MEDIC

MEdical Diagnostic using Intelligent Cameras

# Abschlussdokumentation

Smartphone-basierte Bildverarbeitung
zur Vorbeugung von Pandemien

Auftraggeber:

Dr. Albert Sill

CARL VON OSSIETZKY universität OLDENBURG


Autor:
Projektgruppe MEDIC

Projektleitung:
Patrick Elfert, M. Sc.
Tobias Tiemerding, M. Sc.


30. März 2016

## Vorgelegt von


| Name | E-Mail Adresse | Studiengang |
|---|---|---|
| Danny Fonk | danny.fonk@uni-oldenburg.de | Wirtschaftsinformatik |
| Sebastian Horwege | sebastian.horwege@uni-oldenburg.de | Informatik |
| Raphael Kappes | raphael.kappes@uni-oldenburg.de | Informatik |
| Nicolas Koch | nicolas.koch@uni-oldenburg.de | Wirtschaftsinformatik |
| Timo Raß | timo.rass@uni-oldenburg.de | Wirtschaftsinformatik |
| Christoph Ressel | christoph.ressel@uni-oldenburg.de | Informatik |
| Kevin Sandermann | kevin.sandermann@uni-oldenburg.de | Wirtschaftsinformatik |
| Christian Sandmann | christian.sandmann@uni-oldenburg.de | Informatik |
| Timo Schlömer | timo.schloemer@uni-oldenburg.de | Informatik |
| Jan Philipp Stubbe | jan.philipp.stubbe@uni-oldenburg.de | Informatik |
| Daniel Wegmann | daniel.wegmann@uni-oldenburg.de | Eingeb. Systeme & Mikrorobotik |

# Inhaltsverzeichnis

# Listings

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Glossar

**Account** Vgl. Benutzeraccount.

**Accountinhaber** Eine Person, welche sich mittels eines Account an dem System anmelden kann.

**Administrationsoberfläche** Eine Weboberfläche, welche weder von Probanden, noch von Medizinern eingesehen werden kann. Nur Administratoren des Systems können diese Weboberfläche benutzen.

**Administrator** Nutzer, der exklusiven Zugang zur Administrationsoberfläche hat.

**Administrator** Privilegierte Person, die Zugang zur Administrationsoberfläche des Systems besitzt.

**Android** Ein Betriebssystem von Google, welches für mobile Endgeräte entwickelt wurde.

**Android-Prototyp-App** Prototyp einer App, welche auf einem mobilen Endgerät mit dem Android Betriebssystem ausgeführt wird.

**Android-Studio** Offizielle Entwicklungsumgebung von Google für das Android Betriebssystem.

**Apache-Server** Eine Anwendung, welche Webseiten an andere Computer ausliefern kann.

**App Store** Ein digitaler Marktplatz von Apple, von dem Anwendungen von Apple und weiteren Herstellern bezogen werden können.

**Application Programming Interface** Programmierschnittstelle, die Informationsfluss zwischen Anwendungen ermöglicht.

**Applikation** Eine native Anwendung, welche auf einem mobilen Endgerät ausgeführt wird.

**Arduino** Mikrokontroller-Plattform zum Entwickeln von Hardware-Prototypen.

**Arzt** Vgl. Mediziner.

**Auswertungsoberfläche** Vgl. Weboberfläche.

**Benutzer** Vgl. Proband.

**Benutzeraccount** Daten, mit denen sich ein Proband an dem System anmelden kann.

**Bildverarbeitungsalgorithmus** Algorithmus zur automatischen Detektierung und Auswertung von Teststreifen.

**Canny-Algorithmus** Algorithmus für die Kantenerkennung in einem Bild[2].

**Cascading Style Sheets** Sprache, mit der das Aussehen von Websites angepasst werden kann.

**Comma Separated Values** Textdatei mit Werten, die durch ein Komma voneinander getrennt sind.

**Command Line Interface** Beschreibt das Übergeben von Befehlen an eine Anwendung über die Kommandozeile des Betriebssystems.

**Cookie** Speichermechanismus, mit dem Webseiten Daten auf einem Computer speichern können.

**Corporate Design** Individuelle Designvorgaben für die eiheitliche Gestaltung von Produkten.

**Daemon** Programm, dass als Service im Hintergrund ausgeführt wird.

**Debian** Offene Linux Distribution.

**Diagnose** Vgl. Diagnose des Mediziners.

**Diagnose des Mediziners** Die Diagnose des Mediziners ist das Ergebnis der professionellen Untersuchung des Teststreifens durch einen Mediziner.

**Diagnoseergebnis** Die Diagnose des Mediziners, welche in die Werte 'Unbekannt', 'Positiv' und 'Negativ' eingeteilt werden kann (vgl. Diagnose des Mediziners).

**Doktor** Mediziner.

**Dots per Inch** Einheit, um Druckerauflösung anzugeben.

**Farbumschlagsfeld** Bereich auf dem medizinischen Teststreifen, der als Farbindikator für den Krankheitsfall dient.

**Frames per Second** Bilder pro Sekunde, die angezeigt oder verarbeitet werden können.

**Gemeinsame Codebasis** Code der unverändert auf mehreren Systemen aufgeführt werden kann.

**Geteilte Codebasis** Geteilte Codebasis.

**Git** Ein System, mit dem Quellcode versioniert werden kann. Mehrere Entwickler können so gleichzeitig an einer Anwendung entwickeln.

**Go** Vgl. Golang.

**Golang** Programmiersprache von Google.

**Hash** Ein Hash ist eine Kodierung eines Wortes. Von einem Hash kann nicht auf das ursprüngliche Wort geschlossen werden.

**HSV-Farbraum** Farbraum, der sich in drei Kanäle aufteilt: Wert, Farbwinkel und Sättigung.

**HTTP-Aufruf** Ein Datentransfer über Hypertext Transfer Protocol (HTTP).

**Hypertext Transfer Protocol** Dieses Protokoll wird für Datenübertragungen im Internet eingesetzt und kommt meist beim Anzeigen von Webseiten zum Einsatz.

**Identifikation** Beschriebt einen einzigartigen Bezeichner, wodurch ein Objekt referenziert werden kann.

**iOS** Ein Betriebssystem von Apple, welches für mobile Endgeräte entwickelt wurde.

**iOS-Prototyp-App** Prototyp einer App, welche auf einem mobilen Endgerät mit dem iOS Betriebssystem ausgeführt wird (vgl. Applikation (App) und iOS).

**iOS-SDK** Das Software Development Kit (SDK) von Apple für die Entwicklung von Anwendungen auf dem iOS Betriebssystem.

**iOS-Simulator** Eine Anwendung, welche ein iOS Gerät auf einem regulären Computer simulieren kann.

**iPhone** Ein mobiles Endgerät von Apple, auf dem das iOS Betriebsystem eingesetzt wird.

**JavaEE** Java Enterprise Edition.

**Javascript Object Notation** Ein kompaktes Datenformat in einer einfach lesbaren Textform zum Zweck des Datenaustauschs zwischen Anwendungen.

**Jenkins** Framework für Continuous Integration.

**JQuery** Javascript Framework, das viele Funktionen im Umgang mit Javascript erleichtert.

**Krankheitstest** Test, welcher ein Proband mit Hilfe eines Teststreifens und der mobilen Applikation durchführt.

**Livestream** Ein in Echtzeit laufender Datenstrom.

**Look and Feel** Beschreibt das Aussehen und den Umgang mit einem Produkt.

**Mac** Ein regulärer Computer von Apple, auf dem das Mac OS Betriebssystem zum Einsatz kommt.

**Mac OS** Ein Betriebssystem von Apple, welches für reguläre Computer entwickelt wurde.

**Marker** Vgl. Teststreifendesign.

**Markerdetektion** Prozess, der die automatische Lokalisation eines Markers auf einem Eingabebild beschreibt.

**MEDIC** Name des gesamten Projektes (Medical Diagnostics using Intelligent Cameras).

**Mediziner** Medizinisches Personal, welches Zugang zur Weboberfläche des Systems hat und berechtigt ist, Diagnosen für die Krankheitstests der Probanden anzufertigen.

**Mobile Applikation** Eine Anwendung auf einem Smartphone, welche ein Proband verwendet, um einen Krankheitstest durchzuführen, seinen Krankheitstest zu versenden und eine Diagnose für seinen Krankheitstest zu empfangen.

**Model Viewer Controller** Softwarearchitektur Vorgabe.

**Nutzer** Vgl. Proband.

**OpenCPU Server** Softwaresystem für statistische Berechnungen.

**OpenCV** Bibliothek für die Programmiersprache C, mit der Bildanalyse Algorithmen aufgerufen werden können.

**OTSU-Methode** Adaptive Binarisierungsmethode in der Bildverarbeitung.

**Parser** Anwendung, welche Daten auslesen und auswerten kann.

**PrimeFaces** Java Framework zur Erstellung von Websites.

**Proband** Person, die einen Test durchführt.

**Python** Einfache Programmiersprache, die sich besonders gut für Datenverarbeitung eignet.

**QR-Code** Grafisches, scanbares Element, um einen schnellen Austausch von Informationen zu gewährleisten.

**R** Programmiersprache für Statistiker.

**Referenzfeld** Bereich auf dem medizinischen Teststreifen, der dazu hilft, das Farbumschlagsfeld zu klassifizieren.

**Representational State Transfer** Programmierparadigma für verteilte Systeme, welches den Fokus auf Identifikation und Einheitlichkeit legt.

**Secure Shell** Protokoll zur Kommunikation zwischen verschiedenen Rechnern.

**Server** Geschäftslogik auf Seite des Instituts, um Daten von einem Probanden weiterzuleiten bzw. zu speichern. Der Server beinhaltet u. a. zudem die Logik, damit ein Mediziner Krankheitstests empfangen und eine Diagnose versenden kann.

**Session** Sitzung eines Nutzers.

**SimpleBlob** Algorithmus zum Extrahieren von zusammenängenden Bereichen in einem Bild.

**Smartphone-Applikation** Vgl. App.

**Smartphone-Prototyp** Prototyp einer App.

**Sobel-Algorithmus** Algorithmus für die Kantenerkennung in einem Bild[3].

**Spring** Bibliothek für die Programmiersprache Java, mit der Webseiten und andere Internetdienste bereitgestellt werden können.

**Statistiker** Benutzer der Weboberfläche mit Zugang zur statistischen Auswertung.

**Symfony** PHP-Framework zur Erstellung von Websites.

**System Usability Scale-Test** Einfacher und technologieunabhängiger Fragebogen, zur Bewertung der Gebrauchstauglichkeit eines Systems.

**Test-Framework** Softwaresammlung zur automatischen Auswertung verschiedener Bildverarbeitungsalgorithmen Versionen mithilfe von Videos aus dem Teststand.

**Teststand** Aufbau, mit dem Teststreifen unter reproduzierbaren Bedingungen mit den Smartphones aufgenommen werden können.

**Teststreifen** Papier, welches durch Applizierung einer Körperflüssigkeit Krankheiten indizieren kann.

**Teststreifendesign** Konktrete Definition eines Teststreifenschemas zur Erstellung eines Teststreifens..

**Teststreifengenerator** In der Projektgruppe entwickeltes Software-Werkzeug zur Erstellung von Teststreifen und dem dazugehörigen Teststreifenschema.

**Teststreifenschema** Definition eines Teststreifens, die standartisiert und für einen Computer verständlich ist.

**Teststreifenvariante** Vgl. Teststreifendesign.

**Tomcat-Server** Vgl. Apache-Server.

**UI-Element** Ein einzelnes Bedienelement eines User Interface (UI)..

**Usability** Benutzbarkeit einer Anwendung.

**User Interface** Beschreibt eine meist grafische Darstellung von Bedienelementen mit diesen ein Benutzer eine Anwendung steuern kann.

**Webfrontend** Vgl. Weboberfläche.

**Weboberfläche** Eine Webseite, auf der die Krankheitstests eingesehen werden können.

**Weißabgleich** Methode zur Korrektur der Farbtemperatur einer Aufnahme.

**Xcode** Die offizielle Entwicklungsumgebung von Apple um Anwendungen für das iOS Betriebssystem zu entwickeln.

# Abkürzungsverzeichnis

**Admin** Administrator.

**AJAX** Asynchronous JavaScript and XML.

**API** Application Programming Interface.

**App** Applikation.

**CLI** Command Line Interface.

**CORS** Cross Origin Resource Sharing.

**CSS** Cascading Style Sheets.

**CSV** Comma Separated Values.

**DPI** Dots per Inch.

**FPS** Frames per Second.

**GPS** Global Positioning System.

**HTTP** Hypertext Transfer Protocol.

**ID** Identifikation.

**IP** Internet Protocol.

**JSON** Javascript Object Notation.

**MVC** Model Viewer Controller.

**REST** Representational State Transfer.

**SDK** Software Development Kit.

**SSH** Secure Shell.

**SSL** Secure Socket Layer.

**SUS-Test** System Usability Scale-Test.

**TLS** Transport Layer Security.

**UI** User Interface.

**WLAN** Wireless Local Area Network.

**XML** Extensible Markup Language.

# 1 Einleitung

Das Projekt „MEDIC“ wurde von der Carl-von-Ossietzky Universität unter anderem in Kooperation mit der Universität Tübingen sowie der technischen Universität Braunschweig initiiert. Das übergeordnete Ziel dieses Projektes ist es, papierbasierte Low-Cost-Sensorik umzusetzen und gliedert sich in mehrere Arbeitspakete, welche von den verschiedenen Projektpartnern übernommen werden. Low-Cost-Sensorik ist dabei eine sensorische Auswertung, deren Kosten so gering wie möglich gehalten werden sollen, was in diesem Projekt mittels smartphone-basierter Auswertung realisiert wird. Die Aufgabe der Universität Oldenburg, speziell die der Abteilung Mikrorobotik und Regelungstechnik (AMiR) unter der Leitung von Prof. Dr.-Ing. habil. Sergej Fatikow, liegt dabei auf der smartphone-basierten Auswertung von speziellen Tests - interner Projektname: MEDIC. MEDIC steht dabei für **ME**ical **D**iagnostic using **I**ntelligent **C**ameras.

## 1.1 Motivation

Die Erkennung von Krankheiten ist gleichermaßen für Patienten wie auch für Mediziner äußerst wichtig. Durch stetig wachsende Kosten im Gesundheitswesen (vgl. Abbildung 2) ist es zudem von zunehmender Bedeutung, dass die Erkennung so kostengünstig wie möglich durchgeführt werden kann.

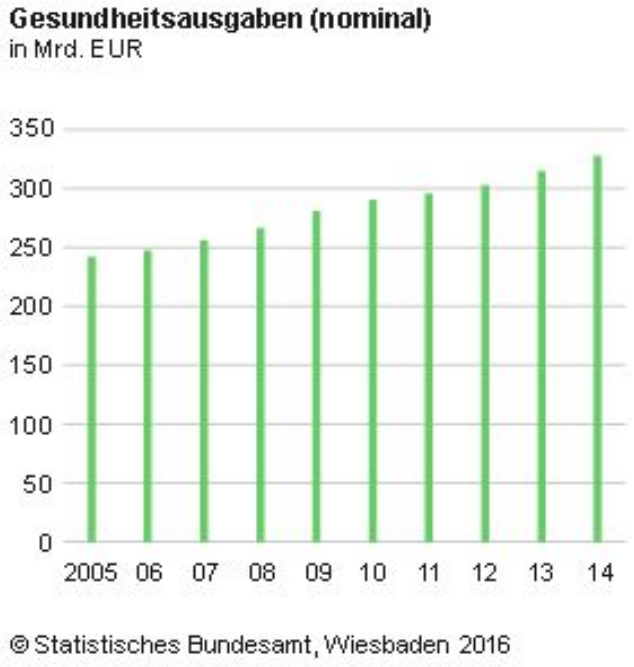



Abbildung 2: Gesundheitsausgaben in den Jahren 2005 bis 2014 [4]

Nimmt man noch die Länder der dritten Welt oder auch die Schwellenländer hinzu, ist es von noch größerer Wichtigkeit die Erkennung der Krankheiten kostengünstig durchführen zu können, da die medizinische Versorgung nicht mit der von beispielsweise Deutschland vergleichbar ist und eine frühe Erkennung von Krankheiten den Krankheitsverlauf und auch die Behandlung signifikant ändern kann. Für Mediziner ist dies ebenfalls wichtig, da vor allem Infektionskrankheiten unkontrolliert zu einer Pandemie (eine länder- und kontinentübergreifende Ausbreitung einer Krankheit) heranwachsen können. Um eine solche Erkennung zu gewährleisten, greift man auf eine Untersuchung am Ort des Patienten zurück (engl. Point-of-Care „POC“). Bekannte Beispiele für das sogenannte Point-of-Care-Testing sind der Schwangerschaftstest oder auch die Blutzuckermessung für Diabetiker [5]. Diese Art der Untersuchung erlaubt es Patienten und Ärzten gleichermaßen, eine schnelle Entscheidung für die weitere Behandlung zu treffen. Die bis dato verfügbaren Testmethoden sind jedoch in ihrer Herstellung oder Auswertung zu teuer, als dass sie breiten Massen zur Verfügung stehen könnten. Daher ist es notwendig, dass die erzeugten Kosten sowohl in der Fertigung als auch in der Auswertung weiter verringert werden. Zudem ist nicht nur die Senkung der Kosten entscheidend, sondern auch die Auswertung über Smartphones für das Vorhaben von größter Relevanz. Durch die Verschiebung des POC allein, ist es noch nicht möglich Aussagen über die Entwicklung einer Krankheit treffen zu können. Erst über die Übermittlung an eine weitere Instanz, welche alle durchgeführten Tests speichert und auswertet, ist es möglich solche Aussagen zu treffen. Dieses Vorhaben stößt jedoch auf weitere Probleme, welche in dem nachfolgenden Kapitel weiter erläutert werden.

## 1.2 Ziele und Problemstellung

Wie bereits beschrieben sind vorhandene Schnelltests für Krankheiten zu teuer um sie in großen Massen zu verteilen. Andere Schnelltests können gar nicht von den Testpersonen selbst ausgewertet werden. Damit müssten sie sich zum Arzt begeben, was zusätzlichen Aufwand darstellt und dazu führen könnte, dass Personen diesen Test gar nicht erst durchführen. Wenn die Personen sich nicht testen lassen, dann ist es auch nicht möglich eine frühe Erkennung von Krankheiten und Krankheitsverläufen zu realisieren. Des Weiteren bedeutet es für Ärzte einen zusätzlichen Aufwand, wenn sie viele Personen bei einem Test auf Krankheiten unterstützen müssen.

Damit die Schnelltests einfach und von einer Vielzahl an Personen durchgeführt werden können, ist es demnach sehr wichtig, dass die Tests an sich nicht zu teuer sind, aber auch die Auswertungsgeräte keine hohen Kosten verursachen. Wichtig dabei ist auch, dass die Ärzte eine Möglichkeit bekommen, die Ergebnisse von Testdurchführungen einzusehen. Ansonsten haben sie nicht die Möglichkeit, Krankheiten und Krankheitsverläufe früh zu erkennen. Dabei sollten sie auch einen geographischen Überblick von der Ausbreitung und Verläufen von Krankheiten bekommen.

Um diese Ziele zu erreichen werden im Rahmen des Projekts sehr günstige Teststreifen entwickelt. Diese können in Papierform an eine große Anzahl von Personen ausgeliefert werden. Für die Auswertung sollen die Smartphones der entsprechenden Personen genutzt werden. Smartphones sind weit verbreitet, sodass eine große Masse an Personen erreicht werden kann. Da fast jeder Zugang zu einem Smartphone hat, fallen damit auch keine weiteren Kosten für Auswertungsgeräte der Tests an. Damit eine möglichst große Zahl von Smartphone-Besitzern die entsprechenden Tests durchführen kann, liegt der Fokus auf besonders kostengünstigen Modellen. Dabei ist davon auszugehen, dass teurere Modelle mit einer besseren Hardwareausstattung diese Tests ebenfalls durchführen können. Dafür wird eine mobile Applikation entwickelt, welche anhand der Verfärbung der Teststreifen erkennt, ob eine Testperson erkrankt ist oder nicht. Diese App schickt die Ergebnisse mit den GPS-Koordinaten an einen Server. Ärzte können sich über ein Webfrontend einloggen und können so die ausgewerteten Testdaten einsehen. Damit bekommen sie einen Überblick über die geographische Ausbreitung von verschiedenen Krankheiten und können ggf. Gegenmaßnahmen ergreifen.

## 1.3 Aufbau der Dokumentation

Die Dokumentation gliedert sich in mehrere Kapitel und soll dabei den eigentlichen Projektverlauf widerspiegeln. Nach dieser Einleitung, in der sich die Motivation, Problemstellung und die Ziele des Projektes wiederfinden, folgt eine Projektplanung, in der das Vorgehen während des gesamten Projektes erläutert wird. Anschließend folgt das Lastenheft, welches zu Projektbeginn erstellt wurde und einen Einblick in die gestellten Anforderungen gewährt. Die Sprintdokumentationen wurden im Anschluss eines jeden Sprints erstellt und bieten somit einen chronologischen Verlauf des Projekts. In den Sprintdokumentationen sind sämtliche Entscheidungen, Rück- und Fortschritte dokumentiert. Die Produktauslieferungen sollen eine Übersicht der Produkte liefern, welche an den Kunden übergeben werden. Dazu dient eine kurze Beschreibung der Produkte und ihr Verwendungszweck. Eine detaillierte Beschreibung der Produkte und wie diese anzuwenden sind, findet sich im Anhang in den jeweiligen Produkt-Handbüchern wieder. Abschließend wird geschildert wie die erzielten Ergebnisse/Produkte evaluiert worden sind und auf Basis dessen ein Fazit für den gesamten Projektverlauf samt Ausblick gezogen.

# 2 Projektplanung

Dieses Kapitel beschreibt den Aufbau und das Vorgehen in der Planung der Projektgruppe. Dabei wird zunächst auf die allgemeine Organisation des Projekts eingegangen und anschließend die genaue Vorgehensweise bei der Sprintplanung beschrieben.

## 2.1 Projektorganisation

Das Projekt wurde in einzelne Abschnitte unterteilt, die „Sprints“ genannt werden. Ziel eines jeden Sprints war es, eine oder mehrere Komponenten des Endprodukts zu erstellen oder zu verbessern. So wurde das Endprodukt sukzessiv erweitert und Stück für Stück aufgebaut. Durch Seminararbeiten wurden zudem Kenntnisse in zusätzlichen Gebieten gewonnen, die anschließend in das Projekt eingebunden werden konnten. Insgesamt wurde das einjährige Projekt in neun Sprints aufgeteilt, die jeweils vier bzw. sechs Wochen andauerten. Das Planen der Sprints wurde dabei vom Projektmanagement–Team übernommen, das zu Beginn des Projekts nur aus Timo Raß bestand und ab dem drittem Sprint durch Daniel Wegmann ergänzt wurde.
Der Fortschritt des Projekts wurde in wöchentlichen Team-Meetings besprochen und am Ende jedes Sprints dem Kunden vorgestellt. In den Meetings wurden ebenfalls Aufgaben an einzelne Mitglieder verteilt, Beschlüsse für das weitere Vorgehen getroffen und Lösungen für bestehende Probleme erörtert. Jedes Meeting wurde von einem Moderator geleitet, der auf die Einhaltung der angesetzten Agenda achtete und von einem weiterem Teammitglied protokolliert. Die angefertigten Protokolle wurden dem übrigen Team per Owncloud zur Verfügung gestellt.
Zum Management des Quellcodes und der Dokumentation wurden Git und GitLab eingesetzt. Bei Git handelt es sich um ein System zur Versionsverwaltung von Dateien. Durch den Einsatz dieser Software ist es zum Einen möglich, ältere Versionen der verwalteten Dateien jederzeit wiederherzustellen, um z.B. auf eine ältere, fehlerfreie Version zurück zu wechseln. Zum Anderen wird durch sogenannte „Branches“ die Zusammenarbeit mehrerer Personen am Projekt erleichtert. Durch das Anlegen eines neuen Branches werden die Dateien in zwei Versionen aufgespalten, so dass sich die Änderungen zweier oder mehrerer Personen nicht gegenseitig beeinflussen können. Nach Abschluss der Arbeit werden die Änderungen im sogenannten „Master–Branch“ zusammengeführt. Bei GitLab handelt es sich um eine Weboberfläche, die zum Management von mit Git verwalteten Projekten eingesetzt wird. Sie erleichtert die Verwaltung des Projekts unter anderem durch eine übersichtliche Darstellung aller aktueller Branches und bietet die Möglichkeit, einen zentralen Informationssammelpunkt („Wiki“) anzulegen.
Um eine hohe Qualität des Quellcodes und der Dokumentation zu gewährleisten wurden jeweils Standards definiert, denen neue oder überarbeitete Elemente entsprechen mussten. Dazu wurden Änderungen nur auf neuen Git–Branches vorgenommen, die nach Fertigstellung über einen Merge–Request in den Master–Branch übernommen wurden. Dieser Merge–Request wurde von den verantwortlichen Teammitgliedern nur akzeptiert, wenn die Qualität den Anforderungen entsprach.

## 2.2 Vorgehensweise

Wie bereits beschrieben, wurde das gesamte Projekt in einzelne Sprints unterteilt. Diese Sprints wiederum setzen sich aus Arbeitspaketen zusammen, die eine feste Aufgabe definieren und – bis auf wenige Ausnahmen – von zwei oder drei Personen bearbeitet wurden. Innerhalb der Arbeitspakete wurde Wert darauf gelegt, die Aufgaben mittels „Pair–Programming“ zu bearbeiten. Dabei handelt es sich um eine Arbeitsweise die vorschreibt, eine Aufgabe stets zu zweit an einem Rechner zu bearbeiten. Das Ziel dieser Technik ist, eine bessere Qualität des Endprodukts zu gewährleisten, indem die zusammenarbeitenden Teammitglieder sich gegenseitig auf Fehler aufmerksam machen und direkt Ideen austauschen können, um so die beste Lösung zu finden. Ein Arbeitspaket definiert dabei sowohl

die eigentliche Aufgabe, als auch die Bedingungen, die erfüllt sein müssen, damit es als abgeschlossen angesehen werden kann.
In den ersten drei Sprints wurden bereits im Voraus alle Arbeitspakete an die Verantwortlichen verteilt und die Zeitplanung fest aufgestellt. Dieses Vorgehen erwies sich jedoch schnell als zu unflexibel. Daher wurden ab dem viertem Sprint nur die „Start"–Arbeitspakete, also die Pakete, die zu Beginn des Sprints angefangen werden müssen, von Anfang an zugewiesen. Sobald ein Team sein Arbeitspaket abgeschlossen hatte, konnten die Mitglieder eines der übrigen offenen Pakete beginnen.
Jedes Arbeitspaket hatte dabei eine Priorität, die entweder „kritisch", „normal" oder „optional" ist. „Kritische" Arbeitspakete haben die höchste Priorität und müssen bis zum Ende des Sprints abgeschlossen sein. Arbeitspakete mit „normaler" Priorität definieren dabei andere essentielle aber nicht zeitkritische Aufgaben. Diese Pakete mussten nicht zwangsweise im entsprechenden Sprint abgeschlossen werden und konnten auch im nächsten Sprint bearbeitet werden. Zusätzlich wurden auch „optionale" Arbeitspakete definiert. Diese Priorität wurde für Zusatzaufgaben definiert, die zwar eine Verbesserung darstellen aber für den Erfolg des Projekts nicht notwendig sind.
Damit das Team sich jederzeit einen Überblick über den Stand des aktuellen Sprints verschaffen konnte, wurde ein Whiteboard in die Abschnitte „TO–DO", „In Bearbeitung" und „Abgeschlossen" unterteilt. In diese Abschnitte wurden die ausgedruckten Beschreibungen der Arbeitspakete gehängt, wodurch jederzeit sichtbar war, welche Arbeitspakete abgeschlossen und welche noch in Bearbeitung oder noch nicht angefangen sind. Auf diese Weise wurde ein agiles Verfahren realisiert, das eine organisierte und zugleich flexible Bearbeitung des Projekts erlaubt.

## 2.3 Zeitplanung

Die folgende Liste gibt eine Übersicht über alle Sprints.

- **Sprint 1** – 01.04.2015 bis 12.05.2015, sechs Wochen – Erstellung des Lastenhefts
- **Sprint 2** – 13.05.2015 bis 09.06.2015, vier Wochen – Erste Prototypen von Android–App, Weboberfläche und Server–Backend – 4.1 auf Seite 43
- **Sprint 3** – 10.06.2015 bis 21.07.2015, sechs Wochen – Erweiterung der Prototypen um neue Funktionen, Erstellen der iOS–App – 4.2 auf Seite 64
- **Sprint 4** – 22.07.2015 bis 08.09.2015, sechs Wochen – Erstellen der neuen Weboberfläche und Verbesserung der Bildverarbeitung – 4.3 auf Seite 99
- **Sprint 5** – 09.09.2015 bis 21.10.2015, sechs Wochen – Fokus auf Bildverarbeitung und Teststand – 4.4 auf Seite 151
- **Sprint 6** – 22.10.2015 bis 23.11.2015, vier Wochen – Optimieren der Bildverarbeitung – 4.5 auf Seite 216
- **Sprint 7** – 24.11.2015 bis 16.12.2015, vier Wochen – Fokus auf Bildverarbeitung und Umsetzen von Lastenheft–Anforderungen – 4.6 auf Seite 268
- **Sprint 8** – 04.01.2016 bis 17.02.2016, sechs Wochen – Fokus auf Bildverarbeitung – 4.7 auf Seite 303
- **Sprint 9** – 18.02.2016 bis 31.03.2016, sechs Wochen – Fertigstellung sämtlicher Produkte und Abschlussdokumentation – 4.8 auf Seite 345

Im folgendem Kapitel werden das Lastenheft und die Ausarbeitung der Anforderungen genauer beschrieben.

# 3 Lastenheft

## 3.1 Einleitung

Die Abteilung Mikrorobotik und Regelungstechnik an der Carl von Ossietzky Universität Oldenburg führt in den Jahren 2015/2016 die Projektgruppe „MEDIC - Smartphonebasierte Bildverarbeitung zur Vermeidung von Pandemien“ durch. Die Aufgabe dieser Projektgruppe ist die Entwicklung eines Systems mit dem Menschen auf Krankheiten getestet werden können. Die Informationen über diese Tests werden an einer zentralen Stelle zusammengetragen, um nach bestimmten Kriterien von Medizinern und Statistikern ausgewertet zu werden. Um eine möglichst große Menge der Bevölkerung zu erreichen, wurde als Auswertungsplattform das Smartphone gewählt. Hiermit sollen Papierteststreifen via Bildverarbeitungsalgorithmen ausgewertet werden. Dieses Projekt wird von 12 Studenten der Masterstudiengänge Eingebettete Systeme und Mikrorobotik, Informatik und Wirtschaftsinformatik innerhalb eines Zeitraums von zwei Semestern umgesetzt.

### 3.1.1 Motivation

Durch den Effekt der Globalisierung ist es für den Menschen immer leichter in fremde Länder rund um den Globus zu reisen. Auch die Chance im eigenen Heimatland auf Menschen aus anderen Ländern zu treffen ist groß. Hierdurch wird das Risiko, dass sich Infektionskrankheiten über Ländergrenzen oder in andere Kontinente verbreiten, immens hoch. Eine solche Infektionskrankheit wird als Pandemie bezeichnet. Diese Pandemien sind z.B. verschiedene Grippearten (Europa und Asien), EHEC (Mitteleuropa), Chikungunya-Fieber (Zentralafrika bis Florida), Pest (Südostafrika) oder Ebolafieber (Afrika). Die Gefahr von Pandemien verdeutlicht sich daran, dass seit dem Jahr 2010 mehr als 16000 Menschen an diesen Krankheiten gestorben sind. Anfang des letzten Jahrhunderts kostete die Spanische Grippe ca. 50 Millionen Menschen das Leben [6]. Neben dem Verlust an Menschenleben verursachen Pandemien auch extreme finanzielle Kosten. Alleine das in Afrika kursierende Ebolafieber wird schätzungsweise Kosten in Höhe von 25 Millionen Euro verursachen [7].
Um solche verheerenden Krankheitsszenarien kontrollieren oder eindämmen zu können, bieten sich prinzipiell zwei Möglichkeiten: Prävention (Impfungen etc.) oder die Identifikation von erkrankten Menschen, um diese zeitnah behandeln zu können. Hierdurch soll die Anzahl der ansteckungsgefährdeten Personen möglichst gering gehalten werden. Wird die Anzahl der potenziell angesteckten Personen auf ein Minimum reduziert, steigt die Chance dass sich die Infektionskrankheit nicht zu einer Pandemie entwickeln kann. Diesen Ansatz verfolgt die Projektgruppe MEDIC, indem ein System zur flächendeckenden Erfassung und Auswertung eines Krankheitsverlaufs entwickelt wird.

### 3.1.2 Zielsetzung

Die technologische Reife von mobilen Geräten wie Smartphones oder Tablets soll mit modernen Techniken der Krankheitserkennung vereint werden. Es wird ermöglicht, dass eine große Masse der Bevölkerung eigenständig Tests auf diverse Krankheiten durchführen kann. Diese Tests werden sowohl von den mobilen Endgeräten, als auch von Experten medizinischer Institute ausgewertet. Dazu werden anonymisierte Daten über das Internet an eine zentrale Datenhaltung gesendet. Dort können diese von einem Mediziner eingesehen werden. Bei Auffälligkeiten wird der Proband von einem Experten kontaktiert. Das Produkt muss dabei in der Lage sein, Anwender ohne medizinische Kenntnisse durch den Ablauf des Testvorgangs zu führen, um ein brauchbares Resultat zu erhalten.

Durch das Produkt wird sowohl für das medizinische Institut als auch für die Probanden ein Mehrwert geschaffen: Durch die Datenintegration kann ein Institut frühzeitig Pandemien erkennen und deren Ausbreitung verhindern. Probanden können ohne technische und fachliche Vorkenntnisse eine medizinische Diagnose erhalten.

Des Weiteren ist es global betrachtet das Ziel einen kostengünstigen Teststreifen zu entwerfen, welcher mit den benötigten Informationen versehen werden kann (z.B. mittels eines QR-Code). Damit ist insbesondere der grafische Aufbau des Teststreifens und nicht die chemische Zusammensetzung gemeint. Der untere Abschnitt des Teststreifens dient dazu ihn in der Hand zu halten. Auf dem mittleren Abschnitt kann der Proband das Körpersekret applizieren. Der obere Abschnitt wird mit der mobilen Endanwendung fotografiert und enthält zudem einen möglichen Farbumschlag des Teststreifens. Der beschriebene Aufbau des Teststreifens kann Abbildung 3 entnommen werden.

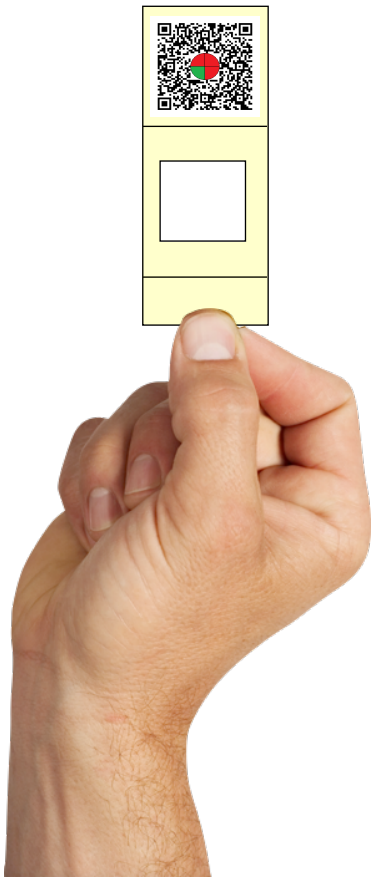


Abbildung 3: Möglicher Aufbau eines Teststreifens

Durch den Einsatz von preiswerten Materialien für die Teststreifen in Verbindung mit dem gesamten technischen System soll zudem erreicht werden, dass sich das Kosten-Nutzen-Verhältnis in einem angemessenen Rahmen befindet, um Massentests durchführen zu können.

## 3.2 Grundlagen

In diesem Kapitel werden die grundlegenden Begrifflichkeiten, die im Rahmen des Projektes verwendet werden, definiert. Des Weiteren wird der technische Ist-Zustand beschrieben, welcher den Ausgangspunkt der Entwicklungsarbeiten darstellt.

### 3.2.1 Begrifflichkeiten

Tabelle 2 auf der nächsten Seite definiert die Begrifflichkeiten, welche in den Anwendungsfällen, Anforderungen und anderen Dokumenten verwendet werden.

| Begriff | Definition |
|---|---|
| Mobile Applikation | Eine mobile Anwendung auf dem Smartphone, welche ein Proband verwendet, um einen Test durchzuführen, seinen Test zu versenden und eine Diagnose für seinen Test zu empfangen. |
| Server | Geschäftslogik auf Seite des Instituts, um Daten von einem Probanden weiterzuleiten bzw. zu speichern. Der Server beinhaltet zudem die Logik, damit ein Mediziner Tests empfangen kann und eine Diagnose versenden kann. |
| Administrationsoberfläche | Die Administrationsoberfläche ist eine Weboberfläche für Systemadministratoren um beispielsweise neue Benutzer anzulegen. |
| Auswertungsoberfläche | Die Auswertungsoberfläche ist eine Weboberfläche für Mediziner und Statistiker. Sie soll z.B. die Tests der Probanden anzeigen und die Möglichkeit bieten eine Diagnose zu jedem Test verfassen zu können. |
| Zu entwickelndes System | Das zu entwickelnde System beinhaltet einen Server mit aller benötigten Geschäftslogik, eine Administrationsoberfläche, eine Auswertungsoberfläche und mobile Applikation für Smartphones. |
| Proband | Ein Proband ist der Test-Durchführende und der Benutzer der mobilen Applikation. |
| Mediziner | Ein Mediziner führt ärztliche Diagnosen durch und kann diese auf der Auswertungsoberfläche verschriftlichen. |
| Systemadministrator | Der Systemadministrator ist für die Verwaltung des zu entwickelnden Systems verantwortlich. |
| Statistiker | Ein Statistiker führt statistische Analysen auf der Auswertungsoberfläche durch. Dies können z.B. geografische Auswertungen der durchgeführten Tests sein. |
| Accountinhaber | Accountinhaber sind Mediziner, Statistiker oder Systemadministratoren. Ein Account wird benötigt, damit nur bestimmte Personen mit unterschiedlichen Kompetenzen eine Sicht auf die Daten erhalten. |
| Ergebnis der Voranalyse | Als Ergebnis der Voranalyse wird die Identifikation eines Farbumschlags auf dem Teststreifen verstanden, welcher durch die mobile Applikation erkannt wird. |
| Teststreifenvariante | Teststreifenvarianten sind die Arten von verschiedenen Teststreifen für unterschiedliche Krankheiten. |
| Diagnose des Mediziners | Die Diagnose des Mediziners ist das Ergebnis der professionellen Untersuchung des Teststreifens durch einen Mediziner. |
| Auswertungskarte | Die Auswertungskarte ist eine interaktive Karte, welche in die Auswertungsoberfläche integriert ist. |
| QR-Code | Ein QR-Code ein grafisches, scanbares Element, um einen schnellen Austausch von Informationen zu gewährleisten. |

Tabelle 2: Begriffe und Definitionen

### 3.2.2 Technischer Ist-Zustand

Der technische Ist-Zustand setzt sich wie folgt zusammen:

- Es existieren sieben Desktop PCs, welche zur Entwicklung der Software genutzt werden können.

- Es existiert ein Rechner, welcher als Server eingesetzt werden kann.
- Es existiert ein MacBook Pro mit 15 Zoll für die Entwicklung von Applikationen für Apple-Geräte.
- Zwölf potenzielle Test-Smartphones der MEDIC-Projektgruppe (iPhone, Android und Windows Phone) sind vorhanden.
- Ein Budget von 5000 Euro für die Aufrüstung weiterer technischer Komponenten ist vorhanden.

## 3.3 Szenarien

Die Szenarien beschreiben die einzelnen Anwendungsbereiche des Projektes. Dazu wurden diese zu vier Szenarien zusammengefasst und in den nachfolgenden Abschnitten beschrieben. Dazu wird ein Anwendungsfalldiagramm verwendet, welches eine Übersicht der Funktionalität eines Szenarios, die Abhängigkeit zwischen den Funktionen und die Beziehung von Akteuren zu den Funktionen darstellt. Ein Szenario setzt sich damit aus einem globalen Anwendungsfalldiagramm, einer Beschreibung, und den Erläuterungen seiner Anwendungsfälle (Funktionen) zusammen. Ein Anwendungsfall beschreibt eine Funktionalität des Produktes und besteht hier aus einem lokalen Anwendungsfalldiagramm sowie einer Beschreibung. Weiterhin werden die Akteure für diesen benannt und der Standardablauf sequenziell aufgelistet. Danach folgen die alternativen Abläufe, sowie Vorbedingungen und Nachbedingungen.

### 3.3.1 Proband führt Test durch

Abbildung 4 zeigt das globale Anwendungsfalldiagramm des Szenarios *Proband führt Test durch*.

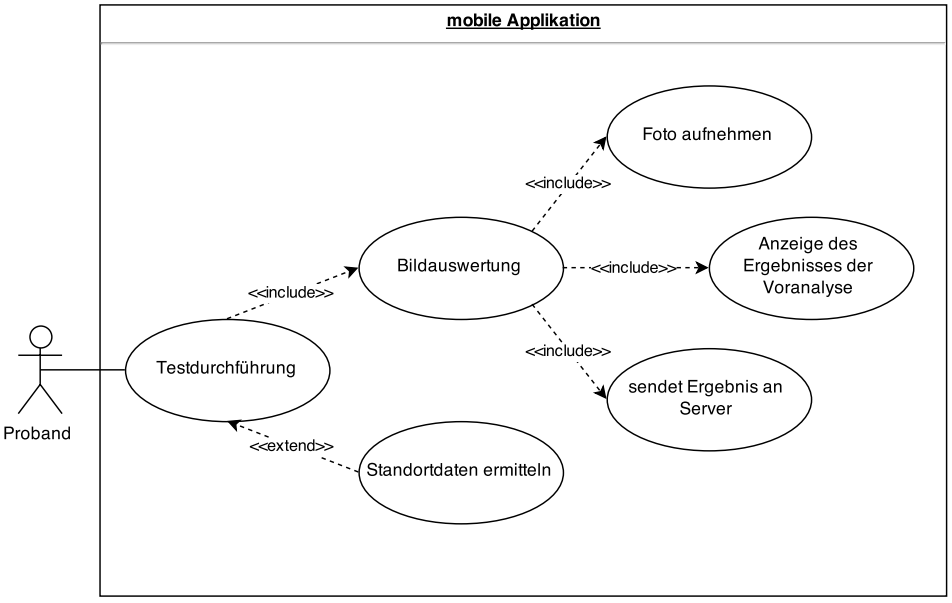



Abbildung 4: Globales Anwendungsfalldiagramm - Proband führt Test durch

**3.3.1.1 Beschreibung** Der Proband möchte einen krankheits-spezifischen Test mittels Smartphone durchführen, um seinen persönlichen Krankenstatus zu erfahren. Dazu gehören die Verwendung der korrekten Teststreifenvariante, sowie die Übermittlung der Daten inklusive Fotografie des benutzten Teststreifens an den Server des Instituts. Dabei dient die Fotografie vor Übermittlung

der Daten an den Server bereits dazu, eine Voranalyse mittels Bilderkennung auf dem Smartphone durchzuführen.
Zur Voranalyse ist es notwendig, dass der Proband eine Teststreifenvariante, auf die ein Sekret appliziert wurde, abfotografiert und optional seine Standortdaten durch die mobile Applikation ermitteln lässt. Zur Validierung durch einen praktizierenden Mediziner ist es notwendig, dass diese Daten an den Server des Dienstleisters übermittelt werden. Während der Ermittlung der Standortdaten und der Durchführung der Voranalyse kann es zu Fehlern kommen, welche durch die mobile Applikation abgefangen und bearbeitet werden. Diese Fehler haben Folgen auf die weitere Auswertung und die Benutzerführung.

**3.3.1.2 Anwendungsfälle** In den folgenden Unterkapiteln werden die einzelnen Anwendungsfälle beschrieben, aus denen das Szenario *Proband führt Test durch* besteht.

**Testdurchführung**

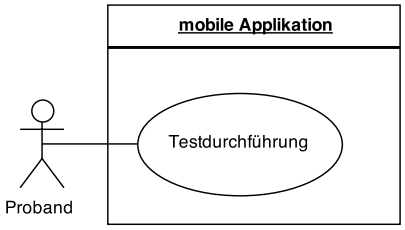



Abbildung 5: Anwendungsfall - Testdurchführung

Der Proband führt einen Test mit Hilfe der mobilen Applikation durch.

- Akteure:
  Proband
- Standardablauf:
  1. Die Mobile Applikation wird gestartet.
  2. Der Proband startet den Testablauf.
  3. Die GPS-Daten werden automatisch ermittelt oder durch den Probanden wird eine Adresse angegeben.
  4. Der Proband nimmt ein Foto von dem mit Sekret appliziertem Teststreifen auf.
  5. Die Voranalyse wird durch Druck auf eine Schaltfläche initiiert.
  6. Die mobile Applikation zeigt das Ergebnis der Smartphone-Analyse an.
  7. Die Daten werden an den Server übermittelt.
- Alternativabläufe:
  - Der Proband startet den Testablauf. Die mobile Applikation stellt fest, dass der Ladestand des Akkus eine bestimmte Schwelle unterschreitet. Dieser Umstand wird dem Probanden mitgeteilt und er wird aufgefordert ein Ladekabel anzuschließen. Sobald dies getan wurde, kann der Test durchgeführt werden.
  - GPS-Daten können aufgrund von ausgeschaltetem Ortungsdienst nicht ermittelt werden und müssen durch den Benutzer manuell eingegeben werden können (Adresse).

- Die Bildaufnahme war unzureichend, sodass der QR-Code nicht ausgelesen oder der Teststreifen nicht ausgewertet werden kann. Die mobile Applikation weist den Probanden darauf hin und fordert ihn auf ein neues Foto zu machen.
- Die Voranalyse ist nicht eindeutig, sodass in der Voranalyse keine Aussage über ein Ergebnis getroffen werden kann. Der Proband wird über diesen Umstand informiert. Das Foto wird dennoch an den Server gesendet, da ein Mediziner unter Umständen auch aus diesem Ergebnis Rückschlüsse ziehen kann.
- Der Proband führt den Test gemäß des Standardablaufs durch. Dabei hat er jederzeit die Möglichkeit die Testdurchführung abzubrechen. Nimmt der Proband diese Möglichkeit wahr, so werden alle bereits erfassten Daten gelöscht und die mobile Applikation kehrt zum Startbildschirm zurück.
- Das Smartphone des Probanden hat keine aktive Internetverbindung oder die Verbindung zum Server schlägt fehl. In diesem Fall wird der Test zu einem späterem Zeitpunkt versandt.

- Vorbedingungen:
  - Der Proband verfügt über einen Teststreifen.
  - Auf den Teststreifen wurde die korrekte Körperflüssigkeit auf die Testfläche appliziert.
  - Die mobile Applikation ist auf dem Smartphone installiert und geöffnet.
  - Das Smartphone verfügt über eine Kamera.
- Nachbedingungen:
  - Dem Probanden liegt das Ergebnis der Voranalyse vor und die Daten wurden erfolgreich an den Server zur Validierung durch einen Mediziner gesandt.

---

**Foto aufnehmen**

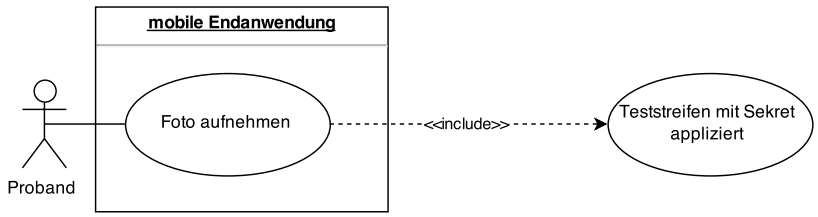



Abbildung 6: Anwendungsfall - Foto aufnehmen

Im Verlauf der Testdurchführung muss ein Foto des Teststreifens aufgenommen werden.

- Akteure:
  Proband
- Standardablauf:
  1. Der Proband bestätigt, dass er bereit ist ein Foto des Teststreifens aufzunehmen.

2. Die mobile Applikation steuert die Kamera des Smartphones an.
3. Der Proband nimmt ein Foto des Teststreifens auf.

- Alternativabläufe:
  - Nachdem der Proband den Teststreifen fotografiert hat, stellt die mobile Applikation fest, dass auf dem Smartphone nicht genügend Speicherplatz für das Foto vorhanden ist. Die mobile Applikation informiert den Probanden über diesen Umstand und fordert dazu auf, Speicherplatz freizuräumen. Nachdem der Proband dies getan hat kann er die Testdurchführung erneut starten.
  - Die Qualität der Bildaufnahme war unzureichend, sodass der QR-Code oder der Teststreifen nicht ausgewertet werden kann. Der Proband wird aufgefordert den Vorgang zu wiederholen, nachdem noch einmal erklärt wurde, wie der Teststreifen zu fotografieren ist.
- Vorbedingungen:
  - Das verwendete Smartphone verfügt über eine Kamera.
  - Der Proband ist innerhalb des Testablaufes bereits an dem Punkt, an dem ein Foto des Teststreifens aufgenommen werden muss. Dies schließt die Vorbedingungen des Anwendungsfalls Testdurchführung mit ein.
- Nachbedingungen:
  - Es existiert ein Foto des Teststreifens, das von der mobilen Applikation zur Voranalyse und späteren Datenübertragung verwendet werden kann.

---

**Bildauswertung**

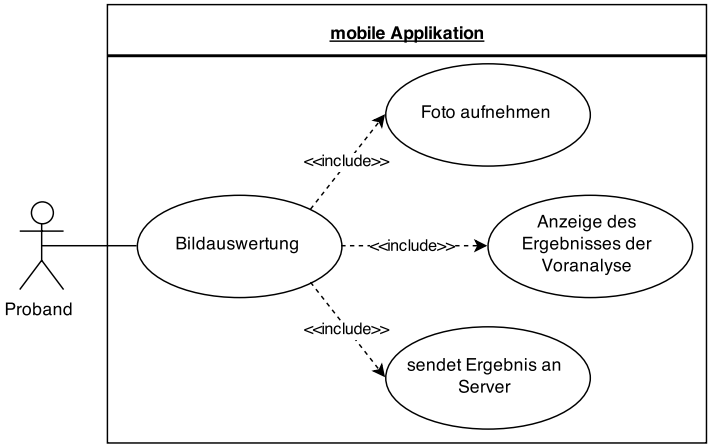



Abbildung 7: Anwendungsfall - Bildauswertung

Die mobile Applikation wertet das Bild des Teststreifens mittels Bilderkennungsalgorithmen aus und stellt ein Ergebnis der Voranalyse für den Probanden bereit. Das Ergebnis kann positiv, negativ oder unbestimmt sein.

- Akteure:
  Proband

- Standardablauf:
  1. Der Proband nimmt ein Foto des Teststreifens auf.
  2. Das Foto wird mittels Bilderkennungsalgorithmen ausgewertet.
  3. Das Ergebnis wird dem Proband mitgeteilt.
- Alternativabläufe:
  - Die Bildaufnahme war unzureichend, sodass der QR-Code oder der Teststreifen nicht ausgewertet werden kann. Der Proband wird aufgefordert den Vorgang zu wiederholen, nachdem noch einmal erklärt wurde wie der Teststreifen zu fotografieren ist.
  - Der Teststreifen ist noch unbenutzt, so dass kein Ergebnis festgestellt werden kann. Der Proband wird aufgefordert zunächst den Teststreifen der Anleitung gemäß zu benutzen und anschließend den Teststreifen zu fotografieren.
- Vorbedingungen:
  - Die Mobile Applikation ist installiert.
  - Das Smartphone ist betriebsbereit.
  - Es ist ein funktionsfähiger Teststreifen vorhanden.
  - Die Testdurchführung wurde gestartet.
  - Das Foto vom Teststreifen wurde bereits aufgenommen.
- Nachbedingungen:
  - Es liegt ein Ergebnis der Voranalyse des Teststreifens vor.

---

**Anzeige des Ergebnis**

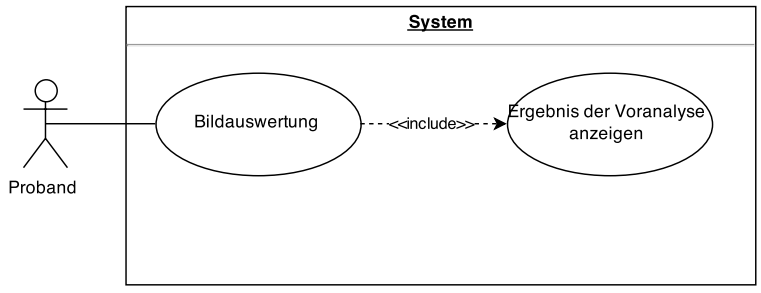



Abbildung 8: Anwendungsfall - Anzeige des Ergebnisses

Die mobile Applikation zeigt das vorläufige Ergebnis der Bilderkennung an.

- Akteure:
  Proband
- Standardablauf:

1. Die Voranalyse wird durchgeführt.
2. Das Ergebnis der Voranalyse wird dem Probanden angezeigt.

- Alternativabläufe:
  - Die Qualität der Bildaufnahme war unzureichend, sodass der QR-Code oder der Teststreifen nicht ausgewertet werden kann. Der Proband wird aufgefordert den Vorgang zu wiederholen, nachdem noch einmal erklärt wurde, wie der Teststreifen zu fotografieren ist.
- Vorbedingungen:
  - Die Mobile Applikation ist installiert.
  - Das Smartphone ist betriebsbereit.
  - Es ist ein funktionsfähiger Teststreifen vorhanden.
  - Die Testdurchführung wurde gestartet.
  - Das Ergebnis der Bilderkennung steht bereit.
- Nachbedingungen:
  - Die Daten stehen bereit, um an den Server versendet zu werden.

---

**Mobile Applikation sendet Ergebnis an Server**

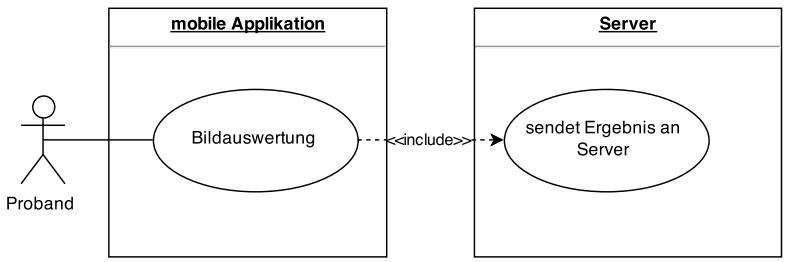



Abbildung 9: Anwendungsfall - mobile Applikation sendet Ergebnis an Server

Das Ergebnis der Bilderkennung, welches im Zuge der Testdurchführung ermittelt wurde, wird an den Server gesendet.

- Akteure:
  Proband
- Standardablauf:
  1. Die Standortinformationen werden ermittelt.
  2. Das Ergebnis der Bilderkennung wird inklusive des aufgenommenen Fotos und den Standortinformationen zu einem Datenpaket verpackt.
  3. Das Datenpaket wird an den Server verschickt.

- Alternativabläufe:
  - Eine Verbindung zum Server lässt sich nicht aufbauen. Der Verbindungsversuch wird später wiederholt. Die Daten werden währenddessen auf dem Smartphone zwischengespeichert.
- Vorbedingungen:
  - Die Mobile Applikation ist installiert.
  - Das Smartphone ist betriebsbereit.
  - Es ist ein funktionsfähiger Teststreifen vorhanden.
  - Die Testdurchführung wurde gestartet.
  - Das Ergebnis der Bilderkennung steht bereit.
  - Es sind Standortinformationen vorhanden.
- Nachbedingungen:
  - Die Daten wurden erfolgreich an den Server übersandt.
  - Auf dem Smartphone wurden die erhobenen Daten wieder gelöscht.

---

### 3.3.2 Institut verwaltet Infrastruktur

Abbildung 10 zeigt das globale Anwendungsfalldiagramm des Szenarios *Institut verwaltet Infrastruktur*.

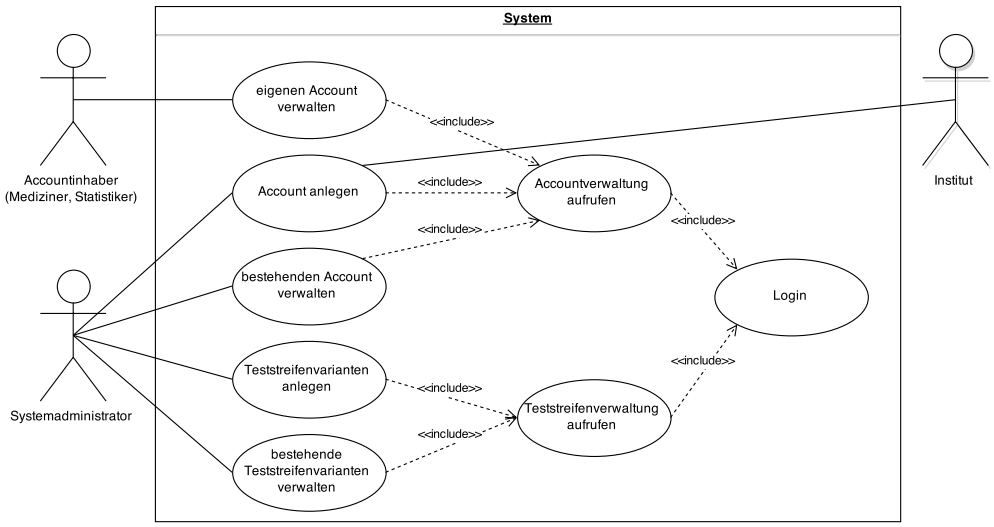



Abbildung 10: Globales Anwendungsfalldiagramm - Institut verwaltet Infrastruktur

**3.3.2.1 Beschreibung** Das Szenario *Institut verwaltet Infrastruktur* beschäftigt sich mit allen administrativen Aufgaben an der im Institut befindlichen Infrastruktur. Beispiele hierfür sind die Verwaltung von Benutzeraccounts für die Nutzung des Systems oder die Datenhaltung und Wartung aller aktiven Teststreifenvarianten. Ebenfalls von Bedeutung ist eine Schnittstelle zwischen der mobilen Applikation und dem Server, welche eine Übertragung aktuell verfügbaren Teststreifenvarianten ermöglicht.

**3.3.2.2 Anwendungsfälle** In den folgenden Unterkapiteln werden die einzelnen Anwendungsfälle beschrieben, aus denen das Szenario *Institut verwaltet Infrastruktur* besteht.

**Account anlegen**

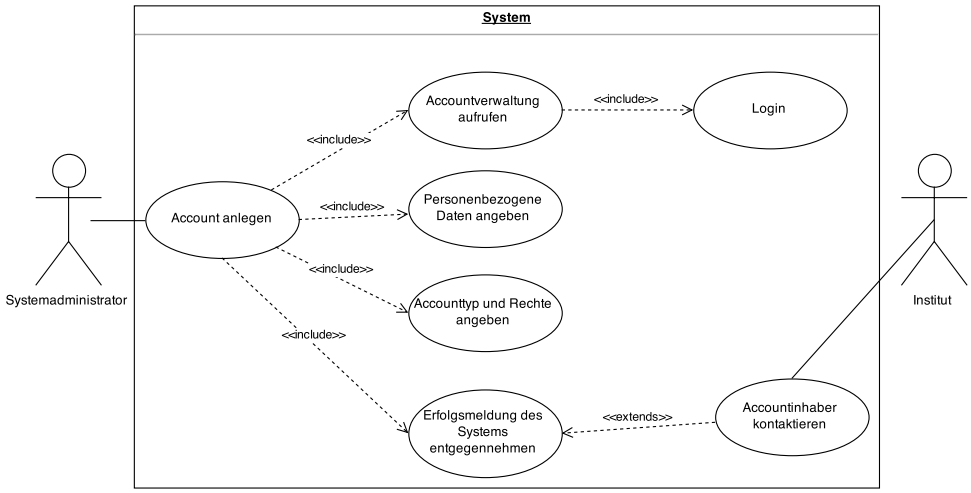



Abbildung 11: Anwendungsfall - Account anlegen

Dieser Anwendungsfall beschäftigt sich mit der Möglichkeit eines Systemadministrators, einen Account zur Nutzung der serverseitigen Funktionen zu erstellen. Hierbei ist zunächst nicht vorgeschrieben, ob dieser Account für einen Mediziner, einen Mitarbeiter des Instituts oder einen Systemadministrator vorgesehen ist.

- Akteure:
  Systemadministrator, Institut.
- Standardablauf:
  1. Der Systemadministrator loggt sich im System ein.
  2. Der Systemadministrator ruft die Oberfläche zur Accountverwaltung auf.
  3. Anschließend erfasst der Systemadministrator über eine geeignete Schaltfläche die Daten, die zum Erstellen eines Account benötigt werden.
  4. Zunächst werden Accounttyp und -rechte angegeben. Hier sind die Optionen eines Mediziners, eines Mitarbeiters des Instituts oder eines Systemadministrators vorgesehen.
  5. Als weiteren Schritt der Accountdatenerfassung werden personenbezogene Daten eingegeben.
  6. Nachdem die Daten erfasst sind, wartet der Systemadministrator auf die Erfolgsrückmeldung des Servers.
  7. Nach erfolgreicher Erstellung des Accounts kann der Accountinhaber hierüber informiert werden und seine Logindaten können ihm mitgeteilt werden.
- Alternativabläufe:
  - Der Loginvorgang des Systemadministrators schlägt fehl: Dem Systemadministrator wird der Zugang verwehrt. Ein neuer Loginversuch kann gestartet werden.

  - Der zu erstellende Account existiert bereits: Dem Systemadministrator wird eine entsprechende Meldung angezeigt. Ihm wird eine Schaltfläche zur Anzeige des Accounts zur Verfügung gestellt.
  - Die Verbindung zum Accountverwaltungssystem bricht ab: Dem Systemadministrator wird eine passende Meldung angezeigt. Der Vorgang wird abgebrochen und muss später erneut gestartet werden.

- Vorbedingungen:
  - Ein Auftrag zur Erstellung eines Accounts inkl. sämtlicher zugehöriger Daten liegt vor.
  - Der beauftragte Systemadministrator verfügt über einen Zugang und passende Berechtigung zum aktiven Accountverwaltungssystem.

- Nachbedingungen:
  - Es existiert eine neuer Account mit passenden Daten und Rechten.
  - Der Accountinhaber wurde eventuell über seinen Account inkl. seiner Logindaten informiert.

---

**Bestehenden Account verwalten**

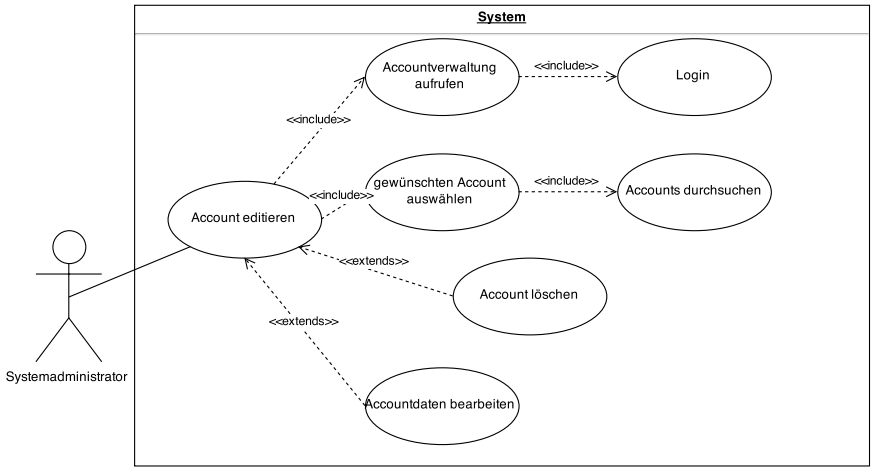



Abbildung 12: Anwendungsfall - Bestehenden Account verwalten

Dieser Anwendungsfall beschäftigt sich mit der Möglichkeit eines Systemadministrators, einen bereits bestehenden Nutzeraccount zu verwalten. Hierbei besteht sowohl die Möglichkeit, dass einem Account zugehörige Daten abgeändert werden, als auch die Möglichkeit, einen vorhandenen Account zu löschen.

- Akteure:
  Systemadministrator

- Standardablauf:
  1. Der Systemadministrator loggt sich im System ein.
  2. Der Systemadministrator ruft die Oberfläche zur Accountverwaltung auf.

3. Anschließend sucht der Systemadministrator mit Hilfe einer Suchfunktion gezielt nach dem zu verwaltenden Account.
4. Der passende Account wird aus einer Liste an Suchergebnissen ausgewählt.
5. Optional kann anschließend eine der folgenden Funktionen durchgeführt werden:
   - Der ausgewählte Account wird gelöscht.
   - Die Daten des ausgewählten Accounts werden bearbeitet.

- Alternativabläufe:
  - Der Loginvorgang des Systemadministrators schlägt fehl: Dem Systemadministrator wird der Zugang verwehrt. Ein neuer Loginversuch kann gestartet werden.
  - Der zu erstellende Account existiert bereits: Dem Systemadministrator wird eine entsprechende Meldung angezeigt. Ihm wird eine Schaltfläche zur Anzeige des Accounts zur Verfügung gestellt.
  - Die Verbindung zum Accountverwaltungssystem bricht ab: Dem Systemadministrator wird eine passende Meldung angezeigt. Der Vorgang wird abgebrochen und muss später erneut gestartet werden.
- Vorbedingungen:
  - Ein Auftrag zum Löschen oder Bearbeiten eines Accounts inkl. sämtlicher zugehöriger Daten liegt vor.
  - Der Account lässt sich anhand der Daten eindeutig identifizieren.
  - Der beauftragte Systemadministrator verfügt über einen Zugang und passende Berechtigungen zum Accountverwaltungssystem.
- Nachbedingungen:
  - Der zu verwaltende Account wurde gelöscht oder bearbeitet.

---

**Verfügbare Teststreifen verwalten**

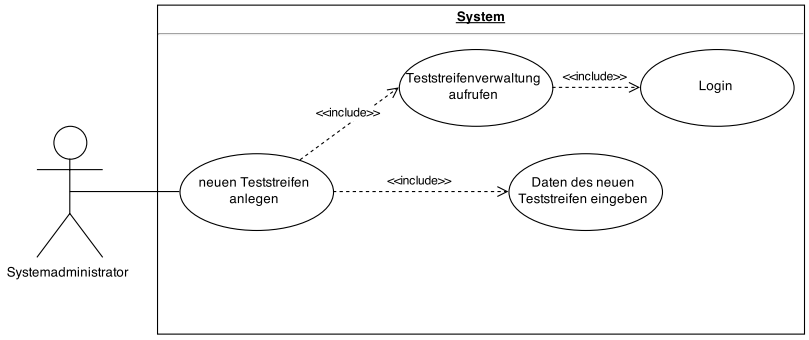



Abbildung 13: Anwendungsfall - Verfügbare Teststreifen verwalten

Dieser Anwendungsfall beschäftigt sich mit der Möglichkeit eines Systemadministrators neue Teststreifenvarianten anzulegen. Teststreifenvarianten beschreiben die Arten von Krankheitstests, die aktuell im Umlauf sind und durchgeführt werden können.

- Akteure:
  Systemadministrator
- Standardablauf:
  1. Der Systemadministrator loggt sich im System ein.
  2. Der Systemadministrator ruft die Oberfläche zur Teststreifenverwaltung auf.
  3. Der Systemadministrator erfasst die Daten eines Teststreifens und legt einen neuen Teststreifen an.
  4. Nach erfolgreichem Anlegen der Teststreifenvariante gibt der Server eine entsprechende Erfolgsmeldung aus.
- Alternativabläufe:
  - Der Loginvorgang des Systemadministrators schlägt fehl: Dem Systemadministrator wird der Zugang verwehrt. Ein neuer Loginversuch kann gestartet werden.
  - Die Verbindung zum Teststreifenverwaltungssystem bricht ab: Dem Systemadministrator wird eine passende Meldung angezeigt. Der Vorgang wird abgebrochen und muss später erneut gestartet werden.
  - Die Daten einer neuen Teststreifenvariante sind syntaktisch oder semantisch fehlerhaft: Dem Systemadministrator wird eine passende Fehlermeldung angezeigt. Die Daten können überarbeitet werden.
- Vorbedingungen:
  - Ein Auftrag zur Erstellung eines Teststreifens inkl. sämtlicher zugehöriger Daten liegt vor.
  - Der beauftragte Systemadministrator verfügt über einen Zugang und passende Berechtigung zum aktiven Teststreifenverwaltungssystem.
- Nachbedingungen:
  - Die neue Teststreifenvariante wurde erstellt.

---

### Bestehende Teststreifen bearbeiten

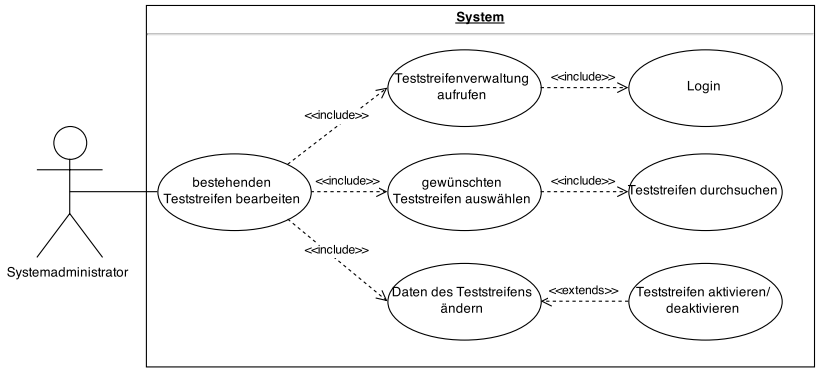



Abbildung 14: Anwendungsfall - Bestehende Teststreifen bearbeiten

Dieser Anwendungsfall beschäftigt sich mit der Möglichkeit eines Systemadministrators, verfügbare Teststreifenvarianten zu verwalten. Verfügbare Teststreifenvarianten beschreiben die Arten von Krankheitstests, die aktuell im Umlauf sind und durchgeführt werden können.

- Akteure:
  Systemadministrator
- Standardablauf:
  1. Der Systemadministrator loggt sich im System ein.
  2. Der Systemadministrator ruft die Oberfläche zur Teststreifenverwaltung auf.
  3. Der Systemadministrator sucht anhand vorliegender Daten gezielt nach dem zu verwaltenden Teststreifen.
  4. Der passende Teststreifen wird aus einer Liste an Suchergebnissen ausgewählt.
  5. Der Systemadministrator ändert die Daten des ausgewählten Teststreifens.
  6. Optional kann der Testreifen aktiviert bzw. deaktiviert werden.
- Alternativabläufe:
  - Der Loginvorgang des Systemadministrators schlägt fehl: Dem Systemadministrator wird der Zugang verwehrt. Ein neuer Loginversuch kann gestartet werden.
  - Die Verbindung zum Teststreifenverwaltungssystem bricht ab: Dem Systemadministrator wird eine passende Meldung angezeigt. Der Vorgang wird abgebrochen und muss später erneut gestartet werden.
  - Die neuen Daten einer Teststreifenänderung sind syntaktisch oder semantisch fehlerhaft: Dem Systemadministrator wird eine passende Fehlermeldung angezeigt. Die Daten können überarbeitet werden.
- Vorbedingungen:
  - Ein Auftrag zur Bearbeitung eines Teststreifens inkl. sämtlicher zugehöriger Daten liegt vor.
  - Der Teststreifen lässt sich anhand der Daten eindeutig identifizieren.

  - Der beauftragte Systemadministrator verfügt über einen Zugang und passende Berechtigung zum aktiven Teststreifenverwaltungssystem.

- Nachbedingungen:
  - Es wurde die zu verwaltende Teststreifenvariante bearbeitet.

---

**Accountinhaber editiert Account**

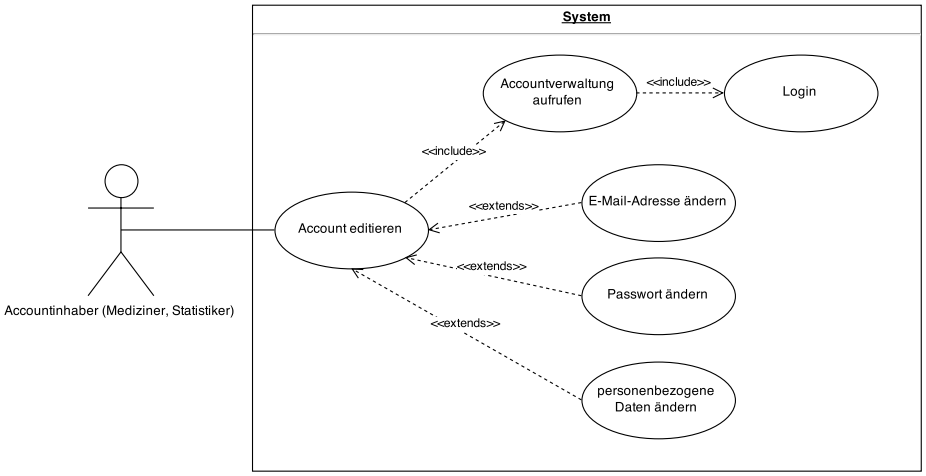



Abbildung 15: Anwendungsfall - Accountinhaber editiert Account

Dieser Anwendungsfall beschäftigt sich mit der Möglichkeit eines Accountinhabers, seinen bereits bestehenden Nutzeraccount zu verwalten. Hierbei besteht sowohl die Möglichkeit, dass dem Account zugehörige personenbezogene Daten abgeändert werden als auch die Zugangsdaten, d.h. die Identifikation und das Passwort zu editieren. Accountinhaber kann ein Mediziner oder Statistiker sein.

- Akteure:
  Accountinhaber (Mediziner, Statistiker)

- Standardablauf:
  1. Der Accountinhaber loggt sich im System ein.
  2. Der Accountinhaber ruft die Oberfläche zur Accountverwaltung auf.
  3. Der Accountinhaber kann seine E-Mail-Adresse ändern.
  4. Der Accountinhaber kann sein Passwort ändern.
  5. Der Accountinhaber kann personenbezogene Daten ändern. Dies beinhaltet Vor- und Nachname, Praxisadresse sowie Telefon- und Faxnummer.

- Alternativabläufe:
  - Der Loginvorgang des Accountinhabers schlägt fehl: Dem Accountinhaber wird der Zugang verwehrt. Ein neuer Loginversuch kann gestartet werden.

- Die Verbindung zum Accountverwaltungssystem bricht ab: Dem Accountinhaber wird eine passende Meldung angezeigt. Der Vorgang wird abgebrochen und muss später erneut gestartet werden.
- Die Inhalte einer Accountdatenänderung sind syntaktisch oder semantisch fehlerhaft: Dem Nutzer wird eine passende Fehlermeldung angezeigt. Die Daten können überarbeitet werden.

- Vorbedingungen:
  - Ein Accountinhaber möchte seine Accountdaten ändern.
  - Der Accountinhaber verfügt über einen Zugang zum aktiven Accountverwaltungssystem.
- Nachbedingungen:
  - Die Accountdaten sind syntaktisch und semantisch fehlerfrei und wurden geändert.

---

### 3.3.3 Kommunikation Mediziner - Proband

Abbildung 16 zeigt das globale Anwendungsfalldiagramm des Szenarios *Kommunikation Mediziner - Proband*.

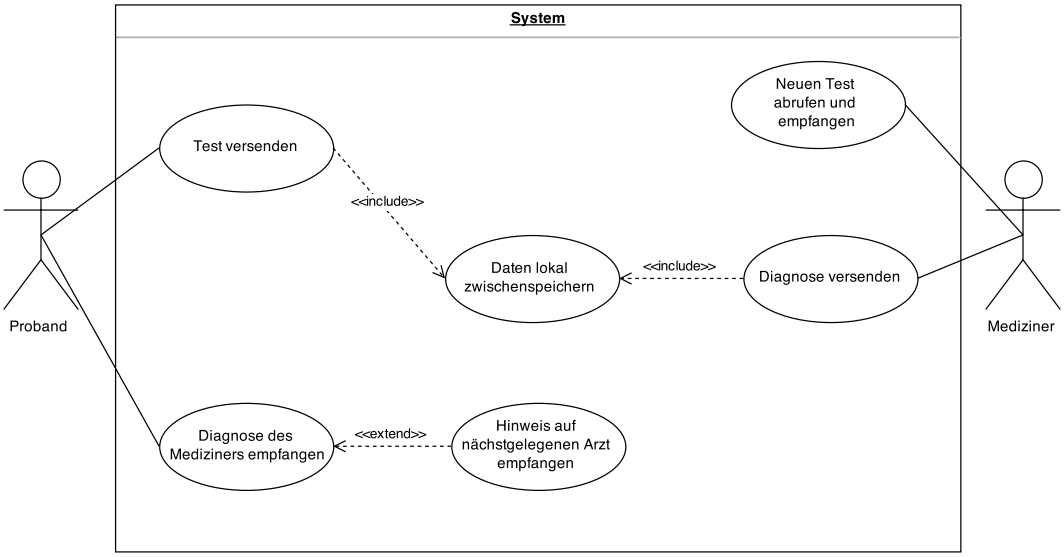



Abbildung 16: Globales Anwendungsfalldiagramm - Kommunikation Mediziner - Proband

**3.3.3.1 Beschreibung** Dieses Szenario beschäftigt sich mit der Kommunikationsschnittstelle, die zwischen dem Proband und dem Mediziner besteht. Diese existiert in zwei Ausprägungen. Zum einen kann der Proband einen Test an das Institut versenden, zum anderen existiert ein Kommunikationskanal vom Mediziner zum Probanden, um Diagnosen zu verschicken.

**3.3.3.2 Anwendungsfälle** In den folgenden Unterkapiteln werden die einzelnen Anwendungsfälle beschrieben, aus denen das Szenario *Kommunikation Mediziner - Proband* besteht.

**Test versenden**

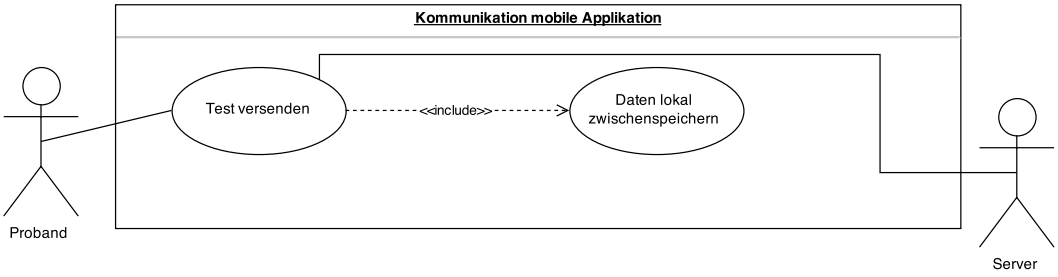



Abbildung 17: Anwendungsfall - Test versenden

Der Anwendungsfall *Test versenden* zeigt den Standardablauf, den Alternativablauf, sowie die dazugehörigen Vor- und Nachbedingungen auf, welche diesem zugeordnet sind.

- Akteure:
  Proband, Server
- Standardablauf:
  1. Bei der Datenerfassung werden die Testdaten lokal zwischengespeichert.
  2. Die Daten werden an einen Server übertragen.
- Alternativabläufe:
  - Es ist keine Internetverbindung vorhanden: Bei der Datenerfassung werden die Testdaten lokal zwischengespeichert. Es ist keine Internetverbindung vorhanden, daher muss darauf gewartet werden, dass diese hergestellt wird. Wenn die Internetverbindung hergestellt werden konnte, wird eine verschlüsselte Verbindung zum Server hergestellt. Die Daten werden an den Server übertragen.
  - Die Internetverbindung bricht während der Übertragung ab: Bei der Datenerfassung werden die Testdaten lokal zwischengespeichert. Die Daten werden an den Server übertragen. Während der Datenübertragung an den Server bricht die Internetverbindung ab. Es wird darauf gewartet, dass die Internetverbindung wiederhergestellt wird. Wenn die Internetverbindung wiederhergestellt werden konnte, wird erneut eine verschlüsselte Verbindung zum Server hergestellt. Die Daten werden erneut an den Server übertragen.
- Vorbedingungen:
  - Die mobile Applikation ist auf dem Smartphone vorhanden.
  - Es ist ein Server vorhanden, der Daten der mobilen Applikation empfangen kann.
  - Es ist eine Internetverbindung vorhanden.
  - Es besteht eine verschlüsselte Verbindung zum Server.
  - Die Test-Durchführung auf dem Smartphone konnte fehlerfrei abgeschlossen werden.
- Nachbedingungen:
  - Die Endanwendung wird nicht vom Smartphone gelöscht.

– Die lokal gespeicherten Daten werden gelöscht.
– Die Daten werden auf dem Server gespeichert.
– Die mobile Applikation ist bereit einen neuen Test durchzuführen.
– Es besteht weiterhin eine Internetverbindung.

---

**Neuen Test abrufen und empfangen**

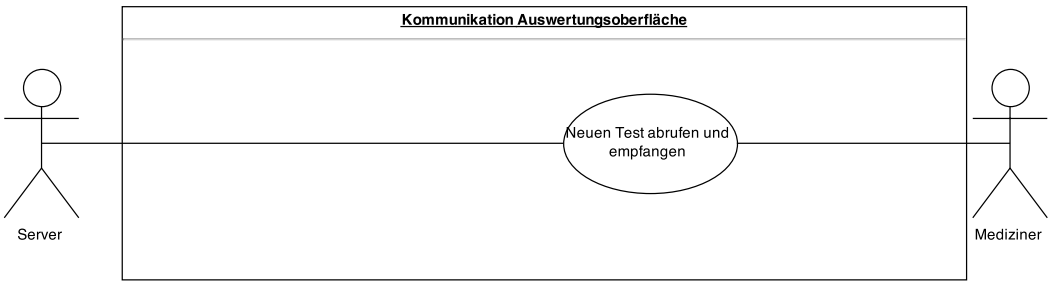



Abbildung 18: Anwendungsfall - Neuen Test abrufen und empfangen

Der Anwendungsfall „Neuen Test abrufen und empfangen“ zeigt den Standardablauf, den Alternativablauf, sowie die dazugehörigen Vor- und Nachbedingungen auf, welche diesem zugeordnet sind.

- Akteure:
  Mediziner, Server
- Standardablauf - Neuen Test abrufen und empfangen:
  1. Die Auswertungsoberfläche des Mediziners fragt beim Server Testdaten eines Probanden ab.
  2. Der Server überträgt den angefragten Datensatz an die Auswertungsoberfläche des Mediziners.
  3. Der Mediziner wird benachrichtigt, nachdem der ausgewählte Datensatz empfangen wurde.
  4. Der empfangene Datensatz wird dargestellt.
- Alternativablauf:
  – Die Internetverbindung bricht während der Übertragung ab: Die Auswertungsoberfläche des Mediziners fragt beim Server Testdaten eines Probanden an. Der Server überträgt den angefragten Datensatz an die Auswertungsoberfläche des Mediziners. Die Verbindung zum Server bricht während der Datenübertragung ab. Die Auswertungsoberfläche des Mediziners wartet darauf, dass die Internetverbindung wiederhergestellt ist. Sobald die Internetverbindung wiederhergestellt ist, wird eine verschlüsselte Verbindung zum Server aufgebaut. Sobald die verschlüsselte Verbindung zum Server hergestellt ist, startet dieser erneut die Übertragung des angefragten Datensatzes an die Auswertungsoberfläche des Mediziners. Der Mediziner wird benachrichtigt, nachdem die ausgewählten Testdaten empfangen wurden. Die empfangenen Testdaten werden dargestellt.

- Vorbedingungen:
  - Es ist eine Internetverbindung vorhanden.
  - Es besteht eine verschlüsselte Verbindung zum Server.
  - Der Mediziner ist in der Auswertungsoberfläche eingeloggt.
  - Der Mediziner hat einen Proband und einen Test ausgewählt.
  - Der Server hat die Testdaten des Probanden erfolgreich empfangen.
  - Der Server hat die Testdaten des Probanden erfolgreich gespeichert.
- Nachbedingungen:
  - Die Auswertungsoberfläche des Mediziners stellt die empfangenen Daten dar und ist bereit, erneut neue Testdaten von Probanden zur Auswertung zu empfangen.

---

**Diagnose versenden**

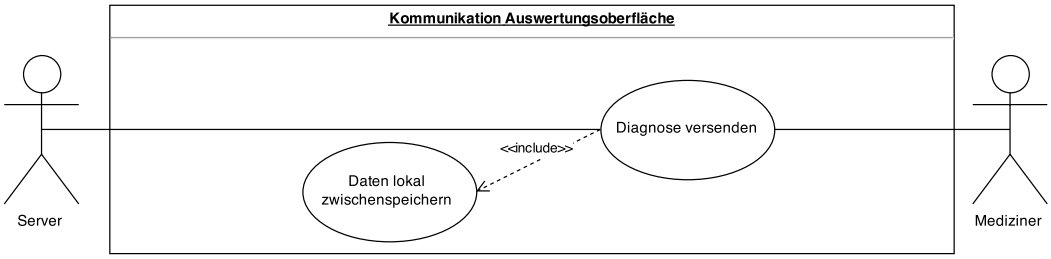



Abbildung 19: Anwendungsfall - Diagnose versenden

Der Anwendungsfall „Diagnose versenden“ zeigt den Standardablauf, den Alternativablauf, sowie die dazugehörigen Vor- und Nachbedingungen auf, welche diesem zugeordnet sind.

- Akteure:
  Mediziner, Server
- Standardablauf - Diagnose versenden:
  1. Die verfasste Diagnose wird lokal zwischengespeichert.
  2. Die Diagnose wird an den Server übertragen.
  3. Sobald die Diagnose erfolgreich übertragen wurde, wird deren lokale Kopie gelöscht.
- Alternativablauf:
  - Die Internetverbindung bricht während der Datenübertragung ab: Die verfasste Diagnose wird lokal zwischengespeichert. Die Diagnose wird an den Server gesendet. Während der Datenübertragung an den Server bricht die Internetverbindung ab. Es wird gewartet, bis die Internetverbindung wieder hergestellt wurde. Es wird eine verschlüsselte Verbindung zum Server hergestellt. Die Diagnose wird erneut an den Server übertragen. Sobald die Diagnose erfolgreich übertragen wurde, wird deren lokale Kopie gelöscht.

- Vorbedingungen:
  - Es ist eine Internetverbindung vorhanden.
  - Es besteht eine verschlüsselte Verbindung zum Server.
  - Der Mediziner hat sich erfolgreich in die Auswertungsoberfläche eingeloggt.
  - Der Mediziner hat eine Diagnose zu dem Test eines Probanden verfasst.
- Nachbedingungen:
  - Die generierte Diagnose ist beim Server persistent gespeichert.
  - Die Auswertungsoberfläche des Mediziners ist bereit neue Testdaten von Probanden zur Auswertung zu empfangen.

---

**Diagnose des Mediziners empfangen**

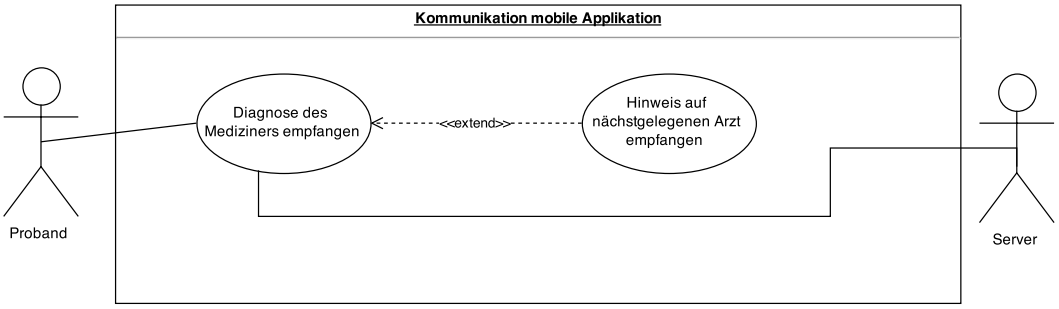



Abbildung 20: Anwendungsfall - Diagnose des Mediziners empfangen

Der Anwendungsfall „Diagnose des Mediziners empfangen“ zeigt den Standardablauf, den Alternativablauf, sowie die dazugehörigen Vor- und Nachbedingungen auf, welche diesem zugeordnet sind.

- Akteure:
  Proband, Server
- Standardablauf:
  1. Der Server schickt die Diagnose des Mediziners und gegebenenfalls einen Hinweis auf einen nächstgelegenen Arzt an die mobile Applikation des Probanden.
- Alternativablauf:
  - Die Internetverbindung bricht während der Übertragung ab: Der Server schickt die Diagnose des Mediziners und gegebenenfalls einen Hinweis auf einen nächstgelegenen Arzt an die mobile Applikation des Probanden. Die Verbindung zum Server bricht während der Datenübertragung ab. Die mobile Applikation wartet auf die Wiederherstellung der Internetverbindung. Die mobile Applikation stellt eine verschlüsselte Verbindung zum Server her. Sobald die Verbindung zum Server hergestellt ist, startet dieser erneut die Datenübertragung an die mobile Applikation des Probanden. Die mobile Applikation des Probanden speichert die empfangenen Daten persistent lokal ab und stellt sie entsprechenden Ansichten in der mobilen Applikation zur Verfügung.

- Vorbedingungen:
  - Die mobile Applikation ist auf dem Smartphone vorhanden.
  - Es ist ein Server vorhanden, der Daten an die mobile Applikation senden kann.
  - Es ist eine Internetverbindung vorhanden.
  - Es besteht eine verschlüsselte Verbindung zum Server.
  - Es wurden erfolgreich Testdaten von der mobilen Applikation an den Server gesendet.
- Nachbedingungen:
  - Die mobile Applikation des Probanden speichert die empfangenen Daten persistent lokal ab und stellt die entsprechenden Ansichten in der mobilen Applikation zur Verfügung.

---

### 3.3.4 Auswertung durch Mediziner

Abbildung 21 zeigt das globale Anwendungsfalldiagramm des Szenarios *Auswertung durch Mediziner*.

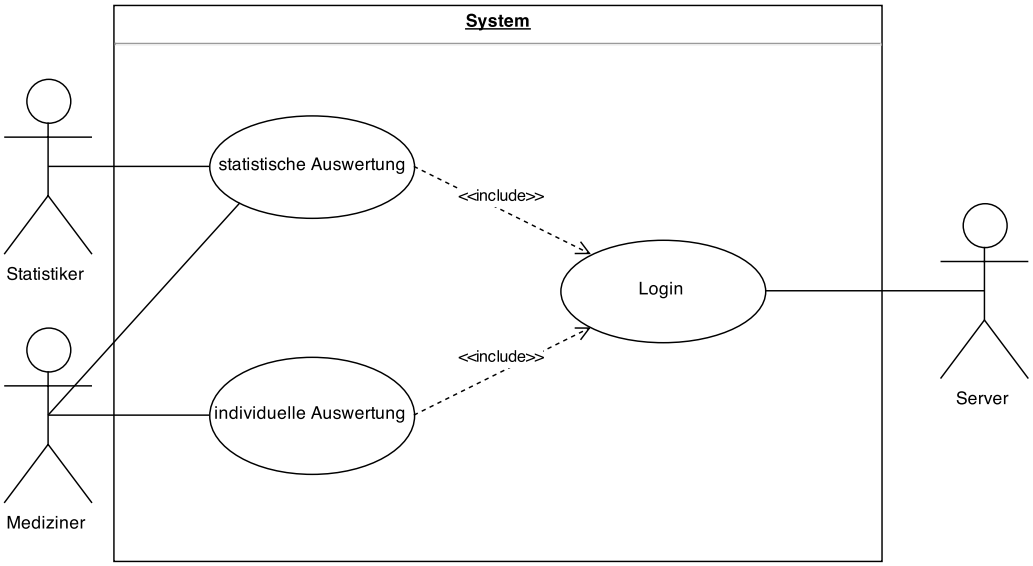



Abbildung 21: Anwendungsfalldiagramm - Auswertung durch Mediziner

**3.3.4.1 Beschreibung** Die Auswertung durch einen Mediziner schließt einerseits die statistische Auswertung ein, wobei diese durch einen Statistiker und einen Mediziner geschehen kann. Anderseits wird die individuelle Auswertung, also die Diagnosestellung für einen Probanden, ebenfalls mit eingeschlossen. Diese ist jedoch nur dem Mediziner vorbehalten. Statistiker und Mediziner müssen, während sie die Auswertungen durchführen, eingeloggt sein, um die Daten an einen Server senden zu können.

**3.3.4.2 Anwendungsfälle** Nun folgen die Erläuterungen der Anwendungsfälle für das Szenario *Auswertung durch Mediziner*.

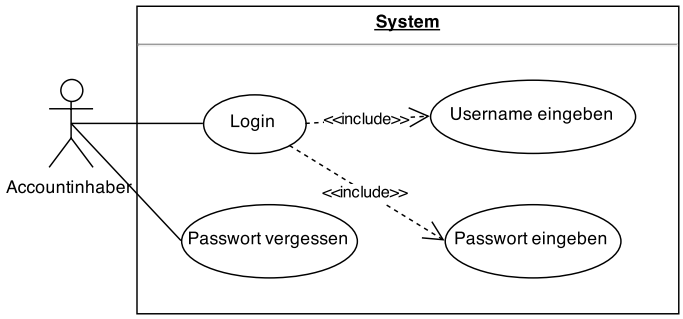



Abbildung 22: Anwendungsfall - Login eines Nutzers

**Login eines Nutzers**

Dieser Anwendungsfall beschreibt, wie ein Benutzer der Weboberfläche (Website) sich in einen geschützten Bereich einloggen kann.

- Akteure:
  Der Akteur ist in diesem Anwendungsfall ein Benutzer der Weboberfläche.
- Standardablauf:
  1. Dem Benutzer wird auf der Weboberfläche eine Loginmaske angezeigt.
  2. Der Benutzer gibt die Logindaten (bestehend aus Nutzerkennung und Passwort) in das vorgegebene Formular ein.
  3. Der Benutzer sendet seine Daten durch die Website an den Server.
  4. Das System validiert die eingegebenen Daten.
- Alternativabläufe:
  - Der Benutzer gibt in Schritt 2 des Standardablaufs falsche Logindaten ein. Dies wird vom System in Schritt 4 erkannt, der Benutzer wird über die falsche Eingabe benachrichtigt und der Standardablauf beginnt wieder bei Schritt 1. Gibt der Benutzer wiederholt (dreimal) falsche Daten ein, wird er aufgefordert, seine Mail Adresse anzugeben, damit er sein Passwort ändern kann.
  - Möglicherweise hat der Benutzer seine Zugangsdaten vergessen. Dann wird er aufgefordert seine Email-Adresse anzugeben, damit er sein Passwort ändern kann.
- Vorbedingungen:
  - Dem Benutzer wurde durch das Institut ein Account bereitgestellt. Der Account besteht mindestens aus einem Passwort, einer eindeutigen Identifikation sowie seiner Email-Adresse.
- Nachbedingungen:
  - Der Benutzer ist eingeloggt und kann den ihm zugeschriebenen geschützten Bereich der Website erreichen.

**Individuelle Auswertung**

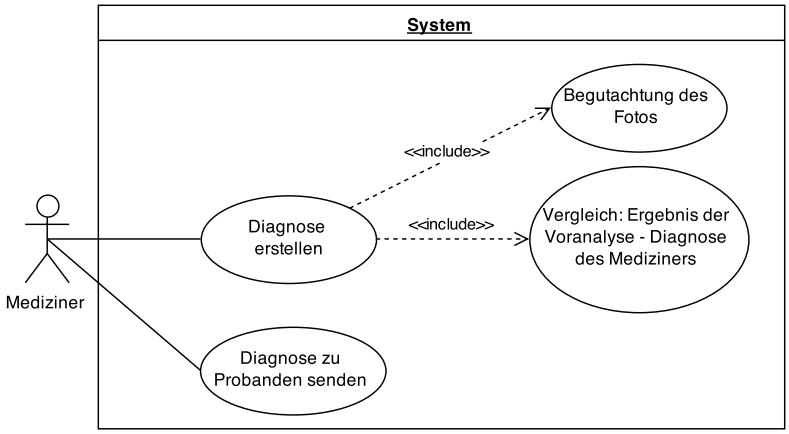



Abbildung 23: Anwendungsfall - Individuelle Auswertung

Dieser Anwendungsfall beschreibt, was passiert, wenn der Mediziner einen einzelnen Teststreifen von einem Probanden auswerten möchte. Dazu begutachtet er zunächst das Foto, welches er ausgewählt hat und vergleicht dieses Ergebnis mit der Voranalyse der mobilen Applikation. Im Anschluss kann er diese Diagnose an den Probanden senden.

- Akteure:
  Der Akteur ist der Mediziner.
- Standardablauf:
  1. Der Mediziner wählt auf der Auswertungsoberfläche die Funktion aus, um sich eine Liste von den Probanden anzeigen zu lassen.
  2. Er wählt aus dieser Liste einen Datensatz aus.
  3. Auf der Auswertungsoberfläche erscheint das Foto des Teststreifens, sowie die Informationen über diesen.
  4. Er überprüft dieses Foto und wählt aus, ob der Test positiv, negativ oder unleserlich ist.
  5. Auf der Auswertungsoberfläche erscheint ein Text, der zu der jeweiligen Option automatisch generiert wurde.
  6. Außerdem erscheint ein Eingabefenster, in dem der Mediziner optional Zusatzinformationen über den Fall eingeben kann.
  7. Der Mediziner wählt die Funktion aus, um seine Diagnose mit dem Ergebnis der Voranalyse zu vergleichen.
  8. Die Auswertungsoberfläche zeigt die Diagnose des Arztes und die der mobilen Applikation an.
  9. Der Mediziner vergleicht die beiden Ergebnisse.
  10. Der Mediziner wählt die Funktion aus, dass sie übereinstimmen.

11. Der Mediziner wählt die Funktion aus, um die Diagnose abzusenden.
12. Auf der Auswertungsoberfläche wird angezeigt, was für eine Diagnose der Mediziner erstellt hat.
13. Der Mediziner hat die Möglichkeit, diese zu verändern.
14. Der Mediziner wählt die Funktion aus, um diese Diagnose an den Probanden zu senden.
15. Auf der Auswertungsoberfläche wird angezeigt, dass das Gutachten versendet wurde.
16. Der Test wird aus der Liste der offenen Teststreifen gelöscht, damit der Mediziner diesen nicht mehrfach überprüft.

- Alternativabläufe:
  - Die Liste der Probanden kann nicht abgerufen werden. In dem Fall muss auf der Auswertungsoberfläche eine aussagekräftige Fehlermeldung über die Ursache erscheinen und eine mögliche Fehlerbehandlung vorgeschlagen werden.
  - Die Diagnose kann nicht gesendet werden. In dem Fall wird dem Mediziner eine aussagekräftige Fehlermeldung angezeigt.
  - Das Programm wird beendet oder das Internet stürzt ab, bevor die Diagnose gesendet werden konnte. In diesem Fall muss die Diagnose zwischengespeichert werden. Bei dem nächsten Start wird dem Mediziner dies mitgeteilt und er kann diese erneut senden.
  - Der Mediziner schließt das Foto ohne eine Diagnose erstellt zu haben.
  - Der Mediziner wählt die Funktion “zurück” aus. In dem Fall muss zwischengespeichert werden, was er bisher angegeben hat.
  - Die Diagnose des Mediziners stimmt nicht mit der Voranalyse der mobilen Applikation überein. In dem Fall wird die Diagnose des Mediziners übernommen und an das Endgerät des Probanden gesendet. Dabei wird dem automatisch generierten Text hinzugefügt, dass es einen Konflikt gab und der Proband einen Arzt aufsuchen sollte. Außerdem wird gespeichert, dass das Institut ebenfalls eine Nachricht erhalten soll.
- Vorbedingungen:
  - Der Mediziner ist eingeloggt.
  - Der Mediziner hat die Daten, also Probanden und Bilder, erhalten.
  - Der Mediziner benötigt eine Internetverbindung zu dem Zeitpunkt, an dem er die Liste der Probanden abruft.
  - Der Mediziner hat zum Zeitpunkt des Sendens eine Internetverbindung.
  - Der Mediziner kann Daten an den Server senden.
- Nachbedingungen:
  - Der Mediziner hat eine Diagnose erstellt, die an das Endgerät des Probanden übertragen wurde.
  - Der Mediziner hat eine Benachrichtigung über den Versand der Diagnose erhalten.
  - Der Mediziner kann im Anschluss die restlichen Funktionen nutzen.
  - Der Test ist nicht mehr in der Liste der offenstehenden Tests erhalten.

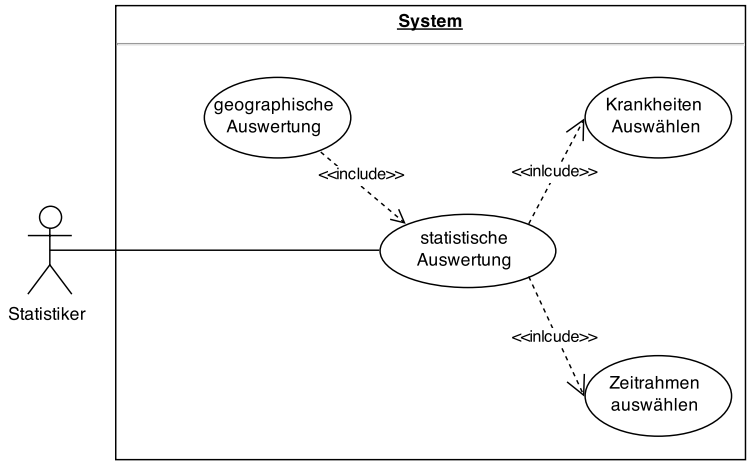



Abbildung 24: Anwendungsfall - Statistische Auswertung

**Statistische Auswertung**

Dieser Anwendungsfall beschreibt die Auswahl und statistische Auswertung von verfügbaren Datensätzen.

- Akteure:
  Der Akteur ist ein Statistiker. Mediziner sind auch Statistiker. Statistiker haben jedoch nicht so viele Rechte wie Mediziner.
- Standardablauf:
  1. Der Statistiker möchte statistische Auswertungen über verschiedene Krankheiten in einem bestimmten Zeitraum erhalten.
  2. Der Statistiker wählt aus einer Liste mit verfügbaren Krankheiten diejenigen aus, die er untersuchen möchte.
  3. Der Statistiker wählt einen Zeitrahmen aus, über den die Testdaten gefiltert werden sollen.
  4. Die Anfrage mit den ausgewählten Parametern wird an den Server geschickt.
  5. Der Server antwortet mit Statistiken (zum Beispiel Anteil Probanden in Stadt- und Landbevölkerung) zu den entsprechenden Daten.
  6. Die Statistiken werden dem Statistiker auf der Auswertungsoberfläche angezeigt.
- Alternativabläufe:
  - In Schritt 1 des Standardablaufs wählt der Benutzer zu viele (keine übersichtliche Darstellung der Daten mehr möglich) oder zu wenige Krankheiten (nicht genügend Daten vorhanden) aus. Darüber muss der Benutzer informiert werden. Solange der Benutzer keine gültige Kombination an Krankheiten ausgewählt hat, werden keine Informationen angezeigt.
  - In Schritt 2 wählt der Benutzer eine ungültige Zeitkombination aus, oder zu dem eingegebenen Zeitrahmen existieren nicht ausreichend Daten. Dann wird der Benutzer über die ungültige Eingabe informiert und aufgefordert, die Eingabe zu korrigieren.
- Vorbedingungen:

- Für den Statistiker existiert ein Account im Institut.
- Der Statistiker ist eingeloggt.
- Der Statistiker besitzt Rechte im System, die ihm erlauben statistische Auswertungen anzuzeigen.

- Nachbedingungen:
  - Die angeforderten Daten wurden dem Statistiker angezeigt.

---

**Geographische Auswertung**

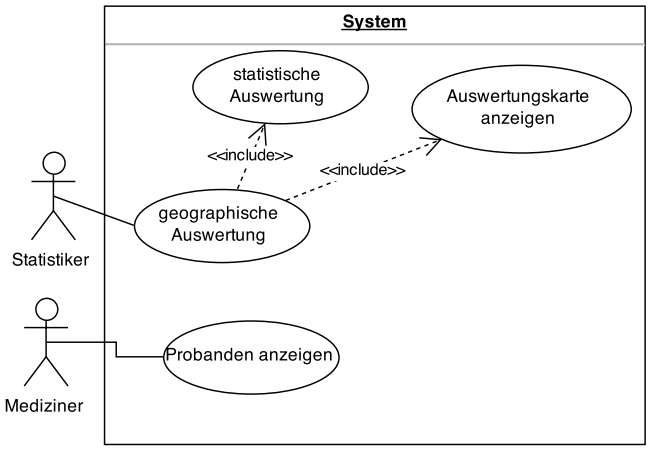



Abbildung 25: Anwendungsfall - Geographische Auswertung

Dieser Anwendungsfall beschreibt welche Informationen dem Statistiker und dem Mediziner angezeigt werden, wenn er eine geographische Auswertung macht.

- Akteure:
  Die Akteure in diesem Anwendungsfall sind Statistiker. Weitere Akteure sind Mediziner, die einem Probanden der ihnen zugewiesenen Region eine Diagnose schicken können.

- Standardablauf:

  1. Basierend auf der Zeit und Krankheitsauswahl aus den Vorbedingungen wird eine interaktive Karte angezeigt. Auf der Karte wird durch eine Heatmap visualisiert, wo welche Krankheit wie stark „ausgebrochen“ ist. Jede Krankheit wird durch eine einzelne Farbe codiert und die Anzahl der Erkrankten durch verschiedene Intensitäten dieser Farbe visualisiert. Es wird eine Legende angezeigt, die diese Codierung erklärt.
  2. Der Benutzer kann die Karte zoomen und verschieben.
  3. Dadurch ändert sich die betrachtete Region der Karte und die Informationen werden vom Server nachgeladen.
  4. Wenn so weit in die Karte gezoomt wurde, dass eine regionale Auswertung möglich ist, werden die Datensätze einzelner Probanden auf der Karte und in einer Liste dargestellt.

5. Ein Mediziner mit Rechten für diese Region kann einzelne Probanden auswählen, um eine Diagnose zu erstellen.

- Alternativabläufe:
  - Für die gewählte Konfiguration sind nicht ausreichend Daten auf dem Server vorhanden. Die Website muss den Statistiker dann über den Datenmangel informieren.
- Vorbedingungen:
  - Der Statistiker hat bereits eine gültige Kombination aus Zeitrahmen und Krankheiten ausgewählt (siehe Anwendungsfall statistische Auswertung 3.3.4.2 auf Seite 32).
  - Dem Mediziner wurde eine Region zugewiesen.
- Nachbedingungen:
  - Der Mediziner ist in der Lage einem ausgewählten Probanden eine Diagnose zu schreiben.
  - Der Statistiker hat geographische Informationen bekommen.

---

## 3.4 Anforderungen

In diesem Kapitel werden die Anforderungen aufgelistet, die sich aus den obigen Anwendungsfällen ergeben. Folgende Kürzel werden hierbei für die Kennzeichnung der Anforderung benutzt:

- FA – funktionale Anforderungen
- NFA – nichtfunktionale Anforderungen

### 3.4.1 Funktionale Anforderungen

Nachfolgend werden die funktionalen Anforderungen, grob gegliedert nach den entsprechenden Szenarien, aufgelistet.

#### 3.4.1.1 Szenario: Proband führt Test durch

FA-1: Die mobile Applikation kann die geographische Position des Probanden bestimmen.

FA-2: Die mobile Applikation nutzt den Ortungsdienst des Smartphones um dessen Position zu bestimmen.

FA-3: Die mobile Applikation kann das Ergebnis des Tests an den Server senden. Der mobile Datenservice oder WLAN wird genutzt um die Daten zu verschicken. Ist keine Verbindung verfügbar, werden die Daten übertragen sobald wieder eine aktive Verbindung besteht.

FA-4: Die mobile Applikation verfügt über einen Bildverarbeitungsalgorithmus, mit dem eine vorläufige Auswertung des Teststreifens vorgenommen werden kann.

FA-5: Die mobile Applikation ist in der Lage zu erkennen, dass der Ladestand des Akkus nicht hoch genug für die Durchführung des Tests ist. Zu Beginn des Tests wird der Ladestand des Akkus abgefragt. Wenn dieser 10% oder weniger beträgt, wird die Testausführung abgebrochen.

FA-6: Die mobile Applikation kann dem Probanden mitteilen, dass der Ladestand des Akkus zu niedrig ist.

FA-7: Die Durchführung des Tests kann bis zu dem Zeitpunkt der Datenübermittlung durch den Probanden abgebrochen werden.

FA-8: Die mobile Applikation ist in der Lage zuvor angelegte Daten eigenständig zu löschen.

FA-9: Der Proband kann die Testdurchführung zu einem selbstgewählten Zeitpunkt starten.

FA-10: Der Proband kann mit der mobilen Applikation ein Foto des Teststreifens aufnehmen. Dazu greift die mobile Applikation auf die Kamera des Smartphones zu, um das Bild aufzunehmen.

FA-11: Die mobile Applikation ist in der Lage ein Foto zwischenzuspeichern.

FA-12: Die mobile Applikation ist in der Lage zu erkennen, ob genügend Speicherplatz für das Foto vorhanden ist. Beim Speichern des Fotos wird geprüft, ob der Speicherplatz ausreicht.

FA-13: Die mobile Applikation kann eine Verbindung zum Server aufbauen.

#### 3.4.1.2 Szenario: Kommunikation Mediziner - Proband

FA-14: Die mobile Applikation des Probanden muss die Daten lokal zwischenspeichern, falls keine Verbindung mit dem Internet besteht, um diese nachträglich versenden zu können.

FA-15: Die mobile Applikation muss eine Funktion enthalten, welche den Test nachträglich versendet, falls zuvor keine Verbindung zum Internet bestand oder die Verbindung während der Übertragung abbricht.

FA-16: Die mobile Applikation muss die Diagnose des Mediziners zum durchgeführten Test vom Server empfangen können.

FA-17: Die mobile Applikation gibt dem Probanden Rückmeldung, sobald eine Diagnose zu einem durchgeführten Test empfangen wurde.

FA-18: Im Falle eines Verbindungsabbruchs zum Server speichert die Auswertungsoberfläche die Daten einer medizinischen Diagnose lokal. Sobald die Verbindung wieder hergestellt ist, wird die Diagnose an den Server übertragen.

FA-19: Sobald der Server von der Auswertungsoberfläche eine Diagnose erhält, ist er in der Lage diese Diagnose der mobilen Applikation des richtigen Probanden zuzuordnen.

#### 3.4.1.3 Szenario: Auswertung durch den Mediziner

FA-20: Die Auswertungsoberfläche muss eine interaktive Karte von Deutschland beinhalten. Diese geographische Auswertungskarte muss für ausgewählte Zeiten und Krankheiten eine Heatmap von Krankheitstests darstellen. Sie wird durch eine entsprechende Legende dokumentiert.

FA-21: Das System muss eine Weboberfläche bereitstellen, die es Accountinhabern ermöglicht, sich durch die Eingabe einer E-Mail-Adresse und eines Passworts einzuloggen.

FA-22: Nach dem Login muss ein Accountinhaber im System als eingeloggt vermerkt werden. Dieser Vermerk kann durch eine Logout-Funktion oder Schließen des Browsers entfernt werden.

FA-23: Die Auswertungsoberfläche stellt dem Mediziner eine Liste mit Probanden bereit, dessen Diagnosen noch ausstehen.

FA-24: Die Auswertungsoberfläche ermöglicht Statistikern, eine statistische Auswertung auf Basis manuell gewählter Daten vorzunehmen. Diese Daten lassen sich zeitlich, geographisch und krankheitsbezogen eingrenzen.

FA-25: Die Auswertungsoberfläche zeigt für einen Benutzer nur die für ihn verfügbaren Funktionen an.

FA-26: Bei der Diagnose eines Tests ermöglicht es die Auswertungsoberfläche dem Mediziner den Test als positiv, negativ oder unlesbar zu erklären.

FA-27: Bei der Erstellung einer Diagnose stellt die Auswertungsoberfläche dem Mediziner Informationen des zugrundeliegenden Tests bereit. Diese Informationen sind das Foto des Teststreifens, die geographischen Daten des Tests und detaillierte Informationen zur Teststreifenvariante.

FA-28: Während der Diagnose ermöglicht es die Auswertungsoberfläche dem Mediziner, seine Diagnose mit der Voranalyse der mobilen Applikation zu vergleichen.

#### 3.4.1.4 Szenario: Institut verwaltet Infrastruktur

FA-29: Die Administrationsoberfläche ermöglicht die Erstellung eines Benutzeraccounts anhand einer E-Mail-Adresse, einem vollständigen Namen und weitergehenden Kontaktdaten.

FA-30: Der Server verfügt über ein Rechtesystem, welches jedem Nutzer mindestens eine Benutzergruppe mit entsprechenden Rechten zuweist. Es existieren die Benutzergruppen der Mediziner, der Systemadministratoren und die der Statistiker.

FA-31: Der Server beherrscht die Speicherung existierender Accounts. Es existiert eine zentrale Datenhaltung aller Accounts und ihrer Daten.

FA-32: Der Server prüft nach Entgegennahme von Accountdaten diese Daten auf Syntax und Semantik. Duplikate müssen automatisch erkannt werden.

FA-33: Das zu entwickelnde System muss bei Auftreten eines Fehlers eine passende Rückmeldung an den Nutzer geben.

FA-34: Jeder Accountinhaber kann bei Erstellung seines Accounts automatisiert per E-Mail über die Accounterstellung informiert werden.

FA-35: Die Administrationsoberfläche muss die Möglichkeit bieten, Benutzeraccounts zu durchsuchen. Als Ergebnis der Suche wird eine Liste mit allen den Suchparametern entsprechenden Accounts angezeigt.

FA-36: Die Administrationsoberfläche ermöglicht die Bearbeitung und das Entfernen von bestehenden Benutzeraccounts.

FA-37: Die Administrationsoberfläche bietet die Möglichkeit, neue Teststreifenvarianten anzulegen.

FA-38: Der Server beherrscht die Speicherung von Teststreifenvarianten. Es existiert eine zentrale Datenhaltung aller Teststreifenvarianten und ihrer Daten.

FA-39: Das System prüft nach Entgegennahme von Teststreifendaten diese auf syntaktische und semantische Korrektheit. Duplikate werden erkannt.

FA-40: Die Administrationsoberfläche ermöglicht die Bearbeitung von bestehenden Teststreifenvarianten.

FA-41: Die Administrationsoberfläche bietet die Möglichkeit, bestehende Teststreifenvarianten zu durchsuchen. Ergebnis der Suche ist eine Liste an Teststreifenvarianten, welche den Suchparametern entsprechen.

FA-42: Die Auswertungsoberfläche ermöglicht einem eingeloggten Benutzer, seine eigenen Accountdaten zu ändern. Eingegebene Daten werden auf Semantik und Syntax überprüft.

### 3.4.2 Nichtfunktionale Anforderungen

Nachfolgend werden die nichtfunktionalen Anforderungen aufgelistet.

#### 3.4.2.1 Zuverlässigkeit

NFA-1: Der Server muss über eine zeitliche Verfügbarkeit von über 97.5% verfügen.

NFA-2: Die mobile Applikation darf durch den Probanden nicht zum Absturz gebracht werden können.

NFA-3: Der Server muss 10.000 gleichzeitigen Übermittlungen pro Sekunde von Testergebnissen standhalten.

NFA-4: Der Server muss 5000 gleichzeitig eingeloggten Accountinhabern standhalten.

#### 3.4.2.2 Usability und Look and Feel

NFA-5: Sowohl die Administrationsoberfläche als auch die Auswertungsoberfläche müssen benutzerfreundlich und einfach gestaltet sein. Etwaige Benutzergruppen können die Oberfläche intuitiv innerhalb von zehn Einarbeitungsminuten verstehen und fehlerfrei bedienen.

NFA-6: Textmeldungen müssen für angesprochene Nutzer verständlich formuliert sein. Fehlermeldungen beinhalten einen eindeutigen Hinweis auf die Art und den Ursprung des Fehlers.

NFA-7: Die mobile Applikation benutzt verständliche Sprache im Sinne von VDE 0039-1.

NFA-8: Die Benutzerfreundlichkeit aller Oberflächen muss durch die Durchführung einer Feldstudie bestätigt werden.

#### 3.4.2.3 Leistung und Effizienz

NFA-9: Die Datenhaltung des Servers muss persistent und performant gelöst sein, d.h. Anfragen an eine entsprechende Datenbank dürfen maximal zwei Sekunden Bearbeitungszeit benötigen.

NFA-10: Suchfunktionen auf persistenten Daten werden innerhalb von maximal fünf Sekunden durchgeführt.

NFA-11: Das Ergebnis der Voranalyse muss dem Probanden nach maximal zehn Sekunden mitgeteilt werden.

NFA-12: Sowohl die Administrationsoberfläche als auch die Auswertungsoberfläche werden innerhalb von maximal drei Sekunden dargestellt. Benutzereingaben werden direkt entgegengenommen.

NFA-13: Die mobile Applikation muss so ressourcenschonend arbeiten, dass sie auch auf low-cost Smartphones ausführbar ist.

#### 3.4.2.4 Wartbarkeit, Änderbarkeit

NFA-14: Die mobile Applikation wird zunächst in deutscher Sprache veröffentlicht und soll die Möglichkeit besitzen, zu einem späterem Zeitpunkt um andere Sprachen erweitert zu werden.

NFA-15: Sämtliche Dokumentationen und Kommentare im Quellcode des zu entwickelnden Systems sind auf Englisch verfasst. Interne Namen von Objekten im Quellcode sind in Englisch formuliert.

NFA-16: Sämtlicher Quellcode des zu entwickelnden Systems ist ausreichend dokumentiert. Ein ausgebildeter Softwareentwickler muss betreffenden Code innerhalb einer angemessenen Einarbeitungszeit verstehen.

NFA-17: Die Dokumentation der einzelner Codeabschnitte erfolgt direkt an entsprechender Stelle im Quellcode. Die Dokumentation muss durch Doxygen erzeugt werden können.

#### 3.4.2.5 Portierbarkeit und Übertragbarkeit

NFA-18: Sowohl die Administrationsoberfläche als auch die Auswertungsoberfläche unterstützen folgende Webbrowser: Google Chrome ab Version 42, Mozilla Firefox ab Version 37, Microsoft Internet Explorer ab Version 11 und Apple Safari ab Version 8.

NFA-19: Sowohl die Administrationsoberfläche als auch die Auswertungsoberfläche unterstützen die mobilen Versionen der vorher genannten Browser.

NFA-20: Die mobile Applikation muss für Android und iOS bereitgestellt werden. Hierdurch wird eine Abdeckung von über 96% aller Smartphones in Deutschland erreicht.

NFA-21: Der Server wird mit einem Linux-Betriebssystem bereitgestellt.

#### 3.4.2.6 Sicherheitsanforderungen

NFA-22: Die Datenhaltung jeglicher Daten darf für unbefugte Personen nicht zugänglich sein.

NFA-23: Die Kommunikation zwischen der mobilen Applikation und dem Server ist nach aktuellen Standards des Bundesamts für Sicherheit verschlüsselt.

NFA-24: Jede Benutzergruppe darf nur die für sie bestimmten Funktionen einsehen und nutzen.

#### 3.4.2.7 Korrektheit

NFA-25: Daten, die zwischen Medizinern und Probanden ausgetauscht werden, sind stets auf Vollständigkeit geprüft.

NFA-26: Jegliche vom Nutzer entgegengenommene Daten sind auf Syntax- und Semantikfehler geprüft.

## 3.5 Risikoanalyse

Während der Planungsphase wurde eine Risikoanalyse durchgeführt, um etwaigen Problemen, welche zu Verzögerungen innerhalb des Projektablaufs oder gar Scheitern des Projekts führen können, frühzeitig entgegen zu wirken. Tabelle 5 auf der nächsten Seite listet diese Probleme unter Angabe einer Schadenshöhe, der erwarteten Wahrscheinlichkeit des Auftretens (Erklärung in Tabelle 3), sowie einem Risikowert (Tabelle 4 auf der nächsten Seite), welcher sich aus der Schadenshöhe und der Wahrscheinlichkeit des Auftretens ergibt. In Tabelle 6 auf Seite 41 werden Gegenmaßnahmen erläutert, welche dazu dienen das Risiko der zu erwartenden Probleme und deren Schadensauswirkungen zu mindern. Abschließend erfolgt die Risikobewertung innerhalb einer Analysematrix vor (Abbildung 26 auf Seite 42) und nach (Abbildung 27 auf Seite 42) den Maßnahmen.
In der Bewertung werden folgende Skalierungen für das Risiko und die Wahrscheinlichkeit verwendet.

| | | |
|---|---|---|
| 1 | unwahrscheinlich | $0\% \leq x \leq 10\%$ |
| 2 | fernliegend | $10\% < x \leq 35\%$ |
| 3 | gelegentlich | $35\% < x \leq 65\%$ |
| 4 | naheliegend | $65\% < x \leq 90\%$ |
| 5 | wahrscheinlich | $90\% < x \leq 100\%$ |

Tabelle 3: Skalierung der Eintrittswahrscheinlichkeit

| | |
|---|---|
| 1 | sehr gering |
| 2 | gering |
| 3 | mittel |
| 4 | hoch |
| 5 | sehr hoch |

Tabelle 4: Skalierung Risiko und Schadenswirkung

### 3.5.1 Risikobeschreibung

| Risiko-Nr. | Risikobeschreibung | Schadens-höhe | W.keit des Auftre-tens | Risiko |
|---|---|---|---|---|
| 1 | Der Bildverarbeitungsalgorithmus ist nicht effizient genug, um die maximale Laufzeit von 10 Sekunden auf Low-End-Smartphones einhalten zu können. | 1 | 2 | 2 |
| 2 | Die Datenbank ist nicht leistungsstark genug, um ein hohes Aufkommen an parallelen Anfragen zu bearbeiten. | 3 | 2 | 6 |
| 3 | Die Farberkennung des Bildverarbeitungsalgorithmus hat eine zu hohe Fehlerquote, sodass die Auswertung der Voranalyse in den meisten Fällen unbrauchbar ist. | 5 | 2 | 10 |
| 4 | Die Rückmeldung durch einen Mediziner bzgl. des validierten Testergebnisses schlägt fehl. | 5 | 1 | 5 |
| 5 | Das Verbinden zum Server zwecks Datenübermittlung schlägt aus unbekannten Gründen fehl. | 1 | 1 | 1 |
| 6 | Die Verschlüsselungsmethodik ist nicht effektiv, wodurch die Sicherheit der Daten nicht mehr gewährleistet werden kann. | 4 | 2 | 8 |
| 7 | Die Auswertungsoberfläche ist nicht ausreichend vor Fremdzugriffen geschützt. | 5 | 3 | 15 |

Tabelle 5: Risikoanalyse

### 3.5.2 Maßnahmenkatalog

Im Zuge des Maßnahmenkatalogs werden Maßnahmen erläutert, welche die erkannten Risiken dahingehend optimieren sollen, dass die Auftrittswahrscheinlichkeit auf ein akzeptables Minimum reduziert wird.

| Risiko - Nr. | Gegenmaßnahme | Schadens-höhe | W.keit des Auftre-tens | Rest-risiko |
|---|---|---|---|---|
| 1 | Laufzeitanalyse, frühzeitige Tests sowie interne Code-Audits um zeitig Optimierungen vornehmen zu können | 1 | 1 | 1 |
| 2 | Skalierbarkeit der Datenbank durch Datenbank-Clustering nutzen sowie Lasttestdurchführungen auf der Datenbank. | 3 | 1 | 3 |
| 3 | Testspektrum auf möglichst viele Geräte/Kameras ausbreiten um Fehler/Unzulänglichkeiten im Algorithmus frühzeitig zu erkennen | 5 | 1 | 5 |
| 4 | Hohe Erreichbarkeit des Servers durch Load-Balancing gewährleisten | 5 | 1 | 5 |
| 6 | Frühzeitige und kontinuierliche Übergabe des Codes an den Kunden um externe Code-Audits zu ermöglichen. | 4 | 1 | 4 |
| 7 | Interne und externe Hacker-Gruppen gezielt darauf ansetzen, dass System zu umgehen. | 5 | 2 | 10 |

Tabelle 6: Gegenmaßnahmen

## 3.6 Entwicklungszyklus

Bei der Durchführung des Entwicklungsprozesses werden Konzepte der agilen Softwareentwicklung maßgeblich sein. Dafür werden Prinzipien des Manifestes der agilen Softwareentwicklung aus dem Jahre 2001 (Manifesto for Agile Software Development[1]) angewandt.

- Frühe und kontinuierliche Auslieferung von Software innerhalb weniger Wochen (Sprint-Zyklus).
- Späte Anforderungsänderungen werden entgegengenommen.
- Fachexperten und Entwickler müssen während des Projektes zusammenarbeiten.
  Dafür werden sich unsere Entwickler zu Experten in notwendigen Bereichen fortbilden.
- Informationen an und innerhalb der/des Entwicklungsteams werden in persönlichen Treffen übermittelt.
- Selbstorganisierte Teams.
- Reflektionen in regelmäßigen Abständen.

[1] vgl. http://agilemanifesto.org/iso/de/principles.html

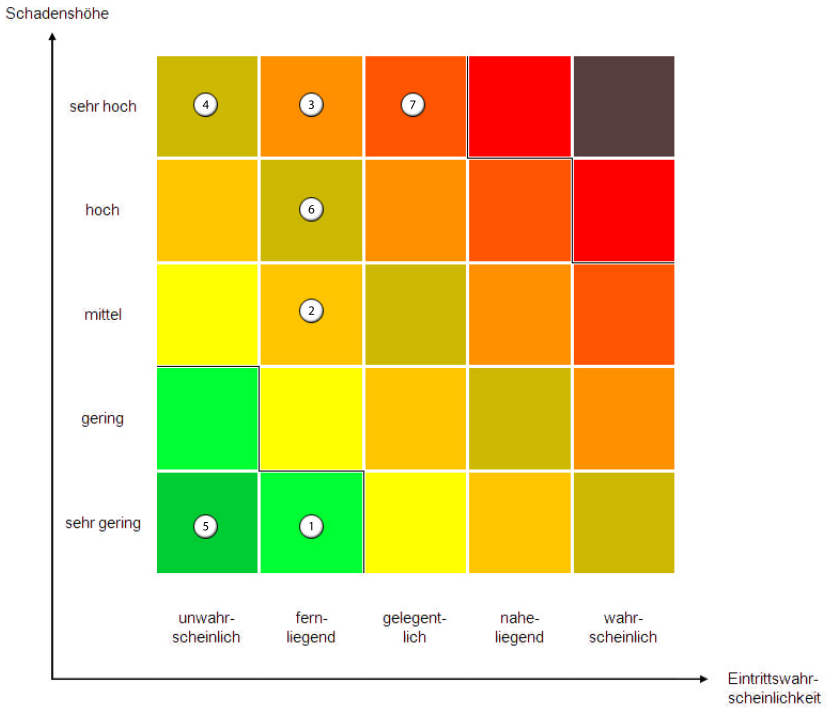



Abbildung 26: Risikomatrix

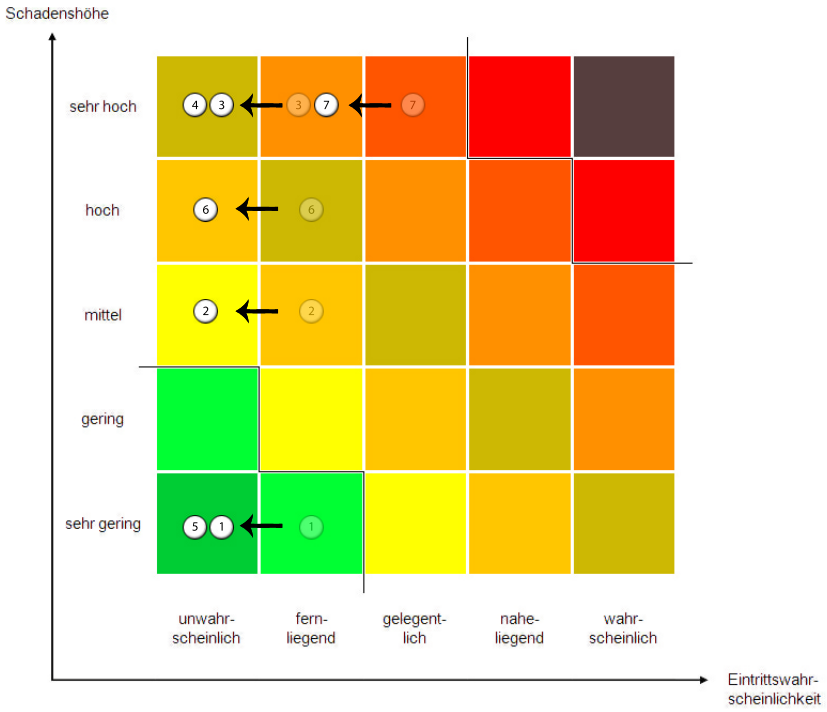



Abbildung 27: Risikomatrix nach Anwendung des Maßnahmenkatalogs

Aus den Prinzipien ergeben sich nachfolgende Kontinuitäten. Das zu entwickelnde Gesamtsystem wird in mindestens zwei Untersysteme (mobile Applikation und Server) geteilt, aus welchen sich mehrere Unteraufgaben ergeben. Diese Aufgaben werden dann in vier Wochen andauernden Intervallen (auch Sprints genannt) bearbeitet. Des Weiteren wird innerhalb des Projektes das Konzept der kontinuierlichen Integration angewandt, sodass am Ende eines jeden Sprints ein lauffähiges Teilprodukt entsteht. Dieses Teilprodukt kann dem Kunden zur Überprüfung von Anforderungen, sowie zur Überprüfung des Produktes selbst übergeben werden.

## 3.7 Lieferumfang

Dem Kunden werden nach Abschluss des Projekts folgende Produkte geliefert:

- Jeweils eine funktionsfähige, auf Android und auf iOS ausführbare, mobile Applikation.
- Die funktionsfähige und ausführbare Serveranwendung.
- Die funktionsfähige und ausführbare Auswertungsoberfläche.
- Die funktionsfähige und ausführbare Administrationsoberfläche.
- Sämtlicher finaler Quellcode der mobilen Applikation, der Serveranwendung, der Auswertungsoberfläche und der Administrationsoberfläche.
- Sämtliche finale Quellcode-Dokumentation.
- Der komplette Abschlussbericht, welcher unter anderem das Lastenheft, die Sprintdokumentationen, die Meeting-Protokolle und die Feldstudien umfasst.
- Bestimmte Mitarbeiter haben sich zu Experten für Teilbereiche des Projekts spezialisiert und stehen dem Kunden als Ansprechpartner zur Verfügung.
- Eine Inbetriebnahmeanleitung für das gesamte entwickelte System.
- Ein lauffähiges Server- und Auswertesystem.

# 4 Sprintdokumentationen

Dieses Kapitel enthält die einzelnen Sprints der Projektgruppe. Ein Sprint bezeichnet einen Projektabschnitt von vier bis sechs Wochen, in dem die Projektgruppe mehrere Arbeitspakete bearbeitet hat. Die folgenden Unterkapitel sind im Laufe der Projektlaufzeit entstanden. Sind inkrementiell Verbesserungen zur Struktur dieser Teilabschnitte entstanden. Da die Verfassung des Lastenheftes den ersten Sprint des Projekts darstellt, ist der erste Sprint dieses Kapitels Sprint 2.

## 4.1 Sprint 2

Der zweite Sprint dieses Projekts hat als Ziel einen ersten Prototypen für die mobile Applikation zu erstellen, einen vorläufigen Server einzurichten und ein Web-Frontend zu designen. Die ermöglicht einen vorläufigen Gesamtablauf des Testprozesses durchzuführen. Dadurch wird die Grundlage für die kontinuierliche Integration geschaffen, welche sicherstellt, dass die Software immer in einem ausführbaren Zustand ist.
Für diesen Sprint wird ein horizontales Vorgehen gewählt, wodurch die Realisierung einer optimalen Funktion nicht im Vordergrund stand sondern der Fokus auf mehreren Bereichen lag. Die folgende Abbildung zeigt den sequentiellen Ablauf aller Arbeitspakete in Relation zueinander.

### 4.1.1 Sprintplanung

Im folgenden wird die Planung des zweiten Sprints (13.05. bis 09.06., vier Wochen) genauer aufgezeigt. Dabei wird ein besonderes Augenmerk auf die folgende Punkte gelegt:

- Fokus des Sprints
- Arbeitspakete und Planung
- Planung der einzelnen Ressourcen

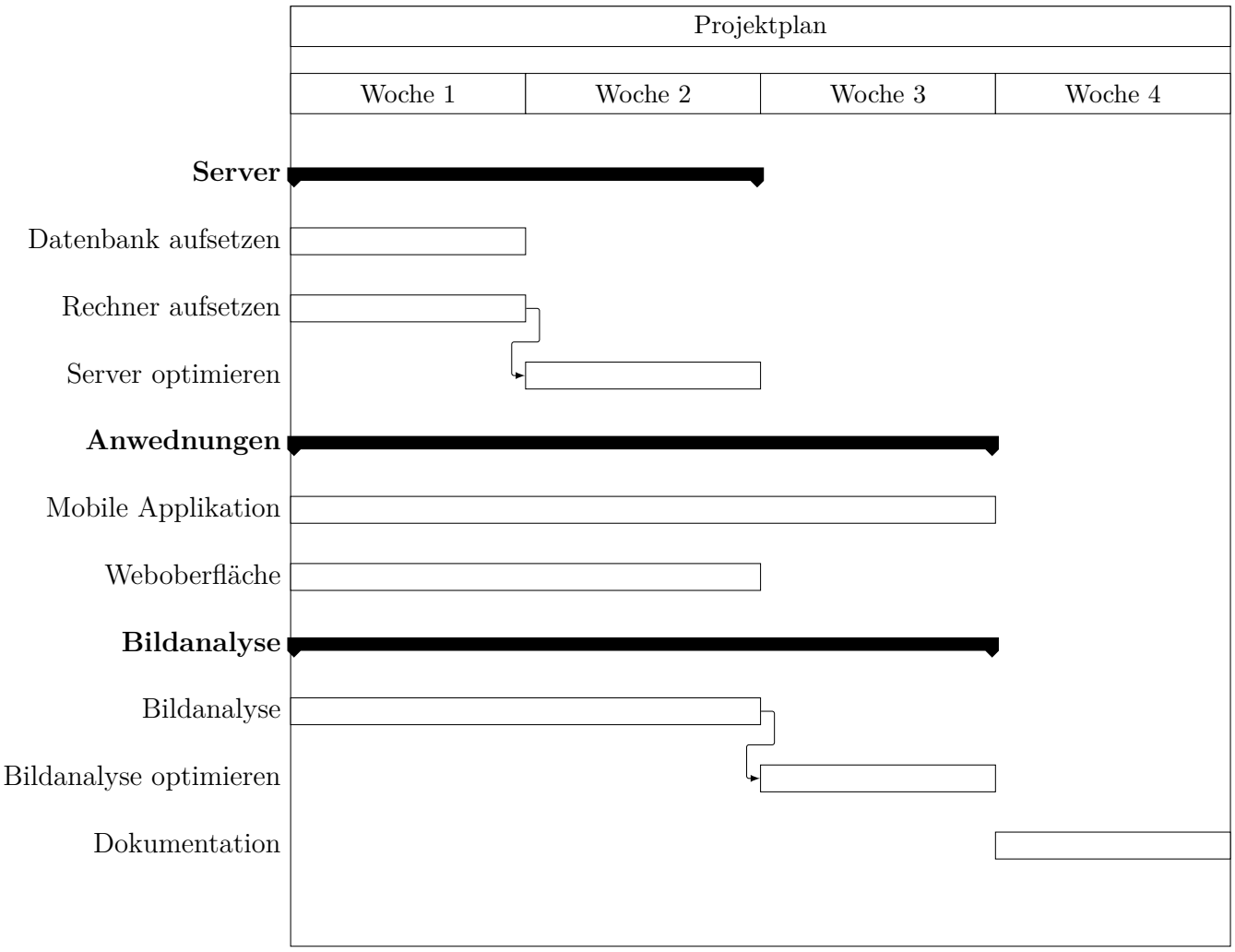



Abbildung 28: Gantt-Diagramm der Planung des zweiten Sprints

#### 4.1.1.1 Fokus des Sprints

Für diesen Sprint wurde ein horizontales Vorgehen gewählt. In diesem sollte ein Prototyp entworfen werden, welcher die Grundfunktionalitäten für die mobilen Applikationen, die Weboberfläche, die Datenbank und die Bildverarbeitung enthält. Durch dieses Vorgehen soll eine Basis geschaffen werden, auf der die zukünftige Arbeit der Projektgruppe aufbauen kann.

**4.1.1.2 Arbeitspakete und Planung** Aus dem Ziel, eine breite Basis erster Prototypen zu schaffen, ergeben sich folgende Arbeitspakete:

- Rechner aufsetzen
- Datenbank einrichten
- Schnittstelle mobile Applikation - Datenbank
- Schnittstelle Auswertungsoberfläche - Datenbank
- Bildanalyse
- Bildanalyse Optimierung
- Server optimieren

#### 4.1.1.3 Planung der Ressourcen

Im folgenden ist die Personalplanung dargestellt. Sie gibt Auskunft darüber, welches Teammitglied an welchem Arbeitspaket gearbeitet hat.

- Raphael Kappes: Bildanalyse (1,5 Wochen), Bildanalyse Optimierung (1 Woche)
- Christian Sandmann: Schnittstelle Auswertungsoberfläche - Datenbank (2,5 Wochen)
- Timo Schlömer: Seminararbeit (6 Wochen)
- Timo Raß: Schnittstelle Auswertungsoberfläche - Datenbank (2,5 Wochen)
- Nicolas Koch: Rechner aufsetzen (0,5 Wochen), Datenbank einrichten (1 Woche), Server optimieren (1 Woche)
- Kevin Sandermann: Rechner aufsetzen (0,5 Wochen), Datenbank einrichten (1 Woche), Server optimieren (1 Woche)
- Sebastian Horwege: Datenbank einrichten (1 Woche), Schnittstelle Auswertungsoberfläche - Datenbank (1,5 Wochen)
- Danny Fonk: Bildanalyse (1,5 Wochen), Bildanalyse Optimierung (1 Woche)
- Jan Philipp Stubbe: Seminararbeit (6 Wochen)
- Christoph Ressel: Schnittstelle mobile Applikation - Datenbank (3 Wochen)
- Daniel Wegmann: Schnittstelle mobile Applikation - Datenbank (3 Wochen)

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und deren Dokumentationen skizziert.

### 4.1.2 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.1.3 Rechner aufsetzen

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Es wird ein Rechner/Server benötigt, auf welchem alle Datenbankdienste und die Weboberflächen bereitgestellt werden können.
- Arbeitspaketbeschreibung:
  Um beim zu entwickelnden System die Seite des Instituts zu simulieren, muss ein Server existieren, auf welchem die Datenhaltung und Serverlogik gehostet werden. Aufgabe dieses Servers ist es, Anfragen von Clients, also den Mobile Applikationen entgegen zunehmen und zu verarbeiten. Des Weiteren agiert der Server als Entwicklungsinstanz der Projektgruppe, auf welcher Entwicklungsarbeiten der Gruppe eingesetzt werden können. Da das Aufsetzen und Konfigurieren dieses Systems eine Grundlage für viele weitere Arbeitspakete darstellt, muss die Umsetzung zeitnah erfolgen. Als Deadline ist daher der 20.05.2015 vorgesehen.

- Vorbedingung:
  Es existiert ein Rechner im Projektgruppenraum, welcher als Server genutzt werden kann.
- Nebenbedingungen:
  -
- Nachbedingungen:
  - Debian ist auf dem Serverrechner installiert.
  - Der Rechner kann aus dem Netzwerk der Universität unter einer IP-Adresse erreicht werden.
  - Ein Webserver ist auf dem Rechner installiert.
  - Der Webserver ist als Daemon konfiguriert, der bei Systemstart automatisch ausgeführt wird.
- Aufwand:
  Eine Woche
- Personen:
  - Nicolas Koch
  - Kevin Sandermann

#### 4.1.4 Datenbank einrichten

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  In diesem Arbeitspaket soll eine erste Funktionalität zur Speicherung von Teststreifendatensätzen auf dem Server bereitgestellt werden. Zusätzlich sollen die Grundlagen für eine REST Schnittstelle zwischen der Datenbank, der Website und der mobilen Applikation bereitgestellt werden.
- Arbeitspaketbeschreibung:
  Die Speicherung der Teststreifen erfolgt in einer relationalen Datenbank. Diese wird durch die Software Postgressql auf dem Server bereitgestellt. Die Datenbank orientiert sich nach einzelnen Testergebnissen und speichert grundlegende Informationen zu diesen: Die Metadaten zu einem Test beinhalten Koordinaten, berechnetes Ergebnis und Zeitpunkt der Übertragung. Zusätzlich wird das aufgenommene Foto in der Datenbank hinterlegt. Es soll außerdem möglich sein, Testergebnisse Benutzern zuzuordnen. Die detaillierte Repräsentation der Daten in der Datenbank ist Abbildung 29 zu entnehmen.

  Die Deadline dieses Arbeitspakets ist der 20.5.2015.

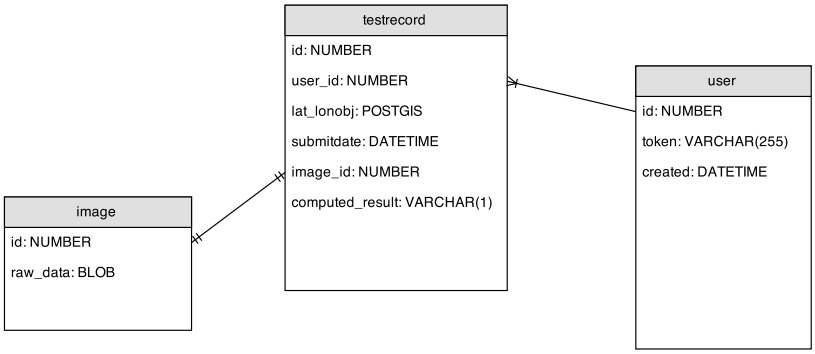



Abbildung 29: UML Diagramm für das Datenbankschema

- Vorbedingung:
  Server ist eingerichtet und betriebsbereit.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Eine Datenbank auf dem Server, sowie eine grundlegende Representational State Transfer (REST)-Schnittstelle für die Weboberfläche und die Mobile Applikation.
- Aufwand:
  Drei Wochen
- Personen:
  - Sebastian Horwege (Datenbank / Schnittstelle)
  - Kevin Sandermann (Schnittstelle)
  - Nicolas Koch (Schnittstelle)

### 4.1.5 Server optimieren

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket baut auf den Arbeitspaketen „Rechner aufsetzen“ (siehe: Kapitel 4.1.3 auf Seite 45) und „Datenbank einrichten“ (siehe: Kapitel 4.1.4 auf der vorherigen Seite) auf. Es dient dazu, den bestehenden Server um zusätzliche Funktionen zu erweitern.
- Arbeitspaketbeschreibung:
  Aus dem Arbeitspaket „Schnittstelle mobile Applikation - Datenbank“ (siehe: Kapitel 4.1.9 auf Seite 55) ist eine funktionsfähige Geschäftslogik entstanden, die sich aus einer Datenbank und einer REST-Schnittstelle zusammensetzt. Diese müssen im vorliegenden Arbeitspaket um folgende Elemente erweitert werden:
  - Authentifizierungsmechanismus mithilfe eines eindeutigen Tokens
  - Integration mit der Weboberfläche aus „Schnittstelle mobile Applikation - Datenbank“ (siehe: Kapitel 4.1.9 auf Seite 55)
  - Dokumentation der entstandenen Application Programming Interface (API)s und des Quellcodes

  - Vergrößerung der Testdatenbasis

  Durch dieses Arbeitspaket wird damit die Grundlage für eine erhöhte Sicherheit des Systems gelegt, damit zuküftige Audits bestanden werden können. Des weiteren wird die Testbarkeit und Wartbarkeits des Systems verbessert.

  Die Deadline dieses Arbeitspakets ist der 27.5.2015.

- Vorbedingung:
  Das Arbeitspaket „Datenbank einrichten“ wurde abgeschlossen.

- Nebenbedingungen:
  -

- Nachbedingungen:

  - Die Serverlogik ist in der Lage, Verbindungen zu authentifizieren.
  - Die Auswertungsoberfläche wird unter derselben Internet Protocol (IP) wie die REST-Schnittstelle bereitgestellt.
  - Jede Funktion des REST-APIs ist hinreichend dokumentiert
  - Die Testdatenbasis ist groß genug, um clientseitige Funktionalitäten, wie z. B. das dynamische Nachladen der Liste mit den durchgeführten Test oder die Heatmap-Funktionalität testen zu können.

- Aufwand:
  Zwei Wochen

- Personen:

  - Nicolas Koch
  - Kevin Sandermann

#### 4.1.6 Dokumentation

Diese Dokumentation bezieht sich auf die Arbeitspakete „Rechner aufsetzen“, „Datenbank einrichten“ und „Server optimieren“. Da diese Arbeitspakete logisch aufeinander aufbauen und chronologisch nacheinander abgearbeitet wurden, ist es an dieser Stelle sinnvoll sie in einer Paketdokumentation zu beschreiben.

**Ablauf** Die durch die Arbeitspakete abgedeckten Anwendungsfälle sind szenarioübergreifend. Daraus ergeben sich folgende zu lösende funktionale Anforderungen:

**Szenario: Proband führt Test durch**

FA-3: Die mobile Applikation kann das Ergebnis des Tests an den Server senden. Der mobile Datenservice oder Wireless Local Area Network (WLAN) wird genutzt um die Daten zu verschicken. Ist keine Verbindung verfügbar, werden die Daten übertragen sobald wieder eine aktive Verbindung besteht.

FA-13: Die mobile Applikation kann eine Verbindung zum Server aufbauen.

**Szenario: Kommunikation Mediziner - Proband**

FA-20: Sobald der Server von der Auswertungsoberfläche eine Diagnose erhält, ist er in der Lage diese Diagnose der mobilen Applikation des richtigen Probanden zuzuordnen.

**Szenario: Institut verwaltet Infrastruktur**

FA-40: Der Server beherrscht die Speicherung von Teststreifenvarianten. Es existiert eine zentrale Datenhaltung aller Teststreifenvarianten und ihrer Daten.

Neben diesen sich bereits aus den Arbeitspaketbeschreibungen ergebenden funktionalen Anforderungen sind weiterführend unten aufgezählte nichtfunktionale Anforderungen für den Prototypen zu beachten. Da Die Arbeitspakete die Implementierung eines Prototypen vorsehen, sind diese nichtfunktionalen Anforderungen gerade in Bezug auf konkrete Belastungs- und Verfügbarkeitskennzahlen nicht verbindlich.

**Zuverlässigkeit**

NFA-1: Der Server muss über eine zeitliche Verfügbarkeit von über 97.5% verfügen.

NFA-3: Der Server muss 10.000 gleichzeitigen Übermittlungen pro Sekunde von Testergebnissen standhalten.

**Leistung und Effizienz**

NFA-9: Die Datenhaltung des Servers muss persistent und performant gelöst sein, d.h. Anfragen an eine entsprechende Datenbank dürfen maximal zwei Sekunden Bearbeitungszeit benötigen.

NFA-10: Suchfunktionen auf persistenten Daten werden innerhalb von maximal fünf Sekunden durchgeführt.

**Wartbarkeit, Änderbarkeit**

NFA-15: Sämtliche Dokumentationen und Kommentare im Quellcode des zu entwickelnden Systems sind auf Englisch verfasst. Interne Namen von Objekten im Quellcode sind in Englisch formuliert.

NFA-16: Sämtlicher Quellcode des zu entwickelnden Systems ist ausreichend dokumentiert. Ein ausgebildeter Softwareentwickler muss betreffenden Code innerhalb einer angemessenen Einarbeitungszeit verstehen.

NFA-17: Die Dokumentation der einzelner Codeabschnitte erfolgt direkt an entsprechender Stelle im Quellcode. Die Dokumentation muss durch Doxygen erzeugt werden können.

**Sicherheitsanforderungen**

NFA-21: Die Datenhaltung jeglicher Daten darf für unbefugte Personen nicht zugänglich sein.

NFA-23: Jede Benutzergruppe darf nur die für sie bestimmten Funktionen einsehen und nutzen.

**Korrektheit**

NFA-24: Daten, die zwischen Medizinern und Probanden ausgetauscht werden, sind stets auf Vollständigkeit geprüft.

NFA-25: Jegliche vom Nutzer entgegengenommene Daten sind auf Syntax- und Semantikfehler geprüft.

Ziel der Arbeitspakete war es, im PG-Raum einen Serverrechner aufzusetzen, welcher intern sowie extern erreichbar ist. Auf diesem Server soll die für den Prototyp im Git-Repository medic-prototype-server entwickelte Server-Logik hinterlegt sein. Ferner war es Ziel eine Datenbank zur Verfügung zu stellen, in welcher Testergebnisse gespeichert sind.
Zunächst wurde ein im PG-Raum befindlicher Rechner als System ausgewählt und als Betriebssystem debian installiert. Anschließend wurde der Rechner im PG-Raum ans Netzwerk angeschlossen und konfiguriert. Der Rechner ist von intern und extern über SSH erreichbar. Bis zu dieser Stelle gab es keinerlei Probleme, das Arbeitspaket „Rechner aufsetzen“ wurde erfolgreich abgeschlossen.
Nachdem der Serverrechner nun zur Verfügung stand, konnte eine Datenbank und ein Applikationsserver installiert und konfiguriert werden. Anschließend wurde die Server-Logik des Prototyp im Git-Repository medic-prototype-server entwickelt und auf den vorher konfigurierten Applikationsserver installiert. Um diese Installation zu automatisieren, wurde außerdem Jenkins auf dem Server installiert, sodass ein neu veröffentlichter Stand jederzeit auf dem Applikationsserver installiert wird.
Neben der Datenbasis stellt der Serverrechner außerdem eine Schnittstelle zur Verfügung, mithilfe welcher Clients mit dem System kommunizieren können. Clients können dabei sowohl eine Mobile Applikation als auch ein Webbrowser sein. Diese werden ebenso wie die Serverlogik von der Jenkinsinstallation ins Gesamtsystem integriert.

**Technologien**

**Debian-Linux** wird als Betriebssystem verwendet, da es ressourcenschonend und kostenlos ist.

**PostgreSQL** dient als Datenbanktechnologie eingesetzt. Sie ist über SQL ansprechbar und ausreichend dokumentiert.

**Apache Tomcat** wird als Servercontainer eingesetzt, um den in Java implementierten Server auszuführen.

**Apache HTTP Server** ist eine gängige HTTP-Server-Variante

**Jenkins** wird eingesetzt, um das Paradigma der kontinuierlichen Integration umzusetzen.

**Spring Framework** ist ein für Java zur Verfügung stehendes Framework, um Server-Schnittstellen schnell und unkompliziert zu implementieren.

**Systembeschreibung** In diesem Kapitel wird beschrieben, mit welchen Tools und auf welche Art und Weise das beschriebene System entwickelt wurde.
Während auf dem Debian-Server stets eine lauffähige Version installiert ist, benutzt jeder Entwickler zur Programmierung einen lokalen Tomcat-Server. Hingegen wird zu Entwicklungszwecken stets die auf dem Server verwaltete Datenbank verwendet.
Als Werkzeug der Entwicklung benutzte die Gruppe die Spring Tool Suite, eine speziell für Spring entworfene Entwicklungsumgebung. Da es sich um ein Java Projekt handelt, wird für die Dependency-Verwaltung Maven benutzt.
Es wurde der Quellcode, die gesamte Serverarchitektur und die Schnittstellen ausgiebig dokumentiert. Diese Dokumentationen finden sich an den folgenden Orten:

- Wiki des Git-Repositories `medic-general`
- README-Datei des Git-Repositories `medic-prototype-server`
- JavaDoc des Server-Projektes (Doxygen-fähig)
- `http://134.106.47.158:8080/medic/`

**Evaluation** Die Evaluation des Systems geschah kontinuierlich durch den Continuous-Integration-Ansatz bei der Entwicklung. Da sowohl die mobile Applikation als auch die Weboberfläche permanent auf die Funktionen des Servers zurückgreifen, wurden Schnittstellen in beide Richtungen als auch die Datenhaltung täglich während der gesamten Entwicklungszeit getestet. Etwaige Fehler konnten direkt kommuniziert und verbessert werden.

**Parameter** In diesem Kapitel werden die Umgebungsvariablen und deren Einfluss auf die Systemfunktionalität beschrieben.

- Netzwerklast: Da das Uni-Netzwerk sehr schnell ist, konnte hier noch kein Grenzwert ermittelt werden.
- Größe der angefragten Daten: Durch die Speicherung der Bilder im base64-Format kommt der Server hier schnell an seine Grenze. Auch wenige Datensätze dauern über eine Sekunde.
- Anzahl der gleichzeitigen Nutzer: Bisher wurde die Serveranwendung mit fünf gleichzeitigen Nutzern getestet.
- Anzahl der neuen Server-Versionen: Wenn Tomcat zu lange läuft, entstehen möglicherweise Speicher-Verschwendungen. Bisher musste dieser daher einmal neu gestartet werden, um die Speichernutzung zu verringern.

**Probleme** Während der Entwicklung der Serverlogik ist klar geworden, dass das initial geplante Datenbankschema nicht ideal für die prototypische Entwicklung geeignet ist. Abbildung 30 zeigt das geplante Datenbankschema aus dem Arbeitspaket „Datenbank einrichten“.

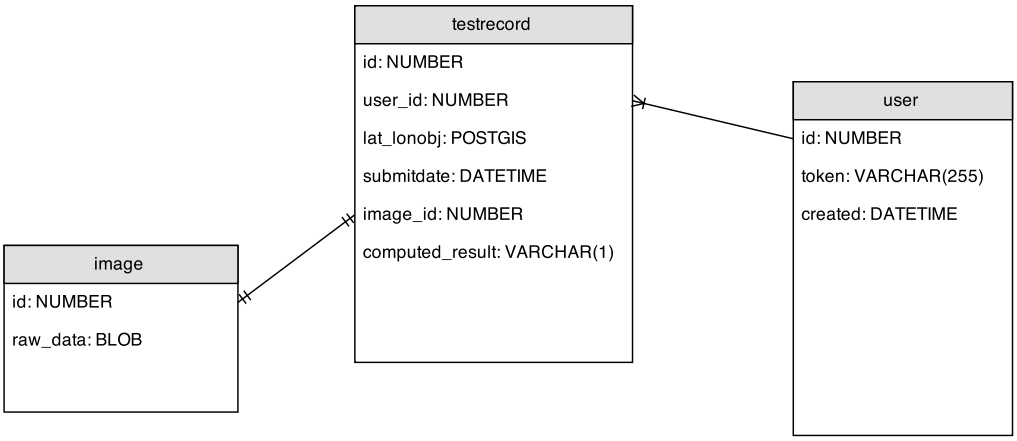



Abbildung 30: Initial geplantes Datenbankschema

Durch den hohen Normalisierungsgrad dieses Schemas wurde die Entwicklung der Serverlogik verkompliziert. Daher wurde das Datenbankschema dahingehend verändert, dass die Anzahl der Relationen verkleinert wurde. Zusätzlich mussten einige Datentypen geändert werden. Abbildung 31 auf der nächsten Seite zeigt das neue Datenbankschema.

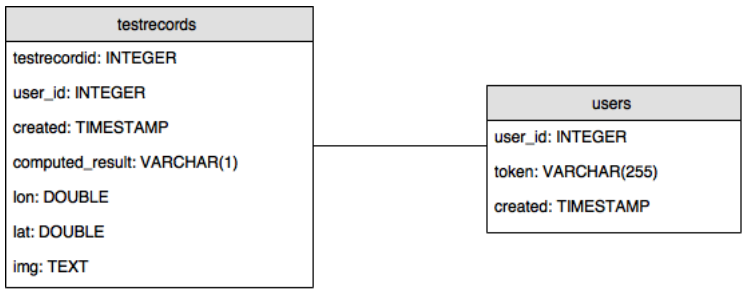



Abbildung 31: Neues Datenbankschema

Ein weiteres Problem hat das Testen der Schnittstelle dargestellt, welches über REST-Schnittstelle angesprochen wird. Dieses konnte nicht, mit Hilfe eines Browsers getestet, sondern durch externe Werkzeuge unterstützt werden. Dazu wurden die Kommandozeilenprogramme wget und curl verwendet.
Das dritte Problem dieses Arbeitspaketes stellte dar, dass die Weboberfläche nicht via Asynchronous JavaScript and XML (AJAX) auf Ressourcen des Servers zugreifen konnte. Um dieses zu lösen, musste Cross Origin Resource Sharing (CORS) auf dem Server aktiviert werden.

**Fazit** Zusammenfassend lässt sich sagen, dass alle Arbeitspakete unter Einhaltung etwaiger Deadlines erfolgreich abgeschlossen werden konnten. Die oben beschriebenen Probleme konnten alle erfolgreich gelöst werden und ergaben für später wichtige Erkenntnisse. Insbesondere der Entwurf einer Serverarchitektur sollte bei folgenden Implementierungen ausführlicher und detaillierter ausfallen, sodass zentrale Strukturen wie etwa das Datenbankschema nicht mehrfach während der Entwicklung geändert werden müssen.

**Ausblick** Ausblickend lässt sich die Erweiterung der prototypischen Serverimplementierung um weitere funktionale Anforderungen und die strengere Einhaltung der nichtfunktionalen Anforderungen planen. Weiter Funktionalitäten wie etwa eine Nutzerverwaltung oder die Verschlüsselung der Übertragung sind zu diesem Zeitpunkt noch nicht umgesetzt und können in folgenden Sprints eingeplant werden. Die Evaluierung von Server- und Schnittstellenarchitekturen und -technologien sind zu diesem Zeitpunkt durch jeweilige Seminararbeiten geplant.

### 4.1.7 Schnittstelle mobile Applikation - Datenbank

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket ist für das Projekt notwendig, um die Kommunikation zwischen der mobilen Applikation und der Datenbank zu ermöglichen.
- Arbeitspaketbeschreibung:
  Als erster Prototyp der mobilen Applikation soll eine Android App entwickelt werden. Diese soll in der Lage sein ein Foto aufzunehmen, die aktuellen Global Positioning System (GPS)-Daten ermitteln und diese gemeinsam mit dem Ergebnis der Prototyp-Analyse (entwickelt in einem anderem Arbeitspaket) an die Datenbank zu schicken. Die Datenbank definiert dazu eine feste Schnittstelle, über die die Daten empfangen werden können. Die empfangenen Daten werden in der Datenbank gespeichert. Die Datenbank wiederum sendet ein Token an die App zurück, das auf dem Smartphone gespeichert wird. Über dieses Token sollen zukünftige Sendungen des gleichen Smartphones in der Datenbank zugeordnet werden können.

  Die Deadline dieses Arbeitspakets ist der 3.6.2015.

- Vorbedingung:
  Ein Server, auf welchem die Schnittstelle bereitgestellt werden kann ist vorhanden.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Für eine Android Applikation existiert eine definierte Serverschnittstelle um einen Datenaustausch zu ermöglichen.
- Aufwand:
  Drei Wochen
- Personen:
  - Christoph Ressel
  - Daniel Wegmann

### 4.1.8 Dokumentation

Diese Dokumentation beschäftigt sich mit der Entwicklung einer Schnittstelle zwischen der mobilen Applikation und der Datenbank von Tests.

**Ablauf** Das Arbeitspaket bezog sich auf folgende Anforderungen:

- FA-1: Die mobile Applikation kann die geographische Position des Probanden bestimmen.
- FA-2: Die mobile Applikation nutzt den Ortungsdienst des Smartphones um dessen Position zu bestimmen
- FA-3: Die mobile Applikation kann das Ergebnis des Tests an den Server senden. Der mobile Datenservice oder WLAN wird genutzt um die Daten zu verschicken.
- FA-4: Der Proband kann ein Analyse des Teststreifens vornehmen lassen. Die mobile Applikation verfügt über einen Bildverarbeitungsalgorithmus, mit dem eine vorläufige Auswertung des Teststreifens vorgenommen werden kann.
- FA-10: Der Proband kann mit der mobilen Applikation ein Foto des Teststreifens aufnehmen. Dazu greift die mobile Applikation auf die Kamera des Smartphones zu, um das Bild aufzunehmen.
- FA-13: Die mobile Applikation kann eine Verbindung zum Server aufbauen.

Als Grundlage wurde die von Jan-Phillip im Laufe der Seminararbeit erstellte Mobile Applikation verwendet. Diese wurde um folgende Funktionen erweitert:

- Zugriff auf die GPS-Funktionen zur Standortermittlung.
- Anbindung zur Server-Schnittstelle zum Versenden der gesammelten Daten.
- Einbinden des Algorithmus zur Bildanalyse, der von einer weiteren Arbeitsgruppe entwickelt wurde.
- Intuitive Nutzerführung durch die Applikation.

Darüber hinaus wurde die Benutzeroberfläche erweitert, sodass der Proband ein Foto zur Testanalyse aufnehmen kann. Sobald dies getan ist, kann mittels eines Buttons die Analyse des Fotos gestartet werden. Das Ergebnis der Analyse wird dem Probanden angezeigt. Über einen weiteren Button können das Ergebnis der Analyse und das Foto über die Schnittstelle an die Datenbank gesendet werden. Hierfür wird das aufgenommene Bild zunächst in das textbasierte Base64-Format kodiert und zusammen mit den restlichen ermittelten Daten (GPS-Position, Bildanalyseergebnis etc.) in ein definiertes Javascript Object Notation (JSON)-Format überführt.
Die Datenbank wurde auf dem Server der Projektgruppe installiert und verfügt über eine Schnitstelle zur Kommunikation mit der mobilen Applikation. Über diese Schnittstelle wird zudem ein Token an das Smartphone zurückgesendet. Über dieses Token kann das Handy bei zukünftigen Übermittlungen identifiziert werden.

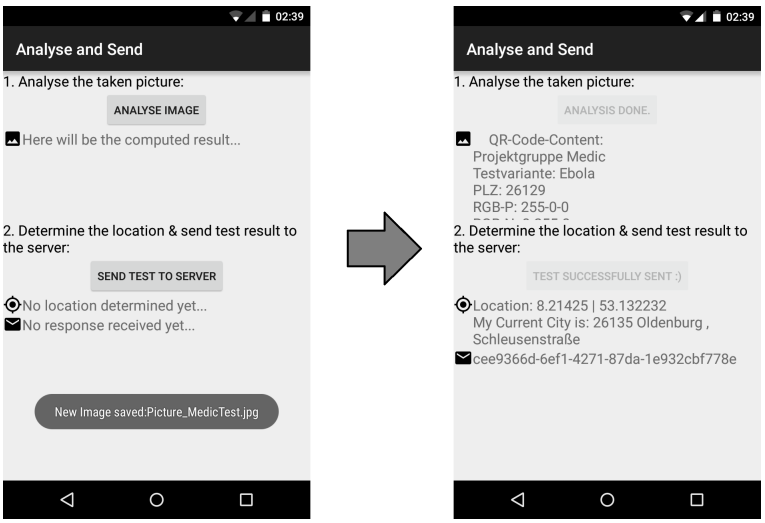



Abbildung 32: Analyseansicht des App-Prototypen.

**Systembeschreibung** Die Mobile Applikation wurde mittels Android Studio nativ für das Betriebssystem Android entwickelt. Als Testplattformen wurden verschiedene Smartphones, unter Anderem ein Google Nexus 4 und ein Samsung Galaxy S3, verwendet. Zu diesem Entwicklungszeitpunkt ist es noch nicht nötig externe Quellen für weitere Tests heranzuziehen, was jedoch für spätere Entwicklungszyklen geplant ist. Darüber hinaus ist geplant die bisher manuellen Tests durch automatisierte Modultests zu ergänzen.

**Evaluation** Die Funktionen wurden per Hand getestet. Zu diesem Zweck wurde die Mobile Applikation auf verschiedenen Android-Smartphones installiert und hinsichtlich der implementierten Funktionalität ausführlich getestet. Zu den Testgeräten zählen neben den in der Systembeschreibung genannten Google Nexus 4 (2012) und Samsung Galaxy S3 (2012), auch ein Google Nexus 6 (2014), ein Samsung Galaxy S2 (2011) und ein OnePlus One (2014). Mit Hilfe dieser Testgeräte wurden verschiedene ausgedruckte QR-Codes abfotografiert, analysiert und das Ergebnis an den Server geschickt. Hierdurch konnte sichergestellt werden, dass die mobile Applikation und die Schnittstelle zur Datenbank den Erwartungen gemäß funktionieren. Die durchgeführten Testdurchläufe ergaben, dass der aktuelle Stand des Prototypen auf den verschiedenen Testgeräten unterschiedlich gut den QR-Code erkennt. Genauere Angaben zu diesen ermittelten Werten sind im Kapitel „Problem“ aufgeführt. Abgesehen von der noch zu verbessernden QR-Code-Erkennung ist festzustellen, dass dieser erste Prototyp alle geforderten funktionalen Anforderungen erfüllt und er somit den Erwartungen entspricht.

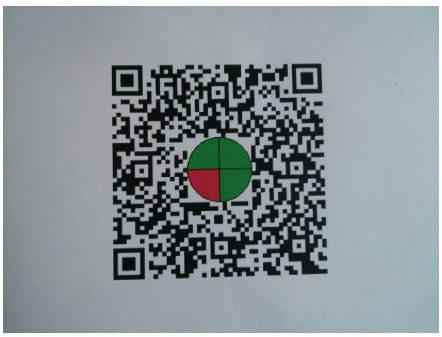


Abbildung 33: Beispiel eines QR-Codes, der vom App-Prototypen fotografiert wurde

**Parameter** Die Funktionalität der Test-Applikation setzt momentan noch bestimmte Parameter an die räumliche Testumgebung voraus. Zum einen muss der zu fotografierende QR-Code ausreichend hell beleuchtet sein. So wurde für diesen Prototyp eine normale künstliche Raumbeleuchtung (weiße Licht) genutzt. Die Wahrscheinlichkeit, den QR-Code richtig zu erkennen ist deutlich größer, wenn keine Schatten auf dem QR-Code sind. Da noch keine Auto-Fokus-Funktionalität implementiert ist, muss darauf geachtet werden, dass der aufgenommene QR-Code in passender Entfernung (beim Testgerät Google Nexus 4 ca. 40 cm) zu der aufnehmenden Kamera ist, um ein möglichst scharfes Bild zu erhalten.

**Fazit** Es konnte ein funktionsfähiger, nativer Prototyp entwickelt werden. Die Funktionen dieses Prototyps sind in ihrem Umfang noch stark eingeschränkt, aber im Einzelnen vollständig umgesetzt. Im aktuellen Stand stellt das Erkennen des QR-Codes das größte Problem dar und muss daher noch verbessert werden. Hierfür können Funktionen wie der Auto-Fokus der Kamera hilfreich sein. Das Ermitteln der geographischen Position und das Fotografieren des Teststreifens hingegen funktionieren schon komplett problemlos. Die Kommunikation der mobilen Applikation mit dem Server funktioniert bei allen bisherigen Testgeräten außer einem Google Nexus 6 einwandfrei. Alles in Allem stellt der Prototyp eine gute Basis für die weitere Entwicklung dar.

**Ausblick** Im weiteren Entwicklungsverlauf müssen verschiedene Aspekte des Prototyps verbessert werden. Dazu gehören vor allem der Bildverarbeitungs-Algorithmus und die Benutzeroberfläche. Außerdem müssen zukünftig Verschlüsselungsalgorithmen implementiert werden, um die Nutzerdaten zu schützen. Zusätzlich muss die gesamte Funktionalität noch für das Betriebssystem iOS implementiert werden, damit auch iPhones und iPads unterstützt werden können.

### 4.1.9 Schnittstelle Auswertungsoberfläche - Datenbank

- Priorität: Kritisch

- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket umfasst die Erstellung einer Auswertungsoberfläche zur Darstellung von Informationen zu Teststreifendaten in der Datenbank (die Datenbank ist in Kapitel 4.1.4 auf Seite 46 beschrieben). Dazu wird auf dem Server (siehe Kapitel 4.1.3 auf Seite 45) eine Schnittstelle entwickelt, die diese Informationen bereitstellen kann.

- Arbeitspaketbeschreibung:
  Zur Realisierung der Auswertungsoberfläche wird eine Weboberfläche implementiert. Diese Weboberfläche soll Medizinern eine einfache Schnittstelle zu den Teststreifendaten aus der Datenbank bereitstellen. Auf der Weboberfläche soll eine Liste aller Teststreifendaten angezeigt werden, die bei Bedarf detaillierte Informationen zu einzelnen Datensätzen liefert. Dies beinhaltet

die Anzeige des aufgenommenen Bildes, sowie das Ergebnis der Voranalyse und die Geoinformationen. Zusätzlich soll das Datum des Teststreifens angezeigt werden.

Zudem soll eine Heatmap integriert werden, welche die Geodaten mit Pins auf der Karte von Deutschland darstellt.

Die Deadline dieses Arbeitspakets ist der 27.5.2015.

- Vorbedingung:
  Dieses Arbeitspaket arbeitet eng mit dem Arbeitspaket „Server optimieren“ (siehe: Kapitel 4.1.5 auf Seite 47) zusammen und ist auf die dort definierten Schnittstellen zur Kommunikation mit der Weboberfläche angewiesen.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Eine existiert eine Weboberfläche mit folgenden Funktionen:
  - Eine Liste der empfangenen Daten der Probanden.
  - Ein Foto wird auf einer separaten Oberfläche angezeigt, wenn eine Element der Liste ausgewählt wird.
  - Eine Heatmap, welche die Geodaten auf der Karte visualisiert.
- Aufwand:
  Drei Wochen
- Personen:
  - Christian Sandmann
  - Timo Raß
  - Sebastian Horwege

### 4.1.10 Dokumentation

Das Ziel des zweiten Sprints war es, eine Weboberfläche zu entwerfen, welche die geographischen Daten, das Ergebnis der Voranalyse und die Identifikation (ID) zu einem durchgeführten Test über eine Schnittstelle aus der Datenbank erhält.

**Ablauf** Die oben genannten Daten sollten in einer Liste und einer Heat-Map angezeigt werden. Bei einem Klick auf einen Eintrag in der Liste wird das Bild, welches zu diesem Test gehört, in einem separaten Feld angezeigt. Um diese Funktionen zu gewährleisten wurden die folgenden Anwendungsfälle bzw. Anforderungen herangezogen.

Anwendungsfall: Auswertung durch Mediziner

FA-20: Die Auswertungsoberfläche muss eine interaktive Karte von Deutschland beinhalten. Diese geographische Auswertungskarte muss eine Heatmap von Krankheitstests darstellen. Sie wird durch eine entsprechende Legende dokumentiert. (Anforderung FA-20 wurde für den Prototypen reduziert umgesetzt.)

FA-23: Die Auswertungsoberfläche stellt dem Mediziner eine Liste mit Probanden bereit, dessen Diagnosen noch ausstehen.

FA-26: Bei der Diagnose eines Tests ermöglicht es die Auswertungsoberfläche dem Mediziner den Test als positiv, negativ oder unlesbar zu erklären.

FA-28: Während der Diagnose ermöglicht es die Auswertungsoberfläche dem Mediziner, seine Diagnose mit der Voranalyse der mobilen Applikation zu vergleichen.

Bei der Entwicklung des ersten Prototyps musste zunächst entschieden werden, welche Web-Technologien verwendet werden sollten.

**Technologien**

Der Prototyp für die Weboberfläche wurde mit den unten aufgelisteten Technologien entwickelt, weil diese einen hohen Etablierungsgrad in der Webentwicklung aufweisen. Da zu diesen noch zahlreiche alternative Technologien existieren, wird es dazu im kommenden Sprint ein Seminarthema geben, um zu evaluieren, ob sich andere Technologien eher eignen, als die für den Prototyp verwendeten.

**HTML** wird verwendet, um den Aufbau der digitalen Elemente in einem HTML-fähigen Browser zu strukturieren.

**JavaScript** ist eine Skriptsprache, mit der die Funktionalität realisiert werden kann. Diese wurde verwendet, weil es eine der am meist verbreiteten Sprachen für die Webentwicklung ist.

**CSS** wurde für die Gestaltung der Weboberfläche verwendet.

**JQuery** ist ein JavaScript Framework, welches Funktionen zur Manipulation von HTML zur Verfügung stellt.

**JQuery mobile** ist ein JavaScript Framework für die Darstellung einer mobilen Webanwendung. Dieses wurde benutzt, um die Listenansicht anzeigen zu lassen. Diese JQuery mobile Liste wurde gewählt, weil diese bereits eine übersichtliche Formatvorlage mit sich bringt.

**AngularJS** bringt den Model Viewer Controller (MVC)-Ansatz in die Entwicklung von Webseiten. Die View-Komponenten der MVC-Struktur sind in AngularJS durch einfaches HTML mit benutzerdefinierten Elementen und Attributen realisiert. Spezielle JavaScript-Objekte, die als Module zusammengefasst werden, übernehmen die Logik. Die Models sind in den Controllern integriert und werden bei Bedarf durch Schnittstellenzugriffen mit Daten gefüllt.

**Leaflet** ist ein Framework, um Karten und Geodaten auf Webseiten anzuzeigen. Der Funktionsumfang ist mit dem von Google Maps zu vergleichen. Die Openstreetmap-Daten, die mit Leaflet angezeigt werden können unterliegen der Creative-Commons-Attribution-ShareAlike-2.0 Lizenz und ermöglichen eine weitestgehend uneingeschränkte Nutzung, im Gegensatz zu den Nutzungsbedingungen von Google Maps. Leaflet ist durch eine Reihe von Plugins erweiterbar.

**Entwicklung**

Bei der Entwicklung der Weboberfläche war es zunächst wichtig, die Verbindung zur Schnittstelle des Servers herzustellen. Dies wurde mit einem AJAX-Aufruf realisiert, der ein Foto und die übrigen Daten zum Test des Probanden auslesen kann. Diese wurden zunächst auf einer HTML Seite anzeigt. Im Anschluss ging es darum, mithilfe von IDs ein Foto zu dem entsprechenden Probanden zuzuordnen. Die Liste, die vorher nur in einem HTML-Container angezeigt werden konnte, wurde nun durch eine richtige Liste von JQuery mobile ersetzt und gestaltet. Danach wurde die Einbindung der Karte vorgenommen. Außerdem wurde die Weboberfläche mit Cascading Style Sheets (CSS) gestaltet.

Nach der Erstellung der Grundfunktionalität der Weboberfläche mit jQuery wurde mithilfe von AngularJS eine MVC-Struktur umgesetzt. Am Ende dieses Umbaus war die Funktionalität der Weboberfläche die gleiche. Dadurch konnten beide Ansätze gut miteinander verglichen werden. Die Erkenntnisse dieses Vergleichs werden im Abschnitt „Fazit“ kurz erläutert.

Das Resultat der Weboberfläche für diesen Prototyp zeigt Abbildung 34. Auf der linken Seite befindet sich die Liste mit den Daten, auf der rechten Seite das Bild zu dem ausgewählten Datensatz aus der Liste. Abbildung 35 auf der nächsten Seite zeigt den Prototyp zur geographischen Auswertung der Daten. Die Visualisierung ist mit Leaflet realisiert. Durch Kontrollkästchen können als positiv, negativ oder unsicher klassifizierte Testergebnisse ein und ausgeblendet werden. Mithilfe des Heatmap-Plugins[2] werden die positiven Testergebnisse als HeatMap-Ebenen über die Karte gelegt. Zusätzlich können die positiven Testergebnisse noch zu Gruppen gebündelt werden. Dies wird durch das MarkerCluster-Plugin[3] realisiert.

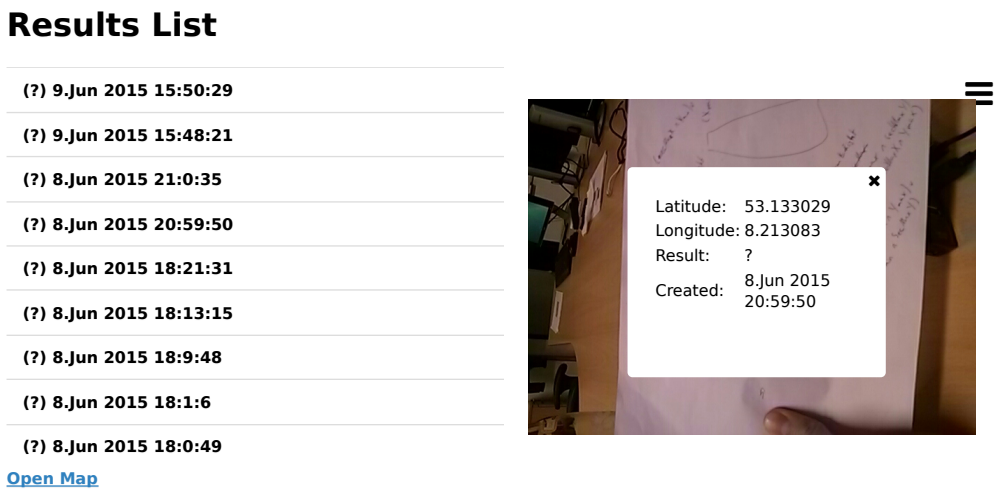



Abbildung 34: Screenshot des derzeitigen Standes des Webfrontends

[2] https://github.com/Leaflet/Leaflet.heat
[3] https://github.com/Leaflet/Leaflet.markercluster

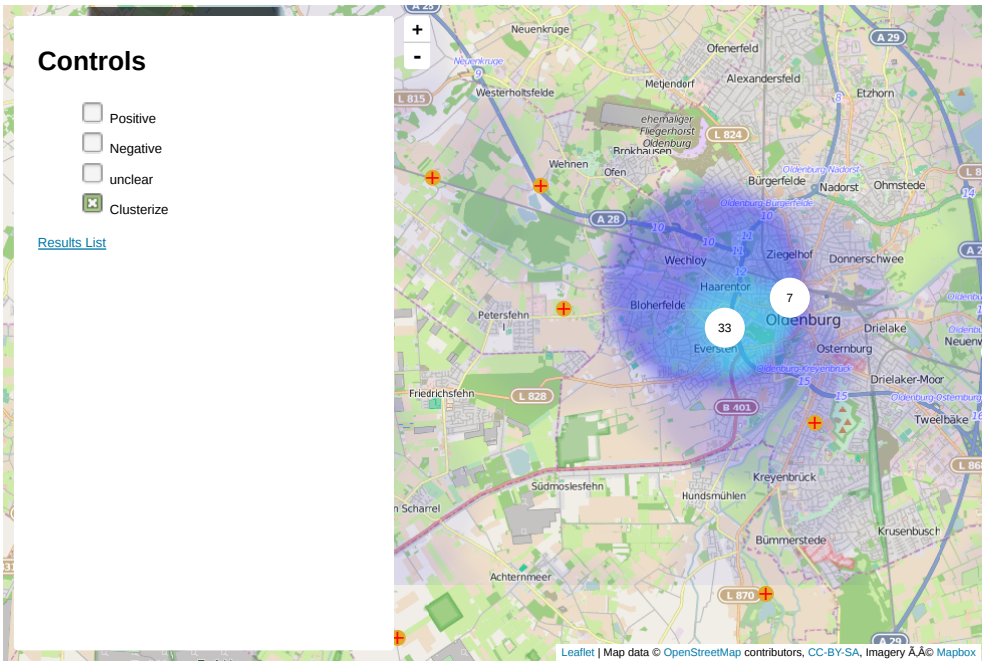



Abbildung 35: Screenshot der geographischen Auswertungsoberfläche

**Systembeschreibung** Das Webfrontend für den Prototyp wurde zunächst auf einem zentralen Desktop-System mit Windows 7 als Betriebssystem entworfen. Zur Entwicklung der Weboberfläche wurde außerdem Brackets[4] verwendet. Nachdem die elementaren Funktionen (Datensätze am Server anfragen, Liste aller Bilder der Datenbank anzeigen, Bild bei Klick auf Liste anzeigen) implementiert wurden, wurde das Webfrontend auf den Server geladen und ist dort unter der IP-Adresse 134.106.47.158 zu erreichen. Änderungen am Webfrontend werden zehn Minuten nach einem `git push` über Jenkins auf dem Server zur Verfügung gestellt.

**Evaluation** Getestet wurde die Weboberfläche in den Browsern Google Chrome, Firefox und Internet Explorer, jeweils in den neuesten Versionen. Des Weiteren wurde ein Test mit einem iPad durchgeführt. Dieser ergab, dass das Webfrontend auch im Safari-Browser auf dem Tablett aufgerufen und genutzt werden kann (iOS 8). Alle Tests bezogen sich auf Funktionen, nicht jedoch auf Gebrauchstauglichkeit oder Design. Als Voraussetzung für die einwandfreie Funktionalität muss JavaScript in den Browsern aktiviert sein.

**Parameter** Die oben beschriebenen Tests wurden von allen elf Gruppenmitgliedern durchgeführt, um möglichst viele verschiedene Browser zu testen. Es wurden keine weiteren Testpersonen herangezogen, da für einen ersten Prototyp eine Testgruppe von elf Personen, welche ausschließlich die Funktionen, nicht jedoch die Gebrauchstauglichkeit testen sollten, ausreichend ist. Diese Testgruppe wird in kommenden Sprints noch erweitert, um eine angemessene Evaluation, die auch Gebrauchstauglichkeit und Design beinhaltet, zu gewährleisten.

**Fazit** Um das Webfrontend weiter zu gestalten, müssen noch andere Webtechnologien berücksichtigt werden, damit eine effiziente und qualitativ hochwertige Implementierung gewährleisten werden kann.

[4] `http://de.wikipedia.org/wiki/Adobe_Brackets`

Da das Webfrontend einen wesentlichen Teil des Projektes darstellt, werden diese in einem Seminar genau untersucht. Im weiteren Verlauf kann der Prototyp mit den bisherigen Technologien um weitere Funktionen ergänzt werden. Die Einarbeitung in AngularJS war anfangs recht aufwendig. Ein MVC-Ansatz für Webfrontends bietet aber auf längere Sicht mehr Flexibilität bei der Entwicklung, als einfaches jQuery.

**Ausblick** Für den nächsten Sprint soll das Webfrontend um folgende Funktionen erweitert werden:

- Die Kommunikation mit dem Server soll erweitert werden. Dies umfasst ein Textfeld und einen Senden-Button für die Diagnose, sowie die Funktion eine solche Diagnose für einen ausgewählten Datensatz zu verfassen, und dann an den Server zu senden. Dort wird diese gespeichert.
- Das Nutzermanagement soll eingebaut werden, sodass es für Ärzte möglich ist, sich zu registrieren und einzuloggen. Dazu muss es eine Seite für diese Funktionen geben. Auch wird es die Möglichkeit zum Ausloggen geben.
- Ebenso wird das Design und die Struktur des Webfrontends überarbeitet und angepasst.
- Die geographische Auswertung wird um einige Logik erweitert. So sollte es zum Beispiel möglich sein, dass einzelne Testergebnisse ausgewählt werden können und dann Informationen zu diesen angezeigt wird. So, wie es bereits in der Ergebnisliste umgesetzt ist. Neben Leaflet können auch noch andere Technologien (wie Google Maps) zur Visualisierung der Daten getestet werden.

### 4.1.11 Bildanalyse

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  In diesem Arbeitspaket soll ein erster Algorithmus zur Erkennung und Auswertung von Teststreifen entwickelt werden. Zudem sollen erste Erfahrungen mit der Bildverarbeitungs-Bibliothek OpenCV gesammelt werden.
- Arbeitspaketbeschreibung:
  Für die benötigte Bilderkennung soll ein erster Prototyp entwickelt werden. Dieser Prototyp wird mit Hilfe von OpenCV entwickelt und soll eine Farberkennung in einem bestimmten Bereich vornehmen können. Der Bereich wird mittels eines QR-Codes, welcher mit dem Foto des Teststreifens aufgenommen wird und mittels QR-Code-Scanner ausgelesen wird, detektiert. Dieser Algorithmus und der QR-Code-Scanner werden dann in die im Arbeitspaket „Schnittstelle mobile Applikation - Datenbank “ (Kapitel 4.1.7 auf Seite 52) bereitgestellte Android-Applikation integriert.

  Die Deadline dieses Arbeitspakets ist der 27.5.2015.
- Vorbedingung:
  Die Android-Applikation, welche die Möglichkeit bietet ein Foto aufzunehmen, ist für die Integration der Bildverarbeitungsalgorithmen vorhanden.
- Nebenbedingungen:
  -
- Nachbedingung:
  Es existiert ein erster Bilderkennungsalgorithmus und ein QR-Code-Scanner, welcher einen QR-Code auslesen kann. Beide Komponenten sind in die Android-Applikation integriert.

- Aufwand:
  Drei Wochen
- Personen:
  - Danny Fonk
  - Raphael Kappes

#### 4.1.12 Bildanalyse Optimierung

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Im Zuge dieses Arbeitspaketes wird die bereits vorhandene Bildanalyse optimiert. Die nachfolgenden Punkte spezifizieren die Aufgabe dieses Arbeitsschrittes.
- Arbeitspaketbeschreibung:

  Die bereits vorhandene QR-Code-, Muster- und Farberkennung bedarf Optimierungen, da eine hohe Erfolgsquote bislang nicht gewährleistet werden konnte. Die dabei eingesetzten Frameworks sind weiterhin OpenCV zwecks Muster-/Farberkennung sowie ZXing für die Detektion und das Auslesen des QR-Codes. Eine genaue Abschätzung bezüglich der Erfolgsquote der QR-Code, Muster- und Farberkennung kann an dieser Stelle nicht vorgenommen werden. Es wird für diesen Prototypen eine Erfolgsquote von 90% unter den nachstehenden Bedingungen angestrebt.

  1. Winkel der Aufnahme darf nicht über 10 Grad sein.
  2. Es müssen annehmbare Lichtverhältnisse herrschen.

  Die Deadline dieses Arbeitspakets ist der 27.5.2015.
- Vorbedingung:
  Die Android-Applikation, mit dem integrierten Prototypen des Bilderkennungsalgorithmus sowie der Fähigkeit einen QR-Code auszulesen sind vorhanden.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Die Android-Prototyp-App kann mit einer Erfolgsquote von 90% unter definierten Bedingungen den Bildbereich mit dem Farbumschlag detektieren.
- Aufwand:
  Zwei Wochen
- Personen:
  - Danny Fonk
  - Raphael Kappes

#### 4.1.13 Dokumentation

Die nachstehende Dokumentation bezieht sich auf die Arbeitspakete „Bildanalyse“ und „Bildanalyse Optimierung“.

**Ablauf** Um ein Farbumschlagsfeld auf einem Teststreifen zu erkennen, wurde unter der Voraussetzung, dass uns das Teststreifendesign unbekannt war zuerst eine QR-Code-Erkennung implementiert. Diese QR-Code-Erkennung dient dazu, dass sich der Bereich in dem sich das Farbumschlagsfeld befindet detektieren lässt, sowie die Testparameter (Testart, RGB-Farbwert für ein positives Testergebnis, RGB-Farbwert für negatives Testergebnis) aus eben diesem QR-Code auszulesen. Im Anschluss daran wird der detektierte Bereich in Abhängigkeit der Größe des erkannten QR-Codes in der Fläche eines Kreises abgetastet und die dort enthaltenen Pixel auf ihren Farbwert hin überprüft. Für die QR-Code-Erkennung sowie für die Musterkennung wurden jeweils bereits vorhandene OpenSource-Bibliotheken genutzt. Der Code wurde vorab in Java geschrieben und im Anschluss daran mit einigen wenigen Anpassungen in die bereits bestehende Android-Prototyp-App integriert und getestet.

**Technologien**

**ZXing** Für die Entwicklung der QR-Code-Erkennung wurde die Bibliothek ZXing[5] in der Version 3.2 genutzt. Sie wurde für diesen Prototypen genutzt, da sie sich einer hohen Beliebtheit erfreut und somit eine große Community vorhanden ist und wichtiger noch, da die Bibliothek sowie für Android als auch für iOS zur Verfügung steht.

**OpenCV** Für die Farb-/Mustererkennung wurde die OpenSource-Bibliothek OpenCV in der Version 2.4.11 für Android genutzt.

**Systembeschreibung** Der Quellcode wurde zunächst in Java geschrieben und wurde nicht in einer mobilen Applikation, sondern auf Desktop-Rechnern mit generierten QR-Codes und später mit Fotos dieser generierten QR-Codes getestet. Nach der Integration in die Android-Prototyp-App wurde die QR-Code-Erkennung und Bildanalyse mittels des in der Android-Prototyp-App aufgenommen Bildes durchgeführt.

**Evaluation** Für die Tests wurde online[6] ein QR-Code mit Logo generiert. Dieses Logo wurde anschließend entfernt und durch einen Kreis, welcher in vier Sektionen unterteilt wurde, ersetzt. Die Sektionen sollen die verschiedenen Farbumschläge innerhalb eines Testes simulieren. Diese Sektionen wurden unterschiedlich eingefärbt, um unterschiedliche Testergebnisse (positiv, negativ oder unbestimmt) zu simulieren und die Farb-/Mustererkennung dahingehend zu überprüfen.

Des Weiteren wurden diese unterschiedlichen QR-Codes ausgedruckt und mit unterschiedlichen mobilen Endgeräten (OnePlus One, Samsung Galaxy SII, LG Nexus S4, Nokia Lumia 920) abfotografiert um anschließend die so entstandenen Fotos wieder für Testdurchläufe zu verwenden und auf ihre Korrektheit hin zu überprüfen. Nach der Integration in die bestehende Android-Prototyp-App wurde auch die Integration auf Stabilität und Zuverlässigkeit überprüft. Des Weiteren konnte die in der Arbeitspaket angestrebte Erfolgsquote (90%, bei 20 Versuchen) unter den dort definierten Bedingungen eingehalten werden.

**Parameter** Die Bildanalyse wurde unter der Annahme entwickelt, dass sich das Farbumschlagsfeld einer Teststreifenvariante in der Mitte eines QR-Codes befindet. Während der Entwicklung kam im Gespräch mit den Projektbetreuern heraus, dass solch ein Teststreifendesign eher unrealistisch ist, was dadurch begründet ist, dass die Mikrokanülen durch einen Druck (Druck des QR-Codes) beschädigt werden können. Aufgrund fehlendem Autofokus ist die Höhe wenig variabel, da das Bild unscharf wird. Weiterhin ist der Winkel aus dem das Foto gemacht wird entscheidend. Wenn das Smartphone direkt auf den QR-Code gerichtet ist, kann es bis zu ca. zehn Grad geneigt werden.

[5] https://github.com/zxing/zxing

[6] `https://www.unitag.io/qrcode`

Abbildung 36: Generierter QR-Code ohne Logo

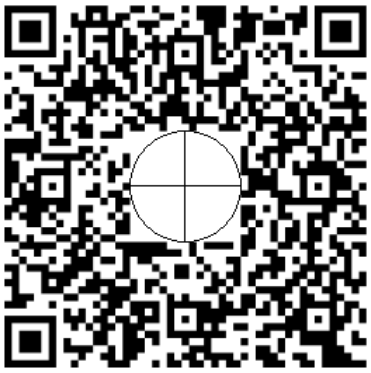


Abbildung 37: Generierter QR-Code mit Kreis und Vier Sektionen

Abbildung 38: Generierter QR-Code mit Kreis und eingefärbt

**Fazit** Die QR-Code- sowie die Farberkennung konnten bereits sehr robust gestaltet werden und werden zu großen Teilen für den weiteren Verlauf dieses Projektes weiterverwendet werden können. In Gänze ist OpenCV als Bibliothek zur Bildverarbeitung sehr tauglich und die Ausführung auf mobilen Endgeräten ist performant genug, sodass eine Gefährdung von Anforderungen (insbesondere bzgl. der Ausführungszeit) an dieser Stelle des Projektes als nicht gegeben scheint. Auch ZXing konnte zielführend eingesetzt werden, jedoch ist die ZXing-Bibliothek bzgl. der Dokumentation ausbaufähig. Diese Umstände lassen folgern, dass ggf. die Bibliothek zum Auslesen von QR-Codes durch eine andere ersetzt werden sollte, aber die Bibliothek zur Farb- und Mustererkennung kann (bisher) ohne weitere Bedenken genutzt werden.

**Ausblick** Der erarbeitete Quellcode wird in C umgeschrieben, sodass er nicht nur in der bestehenden Android-Prototyp-App, sondern auch in der iOS-Prototyp-App genutzt werden kann und der Quellcode nur an einer Stelle erweitert und gewartet werden muss. Des Weiteren steht eine Anpassung an den tatsächlichen Teststreifen noch aus, da zum Zeitpunkt der Entwicklung das endgültige Teststreifendesign noch nicht bekannt war. Daher muss die Detektierung des Farbumschlagsfeldes neu implementiert werden.

## 4.2 Sprint 3

Das Ziel des dritten Sprints ist es, die bisher erstellten Prototypen zu erweitern. Diese Erweiterungen bestehen aus mehreren Teilen: Zum einen soll eine mobile Applikation für iOS erstellt werden, deren Funktionalität identisch zu der im vorherigen Sprint erstellten Android Applikation ist. Auf beiden Betriebssystemen soll außerdem der QR-Code dynamisch aus einem Video-Livestream der Kamera erkannt werden. So muss der Proband nicht länger manuell ein Foto des Teststreifens aufnehmen. Darüber hinaus soll die Heatmap der Weboberfläche um neue Funktionen erweitert und ein Usermanagement eingerichtet werden. Das Usermanagement soll gleichzeitig dazu dienen, die Weboberfläche vor unberechtigten Zugriffen zu schützen. Außerdem wird die Weboberfläche um die Möglichkeit erweitert, eine Diagnose an den Probanden zurückzuschicken. Der Bildanalysealgorithmus wird neu implementiert, um neu definierten Anforderungen entsprechen zu können.
Um zukünftige Arbeiten zu erleichtern, wurden außerdem zwei weitere Arbeitspakete definiert. Zunächst soll in den Apps die Möglichkeit geschaffen werden C-Code zu integrieren um das OpenCV Framework, welches zur Bildanalyse verwendet wird, darüber einzubinden. Dies hat den Grund, dass der Algorithmus so auf beiden Betriebssystemen verwendet werden kann und zukünftige Änderungen nicht jeweils an zwei verschiedenen Stellen vorgenommen werden müssen. Zusätzlich dazu soll ein Testdatengenerator erstellt werden, der es erlaubt, schnell eine große Anzahl fiktiver Tests zu erstellen und an den Server zu schicken. Dies erlaubt es Funktionen, die mit den Testdaten interagieren, schneller und effizienter zu testen.
Die Details der jeweiligen Arbeitspakete sind in den entsprechenden Unterkapiteln in Kapitel 3 zu finden.

### 4.2.1 Sprintplanung

Die Dauer des Sprints wurde auf sechs Wochen (10. Juni bis 21. Juli 2015) angesetzt, wobei die Arbeitslast auf verschiedene Arbeitspakete aufgeteilt wurde. Die Planung wurde dabei zunächst von einer kleinen Gruppe durchgeführt. Dabei wurden die Arbeitspakete grob vordefiniert und eine erste Aufwandsabschätzung vorgenommen. Die Ergebnisse dieser Vorplanung wurden anschließend mit der gesamten Gruppe diskutiert. In dieser Diskussion wurde die Aufwandsabschätzung der einzelnen Pakete angepasst und die bearbeitenden Personen festgelegt.
Jedes Arbeitspaket ist auf eine feste Zeit angelegt und umfasst die vollständige Implementation der gestellten Aufgabe sowie die dazugehörige Dokumentation. Am Ende des Sprints wird der erreichte Fortschritt dem Kunden in einer Präsentation vorgestellt.
Im Nachhinein wurde der Sprint um eine Woche verlängert, um alle Arbeitspakete abschließen zu können. Damit fiel das tatsächliche Ende des Sprints auf den 28. Juli 2015.

**4.2.1.1 Fokus des Sprints** In diesem Sprint lag der Fokus darauf, sämtliche bisher erstellte Prototypen (mit Ausnahme des Server-Backends und der Datenbank) noch einmal zu überarbeiten. Außerdem wird der Prototyp einer iOS-App erstellt.

**4.2.1.2 Arbeitspakete und Planung** Aus dem Fokus und der Planung ergeben sich folgende Arbeitspakete

- Implementierung des Prototyp-Quellcodes in Objective-C
- Erweiterung der Heatmap
- Optimierung Kamera Android
- Optimierung Kamera iOS

- Diagnose-Funktion
- Bildverarbeitungsalgorithmus in C
- Integration des in C geschriebenen Algorithmus in die Apps
- Testdatengenerator
- Usermanagement

#### 4.2.1.3 Planung der Ressourcen

Im folgenden ist die Personalplanung dargestellt. Sie gibt Auskunft darüber, welches Teammitglied an welchem Arbeitspaket gearbeitet hat.

- Raphael Kappes: Testdatengenerator (2 Wochen), Usermanagement (3 Wochen)
- Christian Sandmann: Implementierung des Prototyp-Quellcodes in Objective-C (2 Wochen), Bildverarbeitungsalgorithmus in C (4 Wochen)
- Timo Schlömer: Implementierung des Prototyp-Quellcodes in Objective-C (2 Wochen), Diagnose-Funktion (3 Wochen)
- Timo Raß: Implementierung des Prototyp-Quellcodes in Objective-C (2 Wochen)
- Nicolas Koch: Optimierung Kamera Android (3 Wochen), Optimierung Kamera iOS (1 Woche), Integration des in C geschriebenen Algorithmus in C (2 Wochen)
- Kevin Sandermann: Diagnose-Funktion (3 Wochen), Testdatengenerator (2 Wochen)
- Sebastian Horwege: Seminararbeit
- Danny Fonk: Seminararbeit
- Jan Philipp Stubbe: Erweiterung der Heatmap (1,5 Wochen), Optimierung Kamera iOS (1 Woche), Integration des in C geschriebenen Algorithmus in die Apps (2 Wochen)
- Christoph Ressel: Optimierung Kamera Android (3 Wochen), Bildverarbeitungsalgorithmus in C (4 Wochen)
- Daniel Wegmann: Erweiterung der Heatmap (1,5 Wochen), Usermanagement (3 Wochen)

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und dessen Dokumentation skizziert.

### 4.2.2 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.2.3 Implementierung des Prototyp-Quellcodes in Objective-C

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes: Dieses Arbeitspaket ist notwendig, den prototypischen Quellcode der Android-Applikation in die IOS-Applikation zu integrieren.
- Arbeitspaketbeschreibung:
  Im Arbeitspaket „Implementierung des Prototyp-Quellcodes in Objective-C“ werden zunächst von den ausführenden Teammitgliedern Tutorials für die Entwicklung von iOS Apps mit XCode durchgearbeitet. Nachdem dies erfolgt ist, sollen erste Test-Apps entwickelt werden, welche zum Beispiel die Kamera-Schnittstelle ansprechen. Wenn ein einheitliches Verständnis für die Programmierung in Objective-C geschaffen wurde, wird der bestehende Java-Quellcode der Prototyp-App in Objective-C überführt. Die Integration soll dabei schrittweise erfolgen, um so die Erfahrung in der Entwicklung von iOS-Apps weiter zu schärfen und die Funktionalität sicherzustellen.

  Die Deadline dieses Arbeitspaketes ist der 17.06.2015.
- Vorbedingung:
  Als Vorbedingung gilt, dass der Quellcode der Android-Prototyp-App zur Verfügung steht, um diesen in die Programmiersprache Objective-C zu überführen. Zudem müssen die Apple-Computer (Mac mini und MacBook Pro 15) mit der aktuellen IDE Xcode (Version 6.3.2) von Apple ausgestattet sein.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Am Ende des dritten Sprints muss eine iOS App vorhanden sein, welche es ermöglicht, ein Foto von einem QR-Code aufzunehmen. Dieser muss erkannt und das Bild an den Server, welcher mit der Datenbank verbunden ist, gesendet werden. Des Weiteren beinhaltet die App den Bilderkennungsalgorithmus aus der Android-Prototyp-App, welche die Farbe aus der Mitte des QR-Codes detektieren kann.
- Aufwand:
  Zwei Wochen
- Personen:
  - Timo Schlömer
  - Christian Sandmann
  - Timo Rass

**4.2.3.1 Dokumentation** Die folgende Arbeitspaketdokumentation beschreibt die Erstellung und Implementierung des ersten Entwurfes der iOS-Prototyp-App. Dies ergab sich aus der Erkenntnis, dass iOS neben Android eines der verbreitetsten mobilen Betriebssysteme ist. Somit soll im Projektverlauf neben einer Android-App auch eine App für iOS entwickelt werden, um die Anforderung des Kunden eine möglichst hohe Marktabdeckung mit der App zu erreichen, umzusetzen. Die Anforderungen an die iOS-App ergeben sich aus dem Anwendungsfall **Proband führt Test durch** mit den funktionalen und nichtfunktionalen Anforderungen:

**FA-1** Die mobile Applikation kann die geographische Position des Probanden bestimmen.

**FA-3** Die mobile Applikation kann das Ergebnis des Tests an den Server senden. Der mobile Datenservice oder WLAN wird genutzt um die Daten zu verschicken.

**NFA-20** Die mobile Applikation muss für Android und iOS bereitgestellt werden. Hierdurch wird eine Abdeckung von über 96 % aller Smartphones in Deutschland erreicht.

**Ablauf** Zur Entwicklung der iOS-Prototyp-App wurde die Vorlage aus der Seminararbeit von Jan Philipp Stubbe herangezogen, da diese bereits relevante Funktionen beinhaltete. Nachdem sich die drei Entwickler zunächst mit der Entwicklungsumgebung **Xcode** beschäftigten, wurden anschließend Schritt für Schritt folgende Funktionen und grafische Benutzerelemente implementiert:

- Eine Startseite, welche einen Button enthält, um die Kamera zu öffnen und einen Button mit der Beschriftung **Send & Receive Tests**, welcher zu einem weiteren Auswahlmenü führt (vgl. Abbildung 39).

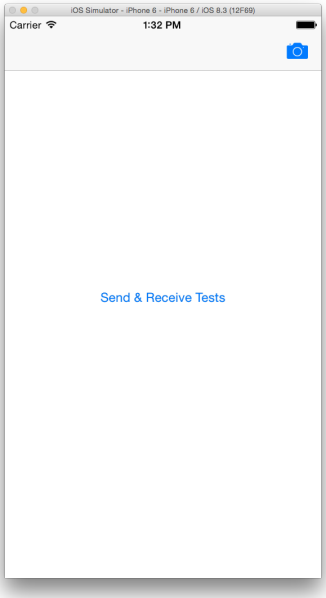



Abbildung 39: Erster Entwurf der iOS-Startseite

- Nachdem der Button **Send & Receive Tests** gedrückt wurde, gelangt der Nutzer zu einer neuen Ansicht. Der Button **Load the latest Tests** lädt die letzten zehn Tests vom Server und zeigt diese in der Konsole der Entwicklungsumgebung an. Der Button **Send a new Test** sendet einen Beispiel-Test an den Server. Dieser Test kann über die Weboberfläche abgerufen und dargestellt werden. (vgl. Abbildung 40 auf der nächsten Seite).

- Wenn der Kamera-Button auf der Startseite der App gedrückt wird, gelangt der Nutzer zur Kameraansicht, um ein Bild aufnehmen zu können. Wenn die Kamera nicht verfügbar ist, weil beispielsweise der durch Xcode mitgelieferte iOS-Simulator verwendet wird, erscheint eine Fehlermeldung. Diese führt den Nutzer wieder zur Startseite der App zurück (vgl. Abbildung 41 auf Seite 69).

- Im derzeitigen Zustand ist es möglich einen Beispiel-Test an den Server zu senden. Hierzu wird eine aktive Internetverbindung benötigt. Wenn der Test erfolgreich versendet wurde erscheint ein Dialogfenster. Dieses bestätigt dem Nutzer die Übermittlung des Tests. Der Nutzer gelangt danach auf die Startseite der App zurück (vgl. Abbildung 42 auf Seite 70).

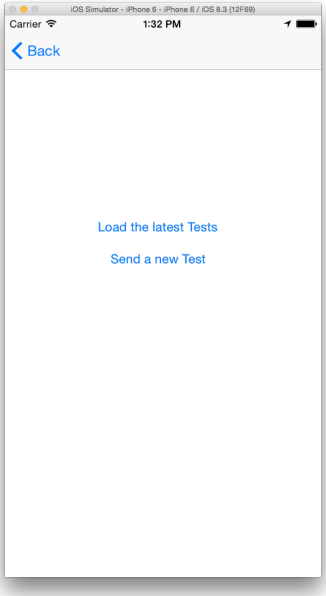



Abbildung 40: Ansicht von Send & Receive Tests

**Technologien** Für die Entwicklung der iOS-App wurde die Entwicklungsumgebung Xcode verwendet. Diese kann über den App Store von Mac OS bezogen werden. Das iOS-SDK für iOS 7 und iOS 8 lagen der Installation von Xcode bei. Als Programmiersprache wurde Objective-C verwendet.

**Entwicklung** Die Implementierung der Kameraansicht konnte mit wenigen Zeilen Code realisiert werden. Für diese Funktionen wurde auf das **AVFoundation Framework** zurückgegriffen, welches von Apple im iOS-SDK bereitgestellt wird. Für den Fall, dass die Kamera nicht verfügbar ist, wird ein Dialog angezeigt, der den Nutzer über diesen Fehler informiert. Dabei musste beachtet werden, dass es unter iOS 7 und iOS 8 zwei verschiedene Klassen für die Darstellung von Dialogen gibt. Es musste deswegen vorher erkannt werden, welches Betriebssystem das Gerät nutzt, um so die korrekte Klasse zu nutzen. Die Aufnahmefunktion konnte während des Arbeitspaketes nicht getestet werden, da die benötigte Hardware nicht vorhanden war.

Das Hochladen und Abrufen der Tests wurde mit Standard-Klassen des iOS-SDK implementiert. Die Klassen **NSURL**, **NSURLConnection** und **NSURLRequest** liefern dabei alle Funktionen um HTTP-Aufrufe mit Parametern und Inhalten auszuführen.

**Systembeschreibung** Die iOS-App wurde auf einem Mac Mini (2,8GHz Intel Core i5, 8 GB 1600MHz DDR3) in der Entwicklungsumgebung Xcode entwickelt. Die iOS-Prototyp-App wurde für die Version 7 und 8 entwickelt.

**Evaluation** Die iOS-Prototyp-App wurde nach dem Arbeitspaket durch das MEDIC-Team getestet und evaluiert, da erst zu diesem Zeitpunkt die benötigte Hardware zur Verfügung stand. Bei der Evaluation traten keine Probleme auf.

**Fazit** Die iOS-Prototyp-App konnte in der vorgegebenen Zeit von zwölf Personentagen umgesetzt werden. Der Funktionsumfang aus der Arbeitspaketbeschreibung wurde implementiert, jedoch wurde der Bildanalysealgorithmus nicht integriert, da dieser zu einem späteren Zeitpunkt innerhalb dieses Sprints überarbeitet wird.


Abbildung 41: Darstellung der Kameraansicht im iOS-Simulator

**Ausblick** Die folgenden Punkte sind relevant für die Weiterentwicklung der iOS-Prototyp-App:

- Der Bildanalysealgorithmus muss integriert werden.
- Der Auslöser der Kamera muss derzeit manuell betätigt werden. Zukünftig sollen Fotos automatisch aufgenommen werden, wenn ein QR-Code aus einer Echtzeit-Vorschau erkannt wird.
- Die iOS-Prototyp-App soll eine Diagnose für einen Test erhalten können, welche der Arzt über die Weboberfläche verfasst hat.

### 4.2.4 Erweiterung der Heatmap

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket umfasst eine Erweiterung der im vorherigen Sprint entwickelten Heatmap, um neue Funktionen. Diese sind Teil der Anforderungen.
- Arbeitspaketbeschreibung:
  Die bereits im ersten Sprint entwickelte Heatmap soll um zwei Funktionen erweitert werden:
  - Zeitliche Eingrenzung der angezeigten Einträge
  - Zeitrafferfunktion zur Visualisierung des zeitlichen Verlaufs der Krankheitsausbreitung

  Zur zeitlichen Eingrenzung soll ein Zeitraum mit Start- und Enddatum eingegeben werden können. Nach Bestätigung dieses Zeitraums werden auf der Heatmap lediglich Einträge angezeigt, die in diesem Zeitraum angelegt wurden.
  Darüber hinaus soll zur besseren Visualisierung der Krankheitsausbreitung eine Zeitrafferfunktion implementiert werden. Hierbei können, ausgehend von einem vom Anwender festzulegendem Startdatum, sukzessiv immer mehr Einträge angezeigt werden. Jeder Schritt stellt dabei einen festen Zeitsprung, zum Beispiel einen Tag, dar. Die Länge des Zeitsprungs kann ebenfalls vom Nutzer eingestellt werden.

  Die Deadline dieses Arbeitspaketes ist der 24. Juni 2015.

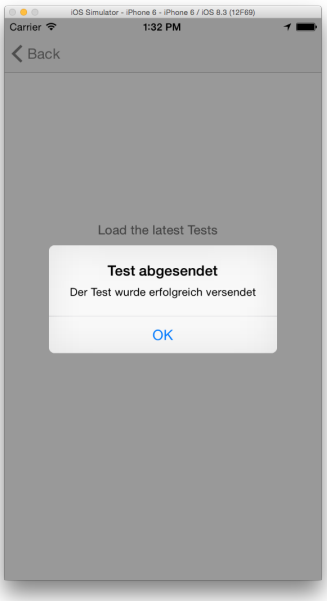



Abbildung 42: Darstellung, nachdem ein Beispiel-Test erfolgreich versendet wurde

- Vorbedingungen:
  Die Weboberfläche liegt in dem Zustand vor, die im vorherigen Sprint erarbeitet wurde.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Die aktualisierte Heatmap mit den neu entwickelten Funktionen steht zur Verfügung.
- Aufwand:
  Zwei Wochen
- Personen:
  - Daniel Wegmann
  - Jan Phillip Stubbe

**4.2.4.1 Dokumentation** Das Arbeitspaket befasst sich mit der Erweiterung der bereits vorhandenen Heatmap. Es wurde eine zeitliche Eingrenzung der betroffenen Fälle vorgenommen und die Möglichkeit geschaffen, die Ausbreitung in einem Zeitraffer zu betrachten. Das Arbeitspaket ergibt sich aus folgender Anforderung:

**FA-22** Die Auswertungsoberfläche muss eine interaktive Karte von Deutschland beinhalten. Diese geographische Auswertungskarte muss für ausgewählte Zeiten und Krankheiten eine Heatmap von Krankheitstests darstellen. Sie wird durch eine entsprechende Legende dokumentiert.

**Ablauf** Die Heatmap wurde im Zuge des Arbeitspakets um mehrere Funktionen erweitert. Zunächst wurde die **JQuery-Datepicker** Bibliothek eingebunden, welche es ermöglicht einen Zeitraum festzulegen, aus dem Ergebnisse auf der Karte angezeigt werden sollen. Das Auswählen und Anzeigen der entsprechenden Elemente wurde anschließend in JavaScript realisiert.
Zusätzlich wurde eine Zeitraffer-Funktion implementiert. Diese ermöglicht es den ausgewählten Zeitraum schrittweise zu durchlaufen. Die Größe und Anzahl der einzelnen Schritte ist dabei variabel

einstellbar. Darüber hinaus wurde die Leistung verbessert, indem nur noch die Daten aus der Datenbank geladen werden, deren Koordinaten in dem sichtbaren Bereich der Karte liegen.

**Systembeschreibung** Die grundlegende Funktionalität der Heatmap war bereits vorher implementiert. Aufgrund dessen wurde bei der Implementierung JavaScript als Programmiersprache beibehalten. Als Editoren wurden Notepad++ und SublimeText eingesetzt.

**Evaluation** Die korrekte Funktion wurde durch manuelle Tests sichergestellt. Diese Tests wurden von mehreren Personen unabhängig voneinander durchgeführt. Getestet wurde auf den neuesten Versionen von Chrome, Firefox und dem Internet Explorer 11.

**Fazit** Sowohl die zeitliche Eingrenzung der angezeigten Einträge, als auch die Zeitrafferfunktion wurden implementiert und das Paket abgeschlossen. Durch diese Eingrenzungen konnten bereits Leistungsvorteile in der Heatmap realisiert werden und die Karte wird flüssiger angezeigt.

**Ausblick** Die Heatmap erfüllt bereits alle geforderten Funktionen. Es ist allerdings zu empfehlen, in einer zukünftigen Version die Leistung noch weiter zu verbessern. Insbesondere bei einer großen Anzahl von Datensätzen wird die Darstellung langsam. Hier kann entweder die Datenmenge reduziert werden, oder es muss ein Weg entwickelt werden, um die Daten frühzeitig in den Speicher zu laden.

### 4.2.5 Optimierung der Kameraansicht in der Android-Applikation

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Das folgende Arbeitspaket verbessert die Nutzerfreundlichkeit der Android-Prototyp-App, indem die Funktionalitäten der Kameraintegration erweitert werden.
- Arbeitspaketbeschreibung:
  In diesem Arbeitspaket geht es darum, die Integration der Kamera in der Android-Prototyp-App zu verbessern. Diese Gesamtaufgabe teilt sich in folgende Unteraufgaben auf:

  1. Korrekte Darstellung des Seitenverhältnisses, um Verzerrungen zu vermeiden.
  2. Zentrierung der angepassten Kameraansicht. (Siehe Abbildung 43)
  3. Automatisierte Erkennung der Existenz eines QR-Codes mithilfe der Kamera.
  4. Anzeige von UI-Elementen als Überlagerung über der Kamera-Vorschau.

  Abbildung 43 zeigt, wie die Vorschau aus der Kamera skaliert werden muss, um bei gleichbleibendem Seitenverhältnis eine möglichst große Fläche des Bildschirms auszunutzen.


Abbildung 43: Differenzierung zwischen Vorschaugröße und Bildschirmgröße

Die Deadline dieses Arbeitspaketes ist der 01. Juli 2015.

- Vorbedingungen:
  Damit dieses Arbeitspaket umgesetzt werden kann, muss die Android-Prototyp-App aus *Sprint 2* bereits existieren. Des weiteren muss das zur Verfügung stehende Smartphone über eine Kamera verfügen. Da noch nicht klar ist, wie rechenaufwendig die QR-Code Erkennung in Echtzeit ist, benötigt dieses Smartphone eventuell überdurchschnittliche Hardware.

- Nebenbedingungen:
  -

- Nachbedingungen:
  Das Ergebnis des Arbeitspaketes ist eine verbesserte Android-Prototyp-App. Diese ist in der Lage, ein nicht verzerrtes Kamerabild anzuzeigen. Über diesem Kamerabild befindet sich eine Fadenkreuz, welches dem Benutzer anzeigt, an welcher Stelle der QR-Code positioniert werden soll. Des Weiteren kann die Applikation in Echtzeit einen QR-Code detektieren, ohne dass ein Foto manuell aufgenommen werden muss.

- Aufwand:
  Drei Wochen

- Personen:
  - Christoph Ressel
  - Nicolas Koch

**4.2.5.1 Dokumentation** Dieses Arbeitspaket befasst sich mit der Verbesserung der Kameraansicht in der Android-Prototyp-App. Dabei sind folgende funktionalen Anforderungen relevant:

**FA-10** Der Proband kann mit der mobilen Applikation ein Foto des Teststreifens aufnehmen. Dazu greift die mobile Applikation auf die Kamera des Smartphones zu, um das Bild aufzunehmen.

**FA-11** Die mobile Applikation ist in der Lage ein Foto zwischenzuspeichern.

**FA-12** Die mobile Applikation ist in der Lage zu erkennen, ob genügend Speicherplatz für das Foto vorhanden ist. Beim Speichern des Fotos wird geprüft, ob der Speicherplatz ausreicht.

Wie der Arbeitspaketbeschreibung zu entnehmen ist, ergeben sich folgende Anforderungen:

- Korrekte Darstellung des Seitenverhältnisses, um Verzerrungen zu vermeiden.
- Zentrierung der angepassten Kameraansicht.
- Automatisierte Erkennung der Existenz eines QR-Codes mithilfe der Kamera.
- Anzeige von UI-Elementen als Überlagerung auf der Kamera-Vorschau.

**Ablauf** Zu Beginn des Arbeitspaketes wurde die automatische Fokussierung des Kamerabildes implementiert. Hierdurch wird ein scharfes Kamerabild erreicht, welches die Erkennung von Elementen auf diesem, wie z.B. QR-Codes erleichtert. Daraufhin wurde sich darum gekümmert, das Seitenverhältnis des Kamerabildes so anzupassen, dass keine Verzerrungen auftreten. Hierzu musste unabhängig von der Bildschirmgröße auf allen Geräten ein einheitliches Seitenverhältnis erreicht werden. Dafür wurde das Kamerabild nicht abgeschnitten, sondern stattdessen skaliert und an die Bildschirmgröße angepasst. Anschließend wurde zur besseren Übersicht das Vorschaubild zentriert und der Hintergrund schwarz eingefärbt. Um den Quellcode lesbar zu halten, wurde das Projekt zum großen Teil neu organisiert.
Um eine automatisierte Erkennung des QR-Codes zu gewährleisten, wird direkt im Livestream der Kamera nach QR-Codes gesucht, ohne dass zunächst manuell ein Bild aufgenommen werden muss. Hierzu wurde eine externe Bibliothek hinzugezogen, die diese Funktionalität zur Verfügung stellt. Sobald ein QR-Code erkannt wurde, nimmt das Smartphone automatisch ein Foto auf. Dieses Bild wird anschließend von dem Bilderkennungsalgorithmus aus dem Arbeitspaket *Bildanalyse* analysiert. Zur verbesserten Nutzerführung wurde außerdem eine Überlagerung auf der Kamera-Vorschau eingeführt. Diese zeichnet dynamisch UI-Elemente über dem Vorschaubild der Kamera, wodurch in Zukunft der erkannte QR-Code angezeigt werden kann. Aktuell wird ein Quadrat angezeigt, welches dem Benutzer signalisiert wo sich auf dem Kamerabild der QR-Code befinden muss.

**Systembeschreibung** Siehe Sprint-Dokumentation Kapitel 4.1.8 auf Seite 53.

**Evaluation** Zum Test der Robustheit der QR-Code Erkennung wurde die Android-Prototyp-App auf folgenden mobilen Endgeräten installiert und ausgeführt:

- Google Nexus 4 (API 22)
- Samsung Galaxy S3 (API 18)
- Google Nexus 6 (API 22)

Auf allen drei Geräten ist die Erkennung von QR-Codes deutlich robuster geworden. Je nach Rechenleistung des Gerätes dauert die Erkennung unterschiedlich lang, allerdings niemals länger als fünf Sekunden nach Fokussierung der Kamera auf den QR-Code. Diese Tests wurden bei guten räumlichen Lichtverhältnissen durchgeführt. QR-Codes werden zuverlässig erkannt, sobald diese mindestens 10% des Kamerabildens bedecken.

**Parameter**

- helle räumliche Umgebung
- QR-Code bedeckt 10% des Kamerabildes
- Smartphone bietet Funktionalität *Auto-Fokus*

**Fazit** Durch die Durchführung dieses Arbeitspaketes wurde jede Anforderung erfüllt. Der vorher geschätzte Aufwand von zwölf Personentagen stellte sich als zu groß heraus. Das Arbeitspaket konnte schon nach acht Personentagen abgeschlossen werden.
Der größte zu verzeichnende Fortschritt, ist die deutlich schlankere und robustere QR-Code-Erkennung aus dem Livestream der Kamera, welche unter anderem durch die Verwendung der **QRCodeReaderView** Bibliothek erreicht werden konnte.

**Ausblick** Folgende Arbeitspakete können auf der QR-Code-Erkennung dieses Arbeitspaketes aufbauen. Insbesondere die Farbumschlagserkennung, die während der zweiten Hälfte dieses Sprints umgesetzt wird, profitiert stark von den Ergebnissen dieses Arbeitspaketes, da die Parameter *Codierter String* und *Eckpunkte des QR-Codes* von dieser Implementierung geliefert werden.

### 4.2.6 Optimierung der Kamera-Ansicht in der iOS-Applikation

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Das folgende Arbeitspaket soll die Nutzerfreundlichkeit der iOS-Prototyp-Applikation verbessern, indem die Funktionalitäten der Kameraansicht erweitert werden.
- Arbeitspaketbeschreibung:
  In diesem Arbeitspaket geht es darum, die Integration der Kamera in die iOS-Prototyp-App zu verbessern. Diese Gesamtaufgabe teilt sich in folgende Unteraufgaben auf:

  1. Automatisierte Erkennung der Existenz eines QR-Codes mithilfe der Kamera.
  2. Anzeige von UI-Elementen als Überlagerung über der Kamera-Vorschau.
  3. Automatisierte Aufnahme und Speicherung eines Fotos mit QR-Code.

  Zur Umsetzung dieses Arbeitspaketes ist es außerdem nötig, die vorhandene iOS-Prototyp-App stellenweise zu verändern.

  Die Deadline dieses Arbeitspaketes ist der 01. Juli 2015.

- Vorbedingungen:
  Damit dieses Arbeitspaket umgesetzt werden kann, muss die iOS-Prototyp-App aus *Sprint 3 (iOS_Implementierung)* bereits existieren. Des Weiteren muss mindestens ein iPhone zur Verfügung stehen, welches über eine Kamera verfügt. Zur Entwicklung der iOS-Prototyp-App wird außerdem ein Mac mit Xcode benötigt.
- Nebenbedingungen:
  -

- Nachbedingungen:
  Das Ergebnis des Arbeitspaketes ist eine verbesserte iOS-Prototyp-App. Diese ist in der Lage, ein Kamerabild anzuzeigen. Über diesem Kamerabild befindet sich eine Fadenkreuz welches die Position, in dem sich der aufzunehmende QR-Code befinden muss, anzeigt. Des Weiteren kann die iOS-Prototyp-App in Echtzeit einen QR-Code detektieren, ohne dass ein Foto manuell aufgenommen werden muss. Ausgabeparameter des Algorithmus sind zum einen die Eckkoordinaten des QR-Codes im Foto und zum anderen der dekodierte Text.

- Aufwand:
  Eine Woche

- Personen:

  – Jan Philipp Stubbe
  – Nicolas Koch

**4.2.6.1 Dokumentation** Dieses Arbeitspaket befasst sich mit der Verbesserung der Kameraansicht in der iOS-Prototyp-App. Dabei sind folgende funktionalen Anforderungen relevant:

**FA-10** Der Proband kann mit der mobilen Applikation ein Foto des Teststreifens aufnehmen. Dazu greift die mobile Applikation auf die Kamera des Smartphones zu, um das Bild aufzunehmen.

**FA-11** Die mobile Applikation ist in der Lage ein Foto zwischenzuspeichern.

**FA-12** Die mobile Applikation ist in der Lage zu erkennen, ob genügend Speicherplatz für das Foto vorhanden ist. Beim Speichern des Fotos wird geprüft, ob der Speicherplatz ausreicht.

Wie der Arbeitspaketbeschreibung zu entnehmen ist, ergeben sich folgende Anforderungen:

- Automatisierte Erkennung der Existenz eines QR-Codes mithilfe der Kamera.
- Anzeige von UI-Elementen als Überlagerung auf der Kamera-Vorschau.
- Automatisierte Aufnahme und Speicherung eines Fotos mit QR-Code.

**Ablauf** Dieses Arbeitspaket wurde anhand einer online verfügbaren Anleitung umgesetzt. Der Ablauf kann dieser entnommen werden. Die Anleitung befindet sich auf dem Webblog von Sam Davies.[7]

**Technologien**

AVFoundation[8]

**Systembeschreibung** Siehe Sprint-Dokumentation *Implementierung des Prototyp-Quellcodes in Objective-C* (Kapitel 4.2.3.1 auf Seite 68).

**Evaluation** Zum Test der Robustheit der QR-Code Erkennung wurde die iOS-Prototyp-App auf einem iPhone 4 installiert und ausgeführt.
Auf dem genannten Gerät ist die QR-Code Erkennung bei räumlichen Lichtverhältnissen sehr zuverlässig. Unter Umständen kann die Erkennung eines Codes bis zu fünf Sekunden nach Fokussierung der Kamera auf den QR-Code dauern. Die Codes werden zuverlässig erkannt, sobald diese mindestens 10% des Kamerabildes bedecken.

[7] `https://www.shinobicontrols.com/blog/ios7-day-by-day-day-16-decoding-qr-codes-with-avfoundation`
[8] `http://developer.apple.com/av-foundation/`

**Parameter**

- helle räumliche Umgebung
- QR-Code bedeckt 10% des Kamerabildes
- Smartphone bietet Funktionalität *Auto-Fokus*

**Fazit** Durch die Durchführung dieses Arbeitspaketes wurde jede Anforderung erfüllt. Die Deadline von vier Personentagen wurde gehalten. Der größte zu Fortschritt ist die automatische QR-Code Erkennung aus dem Livestream der Kamera.

**Ausblick** Nachfolgende Arbeitspakete können auf der QR-Code Erkennung dieses Arbeitspaketes aufbauen. Insbesondere die Farbumschlagerkennung, die während der zweiten Hälfte dieses Sprints umgesetzt wird, profitiert stark von den Ergebnissen dieses Arbeitspaketes, da die Parameter *Codierter String* und *Eckpunkte des QR-Codes* von dieser Implementierung geliefert werden.

### 4.2.7 Entwicklung einer Diagnose-Funktion für Mediziner

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket sieht vor, dass zu durchgeführten Krankheitstests Diagnosen formuliert und abgespeichert werden können. Zusätzlich sollen die Android-Prototyp-App und die iOS-Prototyp-App diese Diagnosen anzeigen können.
- Arbeitspaketbeschreibung:
  In diesem Arbeitspaket soll für die Weboberfläche eine Eingabemaske erstellt werden, mit der Diagnosen zu den durchgeführten Krankheitstests der Nutzer geschrieben werden können. Diese Maske beinhaltet eine Auswahl an folgenden Diagnoseergebnissen:

  – positiv
  – negativ
  – unbekannt

  Neben dem Diagnoseergebnis ist auch ein Textfeld vorgesehen, in welchem ein Kommentar zur Diagnose formuliert werden kann. Über eine weitere Schaltfläche ist es möglich, die vorgenommene Diagnose an den Server zu übertragen.
  Um in der Weboberfläche geschriebene Diagnosen zu speichern, müssen neue Funktionen auf dem Server implementiert werden. Dies beinhaltet die Entwicklung einer weiteren Schnittstelle, welche Diagnosen entgegen nimmt und diese anschließend in neu anzulegende Datenbankspalten ablegt. Außerdem wird eine Schnittstelle benötigt, über welche Diagnosen zu bestimmten Krankheitstests abgefragt werden können.
  Um die Diagnosen von durchgeführten Krankheitstests auf der Android-Prototyp-App und der iOS-Prototyp-App zu empfangen, ist es nötig, die IDs der absolvierten Tests auf den Geräten zu speichern. Zusätzlich muss eine Diagnose-Ansicht entwickelt werden, welche eine Liste aller durchgeführten Krankheitstests darstellt und die vom Server abgefragten Diagnosen anzeigt.

  Die Deadline dieses Arbeitspaketes ist der 15. Juli 2015.
- Vorbedingungen:
  Vorbedingung dieses Arbeitspaket ist es, dass die im zweiten Sprint erstellte Server-Applikation, die Android-Prototyp-App, die iOS-Prototyp-App und die Weboberfläche im Git vorliegen und modifiziert werden können.

- Nebenbedingungen:
  -
- Nachbedingungen:
  Am Ende dieses Arbeitspakets werden Versionen der Server-Applikation, der Android-Prototyp-App, der iOS-Prototyp-App sowie der Weboberfläche bereitgestellt, welche die oben beschriebenen Diagnose-Funktionen enthalten.
- Aufwand:
  Drei Wochen
- Personen:
  - Kevin Sandermann
  - Timo Schlömer

**4.2.7.1 Dokumentation** Das Arbeitspaket bezieht sich auf folgende Anforderungen aus dem Lastenheft:

**FA-16** Die mobile Applikation muss die Diagnose des Mediziners zum durchgeführten Test vom Server empfangen können.

**FA-19** Sobald der Server von der Auswertungsoberfläche eine Diagnose erhält, ist er in der Lage diese Diagnose der mobilen Applikation des richtigen Probanden zuzuordnen.

**FA-26** Bei der Diagnose eines Tests soll es die Auswertungsoberfläche dem Mediziner erlauben den Test als positiv, negativ oder unlesbar zu erklären.

**Ablauf** Zu Beginn des Arbeitspakets wurde mit der Arbeit am Server begonnen, um das Senden und Empfangen der Diagnosedaten zu ermöglichen. Die Datenbank wurde angepasst, um diese Daten verwalten zu können. Im Anschluss wurde die Weboberfläche bearbeitet, welche nun die Eingabe einer Diagnose ermöglicht. Dafür wurde der bisherige Dialog, welcher die Daten eines gesendeten Krankheitstests anzeigt, um weitere Eingabefelder erweitert. Hier können nun alle benötigten Daten für eine Diagnose eingegeben und gesendet werden. Abbildung 44 auf der nächsten Seite zeigt diese Oberfläche.

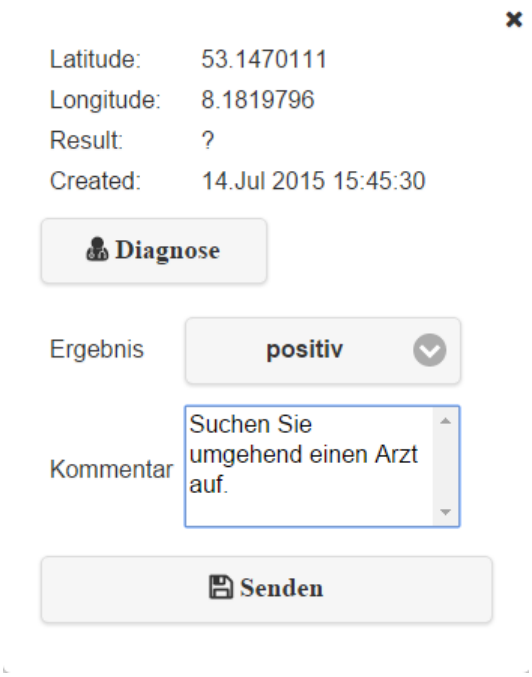



Abbildung 44: Die Diagnose-Eingabemaske auf der Web-Oberfläche

Nachdem die Weboberfläche angepasst war, wurde die Android-Prototyp-App erweitert, um eine Liste aller versendeten Krankheitstests und deren aktuellen Zustand anzuzeigen. Dazu war es nötig, nach jedem versendeten Test die vom Server empfangene Identifikationsnummer lokal zu speichern. Mithilfe dieser Nummer kann die App den aktuellen Zustand des Krankheitstests am Server abfragen. Wenn die Tests in der App angezeigt werden sollen, werden diese Anfragen an den Server gesendet.

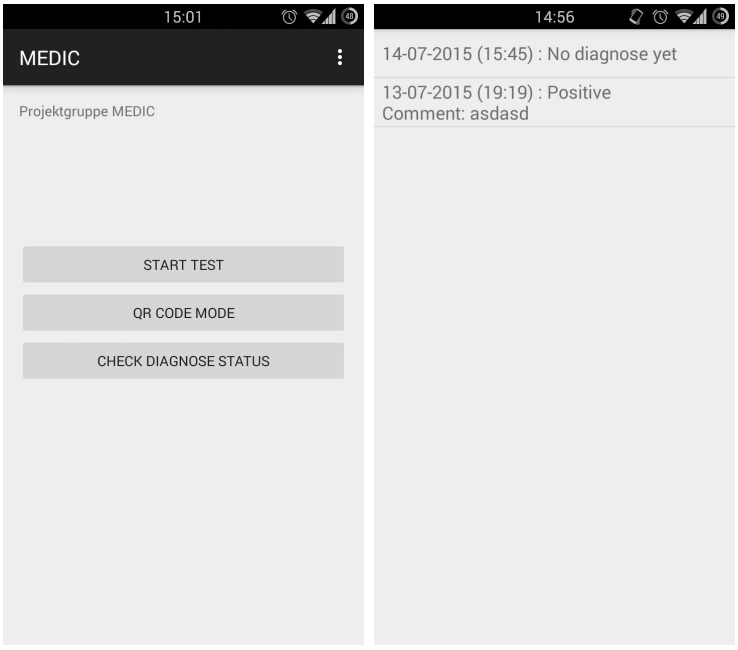



Abbildung 45: Die Android App zeigt den Zustand der gesendeten Tests

Auf die gleiche Art wurde danach die iOS-Prototyp-App angepasst. Auch hier werden die Identifikationsnummern der Krankheitstests lokal gespeichert, um den Zustand dieser später abrufen zu können. Die Tests werden auch in einer einfachen Liste dargestellt. Die Abbildungen 45 und 46 auf der nächsten Seite zeigen diese Oberflächen für die jeweiligen Apps.


Abbildung 46: Die iOS App zeigt den Zustand der gesendeten Tests

**Technologien**

**jQuery** Es wird jQuery eingesetzt, um die Bedienelemente auf der Weboberfläche zu manipulieren und deren Daten auszulesen.

**AngularJS** Es wird AngularJS eingesetzt, um ein Model-View-Controler Prinzip auf der Weboberfläche umzusetzen.

**JSON** Für die Datenübertragung werden alle Daten in das JSON Format übersetzt.

**REST** Die Schnittstellen auf dem Server werden über REST angesprochen.

**Preferences (Android)** Anstelle einer Datenbank werden die Identifikationsnummern der Tests in Android über das **Preferences** System gespeichert.

**NSUserDefaults (iOS)** Anstelle einer Datenbank werden die Identifikationsnummern der Tests in iOS über das **NSUserDefaults** System gespeichert.

**Postgres** Die Datenbank, welche zum Speichern der Tests und Diagnosen verwendet wird.

**Spring** Wird innerhalb einer Java Anwendung auf dem Server verwendet, um die REST Schnittstelle bereitzustellen und mit der Datenbank zu kommunizieren.

**Systembeschreibung** Siehe Abschlussbericht *RA*, *DB_API*, *SO*, *App-Datenbank* und *Schnittstelle Auswertungsoberfläche - Datenbank* aus Sprint 2.

**Evaluation** Die Weboberfläche wurde unter den Browsern Firefox, Internet-Explorer und Google Chrome getestet. Die Android-Prototyp-App wurde mit einem Google Nexus 4 und einem Samsung Galaxy S3 getestet. Die iOS-Prototyp-App wurde auf einem iPhone 4 getestet.

**Parameter** Die Systeme Weboberfläche, Android-Prototyp-App und iOS-Prototyp-App laufen unter den gleichen Parametern des vorherigen Sprints.

**Fazit** Alle geplanten Anforderungen konnten in diesem Arbeitspaket erfüllt werden und die dafür benötigte Zeit lag innerhalb des dafür vorgesehenen Zeitrahmens von zwölf Personentagen.

**Ausblick** Die Apps für Android und iOS zeigen die gesendeten Krankheitstests nach Abschluss des Arbeitspakets in einer Liste an, welche sich aktualisiert, sobald man diese betrachtet. Als nächstes sollten die Apps in der Lage sein, den Zustand der Krankheitstests in regelmäßigen Intervallen abzufragen, um dem Nutzer eine Benachrichtigung zu geben, falls sich der Zustand des Tests ändert. Auch ist die Eingabemaske für das Erstellen einer Diagnose sehr rudimentär und sollte in folgenden Arbeitspaketen verschönert und benutzerfreundlicher gestaltet werden.

### 4.2.8 Bildverarbeitungsalgorithmus in C

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  In diesem Arbeitspaket wird der in Java existierende Bildverarbeitungsalgorithmus in die Programmiersprache C++ konvertiert und um weitere Funktionalitäten ergänzt. Dies ist Notwendig, wenn zukünftig mit der Bildverarbeitungs-Bibliothek OpenCV gearbeitet werden soll.
- Arbeitspaketbeschreibung:
  In diesem Arbeitspaket geht es darum, den Bildverarbeitungsalgorithmus, der den Farbumschlag auf dem Teststreifen detektieren soll, zu verbessern bzw. neu zu implementieren. Um den Quellcode einheitlich für Android und iOS zu entwickeln, wird der Algorithmus in C++ implementiert und durch das Arbeitspaket „Integration des C Algorithmus“ in die jeweiligen Betriebssysteme integriert.
  Der Algorithmus soll so implementiert werden, dass er den Farbumschlag in der Größe detektieren soll, wie er auf einem Schwangerschaftsteststreifen zu finden ist. Als Vorlage dazu dient der Teststreifen der technischen Partneruniversität Braunschweig. Auf dem Teststreifen wird es einen QR-Code geben, der unter anderem mit den folgenden Informationen ausgestattet ist: die Position, an der der Farbumschlag detektiert werden soll und die Farbe, die detektiert werden soll. Außerdem gibt es einen Referenzpunkt, der dazu dient, äußere Lichteinflüsse zu negieren. Es werden folgende Anforderungen berücksichtigt, die bei der Implementierung wichtig sind:
  - Die mobile Applikation verfügt über einen QR-Code-Erkennungs-Algorithmus, mit dem eine vorläufige Detektion des Teststreifens vorgenommen wird.

  Die Deadline dieses Arbeitspaketes ist der 22. Juli 2015.
- Vorbedingungen:
  Um die vollständige Funktionsfähigkeit zu gewährleisten, muss die Integration in Android und iOS abgeschlossen sein.
- Nebenbedingungen:
  -
- Nachbedingungen: Das Ergebnis des Arbeitspaketes ist die Implementierung eines Bildverarbeitungsalgorithmus in C++. Dieser soll den Farbumschlag in der Größe des Teststreifens erkennen und daraus das Testergebnis berechnen können. Dabei wird das Design des Teststreifens, wie es bisher bekannt ist, überarbeitet.
- Aufwand:
  Vier Wochen
- Personen:
  - Christoph Ressel
  - Christian Sandmann

**4.2.8.1 Dokumentation** Dieses Arbeitspaket befasst sich mit der Implementierung des Bildverarbeitungsalgorithmus in der Programmiersprache C bzw. C++. Wie der Arbeitspaketbeschreibung zu entnehmen ist, ergibt sich folgende Anforderung:

- Die mobile Applikation verfügt über einen Bildverarbeitungsalgorithmus, mit dem eine vorläufige Auswertung des Teststreifens vorgenommen werden kann.

**4.2.8.2 Ablauf** Zur Detektion des Farbumschlags wurde der HSV-Farbraum verwendet, da sich mit seiner Hilfe das Bild leicht in verschiedene Farbbereiche separieren lässt. Da eine Überprüfung des gesamten Bildes die Durchführungsdauer des Algorithmus stark erhöhen würde, wird das Bild nur in einem bestimmten Bereich, der relativ zum QR-Code liegt, abgesucht. Dazu wird zunächst eine feste Größe für den QR-Code angenommen und relativ zu seinem Mittelpunkt die Zielkoordinaten gesetzt. In diesem Zielbereich wird ein viereckiges Polygon aufgespannt, welches den Bereich abdeckt, der überhaupt für den Farbumschlag in Frage kommt. Zur Korrektur der äußeren Lichteinflüsse wird zusätzlich der durchschnittliche Farbwert eines Referenzbereichs ermittelt, von dem durch Attribute aus dem QR-Code bekannt ist, welchen Farbwert dieser Bereich aufweisen sollte. Die Diskrepanz zwischen diesem ermittelten Wert und dem eigentlich vorhandenen Wert wird als Korrekturwert in die Auswertung der Pixel des eigentlich zu detektierenden Farbumschlags mit einbezogen.
Während der Bearbeitung dieses Arbeitspakets sind verschiedene Probleme aufgetaucht. Auf Grund falscher Planung der Arbeitspaketlaufzeiten konnte erst spät (nach 3 von 4 Wochen) mit dem Arbeitspaket „Integration C Algorithmus“ die notwendige Schnittstelle kommuniziert werden. Dabei stellte es sich heraus, dass entgegen der Annahme, dass der Farbumschlagsbereich bereits von der Smartphone-Applikation vorgegeben sein würde, die Bildverarbeitungsimplementierung selbst diesen Bereich dynamisch lokalisieren muss. Diese Detektierung konnte bis zum Ende des Sprints nicht beendet werden, wodurch für die Präsentation des Sprintergebnisses feste Koordinaten für diesen Bereich verwendet werden mussten. Das Hauptproblem besteht darin, dass die Größe des QR-Codes variieren kann, sobald man eine andere Eingabe kodiert.
Das größte Problem bei der Entwicklung eines zuverlässigen Bildverarbeitungsalgorithmus ist die zu diesem Zeitpunkt noch viel zu geringe Kenntnis über den Teststreifen und den Farbumschlag. Daher hat es noch keinen Sinn ergeben komplexere Bildverarbeitungsverfahren wie z.B. Regiongrowing[9], Opening[10]/Closing[11], Muster-/Formerkennung etc. zu implementieren, da nicht bekannt ist, ob der Farbumschlag eine bestimmte Form aufweist bzw. überhaupt zusammenhängend ist oder nicht. Auch ist nicht bekannt ab welchen Parametern ein Test ein bestimmtes Ergebnis liefern soll, also z.B. ab wievielen als positiv/negativ detektierten Pixeln im Farbumschlagsbereich das Testergebnis zu 99,9 prozentiger Sicherheit richtig. Auch ist noch völlig unklar welchen Einfluss die Sättigung des Farbwerts auf das Ergebnis hat.

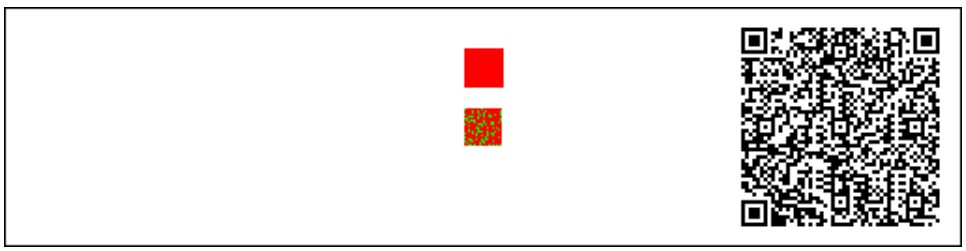


Abbildung 47: Teststreifen generiert vom Teststreifengenerator

[9] `https://de.wikipedia.org/wiki/Region_Growing`, zuletzt geöffnet am 19.07.2015
[10] `https://de.wikipedia.org/wiki/Opening_(Bildverarbeitung)`, zuletzt geöffnet am 19.07.2015
[11] `https://de.wikipedia.org/wiki/Closing`, zuletzt geöffnet am 19.07.2015

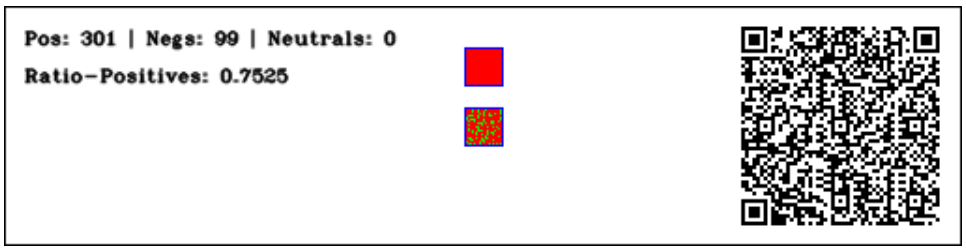


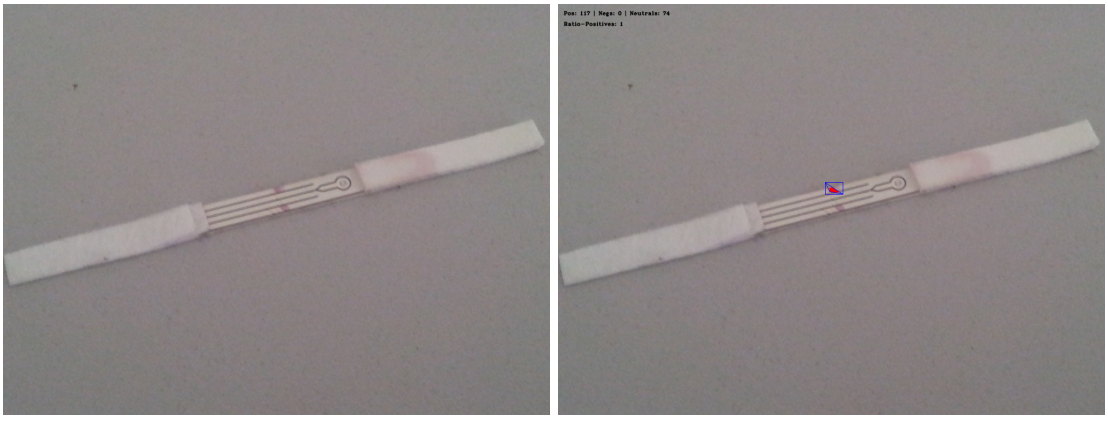

Abbildung 48: Teststreifen mit Analyseergebnis

#### 4.2.8.3 Technologien

**OpenCV** Weit verbreitete in C/C++ implementierte Bibliothek für Bildverarbeitung

**4.2.8.4 Systembeschreibung** Das System wurde auf Grund der umfangreicheren Funktionalität und Vorkenntnisse der Entwickler in C++ statt in C entwickelt. Zur Unterstützung wurde die Bildverarbeitungsbibliothek OpenCV herangezogen und als Entwicklungsoberfläche diente Microsoft Visual Studio 2013.

**4.2.8.5 Evaluation** Zur Evaluierung verschiedenster Testfälle wurde zusätzlich, zu den für das Arbeitspaket geforderten Funktionalitäten, ein Teststreifengenerator entwickelt, der abhängig von der Nutzereingabe einen Teststreifen in der Größe 512 mal 128 Pixel generiert und in einem fest definierten Bereich, Pixel farblich entweder als positiv oder negativ kennzeichnet. In Abbildung 47 auf der vorherigen Seite sieht man einen Teststreifen, der von diesem Generator erzeugt wurde. In Abbildung 48 wurde dieser Test mit dem Algorithmus analysiert. Es wurde ausgegeben, wie hoch die Anteile für positiv und negativ sind.
Der Algorithmus wurde auch erfolgreich auf eine Fotoaufnahme von den durch Braunschweig zur Verfügung gestellten Schwangerschaftstest durchgeführt (Abbildung 49), jedoch mussten hierfür entsprechende feste Parameter für den Farbumschlagsbereich gesetzt werden. Da das Arbeitspaket „Integration C Algorithmus“ nicht abgeschlossen wurde, konnte auch der entwickelte C++-Code nicht unter realen Bedingungen im Smartphone-Prototypen getestet werden.



(a) Vorher (b) Nachher

Abbildung 49: Schwangerschaftstests vor und nach Farbumschlagsdetektion

**4.2.8.6 Parameter** Mit dem Teststreifengenerator konnten verschiedene Teststreifen ausgewertet werden, die verschiedene Anteile an Farben für positiv, negativ und unverändert enthielten.

**4.2.8.7 Fazit** Entsprechend der Arbeitspaketbeschreibung konnte ein Programm in C++ entwickelt werden, dass in einem momentan noch fest definierten Bildbereich den Farbumschlag auswertet. Äußere Lichteinflüsse werden in dem aktuellen Stand beachtet, jedoch kann der Algorithmus an dieser Stelle noch verbessert werden.

**4.2.8.8 Ausblick** Auf diesem Algorithmus können weitere Pakete aufbauen. Die Grundfunktionalität wurde implementiert, doch ist diese noch zu verbessern, insbesondere in Hinblick auf variierende Farbumschlagsbereiche, Performance und Robustheit. Außerdem sollte statt der momentanen Auswertung des Referenzbereichs, ein standardisiertes Verfahren zum Weißabgleich implementiert werden. So werden z.B. in der Fotografie sogenannte Graukarten[12] verwendet um äußere Lichteinflüsse zu eliminieren. Allein hierfür kann also ein neues Arbeitspaket für die nächsten Sprints entstehen.

### 4.2.9 Integration des in C geschriebenen Algorithmus in die mobilen Applikationen

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Das folgende Arbeitspaket beinhaltet die Integration des in C geschriebenen Bilderkennungsalgorithmus aus *Sprint 3 (Bildverarbeitungsalgorithmus in C)* in die Android und iOS App. Dies soll im späteren Verlauf des Projekts den Wartungsaufwand für den Bilderkennungsalgorithmus verringern.
- Arbeitspaketbeschreibung:
  Der in C geschriebene Bilderkennungsalgorithmus muss jeweils aus dem Code der Android und der iOS App aufgerufen und ausgeführt werden können. Dafür ist es zunächst notwendig, dass eine Schnittstelle definiert wird, welche von beiden Apps die gleichen Parameter übergeben bekommt. Da der C-Code Abhängigkeiten zu OpenCV hat, muss die Bibliothek zusätzlich so eingebunden werden, dass es aus dem C-Code heraus aufgerufen werden kann.

  Mit der neuen Version 1.3 von Android Studio soll es einfacher werden, C-Programmcode in Android Applikationen einzubinden. Deshalb soll für dieses Arbeitspaket bereits diese Beta-Version von Android Studio eingesetzt werden. Damit ist zu erwarten, dass die Integration schneller als mit der alten Version funktioniert und auch zukunftsfähiger ist.

  Die Deadline dieses Arbeitspaketes ist der 15. Juli 2015.
- Vorbedingungen:
  Damit dieses Arbeitspaket umgesetzt werden kann, müssen die iOS-Prototyp-App aus *Sprint 3 (iOS_Implementierung)* und die Android-Prototyp-App aus *Sprint 2 (Mobile Applikation - Datenbank)* bereits existieren. Des Weiteren müssen ein iOS-Gerät und Android-Gerät, welche jeweils über eine Kamera verfügen, zum Testen vorliegen.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Ergebnis des Arbeitspaketes sind die Android-Prototyp-App und iOS-Prototyp-App, die eine Bildanalyse mit dem in C geschriebenen Algorithmus durchführen können. Der C-Code soll dafür nicht angepasst werden, sondern über die jeweiligen Schnittstellen aufgerufen werden.
- Aufwand:
  Zwei Wochen

[12] `https://de.wikipedia.org/wiki/Graukarte`

- Personen:
  - Jan Philipp Stubbe
  - Nicolas Koch

**4.2.9.1 Dokumentation** Das Arbeitspaket bezieht sich auf folgende Anforderungen aus dem Lastenheft:

**FA-4** Die mobile Applikation verfügt über einen Bildverarbeitungsalgorithmus, mit dem eine vorläufige Auswertung des Teststreifens vorgenommen werden kann.

**Ablauf** Dieses Arbeitspaket wurde in kleinere Teilpakete eingeteilt, um den Ablauf zu Strukturieren. Diese lauten wie folgt:

1. Definition einer Schnittstelle des *Foreign Function Interfaces (FFI)*
2. Aufruf von arbiträrem nativem Code.
3. Bereitstellung der Parameter (siehe 1.) durch die mobilen Applikationen.
4. Aufruf des C-Algorithmus aus dem Quellcode der Android-Prototyp-App und iOS-Prototyp-App.

Diese Punkte wurden chronologisch abgearbeitet. Da es zu Problemen in der Umsetzung kam, konnte Teilpaket vier nicht innerhalb der Deadline umgesetzt werden. Generell war die Einbindung von arbiträrem nativen Code bei beiden Apps kein Problem. Die Herausforderung entstand, als der native Code eine Abhängigkeit zu OpenCV aufwies. Da es für den Entwickler unzumutbar ist, bei jedem Kompiliervorgang der App auch die gesamte OpenCV Bibliothek zu kompilieren, wurden als Lösung Binärdateien der Bibiothek eingesetzt. Diese sind Architekturabhängig und werden somit in verschiedensten Ausführungen mitgeliefert.
Diese Integration war bis einschließlich der Einbindung der Headerdateien ebenfalls möglich. Das Problem entstand in der Linkphase des Programms, in welcher der Linker nicht die Implementierungen von OpenCV finden konnte. Dies trat sowohl in Android als auch in iOS auf. Da dieses Problem nicht in der geplanten Zeitspanne des Arbeitspaketes gelöst werden konnte, konnte die gesetzte Frist nicht eingehalten werden.
Obwohl das Arbeitspaket nicht fertiggestellt wurde, konnten neue Kenntnisse gewonnen werden. Diese werden in den folgenden Unterkapiteln aufgeführt.

**Technologien**

**JNI** Das Java Native Interface wird benötigt, um nativen Code unter Android auszuführen

**Android Studio 1.3** Diese Beta-Version (Zeitpunkt 18.07.2015) von Android Studio bringt zusätzliche Unterstützung für C und C++.

**Objective-C++** Um die Bindings zwischen C++ und der iOS App zu schaffen, müssen Objective-C++ Header-Files verwendet werden.

**Clang, GCC** Die am weitest verbreiteten Compiler für nativen Code.

**OpenCV** Eine Bildverarbeitungsbibliothek.

**Systembeschreibung** Um Codeduplikation zu vermeiden, sollte das Ziel erreicht werden, den Bilderkennungsalgorithmus der Apps in plattformunabhängigem C++-Code zu schreiben. Die Apps würden dann über eine klar definierte Schnittstelle (siehe Kapitel 4.2.9.1 Unterkapitel *Parameter*) mit dem nativem Teil der Applikationen kommunizieren.

**Evaluation** Da das Arbeitspaket nicht abgeschlossen werden konnte, hat keine Evaluation stattgefunden.

**Parameter** Es wurde eine Schnittstelle definiert, welche die Kommunikation innerhalb einer mobilen Applikation zwischen Java bzw. Objective-C und C beschreibt. Das Codebeispiel 1 zeigt diese Schnittstelle. Der genaue Name der Funktion ist noch offen gehalten worden und ist in dem Beispiel lediglich exemplarisch.

```
float get_sickness_percentage
        (char* qr_string, float* points,
         char** image_data, int image_data_len);
```

Listing 1: API des nativen Teils der Anwendungen

Die Parameter der Schnittstelle sind wie folgt definiert:

**qr_string** Dekodierter String, welcher aus dem QR-Code ausgelesen wurde

**points** Relative Position des QR-Codes auf dem Bild (Eckpunkte)

**image_data** Zwei-dimensionales Byte-Array der dekodierten Bilddaten

**image_data_len** Länge von `image_data`

**return** Zahl zwischen 0 und 1, welche die Wahrscheinlichkeit krank zu sein repräsentiert

**Fazit** Durch die beschriebenen Hindernisse in der Umsetzung konnte das Arbeitspaket nicht vollständig umgesetzt werden. Die Teilpakete eins bis drei wurden erfolgreich abgeschlossen, während Teilpaket vier scheiterte.

**Ausblick** Als Notlösung wird vorerst der Bilderkennungsalgorithmus unabhängig für Android und iOS implementiert werden. Dies ist im Anbetracht der bevorstehenden Kundenpräsentation die robusteste Variante. Eine eventuelle Integration des C-Codes in die Apps wird auf einen zukünftigen Sprint verschoben.

#### 4.2.10 Testdatengenerator

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Um die Weboberfläche und auch die Datenbank effizient Testen zu können, werden Testdaten benötigt. Diese sollen mittels eines Testdatengenerators automatisch generiert werden können.
- Arbeitspaketbeschreibung:
  Dieses Arbeitspaket beschreibt die Entwicklung eines sog. Testdatengenerators für den Prototypen. Dieser Testdatengenerator soll aus definierten Parametern wie etwa Ort und Zeit eine bestimmte Anzahl an Testdaten generieren und an den Server schicken. Anschließend sollen möglichst realistisch weiterführende Testdaten zeitlich versetzt generiert werden, welche etwa durch Ansteckungen entstehen. Um diese Testdaten mit einem bestimmten Zeitpunkt definiert

generieren zu können, muss hierfür zunächst eine passende Schnittstelle im Server implementiert werden. Neben der Implementierung dieser Schnittstelle und der Entwicklung eines geeigneten Ausbreitungsverfahrens umfasst dieses Arbeitspaket außerdem die Nutzung eines geeigneten APIs um auf statistische Daten wie spezifische Bevölkerungsdichten und Zuordnungen von Postleitzahlen zu Städtenamen zugreifen zu können.
Die vom Testdatengenerator generierten Daten sollen anschließend in der Heatmap angezeigt werden, um eine Pandemie zu simulieren. In der Weboberfläche der Mediziner sollen diese generierten Tests direkt, wie in Abbildung 50 dargestellt, markiert sein.

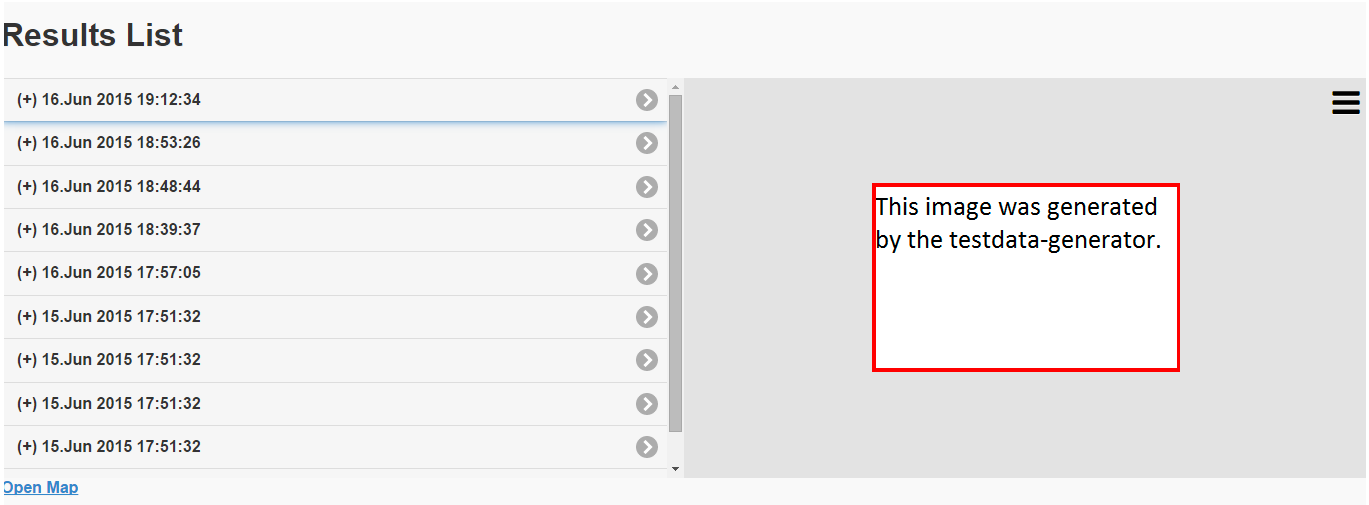



Abbildung 50: Generierter Datensatz auf der Weboberfläche

Die Deadline dieses Arbeitspaketes ist der 24. Juni 2015.

- Vorbedingungen:
  Um dieses Arbeitspaket durchführen zu können, muss der in Sprint_2 entwickelte Server online zur Verfügung stehen und entsprechende Testdaten per API annehmen.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Es wird ein Testdatengenerator bereitgestellt, welcher plattformunabhängig eingesetzt werden kann.
- Aufwand:
  Zwei Wochen
- Personen:
  - Kevin Sandermann
  - Raphael Kappes

**4.2.10.1 Dokumentation** Der Testdatengenerator setzt keine konkrete funktionale Anforderung um, sondern dient als Werkzeug, um die Einhaltung nicht-funktionaler Anforderungen zu gewährleisten. Der Testdatengenerator erstellt automatisiert durchgeführte Krankheitstests und sendet diese an den Server. Diese angelegten Tests können anschließend auf der Weboberfläche und

der Heatmap betrachtet werden. Somit stellt der Testdatengenerator eine Möglichkeit dar, die Komponenten Server, Serverschnittstelle, Datenbank, Weboberfläche und Heatmap jederzeit zu testen. Durch diese durchgehende Testmöglichkeit kann ein bestimmter Qualitätsstandard sichergestellt werden.

**Ablauf** Im Testdatengenerator muss eine Simulation für das Auftreten von Krankheiten in Deutschland möglichst realistisch implementieren. Um diese Implementierung umzusetzen, wurde zunächst entschieden, die in Deutschland vorliegende Bevölkerung anhand einer selbstorganisierenden Karte zu simulieren. Die Datenbasis für diese Karte bilden alle deutschen Städte inklusive ihrer Einwohnerzahlen. Um diese Städte inklusive Einwohnerzahlen und ihrer Positionen aus einer zuverlässigen Datenquelle zu erhalten, wurde eine Schnittstelle zum WikiData-API implementiert, die Daten aller deutschen Städte abfragt.

Schnell wurde klar, dass die WikiData-Daten nicht vollständig sind: Vielen deutschen Städten sind zwar der Name und die Einwohnerzahl zugeordnet, es fehlt allerdings die Position als Längen- und Breitengrade. Um diese Daten ebenfalls zu erhalten, wurde anschließend eine Google-Maps-Schnittstelle implementiert, welche die Längen- und Breitengrade für eine gegebene Postleitzahl oder den Namen einer deutschen Stadt liefert.

Ist anschließend anhand dieser Datenbasis eine Karte von Deutschland mit einer bestimmten Anzahl Einwohner (=Neuronen) aufgebaut, lässt sich die Anzahl aller Neuronen als ein neuronales Netz betrachten. Jedes Neuron hat eine bestimmte Position, einen Krankheitszustand (krank/gesund) und eine bestimmte Anzahl an Nachbarneuronen, welche es beeinflusst bzw. von welchen es beeinflusst wird. Dem Testdatengenerator kann nun eine bestimmte Position übergeben werden, an dem eine Krankheitsepidemie gestartet wird. Es wird das Neuron angesteckt, das dieser Position am nächsten ist und dessen Krankheitswert auf krank gesetzt. Anschließend wird eine konfigurierbare Anzahl an Iterationen durchlaufen, die `Infektionswellen` darstellen, d.h. pro Iteration wird eine bestimmte Anzahl an angesteckten Neuronen errechnet. Jede Ansteckungswelle wird mit einem Zeitpunkt zwei Stunden nach der letzten Ansteckungswelle initialisiert.

Um ein krankes Neuron an den Server zu übertragen, musste außerdem eine Schnittstelle zum Server implementiert werden, welche es erlaubt, einen Krankheitstest zu einem bestimmten Zeitpunkt in der Datenbank anzulegen. Diese Schnittstelle existierte auf Serverseite noch nicht, sodass diese zunächst auf der Server-Seite implementiert werden musste, welche anschließend vom Testdatengenerator angesprochen wird. Um abschließend alle Konfigurationen des Testdatengenerators entgegenzunehmen, wurde ein entsprechend erweiterbarer CLI-Parser mit Hilfe der **Apache CLI** Bibliothek implementiert.

**Systembeschreibung** Für die Entwicklung der Applikation wurde ein eigenes Git-Repository (Ablage) namens **medic-prototype-datagenerator** angelegt. Der Testdatengenerator ist in Java geschrieben und ist unter 1.8 kompilierfähig. Als Entwicklungsumgebung kommt Eclipse in der aktuellen Version Luna zum Einsatz. Der Testdatengenerator kann lokal auf jedem Java-fähigen Rechner ausgeführt werden und wird durch Kommandozeilen-Parameter konfiguriert.

**Evaluation** Die Evaluation des Testdatengenerators geschah kontinuierlich durch Continuous-Integration-Ansatz während der Entwicklung. Täglich wurde automatisch eine vorgegebene Menge an Testdaten an den Server gesendet und konnte über die Web-Oberfläche direkt überprüft werden. Etwaige Fehler konnten direkt kommuniziert und verbessert werden. Ein Beispiel-Testdurchlauf ist durch die Abbildungen im Kapitel `Ergänzende Abbildungen` visualisiert.

**Parameter**

- Netzwerklast: Da das Uni-Netzwerk sehr schnell ist, konnte hier noch kein Grenzwert ermittelt werden.
- Größe der gesendeten Testdaten: Durch die Speicherung der Bilder im Base64-Format kommt der Server hier schnell an seine Grenzen. Dies resultiert in längeren Übertragungszeiten.
- Der Aufbau der selbstorganisierenden Karte ist je nach Anzahl der Neuronen eine rechenintensive Aufgabe und kann auf langsameren Rechnern unter Umständen große Zeiträume in Anspruch nehmen.

**Fazit** Der Testdatengenerator wurde unter Einhaltung der Deadlines erfolgreich fertiggestellt. Dennoch lassen sich weitere Optimierungen vornehmen, die im folgenden Kapitel `Ausblick` genannt werden. Diese Optimierungen sind jedoch nicht Teil des formulierten Arbeitspakets und können bei Bedarf, zu einem späteren Zeitpunkt, in weiteren Arbeitspaketen umgesetzt werden.

**Ausblick** Als mögliche Verbesserung lässt sich die Anpassung der selbstorganisierenden Karte nennen. Da der Zustand der Karte abhängig von der Anzahl der Neuronen und der deutschen Städte und deren Einwohnerzahlen ist, strebt die Karte einen festen Endzustand an. Anstatt die Karte also bei jedem Programmstart neu zu errechnen, wäre der logisch bessere Schritt, die Karte einmalig über ein performantes System mit 82.000.000 Neuronen mit hoch gewählter Iterationsanzahl aufspannen zu lassen. Dieser Endzustand der Karte kann anschließend persistiert und im Projekt jedes Mal an den Testdatengenerator übergeben werden.
Eine Einbindung des Testdatengenerators an das Jenkins CI System ist ebenfalls denkbar.

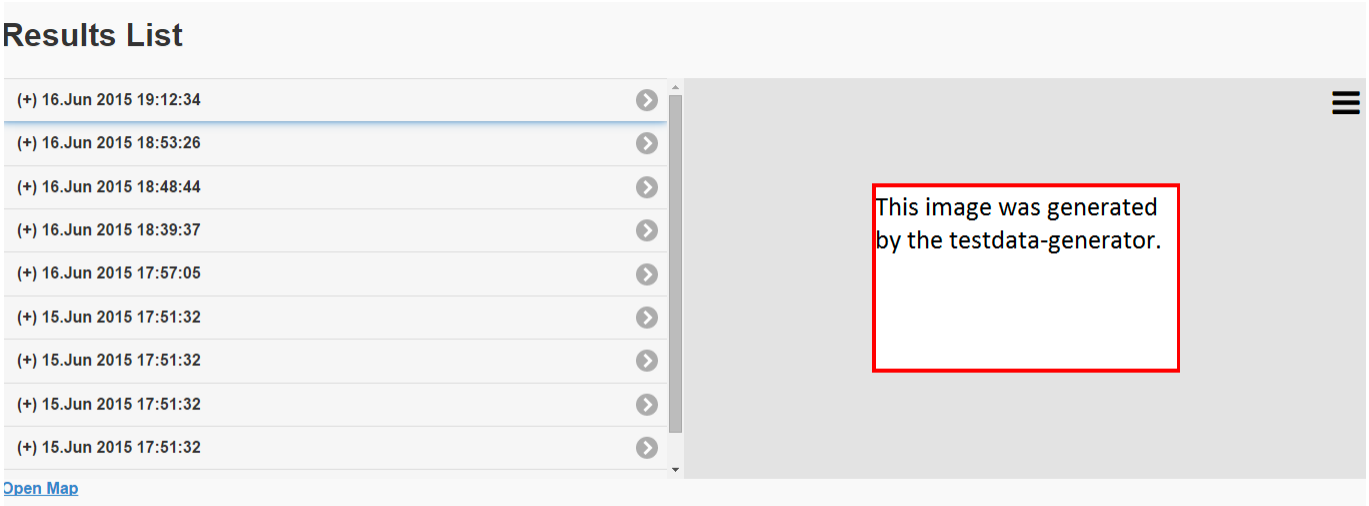



Abbildung 51: Ansicht eines automatisch angelegten Tests im Web-Frontend

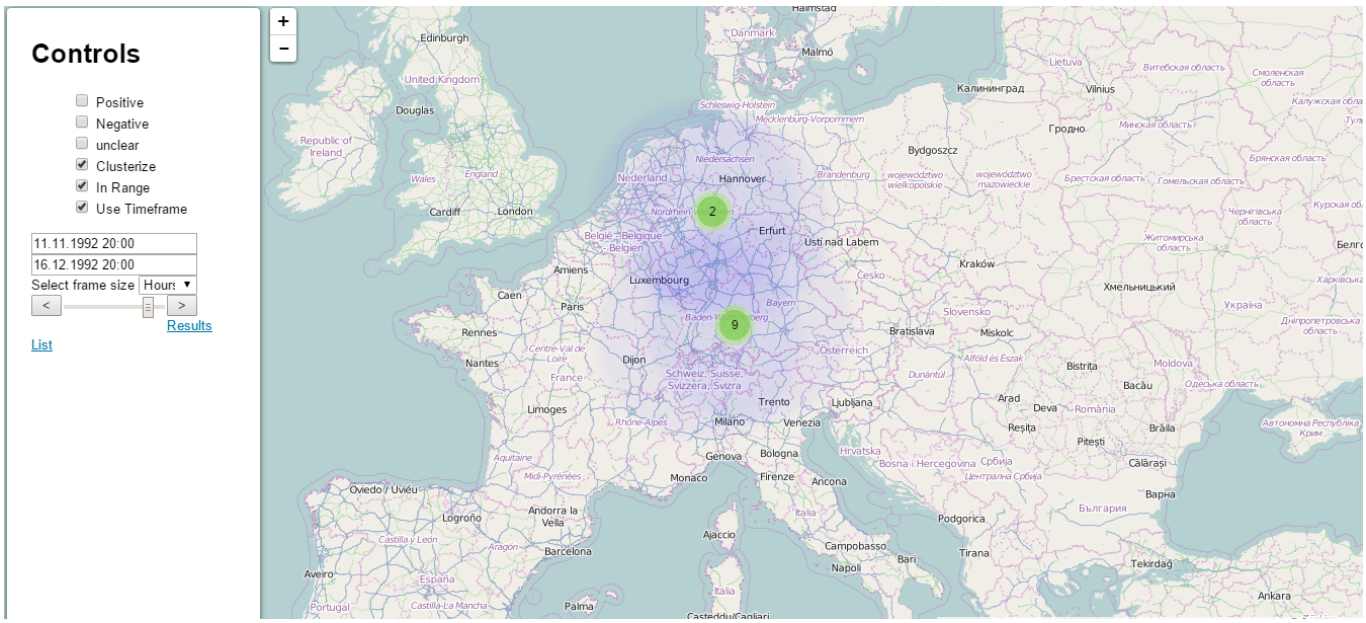



Abbildung 52: Heatmap-Ansicht einer Epidemie 1

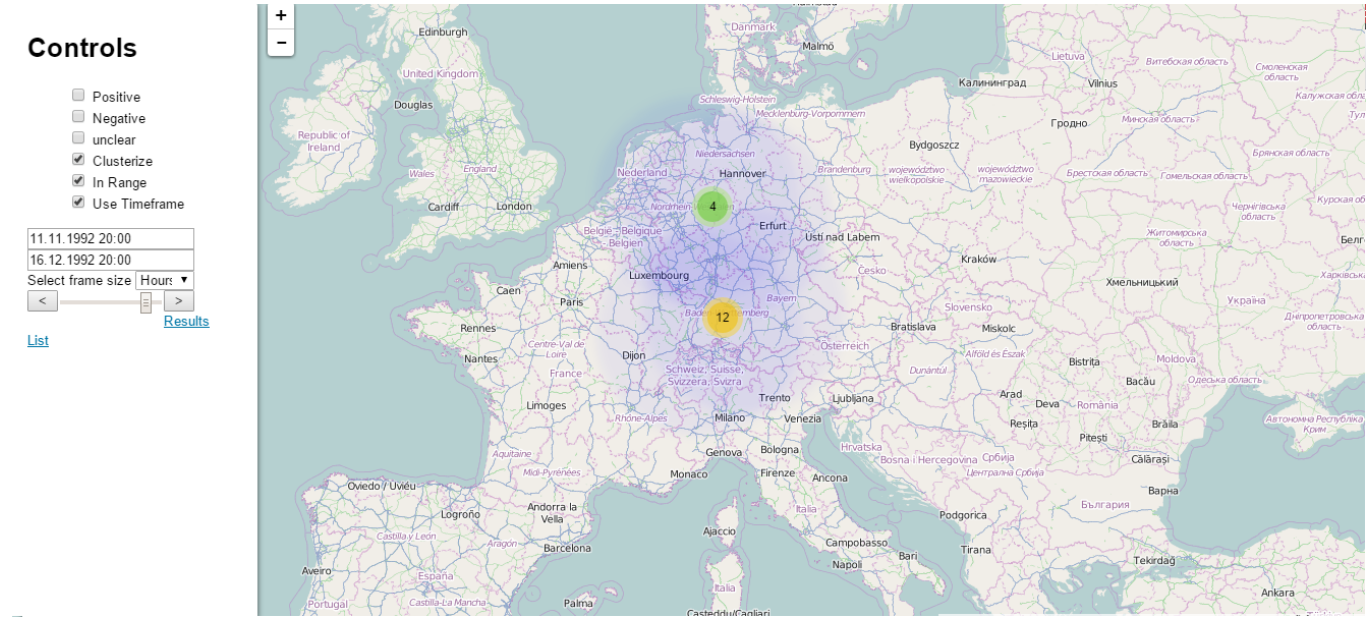



Abbildung 53: Heatmap-Ansicht einer Epidemie 2

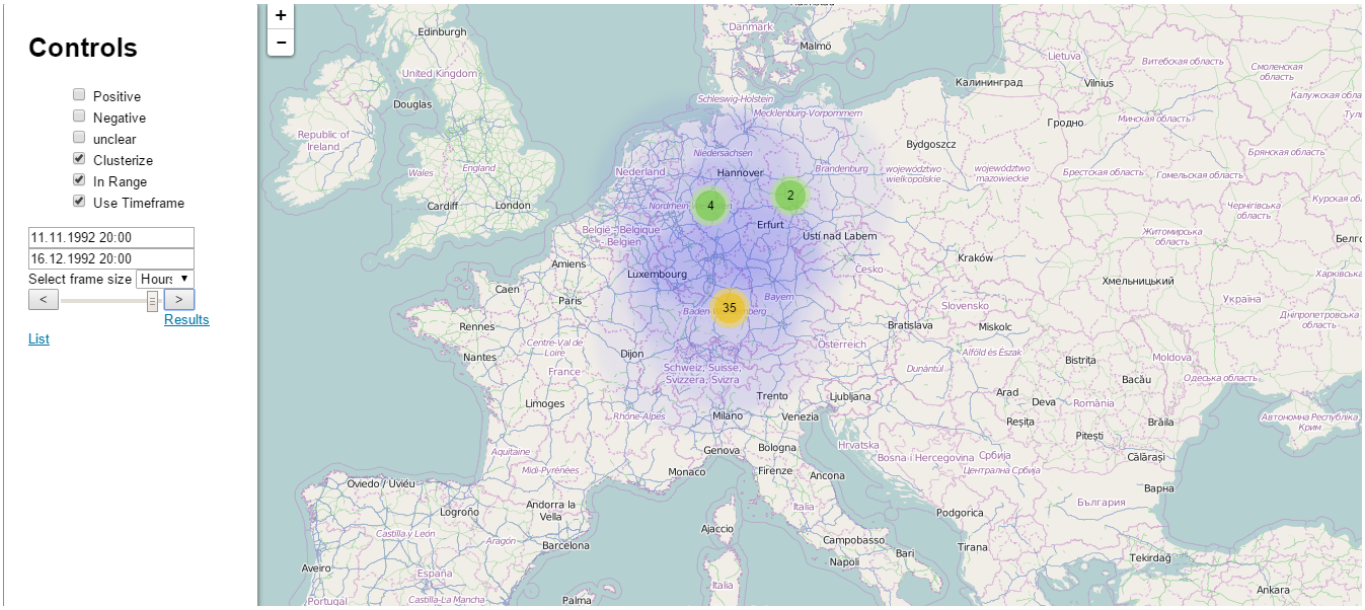



Abbildung 54: Heatmap-Ansicht einer Epidemie 3

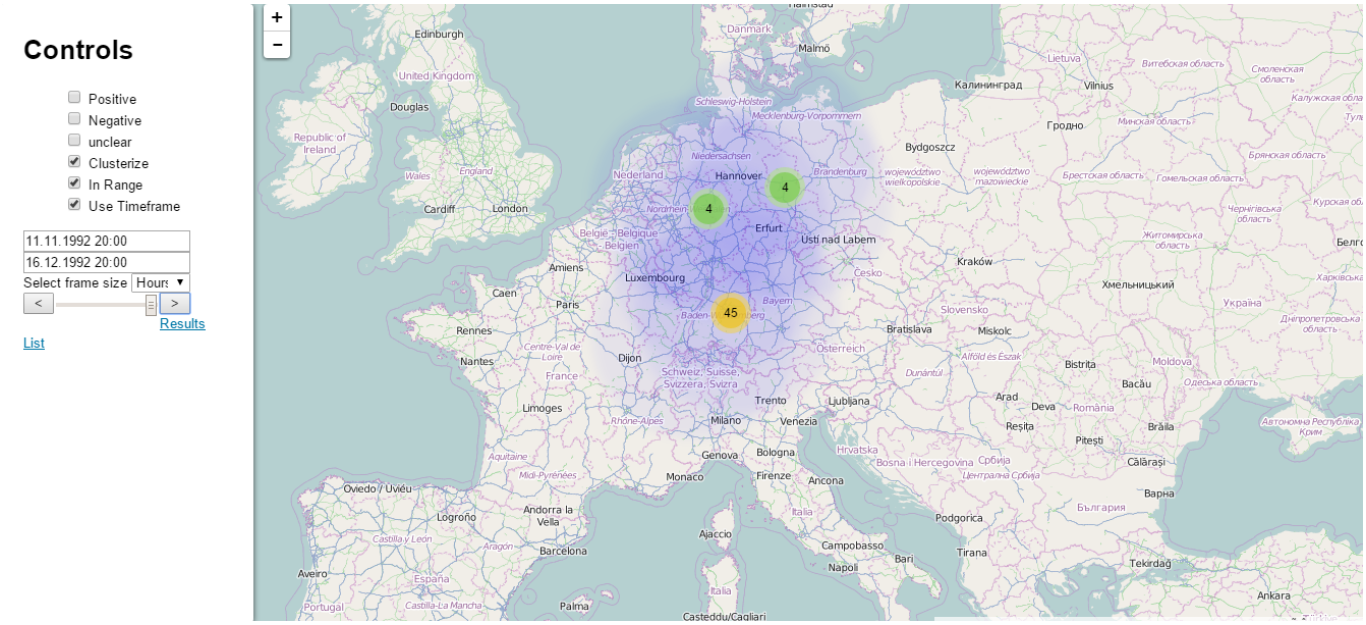



Abbildung 55: Heatmap-Ansicht einer Epidemie 4

**Ergänzende Abbildungen**

### 4.2.11 Entwicklung einer Benutzerverwaltung

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket umfasst die Implementierung Benutzerverwaltung und dient dazu, weitere Anforderungen umzusetzen.
- Arbeitspaketbeschreibung:
  Es soll ein Usermanagement implementiert werden, das folgende Funktionen umfasst:
  - Login für existierende Accountinhaber

  - Registrierung neuer Accounts
  - Zugriff auf Heatmap und Liste von Testdaten nur für eingeloggte Mediziner

  Als Startseite der Weboberfläche muss eine Login-Seite implementiert werden. Hier können sich bereits registrierte Accounts einloggen. Darüber hinaus existiert die Möglichkeit neue Accounts zu erstellen. In einer späteren Iteration soll diese Möglichkeit so beschränkt werden, dass neue Accounts nur durch Administratoren angelegt werden können. Sobald sich ein Accountinhaber eingeloggt hat, wird er auf die Anzeige der Testergebnisse weitergeleitet. Der Zugriff auf diese Seite darf nur nach vorherigem, erfolgreichem Login möglich sein. Die übrigen, bereits implementierten Funktionen der Weboberfläche bleiben unangetastet.
  Auf Seite des Servers müssen entsprechende Methoden implementiert werden, die eine Verifizierung der Login-Daten ermöglichen. Registrierte Accounts werden in der Datenbank gespeichert. Dabei wird das Passwort des Accountinhabers ausschließlich in verschlüsselter Form abgelegt.

  Die Deadline dieses Arbeitspaketes ist der 15. Juli 2015.

- Vorbedingung:
  Die Weboberfläche und der Server liegen im bisher erarbeiteten Zustand vor.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Die überarbeitete Serverstruktur mit den Funktionen der Benutzerverwaltung wird bereitgestellt.
- Aufwand:
  Drei Wochen
- Personen:
  - Daniel Wegmann
  - Raphael Kappes

**4.2.11.1 Dokumentation** Das Usermanagement wurde nach folgenden Anforderungen entwickelt.

**FA-21** Das System muss eine Weboberfläche bereitstellen, die es accountinhabern ermöglicht, sich durch die Eingabe einer E-Mail-Adresse und eines Passworts einzuloggen.

**FA-22** Nach dem Login muss ein Accountinhaber im System als eingeloggt vermerkt werden. Dieser Vermerk kann durch eine Logout-Funktion oder Schließen des Browsers entfernt werden.

**FA-29** Die Administrationsoberfläche ermöglicht die Erstellung eines Benutzeraccounts anhand einer E-Mail-Adresse, einem vollständigen Namen und weitergehenden Kontaktdaten.

**FA-30** Der Server verfügt über ein Rechtesystem, welches jedem Nutzer mindestens eine Benutzergruppe mit entsprechenden Rechten zuweist. Es existieren die Benutzergruppen der Mediziner, der Systemadministratoren und die der Statistiker.

**FA-31** Der Server beherrscht die Speicherung existierender Accounts. Es existiert eine zentrale Datenhaltung aller Accounts und ihrer Daten.

**FA-32** Der Server prüft nach Entgegennahme von Accountdaten diese Daten auf Syntax und Semantik. Duplikate müssen automatisch erkannt werden.

**FA-33** Das zu entwickelnde System muss bei Auftreten eines Fehlers eine passende Rückmeldung an den Nutzer geben.

**FA-34** Jeder Accountinhaber kann bei Erstellung seines Accounts automatisiert per E-Mail über die Accounterstellung informiert werden.

**FA-35** Die Administrationsoberfläche muss die Möglichkeit bieten, Benutzeraccounts zu durchsuchen. Als Ergebnis der Suche wird eine Liste mit allen den Suchparametern entsprechenden Accounts angezeigt.

**FA-36** Die Administrationsoberfläche ermöglicht die Bearbeitung und das Entfernen von bestehenden Benutzeraccounts.

**FA-37** Die Administrationsoberfläche bietet die Möglichkeit, neue Teststreifenvarianten anzulegen.

**FA-40** Die Administrationsoberfläche ermöglicht die Bearbeitung von bestehenden Teststreifenvarianten.

**FA-41** Die Administrationsoberfläche bietet die Möglichkeit, bestehende Teststreifenvarianten zu durchsuchen. Ergebnis der Suche ist eine Liste an Teststreifenvarianten, welche den Suchparametern entsprechen.

**Ablauf** Für die Implementierung des Usermanagements wurde zunächst auf dem Server das Modell **Account** angelegt und eine entsprechende Tabelle in der Datenbank erzeugt. Das Modell beinhaltet die Account-ID, den Account-Namen und den Account-Typ. Der Account-Typ wird unterschieden in **Administrator** (Admin mit dem ganzzahligen Wert 0) und **Mediziner** (Doktor mit dem ganzzahligen Wert 1). Die Tabelle in der Datenbank beinhaltet die Spalten account_id, account_name, account_password und account_type wie in Abb. 56 zu sehen ist.

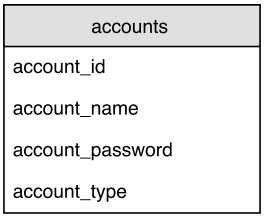



Abbildung 56: Datenbank-Entität eines Accounts

Das Passwort wird in der Datenbank nicht als Klartext gespeichert, sondern stattdessen der Hash-Wert, mit dem das eigentliche Passwort wiedergefunden werden kann. Bei der weiterführenden Implementierung wurden zwei Ansätze betrachtet: Einerseits die vorhandene Weboberfläche über Spring zu verwalten und die vorhanden HTML-Dateien mit „Thymeleaf“[13] zu integrieren oder an die vorhandene Weboberfläche anzuknüpfen und auf getrennten Servern verwalten (die Weboberfläche mit einem Apache-Server und der Server mit einem Tomcat-Server). Für das weitere Vorgehen wurde die erste Möglichkeit gewählt, da zum einen die Integration von HTML-Dateien, JavaScript und CSS in Spring mit „Thymeleaf“ gut dokumentiert ist und zum anderen, da es keine Vorteile bringt, die Verwaltung von Weboberfläche und Server auf getrennt laufen zu lassen, ohne diese physikalisch zu trennen.

[13] http://www.thymeleaf.org/

**Systembeschreibung** Entwickelt wurde das Usermanagement auf einem Windows-Rechner mit Firefox. Dazu wurden die Spring Bibliothek, sowie Maven eingesetzt. Weiterhin wurde die Thymeleaf Bibliothek benutzt, um HTML-Dateien als Vorlagen in Spring zu laden. Zusätzlich wurde die Kendo-JavaScript Bibliothek für die Benutzerverwaltung verwendet.

**Evaluation** Für die Evaluation wurden manuelle Tests der Weboberfläche durchgeführt. Der Ablauf dieser Tests beginnt mit dem Einloggen auf der Seite. Wenn der Login erfolgreich war, lässt sich der Ablauf in zwei Möglichkeiten aufteilen: Wenn der eingeloggte Account die Rolle Administrator hat, so wird er in in den administrativen Bereich, wie in Abb. 57 auf Seite 95 zu sehen, weitergeleitet. Ist die Rolle des Accounts Mediziner, erfolgt eine Weiterleitung in den medizinischen Bereich, der in Abb. 58 auf Seite 96 und Abb. 59 auf Seite 97 zu sehen ist. Wenn das Einloggen fehlschlägt wird „loginFailed“ aufgerufen und es können erneut der Accountname und das Passwort eingegeben werden. In den Tests wurde diese Aufteilung sowie die korrekte Funktion der jeweiligen Seiten überprüft.

Für den administrativen Bereich werden entsprechende Rechte (Rolle **Admin**) benötigt. Sind diese Rechte für den eingeloggten Account vorhanden, wird die „admin.html“ aufgerufen. In dieser Ansicht kann der Administrator neue Accounts (Records) anlegen, vorhandene Accounts bearbeiten und Accounts löschen. Die Rolle des Accounts wird dabei im Cookie gespeichert, so dass der Accountinhaber nur die für sich erlaubten Seiten einsehen kann. Accounts mit der Rolle „Admin“ können nicht auf die „index.html“ und „map.html“ zugreifen. Im Umkehrschluss können Accounts mit der Rolle „Doktor“ nicht auf „admin.html“ zugreifen.

Im medizinischen Bereich kann der Accountinhaber eine Liste mit vorhandenen Testergebnissen sehen und diese bearbeiten. Weiterhin kann er auf die Karte wechseln und hat dort eine geografische Ansicht mit allen abgegebenen Testergebnissen. Wenn der Accountinhaber sich ausloggt, wird der gespeicherte Cookie gelöscht und die Login-Seite wird aufgerufen. Wenn der Accountinhaber den Browser schließt und wieder öffnet kann er seinen Cookie erneuern in dem er sich erneut einloggt oder er direkt auf die vorherige Seite geht. Der alte Zustand wird dabei nicht gespeichert.

Das Usermanagement wurde auf verschiedenen Versionen von Firefox, Chrome, Opera und Internet Explorer getestet. Laut Telerik Kendo UI[14] sollen folgende Browser unterstützt werden:

- Internet Explorer
- Chrome
- Firefox
- Opera
- Safari

Das Usermanagement wurde auf Firefox in den Versionen 33 und 39 (und höher) erfolgreich getestet. Im Internet Explorer 11 konnte das Usermanagement nicht gestartet werden. Die Gründe hierfür müssen noch genauer untersucht werden. Die Tests mit Opera auf Version 15 und 30 und Chrome auf Version 43 wurden ohne Beanstandungen abgeschlossen. Alle anderen Inhalte der Website konnten ohne Probleme in allen Browsern dargestellt werden.

**Parameter** Das Usermanagement funktioniert nur dann korrekt, wenn der Browser JavaScript akzeptiert und dieses nicht veraltet ist. Weiterhin kann die korrekte Funktion nur bei Benutzung der Browser in den entsprechenden Versionen, die in der Evaluation genannt wurden, garantiert werden.

[14] http://docs.telerik.com/kendo-ui/browsers-support

**Fazit** Es wurde ein Usermanagement implementiert, das einen administrativen Bereich zur Accountverwaltung bietet, sowie einen medizinischen Bereich, in dem Testergebnisse von Probanden angezeigt und bearbeitet werden können. Die Deadline wurde nicht eingehalten, jedoch konnte das Usermanagment in der letzten Woche des Sprints nachträglich fertiggestellt werden. Die geographische Ansicht, in der die Testergebnisse von Probanden und dem Prototypen des Testdatengenerators anzeigt werden, blieb unangetastet und funktioniert weiterhin. Zusätzlich können diese Seiten nur noch von eingeloggten Personen besucht und die Sitzungen teilweise gespeichert werden. Daraus ergibt sich, dass folgende Anforderungen erfüllt wurden:

- FA-31: Der Server beherrscht die Speicherung existierender Accounts. Es existiert eine zentrale Datenhaltung aller Accounts und ihrer Daten.
- FA-32: Der Server prüft nach Entgegennahme von Accountdaten diese Daten auf Syntax und Semantik. Duplikate müssen automatisch erkannt werden.
- FA-33: Das zu entwickelnde System muss bei Auftreten eines Fehlers eine passende Rückmeldung an den Nutzer geben.
- FA-36: Die Administrationsoberfläche ermöglicht die Bearbeitung und das Entfernen von bestehenden Benutzeraccounts.

Weiterhin sind folgende Anforderungen nur teilweise erfüllt worden:

- FA-21: Das System muss eine Weboberfläche bereitstellen, die es Accountinhabern ermöglicht, sich durch die Eingabe einer E-Mail-Adresse und eines Passworts einzuloggen.
  - Der Accountname kann die Email-Adresse enthalten, jedoch wird dieser nicht auf die Syntax und die Echtheit dieser Email-Adresse hin überprüft.
- FA-22: Nach dem Login muss ein Accountinhaber im System als eingeloggt vermerkt werden. Dieser Vermerk kann durch eine Logout-Funktion oder Schließen des Browsers entfernt werden.
  - Der Accountinhaber kann sich ein und ausloggen und das wird vom System registiert, jedoch wird beim Schließen des Browser dieses nicht vom System erkannt.
- FA-29: Die Administrationsoberfläche ermöglicht die Erstellung eines Benutzeraccounts anhand einer E-Mail-Adresse, einem vollständigen Namen und weitergehenden Kontaktdaten.
  - Es kann ein neuer Account erstellt werden, jedoch nur ein Name.
- FA-30: Der Server verfügt über ein Rechtesystem, welches jedem Nutzer mindestens eine Benutzergruppe mit entsprechenden Rechten zuweist. Es existieren die Benutzergruppen der Mediziner, der Systemadministratoren und die der Statistiker.
  - Jeder Account wird kann entweder Administrator oder Mediziner sein. Statistiker gibt es noch nicht, da diese noch keine funktionalität haben.
- FA-35: Die Administrationsoberfläche muss die Möglichkeit bieten, Benutzeraccounts zu durchsuchen. Als Ergebnis der Suche wird eine Liste mit allen den Suchparametern entsprechenden Accounts angezeigt.
  - Auf der Administrationsoberfläche können alle Accounts betrachtet werden, jedoch können diese nicht über Suchparameter ausgewählt werden.

Schließlich wurden folgende Anforderungen nicht erfüllt:

- FA-34: Jeder Accountinhaber kann bei Erstellung seines Accounts automatisiert per E-Mail über die Accounterstellung informiert werden.
- FA-37: Die Administrationsoberfläche bietet die Möglichkeit, neue Teststreifenvarianten anzulegen.
- FA-40: Die Administrationsoberfläche ermöglicht die Bearbeitung von bestehenden Teststreifenvarianten.
- FA-41: Die Administrationsoberfläche bietet die Möglichkeit, bestehende Teststreifenvarianten zu durchsuchen. Ergebnis der Suche ist eine Liste an Teststreifenvarianten, welche den Suchparametern entsprechen.

Auf Grund von Fehlplanungen innerhalb des bearbeitenden Teams konnte die Deadline des Arbeitspaketes nicht eingehalten werden. Die noch fehlenden Abschnitte wurden anschließend nachträglich implementiert und eingepflegt.

**Ausblick** Im späteren Verlauf sollten die Sitzungen vollständig gespeichert werden, so dass Bearbeitungsstände von Administratoren und Mediziner zwischen Sitzungen nicht verloren gehen. Zusätzlich sind für die Accountverwaltung Funktionen denkbar, wie das Einlesen einer externen Liste von Accounts - z.B. von der Bundesärztekammer autorisierte Ärzte - und die automatische Erzeugung von Accounts. Weiterhin bleibt die Möglichkeit offen zusätzliche Rollen hinzuzufügen, wie etwa Spezialisten, Allgemeinmediziner und Statistiker.

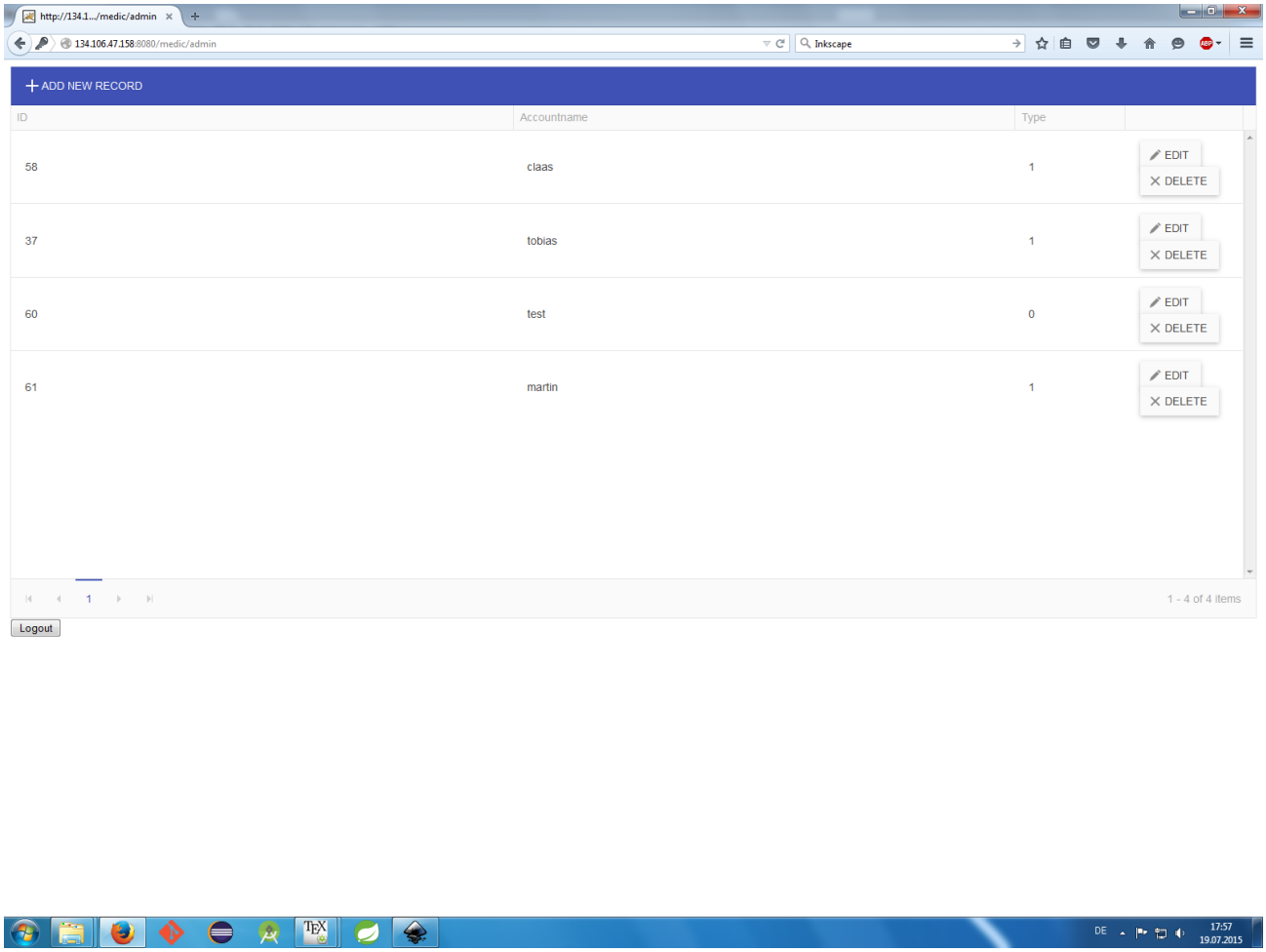



Abbildung 57: Benutzeroberfläche der Accountverwaltung

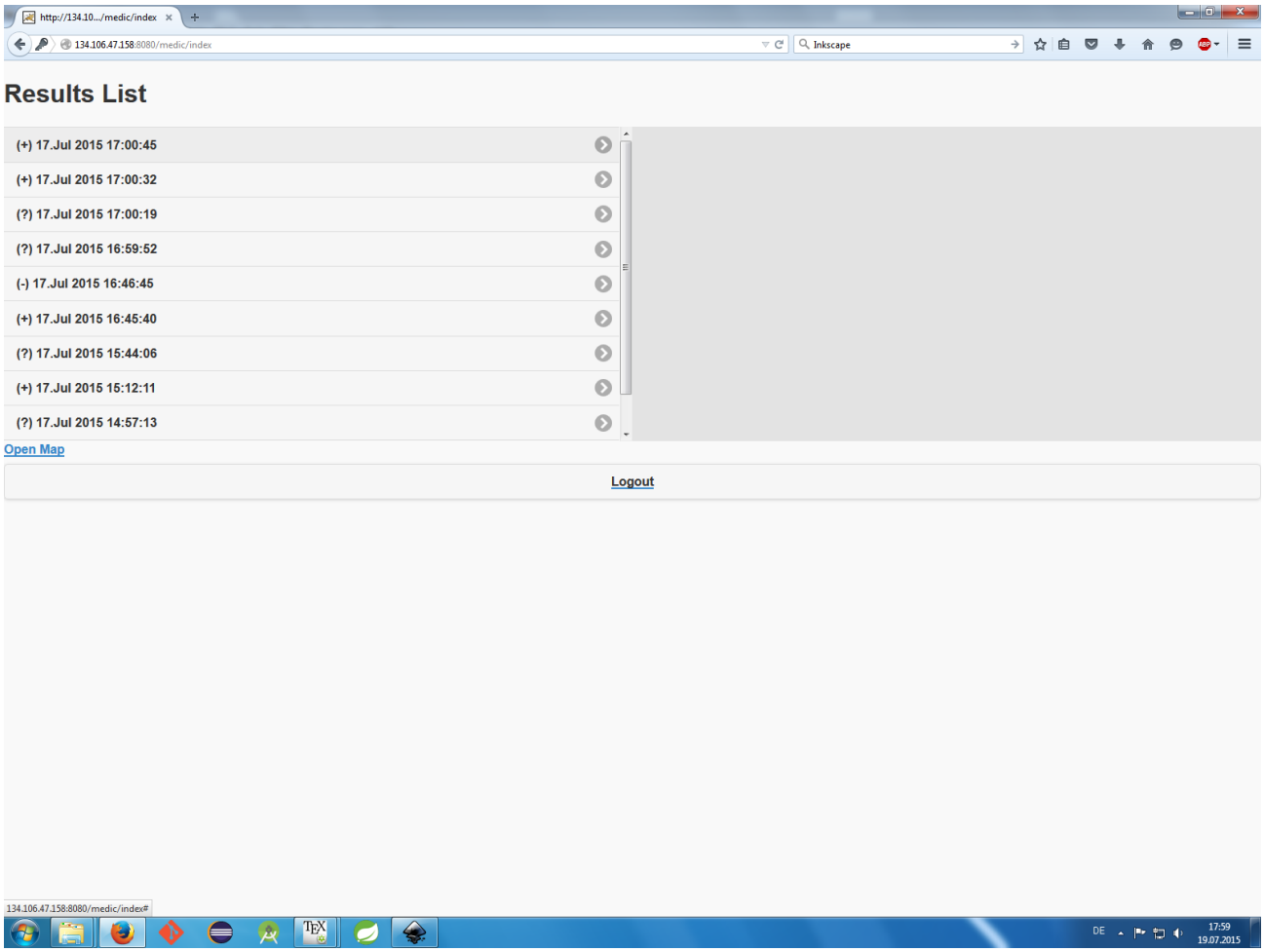



Abbildung 58: Darstellung von Testergebnissen, vgl. Arbeitspaket „Schnittestelle Auswertungsoberfläche - Datenbank“

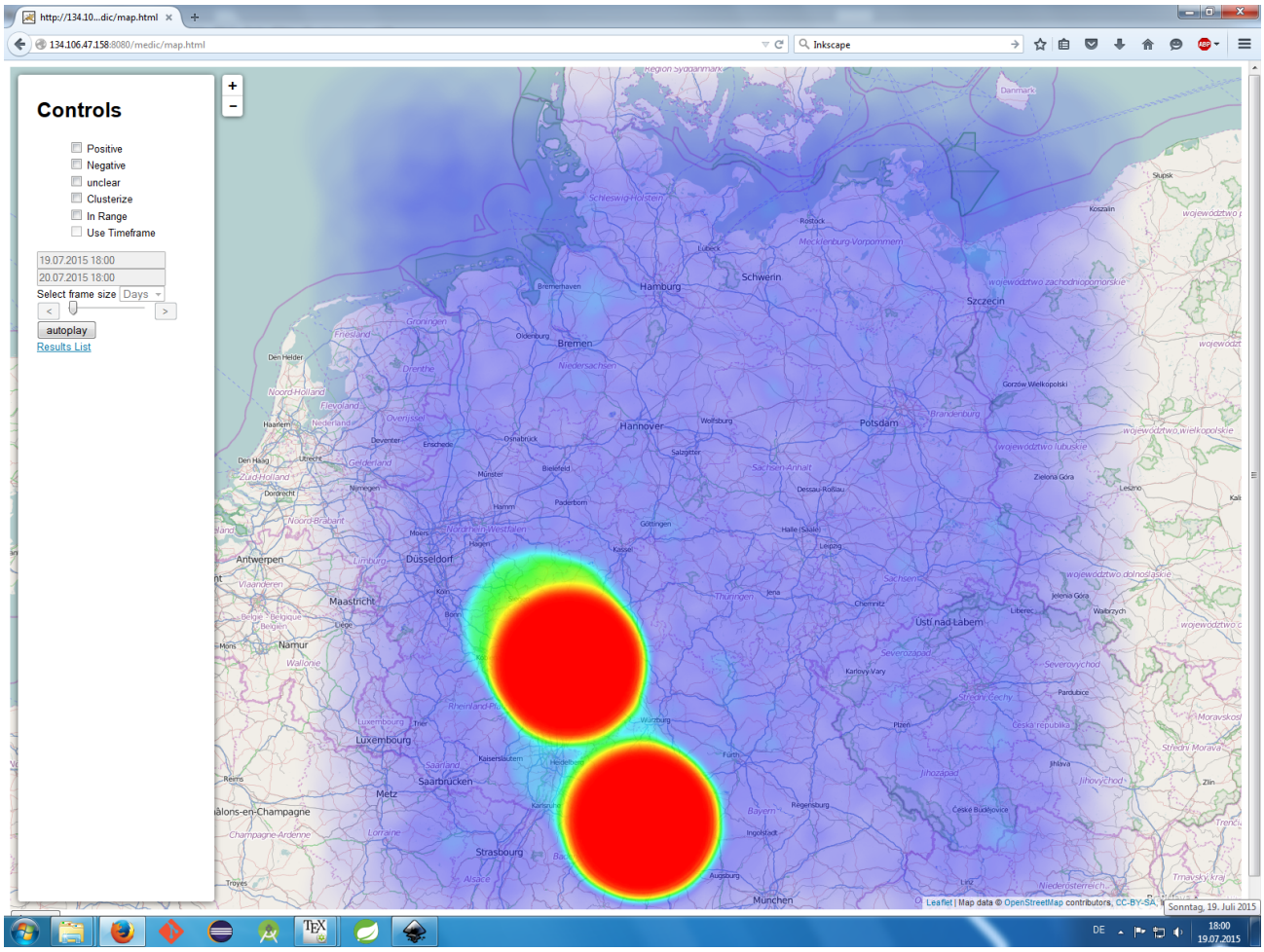



Abbildung 59: Darstellung der Heatmap, vgl. Arbeitspaket „Erweiterung der Heatmap“

**Ergänzende Abbildungen**

### 4.2.12 Fazit des Sprints

Abgesehen von der Integration des OpenCV Frameworks in die mobilen Applikationen konnten alle geplanten Arbeiten vollständig abgeschlossen werden. Damit wurden zum einen neue Features in die bestehenden Prototypen eingebaut und zum anderen Grundlagen gelegt um zukünftige Arbeiten zu erleichtern. Durch das Usermanagement ist die Weboberfläche vor unberechtigten Zugriffen geschützt und gleichzeitig die Verwaltung von Accounts realisiert. Die überarbeitete Heatmap und Funktion zum Senden von Diagnosen erweitern die Weboberfläche um zusätzliche Funktionen.
Durch die Überführung des Prototypen der mobilen Applikation nach iOS wurde das neben Android wichtigste mobile Betriebssystem erschlossen. Dadurch wurde das Ziel erreicht, auf beiden Betriebssystemen eine gleichwertige Applikation zu erstellen. Darüber hinaus wurden die Applikationen um das dynamische Detektieren des QR-Codes erweitert. Indem der Nutzer nicht länger manuell ein Foto aufnehmen muss, konnte die Durchführung des Testablaufs vereinfacht werden.
Durch den Testdatengenerator wurde ein Tool geschaffen, dass es ermöglicht zu Testzwecken schnell große Mengen von Testdaten zu generieren, um damit z.B. die Heatmap und andere Funktionen der Weboberfläche zu testen. In Kombination mit der erweiterten Heatmap hat so ein Mediziner die Möglichkeit, sich den zeitlichen Verlauf einer sich ausbreitenden Krankheit visualisieren zu lassen und eventuelle Rückschlüsse daraus ziehen zu können.
Im Verlauf des Sprints traten diverse Organisationsschwierigkeiten auf. Die Deadlines einiger Arbeitspakete konnten nicht gehalten werden, was durch Mehrarbeit ausgeglichen werden musste. Auch die

Sprintplanung sollte in zukünftigen Sprints effizienter gestaltet werden, um so den den Organisationsaufwand zu reduzieren.
Dennoch kann der Sprint abschließend als erfolgreich bewertet werden, da der Großteil der geplanten Arbeiten abgeschlossen wurde. Die noch nicht umgesetzten Funktionen (Integration von OpenCV) werden im nächsten Sprint weiter bearbeitet.

## 4.3 Sprint 4

Die Ziele des vierten Sprints sind es, die Weboberfläche mit Hilfe anderer Technologien neu zu entwickeln und den Algorithmus für die Bilderkennung zu verbessern. Des Weiteren soll ein Teststreifengenerator, mit dem sich Teststreifen in unterschiedlichen Varianten erstellen lassen, entwickelt werden. Bevor die Weboberfläche neu entwickelt werden kann, muss eine Evaluation durchgeführt werden, aus der sich ergibt, mit welchem Framework die neue Weboberfläche erstellt werden soll. Dabei dient die Seminararbeit von Danny Fonk als Grundlage, so dass als mögliche Frameworks nur noch Symfony und PrimeFaces evaluiert werden müssen.
Die Neuentwicklung der Weboberfläche hat den Hintergrund, dass die alte Version nicht sehr performant lief, ein schlechtes Design hatte und schlecht wartbar gewesen ist. Durch die Neuentwicklung sollen diese Probleme behoben werden. Alle Funktionen, die in der alten Weboberfläche vorhanden sind, sollen am Ende des Sprints auch in der neuen Weboberfläche vorhanden sein. Im Zuge der Neuentwicklung sollen der Weboberfläche eine Filter- und Sortierfunktion und eine Methode zum Wiederherstellen des Passworts hinzugefügt werden. Das Usermanagement wird überarbeitet und um einige Funktionen erweitert.
Für die Verbesserung der Bilderkennung werden zunächst unterschiedliche Tools zur Bildverarbeitung evaluiert. Im Anschluss sollen ein Weißabgleich und eine Intensitätsdetektierung durchgeführt werden. Die Ergebnisse daraus sollen genutzt werden, um den Farbumschlag innerhalb eines definierten Bereichs lokalisieren zu können.
Die Details der jeweiligen Arbeitspakete sind in den entsprechenden Unterkapiteln ab Kapitel 4.3.2 auf Seite 101 zu finden.

### 4.3.1 Sprintplanung

Die Dauer des Sprints wurde auf sechs Wochen (29.07. bis 08.09.) angesetzt, wobei die Arbeitslast auf verschiedene Arbeitspakete aufgeteilt wurde. Die Planung wurde dabei zunächst von einer kleinen Gruppe durchgeführt. Dabei wurden die Arbeitspakete vordefiniert und eine erste Aufwandsabschätzung vorgenommen. Die Ergebnisse dieser Vorplanung wurden anschließend mit der gesamten Gruppe diskutiert. In dieser Diskussion wurde die Aufwandsabschätzung der einzelnen Pakete angepasst. Jedes Arbeitspaket ist auf eine feste Zeit angelegt und umfasst die vollständige Implementierung der gestellten Aufgabe sowie die dazugehörige Dokumentation.
Die Sprintplanung unterschied sich zu den Vorherigen, weil nur die Personen, die in einer Gruppe arbeiten, sowie deren Aufgabe für die erste Woche festgelegt wurden. Diese nahmen sich dann im Laufe des Sprints Arbeitspakete. Von diesen gab es ausreichend viele, dass der ganze Sprint damit abgedeckt wurde. Es wurde unterschieden zwischen kritischen, normalen und optionalen Arbeitspaketen. Sobald eine Gruppe mit einer Aufgabe fertig war, konnte sie sich eine Neue nehmen, wobei die kritischen Arbeitspakete vorzuziehen waren. Dies stellt eine flexiblere Gestaltung dar und es werden unnötige Zeiten, in denen eine Gruppe keine Aufgabe hat, vermieden.
Am Ende des Sprints wird der erreichte Fortschritt dem Kunden in einer Präsentation vorgestellt.

- Fokus des Sprints
- Arbeitspakete und Planung
- Planung der einzelnen Ressourcen

#### 4.3.1.1 Fokus des Sprints

Der Fokus des Arbeitspakets liegt auf der Bildverarbeitung und dem Webfrontend. Das Frontend wird komplett neu aufgesetzt, während für die Bildverarbeitung viele Verbesserungen geplant sind. Die übrigen Bereiche des Projektes werden in diesem Sprint nicht angepasst.

#### 4.3.1.2 Arbeitspakete und Planung

Aus der Planung und dem Fokus ergeben sich folgende Arbeitspakete:

- Struktur der Weboberfläche
- Login in die Weboberfläche
- Usermanagment
- Testübersicht
- Filtern und Sortieren der Liste mit den Tests
- Rückmeldung an den Patienten
- Accountverwaltung vom Benutzer
- Karte
- Evaluation von Bildverarbeitungstools
- Teststreifengenerator
- Weißabgleich
- Intensitätsdetektierung / Farbsättigung
- Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs

#### 4.3.1.3 Planung der Ressourcen

Im folgenden ist die Personalplanung dargestellt. Sie gibt Auskunft darüber, welches Teammitglied an welchem Arbeitspaket gearbeitet hat.

- Raphael Kappes: Struktur der Weboberfläche (1 Woche), Login in die Weboberfläche (3 Wochen), Usermanagment (3 Wochen), Karte (3 Wochen)
- Christian Sandmann: Struktur der Weboberfläche (1 Woche), Testübersicht (2 Wochen),Filtern und Sortieren der Liste mit den Tests (3 Wochen), Accountverwaltung vom Benutzer (2 Wochen)
- Timo Schlömer: Rückmeldung an den Patienten (2 Wochen), Passwort wiederherstellen (2 Wochen), Teststreifengenerator (2 Wochen), Weißabgleich (2 Wochen)
- Timo Raß: Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs (4 Wochen)
- Nicolas Koch: Teststreifengenerator (2 Wochen)
- Kevin Sandermann: Struktur der Weboberfläche (1 Woche), Login in die Weboberfläche (3 Wochen), Usermanagment (3 Wochen), Karte (3 Wochen)
- Sebastian Horwege: Evaluation von Bildverarbeitungstools (1 Woche), Weißabgleich (2 Wochen), Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs (4 Wochen)
- Danny Fonk: Rückmeldung an den Patienten (2 Wochen), Passwort wiederherstellen (2 Wochen), Teststreifengenerator (2 Wochen)

- Jan Philipp Stubbe: Struktur der Weboberfläche (1 Woche), Testübersicht (2 Wochen),Filtern und Sortieren der Liste mit den Tests (3 Wochen), Accountverwaltung vom Benutzer (2 Wochen)
- Christoph Ressel: Evaluation von Bildverarbeitungstools (1 Woche), Weißabgleich (2 Wochen), Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs (4 Wochen)
- Daniel Wegmann: Urlaub

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und dessen Dokumentation skizziert.

### 4.3.2 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.3.3 Neue Struktur der Weboberfläche

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Durch die Umsetzung dieses Arbeitspaketes soll weitere Anforderungen abgearbeitet werden.
- Arbeitspaketbeschreibung:
  Auf dem Server läuft eine Anwendung, die Anfragen unter einer statischen, öffentlichen IP-Adresse entgegennimmt, verarbeitet und eine rudimentäre Weboberfläche darstellt. Diese besteht aus einer simplem Willkommensnachricht und einer Navigationsleiste mit den Einträgen „Testübersicht“, „Karte“ und „Administration“. Diese Anwendung ist so entworfen, dass Entwickler eine lokale Instanz für Entwicklungszwecke starten können, welche unabhängig von der Anwendung auf dem Server ist. Das Continuous Integration System der Projektgruppe ist in der Lage, mehrere Instanzen der Anwendung zu bauen und auf dem Server bereitzustellen.
- Vorbedingungen:
  Das Seminar von Danny Fonk zur Strukturierung der Weboberfläche ist abgeschlossen. Es wurde abgestimmt welche Technologie genutzt wird („PrimeFaces“oder „Symfony 2“).
- Nebenbedingungen:
  -
- Nachbedingungen:
  Wenn die Adresse in einem Browser aufgerufen wird, wird die Startseite mit der Navigationsleiste und der Willkommensnachricht angezeigt. Ein nicht an dem Arbeitspaket beteiligtes Projektgruppenmitglied ist in der Lage, innerhalb von 30 Minuten eine lokale Instanz der Anwendung zu erstellen und zu starten.
- Aufwand:
  Eine Woche
- Personen:

- PrimeFaces:
  - Kevin Sandermann
  - Raphael Kappes
- Symfony:
  - Jan Philipp Stubbe
  - Christian Sandmann

#### 4.3.3.1 Dokumentation Symfony

In diesem Arbeitspaket wurde die Weboberfläche rudimentär in Symfony umgesetzt. Dies diente dazu, eine Grundlage für die Entscheidung zwischen Symfony und PrimeFaces zu bilden.

**Ablauf** Zu Beginn wurde mit der Gruppe für PrimeFaces abgestimmt, welche Parameter zusätzlich untersucht werden sollten. Dabei wurde sich auf folgende Punkte geeinigt:

- Debugging
- Änderung zu Laufzeit
- Community
- eigenes CSS einbauen
- objektrelationales Mapping

Um mit der Implementierung beginnen zu können, wurden **PHP**, **Symfony** und **Composer** installiert. Mit diesen Komponenten konnte ein erstes Symfonyprojekt angelegt werden. Um die Administration umzusetzen, wurde auf die Bibliothek **FOSUserBundle** zurückgegriffen. Dieses wurde versucht über das Paktetverwaltungs-System **Composer** zu installieren, allerdings konnte die Bibliothek nicht in einer Woche eingebunden werden.
Im nächsten Schritt sollte das Framework namens **Bootstrap** eingebunden werden. Dies sollte ebenfalls mit Composer installiert werden, schlug jedoch fehl.
Der Aufruf zum Server, der für die Testdaten benötigt wird, konnte mit PHP realisiert werden. Damit konnte eine Seite für die Testübersicht erstellt werden. Diese zeigt allerdings nur die letzten zehn Tests an, um die Menge der zu übertragenden Daten zu reduzieren. Zusätzlich wurde JQuery eingebunden, um die Bilder zu dem jeweiligen Test dynamisch nachzuladen.
Um die Karte anzuzeigen, wurde **Google Maps** eingebunden. Diese kann auch auf einer eigenen Seite angezeigt werden. Auf der Karte werden positive, negative und unentscheidbare Tests, die von dem Server stammen, als Marker in verschiedenen Farben dargestellt. Dies ist in Abbildung 103 dargestellt.
Zusätzlich wurde eine Navigation implementiert, die auf allen Seiten dargestellt wird.
Es wurde auch untersucht, wie Fehlerbehebung in PHP durchgeführt werden kann. Dazu musste **XDebug** installiert und konfiguriert werden.

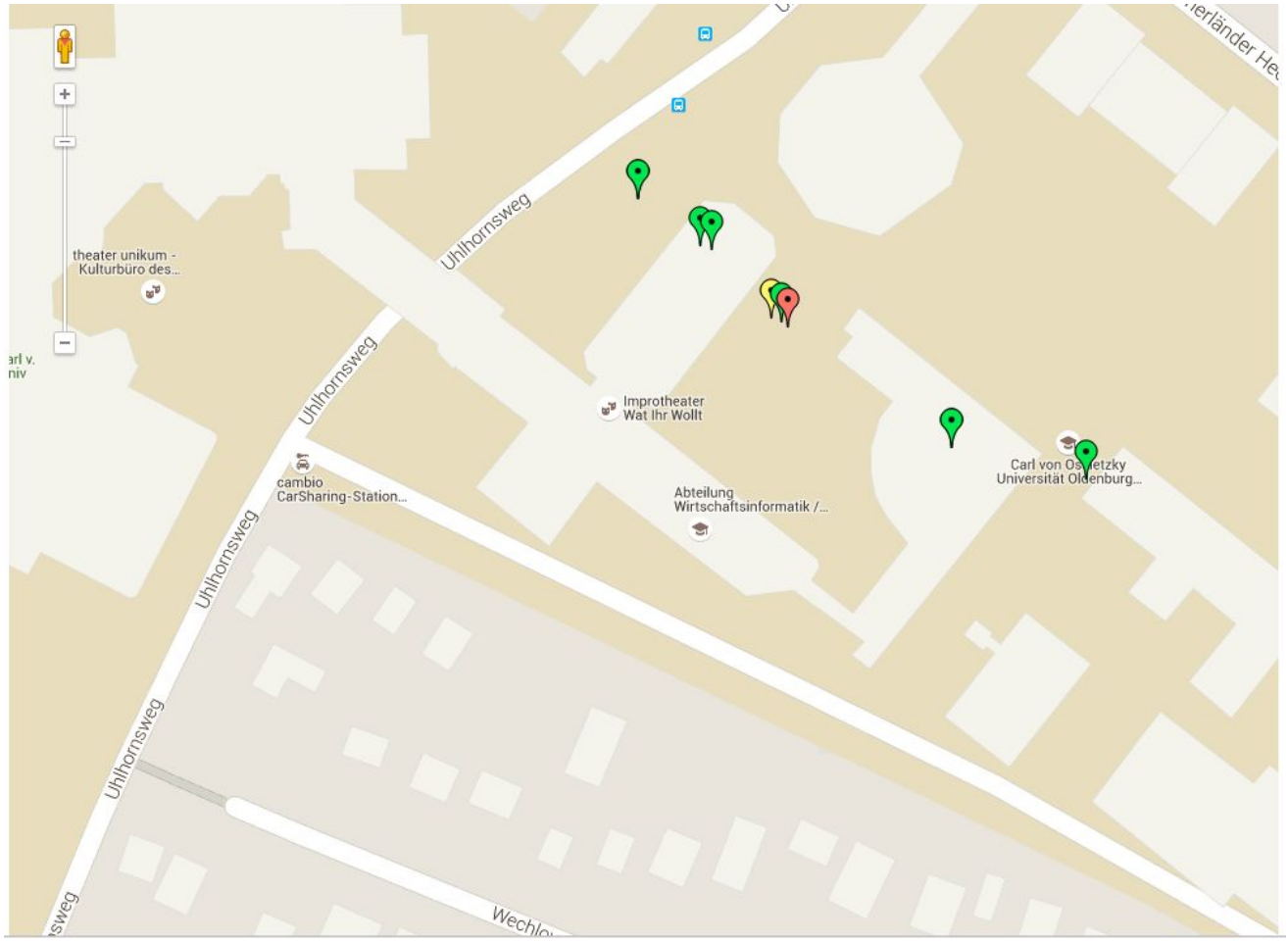



Abbildung 60: Screenshot von der Karte

**Technologien**

**Symfony2** ist ein PHP Framework für die Webentwicklung. Es wurde genutzt um das grundlegende Projekt zu erstellen.

**JQuery** ist eine JavaScript-Bibliothek. JQuery wurde eingesetzt, um asynchrone Aufrufe zum Server durchzuführen und damit die Daten dynamisch neu zu laden.

**Twig** ist ein Framework, um die Darstellung der Weboberfläche zu entwickeln.

**PHP** ist eine Skriptsprache, die hauptsächlich für die Erstellung von Webanwendungen verwendet wird. PHP wurde in Verbindung mit Symfony genutzt, um die Webseite zu erstellen.

**XDebug** ist eine Erweiterung für PHP, mit der sich der Code debuggen lässt.

**Systembeschreibung** Entwickelt wurde die Webseite auf einem Windows-Rechner mit Google Chrome. Zusätzlich wurde PHP 5.6 auf dem System installiert. Außerdem wurde Composer genutzt, um externe Bibliotheken nachzuladen. Als Editor wurde Sublime Text in der Version 2 verwendet.

**Evaluation** Es erfolgte keine Evaluation, die auf objektiven Tests beruht, weil das Ziel des Arbeitspaketes war, zu ermitteln, mit welchem Aufwand die Grundstruktur der Webseite nachgebaut werden kann. Die Evaluation fand in Hinblick auf die im Ablauf genannten Parametern statt.

- Das Debugging wurde mit XDebug durchgeführt. Damit ließ sich der PHP-Code gut debuggen, der Code in den Twig Dateien konnte damit nicht untersucht werden.
- Änderungen am Code wurden nach Aktualisieren der Seite direkt übernommen.
- Wenn Probleme auftraten, gab es im Internet viele Lösungsansätze. Diese waren von unterschiedlicher Qualität.
- Eigene Änderungen konnten in dem CSS-Code durchgeführt werden. In der vorgegebenen Zeit konnte ein Framework wie zum Beispiel Bootstrap nicht eingebunden werden.
- Das objektrelationale Mapping, welches in PHP über Doctrine realisiert wird, konnte in der vorgegebenen Zeit nicht ausreichend untersucht werden. Beim Versuch das **FOSUserBundle** einzubinden, konnte die Datenbank generiert werden.

**Fazit** Die grundlegende Webseite konnte mit Symfony erstellt werden. Eine Administration ließ sich aufgrund der Probleme mit dem Einbinden der **FOSUserBundles** Bibliothek nicht realisieren. Des Weiteren ist das Design der Webseite sehr einfach gehalten, weil sich **Bootstrap** nicht einbinden ließ. Ein Großteil der Arbeit bestand in der Konfiguration von Symfony und anderen Komponenten. Dies macht es für Einsteiger schwierig sich in kurzer Zeit in das Framework einzuarbeiten.

**Ausblick** Es wird abgeschätzt, dass eine Einarbeitungszeit von ungefähr drei Wochen nötig ist, bis ein Entwickler genügend Erfahrung hat, um vernünftig mit Symfony arbeiten zu können. Dies sollte bei der Entscheidung zwischen PrimeFaces und Symfony berücksichtigt werden.

**4.3.3.2 Dokumentation PrimeFaces** Für die Bearbeitung des Arbeitspaketes „Struktur der Weboberfläche“ wurde PrimeFaces benutzt. Dieses sollte mit Symfony verglichen werden und zwar im Hinblick auf die Punkte:

- Änderungen zur Laufzeit
- Debugging
- Community
- Eigene CSS-Elemente einbinden
- Objekt-relationales Mapping - ORM

Es wurden dabei keine Anforderungen berücksichtigt.

**Ablauf** Die Installation von PrimeFaces ist in der IDE Eclipse Mars for Java EE durchgeführt worden, in dem ein neues Maven-Projekt erstellt wurde. Um PrimeFaces in ein Maven-Projekt zu integrieren müssen nur die folgenden Abhängigkeit in die **Pom.xml** hinzugefügt werden:

```
<dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>5.2</version>
</dependency>
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>4.3.1.Final</version>
</dependency>
```

Hibernate ist hier eine ORM-Schnittstelle, die mit in der Evaluation betrachtet werden soll. Weiterhin ist die Nutzung folgender Eclipse Mars Facetten nützlich und notwendig (Diese sind in den Projekt-Einstellungen zu finden unter **Project Facets**) :

- Dynamic Web Module - Eclipse interner Browser
- Java - Perspektive für Java-Elemente
- JavaScript - Perspektive für JavaScript-Elemente
- JavaServer Faces - Nutzung von JSF-Elementen
- JAX-RS (REST Web Service) - Nutzung von REST-Aufrufen

Als Vorlage wurde das entwickelte Gerüst von Dannys Seminararbeit benutzt und diese überarbeitet. Hierzu sind als erstes die Namen der einzelnen Klassen an die Namenskonventionen angepasst worden. Danach wurde in der Postgres-Datenbank eine neue Tabelle angelegt und mit „account_table“ betitelt. Diese enthält eine eindeutige E-Mail-Adresse, ein verschlüsseltes Passwort, einen eindeutigen Nutzernamen und eine Rolle. Anschließend wurde eine Datenbankanbindung mittels Java-Hibernate (ORM) realisiert und diese ausführlich getestet. In den Klassen **RegisterBean.java** und **LoginBean.java** sind Methoden implementiert worden, die das Registrieren und Einloggen von autorisierten Benutzern ermöglichen. Hierfür wurde das Debugging in Eclipse genutzt: Das in einer Tomcat-Umgebung ausgeführte Projekt lässt sich innerhalb von zwei Sekunden neu starten. Während der

Laufzeit konnten Änderungen durchgeführt werden und nach dem Speichern wurden diese dynamisch in die Tomcat-Umgebung geladen, neue Java-Komponenten werden nicht dynamisch hinzugefügt. Weiterhin wurde versucht die bereits entwickelte Heatmap zu integrieren. Grundsätzlich war es problemlos möglich, den HTML- , Javascript- und CSS-Inhalt der Umgebung hinzuzufügen. Da die vom alten Arbeitspaket übernommenen HTML und Javascript-Komponenten allerdings nicht durchgängig valide Syntax enthalten, hätten umfangreiche Änderungen an der alten Heatmap vorgenommen werden müssen, um sie in der PrimeFaces-Umgebung einzusetzen. Da langfristig ohnehin nicht beabsichtigt wird, diese Version der Karte zu verwenden, wurde entschieden eine neue Karte anhand der GoogleMaps-API zu entwickeln.

**Technologien**

**PrimesFaces** Entwicklung der Web-Oberfläche

**Hibernate** Anbindung an die Datenbank

**Google Maps API** Entwicklung der Heatmap

**Eclipse Mars** Benutzte Entwicklungsumgebung

**Tomcat** Bereitstellung der Web-Umgebung

**Systembeschreibung** PrimeFaces wurde in einem Maven-Projekt innerhalb von Eclipse-Mars entwickelt. Dazu bringt Mars eine Tomcat-Umgebung mit und alle Features von der Eclipse-IDE.

**Evaluation** Das Arbeitspaket selber ist die Evaluation von PrimeFaces.

**Parameter** Die Weboberfläche ist unter einer statischen IP-Adresse erreichbar und zwar nur, wenn auf dem Rechner **krtpc32** das Projekt ausgeführt wird. Die IP-Adresse lautet **134.106.47.187** und der Tomcat-Zugriff mit Port **8080** und liegt im Verzeichnis **/Medic/login.xhtml**. Das ist der vollständige Link: `http://134.106.47.187:8080/Medic/login.xhtml`

**Fazit** Mit PrimeFaces ist die Realisierung der Weboberfläche schnell und leicht umsetzbar. Die evaluierten Punkte konnten alle mit gut abschneiden und wir geben unsere Empfehlung dafür. Zusätzlich ist es in den gängigen IDEs (Eclipse, Netbeans und Intellij) möglich PrimeFaces zu integrieren. Durch die parallele Evaluation von Symfony und PrimeFaces konnte der Jenkins-Server zunächst nicht eingerichtet werden.

### 4.3.4 Login für die Weboberfläche

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket dient dazu, weitere Anforderungen umzusetzen.
- Arbeitspaketbeschreibung:
  Die Weboberfläche wird durch einen Login vor unberechtigten Zugriffen geschützt. Personen, die im Besitz eines gültigen Accounts sind, erhalten Zugriff auf die Weboberfläche. Der Login besteht aus einer Eingabemaske, in der ein Nutzername oder alternativ eine E-Mail-Adresse und ein Passwort in Textfelder eingegeben werden müssen und einem Button, mit dem die Eingaben bestätigt werden. Sämtliche anderen Inhalte der Weboberfläche sind nicht erreichbar, ohne dass vorher eine erfolgreiche Anmeldung durchgeführt wurde. Meldet sich ein Benutzer

mit korrekten Daten an, so wird er auf die Startseite der Weboberfläche weitergeleitet. Sind die Anmeldedaten nicht korrekt, so wird eine aussagekräftige Fehlermeldung angezeigt und der Login kann erneut begonnen werden.

- Vorbedingungen:
  Das Arbeitspaket „Grundstruktur des Servers“ muss abgeschlossen sein.
- Nebenbedingungen:
  Accounts werden in verschiedene Rollen unterteilt. Die Rolle eines Accounts bestimmt, auf welche Teilbereiche der Weboberfläche der Benutzer Zugriff erhält. Aus der Fehlermeldung bei falschen Logindaten ist nicht ersichtlich, welche Eingabe fehlerhaft war.
- Nachbedingungen:
  Nach Abschluss des Arbeitspaketes ist es einer Person ohne Account unter keinen Umständen möglich innerhalb von zehn Minuten Zugriff auf die Weboberfläche zu erhalten. Anhand der in der Datenbank gespeicherten Passwörter kann kein Rückschluss auf das tatsächliche Passwort gezogen werden.
- Aufwand:
  Drei Wochen
- Personen:
  - Kevin Sandermann
  - Raphael Kappes

**4.3.4.1 Dokumentation** Ziel dieses Arbeitspakets ist es, die Weboberfläche gegen unbefugte Zugriffe zu schützen. Hierfür ist eine Login-Seite vorgesehen, auf der ein Benutzer sein Account-Login sowie ein zugehöriges Passwort eingeben muss, um Zugriff auf die für ihn vorgesehenen Funktionen zu erhalten. Außer der Login-Seite sind für einen nicht eingeloggten Besucher keine Inhalte einsehbar.

**Ablauf** Zunächst wurde aus dem Arbeitspaket `Struktur der Weboberfläche - PrimeFaces` eine rudimentäre Login-Seite übernommen, welche ein Eingabefeld für das Account-Login und das Passwort enthält. Diese Seite soll später dazu dienen, dass sich ein Nutzer anhand seines Benutzernamens oder seiner E-Mail-Adresse und dem zugehörigen Passwort an der Weboberfläche anmelden kann. Damit nicht eingeloggte Benutzer keinen Zugriff zur restlichen Weboberfläche haben, wurden sämtliche Inhalte der Seite mit Authentifizierungsfiltern geschützt. Sobald eine Seite angefragt wird, überprüft der zugehörige Filter, ob ein Login erfolgt ist (d.h. eine Session angelegt wurde) und ob der evtl. eingeloggte Benutzer berechtigt ist, die Seite einzusehen. Aktuell existieren zwei Filter, die zwischen Seiteninhalten für Administratoren und Inhalten für Doktoren (und Statistikern) unterscheiden.

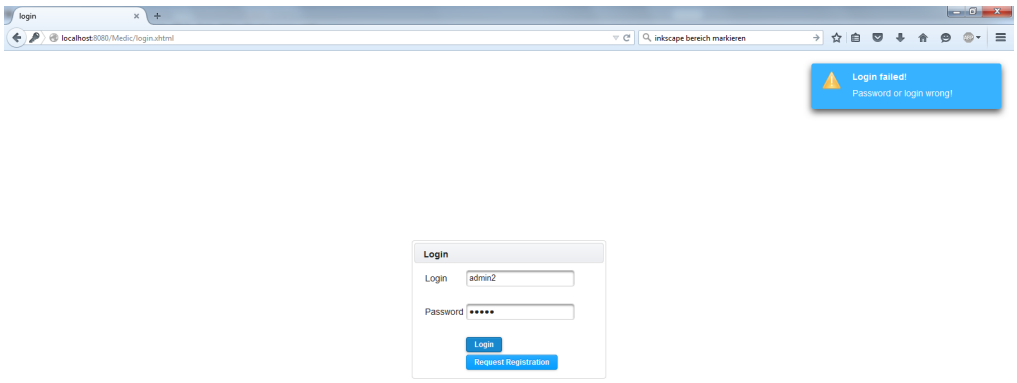



Abbildung 61: Login-Seite der Weboberfläche mit Fehlermeldung eines falschen Logins

Auf der in Abbildung 61 abgebildeten Login-Seite kann ein Nutzer zum Anmelden sowohl seinen Benutzernamen als auch seine E-Mail-Adresse zur Authentifizierung eingeben. Weiterführend muss das zugehörige Passwort eingegeben werden, um anschließend einen Login-Versuch über den Button `Login` vorzunehmen. Bei fehlgeschlagenem Login wird dem Nutzer der Zugriff auf die restliche Weboberfläche verwehrt und die entsprechende Fehlermeldung `Login failed! Password or Login wrong!` angezeigt. An dieser Stelle wird bewusst nicht konkret darauf hingewiesen, ob das Login oder das Passwort falsch ist, um ein Erraten von möglichen Account-Logins zu verhindern.
Bei einem Login-Vorgang auf der Weboberfläche testet der PrimeFaces-Server die eingegebenen Daten auf Korrektheit, um dem Benutzer anschließend Zugriff zu gewähren oder nicht. Hierfür wird in einem ersten Schritt festgestellt, ob der Inhalt der Login-Eingabemaske valide E-Mail-Adressen-Syntax enthält. Ist dies der Fall, wird angenommen, dass sich der Nutzer versucht anhand einer E-Mail-Adresse anzumelden. Ist dies nicht der Fall, wird von einem Benutzernamen als Login ausgegangen. Anschließend wird in der Datenbank in der Tabelle `account_table` nach einem zugehörigen Eintrag gesucht. Existiert kein Eintrag (also kein Account) mit dem zugehörigen Login, wird dem Nutzer der Zugriff zur Seite verwehrt und die oben genannte Fehlermeldung angezeigt. Existiert ein zugehöriger Eintrag, muss als nächstes das Passwort überprüft werden. Da sämtliche Passwörter in der Datenbank gehasht gespeichert sind, wird an dieser Stelle auch der Inhalt des vom Benutzer eingegeben Passwort gehasht und der so errechnete Hash mit dem Inhalt des zugehörigen Datenbankeintrags verglichen. Wurde ein falsches Passwort eingegeben, wird der Zugriff verwehrt und die entsprechende Fehlermeldung angezeigt. Ist das Passwort korrekt, wird der Nutzer eingeloggt. Dieser Vorgang umfasst das Anlegen einer Zuordnung vom validen Login zur vom PrimeFaces-Framework automatisch angelegten HTTP-Session. Dadurch kann der oben genannte Authentifizierungsfilter bei jedem Seitenaufruf direkt anhand der HTTP-Session erkennen, ob ein Benutzer bereits eingeloggt ist oder nicht.
Auf den weiterführenden Seiten der Weboberfläche ist zu jeder Zeit ein Logout-Button verfügbar, welcher es dem Nutzer ermöglicht sich auszuloggen und somit die Zuordnung vom Benutzer zur HTTP-Session zu löschen. Dies geschieht unabhängig von einer Abmeldung außerdem nach 24 Stunden automatisiert. Abbildung 62 zeigt diesen Button.

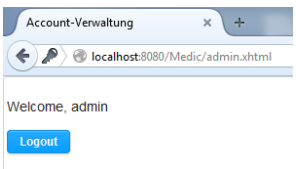



Abbildung 62: Logout-Button, welcher jedem Nutzer in seinen entsprechenden Bereichen zu jeder Zeit zur Verfügung steht.

Wie oben beschrieben wird bei jedem Nutzer außerdem zwischen den Rollen `Systemadministrator`

und `Mediziner` bzw. `Statistiker` unterschieden. Sobald ein Zugriff auf Seiteninhalte erfolgt, überprüft der Authentifizierungsfilter das Login auf seine Rolle und gewährt nur entsprechenden Zugriff. Falls die Berechtigung für eine angeforderte Seite fehlt, wird der Benutzer auf eine zugehörige Standardseite weitergeleitet.

**Technologien** Es wurden die selben Technologien aus dem Arbeitspaket `Struktur der Weboberfläche - PrimeFaces` verwendet.

**Systembeschreibung** Die Entwicklungsumgebung des Projekts ist ebenfalls die Gleiche wie in Arbeitspaket `Struktur der Weboberfläche - PrimeFaces`

**Evaluation** Das Login wurde ausführlich unter den aktuellen Versionen der Browser Mozilla Firefox, Google Chrome, Microsoft Internet Explorer und Opera getestet.

**Parameter** Das Login funktioniert als Teil der Weboberfläche. Diese kann beispielsweise als Projekt in einer Tomcat-Umgebung betrieben werden.

**Fazit** Das Arbeitspaket konnte erfolgreich abgeschlossen werden. Es konnten alle Anforderungen erfüllt werden. Das Arbeitspaket wurde bereits mit einem Drittel des geplanten Zeitaufwands, d.h. zwei Wochen vor der eigentlichen Deadline vom 26.08.2015, abgeschlossen. Da das Login ein kritischer Bestandteil des gesamten Systems ist, sollte vom gesamten Team während der Bearbeitung weiterer Arbeitspakete an der Weboberfläche ständig eine Überprüfung auf Sicherheitslücken und Fehlverhalten erfolgen.

### 4.3.5 Umsetzung des Usermanagment mit PrimeFaces

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Die Benutzerverwaltung soll mit der PrimeFaces Bibliothek umgesetzt werden.
- Arbeitspaketbeschreibung:
  Die Weboberfläche erlaubt es einem Administrator die Benutzer des Systems in einer gesonderten Ansicht zu verwalten. Diese Ansicht stellt eine Liste bereit, in der Benutzer des Systems dargestellt werden. In dieser Liste kann ein Administrator den Namen (sowohl den Accountnamen als auch den realen Namen), die Anschrift und die Rolle des Nutzers, sowie seine E-Mail-Adresse ablesen. Es ist möglich dem System neue Benutzer hinzuzufügen und bestehende Nutzer zu löschen. Beim Erstellen eines Benutzers wird automatisch ein zugehöriges Passwort generiert und an die entsprechende E-Mail-Adresse versendet. Darüber hinaus ist es nicht möglich, ungültige Eingaben wie eine E-Mail-Adresse, die nicht der üblichen Form entspricht, zu speichern.
- Vorbedingungen:
  Das Arbeitspaket „Struktur der Weboberfläche“ muss abgeschlossen sein.
- Nebenbedingungen:
  Passwörter müssen eine Mindestlänge von acht Zeichen haben.

  Es muss ein Administrator existieren, der unter keinen Umständen gelöscht, oder dessen Rolle verändert werden kann. Dieser muss nicht manuell dem System hinzugefügt werden, sondern existiert immer in dem System.

Zusätzlich kann ein Administrator sich nicht selbst löschen oder seine Rolle ändern. Er kann jedoch einen anderen Administrator löschen und dessen Rolle ändern.

Ein Administrator kann sämtliche Daten eines Benutzers, mit Ausnahme des Passworts, ändern. Dabei wird, ebenso wie beim Anlegen eines neuen Benutzers, die Validität der Eingaben verifiziert.

Es können keine zwei Benutzer mit dem gleichen Namen gleichzeitig existieren. Versucht ein Administrator einen Namen zweifach zu vergeben, so wird dies in einer Fehlermeldung angezeigt und der Name nicht vergeben.

Das Usermanagement ist nur von Administratoren einsehbar.

- Nachbedingungen:
  Die Liste mit mindestens zehn Benutzern wird innerhalb von höchstens einer Sekunde angezeigt. Ein neuer Benutzer kann innerhalb einer Minute vom Administrator angelegt werden.
- Aufwand:
  Drei Wochen
- Personen:
  - Kevin Sandermann
  - Raphael Kappes

**4.3.5.1 Dokumentation** Ziel dieses Arbeitspakets ist es, die Weboberfläche dahingehend zu ergänzen, dass es einem Administrator möglich ist bestehende Accounts zu verwalten und ggf. zu editieren oder zu löschen.

**Ablauf** Als Vorbedingung dieses Arbeitspakets galt die aus dem Arbeitspaket `Login in die Weboberfläche` erstellte Login-Seite der Weboberfläche. Diese wurde neben der bereits fertigen Login-Funktion mit einer Funktion zur Anfrage einer Account-Registrierung erweitert. Diese Account-Registrierung stellt eine Anfrage für einen Benutzeraccount dar, welche anschließend von einem Administrator überprüft und bestätigt werden muss. Bei der Registrierung muss der Nutzer zunächst seine E-Mail-Adresse, ein gewünschtes Passwort, die Wiederholung seines gewünschten Passworts sowie seinen Nutzernamen (Account-Loginnamen) vergeben. Ein Passwort muss aus mindestens acht Zeichen bestehen, damit keine Fehlermeldung ausgegeben wird. Weiterführend ist es für einen Mediziner nötig, dass er seine personenbezogenen Daten (Vorname, Nachname, Titel, etc.), seine Praxisadresse, die zuständige Aufsichtsbehörde sowie die Daten der Ärztekammer angibt. Diese Daten werden aktuell noch nicht vom System überprüft, da sie keinen Teil dieses Arbeitspakets darstellen. Im Laufe dieses Pakets wurde allerdings deutlich, dass es ein zukünftiges Arbeitspaket geben muss, welche sich mit dieser Thematik auseinandersetzt und sie löst. Nach dem Eintragen aller Daten kann die Anfrage einer Registrierung entweder mit einem `Schließen`-Button abgebrochen werden oder über die Schaltfläche `Request Doctor-Registration!` abgeschlossen werden. Bei Anwählen dieses Buttons werden die eingegebenen Daten direkt live vom System überprüft, noch bevor sie an die Datenbank oder einen Administrator weitergegeben werden. Stimmen etwa die beiden eingegebenen Passwörter nicht überein, wird eine entsprechende Fehlermeldung angezeigt und der Nutzer zur Korrektur aufgefordert. Sind etwa die E-Mail-Adresse oder der eindeutige Nutzername bereits von einem anderen Nutzer in Gebrauch, wird dies an den Nutzer zurückgemeldet. Nachdem die Daten korrigiert wurden, wird die Registrierungsanfrage entgegengenommen und persistiert. Währenddessen wird der Nutzer per E-Mail darüber informiert, dass seine Anfrage entgegen genommen wurde und nun noch von einem zuständigen Administrator überprüft und freigegeben werden muss. Versucht der Nutzer

sich mit seinem Account anzumelden, bevor dieser bestätigt wurde, wird eine passende Fehlermeldung ausgegeben. Abbildung 63 zeigt die Registrierungsmaske aus Sicht eines Mediziners. Aktuell ist es nur für Mediziner möglich, eine Account-Anfrage auszuführen.

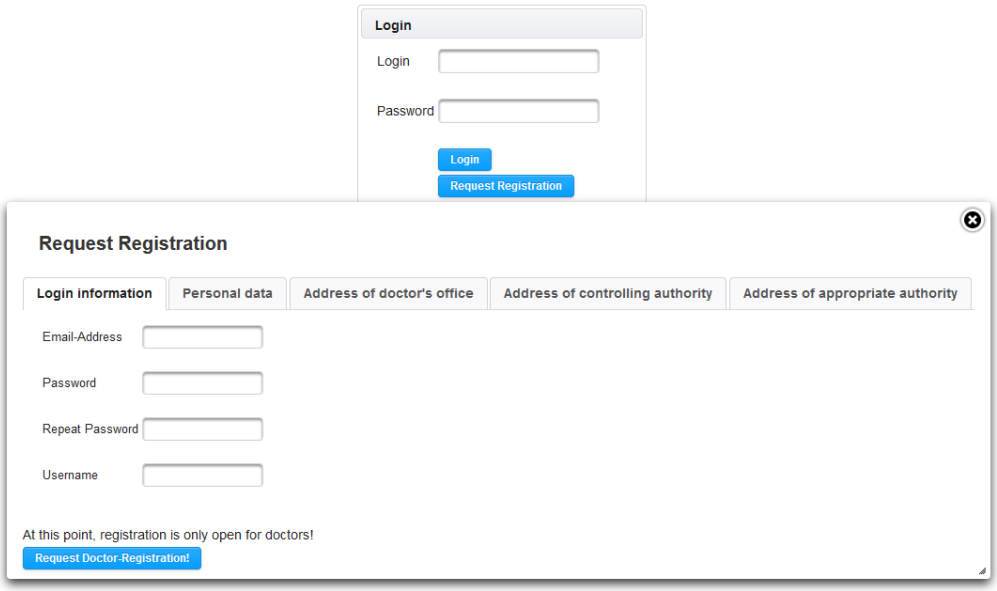



Abbildung 63: Registrierungs-Oberfläche eines Nutzers

Aus Sicht eines Administrators lassen sich zwei verschiedene Sichten auf alle existierenden Accounts auswählen: Eine Liste aller noch nicht bestätigten Accounts und eine Liste aller bereits im System verwendeten und bestätigten Accounts. Die Liste aller noch nicht bestätigten Accounts ist auf Abbildung 64 zu sehen.


Abbildung 64: Liste aller nicht bestätigten Accounts in der Administrationsoberfläche

Wie zu sehen ist, werden einem Administrator zunächst in der Tabelle die wichtigsten dem Account zugehörigen Daten angezeigt. Anhand dieser Informationen muss ein Administrator einen Mediziner auf Zugangsberechtigung prüfen, d.h. seine Praxisdaten und Zulassung müssen bestätigt werden. An dieser Stelle ist es noch nicht möglich, sämtliche Inhalte einer Account-Anfrage anzuzeigen. Diese Funktion wurde im Rahmen dieses Arbeitspaket als nötig empfunden und sollte in einem zukünftigen Arbeitspaket umgesetzt werden. Ist die Überprüfung der Daten geschehen, hat der Administrator über zwei entsprechende Schaltflächen die Möglichkeit, die Registrierung zu bestätigen oder abzulehnen. Ein Nutzer wird per E-Mail über eine etwaige Entscheidung eines Administrators informiert. Als in Zukunft ebenfalls nützliche Funktion lässt sich die optionale Erfassung einer Nachricht vom Administrator an den Nutzer sehen (etwa mit Gründen für eine Ablehnung). Anschließend wird der jetzt bestätigte oder abgelehnte Account aus der Liste gelöscht und findet sich bei eventueller Bestätigung in der unten dargestellten Liste aller bestätigten Accounts wieder:

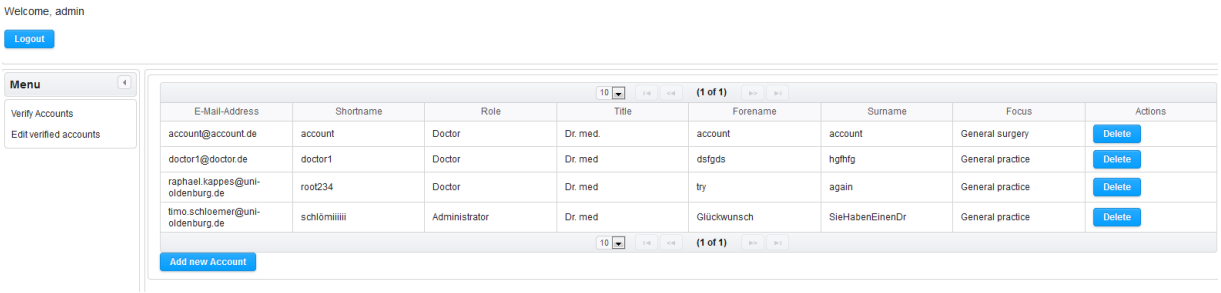



Abbildung 65: Liste aller bestätigten Accounts in der Administrationsoberfläche

In dieser Ansicht werden alle Accounts aufgelistet, die bereits von einem Administrator bestätigt wurden und die bereits zum Login genutzt werden können. In der Tabelle werden wieder die wichtigsten Randdaten des Accounts angezeigt, um einen schnellen Überblick und eine schnelle Identifikation eines Accounts zu ermöglichen. Über die Schaltfläche `Delete` ist es einem Administrator möglich einen Account zu löschen. Wird ein Account gelöscht, wird ein Nutzer hierüber per E-Mail informiert und die Liste des Administrators aktualisiert. Weitergehend hat ein Administrator an dieser Stelle die Möglichkeit, die Daten eines bestehenden Accounts zu editieren. Dies geschieht über einen Klick auf den entsprechenden Eintrag in der Liste. Hierdurch öffnet sich ein entsprechendes Fenster, welches die Änderung der E-Mail-Adresse, des Nutzernamens und der Rolle eines Accounts ermöglicht. Letzteres ist in folgender Abbildung 66 auf der nächsten Seite dargestellt:

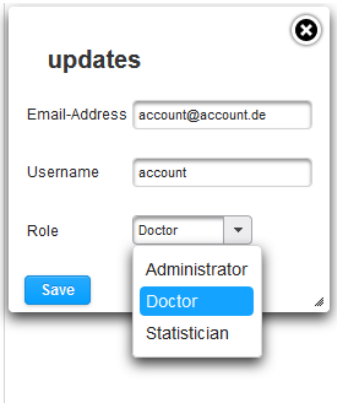



Abbildung 66: Oberfläche zur Account-Editierung aus Sicht eines Administrators

Bei eventuellen Änderungen wird der zugehörige Nutzer per E-Mail informiert. An dieser Stelle ist es einem Administrator nicht möglich, seinen eigenen Account zu editieren oder zu löschen.
Die möglichen Rollen sind, wie bereits in den Anforderungen festgelegt, die eines Mediziners, eines Statistikers und eines Administrators. Neben der Editierung und Löschung von bestehenden Accounts besteht für einen Administrator außerdem die Möglichkeit, neue Accounts anzulegen. Die entsprechende Schaltfläche öffnet das gleiche Fenster, welches auch bei der Account-Registrierung von anderen Benutzern angezeigt wird. Auch die eingegebenen Daten des Administrators werden vom System erst bei Korrektheit entgegengenommen. Erstellt ein Administrator einen Account, ist es ihm allerdings nicht möglich, ein Passwort für den Nutzer zu wählen. Stattdessen wird vom System ein 26-stelliges Zufallspasswort aus Buchstaben, Zahlen und Sonderzeichen generiert. Der Nutzer wird hierüber per E-Mail informiert, d.h. einem Administrator ist es unmöglich, Passwörter einzusehen. Ein vom Administrator angelegter Account ist zunächst ebenfalls nicht bestätigt, d.h. er muss wiederum durch einen Administrator bestätigt oder abgelehnt werden.
Neben den oben genannten Administrator-Accounts existiert ein festes Root-Admin-Konto, welches von keinem anderen Konto eingesehen oder in irgendeiner Weise manipuliert werden kann.

### Technologien

**Google Mail** Es wurde ein Account bei Google Mail angelegt mit dem Registrierungsanfragen und Änderungen eines Accounts dem Accountinhaber mitgeteilt werden kann.

**Systembeschreibung** Die Entwicklungsumgebung des Projekts ist ebenfalls die gleiche wie in Arbeitspaket `Struktur der Weboberfläche - PrimeFaces`

**Evaluation** Das Usermanagement wurde ausführlich unter den aktuellen Versionen der Browser Mozilla Firefox, Google Chrome, Microsoft Internet Explorer und Opera getestet.

**Parameter** Das Usermanagement funktioniert als Teil der Weboberfläche. Diese kann beispielsweise als Projekt in einer Tomcat-Umgebung betrieben werden.

**Fazit** Zusammenfassend lässt sich sagen, dass das Arbeitspaket vollständig umgesetzt und erfolgreich abgeschlossen werden konnte. Alle vorher formulierten Bedingungen und Funktionen konnten umgesetzt und gelöst werden. Als Ausblick lässt sich die Erschließung eines neues Arbeitspakets für die Zukunft sagen, welche sich der Prüfung der gesundheitsspezifischen Daten von Nutzern und dessen Evaluierung annimmt.

### 4.3.6 Anpassung der Testübersicht in der Weboberfläche

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket dient dazu, die in der Weboberfläche dargestellten Tests anzuzeigen. Dazu soll die Bibliothek PrimeFaces verwendet werden.
- Arbeitspaketbeschreibung:
  Die Weboberfläche unterstützt das Anzeigen einer Liste von Krankheitstests, die von den Probanden an den Server übermittelt wurden. Die Liste wird beim Aufrufen nicht sofort mit allen verfügbaren Tests geladen, sondern lädt immer nur einen Teil der vorhandenen Daten. In der Liste werden nur die wichtigsten Informationen zu den Krankheitstests angezeigt (Ergebnis der Voranalyse, Datum, Uhrzeit, Diagnose vorhanden: Ja / Nein) und erst wenn ein Test ausgewählt wird, werden alle weiteren Details zu diesem dargestellt.
- Vorbedingungen:
  Das Arbeitspaket „Struktur der Weboberfläche“ muss abgeschlossen sein.
- Nebenbedingungen:
  Die Liste der Krankheitstests ist nur von Medizinern einsehbar.
- Nachbedingungen:
  Es sind mindestens zehn Tests innerhalb einer Sekunde nach Aufruf der Liste sichtbar. Einem Mediziner ist es möglich eine Menge von mindestens 100 Tests mit der Liste zu betrachten.
- Aufwand:
  Zwei Wochen
- Personen:
  - Jan Philipp Stubbe
  - Christian Sandmann

**4.3.6.1 Dokumentation** Dieses Arbeitspaket befasst sich mit der Darstellung der Liste mit den durchgeführten Tests. Dabei sind folgende Anwendungsfälle von Bedeutung:

**FA-23:** Die Auswertungsoberfläche stellt dem Mediziner eine Liste mit Probanden bereit, dessen Diagnosen noch ausstehen.

**FA-27:** Bei der Erstellung einer Diagnose stellt die Auswertungsoberfläche dem Mediziner Informationen des zugrundeliegenden Tests bereit. Diese Informationen sind das Foto des Teststreifens, die geographischen Daten des Tests und detaillierte Informationen zur Teststreifenvariante.

**NFA-5:** Sowohl die Administrationsoberfläche als auch die Auswertungsoberfläche müssen benutzerfreundlich und einfach gestaltet sein. Etwaige Benutzergruppen können die Oberfläche intuitiv innerhalb von zehn Einarbeitungsminuten verstehen und fehlerfrei bedienen.

**NFA-15:** Sämtliche Dokumentationen und Kommentare im Quellcode des zu entwickelnden Systems sind auf Englisch verfasst. Interne Namen von Objekten im Quellcode sind in Englisch formuliert.

**NFA-18:** Sowohl die Administrationsoberfläche als auch die Auswertungsoberfläche unterstützen folgende Webbrowser: Google Chrome ab Version 42, Mozilla Firefox ab Version 37, Microsoft Internet Explorer ab Version 11 und Apple Safari ab Version 8.

**Ablauf** Als Grundlage für die Implementierung diente der Code von Danny Fonk, der im Rahmen seiner Seminararbeit entwickelt wurde. Zunächst wurde die Darstellung geändert: die Spalten wurden neu aufgeteilt, sodass nun für die Einträge ID, Ergebnis der Voranalyse und Datum des Tests eigene Spalten angezeigt werden, sodass die Übersichtlichkeit verbessert wurde und um die Filterung und Sortierung vorzubereiten. Außerdem wurde eine Funktion hinzugefügt, um die Tests zu zählen. Damit konnte das Blättern der Tests korrekt implementiert werden.

**Technologien**

**PrimeFaces** Framework für die Entwicklung der Weboberfläche

**Systembeschreibung** Für die Entwicklung wurde Eclipse Mars, mit Java Version 8 und Maven verwendet.

**Evaluation** Getestet wurde in den Browsern Chrome, Firefox und Internet Explorer. Die Messung beim Laden der Tests wurde manuell durchgeführt, die Tests konnten innerhalb von einer Sekunde geladen und angezeigt werden.

**Parameter** Die Geschwindigkeit beim Laden der Tests hängt davon ab, wie gut die Verbindung zwischen Server und Datenbank ist.

**Fazit** Das Arbeitspaket wurde erfolgreich abgeschlossen werden, die Tests können angezeigt werden. Es werden nicht alle Tests sofort geladen, sondern es wird nur eine Menge von 20 Tests angezeigt. Über eine Umblätterfunktion können die anderen Tests nachgeladen werden. Das Arbeitspaket konnte eine Woche vor Ablauf der Deadline vollendet werden.

**Ausblick** Es konnte die Grundstruktur für das Arbeitspaket „Filtern und Sortieren der Liste mit den Tests“ (siehe dazu Kapitel 4.3.7.1 auf der nächsten Seite) geschaffen werden.

### 4.3.7 Filtern und Sortieren der Liste mit den Tests

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket dient der Umsetzung weiterer Anforderungen mit der Bibliothek PrimeFaces.
- Arbeitspaketbeschreibung:
  Die Weboberfläche für die Mediziner enthält in der Listen-Ansicht eine Funktion, die eine Sortierung und Filterung der Tests ermöglicht. Eine Sortierung ist nach den folgenden Kriterien möglich:
  - Das Ergebnis des Testes (positiv, negativ, unklar)
  - Die Postleitzahl, die durch den QR-Code mitgesendet wird (Ausgabe des Tests)
  - Ob bereits eine Diagnose durchgeführt und dem Probanden zugesendet wurde oder noch eine Diagnose aussteht
  - Nach der Aktualität des Datums

  Damit nicht immer eine lange Liste an Tests angezeigt wird enthält die Weboberfläche eine Filterfunktion, die eine Eingrenzung nach den folgenden Kriterien ermöglicht:

  - Es werden nur Tests angezeigt die negativ sind.
  - Es werden nur Tests angezeigt die positiv sind.
  - Es werden nur Tests angezeigt die kein eindeutiges Ergebnis liefern konnten.
  - Es werden nur Tests ohne Diagnose angezeigt.
  - Es werden nur Tests der: letzten Stunde, des letztens Tages, der letzten Woche angezeigt.

  Alle Filterfunktionen können zudem kombiniert verwendet werden. Zum Beispiel können alle positiven und negativen Tests der letzten Stunde ohne Diagnose angezeigt werden.

- Vorbedingungen:
  Die Arbeitspakete „Grundstruktur des Servers“ und „Testübersicht“ müssen erfolgreich abgeschlossen sein.

- Nebenbedingungen:
  -

- Nachbedingungen:
  Ein nicht an der Entwicklung beteiligtes Teammitglied oder eine externe Person versteht in maximal drei Minuten die Sortier- und Filterfunktion und kann diese anwenden, um sich die gewünschten Tests anzeigen zu lassen. Zudem ist es durch betätigen der Oberflächenelemente nicht möglich die Liste bzw. die Weboberfläche zum Abstürzen zu bringen. Eine Sortierung oder eine Filterung von 1000 Tests dauert in der Ausführung nicht länger als fünf Sekunden.

- Aufwand:
  Drei Wochen

- Personen:
  - Jan Philipp Stubbe
  - Christian Sandmann

**4.3.7.1 Dokumentation** Ziel dieses Arbeitspaketes war es, die Liste mit den Tests, die auf der Weboberfläche angezeigt wird, nach Kriterien, wie ID, Diagnoseergebnis der Voranalyse, Postleitzahl, Datum und durchgeführte Diagnose, filtern und sortieren zu können. Dabei ist folgende Anforderung zu beachten:

**FA-23:** Die Auswertungsoberfläche stellt dem Mediziner eine Liste mit Probanden bereit, dessen Diagnosen noch ausstehen.

**Ablauf** Für die Umsetzung wurden zunächst die benötigten Spalten in der Tabelle hinzugefügt. Dazu wurden die benötigten Eingabeelemente, wie Textfelder und Kalender, implementiert (siehe Abbildung 67 auf Seite 232). Anschließend wurde das eigentliche Filtern und Sortieren über die Datenbank realisiert. Dabei war zu beachten, dass nicht alle Tests aus der Datenbank geladen werden, damit die zu übertragende Datenmenge reduziert wird. Das führte dazu, dass die von PrimeFaces bereitgestellten Filter- und Sortierfunktionen nicht genutzt werden konnten. Deshalb wurde die Filterung über die Datenbank umgesetzt. Des Weiteren wurde das Laden der Tests so angepasst, dass die Bilder erst aus der Datenbank geladen werden, wenn ein einzelner Test aufgerufen wird. Da die Postleitzahl zu diesem Zeitpunkt noch nicht gespeichert wird und nicht ausgelesen werden kann, wurde diese bei der Filterung und Sortierung nicht berücksichtigt. Wird nach der ID gefiltert, dann werden alle IDs angezeigt, in der die Filterid vorkommt. Um nach dem Ergebnis der Voranalyse zu filtern,

können die Werte aus einem Auswahlmenü mit den Optionen **alle**, **nur positive**, **nur negative** oder **unklare** Testergebnisse gewählt werden. Ähnlich werden auch die Werte für die durchgeführte Diagnose aus einem Auswahlmenü mit den Optionen `Ja` und `Nein` gewählt. Anstatt die Tests nur nach der letzten Stunde, des letzten Tages oder der letzten Woche filtern zu können, kann der Zeitraum über einen Kalender variabel eingestellt werden.

**Technologien**

**Primefaces** Framework für die Entwicklung der Weboberfläche

**Hibernate** Javaframework für objektrelationale Abbildungen aus der Datenbank

**Systembeschreibung** Die Entwicklungsumgebung ist die gleiche wie in dem Arbeitspaket „Testübersicht“ (siehe dazu Kapitel 4.3.6.1 auf Seite 114).

**Evaluation** Die Filterung und Sortierung wurde in den aktuellen Versionen von Firefox, Chrome und Edge getestet. Des Weiteren wurde ein nicht an der Entwicklung beteiligtes Teammitglied hinzugezogen, welches innerhalb von drei Minuten die Funktionen verstehen konnte. Die Filterung und Sortierung dauert in der Ausführung nicht länger als fünf Sekunden.

**Parameter** Die Geschwindigkeit beim Filtern und Sortieren der Tests hängt von der Verbindung zwischen Server und Datenbank ab.

**Fazit** Das Arbeitspaket wurde erfolgreich abgeschlossen. Die Tests können nach allen angezeigten Werten sortiert werden. Außerdem lassen sich die Tests nach allen Kriterien filtern. Dabei lassen sich die Filter beliebig kombinieren. Das Arbeitspaket konnte in der vorgegeben Zeit von zwei Wochen fertig gestellt werden.

**Ausblick** In Zukunft können weitere Spalten hinzugefügt werden. Dies umfasst insbesondere die Postleitzahl, die zu diesem Zeitpunkt noch nicht berücksichtigt wurde.

### 4.3.8 Rückmeldung an den Patienten

- Priorität: Normal

- Motivation und Nutzen des Arbeitspaketes:
  Durch die Umsetzung dieses Arbeitspaketes soll weitere Anforderungen abgearbeitet werden.

- Arbeitspaketbeschreibung:
  Über die Liste der Krankheitstests in der Weboberfläche kann ein Mediziner einen Test auswählen und eine Diagnose für diesen schreiben. Wenn ein Test ausgewählt ist, wird in der Detailansicht des Tests ein Button dargestellt, welcher bei Betätigung die Eingabemaske für die Diagnose anzeigt. Die Eingabemaske enthält ein Textfeld für einen Kommentar, eine Auswahlbox für das vom Mediziner bestätigte Testergebnis (positiv, negativ, nicht identifizierbar) und zwei Buttons. Ein Button dient dazu, die Diagnose auf dem Server zu speichern aber für spätere Änderungen offen zu lassen. Der andere Button speichert die Diagnose auf dem Server, sendet die Diagnose an den Probanden des Tests und sperrt die Diagnose für nachträgliche Änderungen.

- Vorbedingungen:
  Die Arbeitspakete „Struktur der Weboberfläche“ und „Testübersicht “ sind abgeschlossen.

- Nebenbedingungen:
  -
- Nachbedingungen:
  Ein nicht an der Entwicklung beteiligtes Projektgruppenmitglied kann innerhalb von zwei Minuten eine Diagnose verfassen und abspeichern.
- Aufwand:
  Zwei Wochen
- Personen:
  - Danny Fonk
  - Timo Schlömer

**4.3.8.1 Dokumentation** In diesem Arbeitspaket wird es den Medizinern ermöglicht, eine Diagnose in der Testübersicht abzugeben und zu speichern. Dadurch werden folgende Anforderungen erfüllt:

**FA-26** Bei der Diagnose eines Tests ermöglicht es die Auswertungsoberfläche dem Mediziner den Test als positiv, negativ oder unlesbar zu erklären.

**FA-27** Bei der Erstellung einer Diagnose stellt die Auswertungsoberfläche dem Mediziner Informationen des zugrundeliegenden Tests bereit. Diese Informationen sind das Foto des Teststreifens, die geographischen Daten des Tests und detaillierte Informationen zur Teststreifenvariante.

**FA-28** Während der Diagnose ermöglicht es die Auswertungsoberfläche dem Mediziner, seine Diagnose mit der Voranalyse der mobilen Applikation zu vergleichen.

**Ablauf** Die Arbeit basiert auf den Fortschritten der Arbeitspakete „Struktur der Weboberfläche“ (siehe Kapitel 4.3.3 auf Seite 101) und „Testübersicht“ (Kapitel 4.3.6.1 auf Seite 114). So konnte die Anzeige der Diagnosefunktion in die Testübersicht integriert werden, wodurch Mediziner leichter auf die einzelnen Tests zugreifen können. Wählt ein Mediziner aus der Liste der Krankheitstests einen aus, so wird unter den detaillierten Informationen ein Button angezeigt, mit dem sich der Diagnose-Dialog öffnen lässt. In diesem Dialog kann der Mediziner auswählen, ob der Krankheitstest als 'positiv', 'negativ' oder 'unbekannt' zu werten ist. Zusätzlich kann der Mediziner in einem Textbereich einen zusätzlichen Kommentar abgeben. Es gibt auch die Möglichkeit die Diagnose als 'endgültig' zu definieren, so dass keine weiteren Änderungen mehr möglich sind. Wird der Dialog so gespeichert, kann der Diagnose-Dialog nicht mehr geöffnet werden. Andernfalls kann jeder Mediziner eine bestehende Diagnose verändern und abspeichern.

**Systembeschreibung** Das Arbeitspaket wurde unter den gleichen Bedingungen entwickelt wie das Arbeitspaket „Struktur der Weboberfläche - PrimeFaces“ (Kapitel 4.3.3.2 auf Seite 105).

**Evaluation** Ein nicht an der Entwicklung beteiligtes Mitglied der Projektgruppe konnte eine Diagnose in unter einer halben Minute ausfüllen und abspeichern. Außerdem wurde getestet, dass der Diagnose-Dialog eine beliebige Eingabe von Daten korrekt verarbeiten kann.

**Parameter** Die Funktion des Diagnose-Dialogs wird durch keine Parameter eingeschränkt.

**Fazit** Das Arbeitspaket konnte fristgerecht und einfach mittels PrimeFaces und JavaEE entwickelt werden. Die Darstellung des Dialogs und die Anordnung des Buttons betten sich in den Rest der Webseite ein und passen in das Design.

**Ausblick** In einem Bereich des Diagnose-Dialogs wird dem Mediziner angezeigt, welcher die Diagnose als letzter bearbeitet hat. Dies ist bislang aber nur die ID dieses Mediziners. Es sollte in Erwägung gezogen werden, ob dort stattdessen etwas anderes angezeigt werden kann. Hierbei sollte überlegt werden, welche Informationen für andere Mediziner interessant sind und auf welche Weise dies die Privatsphäre der Mediziner beeinflussen könnte.

### 4.3.9 Accountverwaltung vom Benutzer

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Aus der Seminararbeit von Danny Fonk ging hervor, dass alle Komponenten der Weboberfläche mit der Bibliothek PrimeFaces umgesetzt werden sollen.
- Arbeitspaketbeschreibung:
  Die Weboberfläche bietet Benutzern die Möglichkeit eine Verwaltungsansicht für ihren benutzerspezifischen Account aufzurufen. In dieser Ansicht werden der Account-Name, die gespeicherten Personendaten (realer Name und Anschrift) und die mit dem Account assoziierte E-Mail-Adresse dargestellt. Darüber hinaus existieren Funktionen, die es ermöglichen die E-Mail-Adresse, Anschrift und das Passwort zu ändern. Beim Betätigen des entsprechenden Buttons wird eine Eingabemaske aufgerufen, in der die neuen Daten (Email-Adresse, Anschrift oder Passwort) eingeben werden können. Um unberechtigten Zugriff zu unterbinden muss zusätzlich zum neuen Passwort/zur neuen E-Mail-Adresse/zur neuen Anschrift das aktuelle Passwort des Accounts eingegeben werden. Es ist nicht möglich eine E-Mail-Adresse, die nicht der korrekten Form entspricht, zu speichern.
- Vorbedingungen:
  Das Arbeitspaket „Login in die Weboberfläche“ muss abgeschlossen sein.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Einem Benutzer ist es innerhalb von fünf Minuten möglich, sowohl das Passwort als auch die E-Mail-Adresse seines Accounts zu ändern.
- Aufwand:
  Zwei Wochen
- Personen:
  - Christian Sandmann
  - Jan Philipp Stubbe

**4.3.9.1 Dokumentation** Ziel dieses Arbeitspaketes war es, dem eingeloggten Benutzer eine Oberfläche anzubieten, über die der Benutzer seine E-Mail-Adresse und sein Passwort ändern kann. Dabei soll der Benutzer dazu aufgefordert werden, bei jeder Änderung sein Passwort einzugeben, damit keine unberechtigten Personen diese Änderungen durchführen können.

**Ablauf** Für die Umsetzung wurde zunächst ein neuer Menüpunkt in der Ansicht für Administratoren und für Mediziner eingefügt. Über diesen gelangen die Benutzer zu der Ansicht ihres Profils und können ihre E-Mail-Adresse und ihr Passwort bearbeiten (vgl. Abbildung 68). Hier hat der Benutzer die Möglichkeit zwischen den Buttons `change email` und `change password` auszuwählen. Es werden entsprechend die beiden Dialoge aus den Abbildungen 120 und 120 angezeigt, über welche die Änderungen vorgenommen werden können.


Abbildung 68: Profilansicht für die Änderungen am eigenen Account


Abbildung 69: Dialog zum Ändern der E-Mail-Adresse

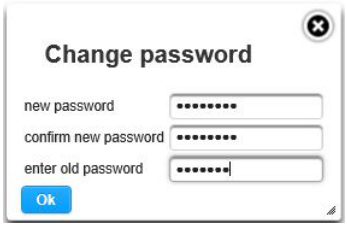



Abbildung 70: Dialog zum Ändern des Passworts

Vor der Durchführung der Änderungen findet jeweils eine Validierung statt. In beiden Fällen wird überprüft, ob das eingegebene Passwort mit dem Passwort des aktuell eingeloggten Benutzers übereinstimmt. Wird die E-Mail-Adresse geändert, dann wird die Syntax der E-Mail-Adresse überprüft und ggf. eine Fehlermeldung ausgegeben. Beim Ändern des Passworts wird überprüft, ob die neuen Passwörter übereinstimmen und ob diese mindestens acht Zeichen lang sind. Ist dies nicht der Fall, dann wird auch hier eine Fehlermeldung angezeigt. Treten mehrere Fehler auf, werden mehrere Meldungen angezeigt, wie in Abbildung 71 auf Seite 233 zu sehen ist. Nur wenn alle Bedingungen erfüllt sind und die Änderung erfolgreich in die Datenbank geschrieben werden konnte, wird das Passwort bzw. die E-Mail-Adresse geändert und der Nutzer erhält eine positive Rückmeldung.

**Technologien**

**Primefaces** Framework für die Entwicklung der Weboberfläche

**Hibernate** Javaframework für objektrelationale Abbildungen aus der Datenbank

**Systembeschreibung** Die Entwicklungsumgebung ist die Gleiche wie in dem Arbeitspaket „Testübersicht“ (Kapitel 4.3.6.1 auf Seite 114).

**Evaluation** Das Ändern der Accountinformationen wurde in den aktuellen Versionen der Browser Firefox, Chrome und Internet Explorer getestet. Des Weiteren wurde eine Testperson, die nicht an der Entwicklung beteiligt war, hinzugezogen, um die Funktionen zu testen. Die Testperson war dazu in der Lage, sich innerhalb der vorgegebenen fünf Minuten in die Funktionen einzuarbeiten und die E-Mail-Adresse und das Passwort zu ändern.

**Parameter** Die Funktion der Accountverwaltung vom Benutzer wird durch keine Parameter eingeschränkt.

**Fazit** Das Arbeitspaket wurde erfolgreich vor Erreichen der Deadline abgeschlossen. Alle geforderten Bedingungen und Nachbedingungen konnten erfüllt werden.

**Ausblick** In Zukunft können weitere Attribute zu den Benutzern hinzugefügt werden. Die Accountverwaltung lässt sich so erweitern, dass die neuen Attribute ebenfalls bearbeitet werden können. Die Gestaltung der Seite könnte ebenfalls verbessert werden, was im Rahmen eines weiteren Arbeitspaketes denkbar wäre.

### 4.3.10 Passwort wiederherstellen in der Weboberfläche

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Durch die Umsetzung dieses Arbeitspaketes soll weitere Anforderungen abgearbeitet werden.
- Arbeitspaketbeschreibung:
  Die Login-Oberfläche bietet eine Möglichkeit, das Passwort eines Accounts neu anzufordern. Dazu wird eine Eingabemaske aufgerufen, in der die E-Mail-Adresse des Accounts eingegeben werden kann. Existiert ein Account mit dieser Adresse, so wird ein Link an diese E-Mail-Adresse gesendet. Über diesen Link kann der Benutzer sein Passwort ändern. Der Benutzer erhält nach dem Ändern des Passworts eine weitere Mail, die die Änderung bestätigt. Eine E-Mail-Adresse, die nicht der korrekten Form entspricht, wird nicht akzeptiert.
- Vorbedingungen:
  Das Arbeitspaket „Login in die Weboberfläche“ muss abgeschlossen sein.
- Nebenbedingungen:
  Die E-Mails enthalten nicht das neue oder alte Passwort. Die Rückmeldung der Eingabemaske lässt keinen Rückschluss darauf zu, ob tatsächlich ein Account mit dieser Adresse existiert.
- Nachbedingungen:
  Nach Abschluss des Arbeitspakets erhält ein Benutzer innerhalb von zehn Minuten, nachdem er ein neues Passwort angefordert hat, eine E-Mail mit dieser er sein Passwort ändern kann.

- Aufwand:
  Zwei Wochen
- Personen:
  - Timo Schlömer
  - Danny Fonk

**4.3.10.1 Dokumentation** In diesem Arbeitspaket wird es den Medizinern ermöglicht, das Passwort ihres Accounts durch ein neues, selbstgewähltes zu ersetzen.

**Ablauf** Die Arbeit basiert auf den Fortschritten der Arbeitspakete „Struktur der Weboberfläche“ (Kapitel 4.3.3 auf Seite 101) und „Login in die Weboberfläche“ (Kapitel 4.3.4.1 auf Seite 107). Auf der Login-Seite der Weboberfläche wird nun ein weiterer Button angezeigt, mit dem sich ein neuer Dialog öffnen lässt. In diesem Dialog kann der Benutzer den Account-Namen oder die E-Mail Adresse, welche mit dem Account assoziiert ist, angeben und die Passwort-Wiederherstellung starten. Wird der Prozess angestoßen, generiert das System einen Link, der 24 Stunden gültig ist. Dieser bezieht sich auf den angegebenen Account und wird dem Benutzer mittels einer E-Mail, an die für diesen Account hinterlegte E-Mail-Adresse, mitgeteilt. Der Nutzer kann über Aufruf des Links ein neues Passwort vergeben, ist nach Vergabe direkt eingeloggt und wird auf die Startseite weitergeleitet.

**Systembeschreibung** Das Arbeitspaket wurde unter den gleichen Bedingungen entwickelt wie das Arbeitspaket „Struktur der Weboberfläche - PrimeFaces“ 4.3.3.2 auf Seite 105. Seitens der Implementierung wurde sich gegen eine konsistente Haltung der Daten entschieden, da der Mehraufwand der Schreib- und Leseoperationen die Datenbank unnötig belasten würden und die Gültigkeit eines Links ohnehin auf 24 Stunden beschränkt ist. Die Daten werden zwischengespeichert, und gehen bei einem Server Neustart verloren.

**Evaluation** Beim Wiederherstellen eines Passworts wird innerhalb der vorgegebenen zehn Minuten ein Link zur neuen Passwortvergabe generiert und an den Benutzer per E-Mail versendet. Auch kann ein möglicher Angreifer nicht erkennen, ob die eingegebenen Registrierungsdaten tatsächlich existieren, wenn die Passwort-Wiederherstellen Funktion verwendet wird. Zudem enthält die versandte E-Mail weder das alte, noch das neue Passwort.

**Parameter** Die Funktion der Passwort-Wiederherstellen Funktion wird durch keine Parameter eingeschränkt.

**Fazit** Das Arbeitspaket konnte eine Woche vor der Frist mittels PrimeFaces und JavaEE abgeschlossen werden.

### 4.3.11 Integration einer Karte in die Weboberfläche

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket dient dazu, weitere Anforderungen umzusetzen.
- Arbeitspaketbeschreibung:
  Die Weboberfläche zeigt eine Karte an, auf der alle übermittelten Krankheitstests oder nur alle, innerhalb eines bestimmten Zeitraumes übermittelten Tests, dargestellt werden. Ein Mediziner kann zwischen einer Heatmap und einer Anzeige mittels Markern wählen. Die Heatmap zeigt

alle positiven Tests als Farbverlauf auf der Karte an. Die Ansicht mit Markern stellt jeden positiven Test als einen Marker auf der Karte dar, welche in Echtzeit zu Clustern zusammengefasst werden. Um einen Zeitraum auswählen zu können, ist es einem Mediziner möglich, einen Startzeitpunkt und einen Endzeitpunkt festzulegen. Wenn ein Zeitraum ausgewählt ist, kann der Mediziner diesen stunden- oder tageweise auf der Karte animieren lassen, wodurch sich die Darstellung auf der Karte dem jeweiligen Zeitpunkt anpasst. Dem Mediziner wird dabei angezeigt, welcher Zeitpunkt gerade auf der Karte dargestellt wird. Um eine ausreichende Datenbasis für die Anzeige von Tests zu generieren, kann der Testdatengenerator aus dem zweiten Sprint herangezogen werden. Die Karte beinhaltet zusammengefasst:

  - Es ist eine Datenbasis an Tests vorhanden, welche zum Testen der Karte herangezogen werden kann.
  - Eine Ansicht als Heatmap und eine Ansicht, welche Tests durch einen Marker auf der Karte repräsentiert
  - Die Möglichkeit einen Zeitraum einzustellen und die Ausbreitung auf der Heatmap oder auf der Marker-Map animieren zu lassen.
  - Die Karte zeigt bei einer Animation den aktuellen Zeitpunkt an. Bei der Animation ist ein sinnvolles Zeitverhalten für die Ausbreitung von Krankheiten zu wählen.

- Vorbedingungen:
  -

- Nebenbedingungen:
  Die Karte ist nur von Medizinern einsehbar.

- Nachbedingungen:
  Die Karte ist in der Lage in der Heatmap und in der Marker-Darstellung mindestens 50000 Krankheitstests gleichzeitig anzuzeigen. Die Karte soll auf allen, der Projektgruppe zur Verfügung stehenden Computern, bei mindestens 50000 gleichzeitig dargestellten Tests weiterhin flüssig bedienbar sein. Wenn ein Mediziner einen Zeitraum betrachten will, darf dieser keinen ungültigen Zeitraum auswählen können (wie etwa den Endzeitpunkt vor dem Startzeitpunkt auszuwählen).

- Aufwand:
  Drei Wochen

- Personen:

  - Kevin Sandermann
  - Raphael Kappes

**4.3.11.1 Dokumentation** Die Motivation für dieses Arbeitspaket ist es den Prototypen der Karte zu überarbeiten und zu verbessern. Dazu ist in Abbildung 72 auf der nächsten Seite die alte Karte zu sehen.

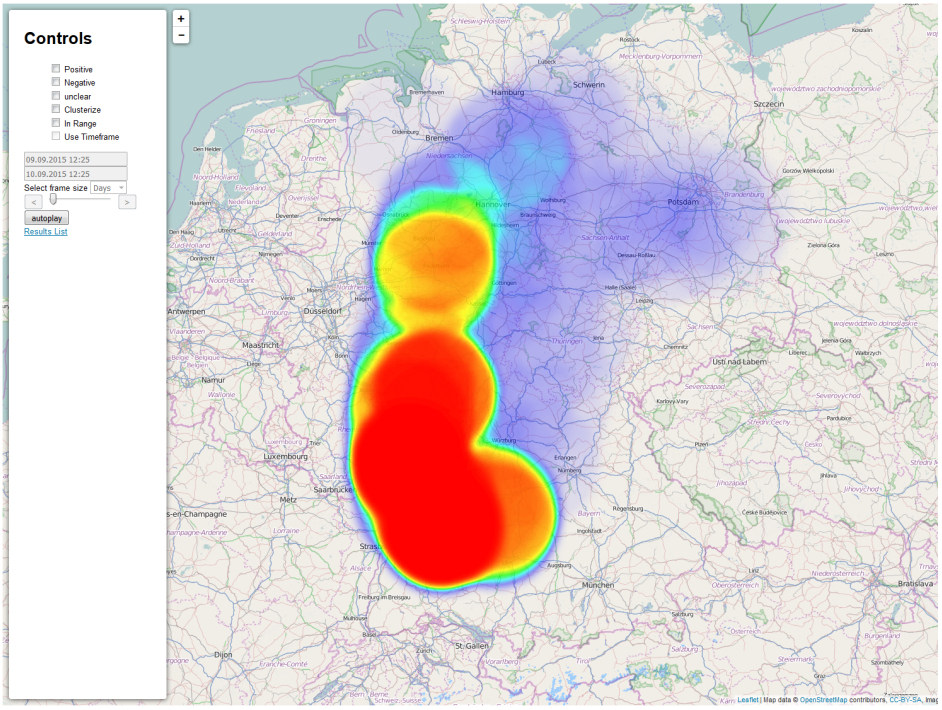



Abbildung 72: Alte Kartenansicht der Heatmap aus vorherigen Sprints

Diese hatte große Probleme mit größeren Datenmengen. Die Benutzeroberfläche war nicht intuitive zu bedienen und hatte unerwartete Kombinationen, die Fehler hervor gerufen hat. Diese Punkte sollen in diesem Arbeitspaket gelöst werden.

**Ablauf** Als Grundlage für dieses Arbeitspaket wurde die aus dem Arbeitspaket `Struktur der Weboberfläche - PrimeFaces` entstandene Weboberfläche in Java PrimeFaces genutzt. Diese wurde dahingehend erweitert, als dass eingeloggten Statistikern und Ärzten eine Kartenansicht über durchgeführte Krankheitstests inklusive deren Ergebnisse zu Verfügung steht. Um diese Ansicht möglichst performant und nutzerfreundlich zu gestalten, wird als Grundlage für die Karte das in PrimeFaces integrierte Google Maps-API genutzt. Diese stellt eine rudimentäre Karte zur Verfügung, über die von der Serverseite aus sogenannte Marker für jeden durchgeführten Krankheitstest gesetzt werden können. Um diese Marker zu setzen, muss aus der Datenbank die gesamte Menge an Krankheitstests gelesen werden. Dieser Vorgang stellt eine intensive Operation auf der Datenbank dar, welche eine Menge Datenverkehr und Last verursacht. Damit diese Operation nicht bei jedem Aufruf der Karte eines Statistikers oder Arztes durchgeführt wird, wurde entschieden im Arbeitsspeicher des PrimeFaces-Servers eine Liste aller Krankheitstests zwischenzuspeichern, welche ohne nennenswerte Performanceverluste gelesen und an den Nutzer gesendet werden kann. Diese im Arbeitsspeicher befindliche Liste aller Krankheitstests wird über einen im Hintergrund ausgeführten Dienst zeitgesteuert alle fünf Minuten aktualisiert, d.h. anstatt bei jedem Seitenaufruf eine große Anfrage an die Datenbank zu schicken, wird bei jedem Seitenaufruf die maximal fünf Minuten alte Liste aller Tests aus dem Arbeitsspeicher gelesen. Der Nachteil, dass neu durchgeführte Tests unter Umständen erst fünf Minuten später auf der Karte angezeigt werden, wurde als hinnehmbar bewertet, da die Kartenansicht als Übersicht aller Krankheitstests im Allgemeinen geplant ist, d.h. ein einzelner Test für sich ist kein kritisches Element.
Um die auf der Datenbank verursachte Last weiter zu minimieren, wurde ein neues Datenmodell auf Serverseite angelegt: Ein sog. `HeatmapRecord` enthält nur die für die Kartenansicht relevanten Informationen, d.h. unnötige Daten wie etwa das Bild werden nicht aus der Datenbank gelesen. Um das Datenmodell mit den passenden Daten zu befüllen, wird die Methode der Projektion genutzt. Um eine spätere Iteration und Filterung der geladenen Datensätze nach Zeitrahmen zu opti-

mieren, wird außerdem bereits auf Datenbankebene eine Sortierung aller Krankheitstestdaten nach Durchführungszeitpunkt vorgenommen.

Wie in der Beschreibung dieses Arbeitspakets beschrieben, lassen sich die durchgeführten Krankheitstests in zwei vom Nutzer frei auswählbaren Ansichten darstellen: Die Ansicht einer Heatmap zeigt alle positiven Krankheitstests, d.h. alle als krank eingeordneten Probanden, in Form einer Heatmap über der Google Maps-Karte an. Diese Heatmap lässt sich vom Nutzer über eine einfache Schaltfläche aktivieren und deaktivieren.

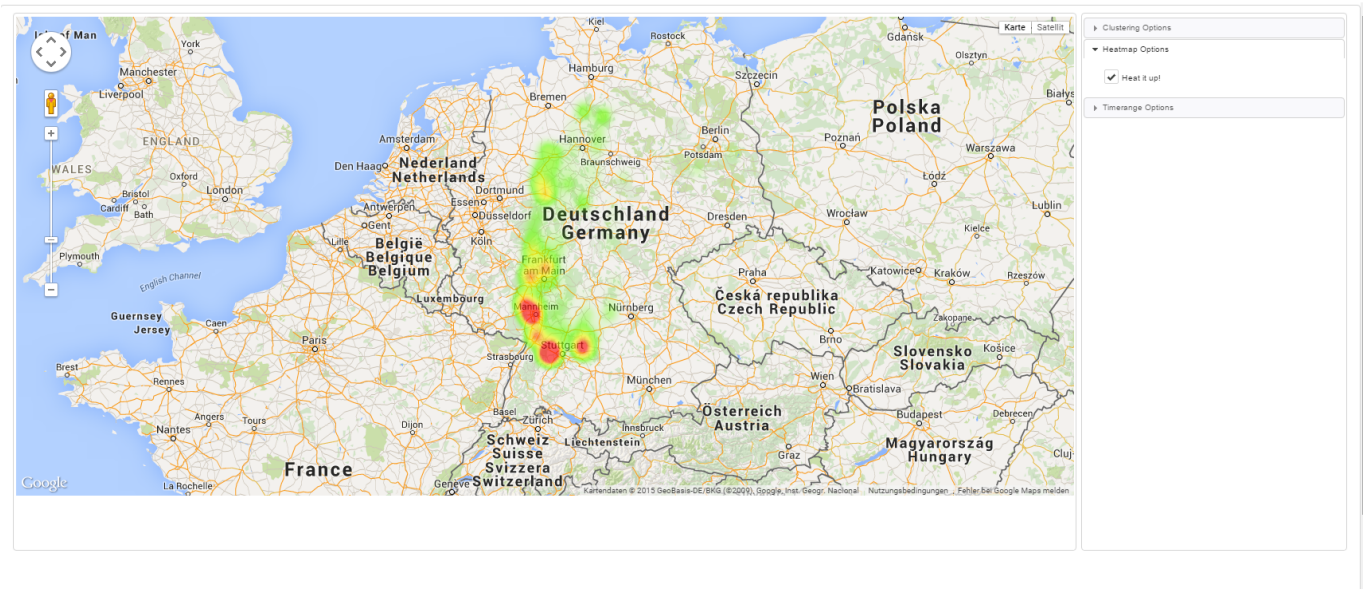



Abbildung 73: Kartenansicht der Weboberfläche mit eingeschalteter Heatmap-Ansicht

Abbildung 73 zeigt diese Heatmap-Ansicht: Auf der linken Seite ist Deutschland als Karte mit positiven Krankheitstests dargestellt, auf der rechten Seite ist die Schaltfläche zum Aktivieren und Deaktivieren dieser Ansicht zu sehen. Um diese Ansicht performant darzustellen, werden die beim Seitenaufruf aus dem Arbeitsspeicher des PrimeFaces-Servers genutzt, welche bereits im Browser des Nutzers vorliegen. Dies hat den Vorteil, dass ein erneutes Nachladen von Daten während der Nutzung der Heatmap Ansicht zu keiner Zeit notwendig wird, sodass eine komplett flüssige Navigation durch die Heatmap ermöglicht wird.

Als zweite Ansicht neben der erwähnten Heatmap ist die sogenannte Cluster-Ansicht aller Krankheitstests implementiert. Diese stellt eine individuell anpassbare Auswahl an Krankheitstests (positive, negative, unbekannte oder Kombinationen hieraus) in sogenannten Clustern dar. Ein Cluster ist ein Verbund mehrerer Krankheitstests, welche in einer bestimmten geographischen Region durchgeführt wurden. Zoomt der Nutzer in der Karte näher in eine bestimmte Region, werden die dort vorliegenden Cluster weiter in kleinere Cluster aufgeteilt, bis schließlich auf maximaler Zoomstufe die einzelnen Krankheitstests als Marker vorliegen. Diese Marker stellen durch Ihre Farbe das Ergebnis des repräsentierten Krankheitstests dar (rot - positiv, grün - negativ, gelb - unbekannt). Fährt der Nutzer mit der Maus über einen Marker, wird ihm der Zeitpunkt der Testdurchführung eben dieses Tests angezeigt.

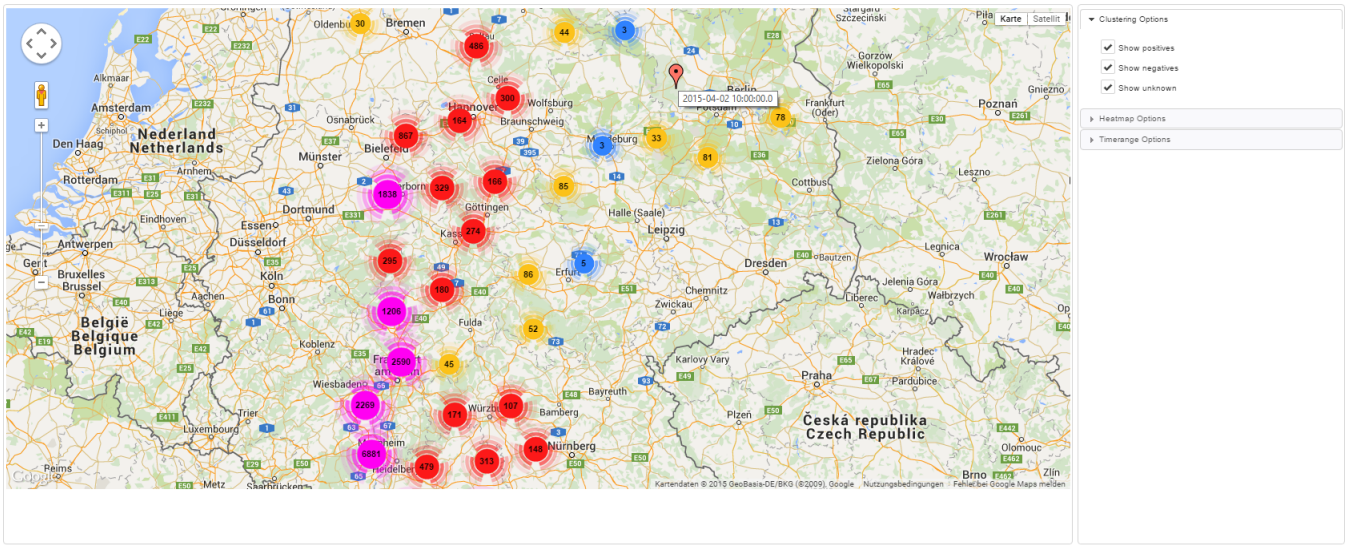



Abbildung 74: Kartenansicht der Weboberfläche mit eingeschalteter Cluster-Ansicht

Abbildung 74 zeigt die erwähnte Cluster-Ansicht der Karte: Die einzelnen Cluster repräsentieren durch ihre Größe und Farbe die Anzahl der durchgeführten Tests. Ebenfalls zu sehen ist ein einzelner positiver Test als roter Marker auf der Karte, welcher durch den auf ihm liegenden Mauszeiger den zugehörigen Testzeitpunkt anzeigt. Auf der rechten Seite sind die Schaltflächen für die jeweiligen Cluster-Möglichkeiten zu sehen. Um diese Ansicht auch bei einer großen Anzahl an Tests flüssig darzustellen, werden die Tests beim Ändern einer dieser Schaltflächen innerhalb des Browsers des Nutzers gefiltert und die passenden Cluster einmalig berechnet. Diese Berechnung und Filterung nimmt bei einem durchschnittlichen Computer bei 50000 vorliegenden Datensätzen etwa zwei Sekunden in Anspruch. Diese Ladezeit gilt als hinnehmbar, da sie einmalig bei der Initialisierung aller Cluster aufkommt. Sobald die Cluster berechnet sind, ist die Kartenansicht komplett flüssig zu bedienen, da keinerlei Berechnungen mehr durchgeführt werden müssen und keine Daten zur Laufzeit nachgeladen werden.

Als dritte Filtermöglichkeit der Karte ist wie in der Arbeitspaketbeschreibung dargestellt eine Filterung nach Zeitrahmen implementiert worden. Diese Zeitfilterung ist grundlegender Bestandteil, um den Datenverkehr zwischen Server und Browser gering zu halten: Durch die Auswahl eines Zeitrahmens, sind für den Nutzer nur die Testdaten relevant, welche innerhalb eben dieses Zeitrahmens liegen. Aufgrund dessen werden bei Auswahl eines Zeitrahmens nur die in diesem Zeitrahmen liegenden Tests vom Server an den Browser gesendet. Um den standardmäßig auftretenden Datenverkehr bei Aufrufen der Karte möglichst gering zu halten, wurde entschieden dass initial ein Zeitrahmen vom aktuellen Datum inklusive der letzten sieben Tage ausgewählt ist. Sobald vom Nutzer ein anderer Zeitrahmen ausgewählt wird, werden die entsprechenden Daten nachgeladen. Ähnlich wie bei der Cluster-Ansicht wird hier ebenfalls initial bei der Auswahl eines Zeitrahmens sämtliche Rechenarbeit durchgeführt, welche bei den 50000 vorliegenden Testdaten ebenfalls etwa zwei Sekunden in Anspruch nimmt. Anschließend werden keinerlei Ladevorgänge mehr benötigt, sodass wiederum komplett flüssig durch die Karte navigiert und gezoomt werden kann. Diese Ladezeit von zwei Sekunden ist dahingehend bereits optimiert, als dass die Iteration über alle vorliegenden Datensätze abgebrochen werden kann, sobald ein erster passender / nicht mehr passenden Datensatz gefunden wurde: Wie oben erwähnt sind die Datensätze auf Datenbankebene nach Datum sortiert worden, d.h. sobald ein erster, nicht mehr dem Zeitrahmen entsprechender Datensatz gefunden wird, sind alle folgenden Datensätze automatisch ebenfalls nicht dem Zeitrahmen entsprechend. Diese Mechanik verkürzt die Ladezeit an dieser Stelle teils um über 50%.

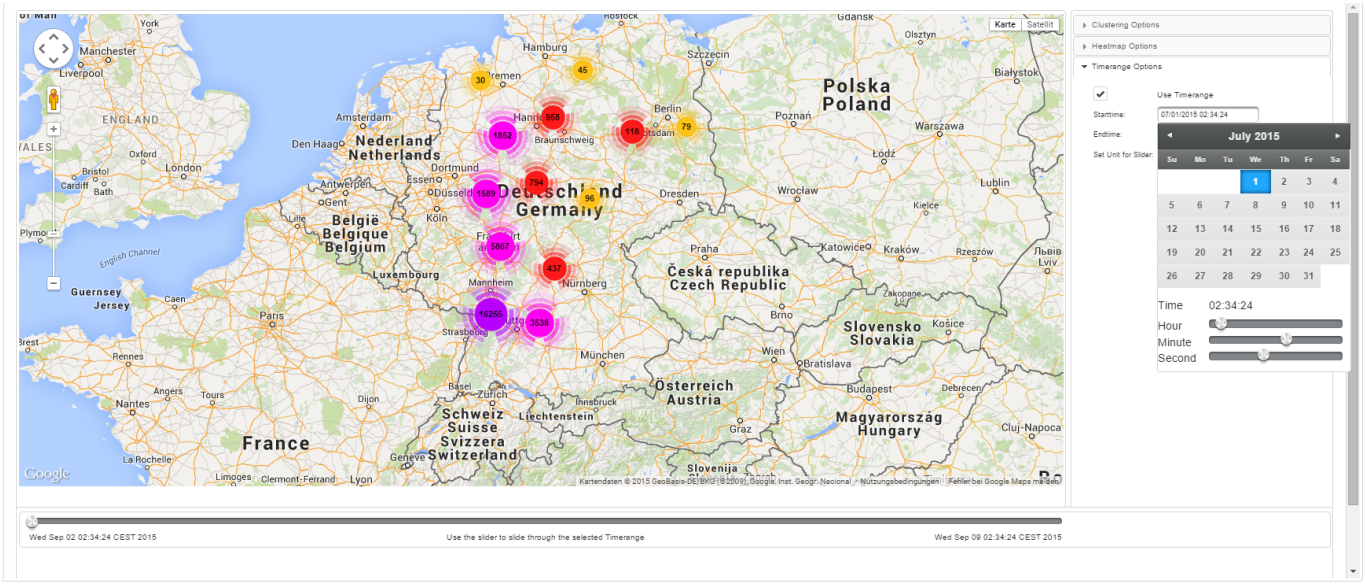



Abbildung 75: Kartenansicht der Weboberfläche: Auswahl eines individuell festlegbaren Zeitrahmens

Abbildung 75 zeigt die Auswahl eines Zeitrahmens durch den Nutzer: Durch die dargestellte Kalender-Ansicht lässt sich intuitiv ein Datum inklusive Zeitpunkt auswählen. Das ausgewählte Datum wird anschließend als gängiges Textformat dargestellt, sodass der Nutzer zu jedem Zeitpunk Einsicht in den ausgewählten Zeitrahmen hat. Die Zeitrahmen-Funktion lässt sich sowohl mit der Heatmap-Ansicht als auch mit der Cluster-Ansicht kombinieren. Ebenfalls ist es dem Nutzer möglich, die Zeitrahmen-Funktionalität komplett zu deaktivieren, um die Gesamtheit aller Krankheitstests darzustellen.

Als letzte implementierte Funktionalität ist der sog. Slider umgesetzt worden. Wie bereits in Abbildung 75 zu sehen, wird der Slider eingeblendet, sobald ein Zeitrahmen definiert wurde. Ist die Zeitrahmen-Funktionalität deaktiviert, wird der Slider automatisch ausgeblendet. Durch den Slider ist es dem Nutzer möglich, durch die einzelnen Zeitpunkte innerhalb des Zeitrahmens schrittweise zu iterieren. Hierbei kann er per Schaltfläche auswählen, ob ein Schritt des Sliders einer Zeitstunde oder einem Tag entspricht. Initial ist der Slider für die Anzeige der Karte nicht relevant, d.h. standardmäßig ist kein spezieller Wert durch den Slider ausgewählt. Sobald allerdings durch den Slider ein gewisser Zeitpunkt innerhalb des Zeitrahmens ausgewählt wird, werden sowohl die Cluster-Ansicht als auch die Heatmap-Ansicht der Karte insofern gefiltert, als dass nur noch die Tests zwischen dem Startzeitpunkt des ausgewählten Zeitrahmens und des durch den Slider definierten Zeitpunkts dargestellt werden.

### Technologien

**Google Maps** Die eigentliche Karte wurde mit dem von PrimeFaces entwickeltem GMap-Objekt realisiert, dass die Google-API nutzt. Weiterhin wurde aus der Google Maps API die Heatmap-Funktion mit benutzt.

**Systembeschreibung** Die Karte wurde in einer Eclipse Mars Java EE Umgebung entwickelt.

**Evaluation** Die Karte wurde mit 50000 Testdaten, die vom Testdatengenerator generiert wurden, getestet. Dabei ist die Ansicht der Heatmap-Funktion zwar langsam beim vorladen, jedoch sobald diese Daten gezeichent wurden sind die Datenmengen nicht mehr bemerkbar. Ebenso ist die Cluster-Funktion schnell in der Anzeige von zusammengefassten Daten. Das Durchlaufen wird mit

einer Slider-Funktion realisiert, die entweder Stunden oder Tage weise durchlaufen werden kann. Dabei verändert sich die Ansicht der dargestellten Daten auf der Karte. Bei mehreren 100000 Tests kann es passieren, dass Windows ein Problem beim Browser vermutet und diesen Schließen möchte.

**Parameter** Die Daten sind, wenn sie zu einer bestimmten Funktion geladen wurden, nicht bemerkbar auf der Karte.

**Fazit** Das Arbeitspaket könnte mit den Nebenbedingungen erfolgreich abgeschlossen werden. Dabei kann es passieren das der Browser, wenn ein Zeitraum, in dem einige Tests durchgeführt wurden, zu groß gewählt wird (wenn mehrere 100000 Tests ausgewählt werden), durch Windows zum beendetet werden soll, da es ein Problem vermutet.

**Primefaces** Dieses Framework bietet keine direkte Unterstützung für das Manipulieren der Google-Maps Karte. Dadurch ist die Implementierung nicht wie bei den anderen Funktionalitäten, einfach in Primefaces zu realisieren, sondern durch zusätzliches JavaScript zu lösen gewesen. Das führte dazu, dass mehr Zeit in die Funktionen, die in dem Arbeitspaket beschrieben wurden, hineingesteckt werden musste.

**Ausblick** Dieses Problem kann später mit dynamischen Nachladen, fallen lassen oder zusammen Fassen der Daten gelöst werden. Für dieses Arbeitspaket ist diese Optimierung jedoch zu aufwendig.

#### 4.3.12 Erstellung eines Teststreifengenerator

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket dient dazu, Teststreifen mit eigenen Parametern für QR-Code und Farbumschlagfelder generieren zu können.
- Arbeitspaketbeschreibung:
  Es existiert ein Programm, das automatisch Beispiel-Teststreifen generieren kann. Auf diesen Beispiel-Teststreifen sind zwei Felder vorhanden. Ein Feld dient als Referenzfeld, um so eine Bildoptimierung zu ermöglichen. Das zweite Feld dient dazu, den Farbumschlag zu repräsentieren, der durch das Applizieren einer Flüssigkeit entsteht. Die Fläche des Farbumschlagsfeldes ist nicht größer als 1x1 Millimeter. Der Beispiel-Teststreifen enthält einen QR-Code, welcher für dieses Arbeitspaket als Platzhalter dient und dessen Inhalt zu einem späteren Zeitpunkt endgültig definiert wird. Der Teststreifengenerator kann zudem die Farbintensität des Farbumschlagfeldes generieren, indem der Benutzer ein Ziel für den Farbwert des (gesamten) Referenzfeldes vorgibt. Der Teststreifengenerator kann mehrere Beispiel-Teststreifen zu einem solchen Zielwert erstellen, wobei die Form und der Farbverlauf zufällig variiert wird. Die Farbwerte des Referenzfelds und des Farbumschlagfeldes sind auf dem Teststreifen abgebildet. Dies dient dazu, die Ergebnisse des Bilderkennungsalgorithmus mit den realen Werten des Teststreifens schnell abgleichen zu können. Für einen möglichen Aufbau des Teststreifens vgl. Abbildung 76 auf Seite 233. Der Teststreifengenerator speichert die generierten Beispiel-Teststreifen als Bilder in einem verlustfreien Dateiformat auf dem Computer ab. Die Dateinamen der generierten Bilder enthalten die Farbwerte des Farbumschlags, damit später schnell Beispiel-Teststreifen mit einem gewünschten Farbwert gefunden werden können.
- Vorbedingungen:
  -

- Nebenbedingungen:
  Optional: Das Programm verfügt über eine grafische Benutzeroberfläche über die die Parameter des Teststreifens eingestellt werden können.

- Nachbedingungen:
  Ein nicht an der Entwicklung des Teststreifengenerators beteiligtes Teammitglied kann mit Hilfe des Generators innerhalb von zehn Minuten 100 neue Beispiel-Teststreifen generieren. Es liegen außerdem mindestens 50 Beispiel-Teststreifen auf Papier gedruckt vor, welche für spätere Arbeitspakete verwendet werden können.

- Aufwand:
  Zwei Wochen

- Personen:
  - Timo Schlömer
  - Nicolas Koch
  - Danny Fonk

**4.3.12.1 Dokumentation** Der Teststreifengenerator setzt keine konkrete funktionale Anforderung um, sondern dient als Werkzeug, um Teststreifen zu erzeugen und auszudrucken. Mit Hilfe dieser kann die Entwicklung des Bildverarbeitungsalgorithmus insofern unterstützt werden, als dass der Algorithmus auf Basis der ausgedruckten Teststreifen evaluiert werden kann.

**Ablauf** Zur Umsetzung des vorliegenden Arbeitspaketes wurde ein bereits entwickelter Prototyp eines Teststreifengenerators herangezogen. Dieser wurde im Rahmen des Arbeitspaketes „Bildverarbeitung“ aus Sprint 3 (siehe Kapitel 4.2.8.1 auf Seite 81) von Christoph Ressel entworfen. Während sich die grafische Oberfläche des neuen Teststreifengenerators an die des Alten anlehnt, wurde die Funktionalität grundlegend geändert. Es wurden dazu folgende Parameter ausgewählt, die das Aussehen eines Teststreifens definieren:

**Intensity** Die Stärke des Farbumschlags auf dem Farbumschlagsfeld.

**Color** Die Farbe des Farbumschlags und des Referenzfeldes.

**Gradient** Der Farbverlauf des Farbumschlags. Hier gibt es die zwei Varianten *radialer* und *linearer* Farbverlauf.

**QR-Code Text** Der im QR-Code enthaltene Text.

**DPI** Die unterstützten Dots per Inch (DPI) des Druckers.

Abbildung 77 auf der nächsten Seite zeigt den Einfluss der Parameter auf die generierten Teststreifen.

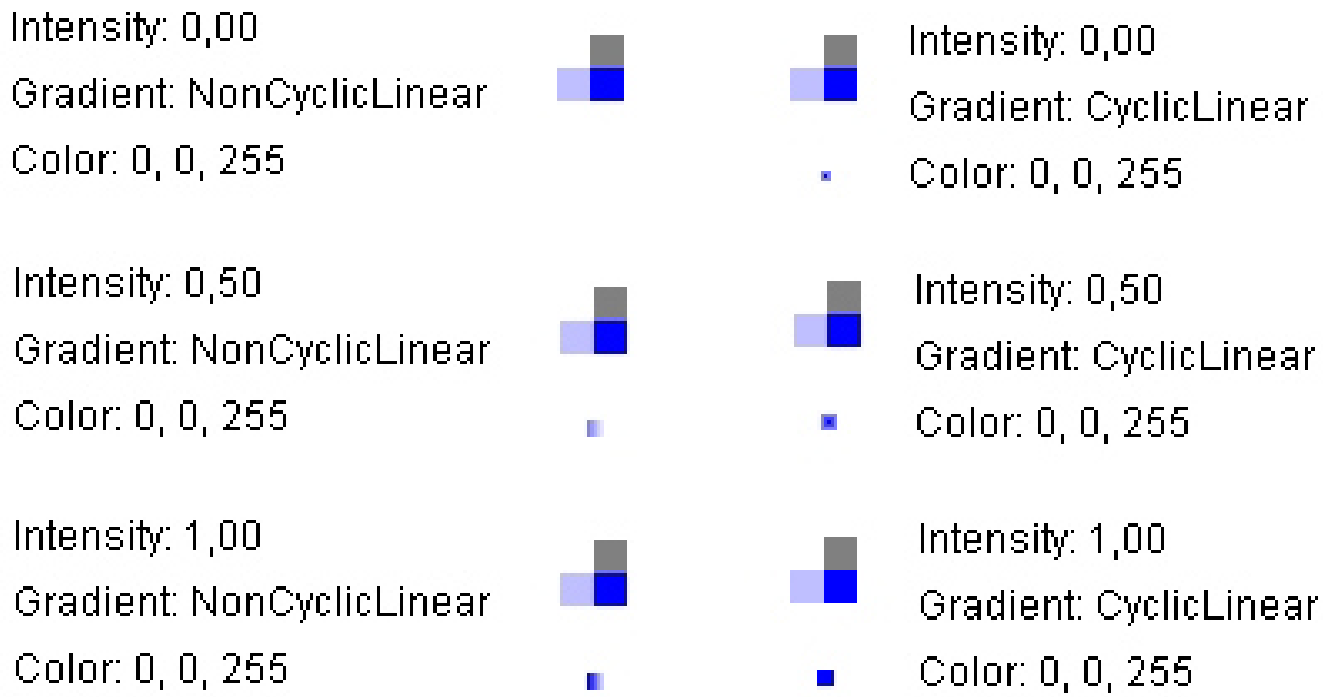



Abbildung 77: Einfluss der verschiedenen Parameter auf den Farbumschlag

Zunächst wurde die Generierung eines einzelnen Teststreifens mit Hilfe einer grafischen Benutzeroberfläche umgesetzt. Hierbei können die oben genannten Parameter spezifiziert werden. Das Programm zeigt die Änderungen der Parameter hierbei direkt in der Benutzeroberfläche an.
Zur Generierung mehrerer Teststreifen auf einmal wurde vorerst eine Kommandozeilenschnittstelle für die Applikation entworfen. Dazu kann der Benutzer die Anzahl der zu erzeugenden Teststreifen und den Speicherort dieser festlegen. Die Schnittstelle wird in Listing 2 auf Seite 132 detailliert beschrieben.
In dem Teammeeting vom 05.08.2015 ist nach einer Feedbackrunde entschieden worden, dass auch für die Generierung mehrerer Teststreifen eine grafische Oberfläche erwünscht ist. Diese wurde in der darauffolgenden Woche zusätzlich implementiert.
Zusätzlich wurde das Referenzfeld aufgrund des Gruppenfeedbacks angepasst. Dieses zeigt nun die Farbwerte $(255, 255, 255, 0), (128, 128, 128, 0), (Color, 0.75), (Color, 0)$ an. Besonders der Grauwert ist für einen späteren Farbabgleich des Bildalgorithmus sehr hilfreich.
Eine weitere Herausforderung bei der Umsetzung war es sicherzustellen, dass ausgedruckte Teststreifen immer eine feste Größe von $13cm \times 2.5cm$ haben müssen. Auch der Farbumschlag muss ausgedruckt stets $1mm^2$ groß sein. Die dafür benötigte Anzahl der Pixel ist abhängig von dem DPI-Wert des Druckers, und muss daher im Teststreifengenerator dynamisch verändert werden können. Dies ist durch ein Auswahlfeld in der Benutzeroberfläche realisiert worden. Zum Abschluss des Arbeitspaketes wurden 50 zufällige Teststreifen ausgedruckt, welche die Entwicklung eines Bildverarbeitungsalgorithmus unterstützen sollen.

**Technologien**

**Java Swing** Bibliothek zur Erstellung von Benutzeroberflächen in Java.

**Apache Commons Imaging** Java Bibliothek, dass der Bildverarbeitung dient.

**ZXing** Bibliothek zur Generierung von Barcodes.

**Systembeschreibung** Abbildung 78 zeigt die grafische Benutzeroberfläche des Teststreifengenerators.

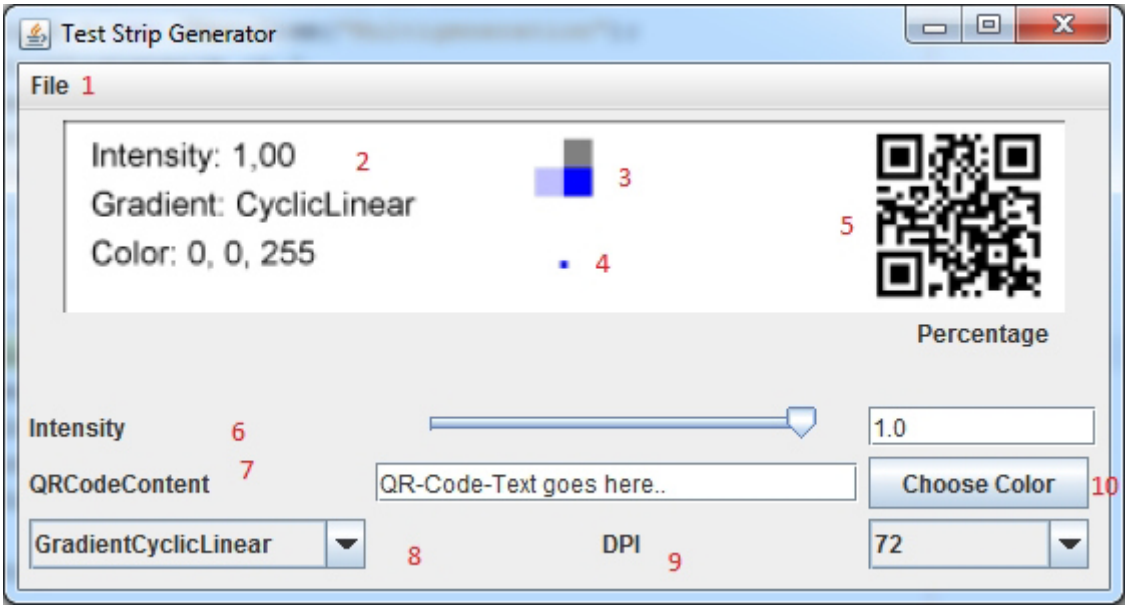



Abbildung 78: Grafische Benutzeroberfläche des Teststreifengenerators

Die einzelnen Elemente (siehe Nummern auf der Abbildung) setzen sich wie folgt zusammen:

1. Menü für Mehrfachgenerierung, Speichern, Hilfe, Drucken und Beenden
2. Anzeige der aktiven Parameter
3. Referenzfeld zur Bildoptimierung
4. Farbumschlagfeld
5. QR-Code
6. Schieberegler und Textfeld zur Einstellung der Intensität
7. Texteingabefeld zur Generierung des QR-Codes
8. Auswahlfeld für den Farbverlauf
9. Auswahlfeld für die *dots per inch* des Druckers.
10. Button zur Auswahl der Farbumschlagfarbe

Abbildung 79 auf der nächsten Seite zeigt die grafische Benutzeroberfläche für die zufällige Generierung mehrerer Teststreifen.

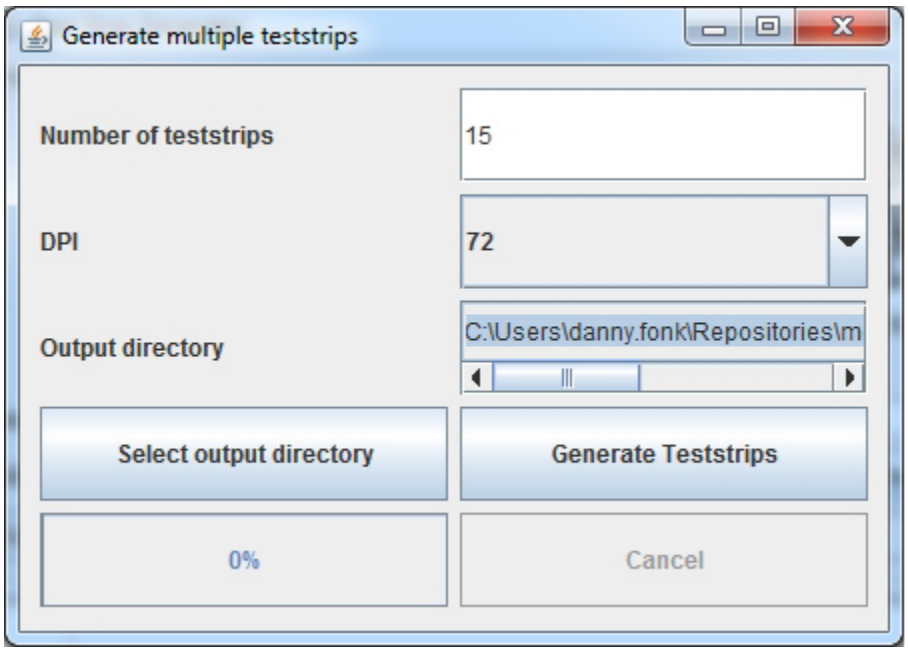



Abbildung 79: Grafische Benutzeroberfläche der Mehrfachgenerierung

Die Parameter können in dieser Ansicht nicht spezifiziert werden, sondern werden von dem Programm zufällig gewählt.
Es existiert außerdem die Kommandozeilenschnittstelle. Das Codebeispiel 2 zeigt die Ausgabe des Parameters `-h`.

```
usage: java -jar tsg.jar [additional arguments]

Generates sample images of test strips with various parameters.
If no arguments are given, a single image with random attributes will be
stored in the programs folder.

-c,--color <arg>                    The color of the generated teststrips in
                                     the form 0-255,0-255,0-255. Will be
                                     randomized if not set.
-d,--dpi <arg>                      Dots per Inch. Sets the DPI-Value of the
                                       outcoming picture('s).
-gui                                Start with the GUI enabled.
-h,--help                           Print this message.
-i,--intensity <arg>                The average intesity of filled pixels in
                                    the changeover. Will be randomized if not
                                    set.
-n,--number-of-images <arg>         The number of teststrips to generate.
-o,--output <arg>                   The folder to save generated images to.
```

Listing 2: Ausgabe des Kommandozeileninterfaces des Teststreifengenerators

**Evaluation** Die Qualität des Produktes wurde durch kontinuierliches Testen während des Entwicklungsprozesses sichergestellt. Des Weiteren wurde ein Unittest erstellt welcher sicherstellt, dass die Generierung zweier Teststreifen mit denselben Parametern stets pixelidentische Bilder generiert. Außerdem wurde nach einer Woche Entwicklungszeit ein Feedbackgespräch mit der Gruppe geführt. Die Anmerkungen daraus wurden zusätzlich in das Programm eingearbeitet.

**Parameter**

**Anzahl der generierten Teststreifen** Es können 1000 Teststreifen auf einmal innerhalb von 10 Sekunden erzeugt werden. Die Laufzeit steigt linear mit der Anzahl der erzeugten Teststreifen.

**Print-Dialog** Der Dialog zum Drucken wird unter bestimmten exotischen Linux-Distributionen nicht angezeigt.

**Fazit** Im Rahmen der Implementierung konnten alle Anforderungen aus der Arbeitspaketbeschreibung, sowie die zusätzlichen Anforderungen aus dem Teammeeting vom 05.08.2015 umgesetzt werden. Die Deadline von zwei Wochen konnte gehalten werden.

**Ausblick** Mit Hilfe des in diesem Arbeitspaket erzeugtem Teststreifengenerators wird in zukünftigen Arbeitspaketen die Evaluation des Bildverarbeitungsalgorithmus unterstützt. Die generierten Teststreifen stellen eine relativ realistische Darstellung von echten Teststreifen dar und dienen hierdurch auch dazu, dass die Teammitglieder sich eine erste Vorstellung über die Komplexität des Bildverarbeitungsalgorithmus machen können.
Es ist denkbar, dass gerade für Low-End Smartphones der Abstand zwischen Farbumschlagfeld und QR-Code zu groß gewählt wurde und die Kamera daher nicht mehr in der Lage ist den Farbumschlag zu erkennen. In einer kurzen Untersuchung wurde festgestellt, dass die für die zur Verfügung stehenden Testgeräte der Projektgruppe nicht der Fall ist, die aber bei noch schlechteren Kameras aber durchaus ein Problem darstellen könnte. Der simpelste Lösungsansatz ist hier, den Abstand zwischen QR-Code und Farbumschlagfeld zu reduzieren.

### 4.3.13 Evaluation von Bildverarbeitungstools

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Um entwickelte Bildverarbeitungsalgorithmen effizient testen zu können, sollen bestehende Bildverarbeitungstools identifiziert und bewertet werden.
- Arbeitspaketbeschreibung:
  Es werden vorhandene Bildverarbeitungstools hinsichtlich ihrer Nutzbarkeit für die Projektgruppe untersucht. Dazu wird ein mit einer Kamera aufgenommenes Bild in dem Tool geöffnet. Es ist möglich einen definierten Bildbereich auszuwählen und einen Weißabgleich und Intensitätsdetektion durchzuführen. Diese Werte können angezeigt werden und mit dem Original des Bildes verglichen werden. Das Bild ist nach dem Weißabgleich und nach der Intensitätsdetektion wieder näherungsweise dem original entsprechend. Als Beispiel kann ein Teststreifen mit festen RGB-Werten für das Referenzfeld und das Farbumschlagfeld erstellt, abfotografiert und das abfotografierte Foto analysiert werden. Das Ziel ist es nicht, die Funktionen in Form von Algorithmen zu implementieren.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Das gesamte Team ist über die Evaluation der Bildverarbeitungstools informiert und die Evaluationsergebnisse wurden vorgestellt.

- Aufwand:
  Eine Wochen
- Personen:
  - Sebastian Horwege
  - Christoph Ressel

**4.3.13.1 Dokumentation** In diesem Arbeitspaket wurden verschiedene Tools zur Manipulation und Analyse von Bilddateien auf Nützlichkeit für das Projekt untersucht. Dabei sollten keine Algorithmen entwickelt, sondern lediglich der Funktionsumfang der einzelnen Tools untersucht werden. Im Konkreten ging es darum, den Weißabgleich des Bildes zu manipulieren und die Farbwerte einer vorher definierten Region zu ermitteln. Dabei sind folgende Anforderungen relevant:

**FA-4** Die mobile Applikation verfügt über einen Bildverarbeitungsalgorithmus, mit dem eine vorläufige Auswertung des Teststreifens vorgenommen werden kann.

**NFA-11** Das Ergebnis der Voranalyse muss dem Probanden nach maximal zehn Sekunden mitgeteilt werden.

**Ablauf** Zunächst ging es darum, sich einen Überblick über die große Auswahl an Tools zur Bildbearbeitung zu verschaffen. Dabei reicht die Bandbreite an verfügbaren Tools von Endbenutzeranwendungen, über Tools für wissenschaftliche Arbeiten bis zu komplexen Programmiersprachen. Aus den zur Verfügung stehenden Tools haben wir einige ausgewählt, sodass die gesamte eben beschriebene Bandbreite abgedeckt wird:

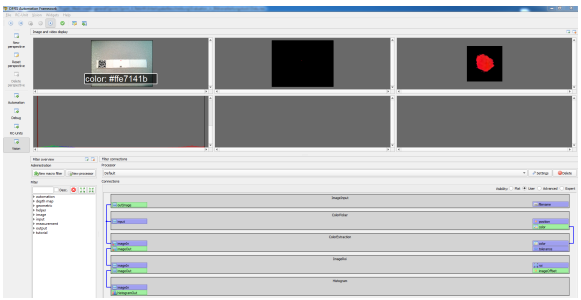
**Technologien**

**FiJi** Einfach zu bedienene Oberfläche für die Bildverarbeitungstoolbox imageJ [8][9].

**Matlab / Octave** Software zur Lösung mathematischer Probleme und der Darstellung von Ergebnissen [10][11].

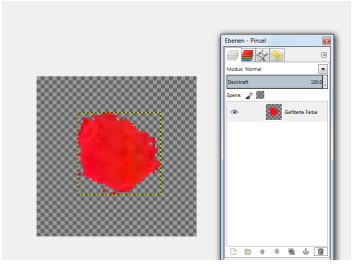
**GIMP**[15] Bildverarbeitungsanwendung, eher für den Privatgebrauch, als für wissenschaftliches Arbeiten.

**OFFIS Automation Framework** Bildverarbeitung und Bildanalyse mit OpenCV ohne zu programmieren.

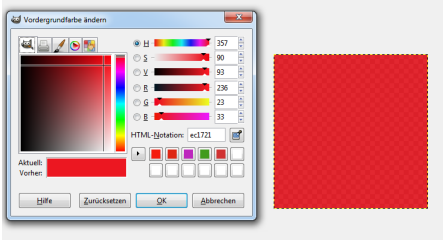


(a) Logische Kombination von OpenCV-Funktionen im OFFIS Automation Framework

(b) Gefilterter und auf Farbumschlag zugeschnittener Bilbereich



(c) Nach Eliminierung der schwarzen Pixel durch GIMP



(d) Nach Skalierung des Bildes auf 1x1 Pixel kann der Farbwert abgefragt werden

Abbildung 80: Office-Automation-Framwork

### Systembeschreibung

**FiJi** FiJi ist eine Zusammenstellung verschiedener Plugins und Bibliotheken um die Bildverarbeitungsumgebung imageJ[16]. Durch FiJi wird der Funktionsumfang von imageJ stark erweitert. Es gibt zahlreiche Möglichkeiten zur Bildverarbeitung und Bildanalyse. In Abbildung 81 auf der nächsten Seite ist eine Profilanalyse eines manuell selektierten Bereichs des Beispielbildes aus Abbildung 82 auf Seite 137 zu sehen. Durch die graphische Anzeige ist leicht zu erkennen, wie der relevante Bildbereich vom Hintergrund zu trennen wäre.

[16] Auswahl der Features von imageJ `http://imagej.nih.gov/ij/features.html`

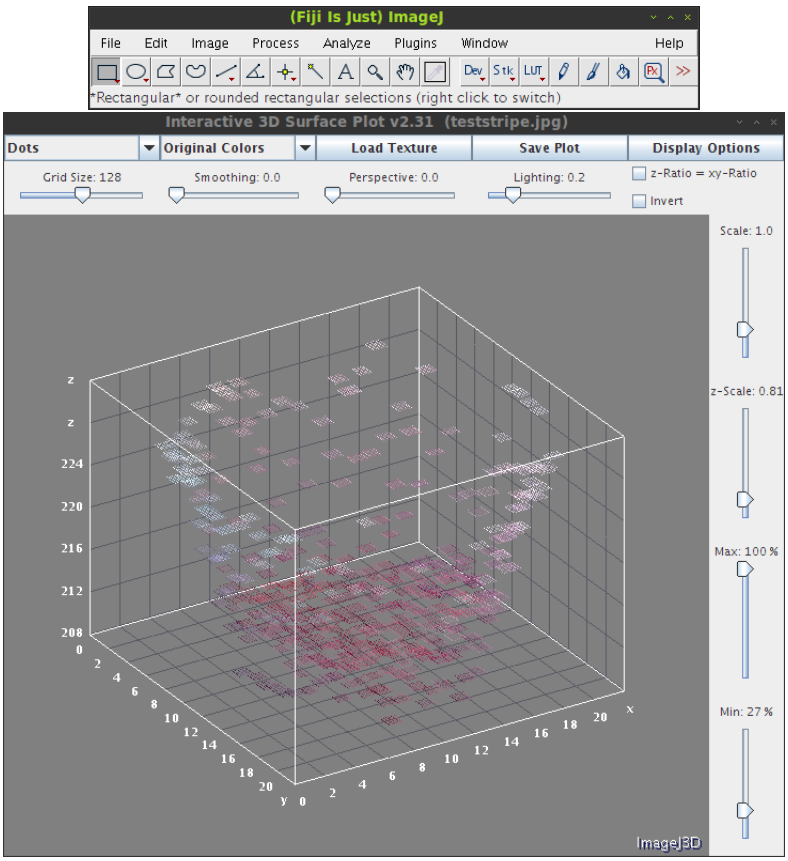



Abbildung 81: FiJi: Menu und Profilanalyse des manuell ausgewählten Bereichs.

**Matlab / Octave** Matlab ist eine Programmierumgebung zur Lösung mathematischer Probleme.

Eigentlich sind digitale Bild bloß zweidimensionale Matrizen, die mit Farbwerten gefüllt sind. Matlab, bei dem prinzipiell alles eine Matrix ist, sollte also bestens geeignet für Bildverarbeitungsalgorithmen sein. Octave wurde als open-source Alternative zu Matlab entwickelt, mit dem Ziel den gleichen Funktionsumfang zu bieten und zu Matlabcode kompatibel zu sein.

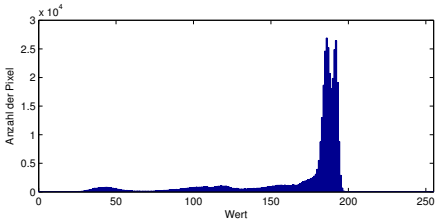
**GIMP** Das Bildverarbeitungstool GIMP richtet sich hauptsächlich an Endnutzer. Es werden verschiedene Werkzeuge zur Manipulation und Anzeige von Bilddateien bereitgestellt. Der Funktionsumfang und Anwendungsbereich ist vergleichbar mit dem bekannteren Adobe Photoshop.

**OFFIS Automation Framework** Das OFFIS Automation Framework [12] benutzt die Bildverarbeitungsverfahren von OpenCV und ermöglicht, mithilfe einer Benutzeroberfläche, verschiedene Filter und Verarbeitungsverfahren auf Bilder anzuwenden und anzuzeigen.

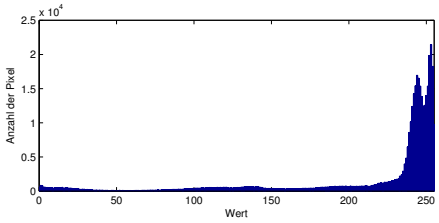
(a) Teststreifen vor Weißabgleich

(b) Teststreifen nach Weißabgleich mit GIMP

Abbildung 82: Teststreifen aufgenommen mit Smartphonekamera vor und nach dem manuellen Weißabgleich mit GIMP




(a) Histogramm vor Weißabgleich




(b) Histogramm nach Weißabgleich

Abbildung 83: Histogrammwerte vor und nach dem Weißabgleich des Fotos aus Abbildung 82

.

### Evaluation

**FiJi** FiJi bietet viele Features, die anfänglich etwas unübersichtlich sortiert und deshalb schwer zu finden waren, nach einer kurzen Einarbeitungszeit und mithilfe der online Dokumentation war das aber kein Problem mehr. Es gibt verschiedene Möglichkeiten, besimmte Features von Bildbereichen in sehr anschaulichen Darstellungen zu visualisieren (siehe Abbildung 136). Außerdem bietet FiJi umfangreiche Exportiermöglichkeiten. So können zum Beispiel sämtliche Daten von erzeugten Diagrammen als Comma Separated Values (CSV) Dateien exportiert werden. Das vereinfacht die weitere Auswertung mit anderen Tools.

**Matlab / Octave** In Matlab musste das Bild zunächst mithilfe der Funktion `imread()` gelesen werden. Danach konnten verschiedene Matrixoperationen auf dem Bild ausgeführt werden. Für den Weißabgleich wurde die auf **mathworks** verfügbare **Auto White Balance Correction** Funktion verwendet[17] Außerdem wurde der Retinex-Algorithmus [18] in einer leicht angepassten Variante implementiert[19]: Statt das globale Maximum für alle Farbkanäle zu suchen, kann eine Region definiert werden, für die dann angenommen wird, dass sie weiß ist.

**GIMP und OFFIS Automation Framework** In einem ersten Iterationsschritt wird zunächst ganz grundlegende Funktionalität getestet und in einer logischen Reihenfolge angewandt, um

[17] `http://www.mathworks.com/matlabcentral/fileexchange/51087-auto-white-balance-correction`

[18] Retinex-Algorithmus Implementierung nach: Kapitel 3.4.2 `http://www.itwm.fraunhofer.de/fileadmin/ITWM-Media/Abteilungen/BV/Pdf/Diplomarbeit_Kenfack.pdf`

[19] Implementierung ist als Snippet im gitlab verfügbar: `https://gitlab.uni-oldenburg.de/medic/medic-general/snippets/5`

ein Ergebnis zu errechnen. Zunächst wird mit Hilfe von GIMP ein Weißabgleich durchgeführt. Hierzu stehen bei GIMP verschiedene Möglichkeiten zu Verfügung, wir nutzen den Weißabgleich per Selektion eines Weiß- und eines Schwarzpixels. Anschließend wird das so korrigierte Bild ins OFFIS Automation Framework geladen. Hier findet zunächst die Selektierung von Pixeln entsprechend einem Farbwert plus möglicher Toleranzgrenze statt. Anschließend wird die betrachtete Bildregion auf den Bereich um den Farbumschlag herum zugeschnitten und als Bild exportiert. Jetzt wird das exportierte Bild wieder mit GIMP geöffnet und alle schwarzen Pixel gelöscht. Schließlich wird das Bild auf die Größe 1x1 Pixel herunter skaliert und hierbei ausgenutzt, dass automatisch der Durchschnittsfarbwert aller Pixel-Farbwerte des Ursprungsbildes generiert wird. Per Pipetten-Tool kann nun der RGB- bzw. HSV-Wert dieser Durchschnittsfarbe abgefragt werden.

**Probleme**

**FiJi** FiJi implementiert eine Reihe nützlicher Funktionen, um Informationen aus Bildern zu extrahieren. Leider sind diese Funktionen durch die undurchsichtige Benutzeroberfläche nicht auf Anhieb zu finden. Außerdem sind einige Funktionen auf bestimmte Bildtypen, wie `binary` oder `greyscale`, beschränkt. Das liegt natürlich hauptsächlich an den zugrunde liegenden Algorithmen, es wäre trotzdem hilfreich, wenn Funktionen, die mit der aktuellen Konfiguration inkompatibel sind, kenntlich gemacht werden.

**Matlab / Octave** Ursprünglich wurde Octave mit in die Evaluation aufgenommen, weil es den Funktionsumfang von Matlab als open-source Projekt zur Verfügung stellt. Zusätzlich sollten die meisten Matlab Funktionen Octave kompatibel sein. Leider ist diese Kompatibilität in manchen Fällen aber nur theoretisch, so konnte das Teststreifenbild nicht auf Anhieb, wie in Kapitel 4.3.13.1 auf Seite 136 beschrieben, geladen werden.

Anders als in den anderen Tools, ist die GUI in Matlab eher nebensächlich. Um die volle Funktionalität von Matlab auszunutzen ist es unbedingt notwendig, die Sprache zu lernen. Das erhöht natürlich die Einarbeitungszeit und erschwert einfaches Ausprobieren.

**GIMP** Dies ist ein Endanwenderprogramm und daher nicht auf wissenschaftliche Arbeit ausgelegt. Aus diesem Grund stößt es schnell an seine Grenzen, wenn es um automatierte Kombination von Bildverarbeitungsverfahren geht. Ausserdem gibt es neben den Bildformaten keine Möglichkeit, Pixelwerte auf Textbasis zu exportieren. Das erschwert die Zusammenarbeit mit anderen Tools.

**OFFIS Automation Framework** Der große Vorteil des OFFIS Automation Frameworks ist es, dass es direkt OpenCV Funktionen implementiert und somit eine spätere Umsetzung in eine eigenständige Implementierung mit OpenCV ermöglicht wird.

**Fazit** Durch die logische Anwendung einer Reihe von Bildverarbeitungsfunktionen (siehe Abbildung 80 auf Seite 135) ist es möglich die Durchschnittsfarbe bzw. Farbintensität des Farbumschlags zu berechnen. Die Herausforderung wird darin bestehen dynamisch einzelne Parameter abhängig vom Input-Bild anzuwenden. Vor Allem Matlab ist ein sehr mächtiges Tool, für das dank der großen Community eine große Bibliothek an Skripten zur Verfügung steht[20]. Unterm Strich ist festzustellen, dass jedes Programm seine Vor- und Nachteile hat und eine ausgewogene Kombination aller Tools am zuverlässigsten zum Ziel führt.

[20] `http://de.mathworks.com/matlabcentral/fileexchange/`, zuletzt geöffnet am 04.08.2015

**Ausblick** Die verwendeten Tools sind ein guter Anfang, um auf einfache Art und Weise Bildverarbeitungsfunktionen auszutesten, jedoch kann keines von ihnen als Hintergrunddienst auf den Zielgeräten verwendet werden. Das Ziel muss es sein, die Algorithmen so zu kombinieren, dass sie auch eine automatische Auswertung der Teststreifen auf den Endgeräten zuverlässig garantieren.

### 4.3.14 Entwicklung eines Algorithmus für den Weißabgleich

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket ist notwendig, um das aufgenommene Foto für die Detektierung der Farbumschlagfelder zu verbessern.
- Arbeitspaketbeschreibung:
  Die Fotos, die von einem Probanden aufgenommen und für die Teststreifen-Analyse verwendet werden, sind von diversen Faktoren beeinflusst, wie etwa der Qualität der Kamera, den Einstellungen der Kamera, den Lichtverhältnissen in der Umgebung des Teststreifens und dem Winkel, unter welchem der Teststreifen aufgenommen wurde. Um diesen Faktoren entgegen zu wirken, wird ein Weißabgleich auf dem aufgenommenen Foto durchgeführt. So werden die Farben auf dem Bild normalisiert und können vom Bildanalysealgorithmus korrekt erkannt werden. Eine beispielhafter Weißabgleich kann der folgenden Abbildung 84 entnommen werden.

Abbildung 84: Beispiel eines Weißabgleiches

- Vorbedingungen:
  -
- Nachbedingungen:
  Ein Testbild mit einem Farbumschlagsfeld wird zweimal unter jeweils unterschiedlichen Lichtbedingungen (überhellt und in einer dunklen Umgebung)aufgenommen. Nachdem der Algorithmus den Weißabgleich durchgeführt hat, müssen die originalen Farbwerte, mit einer erlaubten Abweichung von 1% erkannt werden.
- Nebenbedingungen:
  -
- Aufwand:
  Zwei Wochen
- Personen:
  - Sebastian Horwege
  - Christoph Ressel
  - Timo Raß

**4.3.14.1 Dokumentation** Grundsätzlich wird ein Weißabgleich auf einem Foto durchgeführt, um verfälschte Weißtöne wieder wie im Original darstellen zu können. Das menschliche Auge kann unterschiedliche Weißtöne kompensieren, so dass Weiß auch immer Weiß aussieht. Diese Kompensation der unterschiedlichen Farbtemperaturen wird auch als chromatische Adaption bezeichnet. Die Verfärbungen der unterschiedlichen Weißwerte auf einem Foto entstehen durch unterschiedliche Lichtwerte und werden in Kelvin angegeben. Die Weißtöne können dabei auf unterschiedliche Weise verfälscht sein. So können sie z. B. gelblich (niedrige Kelvinzahl) oder auch bläulich (hohe Kelvinzahl) wirken. Es wird auch von warmen und kalten Weißtönen gesprochen. Eine Skala der Verfärbungen kann der folgenden Abbildung entnommen werden.

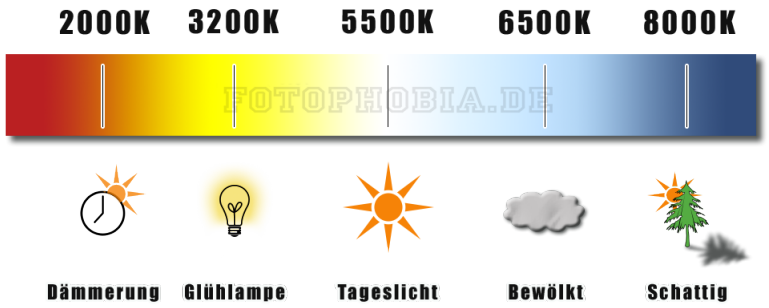



Abbildung 85: Beispiel von Farbtemperaturen

Das Arbeitspaket Weißabgleich ergab sich aus der folgenden Anforderung, welche dem Lastenheft zu entnehmen ist:

- FA-4: Die mobile Applikation verfügt über einen Bildverarbeitungsalgorithmus, mit dem eine vorläufige Auswertung des Teststreifens vorgenommen werden kann.

**Ablauf**

- Zunächst wurde damit begonnen die grundsätzliche Herangehensweise zu definieren, um das Arbeitspaket fokussiert abarbeiten zu können. Da Sebastian bereits Erfahrungen mit MatLab sammeln konnte, wurde entschieden, diese Software für die ersten Versuche des Weißabgleichs zu verwenden.

- Der erste Schritt war es, einen Teststreifen, welcher mit dem Teststreifengenerator erzeugt wurde und einem Laser-Farbdrucker ausgedruckt wurde, abzufotografieren. Dazu wurde das LG-Testgerät aus dem Projektgruppenraum verwendet. Das Bild wurde auf den Computer kopiert und in das MatLab Projekt importiert.

- Um einen Weißabgleich durchzuführen wird in der Regel auf dem Bild eine Stelle gesucht, welche näherungsweise dem exakten Weißwert entspricht ($RGB : 255, 255, 255$). Diese dient als Referenzwert, um die weißen Stellen im Bild im Anschluss wieder weiß erscheinen zu lassen. Dies kann ebenso mit einem grauen Bereich des Bildes erfolgen. In MatLab kann mit dem Befehl **getrect** ein bestimmter Bereich aus einem Bild manuell ausgewählt werden. Dieser wurde verwendet, da noch keine automatische Erkennung der Referenz- und Farbumschlagsfelder implementiert wurde. Somit konnte auf dem importierten Teststreifen mit der Maus eine Stelle markiert werden die näherungsweise weiß bzw. grau ist und als Referenz dient.

- Nachdem der zuvor beschrieben Bereich des Bildes ausgewählt wurde, begann der Weißabgleich. Wurde dieser durchgeführt, wird das überarbeitete Bild automatisch ausgegeben. Der Algorithmus ist im Quellcode kommentiert.

**Technologien**

**MatLab** MatLab ist eine Software, welche zur Lösung mathematischer Probleme und zur grafischen Darstellung der Ergebnisse entwickelt wurde. MatLab ist dabei speziell für die Berechnung nummerischer Probleme ausgelegt. Eine besondere Stärke von MatLab ist der Umgang und das Rechnen mit Vektoren und Matrizen. Dies ist für die Berechnung von Bilddaten von großer Bedeutung, da sich Bilder aus vielen Zahlenwerten (z. B. RGB) matrizenartig zusammensetzen.

MatLab kann auf den Plattformen Windows, Linux, Mac OS genutzt werden.

**Systembeschreibung** Die Entwicklungsumgebung setzte sich aus der Version MatLab R2013a und einem Linux-Betriebssystem zusammen. Zur kollaborativen Entwicklung wurde der Branch Whitebalance des Git-Repository medic-code verwendet.

**Evaluation** Zum Testen des Weißabgleiches wurden die Teststreifen des Teststreifengenerators heranzogen. Diese wurden mit dem PG-Testgerät abfotografiert und in das MatLab-Projekt integriert. Der Kelvinwert des Bildes wurde mehrfach verändert, sodass die Aufnahme von unterschiedlichen Bildern simuliert werden konnte. Es wurde in diesem Schritt explizit darauf verzichtet mehrere Bilder aufzunehmen, da hierdurch viel mehr Faktoren als nur der Farbversatz in eine Richtung stattfindet. Nicht zu unterschätzen ist vor Allem das Rauschen, das durch den Kamerasensor an unterschiedlichen Pixelkoordinaten unterschiedlich stark sein kann. Der Algorithmus zum Weißabgleich orientiert sich am Grey World Algorithmus [13] und der Retinex Theorie [14], dabei wird der Weißbereich des Bildes aber nicht durch den Algorithmus bestimmt, sondern manuell vom Benutzer vorgegeben.

Das Eingabebild $R$ wird als $n \times m$ Matrix über dem RGB Farbraum betrachtet:

$$R \in \begin{pmatrix} r \in [0,255] \\ g \in [0,255] \\ b \in [0,255] \end{pmatrix}^{n \times m}$$

Dabei sind die die Matrizen $R_r$,$R_g$ und $R_b$ die Repräsentation für die einzelnen Farbkanäle. Aus diesem Bild wird dann ein Teilbereich ausgewählt, der eine graue oder weiße Fläche repräsentiert.

Aus dieser Fäche wird dann die am häufigsten auftretende Farbe (zum ausfiltern von Rauschen) als Weißreferenz ausgewählt und anschließend die Korrekturfaktoren berechnet:

$\alpha = \frac{{R_g}^w}{{R_r}^w}$, $\beta = \frac{{R_g}^w}{{R_g}^w}$ und $\gamma = \frac{{R_g}^w}{{R_b}^w}$

Die Faktoren für die einzelnen Farbwerte müssen dann mit jedem Pixel des Bildes verrechnet werden:

$$R^*_r := \alpha R_r$$

$$R^*_g := \beta R_g$$

$$R^*_b := \gamma R_b$$

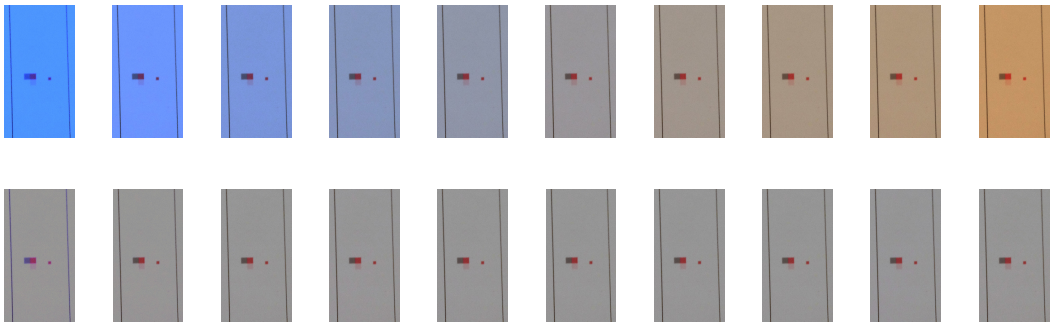


Abbildung 86: Zur Evaluation des Weißabgleichalgorithmus wurde eine Bildreihe aus dem aufgenommenen Teststreifenfoto erstellt

Dieser Algorithmus konnte auf den meisten der simulierten Bildern ein zufriedenstellendes Ergebnis liefern (siehe Abbildung 86). Dabei wurde der Weißabgleich des Ausgangsbildes (Mitte obere Reihe) in 10 Schritten modifiziert. Beim Ausgangsbild wurde von einer Farbtemperatur von $6500K$ ausgegangen. Die bearbeiteten Bilder haben jetzt eine Spanne von $3000K$ (Abbildung 86 ganz links) bis $10000K$ (ganz rechts). In der zweiten Reihe der Abbildung ist das Ergebnis des Weißabgleichs für das darüberliegende Bild zu sehen. A bbildung 87 zeigt eine genauere Auswertung. Tatsächlich konnte aber bei einem Bild mit zu starker Verschiebung die Toleranz von 1% nicht eingehalten werden. Das wird in Abbildung 87 verdeutlicht. Zur besseren Visualisierung wurde die Durchschnittsfarbe in ihre drei RGB Kanäle aufgeteilt und und die Werte auf die Y-Achse aufgetragen. Auf der X-Achse sind die Kelvin Werte der einzelnen Bilder abgebildet. Es ist zu bemerken, dass der Grünkanal über die gesamte Spanne der Variationen konstant ist. Das ist damit zu erklären, dass verschiedene Lichtquellen sich nur im blauen und roten Spektrum unterscheiden. Außerdem ist zu erkennen, dass $3000K$ die einzige Farbtemperatur ist, bei der der Weißabgleich fehlgeschlagen ist. Bei allen anderen konnte die vorgegebene Genauigkeit von 1% eingehalten werden.

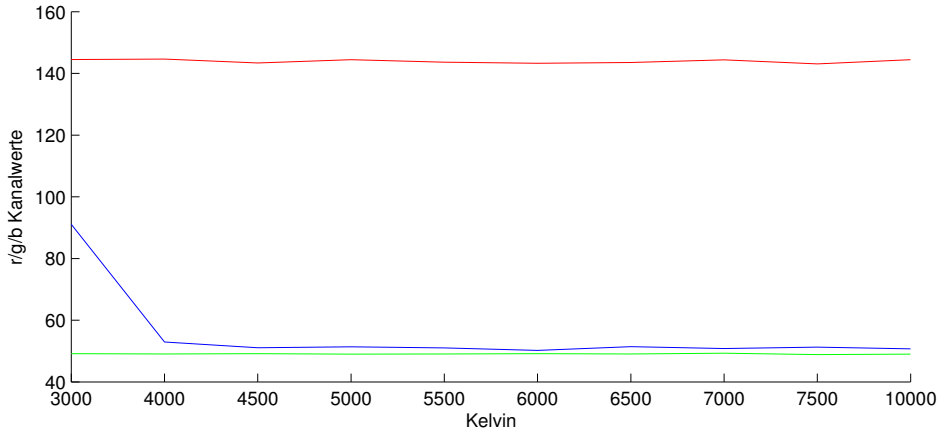



Abbildung 87: Durchschnittsfarbe für die Umschlagbereichs der Bilder aus Abbildung 86 nach dem Weißabgleich

Die aus dem Arbeitspaket Teststreifengenerator (Kapitel 4.3.12.1 auf Seite 129) generierten Teststreifen enthalten zusätzlich zu dem Umschlagfeld Referenzbereiche für die maximale, sowie minimale Farbe des Umschlags. Diese Referenzbereiche sind hilfreich, um die Nützlichkeit und Güte eines Weißabgleichs zu evaluieren. Dazu haben wir zuerst alle drei Bereiche ausgewählt (Abbildung 88) und anschließend verschiedene Verfahren verwendet, um Ähnlichkeiten zwischen den Bereichen zu erkennen.


Abbildung 88: Zum Vergleich wurden vom Teststreifen 3 Bereiche manuell ausgewählt

Dabei ist eine Vergleichsmöglichkeit, die Distanzen zwischen den Durchschnittsfarben der häufigsten Farben der einzelnen Bereiche zu bestimmen. Als Distanzmaß wurde die euklidische Distanz im RGB-Farbraum gewählt. Die Ergebnisse der Vergleiche sind schon sehr vielversprechend (siehe Abbildung 89), es gibt aber noch reichlich Möglichkeiten, den Farbumschlag anders zu qualifizieren (anderes Distanzmaß, anderer Farbraum, andere Auswahl von Referenzpixeln). Abbildung 89 zeigt die Distanz der Durchschnittsfarben (berechnet aus den 1 bis 50 häufigsten Farben in den selektierten Bereichen, X-Achse) zwischen dem Umschlagsbereich und dem maximal Referenzfeld (rot) und dem Umschlagsbereich und dem minimalen Referenzfeld (blau). Es ist zu erkennen, dass in diesem Beispielbild die Auswahl der 6 Häufigsten Farben zum eindeutigsten Ergebnis führt, wohingegen die Genauigkeit stetig abzunehmen scheint, je mehr Farben mit ins Gewicht genommen werden. Natürlich wurden die Häufigkeiten der einzelnen Farben bei der Berechnung der Durchschnittsfarbe berücksichtigt.

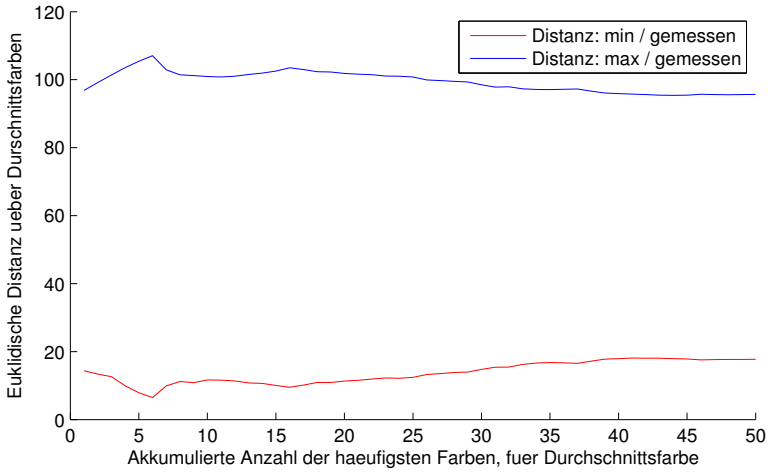



Abbildung 89: Verschiedene Methoden zur Bestimmung der Ähnlichkeit vom Farbumschlagsfeld und den Referenzfeldern

Im Folgenden wird nun der Vergleich des eindeutigsten Ergebnis aus Abbildung 89 mit und ohne Weißabgleich gezeigt:

| ohne Weißabgleich | mit Weißabgleich | |
|---|---|---|
| 103.4130 | 107.0797 | Distanz Umschlag / minimale Referenz |
| 6.5775 | 6.4747 | Distanz Umschlag / maximale Referenz |

Es ist zu erkennen, dass der Weißabgleich die Aussage des Vergleichs leicht verstärken kann.

**Parameter** Das Arbeitspaket wurde unvoreingenommen bearbeitet. Als Parameter dienten lediglich die Teststreifen aus dem Arbeitspaket des Teststreifengenerators. Diese wurden als Grundlage verwendet, um einen Weißabgleich durchzuführen.

**Fazit** Es konnte in diesem Arbeitspaket ein Weißabgleich implementiert werden. Durch eine intensive Einarbeitung in grundlegende Konzepte der Bildverarbeitung konnten neue Erkenntnisse zur Verarbeitung von Bilddaten gewonnen werden. Es kann jedoch darüber diskutiert werden, ob ein Weißabgleich überhaupt notwendig ist, da die Detektierung eines reinen Farbumschlages auch ohne diesen erfolgen kann.

**Ausblick** Aufbauend auf dem Weißabgleich wurde damit begonnen, die Farbintensitäten bzw. den Farbumschlag des Teststreifens nach dem Weißabgleich zu identifizieren. Diese anfänglichen Untersuchungen konnten aufzeigen, dass die Bilder mit einem hohen Rauschen belastet sind, was zur Folge hat, dass das Farbumschlagsfeld verschiedene Farbwerte beinhaltet (z. B. unterschiedliche rote Farben). Für zukünftige Arbeitspakete z. B. die Intensitätsdetektierung ist zu beachten, welche Schwellwerte des Farbumschlags relevant sind. Für die weitere Bearbeitung kann das aktuelle MatLab Projekt aus dem Git-Code Repository Weißabgleich herangezogen werden. Dieses beinhaltet bereits eine Funktionalität, welche die Farbwerte und auch die Häufigkeiten bestimmter Farbintensitäten aus dem Farbumschlagsfeld identifizieren kann. Als nächster Schritt muss der in MatLab entwickelte Algorithmus in die Smartphoneapplikationen integriert werden. Abhängig von zukünftigen Erkenntnissen muss er somit in C/C++ bzw. in Java und Objective-C implementiert werden. Die Erkenntnisse aus diesem Arbeitspaket reichen, um einige weitere Arbeitspakete zu definieren: Durch die Vorkenntnis der Positionen der weißen Bereiche auf dem Teststreifen könnte eine Belichtungskarte des Teststreifens erzeugt werden, um Schatten, die durch den Benutzer entstehen zu eliminieren. Außerdem haben wir erkannt, dass es eine Menge an Möglichkeiten gibt, die Intensität des Farbumschlags zu detektieren und mit den Referenzen zu vergleichen.

#### 4.3.15 Detektierung der Intensität- und Farbsättigung

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Mit diesem Arbeitspaket sollen erste Ergebnisse in der Detektierung des Farbumschlagfeldes erzielt werden.
- Arbeitspaketbeschreibung:
  Der Bildverarbeitungsalgorithmus ist in der Lage die Intensität bzw. die Farbsättigung eines Farbumschlags in einem definierten Bereich festzustellen. Grundsätzlich ist die Intensität von nur einer bestimmten Farbe zu bestimmen (z.B. rot mit dem maximalen rgb 255,0,0). Die Farbintensität vier verschiedener Farben kann der unten stehenden Abbildung 90 auf der nächsten Seite entnommen werden.

Abbildung 90: Vier verschiedene Farbintensitäten

Zusätzlich kann der Algorithmus, das Ergebnis je nach Intensität und Sättigung auszugeben und diesen auf dem Bild zu dokumentieren.

- Vorbedingungen:
  Das Arbeitspaket „Weißabgleich“ muss abgeschlossen sein. Zudem ist Evaluation von Bildverarbeitungs-Tools abgeschlossen, damit die Ergebnisse des Algorithmus durch die Ergebnisse der Bildverarbeitungstools abgeglichen werden können.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der Algorithmus liefert innerhalb von drei Sekunden eine Aussage darüber, ob ein positives, negatives oder unklares Ergebnis eines Testbilds vorliegt.
- Aufwand:
  Drei Wochen
- Personen:
  - Sebastian Horwege
  - Nicolas Koch

**4.3.15.1 Dokumentation** Das Arbeitspaket „Intensitätsdetektierung / Farbsättigung“ wird in den nächsten Sprints genauer behandelt, da es die Quintessenz der Bilderkennung bildet. Zudem wurde das Arbeitspaket „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs “ (siehe Kapitel 4.3.16.1 auf der nächsten Seite) vorgezogen, damit die Bildbereiche, welche der Intensitätsdetektierung zugrunde liegen, nicht immer manuell ausgewählt werden müssen. Dies optimiert den Arbeitsablauf für folgende Arbeitspakete und ist von der Priorität nicht weniger wichtig. Die Intensitätsdetektierung muss somit nur noch kleine, konstant große Bildbereiche analysieren, was zudem die Performance im Nachhinein positiv beeinflussen kann.

**4.3.15.2 Übernahme** Das Arbeitspaket wird in Sprint 5 bzw. nach Absprache in Sprint 6 übernommen.

### 4.3.16 Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Das Arbeitspaket wird benötigt, um den Bereich, welcher das Farbumschlagsfeld beinhaltet enthält, lokalisieren zu können.
- Arbeitspaketbeschreibung:
  Das Bildverarbeitungsprogramm enthält eine Funktionalität die es ermöglicht, innerhalb eines definierten/abgesteckten Bereichs die Pixel zu finden, die den Farbumschlag repräsentieren.

- Vorbedingungen:
  Das Arbeitspaket „Weißabgleich“ ist erfolgreich abgeschlossen. Das zu analysierende Bild ist durch den Weißabgleich korrigiert worden.

- Nebenbedingungen:
  -

- Nachbedingungen:
  Der entwickelte Algorithmus liefert für zehn unterschiedliche Beispielbilder den richtigen Farbumschlagsbereich. Das Detektieren des Bildbereichs dauert auf allen PG-Rechnern nicht länger als zwei Sekunden.

- Aufwand:
  Vier Wochen

- Personen:
  - Sebastian Horwege
  - Christoph Ressel
  - Timo Raß

**4.3.16.1 Dokumentation** Ziel dieses Arbeitspaketes ist es, den Bereich im Bild zu detektieren, der den Farbumschlag repräsentiert.

**Ablauf** Zunächst wurde sich zur Implementierung für die Bibliothek OpenCV und die dafür vorgeschlagene Programmiersprache C++ entschieden. Anders als im vorherigen Arbeitspaket „Bildverarbeitung“ aus dem Sprint 3 (siehe Kapitel 4.2.8.1 auf Seite 81), wurde diesmal als neuer Ansatz das Erkennen von Markern verfolgt, welcher aus dem Bereich der Augmented Reality[21] stammt. Diese Marker werden dazu verwendet, den Bereich abzustecken in dem der Farbumschlag gesucht werden muss. Hierzu wurde ein eigenes Format an Markern erzeugt, das vom Aussehen her einem QR-Code ähnelt, aber deutlich kleiner ist und nur 16 Bit an Informationen abspeichern kann, also z.B. eine Zahl zwischen 0 und 65535. Das Aussehen der Marker orientiert sich am openAR Projekt[22].

Mit Hilfe des neuen Markerformats wurde danach der Teststreifengenerator so modifiziert, dass um den Bereich des Farbumschlags und der Referenzfelder in rechteckiger Form vier unterschiedliche Marker angebracht sind. Ab hier konnte versucht werden, die Marker mit Hilfe von OpenCV in Bildern zu detektieren. Als sehr hilfreicher Ansatz konnte hier bereits bestehender Quellcode von Bharath Prabhuswamy genutzt werden, um die Marker in einem Bild zu detektieren. Darauf aufbauend werden die Mittelpunkte der vier abgrenzenden Marker bestimmt und somit der Bildbereich grob abgesteckt, in dem der Farbumschlag zu suchen ist. Der auf diese Boundingbox zugeschnittene Bildbereich wird nun per affiner Transformation auf ein immer gleich großes Bild projiziert. Hierdurch kann in diesem transformierten Bild immer an der exakt gleichen Position der Farbumschlag angenommen werden.

[21] `https://de.wikipedia.org/wiki/Erweiterte_Realit%C3%A4t`, zuletzt geöffnet am 01.09.2015
[22] openAR: `https://github.com/bharathp666/openAR`

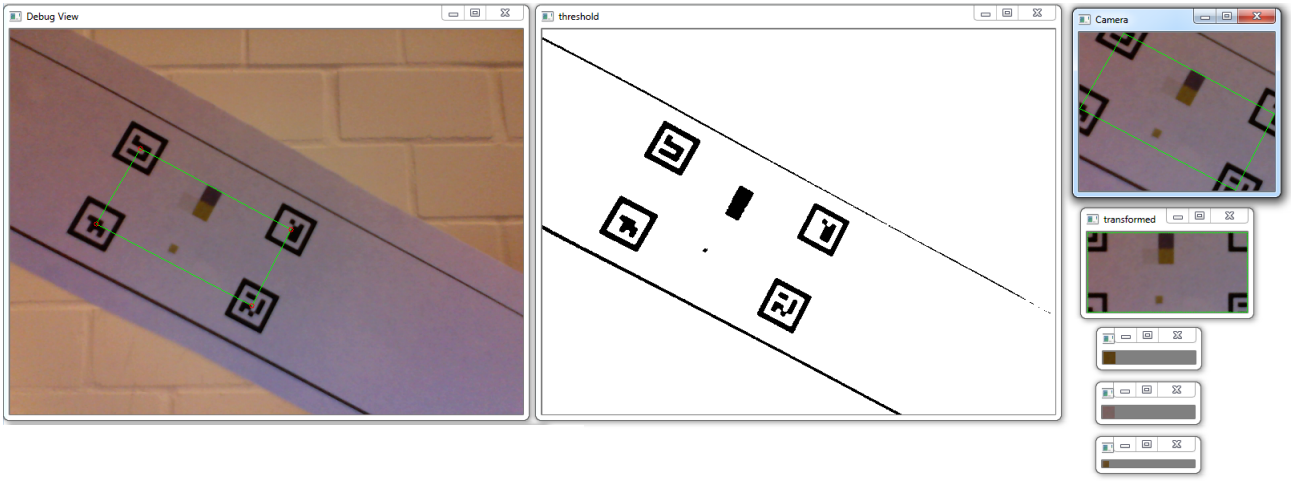



Abbildung 91: Die Ausgabe des mit OpenCV in C++ entwickelten Programms.

**Technologien**

**OpenCV** Eine in C/C++ implementierte Bildverarbeitungsbibliothek.

**Augmented-Reality** Ein Ansatz Objekte der Realität mit künstlichen Informationen zu überlagern und somit einen erweiterte Realität zu erschaffen. Essentiell ist hierbei oftmals das Lokalisieren von Objekten/Markern, um die Informationen an der richtige Stelle (z.B. visuell oder akustisch) darstellen zu können.

**Systembeschreibung** Entwickelt wurde das Programm auf einem Linux und einem Windows 7 Rechner.

**Evaluation** Zum Testen des entwickelten Algorithmus werden zehn unterschiedliche Teststreifen nacheinander vor die Kamera gehalten und aus mehreren Winkeln betrachtet. Sie unterscheiden sich nur in der Farbe der Farbumschlags- und Referenzfelder. Der Test wurde als erfolgreich angesehen, wenn der entwickelte Algorithmus innerhalb von zwei Sekunden auf einem parallel zur Kamera ausgerichteten Teststreifen alle vier Marker gleichzeitig erkannt hat. Auf diese Art und Weise wurden zehn Teststreifen erfolgreich getestet.
Es fiel jedoch auf, dass die Positionierung der Elemente im transformierten Bild minimale Unterschiede aufweisen kann und daher die festen Bereiche für den Farbumschlag und die Referenzfelder etwas großzügiger abgemessen sein müssen. Marker konnten nicht erkannt werden, wenn der Teststreifen um 90 Grad gedreht war, da sie dann zwar als Marker detektiert werden, jedoch auf Grund der quadratischen Form aus Sicht der Kamera andere Daten kodieren, als die die gesucht sind.


Abbildung 92: Die Ergebnisse der Evaluation

**Parameter** Das Arbeitspaket ist unter der Annahme vervollständigt, dass die Form der platzierten Marker unverfälscht sichtbar sind. Das soll heißen, dass die Marker nicht durch kontraststarken und großflächigen Schmutz bedeckt sind.

**Fazit** Der entwickelte Algorithmus liefert sehr zuverlässig den gesuchten Farbumschlagsfeld. Wichtig ist bei diesem Stand zu beachten, dass der Teststreifen nicht zu stark gedreht sein darf (maximal 45 Grad abweichend von paralleler Ausrichtung zur Kamera).

**Ausblick** Zukünftig muss der entwickelte Algorithmus in Android und iOS integriert werden. Mit Hilfe der Marker können theoretisch weitere, für den Endnutzer hilfreiche, Informationen angezeigt werden. Z. B. kann dem Nutzer ein visuelles Feedback gegeben werden, wie viele und welche Marker bereits erkannt wurden. Der Endnutzer kann dadurch ableiten, in wie weit er sein Smartphone richtig auf den Teststreifen ausgerichtet hat und dies ggf. neu justieren.

### 4.3.17 Fazit des Sprints

Der größte Teil der für diesen Sprint definierten Arbeitspakete konnte erfolgreich abgeschlossen werden. Lediglich das Arbeitspaket „Intensitätsdetektierung“ (Kapitel 4.3.15.1 auf Seite 145) wurde nicht in diesem Sprint abgearbeitet. Dafür wurde das Arbeitspaket „Lokalisierung des Farbumschlags innerhalb eines definierten Bereichs“ (Kapitel 4.3.16.1 auf Seite 146) vorgezogen, so dass die Intensitätsdetektierung in einem der nächsten Sprints darauf aufbauen kann.
Nach der Evaluation der Frameworks für die Weboberfläche, konnte diese mit PrimeFaces neu entwickelt werden. Die Wartbarkeit des Quellcodes hat sich damit stark verbessert, da die Entwicklung mit PrimeFaces über Java realisiert wird und das Team hier die meiste Erfahrung und Kompetenz besitzt. Des Weiteren ist die Weboberfläche durch die getätigten Erweiterungen besser zu bedienen und es wurden zusätzliche Funktionen implementiert. Die Performance der Heatmap hat sich stark verbessert, so dass diese um einiges flüssiger zu bedienen ist als vorher.
Mit dem entwickelten Teststreifengenerator lassen sich in kurzer Zeit viele verschiedene Teststreifen erstellen, so dass diese in Zukunft für die Tests in der Entwicklung genutzt werden können. Die Teststreifen werden unter Anderem dafür eingesetzt um den Algorithmus zur Bilderkennung zu testen.
Die Evaluation zu den Bildverarbeitungstools wurde durchgeführt und es wurden probeweise Algorithmen zur Bilderkennung getestet. Aus dieser Evaluation und den bereits vorhandenen Erfahrungen im Team, wurde sich dafür entschieden, den Weißabgleich mit Hilfe von MatLab zu implementieren. Diese Tatsache führte allerdings dazu, dass bei dem Arbeitspaket zur Lokalisierung des Farbumschlags, der Weißabgleich noch nicht integriert werden konnte. Dennoch konnte die Lokalisierung des Farbumschlags ohne den Weißabgleich realisiert werden.
Damit wurde der vierte Sprint erfolgreich abgeschlossen und alle wichtigen Arbeitspakete umgesetzt. Das Arbeitspaket zur Intensitätsdetektierung wird mit in den nächsten Sprint übernommen, so dass es dort abgeschlossen werden kann.

## 4.4 Sprint 5

Im Fokus des Sprints steht mit 75% Anteil (9 von insgesamt 12 Arbeitspaketen) die Bildverarbeitung. Dabei sollen die vorhandenen Ansätze überarbeitet sowie effektiver und effizienter gestaltet werden. Die automatische Detektierung des Farbumschlags, die Integration des Weißabgleichs, sowie die Schaffung einer gemeinsamen Quellcodebasis für die Android- und iOS-Apps sind dabei die im Vordergrund stehenden Ziele.
Des Weiteren soll innerhalb dieses Sprints ein Teststand erschaffen werden (vgl. Abbildung 93), welcher es ermöglicht, verschiedenen Quellcode zur Bildauswertung unter gleichen Bedingungen zu evaluieren. Die Evaluation geschieht zudem unter Zuhilfenahme des im vergangenem Sprint erstellten Teststreifengenerators (Seite 42 im Abschlussbericht von Sprint 4).

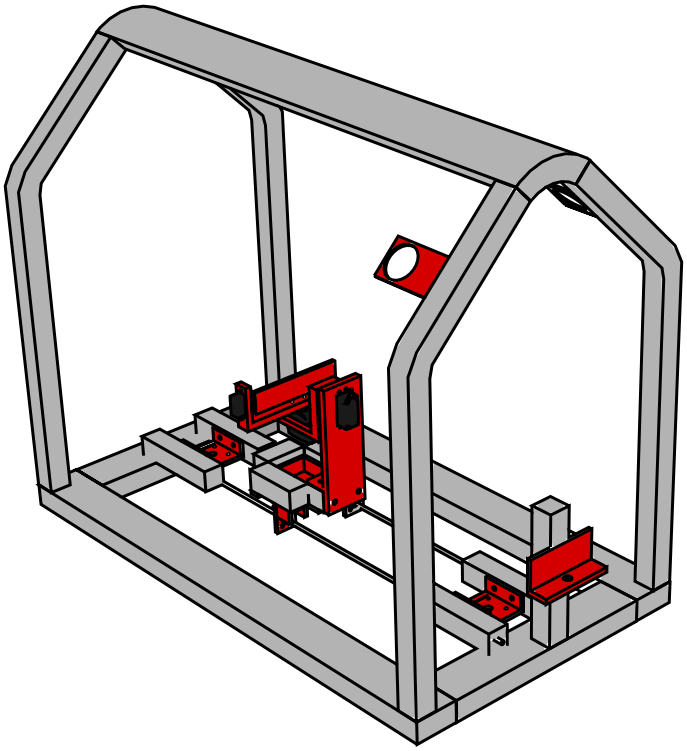


Abbildung 93: Schematische Darstellung des Teststands

Ein weiterer Teil dieses Sprints ist die Schaffung einer Übersicht der Systemlandschaften sowie die ausführliche Dokumentation der Systemkomponenten, welche zukünftig nicht mehr als prototypische Umsetzung angesehen werden. Im Zuge des Arbeitspaketes „Gesamtarchitektur visualisieren“ (Kapitel 4.4.3 auf Seite 158) wird angestrebt, eine Visualisierung der Systemlandschaften vor und nach eines jeden einzelnen Sprints auszuarbeiten. Die daraus resultierende Visualisierung der Systemlandschaft zu Beginn dieses Sprints, findet sich Abbildung 94 auf der nächsten Seite wieder.

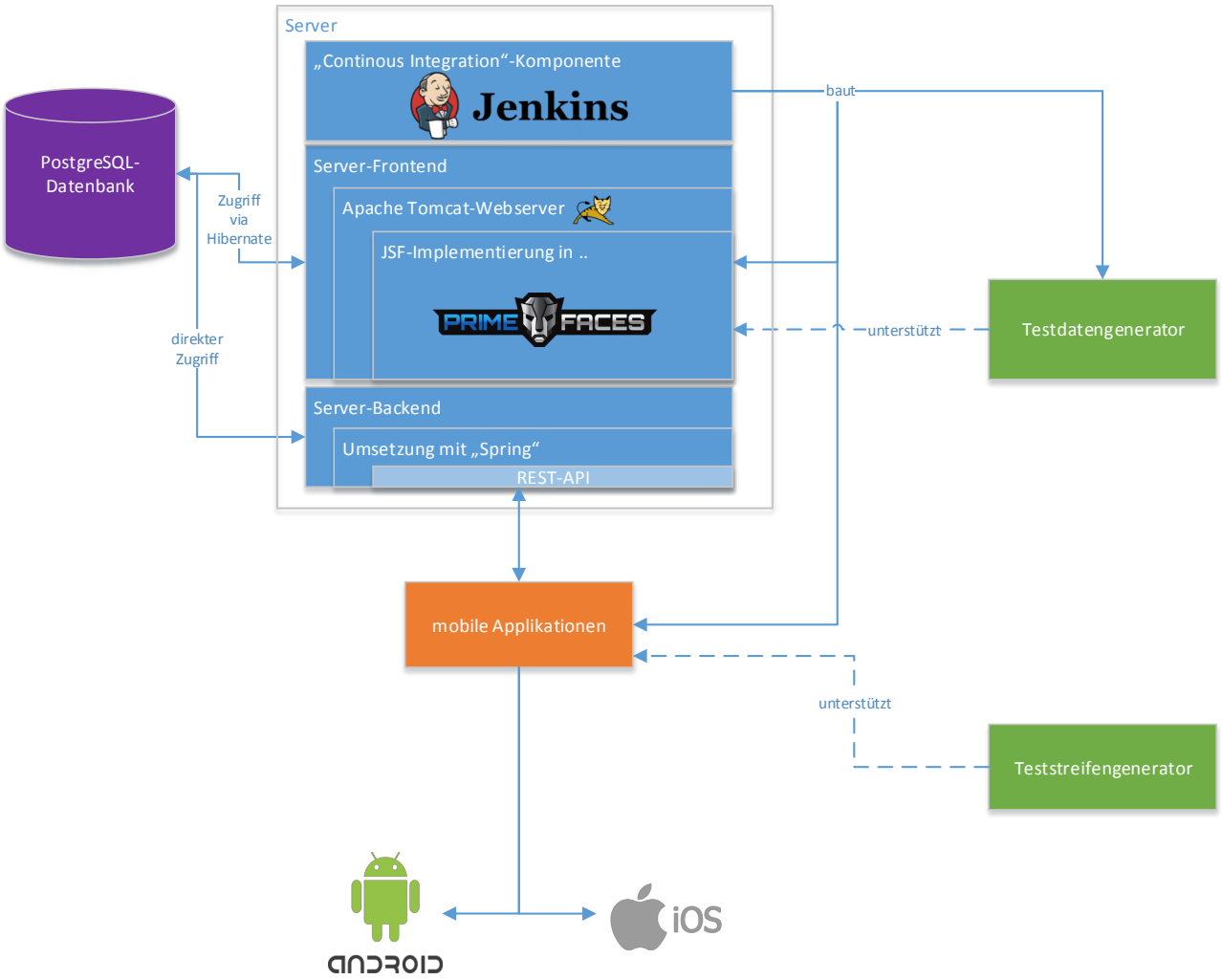



Abbildung 94: Systemlandschaft zu Beginn von Sprint 5

Diese Abbildung dient als Basis, welche für eine weitere Grafik verwendet wurde, in der die Systemlandschaft in zwei Grobsysteme „Server und Weboberfläche“ sowie „mobile Applikation“ unter Angabe eines Fortschrittswertes in Prozent unterteilt wurde (vgl. Abbildung 95 auf der nächsten Seite).

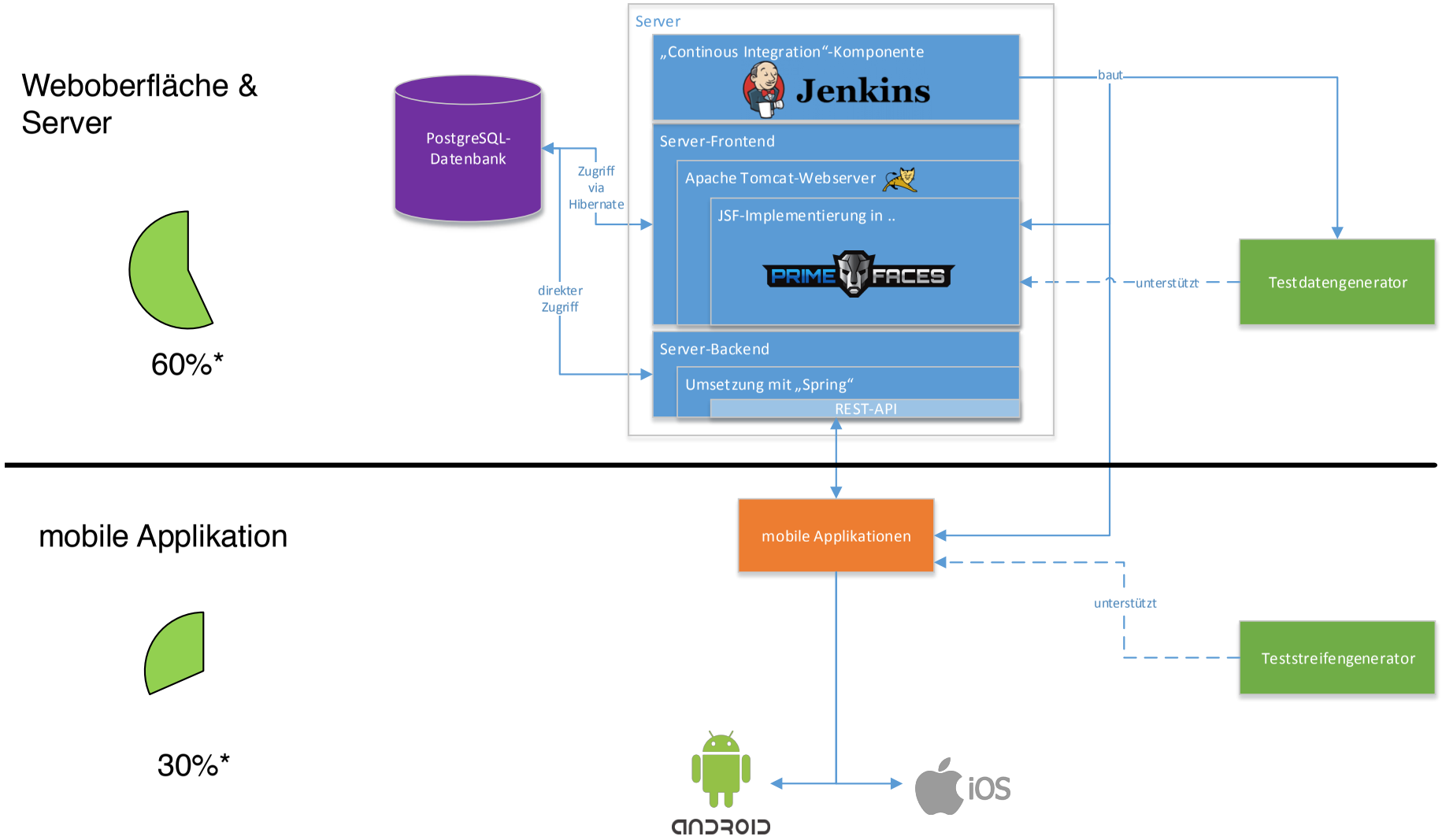



Abbildung 95: Fortschrittsangaben nach Sprint 4

### 4.4.1 Sprintplanung

Im Folgenden wird die Planung des fünften Sprints (09.09.2015-21.10.2015) genauer aufgezeigt. Dabei wird ein besonderes Augenmerk auf die folgende Punkte gelegt:

- Fokus des Sprints
- Arbeitspakete und Planung
- Kritischer Pfad und Planung der Ressourcen

**4.4.1.1 Fokus des Sprints** Wie auch in den zuvor geplanten Sprints wurde in der letzten Woche des vorigen Sprints damit begonnen neue Arbeitspakete, welche sich unter anderem aus den Anforderungen und neuen Erkenntnissen ergaben, zu definieren und zu planen. Diese wurden anschließend ausgedruckt und für die Teammitglieder im Projektgruppenraum ausgehängt. Der Themenschwerpunkt sollte nicht mehr auf dem bereits sehr fortgeschrittenen Webfrontend liegen, sondern in der **Bildverarbeitung** und der **Integration der Bildverarbeitungsalgorithmen** in die iOS und Android Apps. Zudem sollte ein Teststand erstellt werden, welcher das Testen der Bildverarbeitungsalgorithmen unter konstanten und reproduzierbaren Bedingungen ermöglicht.

**4.4.1.2 Arbeitspakete und Planung** Der zeitliche Aufwand der Arbeitspakete wurde anhand einer Methode aus dem Projektmanagement, der sogenannten **Delphi-Methode**, ermittelt. Bei dieser Methode geben mehrere fachkundige Personen ihre Einschätzung zum Aufwand ab. Anschließend wird gemeinsam über den Aufwand der Arbeitspakete diskutiert, bis eine realistische und annehmbare Zeit gefunden wurde. Es ergaben sich zunächst die folgenden Arbeitspakete und Aufwandseinschätzungen:

- Automatische Detektierung der Farbumschlagsfelder in Android (Priorität: kritisch - zwei Wochen à zwei Personen)
- Automatische Detektierung der Farbumschlagsfelder in iOS (Priorität: kritisch - zwei Wochen à zwei Personen)
- Integration des Weißabgleichs in iOS (Priorität: kritisch - zwei Wochen à zwei Personen)
- Integration des Weißabgleichs in Android (Priorität: kritisch - zwei Wochen à zwei Personen)
- Integration von OpenCV in iOS (Priorität: kritisch - eine Woche à zwei Personen)
- Integration von OpenCV in Android (Priorität: kritisch - eine Woche à zwei Personen)
- Code-Refactoring Primefaces (Priorität: normal - vier Wochen à zwei Personen)
- Erstellung eines Teststandes (Priorität: normal - sechs Wochen à drei Personen)
- Gesamtarchitektur visualisieren (Priorität: normal - eine Woche à zwei Personen)
- Entwicklung der Intensitätsdetektierung (Priorität: normal - zwei Wochen à zwei Personen)

Während des Sprints wurden zusätzlich zwei neue Arbeitspakete definiert, da die Überlegung aufkam, eine erneute Machbarkeitsstudie über gemeinsamen Quellcode für die iOS und Android App durchzuführen. Es sollte untersucht werden, ob es möglich ist die Bildverarbeitungsalgorithmen in einem einheitlichen Quellcode verfassen zu können um so die Wartbarkeit und die Qualität des Quellcodes zu erhöhen. Zudem müssten in den zukünftigen Sprints nur noch zwei anstatt vier Personen an den Bildalgorithmen arbeiten, da diese nicht mehr in zwei Programmiersprachen vorliegen müssen. Daraus ergab sich folgendes Arbeitspaket:

- Machbarkeitsstudie Shared C++ Code für die Bildverarbeitungsalgorithmen (Priorität: kritisch - eine Woche à zwei Personen)

Die Machbarkeitsstudie kam zu dem Ergebnis, dass eine geteilte Codebasis möglich ist, wodurch sich ein weiteres Arbeitspaket ergab:

- Bildverarbeitungsalgorithmen in C++ umschreiben und in die iOS und Android App integrieren (Priorität: kritisch - drei Wochen à vier Personen)

**4.4.1.3 Planung der Ressourcen** Zunächst wurde eine Woche eingeplant, um die Apps zu reaktivieren und OpenCV in diese zu integrieren. Darauf aufbauend konnten die Bildverarbeitungs-Arbeitspakete durchgeführt werden. Sobald es in einem dieser Arbeitspakete zu Verzögerungen gekommen wäre, hätte dies auch die folgenden Bildverarbeitungs-Arbeitspakte beeinflusst. Daher war die Priorität dieser Aufgaben kritisch und die Bearbeitungszeit begann, wie auch beim Teststand, direkt in der ersten Woche. Das Arbeitspaket „Erstellung eines Teststandes“ (Kapitel 4.4.14.1 auf Seite 192) war das einzige, welches für mehr als fünf Wochen angesetzt war und somit in die Pufferzone am Ende des Sprints hineinragte, die eigentlich für die Fertigstellung der Dokumentation angesetzt war.

Die anderen Arbeitspakete konnten parallel bzw. zu einem späteren Zeitpunkt begonnen und ohne einen kritischen Pfad geplant werden. Die gesamte Planung der Arbeitspakete und des kritischen Pfades kann der folgenden Abbildung 96 entnommen werden.

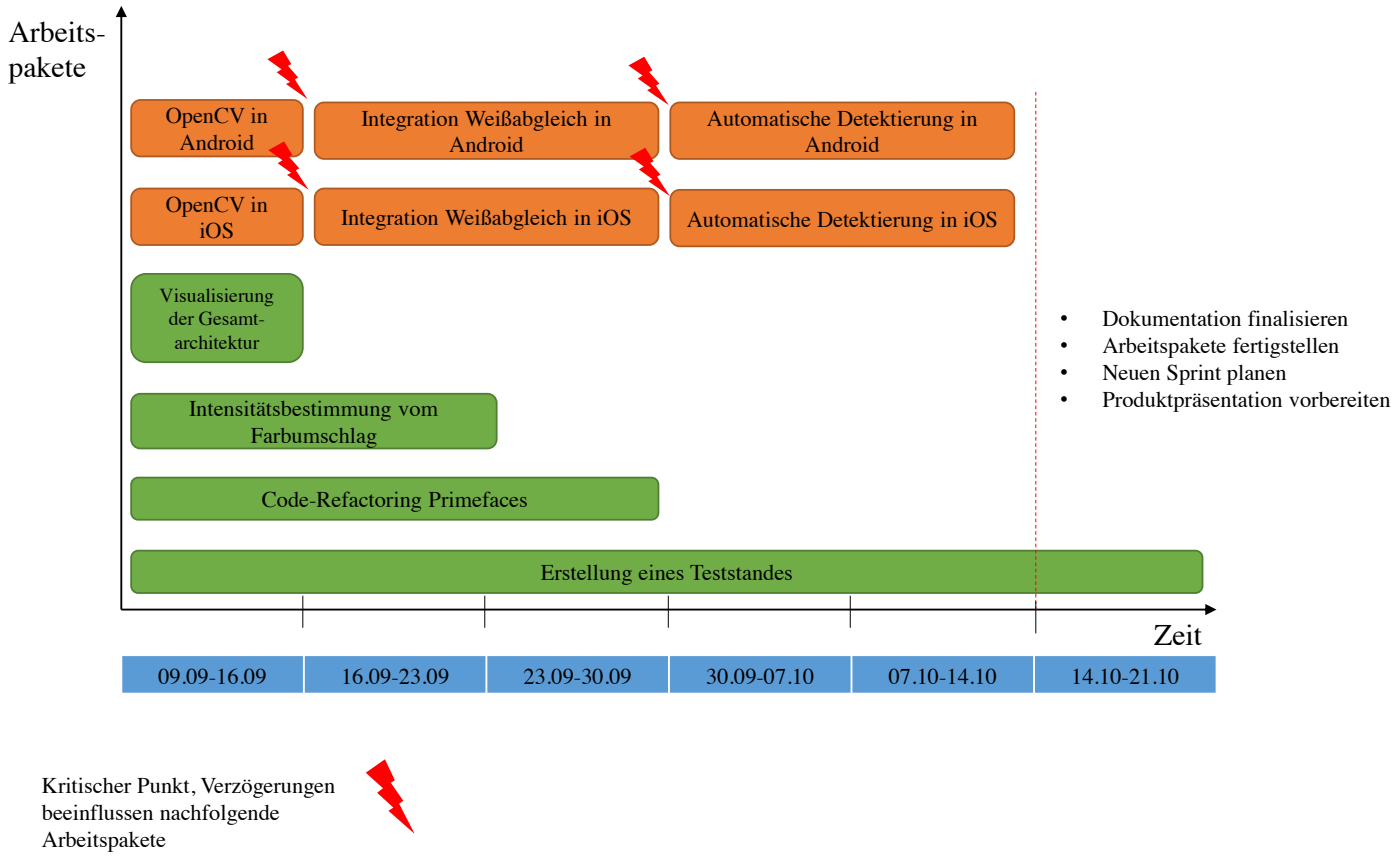



Abbildung 96: Ursprüngliche Projektplanung mit kritischem Pfad

Durch die Machbarkeitsstudie konnte ermittelt werden, dass es möglich ist, den selben C++ Quellcode unter iOS und Android einzubinden und zu verwenden. Daher wurde ab der dritten Woche damit begonnen, die Algorithmen in C++ zu übersetzen und diesen in die Apps zu integrieren. Aus diesem Sachverhalt ergaben sich folgende Änderungen, welche der Abbildung 97 entnommen werden können.

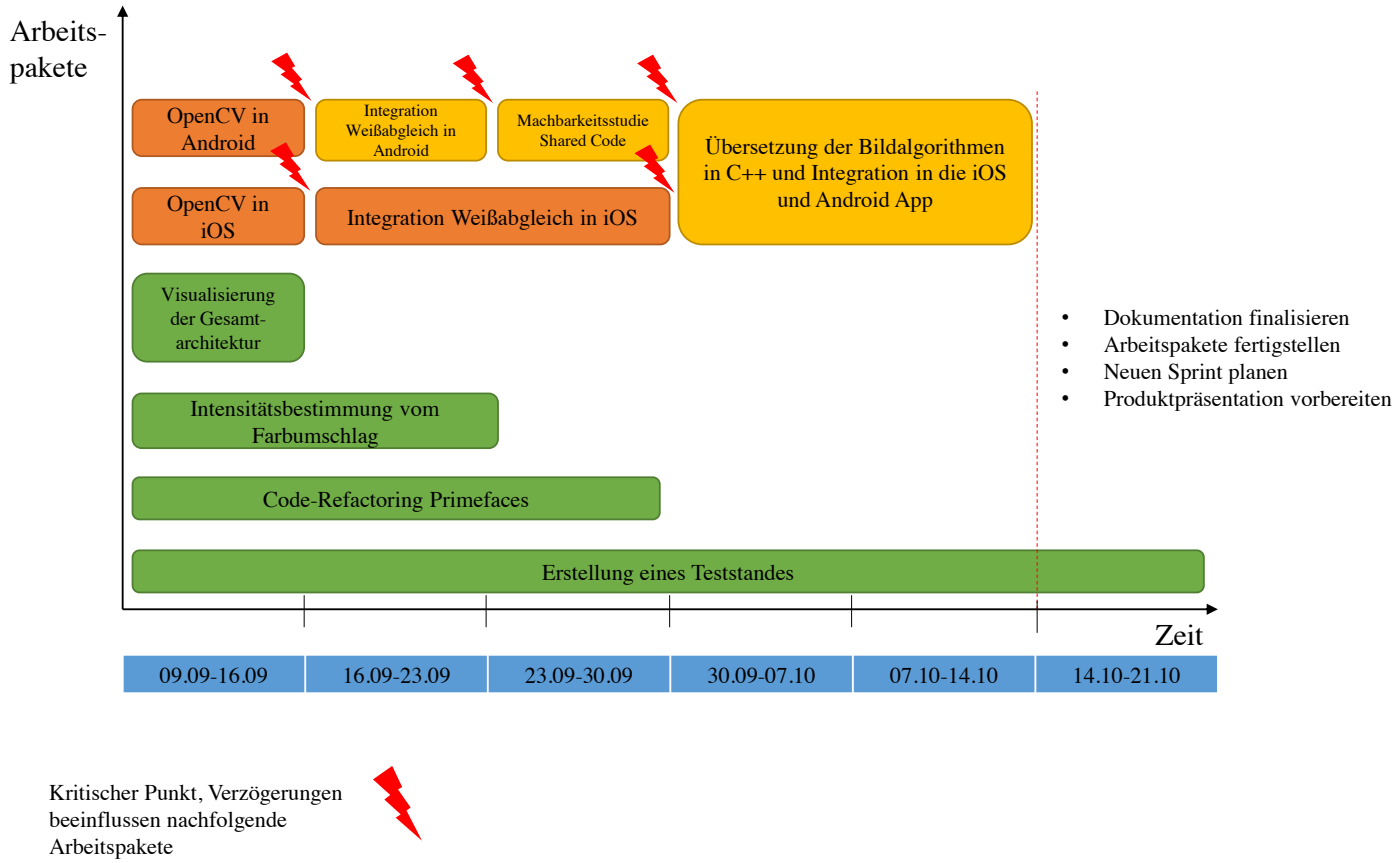



Abbildung 97: Anpassung der Projektplanung und des kritischen Pfades

Die Änderung in der Planung erforderte kontinuierliche Kommunikation zwischen den Teams der Bildverarbeitung und dem Projektmanagement, um die Algorithmen für beide Apps zu übersetzen, redundante Arbeit zu vermeiden und eine Integration in die Apps im zeitlich vorgegebenen Rahmen zu ermöglichen. Anhand dieser restrukturierten Planung konnte die Projektgruppe aufzeigen, dass ein agiles Vorgehen im Projekt sehr gut aufgenommen und umgesetzt wird. Spontane Änderungen wurden schnell und deutlich kommuniziert, neue Konzepte in Kleingruppen erarbeitet und im Arbeitsprinzip der Paarprogrammierung umgesetzt.

Zu Beginn des fünften Sprints konnten neun Teammitglieder für die Arbeitspakete eingeplant werden. Da einige Teammitglieder im Urlaub bzw. mit ihren Seminararbeiten beschäftigt waren, ergab sich folgende Ressourcenplanung:

**Daniel Wegmann** sechs Wochen Teststand, Projektplanung

**Raphael Kappes** sechs Wochen Teststand, Projektplanung

**Timo Schlömer** sechs Wochen Bildverarbeitung und Apps

**Christoph Ressel** fünf Wochen Bildverarbeitung und Apps, danach Einstieg in die Seminarphase

**Nicolas Koch** nach drei Wochen in den Sprint eingestiegen, dann Intensitätsdetektierung und Bildverarbeitung

**Kevin Sandermann** drei Wochen Refactoring Primefaces, danach Einstieg in die Seminarphase

**Sebastian Horwege** nach drei Wochen in den Sprint eingestiegen, dann Intensitätsdetektierung und Unterstützung beim Teststand

**Christian Sandmann** sechs Wochen, Bildverarbeitung und Apps

**Jan Philipp Stubbe** sechs Wochen, Bildverarbeitung und Apps

**Timo Raß** sechs Wochen Teststand, Projektplanung

**Danny Fonk** sechs Wochen Refactoring Primefaces, Visualisierung der Gesamtarchitektur, Planung der Optimierung und Qualitätssicherung vergangener und zukünftiger Dokumentationen

Im folgenden Kapitel werden die Arbeitspakete und deren Umsetzung genauer beschrieben. Dabei wird insbesondere auf die Vorgehensweise und die angefallenen Probleme eingegangen.

### 4.4.2 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete wie gewohnt detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.4.3 Gesamtarchitektur visualisieren

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Für vergangene und zukünftige Projekt-Sprints soll eine Visualisierung der Systemarchitektur erfolgen. Diese soll in die Dokumentationen und Abschlusspräsentationen einfließen und dem Kunden den Fortschrittsgrad aufzeigen.

- Arbeitspaketbeschreibung:
  Es existiert eine grafische Darstellung inklusive einer Beschreibung der gesamten Architektur und aller Technologien, die im MEDIC-Projekt verwendet werden. Für den Kunden soll eine Version dieser Darstellung vorliegen, welche den gleichen Inhalt in vereinfachter Form darstellt. Zudem ist eine Präsentation für den Kunden zu erstellen, welche bei der Produktpräsentation vorgestellt wird. Diese zeigt Ebenfalls die verwendeten Architekturen, Technologien und deren Zusammenspiel untereinander (grafisch).

  Es müssen keine genauen Klassendiagramme aus dem Quellcode generiert werden. Stattdessen ist aufzuzeigen, wo z.B. OpenCV verwendet wird, welche Technologien für die Kommunikation von Medizinern und Probanden gewählt wird und welche Technologien das Webfrontend und der Server verwenden. Eine grafische Illustrierung ist hier angebracht.

- Vorbedingungen:
  -

- Nebenbedingungen:
  -

- Nachbedingungen:
  Eine anpassbare Version der visualisierten Systemarchitektur ist vorhanden.

- Aufwand:
  Eine Woche

- Personen:
  - Danny Fonk

**4.4.3.1 Dokumentation** Zur Ergänzung der Produktpräsentation gegenüber dem Kunden, ist im Rahmen des Arbeitspaketes „Gesamtarchitektur visualisieren“ , eine grafische Darstellung der momentanen Systemarchitektur zu erstellen. Zusätzlich dazu ist die Präsentation für die am Ende von Sprint 5 anstehende Produktpräsentation zu erstellen, welche die erarbeiteten Ergebnisse für den Kunden verständlich aufzeigt.

**Ablauf** Zu Beginn des Arbeitspakets wurde die derzeitige Systemarchitektur anhand des aktuellen Stands des Servers, sowie auf Basis von vergangenen Arbeitspaket-Dokumentationen und Protokollen rekonstruiert, validiert und visualisiert (vgl. Abbildung 94 auf Seite 152).

Die dadurch entstandene Systemarchitektur dient als Grundlage für die nachfolgende Beschreibung der Systemarchitektur.

**Beschreibung der Systemarchitektur** Die nachfolgende Systembeschreibung ist eine zusammengefasste Beschreibung, aufbauend auf den Erläuterungen der Arbeitspaketdokumentationen, der jeweiligen Komponenten.

**PostgreSQL-Datenbank** Die PostgreSQL-Datenbank dient als Datenhaltungs-Instanz für sämtliche anfallende Daten wie bspw. die Testdaten, welche von mobilen Applikationen über die REST-Schnittstelle geliefert werden oder die Verwaltung der Benutzeraccounts der Weboberfläche. Siehe Abbildung 98 auf Seite 234.

**Hibernate** Hibernate ist ein ORM-Framework in der Sprache Java. ORM steht für **Object-relational mapping**, also objektrelationale Abbildung. Das Framework wird für die Datenverbindung vom Webfrontend hin zur PostgreSQL-Datenbank genutzt.

**Server** Der Server ist eine physikalische Komponenten, welche von der Projektgruppe aufgesetzt wurde und eigens gewartet wird. Auf dem Server befinden sich die Software für die kontinuierliche Integration (Jenkins), die Software für die Weboberfläche (Apache Tomcat), sowie das in dem Java-Framework Spring umgesetzte Backend samt REST-Schnittstelle, welche zur Kommunikation zwischen Server und mobiler Applikation genutzt wird.

**Jenkins** Jenkins ist eine Web-basierte Software, welche zur kontinuierlichen Integration von Komponenten eingesetzt wird. Im Umfeld der Projektgruppe Medic wird Jenkins derzeit dazu verwendet, in einem zeitlich festgesetztem Abstand die mobilen Anwendungen sowie die in PrimeFaces umgesetzte Weboberfläche zu kompilieren und auszuliefern.

**Weboberfläche** Unter dem Server-Frontend wird der dafür eingesetzte Apache Tomcat sowie die von der Projektgruppe Medic in PrimeFaces entwickelte Weboberfläche zusammengefasst.

**Apache Tomcat** Ein Apache Tomcat-Webserver ist ein quelloffener Webserver und Webcontainer, welcher es erlaubt Java-Anwendungen auszuführen. Diese Software wurde im Rahmen der Seminararbeit „Evaluation von Web Application Frameworks und Webserver-Software“ ausgewählt und wird seither im Rahmen der Projektgruppe eingesetzt.

**JSF-Implementierung in „PrimeFaces“** Die Weboberfläche wurde in Java, speziell in dem Web-Application-Framework PrimeFaces umgesetzt, welche ebenso wie der Webserver im Rahmen der Seminararbeit „Evaluation von Web Application Frameworks und Webserver-Software“ evaluiert wurde und derzeitig eingesetzt wird.

**Server-Backend** Das Server-Backend ist in dem Java-Framework Spring umgesetzt, wurde in Sprint 2 entwickelt und seitdem von der Projektgruppe als Server-Backend mit REST-Schnittstelle eingesetzt. Das Server-Backend wurde in einigen Sprints erweitert um aktuellen Anforderungen gerecht zu werden. Jedoch wird diese Komponente in einem zukünftigem Sprint durch eine andere Komponente ersetzt (vgl. Abbildung 99 auf der nächsten Seite).

**mobile Applikationen** Die grundsätzliche Anforderung setzt voraus, dass die Auswertung der Krankheitstests auf Smartphones durchgeführt wird. Dazu wurde im ersten Sprint eine Marktanalyse durchgeführt, aus welcher hervorging, dass mit einer Umsetzung für die Betriebssysteme **iOS** und **Android** eine ausreichend große Marktabdeckung umgesetzt werden kann.

**OpenCV-Implementierung mit geteiltem Code** Um eine bessere Wartbarkeit des Algorithmus für die Bildanalyse zu gewährleisten, wird der Code für die Bildanalyse mit einem geteilten Quellcode in der Programmiersprache C++ umgesetzt. Diesen geteilten Code können beide Apps aufrufen.

**Android-Applikation** Umsetzung der Anforderungen an die mobile Applikation für das Betriebssystem Android.

**iOS-Applikation** Umsetzung der Anforderungen an die mobile Applikation für das Betriebssystem iOS.

**Testdatengenerator** *„Der Testdatengenerator [...] dient als Werkzeug, um die Einhaltung nichtfunktionaler Anforderungen zu gewährleisten. Der Testdatengenerator erstellt automatisiert durchgeführte Krankheitstests und sendet diese an den Server. Diese angelegten Tests können anschließend auf der Weboberfläche und der Heatmap betrachtet werden. Somit stellt der Testdatengenerator eine Möglichkeit dar, die Komponenten Server, Serverschnittstelle, Datenbank, Weboberfläche und Heatmap jederzeit zu testen.“*

Quelle: vgl. Sprintdokumentation 3 - Testdatengenerator

**Teststreifengenerator** *„Der Teststreifengenerator [...] dient als Werkzeug, um Teststreifen zu erzeugen und auszudrucken. Mit Hilfe dieser kann die Entwicklung des Bildverarbeitungsalgorithmus in sofern unterstützt werden, als dass der Algorithmus auf Basis der ausgedruckten Teststreifen evaluiert werden kann.“*

Quelle: vgl. Sprintdokumentation 4 - Teststreifengenerator

Im Zuge dieses Arbeitspaketes wurden ebenfalls für die vergangenen Sprints Systemarchitekturen angefertigt, welche die Systemarchitektur am Ende des jeweiligen Sprints zeigen. Des weiteren ist eine angestrebte Systemarchitektur nach Sprint 6 angefertigt worden, um eine verbesserte Planung zu ermöglichen (vgl. Abbildung 99).

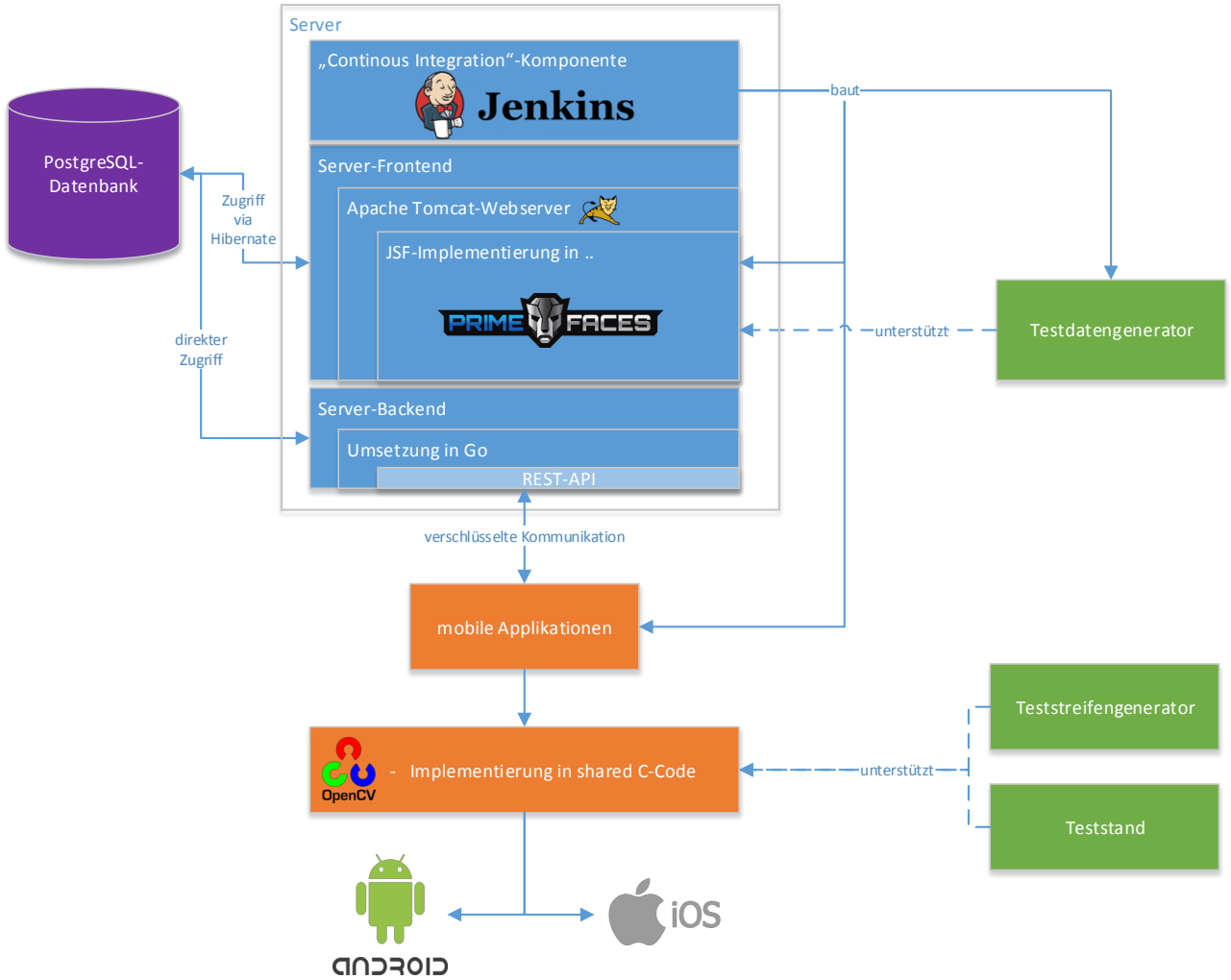



Abbildung 99: zukünftige Systemarchitektur

Zukünftig ist eine Verschlüsselung der Kommunikation zwischen den mobilen Applikationen sowie dem Server zu implementieren. Außerdem wird das Server-Backend, welches derzeitig in dem Java-Framework **Spring** umgesetzt ist, durch ein Server-Backend ersetzt, welches in der Programmiersprache **Go** geschrieben werden wird.

### Technologien

**Microsoft Visio** Die Visualisierungen der Systemarchitekturen wurden mit dem proprietärem Visualisierungsprogramm Visio der Firma Microsoft umgesetzt.

**Systembeschreibung** Die Grafiken wurden in Microsoft Visio erstellt.

**Evaluation** Die entstandenen Visualisierungen wurden nach Erstellung mit den in den Dokumentationen und Protokollen niedergeschriebenen Vorgehensweisen abgeglichen und validiert.

**Parameter** Die Ergebnisse des Arbeitspaketes werden von keinen Parametern eingeschränkt.

**Fazit** Das Arbeitspaket „Visualisierung der Gesamtarchitektur“ konnte innerhalb der geforderten Frist von einer Woche abgeschlossen werden. Die dabei entstandenen Visualisierungen sowie die zugehörige Präsentation kann bei Produktpräsentationen verwendet werden. Ebenfalls ist denkbar, dass die dabei entstandenen Visualisierungen unterstützend bei den zukünftigen Sprintplanungen wirken und zur Erstellung einer sog. Roadmap genutzt werden können. Außerdem wurde zusätzlich ein Entity-Relationship-Diagramm (vgl. Abbildung 98 auf Seite 234) angefertigt, wodurch weitere Erkenntnisse gewonnen werden konnten.

**Ausblick** Im Laufe der Umstellung von Technologien, sollte überprüft werden ob die hier geschilderte „zukünftige Systemarchitektur“(vgl. Abbildung 99 auf der vorherigen Seite) weiterhin gültig ist. Sofern dies nicht der Fall ist, sollte die Dokumentation umgehend angepasst werden, um einen dauerhaft gültigen Stand der Visualisierung der Systemarchitektur zu gewährleisten.

Des Weiteren ist beim Erstellen des ER-Diagramms aufgefallen, dass hier weitere Optimierungen erfolgen können. Ebenfalls sind dort noch Altlasten vorhanden, welche es zu Entfernen gilt.

### 4.4.4 Integration von OpenCV in Android

- Priorität: Kritisch - Dieses Arbeitspaket ist die Basis für die weitere Integration der Bildverarbeitung auf den mobilen Endgeräten.
- Motivation und Nutzen des Arbeitspaketes:
  Um die Anforderung der automatischen Detektierung des Farbumschlagbereiches umsetzten zu können, wird zukünftig die Bildverarbeitungsbibliothek OpenCV eingesetzt.
- Arbeitspaketbeschreibung:
  In die bestehende oder in eine neue Android-App ist das Framework OpenCV integriert. Die App ist zudem so vorbereitet, dass im weiteren Verlauf des fünften Sprints die Algorithmen für den Weißabgleich und die automatische Detektierung des Farbumschlags integriert werden können, ohne dass Integrationsfehler des OpenCV Frameworks auftreten.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Das Framework OpenCV wurde erfolgreich in die Android-App integriert und ausführlich auf seine Lauffähigkeit getestet. Einfache Algorithmen, die ein Testbild manipulieren, können auf dem mobilen Endgeräten ausgeführt werden.

- Aufwand:
  Eine Woche
- Personen:
  - Timo Schlömer
  - Christoph Ressel

#### 4.4.4.1 Dokumentation

**Ablauf** Es wurde entschieden die App für das Android Betriebssystem von Grund auf neu zu schreiben, da der Code der Prototyp-App nicht mehr den Code-Standards des Projekts entspricht und eine Aufbesserung des bestehenden Codes mehr Zeit in Anspruch genommen hätte. Die neue Version der App enthält die bereits bekannte Funktion der Prototyp-App, mit der die aktuellen Zustände einer Liste lokal durchgeführter Tests vom Server heruntergeladen und angezeigt werden können. Auch ist die Einbindung der OpenCV Bibliothek zur Bildverarbeitung wiederhergestellt, nun jedoch mit Version 3.0.0, anstelle der vorher verwendeten Version 2.4.11. Das Aufnehmen eines Fotos und das Analysieren eines Tests wurde vorerst nicht wieder integriert, da diese Funktionen in späteren Arbeitspaketen auf eine neue Weise implementiert werden sollen. Die Abbildungen 100 bis 103 auf der nächsten Seite zeigen die neue Benutzeroberfläche der App.

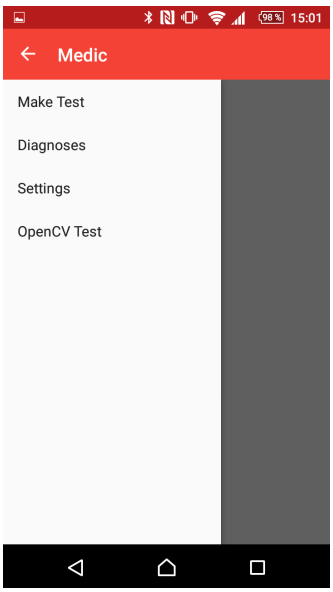



Abbildung 100: Drawer Navigation in Android

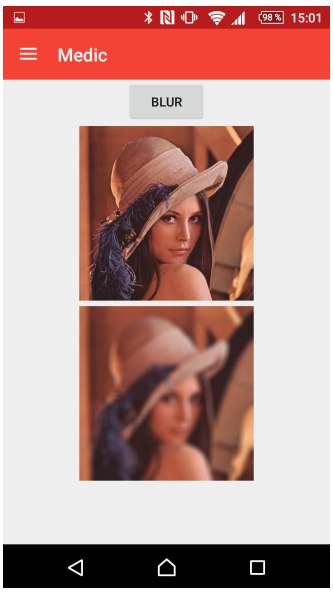



Abbildung 101: OpenCV Beispiel in Android

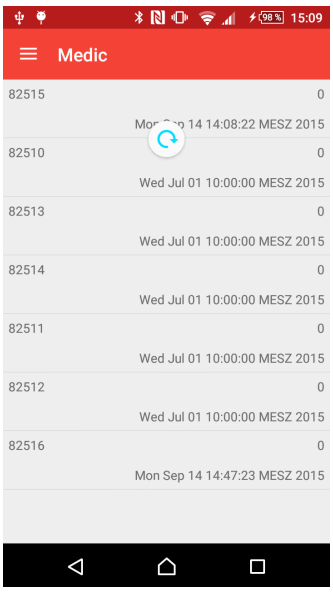



Abbildung 102: Testliste mit Pull-to-Refresh in Android

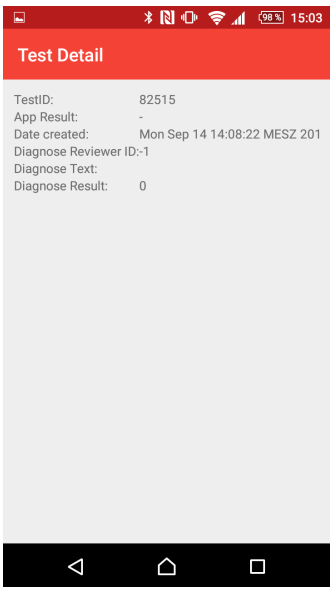



Abbildung 103: Detailierte Testansicht in Android

**Technologien**

**Android Support Library** Die Support Library von Google stellt aktuelle Systemfunktionen für ältere Android Versionen bereit, um Kompatibilität zwischen den verschiedenen Versionen des Betriebssystems zu gewährleisten.

**ION** Eine Bibliothek, welche Netzwerkzugriffe über HTTP vereinfacht.

**Gson** Eine Bibliothek um Java Objekte in JSON zu überführen.

**OpenCV** Bibliothek für Bildverarbeitung.

**Systembeschreibung** Als Entwicklungsumgebung wurde Android Studio in Version 1.3.2 verwendet. Diese Version unterstützt die Integration und das Debugging von nativen C/C++ Code. Die OpenCV-Java-Wrapper-Bibliothek und die dazugehörigen nativen C/C++ Quelldateien sind innerhalb des Projektordners hinterlegt und werden (wie der Quellcode der App) mit dem Versionsverwaltungssystem verwaltet.

**Evaluation** Die App wurde auf einem „Sony Xperia Z3 Compact“ , einem „Sony Xperia E1“ , einem „LG Nexus 4“ und einem „LG Bello“ auf korrekte Funktion getestet. Es wurden dabei häufige Fehlerquellen simuliert, wie das Drehen des Geräts, das Pausieren der App und das Beenden der App. Die Korrektheit der OpenCV Integration wurde mithilfe einer beispielhaften Bildverarbeitungsoperation überprüft.

**Parameter** Die Funktion der App wird von keinen Parametern eingeschränkt.

**Fazit** Der Quellcode der App ist nun in einem guten Zustand und folgt den Standards für Code-Qualität des Projekts. Die Integration der aktuellsten OpenCV Bibliothek war erfolgreich und es konnten beispielhafte Bildverarbeitungsoperationen durchgeführt werden. Die App kann nun mit den Algorithmen für Weißabgleich und Farbumschlagsdetektion erweitert werden.

**Ausblick** Momentan werden von den lokal durchgeführten Tests lediglich die IDs persistent in den „Preferences“ gespeichert. Zukünftig sollte stattdessen eine lokale Datenbank genutzt werden, um die durchgeführten Tests **komplett** abzuspeichern. Hierdurch wird Bandbreite gespart, da zum Anzeigen der Ergebnisse die Datenbank lokal abgefragt werden kann und keine Anfrage an den Server gestellt werden muss.

### 4.4.5 Integration von OpenCV in iOS

- Priorität: Kritisch - Dieses Arbeitspaket ist die Basis für alle weiteren Integration der Bildverarbeitung auf den mobilen Endgeräten.
- Motivation und Nutzen des Arbeitspaketes:
  Um die Anforderung der automatischen Detektierung des Farbumschlagbereiches umsetzten zu können, wird zukünftig die Bildverarbeitungsbibliothek OpenCV eingesetzt.
- Arbeitspaketbeschreibung:
  In die bestehende oder in eine neue iOS-App ist das Framework OpenCV integriert bzw. reaktiviert. Die App ist zudem so präpariert, dass im weiteren Verlauf des fünften Sprints die Algorithmen für den Weißabgleich und die automatische Detektierung des Farbumschlags integriert werden können, ohne dass Integrationsfehler des OpenCV Frameworks auftreten.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Das Framework OpenCV wurde erfolgreich in die iOS-App integriert und ausführlich auf seine Lauffähigkeit getestet. Einfache Algorithmen, die ein Testbild manipulieren, können auf dem mobilen Endgeräten ausgeführt werden.
- Aufwand:
  Eine Woche
- Personen:
  - Christian Sandmann
  - Jan Philipp Stubbe

**4.4.5.1 Dokumentation** Ziel dieses Arbeitspaketes war es, das Programm „OpenCV“, welches für die Bildverarbeitung benötigt wird, in die iOS-App zu integrieren. OpenCV soll ohne Probleme genutzt werden können, um eine einwandfreie Bildverarbeitung auf dem Endgerät vorzubereiten.

**Ablauf** Als Grundlage wurde die iOS-Prototyp-App genommen, die in den vorherigen Sprints entwickelt wurde. In dieser war die Bibliothek für OpenCV schon integriert, sodass im Zuge dieses Arbeitspakets die Umstrukturierung des bestehenden Codes vorgenommen wurde. Um zu zeigen, dass auf die Funktionen von OpenCV zugegriffen werden kann, wurde eine Testfunktion geschrieben, die einen Gauss-Filter über das aufgenommene Foto legt. Bei der Implementierung war darauf zu achten, dass sich der Code, der OpenCV aufruft, in einer **Objective-C++** Klasse befindet.

Für die neue Struktur der Applikation wurde zunächst ein neues Projekt erstellt, welches auf den Klassen des bisherigen Projektes basiert. Auch wurde der Titel der App aktualisiert und lautet nun:„IOSAppMedic“. Im Anschluss wurde eine Klasse für die Bildverarbeitung erstellt, die das

Bild, welches mit der Kamera aufgenommen wurde, in ein OpenCV konformes Format konvertiert, sodass ein Gauss-Filter angewendet werden kann. Das bearbeitete Bild wird dann in ein von iOS darstellbares Bild transformiert und wird dem Nutzer angezeigt.

Zusätzlich wurden die Namen der Klassen und Variablen angepasst, sodass diese aussagekräftig sind und den Coding-Styles der Projektgruppe entsprechen. Auch wurden die bereits bestehenden Klassen und Methoden kommentiert, um die Qualität des Quellcodes weiter zu steigern.

**Technologien**

**Xcode** Entwicklungsumgebung, die für iOS benötigt wird

**OpenCV** Bibliothek, die Funktionen für Bildverarbeitung bereitstellt

**Systembeschreibung** Das System wurde auf einem Mac Mini entwickelt. Als Entwicklungsumgebung wurde Xcode in der Version 7.0 verwendet. Das Endgerät zum Testen der Funktionalität der Applikation war ein iPad Mini mit der iOS Version 9.0.1.

**Evaluation** Durch die Implementierung des Gauss-Filters, konnte evaluiert werden, dass OpenCV integriert wurde und auf die Funktionalitäten zugegriffen werden kann.

**Parameter** Die Funktion der Applikation kann gewährleistet werden, wenn iOS ab der Version 9 verwendet wird.

**Fazit** Durch die Integration von OpenCV in der iOS-App können die weiteren Arbeitspakete, die auf der Bildverarbeitung basieren, begonnen werden. Durch das Anpassen des bisherigen Codes, ist dieser verständlicher und übersichtlicher. Es gibt nun eine eigene Klasse für die Bildverarbeitung, aussagekräftigere Variablen und Methoden, sowie eine Dokumentation des Codes.
Das Arbeitspaket konnte in der vorgegebenen Zeit von einer Woche beendet werden. Die Deadline wurde gehalten, die weiteren Arbeitspakete, die auf diesem aufbauen, können ohne Verzögerungen gestartet werden.

**Ausblick** Um in Zukunft eine stabile Anwendung bereitstellen zu können, müssen die einzelnen Funktionen der App noch intensiver getestet werden. Dazu gehört beispielsweise das Minimieren der Anwendung während die Bildanalyse durchgeführt wird.

### 4.4.6 Integration des Weißabgleichs in die Android-App

- Priorität: Kritisch - Kommt es bei diesem Arbeitspaket zu Verzögerungen, wird die Produktpräsentation für den Kunden davon betroffen. Das Produkt kann eventuell nicht im vollen Umfang präsentiert werden.

- Motivation und Nutzen des Arbeitspaketes:
  Der zuvor entwickelte Weißabgleich soll in die Android-Applikation integriert werden, um das Bild für die Auswertung des Teststreifens zu verbessern.

- Arbeitspaketbeschreibung:
  Der in Matlab entwickelte Algorithmus zum Weißabgleich, eine optimierte Form von diesem oder ein alternativer Algorithmus zum Weißabgleich ist in die Android-App integriert. Das optimierte Bild ist anschließend so zwischengespeichert, dass es von weiteren Arbeitspaketen genutzt werden kann (u.a. Detektierung der Farbumschlagsfelder). Zudem kann das weißabgeglichene Bild auf dem Endgerät angezeigt werden.

- Vorbedingungen:
  Das Arbeitspaket „Integration von OpenCV in Android“ (Kapitel 4.4.4 auf Seite 162) muss abgeschlossen sein.
- Nebenbedingungen:
  Es können weitere alternative Weißabgleichs-Algorithmen getestet werden.
- Nachbedingungen:
  Der Quellcode zum Weißabgleich ist in die Android-App integriert und kann auf einem Bild durchgeführt werden. Das Bild wird nach dem Weißabgleich angezeigt. Zudem wird das optimierte Bild für weitere Arbeitspakete zur Verfügung gestellt.
- Aufwand:
  Zwei Wochen
- Personen:
  - Timo Schlömer
  - Christoph Ressel

**4.4.6.1 Dokumentation** Ziel dieses Arbeitspaketes ist es, in Android einen performanten und zuverlässigen Algorithmus zum Weißabgleich zu implementieren. Als Grundlage kann dafür der im Arbeitspaket „Weißabgleich“ in MatLab entwickelte Algorithmus genutzt werden (Arbeitspaket aus Sprint 4, Seite 54).

**Ablauf** Auf Grund der Ergebnisse des Arbeitspakets „Weißabgleich“ aus Sprint vier wurde versucht den in MatLab prototypisch implementierten Algorithmus in die Android-App zu integrieren. Ab Version 3.0.0 ist in OpenCV bereits ein Weißabgleich implementiert, jedoch steht dieser nicht in der finalen Version zur Verfügung und nicht in der offiziell veröffentlichten Fassung. Somit ist diese Funktion auch noch nicht in der Android Version der Bibliothek nutzbar.

**Technologien** Es wurden keine besonderen Technologien genutzt.

**Systembeschreibung** Es gelten die gleichen Bedingungen wie für das Arbeitspaket „Integration von OpenCV in Android“ (Kapitel 4.4.4 auf Seite 162).

**Evaluation** Zum Evaluieren der Performanz wurde der Algorithmus mit verschiedenen Geräten auf ein Beispielbild angewandt.
Laufzeiten des Algorithmus auf unterschiedlichen Geräten (Mittelwert aus zehn Durchführungen pro Gerät, Bildgröße: 2 Megapixel):
LG Bello: 2285 ms
Sony Xperia Z3: 1010 ms
Sony Xperia E1: 2703 ms

Das Ergebnisbild war nach jeder Durchführung auf jedem Gerät das gleiche. In Abbildung 104 auf der nächsten Seite ist das originale Bild dargestellt, welches in den Algorithmus eingegeben wurde. Das Papier auf dem Bild wurde mit einer Lichtquelle erhellt, welche einen geringen Kelvin-Wert hat und wirkt daher unnatürlich gelb. In Abbildung 105 auf der nächsten Seite ist das Bild nach dem Durchlauf des Algorithmus zu sehen. Der Gelbstich ist entfernt und das Bild wirkt nun weißer.

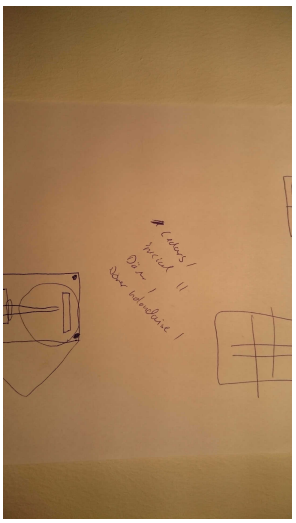


Abbildung 104: Das originale Bild mit Gelbstich

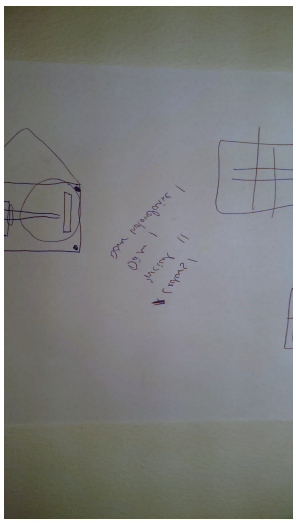


Abbildung 105: Das Bild nach dem Weißabgleich

**Parameter** Der Algorithmus wird durch keine Parameter eingeschränkt.

**Fazit** Der Algorithmus konnte in kurzer Zeit in Java überführt und somit erfolgreich in Android integriert werden. Die Verarbeitung eines hochaufgelösten Bildes ist in einer angemessenen Zeit (ca. eine Sekunde) durchführbar.

**Ausblick** Sobald der Weißabgleich in OpenCV offiziell zur Verfügung steht, kann dieser getestet und mit der momentanen Implementierung verglichen werden. Falls die Immplementierung von OpenCV besser ist, kann der momentane Stand damit ersetzt werden.

### 4.4.7 Integration des Weißabgleichs in die iOS-App

- Priorität: Kritisch - Kommt es bei diesem Arbeitspaket zu Verzögerungen, wird die Produktpräsentation für den Kunden davon betroffen. Das Produkt kann eventuell nicht im vollen Umfang präsentiert werden.

- Motivation und Nutzen des Arbeitspaketes:
  Der zuvor entwickelte Weißabgleich soll in die iOS-Applikation integriert werden, um das Bild für die Auswertung des Teststreifens zu verbessern.

- Arbeitspaketbeschreibung:
  Der in Matlab entwickelte Algorithmus zum Weißabgleich, eine optimierte Form von diesem oder ein alternativer Algorithmus zum Weißabgleich ist in die iOS-App integriert. Das optimierte Bild ist anschließend so gespeichert, dass es von weiteren Arbeitspaketen genutzt werden kann (u.a. Detektierung der Farbumschlagsfelder). Zudem kann das weiß-abgeglichene Bild auf dem Endgerät angezeigt werden.

- Vorbedingungen:
  Das Arbeitspaket „Integration von Open CV in iOS“ (Kapitel 4.4.5 auf Seite 165) muss abgeschlossen sein.

- Nebenbedingungen:
  Es können weitere alternative Weißabgleichs-Algorithmen getestet werden.

- Nachbedingungen:
  Der Quellcode zum Weißabgleich ist in die iOS-App integriert und kann auf einem Bild durchgeführt werden. Das Bild wird nach dem Weißabgleich angezeigt. Zudem wird das optimierte Bild für weitere Arbeitspakete zur Verfügung gestellt.

- Aufwand:
  Zwei Wochen

- Personen:

  - Christian Sandmann
  - Jan Philipp Stubbe

**4.4.7.1 Dokumentation** Ziel dieses Arbeitspakets war es, den Weißabgleich, welcher in Sprint 4 in Matlab entwickelt wurde und zur korrekten Voranalyse entscheidend ist, in die iOS-App zu integrieren.

**Ablauf** Zunächst wurde versucht die Funktion für den Weißabgleich zu nutzen, die von dem Modul **xphoto** aus OpenCV bereitgestellt wird. Doch aufgrund von Problemen beim Erstellen der erforderlichen Version von OpenCV, wurde dieses Paket nicht genutzt.

Da der Weißabgleich schon in der Android-App im Zuge des Arbeitspakets „Integration des Weißabgleichs in die Android-App“(Kapitel 4.4.6.1 auf Seite 167) implementiert wurde, konnte die Grundlage des Algorithmus übernommen und in Objective C überführt werden. Weiter wurde OpenCV auf die Version 3.0 aktualisiert, damit die iOS-App auf dem gleichen Stand ist wie die Android-App.
Für den Weißabgleich werden zunächst die durchschnittlichen RGB-Werte für jeden Pixel des Bildes berechnet, auf dessen Basis die Faktoren für die Anpassung berechnet werden können. Dafür wird zunächst der Gesamtdurchschnitt aus den roten, grünen und blauen Farbwerten der Pixel des Bildes errechnet. Für die Anpassung wird dieser Gesamtdurchschnitt durch den Durchschnitt des jeweiligen Kanals dividiert (z.B. $alpha = Gesamtdurchschnitt/Durchschnittrot$). Diese Vorfaktoren werden mit den einzelnen Farbkanälen für jeden Pixel multipliziert. Das Resultat sieht man in den Abbildungen 106 auf der nächsten Seite (ohne Weißabgleich) und 107 auf der nächsten Seite (nach dem Weißabgleich).

Um die iOS-App für die folgenden Arbeitspakete zur Bildverarbeitung vorzubereiten, wurde die QR-Code Erkennung entfernt, die durch den neuen Ansatz der „Augmented Reality“ nicht mehr benötigt wird. Das Bild kann von dem Benutzer manuell gemacht werden.

Im Zuge des Arbeitspaketes wurden folgende Fehler in der App behoben:

- Wenn ein Foto mit der Kamera im Querformat aufgenommen wurde, wurden die Fotos um 90 Grad verdreht angezeigt.

- Beim Wechsel vom Hochformat ins Querformat nur ein Teil des Bildschirms mit dem Vorschaubild ausgefüllt.

- Das Foto wurde immer im Hochformat abgespeichert.

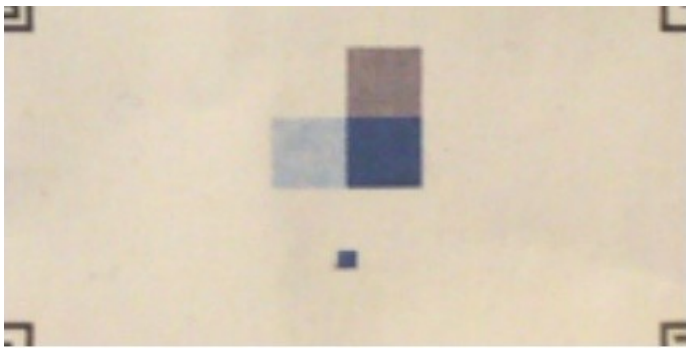


Abbildung 106: Bild vor dem Weißabgleich

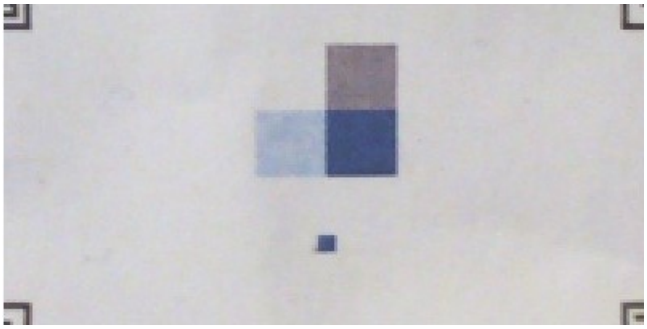


Abbildung 107: Bild bearbeitet mit dem Weißabgleich

**Technologien** Es wurden die gleichen Technologien verwendet wie im Arbeitspaket „Integration von OpenCV in iOS“ (Kapitel 4.4.5.1 auf Seite 165).

**Systembeschreibung** Das System entspricht dem Arbeitspaket „Integration von OpenCV in iOS“.

**Evaluation** Die Evaluation fand anhand von Testbildern statt, ein Beispiel ist in den Abbildungen 106 und 107 zu sehen.
Die Laufzeit für den Weißabgleich wurde auf dem iPad mini getestet. Der Algorithmus dauert im Durchschnitt 243 ms bei 13 durchgeführten Tests.

**Parameter** Die Dauer des Weißabgleichs hängt von der Auflösung des gemachten Fotos ab, da der Algorithmus das Bild Pixel für Pixel durchgeht. Je höher die Auflösung des Bildes ist, desto länger dauert der Algorithmus.

**Fazit** Durch die Vorarbeit des Arbeitspakets „Integration des Weißabgleichs in die Android-App“ konnte die benötigte Zeit für dieses Arbeitspaket reduziert werden, sodass es in einer Woche beendet werden konnte. Die iOS-App wurde um die Funktion mit dem Weißabgleich ergänzt und es blieb Zeit einige Fehler zu beheben.

**Ausblick** Sobald der Weißabgleich in OpenCV offiziell zur Verfügung steht, kann dieser getestet und mit der momentanen Implementierung verglichen werden. Falls die Implementierung von OpenCV besser ist, kann der momentane Stand damit ersetzt werden.

### 4.4.8 Integration der automatischen Detektierung des Farbumschlags in die Android-App

- Priorität: Kritisch - Kommt es bei diesem Arbeitspaket zu Verzögerungen, wird das Arbeitspaket „Integration des Weißabgleichs in Android“ (Kapitel 4.4.6 auf Seite 166) davon betroffen und kann erst später begonnen werden.
- Motivation und Nutzen des Arbeitspaketes:
  Damit die Applikation eine Aussage zur Krankheit treffen kann, muss die automatische Detektierung des Farbumschlages in diese integriert werden.
- Arbeitspaketbeschreibung:
  Die Android-App ist mit dem Algorithmus zur automatischen Detektion der Farbumschlagfelder erweitert. Dazu kann der bereits bestehende Algorithmus aus dem Arbeitspaket „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ aus Sprint 4 (Seite 62) herangezogen werden. Die ausgeschnittenen Felder werden zwischengespeichert (zur weiteren Verarbeitung im Quellcode / Bildverarbeitung) und können zudem auf dem Smartphone ausgegeben und angezeigt werden (für den Kunden interessant für Produktpräsentation am Ende von Sprint 5).
- Vorbedingungen:
  Das Arbeitspaket „Integration des Weißabgleichs in die Android-App“ (Kapitel 4.4.6 auf Seite 166) ist erfolgreich abgeschlossen.
- Nebenbedingungen:
  Für die Detektierung der Felder muss nicht zwangsweise der bestehende Algorithmus gewählt werden. Wenn eine reibungslose Integration schnell erfolgen kann, können alternative Algorithmen und Verfahren getestet werden.
- Nachbedingungen:
  Der Algorithmus ist erfolgreich integriert und weißt eine Robustheit von 70% auf, sodass bei sieben von zehn Aufnahmen die Felder erfolgreich erkannt und ausgeschnitten werden können.
- Aufwand:
  Zwei Wochen
- Personen:
  - Timo Schlömer
  - Christoph Ressel

**4.4.8.1 Dokumentation** Nach diesem Arbeitspaket soll die Android-App in der Lage sein, den Bereich des Farbumschlags und des Referenzfeldes auf dem Bild der Kamera zu lokalisieren und diesen Bereich in eine normalisierte Form zu transformieren. Dafür werden die Ergebnisse des Arbeitspakets „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ (Kapitel 4.3.16.1 auf Seite 146) genutzt.

**Ablauf** Im Arbeitspaket „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ existiert bereits ein Algorithmus, welcher die Anforderungen dieses Arbeitspaketes erfüllt. Nach Analyse des Quelltextes wurde entschieden, den Code nicht direkt in Java zu überführen, sondern äquivalente Funktionen aus der OpenCV Bibliothek zu nutzen um zum selben Ergebnis zu gelangen. Es stellte sich heraus, dass große Teile des bestehenden Codes durch Funktionen der OpenCV Bibliothek ersetzt werden konnten, wodurch der eigene Code auf eine überschaubare Menge reduziert wurde.

Der Ablauf des Algorithmus ist nun wie folgt:

1. Das aktuelle Bild der Kamera wird aus dem **YUV N21** Farbraum[23] in den **BGRA** Farbraum[24] konvertiert.

2. Mit Hilfe des OTSU-Methode Algorithmus[25] wird aus dem Bild ein Binärbild erzeugt.

3. Der SimpleBlob Algorithmus[26] wird durchgeführt um Blobs (wichtige Punkte) im Bild zu finden. Durch die Konvertierung in ein Binärbild und eine gute Parameterwahl für den Simpleblob Algorithmus werden die Marker gut erkannt.

4. Für jeden gefundenen Blob wird überprüft, ob dieser ein Marker ist:

   (a) Es wird eine Kontur-Detektion durchgeführt, wodurch die Begrenzung des Markers gefunden wird (sofern der aktuelle Blob ein Marker ist).

   (b) Mit Hilfe der Konturpunkte, die OpenCV zurück gibt, werden die Eckpunkte des Markers ermittelt. Dafür wird für jeden Eckpunkt des Blobs der Konturpunkt gesucht, welcher am weitesten entfernt liegt. Dieser Konturpunkt ist ein Eckpunkt der schwarzen Umrandung des potentiellen Markers.

   (c) Mit Hilfe der gefundenen Marker-Eckpunkte kann eine Transformation durchgeführt werden, wodurch der Marker in eine normalisierte Form gebracht werden kann.

   (d) Der Inhalt des Markers wird dekodiert und mit den bekannten Werten von Markern verglichen. Für jeden Marker liegt der kodierte Wert in 0, 90, 180 und 270 Grad Rotation vor. Durch die Transformierung in ein quadratisches Bild liegt immer eine von diesen Rotationen vor.

   (e) Stimmt der Wert des aktuellen (potentiellen) Markers mit einem der bekannten Werte überein, wird der Marker als ein korrekter Marker gespeichert.

5. Sind alle vier Marker gleichzeitig gefunden, werden die Mittelpunkte dieser genutzt, um das gesamte Bild in die normalisierte Form zu transformieren und den Bereich zwischen den Markern auszuschneiden. Dieser Bereich enthält den Farbumschlag und das Referenzfeld.

6. Der Farbumschlag und das Referenzfeld befinden sich immer an der jeweils exakt gleichen Position im transformierten Bild und können somit pixelgenau ausgeschnitten werden. Beide Bereiche werden separat als Ergebnis auf dem Bildschirm dargestellt.

Im Verlauf des Arbeitspakets war es nötig, den Teststreifengenerator zu modifizieren, um Marker auf den Teststreifen abzubilden, welche vom Algorithmus konsistent erkannt werden können.

In dem Treffen mit den anderen Universitäten wurde empfohlen, das integrierte Licht der Smartphones bei der Detektierung zu aktivieren. Dies konnte eine deutliche Verbesserung der Detektionsrate bewirken.

In Abbildung 108 auf Seite 235 ist die erfolgreiche Detektierung der Marker auf einem Smartphone (LG Nexus 4) zu sehen. Links oben ist der transformierte Bildbereich dargestellt auf dem durch grüne Grenzen die auszuwertenden Areale gekennzeichnet sind.

#### Technologien

**OpenCV** Bibliothek für Bildverarbeitung.

[23] Das YUV-Farbmodel: `https://de.wikipedia.org/wiki/YUV-Farbmodell`

[24] Das BGRA-Farbmodell: `https://en.wikipedia.org/wiki/RGBA_color_space#ARGB`

[25] Das Schwellwertverfahren von Otsu: `https://de.wikipedia.org/wiki/Schwellenwertverfahren#Verfahren_von_Otsu`

[26] Der Simpleblob Detektor: `http://docs.opencv.org/master/d0/d7a/classcv_1_1SimpleBlobDetector.html`

**Systembeschreibung** Es gelten die gleichen Bedingungen wie für das Arbeitspaket „Integration von OpenCV in Android“ in Kapitel 4.4.4 auf Seite 162.

**Evaluation** Vorherige Versionen des Teststreifens waren mit Markern ausgestattet, welche 6x6 Millimeter groß waren, dies wurde jedoch im Verlauf des Arbeitspaketes geändert, so dass die Marker nur noch 3x3 Millimeter groß sind. Diese Änderung wurde aufgrund der obigen neuen Informationen durchgeführt.

Während die alten Marker von allen Geräten robust erkannt werden konnten, war dies mit der neuen Größe nicht der Fall, wodurch es nötig war einen Test mit präzise eingestellten Parametern durchzuführen. Für den Test wurden die verschiedenen Geräte mittels einer Mikroskophalterung in verschiedenen Höhen über dem Teststreifen fixiert. Dieser Aufbau ist in Abbildung 109 auf Seite 235 und Abbildung 110 auf Seite 235 dargestellt. Es wurde dann untersucht, ob das Gerät in der Lage ist, alle Marker gleichzeitig auf dem Teststreifen zu erkennen und das Ergebnis zu berechnen. Ein Gerät hat den Test erfolgreich absolviert, sofern in einem 30 Sekunden Intervall mindestens alle 5 Sekunden die Marker erkannt wurden.
Die Ergebnisse dieser Evaluation sind in der Tabelle 7 dargestellt. Auf gut ausgestatteten Smartphones mit hochauflösender Kamera, Autofokus und LED-Licht konnten die Marker sehr zuverlässig detektiert werden. Auf Smartphones, die diese Funktionen nicht besitzen, konnte nur ein deutlich schlechteres Resultat erzielt werden. Insbesondere der Autofokus ist eine essenzielle Eigenschaft um Marker von einer kleinen Größe (3x3 mm) zu erkennen. Die Smartphones „Sony E1“ und „Sony Tipo“ konnten auf Grund ihrer geringen Kameraauflösung und des fehlenden Autofokus in keinem Szenario die Marker detektieren. Abbildung 111 auf der nächsten Seite und Abbildung 112 auf der nächsten Seite veranschaulichen die Wichtigkeit dieser Funktion.

**Aufnahmen unterschiedlicher Abstand**

| Abstand (cm) | Rotation | | OnePlus One | LG Nexus 4 | Sony Z3 Compact | Motorola Moto G | LG Bello | Sony E1 | Sony Tipo |
|---|---|---|---|---|---|---|---|---|---|
| | | Autofokus: | Ja | Ja | Ja | Nein | Ja | Nein | Nein |
| 5 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 7 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 8 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 9 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 10 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 11 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 12 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 13 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 14 | 0 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 15 | 0 | | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 16 | 0 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

**Aufnahmen unterschiedliche Rotation**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 20 | | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 10 | 45 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

Tabelle 7: Vergleich zwischen verschiedenen Smartphones und Abstand zum Teststreifen

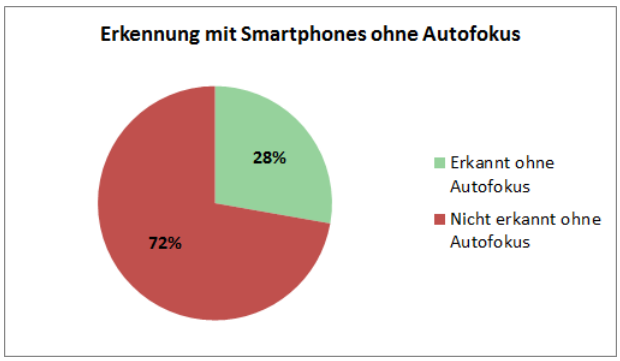



Abbildung 111: Erkennung mit Smartphones ohne Autofokus

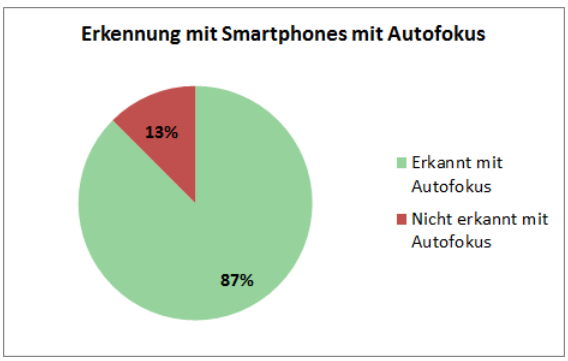



Abbildung 112: Erkennung mit Smartphones mit Autofokus

**Parameter** Die Fähigkeit eines Smartphones, die Marker zu erkennen, hängt von der Größe der Marker, der Distanz zwischen Kamera und Teststreifen, der Auflösung der Kamera und der Fokussierbarkeit der Kamera ab. Größere Marker können leichter von niedrig auflösenden Kameras erkannt werden und ermöglichen eine größere Distanz zwischen Kamera und Marker. Das Aktivieren des LED-Lichtes erhöht die Zuverlässigkeit der Detektion. Die Inhalte der Marker müssen so gewählt werden, das eine zusammenhängende Struktur gebildet wird, damit der im Algorithmus verwendete **SimpleBlob Feature Detector** den gesamten Marker als einen Blob erkennt.

**Fazit** Der Algorithmus konnte in Java überführt und somit erfolgreich in Android integriert werden. Die Reduktion der Marker-Größe hatte zur Folge, dass die Smartphones mit einer geringen Kamera-Auflösung die Teststreifen nicht mehr erkennen können. Das Aktivieren des integrierten Lichts hat die Erkennung auf den Smartphones, welche mit einem solchen ausgestattet sind, deutlich verbessert.

**Ausblick** Um die Erkennungsrate bei Smartphones mit niedrig auflösenden Kameras zu verbessern, kann der Inhalt, die Größe und die Form der Marker überarbeitet werden. Um die Duplizierung von Code zu vermeiden, sollte der Ansatz des geteilten C++ Codes erneut analysiert werden, damit die Android und die iOS App nicht beide die gleichen Algorithmen implementieren müssen.

### 4.4.9 Integration der automatischen Detektierung des Farbumschlags in die iOS-App

- Priorität: Kritisch - Kommt es bei diesem Arbeitspaket zu Verzögerungen, wird das Arbeitspaket „Integration des Weißabgleichs in iOS“ (Kapitel 4.4.7 auf Seite 168) davon betroffen und kann erst später begonnen werden.

- Motivation und Nutzen des Arbeitspaketes:
  Damit die Applikation eine Aussage zur Krankheit treffen kann, muss die automatische Detektierung des Farbumschlages in diese integriert werden.

- Arbeitspaketbeschreibung:
  Die iOS-App ist mit dem Algorithmus zur automatischen Detektierung der Farbumschlagsfelder erweitert. Dazu kann der bereits bestehende Algorithmus aus dem Arbeitspaket „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ aus Sprint 4 herangezogen werden. Die ausgeschnittenen Felder werden zwischengespeichert (zur weiteren Verarbeitung im Quellcode / in der Bildverarbeitung) und können zudem auf dem Smartphone ausgegeben und angezeigt werden (für den Kunden interessant für Produktpräsentation am Ende von Sprint 5).

- Vorbedingungen:
  Das Arbeitspaket „Integration des Weißabgleichs in die iOS-App“ ist erfolgreich abgeschlossen.
- Nebenbedingungen:
  Für die Detektierung der Felder muss nicht zwangsweise der bestehende Algorithmus gewählt werden. Wenn eine reibungslose Integration schnell erfolgen kann, können alternative Algorithmen und Verfahren getestet werden.
- Nachbedingungen:
  Der Algorithmus ist erfolgreich integriert und weißt eine Robustheit von 70% auf, sodass bei sieben von zehn Aufnahmen die Felder erfolgreich erkannt und ausgeschnitten werden können.
- Aufwand:
  Zwei Wochen
- Personen:
  - Christian Sandmann
  - Jan Philipp Stubbe

**4.4.9.1 Dokumentation** Das Ziel dieses Arbeitspakets ist es, die iOS-App so zu erweitern, dass der Bereich des Farbumschlags und des Referenzfeldes auf dem Bild der Kamera lokalisiert wird und dieser Bereich in eine normalisierte Form transformiert werden kann. Dafür werden die Ergebnisse des Arbeitspakets „Integration der automatischen Detektierung des Farbumschlags in die Android-App“ aus Kapitel 4.4.8.1 auf Seite 171 genutzt.

**Ablauf** Im Arbeitspaket „Integration der automatischen Detektierung des Farbumschlags in die Android-App“ existiert bereits ein Algorithmus, dessen einzelne Funktionen übernommen werden konnten. Wie bereits in diesem Kapitel beschrieben, wurde zunächst ein Binärbild aus dem aufgenommenen Bild erzeugt. Dies ist in Abbildung 113 auf der nächsten Seite dargestellt.

Mit der Blob-Erkennung können zusammenhängende Strukturen als wichtige Merkmale im Bild erkannt werden. Diese Erkennung funktioniert und alle wichtigen Punkte werden in der Entwicklungsumgebung ausgegeben. Im Anschluss wurde damit begonnen die Überprüfung der Merkmale zu implementieren.

Im Zuge des Arbeitspaketes „Erneute Machbarkeitsstudie für einen einheitlichen Bildverarbeitungsquellcode“ (Kapitel 4.4.11 auf Seite 181) konnte evaluiert werden, dass gemeinsamer C++ Code in der Android-App und in der iOS-App verwendet werden kann. Dies führte dazu, dass in der Gruppe beschlossen wurde das Arbeitspaket abzubrechen und stattdessen die Bilderkennung direkt im C++ Code zu schreiben. Deshalb wurde darauf verzichtet die Überprüfung der Merkmale vollständig zu implementieren und zu testen, sodass das Arbeitspaket in diesem Zustand abgebrochen und der bereits umgesetzte Code archiviert wurde.

Weil dieser gemeinsame C++ Code von einer anderen Gruppe implementiert wurde, ist eine gemeinsame Schnittstelle bestimmt worden. Diese erwartet ein Bild als Eingabe und liefert das Ergebnis in Form von Eckpunkten, Farbton, Sättigung, Intensität und des bearbeiteten Bildes zurück. Wenn nicht alle Merkmale erkannt werden, besteht das Ergebnis nur aus den gefundenen Eckpunkten.

Im nächsten Schritt wurde die App soweit vorbereitet, dass aus Objective-C++ Code OpenCV aufgerufen werden kann. Des Weiteren wurde die Funktionalität der App so umgestellt, dass diese auf die vereinbarte Schnittstelle reagiert.

Die Benutzerfreundlichkeit der App wurde verbessert indem ein Rahmen in die Kameravorschau gezeichnet wird, der den Bereich angibt, in dem sich die Merkmale befinden sollten. Zusätzlich

wird dem Benutzer ein Hinweis angezeigt, der diesen Rahmen erläutert. Die Kameravorschau ist in Abbildung 114 auf der nächsten Seite dargestellt.

Aufgrund eines Updates des Betriebssystems von iOS 9 traten Fehler bei der Kommunikation mit dem Server auf. Dies wurde durch eine neue Funktion in iOS 9 verursacht, die die Sicherheit der Verbindung zwischen App und Server betrifft. Diese konnten in Zuge dieses Arbeitspaketes vorläufig behoben werden, indem ein Parameter gesetzt wurde, welcher die Kommunikation erlaubt.

Um einen gleichen Stand mit der Android-App zu erreichen, werden in beiden Apps die gleichen Ergebnisse angezeigt (siehe Kapitel 4.4.12.1 auf Seite 184).

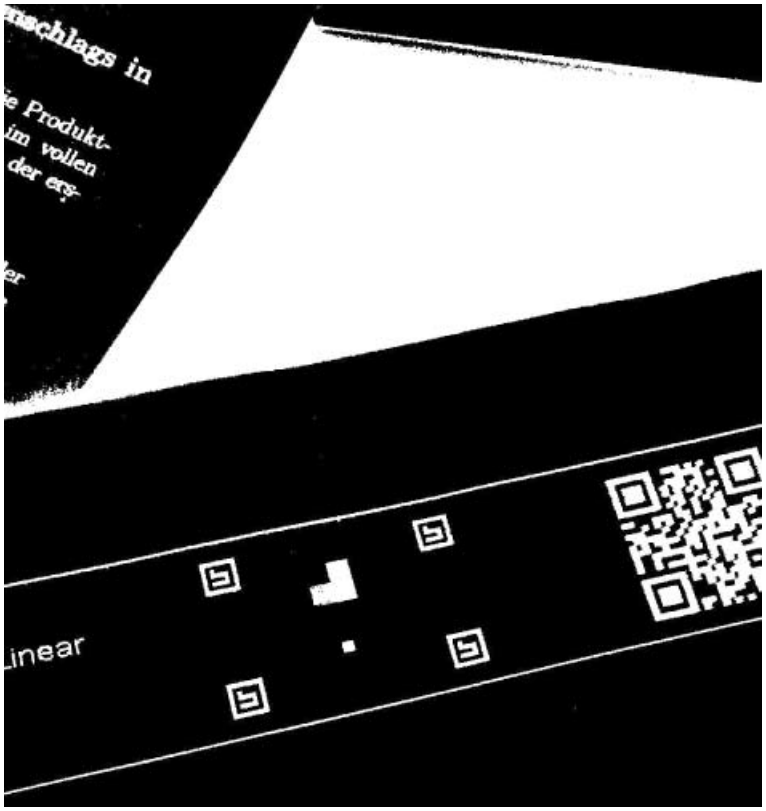



Abbildung 113: Binärbild für die Detektion der Merkmale

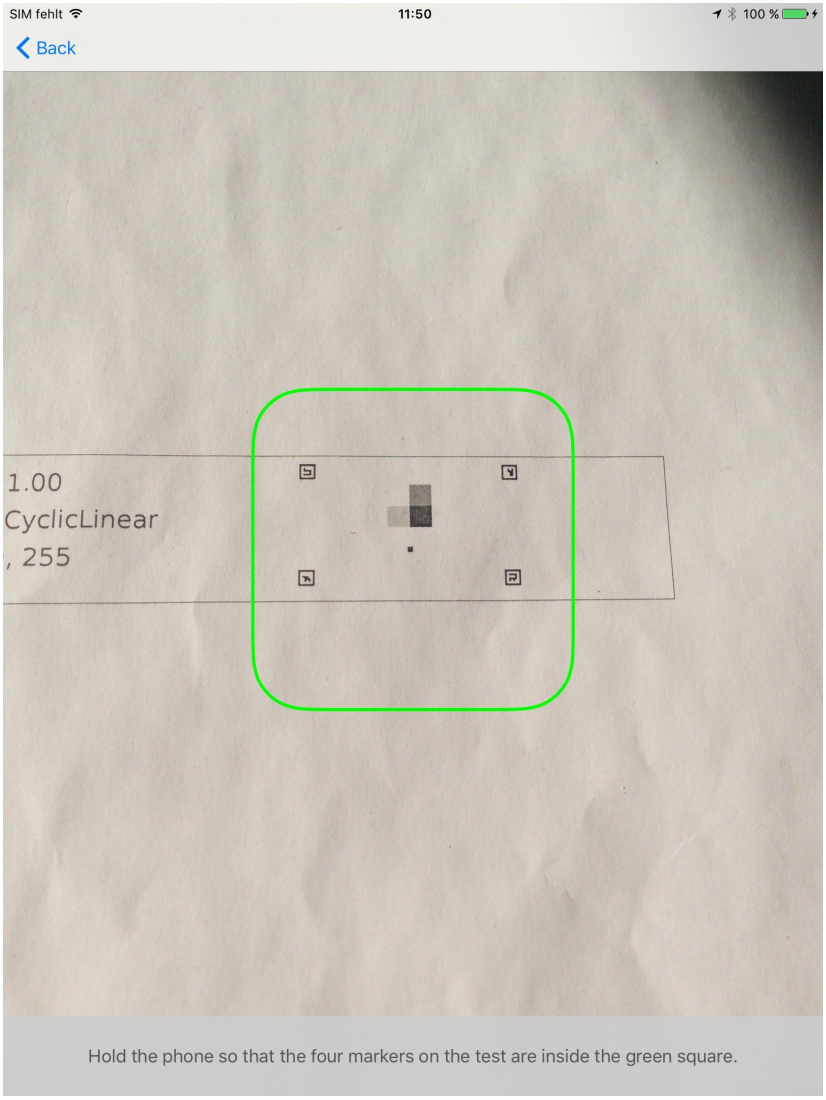



Abbildung 114: Die verbesserte Kameravorschau mit dem Rahmen

**Technologien** Es wurden die gleichen Technologien verwendet wie im Arbeitspaket „Integration von OpenCV in iOS“ (Kapitel 4.4.5.1 auf Seite 165).

**Systembeschreibung** Das System entspricht dem Arbeitspaket „Integration von OpenCV in iOS“ (Kapitel 4.4.5.1 auf Seite 165).

**Evaluation** Die zuverlässige Erkennung der Merkmale und des Farbumschlags wurde in mehreren Durchläufen getestet. Dazu wurden verschiedene Teststreifen erstellt und auf diesen funktionierte der Algorithmus. Es wurden zehn Durchläufe mit drei verschiedenen Teststreifen mit unterschiedlichen Intensitäten und Farben gemacht. Außerdem konnte evaluiert werden, dass die Merkmale am schnellsten erkannt werden, wenn sich diese an den Eckpunkten des Rahmens befinden, der in der Kameravorschau angezeigt wird.

**Fazit** Aufgrund der neu gewonnenen Kenntnisse über die Integration von gemeinsamen C++ Code in beiden Apps, wurde das ursprüngliche Arbeitspaket abgeändert. Der in Objective C entwickelte Code für die Bildverarbeitung wird nicht weiter verwendet.
Stattdessen wird die Bildverarbeitung in C++ umgesetzt. Für die Integration dieses neuen Codes und der Verbesserung der App wurde das Arbeitspaket um eine Woche verlängert. In dieser Woche wurde der Code integriert und die damit einhergehenden Probleme gelöst.

**Ausblick** Die Kamera erkennt die vier Merkmale in hellem Licht gut, in einer dunkleren Umgebung jedoch oft nicht. Um dies zu verbessern, könnte man im nächsten Sprint mit dem erstellten Teststand genauere Untersuchungen anstellen und den Algorithmus so verbessern, dass die Erkennung zuverlässiger erfolgt.

### 4.4.10 Detektierung der Intensität- und Farbsättigung

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Mit diesem Arbeitspaket sollen erste Ergebnisse in der Detektierung des Farbumschlagfeldes erzielt werden. Dies dient im späteren Verlauf dazu, eine Aussage über den Krankheitsstatus treffen zu können.
- Arbeitspaketbeschreibung:
  Der Bildverarbeitungsalgorithmus ist in der Lage die Intensität bzw. die Farbsättigung eines Farbumschlags in einem definierten Bereich festzustellen. Grundsätzlich ist die Intensität von nur einer bestimmten Farbe zu bestimmen (z.B. rot mit dem maximalen rgb 255,0,0). Die Farbintensität vier verschiedener Farben kann der unten stehenden Abbildung 115 entnommen werden.

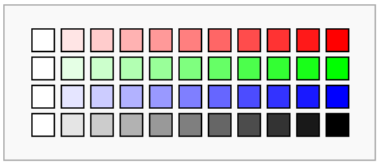


Abbildung 115: Vier verschiedene Farbintensitäten

  Zusätzlich kann der Algorithmus, das Ergebnis je nach Intensität und Sättigung auszugeben und diesen auf dem Bild zu dokumentieren.

- Vorbedingungen:
  Das Arbeitspaket „Weißabgleich“ muss abgeschlossen sein. Zudem ist Evaluation von Bildverarbeitungs-Tools abgeschlossen, damit die Ergebnisse des Algorithmus durch die Ergebnisse der Bildverarbeitungstools abgeglichen werden können.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der Algorithmus liefert innerhalb von drei Sekunden eine Aussage darüber, ob ein positives, negatives oder unklares Ergebnis eines Testbilds vorliegt.
- Aufwand:
  Drei Wochen
- Personen:
  - Sebastian Horwege
  - Nicolas Koch

**4.4.10.1 Dokumentation** In diesem Paket musste die Sättigung und Intensität des Farbumschlagfeldes ermittelt werden. Dies soll automatisiert in C++ Code geschehen, sodass der Algorithmus als geteilte Codebasis der beiden Apps verwendet werden kann.

**Ablauf** Zunächst wurde ein Algorithmus in Java implementiert. Nachdem klar wurde, dass eine Implementierung in C++ möglich ist, wurde der vorhandene Code portiert. Dazu musste neben der eigentlichen Intensitätsdetektierung auch der Algorithmus für den Weißabgleich überführt werden. Als Basis des neuen, nativen Codes wurde das C++ Projekt aus dem Arbeitspaket „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ (Sprint 4, Seite 62) gewählt, da hier die Objektdetektierung bereits funktioniert.

Durch die Ergebnisse der Arbeitspakete „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ (Sprint 4, Seite 62) und „Weißabgleich“ (Sprint 4, Seite 54) liegen dem zu entwickelnden Algorithmus bereits folgende Werte vor:

- Ausgeschnittener Teil des Farbumschlagsfeldes
- Ausgeschnittener Teil des Referenzfeldes (Maximalfarbe)
- Ausgeschnittener Teil des Referenzfeldes (Minimalfarbe)

Alle vorhandenen Bildteile wurden außerdem bereits weißabgeglichen und können somit weniger Fehler haben, die durch schlechte Beleuchtung aufgenommen sein könnten. Diese drei Unterbilder stellen die Eingabeparameter des Algorithmus dar. Die Ausgabe des Algorithmus ist die Intensität, sowie auch die Sättigung des Farbumschlagsfeldes. Da diese beiden Werte allein nicht sehr aussagekräftig sind, werden sie im Verhältnis der Intensität bzw. Sättigung des Referenzfeldes (Maximalfarbe) berechnet.
Resultierend aus diesen Überlegungen ist der folgende Algorithmus entstanden (Ablauf):

1. Konvertiere die drei Eingabebilder in den HSV Farbraum [27].
2. Berechne die Durchschnittsfarbe der drei Bilder.
3. Stelle sicher, dass der H-Wert des Farbumschlags nicht mehr als 10% vom H-Wert der Referenzfelder abweicht.
4. Nehme den S- und V-Wert des maximalen Referenzfeldes ( $S_{max}$ , $V_{max}$ ) und des Farbumschlagsfeldes ($S_{real}$, $V_{real}$).
5. Berechne die relative Intensität des Farbumschlagfeldes mit der Formel $\frac{V_{real}}{V_{max}}$
6. Berechne die relative Sättigung des Farbumschlagfeldes mit der Formel $\frac{S_{real}}{S_{max}}$

In der aktuellen Version wird das Ergebnis dieser Berechnung auf der Konsole ausgegeben. Die Wahl des HSV Farbraums basiert auf der Annahme, dass der Teststreifen später lediglich über die Intensität des Farbumschlags zu bewerten ist, nicht durch die Änderung einer Farbe zu einer anderen (zum Beispiel grün zu rot). Der Farbumschlag sollte deshalb im HSV Farbraum durch die $S$ und $V$ Werte gut zu bewerten sein.

**Technologien**

**OpenCV** Bibliothek zur Bildverarbeitung.

**CMake** Werkzeug zum Bauen von C und C++ Projekten.

[27] https://de.wikipedia.org/wiki/HSV-Farbraum

**Systembeschreibung** Dieses Arbeitspaket wurde unter einem Linux entwickelt. Dabei wurden neben Texteditoren vor allem Standard-Programme, wie make[28] und cmake[29] verwendet. Der entwickelte Code wurde noch nicht unter einer Windows Umgebung getestet.

### Evaluation

**Performanz** Während zur Performanz keine exakten Metriken gemessen wurden, konnte eine Beeinträchtigung der Benutzbarkeit selbst auf alten Smartphones (Samsung Galaxy S3) nicht festgestellt werden.

Es sind keine Schwankungen der Bildrate festzustellen, obwohl der Algorithmus pro Bild[30] ausgeführt wird.

**Korrektheit** Zur Überprüfung der Algorithmen wurden elf Teststreifen mit dem Teststreifengenerator erzeugt. Im Generator wurde der Intensitätswert des Umschlagfelds der Teststreifen in 10.0% Schritten von 0.0% bis 100.0% variiert. Von jedem Teststreifen wurde ein Video mit einem **Samsung Galaxy S 4 Active** gemacht und in der Testumgebung ausgewertet. Die Ergebnisse sind in Abbildung 116 zu sehen. Aus der Abbildung ist eindeutig abzuleiten, dass der **H** Wert der Auswertungen über alle Teststreifen konstant ist. Zur eindeutigen Bestimmung der Intensitätswerte reichen also die **S** und **V** Werte aus.

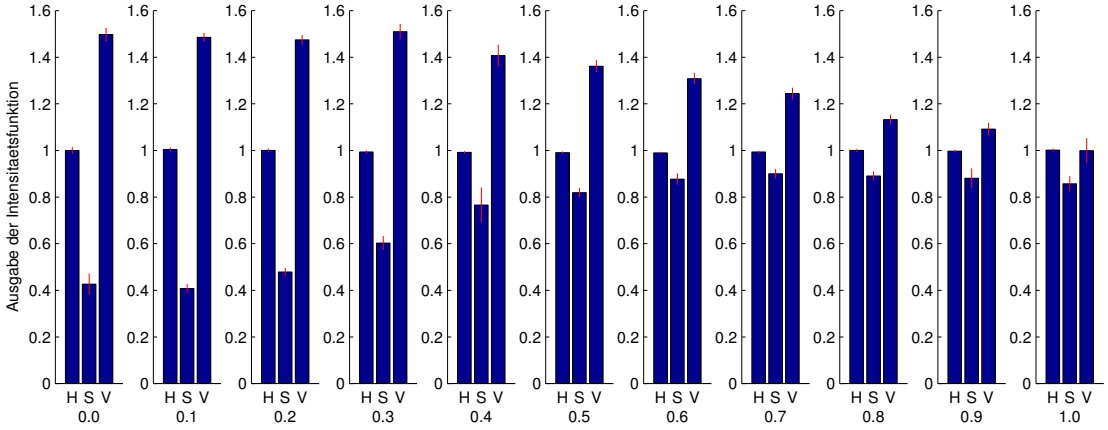



Abbildung 116: Ähnlichkeitswerte für die verschiedenen Teststreifen

**Parameter** Damit sinnvolle Ausgabewerte entstehen, müssen die extrahierten Bereiche eines Teststreifens eingegeben werden.

**Fazit** Durch die Umsetzung wurden alle Anforderungen aus der Arbeitspaketbeschreibung umgesetzt. Anstatt der Verwendung absoluter Werte für Sättigung und Intensität werden relative Werte berechnet, welche einen Aussagewert mit Hinblick auf mögliche Anwendungsfälle haben. Die gesamte Implementierung liegt in C++ vor und kann mit sehr wenig Aufwand in den restlichen C++ Code integriert werden, sobald dieser vorhanden ist[31].

[28] https://de.wikipedia.org/wiki/Make
[29] https://de.wikipedia.org/wiki/CMake
[30] Genauer gesagt: Pro Bild, in dem die Objekterkennung erfolgreich ist.
[31] Die Portierung der Bildverarbeitung in C++ findet parallel zu diesem Arbeitspaket statt

**Ausblick** Durch die Umsetzung dieses Arbeitspaketes kann die Intensität und Sättigung des Farbumschlags bestimmt werden. Dadurch wurde die Bildverarbeitung der mobilen Apps um eine weitere Komponente erweitert. Mithilfe der neuen Informationen kann in Zukunft bestimmt werden, ob ein Proband tatsächlich erkrankt ist. Zusätzlich ist denkbar, eine genauere Aussage über den Krankheitsgrad eines Probanden zu treffen. Während der Bearbeitung des Arbeitspakets ist die Frage aufgekommen, wie der Farbumschlag auf dem Teststreifen später durch einen Algorithmus zu bewerten ist.Dazu gibt es bereits im Arbeitspaket „Weißabgleich“ (siehe Sprint 4, Seite 54) einige Ansätze. Es stellt sich jedoch immer noch die Frage, welche Bewertungsfunktion für die Teststreifen optimal ist. Neben der Auswertung im **HSV** oder **RGB** Farbraum und der euklidischen Distanz als Bewertungsfunktion, wäre eine Bewertung mithilfe der **DeltaE**[32][33] Metrik auch denkbar. Diese berücksichtigt, im Gegensatz zu **HSV** und **RGB**, nämlich die Farbwahrnehmung des menschlichen Auges, das Farben nicht linear wahrnimmt. Bisher wurde das Referenzfeld für den minimalen Farbumschlag noch nicht in die Intensitätsberechnung einbezogen. Aus der Evaluation geht auch hervor, dass dies unter Umständen auch nicht nötig ist.

### 4.4.11 Erneute Machbarkeitsstudie für einen einheitlichen Bildverarbeitungsquellcode

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Da die letzte Machbarkeitsstudie ergeben hat, dass in Android Studios kein C Programmcode verwendet werden kann, soll dies für die neue Android Studio Version erneut überprüft werden.
- Arbeitspaketbeschreibung:
  Es ist eine Machbarkeitsstudie durchgeführt worden, welche erneut untersucht, ob es mit der aktuellen Version von Xcode 7 und der neuen Version von Android Studios 1.3.2 möglich ist, einen einheitlichen Quellcode (C++) für die Bildverarbeitungsalgorithmen zu verwenden. Dazu ist OpenCV in die Projekte eingebunden und es kann C++ Code kompiliert werden. Des Weiteren können Funktionalitäten der OpenCV Bibliothek über C++ Quellcode aufgerufen und verwendet werden.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  -
- Aufwand:
  Zwei Wochen
- Personen:
  - Timo Schlömer
  - Christoph Ressel

#### 4.4.11.1 Dokumentation

[32]Erklärungen zu DeltaE `https://de.wikipedia.org/wiki/Delta_E`
[33]Artikel über Farbmetriken `http://www.compuphase.com/cmetric.htm`

**Ablauf** Die Machbarkeitsstudie für die iOS-Plattform wurde von Seiten der Betreuer des Projekts erfolgreich durchgeführt und es konnte nach sehr kurzer Zeit ein Prototyp fertiggestellt werden, der einen in C++ geschriebenen Code aus bestehendem Objective-C Code aus aufrufen kann. In dem C++ Code wurden beispielhaft Funktionen aus der OpenCV Bibliothek aufgerufen, welche im späteren Algorithmus auch verwendet werden.

Die Studie wurde für die Android Plattform von der Projektgruppe bearbeitet und dauerte deutlich länger als die äquivalente Umsetzung für iOS, blieb aber innerhalb des für das Arbeitpaket vorgesehenen Zeitrahmens. Für die Einbettung von C/C++ Code in Android wird das Android NDK benötigt, welches den nativen Code für die Android Plattform kompilieren kann. Das Konfigurieren und Starten des NDKs kann das offizielle Gradle Plugin übernehmen, das auch den Java Code für Android verwaltet, oder es werden Makefiles[34] verwendet, über welche die NDK Funktionen manuell gesteuert werden können.

Als erstes wurde versucht die Einbindung des C++ Codes mittels des Gradle Buildsystems zu ermöglichen. Das Android Plugin für Gradle existiert in zwei Versionen, einer stabilen und einer experimentellen. Von der Nutzung der stabilen Version wird offiziell abgeraten und auf die neue, experimentelle verwiesen. Es wurden dennoch beide Versionen getestet. Bei beiden Versionen war es möglich, eigenen C++ Code einzufügen, aber unmöglich, die bestehende OpenCV Bibliothek einzubinden. Nach eigener Aussage der Entwickler des Gradle Plugins ist eine Einbindung von externen Bibliotheken bislang nicht möglich.[35]

Die einzig übrig bleibende Alternative war das Nutzen von selbst geschriebenen Makefiles, die die Kompilierung der Quellcode Dateien steuern. Die Syntax dieser Dateien wird auf der offiziellen Webseite des Android NDKs grob beschrieben, bedurfte aber viel eigener Recherche und Tests um die korrekte Deklaration zu finden. Das Ergebnis sind zwei Makefiles (Android.mk und Application.mk) die durch einen eigenen Eintrag in der Gradle Konfigurationsdatei beim Bauvorgang der App gestartet werden. Durch diese beiden Dateien werden die benötigten, nativen Bibliotheken kompiliert, die dann im Java Code verwendet werden können.

**Technologien**

**Android NDK** Eine Sammlung von Werkzeugen um nativen C und C++ Code in das Android System einzubinden.

**Java Native Interface (JNI)** Dient als Schnittstelle zwischen Java und C Code. Methoden können entsprechend einer Konvention aufgerufen und Daten ausgetauscht werden.[36]

**OpenCV** Eine Bildverarbeitungsbibliothek, die im Projekt für die Analyse von Teststreifen genutzt wird. Diese soll durch nativen Code angesprochen werden.

**Gradle** Das Standardwerkzeug zum Bauen von Android-Quellcode

**Systembeschreibung** Als Entwicklungsumgebung für Android wurde Android Studio in Version 1.3.2 verwendet. Diese Version unterstützt die Integration und das Debugging von nativen C/C++ Code. Für die Entwicklung des iOS Codes wurde Xcode 7 verwendet.

**Evaluation** Der native Code konnte auf den Geräten „LG Nexus 4“ , „OnePlus One“ und „Sony Xperia Z3 Compact“ erfolgreich ausgeführt werden.

[34] Offizielle Beschreibung der Makefiles: `https://developer.android.com/ndk/guides/android_mk.html`, zuletzt geöffnet am 18.10.2015

[35] `https://groups.google.com/forum/#!msg/adt-dev/9bWWbfty3gA/qQnHj5_yBAAJ`, zuletzt geöffnet am 18.10.2015

[36] Offizielle JNI Spezifikation: `http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html`, zuletzt geöffnet am 18.10.2015

**Parameter** Der native Code kann unter iOS auf allen Systemen ausgeführt werden, die iOS 9 untersützen. Unter Android 4.0.3 (oder höher) nur mit folgenden Prozessor-Architekturen:

- Arm64-V8a
- ArmEABI
- ArmEABI-V7a
- MIPS
- MIPS64
- x86
- x86-64

**Fazit** Die fehlende Unterstützung des Gradle Plugins für die Einbindung von externen Bibliotheken in Android ist zu bemängeln, kann aber zum jetzigen Zeitpunkt nicht umgangen werden. Der Bauvorgang kann jedoch erfolgreich mit Makefiles durchgeführt werden. Durch den für iOS erfolgreich entwickelten Prototypen, der von den Betreuern bereitgestellt wurde, konnte gezeigt werden, dass eine geteilte Codebasis, die OpenCV beinhaltet, unter iOS ausgeführt werden kann. Somit ist das Entwickeln einer geteilten Codebasis für iOS und Android möglich.

**Ausblick** Die fortlaufende Entwicklung des Gradle Plugins für Android sollte beobachtet werden, um zu einem späteren Zeitpunkt den Bauprozess mittels Gradle erneut zu evaluieren. So könnte der Bauprozess unkomplizierter gestaltet und besser in die Entwicklungsumgebung integriert werden.

Da eine geteilte Codebasis als möglich erwiesen wurde, sollte der Bildverarbeitungsalgorithmus, der bisher in Java entwickelt wurde, durch einen in C++ implementierten Code ersetzt werden.

### 4.4.12 Integration der Bildverarbeitungsalgorithmen in C++

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Für eine effizientere Entwicklung und Wartung, soll der Bildverarbeitungsalgorithmus in nur einer Programmiersprache entwickelt werden.
- Arbeitspaketbeschreibung:
  Da das Ergebnis des Arbeitspaketes „Erneute Machbarkeitsstudie für einen einheitlichen Bildverarbeitungsquellcode“ (Kapitel 4.4.11 auf Seite 181) ergeben hat, dass es aktuell möglich ist einen einheitlichen C++ Quellcode für die iOS als auch die Android App verwenden zu können, soll dies in diesem Arbeitspaket umgesetzt werden. Es gilt die bestehenden Algorithmen der Bildverarbeitung (Weißabgleich und Identifikation der Farbumschlagfelder auf dem Teststreifen) in C++ zu schreiben und in die bestehende iOS und Android App zu integrieren. Zudem sind die iOS und Android am Ende des Arbeitspaketes so weit präpariert, dass Sie dem Kunden vorgestellt werden können (der Kunde kann den Weißabgleich wahrnehmen und sieht, dass die Farbumschlagfelder aus dem Teststreifen extrahiert werden können).
- Vorbedingungen:
  Das Arbeitspaket „Integration der automatischen Detektierung des Farbumschlags in die iOS-App“ (Kapitel 4.4.9 auf Seite 174) wird nicht beendet, da bereits ein lauffähiger Code aus dem Arbeitspaket „Integration der automatischen Detektierung des Farbumschlags in die Android-App“ (Kapitel 4.4.8 auf Seite 171) entwickelt werden konnte. Dieser wird nun in C++ überführt wird.

- Nebenbedingungen:
  Das bereits generierte Wissen aus den Arbeitspakete zur Bildverarbeitung sowie die bestehende Logik der aktuell verwendeten Algorithmen kann verwendet und in C++ umgeschrieben werden.

- Nachbedingungen:
  Die Apps für iOS und Android enthalten einen einheitlichen C++ Quellcode, welcher die Bildverarbeitungsalgorithmen enthält. Zudem wurden die Apps auf ihre Funktionalität und Qualität getestet, um so sicherzustellen, dass diese weiterhin lauffähig sind.

- Aufwand:
  Zwei Wochen

- Personen:
  - Christoph Ressel
  - Timo Schlömer
  - Nicolas Koch

**4.4.12.1 Dokumentation** In diesem Arbeitspaket wurde der C++ Code aus den Arbeitspaketen „Shared C++ Code Bildverarbeitung“ (Kapitel 4.4.11 auf Seite 181) und „Intensitätsdetektierung“ (Kapitel 4.4. auf Seite 178) in die mobilen Apps integriert, sodass dieser anstelle des plattformabhängigen Codes verwendet wird.

**Ablauf** Im Rahmen des Arbeitspaketes wurde die Intensitätsdetektion, sowie auch der Weißabgleich in den restlichen C++ Code integriert. Dieser Code existierte bereits aus dem Arbeitspaket „Intensitätsdetektierung“ . Bei der Bearbeitung des Arbeitspaketes mussten sich die Gruppen der Android und der iOS App auf eine Konvention der Parameterübergabe und des Ergebniswertes einigen, damit sich der geteilte C++ Code nahtlos in die jeweiligen Apps einfügt.

So werden die eingegebenen Bilder von beiden Systemen vor der eigentlichen Verarbeitung in ein Format konvertiert, welches unter beiden Plattformen weiterverarbeitet werden kann. Die Bilder werden in diesem Format im eigentlichen Algorithmus ausgewertet. Nach der erfolgreichen Verarbeitung konvertieren die Apps die Ergebnisse wieder in ein plattformabhängiges Format, um es dem Benutzer anzeigen zu können.

Beide Apps beenden die Aufnahme der Kamera nachdem der Teststreifen erkannt wurde und zeigen in einer anderen Ansicht die aufgenommenen und bearbeiteten Bilder (Abbildung 117 auf der nächsten Seite zeigt die Ansicht unter Android und Abbildung 118 auf der nächsten Seite zeigt die Ansicht unter iOS). Zusätzlich werden die ermittelten Farbwerte des Farbumschlagsfeldes aus dem HSV-Farbraum dargestellt.

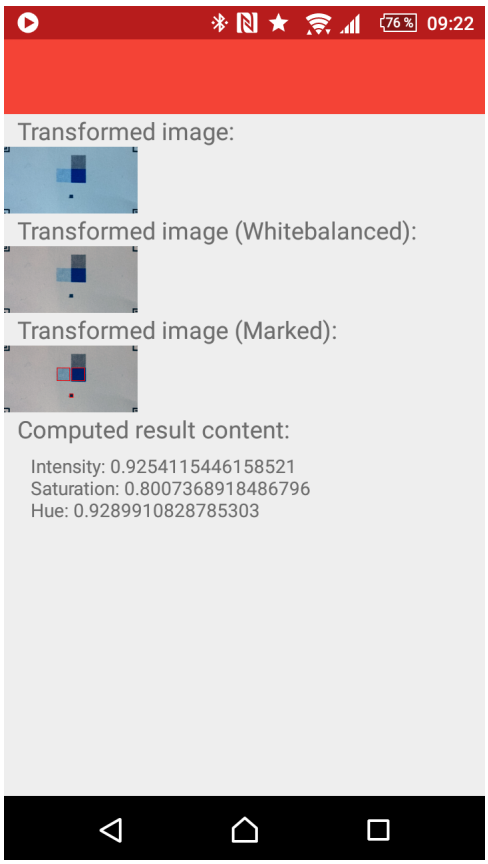



Abbildung 117: Die Android-App hat einen Teststreifen erkannt

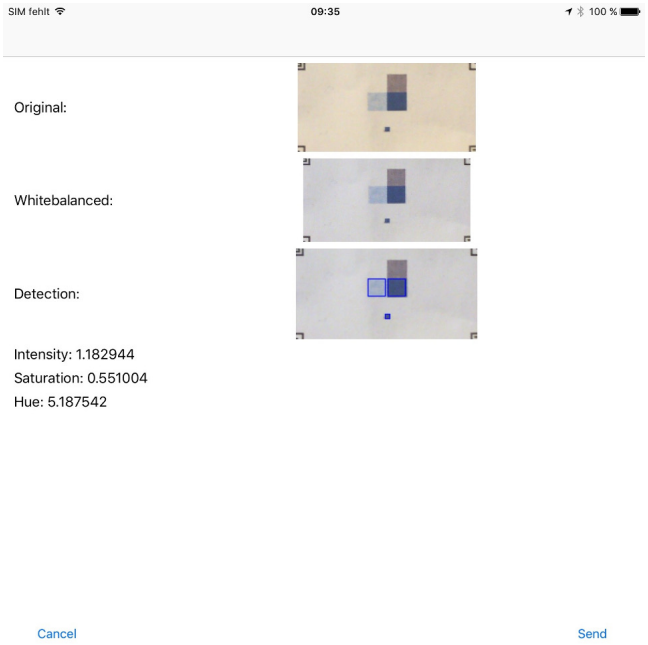



Abbildung 118: Die iOS-App hat einen Teststreifen erkannt

Nachdem der gemeinsame Code in beiden Apps integriert war, wurde festgestellt, dass der verwendete Algorithmus bei der Detektion der Marker noch Probleme aufweist und unter manchem Umständen keine Marker detektieren kann. Dies wurde vor allem bei Tests mit dem noch nicht vollständigen Teststand deutlich. Die Betreuer haben angemerkt, dass bis zur Präsentation vor dem Kunden ein besser funktionierender Algorithmus entwickelt werden sollte, der zuverlässiger demonstriert werden kann. Der Algorithmus wurde dementsprechend abgeändert und verläuft nun wie folgt:

1. Das aktuelle Bild der Kamera wird aus dem jeweiligen Farbraum des Geräts in den **BGR** Farbraum konvertiert.
2. Mit Hilfe des **adaptiven Schwellwertverfahrens**[37] wird aus dem Bild ein Binärbild erzeugt.
3. Der **Simpleblob** Algorithmus wird durchgeführt um Blobs (zusammenhängende Punkte) im Bild zu finden. Durch die Konvertierung in ein Binärbild und eine gute Parameterwahl für den Simpleblob Algorithmus werden die Marker gut erkannt.
4. Für jeden gefundenen Blob wird überprüft, ob dieser ein Marker ist:
   (a) Es wird eine **Kontur-Detektion** durchgeführt, wodurch die Begrenzung des Markers gefunden wird (sofern der aktuelle Blob ein Marker ist).
   (b) Mit Hilfe der Konturpunkte, die OpenCV zurück gibt, werden die Eckpunkte des Markers ermittelt. Dafür wird für jeden Eckpunkt des Blobs der Konturpunkt gesucht, welcher am weitesten entfernt liegt. Dieser Konturpunkt ist ein Eckpunkt der schwarzen Umrandung des potentiellen Markers.

[37] Adaptives Schwellwertverfahren: `http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm`, zuletzt geöffnet am 23.10.2015

(c) Mit Hilfe der gefundenen Marker-Eckpunkte kann eine Transformation durchgeführt werden, wodurch der Marker in eine normalisierte Form gebracht werden kann.

(d) Der Marker wird mit der Liste zu erwartender Marker abgeglichen, wobei der potentielle Marker seine Originalgröße beibehält und der jeweilige Referenzmarker auf die passende Größe vergrößert wird. Der Abgleich erfolgt indem beide Bilder mittels einer **Exklusiven-Oder** Operation vereint werden. Das Ergebnis stellt den Unterschied zwischen den beiden Bildern dar (beispielhaft zu sehen in Abbildung 121 auf der nächsten Seite).

(e) Um Ungenauigkeiten der Kameraaufnahme entgegen zu wirken, wird auf dem Ergebnis des vorherigen Schrittes eine **morphologische Öffnung**[38] angewendet. So werden kleine Differenzen zwischen den Bildern beseitigt (wie etwa ungerade Kanten), aber große Unterschiede bleiben erhalten (beispielhaft zu sehen in Abbildung 122 auf der nächsten Seite).

(f) Die Anzahl an nicht-schwarzen Pixeln in dem Ergebnisbild werden gezählt (diese Pixel sind zwischen dem erwarteten Bild und dem aufgenommenen Bild unterschiedlich) und zur Bildung eines Fehlerwertes verwendet. Dieser gibt den Unterschied (in Prozent) zwischen den beiden Bildern an.

(g) Ein Marker gilt als korrekt detektiert, wenn der Fehlerwert unter 3% liegt.

5. Sind alle vier Marker gleichzeitig gefunden, werden die Zentren dieser genutzt, um das gesamte Bild in die normalisierte Form zu transformieren und den Bereich zwischen den Markern auszuschneiden. Dieser Bereich enthält den Farbumschlag und das Referenzfeld.

6. Der Farbumschlag und das Referenzfeld befinden sich immer an der jeweils exakt gleichen Position im transformierten Bild und können somit pixelgenau ausgeschnitten werden.

7. Die Farben im ausgeschnittenen Farbumschlagsfeld werden gemittelt und in den HSV-Farbraum konvertiert. Die Farbwerte werden zusätzlich zu den ausgeschnitten Feldern als Ergebnis ausgegeben.

In der vorherigen Version des Algorithmus wurde zum Erstellen eines Binärbildes das **OTSU**[39] Verfahren verwendet. Das Verfahren liefert nur gute Ergebnisse, wenn das aufgenommene Bild gleichmäßig ausgeleuchtet ist. So wird ein gutes Binärbild erzeugt, wenn der Teststreifen auf einem Tisch liegt, aber schlägt fehl, wenn der Teststreifen vor die Kamera gehalten wird und der Rest des Raumes sichtbar ist (beispielhaft zu sehen in Abbildung 119 auf der nächsten Seite).

Um diesen Fehler zu beheben wird nun auf ein adaptives Schwellwertverfahren zurückgegriffen. Bei diesem Verfahren wird nicht nur ein Schwellwert erzeugt, welcher auf das gesamte Bild angewendet wird, sondern wird das Bild in mehrere Bereiche unterteilt, welche dann einen eigenen Schwellwert haben. Mit diesem Verfahren ist der Teststreifen mit seinen Markern immer noch gut sichtbar, auch wenn viele andere Objekte (mit unterschiedlicher Helligkeit) im Hintergrund sind (beispielhaft zu sehen in Abbildung 120 auf der nächsten Seite).

[38] Morphologische Öffnung: `https://de.wikipedia.org/wiki/Opening_%28Bildverarbeitung%29`, zuletzt geöffnet am 18.10.2015

[39] OTSU-Schwellwertverfahren: `https://de.wikipedia.org/wiki/Schwellenwertverfahren#Verfahren_von_Otsu`, zuletzt geöffnet am 23.10.2015

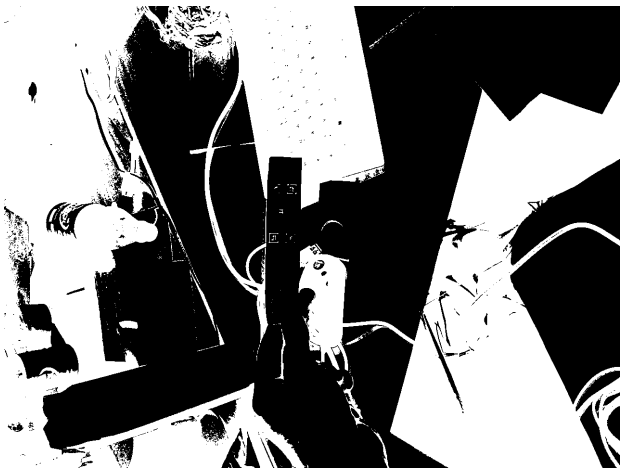


Abbildung 119: Das Binärbild nach dem OTSU Verfahren

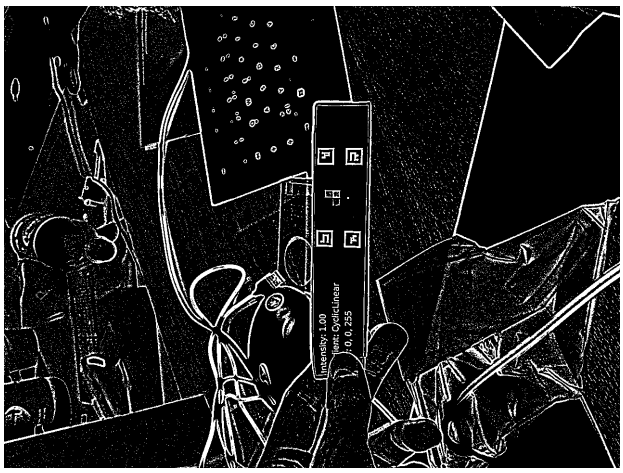


Abbildung 120: Das Binärbild nach dem adaptiven Schwellwertverfahren

Es wurde zusätzlich die Detektion der Marker verändert um den Einfluss von Bildfehlern zu verringern. In der vorherigen Version des Algorithmus wurden die detektierten Marker auf ein 8x8 Pixel großes Bild reduziert und dann Pixel für Pixel mit den erwarteten Markern verglichen. War ein Pixel verkehrt, wurde der Marker zurückgewiesen. Um einen größeren Spielraum bei der Detektion der Marker zu gewähren, werden die aufgenommenen Bilder nun nicht mehr auf eine feste Größe reduziert. Stattdessen wird die Größe auf die nächstbeste, durch acht teilbare, quadratische Einheit vergrößert. Durch dieses Verfahren bleiben mehr Pixel für einen Vergleich übrig. Wenn der Referenzmarker (welcher auf die Größe des aufgenommenen Markers vergrößert wird) mit dem nun quadratischen Marker verglichen wird, bleiben meist noch Differenzen übrig, die durch die Unschärfe der Kamera oder durch das Transformieren des Markers verursacht werden. Diese Fehler werden mit einer **morphologischen Öffnung** kompensiert.

Abbildung 121: Der potentielle Marker nach der Exklusiv-Oder Operation

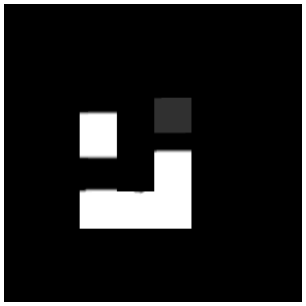


Abbildung 122: Das korrigierte Exklusiv-Oder Ergebnis, nach morphologischer Öffnung

Für den Aufruf des nativen Codes unter Android wird auf das Java Native Interface zurückgegriffen, mit dem die Parameter zwischen dem Java Code und dem C++ Code ausgetauscht werden können. Eine Schwierigkeit bestand darin, die Bilder welche von der OpenCV Bibliothek generiert werden in ein Format zu bringen, welches unter Java eingelesen werden kann. Hierfür werden für alle Bilder Integer-Arrays erzeugt, welche an OpenCV übergeben werden. Die Arrays werden von der Bibliothek als Speicher für die Bilder genutzt und die Pixeldaten dort hinein geschrieben. Nach Durchlauf des Algorithmus können die Arrays wieder an den Java Code übergeben werden, wo diese dann zu einem Bild dekodiert werden können. Damit dieses Vorgehen funktioniert musste darauf geachtet werden, dass die Größe und der Typ der OpenCV Matrix mit der erwarteten Größe des Bildes übereinstimmt, und dass das Bild im BGRA-Farbraum gespeichert wird. Ist das Bild im BGR, RGB oder ARGB Farbraum, kann Android das Bild nicht korrekt dekodieren.

**Technologien**

**Android NDK** Eine Sammlung von Werkzeugen um nativen C und C++ Code in das Android System einzubinden.

**Java Native Interface (JNI)** Dient als Schnittstelle zwischen Java und C Code. Methoden können entsprechend einer Konvention aufgerufen und Daten ausgetauscht werden.[40]

**OpenCV** Eine Bildverarbeitungsbibliothek, die im Projekt für die Analyse von Teststreifen genutzt wird. Diese soll durch nativen Code angesprochen werden.

**CMake** Werkzeug zum Kompilieren von C und C++ Projekten.

**Systembeschreibung** Als Entwicklungsumgebung für Android wurde Android Studio in Version 1.3.2 verwendet. Diese Version unterstützt die Integration und das Debugging von nativen C/C++ Code. Für die Entwicklung des iOS Quellcodes wurde Xcode 7 verwendet.

**Evaluation** Zum Testen des neu eingepflegten Quellcodes wurde dieser sowohl in die iOS als auch die Android Applikation integriert. Beide Apps wurden unter der Verwendung verschiedener Teststreifen getestet. Dabei konnten zuverlässig (in unter 5 Sekunden) die Teststreifen aus dem Bild der Kamera verarbeitet werden und die Farbwerte im HSV-Farbraum des Farbumschlagfeldes ausgegeben werden.

Bei der Bestimmung der Farbe des Teststreifens kann es unter Verwendung niedrig-auflösender Kameras dazu kommen, dass durch Rauschen im Bild die Farbe des Farbumschlagfeldes nicht richtig erkannt wird. Ist dies der Fall, weichen die Farbe des Farbumschlagfeldes und die des Referenzfeldes voneinander ab.

**Parameter** Der native Code kann unter iOS auf allen Systemen ausgeführt werden, die iOS 9 unterstützen. Unter Android 4.0.3 (oder höher) nur mit folgenden Prozessor-Architekturen:

- Arm64-V8a
- ArmEABI
- ArmEABI-V7a
- MIPS
- MIPS64
- x86
- x86-64

**Fazit** Die Android- und iOS-App verwenden nun beide den C++ Code um die Bildanalyse auszuführen. Beide Apps sind außerdem in der Lage den Algorithmus vollständig durchzuführen, sodass ein normalisiertes Bild als Ergebnis vorliegt, aus dem der Farbumschlagsbereich ausgeschnitten wird.

Dem Benutzer werden drei Bilder für diesen Vorgang angezeigt:

- Das normalisierte Bild

[40] Offizielle JNI Spezifikation: `http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html`, zuletzt geöffnet am 18.10.2015

- Das normalisierte und mit einem Weißabgleich verbesserte Bild
- Das normalisierte und mit einem Weißabgleich verbesserte Bild, auf dem der Farbumschlagsbereich und das Referenzfeld markiert werden.

Auch werden nach Erkennen des Teststreifens die Farbwerte für den Farbumschlagsbereich ausgegeben. Diese sind der Farbton, die Farbsättigung und die Intensität aus dem HSV-Farbraum.

Änderungen am Bilverarbeitungsalgorithmus müssen nun nicht mehr in beide Apps eingepflegt werden, sondern können an einer zentralen Stelle geändert werden. Nur bei Änderungen der Ein- und Ausgabewerte müssen die Apps angepasst werden.

**Ausblick** Für die Android-App gilt weiterhin, dass zukünftige Versionen des Gradle-Plugins für Android evaluiert werden sollten. Das Gradle-Plugin muss in der Lage sein, den eignen Code mit der OpenCV Bibliothek zu vereinen.

### 4.4.13 Code-Refactoring PrimeFaces

- Priorität: Normal - Dieses Arbeitspaket wird nicht direkt von anderen Arbeitspaketen beeinflusst. Aufgrund von zeitlichen Aspekten muss es direkt zu Beginn des fünften Sprints bearbeitet werden.
- Motivation und Nutzen des Arbeitspaketes:
  Da an dem Programmcode, welcher in PrimeFaces verfasst wurde, zukünftig noch weiter entwickelt wird, muss dieser „aufgeräumt“ und ausführlicher kommentiert werden.
- Arbeitspaketbeschreibung:
  Der in PrimeFaces entstandene Code weist diverse Altlasten auf und ist zu bereinigen. Am Ende des Arbeitspaketes ist jeglicher alter bzw. nicht verwendeter Quellcode entfernt. Der verwendete Quellcode entspricht zudem den Qualitätsnormen und Konventionen des MEDIC-Projektes (Coding-Guidelines, Dokumentation usw.).
- Vorbedingungen:
  -
- Nebenbedingungen:
  Falls zu diesem Zeitpunkt weiter an der Weboberfläche in PrimeFaces gearbeitet wird, z. B. um das User-Management zu erweitern, gilt es, sich mit den Teams, welche weiter am Quellcode arbeiten absprechen. Dies soll verhindern, dass neue Altlasten entstehen.
- Nachbedingungen:
  Der Quellcode wird vom QM-Beauftragten des Projektes auf seine Qualität geprüft und begutachtet. Erst nachdem dieser den Quellcode abgenommen hat, gilt das Arbeitspaket als abgeschlossen.
- Aufwand:
  Drei Wochen
- Personen:
  - Danny Fonk
  - Kevin Sandermann

**4.4.13.1 Dokumentation** Ziel dieses Arbeitspaketes ist es den im vergangenen Sprint entstandenen PrimeFaces-Code dahingehend zu überarbeiten, dass die Code-Qualität und Stabilität des Frontends verbessert wird.

**Ablauf** Zunächst wurden sämtliche Klassen und Methoden des Quellcodes inhaltlich untersucht, sodass die Anzahl der Klassen auf ein sinnvolles Maß reduziert werden konnte und nun eine logische Struktur erkennbar ist. Jede Methode liegt jetzt in einer ihr semantisch übergeordneten Klasse. Bisher wurde der Datenzugriff innerhalb von Model-Klassen realisiert, was nicht dem MVC-Paradigma entsprach. Das MVC-Paradigma ist ein Muster zur Strukturierung in der Software-Entwicklung. Es gibt eine Datenmodelschicht (Model), eine Präsentationsschicht (View) und eine Programmsteuerung (Controller) (vgl. Abbildung 123 auf Seite 235), um bspw. eine Wiederverwendung des Models unter anderen Bedingungen (bspw. Reimplementierung auf einem anderem Betriebssystem) zu gewährleisten. In unserem Fall ist der Datenzugriff zu nennen: Ändert sich dieser beispielsweise auf eine Datei-basierte Struktur, so sind nur die Klassen anzupassen, welche den Datenzugriff realisieren wobei die restlichen Klassen bestehen und weiterverwendet werden können. Um dies zu gewährleisten wurden sogenannte Data-Access-Objects (DAOs) implementiert, welche den Zugriff auf die Datenbasis realisieren, während ein Model lediglich das Objekt selbst repräsentiert.

Im Anschluss wurden alle Klassen inhaltlich kategorisiert und in sinnvolle Pakete unterteilt. Weiterhin wurde die Hibernate-Implementierung auf eine im Quellcode verankerte Konfiguration vereinheitlicht. Im Zuge dessen wurden die vorher benötigten Konfigurationsdateien überflüssig und gelöscht.

Zur Steigerung der Stabilität wurde die Fehlerbehandlung überarbeitet: Sämtliche Methoden und ihre Aufrufe haben falls benötigt entsprechenden Fehlerbehandlungen. Weiterhin wird im Fall eines Fehlers innerhalb der Fehlerbehandlung der Fehler protokolliert. Dieses sog. **Logging** wurde mit Hilfe der Java-Bibliothek **log4j2** umgesetzt. Die Protokollierung der Fehler ist in einer gesonderten Datei konfigurierbar.

Um die Code-Qualität weiter zu steigern, wurde darauf geachtet, dass bevorzugt Funktionen aus den verwendeten Bibliotheken genutzt werden, anstatt diese selbst zu implementieren. So wurde etwa eine vorher umgesetzte Funktion, welche zusätzliche Klassen und Namensumgebungen benötigte, komplett in PrimeFaces überführt.

Im Zuge der Anpassung von funktionalen Änderungen wurden auch die Dokumentation und Formatierung des Quellcodes angepasst, um den Richtlinien der Projektgruppe zu entsprechen. Sämtliche Methoden und Klassen sind mit **JavaDoc** Kommentaren versehen, welche in englischer Sprache verfasst sind.

Am Ende dieser Arbeitspaketdokumentation ist ein Klassendiagramm der Weboberfläche aufgeführt, das sich in drei Kategorien unterteilen lässt:

- Nutzerverwaltung (vgl. Abbildung 124 auf Seite 236)
- Testrecords (vgl. Abbildung 125 auf Seite 237)
- Einzelne (Hilfs-)Klassen (vgl. Abbildung 126 auf Seite 237)

**Technologien**

**log4j2** Die Protokollierung ist mittels der Java-Bibliothek **log4j2** umgesetzt. Diese ermöglicht, entsprechende Warnungen oder Fehler frei konfigurierbar an beliebiger Stelle zu protokollieren.

**easyUML** Zur Generierung eines Klassendiagramms auf Basis von bestehendem Quellcode wurde easyUML (ein Plugin für die Entwicklungsumgebung NetBeans) genutzt.

**Systembeschreibung** Das PrimeFaces-Frontend ist in den Entwicklungsumgebungen Eclipse und NetBeans entwickelt worden. Andere Bibliotheken werden mit Hilfe von Maven eingebunden. Die Anwendung wurde auf einem Apache-Tomcat Webserver evaluiert.

**Evaluation** Im Anschluss an die Code-Restrukturierung wurde ein Funktionstest sämtlicher bisher implementierter Funktionen durchgeführt um zu gewährleisten, dass während der Code-Restrukturierung unberührter Code durch angepassten Quellcode nicht beeinflusst wird. Ehemalige Abhängigkeiten wurden dabei neu referenziert und die Gesamtkomponente erneut einem Funktionstest unterzogen. Der Funktionstest konnte abschließend ohne Fehler durchgeführt werden.

**Parameter** Die Anwendung wird von keinen Parametern eingeschränkt.

**Fazit** Das Code-Restrukturierung konnte erfolgreich abgeschlossen werden. Jeglicher alter bzw. nicht verwendeter Quellcode ist entfernt. Der verwendete Code entspricht zudem den Qualitätsnormen und Konventionen des MEDIC-Projektes in Hinsicht auf Coding-Guidelines, Dokumentation und Code-Qualität. Letzteres wurde durch den QM-Beauftragten des Projekts geprüft und abgenommen.

**Ausblick** Im Anschluss an die Code-Restrukturierung kann die weiterführende Arbeit am Frontend vorgenommen werden, wie etwa eine Analyse der Performanz, Anpassungen zur Verbesserung der Gebrauchstauglichkeit, das Implementieren von Modultests und die Erweiterung der Benutzerverwaltung.

#### 4.4.14 Erstellung eines Teststandes für reproduzierbare Testdaten

- Priorität: Normal - Dieses Arbeitspaket wird nicht direkt von anderen Arbeitspaketen beeinflusst, muss aber auf Grund zeitlicher Aspekte am Anfang des Sprints begonnen werden.
- Motivation und Nutzen des Arbeitspaketes:
  Um die Algorithmen der Bildverarbeitung zukünftig unter reproduzierbaren Bedingungen Testen zu können, wird ein System benötigt, welches diese Funktionalität zur Verfügung stellt.
- Arbeitspaketbeschreibung:
  Am Ende des Arbeitspaketes ist ein Teststand vorhanden, welcher den Teststreifen und das mobile Endgerät unter verschiedenen Winkeln „einspannen“ kann. Zudem existieren zwei oder mehrere Bereiche, in welchen Lampen mit unterschiedlichen Lichtern und Vorrichtungen für die Erzeugung von Schatten positioniert werden können. Der Teststand ist zudem so gestaltet, dass die Testdaten zu jeder Zeit reproduziert werden können (Winkel von Teststreifen und mobilem Endgerät, Farbeinfall, Farbtemperatur, Schattenposition usw.). Weitere sinnvolle Ideen können optional hinzugefügt werden. Ein möglicher Aufbau des Teststandes kann der Abbildung 127 entnommen werden.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Zusätzlich zum Teststand soll die letzte Woche des Arbeitspaketes dazu genutzt werden, mit dem oben beschriebenen Teststand, Testdaten für den weiteren Verlauf des Projektes zu generieren.

- Aufwand:
  - sechs Wochen
- Personen:
  - Timo Raß
  - Daniel Wegmann
  - Sebastian Horwege
  - Raphael Kappes

**4.4.14.1 Erstellung eines Teststandes** Der Teststand soll dazu dienen, die Qualitätsanforderungen um die Bildverarbeitung im Lastenheft für die Bildverarbeitungsalgorithmen korrekt und Messbar umzusetzen.

**Ablauf** Der allgemeine Ablauf für dieses Arbeitspaket lässt sich in folgende Abschnitte einteilen:

**Allgemeiner Aufbau des Teststandes**
Zunächst wurde damit begonnen, den grundsätzlichen Aufbau des Teststandes zu entwickeln. Dabei entstanden zunächst drei verschiedene Vorschläge, wie der Aufbau des Teststandes aussehen kann. Zur ersten Evaluierung des möglichen Aufbaus wurde ein Schuhkarton als Testobjekt verwendet, welcher den Teststand simulieren sollte.

- Idee 1: Ein vertikaler Aufbau, sodass das Smartphone oben auf dem Teststand liegen würde und der Teststreifen, wie eine Art Fahrstuhl, näher an das Smartphone herangefahren oder weiter entfernt werden kann. Dieser Aufbau kann Abbildung 128 entnommen werden.


Abbildung 128: Prototyp des Teststands - Smartphone lieft auf dem Teststand, vertikale Ausrichtung

- Idee 2: Ein horizontaler Aufbau des Teststandes, bei welchem das Smartphone vor dem Teststand platziert wird. Der Teststreifen kann so auf Schienen dichter an das Smartphone herangefahren oder weiter entfernt werden. Siehe Abbildung 129 auf der nächsten Seite und 130 auf der nächsten Seite.


Abbildung 129: Draufsicht Prototyp - Smartphone vor dem Teststand, horizontale Ausrichtung


Abbildung 130: Prototyp von hinten - Smartphone vor dem Teststand, horizontale Ausrichtung

- Idee 3: Ein horizontaler Aufbau des Teststandes, bei welchem das Smartphone innerhalb des Teststtandes platziert wird. Siehe Abbildung 131.

Abbildung 131: Prototyp Smartphone innerhalb des Teststandes, horizontale Ausrichtung

Nachdem diese Möglichkeiten ausreichend getestet und diskutiert wurden, ergab sich die Erkenntnis, dass ein horizontaler Aufbau, bei welchem das Smartphone vor dem Teststand positioniert wird, am effizientesten ist. Das Smartphone kann von außerhalb konfiguriert und

getauscht werden. Zudem kann beobachtet werden, wie sich der Algorithmus verhält und unter welchen Bedingungen der Algorithmus den Teststreifen identifizieren kann. Dies wäre nur bedingt möglich, wenn das Smartphone innerhalb des Teststandes platziert werden würde. Wichtig ist bei einem horizontalen Aufbau, dass das Smartphone mit einem gewissen Druck von außen an den Teststand gepresst wird, damit die Linse der Kamera nicht durch Umgebungslicht beeinflusst wird.

**Zusammenbau eines Gerüstes für den Teststand mit Bosch-Systemen**

Der Teststand besteht aus einem System von Bosch Profilen [41]. Die Maße sind hier der Abb. 132 zu entnehmen.

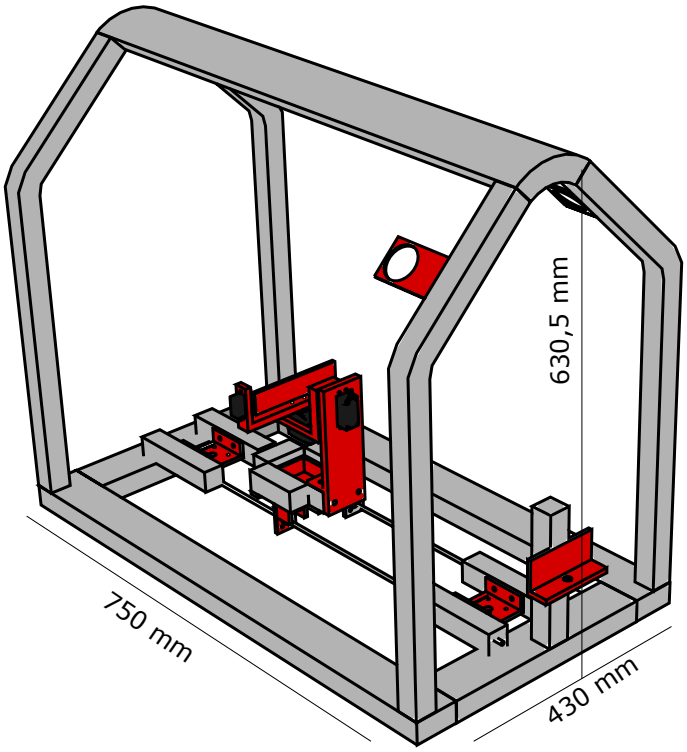



Abbildung 132: Ein 3D-Modell des Teststandes

Das Grundgerüst des Teststandes konnte früh entwickelt werden, da einige Abmaße ersichtlich waren:

- **Höhe:**
  Der Teststand muss eine gewisse Höhe aufweisen. Diese lässt sich aus dem Konzept ermitteln, für das sich früh entschiedenen wurde. Der Teststand muss Platz für einen kippbaren Tisch bieten, dieser wird im späteren kardanische Aufhängung [42], Tisch, Teststreifenhalterung oder Stativ für den Teststreifen genannt, auf dem der Teststreifen angebracht wird. Der Tisch muss so hoch sein, dass ein Tablet, z.B. das iPad Air 2 mit 24 cm Höhe [43], aufrecht mit der Kamera durch das Loch den Teststreifen aufnehmen kann. Weiterhin muss die Lichtanlage oberhalb der kard. Aufhängung angebracht werden, da eine Lichtbestrahlung von unten nicht dem Standard entspricht. Der Standard ist hierbei, dass Räume oder die Umgebung von oben ausgeleuchtet werden [44]. Das heißt, dass für das Lichtsystem mindestens noch 20 cm gebraucht werden, abhängig von der Größe der Glühbirnen. Da ebenfalls keine direkte Bestrahlung durch das Lichtsystem erfolgen soll, muss noch Platz für die Diffusion [45] geschaffen werden, wobei diese mit 10 cm abgeschätzt wurde. Der Teststand hat eine endgültige Höhe von 63 cm, da in dem Stand nicht für diesen Zweck zugeschnittene Profile verbaut wurden. Die Benutzung nicht

[41] http://www.boschrexroth.com/de/de/produkte/produktgruppen/montagetechnik/mechanik-grundelemente/strebenprofile/index
[42] https://de.wikipedia.org/wiki/Kardanische_Aufhängung
[43] http://www.apple.com/de/ipad-air-2/specs/
[44] https://de.wikipedia.org/wiki/Wohnraumbeleuchtung#Abstimmung_auf_die_Wohnr.C3.A4ume
[45] https://de.wikipedia.org/wiki/Diffusion

zugeschnittener Profile lässt sich mit dem Preis und dem Verschnitt begründen. Das Zuschneiden der Bosch Profile hätte Verschnitt verursacht, der kaum hätte wiederverwertet werden können. Daher ist es preiseffizienter gewesen mit größeren Abmaßen zu arbeiten, als mit den abgeschätzten.

- **Länge:**
  Die Länge des Teststandes lässt sich durch einen minimalen und einen maximalen Abstand zwischen Smartphone und Teststreifen beschreiben. Der minimale Abstand ist durch Abschätzung ca. 17 cm lang. Diese Abschätzung beruht darauf, dass der Teststreifen einerseits von dem Blitzlicht der Kamera und andererseits mit momentan 13 cm Länge von der Kamera vollständig erfasst werden soll. Dabei ist der Bildwinkel eines normalen Objektives [46] 50 Grad. Wenn der minimale Abstand mittels eines Dreiecks berechnet wird, kann ein gleichschenkliges Dreieck mit einer Kantenlängen a = b = 17 cm und einem Winkel Gamma mit 50 Grad konstruiert werden, dass eine Kantenlänge c = 14,37 cm hat, wodurch der Teststreifen vollständig aufgenommen werden kann. Der maximale Abstand von 44,5 cm ist mit der Annahme einer Armlänge von 50 cm zu begründen. Die 50 cm stammen aus der Messung eines Armes und der angenehmen Haltung des Teststreifens mit diesem. Damit ergibt sich aus dem minimalen und dem maximalen Abstand eine Mindestlänge von 61,5 cm. Der endgültige Teststand hat dann die Länge von 75 cm ebenfalls auf Grund von nicht zugeschnittenen Bosch Profilen.

- **Breite:**
  Um Platz für das Arduino-Board, die kard. Aufhängung, Elektrik und Elektronik innerhalb des Teststandes zu schaffen, wurde hier eine Breite von ca. 42,5 cm angesetzt. Zum Zeitpunkt der Entwicklung des Gerüstes war die Breite schwer zu bestimmen. Dennoch ist durch den Teststreifen mindestens eine Breite von 13 cm vorausgesetzt. Die Breite des Teststandes von 42,5 cm genügt dieser Vorraussetzung und lässt Platz für zusätzliche Elemente.

Auf Grund dieser Rahmenparameter konnte der Teststand in seiner Funktionalität weiter entwickelt werden. Dazu gehört die endgültige Umsetzung der kard. Aufhängung, das Anbringen einer Lichtanlage und eines Systems, dass den Abstand zwischen Smartphone und Teststreifen regelt.

[46] https:// de.wikipedia.org/wiki/Normalobjektiv

**Aufbau Lampensystem**

Um für einheitliche Lichtverhältnisse innerhalb des Teststandes zu sorgen, wurden zwei mal zwei Glühbirnen (580 Lumen) mit unterschiedlichen Kelvinwerten verbaut [47]. Für warmes oder auch gelbliches Licht wurden zwei 2700 Kelvin Glühbirnen verbaut, für kaltes oder weißes Licht zwei 5700 Kelvin Glühbirnen. Die unterschiedlichen Lichtverhältnisse können den Abbildungen 133, 134 und 135 entnommen werden. Glühbirnen mit einem frei auswählbaren, variablen Kelvinwert hatten den Nachteil, dass der Kelvinwert nur mit dem zuschalten von gelben oder blauen LED-Leuchten erzeugt wird, was die Lichtverhältnisse verfälschen kann. Zudem sind die Lumenwerte dieser Glühbirnen zu gering und die Kosten zu hoch, als dass es sich rentiert hätte diese Glühbirnen zu verbauen.
Wie zudem auf den Abbildungen 133, 134 und 135 zu erkennen ist, sind die Lampenhalterungen vor dem Teststreifen platziert. diese Position hat den Vorteil, dass das Licht nicht direkt in die Linse der Kamera scheint und das Bild so verfälscht werden könnte.

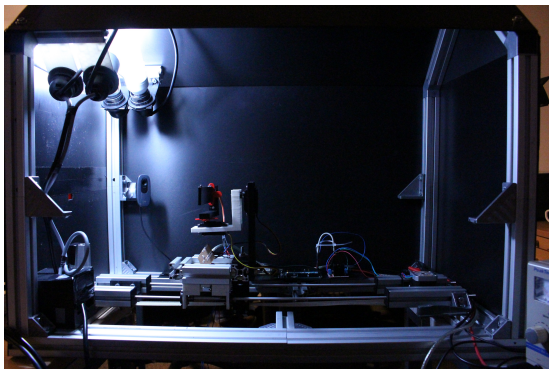


Abbildung 133: Teststand von innen mit 2700 Kelvin Glühbirnen ausgeleuchtet

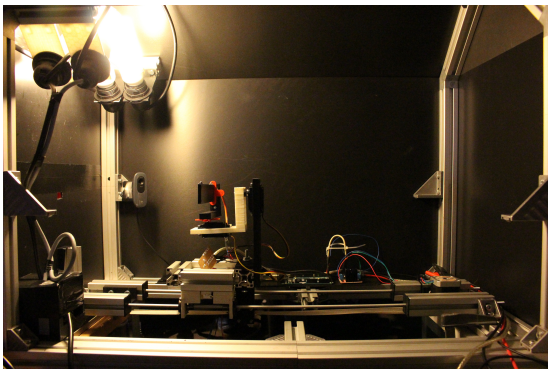


Abbildung 134: Teststand von innen mit 5700 Kelvin Glühbirnen ausgeleuchtet

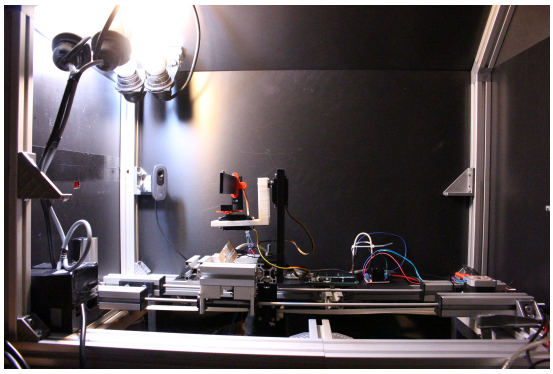


Abbildung 135: Teststand von innen mit 2700 und 5700 Kelvin Glühbirnen ausgeleuchtet

[47] Lumen= Stärke des Lichtstromes

Die Lampenhalterungen wurden mit dem Programm Sketchup entworfen und einem Replicator 5th Generation 3D gedruckt. Sie wurden so konzipiert, dass eine Befestigung am Boschsystem mit den dazugehörigen Schrauben möglich ist und normale Fassungen für Glühbirnen an ihnen angebracht werden können, siehe Abbildung 136. Der Aufbau einer montierten Lampenhalterung am Boschsystem kann Abbildung 137 entnommen werden. Diese können nach belieben auch an anderen, frei wählbaren Positionen innerhalb des Teststandes befestigt werden.

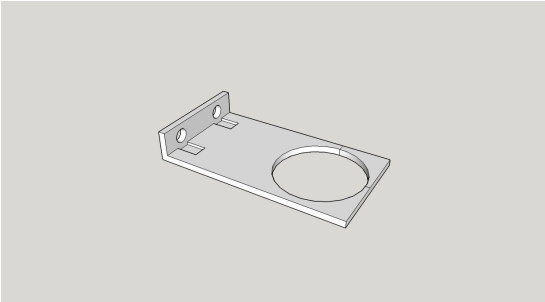


Abbildung 136: Enwurf der Lampenhalterung

Abbildung 137: Umsetzung der Lampenhalterung mit einer eingebauten Fassung

Um das Licht optimal im Teststand zu streuen wurden aus Backpapier Lampenschirme hergestellt. Diese sorgen für eine gleichmäßige Verteilung des Lichtes innerhalb des Teststandes. In diesem Zusammenhang wird auch von diffusem Licht gesprochen. Zudem sorgen die Lampenschirme dafür, Reflexionen zu vermeiden und die Umgebung im Teststand voll auszuleuchten, siehe Abbildung 138.

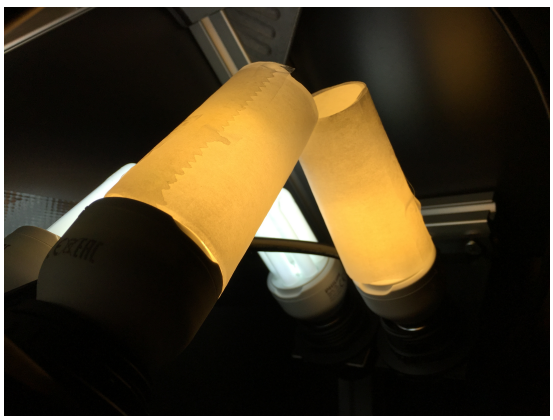


Abbildung 138: Lampenschirme aus Backpapier für die Erzeugung von diffusem Licht

**Aufbau Abschirmung vom Teststand**

Um die Umgebung des Teststandes von der Außenwelt bzw. äußeren Einflüssen abzuschirmen wurden matte Kunststoffplatten verwendet, wie Abbildung 139 zu entnehmen ist. Die Platten sind fest mit dem Boschsystem verschraubt und Schnittkanten miteinander verklebt, um das Eindringen von äußeren Lichteinflüssen so gering wie möglich zu halten. Lediglich eine Seitenwand wurde nicht verklebt und kann mit zwei Schrauben immer wieder schnell gelöst werden, siehe Abbildung 140. Dies dient dazu, die Parameter wie z. B. den Teststreifen innerhalb des Teststandes schnell wechseln zu können bzw. um eventuelle Reparaturen an den Komponenten durchführen zu können. Der Boden des Teststandes wurde zum aktuellen Zeitpunkt offen gelassen, da die unteren Schienen des Boschsystems auf ebenen Stellflächen kein externes Licht eindringen lassen.

Abbildung 139: Seitenansicht des Teststandes

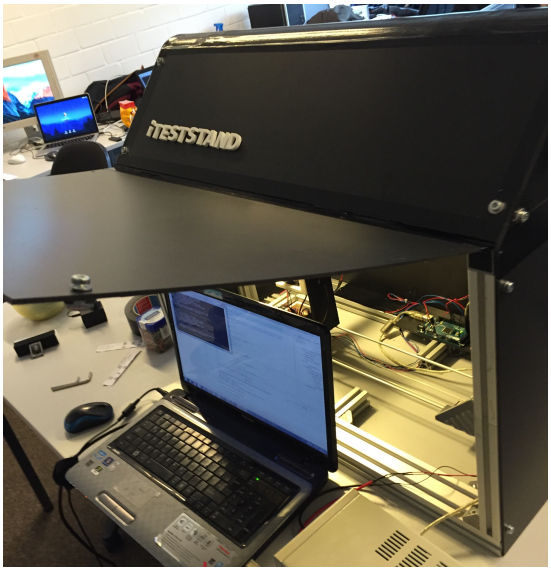



Abbildung 140: Aufklappbare Seite des Teststandes

Damit keine Reflexionen innerhalb des Teststandes auftreten, wurden bewusst schwarze Platten verwendet. Diese können eine große Lichtmenge absorbieren und Reflexionen verhindern, siehe Abbildung 141. Eine Reflexion innerhalb des Teststandes kann den gesamten Bildverarbeitungsalgorithmus und die Testdaten verfälschen, weshalb auch in die Auswahl der Verkleidung des Teststandes einiges an Zeit investiert wurde. Andere Materialien wie z. B. verdunkeltes Plexiglas wurden ebenfalls getestet. Die Ergebnisse waren jedoch enttäuschend, da das Licht viel zu sehr reflektierte. Deshalb fiel die Wahl letztendlich auf matte Hardschaumstoff-Platten.

Abbildung 141: Innenansicht des Teststandes ohne Reflexionen

**Aufbau Smartphonestativs**

Wie bereits zu Beginn erwähnt wurde, wird das Smartphone beim aktuellen Teststanddesign vor dem Teststand positioniert. Dabei muss darauf geachtet werden, dass zwischen Kameralinse und Teststandöffnung möglichst wenig Umgebungslicht einfällt, um die Testdaten reproduzieren zu können. Zudem muss das Stativ in der Lage sein, die Höhe zu verändern, da die Smartphones und deren Kameras unterschiedlicher Größe und Position sind. Hierzu wurde ein Tisch entwickelt, auf welchem das Smartphone positioniert werden kann. Dieser Tisch kann auf einer Schiene des Boschsystems frei nach oben und nach unten bewegt werden, um so die optimale Höhe einstellen zu können. Dies kann Abbildung 142 entnommen werden. Die Abbildung 143 ist die Halterung am Teststand angefügt.

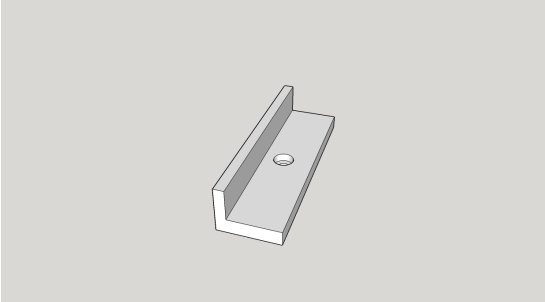


Abbildung 142: 3D Entwurf des Smartphonestativs

Abbildung 143: Stativ / Smartphonehalterung des Teststandes

Mit Hilfe von speziellem Schaumstoff, welcher mit Heißkleber auf dem Tisch befestigt wurde, können die Smartphones an die Außenwand des Teststandes heran gedrückt und fixiert werden, siehe Abbildung 144. Dieser Druck verhindert, dass das Umgebungslicht in die Linse einfällt, da diese ein paar Millimeter in den Teststand hinein gedrückt wird.

Abbildung 144: Stativ / Smartphonehalterung des Teststandes mit Smartphone

Da nicht nur Smartphones mit dem Teststand kompatibel sein sollen, wurde zudem ein größeres Stativ entwickelt, welches die Positionierung und Fixierung von Tablets ermöglicht. Abbildung 145 auf der nächsten Seite zeigt das größere Stativ.

Abbildung 145: Stativ für Tablets

Abbildung 146 zeigt das Stativ mit einem Tablet in vertikaler Ausrichtung. Auch hier wird durch den Schaumstoff ausreichend Druck ausgeübt, um das Tablet in einer Position zu fixieren.

Abbildung 146: Stativ für Tablets mit einem horizontal positionierten iPad Mini 3

Durch die verstellbare Höhe des Stativs ist es sogar möglich kleinere Tablets horizontal zu positionieren.

Abbildung 147: Stativ für Tablets mit einem vertikal positionierten iPad Mini 3

**Aufbau des Stativs für den Teststreifen**

Um die Bedienbarkeit für den Benutzer einfach zu halten, müssen Bildverarbeitungsalgorithmen den Teststreifen aus möglichst vielen Aufnahmewinkeln identifizieren und auswerten können. Die Qualität der Algorithmen kann also unter anderem aus den Intervallen der akzeptierten Aufnahmewinkeln abgeleitet werden. Aus diesem Grund ist ein wesentlicher Bestandteil des Teststandes das Stativ für die Teststreifen. Um verschiedene Aufnahmewinkel zu simulieren ist der Teststreifen in einer kardanische Aufhängung fixiert. Diese ermöglicht es, den Teststreifen um die X-, Y- und Z-Achse zu rotieren wie in Abbildung 148 zu sehen ist. Der Rotationsmittelpunkt aller Achsen befindet sich dabei exakt im Zentrum des Teststreifens. Dadurch ändert sich die Position des Teststreifens im aufgenommenen Bild nicht mit der Rotation der einzelnen Achsen. Der Vorteil dieser Aufhängung ist also die Unabhängigkeit der Position und Rotation des Teststreifens im Teststand.

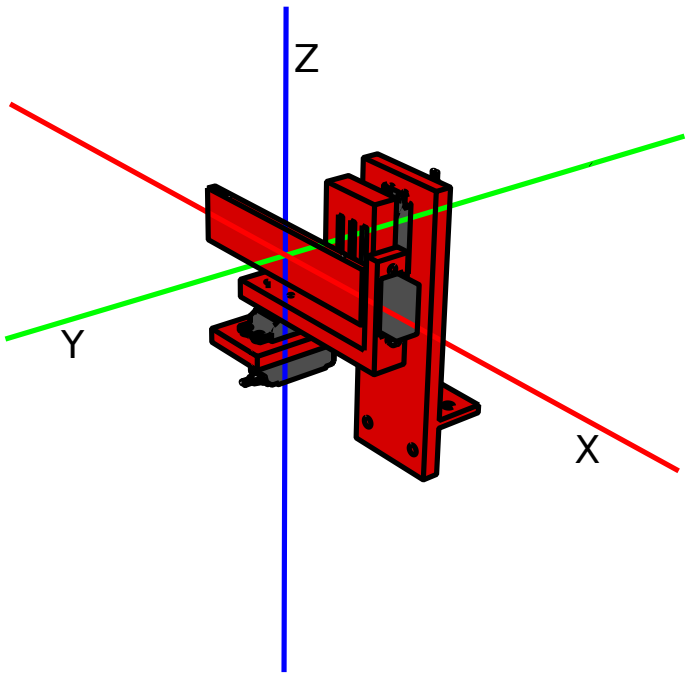
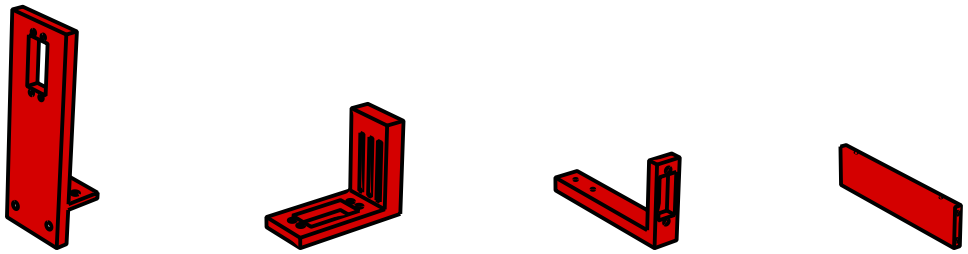



Abbildung 148: Modell der Teststreifenhalterung mit eingezeichneten Achsen

Der Aufbau des Stativs besteht aus vier, in einem 3D-Drucker hergestellten, Halterungen, die durch drei Servomotoren miteinander verbunden sind. Um das Stativ auf dem Schiffchen (siehe unten) zu fixieren, wurde die Stativhalterung in Abbildung 149a kompatibel zum Bosch-System konstruiert. In dieser Halterung ist ein Servomotor verankert, durch diesen kann das Bauteil aus Abbildung 149b um die Y-Achse rotiert werden. Der Teststreifen selbst wird in die Teststreifenhalterung in Abbildung 149d geklemmt. Diese kann über einen zweiten Servomotor, eingelassen in die Halterung aus Abbildung 149b, um die X-Achse rotiert werden. Durch ein letztes Bauteil in Abbildung 149b werden Stativhalterung und Teststreifenhalterung miteinander verbunden und die Rotation um die Z-Achse ermöglicht.



(a) (b) (c) (d)

Abbildung 149: Die gedruckten Teile des Teststreifenstativs.

Alle Komponenten des Stativs wurden des öfteren neu designt, um das Endprodukt optimal nutzen zu können. Anfänglich wurde der Teststreifen nur mit einem Rahmen an dem Tisch befestigt. Dies wurde dann in eine Platte umgestaltet, da durch eine alte Verteilung der Lampen das Licht durch den Rahmen und durch den Teststreifen schien, so dass ungewollte Schatten erzeugt wurden. Beispielhaft wird in den folgenden Abbildungen die Komponente der Teststreifenhalterung dargestellt um aufzuzeigen, welche Änderungen unter anderem vorgenommen wurden. Die Teststreifenhalterung wurde mehrmals neu konzipiert und entworfen, um ein schnelles und stabiles Einspannen des Teststreifens zu ermöglichen. Zunächst wurde der Teststreifen mit Klemmen befestigt, welche den Teststreifen jedoch oft beschädigten und Informationen von diesem verdeckten. Der neue Ansatz Magnete zu verwenden erwies sich als vorteilhaft, da der Teststreifen flexibler und schneller in der Halterung eingespannt werden kann.

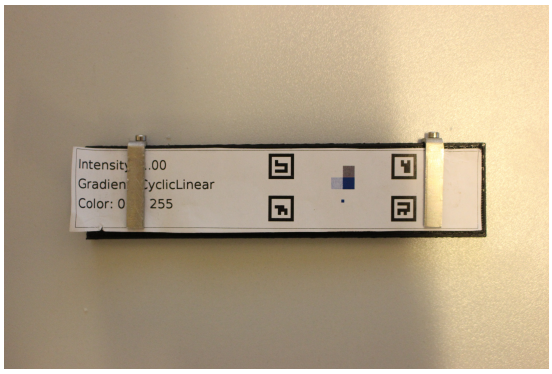


Abbildung 150: Alte Teststreifenhalterung mit Klemmen

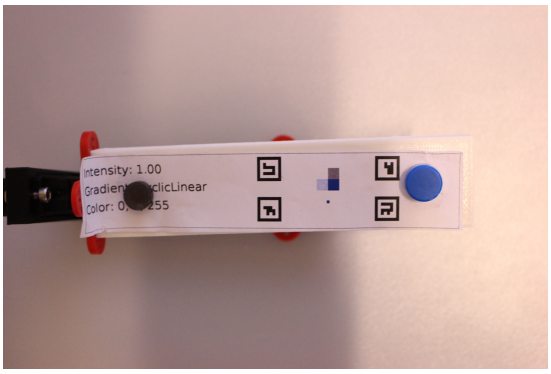


Abbildung 151: Neue Teststreifenhalterung mit eingesetzten Magneten

**Antrieb / Aufbau des Schiffchens**

Um den Rahmen, in dem die entwickelten Bildverarbeitungsalgorithmen funktionieren, zu parametrisieren, ist neben der kardanischen Aufhängung ebenso der Abstand vom Smartphone zum Teststreifen zu verändern ein messbarer Faktor. Zum einen haben ein paar der günstigen Smartphones keinen Autofokus für ihre Kamera wodurch diese in einem eingeschränkteren Bereich ein scharfes Bild aufnehmen können als Smartphones mit Autofokus. Zum anderen ist die Auflösung der Elemente des Teststreifens wie das Farbumschlagsfeld, die Referenzfelder und die Marker abhängig von der Entfernung. Um dieses im Teststand zu realisieren ist ein Antrieb entwickelt worden, der den Abstand verändert. Die kard. Aufhängung ist dabei auf einem Schiffchen (ein Aufbau der sich wie ein Schiff von Position A nach Position B bewegt wie in Abb. 152 sinngemäß gezeigt) angebracht, dass durch den Antrieb hin und her bewegt werden soll. Als erstes wird das Schiffchen und dann der Antrieb erläutert.


Abbildung 152: Ein Schiffchen, das als schwarzer Kasten dargestellt ist und sich von Position A nach Position B bewegt

- **Schiffchen**
  Das Schiffchen, auf dem die kard. Aufhängung befestigt ist, steht mit drei Kugellagern auf zwei Stangen und ist mit einer Halterung an dem sich bewegenden Seil, dass durch den Schrittmotor angetrieben wird, befestigt. Hierfür werden drei Kugellager benötigt, da eine Ebene, welche durch das Schiffchen dargestellt wird, mit mind. drei Stützpunkten definiert werden muss. Das Schiffchen besteht dabei hauptsächlich aus Bosch Profilen mit den Abmaßen 12 cm x 14cm, wodurch leicht aufbauten auf diesem angebracht und gestaltet werden können. Des Weiteren stellt es eine Schnittstelle bereit, an die ein Kabel angeschlossen werden kann. Dadurch beschädigen die Kabel der Motoren, die den Tisch drehen, nicht andere Bauteile und somit ist das Schiffchen ein abgeschlossenes System mit der kard. Aufhängung, welches angesteuert werden kann. In Abbildung 153 wird das Schiffchen nochmal gezeigt.

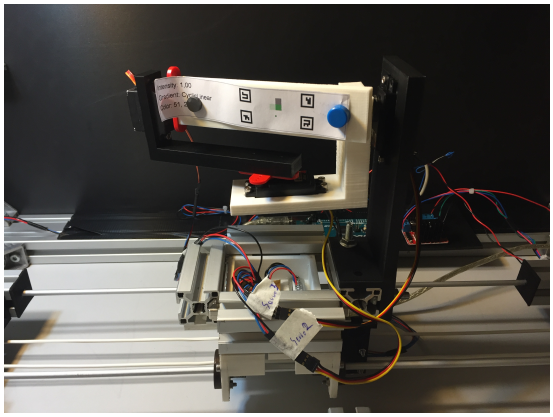


Abbildung 153: Das Schiffchen des Teststands

- **Linearantrieb**
  Die **Definition** für einen solchen Antrieb ist:

Als Linearantrieb oder Linearantriebssystem werden alle Antriebssysteme bezeichnet, die zu einer translatorischen Bewegung (einer linearen oder gradlinigen Bewegung) führen. Linearantriebe ermöglichen die Bewegung von Maschinen(elementen) und Anlagen(teilen) in gerader Linie oder einem anderen vorgegebenen Verlauf. [48]
Hierzu wurden zwei Konzepte ausgearbeitet und getestet, die im folgenden allgemein beschrieben werden:

- **Spindelantrieb**
  Dieser Antrieb setzt sich aus einer Spindel und einem Motor zusammen, die einen Schlitten oder Gondel über eine Halterung translatorisch [49] bewegen kann [50] [51].

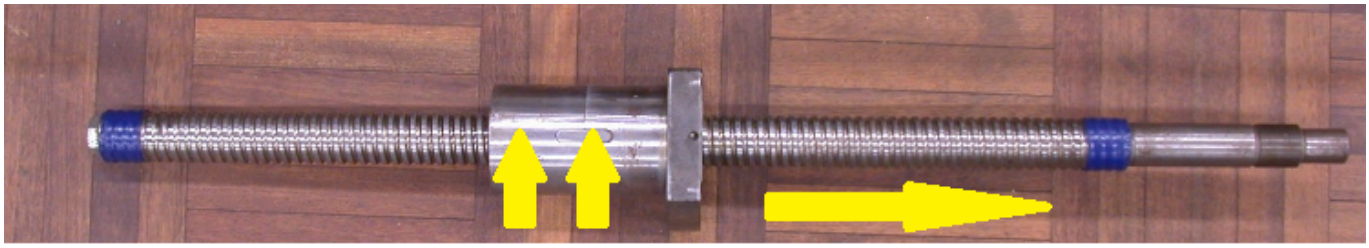


Abbildung 154: Eine Gewindespindel, auf der Schemenhaft die Bewegungsrichtungen eingezeichnet sind

Dieses Konzept ist für Wiederholbarkeit der Position ideal, da das Gewinde einer Gewindestange immer den selben Abstand hat, dadurch eine hohe Präzision (Abhängig von der Fertigung der Gewindestange) besitzt und die Position nicht durch Schwingungen verändert werden kann. Zusätzlich muss ein Haltepunkt für das zu bewegende Objekt bereit gestellt werden, damit dieses sich nicht mit der Gewindestange mitdreht. Der Haltepunkt darf sich nicht in rotatorischer Richtung bewegen, aber in Richtung der umgesetzten tranlatorischen Bewegung.

- **Seiltrieb**
  Als Seiltrieb oder Seiltransmission bezeichnet man eine Vorrichtung zur Übertragung einer Drehbewegung von einer Welle auf eine andere mittels Seilen [52]. In Abbildung 155 ist das Schema dafür zu sehen.

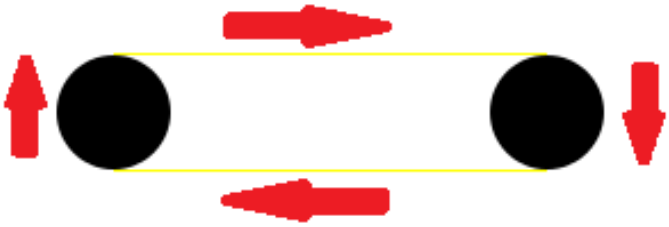


Abbildung 155: Eine schematische Darstellung eines Seiltriebes

Hierbei wurde folgendes Unterschieden:

* Seiltrieb mit Riemen:
  Mit einem Riemen, also einem flachen Seil oder Kabel, haben die Wellen mehr Auflagefläche und die Kraft der Antriebswelle kann besser übertragen werden.

[48] https://de.wikipedia.org/wiki/Linearantrieb
[49] https://de.wikipedia.org/wiki/Translation_(Physik)
[50] https://de.wikipedia.org/wiki/Gewindespindel
[51] https://de.wikipedia.org/wiki/Spindelmotor
[52] https://de.wikipedia.org/wiki/Seiltransmission

* Seiltrieb mit Zahnriemen:
  Mit einem Zahnriemen kann die Kraft der antreibenden Welle noch besser Übertragen werden, wenn die Welle Einlässe für die Zähne des Riemens hat.

Dieses Konzept hat ebenfalls eine hohe Wiederholbarkeit der Position, sowie eine hohe Präzision. Die Wiederholbarkeit und die Präzision entstehen dadurch, dass das Seil oder der Riemen sich nicht in ihrer Länge ändern. Das zu bewegende Objekt kann an einem Haltepunkt befestigt werden, aber es kann ebenso von dem Seil mitgezogen werden, da die Drehbewegung nicht auf das Objekt direkt übertragen wird. Das Objekt folgt dabei dem Verlauf des Seils oder Riemens.

Zunächst wurde das Konzept des Spindelantriebs verfolgt, das jedoch eine hohe Reibung aufweist, da kein Kugellager für die Kraftübertragung zwischen Schiffchen und Gewindestange eingesetzt wurde. Durch die hohe Reibung hätte ein sehr starker Motor eingesetzt werden müssen. Sowohl der starke Motor als auch ein Kugellager für die Kraftübertragung hätten den Preisrahmen des Projekts überstiegen. Von daher musste ein neues Konzept aufgegriffen werden und zwar der Seiltrieb. Dieses Konzept sollte als erstes über einen Zahnriemen umgesetzt werden, konnte jedoch durch Lieferschwierigkeiten der benötigten Teil nicht so umgesetzt werden. Von daher wurde ein flaches Kabel benutzt und zwischen zwei Wellen gespannt. Die Antriebswelle wurde mit einem Schrittmotor [53] bewegt.
Für die Schrittmotorsteuerung musste ein zusätzliches elektronisches Bauteil, eine H-Bridge [54], benutzt werden, um die Vorwärts- und Rückwärtsbewegungen zu steuern. Die Position des Schiffchens, also der Abstand zwischen Smartphone und Teststreifen, kann durch den Schrittmotor mit zählen der Schritte bestimmt werden. Dabei muss jedoch beachtet werden, dass die Position des Schiffchens verändert kann, ohne das der Motor die Schritte mitzählt. Dafür wurden ebenfalls verschiedene Konzepte ausgearbeitet.

- Infrarotsensor:
  Der Abstand des Schiffchens zu einem Infrarotsensor wird durch die Reflexion von Infrarotstrahlen mit dem Sensor gemessen und so die Position des Teststreifens relativ zum Smartphone bestimmt.
- Ultraschallsensor:
  Der Abstand des Schiffchens zu einem Ultraschallsensor wird mittels Schallwellen, die auf das Schiffchen treffen und teilweise reflektiert werden, gemessen und so kann die Position des Teststreifens relativ zum Smartphone bestimmt werden.
- Kamera:
  Der Abstand kann mittels einer Kamera, die auf ein Millimeterlineal gerichtet ist, abgelesen werden, wodurch die Position des Teststreifens relativ zum Smartphone bestimmt werden kann.
- Taster:
  Der Abstand des Schiffchens kann mittels eines Taster bestimmt werden, in dem das Schiffchen zunächst in Richtung des Tasters bewegt wird und von da aus eine Position mittels des Schrittmotors anfährt. Dabei ist diese Position des Tasters fest und die Position des Teststreifens relativ zum Smartphone kann ebenfalls angefahren werden.

Hier wurde im Endeffekt das Konzept des Tasters verfolgt, da alle anderen Konzepte die gefahrene Strecke des Schrittmotor messen, mit diesem jedoch das selbe möglich ist.
In Abbildung 156 auf der nächsten Seite ist das Konzept des Tasters nochmal verdeutlicht.

[53] https://de.wikipedia.org/wiki/Schrittmotor
[54] https://de.wikipedia.org/wiki/Vierquadrantensteller


Abbildung 156: Schiffchen, dass von Position A zu Position B bewegt wurde und an der Position B den Taster betätigt

**Steuerung des Teststreifenstativs**

Die Ansteuerung der Servos und des Linearmotors erfolgt über einen Arduino Mega. Dabei handelt es sich um ein programmierbares Mikrocontroller-Board, für das eine eigene Software geschrieben wurde. Diese Software ist in der Lage die Motoren auf definierte Positionen einzustellen und so sowohl die Winkel des Stativs als auch die Position des Schiffchens zu verändern. Der Arduino ist in Abbildung 157 zu sehen.

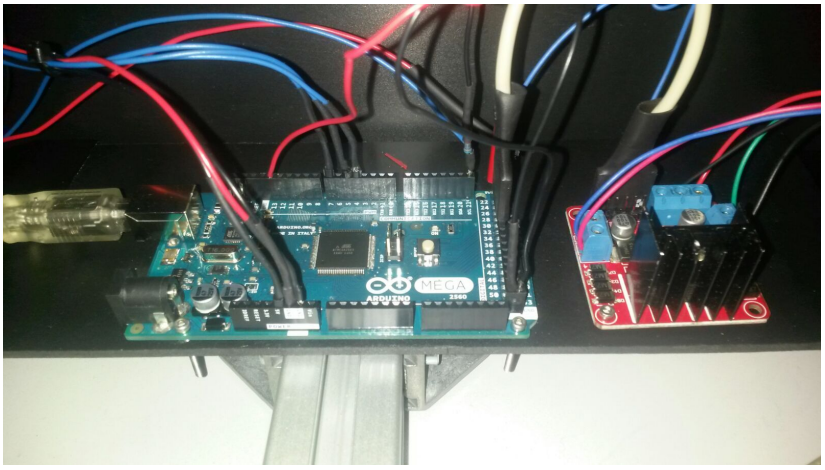


Abbildung 157: Der Arduino Mega mit angeschlossenen Kabeln

Da zu Beginn der Entwicklung die Anforderungen an den Controller noch nicht klar waren wurde die Entscheidung getroffen mit dem Mega den leistungsstärksten Arduino einzusetzen. So wurde sichergestellt, dass der Controller den Anforderungen entspricht. Dies betrifft sowohl die Anzahl verfügbarer Pins zum Anschluss weiterer Hardware als auch die benötigte Prozessorleistung.

Zur Steuerung des Teststands wurde ein Interface entwickelt, das auf einem Windows-PC ausgeführt und über die serielle Schnittstelle des Arduinos kommunizieren kann. Das Interface wurde in Java implementiert und bietet zwei Abschnitte (Tabs), einer für die manuelle Kontrolle der Stativ-Positionen und ein weiterer Tab für das automatische Durchlaufen eines zuvor festgelegten Ablaufs. Dem übergeordnet ist die Auswahl des Ports, über den die Kommunikation erfolgen kann. Nachdem der gewünschte Port ausgewählt ist, kann über den Button Connect die Verbindung zur seriellen Schnittstelle hergestellt werden.

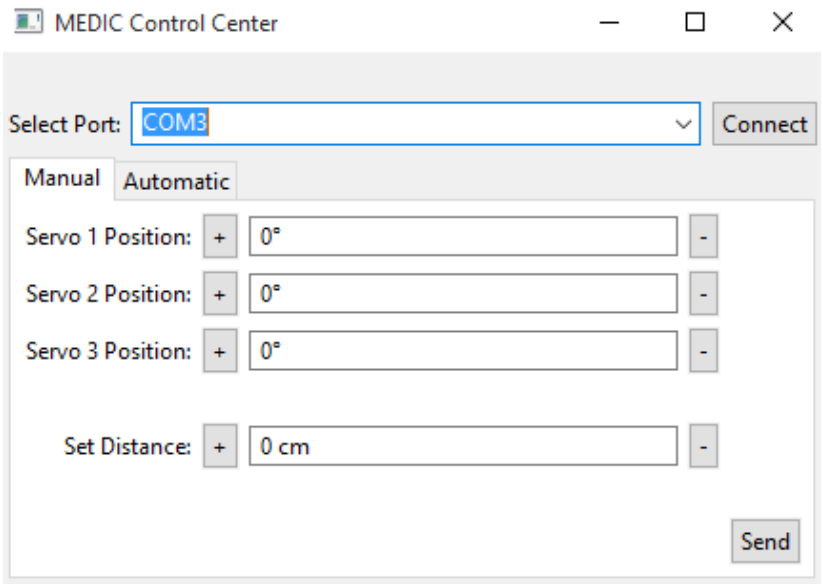



Abbildung 158: Der Manual Tab des Kontrollprogramms

Der Manual Tab ermöglicht die manuelle Einstellung der Positionen und ist in Abbildung 158 auf der vorherigen Seite zu sehen. Dazu existiert für jede Achse ein Eingabefeld, in dem der gewünschte Winkel eingestellt werden kann. Dabei ist es sowohl möglich den Winkel direkt einzutragen, als auch über die + und - Buttons den Winkel in 1 Grad-Schritten zu verändern. Es ist zu beachten, dass die Winkel relativ zur Nullstellung des Systems gewählt werden müssen. Das heißt, dass sowohl positive als auch negative Gradzahlen möglich sind. Da die Servomotoren eine Bewegungsreichweite von insgesamt 180 Grad haben entsteht so ein Spielraum von -90 Grad bis +90 Grad. Darüber hinaus existiert ein weiteres Eingabefeld zur Steuerung des Linearantriebs. Hier kann die gewünschte Position in Zentimeter eingegeben werden, auf die das Schiff mit dem Stativ bewegt werden soll. Sind die gewünschten Eingaben getroffen können die Werte über den Send Button an den Arduino geschickt werden, der die Motoren automatisch auf die entsprechenden Positionen einstellt. In den Eingabefeldern ist es nicht möglich, ungültige Werte wie Buchstaben oder Sonderzeichen einzugeben, mit Ausnahme der jeweiligen Einheit (Grad für die Winkel und cm für den Linearantrieb). Sollten falsche Eingaben getroffen werden, so wird eine Fehlermeldung angezeigt und das Senden der Einstellungen ist nicht möglich.
Dazu siehe Abbildung 159. Die Fehlermeldungen bleiben so lange bestehen, bis korrekte Werte eingegeben wurden.

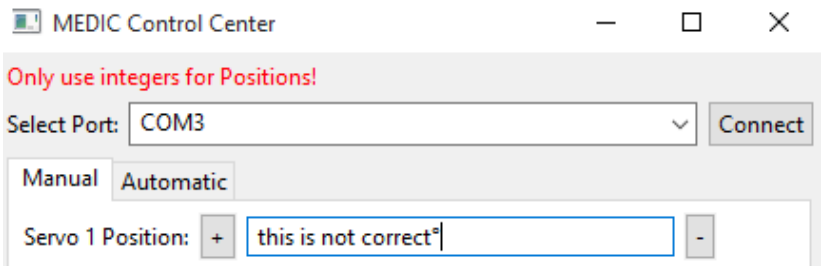



Abbildung 159: Fehlermeldung bei fehlerhaftem Eintrag im Eingabefeld

Abbildung 160 auf der nächsten Seite zeigt den Automatic Tab. Dieses Programm muss zuvor in einer Datei festgelegt werden und einem definiertem Format folgen. In jeder Zeile der Datei muss jeweils ein Eintrag für die Winkel des Stativs vorhanden sein. Die einzelnen Einträge müssen dabei durch Kommata voneinander getrennt sein. Im Path Textfeld muss der Pfad dieser Datei eingegeben werden. Dies kann entweder manuell geschehen oder über den Browse Button. Dieser Button öffnet einen Standard Windows-File Dialog, über den die gewünschte Datei ausgewählt werden kann. Wenn der eingegebene Text einem gültigem Dateipfad entspricht kann die Ausführung des Testdurchlaufs durch Betätigen des Execute Buttons gestartet werden. Hierbei wird zunächst überprüft, ob das Format der angegebenen Datei korrekt ist. Ist dies nicht der Fall wird wiederum eine Fehlermeldung angezeigt und die Ausführung abgebrochen (Abbildung 161 auf der nächsten Seite links). Handelt es sich um eine korrekte Datei, werden die einzelnen Schritte eingelesen. Daraufhin startet die eigentliche Ausführung des Testdurchlaufs in einem asynchronen Prozess, der jederzeit über den Stop Button abgebrochen werden kann. Die einzelnen Schritte werden an den Arduino gesendet und für jeweils fünf Sekunden gehalten. Dies dient der Aufnahme eines Testvideos, das zur Analyse einer großen Anzahl von Winkeln genutzt werden kann.

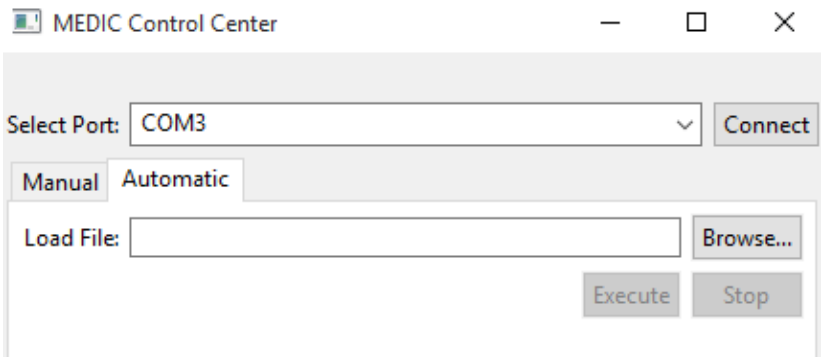



Abbildung 160: Der Automatic Tab des Kontrollprogramms

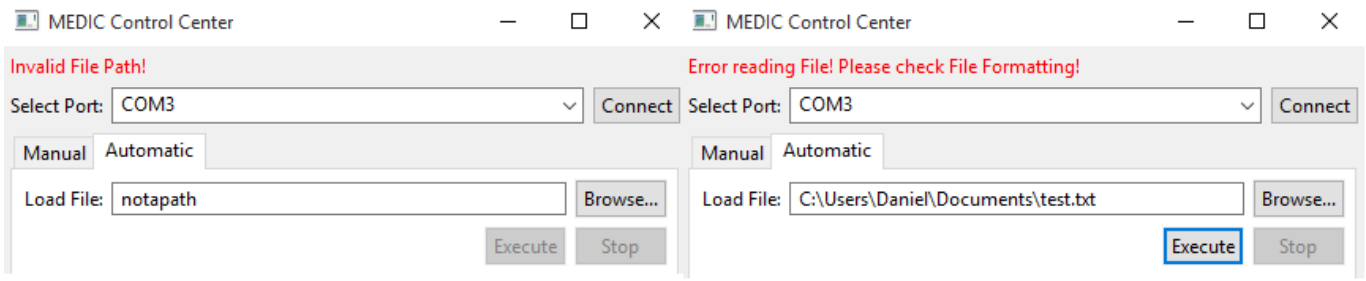



Abbildung 161: Links: Fehlermeldung bei ungültigem Dateipfad, Rechts: Fehlermeldung bei fehlerhaftem Dateiformat

**Technologien** Für die Entwicklung des Teststands wurden verschiedene Technologien eingesetzt. Diese werden im Folgenden kurz beschrieben.

**Arduino Mega** Bei Arduino handelt es sich um eine Familie von programmierbaren Hardware-Boards. Der Arduino Mega ist dabei der größte und leistungsstärkste Vertreter der Familie und wurde gewählt, da im Voraus noch nicht klar war, wie hoch die Anforderungen im Bezug auf Anzahl von benötigten Pins sind.

**Sketchup Make** Ein Programm zur Erstellung dreidimensionaler Designs. Unter anderem wurden hiermit das Stativ und die Halterungen der Lampen entworfen.

**MakerBot Replicator 5th Generation** 3D-Drucker zur Herstellung von Strukturen aus Plastik-Filament. Die in Sketchup entworfenen Designs konnten so in physikalische Objekte überführt und im Aufbau des Teststands verwendet werden.

**Java + SWT** Die Programmiersprache Java wurde zur Erstellung des Kontrollprogramms verwendet. Das Standard Widget Toolkit (SWT) wurde dabei zur Erstellung der Benutzeroberfläche genutzt.

**Systembeschreibung** Die Systembeschreibung wurde in das Kapitel Ablauf des Arbeitspaketes mit aufgenommen.

**Evaluation** Der Teststand konnte bereits für eine kleine Evaluation genutzt werden. Dafür wurde der aktuelle Bildverarbeitungsalgorithmus, wie er zum Abschluss des Arbeitspakets „Integration der Bildalgorithmen in C++“ zur Verfügung stand, als Desktopapplikation ausgeführt. Das Eingabevideo, aufgenommen mit dem MEDIC iPad, zeigt den Teststreifen in 169 Orientierungen. Die eingestellten Winkel reichen von -80 bis 70 Grad auf der X-Achse, von -25 bis 25 Grad auf der Y-Achse und -90 bis 85 Grad auf der Z-Achse. Die Schrittgröße der Winkeln beträgt 25 Grad. Als Beleuchtung wurden die Lichtquellen mit 5700 Kelvin gewählt. Da es sich hierbei um eine erste prototypische Evaluation handelt, kam es leider bei einigen Winkeleinstellungen im Randbereich der Gradwerte zu Aufnahmen, die den Teststreifen selbst für das menschliche Auge unerkennbar machten. Ein Beispiel dafür ist der folgenden Abbildung 162 zu entnehmen.

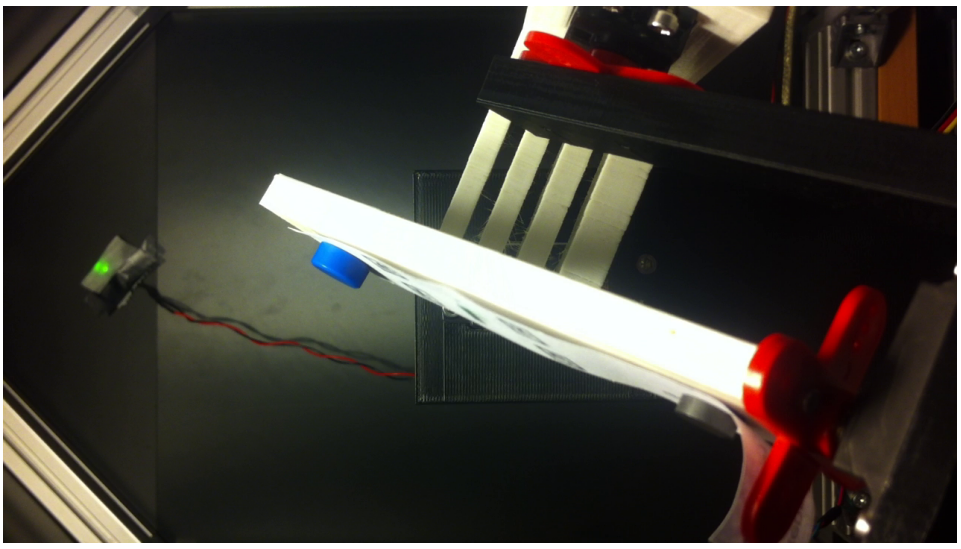


Abbildung 162: Orientierung des Teststreifens

Als Auswertungskriterium wurde die Anzahl der richtig erkannten Marker gewählt. Daraus ergibt sich automatisch für jede Orientierung des Teststreifens eine Bewertung des Algorithmus, die von 0

(keine erkannten Marker) bis 4 (alle Marker wurden richtig erkannt) reicht. In Abbildung 163 ist die Qualität des Algorithmus für jede Orientierung nach der eben beschriebenen Bewertung abgebildet. Dabei sind die Werte 0 bis 4 als Farbübergang von rot nach grün kodiert.

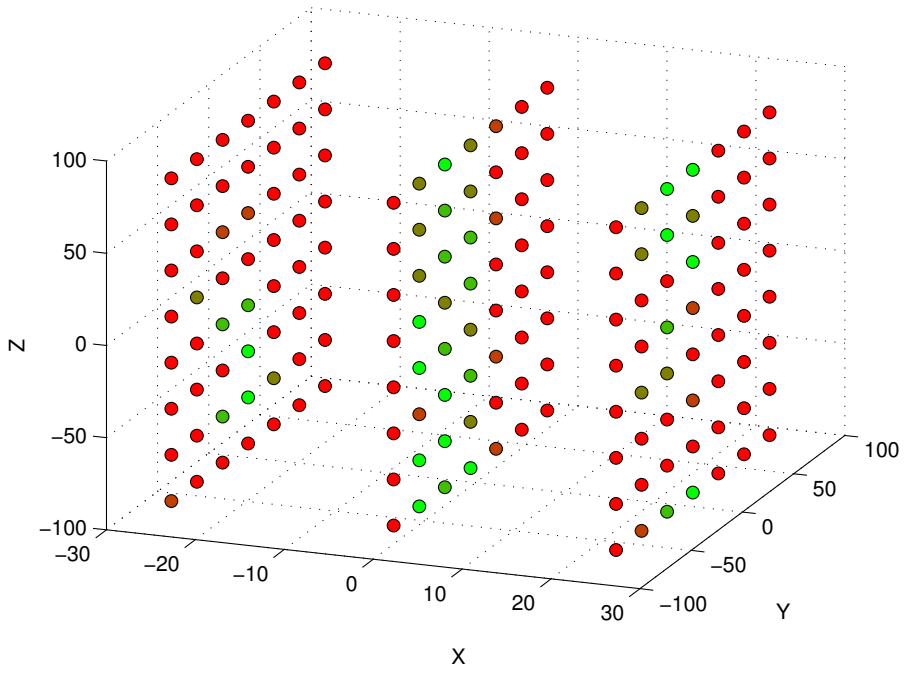



Abbildung 163: Erste Auswertung des Algorithmus mithilfe des Teststands

**Parameter**

- Entfernung und Position des Teststreifen zum Smartphone
- Horizontaler Winkel des Teststreifens
- Vertikaler Winkel des Teststreifens
- Höhe des Stativs für das Smartphone
- Lichtverhältnisse - 2700 Kelvin und 5700 Kelvin
- Endgerät - Smartphone, Tablet

**Fazit** Der Teststand bietet im momentanen Zustand eine solide Quelle für reproduzierbare Testdaten. Durch den Teststand ist es jetzt möglich die Qualitätsanforderungen an die Bildverarbeitungsalgorithmen zu testen und die Anforderungen entsprechend des Lastenheftes zu erfüllen. Daher kann gesagt werde, dass die Erstellung des Teststandes eine notwendige Entwicklung war, um das Endprodukt entsprechend der Kundenwünsche ausliefern zu können. Zudem kann der Teststand zu jeder Produktpräsentation verwendet werden um die Verbesserungen der Bildverarbeitungsalgorithmen unter identischen Bedingungen vorzuführen.
Alle Beteiligten konnten durch die interdisziplinäre Ausrichtung des Arbeitspaketes einen großen Lerneffekt in unterschiedlichsten Bereichen erzielen. Dies resultiert unter anderem daraus, dass viele Konzepte und Prototypen für unterschiedlichste Komponenten entwickelt und deren Funktionsweisen getestet werden konnten. Der anfängliche Gedanke, alle Komponenten manuell von Hand zu steuern wurde positiver Weise nicht weiter verfolgt, da eine digitale Ansteuerung wesentlich genauer und komfortabler ist. Zudem ist es nun möglich den Teststand um weitere Funktionen zu ergänzen. Auf diese wird im Kapitel Ausblick in diesem Arbeitspaket genauer eingegangen.

Nicht sehr optimal bei diesem Arbeitspaket war, dass es oft zu lange gedauert hat, bis erarbeitete Konzepte in die Tat umgesetzt wurden. Dies lag nicht immer am Engagement der Teammitglieder sondern oft auch an misslungenen 3D Drucks oder anderen Materialien, welche im Baumarkt oder im Internet eingekauft werden mussten. Dies führte dazu, dass auch kleine Änderungen oft mit viel Arbeitszeit verbunden waren. Eine Zeit von sechs Wochen war daher eine angebrachte Zeit, um den Teststand in eine Arbeitsfähige Version zu bringen.

**Ausblick** Der Teststand besitzt momentan kein internes Netzteil, wodurch mehrere Kabel von Außen nach Innen verlegt werden müssen, um die Komponenten mit Strom zu versorgen. Dafür soll zu einem späteren Zeitpunkt eine Schnittstelle mit einem entsprechenden Netzteil geschaffen werden, so dass der gesamte Stand nur von einem Kabel mit Strom versorgt wird und mit einem USB-Anschluss für die Programmierung und Steuerung verbunden ist. Weiterhin kann, sobald die Lieferung des Zahnriemens erfolgt ist, der Linearantrieb verbessert werden. Zusätzlich bietet der Teststand nicht nur die Möglichkeit Bildverarbeitungsalgorithmen auf dem Smartphone sondern im Allgemeinen diese Algorithmen zu testen. Dies hängt von dem Aufbau auf dem Schiffchen ab, hier der Tisch. Andere Aufbauten, die an einem Bosch Profil angebracht werden können, lassen sich leicht befestigen. Unter isolierten Lichtverhältnissen kann dann eine beliebige Bildverarbeitung auf definierte Parameter getestet werden.

### 4.4.15 Fazit des Sprints

Der Fokus dieses Sprints lag in der Bildverarbeitung. Dabei lag der Schwerpunkt nicht in der Entwicklung neuer Funktionen des späteren Endproduktes. Es wurden stattdessen Grundlagen geschaffen, sodass die Entwicklung der Algorithmen effizienter und Verbesserungen besser messbar sind. So wurde im Arbeitspaket „Integration der Bildalgorithmen in C++“(Abschnitt 4.4.12.1 auf Seite 184) die

Grundlage für eine gemeinsame Quellcode-Basis für Android- und iOS-Geräte geschaffen. Der erste Versuch für eine gemeinsame Quellcode-Basis wurde bereits in Sprint 3 (Arbeitspaket: „Integration C Algorithmus“) unternommen. Durch die Wiederaufnahme der Arbeitspakete „OpenCV in Android“, Abschnitt 4.4.4 auf Seite 162 und „OpenCV in iOS “, Abschnitt 4.4.5 auf Seite 165) konnten Hindernisse endgültig beseitigt und der dadurch entstandene Quellcode bereits innerhalb dieses Sprints in die jeweiligen mobilen Applikationen integriert werden. Des Weiteren wurde das Arbeitspaket „Erstellung eines Teststandes “(Abschnitt 4.4.14.1 auf Seite 192) innerhalb dieses Sprints durchgeführt. Der dadurch entstandene Teststand bietet die Möglichkeit, Teststreifen in reproduzierbaren Umgebungen aufzunehmen. Die nach Durchführung der beiden erwähnten Arbeitspakete entstandene Systemlandschaft beschreibt Abbildung 164.

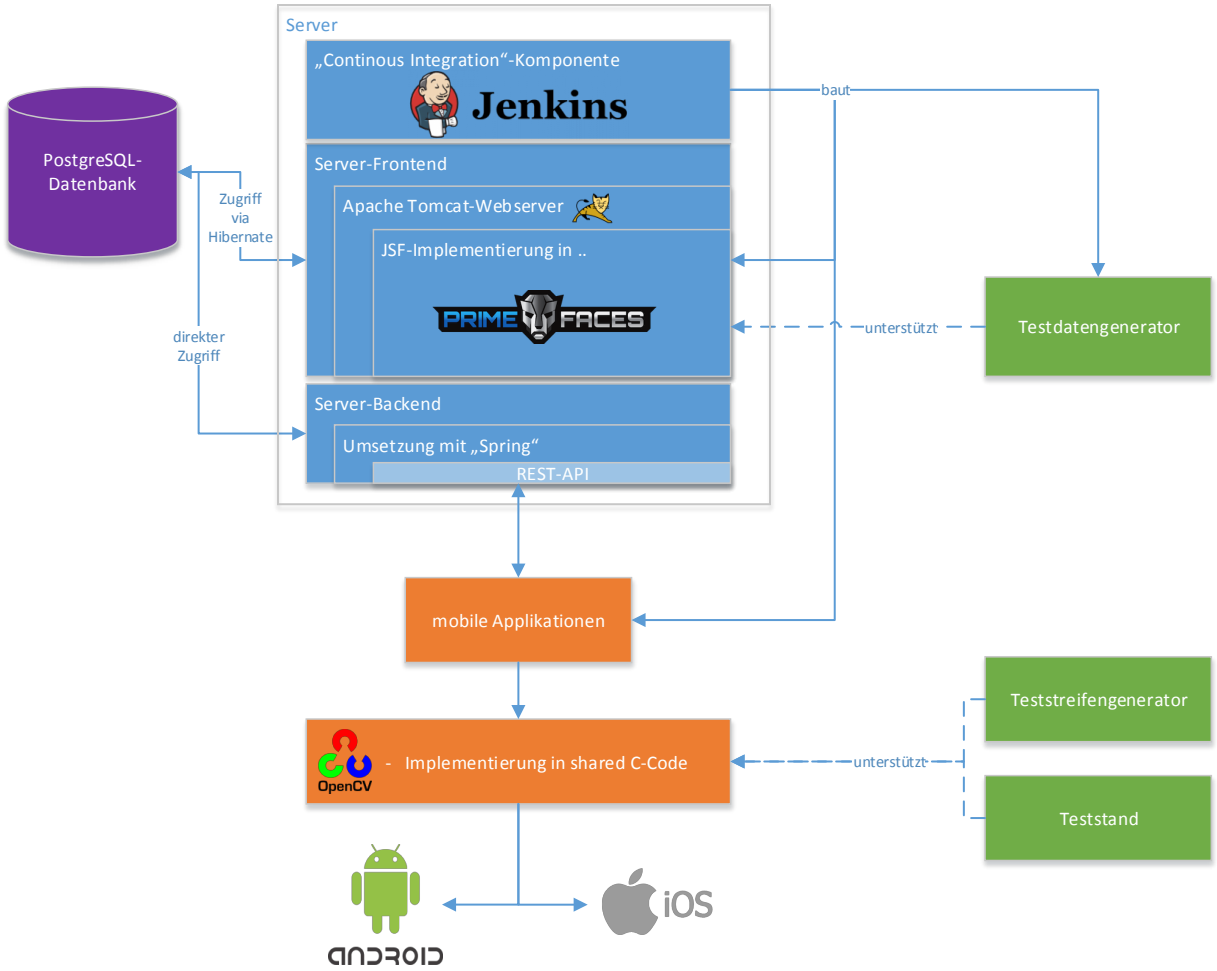



Abbildung 164: Systemlandschaft nach dem Abschluss von Sprint 5

Die nun geschaffene Systemlandschaft stellt zum ersten Mal die Möglichkeit bereit, die Qualität der Bildverarbeitungsalgorithmen standardisiert auf verschiedenen Endgeräten unter gleichbleibenden Bedingungen zu evaluieren. Dieser Umstand steigert die Effektivität und Effizienz mit der zukünftige Algorithmen implementiert und evaluiert werden können.

Des Weiteren wurde am Ende wie auch zu Beginn des Sprints eine Fortschrittsanalyse durchgeführt (vgl. Abbildung 165 auf der nächsten Seite). Diese Fortschrittsanalyse zeigt eine veränderte Systemlandschaft, jedoch keinen prozentualen Fortschritt an funktionalen Anforderungen. Dies ist damit zu begründen, dass dieser Sprint im Sinne der zukünftigen Qualitätssicherung durchgeführt wurde und keine weiteren funktionalen Anforderungen umgesetzt worden sind, was ganz im Sinne der Sprintplanung war.

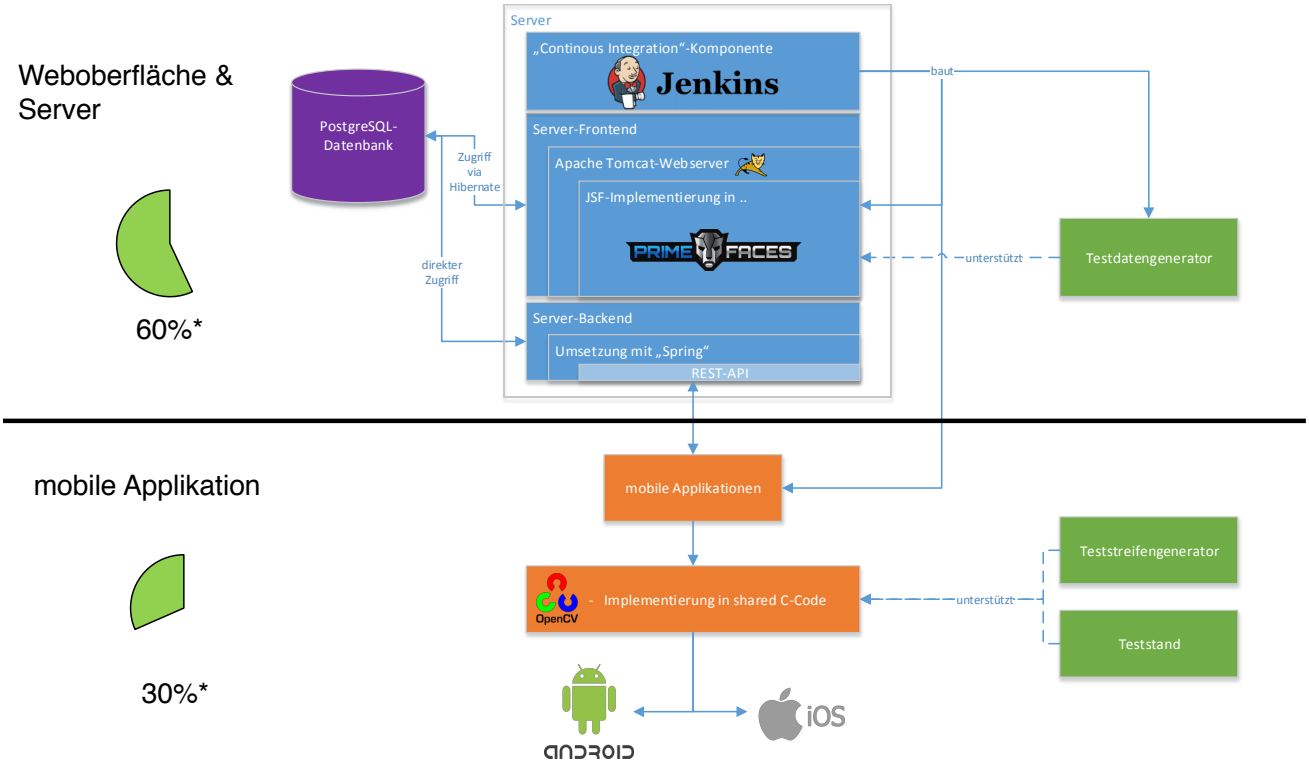



Abbildung 165: Fortschrittsanalyse nach dem Abschluss von Sprint 5

Die am Beginn von Sprint 5 gesetzten Sprintziele konnten erfolgreich umgesetzt werden. Somit kann im nachfolgendem Sprint eine erhöhte Konzentration auf die Bildverarbeitungsalgorithmen, unter Anwendung des Teststandes zur Validierung und Evaluation der Algorithmen, stattfinden.

## 4.5 Sprint 6

Als Basis von Sprint 6 dienen die Ergebnisse des vorherigen Sprints. Abbildung 166 visualisiert die Systemlandschaft zu Beginn des Sprints.

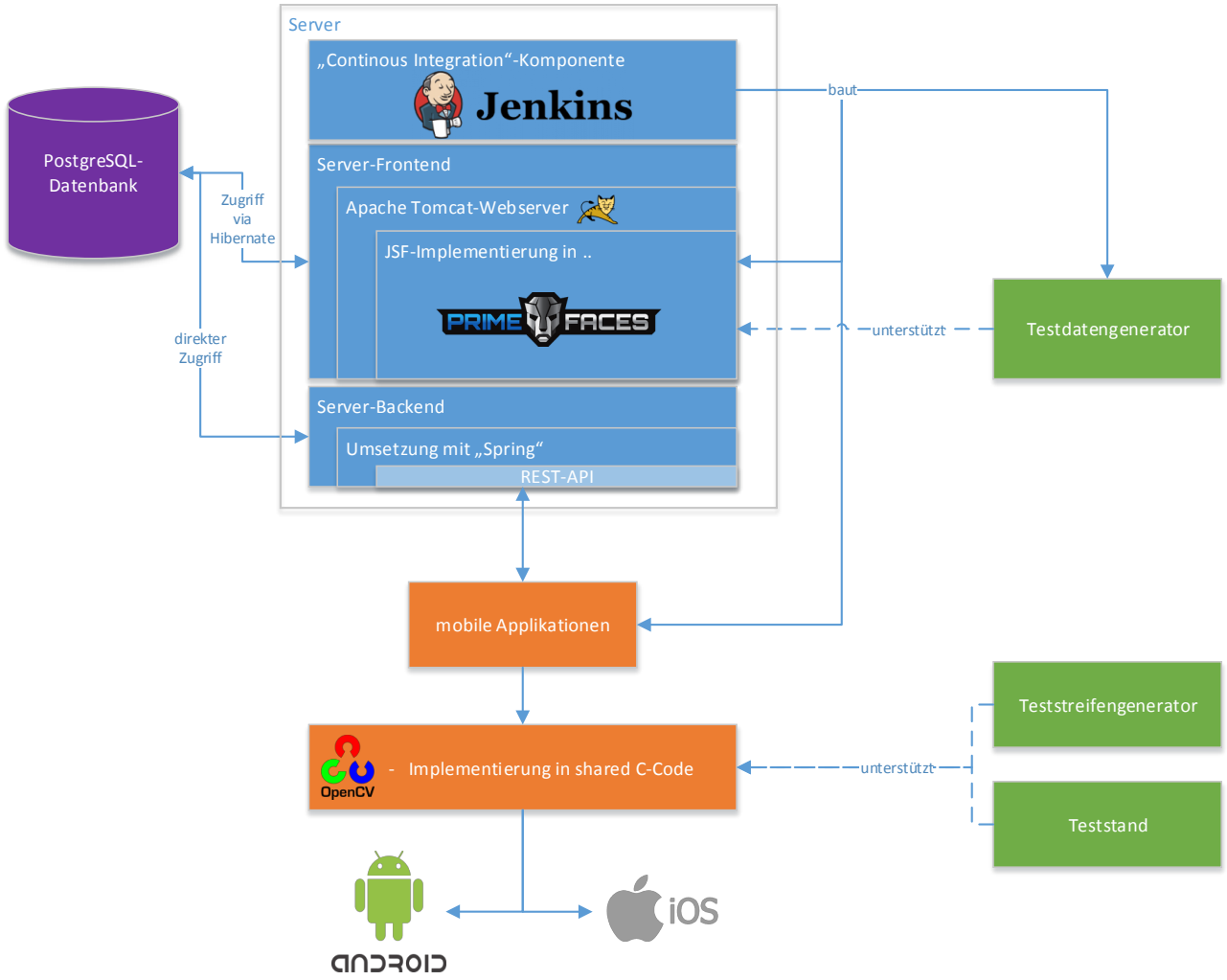



Abbildung 166: Systemarchitektur zu Beginn von Sprint 6

Der vorherige Sprint Nummer 5 legte seinen Schwerpunkt auf die Schaffung von Grundlagen zur späteren Optimierung der Bildverarbeitungsalgorithmen. Wie Abbildung 166 zeigt, stand neben einem Teststand zur Erstellung von Testbildern auch eine gemeinsame Quellcode-Basis für Implementierungen des Bildverarbeitungsalgorithmus zum Ende von Sprint 5 zur Verfügung.
Sprint 6 setzt an genau dieser Stelle an und setzt sich zum Ziel, die Bildverarbeitung weiter zu verbessern. Als erster Schritt hierfür wird die Anforderung des Lastenhefts bearbeitet, dass verschiedene dynamische Teststreifenvarianten verarbeitet werden können. Die Grundvoraussetzung dafür ist die Erstellung dieser Teststreifen. Das Arbeitspaket 4.5.3.1 auf Seite 224 „Anpassung des Teststreifengenerators für ein dynamisches Design und unterschiedliche Teststreifenvarianten“ beschäftigt sich mit genau dieser Umsetzung. Der nächste logische Schritt ist die Anpassung der Bildverarbeitungsalgorithmen an eben diese dynamischen Teststreifen. Hierfür ist das Arbeitspaket „Anpassung der Bildalgorithmen auf dynamische Teststreifen“ (Kapitel 4.5.4.1 auf Seite 241) formuliert worden. Neben der Verarbeitung von dynamischen Teststreifen ist weiterhin die Performanz des Bildverarbeitungsalgorithmus, insbesondere die der Marker-Detektierung, ein aktuelles Problemfeld. Zur Lösung dieser Problematik und zur Verbesserung der Geschwindigkeit des verwendeten Algorithmus,

wurde das Arbeitspaket „Weiterentwicklung der Bildverarbeitung“ (Kapitel 4.5.5.1 auf Seite 245) geplant. Das übergeordnete Ziel dieses Arbeitspakets ist die Ausführung des Algorithmus auch auf kostengünstigen Smartphones zu ermöglichen.
Um den Algorithmus hinsichtlich dieser Funktionserweiterungen, insbesondere der Verarbeitung dynamischer Teststreifen, objektiv und messbar zu evaluieren, wird ein Testframework entwickelt, das es ermöglicht, unterschiedliche Versionen des Bildverarbeitungsalgorithmus gegeneinander zu vergleichen. Das geplante Arbeitspaket „Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen“ (Kapitel 4.5.7.1 auf Seite 259) gibt somit nicht nur Aufschluss über die Qualität der aktuellen Implementierung, sondern bietet zudem auch einen effizienten und automatisierten Arbeitsablauf für zukünftige Evaluationen. Der im letzten Sprint entwickelte Teststand (siehe hierzu Kapitel 4.4.14.1 auf Seite 192) dient hierbei als reproduzier- und kontrollierbare räumliche Umgebung. Mit ihm können vergleichbare Videos generiert werden, die als Eingabeparameter für das Testframework genutzt werden.
Des Weiteren ist eine erste Implementierung des Go-Servers für diesen Sprint vorgesehen.

### 4.5.1 Sprintplanung

Im folgenden wird die Planung des sechsten Sprints (21.10.2015 bis 23.11.2015, vier Wochen) genauer aufgezeigt. Dabei wird ein besonderes Augenmerk auf die folgende Punkte gelegt:

- Fokus des Sprints
- Arbeitspakete, Vorgehen in der Planung und dem kritischer Pfad
- Planung der einzelnen Ressourcen
- Risikoeinschätzung für den weiteren Projektverlauf

#### 4.5.1.1 Fokus des Sprints

Bereits am Ende des fünften Sprints wurde damit begonnen, Ideen für neue Arbeitspakete für den sechsten Sprint zu schaffen und diese zu formulieren. Dabei sollte neben der Optimierung der Bildverarbeitungsalgorithmen auch die Erstellung des Go-Servers im Fokus liegen. Um den prozentualen Fortschritt der Apps weiter auszubauen, sollten zudem weitere funktionale Anforderungen laut Lastenheft in die iOS und Android Apps integriert werden.

Zunächst ergaben sich die folgenden Arbeitspakete:

- Erstellung des Go-Server-Backends (Priorität: normal - vier Wochen à zwei Personen)
- Optimierung der Bildverarbeitungsalgorithmen für die Apps (Priorität: normal - vier Wochen à zwei Personen)
- Entwicklung eines Algorithmus für eine Schattenrückrechnung auf dem Bild (Priorität: normal - zwei Wochen à zwei Personen)
- Integration weiterer funktionaler Anforderungen in die iOS App (Priorität: normal - zwei Wochen à zwei Personen)
- Integration weiterer funktionaler Anforderungen in die Anroid App (Priorität: normal - zwei Wochen à zwei Personen)
- Datenbank Refactoring (Priorität: normal - zwei Wochen à zwei Personen)
- Anpassung des Teststreifengenerators für dynamische Teststreifen (Priorität: kritisch - zwei Wochen à zwei Personen)

- Anpassung des Webfrontends an verschiedene Teststreifenvarianten (Priorität: kritisch - zwei Wochen à zwei Personen)
- Anpassung der Bildverarbeitungsalgorithmen an verschiedene Teststreifendesigns (Priorität: kritisch - zwei Wochen à zwei Personen)

Die Sprintplanung der Arbeitspakete und der kritische Pfad kann Abbildung 167 entnommen werden.

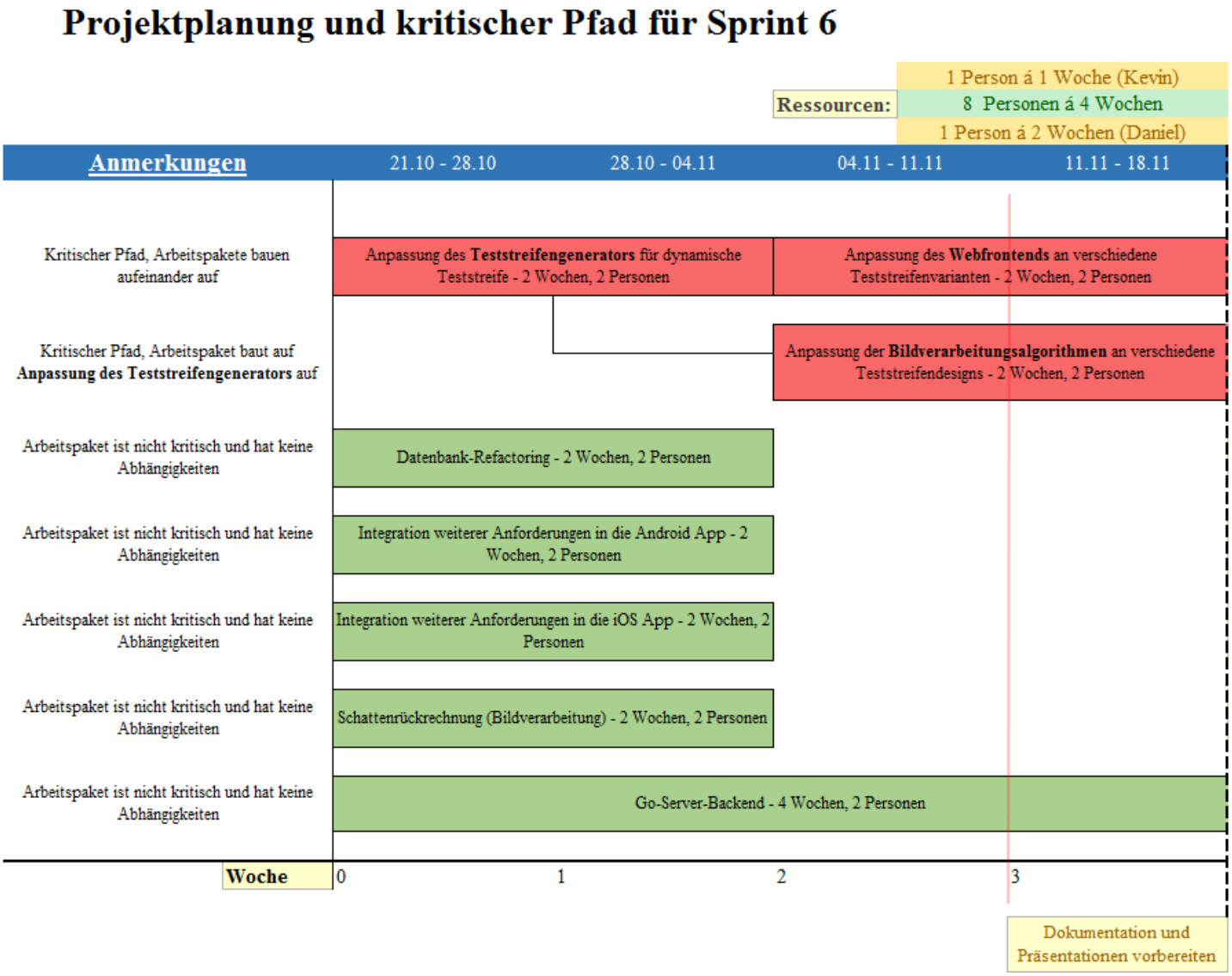



Abbildung 167: Kritischer Pfad und Planung der erste Iteration

Nachdem die Sprintplanung im Teammeeting vorgestellt wurde, kam von den Betreuern der Hinweis den Fokus noch intensiver auf die Optimierung der Bildverarbeitung zu legen, da sich viele Teammitglieder bereits tief in die Materie eingearbeitet haben und der Teststand nun für Testzwecke zur Verfügung steht. Dies wurde zunächst in großer Runde diskutiert, da der Einwand aufkam, andere Arbeitspakete könnten durch weiteres Herauszögern vernachlässigt werden. Die Entscheidung fiel auf anraten der Betreuer darauf, den Go-Server in einem kurzen zwei Wochen Arbeitspaket mit einem konkreten Anwendungsfall durchzuführen. Zudem wurde angemerkt, dass es ein weiteres Arbeitspaket bedarf, welches sich mit der Finalisierung des Teststandes beschäftigt. Daraufhin wurden die Arbeitspakete wie folgt angepasst.

- ~~Erstellung des Go-Server-Backends (Priorität: normal - vier Wochen à zwei Personen)~~
- ~~Integration weiterer funktionaler Anforderungen in die iOS App (Priorität: normal - zwei Wochen à zwei Personen)~~
- ~~Datenbank Refactoring (Priorität: normal - zwei Wochen à zwei Personen)~~
- ~~Integration weiterer funktionaler Anforderungen in die Anroid App (Priorität: normal - zwei Wochen à zwei Personen)~~
- ~~Anpassung des Webfrontends an verschiedene Teststreifenvarianten (Priorität: kritisch - zwei Wochen à zwei Personen)~~
- Optimierung der Bildverarbeitungsalgorithmen für die Apps (Priorität: normal - vier Wochen à zwei Personen)
- Entwicklung eines Algorithmus für eine Schattenrückrechnung auf dem Bild (Priorität: normal - zwei Wochen à zwei Personen)
- Anpassung des Teststreifengenerators für dynamische Teststreifen (Priorität: kritisch - zwei Wochen à zwei Personen)
- Fertigstellung des Konfigurationsprogrammes für den Teststand (Priorität: normal - max. zwei Wochen à eine Personen)
- Anpassung der Algorithmen für die Identifikation des Farbumschlages bei dynamische Teststreifen (Priorität: normal - vier Wochen à zwei Personen)
- Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen (Priorität: normal - zwei Wochen à zwei Personen)
- Auslieferung der Teststreifeninformationen über einen (Go)-Server (Priorität: normal - vier Wochen à zwei Personen)

Aus der Änderung der Arbeitspakete ergaben sich folgende Änderungen für die Sprintplanung und den kritischen Pfad (vgl. Abbildung 168 auf der nächsten Seite).

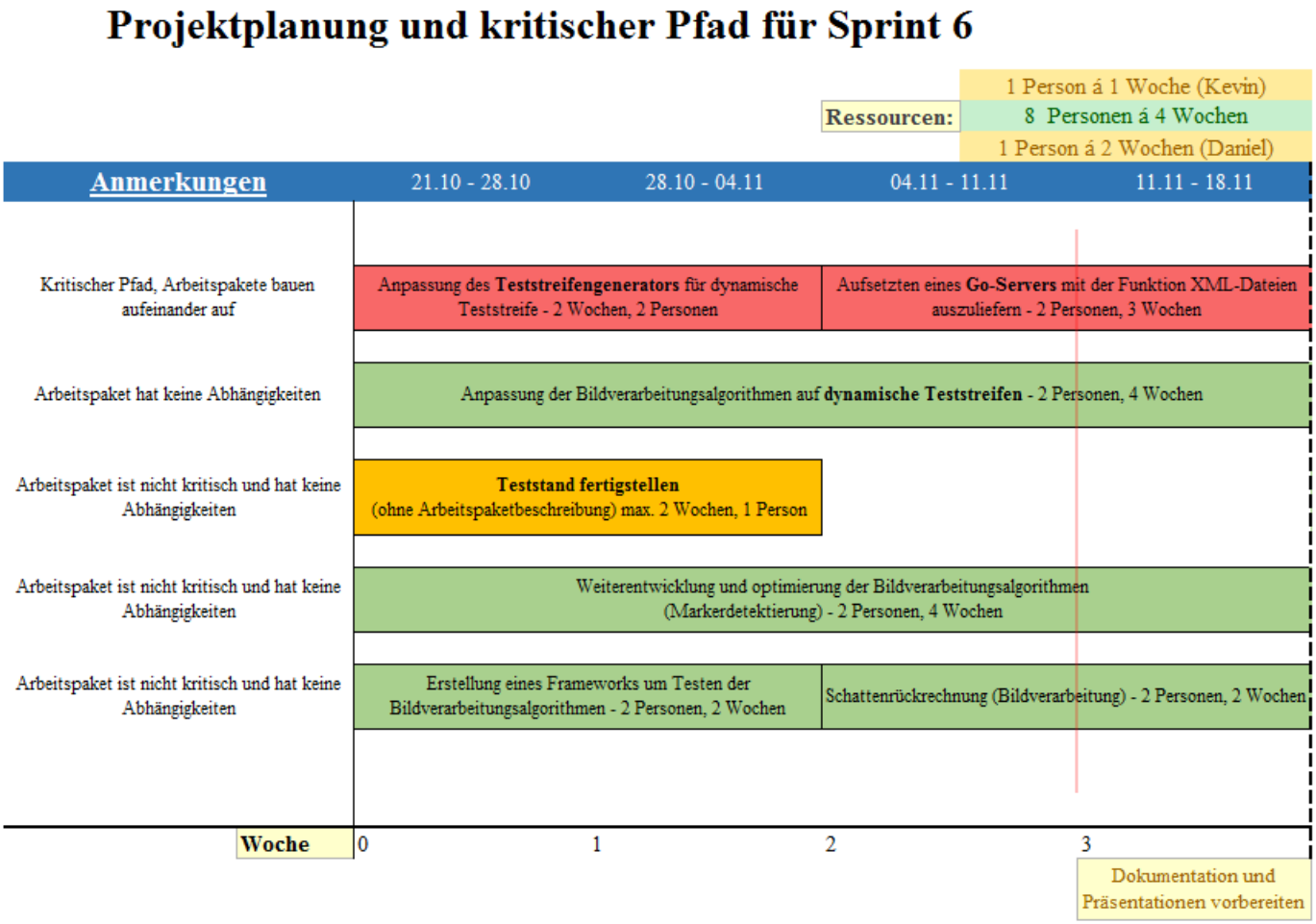



Abbildung 168: Kritischer Pfad und Planung nach der zweiten Iteration

Während des Sprintes wurde lediglich eine weitere Veränderung vorgenommen. Das Arbeitspaket „Aufsetzen des Go-Servers und die Auslieferung der XML-Dateien“ wurde von zwei auf drei Wochen verlängert, um eine ausreichende Einarbeitung in die Programmiersprache zu ermöglichen. Das Arbeitspaket kann in diesem Sprint daher nicht beendet werden und wird in den nächsten Sprint übernommen.

#### 4.5.1.2 Planung der Ressourcen

Die Personalplanung für den sechten Sprint stellt sich für die Arbeitspakete wie folgt dar.

- Raphael Kappes: vier Wochen Anpassung der Algorithmen für die Identifikation des Farbumschlages bei dynamische Teststreifen
- Christian Sandmann: vier Wochen Anpassung der Algorithmen für die Identifikation des Farbumschlages bei dynamische Teststreifen
- Timo Schlömer: vier Wochen Optimierung des Bilderkennungsalgorithmus für die Markerdetektierung
- Timo Raß: vier Wochen Optimierung des Bilderkennungsalgorithmus für die Markerdetektierung
- Nicolas Koch: zwei Wochen Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen; danach Schattenrückrechnung in der Bildverarbeitung

- Sebastian Horwege: zwei Wochen Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen; danach Schattenrückrechnung in der Bildverarbeitung
- Jan Philipp Stubbe: zwei Wochen Anpassung des Teststreifengenerators für dynamische Teststreifen; danach drei Wochen Go Server und Auslieferung der XML-Dateien
- Danny Fonk: zwei Wochen Anpassung des Teststreifengenerators für dynamische Teststreifen; danach drei Wochen Go Server und Auslieferung der XML-Dateien
- Kevin Sandermann: drei Wochen Seminararbeit, danach Einstieg in den Sprint für eine Woche, um die Dokumentation zu verfassen
- Daniel Wegmann: max. zwei Wochen Fertigstellung des Konfigurationsprogrammes für den Teststand, danach Einstieg in die Seminarphase
- Christoph Ressel: Einstieg in die Seminarphase

#### 4.5.1.3 Risikoeinschätzung für den weiteren Projektverlauf

An diesem fortgeschrittenen Zeitpunkt des Projektes ist es wichtig, eine Risikoeinschätzung für das MEDIC-Projekt durchzuführen, um zu verhindern, dass das Projekt scheitert und es im vorgegebenen Zeitrahmen erfolgreich zu Ende bringen zu können. Dazu wurde eine Risikoanalyse durchgeführt. Die einzelnen Schritte können den folgenden Punkten entnommen werden.

**1. Identifikation der Risiken**

Der Grund für eine Risikoeinschätzung ergibt sich aus den Umständen, dass viele Projektmitglieder noch Urlaub haben bzw. Seminararbeiten verfassen müssen und somit nicht für die operative Arbeit im Projekt bereit stehen. Zudem sind im neuen Jahr noch zwei Sprints mit jeweils sechs Wochen geplant. Die folgende Abbildung 169 zeigt die zu Verfügung stehenden Ressourcen für den achten Sprint (in blau). Dies ist jedoch zunächst nur unter Vorbehalt anzusehen, da noch weitere Teammitgleider Urlaub für die Klausurenphase einreichen können.

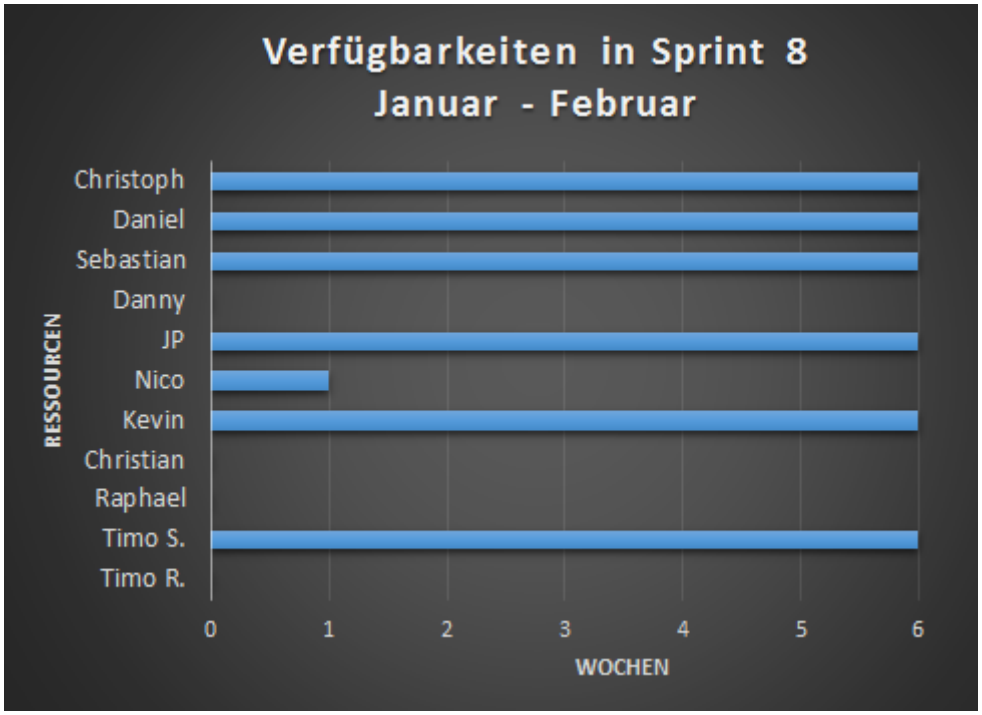



Abbildung 169: Ressourcenverfügbarkeit in Sprint acht

**2. Bewerten von Risiken**

Die zuvor aufgezeigte Grafik zeigt sehr deutlich, die Ressourcenknappheit für den achten Sprint. Da weder Krankheitsfälle noch weitere Urlaubsabwesenheiten in die Planung eingeflossen sind, ist die geringe Anzahl von Teammitgliedern als kritisch zu bewerten und somit der entschiedenste Risikoaktor für den Projekterfolg.

**3. Maßnahmen identifizieren und definieren**

Um die Gefahr des Scheiterns zu mindern, wurden vier sinnvolle Gegenmaßnahmen identifiziert.

**Arbeitspaket-orientierte Seminarthemen**

Zum achten Sprint werden drei Teammitglieder in ihre Seminarphase einsteigen, da der letzte Sprint komplett für operative Arbeit und die Fertigstellung des Projektes geblockt wird. Daher könnten diese Arbeiten einen starken Fokus auf die Umsetzung von funktionalen aber auch nicht funktionalen Anforderungen legen, z. B. in Form von Evaluationen.

**Zusätzliche Programmier-Wochenenden**

Um weitere Anforderungen umzusetzen und dies mit der gesamten Projektgruppe zu realisieren, können Programmier-Wochenenden eingeplant werden, welche zusätzlich zu den 16 Wochenstunden pro Person durchgeführt werden können. Diese haben den Vorteil, dass alle Mitglieder anwesend sind, die Kommunikationswege somit sehr kurz sind und der Fokus über mehrere Stunden und Tage auf ein konkretes Problem / eine konkrete Umsetzung einer Anforderung gelegt werden können. Die Termine hierfür werden im Projektgruppenmeeting am 18.11.2015 besprochen.

**Aufgaben-intensivere Arbeitspakete**

Neben den zuvor genannten Maßnahmen werden die Arbeitspakete im Arbeitsumfang etwas ansteigen. Viele Arbeitspakete bezogen sich zumeist oft auf nur eine Aufgabe mit einem konkreten Anwendungsfall. Dies wird in Zukunft so angepasst, dass ähnliche Aufgaben gebündelt werden und in einem statt zwei oder drei Arbeitspaketen abgearbeitet werden. Komplexe Aufgaben oder Machbarkeitsstudien aus neuen Bereichen werden davon nicht betroffen sein.

**4. Maßnahmen kommunizieren**

Die definierten Maßnahmen werden mit dem Team am 18.11.2015 besprochen und weiter diskutiert bzw. präzisiert.

**5. Risiken überwachen und Auswirkungen prüfen**

Das Projektmanagement für das MEDIC-Projekt wird die Umsetzung der Maßnahmen kontinuierlich und iterativ prüfen und dazu in einem regen Austausch mit der gesamten Projektgruppe stehen. Etwaige Änderungen und Ereignisse werden im nächsten Dokument des siebten Sprints verschriftlicht werden.

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und dessen Dokumentation skizziert.

### 4.5.2 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete wie gewohnt detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.5.3 Anpassung des Teststreifengenerators für ein dynamisches Design und unterschiedliche Teststreifvarianten

- Priorität: Kritisch; Verzögerungen an diesem Arbeitspaket beeinflussen nachfolgende Arbeitspakete.
- Motivation und Nutzen des Arbeitspaketes: Das Ergebnis dieses Arbeitspaketes dient als notwendige Hilfestellung für die Erstellung dynamischer und verschiedener Teststreifenvarianten und die Anpassung der Bildalgorithmen auf die verschiedenen Varianten.
- Arbeitspaketbeschreibung:
  Am Ende dieses Arbeitspaketes ist der Teststreifengenerator so angepasst, dass das Design der Teststreifen variabel gestaltet werden kann. So können z. B. mehrere Farbkanäle auf einem Teststreifen dargestellt werden, siehe Abbildung 170. Die Konfiguration erfolgt dabei über eine Konfigurationsdatei (z. B. XML), welche alle zu verändernden Parameter enthält. Diese kann der Teststreifengenerator einlesen und den entsprechenden Teststreifen generieren.

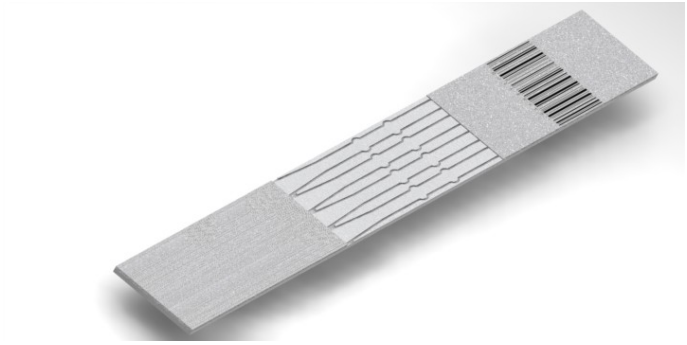


Abbildung 170: Möglicher Aufbau einer anderen Teststreifenvariante

  - Größe des Teststreifens (Länge und Breite)
  - Position des Farbumschlagfeldes
  - Position der Marker
  - Anzahl der Kanäle

  Weitere sinnvolle Parameter für die Konfiguration dürfen hinzugefügt werden.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der angepasste Teststreifengenerator steht für folgende Arbeitspakete zur Verfügung. Im Wiki wurde eine kurze Beschreibung eingefügt, wie der neue Teststreifengenerator zu nutzen ist.
- Aufwand:
  Zwei Wochen
- Personen:
  - Danny Fonk
  - Jan Philipp Stubbe

**4.5.3.1 Dokumentation** Der bisherige Teststreifengenerator erwies sich bei der Erstellung verschiedener Teststreifenvarianten als nicht flexibel genug. Beispielsweise sind Farbkanäle auf den bisher generierten Teststreifen nicht berücksichtigt worden. Deshalb ist innerhalb dieses Arbeitspaketes eine Anpassung des bisherigen Generators auf Basis neuer interner Anforderungen vorgesehen. Des Weiteren ist es nun möglich das Aussehen eines Teststreifens über ein Extensible Markup Language (XML)-Schema zu definieren, welche bei der Bildverarbeitung genutzt werden um die entsprechenden Farbumschläge zu detektieren, respektive generell Informationen über das Aussehen eines Teststreifens zu gewinnen. Der Generator kann diese XML-Dateien einlesen und auch ausgeben.

**Ablauf** Der bisherige Quellcode für den Teststreifengenerator wurde als Basis für die Erweiterungen genutzt. Der neue Funktionsumfang gestaltet sich wie folgt:

1. Marker
   - Variable Anzahl, Position, Größe und enthaltener Wert.
2. Farbumschläge
   - Variable Anzahl, Position, Intensität und verwendeten Wert.
   - Zusätzlich ist die Farbe konfigurierbar, jedoch nur global für alle Farbumschläge.
3. Informationen (Barcode)
   - Variable Position, Größe und enthaltener Wert.
   - Zusätzlich ist derzeit eine Reihe von verschiedenen Barcode-Formaten auswählbar. Zur Verfügung stehen aus Gründen der Vollständigkeit weitaus mehr Formate, als womöglich später genutzt werden. Nachfolgend eine Auflistung der zur Verfügung stehenden Formate:

     – Codabar
     „*Die Kodierung Codabar stellt einen beschränkten alphanumerischen Zeichensatz zur Verfügung. Die Buchstaben a bis d können nur als Start und Stopzeichen verwendet werden. Neben diesen Buchstaben sind noch die Sonderzeichen $ + - : . / möglich.*“
     Quelle: https://de.wikipedia.org/wiki/Codabar

     – Code39
     „*Die Kodierung Code39, auch kurz 3of9 genannt, stellt einen alphanumerischen Zeichensatz zur Verfügung. Außer Ziffern und Großbuchstaben sind sieben Sonderzeichen definiert. Start- und Stoppzeichen sind identisch und werden durch das Sonderzeichen ”*”dargestellt. Der Code39 ist in der ISO/IEC 16388[1] spezifiziert.*“
     Quelle: https://de.wikipedia.org/wiki/Code39

     – Code128
     „*Der Code128 (1981) ist ein alphanumerischer Strichcode (Barcode) mit hoher Informationsdichte. Der Prinzipaufbau eines Code128-Symbols besteht aus einem Startzeichen, der Nutzinformation, der Prüfziffer und dem Stoppzeichen. [...] Der Code 128 ist in der internationalen Norm ISO/IEC 15417 vollständig beschrieben.*“Quelle: https://de.wikipedia.org/wiki/Code128

– EAN-13
„*Die European Article Number (EAN) ist die frühere [...] Bezeichnung für die Globale Artikelidentnummer (Global Trade Item Number, abgekürzt GTIN). Sie stellt eine international unverwechselbare Produktkennzeichnung für Handelsartikel dar. Die Nummer besteht aus 8 bzw. 13 Ziffern, von denen die ersten 2 oder 3 bzw. 7, 8 oder 9 Ziffern zentral durch die GS1-Gruppe verwaltet und an Hersteller auf Antrag als Global Location Number vergeben werden. In Deutschland fallen für die Vergabe Lizenzgebühren an die GS1 Germany an.*“
Quelle: https://de.wikipedia.org/wiki/European_Article_Number.

– EAN-8
„*Die verkürzte Version EAN-8 ist speziell für kleine Artikel gedacht, auf denen eine EAN-13 mehr als 25% des Platzes auf der Vorderseite benötigen würde. Sie hat folgenden Aufbau: GS1-Länderpräfix (2–3 Stellen), Artikelnummer (Reststellen, also 4–5), Prüfziffer (1 Stelle).*“
Quelle: https://de.wikipedia.org/wiki/European_Article_Number.

– ITF
„*Der ITF-Code (Interleaved Two of Five) ist wie die Europäische Artikelnummer aufgebaut, jedoch ist zusätzlich ein einstelliges ITF-Präfix vorangestellt .... [...] In dem Kontext EAN bzw. GS1 wird dieser Code immer als ITF-14 bezeichnet. Der ITF-14 ist ein normaler 2/5 interleaved Code gemäß ISO/IEC 16390, hat aber Einschränkungen: Modulbreite nur von 0,495 mm bis 1,016 mm, immer 14-stellig, immer mit GTIN-Prüfziffer.*“
Quelle: https://de.wikipedia.org/wiki/Strichcode#2.2F5-Familie

– PDF417
„*DF417 (PDF steht für „Portable Data File“) ist ein 2D-Barcode[...]. Die gestapelten Barcodetypen erfordern beim Lesen die vollständige Kongruenz von Abtastlinie und Codesymbol bzw. Codezeile. Diese Einschränkung verringert die erzielbare Informationsdichte, da größere Codehöhen gedruckt werden müssen, als aus theoretischer Sicht notwendig erscheint.*“
Quelle: https://de.wikipedia.org/wiki/PDF417

– QR-Code

Abbildung 171:
Codebeispiel: QR-Code
Quelle: https://de.wikipedia.org

*„Der QR-Code (englisch Quick Response, „schnelle Antwort“, als Markenbegriff „QR-Code“) ist eine Methode, Informationen so aufzuschreiben, dass diese besonders schnell maschinell gefunden und eingelesen werden können. Aufgrund einer automatischen Fehlerkorrektur ist dieses Verfahren sehr robust und daher weit verbreitet. [...] Im Code enthalten sind die Versionsinformation (1) und das benutzte Datenformat (2). Der Datenteil (3) enthält die kodierten Daten in redundanter Form. Zur Feldbegrenzung enthält der QR-Code in nur drei seiner Ecken ein bestimmtes Muster (4.1). Über das fehlende Muster in der vierten Ecke erkennt das Lesegerät die Orientierung. Mit zunehmender Größe des Codes werden weitere Muster (4.2) hinzugefügt, um die Ausrichtung des Codes besser erkennbar zu machen. Zwischen den drei Hauptpositionsmarkierungen befindet sich eine Linie (4.3) aus einer Folge streng abwechselnder Bits, worüber sich die Matrix definiert.“*
Quelle: https://de.wikipedia.org/wiki/QR-Code

– UPC-A


Abbildung 172:
Codebeispiel: UPC-A
Quelle: https://de.wikipedia.org

*„Der Universal Product Code (UPC) ist ein Strichcode, durch den Produkte im Einzelhandel gekennzeichnet werden. Kodiert wird ein Nummerncode, den Barcodescanner berührungslos optisch auslesen. [...] Über die ersten sechs Stellen des UPC A Code [...] kann man ein Produkt eindeutig einem Hersteller oder Händler zuordnen. Die Stellen 7 bis 11 kann der Besitzer des Nummernkreises auf seine Produkte verteilen. Die 12. Stelle ist eine Prüfziffer, die nach Modulo 10 mit der Gewichtung 3 berechnet wird. Unternehmen können UPC-Nummernkreise beim Uniform Code Council gegen Gebühr erwerben. Die Kodierung eines UPC ist identisch mit der Kodierung einer EAN mit führender Null.“*
Quelle: https://de.wikipedia.org/wiki/Universal_Product_Code

4. Sonstiges

   - Als weitere Parameter lassen sich der DPI-Wert („dots per inch“), der Hintergrund, zusätzliche Farbinformationen und der Farbraum der Farbinformationen (RGB oder HSV) einstellen.

**GUI-Beschreibung** Nachfolgend eine Beschreibung der Bedienelemente und ihre Bedeutung. Dabei sind die verschiedenen Bedienelemente zur besseren Verdeutlichung jeweils abgebildet, nummeriert und entsprechend ihrer Nummerierung beschrieben.

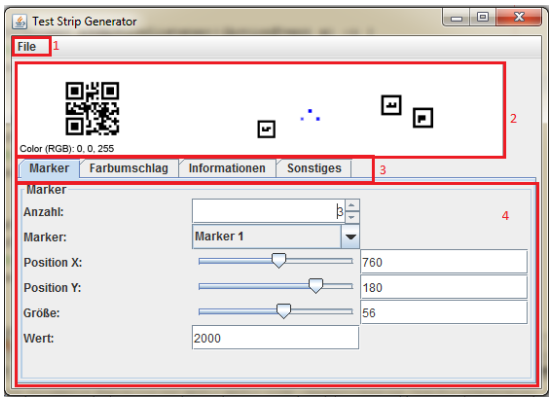



Abbildung 173: Teststreifengenerator - Gesamtübersicht

**Bedienelemente in der Gesamtübersicht**

1. Menü-Bar
   Über die Menübar lassen sich die bereits bekannten Menüsteuerungen erreichen. Zudem ist über das Menü nun der XML Import und auch Export zu erreichen.

2. Teststreifen-Vorschau
   In der Teststreifen-Vorschau wird stets eine aktuelle Darstellung des Teststreifens nach einer erfolgreichen Änderung einer Einstellungsmöglichkeit angezeigt. Eine Bedienung über die Vorschau (durch beispielsweise „Drag and Drop“ ist derzeit nicht möglich und ist innerhalb des Teststreifengenerators nicht geplant (siehe dazu 4.5.3.1 auf Seite 240).

3. Karteireiter zum Wechseln der Einstellungsebenen
   Über diese Karteireiter ist es möglich die Einstellungsebene zu wechseln. Zur Verfügung stehen dabei Marker, Farbumschlag, Informationen und Sonstiges. Die einzelnen Einstellungsmöglichkeiten der einzelnen Karteireiter ist in den nachfolgenden Abschnitten erklärt.

4. Einstellungsebene
   In der Einstellungsebene können spezifische Einstellungen zu den einzelnen Elementen (z. B. Marker oder Farbumschläge) vorgenommen werden.

**Einstellungsebene „Marker“**

1. Anzahl der Marker verändern
   Die Anzahl der Marker ist frei einstellbar, jedoch werden mindestens drei Marker zur Detektion benötigt, weshalb eine Anzahl größer gleich 3 gewählt werden muss.

2. Marker-Auswahl
   Mit diesem Kontrollelement lässt sich ein Marker auswählen für den Einstellungen bzgl. der Position, Größe und seines Wertes vorgenommen werden sollen.

3. Ändern der Position
   Die Position auf der X- sowie auf der Y-Achse lässt sich mittels Schieber oder Freitext einstellen. Dabei ist die maximale Position abhängig von der Größe des Markers, sodass dieser lediglich bis zum äußerem Rand des Teststreifens verschoben werden kann.

4. Verändern der Größe
   Die Größe lässt sich ebenfalls mittels Schieber oder Freitext einstellen. Die eingestellte Größe muss dabei jedoch ein vielfaches von acht sein.

5. Anpassen des dargestellten Wertes
   Der Wert lässt sich über ein Freitext-Feld eintragen. Der dabei gewählte Wert muss zwischen 0 und $2^{15}-1$ liegen.

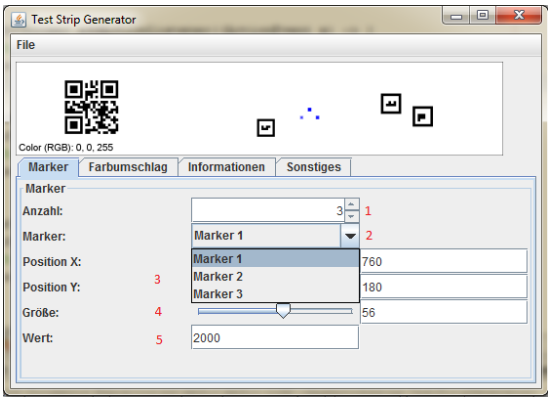



Abbildung 174: Teststreifengenerator - Panel: Marker

**Einstellungsebene „Farbumschläge“**

1. Die Anpassung der Anzahl, die Auswahl des Farbumschlags sowie die Veränderung der Position ist analog zu den Kontrollelementen des Marker-Karteireiters.

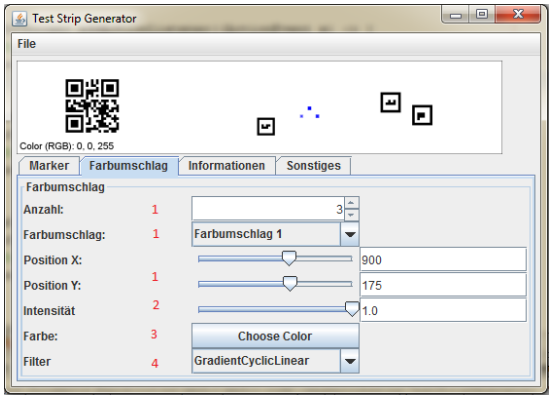



Abbildung 175: Teststreifengenerator - Panel: Farbumschlag

2. Intensität
   Die Farbintensität für einen Farbumschlag lässt sich seperat einstellen. Dabei können Werte zwischen 0.0 und 1.0 gewählt werden. Diese Werte entsprechen einer Prozentangabe (0% - 100%).

3. Farbe
   Die Farbe der Farbumschläge lässt sich global für alle Farbumschläge einstellen. Durch das Betätigen der Schaltfläche, öffnet sich ein Dialog, welcher die Einstellung der Farbe über verschiedene Farbräume zulässt. Siehe dazu Abbildung 176.

4. Filter
   Derzeit existieren zwei Filtermöglichkeiten: Ein linearer, nicht-zyklischer Filter, welcher von links (maximale Intensität) nach rechts (minimale Intensität) verläuft und ein linearer, zyklischer Filter, welcher von innen (maximale Intensität) nach außen (minimale Intensität) verläuft. Die Auswahl des Filters ist dabei für jeden Farbumschlag separat einstellbar.

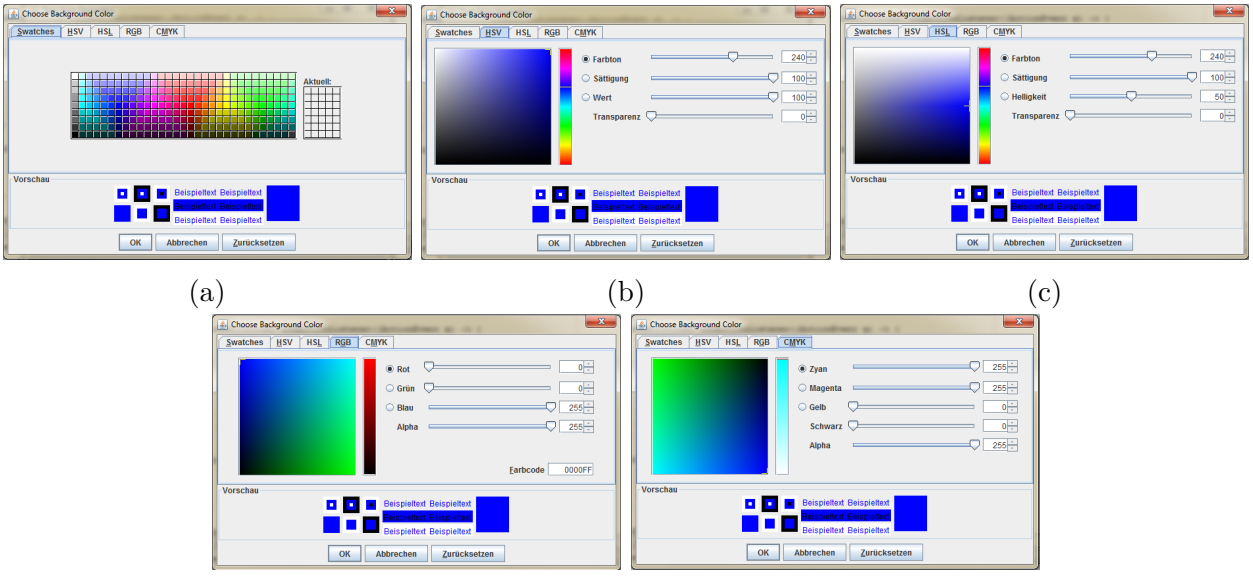



Abbildung 176: Teststreifengenerator - Farbeinstellungen

Die Einstellung der Farbe der Farbumschläge kann unter anderem über mehrere Farbräume vorgenommen werden. Die Einstellungsauswahl kann über Muster (Abbildung 176a), über den HSV-Farbraum ( 176b), HSL-Farbraum ( 176c), den RGB-Farbraum ( 176d) oder über den CMYK-Farbraum ( 176e) vorgenommen werden.

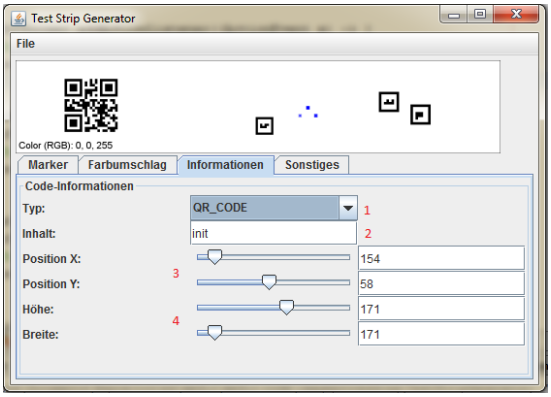
**Einstellungsebene „Informationen“**

1. Typ
   Über eine Schaltfläche kann eine Auswahl eines Informations-Codes erfolgen. Beispielsweise ist die Auswahl eines QR-Codes (vgl. Abbildung 177a) oder des Typs „Codabar“ möglich. Eine Auflistung aller möglichen Informationscodes sowie eine Kurzbeschreibung derer ist in Kapitel 4.5.3.1 auf Seite 224 zu finden.

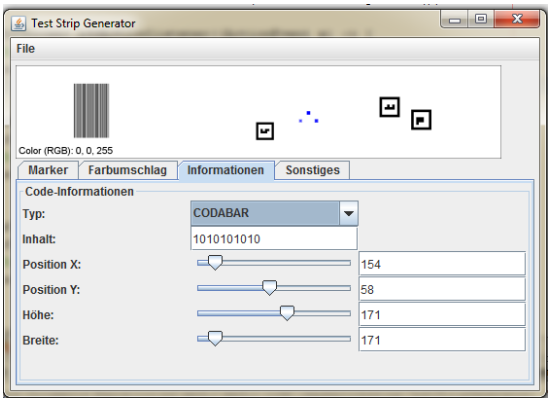
2. Inhalt
   Der zu kodierende Inhalt kann über ein Freitext-Feld angegeben werden. Die angegebenen Informationen werden nach Eingabe kodiert und in Form des ausgewählten Code-Typen dargestellt. Die Länge der Eingabe und auch die Größe des Alphabets kann je nach Typ variieren (vgl. 4.5.3.1 auf Seite 224).

3. Position
   Die Einstellung der Position ist analog zu den Bedienelementen der Positionsregulierungen der Farbumschläge und der der Marker. Die maximale X- und Y-Position ist abhängig von der angegebenen Größe des Informations-Codes.

4. Größe
   Die Größe lässt sich über die Höhe und Breite definieren. Einige Typen haben eine quadratische Form, sodass bei Änderung eines Parameters ohne den anderen keine sichtbare Veränderung der Größe vorgenommen wird. Die freie Fläche wird dann jedoch mit transparenten Pixeln aufgefüllt, was dazu führen kann, dass sich die maximalen Werte für die X- und /oder Y-Position verändern.

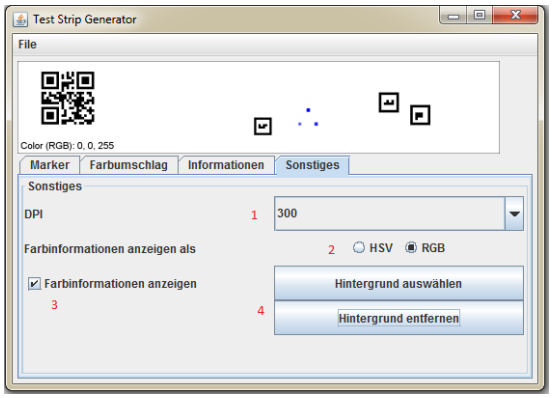



(a) QR-Code

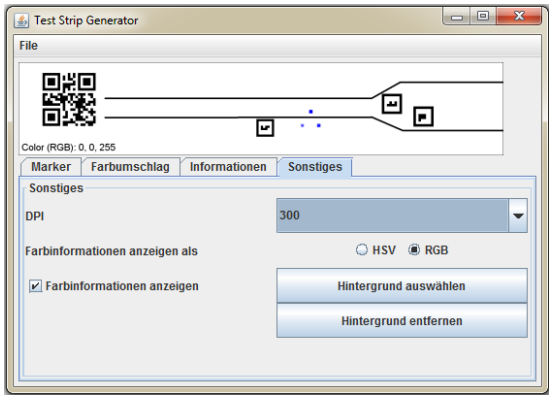



(b) Codebar

Abbildung 177: Teststreifengenerator - Panel: Informationen




(a) ohne Hintergrund




(b) mit Hintergrund

Abbildung 178: Teststreifengenerator - Panel: Sonstiges

**Einstellungsebene „Sonstiges“**

1. DPI-Wert
   Wie bereits im ursprünglichem Teststreifengenerator vorhanden, bleibt die Möglichkeit des Einstellens des DPI-Wertes erhalten. Die Auswahl beschränkt sich dabei jedoch auf 72 oder 300 DPI. Bei einer Auswahl von 72 DPI, wird das ursprüngliche Bild in 300 DPI erstellt und anschließend auf 72 DPI skaliert.

2. Farbinformationen-Typ
   Das Bedienelement erlaubt dem Nutzer einzustellen, ob die Informationen im RGB- oder HSV-Farbraum dargestellt werden sollen.

3. Farbinformationen anzeigen
   Über die Checkbox „Farbinformationen anzeigen“ lässt sich einstellen, ob auf dem Teststreifen die Farbwerte in RGB oder HSV angezeigt werden.

4. Auswahl des Hintergrunds
   Beim Betätigen der Schaltfläche „Hintergrund auswählen“ öffnet sich ein Dialog, welcher es erlaubt eine Bilddatei in das Programm zu importieren. Um den gesamten vorhandenen Platz auf dem Teststreifen auszufüllen, ist es notwendig, dass die Bilddatei eine Abmessung von $1535 \times 295$ vorweist. In Abbildung 178a ist der Teststreifengenerator ohne die Auswahl eines Hintergrundes, während in Abbildung 178b ein eigens erstellter Hintergrund gewählt wurde, auf dem Farbkanäle simuliert werden sollen. Denkbar ist nicht nur die Darstellung von Farbkanälen, sondern auch die Simulation von Verfälschungen wie beispielsweise Verschmutzungen wie Kaffeeflecken oder Nässe.

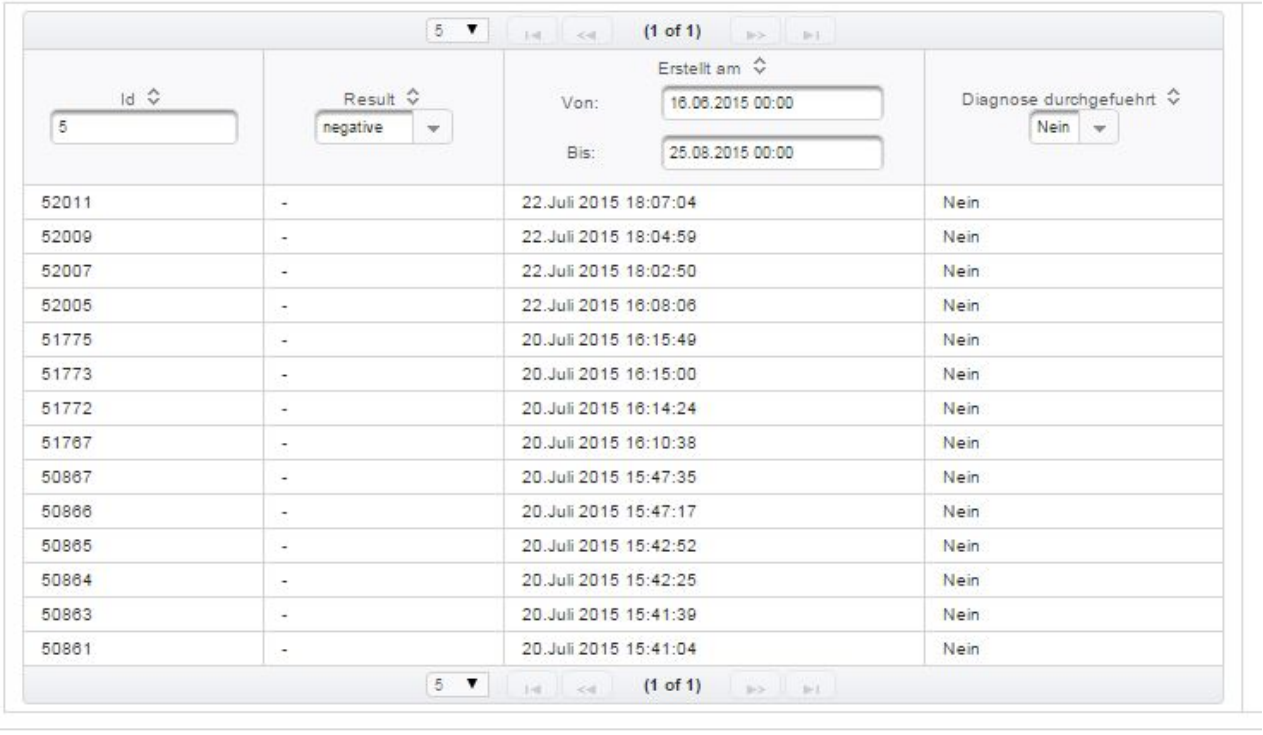



Abbildung 67: Screenshot der Liste mit den Tests und den Möglichkeiten zum Filtern und Sortieren

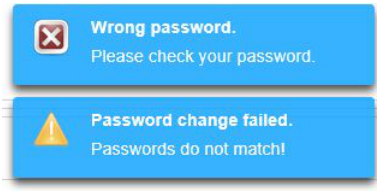



Abbildung 71: Fehlermeldungen beim Ändern des Passworts

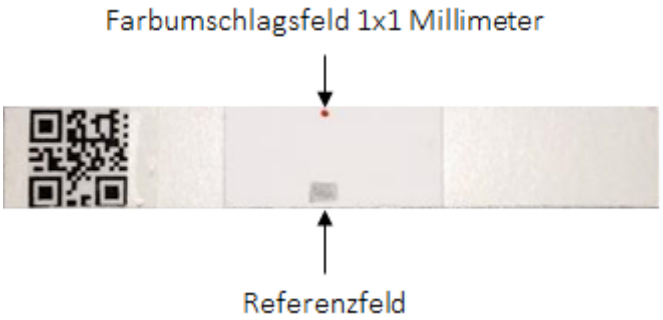



Abbildung 76: Möglicher Aufbau eines Teststreifens

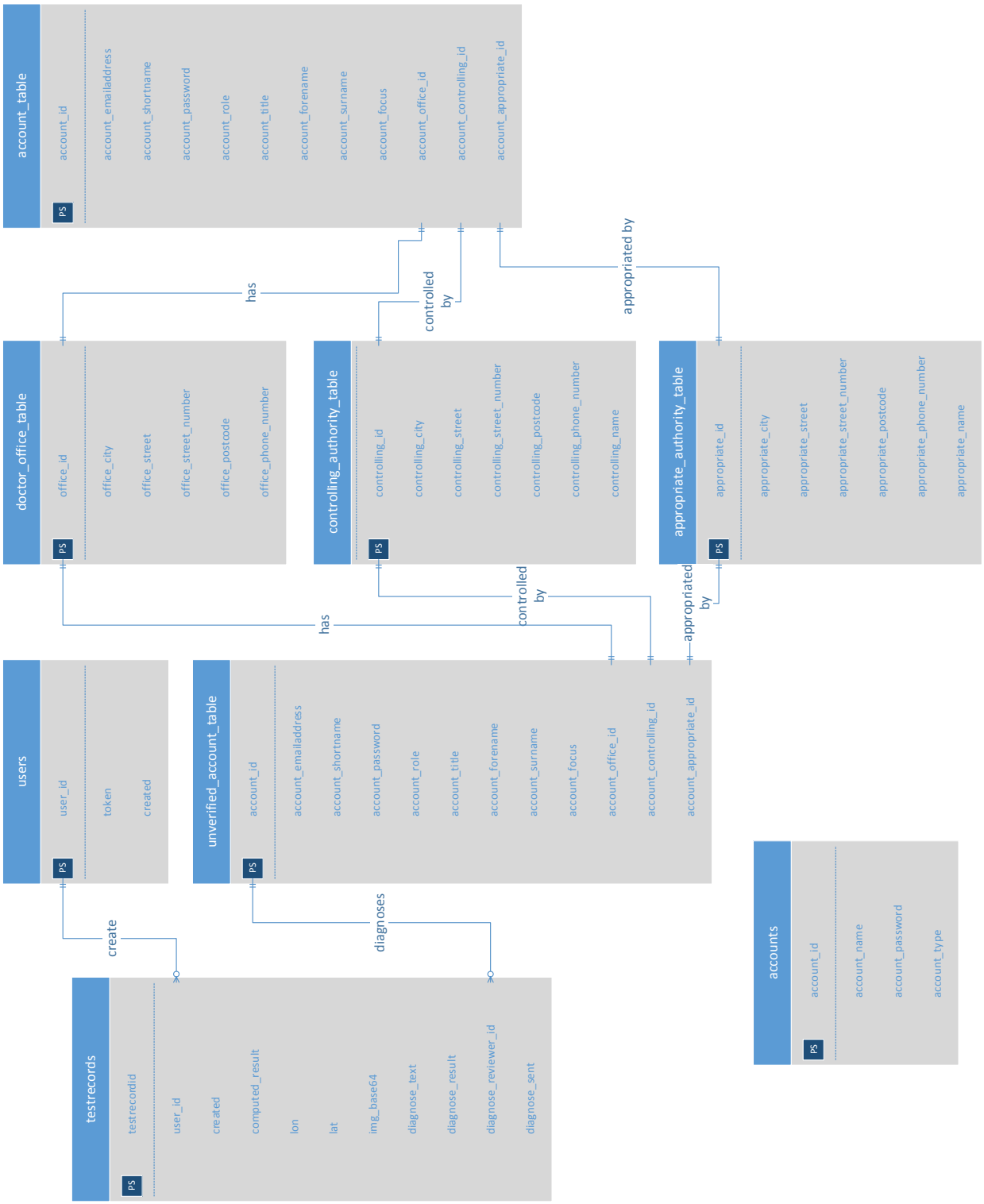



Abbildung 98: Entity-Relationship-Diagramm

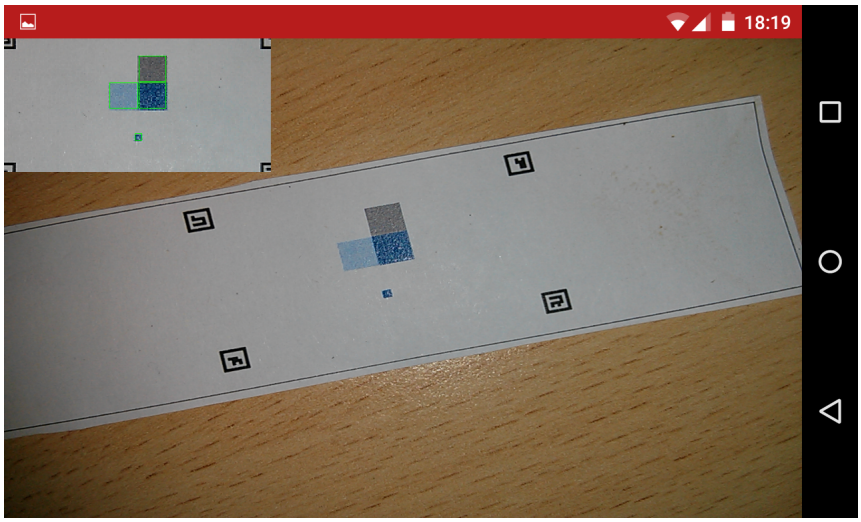



Abbildung 108: Die App erkennt einen Teststreifen

Abbildung 109: Ein Smartphone auf einer Mikroskophalterung

Abbildung 110: Erfolgreiche Detektierung eines Teststreifens

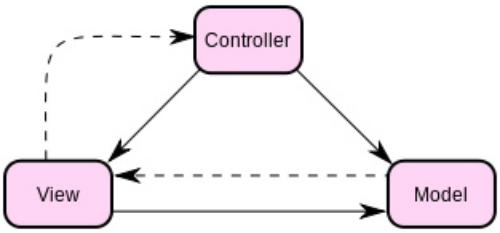



Abbildung 123: Model-View-Controller-Konzept

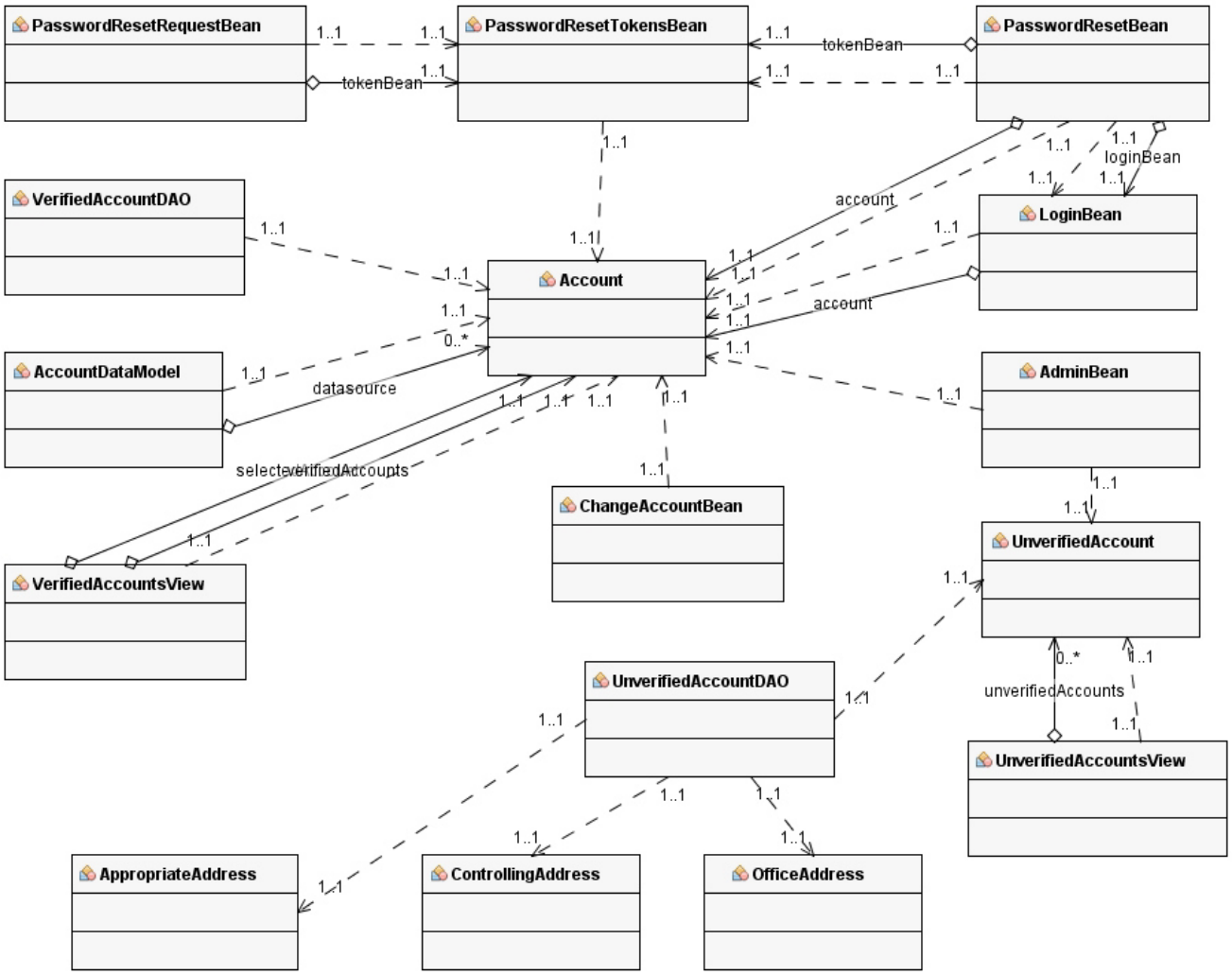



Abbildung 124: Klassendiagramm: Nutzerverwaltung

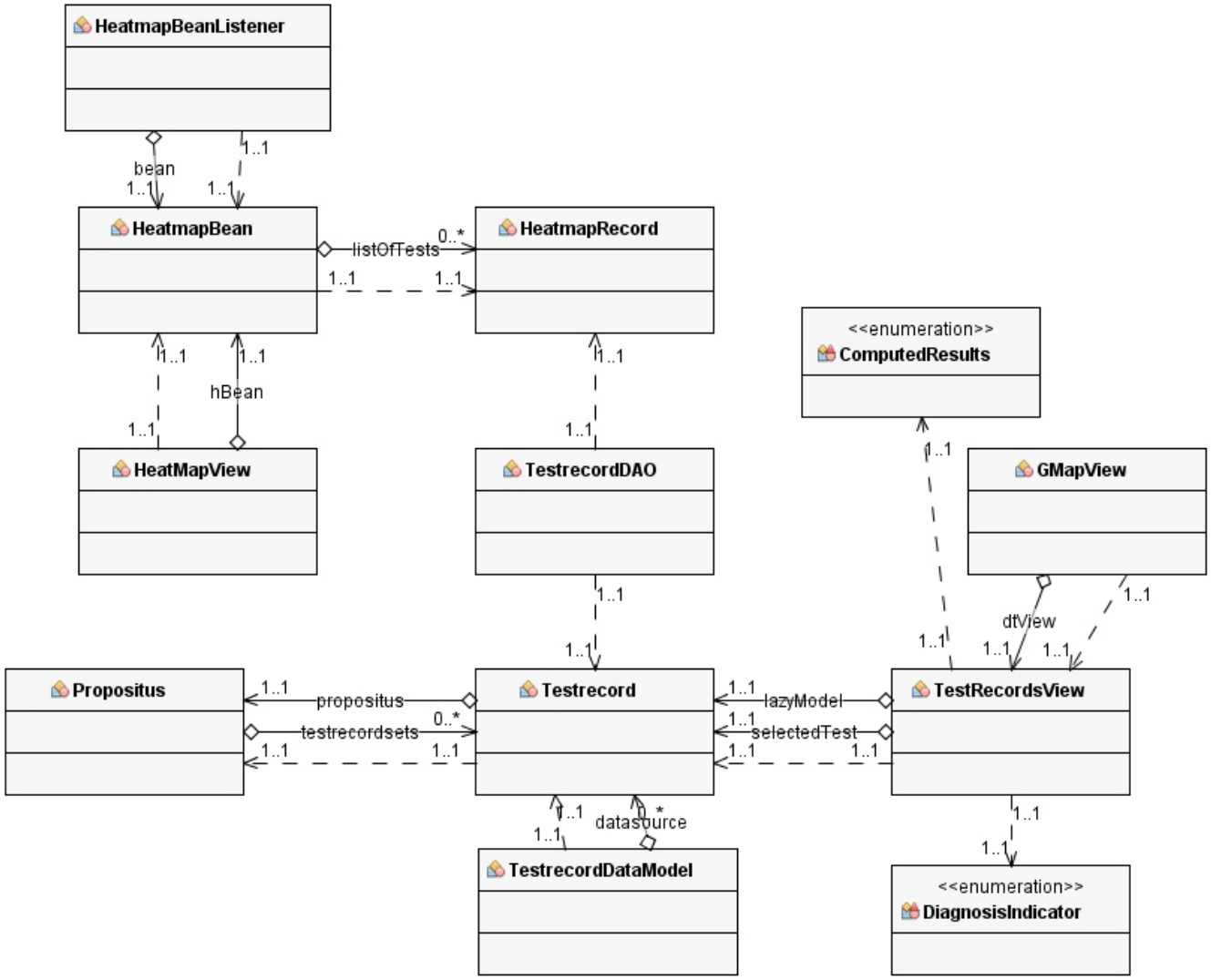



Abbildung 125: Klassendiagramm: Testrecords

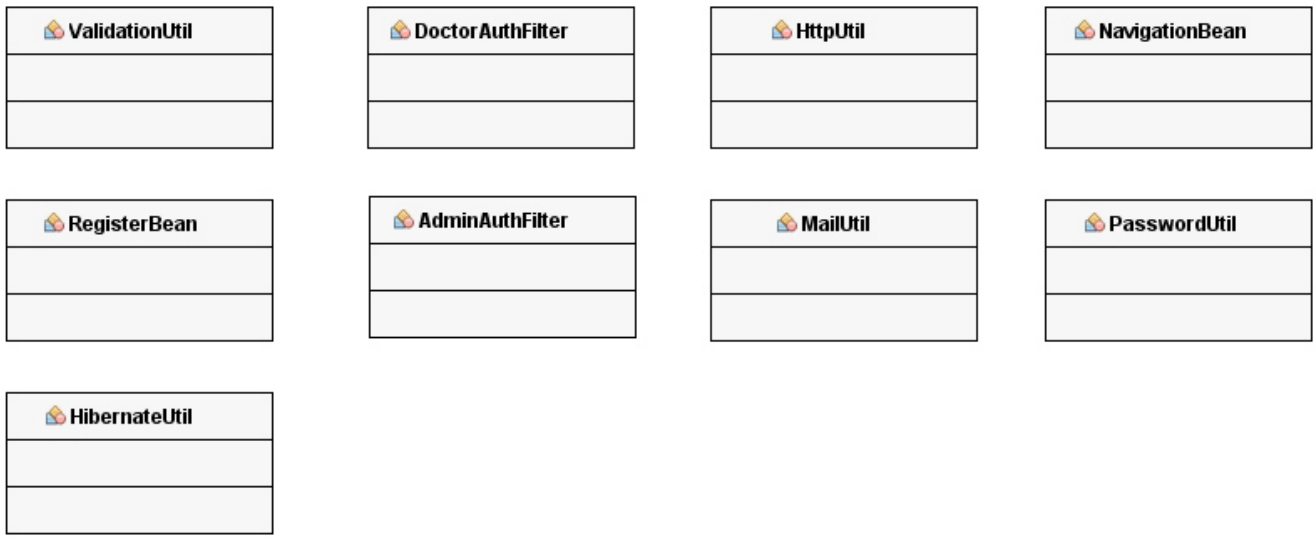



Abbildung 126: Klassendiagramm: Einzelne (Hilfs-)Klassen

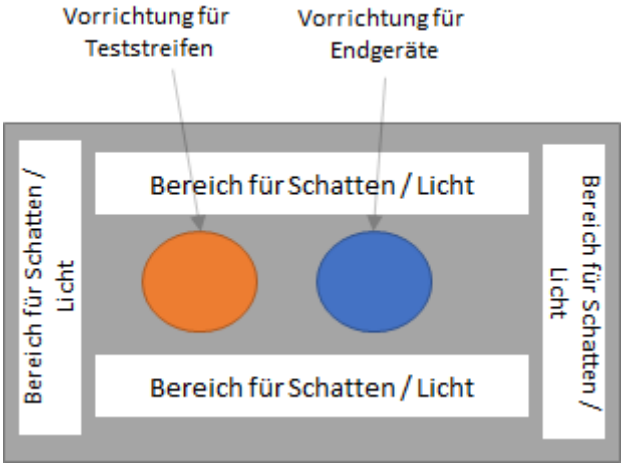



Abbildung 127: Möglicher Aufbau eines Teststandes aus der Vogelperspektive

**CLI-Beschreibung** Die Parameter des Command-Line-Interfaces haben sich stark reduziert. Es ist nicht mehr notwendig sämtliche Parameter des Teststreifens einzeln anzugeben, da das Aussehen eines Teststreifens nun mittels einer XML-Datei beschrieben wird. Die Mehrfacherzeugung von Teststreifen muss nun anders gesteuert werden: Ein Ordner-Pfad kann als Eingabeparameter anstelle einer XML angegeben werden, wodurch alle XML-Dateien in diesem Verzeichnis als Eingabeparameter genutzt werden. Nachfolgend die Parameter und ihre Bedeutung, welche im Programm selbst in Englisch angegeben sind, in deutscher Sprache.

```
Nutzung: tsg.jar [weitere Argumente]

-f,--file <arg>      Die zu importierende XML-Datei, welche das
                     aussehen des Teststreifens beschreibt.
                     Alternativ ist es moeglich einen Ordner-Pfad
                     anzugeben. In diesem Ordner-Pfad werden alle
                     XML-Dateien (nicht-rekursiv) ausgelesen und
                     in die dazugehoerigen Bilder generiert.
-gui                 Das Programm mit grafischer Oberflaeche starten.
-h,--help            (Diese) Hilfenachricht ausgeben.
-o,--output <arg>    Ordner-Pfad unter dem das(/die) Bild(er) ausgegeben
                     werden sollen.
```

Listing 3: Nutzung des Teststreifengenerators per Kommandozeile

**Beschreibung des XML-Exports und Imports** Damit der Bildverarbeitungsalgorithmus auf die verschiedenen, veränderbaren Parameter der Teststreifen reagieren kann, wurde eine Schnittstelle geschaffen, mit der die Parameter in eine XML-Datei exportiert werden können. So können diese später an den Bildverarbeitungsalgorithmus übergeben und verarbeitet werden. Der Export in die XML-Datei erfolgt automatisch, wenn ein Teststreifen als Datei abgespeichert wird. Die XML-Datei bekommt, bis auf die Endung, den gleichen Namen wie die zugehörige Bilddatei. So lassen sich die Dateien später noch einander zuordnen.

Um die XML-Dateien zu erzeugen wurde eine Schema-Datei erstellt. Mit Hilfe von JAXB[55] wurden aus der Schema-Datei die erforderlichen Java-Objekte erzeugt, mit denen im Quellcode gearbeitet werden kann. Über die JAXB-Schnittstelle lassen sich die Objekte mit einem einfachen Methodenaufruf in eine XML-Datei überführen. Des Weiteren ist es über einen weiteren Methodenaufruf möglich, aus einer XML-Datei wieder einen Teststreifen zu erstellen und in der Anwendung anzuzeigen. Dies erfolgt ebenfalls über einen Methodenaufruf der JAXB-Schnittstelle.

Folgende Parameter sind in der XML-Datei enthalten:

- Farbumschläge: Es können beliebig viele Farbumschläge erzeugt werden. Diese werden jeweils mitsamt ihrer Größe, Position, Intensität und ihres Filters in die XML-Datei geschrieben.
- Farbe: Für die Farbumschläge lässt sich eine Farbe festlegen. Die einzelnen Werte für Rot, Grün und Blau werden in der XML-Datei gespeichert.
- Weitere Informationen: Es können verschiedene Codes, wie z.B. Barcodes oder QR-Codes auf den Teststreifen gedruckt werden. In der XML-Datei wird gespeichert, um welchen Code es sich handelt, was in dem Code enthalten ist, sowie die Position und Größe des Codes.
- Marker: Es können beliebig viele Marker erzeugt werden. In der XML-Datei werden die Größe, die Position und der Wert des Markers gespeichert.

[55] Java Architecture for XML Binding: `https://de.wikipedia.org/wiki/Java_Architecture_for_XML_Binding`

- Sonstige Informationen: In den sonstigen Informationen sind der Pfad zu der Hintergrunddatei, die Anzahl der DPI, das Format der Farbinformationen und ein Kontrollkästchen, ob die Informationen auf den Teststreifen gedruckt werden sollen.

**Technologien**

**JAXB** Java Architecture for XML Binding, kurz JAXB wird als neue Technologie innerhalb des Teststreifens verwendet um ein XML Im- sowie Export eines Teststreifendesigns zu ermöglichen.

**Systembeschreibung** Der Teststreifengenerator wurde in den Entwicklungsumgebungen Eclipse und Netbeans weiterentwickelt. Externe Bibliotheken sind mit Hilfe von Maven eingebunden worden. JAXB ist bereits in der Java-Installation enthalten.

**Evaluation** Der Quellcode wurde einem einfachem Funktionstest unterzogen. Tests die in dem vorangegangenem Arbeitspaket für den Teststreifengenerator (JUnit) geschrieben wurden, verlaufen ohne Fehler.

**Parameter** Der Teststreifengenerator wird durch keine Parameter eingeschränkt. Es liegt in der Verantwortung des Benutzers, den Teststreifengenerator so zu benutzen, dass sinnvolle Teststreifen entstehen und die Marker beispielsweise nicht übereinander liegen.

**Fazit** Das Arbeitspaket „Anpassen des Teststreifengenerators“ konnte innerhalb der geforderten Frist von zwei Wochen abgeschlossen werden. Der dabei entstandene Teststreifengenerator wirkt unterstützend für die Arbeitspakete 4.5.4.1 auf der nächsten Seite „Anpassung Bildalgorithmen auf dynamische Teststreifen“ und 4.5.5.1 auf Seite 245 „Weiterentwicklung der Bildverarbeitung“.

**Ausblick** Es ist durchaus sinnvoll, dass die Nutzung des Teststreifengenerators nicht nur der Projektgruppe selbst vorbehalten ist, sondern dass auch bestimmte Nutzergruppen - welche gilt es noch zu identifizieren - der Weboberfläche Zugriff auf die Funktionalitäten erhalten um neue Teststreifenvarianten zu gestalten. Um eine einfache Handhabung und zentrale Verwaltung zu gewährleisten, wäre es denkbar, dass die Logik des Teststreifengenerators zu einem späterem Zeitpunkt in die Weboberfläche integriert wird und dort weitere Funktionen wie „Drag and Drop“ oder das Ändern der Größe durch ziehen von Eckpunkten einer Form realisiert wird.

### 4.5.4 Anpassung der Bildalgorithmen auf dynamische Teststreifen

- Motivation und Nutzen des Arbeitspaketes: Da das Teststreifendesign unterschiedlich sein kann, müssen z. B. die Farbumschlagfelder an unterschiedlichen Stellen detektiert werden können. Dass die Teststreifen variabel sein können, geht unter anderem aus den Anforderungen:

  FA-37: Die Administrationsoberfläche bietet die Möglichkeit, neue Teststreifenvarianten anzulegen.

  FA-27: Bei der Erstellung einer Diagnose stellt die Auswertungsoberfläche dem Mediziner Informationen des zugrundeliegenden Tests bereit. Diese Informationen sind das Foto des Teststreifens, die geographischen Daten des Tests und detaillierte Informationen zur Teststreifenvariante.

  hervor. Diese sagen aus, dass es unterschiedliche Teststreifen geben kann.

- Priorität: Normal

- Arbeitspaketbeschreibung:
  Der Bildverarbeitungsalgorithmus zur Detektierung der Farbumschlagfelder unterstützt dynamische Teststreifen. Das bedeutet, dass z. B. Referenzfelder und Farbumschlagfelder an unterschiedlichen Stellen auf dem Teststreifen positioniert sein können und trotzdem erkannt werden müssen.

- Vorbedingungen:
  -

- Nebenbedingungen:
  Eine kontinuierliche Absprache mit dem Team der Bildverarbeitungsalgorithmen, welches für die Optimierung der Markerdetektierung zuständig ist und mit dem Team, welches den Teststreifengenerator anpasst, ist von Nöten.

- Nachbedingungen:
  Der Bildverarbeitungsalgorithmus erkennt verschiedene Varianten der Teststreifen, die sich in bestimmten Parametern unterscheiden. Diese Parameter wurden im Zuge dieses Arbeitspakets und in Absprache mit dem Team der Markerdetektierung erarbeitet und schriftlich festgehalten.

- Aufwand:
  Vier Wochen

- Personen:
  - Raphael Kappes
  - Christian Sandmann

**4.5.4.1 Dokumentation** In diesem Arbeitspaket ging es darum, den bisherigen Algorithmus für die Bildverarbeitung so anzupassen, dass dynamische Teststreifen erkannt werden. Die dynamischen Teststreifen zeichnen sich dadurch aus, dass sie eine beliebige Anzahl an Markern und Farbumschlägen haben können. Außerdem ist die Anordnung von diesen beliebig und das Farbumschlagsfeld kann außerhalb des Bereichs, der von Markern umrandet wird, liegen. Der Algorithmus soll so angepasst werden, dass am Ende die Positionen der Farbumschlagsfelder bestimmt werden.

**Ablauf** Das bisherige Konzept der Auswertung des Teststreifens ist nicht mehr möglich. Es wurde eine affine Transformation[56] verwendet. Diese hatte einen quadratischen Bereich, der durch die Marker definiert wurde, auf ein 200 Pixel mal 300 Pixel großes Bild abgebildet. Die Transformation hat zusätzlich das Bild bezüglich perspektivischer Fehler und Rotation korrigiert. Da das Farbumschlagsfeld nicht mehr innerhalb dieses Bereichs liegt, müssen die Positionen der Farbumschlagsfelder anders bestimmt werden.
Dafür wurde ein Konzept erarbeitet, das die Positionen der Farbumschläge bestimmt, indem ein Teststreifen, dessen Parameter vollständig bekannt sind, mit einem aufgenommen Teststreifen verglichen wird. Die Parameter des Teststreifens sind die Positionen der Marker und der Farbumschläge und werden vom Teststreifengenerator geliefert. Diese werden im folgenden Referenzwerte genannt.
Neben der Bestimmung der Positionen der Farbumschläge auf dem Teststreifen musste der gesamte bisherige Algorithmus angepasst werden, da dieser vier Marker mit festem Abstand, fest definiertem Inhalt und einer festen Position des Farbumschlagsfeldes erwartete.
Die Positionen der Farbumschlagsfelder können durch Verwendung mehrerer mathematischer Modelle bestimmt werden.

[56] Affine Abbildung: `https://de.wikipedia.org/wiki/Affine_Abbildung`

- Verhältnis:
  Die Referenzwerte sind Positionen in einem zweidimensionalem Koordinatensystem und können als Vektoren dargestellt werden. Das gleiche gilt für die Positionen der detektierten Marker. Der Abstand zwischen den Referenzmarkern und der Abstand zwischen den detektierten Markern sind die Längen der Vektoren zwischen den jeweiligen Positionen. Das Verhältnis der Pixel zwischen den Vektorenlängen berechnet sich indem diese dividiert werden. Alle Referenzvektoren können nun durch das Verhältnis dividiert werden und so auf die Länge der Vektoren des aufgenommenen Bildes abgebildet werden.

- Schwerpunkt einer Figur:
  Der Schwerpunkt einer Figur ist der Punkt auf der Oberfläche um dem die größte Fläche liegt.

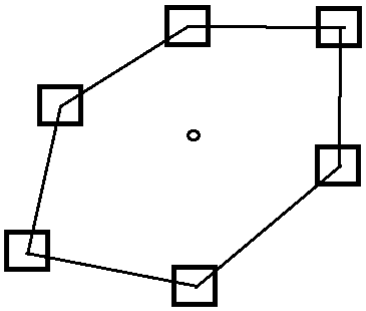


Abbildung 179: Figur mit dem geometrischem Schwerpunkt

Die Abbildung 179 zeigt eine Figur mit ihrem Schwerpunkt. Die Quadrate stellen die erkannten Marker dar und spannen die Figur auf. Die Kanten zwischen den Markern können als Vektoren betrachtet werden und verbinden jeweils die Marker. Der Kreis ist der geometrische Schwerpunkt der Figur. Der Schwerpunkt verschiebt sich durch die Form der Figur. Eine perspektivische Verzerrung verändert eine Figur in einem aufgenommenen Bild, wodurch sich der Schwerpunkt entsprechend der Verzerrung verändert. Vom Schwerpunkt, der durch die Referenzwerte aufgespannten Figur, aus kann ein Vektor auf die Farbumschläge berechnet werden. Dieser wird mit dem Verhältnis dividiert und auf den Schwerpunkt der Figur, die durch die detektierten Marker aufgespannt wird, addiert. Dies ergibt einen Vektor, der auf einen Farbumschlag im aufgenommenen Bild zeigt. Dieser ist jedoch noch durch Rotation fehlerbehaftet.

- Rotation:
  Das aufgenommene Bild ist um eine gedachte z-Achse rotiert. Das heißt, dass auch die Figur, welche durch die detektierten Marker aufgespannt wird, mit rotiert ist. Dabei ist die Figur um ihren Schwerpunkt rotiert. Die Rotation der Figur kann dann über ihre Innenwinkel berechnet werden. Dafür wurde jede Ecke der Figur, wie in Abbildung 180 auf der nächsten Seite dargestellt, beschrieben.

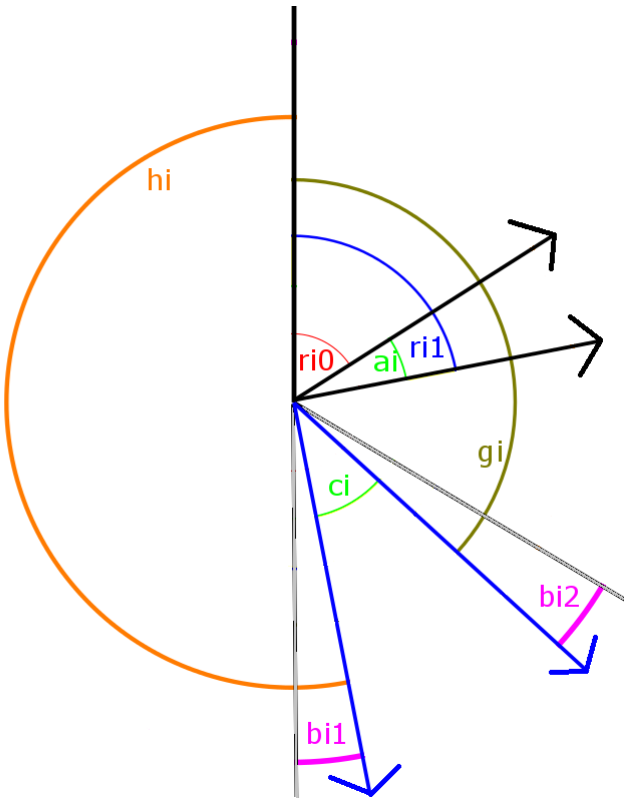



Abbildung 180: Parametrisierung eines Kreises um eine Ecke

Die schwarzen Vektoren stellen dabei die Verbindungen der Ecken in der Referenzfigur dar, die blauen Vektoren sind die Verbindungen der Ecken in dem aufgenommenen Bild. Diese sind in dem aufgenommenen Bild um die gedachte z-Achse um den Winkel $r_i$ rotiert.
Die Veränderung der Innenwinkel um $d_i$ durch eine Rotation der Referenzfigur um die x-Achse oder y-Achse stellt die perspektivische Verzerrung dar.
Diese Verzerrung lässt sich in die Bereiche $b_{i1}$ und $b_{i2}$ mit $d_i = b_{i1} + b_{i2}$ teilen, da die blauen Vektoren in Abbildung 180 unabhängig voneinander verschoben sein können. Der Winkelabstand $a_i$ beider schwarzer Vektoren lässt sich in einen Zusammenhang mit den beiden blauen Vektoren bringen. Dabei gilt dieser Zusammenhang $a_i = c_i + d_i$. Zusätzlich gilt, dass Marker weiter weg sind, wenn $d_i > 0$ und näher dran, wenn $d_i < 0$, wobei $D_p = \mid \sum_{i=0}^{n-1} d_i \mid$ mit allen $d_i > 0$, $D_n = \mid \sum_{i=0}^{n-1} d_i \mid$ mit allen $d_i < 0$ und $D_p = D_n$.

Die folgenden Winkel werden durch einen festgelegten Vektor berechnet, welcher die y-Achse darstellt. Die blauen Vektoren in der Abbildung 180 sind um den Winkel $g_i = r_{i0} + r_i + b_{i2}$ von dieser Achse verschoben. Der Gegenwinkel ist $h_i = 360 - (g_i + c_i + b_{i1})$ der blauen Vektoren. Werden diese Winkel gegeneinander gestellt, ergibt sich für die Rotation folgendes: $r_i = r_{i0} + b_{i2} - g_i$, für $b_{i1} = 360 - (g_i + c_i + h_i)$ und $b_{i2} = a_i - c_i - b_{i1}$.

Durch die Verschiebung der Innenwinkel innerhalb der Figur ist die Rotation für jeden Marker anders und zwar um den Winkel $d_i$. Diese Variation der Rotationen kann gelöst werden, indem die gemittelte Rotation $mr_i$ berechnet wird, da gilt $D_p - D_n = 0$. Die Richtung der Vektoren, die im aufgenommenen Bild vom Schwerpunkt der Figur (es ist die, welche durch die detektierten Marker aufgespannt wird) auf die Position der Farbumschläge zeigen sollen, können nun um den durchschnittlichen Winkel $mr_i$ geändert werden. Dadurch zeigen diese auf die Farbumschläge.

Nachdem nun die Positionen bestimmt wurde, werden diese im aufgenommenen Foto ausgeschnitten (siehe Abbildung 181) und können weiter verarbeitet werden.


Abbildung 181: Teststreifen. Die Farbumschläge sind mit grünen Kästen umrandet

**Technologien**

**OpenCV** Bibliothek für die Bildverarbeitung

**Systembeschreibung** Die verwendete Entwicklungsumgebung war Eclipse Mars, als Betriebssystem wurde Windows 7 verwendet. Für die Einbettung des C/C++ Codes in Eclipse wurde MinGW verwendet.

**Evaluation** Der überarbeitete Algorithmus wurde mit mehreren Videos getestet. Dabei wurden die Positionen der Farbumschläge immer bestimmt, wenn alle Marker erkannt wurden. Aus zeitlichen Gründen konnten nur Teststreifen mit drei Markern getestet werden. Dies begründet sich einerseits dadurch, dass die Apps noch nicht mit dieser Funktion vollständig lauffähig sind (dieses war nicht Teil der Arbeitspaketbeschreibung) und andererseits, dass noch keine Verwaltung für Teststreifenvarianten auf den Smartphones vorhanden ist (dieses war nicht Teil der Arbeitspaketbeschreibung). Dadurch war es sehr aufwendig überhaupt eine Evaluation dieser Funktion durchzuführen. Daher muss zu einem späteren Zeitpunkt diese Funktion ausführlicher getestet werden.

**Parameter** Es müssen alle Marker erkannt werden, damit die Positionen der Farbumschläge bestimmt werden können.

**Fazit** Es besteht nun die Möglichkeit den Farbumschlag für dynamische Teststreifen zu bestimmen. Eine Schwierigkeit war am Anfang die unklare Definition des Arbeitspakets. Eine andere Schwierigkeit war, dass keiner der Beteiligten des Arbeitspakets vorher Erfahrung mit dem Quellcode der Bildverarbeitung hatten. Trotzdem konnte das Arbeitspaket in der vorgegebenen Zeit von vier Wochen beendet werden.

**Ausblick** Die Werte aus dem Idealbild sollen aus der XML-Datei für den jeweiligen Teststreifen gelesen werden. Dies war nicht Teil dieses Arbeitspakets, daher werden im Moment Testdaten eingetragen. Dies ist in kommenden Sprints so zu ändern, dass die Daten automatisch ausgelesen werden.

### 4.5.5 Weiterentwicklung der Bildverarbeitung

- Motivation und Nutzen des Arbeitspaketes: Ziel ist es, die Markererkennung effizienter zu gestalten. Dies führt dazu, dass folgende nicht funktionale Anforderungen teilweise umgesetzt werden können:

  NFA-11: Das Ergebnis der Voranalyse muss dem Probanden nach maximal zehn Sekunden mitgeteilt werden.

  NFA-13: Die mobile Applikation muss so ressourcenschonend arbeiten, dass sie auch auf Low-Cost-Smartphones ausführbar ist.

  Benutzerfreundlichkeit lässt sich nicht nur durch ein schlankes, einfaches Design, sondern auch auf effiziente Algorithmen zurückführen. So muss der Proband das Smartphone z. B. nur eine bis drei Sekunden über den Teststreifen halten, um die elementaren Bereiche zu detektieren und die Voranalyse anzustoßen.

- Priorität: Normal

- Arbeitspaketbeschreibung:
  Nachdem das Arbeitspaket abgeschlossen wurde, können mit den schlechtesten Smartphones (Sony Xperia E1 und LG Bello) die Marker aus unterschiedlichen Entfernungen und unterschiedlichen Lichtverhältnissen in einer maximalen Zeit von drei Sekunden detektiert werden können. Des weiteren wurde eine Tabelle erstellt, welche die Ergebnisse verschriftlicht und eine Hypothese dazu unterstützt, ab welcher Entfernung und bei welchen Lichtverhältnissen die Marker mit Low-Cost-Smartphones erkannt werden können.

- Vorbedingungen:
  -

- Nebenbedingungen:
  Um die unterschiedlichen Licht- und Entfernungsverhältnisse zu simulieren, kann der iTeststand herangezogen werden.

- Nachbedingungen:
  Der Algorithmus erkennt die Marker in maximal drei Sekunden. Die dafür herrschenden Lichtverhältnisse und Entfernungen sind evaluiert.

- Aufwand:
  Vier Wochen

- Personen:
  - Timo Schlömer
  - Timo Raß

**4.5.5.1 Dokumentation** Das Ziel dieses Arbeitspakets war es, die Algorithmen für die Erkennung der Marker auf dem Teststreifen so zu verbessern, sodass die Zeit für die Analyse eines Bildes reduziert wird und die Markerdetektion bei allen Endgeräten erfolgreich die Marker des Teststreifens detektiert.

**Ablauf** Um die Geschwindigkeit und Robustheit der Analyse zu verbessern, wurden diverse Bereiche des Quellcodes angepasst. Die Ergebnisse und Hintergründe der Änderungen sind im Folgenden zusammengefasst:

**Anpassung der Markerstruktur** Nachdem damit begonnen wurde, Optimierungspotentiale zu identifizieren stellte sich heraus, dass einer der vier verwendeten Marker wesentlich schlechter erkannt wurde als die drei anderen. Dies hatte zur Folge, dass das Bild und somit auch der Farbumschlagsbereich nicht identifiziert wurden und die Durchführung eines Tests sehr lange dauerte. Daraufhin wurden unterschiedliche Markeranpassungen durchgeführt. In Abbildung 182 ist der alte Marker dargestellt, in Abbildung 183 der neue, angepasste Marker, welcher sich als beste und am schnellsten detektierbare Alternative herausstellte. Dieser wird durch die Detektierung mit OpenCV jetzt regelmäßig und schneller erkannt.

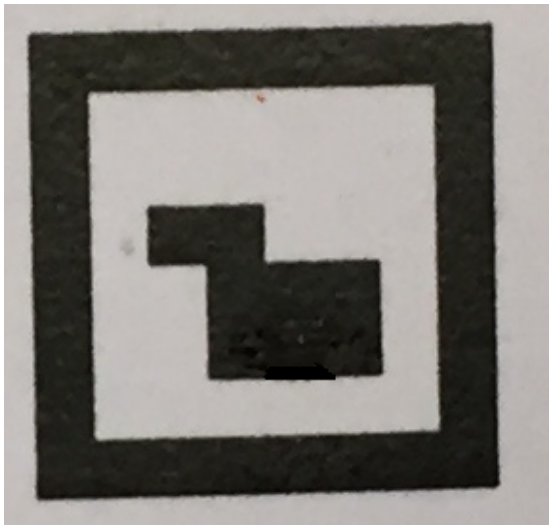


Abbildung 182: Alter, schlecht identifizierbarer Marker

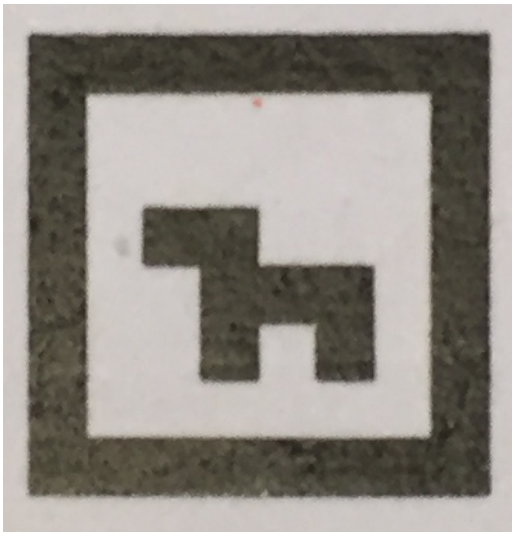


Abbildung 183: Neuer, schnell identifizierbarer Marker

**Das zu analysierende Bild verkleinert** Bislang wurde dem Bildverarbeitungsalgorithmus das komplette aufgenommene Bild übergeben, was bei hochauflösenden Kameras eine große Menge an zu analysierenden Pixeln bedeutet. Es wurde angenommen, dass ein Benutzer den Teststreifen immer mit einer gewissen Distanz zum Smartphone aufnimmt, wodurch der Teststreifen nicht den kompletten Bereich des Bildes ausmacht.

Der zu analysierende Bereich wurde auf ein Drittel der Originalgröße reduziert (ausgehende vom Mittelpunkt des Bildes), wodurch der Algorithmus erheblich schneller lief. Ein Durchlauf dauerte im Durchschnitt nicht mehr **3 Sekunden**, sondern um die **0.5 Sekunden**.

Um den Benutzern den tatsächlich analysierten Bereich des Bildes zu zeigen, wurde der grüne Rahmen, welchen die iOS-Prototyp-App bereits anzeigte (aber keine Bedeutung hatte), auf die korrekte Größe angepasst und auch auf die Android-Prototyp-App portiert.

**Gefundene Marker zwischenspeichern** Es wurde festgestellt, dass der Blobdetektor die Marker in manchen Situationen nicht erkennt, obwohl der restliche Algorithmus mit den aufgenommenen Markern weiter arbeiten könnte. So konnte etwa auftreten, dass im ersten aufgenommenen Bild drei von vier Markern gefunden wurden, im darauf folgenden Bild der vierte Marker auch erkannt wurde, aber nun einer der vorherigen Marker fehlt. So wurden zwar alle Marker gefunden, jedoch nicht im gleichen Bild, wodurch der Algorithmus den Teststreifen nicht erkennt.

Der Algorithmus wurde so verändert, dass er eine Liste von möglichen Positionen von Markern als Eingabeparameter akzeptiert. Diese möglichen Positionen werden wie die gefundenen Blobs des Blobdetektors behandelt und werden auf die gleiche Weise mit den Referenzmarkern verglichen. Die im vorigen Abschnitt beschriebene Situation würde nun den Teststreifen erkennen können, da die Positionen der im ersten Bild gefundenen Marker an die Analyse des zweiten Bildes übergeben werden und der Algorithmus nun die Marker unabhängig vom Blobdetektor detektieren kann. Die Robustheit der Detektierung konnte mit dieser Änderung stark erhöht werden.

In einer ersten Implementierung wurden die Positionen der bereits gefundenen Marker für die restliche Zeit der Anwendung gespeichert. Dadurch wurden bis zu vier zusätzliche Positionen auf mögliche Marker überprüft, zusätzlich zu den Blobs die vom Blobdetektor gefunden wurden. Dies verlangsamte den Algorithmus. Als Reaktion darauf wurde der Algorithmus weiter verändert, sodass die gefundenen Marker nur an das nächste Bild übergeben werden. So werden nicht unnötig Positionen analysiert, wenn im vorherigen Bild keine Marker detektiert werden konnten. Diese Änderung schadete der Robustheit nicht, hat sich jedoch positiv auf die Performanz ausgewirkt.

**Optimierung der morphologischen Öffnung** Die morphologische Öffnung kommt bei der Untersuchung der Marker zum Einsatz, um Ungenauigkeiten, die durch ein unscharfes Bild oder durch Verdrehung entstehen, zu kompensieren. Die Stärke der Öffnung kann durch einen Parameter konfiguriert werden, welcher bislang nicht genauer untersucht wurde. Das Ziel war es, diesen Parameter so zu wählen, dass korrekte Marker akzeptiert werden und falsche Marker abgelehnt werden. Diese Entscheidung wird getroffen, indem die nicht-schwarzen Pixel gezählt werden, die nach diversen Filtern bei einem Marker übrig bleiben.

Wird der Parameter der morphologischen Öffnung erhöht, werden größere Bereiche von nicht-schwarzen Pixeln im Bild entfernt. Ist der Parameter zu groß, werden falsche Marker detektiert. Wird der Parameter reduziert, werden nur kleinere Bereiche von nicht-schwarzen Pixeln entfernt. Ist der Parameter zu gering, werden korrekte Marker nicht akzeptiert.

Nach vielen Tests zeigte sich, dass ein „Structuring Element “[57] mit rechteckiger Form und einer Größe von $(\frac{height}{9} \times \frac{width}{9})$ für ein optimales Ergebnis gewählt werden muss. Das Ergebnis der morpholigischen Öffnung ist ein Graustufen-Bild, was bei korrekten Markern immer noch nicht-schwarze Pixel enthielt. Nach der Öffnung wird nun zusätzlich eine einfache Binarisierung mit einem Schwellwert von **125** durchgeführt, wodurch korrekte Marker **keine nicht-schwarzen Pixel** mehr aufweisen. Ein Marker wird nun nicht mehr bei $\frac{np}{p} < 0.03$ ($np$ = Anzahl nicht-schwarzer Pixel, $p$ = Anzahl Pixel) akzeptiert, sondern bei einer Fehlertoleranz von exakt $\frac{np}{p} = 0$.

**Marker-Referenzen werden nur noch einmal generiert** Bisher wurden die Referenzmarker für jeden analysierten Marker neu erzeugt. Dies kostete für jede Erzeugung bis zu **5 Millisekunden**, welche für jeden der vier Marker und für jede der vier Ausrichtungen anfiel. Wenn alle Marker auf dem Teststreifen detektiert wurden mussten $4Blobs \times 4Marker \times 4Ausrichtungen = 64$ Referenzen generiert werden.

Diese Referenzen werden nun beim Laden der Bibliothek einmalig erzeugt. Der Zeitgewinn durch diese Änderung macht sich vor allem bei vielen gefundenen Blobs bemerkbar.

[57] Structuring Element: `http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html?highlight=getstructuringelement#getstructuringelement`

**Vergleiche mit Marker-Referenzen reduziert** Wie im vorherigen Absatz beschrieben, wurden potentielle Marker immer mit allen Referenzen verglichen, was im schlimmsten Fall **16 Vergleiche** bedeutet. Die Anzahl an Vergleichen konnte jedoch reduziert werden, indem der Algorithmus nach einer erfolgreichen Detektion eines Markers die Rotation von diesem speichert und weitere potentielle Marker nur noch mit Referenzen der gleichen Rotation vergleicht. Ist ein Marker erfolgreich detektiert worden, benötigen die folgenden Marker nur noch bis zu **4 Vergleiche**.

**Fehler in der iOS-Prototyp-App behoben** Die bisherige iOS-Prototyp-App enthielt einen Fehler, durch den jedes aufgenommene Bild der Kamera als Datei auf dem Speicher des Gerätes abgelegt wurde. Nachdem dieser Fehler entfernt wurde, konnten ca. **12-17** Bilder pro Sekunde verarbeitet werden. Zuvor waren es lediglich **ein bis zwei** Bilder.

**Bilder-Pro-Sekunde in Android und iOS anzeigen** Um eine exakte Aussage darüber treffen zu können, wie viele Bilder pro Sekunde verarbeitet und analysiert werden können, wurde ein funktionierender Frames per Second (FPS)-Zähler eingebaut. Dieser zählt, im Gegensatz zum bisherigen Zähler, die Anzahl an Durchläufen der Bildverarbeitung pro Sekunde. Der alte Zähler brachte ausschließlich die Funktion mit sich, alle Bilder, welche insgesamt analysiert wurden zu summieren. Dieser Wert war jedoch wenig aussagekräftig und wurde daher durch die neue Funktion ersetzt.

**Speicherverbrauch in Android optimiert** Nach einer Analyse des Speicherverbrauchs der Android-Prototyp-App zeigte sich, dass an drei Stellen im Code sehr häufig Objekte erzeugt werden, welche den Speicher des Geräts stark beanspruchen und häufig eine „Garbage Collection“[58] auslösen.

Der größte Speicherverbrauch entstand dadurch, dass für jedes aufgenommene Bild der Kamera ein neuer Speicherbereich gefüllt wurde. Diese sammelten sich immer weiter an, bis die Garbage Collection diese wieder frei geräumt hat. Dieses Verhalten konnte korrigiert werden, indem die Daten des vorherigen Bildes mit den Daten des neuen Bildes überschrieben werden. Der gleiche Speicherbereich wird nun für jedes Bild wiederverwendet.

Die anderen beiden Problembereiche entstanden durch Objekte, welche bei jedem Durchlauf des Algorithmus neu erstellt wurden, obwohl dies gar nicht nötig war. Diese Objekte werden nun am Start der Analyse erzeugt und danach weiterverwendet.

Durch diese Anpassungen wurde der Speicherverbrauch und die Anzahl an Garbage-Collections reduziert, was jedoch keine Auswirkungen auf die Dauer der Bildanalyse hatte.

**Autofokus in iOS** Der Autofokus für die iOS-Prototyp-App wurde so angepasst, dass der Bildverarbeitungsalgorithmus erst dann ein Bild analysiert, wenn der Autofokus abgeschlossen ist. Der Autofokus wird in iOS automatisch von der Kamera durchgeführt. Dies führte oft dazu, dass während einer neuen Kalibrierung des Fokus das Bild trotzdem analysiert wurde und in wenigen Fällen auch erkannt wurde. Wenn das Bild erkannt wurde, waren die Bilder jedoch oft sehr verschwommen und unscharf. Nachdem die Anpassung abgeschlossen war, wartet der Bildverarbeitungsalgorithmus so lange, bis das Bild durch den Fokus geschärft wurde und beginnt erst dann die Markerdetektion. Wird das Bild wieder unscharf und der Fokus muss erneut kalibrieren, wartet der Bildverarbeitungsalgorithmus und rechnet erst weiter, wenn die Fokussierung abgeschlossen ist.

**Technologien** Es wurden keine neuen Technologien in diesem Arbeitspaket eingesetzt.

**Systembeschreibung** Es gelten die gleichen Bedingungen wie für das Arbeitspaket „Erneute Machbarkeitsstudie für einen einheitlichen Bildverarbeitungsquellcode“ (Im vorherigen Sprint in Kapitel 4.4.11 auf Seite 181).

[58] Garbage Collection: `https://de.wikipedia.org/wiki/Garbage_Collection`

**Evaluation** Die Evaluation für die Optimierung der Bildverarbeitungsalgorithmen wurde im Teststand und unter Umgebungslicht durchgeführt. Im Teststand wurden abwechselnd die 2700 Kelvin und die 5700 Kelvin Glühbirne verwendet. Es wurden Abstände, von Endgerät zum Teststreifen, in zehn cm Schritten gewählt, begonnen bei 20 cm Entfernung, bis hin zu 50 cm Entfernung. Diese Skala wurde bewusst so gewählt, da bei einer normalen Durchführung des Testes der Abstand zwischen Endgerät und Teststreifen zwischen 20 cm und 50 cm liegt. Der Teststreifen und das Endgerät waren vertikal aufgestellt.

Werden alle Ergebnisse ohne Betrachtung der Lichtverhältnisse und der Entfernung summiert, werden von 108 durchgeführten Tests 20 Teststreifen nicht erkannt, 88 werden erkannt. Dies kann Abbildung 184 entnommen werden.

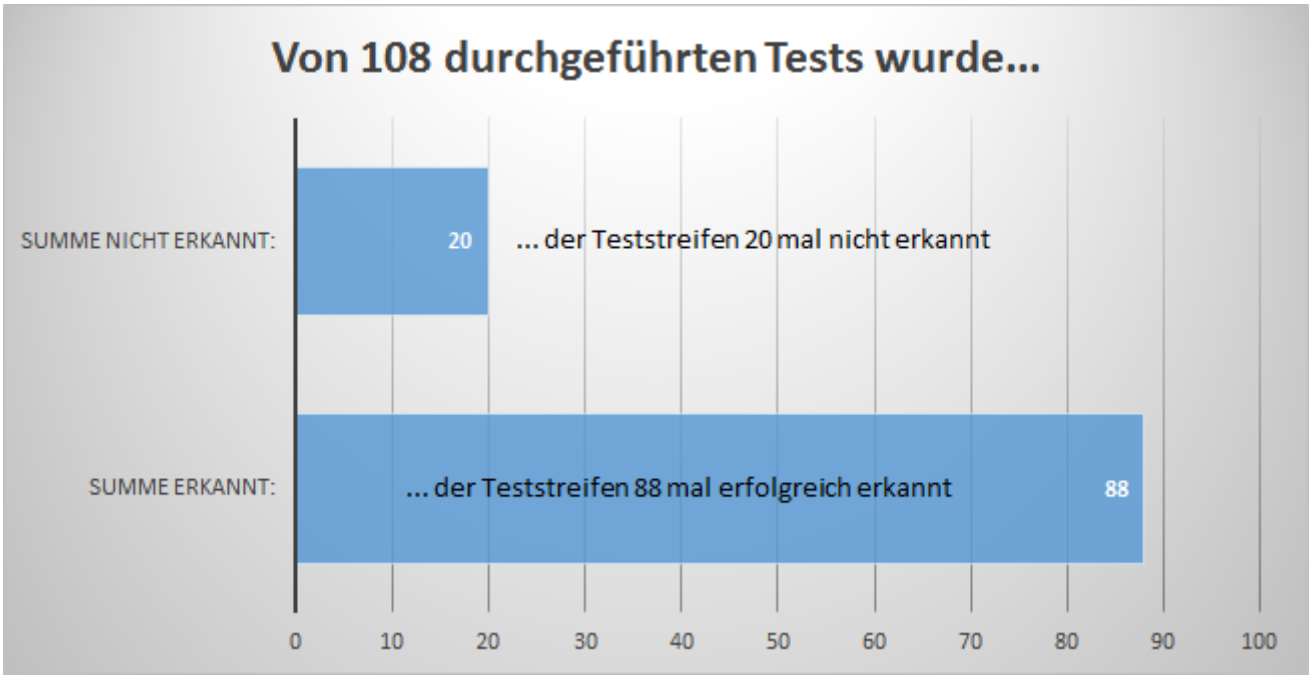



Abbildung 184: Zusammenfassung aller Evaluationen ohne Berücksichtigung von Licht und Entfernung

Da die zuvor gezeigten Ergebnisse nur als zusammenfassender Überblick dienen sollen, werden die exakten Ergebnisse der Evaluation zu jedem Endgerät, jeder Entfernung und den unterschiedlichen Lichtverhältnissen im Folgenden genau skizziert. Für die Evaluation wurden neben den, für das Projekt beschafften Endgeräten, auch diverse private Endgeräte herangezogen. Alle Informationen können den Tabellen entnommen werden.

Wenn der Teststreifen nicht erkannt wurde, wurde die Evaluation für die Entfernung und die Lichtverhältnisse zwei weitere Male durchgeführt. Wurde der Teststreifen in diesen Durchläufen nicht erkannt, galt der Teststreifen als nicht detektier- bzw. erkennbar.

**LG Bello (Low-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 8: Evaluationsergebnisse LG Bello

**Sony Xperia E1 (Low-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Nein |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 9: Evaluationsergebnisse Sony E1

**Motorolla Moto G (Low-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 10: Evaluationsergebnisse Motorola Moto G

**Oneplus One (High-Cost)**


| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 11: Evaluationsergebnisse oneplus one

**Son Xperia Tipo (Low-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Nein |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 12: Evaluationsergebnisse Xperia Tipo

**Sony Z3 Compact (High-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 13: Evaluationsergebnisse Sony Z3

**iPad mini 2 (Low-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 14: Evaluationsergebnisse iPad mini 2

**iPad mini 4 (High-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 15: Evaluationsergebnisse iPad mini 4

**iPhone 6 (High-Cost)**

| Entfernung | Umgebung | Licht | Winkel | Erkannt? |
|---|---|---|---|---|
| 50 cm | Teststand | 2700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 2700 Kelvin | Vertikal | Ja |
| 50 cm | Teststand | 5700 Kelvin | Vertikal | Nein |
| 40 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 30 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 20 cm | Teststand | 5700 Kelvin | Vertikal | Ja |
| 50 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 40 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 30 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |
| 20 cm | Testsreifen auf Tisch | Umgebungslicht | Vertikal | Ja |

Tabelle 16: Evaluationsergebnisse iPhone 6

Die Evaluation hat ergeben, dass es zum jetzigen Zeitpunk möglich ist, mit allen zur Verfügung stehenden Endgeräten den Teststreifen, bei einer Entfernung zwischen 20 cm und 30 cm, erfolgreich zu detektieren und zu analysieren. Das Nicht-Erkennen des Teststreifens vom iPhone 6 im Teststand bei einer Entfernung zwischen 40 und 50 cm ist damit zu begründen, dass durch die gute Kamera zu viele Objekte im Bild detektiert werden (z. B. Kabel und ähnliches). Dennoch kann das Arbeitspaket als erfolgreich abgeschlossen angesehen werden.

**Parameter** Die Funktion des optimierten Algorithmus wird durch keine Parameter eingeschränkt.

**Fazit** Die Durchführung der Optimierung konnte erfolgreich umgesetzt werden. Zum jetzigen Zeitpunkt können die Marker und somit auch der Teststreifen mit allen oben aufgeführten Low-Cost-Smartphones schnell detektiert und analysiert werden. Große Unterschiede in der Zeit für die Detektierung im Vergleich zwischen Low-Cost und High-Cost Endgeräten sind ausgeblieben. Dies resultiert unter anderem daraus, dass die Low-Cost Smartphones zwar weniger Rechenleistung besitzen, die Bilder, welche analysiert werden jedoch auch um ein vielfaches kleiner sind, als es bei High-Cost Endgeräten der Fall ist.

**Ausblick** Es gilt im weiteren Verlauf des Projektes die Optimierungen der Bildverarbeitungsalgorithmen in die Algorithmen für die dynamische Detektierung der Teststreifen zu integrieren. Zudem sind die Bildverarbeitungsalgorithmen aktuell sehr stabil, sodass zwar noch weitere Optimierungen vorgenommen werden könnten, dies aber für die Erfüllung der Anforderungen nicht zwingend notwendig ist.

### 4.5.6 Evaluierung einer Schattenrückrechnung

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes: Eine Schattenrückrechnung kann den Bildverarbeitungsalgorithmus positiv beeinflussen und für genauere Auswertungsergebnisse sorgen.

- Arbeitspaketbeschreibung:
  Am Ende des Arbeitspaketes wurde ein Algorithmus entwickelt, welcher es ermöglicht in einem Bild Schattenpunkte zu identifizieren und diese aus dem Bild herauszurechnen. Ein Schatten auf dem Bild könnte das Ergebnis verfälschen, zudem ist die Wahrscheinlichkeit hoch, dass ein Schatten auf den Teststreifen geworfen wird, wenn der Nutzer das Smartphone über den Teststreifen hält. Eine theoretische Darstellung kann den folgenden Abbildungen 185 und 186 entnommen werden

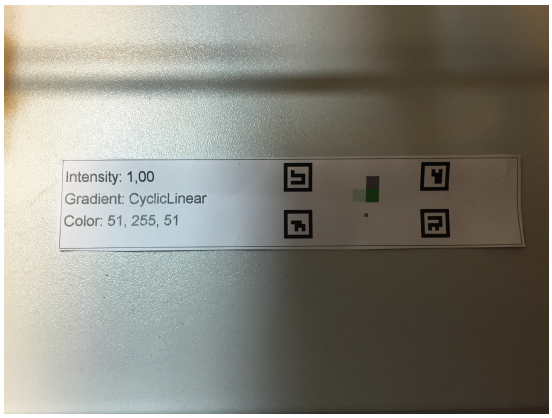



Abbildung 185: Teststreifen mit Schatten

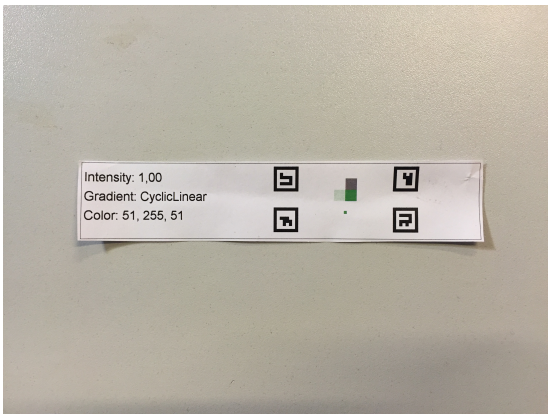



Abbildung 186: Teststreifen ohne Schatten

- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Am Ende soll das Bild ohne Schatteneffekte für weitere Bearbeitungen zur Verfügung gestellt werden.
- Aufwand:
  Zwei Wochen
- Personen:
  - Sebastian Horwege
  - Nicolas Koch

**4.5.6.1 Dokumentation** In diesem Paket sollte ein Algorithmus geschaffen werden, mit dessen Hilfe Schatten aus einem mit dem Smartphone aufgenommenen Bild entfernt werden. Solche Schatten können das Ergebnis des Algorithmus beeinträchtigen und sind daher nicht wünschenswert [15] [16] [17]. Abbildung 187 auf der nächsten Seite zeigt ein Beispiel für ein Bild, in welchem der Schatten programmatisch entfernt wurde.

Abbildung 187: Schattenrückrechnung einer menschlichen Silhouette. Bildquelle: [1]

**Ablauf** Die Schattenrückrechnung kann in die folgenden zwei Unteraufgaben geteilt werden:

1. Detektierung von Schatten auf dem Bild.
2. Entfernung der erkannten Schatten.

Während der Bearbeitung hat sich schnell herausgestellt, dass die Detektierung der Schatten die eigentliche Komplexität der Gesamtaufgabe enthält. Die Rückrechnung der Schatten hingegen ist trivial, sobald die Positionen der Schatten bekannt ist.

Ein weiterer Ansatz, der verfolgt wurde ist die Erstellung von Beleuchtungsinvarianten[59] Bildern [18]. Ist ein solches Bild bekannt, kann die Position des Schattens durch Bildung der Differenz des Beleuchtungsvarianten und Beleuchtungsinvarianten Bildes errechnet werden. Die erste Überlegung war hier, dass Schatten eventuell keine Änderung der Farbe, sondern nur der Intensität bzw. Sättigung des Bildes darstellt. Hierfür bietet sich der **HSV** Farbraum als Grundlage an: Um Schattenkanten zu finden, müssen im Bild Übergänge zwischen hellen und dunklen Bereichen gefunden werden. Der **V**-Kanal des **HSV** Farbraums beschreibt beschreibt die Helligkeit solcher Bereiche. Kanten die über diesem Kanal mit dem Canny-Algorithmus oder Sobel-Algorithmus gesucht werden, treten aber auch an Stellen auf, bei denen es sich nicht um Schatten handelt. Um diese sogenannten „false-positive “ Kanten zu eliminieren, kann ein weiterer Kanal betrachtet werden. Der Farbwinkel (**H**-Kanal) sollte an den Schattenübergängen eine geringere Änderung erfahren, als der Helligkeitskanal. Wählt man nun die Parameter richtig, müssten die Kanten des Farbwinkelkanals, abgezogen der des Helligkeitskanals, nur noch echte Schattenkanten über lassen. Jedoch ist, egal wie die Parameter gewählt werden, mit dieser Methode keine robuste Schattenerkennung möglich. Das liegt daran, dass die Schattenkante nie hart ist. Vielmehr existiert in den meisten Fällen (geschuldet auch durch die kleine Distanz von Kamera und aufgenommenen Objekt) ein weicher Übergang von beleuchteten Regionen, über Halbschatten nach Vollschatten. Dadurch ist es unmöglich, sich überlagernde Kanten zu finden.

Hinzu kommt, dass die Farbtemperatur im Schatten oft auch durch indirektes Licht beeinflusst werden kann. (siehe dazu Abbildung 189 auf der nächsten Seite) Diese Eigenschaft könnte auch zur Schattendetektierung genutzt werden. Jedoch ist die Farbtemperatur nur in wenigen Szenarien eine zuverlässige Aussage über Schatten im Bild. So hat Schatten auf Bildern unter freiem Himmel meist eine Blaufärbung aufgrund der Himmelfarbe. Bei Bildern für die diese Einschränkung nicht gilt wird die Schattenrückrechnung aber auf keinen Fall zuverlässige Ergebnisse liefern. Das macht diesen Ansatz unbrauchbar.

[59] Zum Beispiel Graubilder, die keine Lichteffekte enthalten

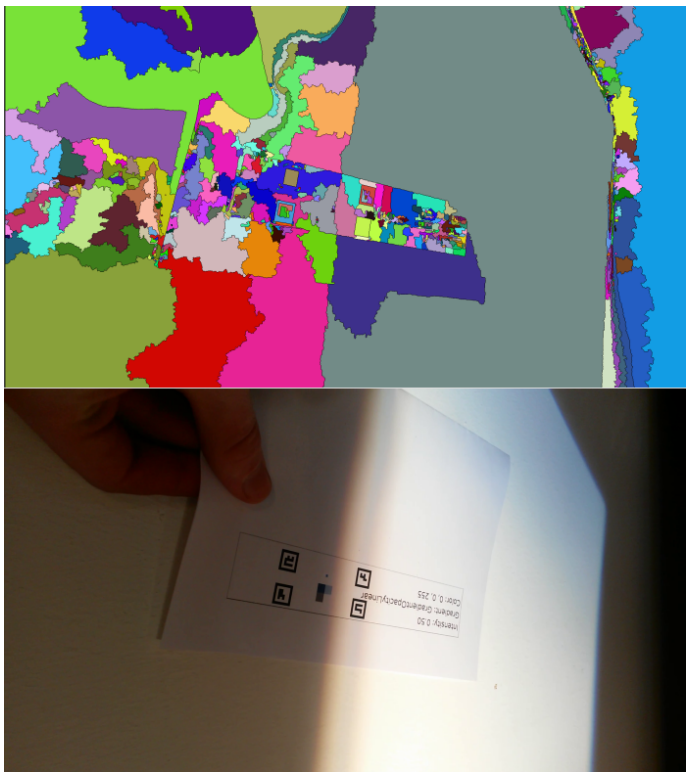


Abbildung 188: Watershed Segmentierung

Der bisher vielversprechendste Ansatz ist die Segmentierung des Bildes in Schatten- und Helligkeitsbereiche mithilfe einer „Watershed Segmentierung “. Dafür wird der L Kanal des „Lab“ - Farbraums[60] im aufgenommenen Bild auf den vollen Dynamikbereich ausgedehnt. Dies gewährleistet, das die dunkelsten Farbbereiche im Bild immer bei 0 und die hellsten immer bei 255 liegen. Über diesem Bild wird dann eine sensible Kantendetektierung mit dem Canny-Algorithmus vorgenommen. Durch das Kantenbild werden nun die Marker detektiert, von denen aus der Watershed Algorithmus im Bild nach homogenen Bereichen sucht. Wie in Abbildung 188 zu erkennen, können dadurch homogen beleuchtete Bereiche gefunden werden. Hierbei kommt es nun darauf an, zusammenhängende Schattenregionen zu erkennen und kleine, dunkle Bereiche zu eliminieren. Dafür muss für jede der gefundenen Regionen die durchschnittliche Beleuchtung ermittelt und ins Verhältnis zu allen anliegenden Regionen gestellt werden. Nach diesen Schritten ist es möglich, alle Regionen relativ zu der am hellsten beleuchteten Region zu korrigieren. Dieser Ansatz konnte aus zeitlichen Gründen nur teilweise umgesetzt werden, verspricht aber gute Ergebnisse zu liefern. Jedoch ist auch zu beachten, dass für diesen Algorithmus einige Schritte nötig sind, die unter Umständen rechenintensiv sein könnten, was sich natürlich auch negativ auf die Performanz des Gesamtalgorithmus auswirken wird.

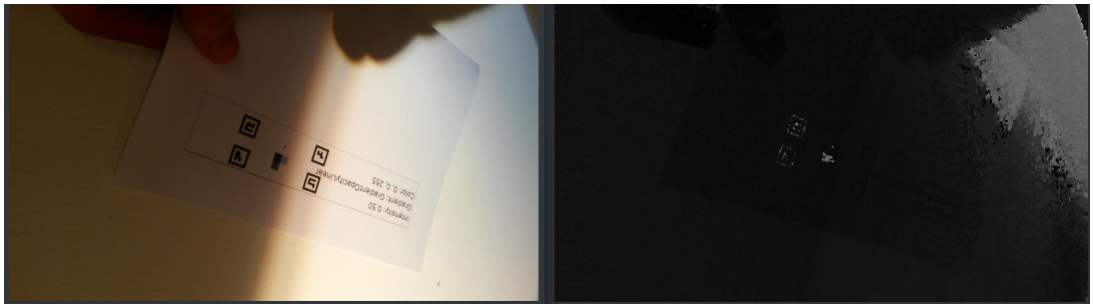


Abbildung 189: Vergleich zwischen originalem Bild und H-Kanal des Bildes

[60] `https://de.wikipedia.org/wiki/Lab-Farbraum`

Ein weiterer Farbraum, der Lichtinvarianz verspricht, ist der Lab Farbraum. Hier wurde der L-Kanal verwendet, um ein möglichst lichtinvariantes Bild zu erzeugen. Leider hat diese Methode auch nicht zum Erfolg geführt, wie Abbildung 190 zeigt.

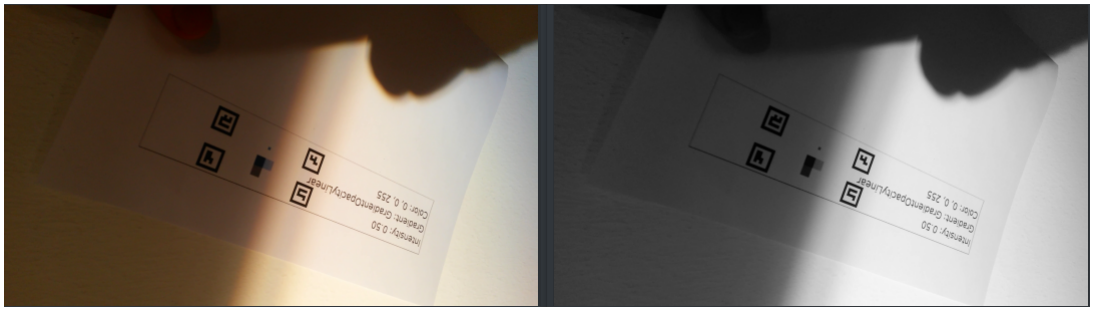


Abbildung 190: Vergleich zwischen originalem Bild und L-Kanal des Bildes

Da die Ansätze Schatteninformationen aus bestimmten Farbräumen zu ziehen leider nicht funktionieren, wurden bekannte Vorgehensweisen zur Schattenerkennung herangezogen. Dabei wurde die Ausarbeitung „Shadow Detection: A Survey and Comparative Evaluation of Recent Methods“ [19] zur Hilfe genommen. Die in der Ausarbeitung beschriebenen Ansätze besitzen zum Teil eine sehr große Komplexität, und würden daher länger als die für das Arbeitspaket zu Verfügung stehenden zwei Wochen in Anspruch nehmen. Ein Ansatz mit dem lichtinvariante Bilder erstellt werden können, ist in [20] beschrieben. Dieser wurde im Rahmen des Arbeitspaketes implementiert, jedoch hat auch er nur zu einem mäßigen Erfolg geführt. Schatten konnten auf einigen Bildern zuverlässig detektiert werden, während bei anderen Bildern, welche eine durchschnittlich niedrige Helligkeit besitzen, auch Nicht-Schatten-Regionen als schattig erkannt werden.
Insgesamt lässt sich sagen, dass zwar einige Ansätze existieren, die sehr erfolgversprechend sind, aber oft detailliertes Wissen über die Kamerasensoren erfordern, welches in diesem Projekt nicht gegeben ist.

**Technologien**

**Testframework** Das im Arbeitspaket „Erstellung eines Test-Frameworks“ (siehe Kapitel 4.5.7.1 auf der nächsten Seite) entstandene Test-Framework erlaubt es, den Bildverarbeitungsalgorithmus außerhalb der mobilen Applikationen auszuführen.

**Fazit** Wie in der Einleitung erwähnt, kann die Schattenrückrechnung generell zur Verbesserung der Performance von Bildverarbeitungsalgorithmen beitragen. Dabei liegt die Schwierigkeit jedoch nicht in der Rückrechnung, sondern in der robusten Erkennung der Schatten.

Es hat sich herausgestellt, dass dieser Teil des Algorithmus nicht nur schwer umzusetzen ist, sondern auch die Performanz des gesamten Algorithmus negativ beeinflusst. Zusätzlich waren einige Ansätze ungeeignet, da diese detaillierte Informationen über den benutzen Kamerasensor benötigen. Diese Informationen sind angesichts der unvorhersehbaren Menge an unterschiedlichen Kameras in den mobilen Endgeräten schlicht unmöglich bereitzustellen. Hätte dieses Arbeitspaket, wie auch die anderen Bildverarbeitungspakete, vier statt zwei Wochen Bearbeitungszeit gehabt, hätte ein besserer Schattenrückrechnungsalgorithmus entstehen können.

**Ausblick** Auch wenn zum Abschluss dieses Arbeitspakets keine Schattenrückrechnung in den Bildverarbeitungsalgorithmus integriert werden konnte, gibt es doch einige Erkenntnisse, die zur Verbesserung des Algorithmus führen können. Eine große Hilfe war das Testframework. Dieses muss jedoch noch weiter ausgebaut werden. Zusätzlich muss die Menge an Testvideos wesentlich erhöht werden, damit Änderungen am Bildverarbeitungsalgorithmus robuster evaluiert werden können.

Daraufhin sollte der Bildverarbeitungsalgorithmus so angepasst werden, dass die Binärisierung des Eingabebildes zur Markerdetektierung über den Luminanz Kanal im **Lab** Farbraum geschieht. Erste Tests haben bestätigt, dass dieser sich besser eignet, als die bloße Graubildberechnung durch OpenCV. Aufgrund der fehlenden Testdaten ist diesbezüglich aber noch keine sichere Aussage möglich.

### 4.5.7 Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen

- Motivation und Nutzen des Arbeitspaketes:
  Ein effizienteres Testen der Bildverarbeitungsalgorithmen soll durch das Testframework ermöglicht werden.
- Priorität: Normal
- Arbeitspaketbeschreibung:
  Um die Bildverarbeitungsalgorithmen nicht nach jeder Änderung neu auf den Smartphones testen zu müssen (ein Kompiliervorgang der App kann ca. eine Minute dauern) und auch kleine Änderungen schnell testen zu können ist ein Testframework vorhanden. Dieses ermöglicht z. B. die Übergabe und anschließende Analyse von Videos des Teststreifen unter verschiedensten Bedingungen.
- Vorbedingungen:
  -
- Nebenbedingungen:
  Falls der zeitliche Rahmen es erlaubt existiert eine grafische Benutzeroberfläche, mit der das Testframework bedient werden kann.
- Nachbedingungen:
  Dem Testframework können Videos als Eingabeparameter übergeben und die Bildverarbeitungsalgorithmen auf diese Videos eingegeben werden. Die Ergebnisse der Bildverarbeitung werden übersichtlich dargestellt.
- Aufwand:
  Zwei Wochen
- Personen:
  - Sebastian Horwege
  - Nicolas Koch

**4.5.7.1 Dokumentation** In diesem Paket wurde ein Gerüst geschaffen, welches sich dazu eignet die Qualität des Bildverarbeitungsalgorithmus zu messen. Dabei soll es vor allem möglich sein, den Fortschritt der Algorithmusqualität über mehrere Versionen hinweg darzustellen. Zusätzlich dient das Framework als Möglichkeit den Algorithmus zu testen, ohne dass ein Smartphone benötigt wird.

**Ablauf** Durch die Anforderungen an das Test-Framework konnten die folgenden drei Kernaufgaben schlussgefolgert werden:

1. Sammeln von repräsentativen Testdaten (Benchmark).
2. Graphische Darstellung der erhobenen Daten.
3. Vergleich zweier verschiedener Datensätze[61]

[61] Verschiedene Algorithmusversionen des gleichen Benchmark-Videos

Die folgenden Absätze beschreiben, wie diese drei Aufgaben umgesetzt wurden. Dazu wurde als Hilfsmittel die Programmiersprache Python herangezogen.

**1. Sammeln von repräsentativen Testdaten (Benchmark)** Zum Sammeln von Testdaten wird das Programm `gather_data.py` zur Verfügung gestellt. Das Programm führt den Algorithmus mehrere Male auf einem Eingabevideo aus, damit Umgebungsfaktoren bei den Messwerten ausgeschlossen werden können.[62] Während der Ausführung von `gather_data.py` werden folgende Messwerte erhoben und als JSON Dokument exportiert:

- Pro Videodatei
  - Dateiname des Videos
  - Anzahl der Marker auf dem Teststreifen
- Pro Frame
  - Framezähler
  - Gemessene Zeit Markerdetektion
  - Gemessene Zeit Transformation
  - Gemessene Zeit Weißabgleich
  - Gemessene Zeit Intensitätsdetektierung
  - Gemessene Zeit gesamt
  - Ausgelesene Farbe des Farbumschlags
  - Anzahl der erkannten Marker

Die pro Bild erhobenen Daten werden über die wiederholten Durchläufe des Algorithmus gemittelt. Die durch diesen Prozess angefallenen Daten[63] können in darauffolgenden Schritten ausgewertet werden.

**2. Graphische Darstellung der erhobenen Daten** Die graphische Darstellung wird durch ein weiteres Programm – `plot_data.py` – umgesetzt. Dieses verwendet als Eingabe die in (1.) beschriebenen Daten und stellt diese in Form von Diagrammen dar. Abbildung 191 auf der nächsten Seite zeigt die Benutzeroberfläche von `plot_data.py`.

[62] z.B. durch Unterbrechnungsroutinen

[63] 200kB JSON Datei bei einem Video aus 700 Bilder

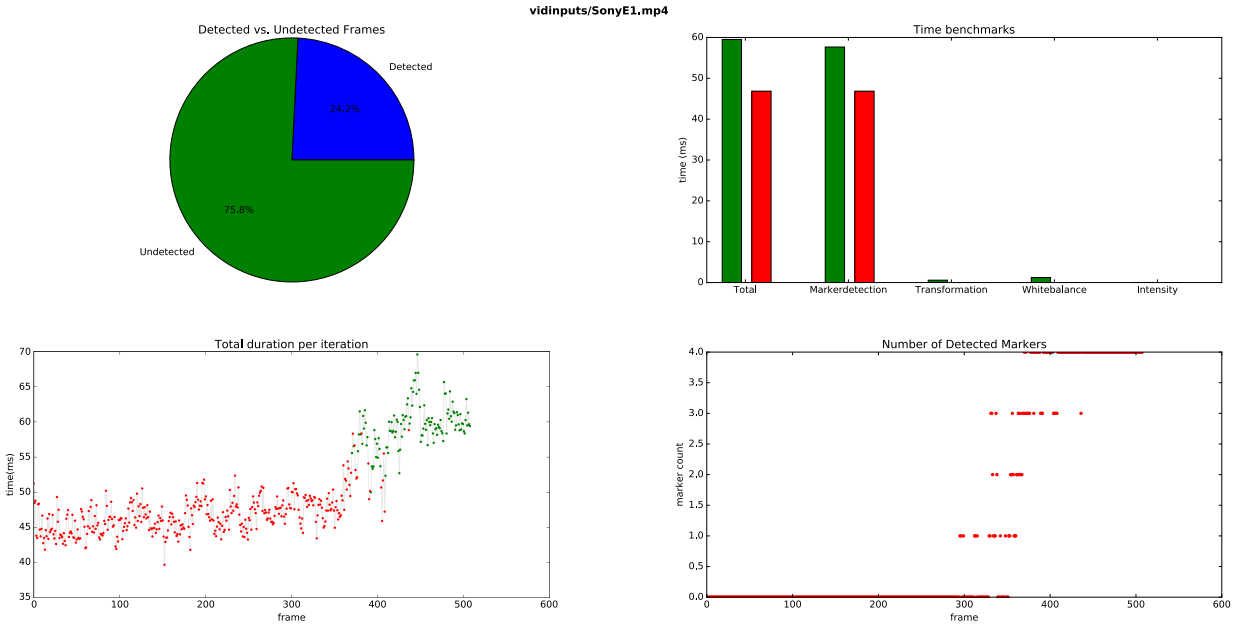



Abbildung 191: Benutzeroberfläche von plot_data.py

Der Graph oben links (detektierten vs. nicht detektierten Bildern) zeigt das Verhältnis zwischen erkannten und nicht erkannten Bildern über das gesamte Video. Erkannte Bilder sind jene, auf denen alle vier Marker erkannt wurden.
Der Graph oben rechts (Time Benchmarks) zeigt in der Abbildung 191, wie viel Zeit durchschnittlich für die einzelnen Teilaufgaben des Algorithmus aufgewendet werden muss.
Das Diagramm „Total duration per iteration“ zeigt die benötigte Gesamtdauer des Algorithmus pro Bild. Hierbei repräsentieren rote Punkte nicht erkannte Bilder und grüne Punkte erkannte Bilder. Um aus der Anwendung Rückschlüsse auf den referenzierten Frame ziehen zu können, kann dieser durch einen Klick auf einen Punkt des Diagramms angezeigt werden. Abbildung 192 zeigt, wie das ausgewählte Bild im Programm angezeigt wird.

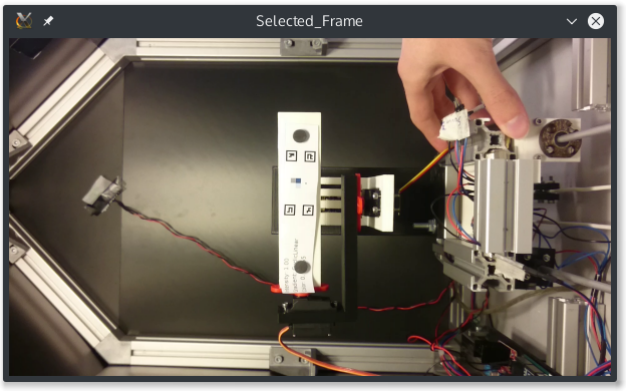



Abbildung 192: Anzeige des ausgewählten Frames

Das Diagramm „Number of Detected Markers“ befasst sich wiederum mit der Anzahl der erkannten Marker, stellt diese aber in einer pro-Bild-Basis dar. Auch hier kann durch Klicken auf das Diagramm das referenzierte Bild angezeigt werden.

**3. Vergleich zweier verschiedener Datensätze** Während `plot_data.py` verwendet werden kann, um einzelne Algorithmusversionen zu analysieren wird dafür ein Werkzeug benötigt um sie

zu vergleichen. Für diesen Zweck wurde das Programm `compare_data.py` entworfen. Mit diesem Werkzeug können zwei Algorithmusversionen verglichen werden, welche auf derselben Videodatei ausgeführt wurden. Abbildung 193 zeigt die Benutzeroberfläche von `compare_data.py`.

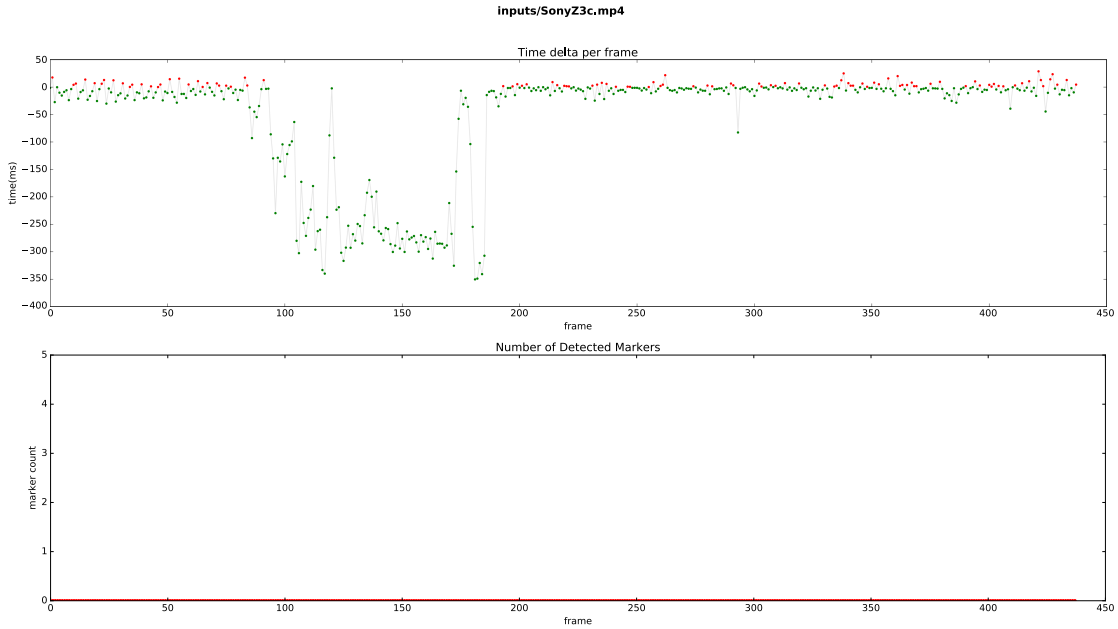



Abbildung 193: Benutzeroberfläche von compare_data.py

Die obere Abbildung zeigt die Zeitdifferenz der Algorithmen pro Bild. Ist der abgebildete Wert negativ bedeutet dies, dass die zweite Version des Algorithmus schneller war, als die erste.
Die untere Abbildung stellt die Differenz der erkannten Marker zwischen den Algorithmen dar. In der Abbildung erkennen die Algorithmen gleich viele Marker pro Bild. Es ist allerdings auch denkbar, dass bei einer Verbesserung des Algorithmus hier andere Werte resultieren.

**4. Automatisierung des Gesamtprozesses** Da es für den Nutzer umständlich ist, mit drei verschiedenen Programmen zu arbeiten, existiert ein weiteres Programm `tell_me_better.py`. Dieses kann genutzt werden, um anhand von zwei Versionen dem Nutzer auszugeben, ob sich der Algorithmus verbessert hat. Dazu kompiliert das Skript die verschiedenen Versionen in einem temporären Ordner und arbeitet dort mit den vorher genannten Programmen.
Dadurch kann der Entwickler ohne großen Aufwand ermitteln, ob seine Änderungen am Algorithmus eine Verbesserung zur folge hatte.

### Technologien

**Python** Programmiersprache, die sich besonders gut für Rapid Development[64] eignet.

**OpenCV** Bibliothek zur Bildverarbeitung [21].

**CMake** Werkzeug zum Bauen von C und C++ Projekten.

**Boost** Bibliothek zur Erstellung von Python Bindings in C++.

**Matplotlib** Python Bibliothek zur Anzeige von Graphen und Diagrammen [22].

[64]Programmiermodell, bei dem Prototyping ausgiebig verwendet wird.

**Git** Verteiltes Versionsverwaltungssystem, mit dem beliebige Versionen des Bildverarbeitungsalgorithmus abgerufen werden können.

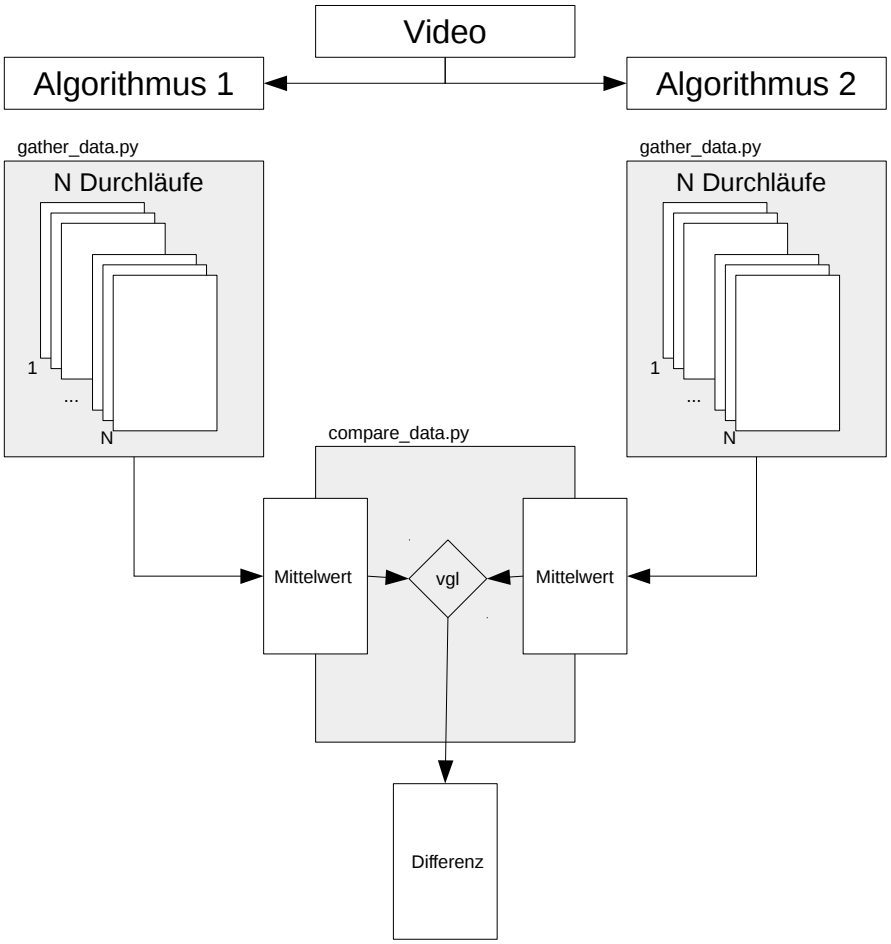



Abbildung 194: Erwarteter Ablauf zur Benutzung der entwickelten Tools

**Systembeschreibung** Das Testframework soll dabei helfen, den Bildverarbeitungsalgorithmus zu optimieren. Für die Integration des Testframeworks in die Entwicklung des Algorithmus ist folgendes Szenario vorstellbar: Um die Qualität des aktuellen Bildverarbeitungsalgorithmus zu messen, müssen zunächst Benchmarkdaten mithilfe von `gather_data.py` und einem Eingabevideo erhoben werden. Da hier auch die Dauer des Algorithmus gemessen wird, muss sich die Anzahl der Durchläufe auf dem Video entsprechend wiederholen. Dies dient dazu, Verfälschungen zu kompensieren, die durch die Prozessverwaltung des Betriebssystems unvermeidbar sind. Um problematische Bilder zu erkennen, können die Testdaten bereits mithilfe von `plot_data.py` analysiert werden.

Die erhobenen Daten können nun mit einer früheren Version des Bildverarbeitungsalgorithmus verglichen werden. Dazu muss dieser aus dem Git-Repository ausgecheckt und mithilfe des beschrieben Verfahrens weiter Daten erhoben werden. Der letzte Schritt ist dann der Vergleich der beiden Algorithmen. Dazu müssen sämtliche Messwerte in `compare_data.py` eingelesen und nach der Verarbeitung ausgewertet werden. Dieser Ablauf wird durch Abbildung 194 zusammengefasst. Das oben

beschriebene Verfahren kann mit dem Programm `tell_me_better.py` vollständig automatisiert werden.

**Parameter** Für jedes der Unterprogramme des Testframeworks gibt es eigene Parameter:

**gather_data.py**

- **`--video`** Video auf dem der Vergleichstest durchgeführt werden soll.
- **`--out`** Ausgabedatei in die die erhobenen Daten nach Ausführung des Programms gespeichert werden.
- **`--runs`** Anzahl der Durchläufe. Jeder Durchlauf wird in einer eigenen Ausgabedatei gespeichert. Die Durchläufe werden durch das Programm durchnummeriert. Jeder Durchlauf wird in einer eigenen Datei gespeichert. Dabei setzt sich die Name aus dem `--out` Parameter und der angehängten Durchlaufnummer zusammen.
- **`--desc`** Eingabedatei mit in den Eigenschaften des aufgenommenen Teststreifens aus `--video`. Die Beschreibung des Teststreifens wird bei der Generierung durch den Teststreifengenerator erzeugt. Hier wird beschrieben, welche Intensität und Farbe der simulierte Farbumschlag hat. Dieser Parameter ist optional und dient später dazu, die Korrektheit des Algorithmus im Sinne der Berechnung der Intensität des aufgenommenen Farbumschlags zu überprüfen.

**plot_data.py**

- **`--data`** 1 bis $N$ sind Eingabedateien in einen Vergleichstest zu einem Video und einem Algorithmus. Das Eingabeformat ist kompatibel zu dem Format der Ausgabedateien auf `gather_data.py`

**compare_data.py**

- **`--data1`** 1 bis $N$ sind Eingabedateien in einen Vergleichstest zu einem Video und einem Algorithmus.
- **`--data2`** 1 bis $N$ sind Eingabedateien in einen Vergleichstest, die sich von `--data1` nur durch den angewandten Bildverarbeitungsalgorithmus unterscheiden sollten.
- **`--outfile`** Ausgabe des Vergleichs von `--data1` und `--data2`. Das Format stimmt mit dem der Eingabedateien überein.
- **`--nogui`** Deaktiviert die grafische Ausgabe des Vergleichs.

**tell_me_better.py**

- **`--commit1`** Hash der Referenzversion aus dem Git-Repository.
- **`--commit2`** Hash der Version mit dem die Referenzversion verglichen werden soll. Aussagen des Programms über die Verbesserung des Algorithmus beziehen sich immer auf die hier angegebene Version.
- **`--video`** Das Eingabevideo, auf dem der Vergleich der Bildverarbeitungsalgorithmen aus `--commit1` und `--commit2` ausgeführt werden soll. Siehe auch `--video` in `gather_data.py`.
- **`--runs`** Anzahl der Durchläufe pro Algorithmus. Siehe auch `--runs` in `gather_data.py`.

**Fazit** Ursprünglich wurde das Testframework in C++ implementiert, um die Integration des Bildverarbeitungsalgorithmus möglichst einfach zu halten. Jedoch hat sich der nicht unerhebliche Aufwand für die Integration des Algorithmus durch Boost in Python mit Sicherheit gelohnt. In Python ist die Serialisierung und Deserialisierung von JSON und Python Objekten trivial. Außerdem wird durch `matplotlib` eine mächtige Datenvisualisierung ermöglicht. Der Zeitaufwand für die Einrichtung der Boost Bindings wurde dadurch schnell relativiert.

Um die Messdaten zu erheben mussten einige Funktionen des Bildverarbeitungsalgorithmus angepasst werden. Damit das Test-Framework zuverlässig funktioniert, müssen diese Änderungen in allen weiteren Versionen des Algorithmus beibehalten werden.

**Ausblick** Aufgrund der Eingriffe in den Bildverarbeitungsalgorithmus sind bis jetzt noch keine aussagekräftige Vergleiche möglich. Diese können aber ab sofort für jede neue Version des Algorithmus durchgeführt werden, um so die stetige Verbesserung der Bildverarbeitung zu dokumentieren. Einige geplante Funktionen wurden noch nicht im Bildverarbeitungsalgorithmus implementiert. Das Test-Framework muss stetig angepasst werden, um eine immer detailliertere Aussage über die Qualität des Algorithmus treffen zu können. Dafür wurde das Testframework soweit modularisiert, dass das Hinzufügen von weiteren Funktionen auch in Zukunft kein Problem ist.

Durch `tell_me_better.py` wurde ein Werkzeug bereitgestellt, das nicht nur einfach zu interpretierende Aussagen über die zu vergleichenden Algorithmen trifft, sondern gleichzeitig auch leicht automatisierbar ist. Es wäre zum Beispiel eine Integration in den bestehenden Jenkinsserver vorstellbar, um für jede neue Version des Algorithmus eine automatisierte Aussage über die Verbesserung der Qualität treffen zu können.

### 4.5.8 Fazit des Sprints

Wie bereits in der Einleitung beschrieben, lag der Fokus von Sprint 6 auf der Verbesserung des Bildverarbeitungsalgorithmus. Durch den erfolgreichen Abschluss der Arbeitspakete 4.5.3.1 auf Seite 224 „Anpassung der Bildalgorithmen auf dynamische Teststreifen“ , 4.5.4.1 auf Seite 241 „Anpassung der Bildalgorithmen auf dynamische Teststreifen“ und 4.5.5.1 auf Seite 245 „Weiterentwicklung der Bildverarbeitung“ konnte der Algorithmus sowohl in seiner Geschwindigkeit verbessert, als auch um die Unterstützung dynamischer Teststreifen erweitert werden. Auf Basis des Arbeitspakets 4.5.7.1 auf Seite 259 „Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen“ ließen sich diese Verbesserungen auch messen: Es ist nun möglich zwei Versionen des Bildverarbeitungsalgorithmus miteinander in den Punkten Performanz und Zuverlässigkeit zu vergleichen. Diese Funktion ermöglicht, zusammen mit dem aus Sprint 5 vorhandenen Teststand (siehe dazu Kapitel 4.4.14.1 auf Seite 192), eine optimale Testumgebung für zukünftige Implementierungen des Algorithmus. Zusätzlich zu dieser Testumgebung wurde auch der Teststreifengenerator im Rahmen des Arbeitspakets „Anpassung des Teststreifengenerators für ein dynamisches Design und unterschiedliche Teststreifenvarianten“ um die Unterstützung dynamischer Teststreifen erweitert.
Neben der Verbesserung des Bildverarbeitungsalgorithmus war ein weiterer Aspekt von Sprint 6 die erste Implementierung des Servers in Go. Im Zuge des sprintübergreifenden Arbeitspakets „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ ist zum jetzigen Zeitpunkt eine erste Version des Servers implementiert, die bereits die Funktionalität umsetzt, XML-Dateien mit Teststreifendaten über eine REST-Schnittstelle zu Verfügung zu stellen.
Zusammenfassend lässt sich die Systemlandschaft zum Ende von Sprint 6 wie folgt darstellen:

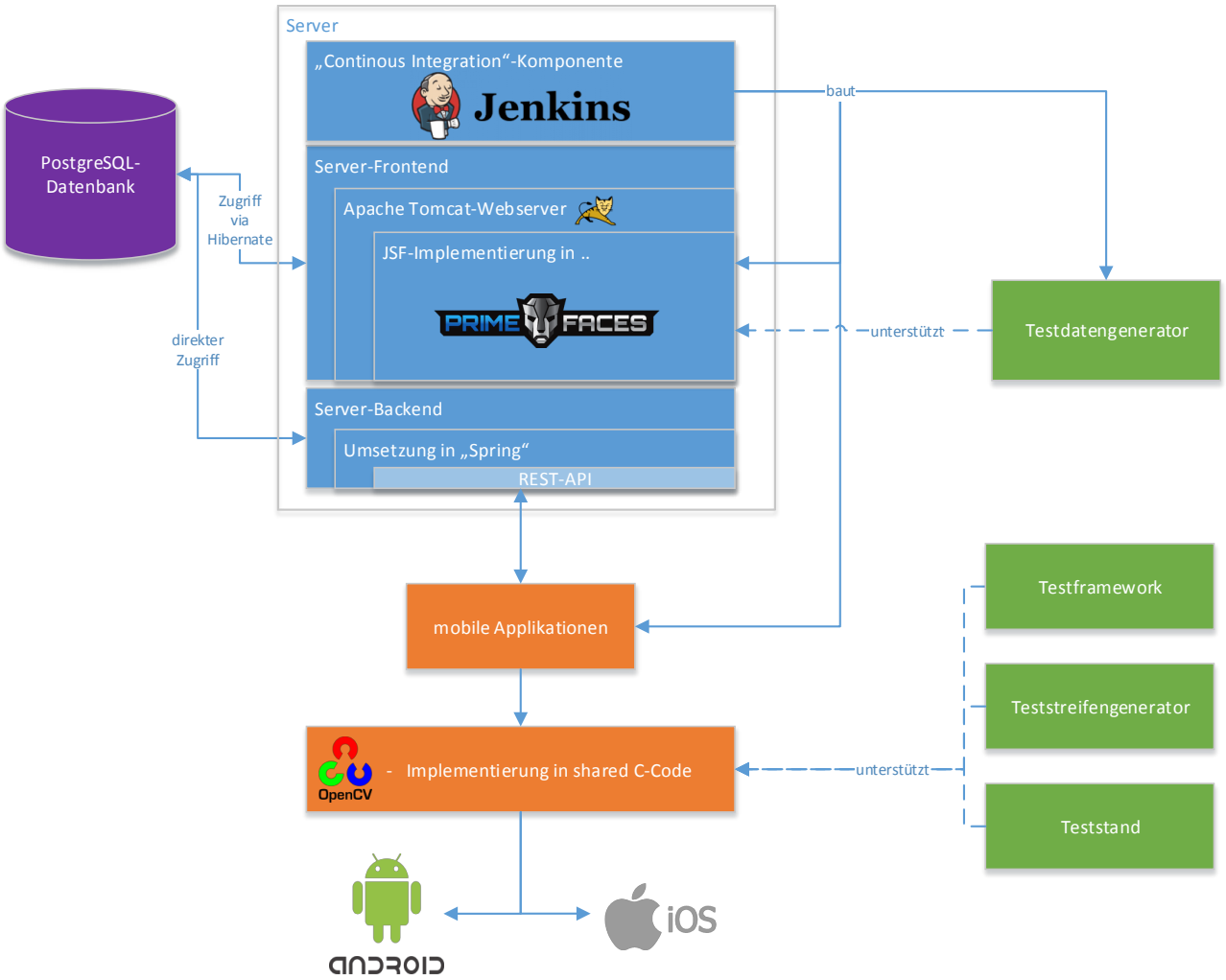



Abbildung 195: Systemarchitektur am Ende von Sprint 6

Wie auf Abbildung 195 zu sehen ist, wurde die Systemlandschaft um das in der Einleitung beschriebene Test-Framework für den Bildverarbeitungsalgorithmus erweitert. Zusammen mit dem erweiterten Teststreifengenerator und dem aus Sprint 5 übernommenen Teststand steht somit eine optimale Evaluierungsumgebung für zukünftige Implementierungen des Algorithmus zur Verfügung. Da die Implementierung des Servers in Go zu diesem Zeitpunkt noch nicht finalisiert ist, wurde die abgebildete Systemlandschaft dahingehend noch nicht erweitert.
Wie schon in den vorherigen Sprints wurde auch in Sprint 6 wieder eine Fortschrittsanalyse über die prozentuale Anzahl aller erfüllten funktionalen Anforderungen des Lastenhefts durchgeführt.

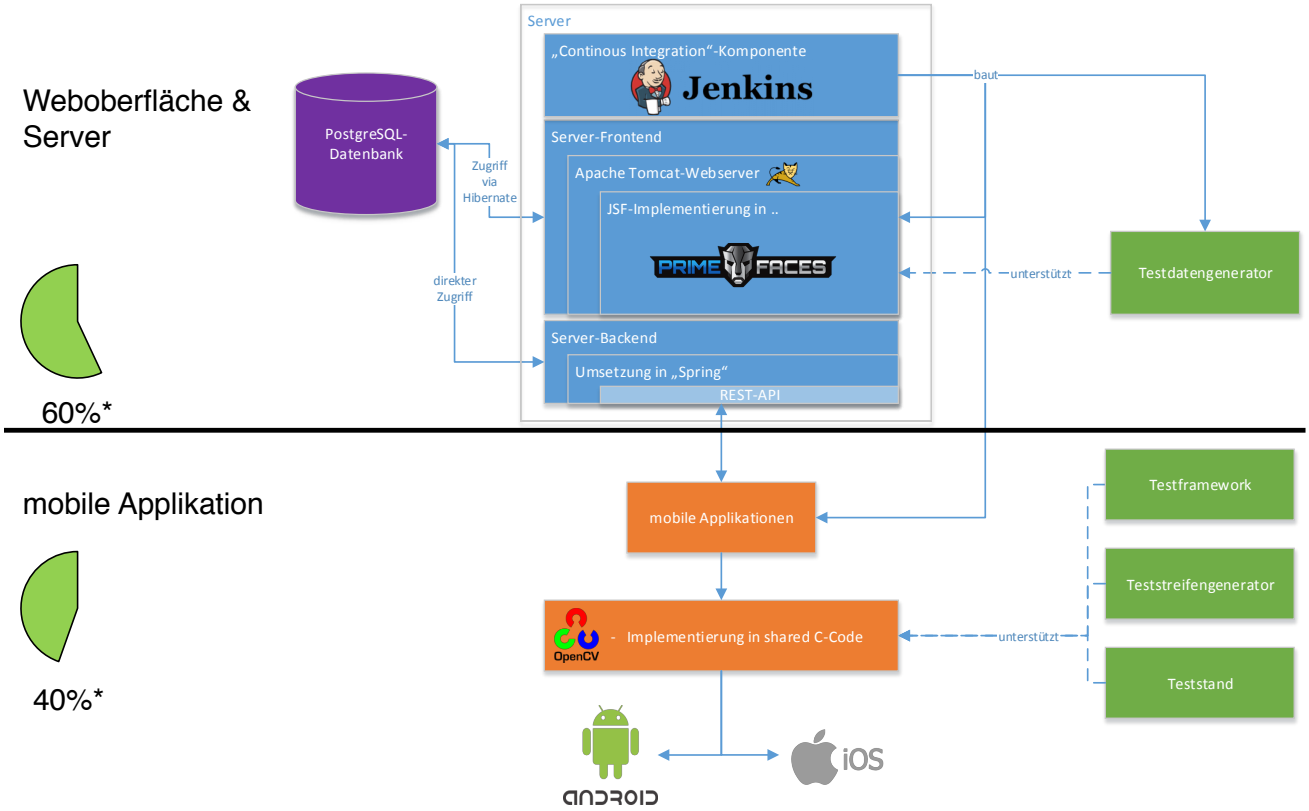



Abbildung 196: Fortschrittsanalyse nach dem Abschluss von Sprint 6

In Abbildung 196 ist zu sehen, dass kein weiterer Fortschritt bei den funktionalen Anforderungen der Weboberfläche und des Servers zu verzeichnen ist (immer noch 60%). Die Gründe hierfür sind zum einen, dass der Fokus in diesem Sprint auf der Verbesserung der Bildverarbeitung lag und zum anderen, dass das oben genannte sprintübergreifende Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ noch nicht abgeschlossen ist und daher auch noch nicht in den Fortschritt der aktuellen Systemlandschaft aufgenommen wurde.
Im Bereich der mobilen Applikationen und des Bildverarbeitungsalgorithmus konnte eine Erhöhung von 10% auf insgesamt 40% umgesetzter funktionaler Anforderungen erreicht werden. Diese Zahl wirkt zunächst nicht nach einer enormen Erweiterung der mobilen Applikationen. Dies täuscht allerdings, da die Geschwindigkeit und Nutzerfreundlichkeit der mobilen Applikationen größtenteils die nichtfunktionalen Anforderungen des Lastenhefts widerspiegeln. Als neue Funktionalität wurde zudem die Unterstützung dynamischer Teststreifen umgesetzt.
Es stellte sich heraus, dass eine Sprintdauer von lediglich vier Wochen problematisch zu betrachten ist. Innerhalb dieser verkürzten Zeit mussten über 25% der Arbeitskapazitäten alleine für Dokumentation und Verwaltungsaufgaben verplant werden. Infolgedessen wird in Zukunft wieder eine Sprintdauer von sechs Wochen angesetzt.

## 4.6 Sprint 7

Im Gegensatz zu Sprint 6, bei dem der Fokus auf der Optimierung der Bildverarbeitung lag, wurde in Sprint 7 das gesamte Projekt weiterentwickelt. Das Ziel ist die Integration der dynamischen Teststreifen in die Apps und der Weboberfläche, sowie die Umsetzung von funktionalen Anforderungen in den Apps. So kann der prozentuale Fortschritt der Anforderungen merklich verbessert und zugleich mit dem Go-Server die Grundlage für eine bessere Performance der Server-Architektur gelegt werden.
In Abbildung 197 ist die Systemlandschaft und ein Überblick über den Fortschritt zu Beginn des Sprints dargestellt.

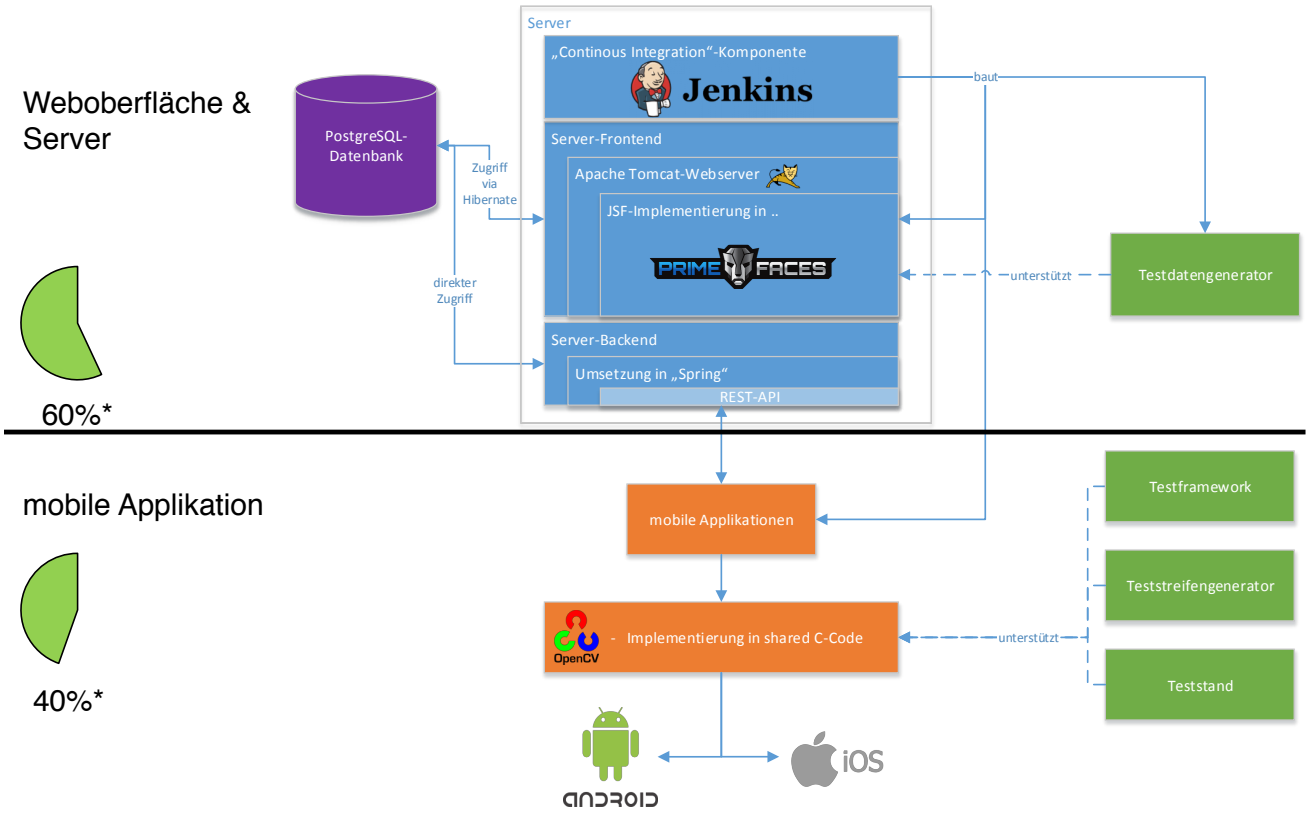



Abbildung 197: Systemarchitektur zu Beginn von Sprint 6

In der Abbildung sieht man auch, dass insgesamt 60% der funktionalen Anforderungen auf der Weboberfläche und dem Server und 40% der funktionalen Anforderungen in den Apps umgesetzt wurde. Zwar wurde der Go-Server im letzten Sprint bereits angefangen, enthielt aber bisher nur rudimentäre Funktionen. Deshalb ist in der Systemlandschaft noch der Spring-Server angegeben. Diese Funktionalität wird in Sprint 7 erweitert, sodass der bisher verwendete Spring-Server vollständig ersetzt werden kann.
In diesem Sprint werden zusätzliche Anforderungen in den Apps umgesetzt, um so den prozentualen Fortschritt zu verbessern. Dies sind unter anderem die Abfrage des Akkustandes, eine bessere Nutzerführung sowie die Abfrage der Rechte, die die App benötigt. Eine genaue Auflistung aller Anforderungen ist in dem Kapitel „Refactoring der iOS App und Integration von weiteren Anforderungen an die iOS App" (Kapitel 4.6.8.1 auf Seite 284).
Zusätzlich dazu wird die in Sprint 6 entwickelte dynamische Teststreifenerkennung integriert. Dies geschieht auf der Seite der Apps, indem die Daten von dem Go-Server geladen, auf dem Smartphone zwischengespeichert und für die Bildverarbeitung bereitgestellt werden. Eine genaue Beschreibung dazu ist in Kapitel 4.6.9.1 auf Seite 290 „Integration der dynamischen Bildverarbeitung in die Apps" zu finden.

Auf der Seite der Weboberfläche wird die Verwaltung der Teststreifenvarianten implementiert. Es wird möglich sein, dass ein Administrator einen eigenen Teststreifen anlegen kann. Als Grundlage dafür wird der Teststreifengenerator genommen, der bereits in früheren Sprints entwickelt wurde. Außerdem wird in diesem Arbeitspaket eine Überarbeitung der bestehenden Datenbank vorgenommen, um die Struktur zu optimieren und Tabellen zu entfernen, die nicht mehr benötigt werden. So soll eine optimale Performance des Servers und der Weboberfläche sichergestellt werden. Eine genaue Beschreibung ist in dem Kapitel „Teststreifenverwaltung in Webfrontend und Refactoring der Datenbank“ (Kapitel 4.6.10.1 auf Seite 295).
Nach Abschluss aller Arbeitspakete ist durch den finalisierten Go-Server der in Sprint 4 begonnene Technologiewechsel abgeschlossen. Darüber hinaus ist zu erwarten, dass durch die Erweiterung des Webfrontends und der mobilen Applikationen der prozentuale Fortschritt des Projekts stark verbessert wird.

### 4.6.1 Sprintplanung

Im Folgenden wird die Planung des siebten Sprints (23.11.2015 bis 16.12.2015, vier Wochen) genauer aufgezeigt. Dabei wird ein besonderes Augenmerk auf die folgenden Punkte gelegt:

- Fokus des Sprints
- Arbeitspakete und Planung
- Planung der einzelnen Ressourcen

### 4.6.2 Fokus des Sprints

Im Fokus des letzten Sprints für 2015 standen unterschiedliche Aufgabenschwerpunkte. Das große Ziel war es jedoch, viele Kundenanforderungen an die Produkte umzusetzen, um den Fortschrittsgrad des Projektes zum Jahresende noch einmal positiv zu beeinflussen. Zudem kann so der Fokus im Jahr 2016 darauf gelegt werden, die letzten Verbesserungen in das Projekt einzuarbeiten und es erfolgreich abzuschließen.

Unter anderem sollte im letzten Sprint der dynamische Bildverarbeitungsalgorithmus in beide Apps integriert werden. Die iOS App sollte neu gestaltet werden und mit weiteren Funktionen laut Anforderungen versehen werden. Zudem sollten die letzten, noch benötigten Funktionen für den Go Server und die Weboberfläche entwickelt werden.

### 4.6.3 Arbeitspakete und Planung

Aus dem Fokus des Sprints und der daraus resultierenden Planung ergaben sich die unten stehenden Arbeitspakete.

- Integration weiterer benötigter Funktionen für den Go-Server (Priorität: normal - zwei Wochen à zwei Personen)
- Integration weiterer benötigter Funktionen und Sicherheitszertifikate in den Go-Server (Priorität: normal - zwei Wochen à zwei Personen)
- Integration des dynamischen Bilderkennungsalgorithmus in die Apps (Priorität: normal - vier Wochen à zwei Personen)
- Refactoring der iOS App und Integration weiterer Funktionen laut Anforderungen (Priorität: normal - vier Wochen à zwei Personen)

- Datenbank-Refactoring, Erweiterung des Web-Frontend und Teststreifenvarianten über Web-Frontend freischalten (Priorität: normal - vier Wochen à zwei Personen)
- Integration weiterer Funktionen in die Android App laut Anforderungen (Priorität: optional - eine Wochen à zwei Personen)

Der folgenden Abbildung 198 können die geplanten Arbeitspakete entnommen werden. Es gab keinen direkten kritischen Pfad, da alle Arbeitspakete unabhängig voneinander erfüllt werden konnten. Die Arbeitspakete mussten lediglich direkt zu Beginn des Sprints begonnen werden, da diese – wie der Sprint – vier Wochen lang sind. Die Grafik beinhaltet nur Arbeitspakete normaler Priorität. Optionale Arbeitspakete wurden aus Platzgründen nicht mit aufgenommen.

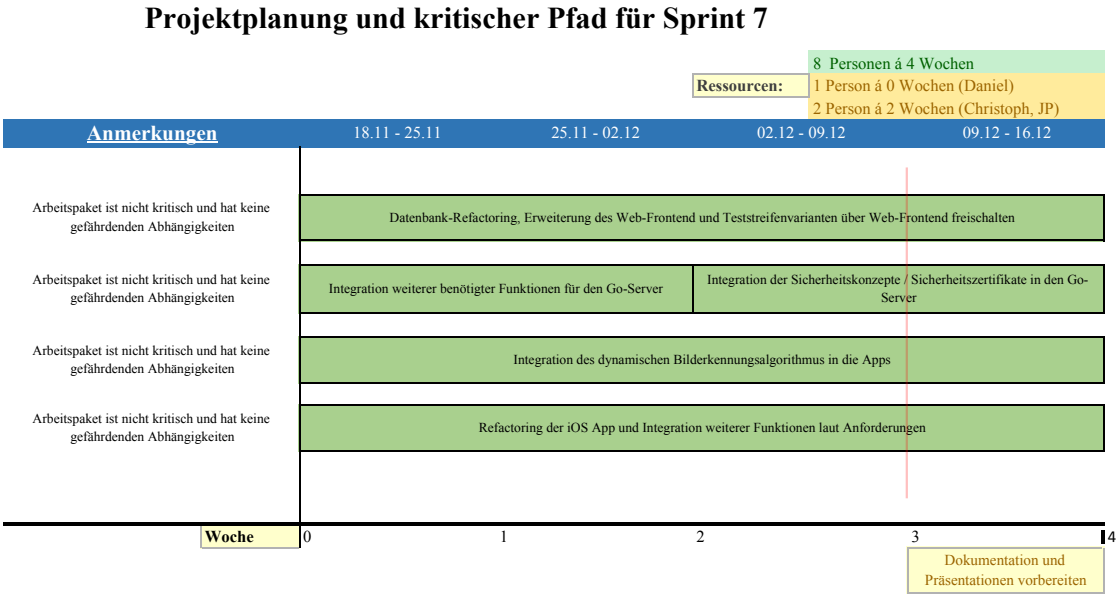



Abbildung 198: Planung und kritischer Pfad für Sprint 7

### 4.6.4 Planung der Ressourcen

Im Folgenden ist die Personalplanung dargestellt. Sie gibt Auskunft darüber, welches Teammitglied an welchem Arbeitspaket arbeitet.

- Raphael Kappes: vier Wochen - Integration des dynamischen Bilderkennungsalgorithmus in die Apps
- Christian Sandmann: vier Wochen - Integration des dynamischen Bilderkennungsalgorithmus in die Apps
- Timo Schlömer: vier Wochen - Refactoring der iOS App und Integration weiterer Funktionen laut Anforderungen
- Timo Raß: vier Wochen - Refactoring der iOS App und Integration weiterer Funktionen laut Anforderungen
- Nicolas Koch: vier Wochen - Datenbank-Refactoring, Erweiterung des Web-Frontend und Teststreifenvarianten über Web-Frontend freischalten

- Kevin Sandermann: vier Wochen - Datenbank-Refactoring, Erweiterung des Web-Frontend und Teststreifenvarianten über Web-Frontend freischalten
- Sebastian Horwege: alle Go-Server Arbeitspakete, zudem zusätzlich weiter am Test Framework arbeiten
- Danny Fonk: alle Go-Server Arbeitspakete
- Jan Philipp Stubbe: zwei Wochen am Go-Server, danach Urlaub
- Christoph Ressel: zwei Wochen - Unterstützung bei der Integration der dynamischen Bildverarbeitungsalgorithmen
- Daniel Wegmann: Seminarphase

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und dessen Dokumentation skizziert.

### 4.6.5 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete wie gewohnt detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.6.6 Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server

- Priorität: Normal - Dieses Arbeitspaket geht von Sprint sechs in Sprint sieben um eine Woche mit ein.
- Motivation und Nutzen des Arbeitspaketes: Es sollen aktuelle Technologien (Go) verwendet werden, um z. B. einer hohen Nutzeranzahl standzuhalten und somit weitere Anforderungen erfüllen zu können.
- Arbeitspaketbeschreibung:
  Am Ende des Arbeitspaketes existiert ein Go-Server, welcher zunächst nur die Funktion enthalten muss die XML-Dateien, welche die verschiedenen Teststreifenvarianten / Designs beschreiben, auszuliefern. Zudem soll der Go-Server so gestaltet werden, dass er in zukünftigen Arbeitspaketen um zusätzliche Funktionen erweitert werden kann. Neben der Dokumentation wurde zudem ein Wiki-Artikel verfasst, welcher die Basis-Funktionalitäten des Go-Servers und wie mit diesem zu Arbeiten ist kurz beschreibt.
- Vorbedingungen:
  Das Format der XML-Dateien für die Teststreifenvarianten muss vorhanden sein. Es müssen auslieferbare XML-Dateien vorhanden sein.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Es existiert eine Schnittstelle, welche von den Apps angesprochen werden kann, sodass die XML-Dateien ausgeliefert werden können.

- Aufwand:
  Drei Wochen
- Personen:
  - Jan Philipp Stubbe
  - Danny Fonk

**4.6.6.1 Dokumentation** Bei einer Gegenüberstellung bzgl. verschiedener Server-Backends im Zuge einer Seminararbeit wurde Go als Lösungsansatz beschrieben. Auf Grund dessen ist die Implementierung der REST-Schnittstelle, welche zur Kommunikation mittels Server und mobilen Endgeräten genutzt wird und vorher in Spring umgesetzt war, in Go erfolgt.

**Ablauf** Nach einer angemessenen Einarbeitungszeit wurde ein rudimentärer Webserver in Go implementiert, welcher als Basis genutzt wurde, um eine REST-Schnittstelle zu realisieren. Die REST-Schnittstelle liefert bei korrekter Benutzung stets ein JSON-Dokument. Bei unsachgemäßer Handhabung (bspw. falscher Methodenname) greifen die standardisierten HTTP-Statuscodes (z.B. 404 falls die angegebene Ressource [bspw. Methode] nicht gefunden wurde). Eine Auflistung möglicher HTTP-Fehlercodes ist auf wikipedia.org[65] zu finden. Bei einer eingehenden Anfrage stellt der Webserver eine Verbindung zur bestehenden Datenbankinstanz her und liest ein Teststreifenschema passend zur übergebenen ID aus. Der ausgelesene Datensatz wird anschließend in ein JSON-Objekt überführt und zurückgeliefert. Bei der Verwendung der Datenbankverbindung wurden Erkenntnisse, welche in der Seminararbeit von Kevin Sandermann erarbeitet worden sind, berücksichtigt, sodass eine missbräuchliche Nutzung der Schnittstelle nicht möglich ist und der Server eine Konfiguration unter Verwendung eines TLS-Protokolls[66] zulässt. Weiterhin wurde eine Fehlerbehandlung implementiert. Diese Fehlerbehandlung sieht vor, dass auftretende Fehler fortlaufend in eine Datei geschrieben werden, sodass eine Auswertung bei schwerwiegenden Fehlern möglich ist.

Die Struktur des JSON-Dokuments bei sachgemäßer Handhabung ist nachfolgend beschrieben:

**REST-Schnittstelle für Teststreifenvarianten**

Der Aufruf der Adresse

`http://[...]/getschema?id=SCHEMAID`

erzeugt bei einem Erfolg folgende Ausgabe:

```
{"Status":"true",
 "Schema": {"id":1,
            "name":"Grippe-Teststreifen",
            "data":"data",
            "created":"2015-11-13T00:00:00Z"
            "active": true}
}
```

Listing 4: Antwort vom Server, wenn Teststreifenschema abgefragt wurde

Dabei ist „data“ ein String und repräsentiert ein weiteres JSON-Objekt, welches die Teststreifenvariante beschreibt. Ein Beispiel für ein solches Objekt kann im Anhang unter 11 auf Seite 276 gefunden werden.

Wenn kein Teststreifen-Schema zu der übergebenen ID gefunden werden konnte, gibt der Server folgende Fehlermitteilung zurück:

[65] https://de.wikipedia.org/wiki/HTTP-Statuscode

[66] https://de.wikipedia.org/wiki/Transport_Layer_Security

```
{"Status":"false",
 "Message":"No schema was found"
}
```

Listing 5: Antwort vom Server, wenn Schema nicht vorhanden

Der Server antwortet auf eine Anfrage mit ungültigen Parametern folgendes:

```
{"Status":"false",
 "Message":"Teststripid has to be an integer number"
}
```

Listing 6: Antwort vom Server bei ungültigem Parameter

Wenn ein serverseitiger Fehler während der Anfrage aufgetreten ist, antwortet der Server wie folgt:

```
{"Status":"false",
 "Message":"An internal server-error occured"
}
```

Listing 7: Antwort vom Server, wenn Serverfehler aufgetreten

**REST-Schnittstelle als Ersatz für Spring-Server**

Die relevanten Schnittstellen des Spring-Servers wurden in den Go-Server übernommen und bei Notwendigkeit angepasst.

Zum Speichern eines neuen Testergebnisses auf dem Server, muss eine Anfrage an folgende Adresse geschickt werden.

```
http://[...]/selftests/create
```

Der Inhalt dieser Anfrage muss alle nötigen Informationen enthalten:

```
{"appresult" : "+",
 "location" : { "lat" : 51, "lon": 8},
 "image" : "YnZrZmc=",
 "tokenid" : "68e59eea-b416-4b11-4f40-c725adf82315"
}
```

Listing 8: Zu sendendes JSON-Dokument des Smartphones

Dabei enthält der Eintrag `appresult` das berechnete Ergebnis auf dem aufgenommenen Teststreifen aus der mobilen Anwendung. Die `location` beinhaltet die Längen- und Breitengrade des Benutzers, die durch die Anwendung ermittelt wurden. Das `image` enthält kodierte Bildinformationen zum aufgenommenen Teststreifen. Fehlt die `tokenid`, wird ein neuer Token zur Identifikation des Benutzers vom Go-Server erzeugt. Die Antwort des Servers auf diese Anfrage wird dann wie folgt aussehen:

```
{"tokenid":"68e59eea-b416-4b11-4f40-c725adf82315",
 "testrecordid":"82574"}
```

Listing 9: Antwort vom Server, wenn Testergebnis erzeugt wurde

Daten zu abgesendeten Testergebnissen können unter Angabe des Benutzertokens und der jeweiligen Teststreifen Identifikationsnummer unter folgender Adresse abgerufen werden:

```
http://[...]/selftests/get?includeImage=false
                          &token=TOKENID&id=TESTSTRIPID
```

Die Antwort auf diese Anfrage bei einem existierenden Datensatz sieht dann folgendermaßen aus:

```
{"id": 8000,
 "lat": 49.466,
 "lon": 8.329,
 "appresult":"+",
 "created":14357376000000,
 "diagnose_reviewer_id":-1,
 "diagnose_text":null,
 "diagnose_result": "0"}
```

Listing 10: Antwort vom Server, wenn Testergebnis abgefragt wurde

Die Antwort enthält die selben Informationen, wie der vorher von der mobilen Anwendung gesendete Datensatz. Zusätzlich werden noch weitere Informationen mitgesendet, die sich auf die vom Arzt gestellte Diagnose beziehen. Ist eine Diagnose bereits vom Arzt vorgenommen wurden, findet diese sich in `diagnose_reviewer_text` wieder. Für fragwürdige Auswertungen durch die mobile Anwendung hat der Arzt auch die Möglichkeit, die Auswertung zu korrigieren. In diesem Fall wird das neue Ergebnis in `diagnose_result` zurückgeliefert.

Das Arbeitspaket wurde während der Laufzeit um eine weitere Woche auf insgesamt vier Wochen verlängert, sodass der Programmierfluss genutzt werden konnte und sämtliche Funktionalitäten des bestehenden Spring-Servers in den Go-Server integriert werden konnten. Dazu wurde eine weitere Person (Sebastian Horwege) mit in das Arbeitspaket als Quereinsteiger aufgenommen. Die Einarbeitungszeit konnte durch eine eigenständige Einarbeitung in Go gering gehalten werden, sodass mit der Implementierung der Funktionen zeitnah begonnen werden konnte. Dadurch konnten alle Serverfunktionen bereits in diesem Arbeitspaket umgesetzt werden. Ein weiteres Arbeitspaket ist deshalb also nicht mehr nötig.

**Anpassung des Teststreifengenerators** Im Zuge der Implementierung des Go-Servers ist aufgefallen, dass eine Speicherung der Beschreibung von Teststreifenvarianten als XML ungünstig ist, da sämtliche Kommunikation mit dem Server in JSON stattfindet. Auf Grund dessen wurde der Teststreifengenerator angepasst, sodass dieser die Teststreifen als JSON speichert und auch einliest.

### Technologien

**Go** *„Go ist eine kompilierbare Programmiersprache, die Nebenläufigkeit unterstützt und über eine automatische Speicherbereinigung verfügt. Entwickelt wurde Go von Mitarbeitern des Unternehmens Google Inc.“* Quelle: `https://de.wikipedia.org/wiki/Go_(Programmiersprache)`

**JSON** JSON (JavaScript Object Notation) ist ein kompaktes Datenformat und wird im Kontext dieses Arbeitspakets zum Datenaustausch zwischen zwei Endgeräten (Server und mobiles Endgerät) genutzt.

**Systembeschreibung** Der Go-Server wurde in Eclipse mit dem Plugin **GoClipse**[67] entwickelt. Weiterhin ist der Go-Server in den bereits laufenden Debian-Server integriert worden und läuft dort als Service.

[67] `https://github.com/GoClipse/goclipse/`

**Evaluation** Alle Funktionen wurden lokal und nach der Installation auf dem Debian-Server, welcher vorher als Server für die Spring-Komponente galt, getestet. Ein Test bzgl. der Performanz ist noch ausstehend und konnte auf Grund von mangelnder Ressourcen nicht durchgeführt werden. Frühere Tests einer Seminararbeit zeigten jedoch, dass der Go-Server bei gleicher Funktionalität dem Spring-Server im Hinblick auf Performanz überlegen ist.

**Parameter** Die Ergebnisse des Arbeitspaketes werden von keinen Parametern eingeschränkt.

**Fazit** Das Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ konnte innerhalb von vier Wochen abgeschlossen werden. Der dabei entstandene Webserver ersetzt zukünftig den bisher vorhandenen Spring-Server und steht den mobilen Anwendungen als Schnittstelle bereit.

**Ausblick** Auch wenn durch die Umstellung von Spring auf Go schon einige Verbesserungen in der Schnittstelle zwischen Server und mobiler Anwendung vorgenommen wurden, gibt es doch noch einige offene Fragen bezüglich der Kommunikation, die beantwortet werden müssen. Beispielsweise sollte das Übertragen der Bildinformationen nicht als **Base64** kodierter Text im **JSON** Dokument mit übergeben werden. Es wäre denkbar, die REST Schnittstelle so zu verändern, dass Binärdaten separat übertragen werden. Zusätzlich ist zu diesem Zeitpunkt nicht definiert, wie die Routine zur Beschaffung neuer Teststreifenschemata in den mobilen Applikationen ausgeführt wird.

```
{
    "teststrip":
            {
                "color": {
                    "r": 0,
                    "b": 255,
                    "g": 0
                },
                "colorchanges": {
                    "colorchange": [
                        {
                            "posX": 900,
                            "intensity": 0.25,
                            "posY": 175,
                            "gradient": "GradientCyclicLinear",
                            "corner-size": 0,
                            "angle": 0
                        },
                        {
                            "posX": 950,
                            "intensity": 0.75,
                            "posY": 175,
                            "gradient": "GradientCyclicLinear",
                            "corner-size": 0,
                            "angle": 0
                        },
                        {
                            "posX": 925,
                            "intensity": 1,
                            "posY": 150,
                            "gradient": "GradientCyclicLinear",
                            "corner-size": 0,
                            "angle": 0
                        }]
                },
                "extrainformation": {
                    "posX": 0,
                    "posY": 0,
                    "width": 0,
                    "type": "QR_CODE",
                    "value": "init",
                    "height": 0
                },
                "MiscOptions": {
                    "backgroundFile": "",
                    "drawText": true,
                    "dpi": 300,
                    "colorType": 1
```

```
                },
                "markers": {
                    "marker": [
                        {
                            "posX": 760,
                            "posY": 0,
                            "size": 56,
                            "value": 2000
                        },
                        {
                            "posX": 1150,
                            "posY": 100,
                            "size": 64,
                            "value": 4000
                        },
                        {
                            "posX": 1250,
                            "posY": 140,
                            "size": 64,
                            "value": 6000
                        }]
                }
            }
}
```

Listing 11: Beschreibung einer Teststreifenvariante

### 4.6.7 Finalisierung des GO Servers

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Die grundsätzliche Motivation für den Go-Server wurde schon ausgiebig im Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server “(Abschnitt 4.6.6 auf Seite 271) behandelt. In diesem Arbeitspaket soll die Funktionalität des Go-Servers nach den neusten Anforderungen an die Schnittstelle zu den mobilen Endgeräten angepasst werden.
- Arbeitspaketbeschreibung:
  Um Bandbreite zwischen dem Server und den mobilen Endgeräten zu minimieren, muss die Schnittstelle überarbeitet werden, über die beide kommunizieren. Die Übertragung der Bilddaten als kodiertes Textfeld ist nicht effizient. Stattdessen sollte das Bild unkodiert, binär übertragen werden.

  Das selbst unterschriebene Secure Socket Layer (SSL)-Zertifikat muss durch ein Zertifikat einer offiziellen Zertifizierungsstelle ersetzt werden. Selbst unterschriebenen Zertifikaten wird im allgemeinen nicht vertraut, weshalb diese nicht auf mobilen Endgeräten mit Android oder iOS funktionieren.

  Den mobilen Endgeräten müssen alle Teststreifenvarianten bekannt sein, die zum aktuellen Zeitpunkt im Umlauf sind. Aus diesem Grund müssen die mobilen Endgräte ein Teststreifenschema zu jedem Teststreifen lokal speichern. Es soll daher eine Routine entworfen und umgesetzt werden, die die Synchronisation zwischen mobilem Endgerät und Server erlaubt.

Die Details zu den Teststreifenschemata können aus dem Arbeitspaket „Anpassung des Teststreifengenerators für ein dynamisches Design und unterschiedliche Teststreifenvarianten“ aus Sprint 6 entnommen werden.

- Vorbedingungen:
  - Das Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ (Abschnitt 4.6.6 auf Seite 271) ist beendet
  - Das Arbeitspaket „Web-Frontend weiterentwickeln, Teststreifen verwalten und Datenbank-Refactoring“ (Abschnitt 4.6.10.1 auf Seite 295) ist soweit fortgeschritten, dass das Datenbankschema nicht mehr geändert wird
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der bestehende Go-Server ist um die oben genannten Funktionen erweitert und ist über den Server der Projektgruppe erreichbar.
- Aufwand:
  Zwei Wochen
- Personen:
  - Danny Fonk
  - Sebastian Horwege

**4.6.7.1 Dokumentation** Als Ergänzung zum Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“, entstand das Arbeitspaket „Finalisierung des Go-Servers“. Während der Entwicklung des Go-Servers sind neue Anforderungen an die mobilen Applikationen entstanden, welche eine Anpassung des bisherigen Codes erforderten.

**Ablauf** Die Finalisierung des Go-Servers verlief in enger Zusammenarbeit mit Arbeitsgruppen anderer Pakete. Hierbei wurden sämtliche Schnittstellen abermals auf ihre Sinnhaftigkeit überprüft und bei Bedarf neu implementiert sowie ggf. neu definiert. Des Weiteren wurde die Methode zum Anlegen eines Datensatzes überarbeitet. Bisher wurde zur erleichterten Übermittlung das Bild auf Seite der mobilen Applikation in Base64 umgewandelt. Dies hatte den Nachteil, dass das Bild bei der Speicherung um den Faktor 1,3 vergrößert wurde. Eine Kompression der Bilddaten ist in Anbetracht des Kompressionsergebnisses im Verhältnis zum Rechenaufwand (auf den mobilen Endgeräten) unsinnig, sodass das Bild nun als solches in der Datenbank gespeichert wird. Außerdem wurden die Datenbanken im Zuge des Arbeitspaketes „Teststreifenverwaltung im Webfrontend und Refactoring der Datenbank“ angepasst, sodass der Go-Server auch dahingehend angepasst werden musste. Eine neue Funktionalität ist ebenfalls eine Methode innerhalb der REST-Schnittstelle, welche es ermöglicht alle Teststreifenschemata abzurufen und so eine Aktualisierungsmöglichkeit der mobilen Applikationen gewährleistet. Ebenfalls wurde ein Schlüssel zur Nutzung der REST-Schnittstelle definiert. Dieser gewährleistet, dass eine Nutzung von nicht autorisierten Quellen ausgeschlossen ist. Weiterhin wurde die Transport Layer Security (TLS)-Funktionalität erweitert. Bisher wurde an dieser Stelle mit einem eigenem Zertifikat gearbeitet. Dieses Zertifikat wurde durch ein Zertifikat von letsencrypt.org[68] ersetzt, sodass es alle gängigen Browser und auch die mobilen Applikationen ohne weitere Zustimmung (wie es bei einem eigenem Zertifikat der Fall war) nutzen können. Eine vollständige Auflistung der Schnittstelle und ihrer Funktionen ist im Kapitel 4.6.6.1 auf Seite 273 zu finden.

[68] `https://letsencrypt.org/` - eine freie Zertifizierungsstelle

**Anpassung des Teststreifengenerators** Ähnlich wie im vorangegangenem Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“, ist eine minimale Anpassung des Teststreifengenerators vorgenommen worden. Siehe dazu 4.6.7.1 auf Seite 281. Speziell wurde das Attribute „corner-size“ aus technischen Gründen in „cornerSize“, sowie zur einheitlichen Benennung das Attribut „MiscOptions“ in „miscOptions“ umbenannt.

**Technologien** Es wurden keine neuen Technologien eingesetzt. Eine Auflistung der verwandten Technologien ist im Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ 4.6.6.1 auf Seite 272 unter dem gleichnamigem Kapitel „Technologien“ zu finden.

**Systembeschreibung** Die Systemlandschaft hat sich im Vergleich zum Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ 4.6.6.1 auf Seite 272 nicht verändert.

**Evaluation** Alle Funktionen wurden lokal und nach der Installation auf dem Debian-Server, welcher vorher als Server für die Spring-Komponente galt, getestet. Ein Test bzgl. der Performanz ist noch ausstehend und konnte auf Grund von mangelnder Ressourcen nicht durchgeführt werden. Frühere Tests einer Seminararbeit zeigten jedoch, dass der Go-Server bei gleicher Funktionalität dem Spring-Server im Hinblick auf Performanz überlegen ist.

**Parameter** Die Ergebnisse des Arbeitspaketes werden von keinen Parametern eingeschränkt.

**Fazit** Das Arbeitspaket „Finalisierung des Go-Servers“ konnte innerhalb von zwei Wochen abgeschlossen werden. Die dabei entstandene REST-Schnittstelle, entstand in enger Zusammenarbeit mit der Arbeitsgruppe der Weboberfläche, sowie denen der mobilen Applikationen.

**Ausblick** Bis jettz sind keine neuen Funktionalitäten zu erwarten.

**Anhang**
Nachfolgend ist eine vollständige Beschreibung der REST-Schnittstelle in englischer Sprache zu finden. Sämtliche Methodenaufrufe, samt Ergebnis sind in ihr beschrieben. Die nachfolgende API, ist ebenfalls im mitgeliefertem Quellcode unter 'server-backend/README.md' zu finden.

Configuration
==========
The server can be configured using a json configuration file with the following content:

```
{
  "db_host" : "134.106.47.158",
  "db_user" : "postgres",
  "db_password" : "x36aKWq-VJwL",
  "db_name" : "medic_db",
  "db_driver" : "postgres",
  "server_port" : "8081",
  "logfile" : "server.log",
  "ssl" : true,
  "public_key" : "ssl/public.pem",
  "private_key" : "ssl/private.key",
  "api_key" : "B61F5B362313C3BF3C47F981D27D9"
}
```

Listing 12: Konfiguration des Backends

after building the server with **go build**, it can be run with the following command:

```
sudo systemctl start medic-go-backend.service
```

API
==========

The **image** tag has moved out of the json string to a new field inside a multipartform.
To send a new request, post the following to your server.

```
#!/bin/bash
metadata=$(cat stripdescription.json)
curl -vX POST http://SERVER:PORT/selftests/create?apikey=KEY \
-F "teststripimg"=@"ff17025366.jpg" -F "metadata"="$metadata"
```

The server handles REST requests for the following tasks:

POST Testrecord
==========
http://SERVER:PORT/selftests/create?apikey=KEY

The generic testrecord format looks as follows:

```
{
  "appresult" : "+",
  "location" : { "lat" : 51, "lon": 8},
  "image" : "YnZrZmc=",
  "tokenid" : "68e59eea-b416-4b11-4f40-c725adf82315"
}
```

Listing 13: Testergebnis in JSON

The tokenid can be omitted, if unknown.
After checking for validity, the server will insert provided record in the database and return a json document, containing the newly generated testrecordid and tokenid.

```
{"tokenid":"68e59eea-b416-4b11-4f40-c725adf82315",
 "testrecordid":"82574"}
```

Listing 14: Antwort des Servers, wenn Testergebnis erstellt wurde

Or, if anything goes wrong, will respond with a corresponding HTTP - Error Code (probably 400).
Run **curl** to verify the functionality of the server:

```
curl -vX POST http://SERVER:PORT/selftests/create?apikey=KEY \
-d @test.json --header "Content-Type: application/json"
```

GET Testrecords by ID and TOKEN
==========
To get a stored record for given user token and testrecord id, call:
http://SERVER:PORT/selftests/get?token=TOKENID&id=TESTSTRIPID&apikey=KEY optionally with the flag includeimage=[true—false], which will determine if the image is contained or not.

The response will look like:

```
  {
    "testrecordid": 82574,
```

```
    "appresult": "+",
    "diagnoseResult": "",
    "diagnoseText": "",
    "diagnoseReviewer": -1,
    "created": "2015-11-23T18:57:07.069143Z",
    "location": {
      "lat": 51,
      "lon": 8
    },
    "image": "YnZrZmc=",
    "diagnoseSent": false
  }
```

Listing 15: Antwort des Servers, wenn Testergebnis abgefragt wurde

GET JSON all Testripschemas
==========
http://SERVER:PORT/getAllSchemas?apikey=KEY

For the update routine there is an option to get all schemas that are stored in the database. Therefore an Array of Teststripschemas is responded, which will look like this:

```
[
 {"id":2,
  "name":"Timos Krankheit",
  "data":{"teststrip":{...}},
  "created":1447372800,
  "active":true
 },
 {"id":1,
  "name":"Ebola",
  "data":{"teststrip":{...}},
  "created":1447372800,
  "active":true
 }
]
```

Listing 16: Antwort des Servers, wenn Teststreifenschema abgefragt wurde

GET JSON Schema for Teststripschema id
==========
http://SERVER:PORT/getSchema?id=1&apikey=KEY

```
{
  "id": 1,
  "name": "Erster streifen",
  "data": {
    "teststrip": {
      "miscOptions": {
        "backgroundFile": "",
        "colorType": 1,
```

```
        "dpi": 300,
        "drawText": true
      },
      "color": {
        "b": 255,
        "g": 0,
        "r": 0
      },
      "colorchanges": {
        "colorchange": [
          {
            "angle": 0,
            "cornerSize": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 0.25,
            "posX": 900,
            "posY": 175
          },
          {
            "angle": 0,
            "cornerSize": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 0.75,
            "posX": 950,
            "posY": 175
          },
          {
            "angle": 0,
            "cornerSize": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 1,
            "posX": 925,
            "posY": 150
          }
        ]
      },
      "extrainformation": {
        "height": 0,
        "posX": 0,
        "posY": 0,
        "type": "QR_CODE",
        "value": "init",
        "width": 0
      },
      "markers": {
        "marker": [
          {
            "posX": 760,
            "posY": 0,
            "size": 56,
```

```
            "value": 2000
          },
          {
            "posX": 1150,
            "posY": 100,
            "size": 64,
            "value": 4000
          },
          {
            "posX": 1250,
            "posY": 140,
            "size": 64,
            "value": 6000
          }
        ]
      }
    }
  },
  "created": 1447372800,
  "active": true
}
```

Listing 17: Teststreifenschema

Error-Struct in case the usage resulted in an error
==========

```
{"Status":"false",
 "Message":"message"
}
```

Listing 18: Fehlermeldung des Servers

If the server is down
==========

```
ssh medic@134.106.47.158
sudo systemctl start medic-go-backend.service
```

#### 4.6.8 Refactoring der iOS App und Integration von weiteren Anforderungen an die iOS App

- Priorität: Normal, muss jedoch nach vier Wochen abgeschlossen sein und die Bearbeitung muss direkt zum Beginn des Sprintes erfolgen.
- Motivation und Nutzen des Arbeitspaketes:
  Die Motivation dieses Arbeitspaketes liegt darin, weitere Anforderungen für das iOS-App laut Lastenheft umzusetzen. Zudem soll weiteres Know-How für iOS Entwicklung aufgebaut werden.

- Arbeitspaketbeschreibung:
  Der iOS-Code muss restrukturiert werden, sodass alte oder nicht verwendete Funktionen aus dem Quellcode entfernt werden. Veraltete Funktionen, welche z. B. für iOS 7 oder 8 verwendet wurden sind an iOS 9 anzupassen. Zudem sind die Anforderungen an die App laut Lastenheft zu integrieren. Dazu gehören z. B. folgenden Anforderungen (Alle weiteren Anforderungen sind dem Lastenheft selbstständig zu entnehmen):
  - FA-3: Die mobile Applikation kann das Ergebnis des Tests an den Server senden. Der mobile Datenservice oder WLAN wird genutzt um die Daten zu verschicken. Ist keine Verbindung verfügbar, werden die Daten übertragen sobald wieder eine aktive Verbindung besteht.
  - FA-5: Die mobile Applikation ist in der Lage zu erkennen, dass der Ladestand des Akkus nicht hoch genug für die Durchführung des Tests ist. Zu Beginn des Tests wird der Ladestand des Akkus abgefragt. Wenn dieser 10% oder weniger beträgt, wird die Testausführung abgebrochen.
  - FA-6: Die mobile Applikation kann dem Probanden mitteilen, dass der Ladestand des Akkus zu niedrig ist.
  - FA-7: Die Durchführung des Tests kann bis zu dem Zeitpunkt der Datenübermittlung durch den Probanden abgebrochen werden.
  - FA-8: Die mobile Applikation ist in der Lage zuvor angelegte Daten eigenständig zu löschen.
  - FA-9: Der Proband kann die Testdurchführung zu einem selbstgewählten Zeitpunkt starten.
  - FA-14: Die mobile Applikation des Probanden muss die Daten lokal zwischenspeichern, falls keine Verbindung mit dem Internet besteht, um diese nachträglich versenden zu könnnen.
  - FA-15: Die mobile Applikation muss eine Funktion enthalten, welche den Test nachträglich versendet, falls zuvor keine Verbindung zum Internet bestand oder die Verbindung während der Übertragung abbricht.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der iOS Code wurde von alten und nicht verwendeten Funktionen befreit.
- Aufwand:
  Vier Wochen
- Personen:
  - Timo Raß
  - Timo Schlömer

**4.6.8.1 Dokumentation** Der folgende Abschnitt der Dokumentation skizziert die Änderungen und Erweiterungen der iOS App. Grundsätzlich wurde das Augenmerk auf das Aufräumen des alten Quellcodes, die Integration eines neuen Designs mit Benutzerführung und die Integration weiterer funktionaler Anforderungen gelegt. Dies wird im folgenden Ablauf genauer beschrieben.

### Ablauf

#### Restrukturierung und Kommentierung des Quellcodes

Der Quellcode wurde analysiert und neu strukturiert um nicht benötigte Methoden oder Variablen zu entfernen. So wurden viele Methoden entfernt, welche nur der Kompatibilität zu iOS 7 dienten.

#### Neues Design mit Benutzerführung

In der neuen iOS App ist es nun möglich, zwischen verschiedenen Tabs zu navigieren. So kann ein neuer Test durchgeführt werden (Tab links), die Diagnosen angezeigt werden (Tab mitte), und in einem dritten Tab können allgemeine Informationen zur App aufgerufen werden (Tab rechts). Dies kann Abbildung 199 entnommen werden. Zudem wurden bereits alle Vorbereitungen getroffen, um in den Tab „New Test“ eine Informations-Animation einzufügen, um den Nutzer visuell zu erklären, wie ein Test durchzuführen ist.

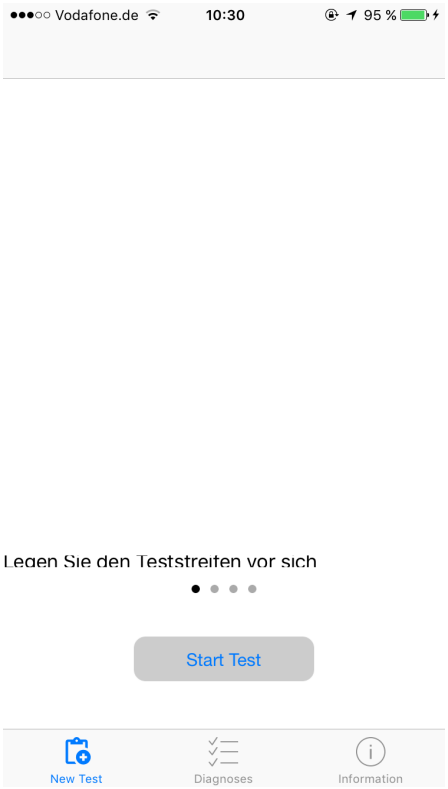



Abbildung 199: Startseite der neuen iOS App

Ebenso wurde die Kameraansicht überarbeitet. Der vom Bildverarbeitungsalgorithmus nicht verwendete Bereich des Bildes wurde ausgegraut und der restliche Ausschnitt des Bildes ist mit einem gelben Rechteck umrandet. Zudem gibt die App dem Nutzer die Information, den Teststreifen im gelben Rechteck zu positionieren. Dies kann Abbildung 200 auf der nächsten Seite entnommen werden.

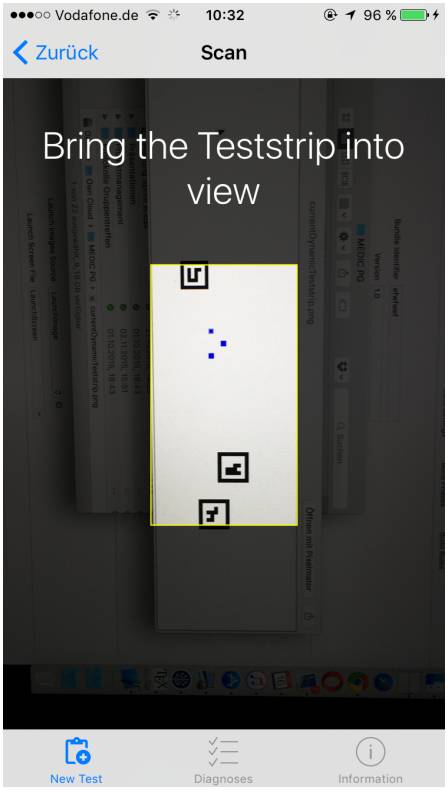



Abbildung 200: Neue Kameraansicht der iOS App

Nachdem der Test durchgeführt wurde, können im Tab „Diagnosen“ alte und neue Tests vom Probanden eingesehen werden. Dies kann Abbildung 201 auf der nächsten Seite entnommen werden. Dort werden ebenfalls die Diagnosen der Ärzte angezeigt. Diese können aufgerufen werden, indem ein gewünschter Test angeklickt wird. Siehe hierzu Abbildung 202 auf der nächsten Seite.

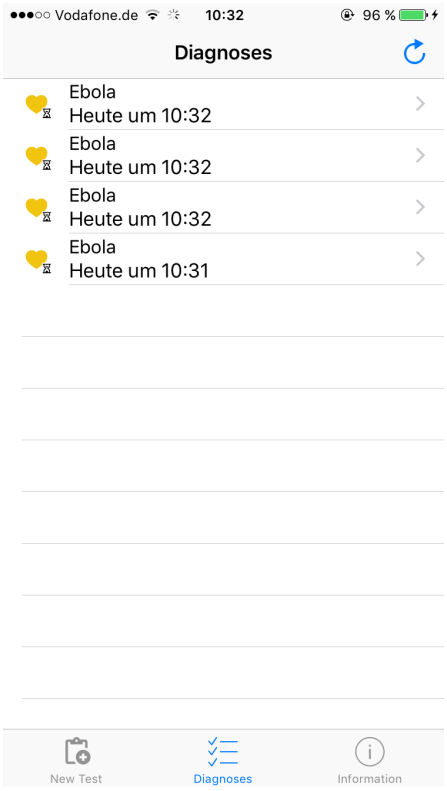



Abbildung 201: Liste mit durchgeführten Tests


Abbildung 202: Detailansicht eines Tests (noch nur Platzhalter)

Der Informations-Tab gibt die Möglichkeit allgemeine Informationen über die App oder auch über die Betreiber unterzubringen. Abbildung 203 zeigt die Ansicht des Informations-Tabs.

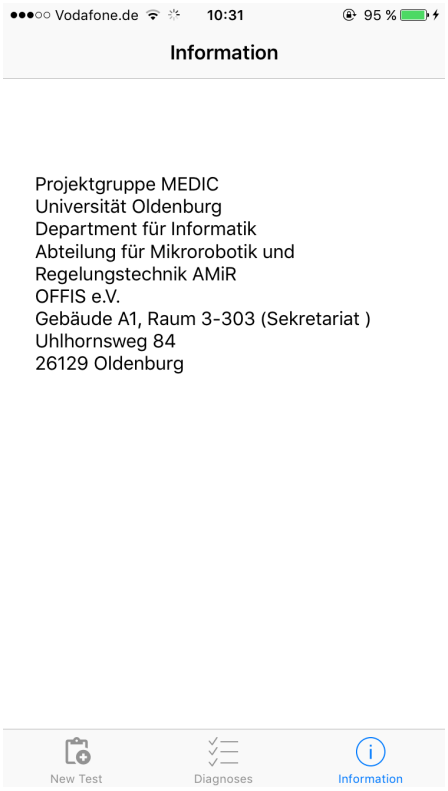



Abbildung 203: Information-Tab der iOS App

Zudem werden beim erstmaligen Starten der App alle notwendigen Abfragen an den Benutzer gestellt. So müssen die GPS-, die Benachrichtungs- und auch die Kamerafunktion durch den Nutzer bestätigt werden. Vgl. Abbildung 204 und 205.

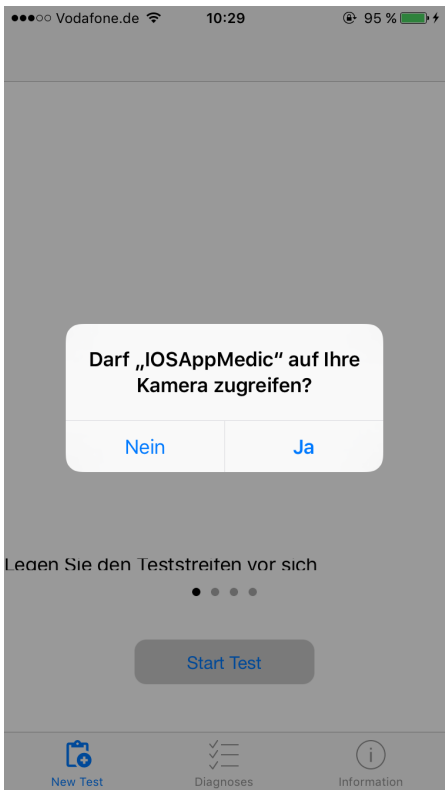



Abbildung 204: Abfrage, ob die App die Kamera nutzen darf

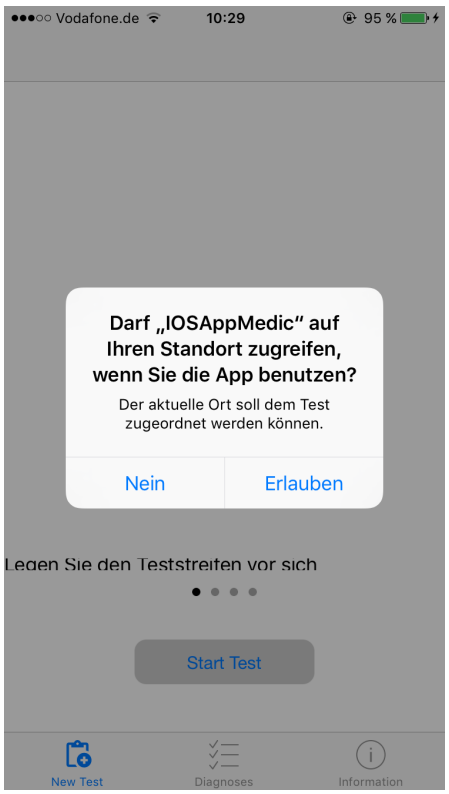



Abbildung 205: Abfrage, ob die App den Standort verwenden darf

### Neue Funktionen der iOS App

Im Folgenden werden die neuen Funktionen der iOS zusammengefasst.

- Abfrage, ob die App GPS nutzen darf.
- Standort aktualisieren, wenn die App in den Vordergrund gerufen wird.
- Abfrage, ob die App die Kamera nutzen darf.
- Abfrage, ob der Akkustand größer als Fünf Prozent.
- Tests werden in der lokalen Datenbank gespeichert.
- Tests werden automatisch versendet falls eine Internetverbidnung vorhanden ist.
- Je nachdem ob die Tests positiv, unbekannt oder negativ sind, wird ein anderes Symbol in der Diagnoseliste angezeigt.
- Integration einer Aktualisierungsfunktion für die Diagnosen.
- Die App kann in vertikaler und horizontaler Ausrichtung ausgeführt werden.
- Ausstehende Diagnosen werden regelmäßig abgerufen.
- Der verwertbare Ausschnitt der Kamera wird dem Nutzer nun deutlicher dargestellt.
- Es kann nun ein einleitender Hilfetext vor dem Start eines Tests dargestellt werden.

**Technologien**

**Core Data** Eine Bibliothek zur Persistierung von Daten für iOS, welche von Apple bereitgestellt wird.

**AFNetworking** Eine Bibliothek zum Erstellen und Verwalten von Anfragen an einen Server über das HTTP-Protokoll.

**Systembeschreibung** Das System wurde auf einem Mac Mini entwickelt. Als Entwicklungsumgebung wurde Xcode in der Version 7.2 verwendet. Die Funktionalität der Applikation wurde auf einem iPad Mini und einem iPhone 6 mit der iOS Version 9.2 getestet.

**Evaluation** Es wurde keine Evaluation durchgeführt.

**Parameter** Die Funktion der App wird durch keine Parameter eingeschränkt.

**Fazit** Durch dieses Arbeitpaket konnten ca. 88 % der Anforderungen an die iOS App umgesetzt werden. Dies ist sehr positiv zu Bewerten, da im restlichen Verlauf des Projektes ein starker Fokus darauf gelegt werden kann, die App weiter zu Designen (optisch), die Benutzerführung zu optimieren und kleinere Fehler auszubessern. Zudem ist die App an einem Punkt angekommen, dass erste Feldtests durchgeführt werden können, um weitere nicht-funktionale Anforderungen evaluieren und umsetzen zu können.

**Ausblick** Die iOS-App muss noch Verbesserungen erhalten, wie etwa Bilder, Texte und Animationen für die Einleitung in den Test, ein Design für die Detailansicht einer Diagnose und korrekte Texte für das Impressum und die Lizenzen in der Informationsseite der App. Zusätzlich muss die letzte Anforderung an die iOS-App umgesetzt werden: Das korrekte Analysieren eines Teststreifens mit einer Entscheidung ob der Proband erkankt ist oder nicht.

### 4.6.9 Integration der dynamischen Bildverarbeitung in die Apps

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Die Apps müssen in der Lage sein die neuen dynamischen Teststreifen zu erkennen, damit Tests sinnvoll durchgeführt werden können.
- Arbeitspaketbeschreibung:
  Die im Arbeitspaket „Anpassung der Bildalgorithmen auf dynamische Teststreifen erarbeiteten Bildverarbeitungsalgorithmen“ genannten Anforderungen werden in die Apps eingearbeitet. Der eigentliche Ablauf des Tests soll dadurch nicht beeinflusst werden. Darüber hinaus wurde ein Konzept erarbeitet und umgesetzt, das die Identifikation von Teststreifen anhand von XML-Dateien erlaubt. Das Konzept muss folgende funktionale Anforderungen beachten:
  - FA-14: Die mobile Applikation des Probanden muss die Daten lokal zwischenspeichern, falls keine Verbindung mit dem Internet besteht, um diese nachträglich versenden zu können.
  - FA-15: Die mobile Applikation muss eine Funktion enthalten, welche den Test nachträglich versendet, falls zuvor keine Verbindung zum Internet bestand oder die Verbindung während der Übertragung abbricht.

- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Die Apps verfügen über den aktualisierten Bildverarbeitungsalghoritmus und sind dadurch in der Lage, dynamische Teststreifen zu erkennen und den Testvorgang darauf durchzuführen. Die Funktionalität wurde durch Tests im Teststand verifiziert.
- Aufwand:
  Vier Wochen
- Personen:
  - Raphael Kappes
  - Christian Sandmann
  - Christoph Ressel (ab Woche 3)

**4.6.9.1 Dokumentation** Ziel dieses Arbeitspaketes war es, die Android- und iOS-App so anzupassen, dass die dynamischen Teststreifen von diesen erkannt werden können. Dazu wurde das Arbeitspaket in mehrere Aufgaben unterteilt:

- **Empfangen neuer Teststreifenvarianten:**
  Der Server stellt eine Datenschnittstelle bereit, die von den Apps unterstützt werden muss. Dies beinhaltet die Bereitstellung eines Datenschemas für die Teststreifenvarianten.
- **Periodische Datensynchronisation:**
  Die Apps müssen in einem bestimmten Zeitintervall neue Teststreifen von dem Server laden. Damit die Apps nicht während des herunterladens der Daten benutzbar bleibt, soll dies im Hintergrund geschehen.
- **Lokale Datenbank:**
  Die erhaltenen Teststreifenvarianten müssen in einer lokalen Datenbank auf dem Smartphone gespeichert werden, damit die Apps auch funktionieren, wenn das Handy kein Internet hat.
- **Datenkonvertierung:**
  Um den Bildverarbeitungsalgorithmus zu benutzen, muss aus den gespeicherten Informationen der Teststreifenvarianten nur die notwendigen Informationen in ein passendes Datenformat überführt werden. Die Konvertierung ist nötig, da die vom Server erhaltenen Daten Informationen enthalten, für die Kommunikation wichtig sind, jedoch nicht für die Bildverarbeitung.

**Ablauf** Für das Empfangen von Teststreifenvarianten ist ein Datenschema in der iOS- und Android-App implementiert worden, das mit Absprache der Arbeitsgruppe des Go-Servers entwickelt wurde. Das Datenschema ist im Arbeitspaket 4.6.7.1 auf Seite 278 definiert. Die Daten werden mittels einer Anfrage vom Server an das Smartphone geschickt. Dabei werden verschiedene Fehlerfälle abgefangen, wie der Verbindungsverlust, keine Erreichbarkeit des Servers und fehlerhafte Daten durch die Übertragung.

Zur periodischen Datensynchronisation wurde in Android ein **Sync Adapter** implementiert. Dieser ermöglicht die regelmäßige Ausführung von Synchronisationsanfragen an den Server, selbst

wenn die App nicht im Vordergrund ist. In iOS hingegen existiert bereits eine vom System bereitgestellte Methode (**performFetchWithCompletionHandler**) zum regelmäßigen Ausführen von Programmcode. Dies ermöglicht eine periodische Datensynchronisation auch in iOS.

Die Datenbank, die für die Speicherung der Informationen auf dem Smartphone benötigt wird, wurde in Android mit „SQLite[69]“ umgesetzt. Es gibt eine Tabelle, in der die Informationen für einen Teststreifen gespeichert werden. Zu jedem Teststreifen wird eine ID, die zur Identifikation benötigt wird und deshalb einzigartig sein muss, abgespeichert. Außerdem enthält die Tabelle den Namen, den Zeitpunkt der Erstellung des Teststreifens und ob dieser gerade aktiv ist oder nicht. Des Weiteren gibt es einen Eintrag mit allen weiteren Informationen. Dies ist ein in JSON formatierter Text, der alles enthält, was für die Bildverarbeitung nötig ist: die Positionen der Farbumschläge und der Marker, die Farbe des Umschlags, sowie einige Zusatzinformationen.
In iOS wurde die Datenbank mit „CoreData“umgesetzt (vgl. Kapitel 4.6.8 auf Seite 284).

Um die Daten zu konvertieren, muss in Android eine Java zu C-Schnittstelle (JNI) benutzt werden, die es ermöglicht Daten in den Bildverarbeitungsalgorithmus (Programmiersprache C) weiterzugeben. Dafür wird zunächst eine Teststreifenvariante aus der Datenbank geladen und die benötigten Parameter ausgelesen. In iOS ist eine solche Schnittstelle nicht nötig, da dort in Objective-C (C-Derivat) entwickelt wird. Jedoch muss auch hier eine Teststreifenvariante aus der Datenbank geladen, die benötigten Parameter ausgelesen und dem Bildverarbeitungsalgorithmus übergeben werden.

**Technologien**

**ION** Eine Bibliothek, welche Netzwerkzugriffe über HTTP vereinfacht.

**Gson** Eine Bibliothek um Java Objekte in JSON zu überführen.

**Sync Adapter** Eine Android-spezifische Komponente, die zur periodischen Ausführung von Aufgaben (zumeist Datenaustausch mit einem Server) benutzt werden kann und unabhängig vom Lebenszyklus der App existiert.

**AFNetworking** Eine Bibliothek, die Netzwerkkommunikation für iOS vereinfacht.

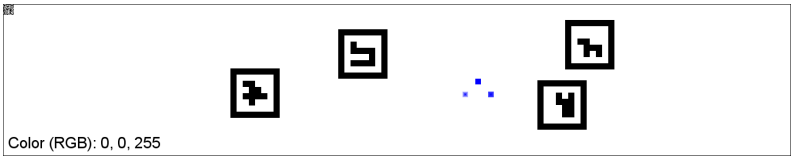
**Systembeschreibung** Die Android-App wurde unter Windows 10 Pro (64 Bit), Windows 7 Professional Service Pack 1 (64 Bit) und ArchLinux entwickelt. Die iOS-App wurde unter MacOS X El Capitan (Version 10.11.2) auf einem Mac mini (Ende 2014) entwickelt.

**Evaluation** Die iOS- und Android-App wurde auf dem teuersten (iPad Mini 3) und dem günstigsten (Sony Xperia E1) Gerät getestet. Dazu wurden sieben verschiedene Teststreifen benutzt (Siehe 206 auf der nächsten Seite und 207 auf der nächsten Seite):

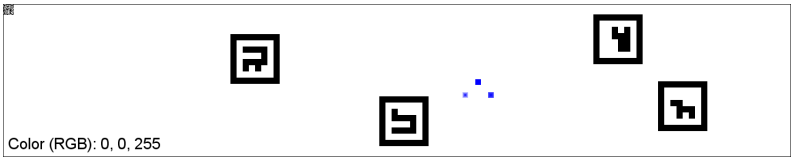
[69] `https://de.wikipedia.org/wiki/SQLite`


(a) 1. Teststreifen


(b) 2. Teststreifen

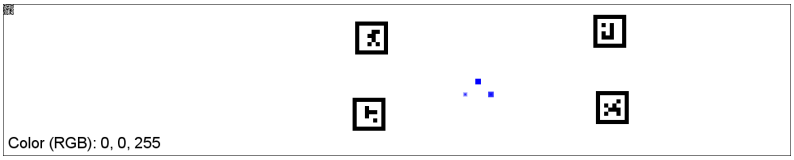



(c) 3. Teststreifen

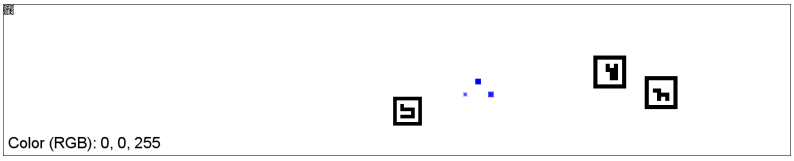



(d) 4. Teststreifen

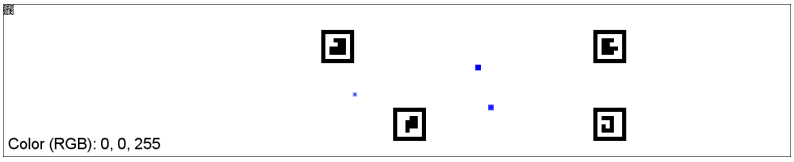
Abbildung 206: Sieben Teststreifenvarianten




(a) 5. Teststreifen




(b) 6. Teststreifen




(c) 7. Teststreifen

Abbildung 207: Sieben Teststreifenvarianten

Diese wurden im Web-Frontend in der Administrationsoberfläche hochgeladen und nach einer gewissen Zeit (die Smartphones fragen nach einer gewissen Zeit beim Server an und dieser verschickt alle Teststreifen) aus der Datenbank des Servers an die Smartphones geschickt. Wenn die Daten vom Smartphone empfangen wurden, ist überprüft worden, ob diese korrekt an den Bildverarbeitungsalgorithmus weitergeleitet wurden. Danach wurde geprüft, ob die Teststreifen erkannt werden. Dafür wurden das Smartphone und der Teststreifen im Teststand befestigt und die Werte aus Tabelle 17 auf Seite 424 bestimmt. (Die Wertung der Spalte „Detektierbar“ ist: Gar nicht, selten, manchmal,

oft, sehr oft, immer - Teststreifen, die beim iPad befriedigend funktionierten, wurden beim Sony Xperia E1 getestet):

Hier sind Teststreifen (Siehe Abbildung 206 auf Seite 292 und Abbildung 207 auf Seite 292), die schlecht detektierbar sind achtmal und Teststreifen, die gut detektierbar sind fünfmal überprüft worden. Daraus ergibt sich einerseits, dass der Algorithmus mit manchen Markern besser arbeiten kann, als mit anderen. Andererseits liefert der Algorithmus die selben Ergebnisse auf dem günstigen, sowie auf dem teuren Gerät. Weiterhin hängt die Detektierbarkeit der Marker von der Rotation des Teststreifens ab. Wenn Teststreifen und Smartphone gleich ausgerichtet sind, hat das Smartphone die meisten Marker erkannt. Sobald die beiden nicht gleich ausgerichtet sind, verschlechtert sich das Ergebnis deutlich.

**Parameter** Der entwickelte Algorithmus vertraut darauf, dass die vom Server empfangenen Teststreifenvarianten valide sind.

**Fazit** Das Arbeitspaket „Integration von Teststreifenvarianten in die Apps“ konnte innerhalb der zur Verfügung stehenden Zeit von vier Wochen erfolgreich fertiggestellt werden. Die neuen App-Versionen sind in der Lage automatisch neue Teststreifenvarianten in die App zu importieren. Hierzu existieren in den Apps Hintergrundprozesse, die unabhängig vom Lebenszyklus der Apps und regelmäßig die lokalen Teststreifenvarianten mit denen des Servers synchronisiert. Zudem ist es jetzt das erste Mal möglich, den Bildverarbeitungsalgorithmus hinsichtlich verschiedener Teststreifenvarianten auf den Smartphones valide zu testen. Hierbei ist aufgefallen, dass die Wahl der Marker eine entscheidenden Einfluss auf den Erfolg des Algorithmus hat.

**Ausblick** Zur Kommunikation mit dem Server werden API-Keys verwendet, die momentan noch als Klartext im Quellcode abgelegt sind. Diese müssen zukünftig unter jeden Umständen verschlüsselt gespeichert werden. Des Weiteren muss der Bildverarbeitungsalgorithmus hinsichtlich der Erkennung von Markern verbessert werden.

### 4.6.10 Teststreifenverwaltung in Web-Frontend und Refactoring der Datenbank

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Weitere Anforderungen sollen umgesetzt werden, um den Projektfortschritt zu verbessern und das Endprodukt in weiteren Iterationen zu finalisieren. Des Weiteren sollen vorhandene alte Datenbankobjekte aufgeräumt bzw. gelöscht werden, um ein aufgeräumtes Produkt ausliefern zu können.
- Arbeitspaketbeschreibung:

  Am Ende dieses Arbeitspakets soll das Szenario „Institut verwaltet Infrastruktur“ zum größten Teil in das Projekt integriert sein. Dazu sind nachfolgend die noch ausstehenden funktionalen Anforderungen sowie weitere Spezifikationen bzgl. der Anforderungen gelistet.

  - funktionale Anforderungen
    - FA-37: Die Administrationsoberfläche bietet die Möglichkeit, neue Teststreifenvarianten anzulegen.
    - FA-38: Der Server beherrscht die Speicherung von Teststreifenvarianten. Es existiert eine zentrale Datenhaltung aller Teststreifenvarianten und ihrer Daten.
    - FA-39: Das System prüft nach Entgegennahme von Teststreifendaten diese auf syntaktische und semantische Korrektheit. Duplikate werden erkannt.
    - FA-40: Die Administrationsoberfläche ermöglicht die Bearbeitung von bestehenden Teststreifenvarianten.

        * FA-41: Die Administrationsoberfläche bietet die Möglichkeit, bestehende Teststreifenvarianten zu durchsuchen. Ergebnis der Suche ist eine Liste an Teststreifenvarianten, welche den Suchparametern entsprechen.
    - weitere Spezifikationen
        * Die Speicherung erfolgt in der bereits vorhandenen Datenbank und wird mittels Hibernate in der JSF-Applikation angesprochen. Dazu muss darauf geachtet werden, dass das MVC-Pattern eingehalten wird (vgl. bestehender Code). Des weiteren ist eine sinnvolles Modell eines Teststreifens zu erarbeiten.
        * Das Anlegen und Verwalten von Teststreifenvarianten über die Web-Oberfläche darf nur von Administratoren vorgenommen werden. Dazu ist die AdminAuthFilter.java dahingehend anzupassen.
        * Für den Aufbau der Oberfläche zum Anlegen und Verwalten von Teststreifenvarianten ist sich an den bereits bestehenden Verwaltungsseiten zu orientieren.

- Vorbedingungen:
  Das Arbeitspaket „Anpassung des Teststreifengenerators“ muss abgeschlossen sein.
- Nebenbedingungen:
  -
- Nachbedingungen:
  Das Szenario „Institut verwaltet Infrastruktur“ ist zum größten Teil in das Projekt integriert. Des weiteren ist das Datenbankschema in einem finalem Zustand, sodass dieses nur noch für Wartungsarbeiten angepasst werden muss.
- Aufwand:
  Vier Wochen
- Personen:
    - Nicolas Koch
    - Kevin Sandermann

**4.6.10.1 Dokumentation** In der Datenbank wurden nicht länger benutzten Tabellen und Spalten entfernt. Weiterhin wurde die Weboberfläche an diese Änderung angepasst und selbst auch überarbeitet um nicht verwendete Methoden zu löschen. Zusätzlich können Administratoren neue Teststreifen anlegen.

**Ablauf** Der erste Teil des Arbeitspaket beschäftigte sich mit der Überarbeitung (engl. Refactoring) der vorhandenen Datenbank. Die relationale PostgreSQL-Datenbank[70] ist fester Bestandteil der gesamten Systemlandschaft des Medic-Projekts und realisiert die Datenhaltung sämtlicher zu persistierender Daten: Neben Benutzerdaten und Passwörtern existieren vor allem Tabellen für durchgeführte Krankheitstests und Teststreifenvarianten. Vor der Durchführung dieses Arbeitspakets wies die Datenbank einige Inkonsistenzen auf, so waren etwa Tabellen- und Spaltennamen nicht einheitlich umgesetzt und es existierten noch alte, ungenutzte Tabellen der Prototypen. Aufgrund dessen wurde zunächst eine einheitliche Benennung für Tabellen und Spalten erarbeitet, um anschließend ein neues Datenbankschema aufzustellen.
Eine Herausforderung hierbei stellte der Stellenwert der Datenbank im Gesamtsystem dar: Sowohl

[70] `https://de.wikipedia.org/wiki/PostgreSQL`

der Go-Server als auch das Web-Frontend benötigen die Datenbank für die Umsetzung jeglicher Funktionalität. Aufgrund dessen war es an dieser Stelle wichtig, das neu erarbeitete Datenbankschema mit den für diese Komponenten zuständigen Arbeitsgruppen abzusprechen. Um außerdem ganz im Sinne des Continious-Integration-Paradigmas durchgehend eine funktionsfähige Version des Systems zu erhalten, wurde das neue Datenbankschema in einer komplett neuen Datenbank umgesetzt, die unter neuem Namen auf dem gleichen Server verfügbar ist. Zusätzlich wurde weiterhin die alte Version die Datenbank weiterhin gehostet, solange nicht alle Systemkomponenten mit der neuen Version kompatibel waren.
Wie oben beschrieben wies das alte Datenbankschema Inkonsistenzen in seiner Benennung auf. Das Schema der alten Datenbank ist in der Dokumentation des Arbeitspaket „Gesamtarchitektur visualisieren“ in der Dokumentation von Sprint 5 zu finden. Das neue Datenbankschema hingegen setzt eine einheitliche Namenskonvention um und ist von sämtlichen veralteten Daten befreit.

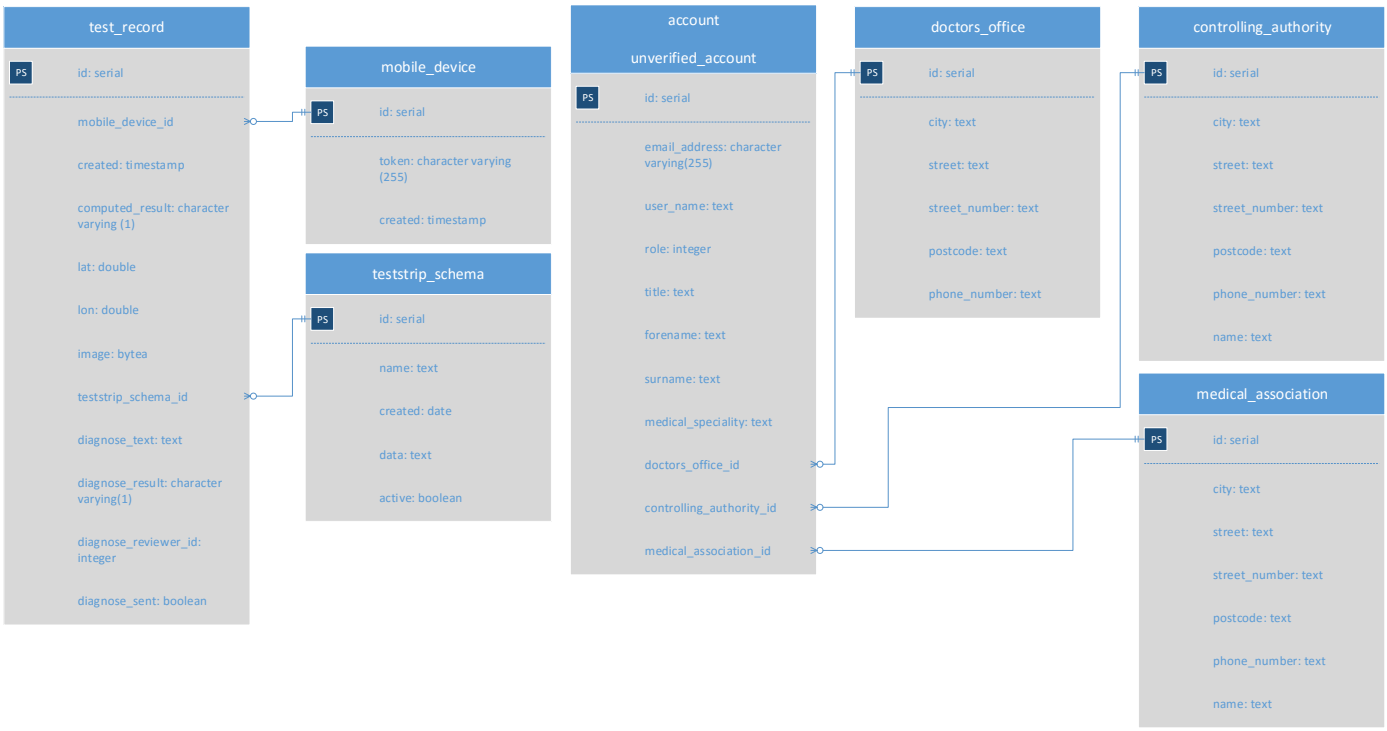



Abbildung 208: ER-Diagramm des neuen Datenbankschemas

Abbildung 208 zeigt das neu erarbeitete Datenbankschema in Form eines ER-Diagramms. Neben den oben erwähnten Neuerungen ist zu erkennen, dass eine einheitliche Verwendung von Datentypen umgesetzt wurde. In Zuge dessen wurde außerdem die Speicherung der Bilddaten eines durchgeführten Krankheitstests von reinem Text im Base64-Format[71] in eine binäre Speicherung der **.png-Datei**[72] geändert. Zusätzlich hierzu ist das abgebildete neuen Datenbankschema um eine Datenstruktur zur Speicherung von verschiedenen Teststreifenvarianten für verschiedene Krankheiten erweitert worden. Des Weiteren wurde ein SQL-Skript geschrieben, welches ermöglicht die Datenbank reproduzierbar und automatisiert zu erstellen. Dazu stellt dieses neben den Relationen auch Administrationsbenutzer und Fremdschlüsselbeziehungen her. Das Skript befindet sich im Wurzelverzeichnisses der Primfaces-Weboberfläche.
Der zweite Teil des Arbeitspakets beschäftigte sich mit der Entwicklung einer Funktion, die die Verwaltung und das Anlegen von Teststreifenvarianten im Web-Frontend ermöglicht und umsetzt.

[71] `https://de.wikipedia.org/wiki/Base64`
[72] `https://de.wikipedia.org/wiki/Portable_Network_Graphics`

Hierfür wurde eine neue Seite in der Benutzeroberfläche der Administratoren erstellt. Jeder Administrator hat nun die Möglichkeit, über die Schaltfläche **Edit Teststripschemata** in der Navigationsleiste des Web-Frontends die Verwaltung der Teststreifenvarianten aufzurufen. Wird diese Seite aufgerufen, wird dem Administrator zunächst eine Liste mit allen aktuell in der Datenbank befindlichen Teststreifenvarianten angezeigt.

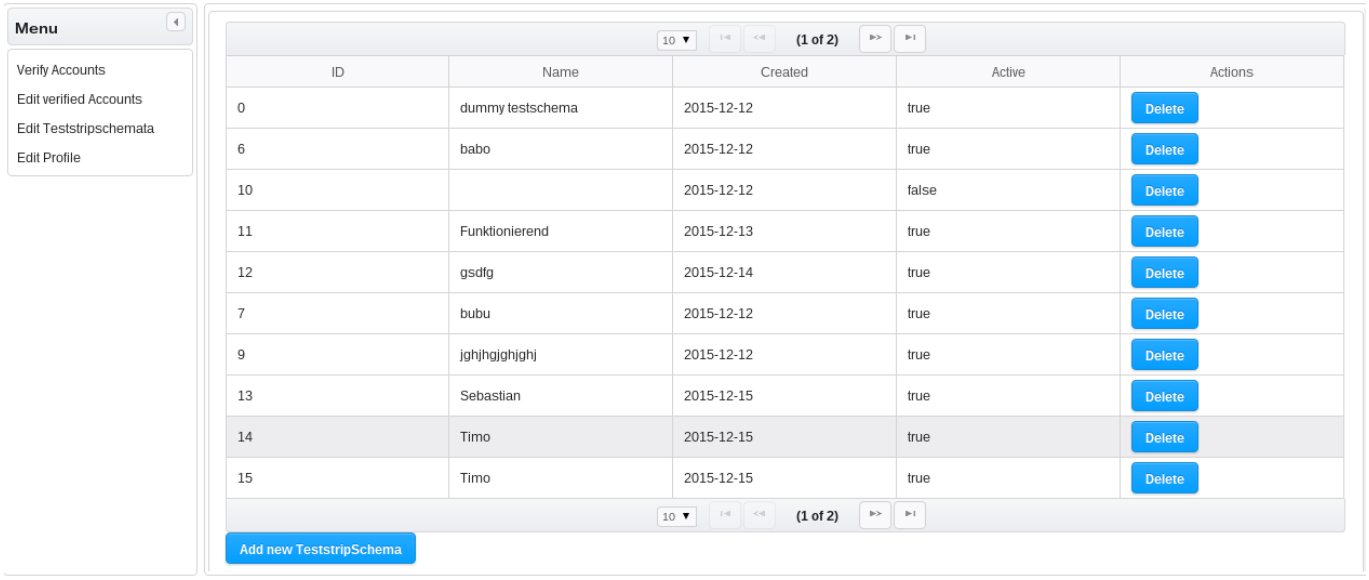



Abbildung 209: Anzeige aller Teststreifenvarianten in der Benutzeroberfläche des Web-Frontends

Abbildung 209 zeigt diese Liste aller aktuellen Teststreifenvarianten. Wie in der Abbildung zu erkennen ist, wird dem Nutzer hier eine Übersicht über die wichtigsten Eigenschaften der Teststreifenvarianten angezeigt: Neben der ID und dem Namen der Teststreifenvariante werden ebenfalls das Erstellungsdatum sowie der aktuelle Status (Teststreifenvariante aktiv/inaktiv) angezeigt. Unter der Spalte **Actions** wird dem Nutzer eine Schaltfläche **Delete** zur Verfügung gestellt, welche die Löschung einer Teststreifenvariante ermöglicht. Neben der Löschung einer Teststreifenvariante hat der Nutzer außerdem die Möglichkeit, einen bestimmten Eintrag in der Liste auszuwählen, um diesen zu bearbeiten.

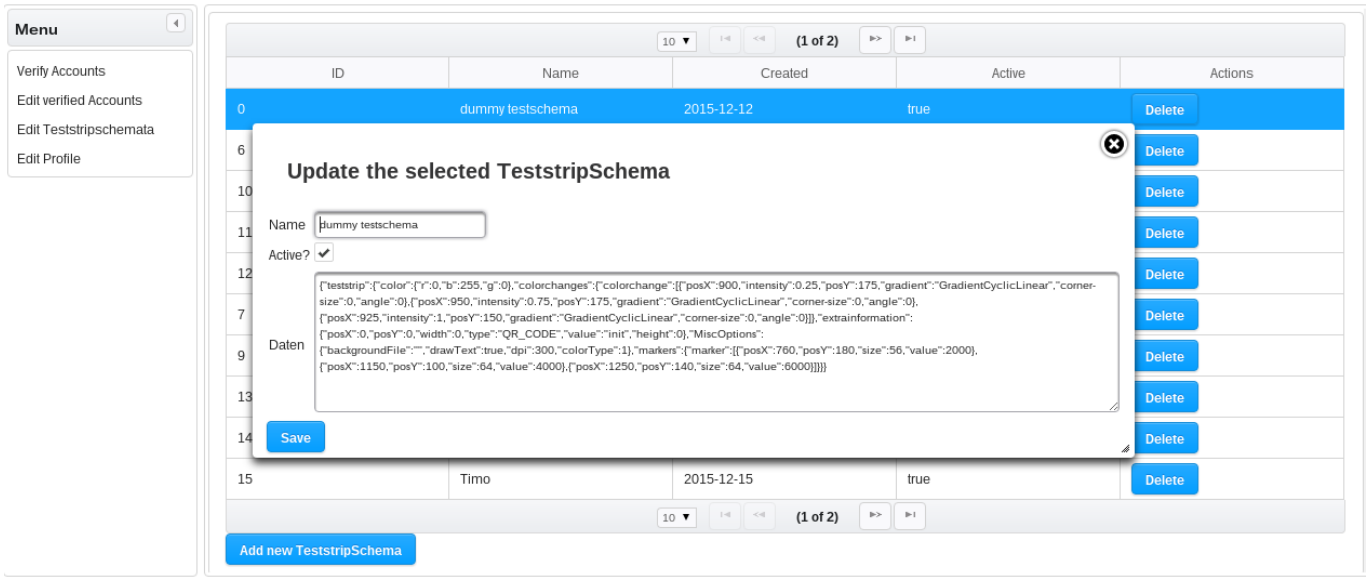



Abbildung 210: Benutzeroberfläche zur Bearbeitung einer Teststreifenvariante

Abbildung 210 auf der vorherigen Seite zeigt das Fenster zur Bearbeitung einer Teststreifenvariante, welches sich bei Auswahl des Teststreifens aus der Liste öffnet. Wie zu erkennen ist, hat der Administrator hier die Möglichkeit, sämtliche Eigenschaften einer Teststreifenvariante zu editieren und die Änderungen anschließend entweder über die **Save**-Schaltfläche zu speichern oder die Änderungen durch einfaches Schließen des Fenster rückgängig zu machen. Neben den bereits in der Übersicht aller Teststreifenvarianten dargestellten Eigenschaften eines Eintrags ist in diesem Fenster außerdem das der Teststreifenvariante zugrunde liegende JSON-Schema[73] sichtbar und editierbar. Dieses JSON-Schema entstammt ursprünglich der Erstellung einer Teststreifenvariante mit Hilfe des Teststreifengenerators. Sobald der Administrator eine Änderung hieran vornimmt, prüft das Web-Frontend serverseitig direkt ob sich das geänderte JSON noch erfolgreich in eine Teststreifenvariante umwandeln lässt. Ist dies nicht der Fall, wird dem Nutzer eine entsprechende Fehlermeldung gezeigt und die Änderungen werden nicht gespeichert. Das entsprechende Eingabefeld lässt dabei ausschließlich Dateien im JSON-Format zu und stellt sicher, dass die Datei nicht größer als 1 MB ist.
Neben dem Entfernen und der Bearbeitung von vorhandenen Teststreifenvarianten ist als letzte Funktionalität zur Verwaltung eine Oberfläche zum Anlegen einer komplett neuen Variante von Teststreifen umgesetzt. Über die Schaltfläche **Add new Teststripschema** lässt sich das in folgender Abbildung dargestellte Menü öffnen:

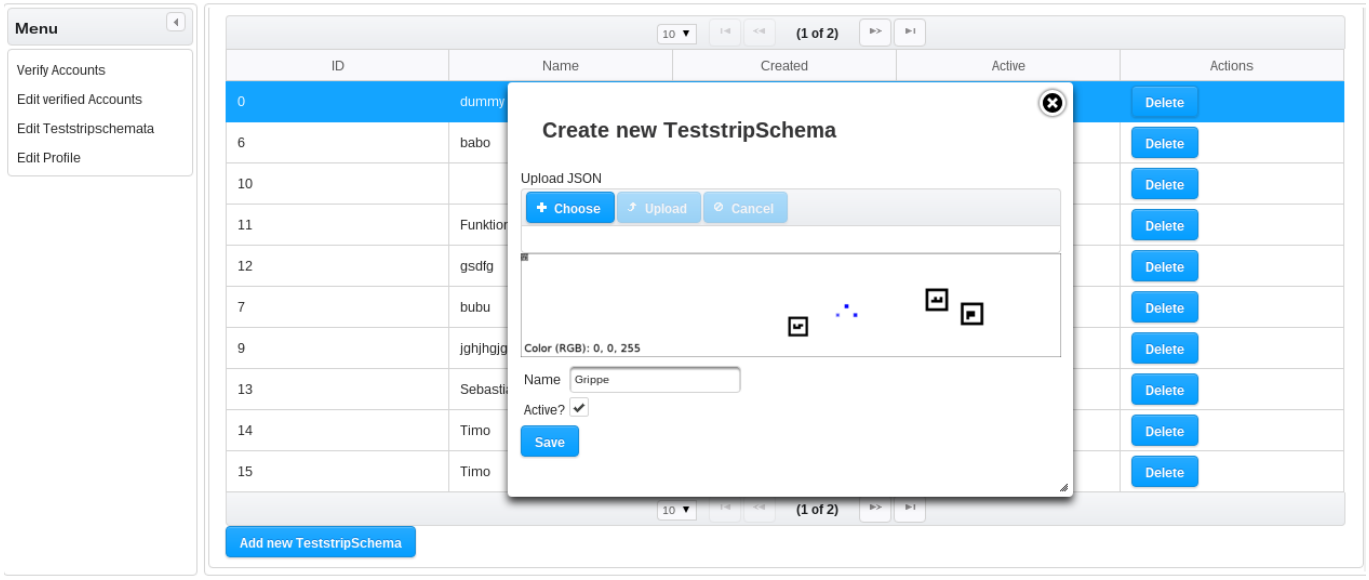



Abbildung 211: Benutzeroberfläche zur Erstellung einer neuen Teststreifenvariante

[73] https://de.wikipedia.org/wiki/JavaScript_Object_Notation

Die in obiger Abbildung dargestellte Benutzeroberfläche ermöglicht es, eine neue Teststreifenvariante anzulegen. Als Grundlage jeder Teststreifenvariante dient wie oben beschrieben eine durch den Teststreifengenerator angelegte JSON-Datei. Nachdem der Administrator mit Hilfe des Teststreifengenerators eine passende JSON-Datei erzeugt hat, kann er diese hochladen. Der Hochladevorgang kann im Web-Frontend über zwei Möglichkeiten angestoßen werden: Die erste Möglichkeit ist die Ablage der JSON-Datei über eine Drag& Drop-Schaltfläche. Zweite Möglichkeit des Hochladens ist die Auswahl der Datei über die Schaltfläche **Choose**. Bei beiden Varianten muss die Dateiauswahl jeweils über die Betätigung des **Upload**-Knopfs bestätigt werden. Sobald der Nutzer eine JSON-Datei hochgeladen hat, wird diese auf Fehler überprüft und gegebenenfalls eine passende Fehlermeldung angezeigt. Ist die Datei valide, wird serverseitig ein Bild der Teststreifenvariante generiert und dem Nutzer angezeigt. Auf technischer Ebene ist diese Funktion durch die Integration des Teststreifengenerators in das Primefaces-Web-Frontend umgesetzt. Diese Integration beinhaltet die Einbindung der immer aktuellsten Version des Teststreifengenerators. Nachdem die JSON-Datei erfolgreich in ein Bild der Teststreifenvariante umgewandelt wurde, kann der Administrator noch einen Namen und den Status der Teststreifenvariante festlegen und diesen speichern. Die Eigenschaften **ID** und **Created** werden automatisch vom Server vergeben. Da durchgeführte Tests nun Informationen über die Art des Teststreifens enthalten, wird diese nun auch auf der Benutzeroberfläche in der Spalte **Disease** angezeigt. Das neue Anzeigefeld ist in Abbildung 212 zu sehen.

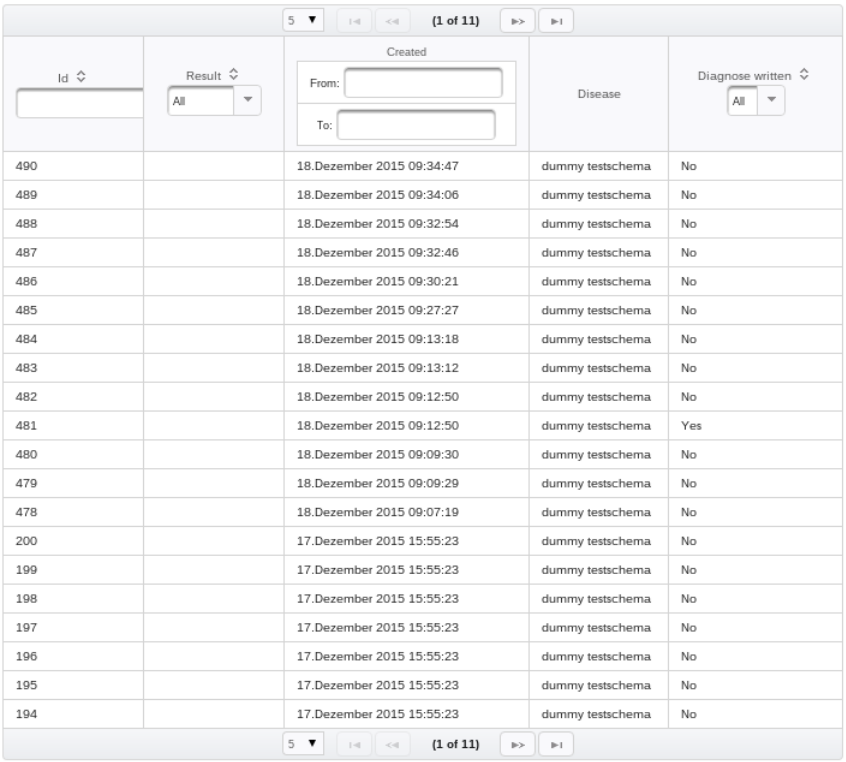



Abbildung 212: Anzeige der Krankheitsart in der Liste der Testergebnisse

In einem Produktionsumfeld werden die Teststreifenvarianten aussagekräftige Namen haben. Dies ist in den aktuell verwendeten Testdatensätzen nicht der Fall.

### Technologien

**Teststreifengenerator** Der Teststreifengenerator des Projektes wurde in die Serverlogik eingebunden, sodass eine Vorschau eines Teststreifens angezeigt werden kann.

**Systembeschreibung** Das Primefaces-Frontend ist in den Entwicklungsumgebungen Eclipse und NetBeans entwickelt worden. Andere Bibliotheken werden mit Hilfe des Build-Management-

Tools[74] Maven eingebunden. Die Anwendung wurde auf einem Apache-Tomcat Webserver evaluiert. Des Weiteren ist durch den Teststreifengenerator eine neue Abhängigkeit hinzugekommen, die ebenfalls mit Hilfe von Maven in das Projekt eingebunden wurde.

**Evaluation** Zur Evaluation der Korrektheit des neuen Datenbankschemas wurde das Web-Frontend ausgiebig getestet. Dazu haben zwei Entwickler jeden vor der Überarbeitung möglichen Anwendungsfall manuell durchgeführt und anschließend die entstandenen Datensätze in der Datenbank überprüft.
Die neuen Funktionalitäten zum Anlegen, Bearbeiten und Löschen von Teststreifenvarianten wurden ebenfalls durch manuelle Tests im Web-Frontend getestet. Außerdem konnten parallel zu diesem Arbeitspaket andere Systeme, wie z.B. der Go-Server einwandfrei auf die neuen Datensätze zugreifen.

**Parameter** Alle Änderungen die am Datenbankschema durchgeführt wurden sind rückwärtskompatibel. Wie zuvor ist auch die neue Version nicht durch Parameter eingeschränkt (siehe vorherige Version, im Sprint Fünf, im Kapitel 4.4.13.1 auf Seite 191)
Zum Anlegen der Teststreifenvarianten müssen folgende Parameter durch den Benutzer zur Verfügung gestellt werden:

- Name der neuen Teststreifenvariante
- Status der neuen Teststreifenvariante (Aktiv oder inaktiv)
- Eine JSON-Datei, die das Layout des Teststreifens beschreibt.

**Fazit** Durch die Umsetzung dieses Arbeitspaketes konnten die Anforderungen an das Web-Frontend zu einem großen Teil umgesetzt werden. Die einzigen Anforderungen aus dem Lastenheft, welche noch nicht umgesetzt worden sind, sind FA-25 („Die Auswertungsoberfläche stellt dem Mediziner eine Liste mit Probanden bereit, dessen Diagnosen noch ausstehen“) und FA-30 („Nach der Diagnose ermöglicht es die Auswertungsoberfläche dem Mediziner, seine Diagnose mit der Voranalyse der mobilen Applikation zu vergleichen“).

**Ausblick** In folgenden Arbeitspaketen müssen die zwei noch ausstehenden funktionalen Anforderungen umgesetzt werden. In Anschluss daran befindet sich das Web-Frontend in einem finalen Zustand und sollte nur noch für Wartungszwecke, nicht aber für neue Funktionalitäten verändert werden müssen.

### 4.6.11 Fazit des Sprints

Wie in der Einleitung beschrieben, lag der Fokus dieses Sprints auf der breit gefächerten Umsetzung der funktionalen Anforderungen. Die Weboberfläche sowie der Go-Server wurden stark weiterentwickelt, wodurch der Fortschrittsgrad von vorher 60% auf 91% verbessert werden konnte (vgl. Abbildung 213 auf der nächsten Seite, Fortschrittsgrad errechnet auf Basis der abgeschlossenen funktionalen Anforderungen). Der bisher verwendete Spring-Server als Kommunikationsschnittstelle zwischen Verwaltungsinstanz und mobilen Endgeräten, wurde durch einen neuen Backend-Server (geschrieben in Go) ersetzt, wodurch sämtliche, ausstehende und geplanten Technologiewechsel erfolgreich abgeschlossen wurden. Auch die mobilen Applikationen wurden innerhalb dieses Sprints erfolgreich erweitert. Die dynamische Teststreifenerkennung konnte integriert und die Applikationen um zusätzliche Funktionen, wie beispielsweise das Auslesen der GPS-Koordinaten oder des Akkustandes erweitert werden. Des Weiteren wurden auch Arbeiten verrichtet, die den allgemeinen Projektfortschritt nicht

[74] `https://de.wikipedia.org/wiki/Erstellungsprozess`

vorantrieben, jedoch zusätzliche Stabilität gewährleisten und somit erheblich zur Projektzielerreichung beitragen. Darunter fallen die Überarbeitung der Datenbankstruktur und das Überarbeiten der Quellcode-Struktur der iOS-Applikation.

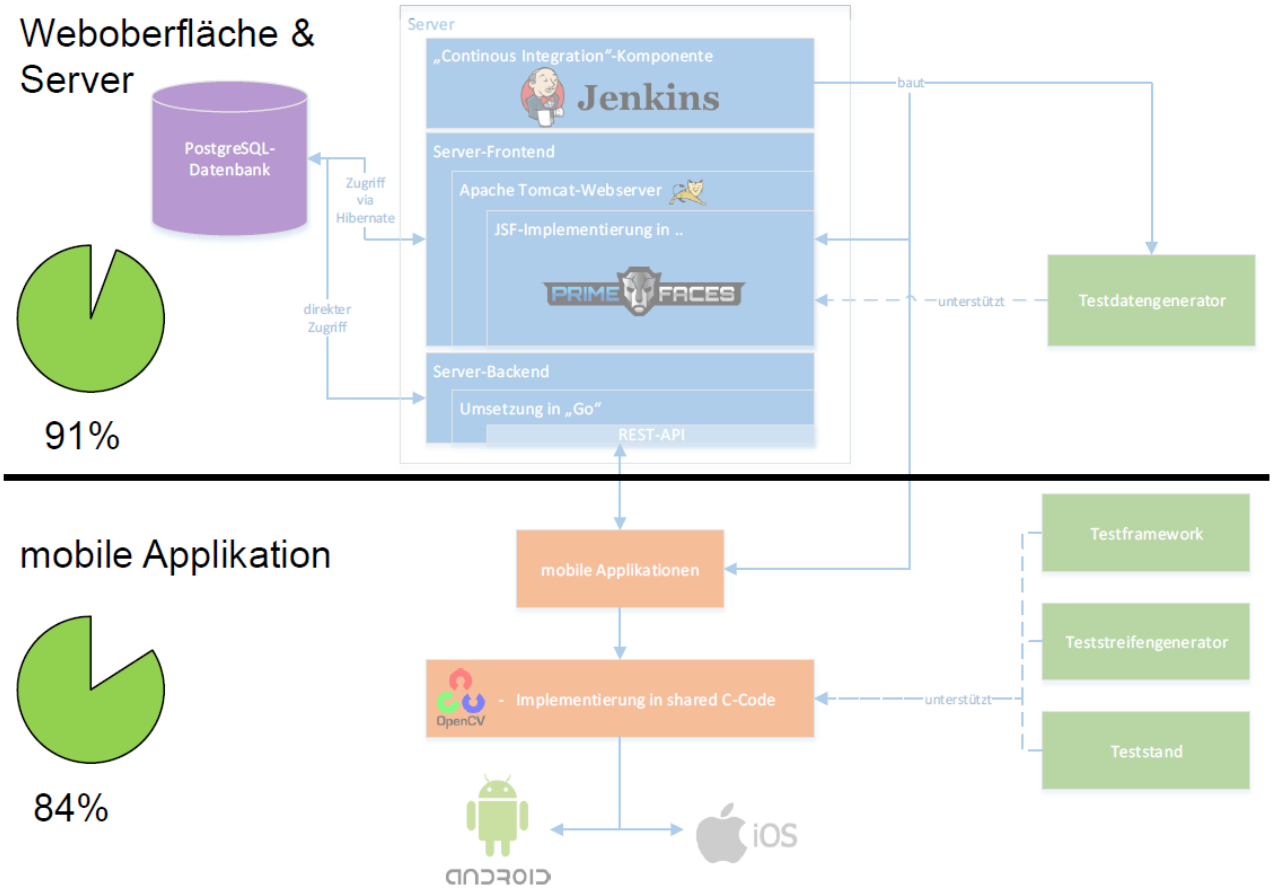



Abbildung 213: Fortschrittsanalyse nach dem Abschluss von Sprint 7

Die in Abbildung 213 gezeigte Systemlandschaft samt Fortschrittsanalyse unterscheidet sich lediglich in der Komponente „Server" von der in der Einleitung gezeigten Systemlandschaft. Die Server-Komponente wurde, wie bereits beschrieben, durch eine effizientere Lösung ersetzt. Zusätzlich konnten bereits erste Ansätze bezüglich der Gestaltung im Web-Frontend, wie auch in den mobilen Applikationen umgesetzt werden. Hierbei handelt es sich jedoch nur um Ansätze, die in den nachfolgenden Sprints finalisiert werden.

| Szenario: Proband führt Test durch | | | | | |
|---|---|---|---|---|---|
| FA-1 | ✓ | FA-6 | ✓ | FA-11 | ✓ |
| FA-2 | ✓ | FA-7 | ✓ | FA-12 | ✓ |
| FA-3 | ✓ | FA-8 | ✗ | FA-13 | ✓ |
| FA-4 | ✗ | FA-9 | ✓ | | |
| FA-5 | ✓ | FA-10 | ✓ | | |
| | | | | | |
| **Szenario: Kommunikation Mediziner – Proband** | | | | | |
| FA-14 | ✓ | FA-16 | ✓ | FA-18 | ✗ |
| FA-15 | ✓ | FA-17 | ✓ | FA-19 | ✗ |
| | | | | | |
| **Szenario: Auswertung durch den Mediziner** | | | | | |
| FA-20 | ✓ | FA-23 | ✗ | FA-26 | ✓ |
| FA-21 | ✓ | FA-24 | ✓ | FA-27 | ✓ |
| FA-22 | ✓ | FA-25 | ✓ | FA-28 | ✗ |
| | | | | | |
| **Szenario: Institut verwaltet Infrastruktur** | | | | | |
| FA-29 | ✓ | FA-34 | ✓ | FA-39 | ✓ |
| FA-30 | ✓ | FA-35 | ✓ | FA-40 | ✓ |
| FA-31 | ✓ | FA-36 | ✓ | FA-41 | ✓ |
| FA-32 | ✓ | FA-37 | ✓ | FA-42 | ✓ |
| FA-33 | ✓ | FA-38 | ✓ | | |

Tabelle 18: Erfüllung funktionaler Anforderungen

Zusätzlich wurde eine Auflistung der Schlüsselnummern der funktionalen Anforderungen und ihr Erreichungsgrad (erfüllt / nicht erfüllt) erstellt, welche sich in Tabelle 18 wiederfindet. Zur Aufschlüsselung der einzelnen Anforderungen, ist das Lastenheft heranzuziehen. Durch eine Auswertung der Tabelle, ergibt sich ein Projekt-Gesamtfortschritt von 88% (vgl. Abbildung 214).

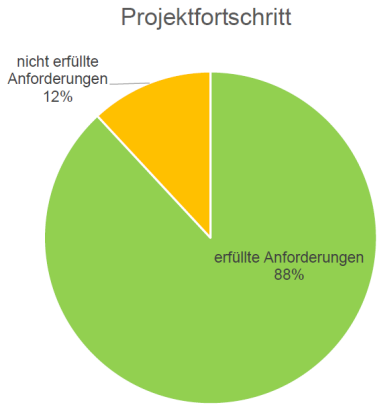



Abbildung 214: Gesamtfortschritt des Projekts nach Sprint 7

Die am Beginn von Sprint 7 definierten Sprintziele konnten erfolgreich umgesetzt werden. Dadurch ist es uns möglich in dem nachfolgenden Sprint, einen erhöhten Fokus auf die Finalisierung der einzelnen Komponenten zu legen. Ebenfalls ist die Umsetzung der nicht-funktionalen Anforderungen ein nächster, wichtiger Schritt im Projektverlauf. Durch die Berücksichtigung dieser ergibt sich innerhalb des nächsten Sprints ein aussagekräftiger Projektfortschritt.

## 4.7 Sprint 8

Sprint 8 des Projekts legt seinen Fokus vor allem auf die Verbesserung des Bildverarbeitungsalgorithmus der mobilen Applikationen. Aufgrund von mehreren parallel ablaufenden Seminararbeiten und Urlaubsfällen, besteht der Sprint lediglich aus vier Arbeitspaketen:
Das Paket „Stabilisierung der Bildverarbeitung“ (Kapitel 218 auf Seite 310) beschäftigt sich vor allem mit der Verbesserung der Markerdetektierung der mobilen Applikationen. Zusätzlich soll die Erkennung von Bar- und QR-Codes der Teststreifen umgesetzt bzw. optimiert werden.
Ein weiteres Arbeitspaket von Sprint 8 ist die „Erweiterung des Teststands“ (Kapitel 4.7.3 auf Seite 306). Dieses Paket setzt es sich zum Ziel, den Teststand endgültig zu finalisieren, sodass zukünftige Tests den Bildverarbeitungsalgorithmus zuverlässig evaluieren. Die Erweiterungen des Teststands umfassen sowohl physische Optimierungen als auch Änderungen an der Software.
Das Paket „Vergleich der Bildverarbeitungsalgorithmen“setzt es sich zum Ziel, das in Sprint 6 erstellte Test-Framework (siehe Kapitel 4.5.7.1 auf Seite 259) so zu erweitern, dass mehrere verschiedene Versionen des Bildverarbeitungsalgorithmus miteinander verglichen werden können.
Das letzte geplante Arbeitspaket ist die „Dokumentationsvorbereitung“ (Kaptiel 4.7.6 auf Seite 342). Dieses Arbeitspaket beschäftigt sich bereits mit der Gliederung der im nächsten Sprint anzufertigenden Abschlussdokumentation. Um spätere Zeitprobleme zu minimieren, wird bereits in diesem Sprint die Grundstruktur des Dokuments erstellt.

In Abbildung 215 ist die Systemlandschaft und ein Überblick über den Fortschritt zu Beginn dieses Sprints dargestellt.

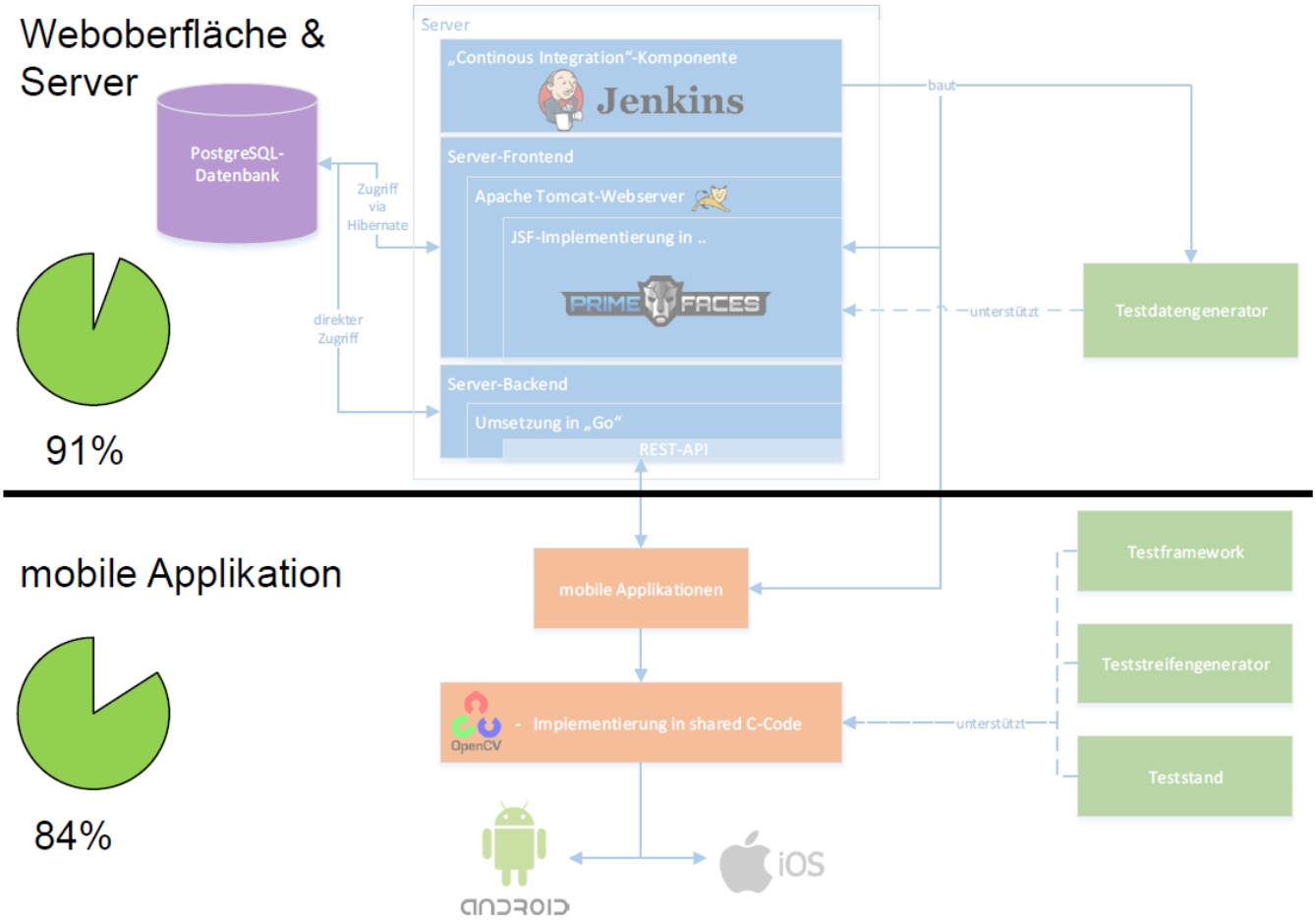



Abbildung 215: Systemarchitektur zu Beginn von Sprint 8

Wie in der Abbildung zu sehen ist, sind 91 % der funktionalen Anforderungen im Bereich der Weboberfläche und des Servers umgesetzt. Die funktionalen Anforderungen der mobilen Applikation sind zu 84 % realisiert. Wie bereits beschrieben beinhalten die Arbeitspakete von Sprint 8 vor allem die Optimierung bereits bestehender Systeme sowie Vorbereitungen für Aufgaben des zukünftigen Sprints, weswegen eine Steigerung der prozentualen Erfüllung der funktionalen Anforderungen nicht geplant ist. Weiterhin wurde die ursprüngliche Planung, in diesem Sprint den Großteil der nichtfunktionalen Anforderungen umzusetzen, aufgrund der geringen Verfügbarkeit der Projektmitglieder auf den nächsten Sprint verschoben.

### 4.7.1 Sprintplanung

Im Folgenden wird die Planung des achten Sprints (04.01.2016 bis 17.02.2016, sechs Wochen) genauer aufgezeigt. Dabei wird ein besonderes Augenmerk auf die folgende Punkte gelegt:

- Fokus des Sprints
- Arbeitspakete und Planung
- Planung der einzelnen Ressourcen

**Fokus des Sprints**
Der Fokus für den Sprint 8 liegt auf der Optimierung bestehender Systeme. Vor allem die Bildverarbeitungsalgorithmen sollen effizienter und robuster gemacht werden. Auf Grund begrenzter Ressourcen, die durch Urlaube und Seminararbeiten zu Stande kommen, können weniger Arbeitspakete als im vorherigen Sprint vollendet werden.

**Arbeitspakete und Planung** Aus dem Fokus des Sprints und der daraus resultierenden Planung ergaben sich die unten stehenden Arbeitspakete.

- Stabilisierung der Bildverarbeitung (Priorität: Normal - Sechs Wochen á drei Personen)
- Erweiterung des Teststandes (Priorität: Normal - Zwei Wochen á zwei Personen)
- Erweiterung des Testframeworks (Priorität: Normal - Zweieinhalb Wochen á eine Person)
- Vorbereitungen für die Abschlussdokumentation (Priorität: Normal - Eine Woche á zwei Personen)

Der folgenden Abbildung 216 auf der nächsten Seite können die geplanten Arbeitspakete entnommen werden. Es gibt keinen direkten kritischen Pfad, da alle Arbeitspakete unabhängig voneinander erfüllt werden können. Die Grafik beinhaltet nur normale Arbeitspakete. Optionale Arbeitspakete sind aus Platzgründen nicht mit aufgenommen.

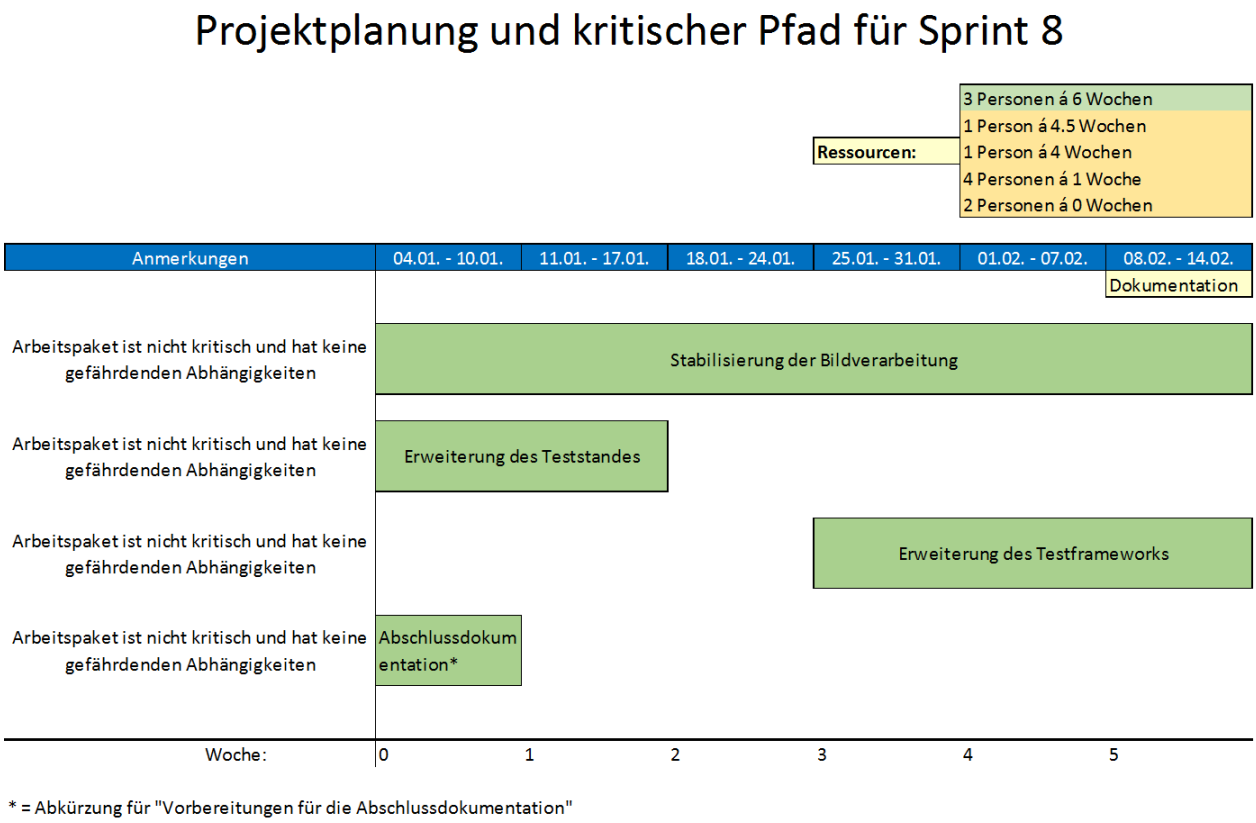



Abbildung 216: Planung und kritischer Pfad für Sprint 8

**Planung der Ressourcen**

Im Folgenden ist die Personalplanung dargestellt. Diese gibt Auskunft darüber, welches Teammitglied an welchem Arbeitspaket gearbeitet hat.

- Raphael Kappes: Null Wochen - Die komplette Zeit an seiner Seminararbeit gearbeitet
- Timo Raß: Null Wochen - Die komplette Zeit an seiner Seminararbeit gearbeitet
- Danny Fonk: Eine Woche - Serverseitige Integration der Pushtechnologien, fünf Wochen Urlaub
- Nicolas Koch: Eine Woche - Allgemeine Sprintdokumentation, fünf Wochen Urlaub
- Kevin Sandermann: Eine Woche - Allgemeine Sprintdokumentation, fünf Wochen Urlaub
- Christian Sandmann: Eine Woche - Vorbereitungen für die Abschlussdokumentation, fünf Wochen an seiner Seminararbeit gearbeitet
- Christoph Ressel: Vier Wochen - Stabilisierung der Bildverarbeitung, zwei Wochen Urlaub
- Sebastian Horwege: Viereinhalb Wochen - Zwei Wochen Erweiterung des Teststandes, zweieinhalb Wochen Erweiterung des Testframeworks, eineinhalb Wochen Urlaub
- Timo Schlömer: Sechs Wochen - Stabilisierung der Bildverarbeitung
- Jan Philipp Stubbe: Sechs Wochen - Stabilisierung der Bildverarbeitung
- Daniel Wegmann: Sechs Wochen - Vier Wochen Stabilisierung der Bildverarbeitung, zwei Wochen Erweiterung des Teststandes

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und dessen Dokumentation skizziert.

### 4.7.2 Arbeitspaketdokumentation

In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete wie gewohnt detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.7.3 Erweiterung des Teststandes

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Das Arbeitspaket soll dazu dienen den Teststand zu optimieren, damit die Bildverarbeitungsalgorithmen noch effizienter getestet werden können.
- Arbeitspaketbeschreibung:
  Am Ende des Arbeitspaketes ist der Teststand soweit erweitert bzw. modifiziert, dass auch mehrstündige Tests reibungslos durchgeführt werden können. Darunter werden sowohl alle physischen Modifizierungen (u.a. Integration des Zahnriemens), als auch alle Anpassungen am Quellcode verstanden.
  Zudem existiert ein Programm, welches den Teststand in unterschiedlichen zeitlichen Abständen automatisch fahren lässt und die Winkel automatisch verstellt. Dieses Programm wird für den Schülertag am 22.01.2016 benötigt, kann aber auch im weiteren Projektverlauf eingesetzt werden.
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der Teststand kann ohne Probleme mehrere Stunden genutzt werden. Zudem kann das Stativ des Teststandes durch die Modifizierung des Zahnriemens exakt durch das Programm positioniert werden (am Besten auf einen cm genau).
- Aufwand:
  Zwei Wochen
- Personen:
  - Sebastian Horwege
  - Daniel Wegmann

**Dokumentation** Diese Arbeitspaket ist eine Erweiterung des Arbeitspakets „Erstellung eines Teststandes“ aus Sprint 5 (siehe Kapitel 4.4.14.1 auf Seite 192). Im Folgenden wird beschrieben, welche Probleme beim Teststand auftraten, und wie diese behoben wurden.

**Ablauf** Die Steuerungssoftware besaß einige kleinere Fehler, die die Bedienung unnötig kompliziert gestalteten. Diese Fehler wurden behoben. Es ist nicht länger möglich in den Textfeldern der manuellen Steuerung Winkel einzugeben, die von den Motoren nicht umgesetzt werden können. Zwar wurden fehlerhafte Eingaben zuvor bereits korrigiert, diese Korrektur dem Nutzer jedoch nicht angezeigt. Zusätzlich wurden eine Fortschrittsanzeige bei der automatischen Testausführung eingeführt und mehrere Fehlermeldungen verständlicher formuliert.

Im Teststand selbst wurde eine interne Stromversorgung in Form eines PC-Netzteils eingebaut. Diese übernimmt die vollständige Versorgung des Teststands. Lediglich der Arduino wird weiterhin über die USB-Verbindung versorgt. Im Zuge dieses Umbaus wurde auch die Stromversorgung der Servomotoren vom Arduino an das Netzteil verlegt. Dadurch konnte erreicht werden, dass die Stromversorgung der Motoren unter Last aufrecht erhalten werden konnte, was zuvor die Kapazität des Arduino überschritt. Diese Änderung führte ebenfalls dazu, dass keine Abbrüche der seriellen Verbindung mehr auftraten, da diese ebenfalls durch die Überlastung des Arduino verursacht wurden.

Die Veränderung der Distanz des Stativs zum Aufnahmegerät konnte, auf Grund verzögerter Lieferung benötigter Bauteile, erst im Zuge dieses Arbeitspakets zuverlässig implementiert werden. Es wurde ein Zahnriemen an den Motor gebaut und mittels einer speziell entworfenen Halterung am Stativ befestigt. Dadurch tritt nicht länger ein „Durchrutschen“ des Antriebsriemens beim Bewegen des Stativs auf. So kann das Stativ zuverlässig und präzise bewegt werden. Hierzu mussten ebenfalls die Achsen des Antriebssystems überarbeitet und auf Kugellagern gelagert werden, da sonst zu hohe Reibungskräfte auftraten. Die Programmierung des Arduino wurde entsprechend der neuen Parameter angepasst.

Auf Grund des geänderten Teststreifendesigns erwies sich die Halterung für die Streifen am Stativ als zu klein, weswegen eine vergrößerte Version gedruckt und angebracht wurde. Außerdem wurde die Beleuchtung innerhalb des Teststands reduziert. Dadurch treten weniger Lichtreflexe auf, die bei der Bildverarbeitung zu Fehlern führen.

Abbildung 217: Der überarbeitete Teststand

**Technologien** Die verwendeten Technologien decken sich mit denen des vorherigen Arbeitspakets „Erstellung eines Teststandes“ (siehe Kapitel 4.4.14.1 auf Seite 192).

**Systembeschreibung** Alle getroffenen Maßnahmen haben dazu beigetragen, dass das System wesentlich stabiler läuft. Abgesehen davon haben sich die Funktionen des Teststands im Vergleich zum Arbeitspaket „Erstellung eines Teststandes“ nicht verändert.

**Evaluation** Die Funktionalität des Teststands wurde während der Umbaumaßnahmen kontinuierlich überprüft. Dabei konnte die Stabilität des gesamten Systems schrittweise verbessert werden. Vor allem die Fehlerbeseitigung im Steuerungsprogramm, sowie die nun ausreichende Stromversorgung, sorgen zum Abschluss dieses Arbeitspakets dafür, dass der Teststand in seiner Funktionalität nicht mehr eingeschränkt ist. Der Teststand konnte mehrfach automatisch eine Sequenz von über 500 Einstellungen ohne Komplikationen abfahren. Aus weiteren Testläufen mit zwei verschiedenen mobilen Endgeräten, konnten bereits 506 verschiedene Einstellungen aufgenommen und mithilfe des Test-Frameworks evaluiert werden.

Alle vom Test-Framework erhobenen, relevanten Daten werden in einer Zeitreihendatenbank auf dem PG-Medic Rechner zwischengespeichert. Diese Daten können auf einer, durch das Framework bereitgestellten Weboberfläche[75] verwaltet werden.

**Parameter** Der Teststand akzeptiert manuelle und automatisierbare Eingaben. Für automatisierte Eingaben wird eine Eingabedatei mit Informationen zu allen Winkeleinstellungen, sowie der Distanzeinstellung benötigt. Die Wertebereiche der beiden Eingabemöglichkeiten sind gleich: Für die Winkeleinstellungen werden ausschließlich ganzzahlige Eingaben akzeptiert. Die Minimal- und Maximalwerte sind für jeden Winkel individuell:

**Winkel 1** $-90$ bis 90

**Winkel 2** $-25$ bis 25

**Winkel 3** $-65$ bis 65

Die erlaubte Distanz des Schlittens zur Initialposition liegt zwischen 0 und 33 Zentimetern.

**Fazit** Zum Ende des ersten Arbeitspakets zur „Erstellung eines Teststandes“ entstand ein eingeschränkt funktionsfähiger Prototyp. Im Laufe dieses Arbeitspakets konnten alle funktionalen Einschränkungen erfolgreich beseitigt werden, sodass dem produktiven Einsatz des Teststandes nichts mehr im Wege steht. Die Gesamtstabilität des Systems konnte hergestellt werden. Der überarbeitete Teststand ist in Abbildung 217 auf der vorherigen Seite dargestellt.

**Ausblick** Der Teststand ist einsatzbereit und ermöglicht es nun, Videos reproduzierbar mit unterschiedlichen mobilen Endgeräten und Teststreifenvarianten zu erzeugen. Der aktuelle Datensatz von 506 Videos umfasst jedoch noch nicht die gesamte Bandbreite an Einstellungsmöglichkeiten, die der Teststand zu bieten hat. Der Teststand muss daher ab jetzt intensiv zur Erstellung von Testdaten mit unterschiedlichen mobilen Endgeräten und Teststreifenvarianten genutzt werden. Das Test-Framework wurde dementsprechend angepasst, sodass nun der Bildverarbeitungsalgorithmus evaluiert werden kann.

### 4.7.4 Stabilisierung der Bildverarbeitung

- Priorität: Normal
- Arbeitspaketbeschreibung:
  Die Algorithmen der Bildverarbeitung werden in folgenden Punkten überarbeitet bzw. verbessert:

[75] Weboberfläche zur Verwaltung: `http://134.106.47.158:3000/`

1. Detektierung der Marker: Die Detektierung muss alle Marker zuverlässig erkennen. In diesem Zusammenhang ist das GitLab-Issue # 5 zu analysieren.
2. Schwellwert des Farbumschlags: Es existiert ein Schwellwert, anhand dessen der Algorithmus ein Ergebnis liefern kann, ob die Probe auf eine Krankheit hindeutet oder nicht.
3. Evaluierung Barcode oder QR-Code: Es wird evaluiert, ob der Einsatz eines Barcodes die Zuverlässigkeit bei der Identifizierung des Teststreifens erhöht. Die Anzahl von einzelnen Scan-Vorgängen ist dabei möglichst gering zu halten.

- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Der Bildverarbeitungsalgorithmus kann dynamisch die Teststreifenvariante identifizieren und erkennt zuverlässig genügend Marker, um in jedem Fall das Farbumschlagsfeld finden zu können. Als Ergebnis wird eine Aussage darüber geliefert, ob das Farbumschlagsfeld einen Hinweis auf eine Erkrankung liefert. Die korrekte Funktion wurde mittels umfangreicher Tests im Teststand sichergestellt.
- Aufwand:
  Sechs Wochen
- Personen:
  - Timo Schlömer
  - Jan Philipp Stubbe
  - Christoph Ressel
  - Daniel Wegmann

**Dokumentation**

**Ablauf** Im ersten Schritt wurde analysiert, in welchen Bereichen noch Probleme im Bildverarbeitungsalgorithmus auftraten. Dabei stellte sich heraus, dass die Teststreifen nicht in allen möglichen Rotationen erkannt werden konnten. Der Grund dafür lag darin, dass durch einen Fehler nicht alle Markerrotationen abgeglichen wurden. Nach der Behebung dieses Fehlers wurden schließlich alle möglichen Rotationen erkannt.

Eine weitere Schwäche in der Markererkennung bestand darin, dass bei großen Aufnahmen der Rand der Marker nicht mehr als zusammenhängende Struktur erkannt werden konnte, wenn das Bild binarisiert wurde (vergleiche Abbildung 218 auf der nächsten Seite).

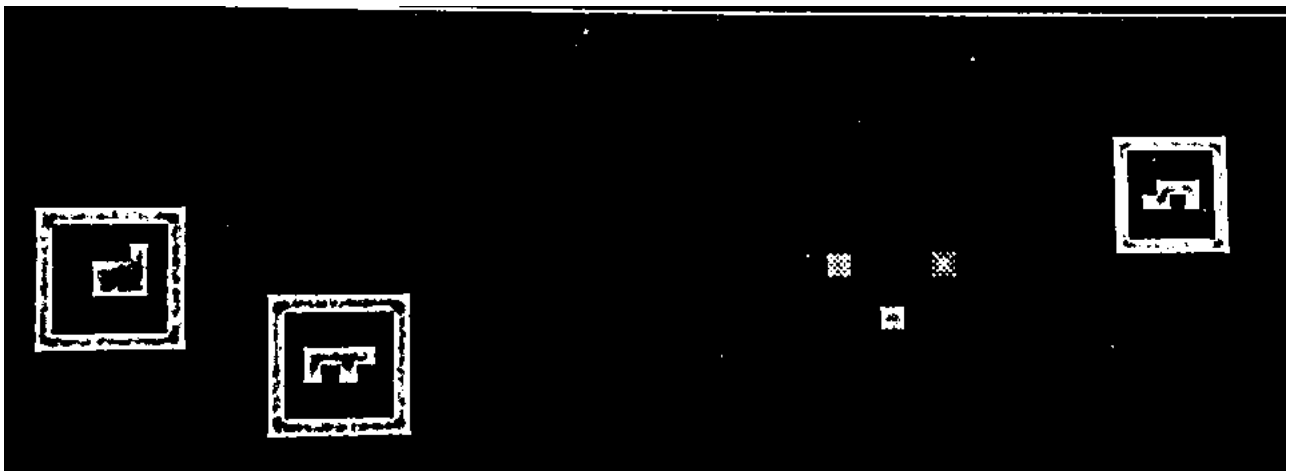


Abbildung 218: Der festgestellte Fehler in der Binarisierung

Dieses Problem trat auf, da die Binarisierung des Bildes mit einem adaptiven Schwellwertverfahren durchgeführt wurde. Dabei wird für jeden Pixel seine „Pixel Nachbarschaft“ betrachtet und daraufhin bestimmt, ob der Pixel im binarisierten Bild weiß oder schwarz sein muss. Abhängig von der Wahl der Parameter des Algorithmus und der Größe eines Markers auf dem Bild konnte so der Fall eintreten, dass für manche Pixel nur die schwarzen Bereiche des Markers als Nachbarschaft betrachtet wurden. Durch Ungenauigkeiten der Kamera oder des ausgedruckten Teststreifens kann ein Pixel somit heller als seine schwarze Umgebung wirken und damit als weiß erkannt werden. So entsteht der Effekt, dass die Ränder der Marker gut getrennt sind (weiße und schwarze Pixel treffen aufeinander) und die schwarzen Flächen innere weiße Flächen erhalten. In Abbildung 218 sind die Farben vertauscht, da der Algorithmus mit einem invertierten Binärbild arbeitet.

Es wurde versucht diesem Problem zu begegnen, indem andere Schwellwertverfahren und andere Parameter getestet wurden. Jedoch zeigte sich, dass es kein Verfahren gab, welches zuverlässig unter allen Bedingungen arbeitet, die beim Einscannen eines Teststreifens eintreten können. Als Alternative wurde beschlossen das Bild nicht in seiner Gesamtheit zu binarisieren und dem SimpleBlob das Bild im BGR-Format zu übergeben. In diesem Fall führt der Algorithmus aus OpenCV[21] eine Konvertierung in Graustufen aus und erstellt mehrere Binärbilder mit unterschiedlichen Schwellwerten. Es kann somit ohne vorherige Binarisierung eine Blob-Detektion durchgeführt werden. Erst wenn die Marker aus dem Originalbild ausgeschnitten werden, wird ein Binärbild mit der OTSU-Methode erstellt. Dies funktioniert gut, da auf den ausgeschnitten Bildern in den meisten Fällen nur das weiße Papier und der schwarze Marker zu sehen sind.

Das Problem, welches mit Hilfe von Abbildung 218 skizziert wurde, konnte mit dieser neuen Methode gelöst werden, jedoch wurden weitere Unzulänglichkeiten des SimpleBlob Algorithmus festgestellt. Da mit diesem Verfahren nur Blobs gefunden werden können, die durch eine zusammenhängende Struktur hervorstehen, sind die Inhalte der Marker limitiert. Ein Marker, der nur aus vielen einzelnen schwarzen Pixeln besteht, kann nur sehr schwer vom Algorithmus detektiert werden. Auch konnten eigentlich erkennbare Marker durch Ungenauigkeiten bei der Aufnahme des Bildes manchmal nicht als zusammenhängende Struktur erkannt werden. In diesen Fällen hätten die folgenden Schritte der Bildverarbeitung problemlos funktioniert, wäre die Position des Markers durch den SimpleBlob bekannt gewesen.

Daher wurde nach einer Alternative für die Detektion der Marker gesucht, die weniger vom Inhalt dieser abhängig ist. Eine Kantendetektion mit dem **Canny-Algorithmus**[2] stellte sich nach einer Evaluation verschiedener anderer Verfahren als vielversprechend heraus. Mit Hilfe der Kantendetektion kann die schwarze Umrandung des Markers sehr zuverlässig erkannt werden. Die Detektion der Marker wurde im Bildverarbeitungsalgorithmus auf die Kantendetektion umgestellt und es ergab sich der Ablauf, der in Abbildung 219 auf der nächsten Seite dargestellt ist.

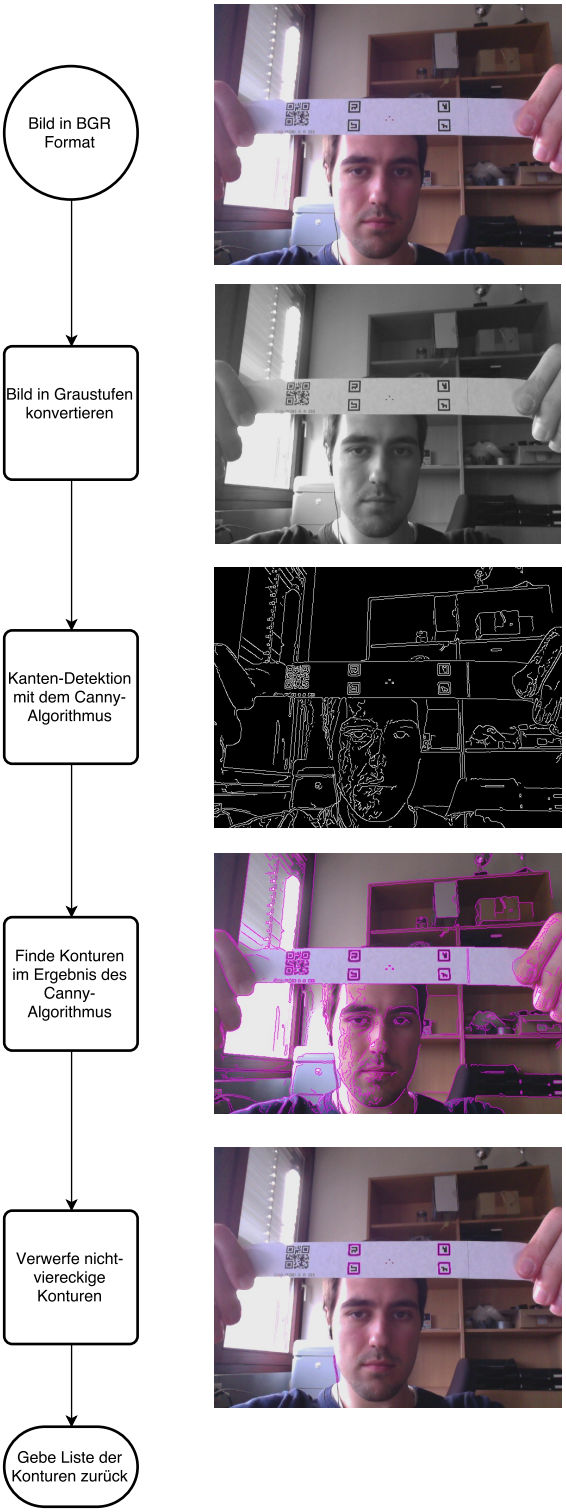



Abbildung 219: Die Marker Detektion im Algorithmus

Wie in Abbildung 219 auf der vorherigen Seite zu sehen ist, kann der Canny-Algorithmus sehr robust alle Kanten im Bild erkennen, was die Umrandung der Marker beinhaltet. Danach werden mit Hilfe von weiteren OpenCV Funktionen zusammenhängende Kanten zu einer Kontur zusammengefasst. Diese Konturen werden mit dem **Douglas-Peucker-Algorithmus**[23] geglättet, wodurch die Konturen von Markern nur noch vier Eckpunkte haben. So können Konturen mit zu vielen oder zu wenigen Eckpunkten verworfen werden, wodurch Berechnungszeit gespart wird.

Nachdem die Position und Form der (potentiellen) Marker bekannt ist, können diese aus dem Originalbild ausgeschnitten werden. Der Vorgang ist in Abbildung 220 auf der nächsten Seite beschrieben. Wichtig ist hierbei, dass der Marker mit ausreichend Rand ausgeschnitten wird, damit die eigentliche Umrandung des Markers bei jeder Rotation des Teststreifens komplett sichtbar ist. Das ausgeschnittene Bild wird im Anschluss mit der OTSU-Methode [24] binarisiert. Da, wie in der Abbildung zu sehen, hauptsächlich der Teststreifen mit dem Marker erkennbar ist, kann ein korrektes Binärbild erstellt werden, auf dem der Marker hervorsticht. Auf diesem Bild wird nun erneut nach Konturen gesucht, jedoch ohne vorher den **Canny-Algorithmus** durchzuführen. So werden die Eckpunkte der Marker präziser gefunden. Es wird angenommen, dass die größte Kontur die des Markers ist. Diese wird dann ausgewählt. Mit Hilfe der Kontur können der Mittelpunkt des Markers und seine Eckpunkte berechnet werden. Diese können genutzt werden, um den Marker in eine quadratische Form zu bringen, damit dieser mit den Referenzmarkern verglichen werden kann, die spezifisch für den Teststreifen definiert sind.

Bislang konnte der Bilderverarbeitungsalgorithmus einen Teststreifen nur erfolgreich erkennen, wenn alle Marker auf dem Streifen erfolgreich gefunden wurden. Theoretisch wäre es aber auch möglich mit nur drei gefundenen Markern die Position des Teststreifens auf dem Bild zu berechnen. Der Algorithmus wurde in Folge dessen so abgeändert, dass dieser ein Bild mit mindestens drei gefundenen Markern als detektiert betrachtet, unabhängig davon wie viele Marker tatsächlich auf dem Teststreifen sind. Alle zusätzlichen Marker sind somit optional und dienen nur der Verbesserung der Genauigkeit des Ergebnisses.

Während verschiedener Tests fiel auf, dass die Position der Farbumschläge nicht mehr richtig berechnet wird, wenn der Teststreifen mit einer Verkippung aufgenommen wird. Dies ließ sich auf einen Fehler im bisher verwendeten mathematischen Modell zurückführen, das für die Berechnung der Positionen der Farbumschlagsfelder genutzt wird. Es wurde versucht den Fehler im Modell zu korrigieren, jedoch konnte keine Lösung gefunden werden, die für alle Rotationen und Verkippungen ein korrektes Ergebnis liefert.

Um dennoch eine Aussage über die Erkrankung eines Probanden zu treffen, wurde eine neue Lösung konzipiert, die sich auf Funktionen stützt, die bereits in OpenCV implementiert wird. In Abbildung 221 auf Seite 315 ist der neue Ablauf dargestellt. So werden im ersten Schritt die Eckpunkte der Marker und deren Zentren auf ihre Position im Originalbild zurückgerechnet. Mit diesen Punkten kann anschließend mittels einer Funktion aus OpenCV eine Bildtransformation durchgeführt werden, die das Bild in die gleiche Form bringt, wie sie der Teststreifengenerator ausgibt. Somit hat das Bild nun die gleiche Größe wie die Bilder des Teststreifengenerators und die Positionen der Farbumschlagsfelder können anhand ihrer Pixel-Koordinaten bestimmt werden. In einer ersten Version wurden für diesen Vorgang nur die Zentren der Marker verwendet, dies führte aber zu Problemen, wenn die gefundenen Marker z.B. in einer horizontalen oder vertikalen Linie angeordnet waren. Das transformierte Bild wies dann erhebliche Fehler auf. Da für jeden einzelnen Marker nun fünf Positionen zur Verfügung stehen und eine Anzahl von drei unterschiedlichen Markern gefordert ist, sind somit insgesamt 15 Referenzpunkte und damit genug Informationen für eine genaue Transformation vorhanden.

Aus dem transformiertem Bild werden dann die Referenz- und Farbumschlagsfelder ausgeschnitten. Um Ungenauigkeiten der Kamera oder der Transformation vorzubeugen, wird auch ein zusätzlicher Rand ausgeschnitten. Danach werden die einzelnen Bilder mit der OTSU-Methode binarisiert. Dieses Binärbild dient als Maske, um im farbigen Bild den Farb-Mittelwert zu extrahieren. Vor der Extrak-

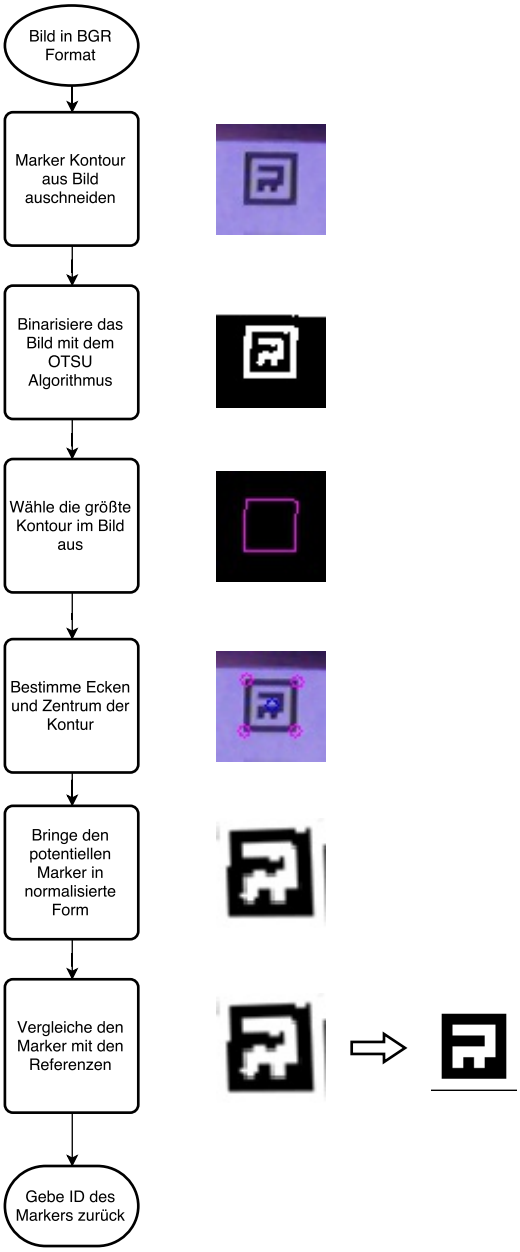



Abbildung 220: Validierung eines Markers im Algorithmus

tion werden die Bilder in den HSV-Farbraum konvertiert. Die Differenzen der einzelnen Farbkanäle werden addiert und dieses Ergebnis zur Einschätzung des Teststreifen-Ergebnisses genutzt. Eine hohe Differenz stellt einen negativen Test dar, eine geringe Differenz einen positiven Test.

Bei verwackelten Bildern, oder bei Bildern in denen der Fokus nicht korrekt ist, kam es jedoch zu Problemen mit diesem Ansatz, denn dort war oft auch das Referenzfeld verwaschen und hob sich nicht deutlich genug vom weißen Hintergrund des Teststreifens ab. Um dieses Problem zu vermeiden wird für das Referenzfeld nicht nur der Mittelwert der eigentlichen Farbe berechnet, sondern auch der Mittelwert des Hintergrundes. Ist die Differenz zwischen diesen beiden Werten nicht groß genug kann davon ausgegangen werden, dass das Bild verwackelt ist und dem Ergebnis nicht vertraut werden kann. In diesem Fall wird das Bild so behandelt, als wären keine Marker detektiert worden. Diese Idee verhindert falsche Ergebnisse durch geringe Bildqualität.

Als letztes wurde eine Verbesserung in der **Android-App** und der **iOS-App** vorgenommen, um die Robustheit des Ergebnisses weiter zu erhöhen. So wird nicht mehr nur das erste Bild, auf dem der Teststreifen erkannt wurde, als Ergebnis genommen, sondern es wird nach dem ersten erkannten Bild eine weitere Sekunde gewartet, in der das Video der Kamera weiter ausgewertet wird. So stehen der App mehrere Ergebnisse zur Verfügung, aus der dann das Ergebnis ausgewählt wird, welches am häufigsten auftritt. So werden Ausreißer eliminiert und nicht als Ergebnis angezeigt.

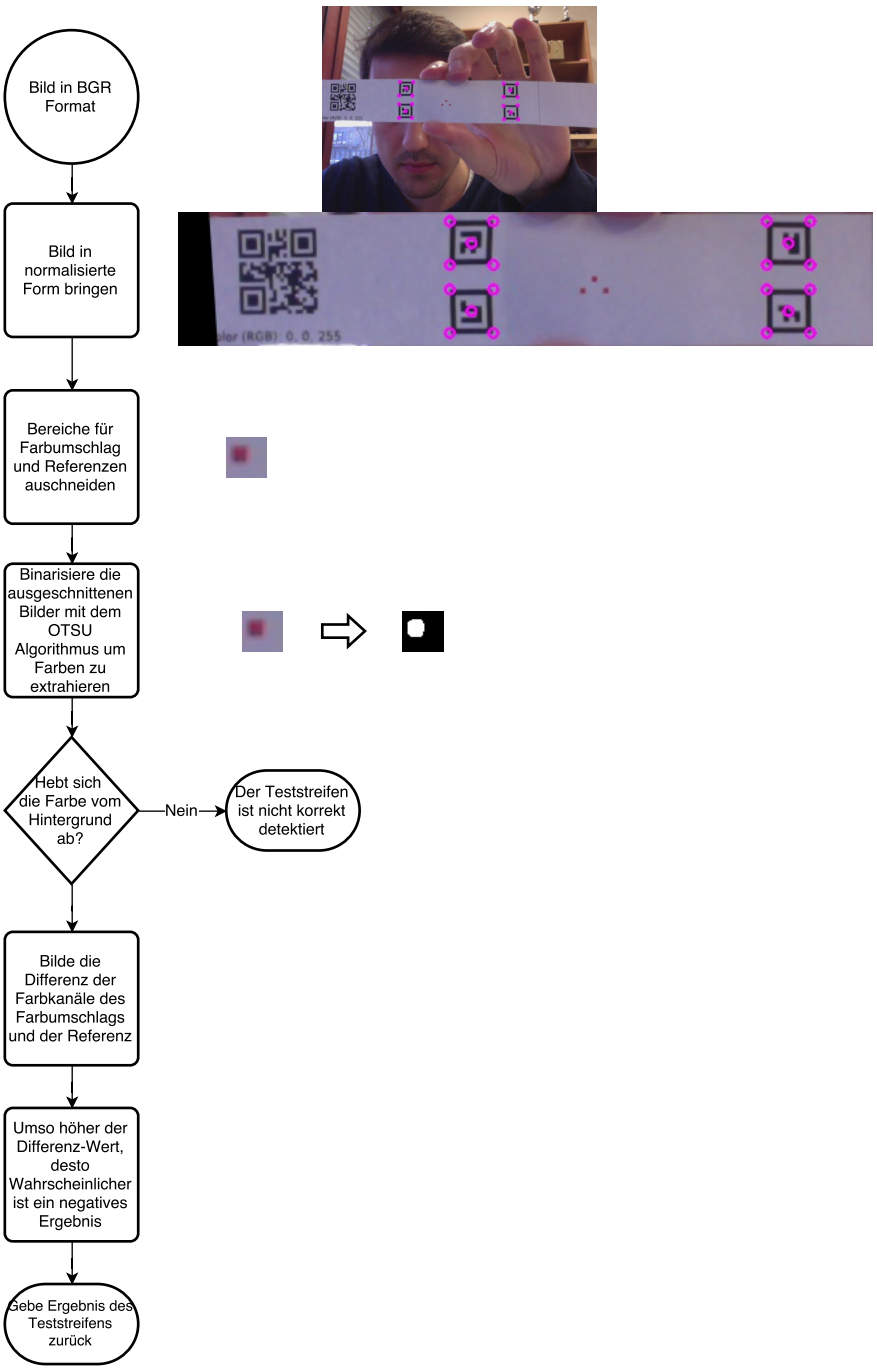



Abbildung 221: Bestimmung des Teststreifen Ergebnisses

**Technologien**

**OpenCV** Bibliothek für die Bildverarbeitung

**Teststand & Testframework** Zum Evaluieren des Algorithmus

**Systembeschreibung** Die Änderungen am Code für die Bildverarbeitung wurden unter Windows 10, Linux und OSX vorgenommen und getestet. Unter Windows und Linux wurde Eclipse Mars als IDE eingesetzt und unter OSX Xcode 7.2. Die Anpassungen für die Android-App wurde unter Linux mit Android-Studio 1.5 und die Änderungen der iOS App unter OSX mit Xcode 7.2 durchgeführt. Die Videos wurden mit den entsprechenden Testgeräten für die definierten Einstellungen (Winkel, Licht, Abstand) im Teststand aufgenommen.

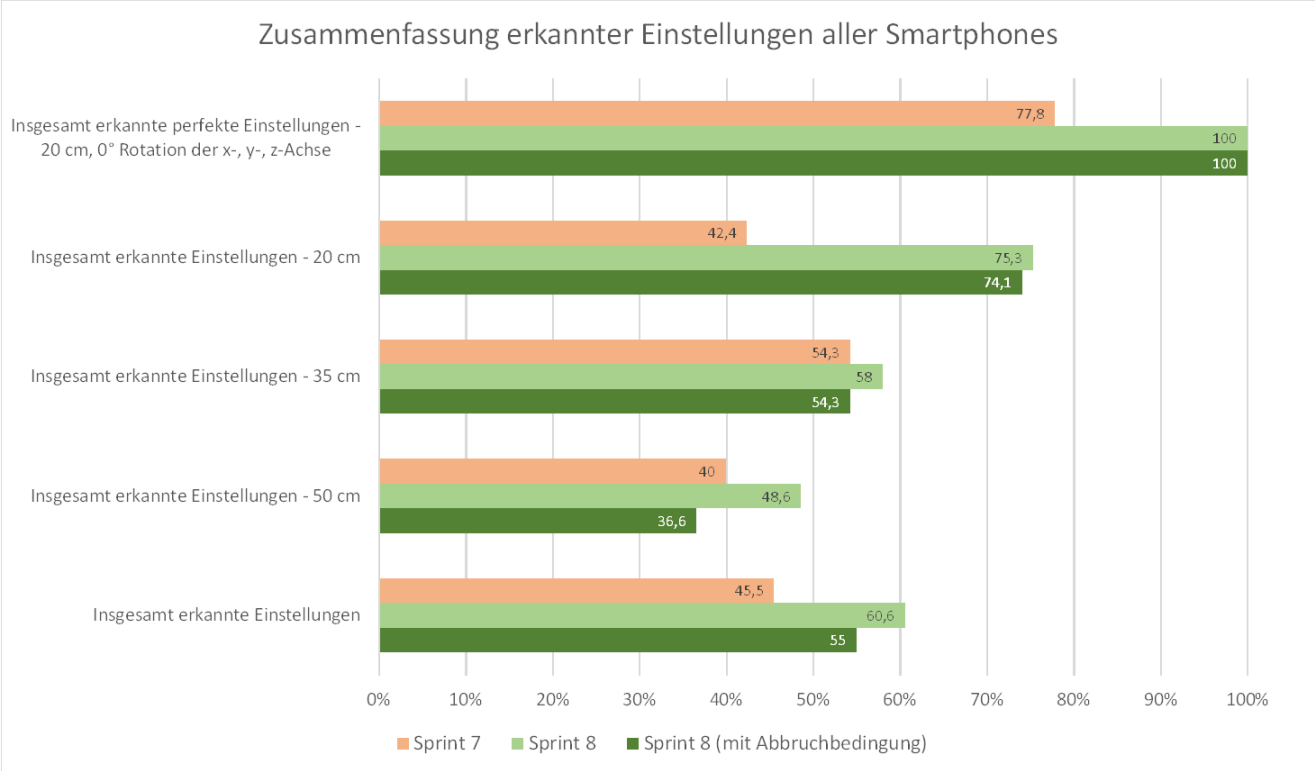



Abbildung 222: Zusammenfassung der Evaluation

**Evaluation** Um zu überprüfen, ob mit diesem Arbeitspaket eine Verbesserung des Algorithmus erzielt werden konnte, wird ein Vergleich zu der vorherigen Version gezogen. Bei diesem Vergleich soll getestet werden, unter welchen Parametern der Algorithmus einen Teststreifen anhand seiner Marker erfolgreich in einem Bild detektieren kann. Da in der Version aus Sprint 8 der Algorithmus ein Bild als „nicht detektiert“ bewertet, wenn die Bildqualität nicht gut genug für eine eindeutige Auswertung des Farbumschlags ist, wurde zusätzlich eine modifizierte Version verwendet. So kann die Vergleichbarkeit der beiden Versionen des Algorithmus sichergestellt werden.

Die ausführlichen Tabellen für jedes Smartphone mit allen Einstellungen sind im Anhang dieser Dokumentation zu finden. In diesem Kapitel wird der Fokus auf eine gekürzte Fassung gelegt, die einen besseren Überblick über den Algorithmus geben kann.

Für die Tests wurden mit jedem Smartphone Videos erzeugt, die den Teststreifen unter verschiedenen Abständen und Winkeleinstellungen im Teststand zeigen. Welche Rotation durch welchen Winkel beschrieben wird, ist in Abbildung 224 auf Seite 318 visualisiert. Die x-Rotation gibt beispielsweise die Rotation um die x-Achse herum an. Somit liegen eine Menge von kurzen Videos vor,

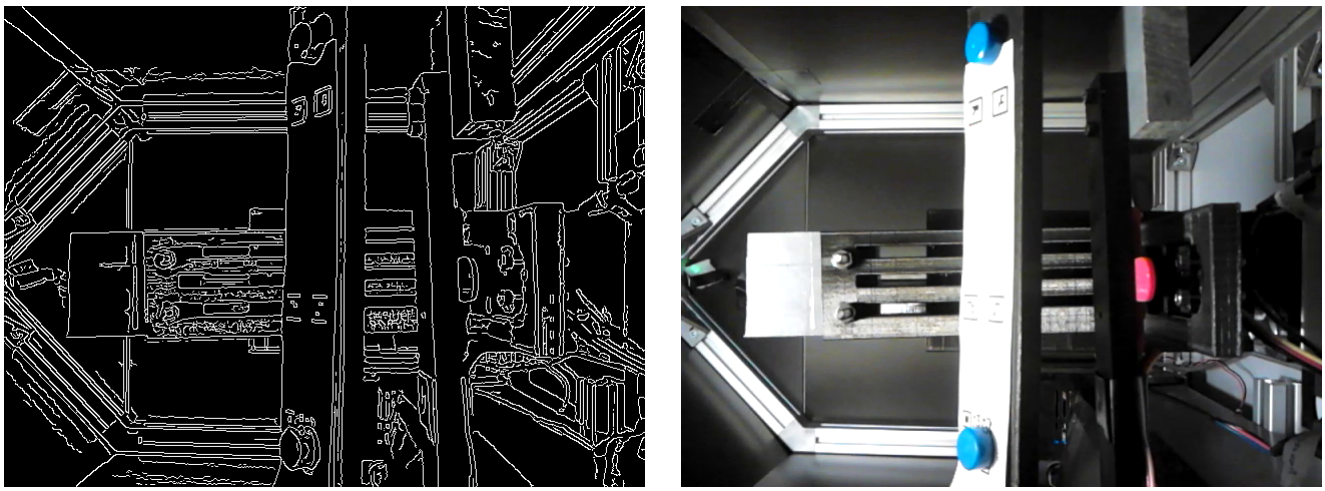


Abbildung 223: Teststreifen mit Ergebnis des Canny-Detektors (links) und Eingabe (rechts)

auf denen jeweils eine Konfiguration zu sehen ist. Mit Hilfe des Testframeworks wurden anschließend die Versionen des Algorithmus nach dem vorherigen Sprint und dem aktuellen Sprint miteinander verglichen. Es wurde überprüft, ob der Algorithmus für ein gegebenes Video mindestens einmal die Marker des Teststreifen detektieren und den Algorithmus erfolgreich beenden konnte.

Aus der Abbildung 222 auf der vorherigen Seite ist ersichtlich, dass durch die Anpassungen in diesem Arbeitspaket der Algorithmus eine generelle Verbesserung gegenüber der älteren Version darstellt. Vor allem bei naher Distanz (20cm) ist eine erhebliche Verbesserung der Detektion zu beobachten. Treten bei dieser Distanz nur Verdrehungen von weniger als 10° um die x- und y-Achse auf (vergleiche Abbildung 224 auf der nächsten Seite für eine Visualisierung der Achsen), ist die Detektionsrate 100%. Nur wenn alle Achsen auf ihre maximalen Werte im Teststand eingestellt sind, ist eine Verschlechterung eingetreten. In allen anderen Fällen war die neue Version des Algorithmus besser oder mindestens gleich gut. Insgesamt konnte die Detektionsrate (in allen Einstellungen) von ∼45% auf ∼60% verbessert werden. Die Tabellen 19 auf Seite 425, 20 auf Seite 425 und 21 auf Seite 426 zeigen diese Effekte in einer Zusammenfassung von allen getesteten Handys.

Es konnte ermittelt werden, dass die schlechten Ergebnisse bei hohen Verkippungen auf Überblendung des Teststreifens zurückzuführen sind. In einer solchen Situation scheint das Licht des Teststandes so auf den Teststreifen, dass das Licht reflektiert wird und die schwarze Umrandung der Marker nicht mehr kontinuierlich zu erkennen ist. Das Innere des Markers ist davon nicht immer betroffen, wodurch die vorherige Version des Algorithmus den Marker in manchen Fällen als Blob detektieren kann. Der **Canny-Algorithmus** hingegen kann die Konturen dadurch nicht mehr korrekt bestimmen, wie in Abbildung 223 zu erkennen ist.

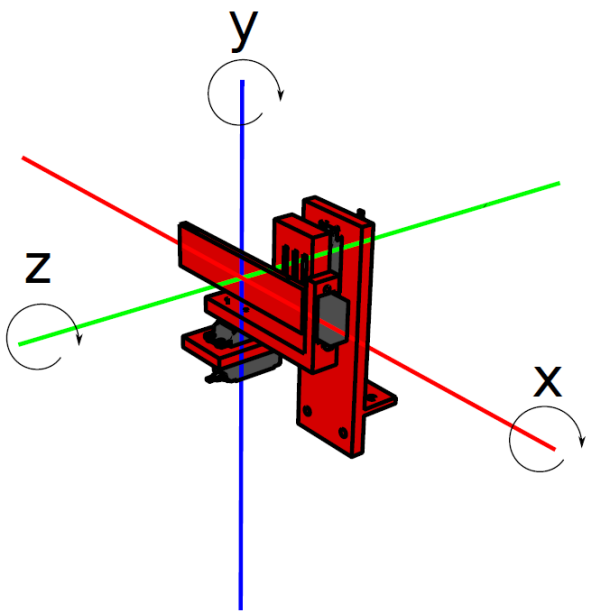



Abbildung 224: Visualisierung der genutzten Rotationen

**Parameter** Der Bildverarbeitungsalgorithmus setzt voraus, dass in einem Bild immer nur ein Teststreifen gleichzeitig zu sehen ist. Des Weiteren muss das Teststreifenschema bekannt sein, um nach den richtigen Markern und den richtigen Positionen der Farbumschläge zu suchen. Auch muss der Farbumschlag von einer hohen Intensität sein, damit er korrekt als positiv erkannt wird. Das Bild darf nicht zu stark verwaschen oder verwackelt sein, da der Algorithmus sonst keine genaue Auswertung geben kann und den Teststreifen als „nicht detektiert“ zurückweist. Zusätzlich darf Licht aus der Umgebung nicht so auf den Teststreifen scheinen, dass Teile der Umrandung oder des Marker-Inhalts reflektieren oder überblendet werden. Falls nur der Farbumschlagsbereich überblendet wird, kann es passieren, dass der Algorithmus eine falsche Aussage über das Testergebnis trifft.

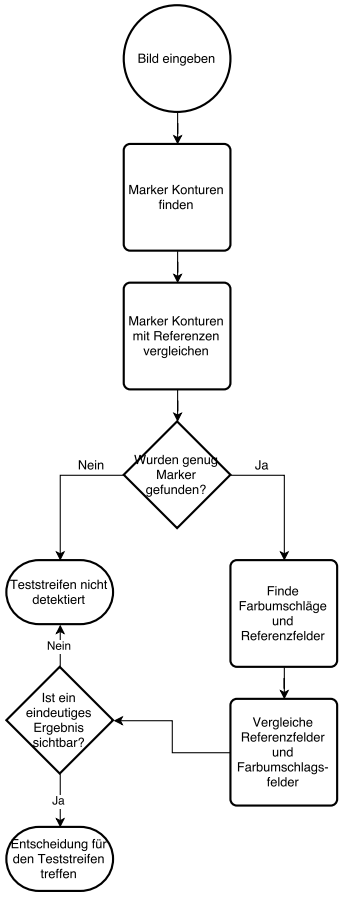



Abbildung 225: Grober Ablauf des Bildverarbeitungsalgorithmus

**Fazit** Zunächst wurde die Detektion der Marker verbessert, indem ein Fehler behoben wurde, durch den die Marker nur in der Hälfte der möglichen Rotationen erkannt werden konnten. Darüber hinaus ist die Grundlage der Markerdetektion nicht mehr Blob-Detektion sondern Kantenerkennung. Das Suchen nach den Konturen der Marker ist schneller und lässt vor allem mehr Spielraum für deren Inhalt. Da im Teststreifengenerator nun mehr als zehn Marker auf einen Teststreifen gelegt werden können, hilft diese Freiheit beim Erstellen neuer Teststreifendesigns, da

der Inhalt der Marker nicht mehr für deren Detektierbarkeit relevant ist.

Neben den Verbesserungen in der Marker Detektion wurde auch die Auswertung der Farbumschläge verändert und liefert nun deutliche Ergebnisse. Die Änderungen konnten meist mit in OpenCV bereitgestellten Funktionen durchgeführt werden, wodurch die Übersichtlichkeit und Wartbarkeit des Quellcodes zusätzlich verbessert werden konnte. Der neue Ablauf des Bildverarbeitungsalgorithmus ist in Abbildung 225 auf der vorherigen Seite abschließend zusammengefasst.

Der gesamte Bildverarbeitungsalgorithmus wurde vollständig und funktionsfähig sowohl in die Android-Prototyp-App als auch in die iOS-Prototyp-App integriert. Hierzu wurden die Schnittstellen auf den entsprechenden Betriebssystemen angepasst.

Insgesamt können jetzt mehr Bilder pro Sekunde analysiert werden als noch in Sprint 7, was hauptsächlich durch das Ersetzen der Blob-Detektion durch Kantenerkennung ermöglicht wurde. So schaffen es die Smartphones, bis zu zehn Bilder pro Sekunde zu analysieren, wohingegen es im vorherigen Sprint höchstens sieben waren. Die schwächeren Smartphones schaffen dabei meistens genauso viele Bilder wie ihre stärkeren Konkurrenten, da sie zwar schlechtere Prozessoren haben, aber auch eine geringere Kameraauflösung, wodurch die Unterschiede sich annähernd ausgleichen.

Der neue Algorithmus ermöglicht sogar den leistungsschwächsten Smartphones eine erfolgreiche Teststreifenanalyse. Das **Sony Xperia E1** und das **Sony Xperia Tipo** konnten zwar in den Tests mit der vorherigen Version des Algorithmus keine Teststreifen erfolgreich detektieren (vergleiche Abbildung 40 auf Seite 329 und Abbildung 43 auf Seite 330), sind aber mit der neuen Version dazu in der Lage. Auch das **Google Nexus 4** mit stärkerem Prozessor und höher auflösender Kamera kann nun in deutlich mehr Fällen den Teststreifen detektieren (vergleiche Abbildung 28 auf Seite 323 mit vorheriger Detektionsrate von 25% und jetzigen 89% bei 20cm Abstand). Genauso kann das **iPad Mini 2** nun über doppelt so viele Konfigurationen (Abstände und Winkel) erkennen, als noch im letzten Sprint.

**Ausblick** Die Auswertung der Farbumschlagsbereiche kann in Zukunft noch weiter verbessert werden, um auch Farbumschläge mit geringer Intensität korrekt zu detektieren. Des Weiteren müssen die Informationen zur Lokalisierung der benötigten Referenzfelder zum Berechnen des Farbumschlags im kommenden Sprint in die Teststreifenschemata eingebettet werden, damit diese für jeden Teststreifen konfigurierbar sind. Hierfür müssen mehrere Komponenten des gesamten Projekts angepasst werden: der Teststreifengenerator, die Administrationsoberfläche und die Kommunikation der Apps mit dem Server.

**Anhang** Hier eine genaue Auflistung aller Geräte und ihrer Testergebnisse. Alle Aufnahmen wurden bei einer Lichtintensität von 2700 Kelvin im Teststand gemacht.

| iPad mini 2 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 20 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Nein | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 20 cm | 0° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 20 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 0° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 20 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 22: Zusammenfassung des iPad Mini 2 bei 20cm Abstand

| iPad mini 2 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Nein | Ja | Ja |
| | 35 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 0° | Nein | Ja | Ja |
| | 35 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 35 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 35 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 0° | 50° | Nein | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 35 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 50° | Nein | Ja | Ja |
| | 35 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 23: Zusammenfassung des iPad Mini 2 bei 35cm Abstand

| iPad mini 2 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 50 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 50 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 25° | Nein | Ja | Ja |
| | 50 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 25° | Nein | Ja | Ja |
| | 50 cm | 30° | 10° | 25° | Nein | Ja | Ja |
| | 50 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 50° | Nein | Ja | Ja |
| | 50 cm | 30° | 0° | 50° | Nein | Ja | Ja |
| | 50 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 50 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 50 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 24: Zusammenfassung des iPad Mini 2 bei 50cm Abstand

| LG Bello (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Ja | Ja | Ja |

Tabelle 25: Zusammenfassung des LG Bello bei 20cm Abstand

| LG Bello (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 26: Zusammenfassung des LG Bello bei 35cm Abstand

| LG Bello (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 0° | Ja | Ja | Nein |
| | 50 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 0° | Nein | Ja | Nein |
| | 50 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 50 cm | 30° | 25° | 0° | Nein | Ja | Nein |
| | 50 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 25° | Ja | Nein | Nein |
| | 50 cm | 60° | 0° | 25° | Ja | Ja | Nein |
| | 50 cm | 0° | 10° | 25° | Ja | Ja | Nein |
| | 50 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 25° | Ja | Ja | Nein |
| | 50 cm | 0° | 25° | 25° | Ja | Ja | Nein |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Ja | Nein | Nein |
| | 50 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 50° | Ja | Nein | Nein |
| | 50 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 27: Zusammenfassung des LG Bello bei 50cm Abstand

| Google Nexus 4 | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Ja | Ja | Ja |

Tabelle 28: Zusammenfassung des Google Nexus 4 bei 20cm Abstand

| Google Nexus 4 | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Nein | Ja | Ja |
| | 35 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Ja | Ja | Ja |

Tabelle 29: Zusammenfassung des Google Nexus 4 bei 35cm Abstand

| Google Nexus 4 | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 0° | Ja | Ja | Nein |
| | 50 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 25° | Nein | Ja | Ja |
| | 50 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 25° | Ja | Ja | Nein |
| | 50 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Ja | Nein | Nein |

Tabelle 30: Zusammenfassung des Google Nexus 4 bei 50cm Abstand

| Motorolla Moto G (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 31: Zusammenfassung des Motorola Moto G bei 20cm Abstand

| Motorola Moto G (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Nein | Ja | Nein |
| | 35 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 0° | Nein | Ja | Nein |
| | 35 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 0° | Nein | Ja | Ja |
| | 35 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Nein | Ja | Nein |
| | 35 cm | 60° | 25° | 0° | Nein | Ja | Ja |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Nein | Ja | Ja |
| | 35 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Nein | Ja | Nein |
| | 35 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 32: Zusammenfassung des Motorola Moto G bei 35cm Abstand

| Motorola Moto G (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Nein | Ja | Ja |
| | 50 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 0° | Nein | Ja | Ja |
| | 50 cm | 0° | 10° | 0° | Nein | Ja | Ja |
| | 50 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 25° | Nein | Ja | Ja |
| | 50 cm | 30° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 33: Zusammenfassung des Motorola Moto G bei 50cm Abstand

| Oneplus One (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 34: Zusammenfassung des OnePlus One bei 20cm Abstand

| Oneplus One (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 0° | Nein | Ja | Ja |
| | 35 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 35: Zusammenfassung des OnePlus One bei 35cm Abstand

| Oneplus One (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 0° | Ja | Ja | Nein |
| | 50 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 0° | Nein | Ja | Nein |
| | 50 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 50 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 25° | Nein | Ja | Ja |
| | 50 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 25° | Nein | Ja | Nein |
| | 50 cm | 60° | 10° | 25° | Nein | Ja | Ja |
| | 50 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 25° | Nein | Ja | Nein |
| | 50 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Nein | Ja | Ja |
| | 50 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 36: Zusammenfassung des OnePlus One bei 50cm Abstand

| Samsung Galaxy S5 Neo (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Ja | Nein | Nein |

Tabelle 37: Zusammenfassung des Samsung Galaxy S5 Neo bei 20cm Abstand

| Samsung Galaxy S5 Neo (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 0° | Nein | Ja | Ja |
| | 35 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 35 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Ja | Nein | Nein |

Tabelle 38: Zusammenfassung des Samsung Galaxy S5 Neo bei 35cm Abstand

| Samsung Galaxy S5 Neo (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 0° | Nein | Ja | Ja |
| | 50 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 50 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 50 cm | 60° | 25° | 0° | Nein | Ja | Ja |
| | 50 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 25° | Ja | Ja | Nein |
| | 50 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 25° | Ja | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 50 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 50 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 30° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 50 cm | 0° | 25° | 50° | Nein | Ja | Ja |
| | 50 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 39: Zusammenfassung des Samsung Galaxy S5 Neo bei 50cm Abstand

| Sony Xperia E1 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 20 cm | 0° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 20 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 20 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 20 cm | 0° | 0° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Nein | Ja | Nein |
| | 20 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 20 cm | 0° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 20 cm | 0° | 25° | 25° | Nein | Ja | Nein |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 0° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Nein |
| | 20 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 20 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 40: Zusammenfassung des Sony Xperia E1 bei 20cm Abstand

| Sony Xperia E1 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 30° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 41: Zusammenfassung des Sony Xperia E1 bei 35cm Abstand

| Sony Xperia E1 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 42: Zusammenfassung des Sony Xperia E1 bei 50cm Abstand

| Sony Xperia Tipo (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Nein | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Nein | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Nein | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 43: Zusammenfassung des Sony Xperia Tipo bei 20cm Abstand

| Sony Xperia Tipo (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 35 cm | 0° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 30° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 0° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 35 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 35 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 44: Zusammenfassung des Sony Xperia Tipo bei 35cm Abstand

| Sony Xperia Tipo (Low-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 0° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 0° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 45: Zusammenfassung des Sony Xperia Tipo bei 50cm Abstand

| Sony Xperia Z3 Compact (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 20 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 20 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja | Ja |
| | 20 cm | 60° | 10° | 50° | Ja | Ja | Ja |
| | 20 cm | 0° | 25° | 50° | Ja | Ja | Ja |
| | 20 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Ja | Nein | Nein |

Tabelle 46: Zusammenfassung des Sony Xperia Z3 Compact bei 20cm Abstand

| Sony Xperia Z3 Compact (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 60° | 0° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 0° | Ja | Ja | Nein |
| | 35 cm | 60° | 10° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 0° | Ja | Ja | Nein |
| | 35 cm | 60° | 25° | 0° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 25° | Ja | Ja | Nein |
| | 35 cm | 60° | 0° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 10° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 35 cm | 60° | 25° | 25° | Ja | Ja | Ja |
| | 35 cm | 0° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 35 cm | 60° | 0° | 50° | Ja | Ja | Ja |
| | 35 cm | 0° | 10° | 50° | Ja | Ja | Ja |
| | 35 cm | 30° | 10° | 50° | Ja | Ja | Nein |
| | 35 cm | 60° | 10° | 50° | Ja | Ja | Nein |
| | 35 cm | 0° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 35 cm | 60° | 25° | 50° | Ja | Nein | Nein |

Tabelle 47: Zusammenfassung des Sony Xperia Z3 Compact bei 35cm Abstand

| Sony Xperia Z3 Compact (High-Cost) | Entfernung | Winkel | | | Erkannt? | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | Ja | Ja | Nein |
| | 50 cm | 30° | 0° | 0° | Ja | Ja | Ja |
| | 50 cm | 60° | 0° | 0° | Ja | Ja | Nein |
| | 50 cm | 0° | 10° | 0° | Ja | Ja | Nein |
| | 50 cm | 30° | 10° | 0° | Nein | Ja | Nein |
| | 50 cm | 60° | 10° | 0° | Ja | Ja | Nein |
| | 50 cm | 0° | 25° | 0° | Ja | Ja | Ja |
| | 50 cm | 30° | 25° | 0° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 0° | Ja | Ja | Nein |
| | 50 cm | 0° | 0° | 25° | Ja | Ja | Nein |
| | 50 cm | 30° | 0° | 25° | Ja | Ja | Nein |
| | 50 cm | 60° | 0° | 25° | Ja | Ja | Nein |
| | 50 cm | 0° | 10° | 25° | Ja | Ja | Nein |
| | 50 cm | 30° | 10° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 10° | 25° | Ja | Ja | Nein |
| | 50 cm | 0° | 25° | 25° | Ja | Ja | Nein |
| | 50 cm | 30° | 25° | 25° | Nein | Nein | Nein |
| | 50 cm | 60° | 25° | 25° | Nein | Ja | Nein |
| | 50 cm | 0° | 0° | 50° | Ja | Ja | Nein |
| | 50 cm | 30° | 0° | 50° | Nein | Nein | Nein |
| | 50 cm | 60° | 0° | 50° | Ja | Nein | Nein |
| | 50 cm | 0° | 10° | 50° | Nein | Nein | Nein |
| | 50 cm | 30° | 10° | 50° | Ja | Nein | Nein |
| | 50 cm | 60° | 10° | 50° | Nein | Ja | Nein |
| | 50 cm | 0° | 25° | 50° | Ja | Nein | Nein |
| | 50 cm | 30° | 25° | 50° | Ja | Nein | Nein |
| | 50 cm | 60° | 25° | 50° | Nein | Nein | Nein |

Tabelle 48: Zusammenfassung des Sony Xperia Z3 Compact bei 50cm Abstand

### 4.7.5 Vergleich der Bildverarbeitungsalgorithmen

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Das Arbeitspaket soll dazu dienen, verschiedene Versionen des Bildverarbeitungsalgorithmus zu untersuchen und zu bewerten.
- Arbeitspaketbeschreibung:
  Das bestehende Testframework aus dem Arbeitspaket „Erstellung eines Testframeworks für die Bildverarbeitung“ aus Sprint 6 soll erweitert werden. Für einen aussagekräftigen Vergleich zweier Versionen des Bildverarbeitungsalgorithmus muss das Testframework eine Menge von Eingabevideos verwalten können, auf denen die Algorithmen evaluiert werden.
  Das Testframework soll die Einzelergebnisse der Evaluationen außerdem persistent speichern können, damit nachträglich Vergleiche zwischen den Versionen des Algorithmus angestellt und Visualisierungen ermöglicht werden.
- Vorbedingungen:
  Der Teststand muss durch das Arbeitspaket „ Erweiterung des Teststands“in ein zuverlässig laufendes System umgesetzt worden sein.
- Nebenbedingungen:
  Für den Vergleich des aktuellen Sprints mit dem vorherigen muss das Arbeitspaket „Stabilisierung der Bildverarbeitung“ die geplanten Änderungen des Algorithmus vollständig umgesetzt haben
- Nachbedingungen:
  Es existiert ein aussagekräftiger Vergleich zwischen den verschiedenen Sprints und es wurde eine Gesamtaussage über den aktuellen Bildverarbeitungsalgorithmus getroffen.

- Aufwand:
  Drei Wochen
- Personen:
  - Sebastian Horwege

**Dokumentation** Dieses Arbeitspaket erweitert das im Arbeitspaket „Erstellung eines Testframeworks für die Bildverarbeitungsalgorithmen“ aus Sprint 6 entstandene Test-Framework (siehe Kaptiel 4.5.7.1 auf Seite 259). In der initialen Version des Testframeworks war es möglich, zwei verschiedene Versionen des Bildverarbeitungsalgorithmus auf Basis eines Eingabevideos miteinander zu vergleichen. Der nächste Schritt in der Entwicklung des Test-frameworks wird in diesem Arbeitspaket ausgeführt: Statt eines einzelnen Eingabevideos soll es möglich sein, zwei Algorithmusversionen über einer beliebigen Menge an Videos zu evaluieren. Das ermöglicht, unabhängigere Aussagen über die Performance, als Robustheit und Geschwindkeit, der Algorithmen. Änderungen am Bildverarbeitungsalgorithmus, die auf Grundlage der Aussagen des Test-frameworks gemacht werden, sollten so weniger dazu neigen, an bestimmte Faktoren überangepasst zu sein.

**Ablauf** Im Arbeitpaket „Erweiterung des Teststandes“ in diesem Sprint (Kapitel 4.7.3 auf Seite 306) ist der für den produktiven Einsatz vorbereitet worden. Den Vergleichen in diesem Arbeitspaket liegt ein Videodatensatz zugrunde, der mithilfe des Teststandes erstellt wurde. Der funktioniert unabhängig vom mobilen Endgerät, das die Bewegung des sich im Inneren befindenden Teststreifens aufnimmt. Diese Entscheidung befreit die mobilen Endgeräte von vielen Einschränkungen, wie vorinstallierten Applikationen, oder Eigenschaften der Hardware. Es bedarf also keiner Kommunikation der Endgeräte mit dem Teststand.

Diese Einfachheit birgt aber gleichzeitig auch eine Verkomplizierung des Aufnahmeprozesses. Denn um den Bedarf an Testvideos decken zu können, werden automatisch beliebig viele Einstellungen vom Teststand nacheinander abgefahren. Statt für jede Einstellung ein einzelnes Video aufzunehmen, entsteht langes Ablaufvideo.

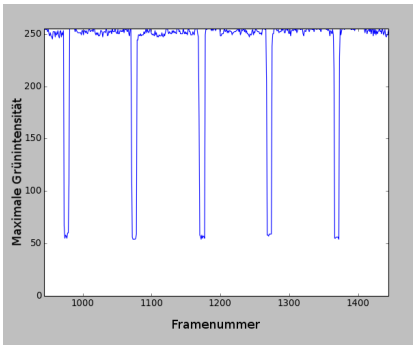



Abbildung 226: Das Ablaufvideo wird an Steigungsänderungen der gemessenen Grünwerte der Indikatorlampe geschnitten.

Um dem Test-Framework unabhängige Eingabevideos für beliebige Orientierungen bereitzustellen, müssen die Ablaufvideos an den richtigen Stellen geschnitten werden. Auch das kann manuell passieren, nimmt aber bei entsprechenden Datenmenge unverhältnismäßig viel Zeit und Aufmerksamkeit in Anspruch. Stattdessen kann durch eine im installierte Indikatorlampe die Schnittposition im Ablaufvideo automatisch detektiert werden. Denn bei jeder neuen Einstellung des Teststandes leuchtet die Indikatorlampe grün auf. Das ensprechende Schnittprogramm misst den Grünwert der Pixel im Video, an dem sich die Indikatorlampe befindet und schneidet das Video an entsprechender

Stelle, wenn die Grünwerte eine festgelegten Steigung überschreiten. Abbildung 226 auf der vorherigen Seite zeigt, wie eindeutig diese Schnittstellen in einem Ablaufvideo zu identifizieren sind. So können zuverlässig Videos mit bekannten Orientierungen von Teststreifen aus dem Ablaufvideo extrahiert werden.

**Technologien**

**InfluxDB** Schemalose Zeitreihendatenbank

**matplotlib** Python Bibliothek zur Visualisierung von Daten[22].

**Git** verteiltes Versionskontrollsystem, das eigentlich in jedem Arbeitspaket verwendet wird, hier aber nochmal aufgeführt werden soll, da es benutzt wurde, um vergangene Versionen der Algorithmen wiederherzustellen.

**Systembeschreibung** Durch das Versionsverwaltungssystem Git ist es möglich vergangene Versionen des Algorithmus wiederherzustellen. Denn zu jeder Änderung am Algorithmus wird eine eindeutige Identifikation hinterlegt, die den Zugriff auf diese Version ermöglicht. Diese Git-Commits bilden die Grundlage für die Vergleiche der Bildverarbeitungsalgorithmen der verschiedenen Sprints. Durch das Test-Framework kann ein beliebiger Commit, also Zustand des Quellcodes untersucht werden. Dafür wird lediglich die Version hinter dem Commit wiederhergestellt und mithilfe der Testvideos evaluiert.

Die Testvideos sind hierarchisch sortiert und stehen auf dem PG-Medic Rechner zur Verfügung. Auf oberster Ebene wird nach Teststandkonfiguration unterschieden. Für jede der Konfigurationen, also einer festen Ablaufsequenz für den Teststand, gibt es eine eigene Menge an Videos. Aus der Struktur der Videodaten ist eindeutig ersichtlich, welches Video welche Teststandeinstellung aufnimmt. Außerdem wird bei den Videos nach Aufnahmegerät und Teststreifenschema unterschieden.

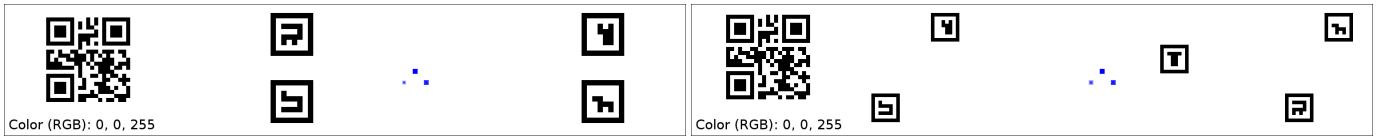
Das Test-Framework kann nun für eine fast beliebige Version des Bildverarbeitungsalgorithmus (siehe Parameter Abschnitt für Einschränkungen) die Performance auf den Eingabevideos untersuchen. Die individuellen Ergebnisse der Untersuchung für jedes Eingabevideo werden schließlich in einer Datenbank mit Verweis auf die Version des Algorithmus gespeichert. Nachfolgend können die gespeicherten Daten in Relation zueinander gesetzt und visualisiert werden.

**Parameter** Das Test-Framework ist von fünf Parametern abhängig. Der Commit bestimmt den Zustand, indem sich der Quellcode des Bildverarbeitungsalgorithmus zu einem festen Zeitpunkt befindet. Hier ist es sinnvoll, einen Commit am Beginn und Ende eines Sprints oder Arbeitspakets zu wählen. Natürlich ist es auch möglich, den Zustand des Quellcodes inmitten eines Arbeitspakets zu evaluieren, um gegebenenfalls die Relevanz einzelner Änderungen zu überprüfen.

Die Distanz und Winkeleinstellungen des Teststands (siehe dazu das Arbeitspaket„Erstellung eines Teststands“ aus Sprint 5) sind wichtige Parameter des Test-frameworks, die bei der Auswertung berücksichtigt werden. Alle folgenden Distanzwerte beziehen sich auf die Parameter des Teststandes. Um die wirkliche Distanz zwischen dem mobilen Endgerät und dem Teststreifen im zu erhalten, muss auf die Werte der Mindestabstand von $17.5cm$ addiert werden. Dadurch, dass der ein nach Außen hin abgeschirmtes System darstellt, und es durch die Steuerungssoftware die Möglichkeit gibt, den reproduzierbar Einstellungen abfahren zu lassen, entsteht ein weiterer Parameter. Die Kameras der mobilen Endgeräte können so unabhängig untersucht werden.

Ein letzter Parameter sind die Teststreifenvarianten (Arbeitspaket „Teststreifengenerator„ aus Sprint 4 (siehe Kapitel 4.3.12.1 auf Seite 129) und “Anpassung des Teststreifengenerators für ein dynamisches Design und unterschiedliche Teststreifvarianten “ aus Sprint 6, Kapitel 4.5.3.1 auf Seite 224). Durch den Teststreifengenerator sind abzählbar unendlich viele Teststreifenvarianten denkbar. Die Teststreifenvariante ist der am schwersten handhabbare Parameter. Im Gegensatz zu den

Winkel- und Distanzeinstellungen, bei denen repräsentative Werte gewählt werden können, um die zu untersuchende Menge einzuschränken, können die Teststreifenvarianten nur manuell erstellt und evaluiert werden. Deshalb kann der Parameterraum nur mit wenigen Stichproben durchlaufen werden, ein gleichmäßiges Abtasten ist aber undenkbar. In Abbildung 227 sind die untersuchten Teststreifenvarianten abgebildet.




(a) Teststreifen 1 (b) Teststreifen 2

Abbildung 227: Die untersuchten Teststreifenvarianten

**Evaluation** Für die Evaluation wurden drei Versionen des Bildverarbeitungsalgorithmus herangezogen, die bis zurück zu Sprint 6 reichen. Ein Vergleich über diesen Zeitraum hinaus ist nicht möglich, da erst mit dem Arbeitpaket „Anpassung der Bildalgorithmen auf dynamische Teststreifen“ in Sprint 6 (siehe Kapitel 4.5.4.1 auf Seite 241) der Umstieg auf dynamische Teststreifen erfolgte. Für Sprint 6 und 7 wurden die jeweils letzten Versionen des Bildverarbeitungsalgorithmus herangezogen. Die Evaluation des Algorithmus im aktuellen Sprint geschah kurz vor Beendigung des Arbeitspakets „Stabilisierung der Bildverarbeitung“ (siehe Kapitel 4.7.4 auf Seite 309).

Die Eingabevideos wurden mit fünf unterschiedlichen mobilen Endgeräten erzeugt, die in Tabelle 228 kurz beschrieben sind.

| Gerät | Beschreibung | Auflösung |
|---|---|---|
| Sony Xperia | Kostengrünstigstes Gerät im Vergleich. Geringste Auflösung. | 640 x 480 |
| LG Bello | Bessere Kamera als Sony Xperia, ansonsten aber mit ähnlicher Leistung | 1920 x 1080 |
| iPhone 4 | Geringere Auflösung, als aktuelle Geräte. Geringster Bildbereich. | 1280 x 720 |
| iPad | Tablet Gerät, gleicher Bildbereich, wie iPhone 4 | 1280 x 720 |
| Samsung Galaxy S5 Neo | Leistungsstärkstes Gerät im Test | 1920 x 1080 |

Abbildung 228: Die mobilen Endgeräte kurz vorgestellt

Insgesamt liegen den folgenden Vergleichen 1311 Videos zugrunde. In jedem Vergleich wird jedoch meistens ein Aspekt isoliert betrachtet, die relevante Teilmenge der Videos wird aber jeweils beschrieben.

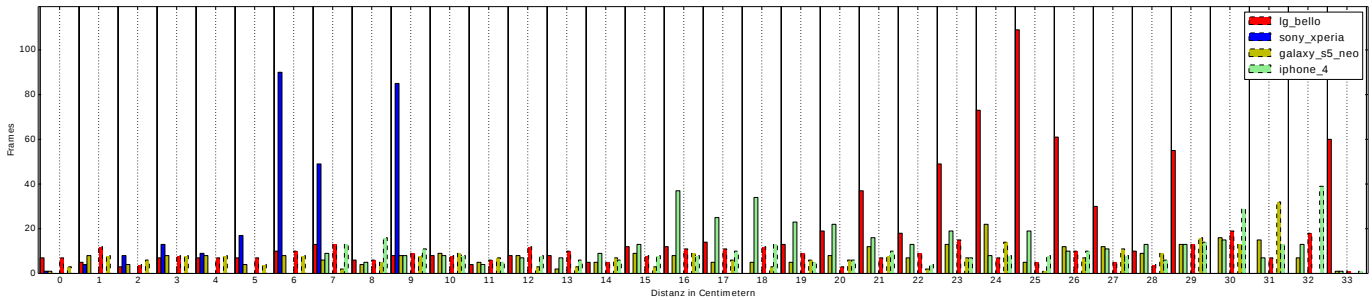


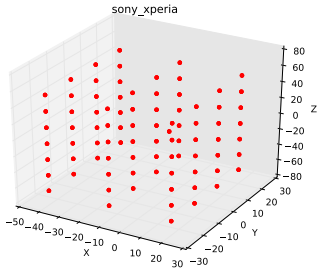

Abbildung 229: Anzahl benötigter Frames bei variierender Distanz, bis Teststreifen erkannt wurde

**Distanz** Abbildung 229 auf der vorherigen Seite zeigt die Performance der Algorithmen aus Sprint 7 und 8. Das Balkendiagramm teilt sich in 34 Einzelbereiche auf, die jeweils für einen Abstand des Teststreifens zur Kamera stehen. Die Winkeleinstellungen der Teststreifenaufhängung wurden nicht variiert, sodass der Teststreifen direkt auf die Kameras gerichtet war. Jeder Bereich ist eindeutig mit durchgezogenen Linien gekennzeichnet. Innerhalb jeden Bereichs sind die Balken links der gestrichelten Linie Sprint 8 und rechts davon Sprint 7 zuzuordnen. Die Farbkodierung ist der Legende zu entnehmen.

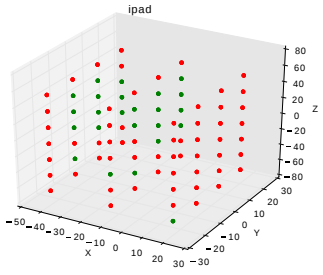
Für das iPhone 4 gibt es im Bereich zwischen 0 und 7 Zentimetern keine Daten, da hier das Sichtfeld der Kamera zu eingeschränkt war und der Teststreifen nicht mehr komplett abgebildet werden konnte.

**Winkel** In Abbildung 230 auf der nächsten Seite werden die Daten in einer weiteren Dimension erkundet. Statt, wie in Abbilung 229 auf der vorherigen Seite, die Distanz bei festen Winkeleinstellungen zu variieren, wurde hier die Distanz auf $15cm$ festgelegt, und die Winkel auf drei Achsen variiert. Ein grüner Punkt im Graphen stellt eine Winkeleinstellung dar, bei der der Teststreifen erkannt wurde. Die roten Punkte im Gegenteil dazu, bilden die Winkel ab, bei denen der Teststreifen nicht detektiert werden konnte. Es fällt auf, dass in den Aufnahmen des Sony Xperia keine Teststreifen gefunden werden konnten, was die These aus obigem Abschnitt stützt: Ab einem gewissen Abstand sind die Marker bei zu geringen Auflösung nicht mehr zu erkennen. Für die restlichen Geräte ist eine eindeutige Verbesserung in Sprint 8 erkennbar.

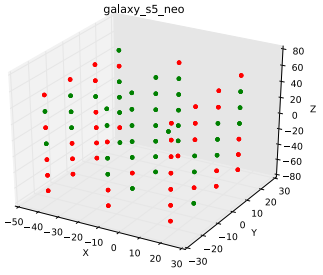
Anders ist dies bei hohen Distanzen, was in Abbildung 229 auf der vorherigen Seite angedeutet werden konnte, ist in Abbildung 231 auf Seite 339 verdeutlicht. Hier ist die Erkennungsrate für ein Gerät und verschiedenen Winkeln bei $0cm$ und $30cm$ Entfernung abgebildet. Es ist erkennbar, dass für $0cm$ Entfernung das Fenster in dem die Winkeleinstellungen variiert werden können stark vergrößert wurde. Bei der Distanz von $30cm$ ist die Performance im neuen Sprint aber eindeutig schlechter. In Tabelle 49 auf Seite 426 kann zwar gezeigt werden, dass diese Verschlechterung die Ausnahme ist, und der Bildverarbeitungsalgorithmus aus Sprint 8 auch für $30cm$ dem aus Sprint 7 überlegen ist, es zeigt sich aber auch, dass der Unterschied zwischen den Sprints bei diesen Distanzen minimal ist.



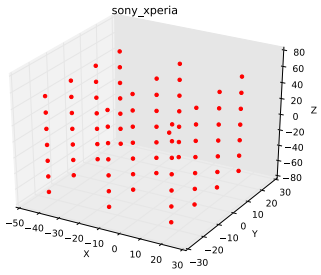

(a) Sprint 7, Sony Xperia



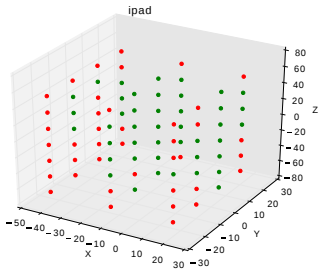

(b) Sprint 7, iPad



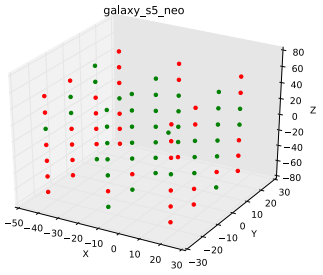

(c) Sprint 7, Galaxy S5 Neo




(d) Sprint 8, Sony Xperia

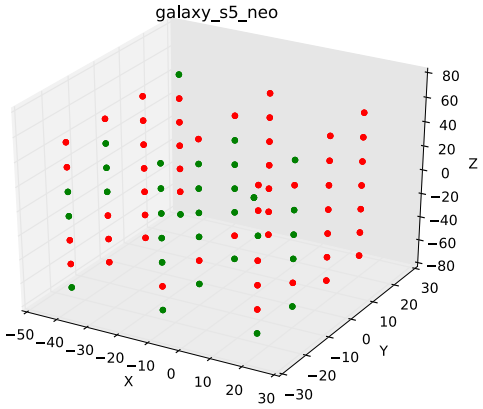



(e) Sprint 8, iPad

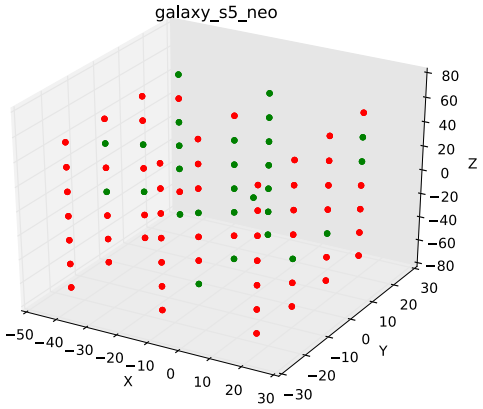



(f) Sprint 8, Galaxy S5 Neo

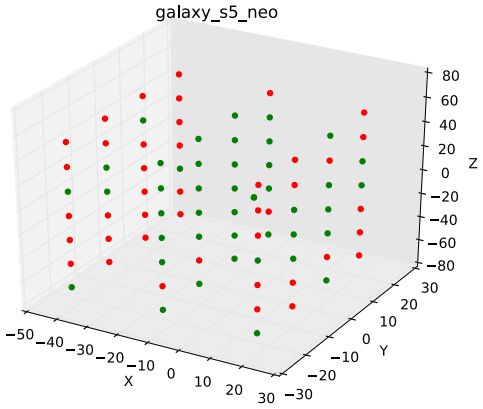
Abbildung 230: Performance der Algorithmen in Sprint 7 und 8 bei verschiedenen Winkeln und 15 cm Entfernung



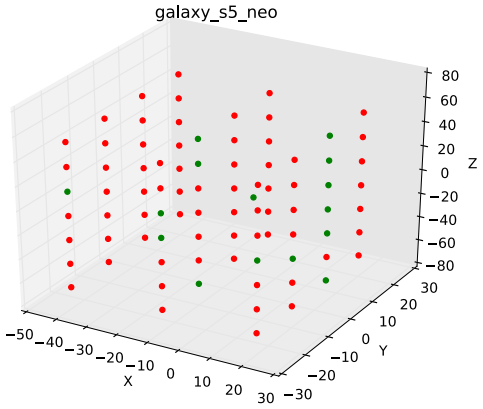

(a) Sprint 7, 0cm Entfernung, Galaxy S5 Neo




(b) Sprint 7, 30cm Entfernung, Galaxy S5 Neo




(c) Sprint 8, 0cm Entfernung, Galaxy S5 Neo




(d) Sprint 8, 30cm Entfernung, Galaxy S5 Neo

Abbildung 231: Performance der Algorithmen in Sprint 7 und 8 für unterschiedliche Entfernungen / Winkel

**Gesamtvergleich** Während der Evaluation ist aufgefallen, dass die Algorithmen in bestimmten Bedingungen nicht fehlerfrei ausgeführt wurden. Das kann unterschiedlichste Gründe haben, auf die an dieser Stelle nicht genauer eingegangen werden soll. Das Test-Framework registriert und speichert diese Fälle aber ab. In 232 ist die Fehlerrate für Sprint 6 und 7 für alle verfügbaren Videos abgebildet. Die Grafik enthält keine Informationen über den aktuellen Sprint, da es hier nie zu unerwarteten Programmabbrüchen kam.

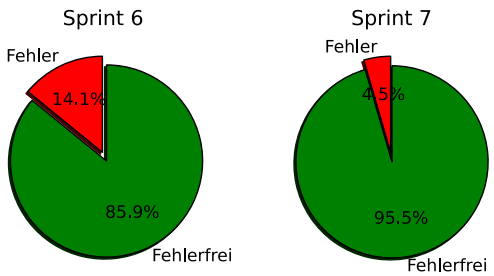



Abbildung 232: Rate der fehlerhaften Programmabbrüche für Sprint 6 und 7

Tabelle 49 auf Seite 426 zeigt die Gesamtperformance über die oben genannte Videomenge. Abgebildet ist der Anteil an Videos, in denen der Teststreifen erkannt wurde. Dabei unterscheidet die Tabelle zwischen den beiden verschiedenen Teststreifenvarianten (siehe Abbildung 336) und un-

terschiedlichen Distanzen im Teststand. Wie oben bereits erwähnt, wird deutlich, dass es in jedem untersuchten Kriterium zu einer Verbesserung des Algorithmus kam. Bei maximaler Distanz ist der Unterschied aber nicht mehr so eindeutig. Auffallend ist die überragende Verbesserung bei der Erkennung des Teststreifen 2. Das Extrem ist hier bei $0cm$ Entfernung, während in Sprint 7 nur in 7% der Videos der Teststreifen erkannt wird, steigt der Anteil in Sprint 8 auf knapp 40%.

Neben der Robustheit des Algorithmus ist als weiteres Ziel die Geschwindigkeit des Algorithmus im Lastenheft festgelegt. In Abbildung 340 ist das Verhältnis der Zeit bis zur erfolgreichen Detektierung eines Teststreifens zwischen Sprint 7 und Sprint 8 abgebildet. Hierfür wurden alle Videos ausgewählt, bei denen die Algortihmen beider Sprints den Teststreifen erkannt haben. Durch das Histogramm, das die Videos in Intervallen der Geschwindkeitsverbesserung akkumuliert, ist nun zu erkennen, dass der aktuelle Algorithmus den Teststreifen in den meisten Fällen in einem Bruchteil der Zeit detektiert, die der alte Algorithmus gebraucht hätte. Die Zeiten wurden auf dem Medic-PG Rechner gemessen, das Verhältnis zwsichen den Algorithmen sollte sich aber auf den mobilen Endgeräten ähnlich verhalten.

**Fazit** Durch das Test-Framework konnte die Verbesserung des Bildverarbeitungsalgorithmus durch das Arbeitspaket „Stabilisierung der Bildverarbeitung“ aus diesem Sprint (Kapitel 4.7.4 auf Seite 309) umfassend untersucht werden.

Die Aussagen des Test-frameworks über die verschiedenen Entwicklungsstadien des Bildverarbeitungsalgorithmus ermöglichen eine Richtung und gegebenenfalls die Steigung der Performanceänderung des Algorithmus zu erkennen. Es sollte aber kein Trugschluss über die Aussagen gemacht werden. Das Test-Framework evaluiert den Algorithmus nur im geschlossenen System des Teststandes. Parameter, wie Kamerabewegungen durch den Nutzer und sich während der Erkennung ändernde Lichtbedingungen, können nicht berücksichtigt werden. Auch bekommt der Algorithmus durch die mobilen Endgeräte nur einen stark zugeschnittenen Bereich des Gesamtbildes als Eingabe (siehe Abbildung 234 auf der nächsten Seite). Dadurch werden große Bildbereiche eliminiert, die den Algorithmus sonst verlangsamen würden.

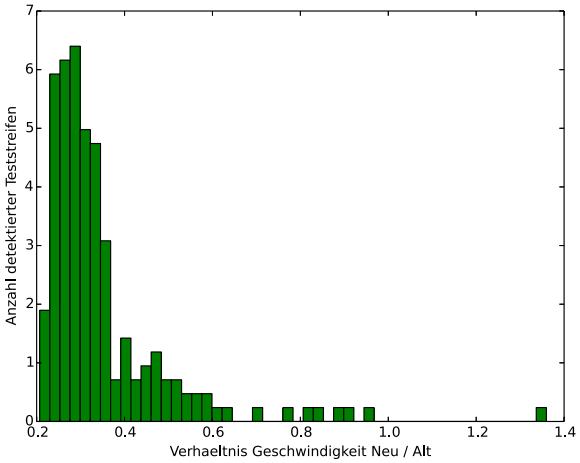
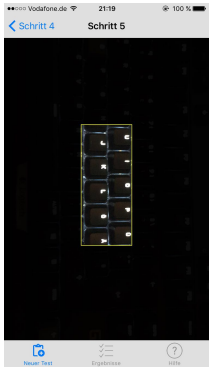
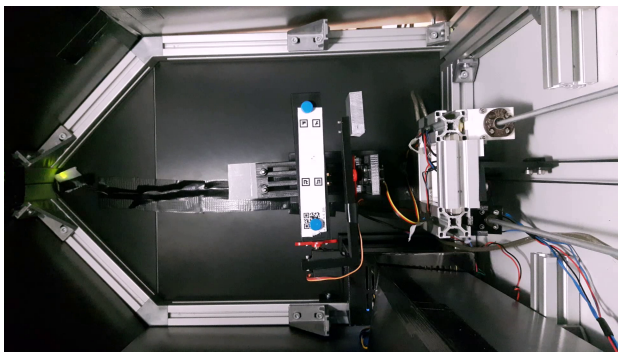



Abbildung 233: Vergleich der Dauer der Teststreifendetektierung in Sprint 7 und 8




(a) Das Eingabevideo in den mobilen Applikationen ist stark zugeschnitten



(b) Das Test-Framework bekommt die unbearbeiteten Videos als Eingabe

Abbildung 234: Videoeingabe für Testframeworkalgorithmus und Applikationsalgorithmus

**Ausblick** Jede weitere Änderung des Bildverarbeitungsalgorithmus kann nun auf den zur Verfügung stehenden Testvideos evaluiert werden. Kommt es im nächsten Sprint wieder zu umfangreichen Veränderungen des Algorithmus, wäre auch die Messung der Performance durch die einzelnen Änderungen denkbar. Damit würde in Zukunft die Gewichtung der Relevanz einzelner Änderungen ermöglicht werden.

### 4.7.6 Vorbereitungen für die Abschlussdokumentation

- Priorität: Normal
- Motivation und Nutzen des Arbeitspaketes:
  Das Arbeitspaket soll dazu dienen eine ordentliche Gliederung für das Abschlussdokument zu erstellen, um so die Qualität der Abschlussdokumentation maßgeblich zu verbessern.
- Arbeitspaketbeschreibung:
  Um die Abschlussdokumentation nicht nur formal, sondern auch inhaltlich gut auszuarbeiten, werden alte Abschlussdokumentation gelesen und sollen somit als Inspiration für das Abschlussdokument der Projektgruppe MEDIC dienen. Am Ende des Arbeitspaketes existiert eine Gliederung für das Abschlussdokument der Projektgruppe MEDIC. Diese muss nicht final sein und kann im Nachhinein weiter angepasst werden. Die Gliederung muss alle wichtigen Punkte enthalten, welche neben den Sprintdokumenten zusätzlich benötigt werden. Zudem können zu den identifizierten Punkten bereits erste Stichpunkte erarbeitet werden um aufzuzeigen welche Themen unter dem entsprechenden Kapitel beschrieben werden sollen.
- Vorbedingungen:
  -
- Nebenbedingungen:
  Als Inspiration können die alten Projektgruppenberichte, welche in der OwnCloud abgelegt sind, herangezogen werden.
- Nachbedingungen:
  Es existiert ein Dokument (im Optimalfall erstellt mit LaTeX), welches die groben Kapitel der Gliederung für die Abschlussdokumentation skizziert. Zudem sind für die dementsprechenden Kapitel bereits erste Stichpunkte niedergeschrieben.
- Aufwand:
  Eine Woche

- Personen:
  - Danny Fonk
  - Christian Sandmann

**Dokumentation** Um die Qualität der Abschlussdokumentation zu gewährleisten, wurde im Zuge dieses Arbeitspaketes eine Gliederung auf Basis der Abschlussdokumentationen vergangener Projektgruppen erstellt.

**Ablauf** Es wurde auf Basis der vorangegangenen Abgaben früherer Projektgruppen eine Literaturrecherche durchgeführt. Die dabei gewonnen Eindrücke wurden festgehalten und aus diesen Eindrücken entstand eine unserer Meinung nach geeignete Struktur für unsere Abschlussdokumentation. Die dabei entstandene Gliederung ist eine lose Vorgabe und kann sich während des Schreibprozesses weiterentwickeln. Nachfolgend ist die Gliederung kurz vorgestellt:

1. Einleitung
   (a) Motivation
   (b) Ziele und Problemstellung
   (c) Aufbau der Dokumentation
2. Projektplanung
   (a) Vorgehensmodell
   (b) Entwicklungsprozesse
   (c) Ressourcenplanung
3. Lastenheft
4. Sprintdokumentationen
5. Produktauslieferungen
   (a) Webfrontend
   (b) Backend
   (c) Teststreifengenerator
   (d) mobile Applikationen
      i. iOS-Applikation
      ii. Android-Applikation
   (e) Installation und Inbetriebnahme des Gesamtsystems
6. Fazit
7. Ausblick
8. Quellenverzeichnis
9. Abkürzungsverzeichnis
10. Glossar
11. Anhang
    (a) Entwicklerhandbücher
    (b) Seminararbeiten

**Technologien** Im Zuge dieses Arbeitspaketes wurden keine neuen Technologien eingesetzt.

**Systembeschreibung** Eine Änderung des bestehenden Systems fand nicht statt.

**Evaluation** Die entstandene Dokumentenstruktur wurde im Nachgang mit bestehenden und abgenommenen Dokumentenstrukturen validiert.

**Parameter** Es entstanden keine neuen Laufzeitparameter jedweder Programme oder Systeme.

**Fazit** Die während dieses Arbeitspaketes entstandene Ordner- und Dokumentenstruktur stellt eine nahtlose Arbeit an der Dokumentation sicher, ist jedoch nur als lose Vorgabe anzusehen.

**Ausblick** Die Ordnerstruktur wird ggf. angepasst werden müssen um den sich ändernden Anforderungen an das Dokument gerecht zu werden.

### 4.7.7 Fazit des Sprints

Wie sich bereits angekündigt hat, war das Projektteam in diesem Sprint aufgrund von Urlaub und Seminararbeiten unvollständig. Daher konnten in diesem Sprint lediglich vier Arbeitspakete umgesetzt werden.
Zum einen wurde der Bildverarbeitungsalgorithmus stark optimiert. Dies zeichnet sich sowohl in einer Verbesserung der Qualität der Ergebnisse, als auch in einer besseren Laufzeit ab. Ein weiteres Arbeitspaket hat sich mit dem Beheben von Fehlern im Teststand befasst, damit dieser auf einer Informationsveranstaltung vorgestellt werden konnte. Außerdem wurde das Testframework um die Möglichkeit, verschiedene Versionen des Bildverarbeitungsalgorithmus miteinander zu vergleichen, erweitert. Zu guter Letzt wurde die Endphase des Projektes thematisch abgesteckt. Dazu wurde die Struktur des Abschlussdokumentes erarbeitet.
Da der Fokus des Sprints auf der Optimierung von bestehenden Systemen lag, konnten keine neuen funktionalen Anforderungen erfüllt werden. Auch die Struktur der Systemkomponenten wurde in diesem Sprint nicht geändert. Somit entspricht diese der Struktur nach Sprint 7. Abbildung 235 auf der nächsten Seite zeigt die Komponenten des Systems erneut, zusammen mit dem Anteil der bisher erfüllten Anforderungen.

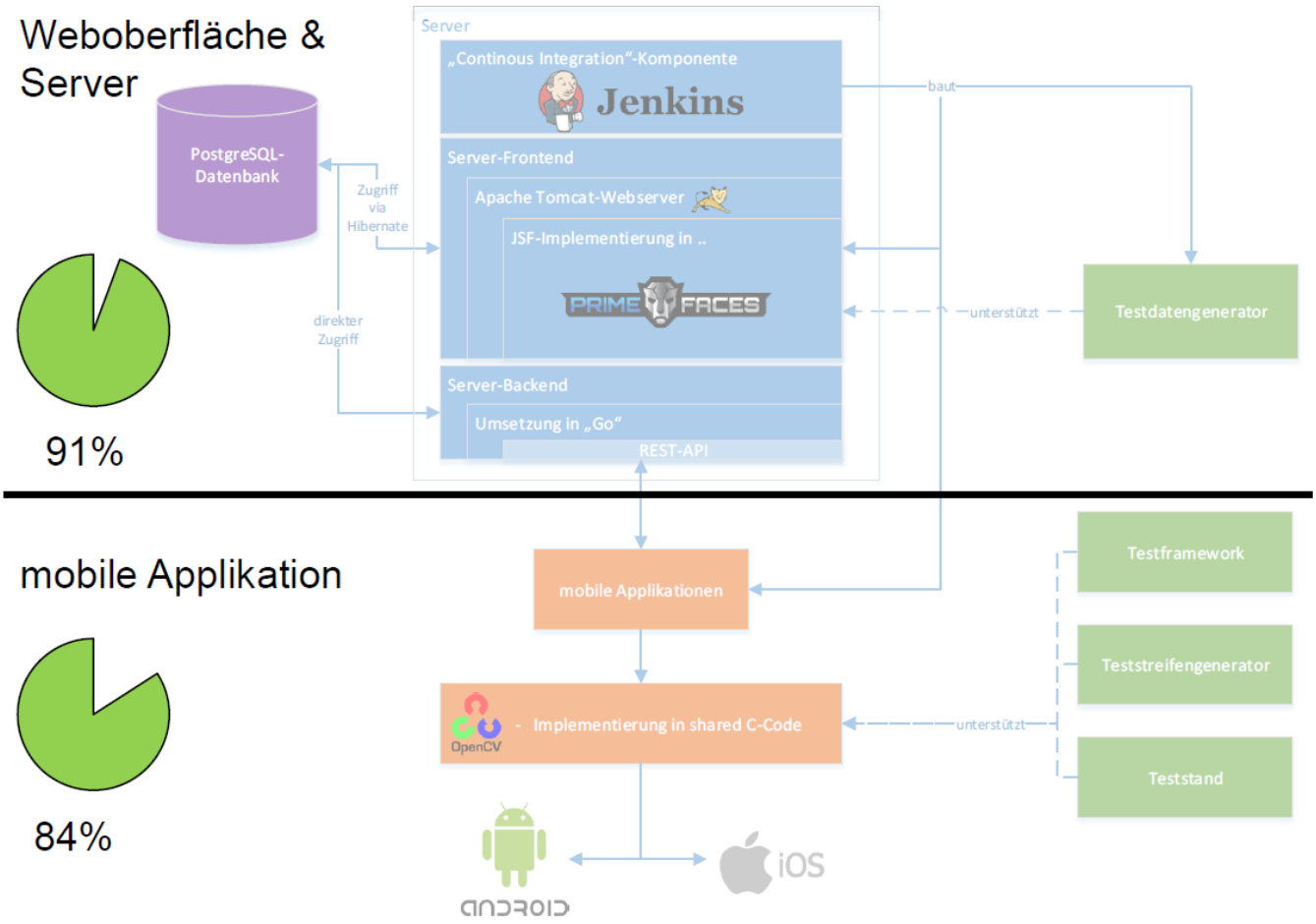



Abbildung 235: Systemkomponenten nach Abschluss von Sprint 8

Neben den in diesem Dokument beschriebenen Arbeitspaketen wurden die folgenden zusätzlichen Workloads abgearbeitet:

- Anpassung des Testframeworks an die neue Schnittstelle des Bildverarbeitungsalgorithmus
- Recherche und teilweise Implementierung von serverseitigen Pushtechnologien

Durch die Konzentration auf Optimierung bestehender Algorithmen in Sprint 8, verschiebt sich die weitere Umsetzung der nichtfunktionalen Anforderungen auf Sprint 9. Diese werden die Usability der mobilen Applikationen sowie der Weboberfläche erheblich verbessern und so ein aus Kundensicht hochwertigeres Produkt schaffen. Im nächsten Sprint muss außerdem die Dokumentation der Produkte finalisiert werden. Diese besteht aus einem ausführlichem Projektbericht, Installationsanleitungen und einem Nutzerleitfaden.

## 4.8 Sprint 9

Das Ziel des neunten und letzten Sprints des Projekts liegt in der Erfüllung aller im Lastenheft definierten Anforderungen. Hierzu wird am Anfang des Sprints von allen Hauptkomponenten des Projektes (Server, Weboberfläche, iOS-App, Android-App, Bildverarbeitung) ermittelt, wo genau noch Verbesserungsbedarf besteht und diese als sogenannte „Issues“ im GitLab gespeichert.

Diese Liste wird während des Sprints iterativ erweitert, wenn neue Probleme bemerkt werden. Daraus ergeben sich die Arbeitspakete „Fertigstellung der Android App“ (Seite 347), „Fertigstellung der iOS App“ (Seite 380), „Fertigstellung des Serverbackends“ (Seite 390), „Fertigstellung des Webfrontends“ (Seite 396) und „Fertigstellung des Teststreifengenerators“ (Seite 414). Neben der Ausbesserung bestehender Probleme wird im Arbeitspaket „Bildverarbeitungsalgorithmus optimieren“ (Seite 353) weiter an der Auswertung des Farbumschlags gearbeitet. Zudem kommt diesen Sprint die Finalisierung der gesamten Projektdokumentation hinzu. Hierzu wird die bereits im letzten Sprint erarbeitete Struktur des Abschlussdokuments als Grundlage für die Umsetzung genutzt.

In Abbildung 236 ist die Systemlandschaft und ein Überblick über den Fortschritt zu Beginn dieses Sprints dargestellt.

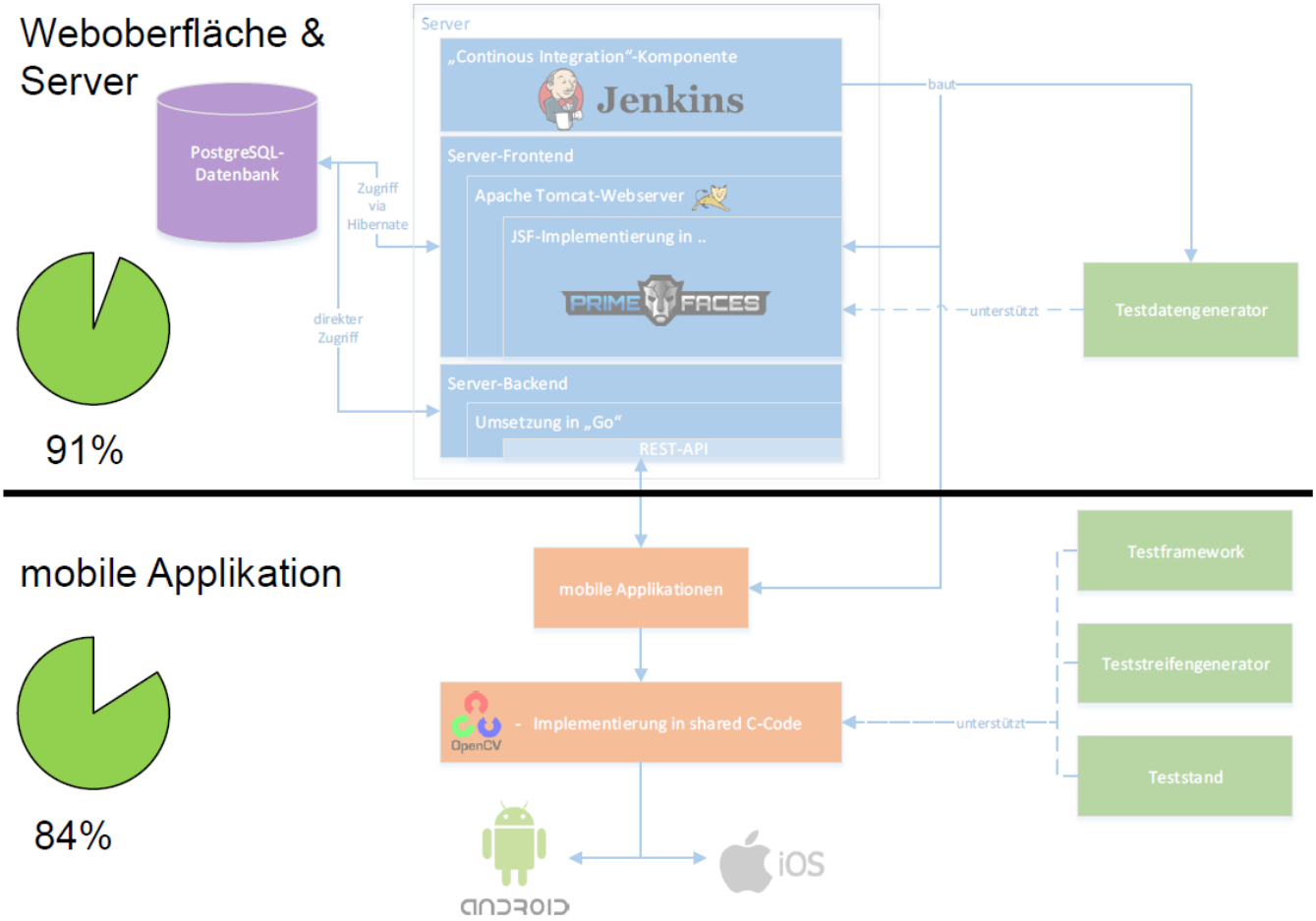



Abbildung 236: Systemarchitektur zu Beginn von Sprint 9

Der Abbildung 236 ist zu entnehmen, dass zu Beginn von Sprint 9 91% der funktionalen Anforderungen im Bereich der Weboberfläche und des Servers umgesetzt sind. Die funktionalen Anforderungen der mobilen Applikation sind zu 84% erfüllt. Da das Ziel dieses Sprints die Erfüllung aller im Lastenheft definierten Anforderungen ist, ist nach Beendigung des Sprints zu erwarten, dass der Gesamtfortschritt bei 100% liegt und das Projekt somit abgeschlossen ist.

### 4.8.1 Sprintplanung

Im Folgenden wird die Planung des neunten Sprints (17.02.2016 bis 31.03.2016, sechs Wochen) genauer aufgezeigt. Dies ist der letzte Sprint der Projektgruppe Medic und schließt das Projekt ab. Dabei wird ein besonderes Augenmerk auf die folgende Punkte gelegt:

- Fokus des Sprints
- Arbeitspakete und Planung der Ressourcen

**4.8.1.1 Fokus des Sprints** Der Fokus für den letzten Sprint liegt auf der Fertigstellung aller Produkte, welche an den Kunden ausgeliefert werden. Zu diesen gehören die Bildverarbeitung, der Teststreifengenerator, die Weboberfläche für Mediziner, die Weboberfläche für Statistiker, die Weboberfläche für Administratoren, das Server-Backend, die iOS Applikation, die Android Applikation, die Abschlussdokumentation, die Abschlusspräsentation. Der Teststand wird ebenfalls an den Kunden ausgeliefert und ist bereits in einem auslieferbaren Zustand. Die Produkte müssen nach dem Ende dieses Sprint alle funktionalen Anforderungen integriert haben. Zudem müssen alle nicht funktionalen Anforderungen abgearbeitet sein was zur Folge hat, dass intensive Tests durchgeführt werden. Diese beinhalten die einzelnen Produkte, als auch das Zusammenspiel aller Systeme (von der Durchführung eines Schnelltests bis zur Diagnose eines Doktors).

**4.8.1.2 Arbeitspakete und Planung** Um die Fertigstellung aller Produkte so effizient wie möglich zu gestalten, wurden Expertengruppen gebildet. Jede dieser Gruppen beschäftigt sich innerhalb dieses Sprints mit einem der oben aufgelisteten Produkten. Die Zuteilung der Gruppen gliedert sich wie folgt:

- **Bildverarbeitung**: Jan Philipp Stubbe und Christian Sandmann
- **Teststreifengenerator**: Danny Fonk
- **Weboberfläche für Mediziner**: Kevin Sandermann, Nicolas Koch, Daniel Wegmann, Raphael Kappes
- **Weboberfläche für Statistiker**: Kevin Sandermann, Nicolas Koch, Daniel Wegmann, Raphael Kappes
- **Weboberfläche für Administratoren**: Kevin Sandermann, Nicolas Koch, Daniel Wegmann, Raphael Kappes
- **Server-Backend**: Sebastian Horwege und Danny Fonk
- **iOS Applikation**: Timo Raß und Timo Schlömer
- **Android Applikation**: Timo Schlömer und Christoph Ressel
- **Abschlussdokumentation**: die gesamte Projektgruppe
- **Abschlusspräsentation**: Danny Fonk und Timo Raß

Um alle notwendigen Arbeitsaufwände zu erfassen, wurde im letzten Sprint mit der Issues-Funktion des GitLab gearbeitet. Es wurden alle anstehenden Aufgaben zu beginn des Sprints gesammelt und als Issue eingetragen. Dies Art der Aufgabenverwaltung wurde auch während des Sprintes weiter fort geführt. Wenn ein Fehler in den Programmen identifiziert oder andere Änderungen vorgenommen werden sollten, wurden neue Issues mit einer entsprechenden Beschreibung erstellt und dem Expertenteam zugewiesen.

Im weiteren Verlauf des Dokumentes werden die Arbeitspakte beschrieben und dessen Dokumentation skizziert.

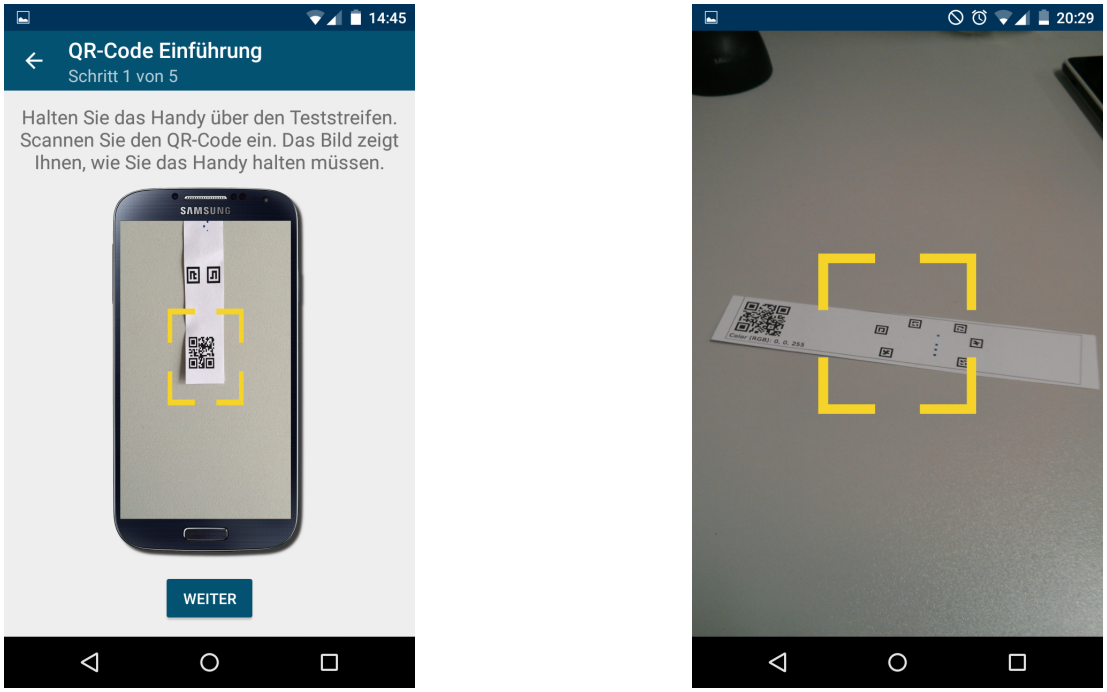
**Arbeitspaketdokumentation** In den folgenden Unterkapiteln werden die einzelnen Arbeitspakete wie gewohnt detailliert beschrieben. Dabei wird jeweils zunächst die Definition des Arbeitspakets mit Aufgabe, angesetzter Zeit und bearbeitenden Personen vorgestellt. Im darauf folgenden Unterkapitel folgt ein Abschlussbericht, der den erreichten Fortschritt zusammenfasst. Dies umfasst auch aufgetretene Probleme, wie das Erreichte evaluiert wurde und welche Technologien verwendet wurden.

### 4.8.2 Fertigstellung der Android-Applikation

- Priorität: Kritisch
- Arbeitspaketbeschreibung:
  Das Hauptaugenmerk dieses Arbeitspaketes liegt auf der Benutzerführung und der grafischen Oberfläche der Android-App. Dabei sollen die durch Timo Raß' Seminararbeit erarbeiteten Konzepte und Erweiterungen in die Android integriert werden.
  - Es muss ein System zum Herunterladen, Speichern und Darstellen von Test-Anleitungen implementiert werden, damit diese dynamisch für jeden Teststreifen offline verwendet werden können
  - Die Accessibility-Verbesserungen aus der Seminararbeit von Timo Raß müssen in die Android-App integriert werden Dazu kann auf die bereits angepasste iOS-App verwiesen werden
  - Der aktuell ausgewählte Tab muss sich visuell stärker vom Hintergrund abheben
  - Noch nicht betrachtete Diagnosen sollen in der Liste der Diagnosen hervorgehoben werden
  - Das Tippen auf die Benachrichtigung bei einer neuen Diagnose soll den Nutzer direkt zur korrekten Ansicht bringen
  - Wenn keine Diagnosen in der Liste dargestellt werden, soll stattdessen ein Platzhalter angezeigt werden
  - Die Platzhalter und Test-Ausgaben auf der **Detailansicht**, der **Ansicht nach Beenden eines Tests** und der **Hilfeseite** müssen durch ästhetische Endversionen ersetzt werden
  - Alle darstellbaren Texte müssen eine deutsche Fassung haben
  - Die Diagnoseliste muss sich nach dem Erstellen eines Tests automatisch aktualisieren
  - Die App muss in der Lage sein einen QR-Code einzuscannen und dessen Inhalt für die Weiterverwendung verfügbar zu machen
  - Der API-Schlüssel der Android-App darf nicht als Klartext im Quellcode gespeichert sein
- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Alle offenen Anforderungen sind in die Applikation integriert und alle Fehler behoben.
- Aufwand:
  6 Wochen
- Personen:
  - Christoph Ressel
  - Timo Schlömer

### Dokumentation

**Ablauf** Das Ziel dieses Arbeitspakets bestand darin, offene Probleme der Android Applikation zu lösen und die Funktionalitäten entsprechend den Anforderungen im Lastenheft zu implementieren. Hierfür sind die bekannten Probleme gesammelt worden, die im späteren Verlauf abgearbeitet und gegebenenfalls um weitere Aufgaben erweitert wurden. Die erste Aufgabe bestand darin, die Testdurchführung entsprechend den Empfehlungen aus Timo Raß' Seminararbeit hinsichtlich Nutzerfreundlichkeit und Einfachheit anzupassen. Hierzu wurde als erstes dafür gesorgt, dass verschiedene Teststreifenvarianten anhand des QR-Codes identifiziert werden können. Dazu wurde eine Kameraansicht zum Scannen von QR-Codes implementiert und in die Testdurchführung eingebettet (vgl. Abbildung 237). Als Bibliothek wurde hier neu die „Google Play Services Vision API Version 8.4.0“ [76] eingebunden. Es wurde sich dagegen entschieden die in früheren Sprints verwendete „ZXing“ Bibliothek zu nutzen, da sich die Integration dieser in der Vergangenheit als schwierig erwies und die Wartbarkeit des Programmcodes im Vergleich zur Alternative von Google erschweren würde.

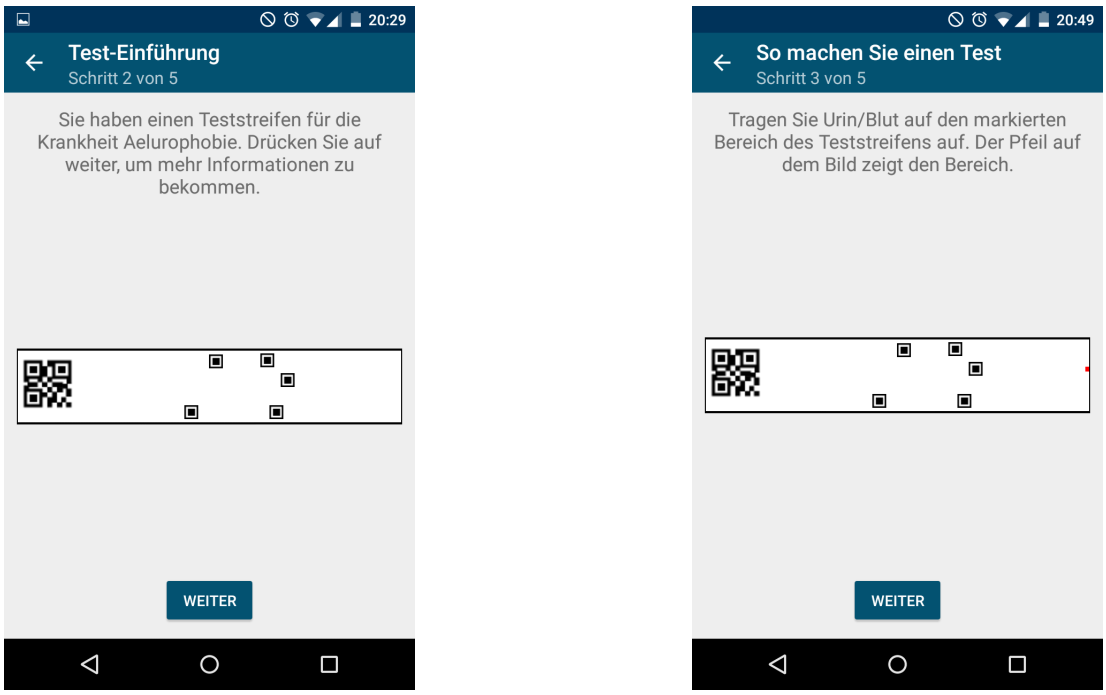



(a) QR-Code-Scanner Erklärungsansicht (b) Die neue QR-Code-Scanner-Ansicht

Abbildung 237: Die neue QR-Code-Scanner-Ansicht

Wenn das durch den QR-Code-Scanner bestimmte Teststreifenschema noch nicht auf dem Smartphone zur Verfügung steht, wird automatisch versucht dieses herunterzuladen, um den Test ungehindert fortsetzen zu können. Im weiteren Verlauf der Testdurchführung wird der aktuelle Teststreifen schematisch dargestellt. Diese schematische Darstellung wird genutzt um zu zeigen, wo sich der Bereich zum Applizieren der Trägerflüssigkeit auf dem Teststreifen befindet (vgl. Abbildung 238 auf der nächsten Seite).

[76] `https://developers.google.com/vision`

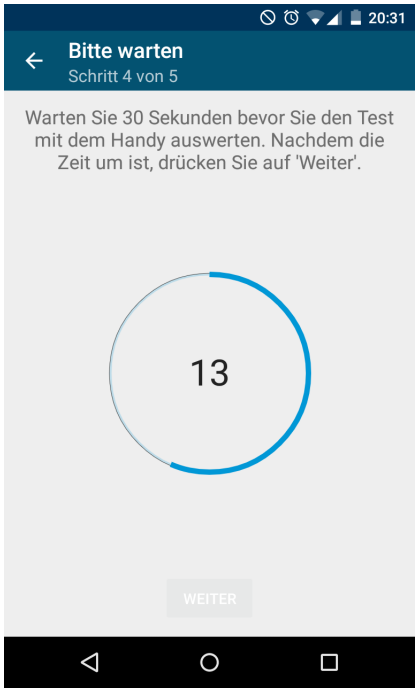



(a) Schritt 2 von 5

(b) Schritt 3 von 5

Abbildung 238: Schritt 2 und 3 von 5 in der Testdurchführung

Es ist wichtig, dass der Nutzer nach dem Applizieren des Sekrets lange genug wartet, bis er den Teststreifen mit der App einscannt. Vorher ist der Farbumschlag womöglich noch nicht in voller Intensität sichtbar. Hierfür ist mit Schritt 4 von 5 eine Timer-Ansicht erstellt worden, auf der der Nutzer sehen kann wie lange er noch warten muss. Erst nach Ablauf des Timers wird die „Weiter“-Taste freigeschaltet, sodass die App weiter zu der Erklärungsansicht für das Einscannen gelangt (vgl. Abbildung 239 auf der nächsten Seite).

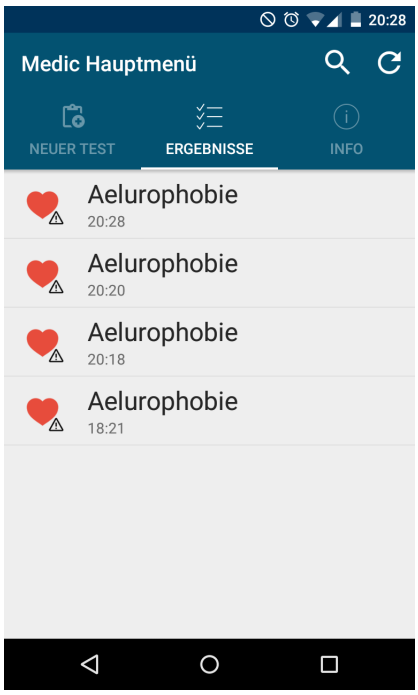



(a) Timer-Ansicht

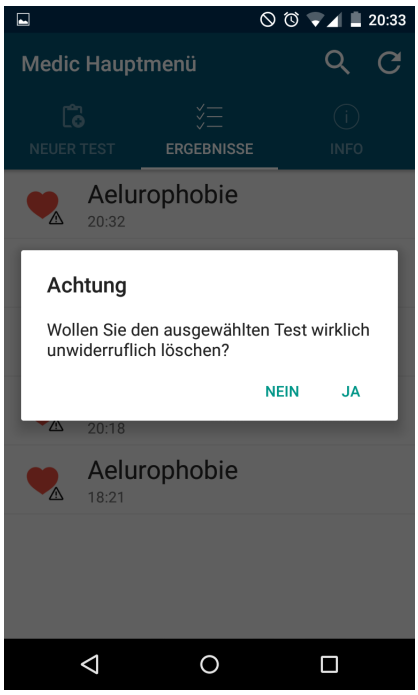

(b) Erklärungsansicht für das darauf folgende Einscannen des Teststreifens

Abbildung 239: Erst wenn der Timer abgelaufen ist, kann der Nutzer mit dem Einscannen des Teststreifens fortfahren

Neben der Testdurchführung, mussten auch alle restlichen Ansichten der App überarbeitet werden. So wurde das Aktualisieren der Diagnoseliste optimiert, damit die Nutzerinteraktion während des Ladens nicht mehr beeinträchtigt wird. Zudem kann diese Liste jetzt entsprechend des Namens der Krankheiten gefiltert werden (vgl. Abbildung 240 auf der nächsten Seite). Ergebnisse zu denen neue Diagnosen vorliegen, werden nun durch dickere Schrift und andere Farbe hervorgehoben. Falls keine Tests vom Nutzer durchgeführt wurden, wird ein Platzhalter anstelle der Liste angezeigt. Es ist jetzt möglich Testergebnisse komplett zu löschen, falls der Nutzer dies explizit bestätigt.



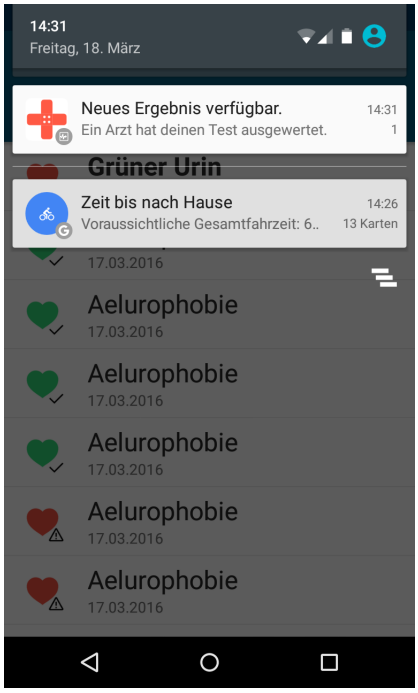

(a) Ergebnisübersicht



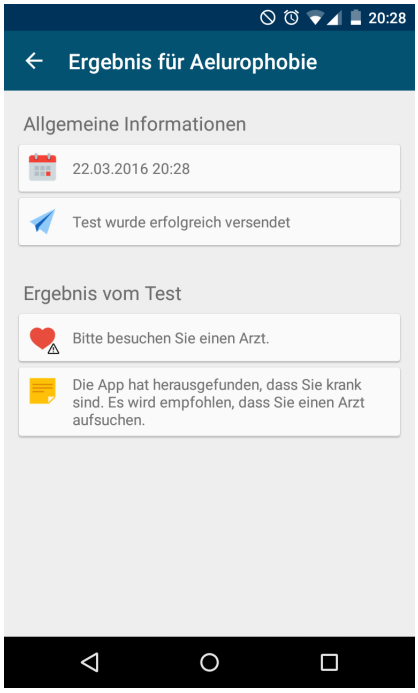

(b) Ergebnis aus Ergebnisübersicht löschen

Abbildung 240: L.: Ergebnisübersicht, R.: Löschen eines Ergebnisses

Wenn der Nutzer die Benachrichtigung über die Google Push Dienste empfängt, dass ein neues Diagnoseergebnis vorhanden ist, gelangt er durch tippen auf selbige direkt zu der Detailansicht, die den aktualisierten Inhalt des Krankheitstests anzeigt (vgl. Abbildung 241). Falls mehrere Ergebnisse gleichzeitig aktualisiert wurden, gelangt er zu der Ergebnisübersicht, in der die entsprechenden Einträge wie bereits erwähnt hervorgehoben sind. Wenn sich der Nutzer bereits auf der Detailansicht befindet, während die Benachrichtigung empfangen wird, aktualisiert sich die Ansicht automatisch.




(a) Neue Notification empfangen

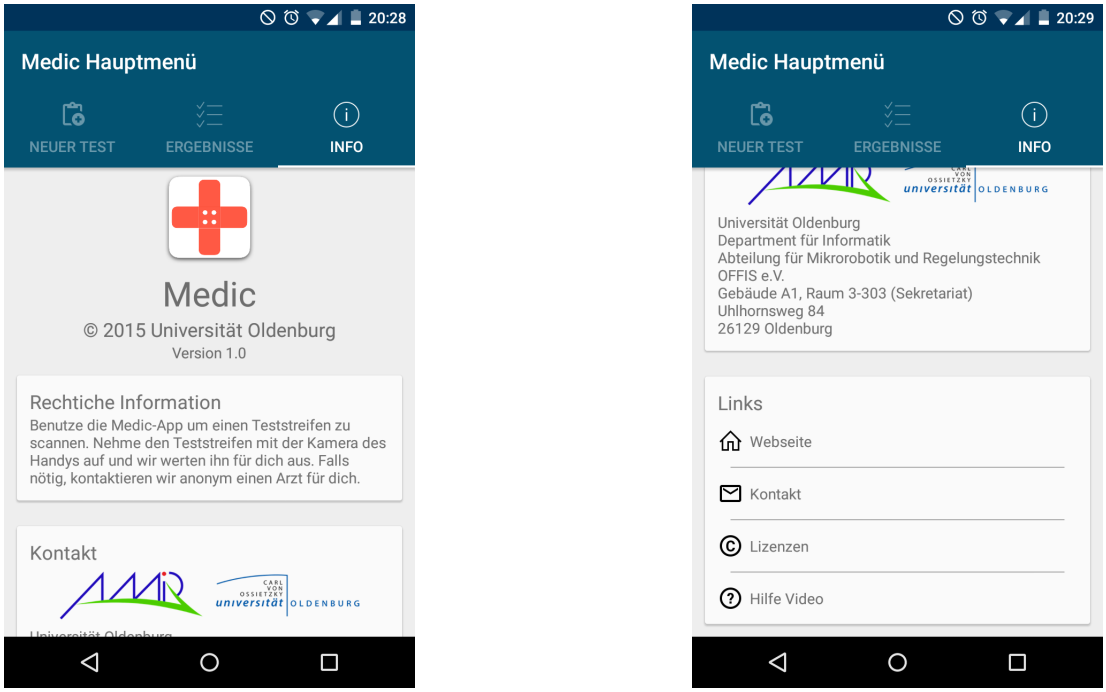



(b) Ergebnis-Detailsansicht

Abbildung 241: Der Nutzer wird per Notification auf neue Diagnosen hingewiesen

Der zur Kommunikation mit dem Server-Backend genutzte API-Schlüssel ist mit Hilfe der Kryptographie-Funktionen verschlüsselt, welche in Java offiziell ausgeliefert werden (und somit auch in Android). Da das Zurückführen der App in Quellcode ein großes Problem bei der Verschlüsselung darstellt, wurden zusätzlich die Variablen- und Methodennamen des zur Verschlüsselung verwendeten Quellcodes als andere Funktionen getarnt, wodurch Angreifer diese nur erschwert finden können.

Auf der Info-Ansicht sind allgemeine Informationen zu der MEDIC-Projektgruppe, der Abteilung AMiR und den genutzten Lizenzen vermerkt. Darüber hinaus ist ein Demo-Video vorhanden, das als zusätzliche Hilfe dient, um dem Nutzer die Bedienung der App zu erklären (vgl. Abbildung 242).




(a) Info-Ansicht erste Hälfte

(b) Info-Ansicht zweite Hälfte

Abbildung 242: Info-Ansicht

Die Texte der gesamten App sind jetzt in der Sprache dargestellt, die der Nutzer auf seinem Android Gerät eingestellt hat. Als Sprachen sind Englisch und Deutsch verfügbar, wobei Englisch die Standardeinstellung ist, falls der Nutzer keine der beiden Sprachen eingestellt hat. Neben den sichtbaren Texten sind zusätzliche Texte vorhanden, die mit der Talk-Back-Funktion[77] z.B. für Blinde vorgelesen werden können. Diese ermöglichen auch diesen Nutzern das Durchführen eines Tests, indem zusätzliche Informationen vorgelesen werden, welche sonst nur durch Bilder dargestellt werden.

Das gesamte Design der Android-Applikation wurde entsprechend einem für alle Projektkomponenten bestimmten Corporate-Design angepasst und bettet sich damit in die Reihe der anderen Produkte ein, welche von der MEDIC-Projektgruppe erstellt wurden. Die Tab-Navigation der App wurde um Symbole erweitert, die sich nun über den Titeln der Tabs präsentieren und das finden der richtigen Ansichten für die Nutzer vereinfachen.

**Technologien** Zur Umsetzung des Arbeitspaketes wurde folgende Technologien neu in die Android-App integriert:

**Google Play Services Vision** Bibliothek zur Bildverarbeitung die in der Android-App zum Scannen von QR-Codes verwendet wird.

[77] ScreenReader von Google, welcher Bildschirminhalte für Nutzer mit Sehbehinderungen vorliest

**Systembeschreibung** Die Android-App wurde mit Android Studio 1.5.1 unter ArchLinux und Windows 10 entwickelt.

**Evaluation** Die Android-App wurde auf folgenden Geräten getestet: Sony Xperia E1, Sony XPeria Tipo, Sony XPeria Z3 Compact, Samsung Galaxy S5 Neo, One Plus One, Motorolla Moto G und LG Bello. Mit allen aufgeführten Endgeräten wurde die App hinsichtlich der im Lastenheft definierten Funktionen, von insgesamt zehn unterschiedlichen Nutzern getestet. Die Applikation ist bis auf eine Einschränkung komplett nutzbar. Wenn die Marker auf dem Teststreifen klein sind und weit auseinander liegen, kann es passieren, dass Low-Cost-Endgeräte (wie z.B. das Sony Xperia E1) nicht in der Lage sind bei der Testdurchführung in Schritt 5 von 5 den Teststreifen einzuscannen, da die Marker weit von einander entfernt sind und sonst nicht alle Marker im Bild sind. Bei solchen Distanzen ist die Auflösung der Kamera jedoch zu gering und die Marker können nicht mehr korrekt detektiert werden.

**Parameter** Um die Android-App zu verwenden muss auf dem Smartphone oder Tablet das Android Betriebssystem in Version 4.0.3 oder höher installiert sein. Das Gerät muss zusätzlich über eine Kamera verfügen, Zugriff auf das Internet haben und eine Installation der Google Play Dienste besitzen.

Für optimalen Betrieb sollte das Android Gerät zusätzlich über eine Kamera mit Autofokus verfügen und Ortungsdienste aktiviert haben.

**Fazit** In diesem Arbeitspaket konnte die Entwicklung der Android-App erfolgreich abgeschlossen werden. Alle im Lastenheft definierten Anforderungen konnten vollständig umgesetzt werden. Zudem wurde ein Handbuch verfasst, das die Installation der Entwicklungsumgebung auf einem Windows 10 Rechner erklärt und alle Funktionen der App, anhand ihrer verschiedenen Ansichten, darstellt.

Somit kann die Android-App zusammen mit dem Handbuch pünktlich zum 31.03.2016 ausgeliefert werden.

**Ausblick** Bei tatsächlicher Veröffentlichung der App wäre es empfehlenswert auf der Info-Ansicht Adressen und Telefonnummern von einer Auswahl an Ärzten aufzulisten, die sich an der Auswertung unklarer Teststreifen beteiligen. So könnte der Nutzer sofort wichtige Informationen erhalten, falls Unklarheiten zu seinem Test bestehen. Diese Telefonnummern könnten auch mit auf der Testdetail-Ansicht angezeigt werden, falls der Test positiv ausfällt.

Um die Accessibility der App nochmals zu steigern, können die „Weiter“ Tasten bei der Testdurchführung durch ein eigenes Design deutlicher hervorgehoben werden. Hierzu zählt auch das Hinzufügen von Icons zu den Tasten.

Durch die in diesem Sprint erarbeitete neue Version des Bildverarbeitungsalgorithmus dauert es jetzt oftmals deutlich länger den Teststreifen einzuscannen. Aus diesem Grund ist es empfehlenswert den Nutzer über die Arbeit der App auf der Scan-Ansicht auf dem Laufenden zu halten. Der Bildverarbeitungsalgorithmus teilt hierzu der App per Fehlercode mit, aus welchen Gründen noch kein Teststreifen ausgewertet werden konnte. Dieser Fehlercode kann dazu verwendet werden dem Nutzer durch ein für ihn verständliches Symbol darzustellen, warum das Einscannen noch nicht erfolgreich war. Hierdurch würde verhindert werden, dass beim Nutzer das Gefühl entsteht, die sei App fehlerhaft oder abgestürzt.

### 4.8.3 Fertigstellung der Bildverarbeitung

- Priorität: Kritisch

- Arbeitspaketbeschreibung:
  Der Algorithmus muss, um für unterschiedliche Teststreifenschemas die Referenzfelder von den Farbumschlagsfeldern trennen zu können, mit zusätzlichen Informationen im Schema umgehen können. Das Schema der Teststreifen muss dementsprechend angepasst werden.

  Zusätzlich muss überprüft werden, ob der Algorithmus hinsichtlich folgender Aspekte weiter optimiert werden kann:

  - Es müssen Farbumschläge mit geringer Intensität detektiert werden können.
  - Die Ergebnisse von Christian Sandmanns Seminararbeit müssen (falls vorliegend) implementiert werden.

  Wenn Optimierungen möglich sind, müssen diese in den Algorithmus integriert werden.

  Es müssen die zehn besten Marker hinsichtlich ihrer Detektierbarkeit ermittelt und für das Erstellen neuer Teststreifen zur Verfügung gestellt werden.

- Vorbedingungen:
  -

- Nebenbedingungen:
  Der Teststreifengenerator ist in der Lage Teststreifen mit dem angepassten Schema zu erstellen. Die Web-Oberfläche ist in der Lage die neuen Teststreifenschemas zu verarbeiten. Das Server-Backend ist in der Lage die neuen Teststreifenschemas an die iOS- und Android-App auszuliefern.

- Nachbedingungen:
  Die in der Beschreibung genannten Optimierungen sind (falls vorhanden) in den Algorithmus integriert. Es existiert eine Liste mit zehn Markern, welche von allen möglichen am besten detektiert werden können und sich soweit von einander unterscheiden, dass keine Fehldetektionen/Verwechslungen auftreten. Es steht ein ausgewähltes Teststreifenschema für die Endpräsentation zur Verfügung. Die Änderungen am Algorithmus sind in die Android- und iOS-App integriert.

- Aufwand:
  6 Wochen

- Personen:

  - Christian Sandmann
  - Sebastian Horwege
  - Jan Philipp Stubbe

**Dokumentation** Das Arbeitspaket befasst sich mit der Finalisierung der Bildverarbeitung. Dabei wurden die noch offenen Punkte aus dem letzten Sprint bearbeitet, eine Klassifizierung der Marker vorgenommen und Fehlercodes eingeführt.

**Ablauf** Ein offener Punkt für den Bildverarbeitungsalgorithmus war es, die möglichen Marker nach solchen zu filtern, die sich besonders gut detektieren lassen. Die Form, sowie die Information in den Markern ist seit dem Arbeitspaket „Lokalisieren des Farbumschlags innerhalb eines definierten Bereichs“ aus Sprint 4 fix. In jedem Marker kann eine Zahl zwischen 0 und 65535 kodiert werden, denn es werden zwei Byte in einer $4 \times 4$ Matrix dargestellt. Da es vom zeitlichen Aufwand her unmöglich ist, alle möglichen Marker aufzunehmen und mit dem Bildverarbeitungsalgorithmus zu testen, werden die Marker nach folgenden Annahmen gefiltert und sortiert:

1. Einzeln liegende gesetzte Bits können schlecht erkannt werden, wenn die Kameraauflösung zu gering, oder der Abstand zum Teststreifen zu hoch ist. Da es bei zu wenig abgebildeten gefärbten Pixeln zu einer Falschklassifizierung kommen kann. Gleiches gilt für gesetzte Bits, in deren Nachbarschaft nur diagonale Nachbarn gesetzt sind. Folglich werden alle Marker aussortiert, die gesetzte Bits aufweisen, die keine gesetzten Nachbarbits in der von Neumann Nachbarschaft haben (siehe Abbildung 243).

   Insgesamt gibt es 59777 Marker, für die dieses Kriterium gilt.

Abbildung 243: Marker mit rot markierten Bits, die die von Neumann Nachbarschaft nicht einhalten

2. Bei zu wenigen gesetzten Bits im Marker, ist es schwerer, diesen zu erkennen und es kann bei einfachen Mustern zu Verwechslungen mit anderen Objekten auf dem Bild kommen. Aus diesem Grund werden Marker aussortiert, die weniger als vier gesetzte Bits haben.

3. Symmetrische Marker, also solche, die in mindestens zwei der vier möglichen Rotationen den gleichen Wert repräsentieren, können nicht zur eindeutigen Detektierung der Orientierung des Teststreifens genutzt werden. Marker, wie in Abbildung 244, werden deshalb aussortiert. Insgesamt gibt es 130 symmetrische Marker mit mehr als vier gesetzten Bits.

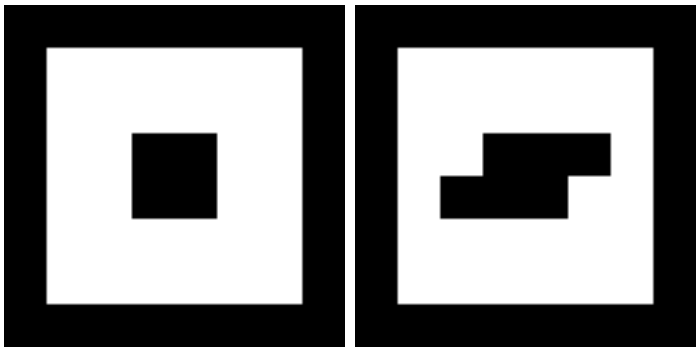


Abbildung 244: Zwei Beispiele für symmetrische und deshalb nicht valide Marker

4. Die im vorherigen Punkt beschriebenen Marker sind nicht besonders häufig, jedoch repräsentiert ein rotierter Marker immer einen anderen (oder im Fall der Symmetrie sogar den gleichen) Wert. Um die Einzigartigkeit der Marker zu gewährleisten, werden für einen Marker, der nicht durch die vorherigen Punkte ausgefiltert wurde, alle vier Rotationen aus der Menge der möglichen Marker entfernt.

5. Die verbleibenden Marker (ca. 6000 von den oben genannten 65535) werden nun nach ihrer Eignung sortiert. Hierfür wird zwischen zwei Markern eine Distanz berechnet, die aussagt, wie viele Bits sich zwischen den beiden Markern unterscheiden. Je größer dieser Wert, desto besser sind die beiden Marker voneinander zu unterscheiden. Es wird jeder Marker mit allen anderen

verglichen, um eine globale Aussage für die Einzigartigkeit des Markers treffen zu können. Dabei werden auch die möglichen Rotationen der Marker berücksichtigt.

Das Ergebnis dieser Filter- und Sortieroperationen ist eine Liste mit Markern, absteigend sortiert nach der Einzigartigkeit der Marker. Diese Liste wurde in den Teststreifengenerator eingebaut, so dass potentiell ungeeignete Marker nicht mehr ausgewählt werden können.

Des Weiteren wurde der Bilderkennungsalgorithmus erweitert. Da der Teststreifengenerator verschiedene Farbfelder erstellen kann, wurde der Algorithmus so verbessert, dass er jetzt auswerten kann, welche von den Farbfeldern Referenzfelder sind und welches das Farbumschlagsfeld ist. Da es mehrere Referenzfelder geben kann, wird aus den Farbwerten der verschiedenen Referenzfelder ein Durchschnittswert gebildet und mit dem Farbumschlagsfeld verglichen. So wird eine höhere Genauigkeit bei dem Farbabgleich erreicht.

Bisher wurde das berechnete Farbumschlagsfeld nur mit den daneben liegenden Referenzfeldern verglichen. Das hatte zur Folge, dass es zu Fehlern kommen kann, wenn beispielsweise ein Finger über den Bereich der Referenzfelder und Farbumschlagsfelder gehalten wurde (vgl. Abbildung 245). Da das Objekt, was über diesen Bereich gehalten wird ggf. einfarbig ist, führt dies dazu, dass der Test fälschlicher Weise als positiv gewertet wird. Aus diesem Grund werden die Farben der Referenzfelder mit den vorher definierten Farben des Teststreifens abgeglichen. Für den Farbabgleich wurden Intervalle für die verschiedenen Farben im HSV-Raum gebildet, damit eine möglichst hohe Genauigkeit erreicht werden kann. Es mussten Intervalle definiert werden, da es nicht möglich war einen festen Schwellwert für alle Farben zu finden. Dies lässt sich an der Farbskala des HSV-Raums erkennen (vgl. Abbildung 246). Vor dem Vergleich der Farben wird ein Weißabgleich bei den Referenzfeldern durchgeführt, sodass Verfärbungen im Bild ausgeglichen werden. Liegt die Farbe des aufgenommenen Referenzwertes nicht in dem Intervall das über die gegebene Farbe aus dem Schema bestimmt wurde, dann wird die Aufnahme verworfen.


Abbildung 245: Testreifen mit Finger über dem Farbumschlagsbereich


Abbildung 246: HSV-Farbraum H-Werte[78]

Bisher konnten noch Fehler entstehen, wenn sich mehrere Teststreifen auf einem Bild befanden. Deshalb wurde der Algorithmus dahingehend weiter entwickelt, dass ein Winkel zwischen einem

Referenzpunkt und einem Eckpunkt eines Markers berechnet wird. Ist der Unterschied zwischen den Winkeln der verschiedenen Marker zu groß, dann wird davon ausgegangen, dass sich ein zweiter Teststreifen auf der Aufnahme befindet und die Aufnahme wird verworfen.

Des Weiteren konnte es zu falschen Auswertungsergebnissen kommen, wenn der Teststreifen verzerrt war. Die Verzerrung hatte zur Folge, dass die Positionen der Referenzfelder und des Farbumschlagfelds nicht richtig berechnet werden konnten. Das führte dazu, dass undefinierte Farben miteinander verglichen wurden und das Ergebnis nicht mehr stimmt (vgl. Abbildung 247). Jetzt werden die Transformationen auf dem Bild immer mit einem Marker weniger durchgeführt und damit überprüft, ob die berechneten Punkte des fehlenden Markers mit den Punkten des definierten Markers übereinstimmen. Wenn dies nicht der Fall ist, dann wird die Aufnahme verworfen.

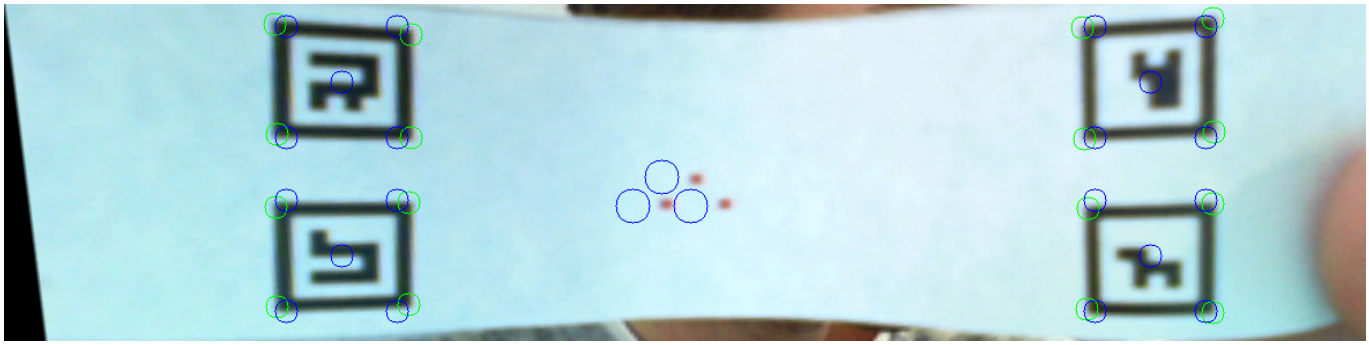


Abbildung 247: Verzerrter Teststreifen mit berechneten Farbumschlagsfeldern

Während die Rückgabe der vorherigen Version des Algorithmus lediglich beinhaltete, ob der Teststreifen erkannt wurde oder nicht, gibt es nun in der neuen Version einen Fehlercode, der über die Art der Fehler informiert. Die Fehlercodes sind in Tabelle 50 auf Seite 359 aufgelistet. Tritt einer dieser Fehler auf, dann wird die Durchführung des Algorithmus unterbrochen und der Fehlercode wird zurückgegeben.

Durch die Rückgabe der Fehlercodes und den Abgleich der Referenzfelder mit den definierten Farben hat sich der Ablauf des Algorithmus verändert. Wie in Abbildung 248 auf der nächsten Seite zu erkennen ist, sind an vielen Stellen im Algorithmus Prüfungen hinzugekommen, an denen die Durchführung unterbrochen werden kann. Erst wenn alle möglichen Fehlerfälle überprüft wurden, wird ein Farbabgleich zwischen dem Farbumschlagsfeld und den Referenzfeldern durchgeführt.

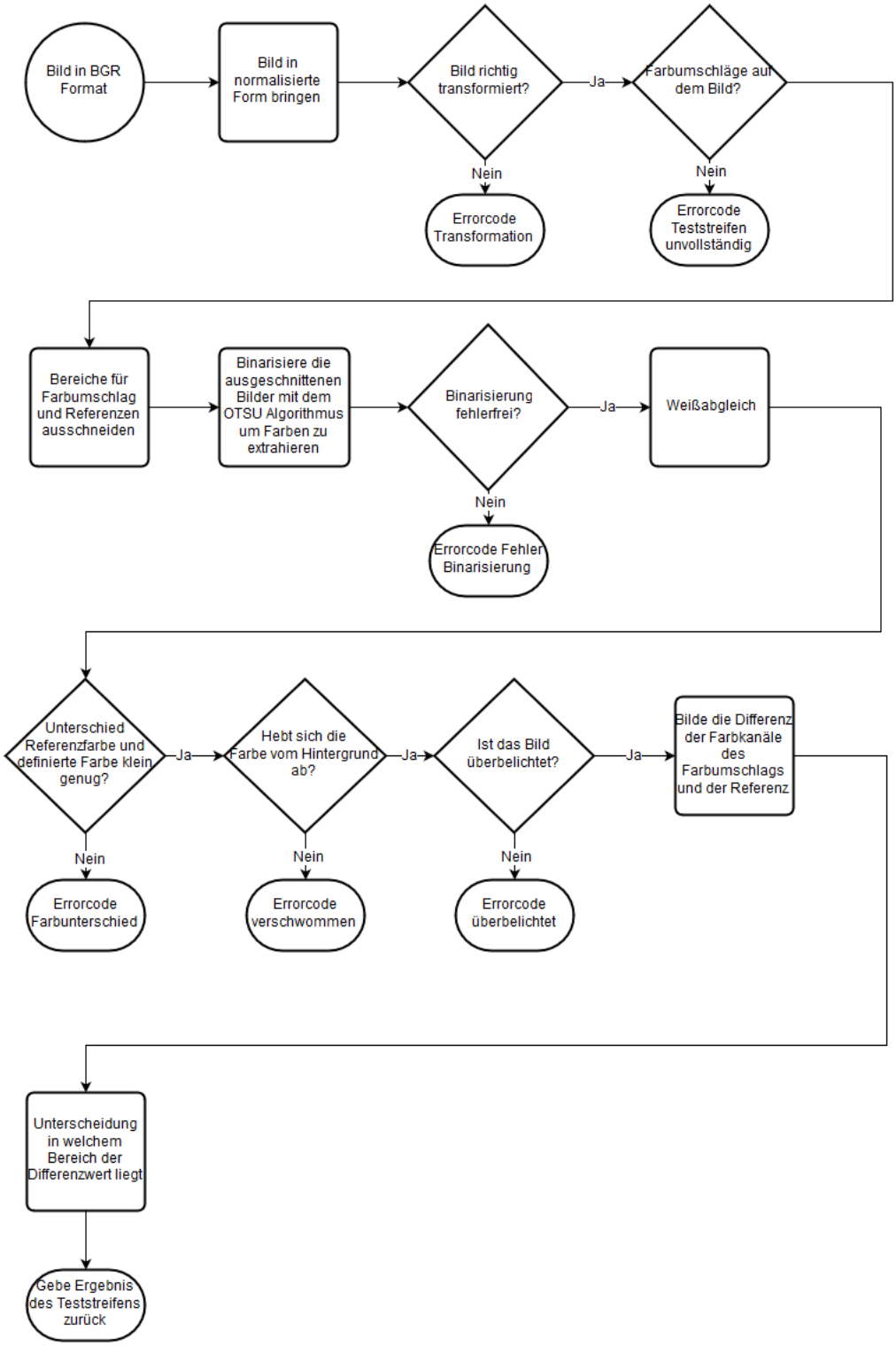



Abbildung 248: Ablauf der Bildverarbeitung

| Bezeichnung | Wert | Beschreibung |
|---|---|---|
| NO_ERROR | 0 | Der Algorithmus ist ohne Probleme durchgelaufen und gibt ein valides Ergebnis zurück. |
| ERROR_COLOR_DIFFERENCE | 1 | Die Farbumschlagfelder haben nicht die erwarteten Farbwerte. Dies ist wahrscheinlich zurückzuführen auf die Verdeckung der Farbumschlagsfelder im aufgenommenen Bild. |
| ERROR_TRANSFORMED_DISTORTION | 2 | Der Teststreifen ist so stark verzerrt, dass eine zuverlässige Berechnung der Farbumschlagsfelder nicht mehr möglich ist. Dies ist auch der Fall, wenn Teststreifen überlagert sind und deshalb Marker von unterschiedlichen Teststreifen gescannt wurden. |
| ERROR_BLURRY | 3 | Das Bild ist zu verschwommen. Der Algorithmus kann die Referenzfelder nicht erkennen, obwohl aber alle Marker gefunden wurden. |
| ERROR_NOT_ENOUGH_MARKERS | 4 | Es konnte eine unzureichende Menge an Markern gefunden werden. |
| ERROR_TESTSTRIP_INCOMPLETE | 5 | Dieser Fehler tritt auf, wenn der Teststreifen nicht vollständig gescannt werden konnte, es aber genug Marker für die Detektierung gab. |
| ERROR_MAX_NON_ZEROS_REFERENCE_MASK | 6 | Dieser Fehler tritt auf, wenn beim Binarisieren des Bildes ein Fehler auftritt. |
| ERROR_LIGHT | 7 | Das Bild ist überbelichtet, sodass die Referenzfelder und das Farbumschlagsfeld nicht mehr richtig erkannt werden können. |

Tabelle 50: Tabelle mit Fehlercodes

**Technologien**

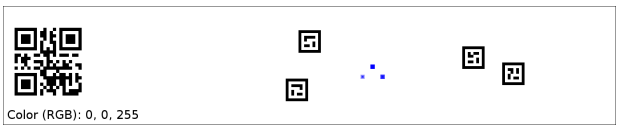
**OpenCV** Bibliothek für die Bildverarbeitung

**Teststand & Testframework** Zum Evaluieren des Algorithmus

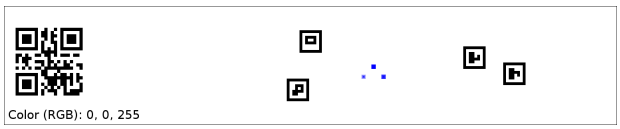
**Systembeschreibung** Die Änderungen am Code für die Bildverarbeitung wurden unter Windows 10 vorgenommen und getestet. Es wurde Eclipse Mars als IDE eingesetzt. Während der Entwicklung wurden anhand von aufgenommenen Videos Tests durchgeführt. Die Auswertung basiert auf Videos, die im Teststand angefertigt wurde.

**Evaluation** Die oben beschriebene Sortierung der Marker nach Eignung für den Bildverarbeitungsalgorithmus muss als Heuristik verstanden werden. Da es aber zu aufwendig ist, jeden Marker zu testen, ist diese Vorgehensweise auf jeden Fall gerechtfertigt.

Exemplarisch wurden zwei Teststreifen mit guten (Abbildung 249a) und schlechten (Abbildung 249b) Markern erzeugt und für eine Evaluation mit der aktuellen Version des Bildverarbeitungsalgorithmus herangezogen.




(a) Testreifen mit guten Markern




(b) Testreifen mit schlechten Markern

Die Teststreifen wurden jeweils mit 80 Einstellungen im Teststand abgefahren und mit dem Galaxy S 5 Neo aufgenommen. Für beide Teststreifenvarianten ergab sich damit also exakt die gleiche Konfiguration - abgesehen von den Markern. Im Vergleich zu den schlechter geeigneten Markern, konnte bei der guten Teststreifenvariante zwar nur in zwei weiteren Fällen der Teststreifen detektiert werden, aber statt der vorher durchschnittlich benötigten 20 Bildern bis zur Detektieren, waren bei den guten Markern nur 8 nötig.

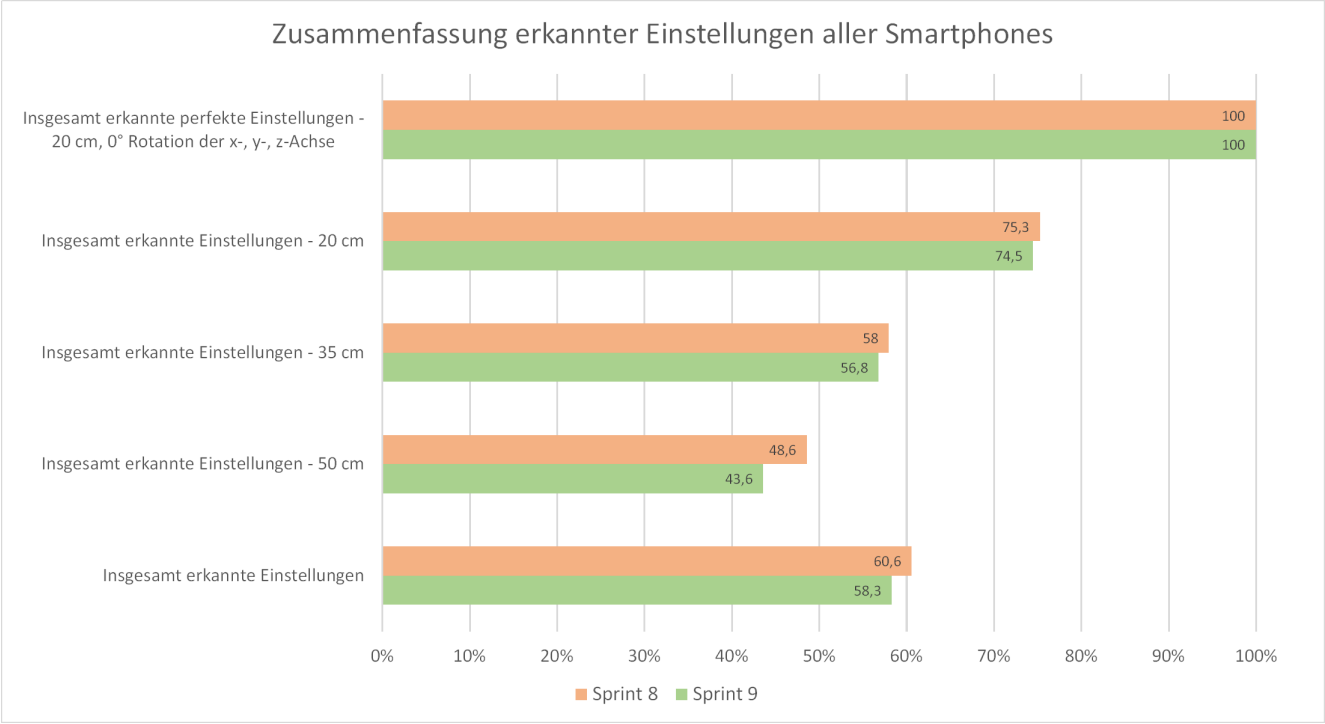



Abbildung 250: Zusammenfassung der Evaluation

| Alle Smartphones | Entfernung | Winkel | | | Erkannt (in %) | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | 100 | 100 |
| | 20 cm | 30 | 0 | 0 | 88,9 | 88,9 |
| | 20 cm | 60 | 0 | 0 | 88,9 | 88,9 |
| | 20 cm | 0 | 10 | 0 | 100 | 100 |
| | 20 cm | 30 | 10 | 0 | 88,9 | 88,9 |
| | 20 cm | 60 | 10 | 0 | 77,8 | 77,8 |
| | 20 cm | 0 | 25 | 0 | 100 | 100 |
| | 20 cm | 30 | 25 | 0 | 88,9 | 88,9 |
| | 20 cm | 60 | 25 | 0 | 77,8 | 77,8 |
| | 20 cm | 0 | 0 | 25 | 100 | 100 |
| | 20 cm | 30 | 0 | 25 | 100 | 100 |
| | 20 cm | 60 | 0 | 25 | 77,8 | 77,8 |
| | 20 cm | 0 | 10 | 25 | 100 | 100 |
| | 20 cm | 30 | 10 | 25 | 55,6 | 55,6 |
| | 20 cm | 60 | 10 | 25 | 77,8 | 77,8 |
| | 20 cm | 0 | 25 | 25 | 100 | 100 |
| | 20 cm | 30 | 25 | 25 | 0 | 0 |
| | 20 cm | 60 | 25 | 25 | 77,8 | 66,7 |
| | 20 cm | 0 | 0 | 50 | 100 | 100 |
| | 20 cm | 30 | 0 | 50 | 0 | 0 |
| | 20 cm | 60 | 0 | 50 | 77,8 | 77,8 |
| | 20 cm | 0 | 10 | 50 | 100 | 88,9 |
| | 20 cm | 30 | 10 | 50 | 88,9 | 88,9 |
| | 20 cm | 60 | 10 | 50 | 77,8 | 77,8 |
| | 20 cm | 0 | 25 | 50 | 66,7 | 66,7 |
| | 20 cm | 30 | 25 | 50 | 0 | 0 |
| | 20 cm | 60 | 25 | 50 | 22,2 | 22,2 |

Tabelle 51: Zusammenfassung aller Smartphones bei Distanz von 20cm

| Alle Smartphones | Entfernung | Winkel | | | Erkannt (in %) | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | 77,8 | 77,8 |
| | 35 cm | 30 | 0 | 0 | 55,6 | 33,3 |
| | 35 cm | 60 | 0 | 0 | 66,7 | 66,7 |
| | 35 cm | 0 | 10 | 0 | 77,8 | 77,8 |
| | 35 cm | 30 | 10 | 0 | 44,4 | 44,4 |
| | 35 cm | 60 | 10 | 0 | 66,7 | 66,7 |
| | 35 cm | 0 | 25 | 0 | 77,8 | 77,8 |
| | 35 cm | 30 | 25 | 0 | 77,8 | 77,8 |
| | 35 cm | 60 | 25 | 0 | 66,7 | 66,7 |
| | 35 cm | 0 | 0 | 25 | 77,8 | 77,8 |
| | 35 cm | 30 | 0 | 25 | 77,8 | 77,8 |
| | 35 cm | 60 | 0 | 25 | 66,7 | 66,7 |
| | 35 cm | 0 | 10 | 25 | 77,8 | 77,8 |
| | 35 cm | 30 | 10 | 25 | 22,2 | 22,2 |
| | 35 cm | 60 | 10 | 25 | 66,7 | 66,7 |
| | 35 cm | 0 | 25 | 25 | 77,8 | 77,8 |
| | 35 cm | 30 | 25 | 25 | 0 | 0 |
| | 35 cm | 60 | 25 | 25 | 66,7 | 66,7 |
| | 35 cm | 0 | 0 | 50 | 77,8 | 77,8 |
| | 35 cm | 30 | 0 | 50 | 0 | 0 |
| | 35 cm | 60 | 0 | 50 | 66,7 | 66,7 |
| | 35 cm | 0 | 10 | 50 | 77,8 | 77,8 |
| | 35 cm | 30 | 10 | 50 | 77,8 | 77,8 |
| | 35 cm | 60 | 10 | 50 | 66,7 | 66,7 |
| | 35 cm | 0 | 25 | 50 | 44,4 | 33,3 |
| | 35 cm | 30 | 25 | 50 | 0 | 0 |
| | 35 cm | 60 | 25 | 50 | 11,1 | 11,1 |

Tabelle 52: Zusammenfassung aller Smartphones bei Distanz von 35cm

| Alle Smartphones | Entfernung | Winkel | | | Erkannt (in %) | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | 77,8 | 77,8 |
| | 50 cm | 30 | 0 | 0 | 55,6 | 55,6 |
| | 50 cm | 60 | 0 | 0 | 66,7 | 55,6 |
| | 50 cm | 0 | 10 | 0 | 77,8 | 66,7 |
| | 50 cm | 30 | 10 | 0 | 55,6 | 44,4 |
| | 50 cm | 60 | 10 | 0 | 55,6 | 55,6 |
| | 50 cm | 0 | 25 | 0 | 66,7 | 66,7 |
| | 50 cm | 30 | 25 | 0 | 55,6 | 55,6 |
| | 50 cm | 60 | 25 | 0 | 55,6 | 44,4 |
| | 50 cm | 0 | 0 | 25 | 77,8 | 66,7 |
| | 50 cm | 30 | 0 | 25 | 55,6 | 55,6 |
| | 50 cm | 60 | 0 | 25 | 55,6 | 44,4 |
| | 50 cm | 0 | 10 | 25 | 66,7 | 66,7 |
| | 50 cm | 30 | 10 | 25 | 44,4 | 33,3 |
| | 50 cm | 60 | 10 | 25 | 55,6 | 44,4 |
| | 50 cm | 0 | 25 | 25 | 66,7 | 66,7 |
| | 50 cm | 30 | 25 | 25 | 11,1 | 0 |
| | 50 cm | 60 | 25 | 25 | 44,4 | 44,4 |
| | 50 cm | 0 | 0 | 50 | 66,7 | 66,7 |
| | 50 cm | 30 | 0 | 50 | 11,1 | 11,1 |
| | 50 cm | 60 | 0 | 50 | 33,3 | 22,2 |
| | 50 cm | 0 | 10 | 50 | 44,4 | 44,4 |
| | 50 cm | 30 | 10 | 50 | 44,4 | 44,4 |
| | 50 cm | 60 | 10 | 50 | 44,4 | 33,3 |
| | 50 cm | 0 | 25 | 50 | 22,2 | 11,1 |
| | 50 cm | 30 | 25 | 50 | 0 | 0 |
| | 50 cm | 60 | 25 | 50 | 0 | 0 |

Tabelle 53: Zusammenfassung aller Smartphones bei Distanz von 50cm

Zur Überprüfung, wie sich der Algorithmus verändert hat, wurde ein Vergleich mit der vorherigen Version erstellt. Da an der Markererkennung keine Veränderungen vorgenommen wurden, werden lediglich die Ergebnisse, welche die Version mit der Fehlerbehandlung enthalten, verglichen.

Der Abbildung 250 auf Seite 360 ist zu entnehmen, dass insgesamt weniger Teststreifen erkannt werden. Das hat den Grund, dass mehr Überprüfungen durchgeführt werden, sodass fehlerhafte Bilder, die vorher nicht als fehlerhaft erkannt wurden, verworfen werden. Es werden damit Teststreifen verworfen, bei denen keine zuverlässige Ergebnisermittlung möglich ist. Sind die Einstellungen perfekt, dann erkennt der Algorithmus weiterhin alle Teststreifen genauso gut wie vorher. Speziell bei einem Abstand von 50cm wurden mehrere Bilder verworfen. Der Tabelle 53 lässt sich entnehmen, dass dies nicht mit bestimmten Winkeln zusammenhängt. Durch die größere Entfernung ist die Aufnahme des Teststreifens nicht mehr so gut. Das führt dazu, dass die Referenzfelder nicht mehr richtig erkannt werden können. In den Tabellen 51 auf der vorherigen Seite und 52 auf der vorherigen Seite ist zu erkennen, dass nur vereinzelte Bilder nicht erkannt werden konnten.

Während die vorherigen Dokumentationen zu Verbesserung und Auswertung des Bildverarbeitungsalgorithmus (Sprint 8 „Stabilisierung der Bildverarbeitung“ Kapitel 4.7.4 auf Seite 309 und „Algorithmenvergleich“ Kapitel 4.7.4 auf Seite 309) alle Einstellungen des Teststandes als gleichwertig betrachteten, berücksichtigt die folgende Auswertung die Abweichung von der erwarteten Eingabe. Dazu wird für eine gegebene Orientierung des Teststands, also dem 4-Tupel $A = (\epsilon, \alpha, \beta, \gamma)$, mit $\epsilon$ als Teststandeinstellung für die Entfernung des Teststreifens und $\alpha, \beta, \gamma$ als Winkel eins bis drei (siehe hierzu „Erweiterung des Teststandes“ Kapitel 4.7.3 auf Seite 306), der Abstand von der erwarteten Einstellung $A_E = (5cm, 0, 0, 0)$ mit Gewichtung über die Extremeinstellung des Teststands $A_{MAX} = (33cm, 90, 25, 65)$ berechnet.

Die berechnete Abweichung von der erwarteten Eingabe für eine gegebene Einstellung $A_{GEG}$ ergibt sich dann wie in Abbilung 251.

$$D(A_E, A_{GEG}) := \frac{\frac{abs(A_E(\epsilon)-A_{GEG}(\epsilon))}{abs(A_E(\epsilon)-A_{MAX}(\epsilon))} + \frac{abs(A_E(\alpha)-A_{GEG}(\alpha))}{abs(A_E(\alpha)-A_{MAX}(\alpha))} + \frac{abs(A_E(\beta)-A_{GEG}(\beta))}{abs(A_E(\beta)-A_{MAX}(\beta))} + \frac{abs(A_E(\gamma)-A_{GEG}(\gamma))}{abs(A_E(\gamma)-A_{MAX}(\gamma))}}{4}$$

Abbildung 251: Abweichung von der erwarteten Eingabe für eine gegebene Einstellung

Der Wertebereich für die Abweichung von der erwarteten Eingabe liegt offensichtlich im Interval $[0, 1]$. Wobei der Wert 1 der Teststandeinstellung von $A_{MAX}$ entspricht. Die Performance der Algorithmen aus Sprint 7, 8 und 9 wurde nun mit dieser neuen Bewertung nochmals verglichen. Dabei wurden sämtliche Eingabevideos ausgewertet und daraufhin untersucht, ob der Teststreifen im Video erkannt werden konnte, oder nicht. Der Datensatz betrug zum Zeitpunkt der Auswertung ca. 2300 Videos.

In den Abbildungen 252 bis 254 auf der nächsten Seite sind die normalisierten Histogramme zur Performance der Teststreifenerkennung in Abhängigkeit von der Abweichung von der erwarteten Eingabe für Sprint 7 bis 9 abgebildet. Die Klassen des Histogramms sind hierbei in 1% Intervallen von 0% bis 100% Abweichung von der erwarteten Eingabe eingeteilt. Hierbei fällt auf, dass Sprint 7 (Abbildung 252) eine niedrigere Erkennungsrate bei wahrscheinlicheren Orientierungen hat, als Sprint 8 und 9. Dafür liegt die Grenze, bis zu der Teststreifen erkannt werden bie etwa 85% im Vergleich zu 65% (Sprint 8) und 75% (Sprint 9). Die Änderungen am Bildverarbeitungsalgorithmus in Sprint 9 haben erhebliche Verbesserungen bei der Erkennung der Teststreifen bei wahrscheinlichen Eingaben gebracht. In Sprint 9 wurden zum ersten Mal umfassende Verifizierungen am Bildverarbeitungsalgorithmus vorgenommen, die Teststreifen verwerfen, für die zwar genügend Marker erkannt wurden, die aber gleichzeitig nicht richtig zurückgerechnet werden konnten. Deshalb sinkt an einigen Stellen die Erkennungsrate.

In Abbildung 255 sind Histogramme abgebildet, die bei der Auswertung von Sprint 9 jeweils einen der vier Freiheitsgrade des Teststandes berücksichtigen. Hierbei wird ersichtlich, dass die Qualität des Algorithmus vor allem von den Winkeln abhängt und nicht von der Entfernung des Teststreifens von der Kamera.

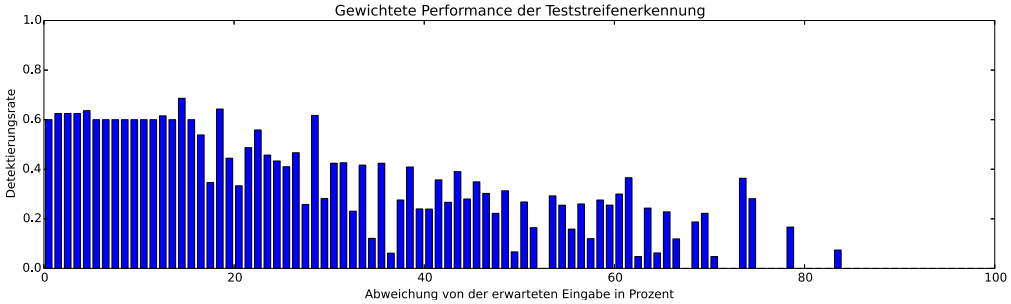



Abbildung 252: Gesamtperformance Algorithmus zu Sprint 7

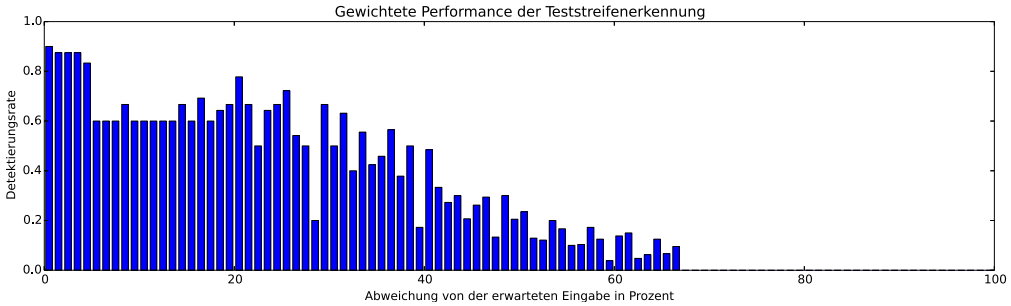



Abbildung 253: Gesamtperformance Algorithmus zu Sprint 8

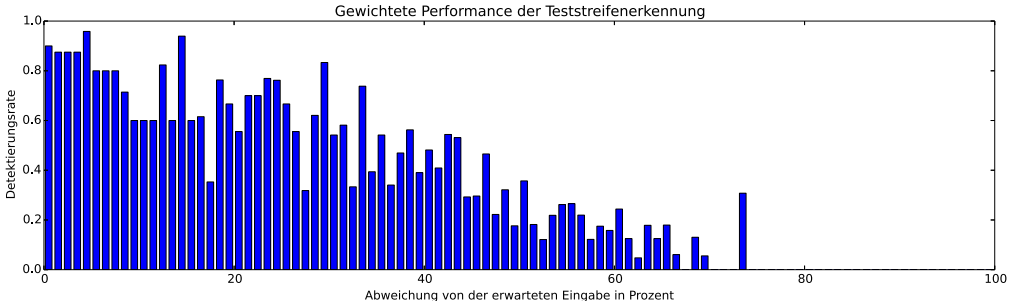
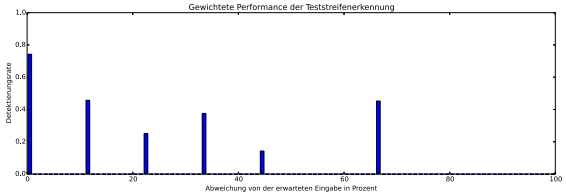
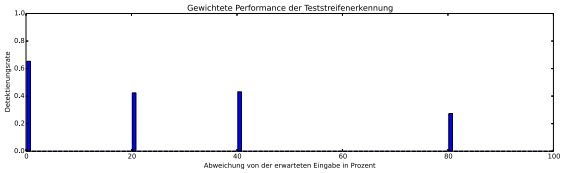


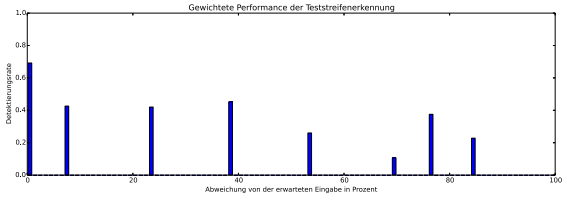

Abbildung 254: Gesamtperformance Algorithmus zu Sprint 9



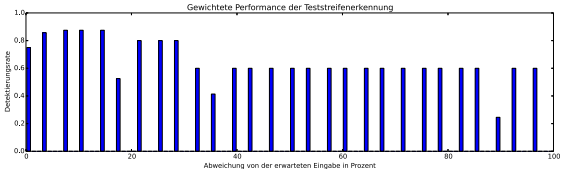

(a) Performance abhängig von Winkel 1




(b) Performance abhängig von Winkel 2




(c) Performance abhängig von Winkel 3




(d) Performance abhängig von Distanz

Abbildung 255: Untersuchung der Performance für einzelne Parameter

**Parameter** Der Bildverarbeitungsalgorithmus setzt voraus, dass der aufgenommene Teststreifen nicht verzerrt ist, da sonst die Positionen der Farbumschlagsfelder nicht mehr richtig berechnet werden können. Des Weiteren darf nur ein Teststreifen auf dem aufgenommenen Bild vorhanden sein. Die Referenzfelder und das Farbumschlagsfeld müssen auf dem aufgenommenen Teststreifen erkennbar sein und die Referenzfelder müssen die gleiche Farbe haben, die in dem Teststreifenschema angegeben ist. Ist einer dieser Fälle nicht gegeben, dann wird die Bildverarbeitung unterbrochen und ein Fehlercode zurückgegeben.

**Fazit** Die Sortierung der Marker nach Eignung für den Bildverarbeitungsalgorithmus brachte zwar keine großen Erfolge bei der Anzahl an detektierten Teststreifen, jedoch konnte die Geschwindigkeit, in der die Teststreifen erkannt wurden durch die richtige Wahl der Marker mehr als halbiert werden.

Insgesamt werden weniger Teststreifen erkannt. Dies ist damit zu begründen, dass eine Fehlerbehandlung durchgeführt wird. Diese wurde in den vorherigen Sprints nicht durchgeführt. Dies hat zur Folge, dass der Algorithmus keine falschen Ergebnisse mehr liefert, wenn beispielsweise der Farbumschlagsbereich auf dem Teststreifen verdeckt wird.

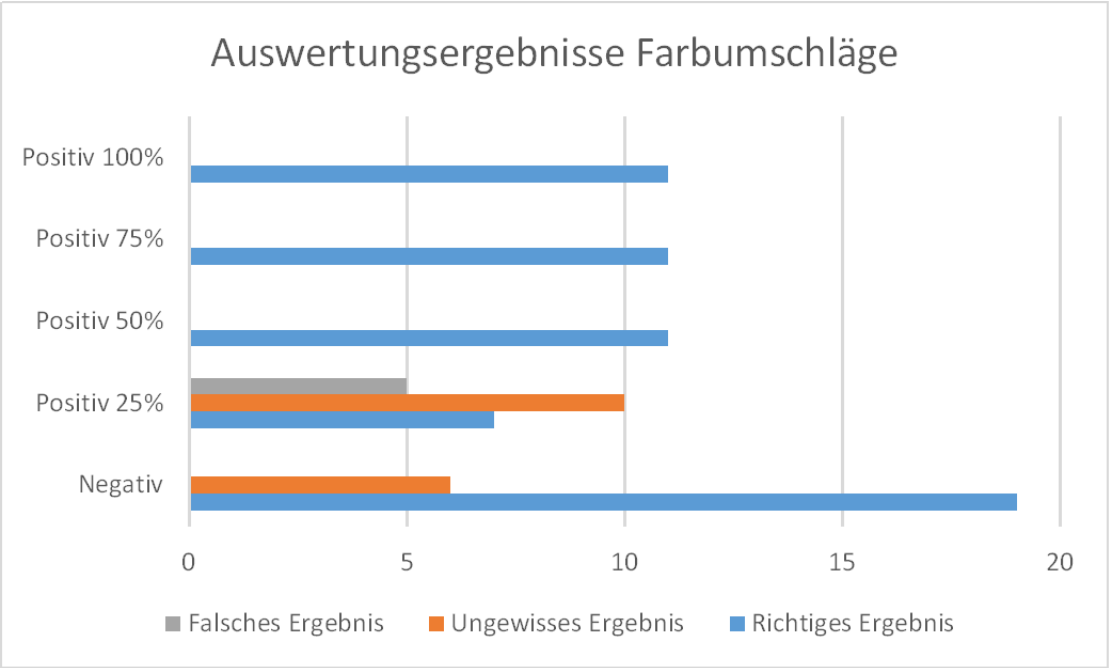



Abbildung 256: Auswertungsergebnisse der Farbumschläge Sprint 9

Die Diagnose, die von dem Algorithmus ausgegeben wird, bekommt damit eine höhere Qualität, die vorher nicht gegeben war. Die Abbildung 256 zeigt, dass fast alle Einstellungen richtig erkannt werden. Einzig bei dem positiven Teststreifen, welcher eine Farbumschlagsintensität von 25% hat wird in einigen Fällen falsch detektiert. In der Abbildung 257 ist ein Teststreifen, der falsch detektiert wurde. Der Farbumschlag, welcher ganz links hätte sein sollen, ist nicht mehr zu erkennen. Zum Vergleich kann die Abbildung 258 auf der nächsten Seite, auf der eine optimale Aufnahme des Teststreifens ist, herangezogen werden. Bei einer Farbumschlagsintensität von 25% ist das ermittelte Ergebnis demnach nicht 100% ig zuverlässig. Der alte Algorithmus hat hingegen bei allen Aufnahmen ein unbestimmtes Ergebnis geliefert. Eine Diagnose war somit nicht möglich.

Abbildung 257: Teststreifenaufnahme auf der der Farbumschlag nicht mehr zu erkennen ist

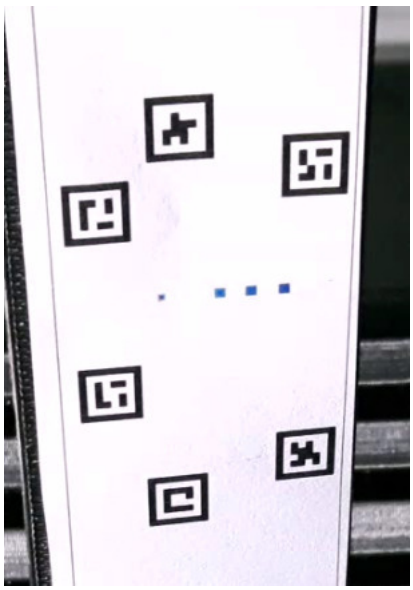


Abbildung 258: Optimale Aufnahme des Teststreifens

**Ausblick** In diesem Sprint wurde der Algorithmus fertiggestellt und alle bekannten Fehlergründe wurden abgefangen. Am Algorithmus gibt es demnach keine weiteren Verbesserungen mehr durchzuführen. Durch die eingeführte Fehlerbehandlung wird die Möglichkeit geschaffen in den Apps auf die einzelnen Fehler zu reagieren und dem Benutzer eine entsprechende Rückmeldung zu geben.

**Anhang** Hier eine genaue Auflistung aller Geräte und ihrer Testergebnisse. Alle Aufnahmen wurden bei einer Lichtintensität von 2700 Kelvin im Teststand gemacht.

| iPad mini 2 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0° | 0° | 0° | Ja | Ja |
| | 20 cm | 30° | 0° | 0° | Ja | Ja |
| | 20 cm | 60° | 0° | 0° | Nein | Ja |
| | 20 cm | 0° | 10° | 0° | Ja | Ja |
| | 20 cm | 30° | 10° | 0° | Nein | Ja |
| | 20 cm | 60° | 10° | 0° | Nein | Nein |
| | 20 cm | 0° | 25° | 0° | Nein | Ja |
| | 20 cm | 30° | 25° | 0° | Nein | Ja |
| | 20 cm | 60° | 25° | 0° | Nein | Nein |
| | 20 cm | 0° | 0° | 25° | Ja | Ja |
| | 20 cm | 30° | 0° | 25° | Nein | Ja |
| | 20 cm | 60° | 0° | 25° | Nein | Nein |
| | 20 cm | 0° | 10° | 25° | Nein | Ja |
| | 20 cm | 30° | 10° | 25° | Nein | Nein |
| | 20 cm | 60° | 10° | 25° | Nein | Nein |
| | 20 cm | 0° | 25° | 25° | Nein | Ja |
| | 20 cm | 30° | 25° | 25° | Nein | Nein |
| | 20 cm | 60° | 25° | 25° | Nein | Nein |
| | 20 cm | 0° | 0° | 50° | Nein | Ja |
| | 20 cm | 30° | 0° | 50° | Nein | Nein |
| | 20 cm | 60° | 0° | 50° | Nein | Nein |
| | 20 cm | 0° | 10° | 50° | Nein | Ja |
| | 20 cm | 30° | 10° | 50° | Nein | Ja |
| | 20 cm | 60° | 10° | 50° | Nein | Nein |
| | 20 cm | 0° | 25° | 50° | Nein | Nein |
| | 20 cm | 30° | 25° | 50° | Nein | Nein |
| | 20 cm | 60° | 25° | 50° | Nein | Nein |

Tabelle 54: Zusammenfassung des iPad Mini 2 bei 20cm Abstand

| iPad mini 2 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Ja | Ja |
| | 35 cm | 60 | 0 | 0 | Ja | Ja |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Ja | Ja |
| | 35 cm | 60 | 10 | 0 | Nein | Nein |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Nein | Nein |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Nein | Nein |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Nein | Nein |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Nein | Nein |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Nein | Nein |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Nein | Nein |
| | 35 cm | 0 | 25 | 50 | Nein | Nein |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 55: Zusammenfassung des iPad Mini 2 bei 35cm Abstand

| iPad mini 2 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Ja | Ja |
| | 50 cm | 60 | 0 | 0 | Nein | Nein |
| | 50 cm | 0 | 10 | 0 | Ja | Ja |
| | 50 cm | 30 | 10 | 0 | Ja | Ja |
| | 50 cm | 60 | 10 | 0 | Nein | Nein |
| | 50 cm | 0 | 25 | 0 | Ja | Ja |
| | 50 cm | 30 | 25 | 0 | Ja | Ja |
| | 50 cm | 60 | 25 | 0 | Nein | Nein |
| | 50 cm | 0 | 0 | 25 | Ja | Ja |
| | 50 cm | 30 | 0 | 25 | Ja | Ja |
| | 50 cm | 60 | 0 | 25 | Nein | Nein |
| | 50 cm | 0 | 10 | 25 | Ja | Ja |
| | 50 cm | 30 | 10 | 25 | Ja | Ja |
| | 50 cm | 60 | 10 | 25 | Nein | Nein |
| | 50 cm | 0 | 25 | 25 | Ja | Ja |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Nein | Nein |
| | 50 cm | 0 | 0 | 50 | Ja | Ja |
| | 50 cm | 30 | 0 | 50 | Ja | Ja |
| | 50 cm | 60 | 0 | 50 | Nein | Nein |
| | 50 cm | 0 | 10 | 50 | Ja | Ja |
| | 50 cm | 30 | 10 | 50 | Ja | Ja |
| | 50 cm | 60 | 10 | 50 | Nein | Nein |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 56: Zusammenfassung des iPad Mini 2 bei 50cm Abstand

| LG Bello (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Ja | Ja |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Ja |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Nein | Nein |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Ja | Ja |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Ja | Ja |

Tabelle 57: Zusammenfassung des LG Bello bei 20cm Abstand

| LG Bello (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Nein | Nein |
| | 35 cm | 60 | 0 | 0 | Ja | Ja |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Nein | Nein |
| | 35 cm | 60 | 10 | 0 | Ja | Ja |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Ja | Ja |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Ja | Ja |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Ja | Ja |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Ja | Ja |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Ja | Ja |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Ja | Ja |
| | 35 cm | 0 | 25 | 50 | Nein | Nein |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 58: Zusammenfassung des LG Bello bei 35cm Abstand

| LG Bello (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Nein | Nein |
| | 50 cm | 60 | 0 | 0 | Ja | Ja |
| | 50 cm | 0 | 10 | 0 | Ja | Ja |
| | 50 cm | 30 | 10 | 0 | Nein | Nein |
| | 50 cm | 60 | 10 | 0 | Ja | Ja |
| | 50 cm | 0 | 25 | 0 | Ja | Ja |
| | 50 cm | 30 | 25 | 0 | Ja | Ja |
| | 50 cm | 60 | 25 | 0 | Ja | Nein |
| | 50 cm | 0 | 0 | 25 | Ja | Ja |
| | 50 cm | 30 | 0 | 25 | Nein | Nein |
| | 50 cm | 60 | 0 | 25 | Ja | Ja |
| | 50 cm | 0 | 10 | 25 | Ja | Ja |
| | 50 cm | 30 | 10 | 25 | Nein | Nein |
| | 50 cm | 60 | 10 | 25 | Ja | Ja |
| | 50 cm | 0 | 25 | 25 | Ja | Ja |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Nein | Nein |
| | 50 cm | 0 | 0 | 50 | Ja | Ja |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Nein | Nein |
| | 50 cm | 0 | 10 | 50 | Nein | Nein |
| | 50 cm | 30 | 10 | 50 | Nein | Nein |
| | 50 cm | 60 | 10 | 50 | Nein | Nein |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 59: Zusammenfassung des LG Bello bei 50cm Abstand

| Google Nexus 4 | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Ja | Ja |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Ja |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Ja | Ja |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Ja | Ja |

Tabelle 60: Zusammenfassung des Google Nexus 4 bei 20cm Abstand

| Google Nexus 4 | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Ja | Ja |
| | 35 cm | 60 | 0 | 0 | Ja | Ja |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Ja | Ja |
| | 35 cm | 60 | 10 | 0 | Ja | Ja |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Ja | Ja |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Ja | Ja |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Ja | Ja |
| | 35 cm | 60 | 10 | 25 | Ja | Ja |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Ja | Ja |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Ja | Ja |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Ja | Ja |
| | 35 cm | 0 | 25 | 50 | Ja | Ja |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Ja | Ja |

Tabelle 61: Zusammenfassung des Google Nexus 4 bei 35cm Abstand

| Google Nexus 4 | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Ja | Ja |
| | 50 cm | 60 | 0 | 0 | Ja | Ja |
| | 50 cm | 0 | 10 | 0 | Ja | Ja |
| | 50 cm | 30 | 10 | 0 | Ja | Ja |
| | 50 cm | 60 | 10 | 0 | Ja | Ja |
| | 50 cm | 0 | 25 | 0 | Ja | Ja |
| | 50 cm | 30 | 25 | 0 | Ja | Ja |
| | 50 cm | 60 | 25 | 0 | Ja | Ja |
| | 50 cm | 0 | 0 | 25 | Ja | Ja |
| | 50 cm | 30 | 0 | 25 | Ja | Ja |
| | 50 cm | 60 | 0 | 25 | Ja | Ja |
| | 50 cm | 0 | 10 | 25 | Ja | Ja |
| | 50 cm | 30 | 10 | 25 | Ja | Ja |
| | 50 cm | 60 | 10 | 25 | Ja | Ja |
| | 50 cm | 0 | 25 | 25 | Ja | Ja |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Ja | Ja |
| | 50 cm | 0 | 0 | 50 | Ja | Ja |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Ja | Ja |
| | 50 cm | 0 | 10 | 50 | Ja | Ja |
| | 50 cm | 30 | 10 | 50 | Ja | Ja |
| | 50 cm | 60 | 10 | 50 | Ja | Ja |
| | 50 cm | 0 | 25 | 50 | Ja | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 62: Zusammenfassung des Google Nexus 4 bei 50cm Abstand

| Motorolla Moto G (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Nein | Nein |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Ja |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Ja | Ja |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 63: Zusammenfassung des Motorola Moto G bei 20cm Abstand

| Motorola Moto G (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Nein | Nein |
| | 35 cm | 60 | 0 | 0 | Nein | Nein |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Nein | Nein |
| | 35 cm | 60 | 10 | 0 | Ja | Ja |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Ja | Ja |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Ja | Ja |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Ja | Ja |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Ja | Ja |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Ja | Ja |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Ja | Ja |
| | 35 cm | 0 | 25 | 50 | Nein | Nein |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 64: Zusammenfassung des Motorola Moto G bei 35cm Abstand

| Motorola Moto G (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Nein | Nein |
| | 50 cm | 60 | 0 | 0 | Ja | Nein |
| | 50 cm | 0 | 10 | 0 | Ja | Nein |
| | 50 cm | 30 | 10 | 0 | Nein | Nein |
| | 50 cm | 60 | 10 | 0 | Nein | Nein |
| | 50 cm | 0 | 25 | 0 | Nein | Nein |
| | 50 cm | 30 | 25 | 0 | Nein | Nein |
| | 50 cm | 60 | 25 | 0 | Nein | Nein |
| | 50 cm | 0 | 0 | 25 | Ja | Nein |
| | 50 cm | 30 | 0 | 25 | Nein | Nein |
| | 50 cm | 60 | 0 | 25 | Nein | Nein |
| | 50 cm | 0 | 10 | 25 | Nein | Nein |
| | 50 cm | 30 | 10 | 25 | Nein | Nein |
| | 50 cm | 60 | 10 | 25 | Nein | Nein |
| | 50 cm | 0 | 25 | 25 | Nein | Nein |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Nein | Nein |
| | 50 cm | 0 | 0 | 50 | Nein | Nein |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Nein | Nein |
| | 50 cm | 0 | 10 | 50 | Nein | Nein |
| | 50 cm | 30 | 10 | 50 | Nein | Nein |
| | 50 cm | 60 | 10 | 50 | Nein | Nein |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 65: Zusammenfassung des Motorola Moto G bei 50cm Abstand

| Oneplus One (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Nein | Nein |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Ja |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Ja | Ja |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 66: Zusammenfassung des OnePlus One bei 20cm Abstand

| Oneplus One (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Ja | Nein |
| | 35 cm | 60 | 0 | 0 | Ja | Ja |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Nein | Nein |
| | 35 cm | 60 | 10 | 0 | Ja | Ja |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Ja | Ja |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Ja | Ja |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Ja | Ja |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Ja | Ja |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Ja | Ja |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Ja | Ja |
| | 35 cm | 0 | 25 | 50 | Ja | Ja |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 67: Zusammenfassung des OnePlus One bei 35cm Abstand

| Oneplus One (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Ja | Ja |
| | 50 cm | 60 | 0 | 0 | Ja | Ja |
| | 50 cm | 0 | 10 | 0 | Ja | Ja |
| | 50 cm | 30 | 10 | 0 | Ja | Nein |
| | 50 cm | 60 | 10 | 0 | Ja | Ja |
| | 50 cm | 0 | 25 | 0 | Ja | Ja |
| | 50 cm | 30 | 25 | 0 | Ja | Ja |
| | 50 cm | 60 | 25 | 0 | Ja | Ja |
| | 50 cm | 0 | 0 | 25 | Ja | Ja |
| | 50 cm | 30 | 0 | 25 | Ja | Ja |
| | 50 cm | 60 | 0 | 25 | Ja | Ja |
| | 50 cm | 0 | 10 | 25 | Ja | Ja |
| | 50 cm | 30 | 10 | 25 | Ja | Nein |
| | 50 cm | 60 | 10 | 25 | Ja | Nein |
| | 50 cm | 0 | 25 | 25 | Ja | Ja |
| | 50 cm | 30 | 25 | 25 | Ja | Nein |
| | 50 cm | 60 | 25 | 25 | Ja | Ja |
| | 50 cm | 0 | 0 | 50 | Ja | Ja |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Ja | Ja |
| | 50 cm | 0 | 10 | 50 | Ja | Ja |
| | 50 cm | 30 | 10 | 50 | Ja | Ja |
| | 50 cm | 60 | 10 | 50 | Ja | Ja |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 68: Zusammenfassung des OnePlus One bei 50cm Abstand

| Samsung Galaxy S5 Neo (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Ja | Ja |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Ja |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Ja | Ja |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 69: Zusammenfassung des Samsung Galaxy S5 Neo bei 20cm Abstand

| Samsung Galaxy S5 Neo (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Ja | Nein |
| | 35 cm | 60 | 0 | 0 | Ja | Ja |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Ja | Ja |
| | 35 cm | 60 | 10 | 0 | Ja | Ja |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Ja | Ja |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Ja | Ja |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Ja | Ja |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Ja | Ja |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Ja | Ja |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Ja | Ja |
| | 35 cm | 0 | 25 | 50 | Ja | Ja |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 70: Zusammenfassung des Samsung Galaxy S5 Neo bei 35cm Abstand

| Samsung Galaxy S5 Neo (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Ja | Ja |
| | 50 cm | 60 | 0 | 0 | Ja | Ja |
| | 50 cm | 0 | 10 | 0 | Ja | Ja |
| | 50 cm | 30 | 10 | 0 | Ja | Ja |
| | 50 cm | 60 | 10 | 0 | Ja | Ja |
| | 50 cm | 0 | 25 | 0 | Ja | Ja |
| | 50 cm | 30 | 25 | 0 | Ja | Ja |
| | 50 cm | 60 | 25 | 0 | Ja | Ja |
| | 50 cm | 0 | 0 | 25 | Ja | Ja |
| | 50 cm | 30 | 0 | 25 | Ja | Ja |
| | 50 cm | 60 | 0 | 25 | Ja | Ja |
| | 50 cm | 0 | 10 | 25 | Ja | Ja |
| | 50 cm | 30 | 10 | 25 | Ja | Ja |
| | 50 cm | 60 | 10 | 25 | Ja | Ja |
| | 50 cm | 0 | 25 | 25 | Ja | Ja |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Ja | Ja |
| | 50 cm | 0 | 0 | 50 | Ja | Ja |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Ja | Nein |
| | 50 cm | 0 | 10 | 50 | Ja | Ja |
| | 50 cm | 30 | 10 | 50 | Ja | Ja |
| | 50 cm | 60 | 10 | 50 | Ja | Ja |
| | 50 cm | 0 | 25 | 50 | Ja | Ja |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 71: Zusammenfassung des Samsung Galaxy S5 Neo bei 50cm Abstand

| Sony Xperia E1 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Nein | Nein |
| | 20 cm | 60 | 0 | 0 | Nein | Nein |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Nein | Nein |
| | 20 cm | 60 | 10 | 0 | Nein | Nein |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Nein | Nein |
| | 20 cm | 60 | 25 | 0 | Nein | Nein |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Nein | Nein |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Nein | Nein |
| | 20 cm | 60 | 10 | 25 | Nein | Nein |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Nein | Nein |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Nein | Nein |
| | 20 cm | 0 | 10 | 50 | Ja | Nein |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Nein | Nein |
| | 20 cm | 0 | 25 | 50 | Nein | Nein |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 72: Zusammenfassung des Sony Xperia E1 bei 20cm Abstand

| Sony Xperia E1 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Nein | Nein |
| | 35 cm | 30 | 0 | 0 | Nein | Nein |
| | 35 cm | 60 | 0 | 0 | Nein | Nein |
| | 35 cm | 0 | 10 | 0 | Nein | Nein |
| | 35 cm | 30 | 10 | 0 | Nein | Nein |
| | 35 cm | 60 | 10 | 0 | Nein | Nein |
| | 35 cm | 0 | 25 | 0 | Nein | Nein |
| | 35 cm | 30 | 25 | 0 | Nein | Nein |
| | 35 cm | 60 | 25 | 0 | Nein | Nein |
| | 35 cm | 0 | 0 | 25 | Nein | Nein |
| | 35 cm | 30 | 0 | 25 | Nein | Nein |
| | 35 cm | 60 | 0 | 25 | Nein | Nein |
| | 35 cm | 0 | 10 | 25 | Nein | Nein |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Nein | Nein |
| | 35 cm | 0 | 25 | 25 | Nein | Nein |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Nein | Nein |
| | 35 cm | 0 | 0 | 50 | Nein | Nein |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Nein | Nein |
| | 35 cm | 0 | 10 | 50 | Nein | Nein |
| | 35 cm | 30 | 10 | 50 | Nein | Nein |
| | 35 cm | 60 | 10 | 50 | Nein | Nein |
| | 35 cm | 0 | 25 | 50 | Nein | Nein |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 73: Zusammenfassung des Sony Xperia E1 bei 35cm Abstand

| Sony Xperia E1 (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Nein | Nein |
| | 50 cm | 30 | 0 | 0 | Nein | Nein |
| | 50 cm | 60 | 0 | 0 | Nein | Nein |
| | 50 cm | 0 | 10 | 0 | Nein | Nein |
| | 50 cm | 30 | 10 | 0 | Nein | Nein |
| | 50 cm | 60 | 10 | 0 | Nein | Nein |
| | 50 cm | 0 | 25 | 0 | Nein | Nein |
| | 50 cm | 30 | 25 | 0 | Nein | Nein |
| | 50 cm | 60 | 25 | 0 | Nein | Nein |
| | 50 cm | 0 | 0 | 25 | Nein | Nein |
| | 50 cm | 30 | 0 | 25 | Nein | Nein |
| | 50 cm | 60 | 0 | 25 | Nein | Nein |
| | 50 cm | 0 | 10 | 25 | Nein | Nein |
| | 50 cm | 30 | 10 | 25 | Nein | Nein |
| | 50 cm | 60 | 10 | 25 | Nein | Nein |
| | 50 cm | 0 | 25 | 25 | Nein | Nein |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Nein | Nein |
| | 50 cm | 0 | 0 | 50 | Nein | Nein |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Nein | Nein |
| | 50 cm | 0 | 10 | 50 | Nein | Nein |
| | 50 cm | 30 | 10 | 50 | Nein | Nein |
| | 50 cm | 60 | 10 | 50 | Nein | Nein |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 74: Zusammenfassung des Sony Xperia E1 bei 50cm Abstand

| Sony Xperia Tipo (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Ja | Ja |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Nein |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Nein | Nein |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 75: Zusammenfassung des Sony Xperia Tipo bei 20cm Abstand

| Sony Xperia Tipo (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Nein | Nein |
| | 35 cm | 30 | 0 | 0 | Nein | Nein |
| | 35 cm | 60 | 0 | 0 | Nein | Nein |
| | 35 cm | 0 | 10 | 0 | Nein | Nein |
| | 35 cm | 30 | 10 | 0 | Nein | Nein |
| | 35 cm | 60 | 10 | 0 | Nein | Nein |
| | 35 cm | 0 | 25 | 0 | Nein | Nein |
| | 35 cm | 30 | 25 | 0 | Nein | Nein |
| | 35 cm | 60 | 25 | 0 | Nein | Nein |
| | 35 cm | 0 | 0 | 25 | Nein | Nein |
| | 35 cm | 30 | 0 | 25 | Nein | Nein |
| | 35 cm | 60 | 0 | 25 | Nein | Nein |
| | 35 cm | 0 | 10 | 25 | Nein | Nein |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Nein | Nein |
| | 35 cm | 0 | 25 | 25 | Nein | Nein |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Nein | Nein |
| | 35 cm | 0 | 0 | 50 | Nein | Nein |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Nein | Nein |
| | 35 cm | 0 | 10 | 50 | Nein | Nein |
| | 35 cm | 30 | 10 | 50 | Nein | Nein |
| | 35 cm | 60 | 10 | 50 | Nein | Nein |
| | 35 cm | 0 | 25 | 50 | Nein | Nein |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 76: Zusammenfassung des Sony Xperia Tipo bei 35cm Abstand

| Sony Xperia Tipo (Low-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Nein | Nein |
| | 50 cm | 30 | 0 | 0 | Nein | Nein |
| | 50 cm | 60 | 0 | 0 | Nein | Nein |
| | 50 cm | 0 | 10 | 0 | Nein | Nein |
| | 50 cm | 30 | 10 | 0 | Nein | Nein |
| | 50 cm | 60 | 10 | 0 | Nein | Nein |
| | 50 cm | 0 | 25 | 0 | Nein | Nein |
| | 50 cm | 30 | 25 | 0 | Nein | Nein |
| | 50 cm | 60 | 25 | 0 | Nein | Nein |
| | 50 cm | 0 | 0 | 25 | Nein | Nein |
| | 50 cm | 30 | 0 | 25 | Nein | Nein |
| | 50 cm | 60 | 0 | 25 | Nein | Nein |
| | 50 cm | 0 | 10 | 25 | Nein | Nein |
| | 50 cm | 30 | 10 | 25 | Nein | Nein |
| | 50 cm | 60 | 10 | 25 | Nein | Nein |
| | 50 cm | 0 | 25 | 25 | Nein | Nein |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Nein | Nein |
| | 50 cm | 0 | 0 | 50 | Nein | Nein |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Nein | Nein |
| | 50 cm | 0 | 10 | 50 | Nein | Nein |
| | 50 cm | 30 | 10 | 50 | Nein | Nein |
| | 50 cm | 60 | 10 | 50 | Nein | Nein |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 77: Zusammenfassung des Sony Xperia Tipo bei 50cm Abstand

| Sony Xperia Z3 Compact (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 20 cm | 0 | 0 | 0 | Ja | Ja |
| | 20 cm | 30 | 0 | 0 | Ja | Ja |
| | 20 cm | 60 | 0 | 0 | Ja | Ja |
| | 20 cm | 0 | 10 | 0 | Ja | Ja |
| | 20 cm | 30 | 10 | 0 | Ja | Ja |
| | 20 cm | 60 | 10 | 0 | Ja | Ja |
| | 20 cm | 0 | 25 | 0 | Ja | Ja |
| | 20 cm | 30 | 25 | 0 | Ja | Ja |
| | 20 cm | 60 | 25 | 0 | Ja | Ja |
| | 20 cm | 0 | 0 | 25 | Ja | Ja |
| | 20 cm | 30 | 0 | 25 | Ja | Ja |
| | 20 cm | 60 | 0 | 25 | Ja | Ja |
| | 20 cm | 0 | 10 | 25 | Ja | Ja |
| | 20 cm | 30 | 10 | 25 | Ja | Ja |
| | 20 cm | 60 | 10 | 25 | Ja | Ja |
| | 20 cm | 0 | 25 | 25 | Ja | Ja |
| | 20 cm | 30 | 25 | 25 | Nein | Nein |
| | 20 cm | 60 | 25 | 25 | Ja | Ja |
| | 20 cm | 0 | 0 | 50 | Ja | Ja |
| | 20 cm | 30 | 0 | 50 | Nein | Nein |
| | 20 cm | 60 | 0 | 50 | Ja | Ja |
| | 20 cm | 0 | 10 | 50 | Ja | Ja |
| | 20 cm | 30 | 10 | 50 | Ja | Ja |
| | 20 cm | 60 | 10 | 50 | Ja | Ja |
| | 20 cm | 0 | 25 | 50 | Ja | Ja |
| | 20 cm | 30 | 25 | 50 | Nein | Nein |
| | 20 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 78: Zusammenfassung des Sony Xperia Z3 Compact bei 20cm Abstand

| Sony Xperia Z3 Compact (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 35 cm | 0 | 0 | 0 | Ja | Ja |
| | 35 cm | 30 | 0 | 0 | Ja | Ja |
| | 35 cm | 60 | 0 | 0 | Ja | Ja |
| | 35 cm | 0 | 10 | 0 | Ja | Ja |
| | 35 cm | 30 | 10 | 0 | Ja | Ja |
| | 35 cm | 60 | 10 | 0 | Ja | Ja |
| | 35 cm | 0 | 25 | 0 | Ja | Ja |
| | 35 cm | 30 | 25 | 0 | Ja | Ja |
| | 35 cm | 60 | 25 | 0 | Ja | Ja |
| | 35 cm | 0 | 0 | 25 | Ja | Ja |
| | 35 cm | 30 | 0 | 25 | Ja | Ja |
| | 35 cm | 60 | 0 | 25 | Ja | Ja |
| | 35 cm | 0 | 10 | 25 | Ja | Ja |
| | 35 cm | 30 | 10 | 25 | Nein | Nein |
| | 35 cm | 60 | 10 | 25 | Ja | Ja |
| | 35 cm | 0 | 25 | 25 | Ja | Ja |
| | 35 cm | 30 | 25 | 25 | Nein | Nein |
| | 35 cm | 60 | 25 | 25 | Ja | Ja |
| | 35 cm | 0 | 0 | 50 | Ja | Ja |
| | 35 cm | 30 | 0 | 50 | Nein | Nein |
| | 35 cm | 60 | 0 | 50 | Ja | Ja |
| | 35 cm | 0 | 10 | 50 | Ja | Ja |
| | 35 cm | 30 | 10 | 50 | Ja | Ja |
| | 35 cm | 60 | 10 | 50 | Ja | Ja |
| | 35 cm | 0 | 25 | 50 | Nein | Nein |
| | 35 cm | 30 | 25 | 50 | Nein | Nein |
| | 35 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 79: Zusammenfassung des Sony Xperia Z3 Compact bei 35cm Abstand

| Sony Xperia Z3 Compact (High-Cost) | Entfernung | Winkel | | | Erkannt? | |
|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 8 | Sprint 9 |
|  | 50 cm | 0 | 0 | 0 | Ja | Ja |
| | 50 cm | 30 | 0 | 0 | Ja | Ja |
| | 50 cm | 60 | 0 | 0 | Ja | Ja |
| | 50 cm | 0 | 10 | 0 | Ja | Ja |
| | 50 cm | 30 | 10 | 0 | Ja | Ja |
| | 50 cm | 60 | 10 | 0 | Ja | Ja |
| | 50 cm | 0 | 25 | 0 | Ja | Ja |
| | 50 cm | 30 | 25 | 0 | Nein | Nein |
| | 50 cm | 60 | 25 | 0 | Ja | Ja |
| | 50 cm | 0 | 0 | 25 | Ja | Ja |
| | 50 cm | 30 | 0 | 25 | Ja | Ja |
| | 50 cm | 60 | 0 | 25 | Ja | Nein |
| | 50 cm | 0 | 10 | 25 | Ja | Ja |
| | 50 cm | 30 | 10 | 25 | Nein | Nein |
| | 50 cm | 60 | 10 | 25 | Ja | Ja |
| | 50 cm | 0 | 25 | 25 | Ja | Ja |
| | 50 cm | 30 | 25 | 25 | Nein | Nein |
| | 50 cm | 60 | 25 | 25 | Ja | Ja |
| | 50 cm | 0 | 0 | 50 | Ja | Ja |
| | 50 cm | 30 | 0 | 50 | Nein | Nein |
| | 50 cm | 60 | 0 | 50 | Nein | Nein |
| | 50 cm | 0 | 10 | 50 | Nein | Nein |
| | 50 cm | 30 | 10 | 50 | Nein | Nein |
| | 50 cm | 60 | 10 | 50 | Ja | Nein |
| | 50 cm | 0 | 25 | 50 | Nein | Nein |
| | 50 cm | 30 | 25 | 50 | Nein | Nein |
| | 50 cm | 60 | 25 | 50 | Nein | Nein |

Tabelle 80: Zusammenfassung des Sony Xperia Z3 Compact bei 50cm Abstand

### 4.8.4 Fertigstellung der iOS-Applikation

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Das Arbeitspaket soll dazu dienen, die iOS-Applikation entsprechend der Kundenanforderungen aus dem Lastenheft fertig zu stellen.
- Arbeitspaketbeschreibung:
  Am Ende dieses Arbeitspaketes ist die iOS-Applikation entsprechend aller Anforderungen laut Lastenheft fertig gestellt. Dazu müssen folgende Aufgaben abgearbeitet werden:
  - Integration des QR-Code-Readers
  - Bereitstellung der Informationen entsprechend der Teststreifenart in den Nutzer-Workflow
  - Integration der Hilfestellungs-Videos
  - Durchführung von ausführlichen Tests, um die Qualität der Applikation sicher zu stellen
  - Noch nicht betrachtete Diagnosen sollten in der Liste hervorgehoben werden
  - Benachrichtigungen sollen den Nutzer direkt zur richtigen Ansicht bringen
  - Die Liste der Diagnose soll einen Platzanhalter anzeigen wenn keine Einträge vorhanden sind
  - Integration von englischer Sprache
  - Die Detailansichten der Diagnosen müssen gestaltet werden
  - Das Befüllen der Diagnoseliste mit Inhalt darf nicht zum Stillstand iOS-App führen
- Vorbedingungen:
  -

- Nebenbedingungen:
  -
- Nachbedingungen:
  Alle offenen Anforderungen sind in die Applikation integriert und alle Fehler behoben.
- Aufwand:
  sechs Wochen
- Personen:
  - Timo Raß
  - Timo Schlömer

**Dokumentation** Im folgenden Teil der Dokumentation wird die Fertigstellung der iOS-Applikation genauer beschrieben. Dabei wird insbesondere auf den neuen Aufbau der Benutzeroberfläche und die Integration der letzten Anforderungen eingegangen.

**Ablauf** Der Ablauf zur Fertigstellung der mobilen iOS-Applikation gliederte sich in die folgenden Teilbereiche:

**Änderung der Darstellungsarchitektur in eine Tabellenform:** Da es in der iOS-Applikation immer wieder zu Darstellungsfehlern kam, wurde jede Ansicht, exklusive der Kameraansichten, in Form von Tabellen aufgebaut. Die tabellarische Darstellung ist eine State of the Art Methode und wird nicht mehr über das Storyboard in Xcode erzeugt, sondern über speziell entworfene Klassen und Programmcode generiert. Somit wurde die Darstellungs-Architektur der Applikation von Grund auf neu entworfen. Diese Form der Darstellung brachte die folgenden positive Effekte mit sich:

- Grafiken und Texte werden auf allen iOS-Endgeräten nun korrekt dargestellt und nicht abgeschnitten oder falsch skaliert.
- Je nach der Displaygröße des Endgerätes, kann der Bildlauf der einzelnen Ansichten nach oben und unten bewegt werden. Diese Funktion stellt iOS automatisch bereit.
- Die Zellen für variable Texte, welche z. B. durch einen Arzt verfasst werden, können nun beliebig lang sein und sind korrekt dargestellt.
- Inhalte in allen Ansichten können um beliebig viel Inhalt ergänzt und durch den dynamischen Bildlauf eingesehen werden.

Abbildung 259 auf der nächsten Seite zeigt die neue Darstellung. Die Trennlinien der einzelnen Zellen sind aus ästhetischen Gründen ausgeblendet. Zudem wurde die Unterschrift-Funktion eingefügt, welche dem Nutzer zeigt, in welchem Schritt der Testdurchführung er sich befindet. Diese wird im iOS-Betriebssystem standardmäßig über und nicht unter der Überschrift dargestellt.

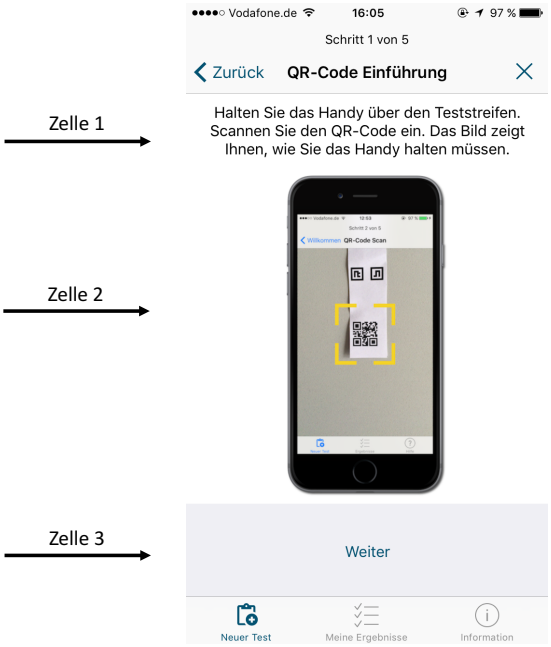



Abbildung 259: Ansicht der iOS-Applikation in Tabellenform

**Änderung in der Nutzerführung zur Durchführung eines neuen Testes:** Die Benutzerführung wurde weiter verbessert. Es wurden zusätzliche Schritte eingefügt, um die Testdurchführung noch intuitiver zu gestalten und eine Fehlbedienung vom Nutzer auszuschließen. Die Nutzerführung, um einen neuen Test durchzuführen, läuft wie folgt ab:

- Eine Willkommens-Ansicht, welche den Nutzer informiert, dass er diese Applikation nutzen kann, wenn er einen MEDIC-Teststreifen erhalten hat.
- Schritt eins von fünf erklärt dem Nutzer wie der QR-Code des Teststreifens einzuscannen ist und dass nach dem Scan weitere Informationen zum Test und zur Durchführung bereit gestellt werden.
- Die nächste Ansicht zeigt die Kamera mit einem Fadenkreuz, in welchem der QR-Code positioniert werden muss. Wenn der Teststreifen erkannt wurde, vibriert das Endgerät, lädt das Teststreifenschema und führt den Test automatisch fort. Das Schema enthält alle Metadaten für den Test. Die Ansicht zum Scannen des QR-Codes war in den vorherigen Sprints noch nicht vorhanden und wurde implementiert.
- In Schritt zwei von fünf werden dem Nutzer Informationen zu der Testart gegeben, z. B. für welche Krankheit der Test vorgesehen ist. Zudem wird das angezeigte Teststreifenbild nicht in Form eines statischen Bildes dargestellt sondern immer, je nach Aufbau des Teststreifens, neu gezeichnet. Das Bild des Teststreifens in der Nutzerführung entspricht somit dem Teststreifen, welcher eingescannt wurde (vgl. Abbildung 259). Dies ist aufgrund der dynamischen Teststreifendesigns sehr vorteilhaft und führt den Nutzer so nicht in die Irre. Die notwendigen Daten, um den Teststreifen zu zeichnen, werden mit dem Teststreifenschema ausgeliefert und stehen nach dem Scannen des QR-Codes bereit. Die Informationen zum Test werden durch das Teststreifenschema bereitgestellt und von einem Mediziner verfasst.


Abbildung 260: In der iOS-Applikation gezeichneter Teststreifen

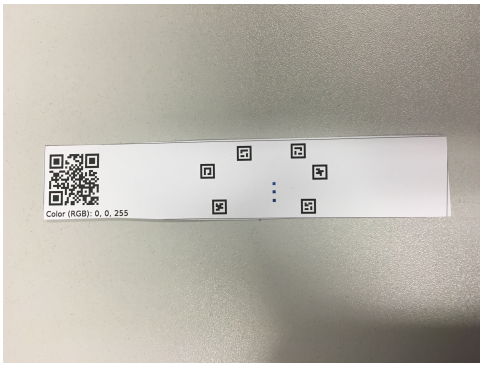



Abbildung 261: Originales Foto des realen Teststreifens

- In Schritt drei von fünf wird dem Nutzer erklärt, welche Flüssigkeit wo auf dem Teststreifen aufgetragen werden muss. Dies wird auf dem Bild des Teststreifens ebenfalls dynamisch dargestellt und von der iOS-Applikation eingezeichnet (vgl. Abbildung 262 auf der nächsten Seite). Die Koordinaten für den Bereich des zu applizierenden Sekrets und die Beschreibung zur Durchführung des Testes werden durch das Teststreifenschema bereitgestellt und von einem Mediziner in der Weboberfläche verfasst. Nachdem der Nutzer das Sekret auf den Test appliziert hat, muss er in Schritt vier von fünf eine von einem Mediziner definierte Zeit warten, bis eine Reaktion auf dem Teststreifen vollzogen ist. Erst danach kann dieser ausgewertet werden. Die Zeit wird durch einen Timer visualisiert (vgl. Abbildung 263 auf der nächsten Seite). Die zu wartende Zeit ist spezifisch für eine Testart und wird mit dem Teststreifenschema ausgeliefert.


Abbildung 262: Eingezeichneter Bereich, um Sekret auf den Teststreifen aufzutragen (rot)

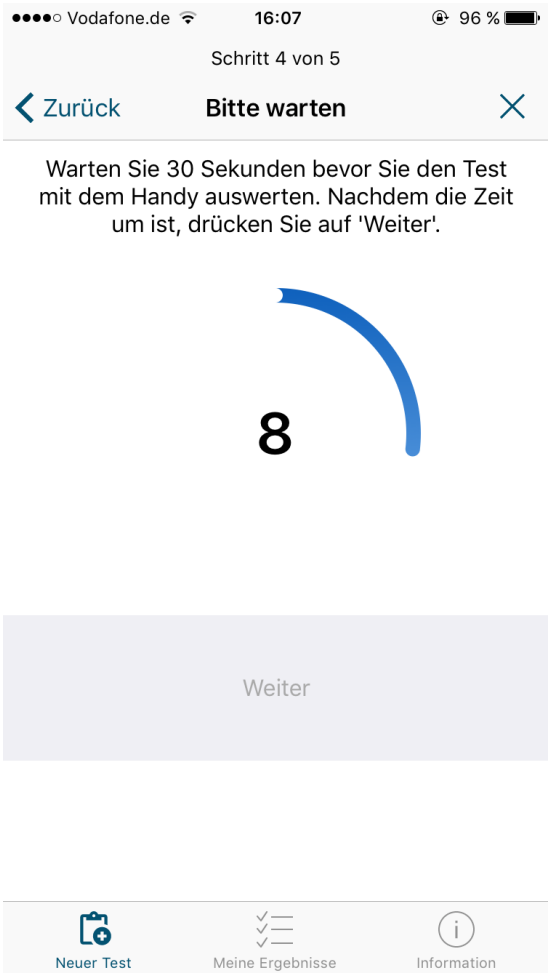



Abbildung 263: Timer-Ansicht, bevor der Test ausgewertet werden kann

- Nachdem die Reaktion auf dem Teststreifen stattfinden konnte, wird der Nutzer in Schritt fünf von fünf darüber informiert, wie er den Teststreifen einscannen muss, damit dieser ausgewertet werden kann (vgl. Abbildung 264 auf der nächsten Seite). Der Text und das Bild in dieser Ansicht werden nicht durch das Teststreifenschema bereitgestellt, da die Anleitung für jeden Test identisch ist.


Abbildung 264: Erklärung für den Nutzer, wie der Test für die Analyse eingescannt werden muss

- Als letztes muss der Nutzer den Test einscannen. Dafür muss der Teststreifen im innerhalb des gelben Rechtecks positioniert werden (vgl. Abbildung 265 auf der nächsten Seite). Das Endgerät vibriert und es ertönt ein akustisches Signal, wenn der Teststreifen erfolgreich erkannt wurde. Nachdem die Analyse abgeschlossen wurde erscheint automatisch die Ergebnis-Ansicht, welche dem Nutzer über seine Befindlichkeiten informiert (vgl. Abbildung 266 auf der nächsten Seite). Diese wurde gänzlich neu entworfen.

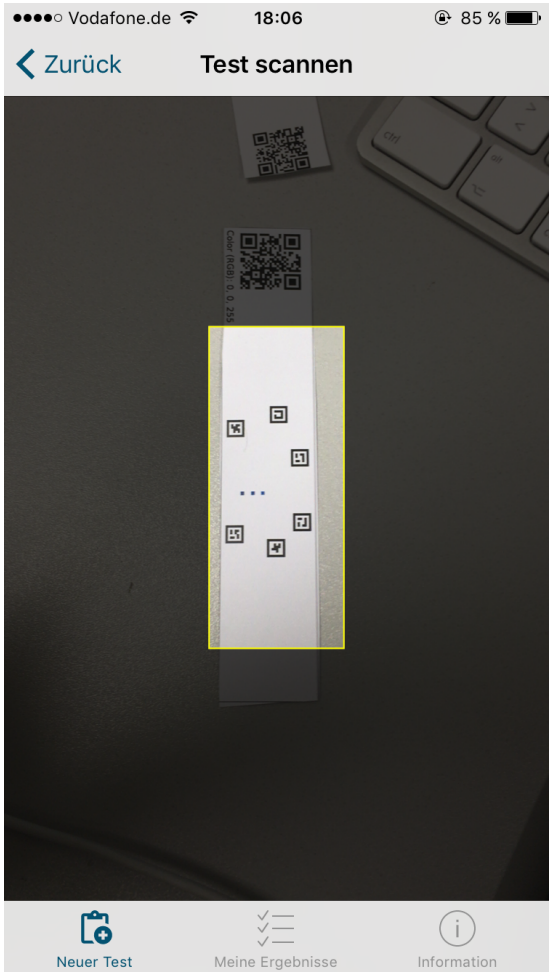



Abbildung 265: Ansicht der Analyse und Auswertung des Teststreifens

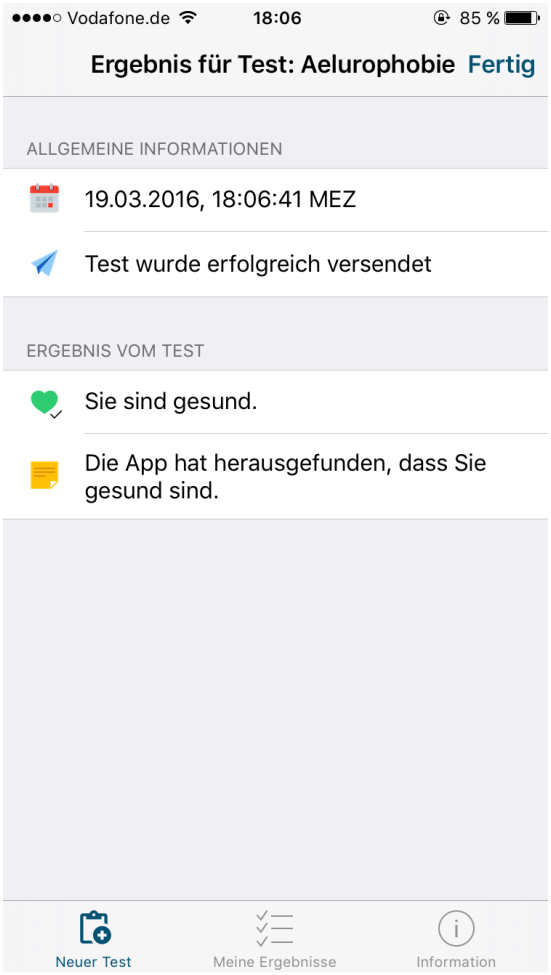



Abbildung 266: Ergebnis-Ansicht direkt nach der Analyse

**Erneute Integration von deutscher Sprache für die Accessibility:** Die Texte für die Accessibility der Applikation mussten neu integriert werden, da die Benutzeroberfläche nicht mehr durch das Storyboard, sondern im Programmcode generiert wird. Mit den unter iOS zur Verfügung stehenden Bedienungshilfen können beliebige Bezeichner, Hinweise und normale Texte integriert werden, welche vom Screenreader vorgelesen werden. Zudem wurden im Programmcode unnötige Steuerungselemente für die Blindensteuerung deaktiviert. Ein Arzt kann in der Weboberfläche neben dem dargestellten Text zur Testdurchführung auch gesondert eine Beschreibung für Sehbehinderte verfassen. Diese wird anstelle des dargestellten Textes durch den Screenreader vorgelesen.

**Integration von englischer Sprache:** Nachdem die gesamte Applikation neu gestaltet wurde, konnte damit begonnen werden, Englisch als Zweitsprache zu integrieren. Dafür wurden im Programmcode statt statischen Texten Platzhalter verwendet. Diese können für verschiedene Sprachen, unabhängig vom Quellcode, in separaten Datei übersetzt werden. Je nachdem welche Sprache im Betriebssystem eingestellt ist werden die entsprechenden Texte dargestellt. Beschreibungen zu einem Test, welche von einem Arzt im Teststreifenschema verfasst werden, müssen ebenfalls in deutscher und englischer Sprache geschrieben sein.

**Gestaltung der Ergebnis-Ansichten:** Die Liste der Testergebnisse (vgl. Abbildung 267) und die detaillierte Ansicht eines Tests (vgl. Abbildung 268) wurden ebenfalls neu entworfen. Sie werden in tabellarischer Form dargestellt und können dynamisch Daten, je nach Testergebnis, darstellen. Abbildung 267 zeigt zudem, dass neue Testdiagnosen hervorgehoben angezeigt werden. Die Schrift wird fett dargestellt und es wird ein blauer Kreis angezeigt. Dieser wird im iOS-Betriebssystem verwendet, um zu visualisieren, dass neue Inhalte zur Verfügung stehen.

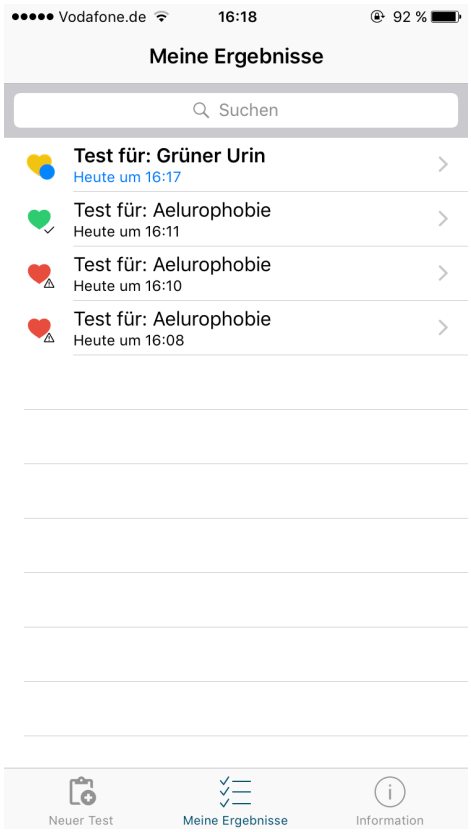



Abbildung 267: Liste der Testergebnisse

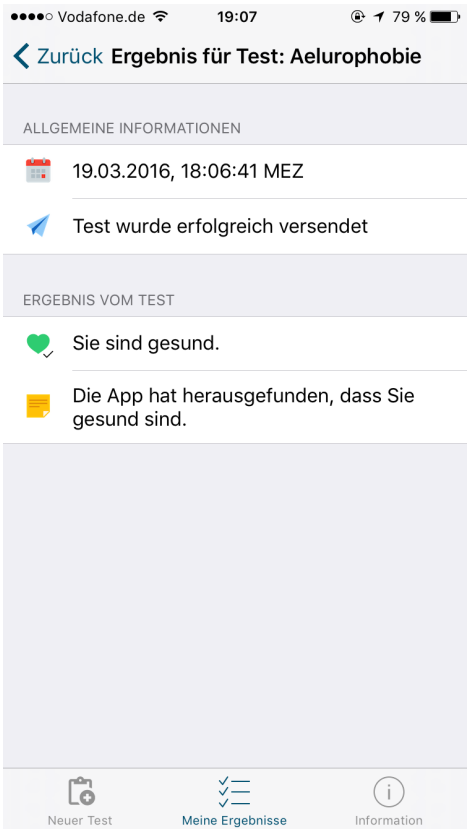



Abbildung 268: Detaillierte Ansicht eines ausgewählten Tests

Wenn in der Listenansicht ein Test nach links gewischt wird, kann der Test gelöscht werden. Eine Suchoption wurde integriert, um die Liste der durchgeführten Tests zu filtern (vgl. Abbildung 269). Wenn in der Ergebnisliste keine Tests vorhanden sind, wird ein Platzhalter angezeigt (vgl. Abbildung 270).


Abbildung 269: Einen Test löschen in der Ergebnis-Ansicht


Abbildung 270: Platzhalter, falls keine Tests vorhanden sind

**Gestaltung der Informationsansicht:** In den vergangenen Versionen der iOS-Applikation war in der Informationsansicht nur die Adresse der Abteilung AMiR vorhanden. Nach der Überarbeitung dieser Ansicht sind nun das Logo, die Versionsnummer, rechtliche Hinweise, ein Link zur Abteilung AMiR, eine Möglichkeit einen E-Mail Kontakt herzustellen, die Lizenzen der verwendeten Bibliotheken und ein Hilfe-Video integriert worden (vgl. Abbildung 271 und 272).

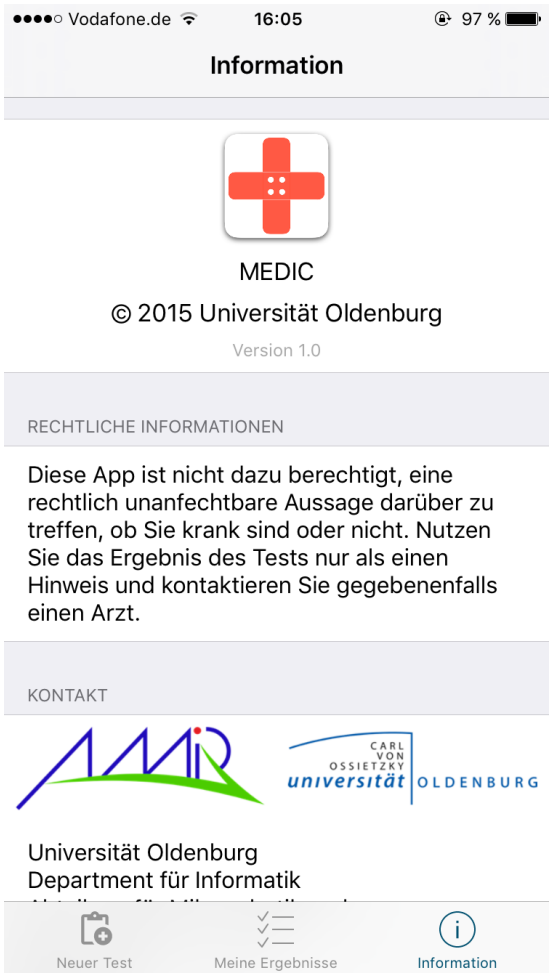



Abbildung 271: Informations-Ansicht Teil 1

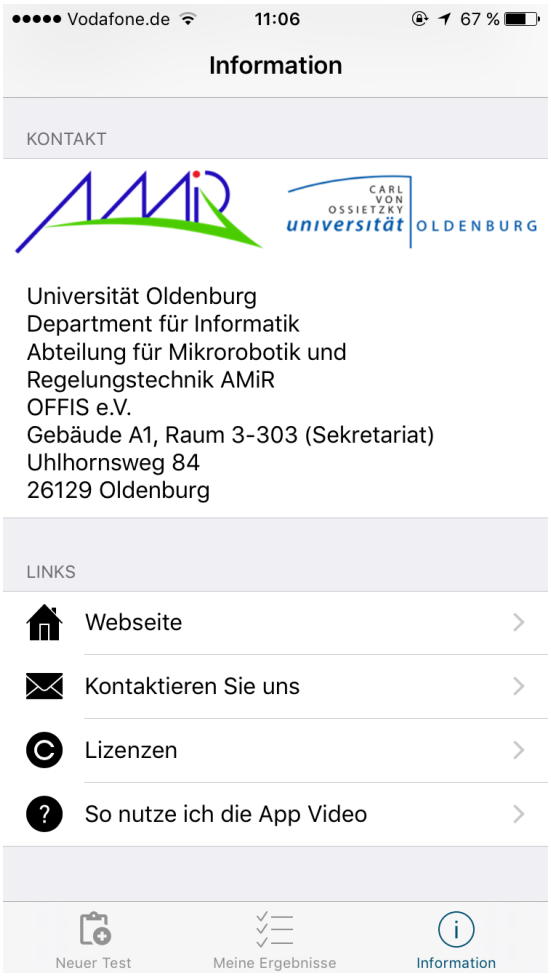



Abbildung 272: Informations-Ansicht Teil 2

Für die Verlinkung der AMiR-Webseite wurde eine in iOS 9 eingeführte Funktion verwendet, um eine vereinfachte Version des Safari-Browsers innerhalb der Applikation darzustellen. Durch diese ist es möglich, eine Webseite zu öffnen, ohne dass die Applikation geschlossen werden muss. Diese Ansicht enthält die wichtigsten Safari-Funktionen und wird über die Informations-Ansicht gelegt (vgl. Abbildung 273 auf der nächsten Seite). Zudem ist es möglich im Punkt Lizenzen, die Webseiten aller verwendeten Bibliotheken aufzurufen.


Abbildung 273: Ansicht des Safari-Browsers innerhalb der Medic-Applikation

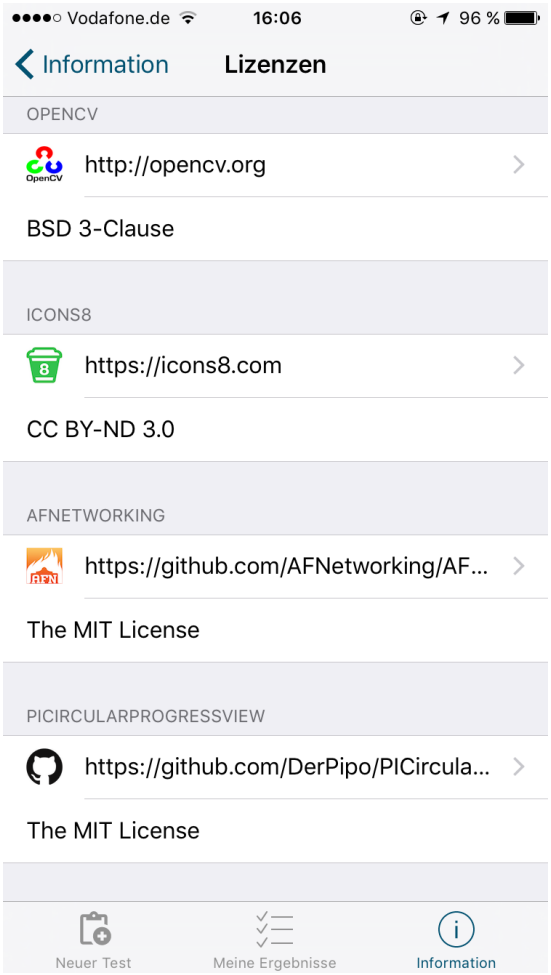



Abbildung 274: Ansicht der verwendeten Bibliothekslizenzen

**Erstellung und Integration eines Hilfe-Videos:** Um dem Nutzer eine weitere Hilfestellung zu geben und so die Wahrscheinlichkeit einer Fehlbedienung zu verringern, wurde ein Hilfe-Video mit Animationen erstellt. Dies kann unter dem Menüpunkt „So nutze ich die App Video “ aufgerufen werden (vgl. 272 auf der vorherigen Seite). In diesem wird dem Nutzer, in einfacher Sprache, Schritt für Schritt, jede Ansicht der Applikation verständlich gemacht. Zudem wird explizit darauf eingegangen, wie ein Test durchgeführt wird auf was besonders zu achten ist (z. B. das der Teststreifen nicht geknickt oder verdeckt sein darf, wenn er für die Analyse eingescannt wird).

### Technologien

**SafariViewController** Der in iOS 9 eingeführte Safari View Controller ermöglicht es, eine Safari-Browser-Ansicht in eine Applikation zu integrieren. Die Applikation muss nicht mehr verlassen werden, wenn auf einen Link verwiesen wird. Zudem stehen die meisten Funktionen des Safari-Browsers zur Verfügung.

**TableViewController** Der Table View Controller beinhaltet die Logik und Darstellungsparameter für den Aufbau einer Tabelle. Die iOS-Applikation wurde grundlegend nach dem Konzept der tabellarischen Ansichten erneuert.

**AVPlayerViewController** Der AV Player View Controller wurde verwendet, um ein Hilfe-Video in die Applikation zu integrieren. Er erzeugt eine Media Player Ansicht und spielt das Video

automatisch ab. Die gängigen Steuerungselemente für die Videowiedergabe werden von diesem bereit gestellt.

**PICircularProgressView** Die PI Circular Progress View ermöglicht es, einen runden, sich füllenden Ladebalken zu erzeugen. Dieser wird für die Timer-Ansicht benötigt.

**UAObfuscatedString** Die UAObfuscatedString Bibliothek ermöglicht das „unkenntlich machen“ von Textkonstanten innerhalb des Programmcodes. So werden die Konstanten durch Aufrufe von Methoden ersetzt, welche den selben Text produzieren, aber das auslesen durch Angreifer erschweren. So werden wichtige Textkonstanten wie etwa API-Schlüssel gesichert.

**Systembeschreibung** Die Entwicklung wurde unter einem Mac mini mit Xcode 7.2 für das Betriebssystem iOS 9.2 durchgeführt.

**Evaluation** Die iOS Applikation wurde von allen Teammitgliedern sowie von zehn externen Personen getestet. Aufgetretene Fehler wurden sofort behoben. Die Applikation ist ohne Einschränkungen verwendbar. Als Testgeräte wurden ein iPad mini 3, ein iPhone 5 und ein iPhone 6s verwendet.

**Parameter** Die Funktion der App wird durch keine Parameter eingeschränkt.

**Fazit** Alle angefallenen Aufgaben und Anforderungen konnten erfolgreich in die Applikation integriert werden. Diese ist somit in einem auslieferbaren Zustand und kann dem Kunden, inklusive Handbuch, am 31.03.2016 übergeben werden.

**Ausblick** Die Entwicklung konnte erfolgreich abgeschlossen werden.

### 4.8.5 Fertigstellung des Serverbackends

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Dieses Arbeitspaket soll dazu dienen, den Go-Server des Backends für die Auslieferung an den Kunden zu finalisieren.
- Arbeitspaketbeschreibung:
  Es muss sich in diesem Arbeitspaket mit dem Webfrontend Arbeitspaket und denen der mobilen Applikationen abgesprochen werden, um ein vollständiges Backend mit allen benötigten Funktionen bereitzustellen.

  Am Ende muss das Backend folgende Anforderungen implementieren:

  - Korrekte Rückgabe bei fehlerhaften Eingaben. (Fehlermeldung und Status-Code)
  - Unterstützung von Internationalisierung für die Teststreifenbeschreibung
  - Unterstützung von Internationalisierung für die Accessability-Teststreifenbeschreibung
  - Unterscheidung der verschiedenen Applizierungstypen
- Vorbedingungen:
  -
- Nebenbedingungen:
  -

- Nachbedingungen:
  Alle offenen Anforderungen sind in das Backend integriert und alle Fehler behoben.
- Aufwand:
  sechs Wochen
- Personen:
  - Sebastian Horwege
  - Danny Fonk

**Dokumentation** Für die Auslieferung der Komponente „Go-Server-Backend“ an den Endkunden sind finale Änderungen vonnöten. Diese Änderungen betreffen einen sich gewandelten Prozessablauf zur Auslieferung von Teststreifendaten, sowie das Empfangen von durchgeführten Tests auf den mobilen Endgeräten. Des Weiteren sind während der vorangegangenen Projektphase diverse Probleme aufgetreten, die im Zuge dieses Arbeitspaketes behoben wurden.

**Ablauf** Nachdem innerhalb der Projektgruppe alle ausstehenden oder sich gewandelten Anforderungen identifiziert wurden, konnten die an das Server-Backend gestellten Anforderungen formuliert werden. Das Datum, welches von den mobilen Applikationen an den Server übermittelt wurde, wurde ignoriert und stattdessen der Zeitpunkt des Eingangs auf dem Server verwendet - das geht konträr den Anforderungen, des Lastenhefts. Zwischen den Arbeitsgruppen war eine Abstimmung bezüglich der Rückgabewerte noch ausstehend, sodass neu kommuniziert werden musste, welche Eingabeparameter erforderlich und welche Rückgabewerte zu erwarten sind. Dazu kam, dass eine Ortsbestimmung nicht stattfinden konnte, wenn die mobilen Endgeräte keine Geodaten mitsendeten. Innerhalb des Teststreifengenerators kann nun eine Postleitzahl eingegeben werden, die innerhalb des QR-Codes gespeichert wird. Tritt nun der Fall ein, dass keine Geodaten von den Smartphones aus mitgesendet werden können, wird die mitgesendete Postleitzahl in Geodaten (Längen- und Breitengrad) umgewandelt und erst dann auf der Datenbank gespeichert. Durch die Anforderung, dass Teststreifenschemas auch Kontextinformationen wie beispielsweise die Angabe, welche Flüssigkeit appliziert werden muss, sind auch innerhalb des Server-Backends Anpassungen erforderlich gewesen. Diese Kontextinformationen, welche über die Weboberfläche erfasst werden können, werden über das Server-Backend an die mobilen Applikationen verteilt, sodass diese Informationen innerhalb der Applikationen angezeigt werden kann.

**Technologien** Es wurden keine neuen Technologien eingesetzt. Eine Auflistung der verwandten Technologien ist im Arbeitspaket „Auslieferung der XML Dateien für dynamische Teststreifen mit einem Go-Server“ 4.6.6.1 auf Seite 272 unter dem gleichnamigem Kapitel „Technologien“zu finden.

**Systembeschreibung** Abbildung 275 auf der nächsten Seite zeigt den schematischen Aufbau des Server-Backends. Die Klassen werden mittels Umrandung in einer durchgezogenen Linie dargestellt. Ersichtlich ist hier das Zusammenspiel der Klassen untereinander. Auf unterster Ebene sind die Funktionen (Umrandung in gestrichelter Linie), welche von der Klasse „SelftestService“ nach außen hin bereitgestellt wird, dargestellt. Diese Funktionen definieren die bidirektionale Schnittstelle zu den mobilen Applikationen.

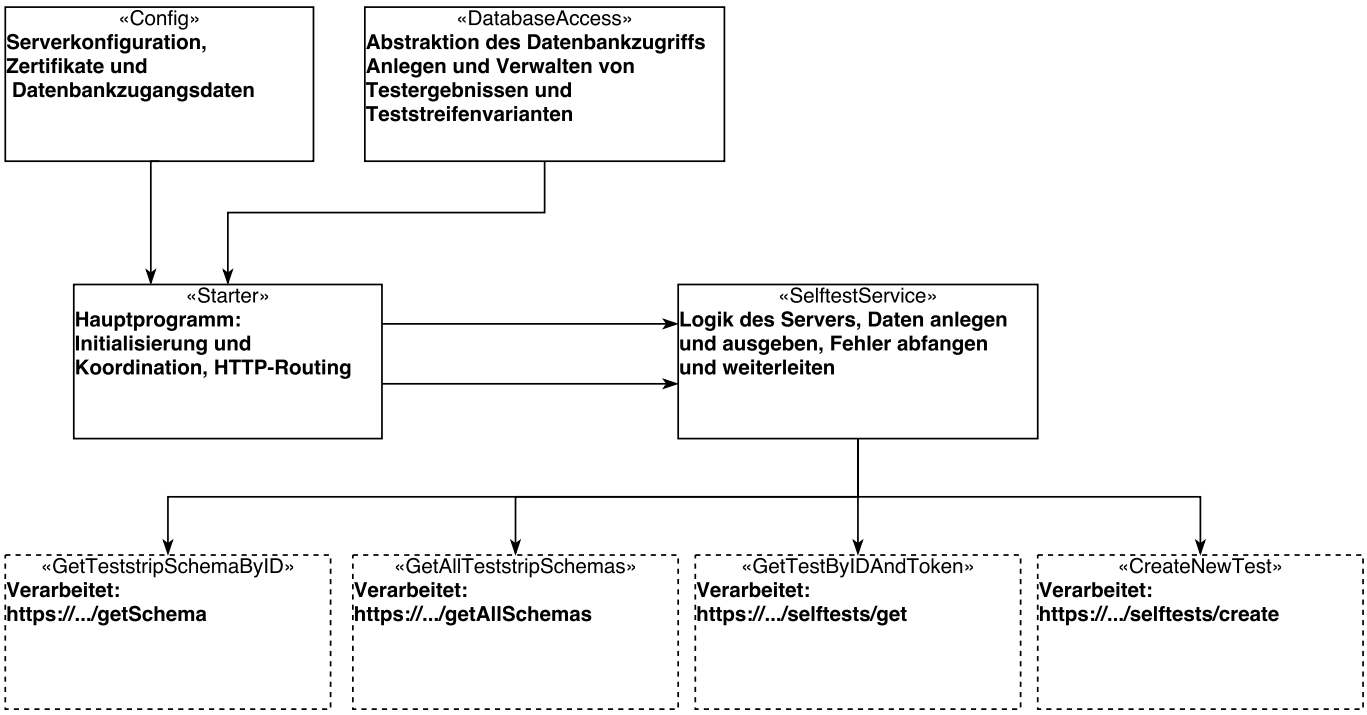



Abbildung 275: Schematische Darstellung des Server-Backends

**Evaluation** Es wurde ein Integrationstest durchgeführt, in dem die mobilen Applikationen mit der neuen REST-Schnittstelle kommunizierten. Die durchgeführten Tests verliefen erfolgreich und es konnten keine Fehler festgestellt werden.

Um die weitere Entwicklung am Backend zu erleichtern wurden außerdem Modultests (Unittests) geschrieben, die die Funktionsweisen der einzelnen Komponenten testen.

**Parameter** Durch die Anpassung des Server-Backends, mussten die Parameter die zur Nutzung der vom Go-Server-Backend bereitgestellten REST-API angepasst werden. Eine aktuelle Beschreibung der API befindet sich im Anhang in dem Kapitel 4.8.5.

**Fazit** Der Go-Server konnte innerhalb der angesetzten Dauer von sechs Wochen finalisiert werden. Eine Weiterentwicklung ist bei gleichbleibenden Anforderungen nicht erforderlich.

**Anhang**

Nachfolgend ist eine vollständige Beschreibung der REST-Schnittstelle zu finden. Sämtliche Methodenaufrufe, samt Ergebnis, sind in ihr beschrieben. Die nachfolgende API, ist ebenfalls im mitgelieferten Quellcode unter 'server-backend/README.md' zu finden.

**Testergebnis erzeugen** Ein Testergebnis kann von den mobilen Endgeräten an den Server geschickt werden. Nach dem REST-Standard ist hierfür ein POST-Aufruf nötig.

```
POST http[s]://SERVER:PORT/selftests/create?apikey=KEY
```

Der Aufruf wird unterteilt in Metadaten, welche die Testergebnisinformationen enthalten und Bilddaten, die als binäres Objekt in der Formvariable `teststripimg` übertragen werden.

Die Metadaten werden als JSON-Dokument in der Formvariable `metadata` übertragen und enthalten folgende Informationen, die sich aus Informationen aus dem QR-Code, dem Ergebnis des Bildverarbeitungsalgorithmus und Informationen über das mobile Endgerät zusammensetzen.

```
1 {
```

```
"testSchemaId": 33,
"computedResult": "-",
"location": {
        "lat": 53.13513513513514,
        "lon": 8.169124952332199
        },
"operatingSystem": "android",
"postalCode": "26129",
"tokenId": "uniqueidentifier"
}
```

Listing 19: Antwort des Servers, wenn Teststreifenschema abgefragt wurde

Auf den Aufruf antwortet der Server mit nachfolgendem Dokument. Das Feld `testRecordId` enthält die Identifikation, über die das Teststreifenergebnis auf dem Server gefunden werden kann.

```
{
  "tokenId": "uniqueidentifier",
  "testRecordId": "2071"
}
```

Listing 20: Antwort des Servers, wenn Testergebnis erstellt wurde

**Testergebnis abrufen** Um den aktuellen Status eines gesendeten Tests abzufragen, muss folgende Anfrage an den Server gestellt werden:

```
GET http[s]://SERVER:PORT/selftests/get?apikey=KEY&tokenid=TOKEN&id=IDENTIFIKATION[&includeimage=true]
```

```
{
  "testRecordId": IDENTIFIKATION,
  "testSchemaId": 33,
  "computedResult": "-",
  "diagnoseResult": "",
  "diagnoseText": "",
  "diagnoseReviewer": -1,
  "created": 1457466107,
  "location": {
    "lat": 53.13513513513514,
    "lon": 8.169124952332199
  },
  "image": null,
  "tokenId": "TOKEN",
  "diagnoseSent": false,
  "operatingSystem": "",
  "postalCode": ""
}
```

Listing 21: Antwort des Servers, wenn Testergebnis abgefragt wurde

Die Parameter `TOKEN` und `IDENTIFIKATION` ergeben sich aus der Antwort des Servers beim Erstellen eines Testergebnisses. Ist `includeimage=true` gesetzt, so wird das gesendete Bild als Base64-Zeichenkette im JSON-Dokument eingefügt.

GET http[s]://SERVER:PORT/getSchema?apikey=KEY&id=IDENTIFIKATION

Das empfangene JSON-Dokument enthält alle Informationen über das Teststreifenschema, wie Teststreifenbeschreibungen in allen verfügbaren Sprachen, die QR-Code Informationen und den Namen des Teststreifens:


```
{
  "id": IDENTIFIKATION,
  "name": "PG_Debug_33",
  "data": {
    "teststrip": {
      "applyField": {
        "posX": 1515,
        "posY": 131,
        "size": 20,
        "type": "Blut"
      },
      "color": {
        "b": 255,
        "g": 0,
        "r": 0
      },
      "colorchanges": {
        "colorchange": [
          {
            "angle": 0,
            "corner-size": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 0.25,
            "posX": 900,
            "posY": 193,
            "referenceField": true
          },
          {
            "angle": 0,
            "corner-size": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 0.75,
            "posX": 950,
            "posY": 175,
            "referenceField": false
          },
          {
            "angle": 0,
            "corner-size": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 1,
            "posX": 925,
            "posY": 150,
            "referenceField": false
          },
          {
```

```
            "angle": 0,
            "corner-size": 0,
            "gradient": "GradientCyclicLinear",
            "intensity": 0.75,
            "posX": 868,
            "posY": 161,
            "referenceField": false
          }
        ]
      },
      "extrainformation": {
        "height": 179,
        "posX": 0,
        "posY": 50,
        "type": "QR_CODE",
        "value": "{\"id\":\"33\",\"plz\":\"26129\"}",
        "width": 179
      },
      "markers": {
        "marker": [
          {
            "posX": 341,
            "posY": 209,
            "size": 56,
            "value": 1634
          },
          {
            "posX": 1347,
            "posY": 213,
            "size": 56,
            "value": 1636
          },
          {
            "posX": 1347,
            "posY": 56,
            "size": 56,
            "value": -18371
          },
          {
            "posX": 341,
            "posY": 28,
            "size": 56,
            "value": -11861
          }
        ]
      },
      "miscOptions": {
        "colorType": 1,
        "dpi": 300,
        "drawText": false
```

```
      }
    }
  },
  "created": 1457308800,
  "active": true,
  "timer": 0,
  "applyLiquid": 0,
  "loc": {
    "de": {
      "name": "kein name definiert",
      "tutorial": "kein tutorial gegeben",
      "tutorialAccess": "kein accessability tutorial gegeben"
    }
  }
}
```

Listing 22: Teststreifenschema

Falls kein Teststreifenschema unter der angegebenen `IDENTIFIKATION` existiert, antwortet der Server mit einer entsprechenden Fehlermeldung:

```
{
  "Status": "false",
  "Message": "Teststripschema does not exist in database"
}
```

**Alle Testschemata abrufen** Um alle Teststreifenschemata vom Server zu holen, wird folgender Aufruf gesendet.

```
GET http[s]://SERVER:PORT/getAllSchemas?apikey=KEY
```

Die Antwort des Servers ist ein JSON-Array aus Teststreifenschemadokumenten, wie sie auch beim Aufruf für Testschema über Identifikation zurückgegeben werden.

### 4.8.6 Fertigstellung des Webfrontend

- Priorität: Kritisch
- Motivation und Nutzen des Arbeitspaketes:
  Das Arbeitspaket soll dazu dienen, das Webfrontend entsprechend der Kundenanforderungen aus dem Lastenheft fertig zu stellen.
- Arbeitspaketbeschreibung:
  Am Ende dieses Arbeitspaketes ist das Webfrontend entsprechend aller Anforderungen laut Lastenheft fertig gestellt. Dazu müssen folgende Aufgaben abgearbeitet werden:
  - Wechsel von Heatmap weg führt zu Lags
  - Upload der JSON-Dateien anpassen
  - Css-Dateien für Webfrontend designen und einbauen
  - Integration der Seminararbeit von Raphael (Statistiker-Ansicht)
  - Integration der Seminararbeit von Daniel (Usability)
  - Ausführliche Tests bezüglich aller Funktionen laut Lastenheft (Diagnose schreiben, speichern, versenden, Login usw.)

- Vorbedingungen:
  -
- Nebenbedingungen:
  -
- Nachbedingungen:
  Alle offenen Anforderungen sind in das Webfrontend integriert und alle Fehler behoben.
- Aufwand:
  sechs Wochen
- Personen:
  - Kevin Sandermann
  - Nico Koch
  - Raphael Kappes
  - Daniel Wegmann

#### 4.8.6.1 Dokumentation

**Ablauf** Der erste Schritt dieses Arbeitspakets bestand darin, die Ergebnisse der Seminararbeiten von Daniel Wegmann und Raphael Kappes in das Webfrontend zu integrieren.

Dadurch wurde die Basis dieses Arbeitspakets, dessen Ziel die Verbesserung der Usability und die Integration der neuen Statistiker-Perspektive ist, gebildet. Anschließend wurden die folgenden, in kleinere Arbeitspakete unterteilte, Verbesserungen vorgenommen.

**Usability-Verbesserungen und Erstellen eines Corporate Designs**
Zu Beginn des Arbeitspakets wurde die Optik des Webfrontend durch die Standard-Optik der verwendeten PrimeFaces-Elemente bestimmt. Dadurch entstand ein eher trockener, klinischer Eindruck. Zudem war eine einwandfreie Abtrennung der Elemente nicht immer möglich, da als Farben nur Weiß und einige Grau-Abstufungen verwendet wurden. Um diesen Umstand zu verbessern wurde ein farbliches Corporate Design entwickelt, das auch auf die Apps angewendet wurde. Ziel dessen war zum Einen, die zuvor beschriebenen optischen Probleme zu beheben, und zum Anderen eine systemübergreifende einheitliche Darstellung der verschiedenen Produkte zu gewährleisten. Dazu wurde ein Farbschema entwickelt, welches in den Apps und dem Webfrontend verwendet wird. So kann ein systemweit einheitliches Look and Feel erreicht werden. Abbildung 276 zeigt das Farbschema.

Abbildung 276: Farbschema für Webfrontend und Apps

Auf Seiten des Webfrontends wurde dieses Design mittels „Cascading Style Sheets“ (CSS) implementiert. Abbildung 277 auf der nächsten Seite zeigt, wie sich das Aussehen der Webseite geändert

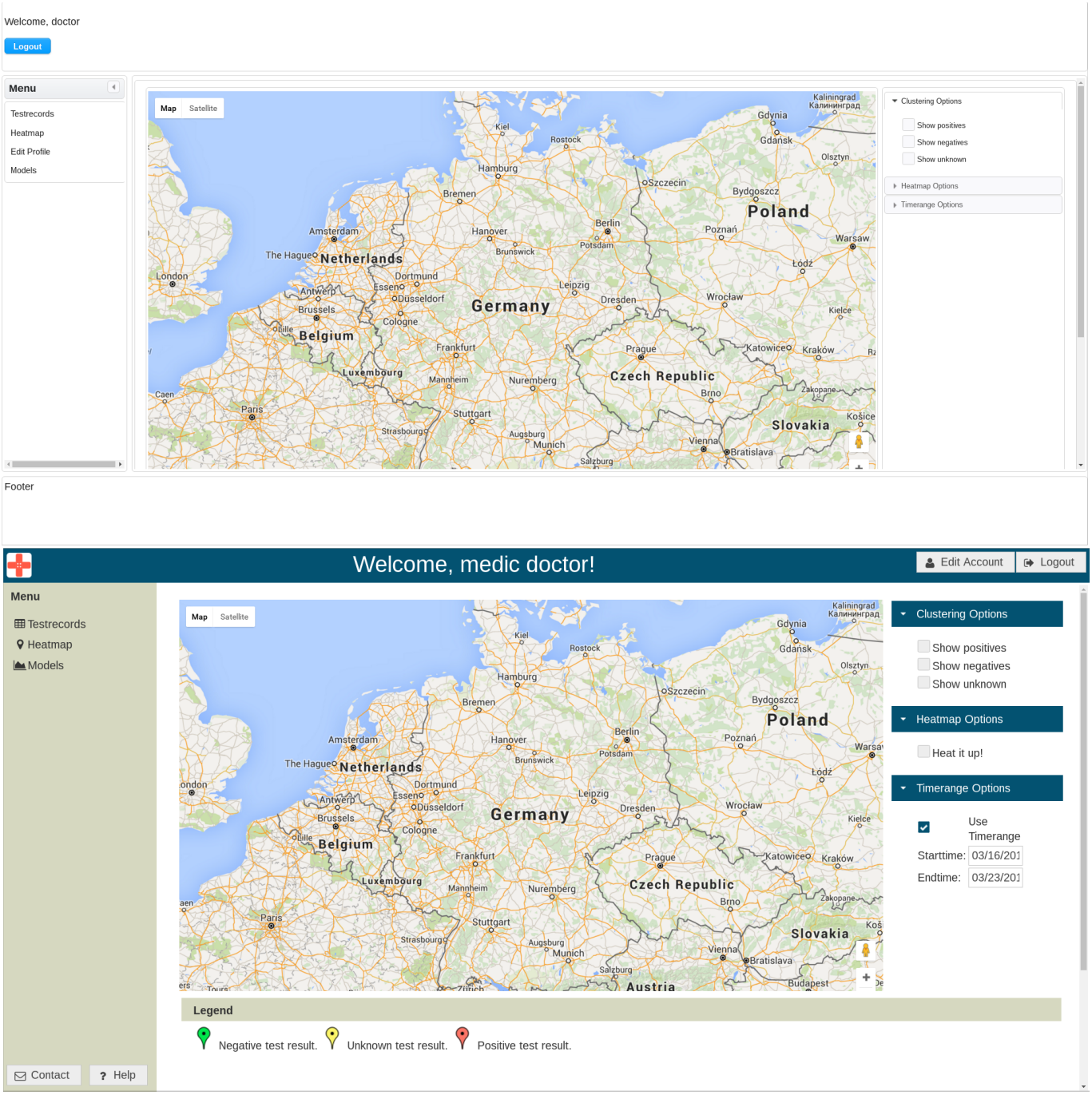



Abbildung 277: Veränderung im Aussehen der Webseite (Heatmap Ansicht)

hat. So wurde beispielsweise die Kopfleiste überarbeitet und das Logout-Menü nach oben rechts verschoben.

Zusätzlich zum Farbschema wurden weitere optische Verbesserungen vorgenommen. So wurden die verschiedenen Buttons um Symbole erweitert, die die Funktionen der jeweiligen Buttons passend visualisieren. Abbildung 278 auf der nächsten Seite zeigt exemplarisch hierfür das neue Navigationsmenü im Vergleich zum alten.

Die Ansicht der Testdetails wurde überarbeitet, sodass die Karte nun unter den Details angezeigt wird. Darüber hinaus wurden unnötige Freiflächen entfernt, indem dafür gesorgt wurde, dass die dargestellten Inhalte sich an die Auflösung des anzeigenden Monitors anpassen.

**Verantwortliche Personen:**

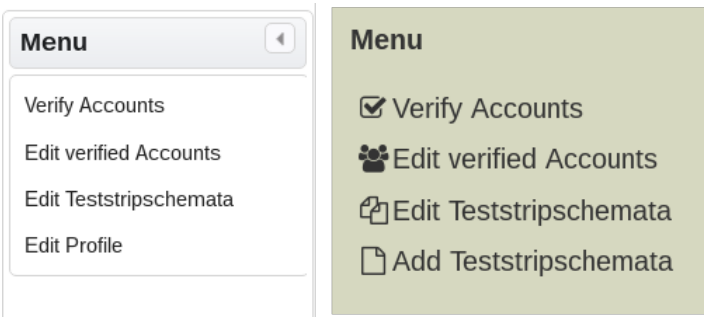



Abbildung 278: Altes Navigationsmenü im Vergleich zum neuen Navigationsmenü

- Kevin Sandermann
- Nicolas Koch

---

**Nutzerfeedback des Projektmeetings**

Im Rahmen des Projektgruppenmeetings vom 09.03.2016 wurde die zu der Zeit aktuelle Version der Weboberfläche der Gruppe vorgestellt. Diese Version enthielt bereits die an das neue Corporate Design angepasste Oberfläche. Im Rahmen dieser Präsentation sind durch die Gruppe viele kleinere Erweiterungs- und Verbesserungsvorschläge formuliert worden, welche allesamt umgesetzt wurden.

**Umgesetzte Verbesserungsvorschläge:**

- Die Filterfunktionsfelder sämtlicher Listen sind nicht länger ausgegraut, da sie jetzt funktionieren. (vgl. Abbildung 279 auf der nächsten Seite)
- Die Karte der Medizineransicht ist jetzt zusammen mit den Testdetails in ein Akkordeonmenü gegliedert, um die gesamte Größe des Browserfensters auszunutzen. Die Karte kann hierbei nach Belieben ein- und ausgeblendet werden. (vgl. Abbildung 279 auf der nächsten Seite)
- Das Bild des Teststreifens ist durch einen Klick darauf vergrößerbar.
- Die Schaltfläche „Edit Profile“ ist nun nicht mehr Teil des Navigationsmenüs am linken Bildschirmrand, sondern als Schaltfläche „Edit Account“ am rechten oberen Bildschirmrand neben der „Logout“-Funktion platziert. (vgl. Abbildung 279 auf der nächsten Seite)
- Das Navigationsmenü der Statistikeransicht ist nun nicht mehr auszublenden.
- In den Tabellen der Administrationsoberfläche sind die „Delete“-Schaltflächen nicht weiter unter einer eigenen Spalte „Actions“ platziert, sondern Teil des Dialogs zur Bearbeitung einer Spalte.
- Sämtliche Ein- und Ausblendeanimationen der Weboberfläche sind nun konsistent.
- Die Möglichkeit, dass ein Nutzer die Größe eines Dialogs ändern kann, ist jetzt konsistent nur bei sinnvollen Dialogen freigeschaltet.
- Die Position der Dialoge ist jetzt konsistent in der Mitte des Bildschirms umgesetzt.
- Die Platzhalter für grafische Inhalte von R-Skripten sind eingefügt worden (vgl. Abb. 280 auf der nächsten Seite).

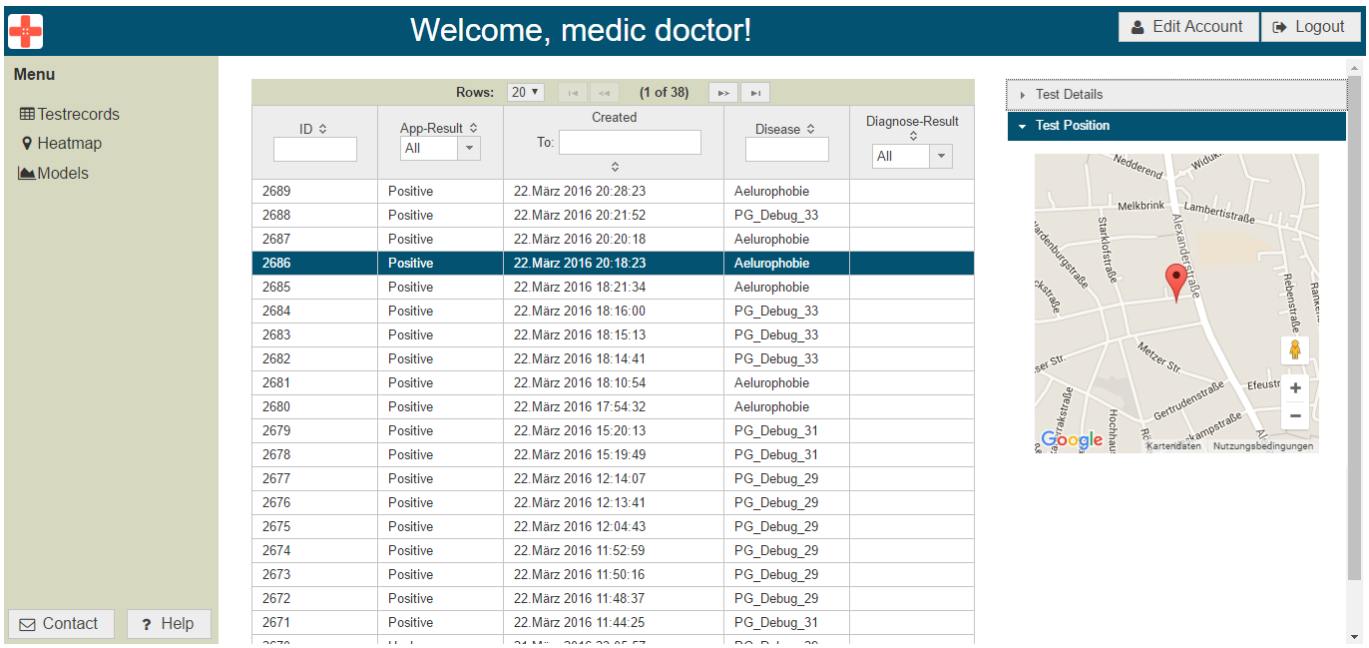



Abbildung 279: Verbesserte Standardansicht für Mediziner

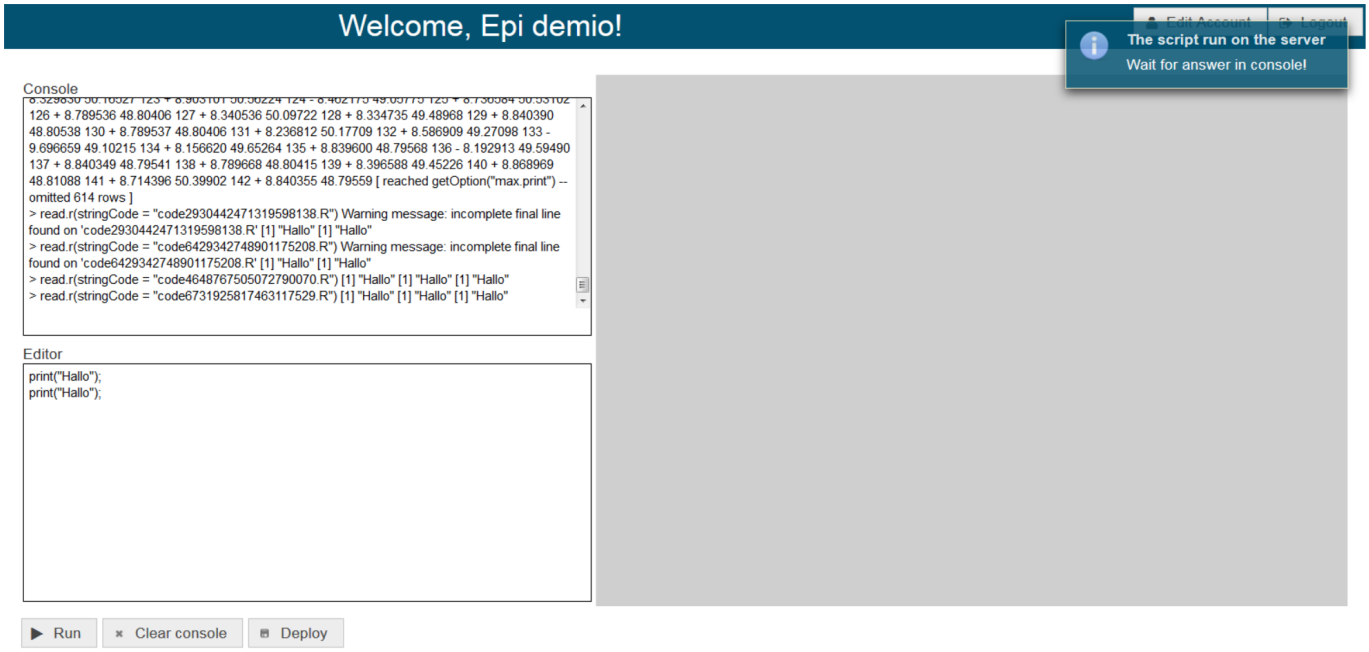



Abbildung 280: Startseite eines Statistikers mit Platzhalter für grafische Inhalte.

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch
- Raphael Kappes

---

**Erweiterung der Weboberfläche für Statistiker und Anpassung der Modell-Ansicht für Mediziner**

Im Rahmen des letzten Sprints wurde die Weboberfläche für Statistiker erweitert. Dies schloss

die Überarbeitung der Modell-Ansicht für Mediziner ein, sodass diese nur noch Modelle in der Liste auswählen können, die grafische Inhalte haben, sodass nur noch der Name des Modells, der Quellcode und eine kurze Beschreibung des Modells zu sehen sind. Weiterhin wird der grafische Inhalt eines Modells in einem separaten Dialog angezeigt, sodass die gesamte Seite mit den Modellen ausgefüllt werden kann (vgl. Abb. 281).

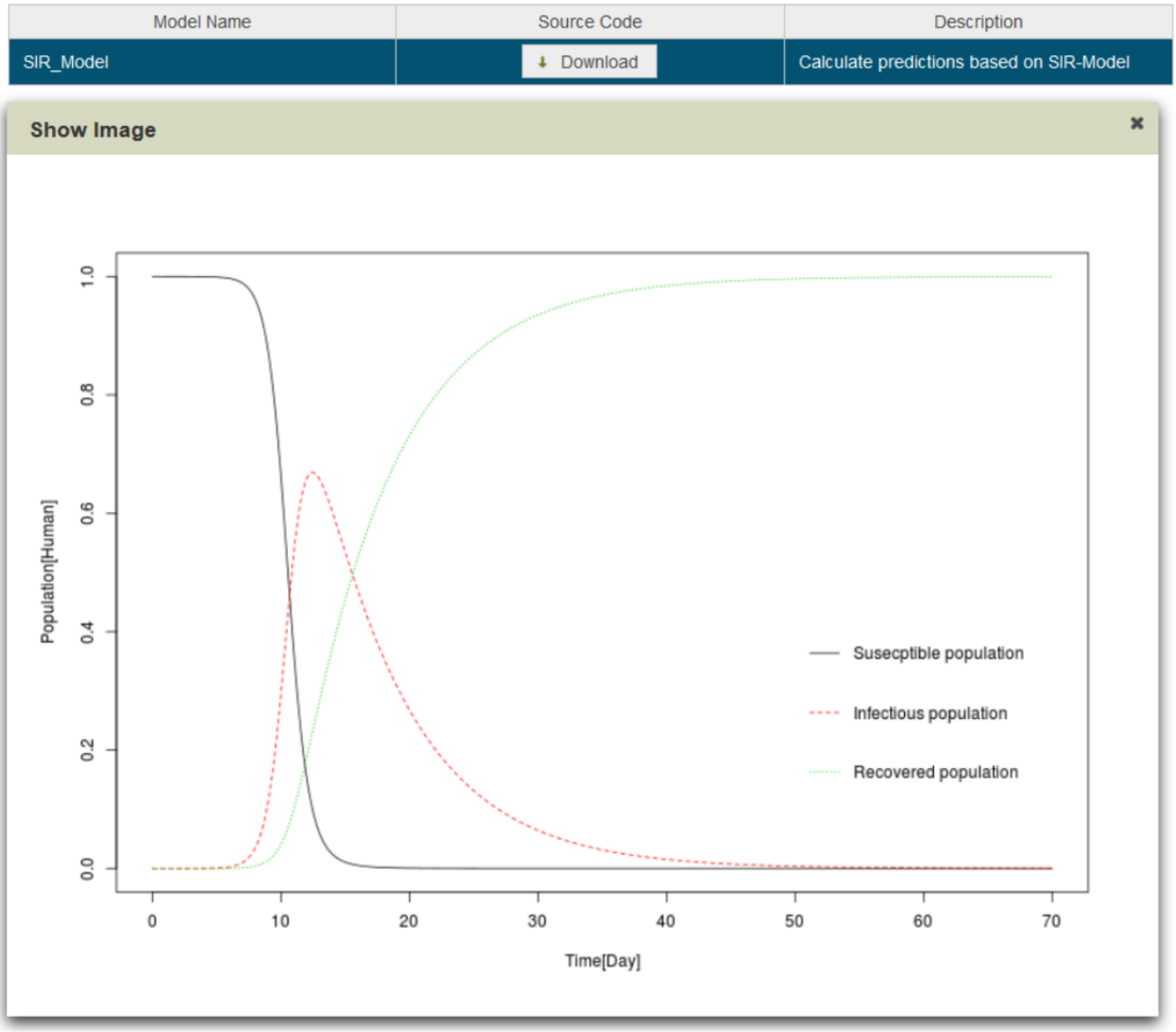



Abbildung 281: Modell-Übersicht eines Mediziners mit ausgeführten Modell

Zwischen Statistikern und Medizinern wird in der Modell-Übersicht in soweit unterschieden, dass ein Mediziner nur Modelle einsehen kann und Statistiker R-Funktionen. Ein Statistiker hat die Übersicht über alle R-Funktionen, die bisher in der Weboberfläche umgesetzt wurden. Dieser kann den Funktionsnamen einsehen, den Quellcode herunterladen und die detaillierte Dokumentation über Benutzung, Aufrufparameter und Beispielen jeder R-Funktion einsehen (Siehe Abb. 282 auf der nächsten Seite). Weiterhin kann der Statistiker die R-Funktionen nicht durch das Auswählen ausführen, sondern muss die R-Funktionen im Entwicklungsbereich ausführen. Da der Entwicklungsbereich eines Statistikers solange erhalten bleibt wie dieser eingeloggt ist, kann zwischen der Übersicht der R-Funktionen und der Entwicklungsumgebung problemlos gewechselt werden.

**Verantwortliche Personen:**

- Raphael Kappes

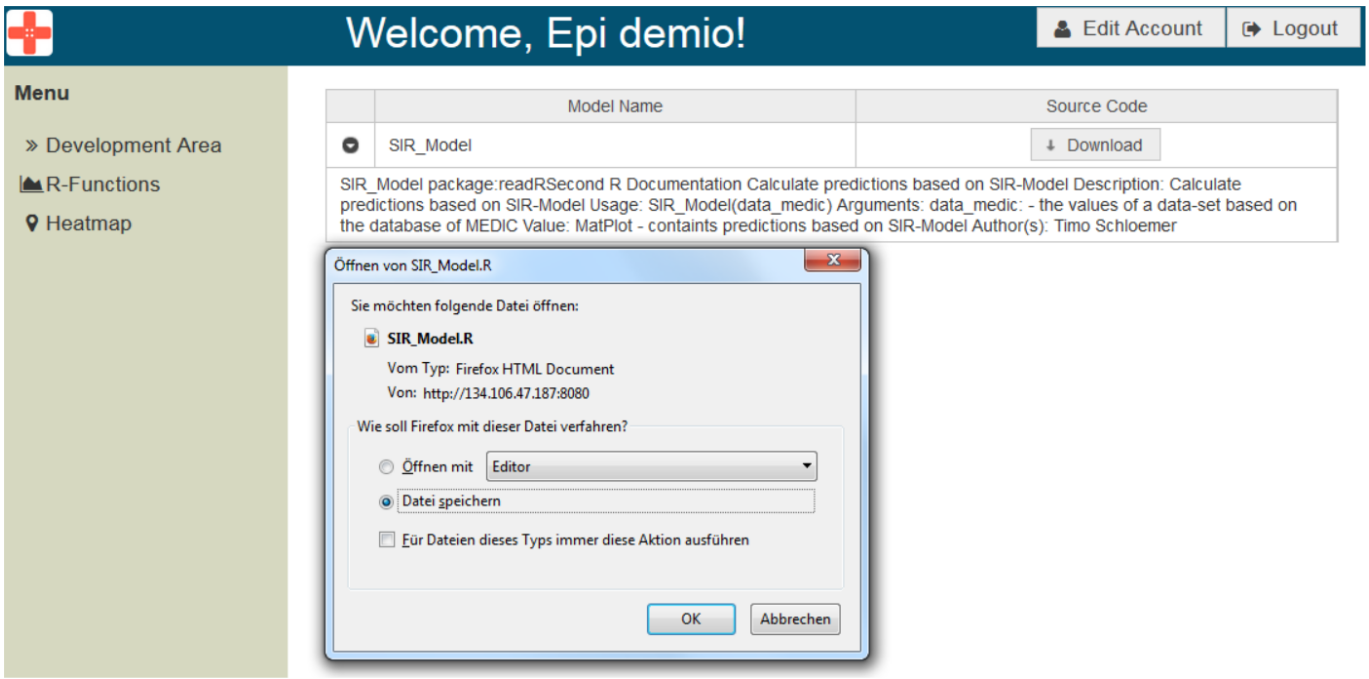



Abbildung 282: Die Übersicht eines Statistikers aller R-Funktionen und die zum Download stehende Datei

---

**Filter- & Sortierfunktion sämtlicher Tabellen**

Im Laufe der Bearbeitung wurde festgestellt, dass das Filtern und Sortieren der verschiedenen Listen (eingegangene Accounts, verifizierte und unverifizierte Accounts, Teststreifenschemata) teilweise nicht funktionierte und teilweise überhaupt nicht implementiert war. Hier wurden die vorhandenen Probleme behoben und das Filtern und Sortieren für die Listen implementiert. Dadurch ist es möglich, in allen Listen nach den Inhalten der verschiedenen Spalten sowohl zu filtern als auch zu sortieren. In Abbildung 283 ist die Liste der Teststreifenschemata zu sehen, bei der nach dem Namen des Schemas sortiert wurde und nur aktive Schemata angezeigt werden (Filterung).

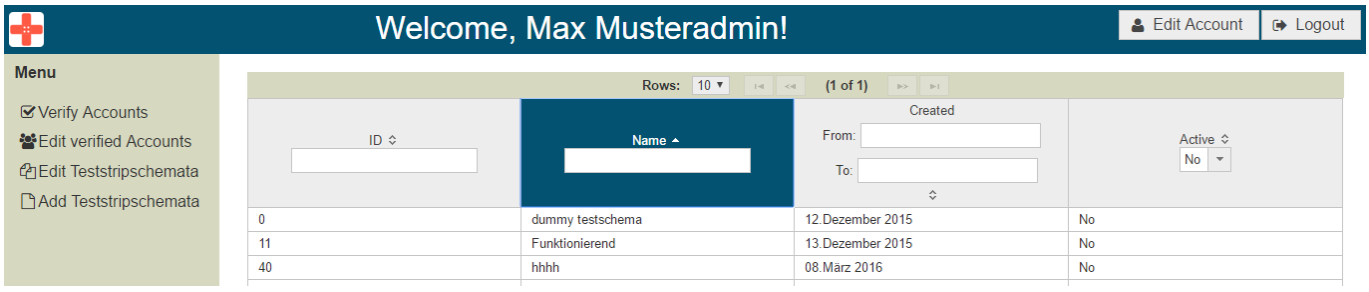



Abbildung 283: Liste der Teststreifenschemata

Die Liste eingegangener Tests wurde außerdem so überarbeitet, dass nicht mehr lediglich angezeigt wird, ob eine ärztliche Diagnose vorhanden ist. Stattdessen zeigt die Spalte jetzt direkt das Ergebnis der Diagnose, sofern vorhanden. Damit wird ein direkter Vergleich des Ergebnisses der Voranalyse und der ärztlichen Diagnose möglich.
Eine Vielzahl von kleineren Fehlern konnten darüber hinaus während der Bearbeitung des Arbeitspakets identifiziert und behoben werden. Dazu gehörten z.B., dass unter bestimmten Umständen der Inhalt der verschiedenen Listen nicht angezeigt werden konnte.

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch
- Daniel Wegmann

---

**Verarbeitung der neuen Teststreifen**

Im Rahmen der Finalisierung der Struktur von Teststreifendesigns sind einige Änderungen an den Teststreifenschemata getätigt worden. Diese Änderungen machten es nötig, eine modifizierte Version des Teststreifengenerators in das Webfrontend einzubetten. Des Weiteren ist es jetzt nötig, dass das Webfrontend den Inhalt der QR-Codes einer Teststreifenvariante in dessen JSON-Daten schreibt. Letzteres ist darin begründet, dass erst das Webfrontend die Teststreifenvariante in der Datenbank persistiert, sodass erst beim Anlegen der Teststreifenvariante im Webfrontend die Schema-ID in den QR-Code geschrieben werden kann. Dieser Mechanismus ist jetzt vollständig implementiert und ausführlich getestet. Außerdem sind jedem Teststreifenschema jetzt weiterführende Eigenschaften zugeordnet, die durch Usability-Verbesserungen der mobilen Applikationen entstanden sind: Neben speziellen Anleitungen für sehbehinderte Nutzer sind weiterführende Felder wie z.B. eine Wartezeit nach Testdurchführung eingeführt worden. Im Webfrontend wurden diese Felder alle in den Ansichten „Edit Teststripschemata“ und „Add Teststripschemata“ umgesetzt.

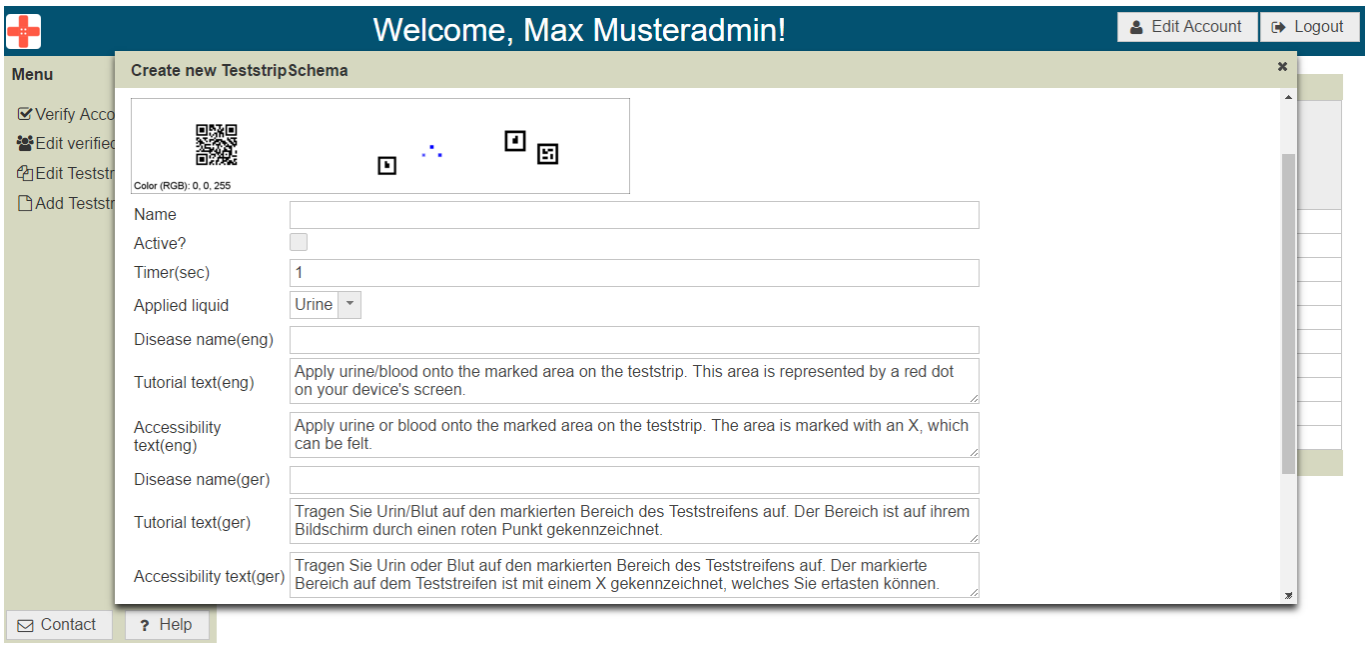



Abbildung 284: Oberfläche für neue Teststreifenschemata

Wie Abbildung 284 zeigt, enthalten diese neu eingeführten Eingabefelder direkt Standardtexte, welche ein Nutzer bei Bedarf anpassen kann.

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch

---

**Anpassung des Datenbankschemas an die neuen Teststreifenschemata**

Wie bereits unter „Verarbeitung der neuen Teststreifen“ beschrieben, wurden die Teststreifenschemata um einige Felder erweitert. Diese Felder sind die zu applizierende Flüssigkeit, die Wartezeit nach Testdurchführung sowie beliebig viele Varianten der Felder Anleitungstext, Anleitungstext für sehbehinderte Nutzer und Krankheitsnamen. Letztere drei Felder können in verschiedenen Sprachen vorhanden sein, sodass die Datenhaltung eine entsprechende Speicherstruktur zu Verfügung stellen muss.

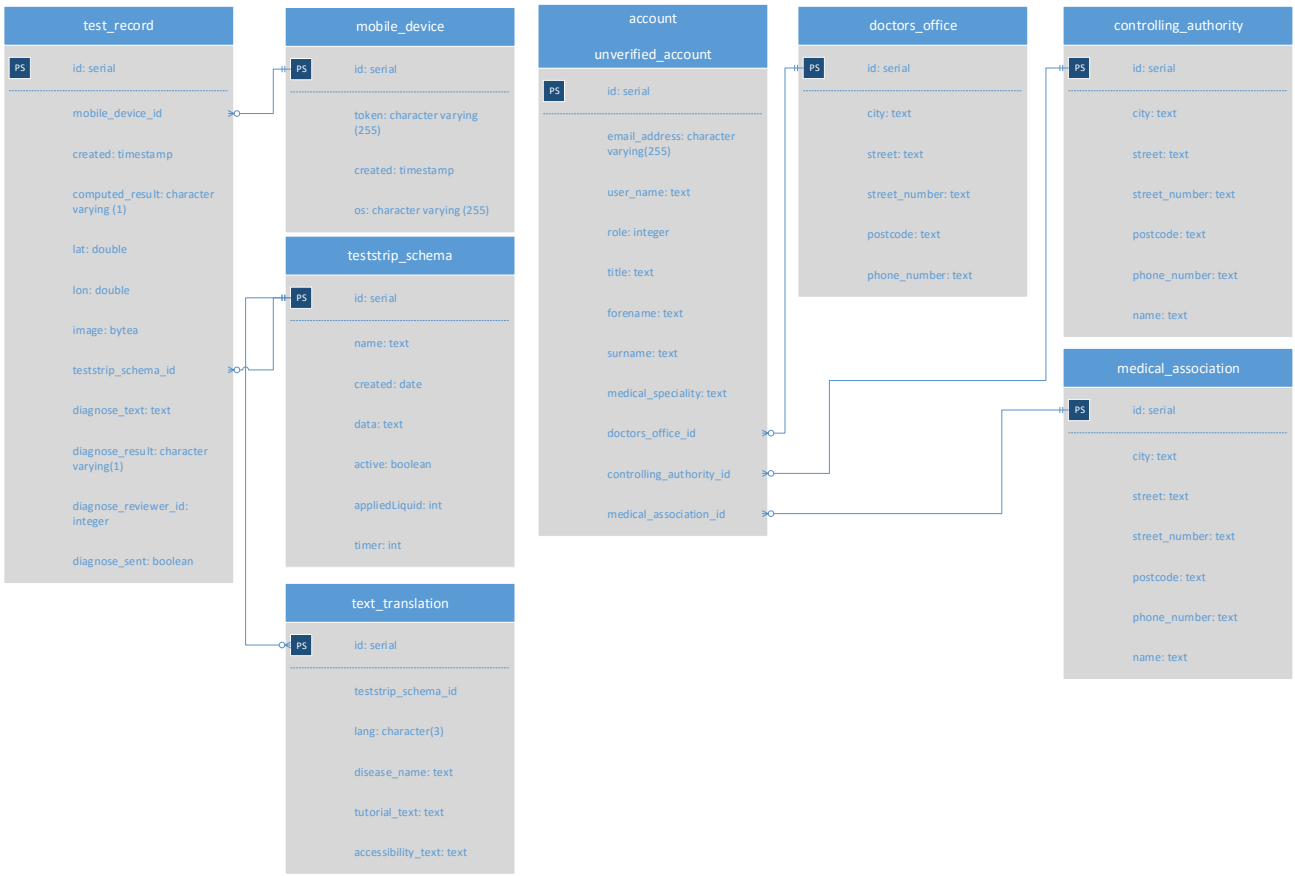



Abbildung 285: An neue Teststreifenschemata angepasstes Datenbankschema

Abbildung 285 zeigt das angepasste Datenbankschema, welches die neuen Anforderungen umsetzt. Das Datenbankschema wurde durch die Tabelle „text_translation“ ergänzt. Ein Element dieser Tabelle entspricht einer Übersetzung sämtlicher sprachenabhängiger Texte eines Teststreifenschemas. Ist ein Teststreifenschema beispielsweise für deutsch- und englischsprachige Applikationen verfügbar, gibt es für diese Variante zwei Einträge in der Tabelle „text_translation“, die jeweils eine Übersetzung in Deutsch und Englisch enthalten. Neben dieser neuen Tabelle wurde die Teststreifenschema-Tabelle um die Spalten „appliedLiquid“ und „timer“ ergänzt, da diese als Integer-Werte abgebildet sind und somit unabhängig von einer Übersetzung gespeichert werden können. Eine Umsetzung in die entsprechende Sprache wird für die Applizierungsflüssigkeit im Webfrontend vorgenommen, da es hier eine sehr begrenzte Anzahl an möglichen Werten gibt (aktuell zwei).

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch

**Einbindung des Teststreifengenerators in die Weboberfläche**

Damit potentielle Kunden auch selbst Teststreifen (bzw. Teststreifenvarianten) erstellen können, wurde schon im Laufe des Projekts entschieden, den Teststreifengenerator mit an den Kunden auszuliefern. Da dieser vorerst nur als Entwicklungshilfe entworfen wurde, musste er mehrmals überarbeitet und neu gestaltet werden. Dieser Teil des Arbeitspakets befasst sich mit der Einbindung des neuen Teststreifengenerators in die Weboberfläche. Dazu wurde der Teststreifengenerator als Abhängigkeit des Webfrontends in das Projekt eingebunden.
Des Weiteren wurde eine Strategie zur Auslieferung des Teststreifengenerators erarbeitet. Diese findet nun – samt Benutzerhandbuch – komplett über die Weboberfläche statt. So können Administratoren ohne viel Aufwand das fertige Produkt herunterladen und sofort nutzen. Abbildung 405 zeigt die Benutzeroberfläche der Seite „Add Teststripschemata“, welche den Teststreifengenerator zur Verfügung stellt.

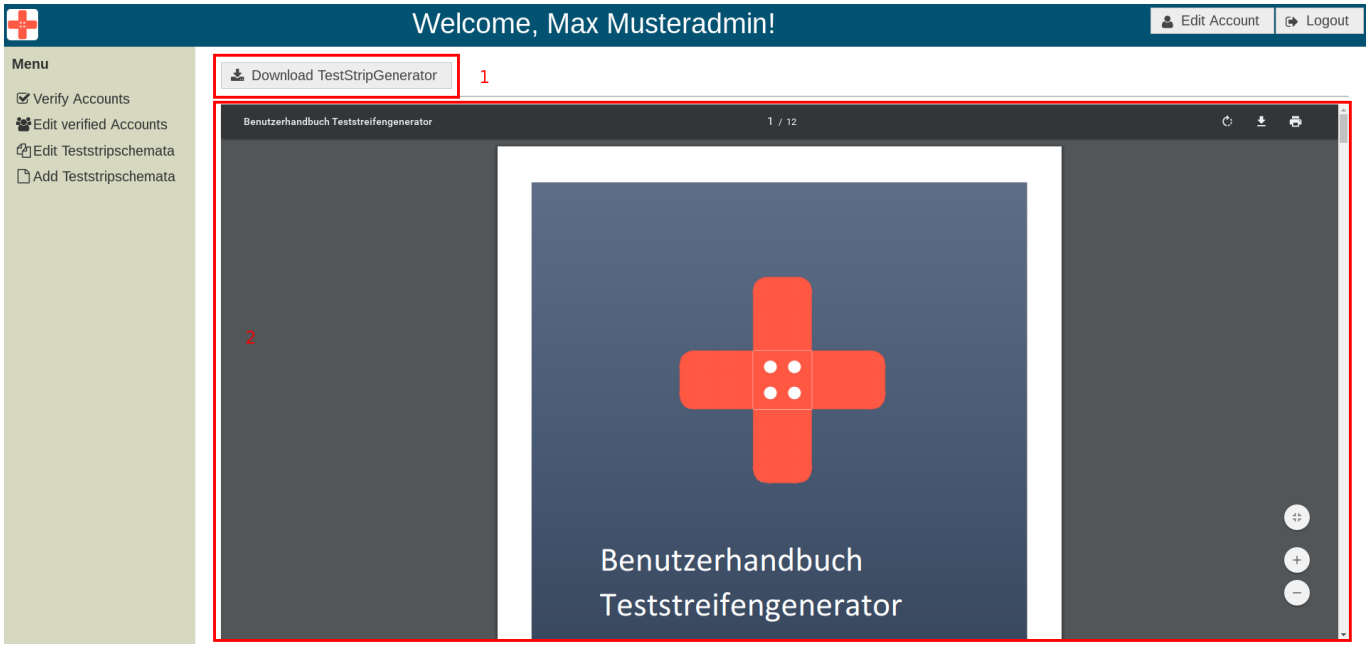



Abbildung 286: Bereitstellung des Teststreifengenerators auf der Webseite

Es befinden sich folgende Elemente auf dieser Seite:

1. Download-Button: Diese Schaltfläche startet den Download des Teststreifengenerators. Die Datei ist ein ausführbares Java-Archiv, welches zur Ausführung das Java Runtime Environment 8 benötigt.

2. Benutzerhandbuch: PDF-Vorschau des Benutzerhandbuchs des Teststreifengenerators. Diese enthält detaillierte Informationen über die Verwendung des Teststreifengenerators.

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch

---

**Erstellung neuer Accounts mit Rollen und Änderung weitergehender Accountdaten**

Bisher war es nicht möglich über die Weboberfläche Accounts für Statistiker und Administratoren anzulegen. Dieses wurde in der Oberfläche für Administratoren umgesetzt, sodass beim Anlegen

neuer Accounts nun die Wahl zwischen „Doctor“, „Statistician“ und „Administrator“ besteht. Der Dialog zum Anlegen neuer Accounts ändert sich dazu dynamisch und zeigt passend die Felder der jeweiligen Rolle. Auf der Login-Seite können wie bisher nur Mediziner Anfragen zum Anlegen eines neuen Accounts stellen. Das Hinzufügen von Statistikern und Administratoren soll einem Administrator vorbehalten sein, da diese im selben Institut tätig sind. Weiterhin sind in der Oberfläche der Administratoren die Accounts, in den Tabellen für unbestätigte und bestätigte Accounts, auswählbar. Durch das Auswählen wird ein Dialog geöffnet, der es einem Administrator ermöglicht den Account auf Korrektheit zu prüfen, in dem die rollenspezifischen Felder betrachtet werden und gegebenenfalls Änderungen eingefügt werden können. Abhängig davon welche angezeigt Tabelle wird, sind die Schaltflächen bei der Tabelle mit unbestätigten Accounts „Confirm Request“ und „Decline Request“ und bei der Tabelle mit bestätigten Accounts die „Save“ und „Delete“sichtbar. In Abb. 287 wird diese für bestätigte Accounts gezeigt. Ändert ein Administrator den Inhalt eines Feldes in dem

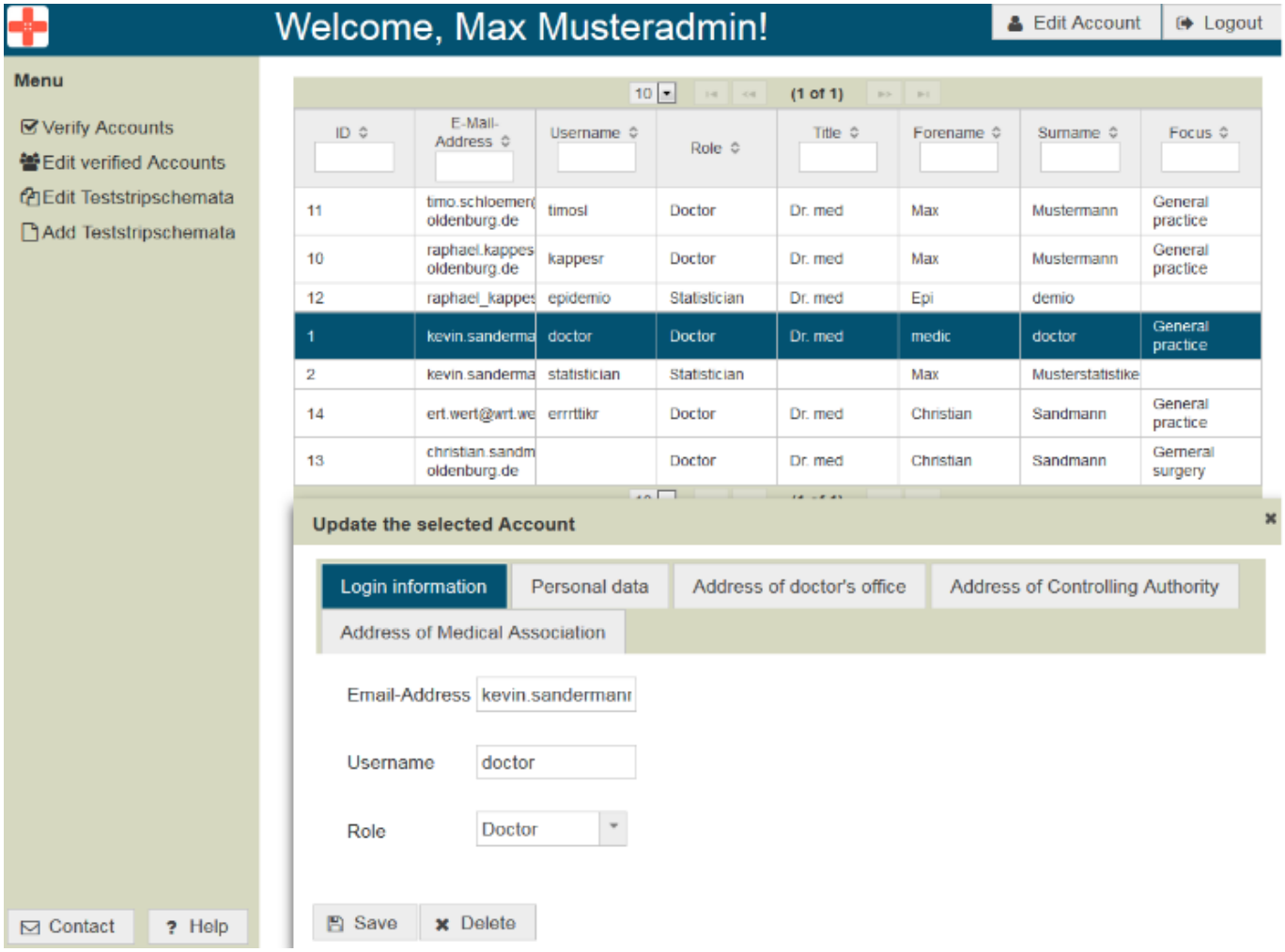



Abbildung 287: Dialog zur Bearbeitung eines bestätigten Accounts mit der Rolle „Doctor“

Dialog, so wird vom Webfrontend überprüft, wie stark sich der alte Wert vom neuen unterscheidet. Dies muss berücksichtigt werden, da zwei Mediziner in der selben Praxis arbeiten können und einer der beiden die Praxis wechseln kann. Der bisherige Eintrag muss in der Datenbank bestehen bleiben, da ein Mediziner in der Praxis verbleibt. Wird die Praxis von keinem Mediziner mehr benutzt, verbleibt diese in der Datenbank, da die Adresse durch andere Mediziner wieder benutzt werden könnte z.B. durch Nachfolger. Um dies zu bestimmen wird bei einer Änderung eines Feldes überprüft, ob sich der neue Wert mehr als 20% von dem vorherigen Wert unterscheidet. Wenn es mehr als 20% sind wird ein neuer Eintrag in die Datenbank geschrieben, wenn es weniger sind wird der alte Eintrag bearbeitet. Der Unterschied wird mit den Teilwörtern des alten und des neuen Wertes bestimmt in

dem geprüft wird, ob alle Teilwörter des neuen Wertes im alten Wert enthalten sind. Dies soll im folgenden mit einem einfachen Beispiel mit den Wörtern „AC“ als neuer Wert und „AB“ als alter Wert erläutert werden:

- Zunächst werden die Teilwörter des alten Wertes bestimmt.
  Diese sind „A“, „AB“ und „B“.
- Dann werden die Teilwörter des neuen Wertes bestimmt.
  Diese sind „A“, „AC“ und „C“.
- Dann wird das Vorkommen der Teilwörter beider Werte im alten Wert ermittelt. Die Teilwörter des alten Wertes kommen drei Mal vor, die Teilwörter des neuen Wertes kommen ein Mal vor.
- Daraus kann die relative Häufigkeit bestimmt werden in dem das Vorkommen der Teilwörter des alten und des neuen Wertes mit einander dividiert werden.
- Für dieses Beispiel ergibt sich eine Ähnlichkeit von $\frac{1}{3}$ bzw. ein Unterschied von $\frac{2}{3}$ ca. $66,66\%$.

Das gleiche Vorgehen wird bei einer Änderung der Ärztekammer - Medical Association - oder der Aufsichtsbehörde - Controlling Authority - durchgeführt. Zusätzlich zu den Änderungsmöglichkeiten des Administrators kann ein Mediziner seine Praxis wechseln (Siehe Abb. 288), bei der die selbe Überprüfung des Servers getätigt wird.

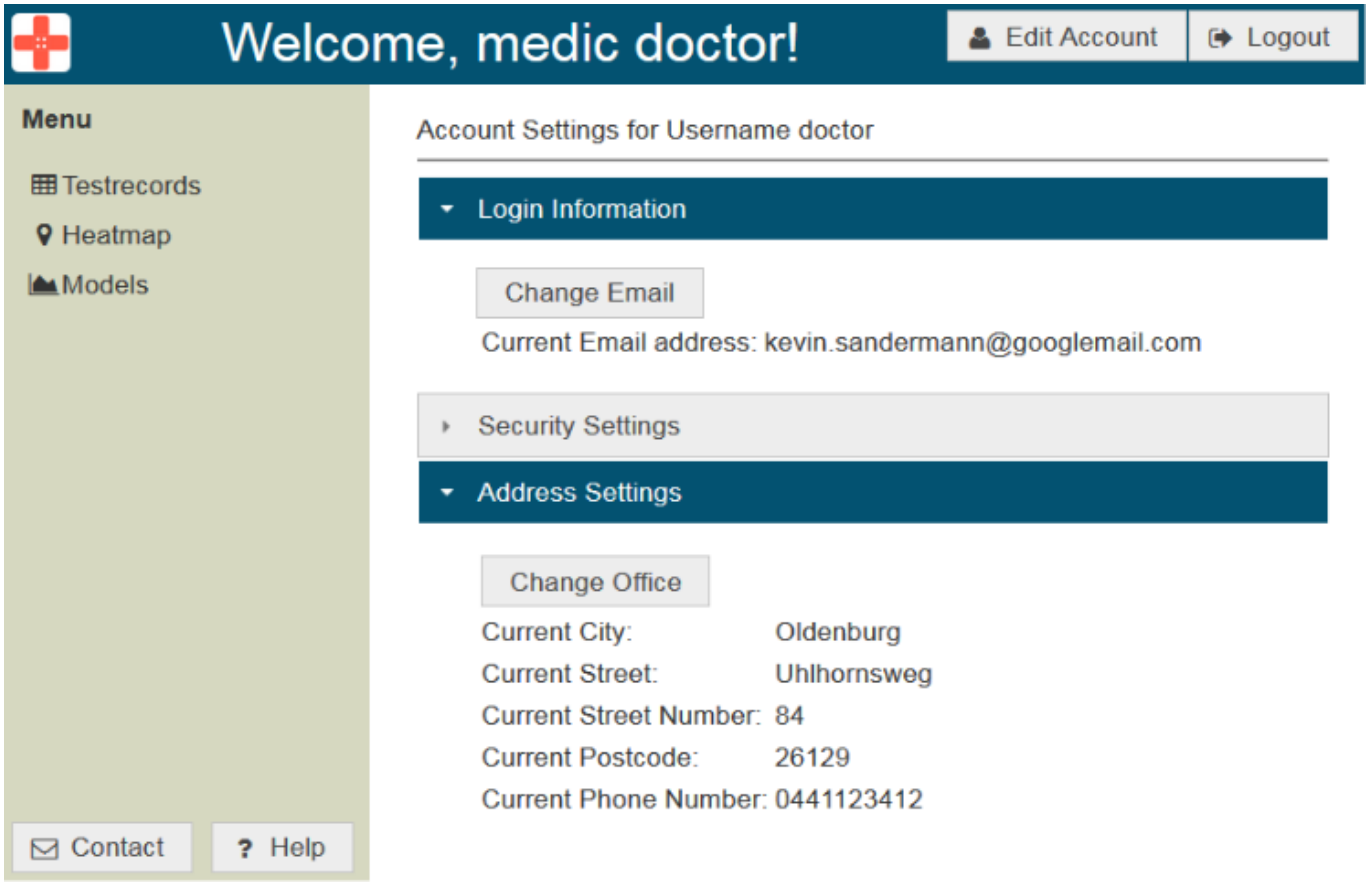



Abbildung 288: Oberfläche des Mediziners zur Bearbeitung seines Accounts

**Verantwortliche Personen:**

- Raphael Kappes

**Handbücher für das Webfrontend erstellen**

Damit Nutzer der Weboberfläche einen Überblick über alle Funktionen erhalten, wurden Handbücher erstellt. Um Informationen rollenspezifisch trennen zu können, hat sich die Gruppe dazu entschieden, drei verschiedene Handbücher zu erstellen. Folgende Hanbücher wurden erstellt:

- Handbuch Doktor (Kevin Sandermann)
- Handbuch Statistiker (Raphael Kappes)
- Handbuch Administrator (Nicolas Koch)

Diese besitzen einen Umfang von circa 10-12 Seiten und enthalten detaillierte Informationen über die Benutzeroberfläche der Webseite.

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch
- Raphael Kappes

---

**Hilfeseite für Nutzer erstellen**

Um dem Nutzer der Weboberfläche bei Bedarf Hilfestellung zu leisten, ist es von Nöten eine zentrale Hilfeseite einzurichten, die jederzeit durch den Nutzer erreichbar ist. Diese Hilfeseite sollte Informationen zur korrekten Nutzung der Weboberfläche bereitstellen. Hierfür wurden dem Navigationsmenü am linken Seitenrand die zwei Schaltflächen „Contact“ und „Help“ hinzugefügt. Diese Schaltflächen sind, unabhängig von der aktuellen Ansicht, jederzeit für den Benutzer sichtbar. Während durch einen Klick auf „Contact“ ein leeres E-Mailformular mit Zieladresse des Entwicklerteams geöffnet wird, wird durch Betätigen der „Help“-Schaltfläche folgenden in Abbildung 289 dargestellt Seite anzeigt.

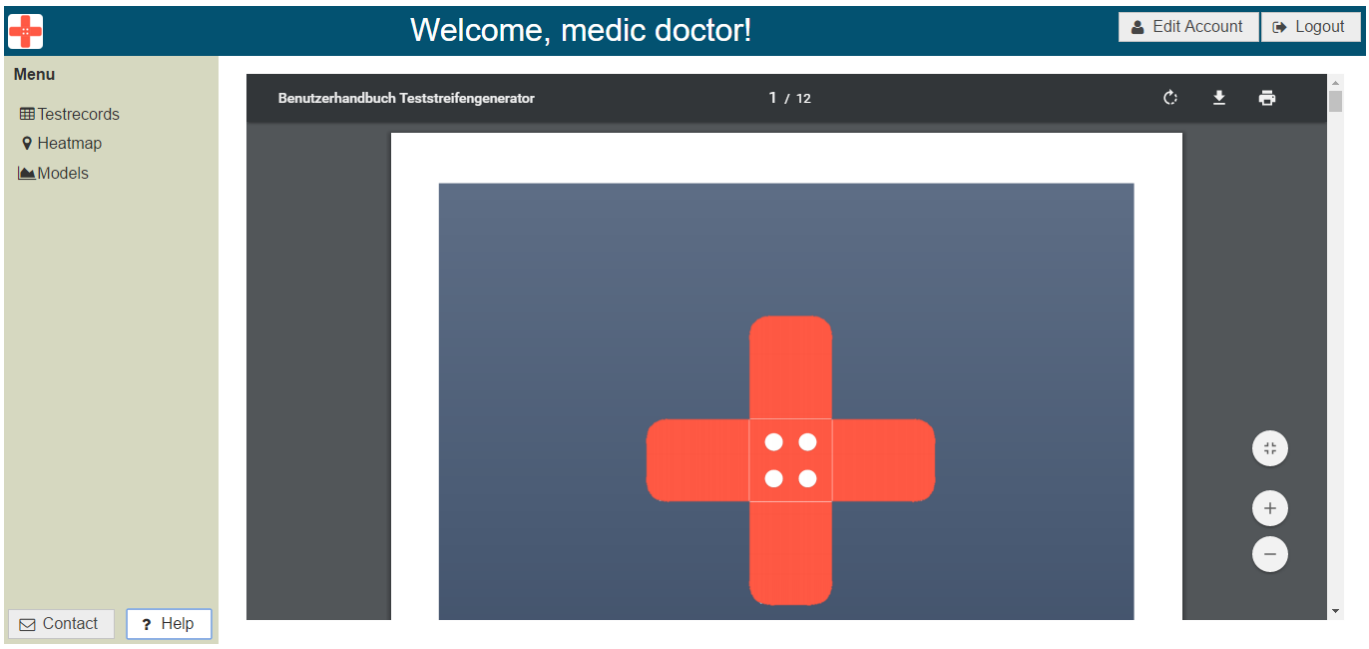



Abbildung 289: Dynamische Hilfseite für alle Accountrollen

Wie Abbildung 289 zeigt, wird dem Nutzer auf dieser Hilfeseite das Benutzerhandbuch für seine Rolle angezeigt. Einem eingeloggten Mediziner wird so beispielsweise das Medizinerhandbuch angezeigt. Neben dem Lesen des Handbuchs ist ebenfalls das Herunterladen als `.pdf`-Datei möglich.

**Verantwortliche Personen:**

- Kevin Sandermann

---

**Testdatengenerator**

Um die Funktionen der Weboberfläche, etwa die Tabelle mit den Testergebnissen oder die Heatmap, zu testen, musste der Testdatengenerator überarbeitet werden, da die Schnittstellen, auf die nach dem vierten Sprint zugegriffen wurden, im neunten Sprint nicht mehr benutzbar waren. Der Testdatengenerator wurde an die neuen Schnittstellen und das neue Datenbankschema angepasst, sodass neue Testergebnisse erzeugt werden können. Die Testergebnisse vom Testdatengenerator erhalten dabei wie echte Testergebnisse eine Smartphone-Identifikation, sodass jedes Testergebnis einem anderen fiktivem Smartphone zugewiesen werden kann. Damit der Go-Server diese fiktiven von den echten Smartphones unterscheiden kann, wird dem Server mitgeteilt, dass das Testergebnis von einem Smartphone mit dem Betriebssystem „DEBUG“ kommt anstatt von einem mit dem Betriebssystemen „Android“ und „iOS“. Dies verhindert, dass das Backend den fiktiven Smartphones Benachrichtigungen über neue Teststreifenvarianten oder Diagnosen sendet. Weiterhin wird durch die eindeutige Zuordnung eines fiktiven Smartphones zu einem Testergebnis die Qualität der Testdaten erhöht, da vorher alle Testergebnisse einem fiktives Smartphone in der Datenbank zugeordnet waren.

**Verantwortliche Personen:**

- Raphael Kappes

---

**Filterfunktion nach Krankheitsarten zur Heatmap hinzufügen**

Die funktionale Anforderung FA-20 des Lastenhefts formuliert eine weitere Filterfunktion der Heatmap: Durch Auswahl einer entsprechenden Krankheit sollen nur Ergebnisse von Tests dieser Krankheitsart angezeigt werden. Diese Funktion war bisher noch nicht umgesetzt und musste daher komplett implementiert werden. Hierzu wurde der Heatmap-Ansicht zunächst unter den Filterfunktionen ein weiteres Menü „Disease Options“ hinzugefügt. Wie die Abbildung 290 auf der nächsten Seite zeigt, ist es dem Mediziner unter diesem Menüpunkt möglich, eine Krankheit aus einer Liste aller Krankheiten auszuwählen.

Sobald eine Krankheit ausgewählt ist, werden die auf der Karte eingezeichneten Testergebnisse der Cluster- als auch der Heatmap-Ansicht aktualisiert. Um den Filter zurückzusetzen, wird dem Nutzer immer an oberste Stelle der Liste die Option „Show All“ zur Verfügung gestellt. Als technische Besonderheit ist dabei zu erwähnen, dass die Krankheitsarten an dieser Stelle bewusst nach Namen, nicht nach Identifikationsnummern, unterschieden werden. Dies hat den Sinn, dass Aktualisierungen von Teststreifenschemata trotzdem der gleichen Krankheit zugeordnet werden: Wird beispielsweise ein neues Teststreifenschema der Krankheit „Grippe“ erstellt, bekommt dieses einen neue Identifikationsnummer. Da allerdings auch der neue „Grippe“-Test die gleiche Krankheit wie der alte untersucht, werden die Teststreifenschemata an dieser Stelle nach Krankheitsnamen zugeordnet.

**Verantwortliche Personen:**

- Kevin Sandermann

---

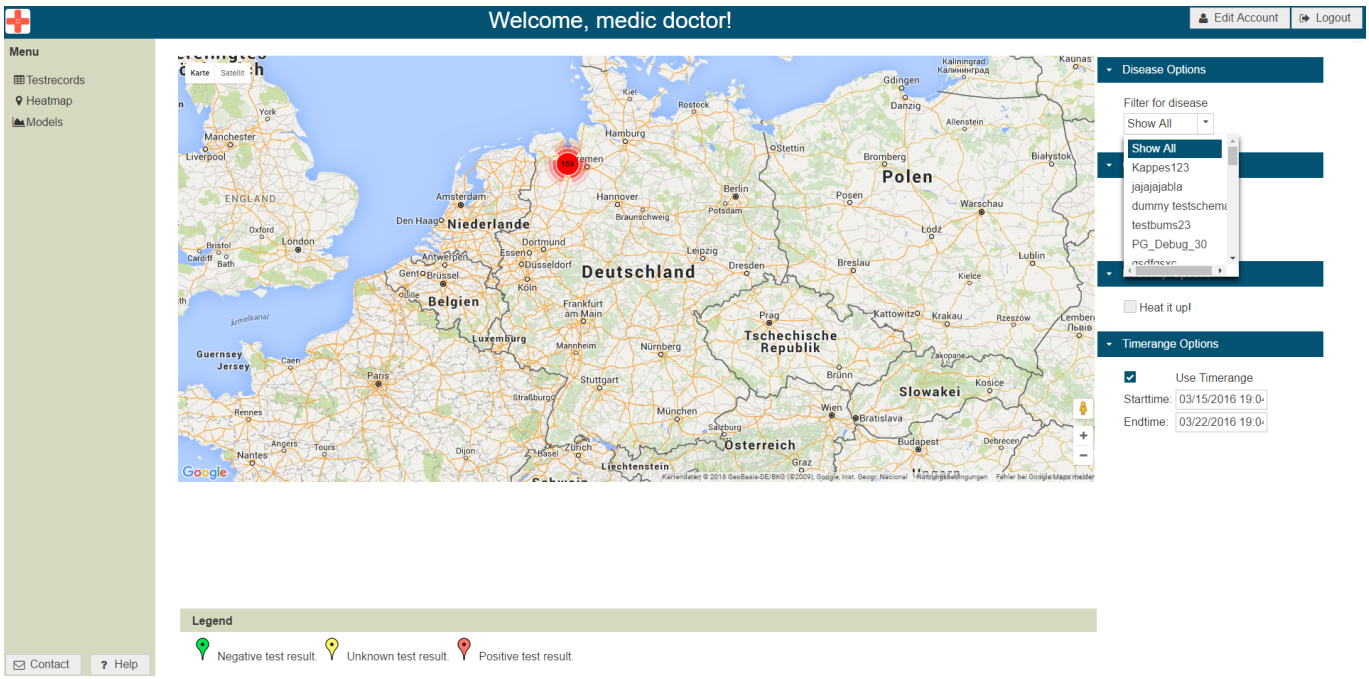



Abbildung 290: Filterfunktion nach Krankheiten auf der Heatmapansicht

**Layout der „Account-Editieren“-Seite anpassen**

Jeder Nutzer der Weboberfläche hat die Möglichkeit, seine eigenen Accountdaten zu editieren. Diese Funktion ist elementar für jede Benutzergruppe. Bisher ist die Ansicht, auf der diese Funktion umgesetzt ist, allerdings nicht mit dem restlichen Design der Weboberfläche abgestimmt.

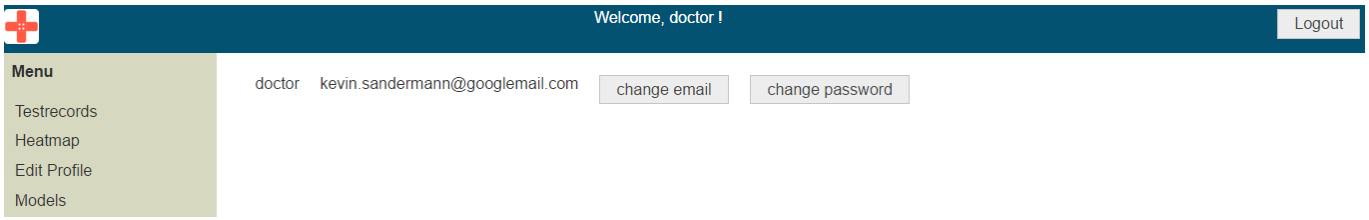



Abbildung 291: Bisherige Ansicht der Seite zum Editieren eines Accounts

Wie Abbildung 291 zeigt, sind die aufgeführten Informationen wie der Benutzername und die E-Mailadresse in keinerlei Kontext gesetzt und willkürlich positioniert. Außerdem sind die Informationen nicht vollständig, da durch neue Funktionen ebenfalls die Änderung von Praxisdaten möglich ist. Nachdem im Rahmen der Usability-Verbesserung und Einführung eines Corporate Designs Akkordeonmenüs als Standardaufbau von Menüs festgelegt wurden, wurde diese neue Seite ebenfalls in Form eines Akkordeonmenüs umgesetzt.

Abbildung 292 auf der nächsten Seite zeigt nun die verbesserte Seitenansicht: Alle Informationen sind durch ein auf- und zuklappbares Akkordeonmenü in die Rubriken „Login Information“, „Security Settings“ und „Address Settings“ gegliedert. Jede Rubrik enthält neben den aktuellen Daten eine Schaltfläche zum Ändern dieser Daten. Das Passwort wird dem Nutzer aus Sicherheitsgründen nicht angezeigt. Da der Nutzername des Accounts nicht editierbar ist, wird er nicht als Teil eines der Untermenüs angezeigt sondern ist Teil der Seitenüberschrift. Ebenfalls zu erwähnen ist, dass die Seiten nicht weiter über das linke Navigationsmenü zu erreichen sind, sondern am rechten oberen Seitenrand neben der „Logout“-Schaltfläche ein weitere Schaltfläche „Edit Account“ zu Verfügung steht.

**Verantwortliche Personen:**

- Kevin Sandermann


Abbildung 292: Verbesserte Ansicht der Seite zum Editieren eines Accounts

---

**Syntax- und Semantikprüfung sämtlicher Eingabefelder**

Im Laufe der ausführlichen Tests der Weboberfläche fiel auf, dass nicht alle Eingabefelder vollständig auf Semantik und Syntax überprüft werden. So war es beispielsweise möglich, im Formular für das Zurücksetzen des Passworts eine leere E-Mailadresse anzugeben. Ebenfalls war ein Fehler, dass beim Editieren von Accountdaten wie der Telefonnummer oder der Postleitzahl diese auf keinster Weise validiert wurden, sodass es beispielsweise möglich war, Buchstaben als Postleitzahl einzugeben. Sämtliche Eingabefelder des Webfrontends wurden auf eine fehlende Validierung überprüft und diese anschließend implementiert. Bei falscher Eingabe durch den Nutzer wird eine entsprechende Fehlermeldung angezeigt und der Nutzer hat die Möglichkeit, die Eingaben entsprechend zu korrigieren.

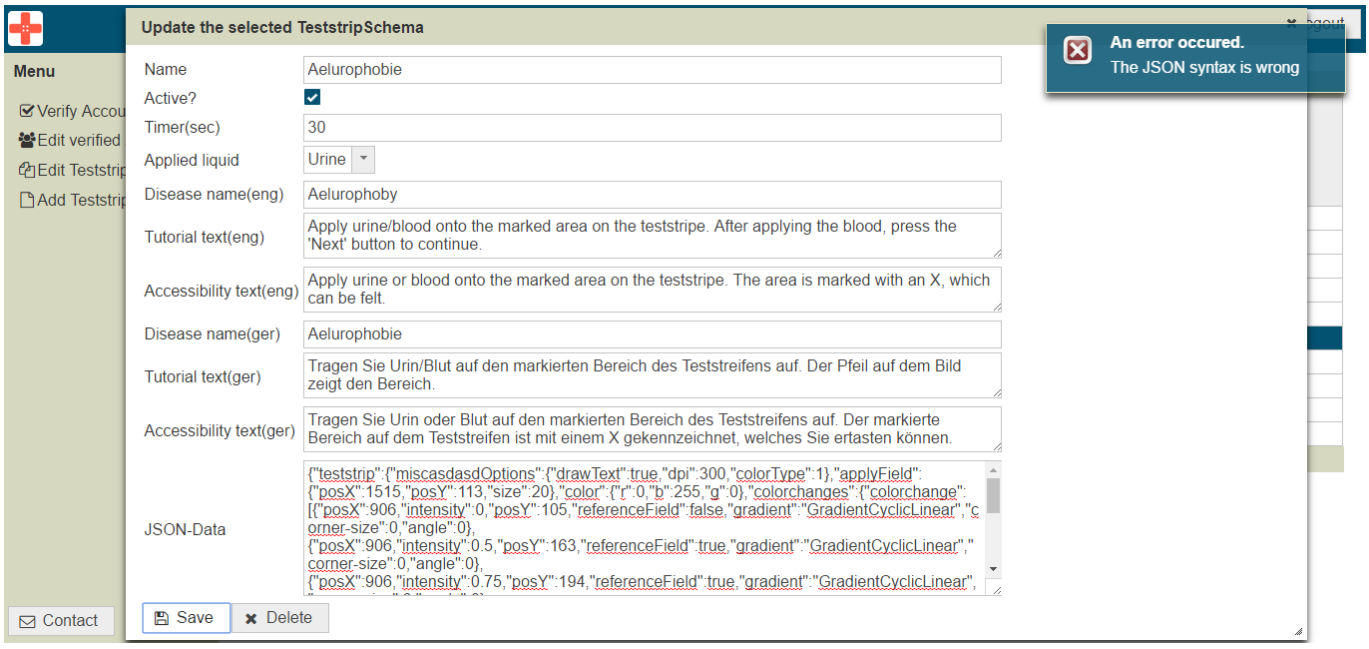



Abbildung 293: Validierung der eingegebenen JSON-Daten

Eine besonders kritische Umsetzung war dabei die in Abbildung 293 auf der vorherigen Seite dargestellte Validierung des JSON-Inhalts der Teststreifenschemata. Um diesen JSON-Inhalt sowohl strukturell als auch inhaltlich zu validieren, wurde der in das Webfrontend eingebettete Teststreifengenerator genutzt: Sobald der Nutzer versucht die geänderten JSON-Daten zu speichern, wird durch den Teststreifengenerator evaluiert, ob anhand dieser Daten ein korrekter Teststreifen generiert werden kann. Ist dies nicht der Fall, sind die Daten als fehlerhaft anzusehen und dem Nutzer wird die in Abbildung 293 auf der vorherigen Seite dargestellt Fehlermeldung angezeigt.

**Verantwortliche Personen:**

- Kevin Sandermann

---

**Behebung von Fehlern und Umsetzung kleinerer Funktionen**

Im Rahmen der Finalisierung des Webfrontends wurden ausgiebige Tests der Oberfläche durchgeführt. Durch diese Tests fielen einige kleinere Fehler sowie benötigte kleinere Funktionen auf. Diese Funktionen bzw. Fehler wurden während der Bearbeitung der anderen Arbeiten am Webfrontend behoben:

- Beim Bearbeiten eines Teststreifenschemas wurde eine Fehlermeldung für jedes nicht erreichbare mobile Endgerät in das Fehlerprotokoll der Weboberfläche geschrieben. Dies ist jetzt nicht mehr der Fall.
- Ein Fehler verursachte, dass auf der Karte der Medizineransicht keine Markierung für den Ort eines durchgeführten Tests mehr angezeigt wurde.
- An einigen Stellen der Weboberfläche wurden keine passenden Rückmeldungen bei erfolgreicher Ausführung einer Aktion gegeben. So wurde z.B. das erfolgreiche Anlegen eines neuen Teststreifenschemas nicht an den Nutzer gemeldet.
- Durch einen Fehler war es einem Statistiker nicht möglich, Zugriff auf die Heatmap-Ansicht zu erlangen.
- Die Legende zur Heatmap-Ansicht, wie sie in den Anforderungen beschrieben ist, fehlte und wurde ergänzt.
- Das Editieren und Anlegen von neuen Accounts war durch einen Fehler nicht mehr möglich.
- Durch einen falsch angegebenen Pfad wurde das Fehlerprotokoll des Webfrontends nicht gespeichert.
- Durch einen falsch angegebenen Pfad war es teilweise nicht möglich, den Teststreifengenerator aus der Weboberfläche heraus herunterzuladen.
- Die Diagnose-Details zeigen nun den Namen des Mediziners, welcher die Diagnose geschrieben hat, an.
- Die Tabelle aller durchgeführten Tests zeigt jetzt auch das Ergebnis der Diagnose als Spalte an.
- Eine nicht dokumentierte Änderung in der Schnittstelle von Google Maps verhinderte die Funktion der Heatmap Ansicht komplett. Nach zeitaufwändiger Behebung dieses Fehlers wurde deutlich, dass die Funktionswiederherstellung des Schiebereglers ebenfalls sehr aufwändig wäre. Da der Schieberegler allerdings nicht Teil der funktionalen Anforderungen ist, wurde er aus dem Webfrontend entfernt.

- An einigen Stellen war die Stabilität der Weboberfläche nicht gewährleistet, daher wurde der PrimeFaces-Server überarbeitet, so das jede mögliche Fehlermeldung abgefangen und protokolliert wird.

**Verantwortliche Personen:**

- Kevin Sandermann
- Nicolas Koch
- Raphael Kappes
- Daniel Wegmann

---

**Technologien**

**Cascading Style Sheets (CSS)** Bei CSS handelt es sich um eine sog. „Stylesheet-Sprache“. Sie wird unter Anderem dazu verwendet, Gestaltungsanweisungen für auf HTML-basierende Webseiten zu geben und so deren Optik zu gestalten.

**Systembeschreibung** Das System besteht zum Einen aus dem Tomcat-Webserver, auf dem das, mit Primefaces entwickelte, Webfrontend gehostet ist. Der andere Systemteil ist die PostgreSQL-Datenbank, die zur Datenhaltung dient.

**Evaluation** Das Webfrontend wurde durch intensive interne Tests und Kontrollen evaluiert. Die dadurch aufgedeckten Probleme wurden schriftlich festgehalten. Anhand der so entstandenen Liste von Problemen wurde das Frontend nach Überarbeitung erneut gemessen, um so feststellen zu können, dass alle gefundenen Probleme behoben werden konnten. Als zusätzliche Metrik wurden die im Lastenheft definierten funktionalen Anforderungen herangezogen. In Absprache mit dem Kunden wurde eine Anforderung, welche das lokale Speichern einer geschriebenen Diagnose bei Verbindungsabbruch fordert, gestrichen. Anschließend konnten alle Anforderungen erfüllt werden.

**Parameter** Wie in den vorherigen Sprints ist die Funktion des Webfrontends nicht durch Parameter eingeschränkt (siehe Sprints 5 und 7).

**Fazit** Alle geforderten funktionalen Anforderungen, die im Lastenheft festgelegt wurden, konnten erfüllt werden. Damit kann das Webfrontend als erfolgreich abgeschlossen angesehen werden. Darüber hinaus konnten z.B. mit der Integration des Teststreifengenerators sogar noch zusätzliche, zu Beginn des Projekts nicht definierte, Funktionen implementiert werden.

**Ausblick** Durch Abschluss der Anforderungen ist das Webfrontend in finaler Version 1.0 abgeschlossen. Denkbare zukünftige Erweiterungen wären die Unterstützung mehrerer Sprachen (gegenwärtig nur Englisch) und die Einführung eines Kommunikationssystems der Mediziner untereinander und zu den Administratoren und Statistikern.

### 4.8.7 Fertigstellung des Teststreifengenerators

- Priorität: Normal
- Arbeitspaketbeschreibung:
  Im Zuge der finalen Projektphase werden alle Komponenten fertig gestellt - darunter auch der Teststreifengenerator. Ziel ist es, dass der Teststreifengenerator an die aktuellen Anforderungen an das Teststreifendesign angepasst wird. Darunter fällt zum einen, dass bei Farbumschlägen unterschieden werden kann, ob es sich um ein Farbumschlagsfeld oder ein Referenzfeld handelt. Des Weiteren muss angegeben werden können, wo sich das Applizierungsfeld befindet, sowie was appliziert werden soll.

  Optional ist die Unterstützung mehrerer Sprachen zu implementieren. Dazu sollen anfänglich die Sprachen Deutsch und Englisch implementiert und die Erweiterbarkeit um weitere Sprachen gegeben sein.

  Weiterhin ist im Zuge dieses Arbeitspakets ein Handbuch für den Teststreifengenerator zu erstellen. In diesem Handbuch muss die Inbetriebnahme, der Aufbau und die Funktionsweise des Teststreifengenerators beschrieben werden.
- Vorbedingungen:
  Das Teststreifendesign ist genau definiert und die daraus entstehenden Anforderungen sind zwischen den Arbeitsgruppen kommuniziert worden.
- Nebenbedingungen:
  Optional unterstützt der Teststreifengenerator die Sprachen Englisch und Deutsch und lässt sich auch um weitere Sprachen erweitern.
- Nachbedingungen:
  Der Teststreifengenerator ist final, funktioniert fehlerfrei beziehungsweise verfügt über eine lückenlose Fehlerbehandlung und es existiert ein Handbuch in dem der Aufbau und die Funktionsweise des Teststreifengenerators genau beschrieben ist.
- Aufwand:
  drei Wochen
- Personen:
  - Danny Fonk

**Dokumentation** Im Zuge der Finalisierung aller Komponenten, muss diese auch für den Teststreifengenerator vorgenommen werden. Darunter fallen die Ergänzung ausstehender Funktionen, Fehlerbehandlung sowie die Erstellung eines Handbuchs.

**Ablauf** Die Fertigstellung des Teststreifengenerators verlief in Zusammenarbeit mit der Arbeitsgruppe die das Go-Server-Backend betreut und mit der Arbeitsgruppe die für die mobilen Applikationen zuständig ist. Dies war notwendig, da sich wandelnde Anforderungen an das Teststreifendesign eine Anpassung aller Komponenten fordert. Zudem wurde bezüglich der Auslieferung von zusätzlichen Kontextinformationen an die mobilen Endgeräte mit der Arbeitsgruppe der Weboberfläche zusammengearbeitet. Des Weiteren ist die Erstellung und Weitergabe von Kontextinformationen zum Teststreifen als eine weitere Anforderung entstanden. Darunter ist beispielsweise die Zeit, die nach dem Auftragen einer Körperflüssigkeit (Blut oder Urin) gewartet werden muss, bis die Reaktion vollständig abgeschlossen ist und das Ergebnis ausgewertet werden kann, zu verstehen. Diese Kontextinformationen sind jedoch nicht Teil dieses Arbeitspakets und werden innerhalb

der Weboberfläche erfasst und über das Server-Backend an die mobilen Applikationen verteilt. Der Teststreifengenerator wurde um die Einstellungen bezüglich eines „Applizierungsfeldes“ erweitert.

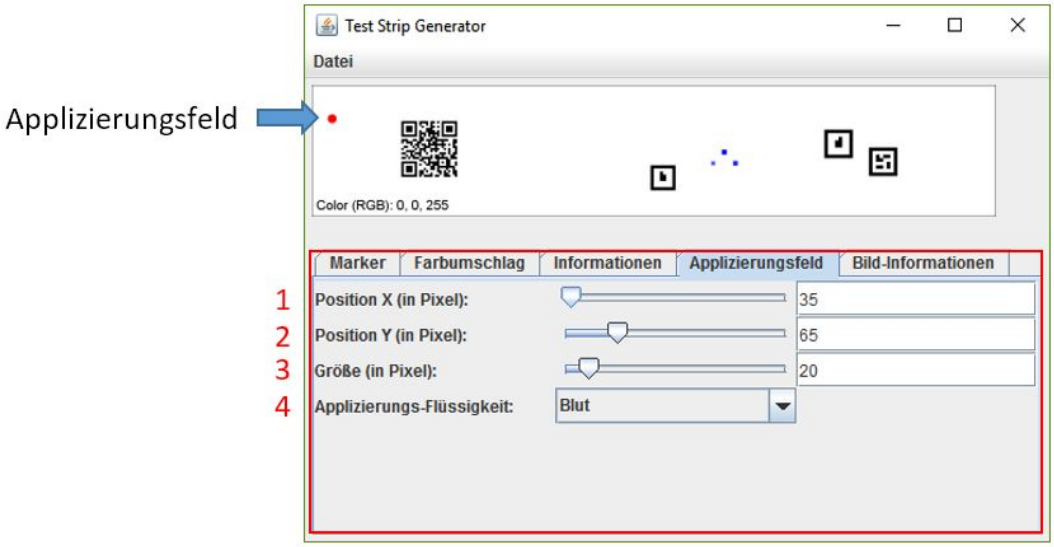



Abbildung 294: Aufbau des Menü-Reiters „Applizierungsfeld“

Der Aufbau gliedert sich wie folgt:

1. Die Position auf der X-Achse lässt sich mit einem Schieberegler oder dem nebenstehenden Eingabetextfeld einstellen.

2. Die Position auf der Y-Achse lässt sich mit einem Schieberegler oder dem nebenstehenden Eingabetextfeld einstellen.

3. Die Größe des Applizierungsfeldes lässt sich ebenfalls mittels Schieberegler oder nebenstehenden Eingabefeld einstellen.

4. Die Flüssigkeit, welche appliziert werden muss, lässt sich über ein Auswahlmenü festlegen. Derzeitig stehen Blut und Urin zur Auswahl.

Des Weiteren wurde die Möglichkeit ein Hintergrund-Bild zu setzen entfernt, da dies nur innerhalb der Entwicklung genutzt wurde und in der Praxis keine weitere Anwendung findet. Als zusätzliche Funktion wurde eine Sprachauswahl implementiert. Derzeitig sind die Sprachen Deutsch und Englisch vordefiniert. Eine Erweiterung durch andere Sprachen ist möglich, jedoch vor Projektende weder gefordert noch angedacht. Das Sprachauswahlmenü ist unter dem Menüpunkt „Datei“ zu finden (vgl. Abbildung 295 auf der nächsten Seite).

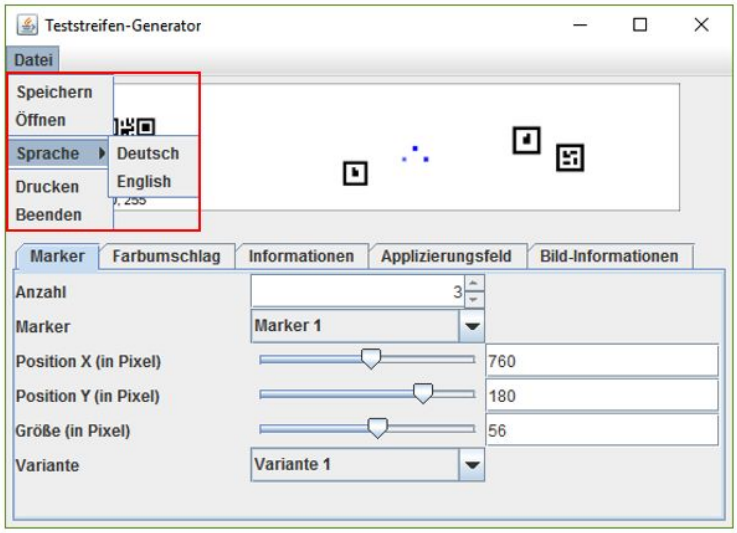



Abbildung 295: Menüstruktur des Teststreifengenerators

**Technologien** Es wurden für die Fertigstellung des Teststreifengenerators keine neuen Technologien eingesetzt.

**Systembeschreibung** Die Fertigstellung des Teststreifengenerators bzw. die Anpassung des Teststreifendesigns hatte Auswirkungen auf die Datenhaltung, genauer auf das Datenmodell. Eine aktualisierte Version des Datenmodells ist in Kapitel 4.8.6 zu finden.

**Evaluation** Der Teststreifengenerator wurde ausgiebig getestet. Die Fehlerbehandlung ist vollständig und aussagekräftig. Das entstandene Handbuch wurde gegengelesen und auf Korrektheit und Vollständigkeit hin überprüft.

**Parameter** Die Ergebnisse des Arbeitspaketes werden von keinen Parametern eingeschränkt.

**Fazit** Das Arbeitspaket „Fertigstellung des Teststreifengenerators“ konnte innerhalb von drei Wochen abgeschlossen werden. Die dabei entstandene Applikation entstand in enger Zusammenarbeit mit der Arbeitsgruppe der Weboberfläche, sowie denen der mobilen Applikationen und dem Go-Server.

**Ausblick** Durch die noch nicht vollständig definierten Anforderungen an das Teststreifendesign ist der Teststreifengenerator noch in seiner Handhabung variabel, sodass Einstellungen vorgenommen werden können die gegebenenfalls nicht vom Bildverarbeitungsalgorithmus erkannt werden können. Sofern alle Anforderungen definiert sind, ist der Teststreifengenerator dahingehend anzupassen, dass eine Erstellung von „fehlerhaften“ Teststreifenvarianten nicht mehr möglich ist. Dies wird vor Projektabschluss jedoch nicht der Fall sein, sodass eine nachträgliche Änderung des Teststreifengenerators nach Projektende denkbar - jedoch nicht zwingend erforderlich - ist.

### 4.8.8 Integration von Push-Benachrichtigungen

- Priorität: Normal
- Arbeitspaketbeschreibung:
  Die aktuelle Implementierung einer „pull“-orientierten Aktualisierung, soll durch eine „push“-gesteuerte Implementierung ersetzt werden. Dazu müssen die mobilen Applikationen insofern

angepasst werden, dass diese Push-Benachrichtigungen empfangen können (clientseitig). Außerdem muss serverseitig eine Logik implementiert werden, sodass Push-Benachrichtigungen versandt werden können. Eine Voruntersuchung, durchgeführt von Timo R. und Timo S., hat bereits die zu verwendeten Technologien festgelegt:

  - iOS: APNS[79] (Apple Push Notification Service)
  - Android: GCM[80] (Google Cloud Messaging)

  Es wäre möglich, dass man über GCM sowohl die Android-, als auch die iOS-Geräte mittels Push-Benachrichtigungen erreicht, jedoch steht hier die native Implementierung im Vordergrund, sodass sich für eine Implementierung entschieden wurde, welche die native Schnittstelle von Apple unterstützt. Es gibt zwei Anwendungsfälle für das Senden von Push-Benachrichtigungen:

  - Es gibt eine Änderung in den Teststreifenvarianten

    Werden die Teststreifenvarianten in der Weboberfläche verändert, dann ist es notwendig, dass diese Änderungen an die Geräte weitergereicht werden.
  - Es wurde eine Diagnose geschrieben

    Im Falle, dass eine Diagnose über die Weboberfläche geschrieben wurde, soll das entsprechende Gerät via Push-Benachrichtigung darüber informiert werden.

- Vorbedingungen:
  Die Weboberfläche verfügt über die Möglichkeiten eine Diagnose zu einem Test zu schreiben und über die Möglichkeit, dass die Teststreifenvarianten angepasst werden können.

- Nebenbedingungen:
  Es gibt keine Nebenbedingungen.

- Nachbedingungen:
  Über die Weboberfläche wird der Versand von Push-Benachrichtigungen initiiert. Diese Nachrichten können von den mobilen Applikationen empfangen und verarbeitet werden, sodass die neue Diagnose bzw. die neuen Teststreifenvarianten innerhalb der mobilen Applikationen abgelegt und/oder angezeigt werden.

- Aufwand:
  drei Wochen

- Personen:

  - Danny Fonk (serverseitig)
  - Timo Raß (clientseitig)
  - Timo Schlömer (clientseitig)

**Dokumentation für Push-Technologien** Um eine effiziente Abwicklung des Ablaufes von Veröffentlichungen von Teststreifenvarianten und dem Versand von Ergebnissen zu einem Test zu gewährleisten, wurde das bisher implementierte Pull-Verfahren durch ein Push-Verfahren ersetzt. Das Pull-Verfahren wurde clientseitig (vom Smartphone) initiiert und fragte in einem festgelegtem Intervall am Server nach Aktualisierungen von Testergebnissen sowie von Teststreifenvarianten. Das Push-orientierte Verfahren wird serverseitig ausgelöst und verringert so die Anfragen, welche auf dem Server eingehen. Diese werden nun nur noch ausgeführt, wenn eine Aktualisierung auch wirklich vorliegt.

[79] `https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html`

[80] `https://developers.google.com/cloud-messaging/`

**Ablauf** Das Arbeitspaket selbst wurde in zwei Teile untergliedert. Das Empfangen und Verarbeiten von Push-Benachrichtigungen, was auf den mobilen Endgeräten implementiert werden muss (clientseitig) sowie das Senden von Push-Benachrichtigungen bei verschiedenen Ereignissen (serverseitig). Der Ablauf einer Push-Benachrichtigung ist dabei wie folgt (vgl. Abbildung 296):

1. Die Push-Benachrichtigung wird durch die zwei Anwendungsfälle (das Schreiben einer Diagnose oder das Verändern von Teststreifenvarianten) initiiert.

2. Je nach Anwendungsfall, wird der Inhalt der zu sendenden Nachricht angepasst.

   Wenn eine Diagnose geschrieben wurde, ist der Inhalt der Nachricht „syncTests“.

   Wenn Teststreifenvarianten erzeugt, gelöscht oder verändert wurden, dann „syncSchemas“.

3. Die Push-Nachricht, wird abhängig vom Betriebssystem über APNS[81] oder GCM[82] an das Smartphone geschickt.

4. Das Smartphone fragt daraufhin, abhängig von dem in der Push-Nachricht enthaltenen Text, an der REST-Schnittstelle des Go-Server-Backends nach den aktualisierten Datensätzen.

5. Der Go-Server gibt dann die angefragten Datensätze für das anfragende mobile Endgerät heraus.

6. Die Datensätze werden auf dem mobilen Endgerät gespeichert und können dort angezeigt / verarbeitet werden.

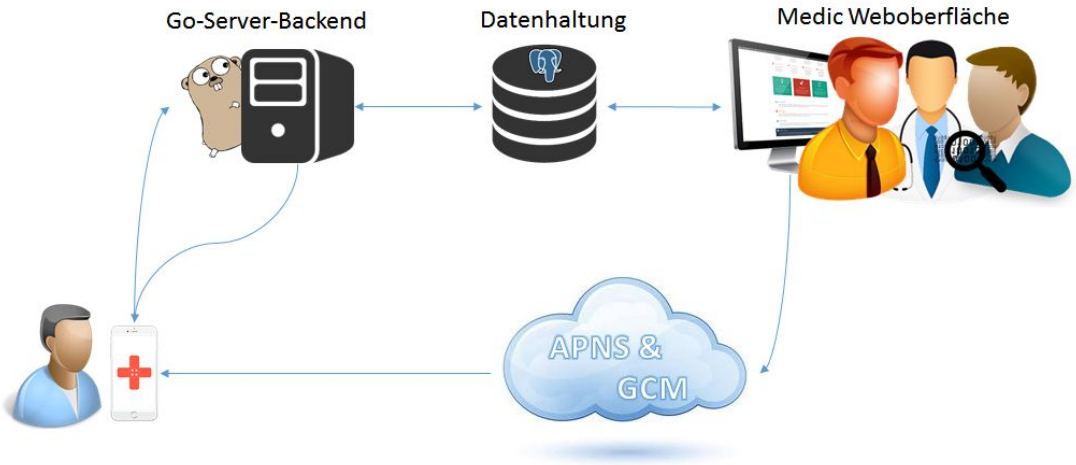



Abbildung 296: Ablauf einer Push-Benachrichtigung

Wichtig ist dabei der Fakt, dass die jeweiligen Daten (das Ergebnis eines Tests oder die verschiedenen Teststreifenvarianten) aus datenschutzrechtlichen Gründen nicht direkt in der Push-Benachrichtigung enthalten sind, sondern die Auslieferung selbst erst über den Go-Server initiiert wird.

Wie angesprochen, wurde die Implementierung in zwei Teile untergliedert - server- und clientseitig. Wie diese Implementierung verlief, wird nachfolgend aufgezeigt werden.

[81] Apple Push Notification Service Provider - Push-Implementierung für iOS-Geräte
[82] Google Cloud Messaging - Push-Implementierung für Android-Geräte

**Clientseitige Implementierung**

**Umsetzung für Android:** Zunächst wurde auf der Entwicklerseite von Google (Google Developers Console) ein neues Projekt angelegt. In diesem konnten die benötigten Schlüssel für die Authentifizierung mit den Push-Diensten von Google generiert werden. Insgesamt mussten zwei Schlüssel erzeugt werden, einer für das Server-Backend und einer für die mobile Android Applikation. Diese ermöglichen die Kommunikation vom Server-Backend, über die Push-Dienste zum mobilen Endgerät (vgl. Abbildung 297).


Abbildung 297: Verbindung mit dem Google Cloud Messaging Dienst

Die Nutzung des Push-Dienstes läuft wie folgt ab:

1. Das mobile Endgerät generiert eine Geräte ID:
   (a) Eine Verbindung mit dem GCM-Dienst wird aufgebaut.
   (b) Eine Geräte ID wird angefordert.
   (c) Der GCM-Dienst schickt eine Geräte ID an das Gerät.
2. Die Geräte ID wird im lokalen Speicher des Geräts hinterlegt.
3. Beim Versenden eines Tests wird die Geräte ID und das Betriebssystem (Android) als Teil der Daten zum Server übermittelt.

Die App deklariert einen Dienst, der vom Android System aufgerufen wird, wenn über die GCM-Verbindung des Geräts eine Nachricht empfangen wurde, welche an diese adressiert ist. Die App wird anschließend automatisch gestartet und verarbeitet die Nachricht.

Google bietet seinen Entwicklern zudem die Möglichkeit, mit dem Dienst für Push-Nachrichten auch iOS-Endgeräte zu unterstützen. Dies kann jedoch nur genutzt werden, wenn alle notwendigen Zertifikate und Einstellung bei Apple erzeugt und registriert wurden. Dies würde die Implementierung auf dem Server-Backend vereinfachen, da für alle mobilen Endgeräte nur der GCM-Dienst genutzt werden kann. Jedoch sind die geheimen Schlüssel und Zertifikate dadurch auch von Google einsehbar, was ein höheres Sicherheitsrisiko mit sich bringt. Da die Sicherheitsaspekte eine wesentliche Rolle im Medic Projekt darstellen, wurde entschieden die nativen Dienste der mobilen Betriebssysteme zu nutzen. Zusätzlich wären bei der Nutzung von GCM unter Apple beide Dienste aktiv, da das Server-Backend mit dem GCM-Dienst kommuniziert, welcher mit den APN-Diensten kommuniziert, welche dann mit der App auf dem mobilen Endgerät kommunizieren. Dies ist eine unnötig komplexe Kommunikationskette.

**Umsetzung für iOS:** Zunächst musste das Entwicklerkonto von den Betreuern zu einem Administrator hochgestuft werden, damit die Push-Dienste für das Projekt aktiviert werden konnten. Für die Freischaltung dieser Dienste ist im Normalfall das kostenpflichtige Entwicklerkonto notwendig. Da das Entwicklerkonto Teil des Universitätsprogramms von Apple ist, sind diese Funktionen für das Projekt ohne Kosten aktivierbar. Nachdem die Push-Dienste für das Projekt aktiviert wurden, mussten Zertifikate auf dem Mac-Rechner der Projektgruppe generiert werden. Danach war es erforderlich diese an Apple zu senden, um eine signierte Version des Zertifikats zu erhalten, welches dann in Xcode eingebunden werden musste. Der Kommunikationsweg vom Server-Backend, über die

Push-Dienste zum mobilen Endgerät ist dem GCM-Dienst ähnlich (vgl. Abbildung 298). Lediglich das Übertragungsprotokoll ist nicht HTTP sondern TCP.

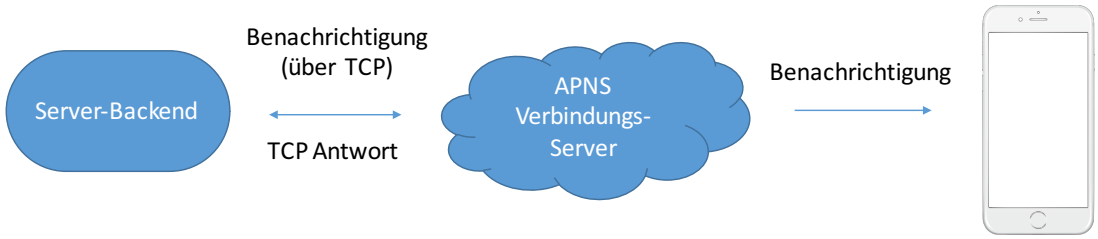



Abbildung 298: Verbindung mit dem Apple Push Notification Dienst

Die Nutzung des Push-Dienstes läuft wie folgt ab:

1. Das mobile Endgerät generiert eine Geräte ID:
   (a) Eine Geräte ID wird beim Betriebssystem angefordert.
   (b) Die App erhält vom Betriebssystem eine Geräte ID.
2. Die Geräte ID wird im lokalen Speicher des Geräts hinterlegt.
3. Beim Versenden eines Tests wird die Geräte ID und das Betriebssystem (iOS) als Teil der Daten zum Server übermittelt.

Wenn eine Push-Benachrichtigung an das mobile Gerät gesendet wird, startet das Betriebssystem die App und übermittelt die Daten aus der Nachricht. Abschließend musste das signierte Push-Zertifikat in ein .p12 Format umgewandelt werden. Dieses muss in das Server-Backend integriert werden, um die Push-Dienste nutzen zu können.

**Serverseitige Implementierung**

Die Implementierung für das Versenden von Push-Benachrichtigungen für Android-Geräte über GCM, verlief weitestgehend problemlos. Lediglich die im Fehlerfall nicht immer aussagekräftigen Fehlermeldungen, sind hier als Schwierigkeit zu vermerken, da dadurch die Fehlerbeseitigung unnötig erschwert wird. Aufgetretene Fehler waren beispielsweise, dass der API-Key nicht mehr gültig war, weil mittlerweile ein neuer generiert wurde oder dass das übermittelte Token (ein Token dient zur Identifikation, in diesem Fall zur Identifikation von mobilen Endgeräten) nicht erreicht werden konnte. Letzteres rührte daher, dass in der Datenbank noch eigens generierte Token verwaisten. Zur Identifikation von mobilen Endgeräten, stützen wir uns bisher auf selbst generierte Token, während nun die Token von den jeweiligen Providern (Apple und Google) selbst generiert und diese dann an das Go-Server-Backend gesandt werden. Die Verwendung erfolgt mittels API-Key, welcher auf der Seite für Entwickler (Google Developers[83]) generiert und jederzeit eingesehen werden kann. Die Implementierung für iOS-Geräte über APNS hingegen, war im Java-Umfeld etwas schwieriger, da seitens Apple keine Tutorials oder Vorgehensweisen hierfür beschrieben sind. Es wurde eine Recherche bezüglich verschiedener Bibliotheken durchgeführt, welche eine Nutzung von APNS unter Java ermöglichen. Darunter haben sich zwei Projekte aufgetan, welche auf den Online-Plattformen wie bspw. Foren wiederholt genannt wurden:

- Pushy[84] - „A Java library for sending APNs (iOS/OS X) push notifications“
- java-apns[85] - „Java Apple Push Notification Service Provider“

[83] https://developers.google.com
[84] https://github.com/relayrides/pushy
[85] https://github.com/notnoop/java-apns

Anfangs wurde damit begonnen Pushy zu verwenden, was jedoch an verschiedensten Stellen scheiterte. Nach einer gewissen Anzahl an Rückschlägen, wurde das Projekt auf java-apns umgestellt, womit eine Integration schnell umgesetzt werden konnte. Die Bibliothek ist sehr gut dokumentiert und verfügt über ausreichend Beispiel-Implementierungen, welche adaptiert und im Anschluss erfolgreich an unsere Bedürfnisse angepasst werden konnten. Für das Senden von Push-Benachrichtigungen, benötigt man ferner noch ein Zertifikat um sich an den Apple-Servern zu authentifizieren. Dieses Zertifikat kann im Apple's iOS Provisioning Portal[86] generiert und heruntergeladen werden. Das Zertifikat, wird bei jedem Aufruf überprüft, sodass dies dauerhaft im Projekt eingebunden werden muss. Das führte anfangs zu Problemen, da das Zertifikat eine Binär-Datei ist. Dadurch, dass Git zur Versionsverwaltung eingesetzt wird, kam es hier zu Schwierigkeiten, da Git mit Binärdateien nicht umgehen kann. Lädt man sich das Zertifikat über das Git herunter und versucht das Projekt zu starten, kommt es zu Fehlern und das Zertifikat scheint ungültig. Eine Fehlerbehandlung über das Git wurde nicht implementiert und ist auch nicht angedacht. Die derzeitige Fehlervermeidungsstrategie sieht vor, dass die Zertifikat-Datei manuell im Pfad (\Primefaces\Medic\src\main\resources\) abzulegen ist.

**Technologien** Für die serverseitige Implementierung, wurde eine Java-Bibliothek verwendet, welche es erlaubt aus Java-Quellcode heraus Push-Benachrichtigungen über APNS zu versenden - Java Apple Push Notification Service Provider[85]. Dabei handelt es sich um ein Github-Projekt, was unter der BSD 3-Clause License[87] geführt wird. Für das Senden von Push-Benachrichtigungen an Android-Geräte, wurde keine weitere Bibliothek eingebunden. Es wurde sich vollständig auf eine Eigenentwicklung gestützt, da das Versenden von Push-Benachrichtigungen über einen HTTP-Post-Request durchgeführt werden kann.

**Systembeschreibung** Das Datenmodell, genauer die Tabelle „mobile_device" musste um die Spalte „os" (Abkürzung für Operating System, dt. Betriebssystem) erweitert werden, sodass die jeweiligen Dienste (APNS und GCM) nicht mit „falschen" Nachrichten versorgt werden und ein effizienter Versand an die jeweiligen Geräte damit gewährleistet ist.

**Evaluation** Die Push-Benachrichtigungen, wurden auf den folgenden Geräten erfolgreich getestet:

- iPhone 5 - iOS 9
- iPhone 6s - iOS 9
- iPad mini 2 - iOS 9
- Sony Xperia Z3 Compact - Android

**Parameter** Die Ergebnisse des Arbeitspaketes werden durch die Gültigkeit der Push-Zertifikate eingeschränkt. Nach dem Ablauf (31.01.2017) eines solchen Zertifikates, muss dieses erneuert werden, um weiterhin Push-Nachrichten versenden zu können.

**Fazit** Das Arbeitspaket „Integration von Push-Benachrichtigungen", konnte innerhalb von drei Wochen abgeschlossen werden. Eine Integration in die Weboberfläche erfolgte jedoch aus organisatorischen Gründen erst zu einem späteren Zeitpunkt.

[86]https://developer.apple.com/membercenter/index.action
[87]https://opensource.org/licenses/BSD-3-Clause

### 4.8.9 Fazit des Sprints

Wie in der Einleitung bereits in Aussicht gestellt, wurden in diesem Sprint alle Anforderungen des Lastenhefts vollständig erfüllt.

Durch die Arbeitspakete „Fertigstellung der iOS-App“ ( 4.8.2 auf Seite 347) und „Fertigstellung der Android-App“ ( 4.8.4 auf Seite 380) wurden nicht nur die im Lastenheft definierten, sondern darüber hinaus auch zusätzliche Funktionen, wie die Zugänglichkeit für körperlich beeinträchtigte Personen, z.B. Blinde, umgesetzt. Im Arbeitspaket „Bildverarbeitungsalgorithmus optimieren“ ( 4.8.3 auf Seite 353) konnte die Auswertung des Farbumschlags deutlich verbessert werden. Dadurch werden nun auch bei Aufnahmen des gleichen Teststreifens durch verschiedene Kameras immer die gleichen Auswertungen zurückgeliefert. Das Arbeitspaket „Fertigstellung des Webfrontends“ ( 4.8.6 auf Seite 396) hat sich besonders auf die Aufwertung der Nutzeroberfläche konzentriert und zudem die durch die Seminararbeiten „Integration und Evaluation von epidemiologischen Vorhersagemodellen in medizinische Datenhaltungssysteme“ und „Evaluation der Usability eines Onlineportals für Mediziner“ gewonnenen Erkenntnisse in das Webfrontend integriert. Beim Arbeitspaket „Fertigstellung des Teststreifengenerators“ ( 4.8.7 auf Seite 414) wurde der Teststreifengenerator hinsichtlich der aktualisierten Eigenschaften der Teststreifenschemata erweitert. So kann jetzt z.B. die Position des Applizierungsfeldes festgelegt und zwischen Referenz- und Farbumschlagsfeldern unterschieden werden. Im Arbeitspaket „Fertigstellung des Serverbackends“ ( 4.8.5 auf Seite 390) wurde der Go-Server an den veränderten Prozessablauf angepasst, sodass jetzt auch Push-Nachrichten an die iOS- und Android-App verschickt werden können.

Abbildung 299 zeigt die Komponenten des Systems nach Beendigung von Sprint 9 zusammen mit den Anteilen an erfüllten Anforderungen.

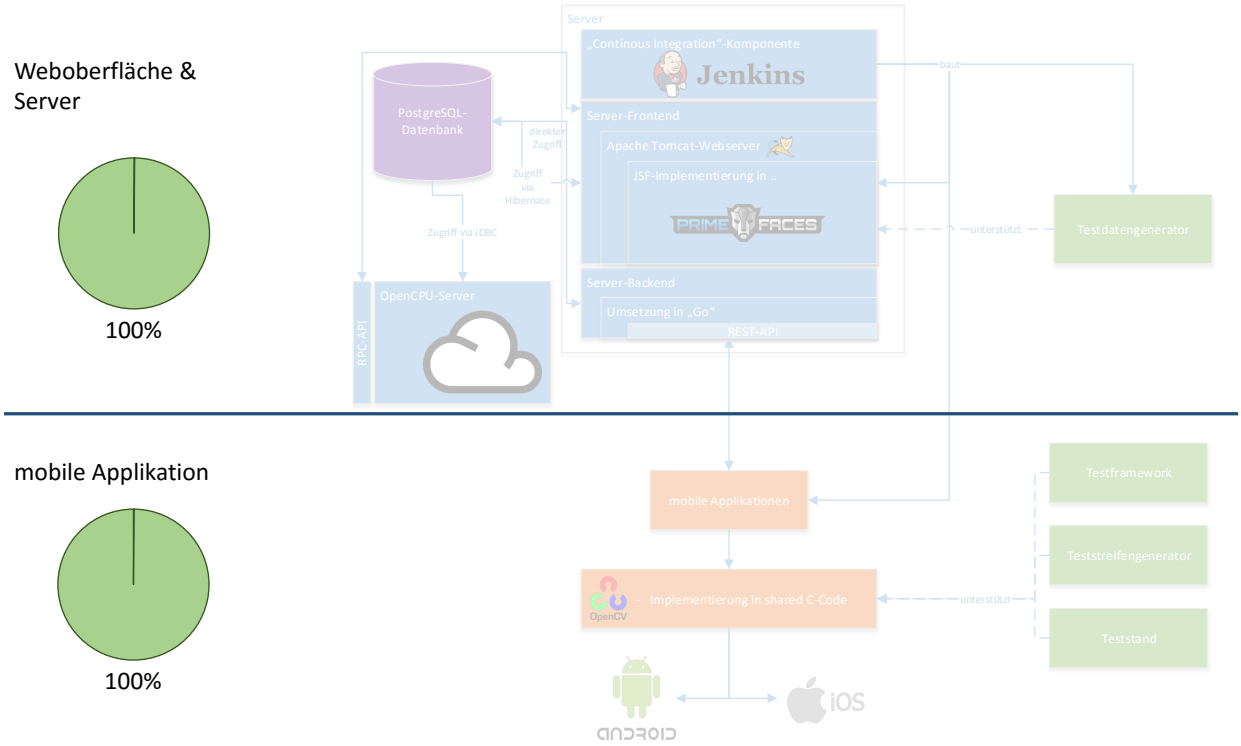



Abbildung 299: Systemkomponenten nach Abschluss von Sprint 9

Wie aus der Abbildung ersichtlich ist, konnten 100% aller Anforderungen erfüllt werden, womit das Projekt aus entwicklungstechnischer Sicht als abgeschlossen betrachtet werden kann. Neben den funktionalen wurden in diesem Sprint auch die noch offenen nichtfunktionalen Anforderungen erfüllt. Durch diese wurde nicht nur die Optik aller grafischer Nutzeroberflächen stark aufgebessert, sondern

auch deren Bedienung deutlich intuitiver und übersichtlicher gestaltet.

Abschließend wurde die Gesamtdokumentation, die das komplette Projekt beschreibt, erstellt. Mit diesem letzten Schritt konnte das Projekt auch aus organisatorischer Sicht vollständig abgeschlossen werden. Hierbei wurde besonders darauf geachtet, dass der Aufbau der einzelnen Sprintdokumentationen konsistent zueinander ist.

| Teststreifen | Gerät | Detektierbar | Problem |
|---|---|---|---|
| 206a auf Seite 292 | iPad Mini 3 | Gar nicht | Wenn irgendwelche Marker erkannt wurden, wurden immer nur zwei bis drei erkannt, der vierte und fünfte aber nie. |
| 206b auf Seite 292 | iPad Mini 3 | Manchmal | Wenn irgendwelche Marker erkannt wurden, wurden immer drei erkannt, der vierte nur manchmal Mal. |
| 206c auf Seite 292 | iPad Mini 3 | Manchmal | Wenn irgendwelche Marker erkannt wurden, wurden immer drei erkannt, der vierte nur manchmal Mal. |
| 206d auf Seite 292 | iPad Mini 3 | Sehr oft | Wenn irgendwelche Marker erkannt wurden, wurden immer drei erkannt, der vierte ist sehr oft erkannt worden. |
| 207a auf Seite 292 | iPad Mini 3 | Oft | Wenn irgendwelche Marker erkannt wurden, wurden immer zwei erkannt, der dritte und vierte sind oft erkannt worden. |
| 207b auf Seite 292 | iPad Mini 3 | Immer | Wenn irgendwelche Marker erkannt wurden, wurden immer alle erkannt. |
| 207c auf Seite 292 | iPad Mini 3 | Gar nicht | Wenn irgendwelche Marker erkannt wurden, wurden immer zwei erkannt, der dritte und vierte jedoch nicht. |
| 206d auf Seite 292 | Sony Xperia E1 | Sehr oft | Wenn irgendwelche Marker erkannt wurden, wurden immer drei erkannt, der vierte ist sehr oft erkannt worden. |
| 207a auf Seite 292 | Sony Xperia E1 | Oft | Wenn irgendwelche Marker erkannt wurden, wurden immer zwei erkannt, der dritte und vierte sind oft erkannt worden. |
| 207b auf Seite 292 | Sony Xperia E1 | Immer | Wenn irgendwelche Marker erkannt wurden, wurden immer alle erkannt. |

Tabelle 17: Ergebnisse der Evaluation der Smartphones

| Alle Smartphones | Entfernung | Winkel | | | Erkannt (in %) | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 20 cm | 0° | 0° | 0° | 77,8 | 100 | 100 |
| | 20 cm | 30° | 0° | 0° | 66,7 | 88,9 | 88,9 |
| | 20 cm | 60° | 0° | 0° | 55,6 | 88,9 | 88,9 |
| | 20 cm | 0° | 10° | 0° | 55,6 | 100 | 100 |
| | 20 cm | 30° | 10° | 0° | 55,6 | 88,9 | 88,9 |
| | 20 cm | 60° | 10° | 0° | 44,4 | 77,8 | 77,8 |
| | 20 cm | 0° | 25° | 0° | 33,3 | 100 | 100 |
| | 20 cm | 30° | 25° | 0° | 44,4 | 88,9 | 88,9 |
| | 20 cm | 60° | 25° | 0° | 44,4 | 77,8 | 77,8 |
| | 20 cm | 0° | 0° | 25° | 77,8 | 100 | 100 |
| | 20 cm | 30° | 0° | 25° | 66,7 | 100 | 88,9 |
| | 20 cm | 60° | 0° | 25° | 66,7 | 77,8 | 77,8 |
| | 20 cm | 0° | 10° | 25° | 44,4 | 100 | 100 |
| | 20 cm | 30° | 10° | 25° | 11,1 | 55,6 | 55,6 |
| | 20 cm | 60° | 10° | 25° | 44,4 | 77,8 | 77,8 |
| | 20 cm | 0° | 25° | 25° | 33,3 | 100 | 88,9 |
| | 20 cm | 30° | 25° | 25° | 0 | 0 | 0 |
| | 20 cm | 60° | 25° | 25° | 44,4 | 77,8 | 77,8 |
| | 20 cm | 0° | 0° | 50° | 55,6 | 100 | 100 |
| | 20 cm | 30° | 0° | 50° | 0 | 0 | 0 |
| | 20 cm | 60° | 0° | 50° | 33,3 | 77,8 | 77,8 |
| | 20 cm | 0° | 10° | 50° | 44,4 | 100 | 100 |
| | 20 cm | 30° | 10° | 50° | 11,1 | 88,9 | 77,8 |
| | 20 cm | 60° | 10° | 50° | 33,3 | 77,8 | 77,8 |
| | 20 cm | 0° | 25° | 50° | 33,3 | 66,7 | 66,7 |
| | 20 cm | 30° | 25° | 50° | 22,2 | 0 | 0 |
| | 20 cm | 60° | 25° | 50° | 44,4 | 22,2 | 22,2 |

Tabelle 19: Zusammenfassung aller Smartphones bei Distanz von 20cm

| Alle Smartphones | Entfernung | Winkel | | | Erkannt (in %) | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 35 cm | 0° | 0° | 0° | 55,6 | 77,8 | 66,7 |
| | 35 cm | 30° | 0° | 0° | 33,3 | 55,6 | 55,6 |
| | 35 cm | 60° | 0° | 0° | 55,6 | 66,7 | 66,7 |
| | 35 cm | 0° | 10° | 0° | 66,7 | 77,8 | 66,7 |
| | 35 cm | 30° | 10° | 0° | 22,2 | 44,4 | 33,3 |
| | 35 cm | 60° | 10° | 0° | 55,6 | 66,7 | 66,7 |
| | 35 cm | 0° | 25° | 0° | 66,7 | 77,8 | 77,8 |
| | 35 cm | 30° | 25° | 0° | 55,6 | 77,8 | 55,6 |
| | 35 cm | 60° | 25° | 0° | 55,6 | 66,7 | 66,7 |
| | 35 cm | 0° | 0° | 25° | 77,8 | 77,8 | 77,8 |
| | 35 cm | 30° | 0° | 25° | 66,7 | 77,8 | 66,7 |
| | 35 cm | 60° | 0° | 25° | 66,7 | 66,7 | 66,7 |
| | 35 cm | 0° | 10° | 25° | 77,8 | 77,8 | 77,8 |
| | 35 cm | 30° | 10° | 25° | 11,1 | 22,2 | 22,2 |
| | 35 cm | 60° | 10° | 25° | 66,7 | 66,7 | 66,7 |
| | 35 cm | 0° | 25° | 25° | 66,7 | 77,8 | 77,8 |
| | 35 cm | 30° | 25° | 25° | 22,2 | 0 | 0 |
| | 35 cm | 60° | 25° | 25° | 66,7 | 66,7 | 66,7 |
| | 35 cm | 0° | 0° | 50° | 66,7 | 77,8 | 77,8 |
| | 35 cm | 30° | 0° | 50° | 0 | 0 | 0 |
| | 35 cm | 60° | 0° | 50° | 66,7 | 66,7 | 66,7 |
| | 35 cm | 0° | 10° | 50° | 66,7 | 77,8 | 77,8 |
| | 35 cm | 30° | 10° | 50° | 55,6 | 77,8 | 55,6 |
| | 35 cm | 60° | 10° | 50° | 66,7 | 66,7 | 55,6 |
| | 35 cm | 0° | 25° | 50° | 55,6 | 44,4 | 44,4 |
| | 35 cm | 30° | 25° | 50° | 66,7 | 0 | 0 |
| | 35 cm | 60° | 25° | 50° | 33,3 | 11,1 | 11,1 |

Tabelle 20: Zusammenfassung aller Smartphones bei Distanz von 35cm

| Alle Smartphones | Entfernung | Winkel | | | Erkannt (in %) | | |
|---|---|---|---|---|---|---|---|
| | | x-Rotation | y-Rotation | z-Rotation | Sprint 7 | Sprint 8 | Sprint 8 (mit Abbruchbedingung) |
|  | 50 cm | 0° | 0° | 0° | 66,7 | 77,8 | 66,7 |
| | 50 cm | 30° | 0° | 0° | 44,4 | 55,6 | 33,3 |
| | 50 cm | 60° | 0° | 0° | 55,6 | 66,7 | 44,4 |
| | 50 cm | 0° | 10° | 0° | 66,7 | 77,8 | 66,7 |
| | 50 cm | 30° | 10° | 0° | 11,1 | 55,6 | 33,3 |
| | 50 cm | 60° | 10° | 0° | 44,4 | 55,6 | 33,3 |
| | 50 cm | 0° | 25° | 0° | 55,6 | 66,7 | 66,7 |
| | 50 cm | 30° | 25° | 0° | 11,1 | 55,6 | 44,4 |
| | 50 cm | 60° | 25° | 0° | 44,4 | 55,6 | 44,4 |
| | 50 cm | 0° | 0° | 25° | 66,7 | 77,8 | 66,7 |
| | 50 cm | 30° | 0° | 25° | 33,3 | 55,6 | 44,4 |
| | 50 cm | 60° | 0° | 25° | 55,6 | 55,6 | 33,3 |
| | 50 cm | 0° | 10° | 25° | 55,6 | 66,7 | 44,4 |
| | 50 cm | 30° | 10° | 25° | 22,2 | 44,4 | 11,1 |
| | 50 cm | 60° | 10° | 25° | 44,4 | 55,6 | 33,3 |
| | 50 cm | 0° | 25° | 25° | 55,6 | 66,7 | 44,4 |
| | 50 cm | 30° | 25° | 25° | 11,1 | 11,1 | 0 |
| | 50 cm | 60° | 25° | 25° | 44,4 | 44,4 | 33,3 |
| | 50 cm | 0° | 0° | 50° | 55,6 | 66,7 | 55,6 |
| | 50 cm | 30° | 0° | 50° | 0 | 11,1 | 11,1 |
| | 50 cm | 60° | 0° | 50° | 33,3 | 33,3 | 33,3 |
| | 50 cm | 0° | 10° | 50° | 33,3 | 44,4 | 44,4 |
| | 50 cm | 30° | 10° | 50° | 55,6 | 44,4 | 44,4 |
| | 50 cm | 60° | 10° | 50° | 33,3 | 44,4 | 33,3 |
| | 50 cm | 0° | 25° | 50° | 22,2 | 22,2 | 22,2 |
| | 50 cm | 30° | 25° | 50° | 44,4 | 0 | 0 |
| | 50 cm | 60° | 25° | 50° | 11,1 | 0 | 0 |

Tabelle 21: Zusammenfassung aller Smartphones bei Distanz von 50cm

| | | 0cm | 15cm | 30cm | Total |
|---|---|---|---|---|---|
| Sprint 7 | Teststreifen 1 | 0.245247148289 | 0.269180754226 | 0.132639791938 | 0.286837748344 |
| | Teststreifen 2 | 0.0704500978474 | 0.029702970297 | 0.0 | 0.0335747202107 |
| Sprint 8 | Teststreifen 1 | 0.403409090909 | 0.342412451362 | 0.15953307393 | 0.362282878412 |
| | Teststreifen 2 | 0.397660818713 | 0.195266272189 | 0.0236686390533 | 0.207100591716 |

Tabelle 49: Übersicht der Gesamtperformance über alle Videos

# 5 Produktauslieferungen

In diesem Kapitel werden die einzelnen Softwareprodukte, welche dem Kunden zum Abschluss des Projektes übergeben werden, erklärt. Dabei wird grundlegend auf den Lieferumfang der einzelnen Komponenten eingegangen. Detaillierte Informationen können den Handbüchern entnommen werden, welche sich im Anhang dieses Dokumentes befinden. ( 7.2 auf Seite XIX)

## 5.1 Webfrontend

Das Webfrontend stellt die zentrale Schnittstelle für Doktoren, Statistiker und Administratoren dar. Es handelt sich dabei um eine Webapplikation, die über den Browser aufgerufen werden kann. Unterstützt werden dabei die Webbrowser Mozilla Firefox, Google Chrome, Safari und Internet Explorer, jeweils in aktuellen Versionen.
Neben der Weboberfläche enthält das Produkt „Webfrontend“ zusätzlich die Komponenten „Datenbank“, „Webserver“ und „OpenCPU Server“. Diese sind allerdings für den Nutzer nicht ersichtlich und stellen im Hintergrund die benötigten Funktionalitäten zur Verfügung. Detaillierte Beschreibungen zu diesen Komponenten sind der mitgelieferten Installationsanleitung zu entnehmen.
Die folgenden Unterkapitel zeigen kurz die bereitgestellten Funktionalitäten für die genannten Rollen auf. Für detaillierte Beschreibungen der Weboberfläche können die Benutzerhandbücher herangezogen werden (vgl. Handbuch für Mediziner A, Handbuch für Statistiker B und Handbuch für Administratoren C).

### 5.1.1 Doktorenoberfläche

Die Doktorenoberfläche stellt Medizinern Funktionen zur Auswertung von Probandentests zur Verfügung. Abbildung 300 zeigt den Grundaufbau der Webseite.

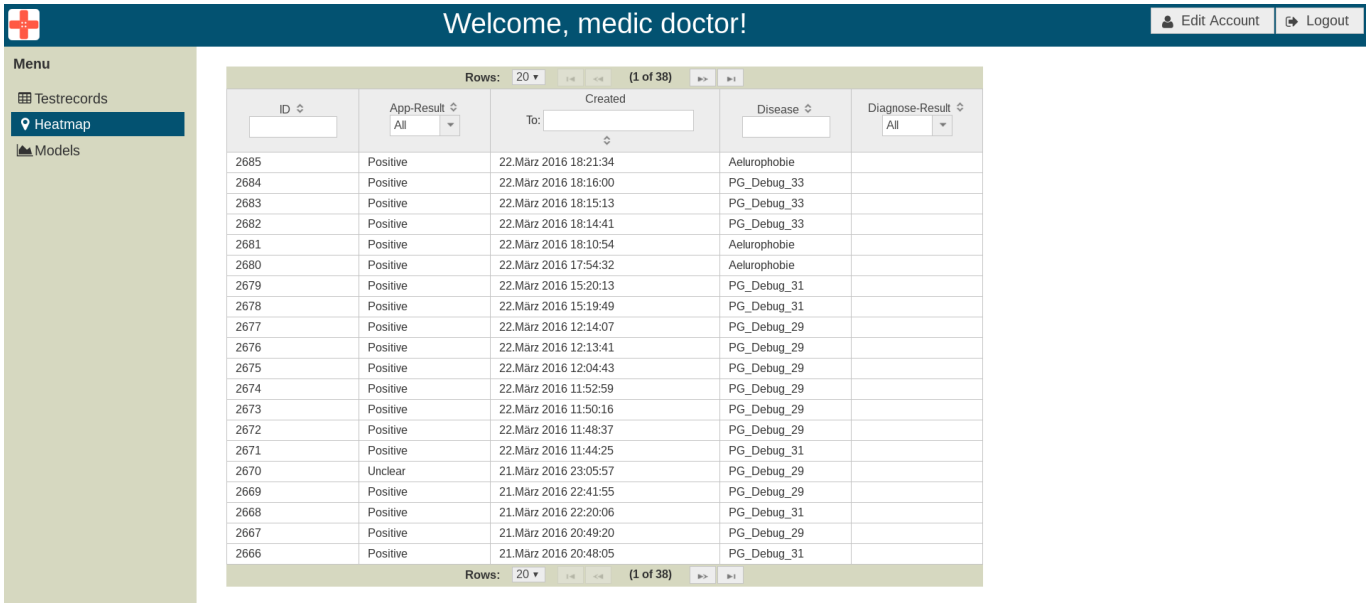



Abbildung 300: Aufbau der Doktorenoberfläche

Der Menüpunkt *Testrecords* bietet Funktionalitäten zur Auswertung und Einsicht von durchgeführten Krankheitstests. Mit der *Heatmap* können geographische Daten zu den durchgeführten Krankheitstests visualisiert werden. Abschließen bietet der Menüpunkt *Models* Möglichkeiten zur statistischen Auswertung. So können zum Beispiel Graphen generiert werden.

### 5.1.2 Statistikeroberfläche

Mit der Statistikeroberfläche können statistische Modelle erstellt und ausgeführt werden. Die so generierten Informationen können zum Beispiel Wissen erzeugen, welches bei der Vorbeugung von Pandemien unterstützt. Abbildung 301 auf der nächsten Seite zeigt den grundlegenden Aufbau der bereitgestellten Weboberfläche.

In der *Development Area* können neue Modelle in der Programmiersprache R erstellt werden. Die Reiter *Models* und *Heatmap* sind identisch zu den entsprechenden Menüpunkten der Doktorenoberfläche.

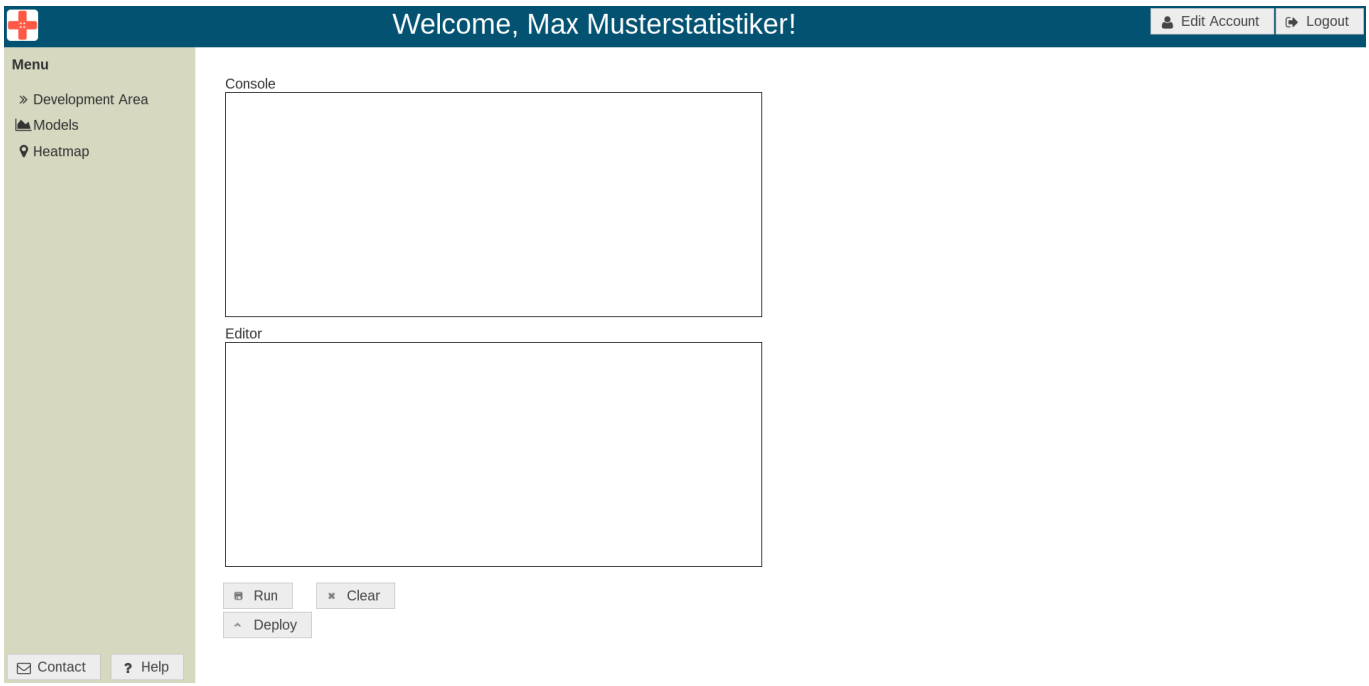



Abbildung 301: Aufbau der Statistikeroberfläche

### 5.1.3 Administratorenoberfläche

Die Administratorenoberfläche dient zur Verwaltung der Infrastruktur. Abbildung 302 zeigt den Aufbau dieser Benutzeroberfläche.


Abbildung 302: Aufbau der Administratorenoberfläche

Mit den Reitern *Verfify Accounts* und *Edit Verified Accounts* können angeforderte bzw. vorhandene Nutzeraccounts verwaltet werden. Es werden Funktionen zur Erstellung, Bearbeitung und Löschung dieser Accounts zur Verfügung gestellt.
Die Menüpunkte *Edit Teststripschemata* bzw.*Add Teststripschemata* dienen zur Verwaltung von Testreifenvarianten. Sobald es neue Krankheitstests im Umlauf gibt, können diese hier ins System eingepflegt werden.

## 5.2 Backend

Die Aufgabe des Backends ist es, die Kommunikation zwischen den mobilen Endgeräten und dem Server zu ermöglichen. Aufgrund dieser Aufgabe und dem Kontext, für diesen das Produkt entwickelt wurde, ergeben sich folgende Anforderungen an das Backend:

- Hohe Skalierbarkeit und Funktionalität bei Stoßzeiten, zum Beispiel bei der massenhaften Ausgabe von Teststreifen an die Bevölkerung.
- Die Schnittstelle, die durch das Backend zur Verfügung gestellt wird, muss kompatibel zu iOS und Android sein, um einen möglichst großen Markt erschließen zu können.
- Die Kommunikation zwischen Backend und den mobilen Endgeräten muss nach aktuellen Standards verschlüsselt sein. Dies trägt zur Anonymität der Ergebnisse bei.

Um die Skalierbarkeit zu gewährleisten wurde Golang als Programmiersprache gewählt. Golang ist eine junge Programmiersprache, die sich stetig wachsender Beliebtheit erfreut und mittlerweile zu den beliebtesten Programmiersprachen gehört [25][26]. Zudem konnte in zahlreichen Fallbeispielen gezeigt werden, dass Golang ressourcenschonend, robust und skalierbar ist [27][28].

Um die Kommunikation zwischen Backend und den mobilen Applikationen möglichst kompatibel zu allen Betriebssystemen zu halten, wurde ein REST Protkoll gewählt. Dieses textbasierte Protokoll erfüllt alle Anforderungen zur Datenübertragung und ist einfach in Android und iOS zu integrieren.

Die REST Schnittstelle zwischen Backend und mobilen Applikationen ist mit dem *de facto* Standard für Verschlüsselung im Web, SSL, gesichert [29]. Das Backend ermöglicht den Zugang der Mobilen Applikationen über die REST Schnittstelle an die Datenbank, die auch von der Weboberfläche angesprochen wird. Dadurch ist eine nahtlose Verbindung zwischen der Weboberfläche und der mobilen Applikationen hergestellt.

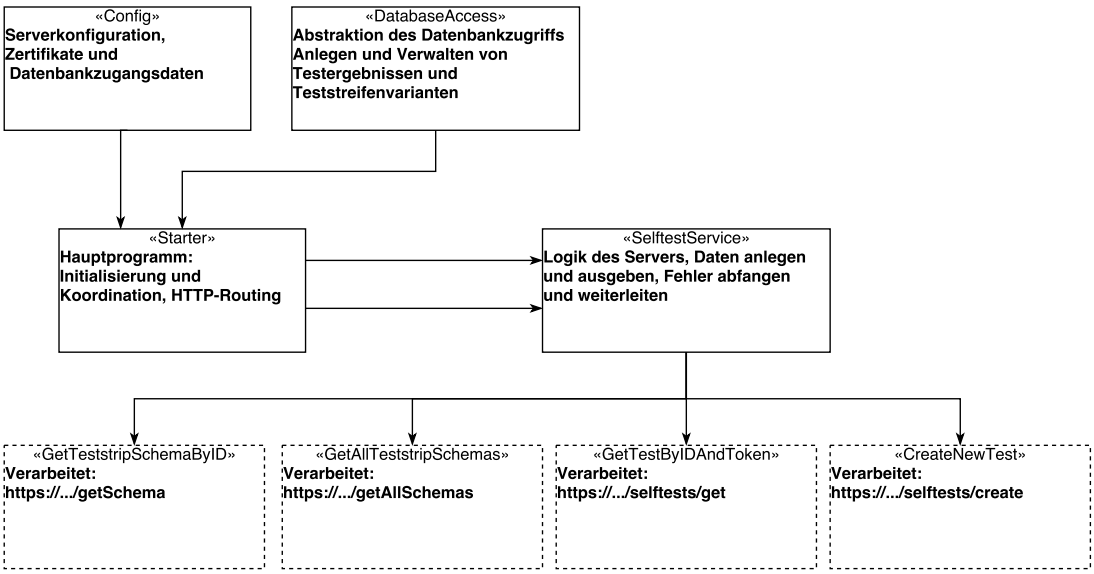



Abbildung 303: Schematische Darstellung des Server-Backends

In Abbildung 303 wird die grobe Struktur des Backends aufgezeigt. Hierbei wird die direkte Verbindung zur Datenbank (Kasten *DatabaseAccess*) deutlich. Zudem wird eine Übersicht über die Schnittstelle für die mobilen Applikationen gegeben (gestrichelte Kästen). Den mobilen Appilkationen stehen Funktionen zur Verwaltung von Krankheitstests und Teststreifenschemata zur Verfügung. So kann das berechnete Ergebnis zu einem Krankheitstest an den Server geschickt und die Diagnose

des Mediziners abgefragt werden. Außerdem ist es möglich, Informationen zu den Teststreifenschemata zu erhalten, welche aktuell im Umlauf sind. Die Informationen über die Teststreifenschemata ermöglichen erst die automatische Auswertung auf den mobilen Endgeräten.

Detaillierte Beschreibung der REST-API ist in der Arbeitspaketdokumentation „Fertigstellung des Backends “aus Sprint 9 (Kapitel 4.8.5 auf Seite 391) festgehalten.

## 5.3 Teststreifengenerator

Der Teststreifengenerator kann genutzt werden, um neue Teststreifenvarianten im Vorfeld zu gestalten und zu visualisieren. Im Zusammenhang mit der Weboberfläche können neue Teststreifenvarianten dann auch auf den mobilen Endgeräten verfügbar gemacht werden. Teststreifenvarianten sind Varianten von Schnelltests, welche dafür verwendet werden bestimmte Krankheiten bei einem Patienten nachzuweisen. Nachfolgend ist der Aufbau eines Teststreifens (Abbildung 304) aufgezeigt und erklärt.

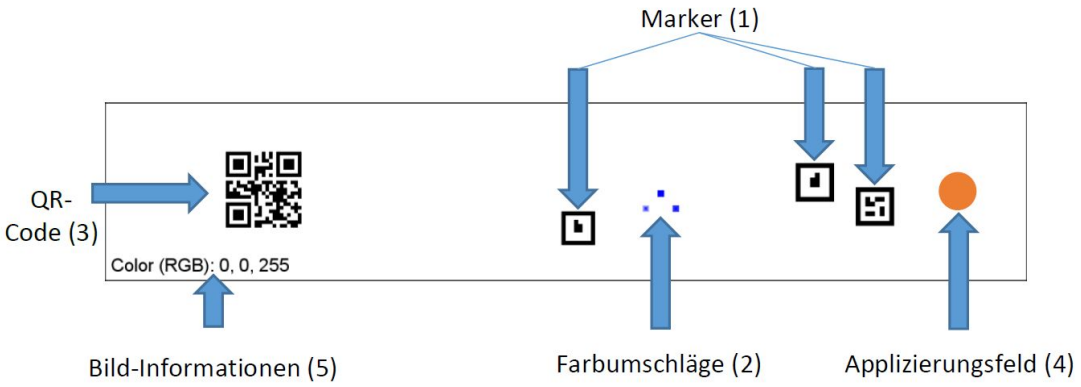



Abbildung 304: Aufbau des Menü-Reiters „Applizierungsfeld“

1. **Die Marker**
   Sie werden zur Erkennung des Teststreifens innerhalb des Bildverarbeitungsalgorithmus benötigt.

2. **Die Farbumschläge**
   Die Farbumschläge sind die Testzonen innerhalb eines Teststreifens. In ihnen werden sich die Antikörper mit der applizierten Flüssigkeit oder mit anderen transportierten Anti-Körpern, welche bereits Kontakt mit der Applizierungsflüssigkeit hatten, in Abhängigkeit des Krankheitsstatus an Farbstoff-Komplexe binden.

3. **Der QR-Code**
   Dieser Code dient zur Hinterlegung von Informationen wie der Teststreifenschema-ID und der Postleitzahl für einen Bereich.

4. **Das Applizierungsfeld**
   Auf diesem Feld appliziert der Patient seine Flüssigkeit. Innerhalb dieses Projektes wurde sich dabei auf die Flüssigkeiten Blut und Urin festgelegt.

5. **Bild-Informationen**
   Allgemeine Informationen zum Bild selbst.

## 5.4 Bildverarbeitung

Die Bildverarbeitung stellt ein zentrales Element des Projektes dar und wurde während des gesamten Projekts stetig weiterentwickelt. Mit dem entwickelten Bildverarbeitungsalgorithmus werden die Teststreifen ausgewertet, sodass der Benutzer in der App eine Rückmeldung erhält, ob er krank oder gesund ist. Die Bildverarbeitung wurde dabei plattformunabhängig in C++ implementiert und in beide Apps eingebunden. Das hat den Vorteil, dass bei Änderungen des Bildverarbeitungsalgorithmus, die Apps nur an einer Stelle angepasst werden müssen und kein doppelter Quellcode entsteht. Für die Bildverarbeitung wird OpenCV als Framework genutzt, sodass die verwendeten Algorithmen, wie z.B. der Canny-Algorithmus, nicht neu implementiert werden mussten.

Für die Bildverarbeitung wurden eigene, spezielle Marker erstellt. Diese sind in der Abbildung 305, welche einen Teststreifen darstellt, zu erkennen. Der Algorithmus kann diese Marker erkennen und

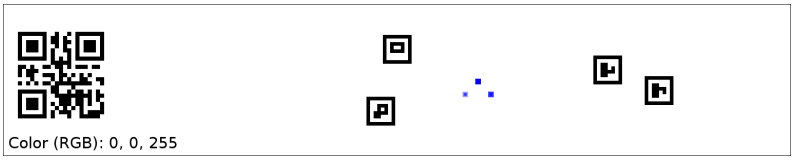



Abbildung 305: Beispiel für einen Teststreifen

anhand ihrer Position berechnen, wo sich die Referenzfelder und Farbumschlagsfelder auf dem Bild befinden. Dafür werden die zu dem Teststreifen zugehörigen Parameter mit an den Algorithmus übergeben.

Im Laufe der Entwicklung wurden verschiedene Phasen durchlaufen. Zunächst wurde der Algorithmus so implementiert, dass er statische Teststreifen erkennen konnte. Dies sollte über einen QR-Code realisiert werden. Es stellte sich allerdings heraus, dass dies kein optimales Vorgehen war, sodass die eigenen Marker erstellt wurden. Anhand dieser wurde ein Modell entwickelt mit dem sich die Position der Referenzfelder und Farbumschläge berechnen ließ. Da dieses Modell Fehler aufwies, wurde nach weiteren Recherchen festgestellt, dass die Berechnung am Besten nach einer Transformation des aufgenommenen Bildes durchgeführt werden kann. Durch das Übergeben von verschiedenen Parametern wurde es ermöglicht, dass der Algorithmus dynamische Teststreifen erkennen kann. Nachdem es möglich war die Positionen der Umschlagsfelder zu berechnen, musste ein Farbabgleich zwischen den Feldern realisiert werden. Hier wurde zunächst der RGB-Farbraum gewählt. Allerdings ergaben Recherchen, dass sich der HSV-Farbraum besser für den Farbabgleich zwischen den Feldern eignet. Gegen Ende des Projekts wurde eine Fehlerbehandlung implementiert, damit zuverlässige Ergebnisse von dem Algorithmus zurück geliefert werden können. Diese prüft das Bild auf verschiedene Merkmale, wie z.B. Überbelichtung und Verzerrung, sodass ein spezifischer Fehlercode zurückgegeben werden kann.

Die eigentliche Auswertung des Bildes beginnt, sobald mindestens drei Marker auf dem Bild gefunden wurden. Die einzelnen Schritte dabei sind in Abbildung 306 auf der nächsten Seite aufgezeigt. Tritt bei der Auswertung einer der beschriebenen Fehler auf, dann wird der Algorithmus unterbrochen und liefert einen spezifischen Fehlercode an die Apps zurück. Dabei ist zu beachten, dass die Fehler nur auftreten, wenn das Bild von einer so schlechten Qualität ist, dass der Farbumschlag oder die Referenzfelder so verfälscht sind, dass kein verlässliches Ergebnis mehr geliefert werden kann.

Der Bildverarbeitungsalgorithmus wird als Quellcode und integriert in der Android- und iOS-App ausgeliefert. Von dort aus wird er aufgerufen und liefert ein entsprechendes Testergebnis an die jeweilige App zurück.

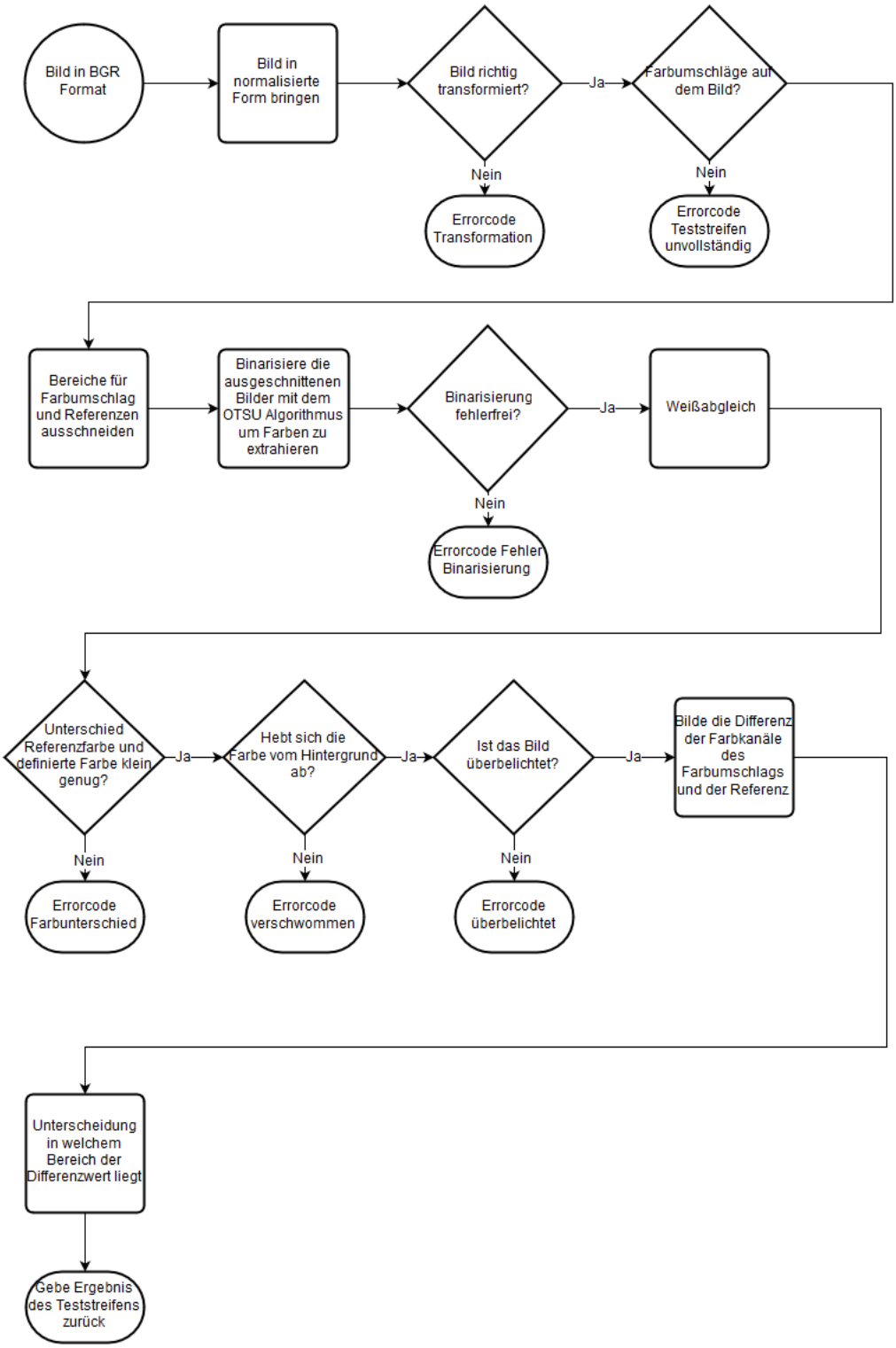



Abbildung 306: Ablauf der Bildverarbeitung

## 5.5 Mobile Applikationen

Die mobilen iOS und Android Applikationen sind die Endanwendungen, mit welchen die Probanden einen Test durchführen. Der Test wird auf dem Smartphone analysiert, an den Server gesendet und empfängt bei einem unklaren Ergebnis eine Diagnose von einem Mediziner. Die Apps werden nicht über die digitalen Marktplätze (App Store und Google Play Store) bereitgestellt, sondern müssen über die jeweiligen Entwicklungsumgebungen auf die Endgeräte übertragen werden. Eine Anleitung zum genauen Vorgehen kann den Handbüchern der Android (siehe E) und iOS App (siehe D) entnommen werden. Die folgenden Abbildungen 307 und 308 zeigen die Grundstruktur der beiden Endanwendungen. Diese untergliedert sich in drei unterschiedliche Ansichten:

**Neuer Test** Die Ansicht „Neuer Test“ führt den Nutzer durch den Testablauf und zeigt ihm in einzelnen Schritten das genaue Vorgehen.

**Ergebnisse** Die Ansicht „Ergebnisse“ zeigt dem Nutzer die Ergebnisse aller durchgeführten Tests in einer Liste. Bei der Auswahl eines spezifischen Tests werden weitere Informationen angezeigt.

**Information** Die Ansicht „Information“ zeigt dem Nutzer allgemeine Informationen zur mobilen Applikation. Darunter fallen zum Beispiel die verwendeten Bibliotheken, die Webseite der Abteilung **AMiR** und ein Hilfe-Video zur App.


Abbildung 307: Grundstruktur der Android App

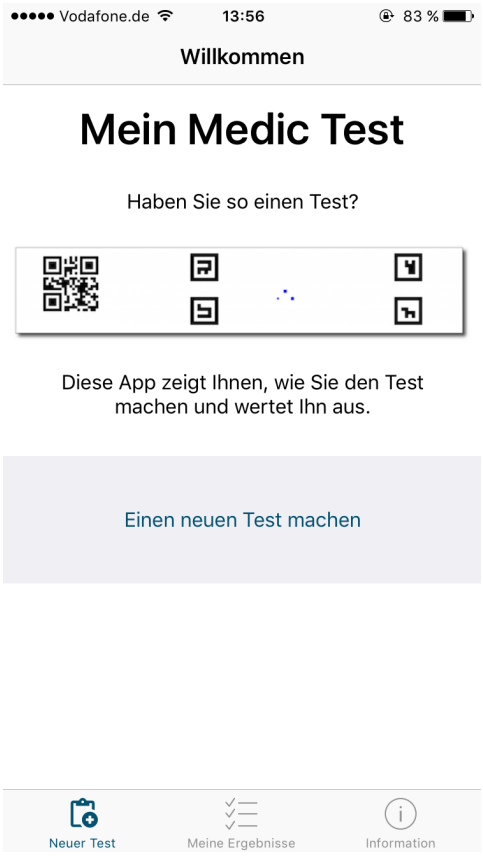



Abbildung 308: Grundstruktur der iOS App

# 6 Evaluation

Dieses Kapitel befasst sich mit der abschließenden Evaluation der Anforderungen, die im Lastenheft 3.4 auf Seite 35 definiert wurden. Sie wurde unabhängig von den Evaluationen der Sprintdokumentation durchgeführt, sodass die Anforderung im folgenden nochmals überprüft werden. Dazu wurden die Szenarien „Proband führt Test durch “ , „Kommunikation Mediziner - Proband“ , „Auswertung durch den Mediziner“ und „Institut verwaltet Infrastruktur“ aus dem Lastenheft analysiert und die erfüllten funktionalen Anforderungen genannt. In der Analyse der Szenarien werden die Zusatzleistungen, die über die funktionalen Anforderungen hinaus gehen, explizit hervorgehoben. Die nichtfunktionalen Anforderungen werden anschließend betrachtet, zu ihren Schwerpunkten zusammengefasst und bewertet. Außerdem wird der Bildverarbeitungsalgorithmus, der zur Auswertung der Teststreifen benutzt wird, gesondert betrachtet, da er in den definierten Anforderungen kaum Erwähnung fand und zusätzliche Bewertungsmittel geschaffen wurden.

## 6.1 Funktionale Anforderungen

Im Folgenden werden zunächst die funktionalen Anforderungen betrachtet, die in dem Projekt umgesetzt wurden.

### 6.1.1 Szenario: Proband führt Test durch

Die Anforderungen, dass ein Proband einen Test mit einer mobilen Applikation durchführen kann, konnten für iOS und Android vollständig umgesetzt werden. Die geographische Position wird durch den Ortungsdienst des Smartphones bestimmt (FA-2). Falls dieser nicht zur Verfügung steht, kann eine Positionierung über eine Postleitzahl, die im QR-Code gespeichert ist, vorgenommen werden (FA-1). Eine spätere Übertragung des Testergebnisses und des Fotos (FA-10) an den Server (FA-13) ist auch im Fall des Verbindungsabbruchs, durch die Speicherung in eine lokale Datenbank auf dem Smartphone, gewährleistet (FA-14). Es konnte evaluiert werden, dass auch wenn das Smartphone neu gestartet wird, eine Übertragung stattfindet, sobald eine aktive Verbindung besteht (FA-3). Der Test kann nicht gestartet werden, wenn der Ladestand des Akkus weniger als 10% beträgt (FA-5). Dies wird dem Probanden mitgeteilt (FA-6). Der Proband hat die Möglichkeit einen Test eigenständig zu starten (FA-9) und bis zur Datenübermittlung zu beenden (FA-7). Dabei wurde auch darauf geachtet, den Probanden durch einfache Sprache durch den Test zu führen, um so die Barrierefreiheit zu gewährleisten. Eine genauere Betrachtung ist in Kapitel 6.2.2 auf Seite 444 nachzulesen. Die Testergebnisse, die in der Datenbank des Smartphones gespeichert sind, können durch den Probanden über die mobile Applikation gelöscht werden (FA-8). Die Zwischenspeicherung des Fotos erfolgt in der Datenbank auf dem Smartphone (FA-11). Wenn kein Speicherplatz verfügbar ist, teilt die mobile Applikation dies dem Probanden mit (FA-12).

Bei der Umsetzung der mobilen Applikationen wurden einige zusätzliche Funktionen implementiert. Diese sind im folgenden Abschnitt beschrieben.
Die mobilen Applikationen sind nativ in iOS und Android umgesetzt worden, was einen deutlichen Mehraufwand bei der Entwicklung darstellte. Das hatte den Vorteil, dass eine intuitive Nutzung der mobilen Applikationen für die Benutzergruppen (Android und iOS) möglich ist, weil einige Bedienelemente bei den Systemen unterschiedlich sind. Des Weiteren haben die nativen Apps einen besseren Umgang mit den Systemressourcen und dadurch eine höhere Performanz. Dadurch konnte ein qualitativ hochwertigeres Produkt entwickelt werden.
Die Benachrichtigung, wenn ein Arzt eine Diagnose stellt oder ein neues Teststreifenschema zur

Verfügung steht, wird mit Pushtechnologien[88] realisiert. Pulltechnologien[89] verbrauchen durch regelmäßige Anfragen an den Server die Akkulaufzeit und das Datenvolumen, welches bei den meisten Smartphones beschränkt ist. Durch die Pushtechnologien kann dieser Verbrauch reduziert werden, weil das Smartphone nur dann Anfragen an den Server stellt, wenn der Server es vorher mitteilt.

### 6.1.2 Szenario: Kommunikation Mediziner - Proband

Die Kommunikation zwischen dem Mediziner und dem Probanden wird durch einen Server gewährleistet, der in Go implementiert wurde, wodurch Diagnosen vom Mediziner gesendet und vom Probanden empfangen werden können (FA-16). Die mobile Applikation gibt dem Probanden Rückmeldung über eine Diagnose, indem eine Benachrichtigung auf dem Smartphone erscheint, sobald diese empfangen wurde (FA-17) (Siehe Abb. 309).


Abbildung 309: Pushbenachrichtigung für eine erstellte Diagnose.

Durch eine eindeutige Identifikationsnummer auf dem Server kann ein Test einem Smartphone zugeordnet werden. Ein Proband ist dabei der Besitzer eines Smartphones. Eine Push-Benachrichtigung wird über die Identifikationsnummer an das entsprechende Smartphone gesendet (FA-19). Eine lokale Zwischenspeicherung der Diagnose bei einem Verbindungsabbruch zum Webserver ist nicht möglich, da das Betriebssystem und der Browser dieses verhindern . Ein nachträgliches Senden einer Diagnose kann dadurch nicht erfolgen. Die Anforderung FA-18 kann daher nicht umgesetzt werden. Für dieses Szenario sind alle Anforderungen, die realisierbar waren, umgesetzt worden.

[88] `http://www.it-administrator.de/lexikon/push_technologie.html`
[89] `https://en.wikipedia.org/wiki/Pull_technology`

### 6.1.3 Szenario: Auswertung durch den Mediziner

Dieses Szenario umfasst die Anforderungen, die für die Weboberfläche des Mediziners aufgestellt wurden.
Dies schließt ein, dass ein Mediziner die Möglichkeit hat ein Benutzerkonto anzulegen. Dazu muss eine gültige E-Mail-Adresse mit einem Passwort angegeben werden (FA-21), damit der Mediziner eine Rückmeldung in Form einer E-Mail für die Erstellung seines Benutzerkontos erhalten kann (FA-33). Das Passwort muss mindestens acht Zeichen lang sein, um eine ausreichende Sicherung zu gewährleisten. Bei der Erstellung eines Benutzerkontos sollten weitere Daten vom Mediziner mit angegeben werden. Diese zusätzlichen Informationen sind unter anderem die Adresse seiner Praxis, wie dieser sich spezialisiert und welchen Titel er hat. Alle Eingaben, die ein Mediziner oder ein Administrator machen kann, werden auf Syntax und Semantik geprüft (FA-32), wie etwa, dass die E-Mail eine gültige Form hat, dass in einigen Feldern wie die Postleitzahl und die Telefonnummer nur Zahlen eingegeben werden können und dass das Passwort mindestens aus acht Zeichen besteht. Bei einer ungültigen Eingabe gibt das System eine entsprechende, aussagekräftige Rückmeldung (FA-33). Bei der Rückmeldung wird unterschieden, ob eine falsche oder keine Eingabe getätigt wurde und welches Eingabefeld zu bearbeiten ist. Ein Administrator muss ein Benutzerkonto bestätigen, bevor dieser Zugriff auf die Funktionen der Weboberfläche erhält (FA-29). Sobald das Benutzerkonto bestätigt wurde, kann der Nutzer sich anmelden. Wenn ein Nutzer angemeldet ist, kann der Nutzer sich wieder abmelden oder durch das schließen des Browsers im System abgemeldet werden (FA-22). Durch eine Authentizitätsprüfung auf Basis der Rolle des Benutzerkontos, hat jeder Nutzer nur die Funktionen, die für die jeweilige Rolle freigeschaltet sind (FA-25). Wenn ein Nutzer versucht sich unbefugten Zugang zu einer anderen Oberfläche (z.B. der Administrator Oberfläche) zu verschaffen, wird der Nutzer zurück auf die für ihn erreichbare Seite geleitet.

Einem Mediziner werden alle bisher erfassten Testergebnisse in einer Tabelle angezeigt. Ein Testergebnis beinhaltet eine eigene Identifikationsnummer, das Datum an dem der Test durchgeführt und den Namen der Krankheit auf die getestet wurde, sowie das Ergebnis der Voranalyse (positiv, negativ oder unbestimmt) und ob für das Testergebnis eine Diagnose vorliegt (FA-27). In der Tabelle kann ein Testergebnis ausgewählt werden, wodurch sich eine Ansicht öffnet. Die Ansicht zeigt das aufgenommene Bild, das während der Durchführung des Tests vom Probanden erstellt wurde. In der Ansicht ist ebenfalls die Position in einer Karte angegeben, an der der Test durchgeführt wurde, wobei die Karte optional deaktiviert werden kann. Für ein Testergebnis, für das die Voranalyse kein Ergebnis ermitteln konnte (unbestimmt), kann der Mediziner eine Diagnose stellen (FA-23). Innerhalb dieser Diagnose kann er den Test als positiv oder negativ bewerten(FA-26), mit dem Ergebnis der Voranalyse vergleichen (FA-28).

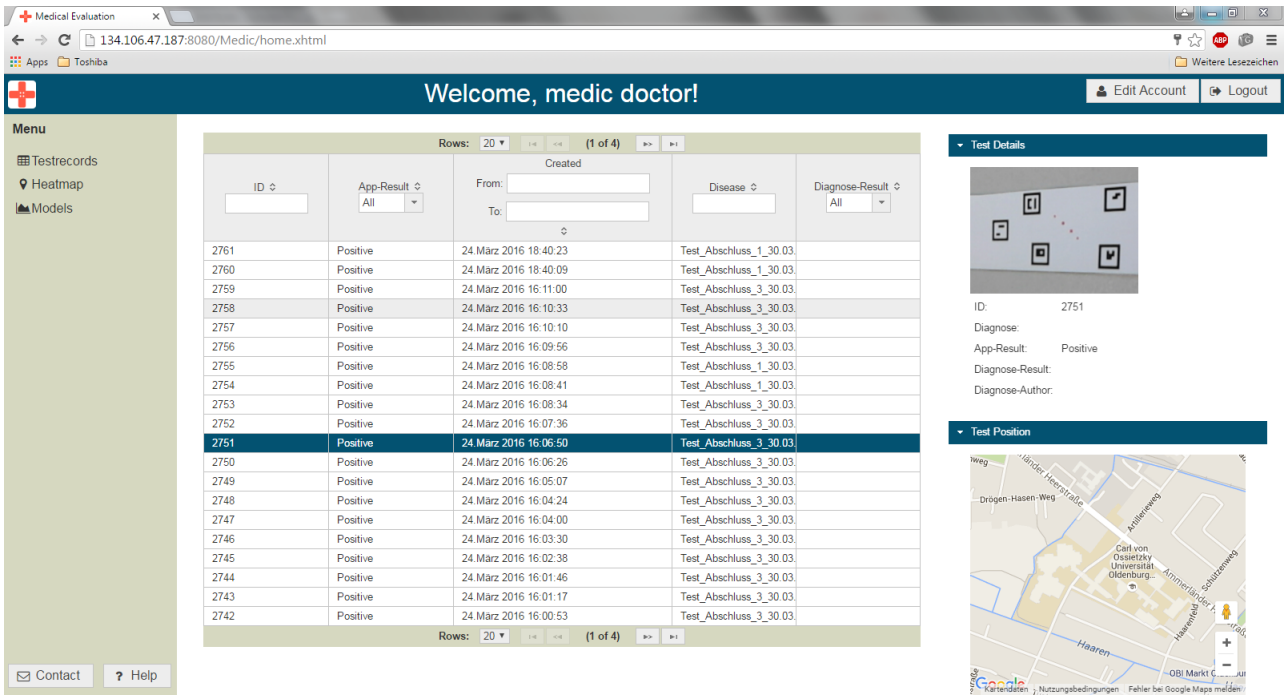



Abbildung 310: Die Hauptansicht der Mediziner auf der Weboberfläche

Zusätzlich sei herausgestellt, dass der Mediziner die Möglichkeit hat, eine detaillierte Diagnose zu diesem Testergebnis zu schreiben, die ebenfalls an das Smartphone des Probanden gesendet wird. Des Weiteren hat ein Mediziner die Möglichkeit, in der Tabelle die Testergebnisse nach den einzelnen Werten zu filtern und zu sortieren. Die beschriebene Ansicht ist in Abbildung 310 dargestellt.

In der Weboberfläche des Mediziners gibt es die Ansicht der interaktiven Karte von Deutschland, in der Testergebnisse für ausgewählte Krankheiten mit ihren Positionen, dem Ergebnis der Voranalyse dargestellt werden und die mit einer Legende beschrieben wurde. Der Mediziner hat die Möglichkeit diese auf Basis ihres Ergebnisses zu filtern, sodass zum Einen nur die positiven, negativen oder unbestimmten Resultate angezeigt werden können, zum Anderen alle Kombinationen dieser möglich sind. Die gefilterten Ergebnisse werden dann als kumulierte Häufigkeiten in einzelnen Gebieten zusammengefasst dargestellt (vgl Abbildung 311 auf der nächsten Seite).

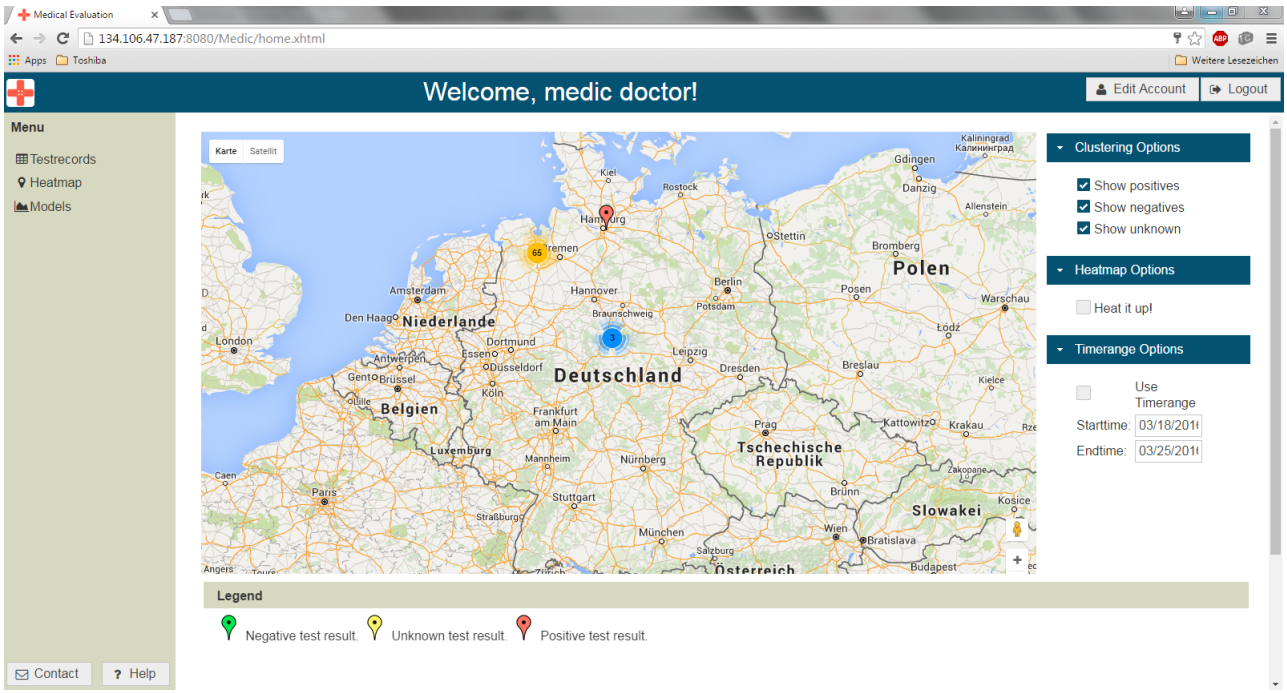



Abbildung 311: Beispiel für die kumulierten Häufigkeiten auf der Heatmap

Des Weiteren gibt es die Möglichkeit eines Heatmapdiagramms, welches die positiven Testergebnisse zusammenfasst (FA-20).

In der Oberfläche für Statistiker können statistische Auswertungen auf Basis von gewählten Daten durchgeführt werden, indem R-Skripte im webbasierten Entwicklungsbereich entwickelt werden können. R ist eine Programmiersprache, die entwickelt wurde, um statistische Aufgaben und Problemstellungen zu lösen. In R können Funktionen, Berechnungen, Modelle und Graphen zur Auswertung der ausgewählten Daten genutzt werden, um epidemiologische Aussagen zu tätigen. Dafür wurde ein OpenCPU Server bereitgestellt, auf dem die Berechnungen der R-Skripte durchgeführt werden, wodurch keine lokale Installation von R notwendig ist.
Eine genaue Darstellung der Funktionen für die Auswertungsoberfläche des Administrators wird in dem folgenden Szenario 6.1.4 beschrieben.

### 6.1.4 Szenario: Institut verwaltet Infrastruktur

In dem Szenario, dass ein Institut die Infrastrukturen verwaltet, wurden alle relevanten funktionalen Anforderungen für diesen zusammengefasst und erfüllt.
Wie im Abschnitt 6.1.3 auf Seite 438 erwähnt, existiert ein Rechtesystem, welches aus Medizinern, Administratoren und Statistikern besteht (FA-30). Um dieses Szenario abzudecken wurde die Weboberfläche für Administratoren entwickelt (Abbildung 312 auf der nächsten Seite).

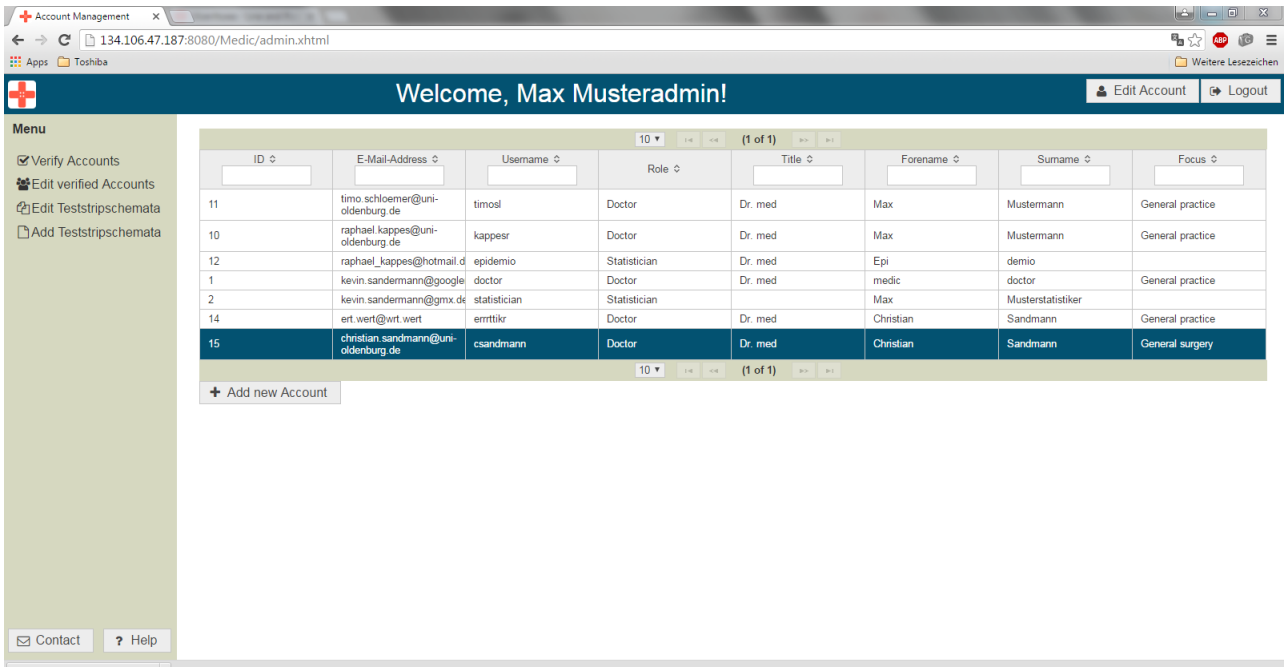



Abbildung 312: Liste der noch ausstehenden Mediziner auf der Oberfläche der Administratoren

Diese ermöglicht Administratoren das Bearbeiten und Löschen vorhandener und das Anlegen, Bestätigen oder Ablehnen neuer Benutzerkonten (FA-36). Benutzerkonten werden in bestätigte und unbestätigte unterteilt und so auch in getrennten Tabellen in einer zentralen Datenbank gespeichert (FA-31). In der Weboberfläche eines Administrators werden diese mit verschiedenen Ansichten angezeigt, so dass ein Administrator einsehen kann welche Benutzerkonten bereits bestätigt wurden und welche nicht. Jedes Benutzerkonto beinhaltet eine E-Mail-Adresse, einen Benutzernamen, einen oder keinen Titel sowie dem Vor- und Nachnamen des Nutzer. Für Mediziner muss darüber hinaus die Praxisadresse, die Adresse der Ärztekammer und die der Aufsichtsbehörde angegeben werden (FA-29). Dabei wird überprüft, ob die Eingaben beim Anlegen eines Benutzerkontos bereits in der Datenbank vorhanden sind. Wenn die E-Mail-Adresse oder der Benutzername bereits vorhanden sind, kann das Benutzerkonto nicht angelegt werden. Beim Anlegen eines Benutzerkontos für einen Mediziner wird zusätzlich überprüft, ob die Telefonnummer der Praxisadresse bereits vorhanden ist, so dass das Benutzerkonto, wenn die Telefonnummer bereits existiert, nicht angelegt wird (FA-32). Wenn während der Verwaltung der bestätigten und unbestätigten Benutzerkonten Fehler auftreten, erhält der Administrator eine Benachrichtigung über die Ursache des Fehlers und die Änderung wird nicht übernommen (FA-33).

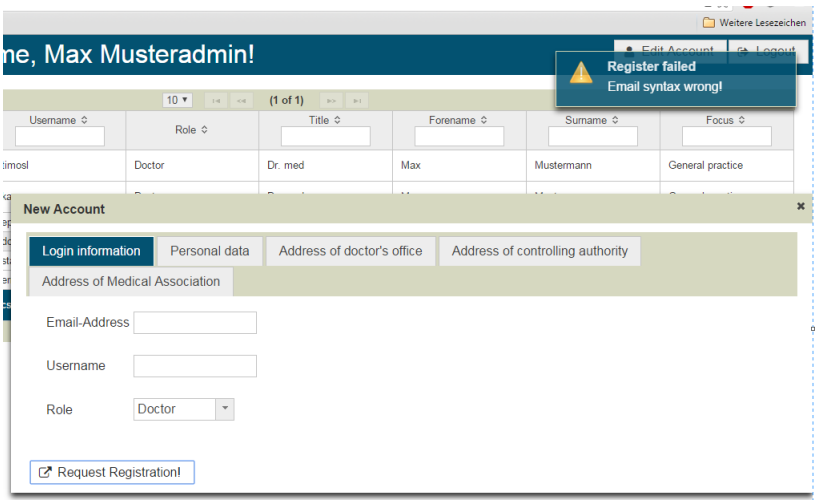



Abbildung 313: Darstellung von Fehlermeldungen in der Weboberfläche.

Nachdem ein Benutzerkonto angelegt wurde, wird der Nutzer über die angegebene E-Mail-Adresse informiert (FA-34). Schließlich können die Tabellen in der Weboberfläche für die bestätigten und unbestätigten Benutzerkonten nach ihren Werten gefiltert und sortiert werden (FA-35).

Eine weitere Ansicht der Weboberfläche eines Administrators ist das Hinzufügen neuer Teststreifenvarianten.

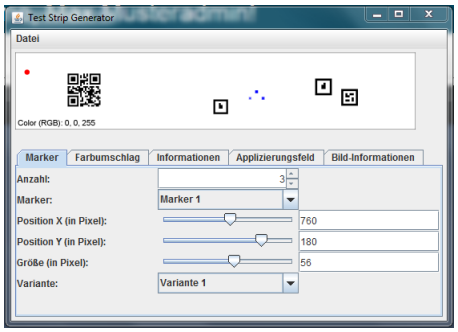



Abbildung 314: Der Teststreifengenerator zur Erstellung von neuen Teststreifenvarianten

Dazu wird der Teststreifengenerator (Abbildung 314) benutzt, mit dem ein neuer Teststreifen erstellt und über die Ansicht in eine zentrale Datenbank geladen werden kann. Der Teststreifengenerator kann direkt von der Weboberfläche des Administrators heruntergeladen werden. Mit dem Teststreifengenerator lassen sich beliebige Teststreifen erstellen. Die Anzahl und Positionen der Farbumschläge, Referenzfelder und Marker, welche für die Erkennung erforderlich sind, sind flexibel einstellbar. Dabei wird überprüft, ob alle nötigen Anforderungen gegeben sind, also dass immer mindestens drei Marker vorhanden sind (FA-39). Nachdem der Teststreifen hochgeladen wurde, besitzt er eine Identifikationsnummer und wird als Eintrag in der Tabelle auf der Oberfläche angezeigt (FA-37). In der Tabelle kann der Eintrag ausgewählt werden, wodurch ein Dialog geöffnet in dem der Teststreifen bearbeitet werden kann (FA-40). Die Tabelle für die Teststreifenvarianten in der Weboberfläche lassen sich über die Identifikationsnummer, den Krankheitsnamen, das Erstellungsdatum und der Aktivität filtern und sortieren (FA-41).

Jeder Nutzer hat die Möglichkeit auf einer gesonderten Oberfläche seine Daten zu verändern, die auf Syntax und Semantik überprüft werden (FA-42).

## 6.2 Nichtfunktionale Anforderungen

Im folgenden Kapitel wird beschrieben, welche nichtfunktionalen Anforderungen, die an das Projekt gestellt wurden, umgesetzt und evaluiert wurden.

### 6.2.1 Zuverlässigkeit

Die Verfügbarkeit des Servers wird durch die Anbindung des Instituts an das Internet, Stromausfälle und technische Probleme eingeschränkt. Diese Einschränkungen können nicht durch die Umsetzung des Servers beeinflusst werden. Der Go-Server, die Datenbank, der Primefaces-Server und der OpenCPU Server werden in den Betriebssystemen Linux Debian und Ubuntu 14.04 betrieben und hängen von deren Stabilität ab (NFA-21). Die Verfügbarkeit von 97,5% kann von daher nur durch Langzeittests an dem entsprechenden Standort des Instituts geprüft werden (NF-1).
Um die Zuverlässigkeit des Servers zu garantieren, sodass dieser 10000 gleichzeitigen Übermittlungen pro Sekunde und 5000 gleichzeitig angemeldeten Nutzern standhalten kann (NFA-3, NFA-4), wurde ein sogenannter „Load-Balancer“ implementiert und evaluiert, der dies umsetzt.

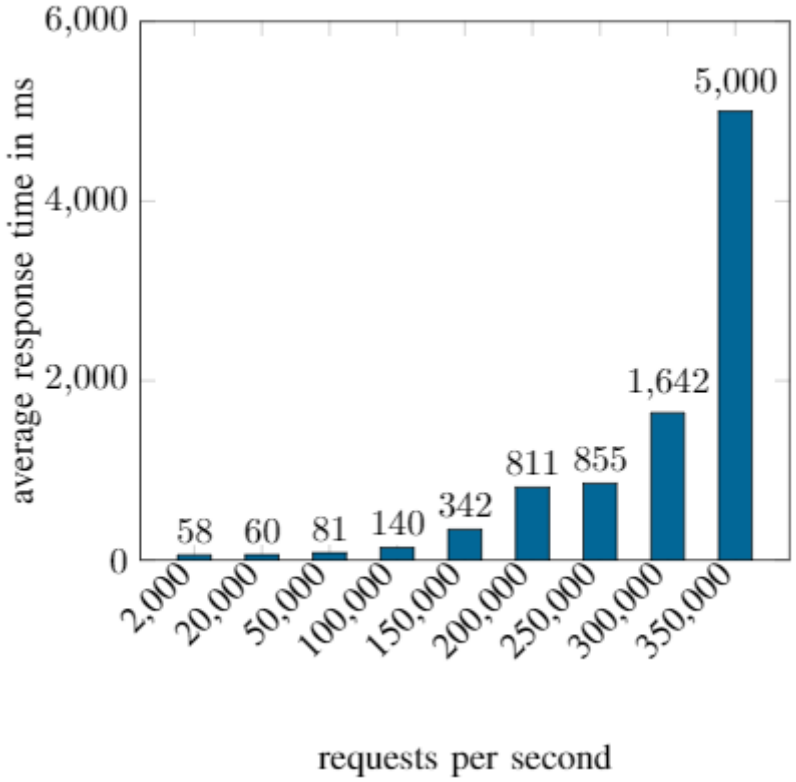



Abbildung 315: Durchschnittliche Antwortzeit des Servers in Sekunden mit dem Load-Balancer

Abbildung 315 zeigt das Ergebnis der Auswertung. Der Server antwortet bei 250000 gleichzeitigen Anfragen in unter einer Sekunde, wenn über 300000 gleichzeitige Anfragen kommen, kann der Server standhalten.

Die mobile Applikationen wurde von insgesamt 21 Personen bestehend aus Entwicklern und Außenstehenden ausgiebig getestet. Die Resultate dieser Studie sind in Abb. 316 auf der nächsten Seite zu sehen.

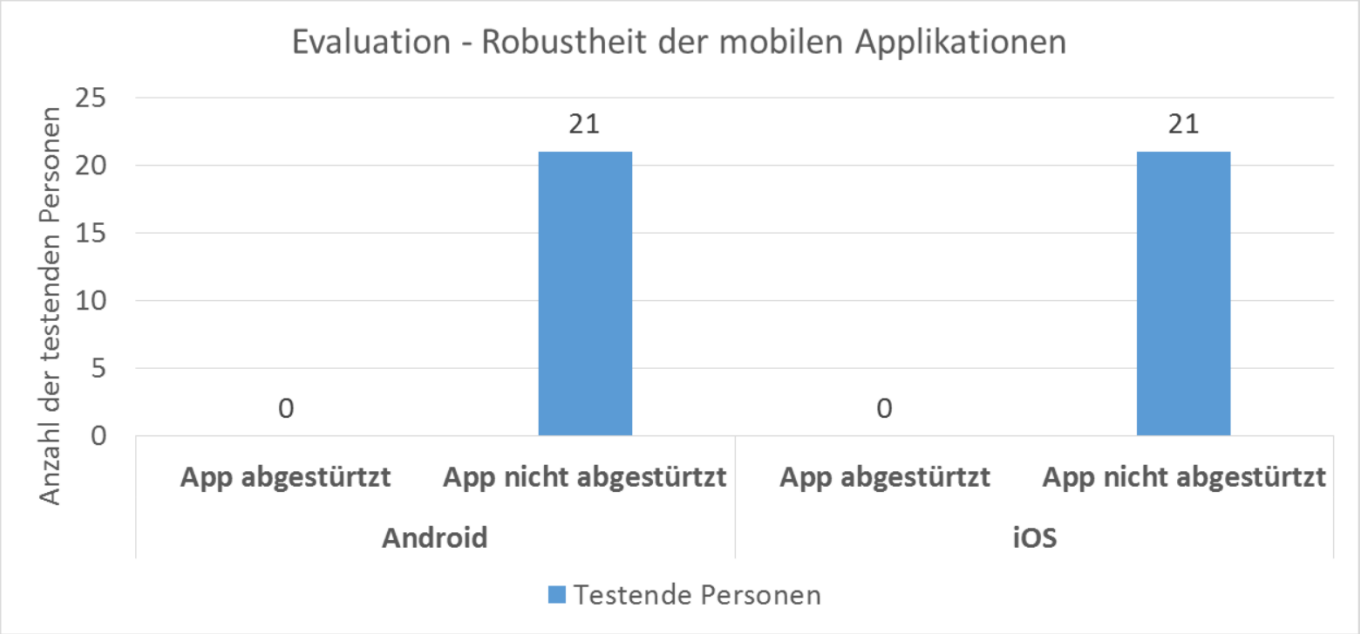



Abbildung 316: Studie zur absturzfreien Nutzung der Applikationen.

Dabei ist es nicht vorgekommen, dass die mobilen Applikationen, sowohl für Android als auch für iOS, abgestürzt sind (NF-2).

### 6.2.2 Usability und Look and Feel

Um die Usability zu gewährleisten wurden Feldstudien und Umfragen durchgeführt und diese anschließend mit dem System Usability Scale-Test (SUS-Test) bewertet. Ein gutes Produkt sollte nach Dr. Bangor einen Wert von 71.4 Punkten aufweisen, damit die Usabilty mit dem Adjektiv „gut“ bezeichnet werden kann [30].

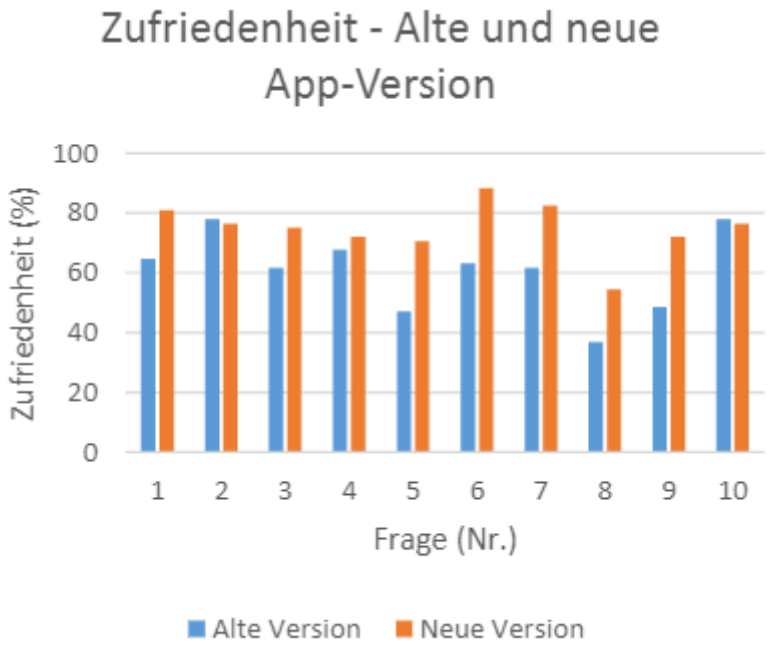



Abbildung 317: Vergleich der Zufriedenheit der alten Version der Android-App mit der neuen Version

Das Ergebnis des SUS-Test für die Android-App ist in Abbildung 317 dargestellt. Diese konnte

einen Wert von 75 Punkten erlangen und wird damit als „gut“ eingestuft. Die iOS-App ist vom Aufbau mit der Android-App vergleichbar, sodass sich der Wert des SUS-Test nicht unterscheidet. Die Weboberfläche konnte bei dem SUS-Test einen Mittelwert von 77.9 Punkten erlangen. Mit Feldstudien könnte gezeigt werden, dass die Usability sowohl für die mobilen Applikationen, als auch für die Weboberfläche gewährleistet ist (NFA-8).

Neben der Usability wurde eine umfassende Arbeit über die Barrierefreiheit erstellt und dazu eine Studie durchgeführt. Insbesondere wurde der Aspekt der einfachen Sprache analysiert und die Ergebnisse in die Applikationen integriert (NFA-6, NFA-7). Zusätzlich wurden Anweisungstexte für blinde Menschen entworfen, sodass diese die Möglichkeit haben, die mobilen Applikationen zu nutzen. Dadurch weisen die mobilen Applikationen des Projekts MEDIC im Vergleich zu bereits vorhandenen Applikationen eine bessere Umsetzung der Barrierefreiheit auf.

### 6.2.3 Leistung und Effizienz

Damit die Datenhaltung des Servers möglichst persistent und performant ist, wurde evaluiert, dass eine Implementierung des Servers in der Programmiersprache Go für diese Anforderung am Besten geeignet ist.

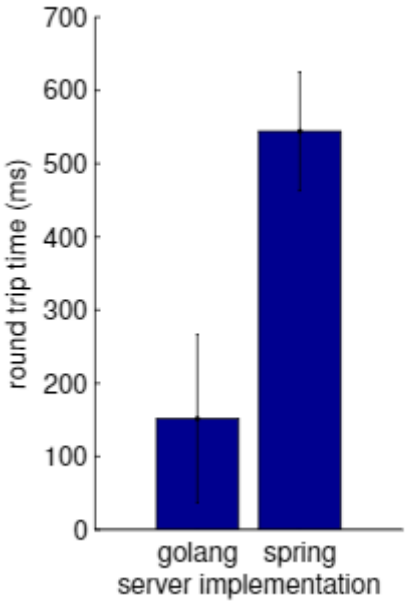



Abbildung 318: Vergleich der Serverperformance mit einer Implementierung in Go und in Spring/Java

Ein Vergleich der Performance des Servers mit einer Implementierung in Go und in Spring/Java ist in Abbildung 318 dargestellt. Die Abbildung zeigt, dass ein in Go implementierter Server deutlich performanter ist als eine Implementierung in Spring/Java.
Für die Kommunikation zwischen der Weboberfläche/Primefaces und der Datenbank wird Hibernate benutzt, was ebenfalls zu einer persistenten Datenhaltung führt (NFA-9).

Mit dem Testframework konnte gezeigt werden, dass die Voranalyse des Bildverarbeitungsalgorithmus niemals länger als zehn Sekunden dauert (NFA-11). Der Bildverarbeitungsalgorithmus war eine der Hauptaufgaben des Projektes, daher ist eine genauere Analyse zu diesem in Kapitel 6.3 auf Seite 447.

Die mobile Applikation konnte auch auf preisgünstigen Smartphones wie dem Sony XPeria E1 einwandfrei getestet werden (NFA-13).

### 6.2.4 Wartbarkeit und Änderbarkeit

Im Laufe des Projektes wurden verschiedene Richtlinien erstellt, die klar aufzeigen, wie Kommentare und Dokumentationen auszusehen haben. Die Richtlinien konnten eingehalten werden, in dem sämtlicher erstellter Quellcode durch ein Projektmitglied auf die Qualität hin überprüft wurde, welches nicht an der Entwicklung beteiligt war. Erst nachdem eventuelle Anmerkungen dieses Mitglieds durch einen Entwickler eingearbeitet wurden, wurde der Quellcode angenommen. Diese Kontrollstruktur hatte zur Folge, dass neben der Sicherstellung der Qualität des Quellcodes auch eine ausreichende (NFA-16) Dokumentation vorhanden ist, die auf englisch verfasst wurde (NFA-15). Die Dokumentation des Projekts soll durch das Werkzeug **Doxygen** erstellt werden können. Doxygen ermittelt über die Dateiendungen wie der Quellcode zu übersetzen ist und erstellt daraus HTML-Seiten, die die Zusammenhänge der einzelnen Funktionen und Strukturen zeigen. Weiterhin werden die sprachenspezifischen Dokumentationsarten vollständig in die Seiten übernommen. Das Werkzeug unterstützt die in diesem Projekt benutzten Programmiersprachen Java, Objective-C, Python und C++. Folgenden Produkte können dadurch mit Doxygen dokumentiert werden:

- Weboberfläche in Java
- Teststreifengenerator in Java
- Testdatengenerator in Java
- iOS-Applikation in Objective-C
- Android-Applikation in Java
- Testframework in Python
- Bildverarbeitungsalgorithmus in C++
- Teststand in C++/Java

Für die übrigen Produkte wie den Go-Server der in der Programmiersprache Go geschrieben wurde muss ein anderes Dokumentationswerkzeug genutzt werden. Der Go-Server kann durch das Dokumentationswerkzeug GoDocs dokumentiert werden und erstellt ähnlich wie Doxygen HTML-Seiten, die die Zusammenhänge der einzelnen Funktionen erläutern und die im Quellcode geschriebene Dokumentation übernehmen (NFA-17).

Die mobilen Applikationen sind in deutscher Sprache und haben die Möglichkeit, auch in verschiedene Sprachen überführt zu werden (NFA-14). Dies wird durch die Datenbank erreicht, in der Übersetzungen sämtlicher Texte, die in den mobilen Applikationen angezeigt werden können, abgespeichert sind und können je nach Bedarf angefordert werden können.

### 6.2.5 Portierbarkeit und Übertragbarkeit

Bei der Entwicklung des Systems wurde darauf geachtet, dass verschiedene Browser verwendet wurden, sodass gewährleistet werden kann, dass die im Lastenheft angegebenen Versionen von Google Chrome, Mozilla Firefox, Microsoft Internet Explorer und Apple Safari unterstützt werden (NFA-18). Dies gilt auch für die mobilen Versionen dieser Browser (NFA-19).
Durch die in Kapitel 6.1.1 auf Seite 436 beschriebene native Implementierung in iOS und Android ist das System für beide Betriebssysteme verfügbar, sodass eine Abdeckung von 96% der Smartphones

in Deutschland ermöglicht wird.
Der Go-Server, die Datenbank und der OpenCPU Server werden in den Linux-Betriebssystemen Debian und Ubuntu bereitgestellt (Bereits in Abschnitt 6.2.1 auf Seite 443 erfüllt).

### 6.2.6 Sicherheitsanforderungen

Es ist sichergestellt, dass keine unbefugten Personen auf Funktionen zugreifen können. Dies wird durch die Authtentizitätsprüfung in der Weboberfläche realisiert, wie es in Kapitel 6.1.3 auf Seite 438 beschrieben wurde (NFA-24). Weiterhin sind für die Kommunikationschnittstellen zwischen den mobilen Applikationen und dem Go-Server sowie zwischen Weboberfläche und OpenCPU Server weitere Authentizitätsprüfungen umgesetzt worden (NFA-23). Die Kommunikation zwischen Weboberfläche und den mobilen Applikationen benötigt keine weitere Überprüfung seitens des Projekts, da ein Mediziner an der Stelle nur einen Text eingeben kann und dieser in den mobilen Applikationen nur als Text angezeigt wird. Diese Dienste überprüfen die Berechtigung der Datenübermittlung. Eine direkte Übertragung von den mobilen Applikationen zu der Weboberfläche ist nicht vorhanden. Zusätzlich muss jedes Benutzerkonto zunächst von einem Administrator überprüft werden, bevor er auf die Funktionen der Weboberfläche zugreifen kann. Dadurch wurde sichergestellt, dass keine unbefugte Personen auf Daten von Probanden zugreifen können (NFA-22).

### 6.2.7 Korrektheit

In der Weboberfläche werden Eingaben von Nutzern wie bei der Anmeldung oder bei Änderungen auf Syntax- und Semantikfehler überprüft (Siehe Abschnitt 6.1.3 auf Seite 438). Die Daten, die vom Nutzer entgegengenommen werden, werden auf Syntax- und Semantikfehler überprüft (NFA-26). Die mobilen Applikationen benötigen keine Eingaben von Probanden, sodass keine direkte Prüfung stattfinden kann. Eine indirekte Prüfung findet durch die Nutzung der Kommunikationschnittstelle mit dem Go-Server statt, die nur ein vollständiges Testergebnis überträgt. Das zu übertragende Foto, das durch die Testdurchführung entsteht, wird vom Bildverarbeitungsalgorithmus analysiert und bei einer erfolgreichen Auswertung versendet (Siehe Abschnitt 4.8.3 auf Seite 354) (NFA-25).

## 6.3 Bildverarbeitungsalgorithmus - Sonderleistungen

Im folgenden Kapitel wird der Bildverarbeitungsalgorithmus bewertet, der eine der Kernaufgaben des Projektes darstellte und zu dem es nur eine Anforderung im Lastenheft gab.
Um eine qualitative Bewertung der Bildverarbeitung vornehmen zu können, wurden im Laufe des Projektes einige Hilfsmittel entwickelt. Ein Hilfsmittel ist der Teststand (Kapitel 4.4.14.1 auf Seite 192), welcher entwickelt wurde, um reproduzierbare Messwerte zu erzeugen und verschiedene Versionen des Bildverarbeitungsalgorithmus gegeneinander zu bewerten.

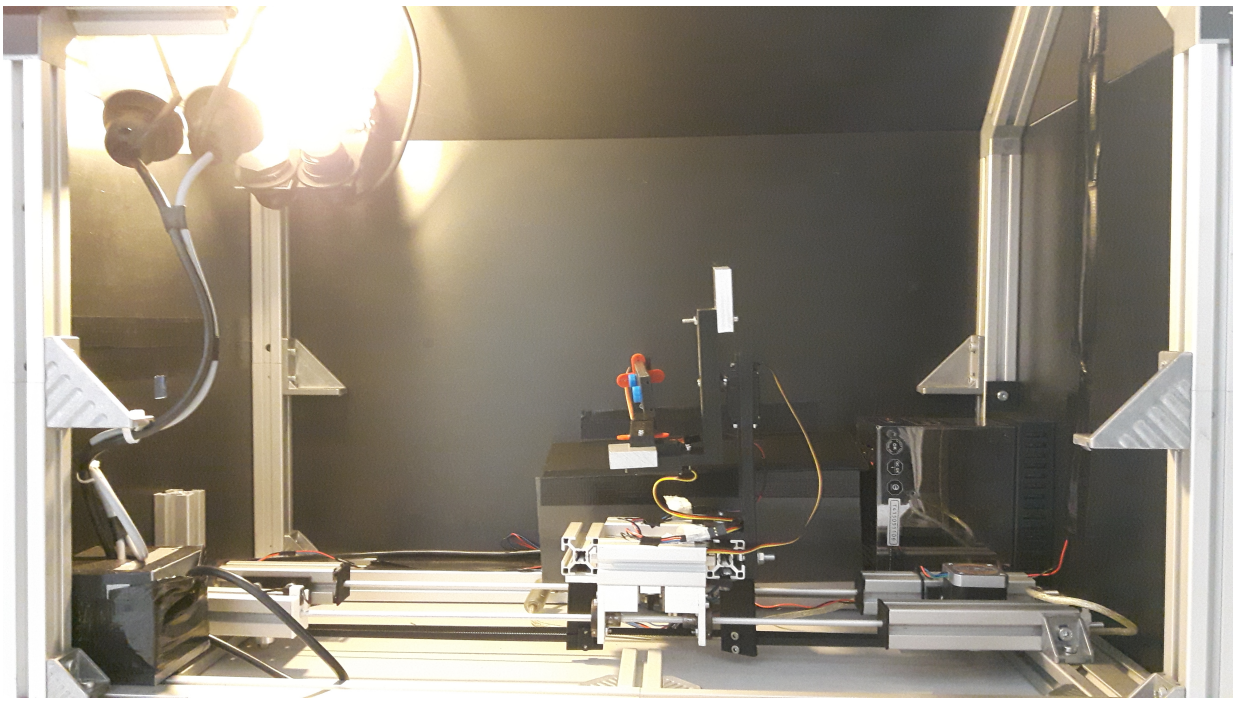


Abbildung 319: Der entwickelte Teststand von innen.

In diesem Teststand kann ein Teststreifen eingespannt und dieser vor der Kamera unter kontrollierten Lichtbedingungen bewegt werden. Ein Teststreifen kann in vier Freiheitsgraden bewegt werden, sodass Teststreifen in x-, y-, und z-Achse rotiert und diese in y-Richtung zwischen 20cm und 50cm in bewegt werden können (Siehe Abb. 320).

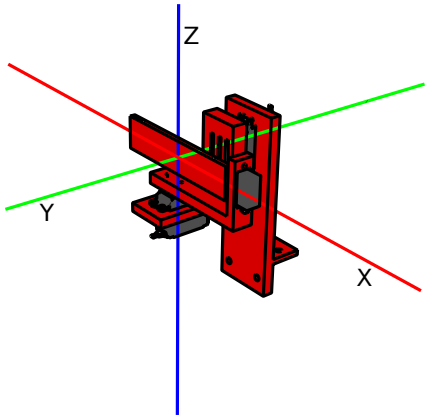



Abbildung 320: Das Koordinatensystem in dem ein Teststreifen im Teststand bewegt wird.

Dadurch werden alle Randfälle berücksichtigt, wie ein Proband einen Test halten kann.
Im laufe des Projekts wurden mit diesem Teststand verschiedene Videos mit unterschiedlichen Teststreifen und verschiedenen Smartphones erstellt. Um die Videos auszuwerten wurde das Testframework (Kapitel 4.5.7.1 auf Seite 259) entwickelt, welches den Bildverarbeitungsalgorithmus aus verschiedenen Entwicklungsstadien miteinander vergleichen kann. Mit diesem kann analysiert werden, wie viele Marker erfolgreich erkannt werden und ob das berechnete Ergebnis der Voranalyse korrekt ist oder nicht.
Im Folgenden wird die Qualität des Bildverarbeitungsalgorithmus in verschiedenen Entwicklungsstadien bewertet. Dazu wurden die aufgenommenen Videos von dem Testframework analysiert.

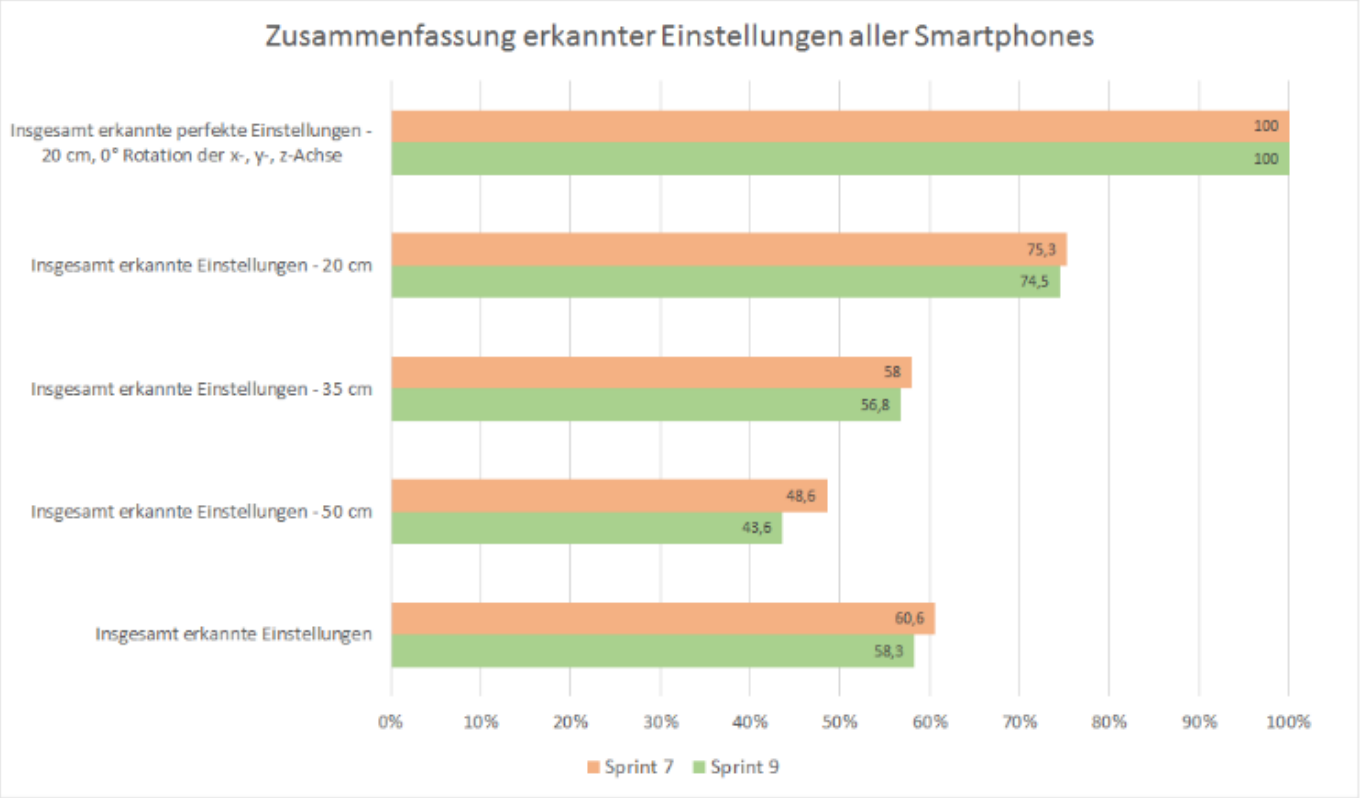



Abbildung 321: Analyse des Bildverarbeitungsalgorithmus für Sprint 7 und Sprint 9.

Die Zusammenfassung der Ergebnisse für die verschiedenen Einstellungen, die mit dem Teststand möglich sind, ist in Abbildung 321 zu sehen. Es zeigt den Bildverarbeitungsalgorithmus von Sprint 7 im Vergleich zu der Version in Sprint 9. Die beiden Versionen werden über die Anzahl an erfolgreich ausgewerteten Teststreifen verglichen. Die Erkennung wurde für den Idealfall, dass die Kamera gerade über dem Teststreifen in einem Abstand von 20cm liegt, perfektioniert. Auffällig ist, dass im Vergleich zu Sprint 7 die Erkennungsrate der Teststreifen in Sprint 9 abgenommen hat. Dies hat den Grund, dass die Erkennung in Sprint 9 genauer geworden ist, sodass viele Fehlerfälle, die in Sprint 7 noch auftreten konnten, in Sprint 9 abgefangen werden und der Algorithmus qualitativ bessere Ergebnisse zurück liefert.

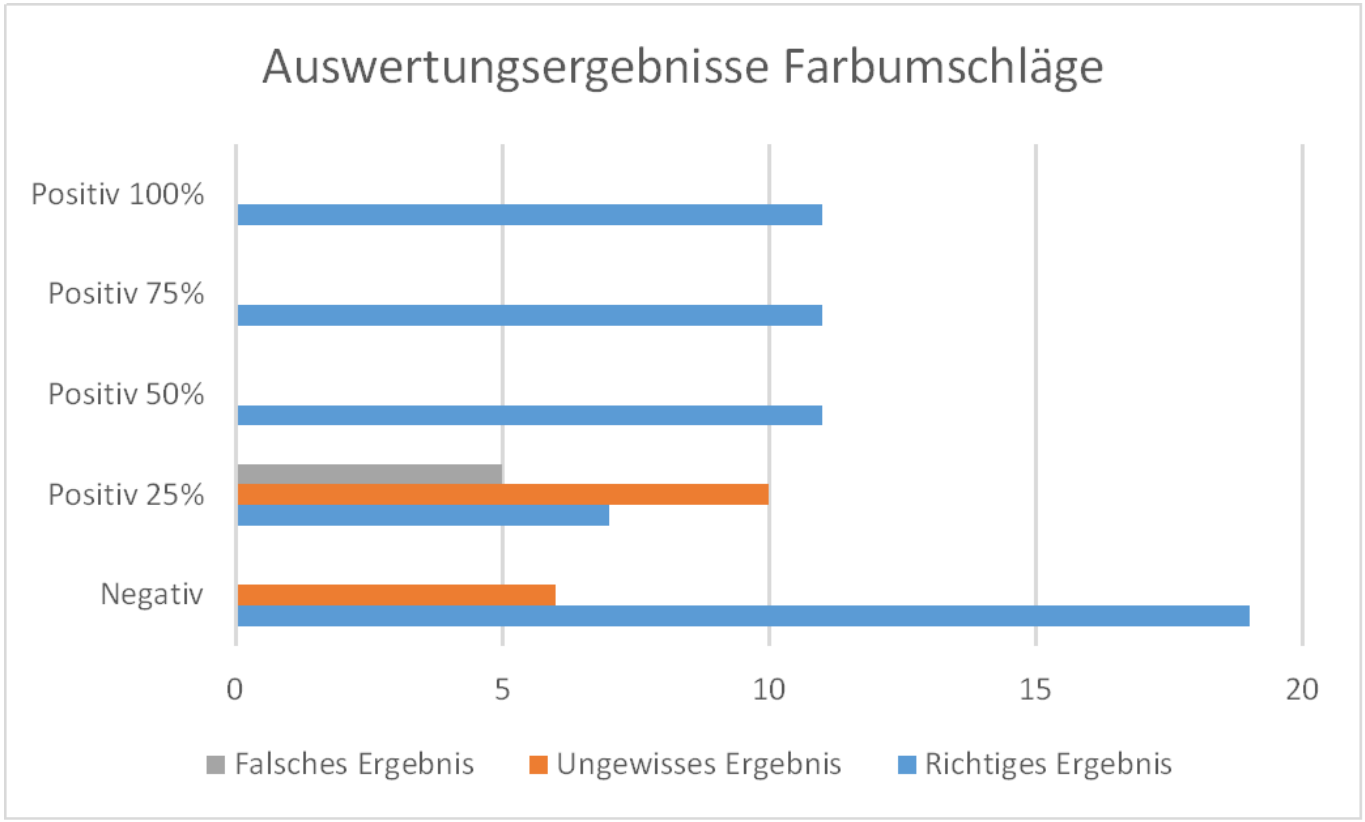



Abbildung 322: Qualität des Bildverarbeitungsalgorithmus am Ende von Sprint 9

In Abbildung 322 ist dargestellt, wie gut der Algorithmus am Ende von Sprint 9 ein korrektes Ergebnis bei der Voranalyse liefert. Wenn der Farbumschlag mit hoher Intensität, also mehr als 50%, stattfindet, liefert der Algorithmus immer ein korrektes Ergebnis. Wenn kein Farbumschlag auftritt, erkennt der Algorithmus in zwei Drittel aller Fälle, dass der Test auch tatsächlich negativ ist. Ansonsten liefert er ein ungewisses Ergebnis zurück. Es kommt also nicht vor, dass der Fall auftritt, dass ein Proband fälschlicherweise als krank eingestuft wird, obwohl er eigentlich gesund ist. Wenn der Farbumschlag sehr undeutlich ist, also nur eine Intensität von 25% vorliegt, trifft der Algorithmus in den meisten Fällen keine Aussage. Dadurch dass die Intensität in diesem Fall sehr undeutlich ist, kann es vorkommen, dass der Algorithmus ein falsches Ergebnis zurück liefert. Eine genauere Analyse der Teststreifen, die nicht erkannt wurden, ist in Kapitel 4.8.3 auf Seite 354.
Abschließend ist im Vergleich zu Sprint 7 die Erkennung des Bildverarbeitungsalgorithmus deutlich besser geworden, da dieser bis dahin zu keinem der Teststreifen eine Aussage treffen konnte. Insgesamt ist der Algorithmus robust genug geworden, um die funktionale Anforderung (FA-4) zu erfüllen und liefert nur in einigen Grenzfällen ein fehlerhaftes Ergebnis.

# 7 Fazit

In diesem abschließenden Kapitel wird ein Fazit der Projektgruppe gezogen. Dazu wird zunächst in Kapitel 7.1 reflektiert, was in den vergangenen zwei Semestern erreicht wurde. Im Anschluss wird in Kapitel 7.2 auf der nächsten Seite aufgezeigt, was in dem Projekt noch hätte umgesetzt oder verbessert werden können.

## 7.1 Reflexion

Insgesamt war das Projekt MEDIC ein vielseitiges Projekt, was durch die verschiedenen Anwendungsbereiche entstand. Durch die unterschiedlichen Nutzergruppen und Anforderungen wurden vielseitige Systeme entwickelt. Einerseits musste ein Bildverarbeitungsalgorithmus entwickelt werden, der zur Auswertung der Teststreifen genutzt werden sollte. Weiterführend musste dieser in mobile Applikationen eingebettet werden. Diese beiden Systeme stellte die Projektgruppe bereits vor eine große Aufgabe. Andererseits kam die Entwicklung einer Weboberfläche, die für verschiedene Nutzer erstellt werden musste, sowie eine Möglichkeit der Kommunikation über einen Server, hinzu.

Der Kern dieses Projekts war die Entwicklung des Bildverarbeitungsalgorithmus, der für die automatische Auswertung der Teststreifen implementiert wurde. Die Schwierigkeit war es, diesen robust gegenüber Fehlern zu machen, sodass er unter realistischen Bedingungen einwandfrei funktioniert und kein falsches Ergebnis zurückliefert. Diese Aufgabe wurde durch das unbekannte Design eines Teststreifens erschwert. Im Laufe des Projektes wurden die Anforderungen an diese Teststreifen immer wieder neu definiert, sodass diese so dynamisch wie möglich gestaltet werden mussten. Dazu musste zusätzlich ein Teststreifengenerator entwickelt und immer wieder überarbeitet werden, mit dem Teststreifen erstellt werden konnten, die durch den Bildverarbeitungsalgorithmus erkannt werden. Aufgrund der hohen Anforderungen an den Bildverarbeitungsalgorithmus wurden außerdem Möglichkeiten zur Bewertung entwickelt: Dies umfasst zum Einen den Teststand, der die Aufnahme eines Teststreifens unter allen Aufnahmewinkeln, Abständen und unter reproduzierbaren Bedingungen simulieren kann. Zum Anderen wurde das Testframework entwickelt, welches die Qualität des Algorithmus angeben kann, indem die Erkennungsrate von verschiedenen Entwicklungsstufen des Algorithmus verglichen werden kann.

Eine weitere Kernaufgabe war die Implementierung einer mobilen Applikation, mit der die Auswertung eines Teststreifens vorgenommen werden soll. Um eine möglichst große Personengruppe zu erreichen, wurde die mobile Applikation nativ in Android und iOS implementiert. Diese Anwendungen beinhalten nicht nur den Bildverarbeitungsalgorithmus, der in C++ implementiert wurde, um eine einheitliche Auswertung zu gewährleisten und um Änderungen zu vereinfachen, sondern führen den Nutzer mit einfacher Sprache durch den gesamten Testablauf. Durch Texte mit zusätzlichen Informationen für blinde Nutzer kann auch die Sprachausgabe genutzt werden, welche die Nutzung der mobilen Applikationen für sehbehinderte Personen ermöglicht.

Eine weitere Aufgabe bestand in der Umsetzung einer Weboberfläche, die für verschiedene Nutzergruppen erstellt werden sollte:

- Mediziner sollten bei zweifelhaften Testresultaten eine Diagnose abgeben können.
- Statistiker haben die Möglichkeit Auswertungen zu treffen, die auf den gesammelten Informationen basieren, um mögliche Krankheitsverläufe, die zu Pandemien führen, frühzeitig zu erkennen.
- Administratoren, die neue Teststreifen für verschiedene Krankheiten anlegen können und verwaltende Aufgaben haben, wie das Bestätigen eines Mediziners.

Für die Erstellung neuer Teststreifenvarianten wurde der oben genannte Teststreifengenerator verwendet. Um eine reibungslose Kommunikation der einzelnen Komponenten zu gewährleisten, wurde ein Server benötigt. Dieser hatte zahlreiche Anforderungen, wie etwa dass er 10.000 gleichzeitigen Anfragen standhalten musste. Zu diesem Zweck wurde dieser in der Programmiersprache Go implementiert.

Diese Arbeit konnte nur mit einem engagierten Team umgesetzt werden, das koordiniert zusammenarbeitete. Dies wurde durch eine positive Einstellung der Teammitglieder erreicht, die durch das Projekt motiviert wurden. Unterstützt wurde dies weiterhin durch ein gut durchdachtes Projektmanagement, welches auf Scrum[31] basierte und mit dem die Aufgaben gleichmäßig verteilt wurden. Durch die Verwendung von Arbeitspaketen, die jedes Projektmitglied zum Bearbeiten wählen konnte, konnten Wartezeiten minimiert werden, da die Verteilung der Pakete sonst erst zu den wöchentlichen Treffen geschehen wäre. Unterstützt wurde dieses Vorgehen durch Programmierwochenenden, an denen die gesamte Gruppe teilnahm, sodass die Kommunikationswege kurz gehalten werden konnten. Dadurch wurde das Projekt noch weiter vorangetrieben und es konnten weitere, über die definierten Anforderungen hinweg gehende Systeme entwickelt werden.

Das Projekt kann als Erfolg betrachtet werden. Zum Einen konnten alle funktionalen und nichtfunktionalen Anforderungen an das System, die vorher definiert wurden, umgesetzt werden. Auf Anforderungen, die während des Projektes aufkamen, wie die ständige Überarbeitung des Teststreifendesigns, konnte gut reagiert werden. Während des Projektverlaufs wurden einige zusätzliche Komponenten, wie der Teststand und das Testframework, entwickelt.

## 7.2 Ausblick

Es konnten alle Anforderungen in dem Projekt umgesetzt werden. Trotzdem gibt es noch Möglichkeiten zur Verbesserung und Weiterentwicklung.
Bisher wurde der Farbumschlag eines Teststreifen durch den Teststreifengenerator erstellt. Dieser hat den Bereich für den Farbumschlag gut simuliert, doch auf den echten Streifen ist dies eine chemische Reaktion. Wenn die Teststreifen von den Projektpartnern entwickelt wurden, sodass diese tatsächlich die gewünschte Farbreaktion erwirken, wäre die Anwendung unseres Systems unter realen Bedingungen denkbar. Da die Betreuung unseres Systems im Anschluss an die Projektgruppe nicht garantiert werden kann, wurden alle Produkte für Entwickler ausreichend dokumentiert. Außerdem gibt es Handbücher zur Installation und Nutzung der einzelnen Komponenten, sodass ein Mediziner diese ohne weitere Hilfe nutzen kann.
Im Fall, dass das System tatsächlich eingesetzt wird, sollte in den mobilen Applikationen eine Liste mit Telefonnummern von Medizinern angegeben werden, damit der Proband direkt alle Informationen erhalten kann, falls Unklarheiten zu dem Testergebnis bestehen.

Der Bildverarbeitungsalgorithmus bietet Möglichkeiten der Verbesserung. Dieser wurde im Verlauf des Projektes zwar optimiert, jedoch könnten verschiedene Aspekte wie die Schattenrückrechnung, die Anpassung der Lichtverhältnisse sowie beliebig verzerrte, zerknitterte und verknickte Teststreifen genauer untersucht werden. Die meisten dieser Aspekte wurden im Projektverlauf zwar analysiert, konnten aber auf Grund von Zeit -oder Ressourcenmangel nicht weiter betrachtet werden, sodass einige Implementierungen noch denkbar sind. Es war allerdings möglich zu bestimmen, wenn diese Fehlerquellen auftreten, sodass der Algorithmus in dem Fall abbricht, den Fehler erkennt und dies an die mobilen Applikationen zurück gibt. In zukünftigen Arbeiten könnte eine erneute Studie zu diesen Themen gemacht werden, sodass diese Teststreifen eventuell doch ausgewertet werden können.

# Literatur


[1] R. Guo, Q. Dai, and D. Hoiem, "Single-image shadow detection and removal using paired regions," in Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on, pp. 2033–2040, IEEE, 2011.

[2] J. Canny, "A computational approach to edge detection," Pattern Analysis and Machine Intelligence, IEEE Transactions on, no. 6, pp. 679–698, 1986.

[3] I. Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," a talk at the Stanford Artificial Project in, pp. 271–272, 1968.

[4] S. Bundesamt, "Gesundheitsausgaben im jahr 2014 bei 328 milliarden euro," 2016. [Online; Stand 17. März 2016].

[5] Wikipedia, "Point-of-care-testing — wikipedia, die freie enzyklopädie," 2015. [Online; Stand 17. März 2016].

[6] Statista GmbH, "Geschätzte anzahl von todesfällen aufgrund von grippepandemien weltweit," Juli 2009.

[7] WeltN24 GmbH, "Afrika droht jetzt auch die finanzielle katastrophe," Oktober 2014.

[8] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, et al., "Fiji: an open-source platform for biological-image analysis," Nature methods, vol. 9, no. 7, pp. 676–682, 2012.

[9] C. A. Schneider, W. S. Rasband, K. W. Eliceiri, et al., "Nih image to imagej: 25 years of image analysis," Nat methods, vol. 9, no. 7, pp. 671–675, 2012.

[10] J. W. Eaton et al., "Gnu octave."

[11] MATLAB, version 7.10.0 (R2010a). Natick, Massachusetts: The MathWorks Inc., 2010.

[12] D. J. Tobias Tiemerding, Claas Diederichs, "Offis automation framework."

[13] J. M. Buenaposada and B. Luis, "Variations of grey world for face tracking," Image Processing & Communications, vol. 7, no. 3-4, pp. 51–61, 2001.

[14] E. H. Land and J. J. McCann, "Lightness and retinex theory," JOSA, vol. 61, no. 1, pp. 1–11, 1971.

[15] Y. Li, T. Sasagawa, and P. Gong, "A system of the shadow detection and shadow removal for high resolution city aerial photo," in XXth ISPRS Congress, pp. 12–23, 2004.

[16] Y. Wang, "Real-time moving vehicle detection with cast shadow removal in video based on conditional random field," Circuits and Systems for Video Technology, IEEE Transactions on, vol. 19, no. 3, pp. 437–441, 2009.

[17] G. D. Finlayson, M. S. Drew, and C. Lu, "Entropy minimization for shadow removal," International Journal of Computer Vision, vol. 85, no. 1, pp. 35–57, 2009.

[18] R. Gross and V. Brajovic, "An image preprocessing algorithm for illumination invariant face recognition," in Audio-and Video-Based Biometric Person Authentication, pp. 10–18, Springer, 2003.

[19] A. Sanin, C. Sanderson, and B. C. Lovell, "Shadow detection: A survey and comparative evaluation of recent methods," Pattern recognition, vol. 45, no. 4, pp. 1684–1695, 2012.

[20] T. Gevers and A. W. Smeulders, "Color-based object recognition," Pattern recognition, vol. 32, no. 3, pp. 453–464, 1999.

[21] G. Bradski Dr. Dobb's Journal of Software Tools, 2000.

[22] J. D. Hunter, "Matplotlib: A 2d graphics environment," Computing In Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.

[23] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," Cartographica: The International Journal for Geographic Information and Geovisualization, vol. 10, no. 2, pp. 112–122, 1973.

[24] N. Otsu, "A threshold selection method from gray-level histograms," Automatica, vol. 11, no. 285-296, pp. 23–27, 1975.

[25] S. O'Grady, "The redmonk programming language rankings: January 2013." `https://redmonk.com/sogrady/2013/02/28/language-rankings-1-13/`, 2013. Accessed: 2016-03-21.

[26] S. O'Grady, "The redmonk programming language rankings: January 2016." `https://redmonk.com/sogrady/2016/02/19/language-rankings-1-16/`, 2016. Accessed: 2016-03-21.

[27] C. Majors, "How we moved our api from ruby to go and saved our sanity." `http://blog.parse.com/learn/how-we-moved-our-api-from-ruby-to-go-and-saved-our-sanity/`, 2015. Accessed: 2016-03-21.

[28] T. Reeder, "How we went from 30 servers to 2: Go." `https://www.iron.io/how-we-went-from-30-servers-to-2-go/`, 2013. Accessed: 2016-03-21.

[29] A. Freier, P. Karlton, and P. Kocher, "The secure sockets layer (ssl) protocol version 3.0," 2011.

[30] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," J. Usability Studies, vol. 4, pp. 114–123, May 2009.

[31] K. Schwaber and M. Beedle, "Agilè software development with scrum," 2002.