

Dokumentation

Marcel Gauss, Nitin Kishore,

René Gunesch, Antonio Lombardi

# Inhaltsverzeichnis

Inhaltsverzeichnis 2

Überblick / Aufgabenstellung / Motivation 3

1 Vorgehen 4

2 Ziele / Requirments 6

3 Entwicklungsumgebung 7

4 Konzeption 8

4.1 Benutzungskonzept 8

4.2 Persona 8

4.3 User Story 10

4.4 Use Case 11

4.5 Design 13

4.5.1 Style Guide 13

4.5.2 Mockup 14

4.5.3 Logo 15

5 Technologien 16

5.1 OCR 16

5.2 Text to speech (TTS) 17

5.3 Speech to text (STT) 17

6 Umsetzung 19

6.1 Grafische Oberfläche 19

6.2 Architektur 21

6.2.1 Allgemein 22

6.3 Einbindung 22

6.3.1 OCR 22

6.3.2 TTS 23

6.3.3 STT 25

6.4 Sprachen 26

7 Ausblick 27

8 Fazit 28

Quellenverzeichnis 29

# 

# Aufgabenstellung / Motivation

Die Motivation des Projektes, war eine Applikation zu entwickeln, welche Menschen mit Sehschwäche, Analphabeten genauso wie Touristen zu helfen. Eine Applikation für Android Smartphones, die handgeschriebene, sowie Computer-geschriebene Texte erkennt, verarbeitet und dann auditiv ausgibt. Diese App soll das Leben der oben genannten Menschen erleichtern und alltägliche Probleme aus dem Weg räumen. Für Analphabeten und Menschen mit eingeschränkter Sehfähigkeit soll die App als „Vorleser“ fungieren, für Touristen soll sie eher als Übersetzer arbeiten. Das sind die 3 größten Zielgruppen, welche mit der Applikation unterstützt werden sollen.

# Vorgehen

Das Projekt wurde mit Hilfe von Scrum durchgeführt. Scrum gehört zu den Agilen-Entwicklungsmethoden. Diese sind grundlegend verschieden von den sonst so normalen Prozessmanagement Methoden. Einer der Hauptmerkmale der Traditionellen Prozessmanagement Methoden ist die Prozesslenkung durch Prozessbeschreibung und Planung. Beim Scrum wird das Problem in kleine Teilaufgaben zerlegt. In Scrum gibt es sogenannte Sprints. Sprints sind in kurzen und zyklischen Abständen eingeteilt, welche meistens 30 Tage andauern. In diesen wird versucht eine Teilaufgabe zu lösen. Eine enge Zusammenarbeit zwischen den Entwicklern und Kunden ist die wesentliche Stärke von Scrum. Die Entwickler werden in die strategischen Überlegungen miteinbezogen. Dadurch kennen die Entwickler die Hintergrundinformationen, wie etwa bestimmte Features welche der Kunde haben möchte. Mit diesen Informationen können die Entwickler innovative Problemlösungsansätze entwickeln.

In der agilen Methode Scrum wird in verschiedene Rollen unterschieden. Im Folgenden werden die zwei wichtigsten Rollen unterschieden:

**Der Product Owner** – Visionär:

Die Produktentwicklung wird von dem Product Owner geplant und gesteuert. Er ist dafür verantwortlich, dass die Teams das korrekte Produkt erstellen und die finanziellen Rahmenbedingungen eingehalten werden. Er arbeitet eng mit den Entwicklungsteams zusammen, besitzt die Entscheidungsmacht und arbeitet am Product Backlog.

**Das Entwicklungsteam** – Lieferanten:

Das Entwicklungsteam leistet die eigentliche Arbeit an dem Produkt. Es ist für seine Aufgaben selbst verantwortlich und berechtigt all das zu tun, was nötig ist, um das geplante Ergebnis zu erreichen. Die Menge der jeweiligen Aufgaben wird von den Entwicklungsteams eigenständig festgelegt.

Was macht ein Entwicklungsteam erfolgreich?

Damit ein Team erfolgreich ist, sollte es diese drei Eigenschaften besitzen:

* Autonom: eigenständig organisiertes Entwicklungsteam.
* Funktionsübergreifend: besitzt das notwendige Wissen und die geeigneten Fertigkeiten.
* Transzendent: hat den Willen sich und das Produkt stets zu verbessern.

Der Kunde als Stakeholder

Eine weitere Rolle des Scrums sind die Stakeholder, die sich außerhalb des Unternehmens befinden. Im Folgenden wird der Kunde als Stakeholder genauer beschrieben:

Der Kunde bekommt das Produkt nach Fertigstellung. Kunden können interne als auch externe Personen sein.

Die erfolgreiche Einführung von Scrum

Um Scrum erfolgreich als Methode einzuführen sind fünf Tätigkeiten erforderlich:

* **Situationsbewusstsein** dafür, dass die Abläufe momentan nicht zu optimalen Ergebnissen führen.
* **Wille** zur Ausführung von Scrum, um die aktuellen Hindernisse zu umgehen.
* **Befähigung** mit Scrum erfolgreich zu sein.
* **Bekanntmachung** von Scrum, durch Weitergabe der Erfahrungen.
* **Ausweitung** von Scrum, sodass das gesamte Unternehmen eingebunden wird.

Für die Umsetzung von Scrum wurde Kanbanboard verwendet. In diesem wurde die Aufgabe in kleine Teilaufgaben unterteilt. Dieses Board ist eine Tabellarische Darstellung der Aufgaben und in welchen Status sich diese Befinden.

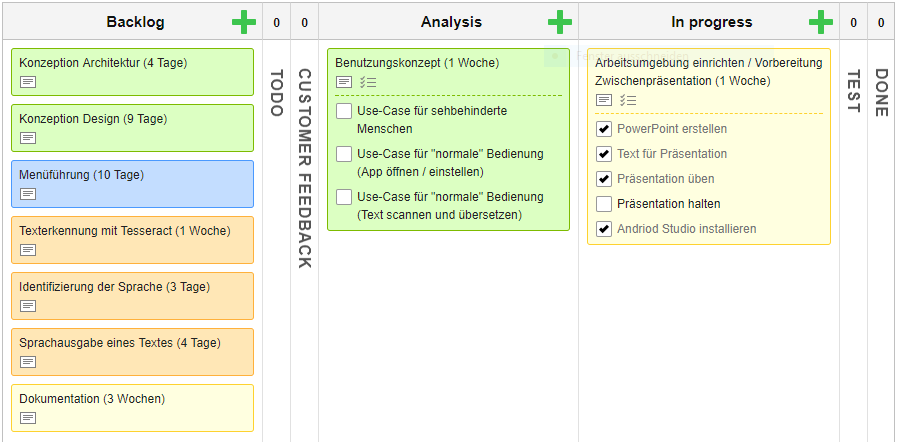


Abbildung 1: Kanban Board

# Ziele / Requirments

Ziele des Projektes waren zum einen, das Anwenden einer agilen Entwicklungsmethode. Hierfür wurde, wie oben bereits beschrieben, die agile Methode Scrum angewendet. Weiterhin war ein weiteres Ziel, eine lauffähige Applikation zu entwickeln, welche Texte mittels der Smartphone eigenen Kamera einlesen und erkennen, diese Texte dann separieren und am Ende wieder auditiv ausgeben kann. Ebenso war es wichtig, Menschen mit eingeschränkter Sicht, Feedback zu geben, damit diese wissen, wann ein Text erkannt wurde oder wann eine Einstellung geändert wurde. Dies sollte mittels haptischen Impulsen geschehen. Für die Verarbeitung des Textes wurde die Tesseract-Engine verwendet. Diese dient dazu, einen Text zu erkennen, als String zu separieren und ebendiesen String zurückzugeben. Weiterhin sollte die Sprache des eingelesenen Textes erkannt und zurückgegeben werden, um danach eine flüssige auditive Ausgabe des Strings in der erkannten Sprache zu gewährleisten. Das Design sollte responsive umgesetzt werden, also anpassbar auf alle gängigen Smartphone-Größen, um die Benutzung nicht auf eine Smartphone-Größe zu beschränken. Auch sollte das Design eine barrierefreie Nutzung unterstützen und somit von sowohl Sehgeschwächten, als auch Sehstarken Menschen gleichermaßen bedient werden können. Des Weiteren sollte es möglich sein, die Applikation über die Sprache zu bedienen. Das sollte umgesetzt werden, um die Barrierefreiheit weiter zu unterstützen. Eine weitere Anforderung, war es die App in verschiedenen Sprachen benutzen zu können. So sollte die Applikation beim ersten Öffnen erkennen, welche Systemsprache eingestellt wurde und diese dann übernehmen. Ebenso sollte die App aber auch eigens auf eine andere Sprache umgestellt werden können.

# Entwicklungsumgebung

Als Entwicklungsumgebung wurde Android Studio verwendet. Dies wurde aufgrund der einfachen Bediendung, als auch auf Grund der hervorragenden bereits eingebundenen Unterstützungen zum Entwickeln für Android Anwendungen verwendet. Zu diesen gehört unter anderem der Android Studio Emulator, anhand dessen lassen sich die Android Anwendungen, ohne ein Android Gerat physikalisch zur Hand zu haben, testen. Zudem ist der Emulator auch eine sehr gute Unterstützung Fehler zu finden und auch verwendete Application programming interfaces (im Folgenden API‘s) nachzuvollziehen und zu verstehen. Ein großer Aspekt zum Fehler finden ist das Debuggen. Der Debugger erlaubt es den Code Schritt für Schritt zu inspizieren. Hierzu hat das Android Studio hervorragende Werkzeuge, wie zum Beispiel den Stacktrace und das Überwachen der verwendeten Variablen, in Hinsicht auf Kapselung, Struktur und Inhalt. Ein weiterer sehr großer Vorteil von Android Studio ist die Möglichkeit die Grafische Oberfläche einfach zusammenzustellen. Hierzu gibt es in Android Studio einen „Designer“ in dem kann man per „Drag and Drop“ Elemente hinzufügen und diese auch direkt visuell manipulieren. Dies ist eine Funktion die von sehr großem Vorteil ist wenn noch nicht allzu viel Kontakt mit der Entwicklung von Android Anwendungen hat. Zudem kann man für die einzelnen Seiten der Anwendung auf vorgefertigte Templates zurückgreifen.

Die Struktur und die Kenntnis welche Dateien wo in der Struktur an- bzw. abgelegt werden müssen kann einen Anwender die den ersten Kontakt zur Entwicklung von Android Anwendungen hat überfordern. Aber diese Hürde wird auch größtenteils von Android Studio erleichtert. Wird zum Beispiel eine neue Seite für die Anwendung generiert, so wird nicht zur Datei automatisch an der richtigen Stelle generiert, es wird zusätzlich auch noch direkt die Codebehind Datei an der richtigen Stelle generiert und der Anwender muss sich nicht mehr Gedanken über Fehlende Dateien oder die Implementierung der Logik zum Aufruf der Seite machen.

# Konzeption

## Benutzungskonzept

## Persona

Zu Beginn des Projektes wurden Personas erstellt, diese sollten dazu dienen, jegliche Zielgruppen abzudecken, die von einer Applikation dieser Art profitieren können. Diese Zielgruppen flossen dann in der Konzeption des Projektes mit ein. Infolgedessen wurden das Benutzungskonzept sowie die Funktionalität in der Konzeption allen Zielgruppen gerecht. Die Hauptanwendung, das Unterstützen beim Erkennen von Texten für Menschen mit eingeschränktem Sehvermögen wurde hierdurch nicht beeinträchtigt.

**Yusuf Jilmaz**

Abbildung 2:Yusuf

* 66 Jahre alt
* Wohnort: Hamburg
* Verheiratet, 2 Kinder
* Rentner

Erfahrungen:

* Yusuf leidet am Grünen Star (Glaukom) dadurch ist seine Sehfertigkeit stark eingeschränkt
* Das Lesen ist für Yusuf eine große Herausforderung, da sein Sehfeld nur noch 20 % beträgt.
* Aufgrund seines Alters ist Yusuf nicht sehr Technik affin.

Aufgaben und Ziele der App die Yusuf erwartet

* Aufgrund seines Sehvermögens erwartet Yusuf eine Applikation die sich per Spracheingabe steuern lässt.
* Da Yusuf keine große Technik Affinität vorweisen kann, erwartet er eine einfache Handhabung der Applikation.

Anforderungen von Yusuf - User Story

* Als User möchte Ich die App möglichst einfach ohne besonderen Vorkenntnisse für alltäglichen Aufgaben, zum Beispiel einkaufen, bedienen können.

**Tayo Modibo**

Abbildung 3: Tayo

* 32 Jahre alt
* Wohnort: Frankfurt
* Ledig, keine Kinder
* Arbeitslos

Erfahrungen:

* Tayo floh 2016 aus Uganda vor dem Bürgerkrieg nach Deutschland
* Aufgrund seines Heimatslandes und seiner finanziellen Lage dort, konnte Tayo das Lesen und Schreiben nicht erlernen.
* Tayo ist Analphabet.

Aufgaben und Ziele der App die Tayo erwartet

* Die Applikation soll mittels Spracheingabe funktionieren
* Die Applikation sollte während der Hauptbedienung kein Verständnis der Textelemente erwarten.

**Sabine Müller**

Abbildung 4:Sabine

* 20 Jahre alt
* Wohnort: Stuttgart
* Ledig, keine Kinder
* Studiert Lehramt in Stuttgart

Erfahrungen:

* Sabine kann ein gewisses Maß an Technik Affinität vorweisen, seit Ihrer Kindheit ist Sie mit Computer aufgewachsen und kennt unterschiedliche User Interfaces.
* Da Sabine sehr oft reisen geht, vor allem auch in Länder mit einem anderen Schriftensystem findet sich Sabine dort nur schwer zurecht.
* Sabine spricht nur Deutsch und Englisch, reißt jedoch auch in Länder in denen diese Sprachen nicht üblich sind.

Aufgaben und Ziele der App die Sabine erwartet

* Da Sabine nur das lateinische Schriftensystem beherrscht, wünscht Sie sich eine Übersetzung der Schriften die Sie nicht versteht.
* Da Sie viele der Sprachen nicht versteht, auch wenn Sie im lateinischen Schriftensystem geschrieben sind, wünscht sich Sabine davon ebenfalls eine Übersetzung.

## User Story

Um die Anforderungen der unterschiedlichen Zielgruppen eindeutig zu identifizieren, wurden User Storys erstellt, die die Funktionalität der Applikation aus Sicht der Benutzer definieren.

* Als User möchte ich die Anwendung auch für Übersetzungen im Ausland verwenden können.
* Als User möchte ich die Anwendung mit wenigen Klicks starten können.
* Als User möchte ich die Anwendung bedienen können, ohne gewisse Textelemente der Anwendung lesen können zu müssen.
* Als User möchte ich Texte die ich zuvor abfotografiert habe und auf meinem Handy gespeichert habe, später Auditiv ausgeben lassen.
* Als User möchte Ich die App möglichst einfach ohne besonderen Vorkenntnisse für alltäglichen Aufgaben, zum Beispiel einkaufen, bedienen können.
* Als User möchte ich die Anwendung ohne Sehvermögen bedienen können
* Als User möchte ich Texte abfotografieren können und Sie Auditive ausgeben lassen.

## Use Case

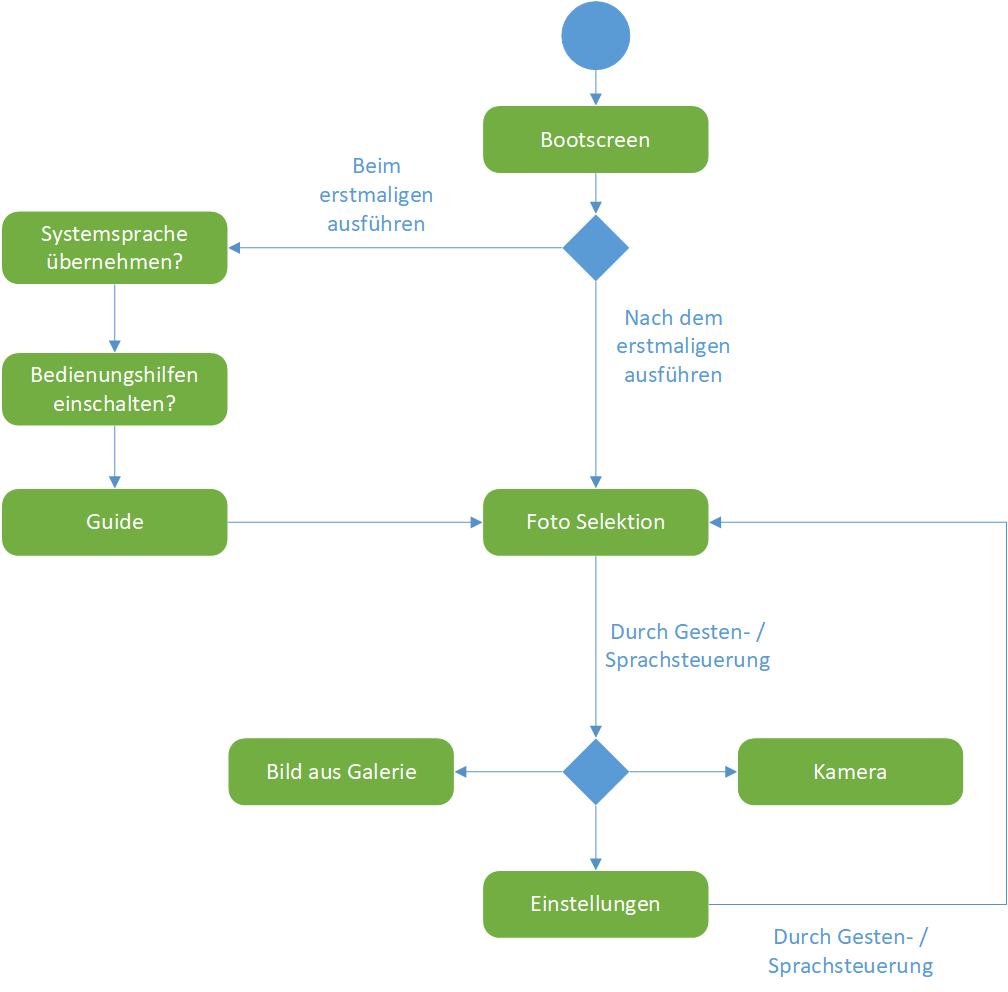


Abbildung 5: Use Case App

Beim Starten der Applikation wird der Bootscreen angezeigt. Dann wird differenziert, ob der Benutzer die Applikation zum ersten Mal startet oder sie schon einmal gestartet hat. Wird sie zum ersten Mal ausgeführt, so wird einmalig gefragt ob der Benutzer die aktuelle Systemsprache übernehmen möchte. Dies dient dazu, den Inhalt sowie die Auditive Ausgabe der Applikation an den Benutzer anzupassen. Anschließend wird der Benutzer gefragt ob er die Bedienungshilfen einschalten möchte. Zu den Bedienungshilfen gehören das Auditive sowie das Haptische Feedback. Ist dieses eingeschalten, erhält der Benutzer auf jeder Seite der Applikation auditives Feedback. Da der weitere Verwendungszweck, das Übersetzten von Texten auch für Menschen ohne jegliche Barrieren Verwendung findet, wird diese Abfrage einmalig getätigt. Anschließend wird der Guide ausgeführt. Hier wird dem Benutzer das Benutzungskonzept der Applikation nähergebracht. Dieser erklärt auf drei Seiten welche Funktionalität geboten wird und wie der Benutzer zu diesen gelangt. Folglich, nach dem Guide wird eine neue Seite aufgerufen, die Foto Selektion. Nachdem der Benutzer die einmalige Konfiguration durchlaufen hat, gelangt er stets nach dem Bootscreen auf diese Seite. Hier hat der Benutzer die Möglichkeit Bilder selber zu schießen sowie Bilder aus der Galerie zu wählen. Diese Bilder werden anschließend verarbeitet und dann auf einer neuen Seite Auditive ausgegeben. Auf jeder Seite besteht die Möglichkeit in die Einstellungen zulangen, sei es über die Sprachsteuerung oder über Gesten. Der Benutzer kann auch via Geste oder Sprachsteuerung von jeder Seite auf die Foto-Selektion Seite gelangen.

Die genaue Informationsarchitektur wird im folgenden Abschnitt erläutert.

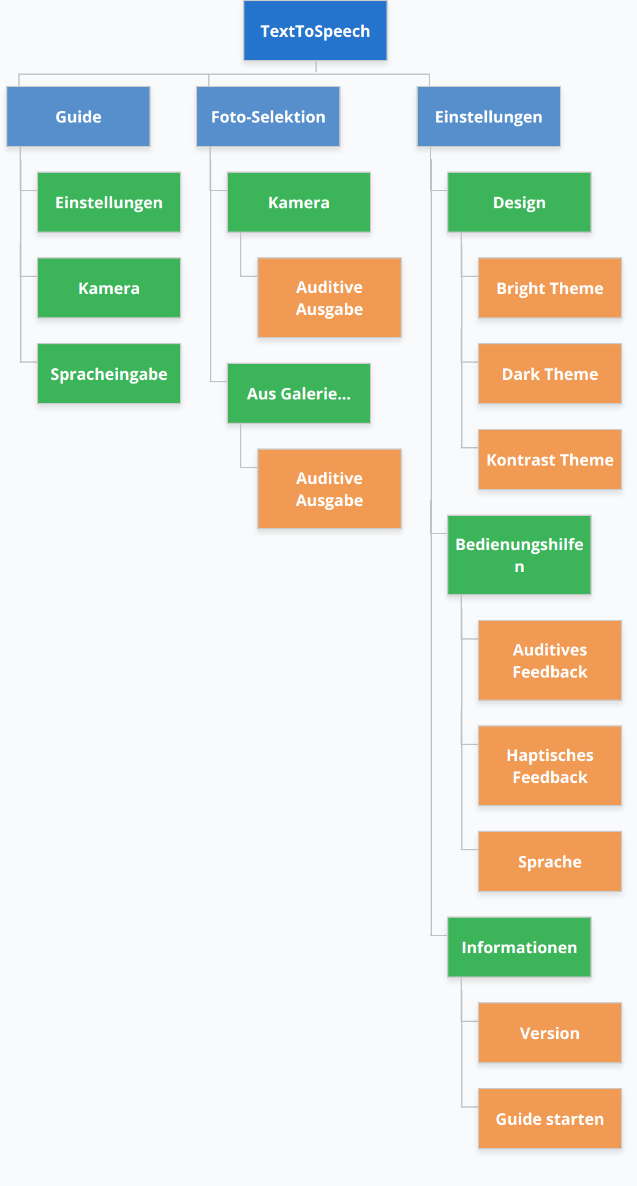


Abbildung 6: Informationsarchitektur

In den Einstellungen gibt es folgende Reiter. Design, Bedienungshilfen sowie Informationen. Im Reiter “Bedienungshilfen” besteht die Möglichkeit die Bedienungshilfen inklusive Sprache zu ändern. Hier lässt sich das Auditive sowie Haptische Feedback an- und ausschalten. Sprachen, die wir zu Beginn unterstützen sind Deutsch und Englisch. Im Design werden basierend auf Präferenzen des Benutzers andere Themes angeboten. So wird ein “Dark Theme”, “Brigth Theme” sowie “Contrast Theme” angeboten. Im Reiter Informationen, lassen sich die Version sowie der Guide anzeigen.

## Design

### Style Guide

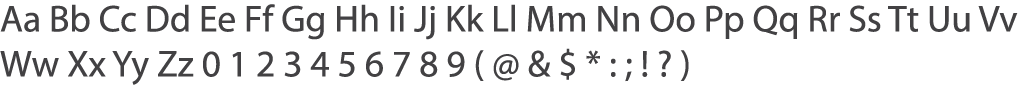
Um ein Einheitliches Design der Applikation bei der Entwicklung zu gewährleisten wurde ein Style Guide entwickelt. Dieser Beschreibt die Hauptelemente der Seiten.

#### Farben



Abbildung 7: Farbpalette

#### Typographie

****

**Schriftart:** Material.Small

**Schriftschnitt:** Sans-Serif

**Schriftgrad:** 20sp, 28sp, 36sp

#### Grafische Elemente

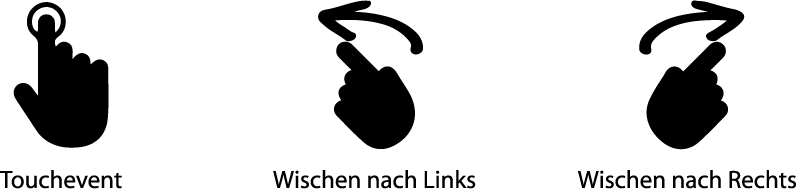
****

Abbildung 8:Touchevents

Die verwendeten Icons stammen aus einer Icon-Bibliothek. Sie sind kostenlos sowie für den Kommerziellen Gebrauch zu nutzen, sie erfordern keine Verlinkung auf die Webseite des Designers. Diese wurden etwas modifiziert um die Semantik auf jeglichen Displaygrößen zu gewährleisten.

### Mockup

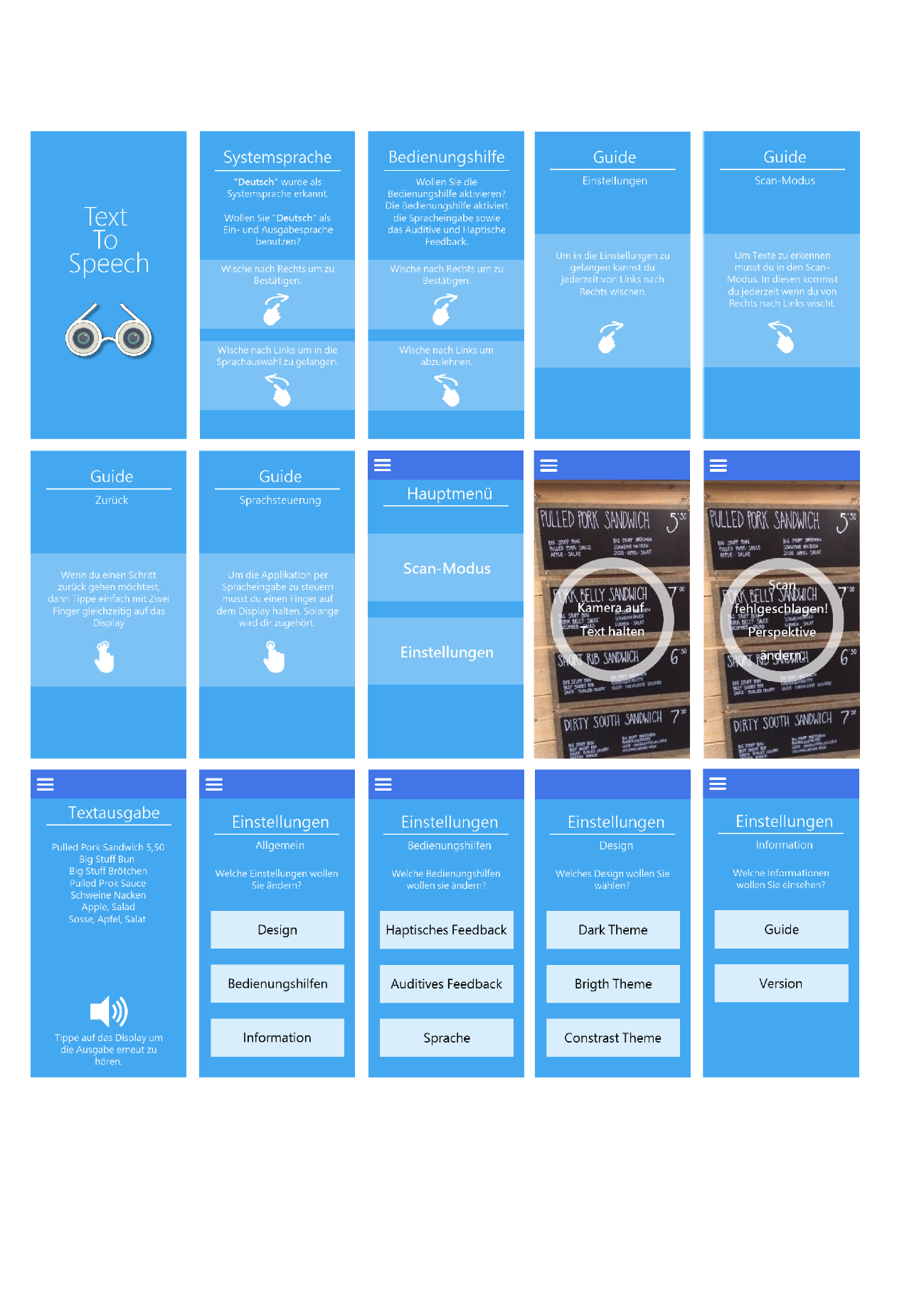
Um zu Beginn einen Überblick für den Aufbau der Applikation zu erhalten, wurde in Azure ein Mockup erstellt. Hier wurde vorab das Design der Applikation definiert, jedoch war dieses in diesem Schritt noch nicht Endgültig wie beim Style Guide. Die einzelnen Seiten des Mockups werden nun abgebildet.

Abbildung 9:Mockup Anwendung

Im Laufe des Entwicklungsprozesses haben sich einzelne Seiten verändert. So wurde die Seite „Guide – Zurück“ entfernt, die Geste hierzu wurde auch nicht implementiert.

Des Weiteren wurde die Seite „Hauptmenü“ entfernt und durch die Seite „Fotoselektion“ ersetzt. Da man durch eine einfache Geste in die Einstellungen gelangt, wurde die Seite „Hauptmenü“ entfernt. Die Seite „Fotoselektion“ bietet stattdessen die Auswahl an, Fotos selber zu schießen oder aus der Galerie auszuwählen.

### Logo

Um ein einzigartiges und passendes Erscheinungsbild zu gestalten wurde ein Logo erstellt. Ziel des Logos war es einen hohen Wiedererkennungswert zu schaffen.

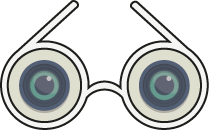


Abbildung 10: Logo

Das Hauptlogo besteht aus einer Brille sowie zwei Kameralinsen. Die Kameralinsen dienen im Logo als Brillengläser. Das Logo soll symbolisieren, dass via Kamera eine Sehhilfe angeboten wird.

Dieses wurde in Adobe Illustrator als Vektor-Grafik erstellt um Aliasing-Effekte bei der Vergrößerung der Grafik zu vermeiden. Als Vektor Datei lässt sich das Logo in beliebigen Größen erstellen.

**Launcher Icons**

Des Weiteren wurde passend zum Logo Zwei Launcher Icons entworfen, die sowohl auf dem Betriebssystem als auch im Playstore als Logo der Applikation dienen.

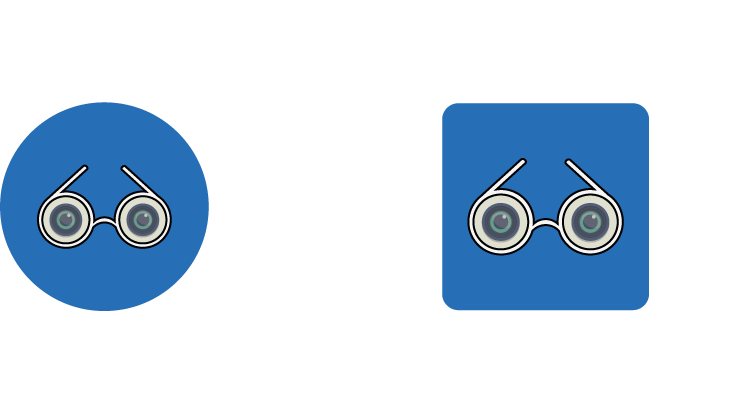
Diese wurden in Adobe Illustrator als Vektor-Grafiken erstellt um Variable auf die unterschiedlichen Displaygrößen reagieren zu können.

Abbildung 11: Icons

# Technologien

Um alle Use-Cases zu erfüllen wurden drei Technologien verwendet.

## OCR

Zur Optischen Zeichenerkennung (OCR) wurde eine Gabelung von *Tesseract Tool for Android* Namens *tess-two* verwendet. Diese Engine erlaubt es, eine IMG Datei zu übergeben um ein String zu erhalten, der idealerweise den Inhalt der IMG Datei enthält. Dieser String kann dann auch als TXT Datei gespeichert und weiterverwendet werden.

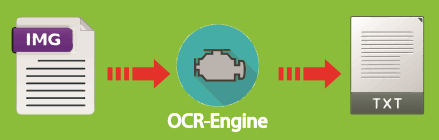


Abbildung 12: OCR Engine

Zur Verbesserung der Texterkennung werden Trainingsdaten für *tess-two* verwendet. Diese Dateien sind an den gewünschten Sprachen gebunden, das heißt möchte man einen deutschen Text erkennen, werden die deutschen Trainingsdaten verwendend.

Anders wie bei einem herkömmlichen Scanner, indem ein optisches Medium in eine Bilddatei umgewandelt wird, wird bei einer OCR jeder individuell erkannte Buchstabe auf einem Bild gescannt um diese dann als veränderbare Datei zu konvertieren. Moderne OCR erreichen dies durch Bereinigung, erkennen individueller Buchstaben und Vergleiche mit Hilfe von Wörterbücher.

Bei der Bereinigung wird das Bild verändert um den Text leichter erkennbar zu machen. Staub, geknicktes oder verfärbtes Papier wird bereinigt. Farben und Graustufen werden zu Schwarzweiß konvertiert.

Bei der individuellen Buchstabenerkennung werden dann die gescannten Elemente in ein Raster gelegt und mit einer Datenbank aus Schriftarten verglichen. Der Buchstabe mit der meisten Ähnlichkeit wird dann übernommen.

Im übernommenen Text werden dann die einzelnen Wörter in einem Wörterbuch oder mit einer Trainingsdatei überprüft. Diese Überprüfung kann auch angepasst werden nur Buchstaben oder nur Zahlen zu erkennen.

## Text to speech (TTS)

Um den gespeicherten Text von der *tess-two* Engine auditiv zurück zu geben wurde die Sprachsynthese TextToSpeech (TTS) verwendet. TTS gehört zu einer in Android standardmäßig eingebundene Bibliothek seit API Level 4. Eine Sprachsynthese verwendet Sprachaufnahmen die in Elementen aufgeteilt sind. Aufsteigend werden einzelne Buchstaben, über Wörter, zu ganzen Sätzen bis hin zu ganzen Satzpassagen in Elemente gespeichert die dann Modular von der TTS Engine, mit einer Textvorlage zusammengestellt werden. Hierbei wird auf die Tonlage und den Takt der Stimme geachtet. Wenn die Engine seine Arbeit geleistet hat, gibt es eine Audio Datei zurück.

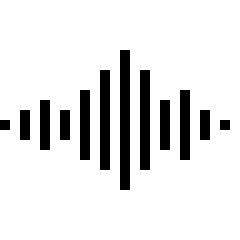


Abbildung 13: TTS

## Speech to text (STT)

Um eine Barriere lose Bedienung zu erreichen wurde eine Speech to Text (STT) Engine genutzt. Mit Hilfe der *SpeechRecognizer* Klasse, die in Android standardmäßig seit API Level 8 eingebunden ist, wird das gesprochene Wort von einem externen Server (Google) interpretiert und als eine Liste möglicher Strings zurückgegeben.

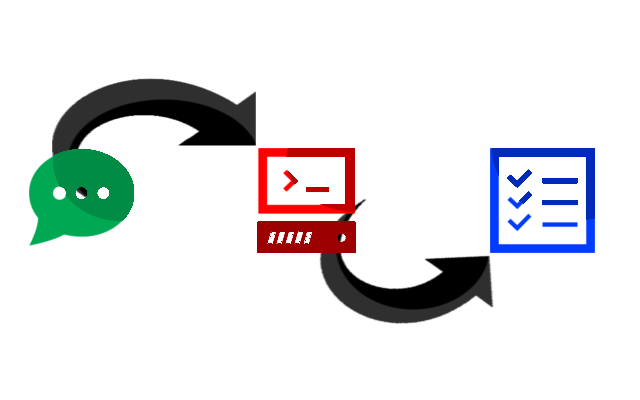


Abbildung 14: STT

Moderne Spracherkennungen nutzten Algorithmen, die sowohl die Wörter als auch den Kontext des gesprochenen Satzes bei der Interpretation berücksichtigten. Da es sich bei der Interpretation um aufwändige Kalkulationen handelt, werden die meisten Spracherkennungen auf *Cloud*-Rechnern oder Servern verlagert. Dies ermöglicht eine schnelle Verarbeitung der Daten, soweit eine gute Internetverbindung gewährleistet ist. Allerdings erschweren große Datensätze diese Funktion im mobilen Netz.

Im Falle der TextToSpeech App wurde der *Google*-Server genutzt um die Sprache zu erkennen. Dabei wird von *Google* eine Liste an Wörtern oder Sätzen zurückgegeben, die nach ihrer Wahrscheinlichkeit sortiert sind. Um die Spracherkennung als Befehlseingabe zu nutzen wurde auf bestimmte Schlagwörter geachtet.

# Umsetzung

## Grafische Oberfläche

Zu aller Erst wurde die grafische Oberfläche erstellt. Als Vorlage dienten die zuvor erarbeiteten Mockups (siehe 6.2). Es wurde der in Android Studio enthaltene XML-Editor verwendet. Dabei wurden ConstraintLayouts verwendet um ein responsive Design zu erreichen. Dies funktioniert so, dass Elemente sich automatisch anpassen können, wenn zum Beispiel ein Gerät einen größeren Bildschirm hat als das wofür das Design ursprünglich entwickelt wurde. Dies ist mit dem im Editor enthaltenen Blaupausen erkennbar.

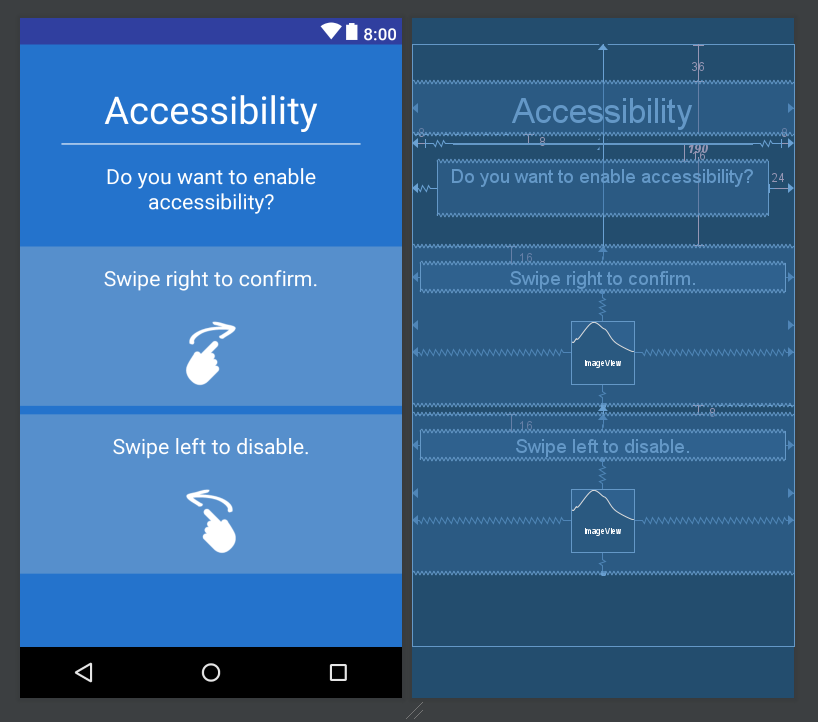


Abbildung 15: Screen Accessibility

An der rechten Seite des Bildes kann man erkennen, wie die einzelnen Elemente mit sich und den begrenzten Platz interagieren. Sie sind mit Pfeilen verbunden die entweder einen fixen Wert besitzen (durchgezogenen Pfeile) oder sich Variabel an den gegebenen Patz anpassen (gezackte Pfeile). Die verwendeten Elemente, für das gesamte Projekt, begrenzen sich auf die von Android Studio standartmäßig enthaltenen Widgets. Lediglich die Icons mussten eingebunden werden. Da die unterstützte Android Version 5.0 keine Unterstützung für veränderbaren Schriftarten besaß entschied man sich auf die Standardschriftart des Betriebssystems.

Weitere Punkte die beachtet wurden waren ein nutzbares Querformat und Unterstützung für Tabletts. Da alle Arten und Bildschirmgrößen der mobilen Geräte berücksichtigt werden, wurden drei verschiedene Layout-Arten verwendet:

**Layout**: Für Smartphones bis zu Bildschirmgrößen von 10 Zoll im Hochkant Modus.

**Layout-Land**: Für Smartphones bis zu Bildschirmgrößen von 10 Zoll im Querformat.

**Layout-XLarge**: Für Tabletts mit einer Bildschirmgröße von über 6 Zoll sowohl im Querformat als auch im Hochkant Modus.

Bei Letzterem sind die Maße des Bildschirms so groß, dass die Elemente nicht verschoben werden mussten um eine ausreichende Lesbarkeit zu gewähren.

Darüber hinaus wurde darauf geachtet, mit der kleinsten Bildschirmgröße des jeweiligen Layouts zu beginnen, da man andernfalls das Risiko eingeht, dass wichtige Elemente von dem begrenzten Platz kleinerer Bildschirme, teilweise oder gänzlich abgeschnitten werden.



Abbildung 16: Landscape Hochformat

Abbildung 17: Landscape Querformat

## Architektur

Als Architektur wurde das MVC Pattern verwendet. Hierzu wurde das Tesseract Objekt in eine Control-Klasse implementiert. In dieser ist die komplette Logik von Tesseract implementiert, zu diesen gehören das initialisieren von Tesseract mit der festgelegten Sprache, das Bereitstellen der Sprachdateien und das Erkennen von Texten aus Bildern.

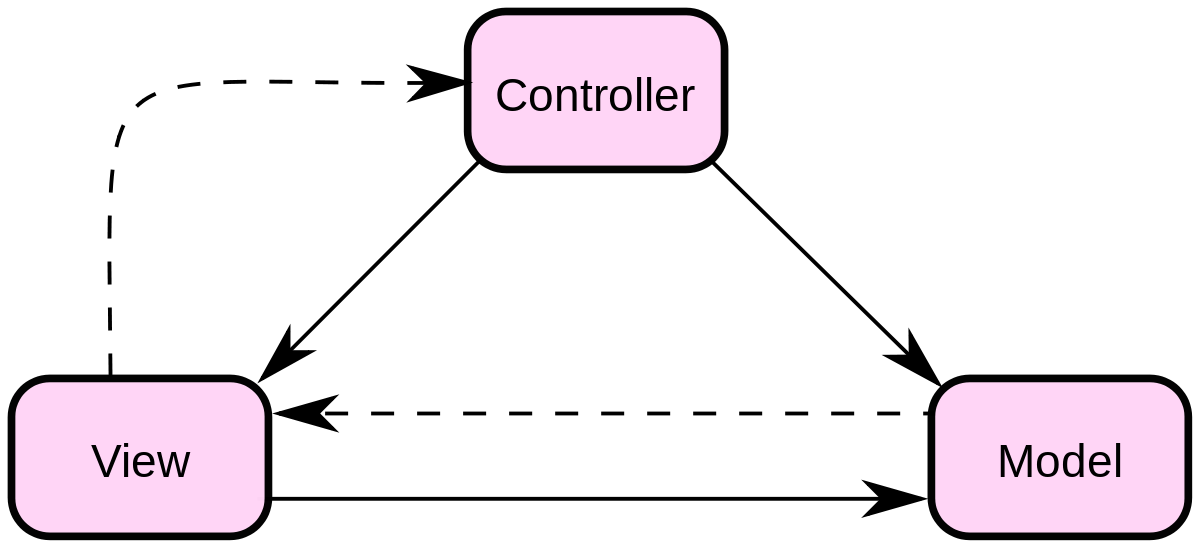


Abbildung 18:MVC Architektur

Von der View, also der Activity wird der Pfad des ausgewählten Bildes bereitgestellt und in einer Datenklasse (Model) abgelegt. Auf das Model Element wird vom Control Objekt automatisch zugegriffen und zur Erkennung des Textes verwendet.

Am Ende wird von der „Text Ausgabe“ Activity der von Tesseract erkannte Text sowohl visuell und auditiv ausgegeben.

### Allgemein

Es wurde mit einer Vererbungshierarchie entwickelt, hierfür wurden alle sich wiederholden Funktionalitäten in eine Klasse („DeviceInteraction“) ausgelagert.

Abbildung 19: Anwendung Architektur

Im genauen sind in dieser Klasse die Funktionalitäten, die Swipe Gesten, die Logik für das doppelte Klicken des Zurück Button und natürlich auch die Sprachausgabe und Sprachsteuerung. Diese Architektur hat den Vorteil, dass der Code nicht an allzu vielen Stellen wiederholt wird, dadurch ist es einfacher Änderungen vorzunehmen oder auch Fehler zu korrigieren. Außerdem ist der Code der Anwendung hiermit übersichtlicher, einfacher zu verstehen und auch schneller.

## Einbindung

### OCR

Wie in 6.1 beschrieben wurde eine Gabelung von *Tesseract Tool for Android* Namens *tess-two* für die Texterkennung (OCR) verwendet. Um die *tess-two* Engine in der App zu verwenden wurde als erstes eine Abhängigkeit zur aktuellen Version der Engine definiert. Für eine höhere Genauigkeit wurden auch Trainingsdaten für die deutsche und englische Sprache verwendet.

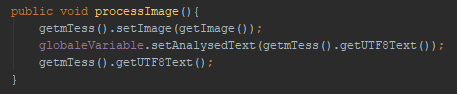
Zu Beginn wurde mit Bildern gearbeitet die als Ressource in der App gespeichert wurde, dies wurde im Verlauf der Entwicklung geändert. Nun kann der Benutzer aus einer Galerie wählen oder ein eigenes Foto schießen. Das geschossene Bild wird dann in der Galerie gespeichert und darauf zugegriffen. In der App wird das gewählte Bild zwischengespeichert und in eine Bitmap umgewandelt.

Zur Initialisierung der tess-two Engine werden die gewählte Sprache und der Dateipfad der Trainingsdatei benötigt. Die Sprache wurde über eine globale Variable vergeben, die berücksichtigt, dass die in der App gewählte Sprache verwendet wird. Die Initialisierung erfolgt automatisch beim Erreichen der Bildauswahl.

Die Bildauswahl wurde so eingestellt, dass nach Bestätigung, ob durch die Auswahl in der Galerie als auch beim Aufnehmen eines Fotos, zwei Methoden (setImage() und processImage()) aufgerufen werden, die das Bild umgewandelt und in die tess-two Engine einsetzt. Das Ergebnis der API wird daraufhin in eine globale Variable gespeichert. Die Variable wird zur Ausgabe genutzt.



„setImage“ nutzt die „BitmapFactory“-Methode um die im Dateipfad enthaltene Bilddatei zu einer Bitmap zu wandeln und in der Klassenvariablen „image“ zu speichern.



„image“ wird daraufhin von der Methode „processImage“ in die Tesseract API eingefügt um einen String an die globale Variable „analysedText“ senden. Dies wurde durch die „getUTF8Text“-Methode der tess-two API erreicht.

### TTS

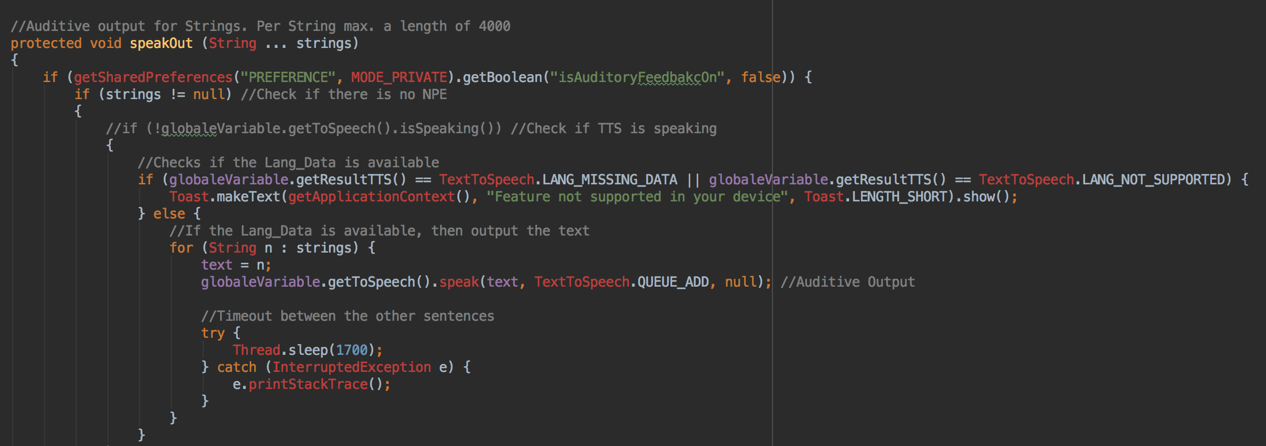
Die Text to Speech Funktion kann per Android Library importiert werden und wurde in der Vaterklasse „DeviceInteraction“ implementiert. Zu Beginn einer jeden Methode finden wir eine if-Abfrage, welche prüft ob es bereits einen Wahrheitswert für die Variable „isAuditoryFeedbackOn“ gibt und falls nicht, den Wert *false* zuweist.

../Desktop/Bildschirmfoto%202018-01-14%20um%2012.25.24.png

Die *Activity* eigenen Strings werden in den Methoden „speakOutOnStart“ und „speakOut“ verarbeitet. In ersterer wird ein Handler initialisiert, welcher dazu dient, die TTS engine zu verzögern. In der zweiten wird abgefragt, ob die gewählte Sprache überhaupt unterstützt wird und falls nicht wird einer kleiner *Toast* ausgegeben, welcher dem Benutzer mitteilt, dass die gewünschte Sprache nicht unterstützt wird.

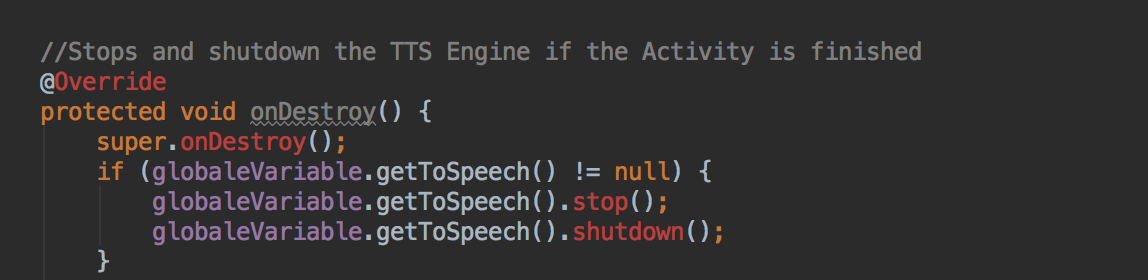
Falls die gewünschte Sprache unterstützt wird, wird der Text auditiv ausgegeben.

Mit dem Aufruf „Thread.sleep(1700)“ wird ein Timeout provoziert, um die Sätze zu trennen und somit einen verständlichen Ablauf der Texte zu konstruieren.



Selbiges passiert auch in den Methoden „speakOutOnStartOutput“ und „speakOutOutput“.

Nachdem die *Activity* beendet wurde, wird auch die TTS engine gestoppt und beendet.

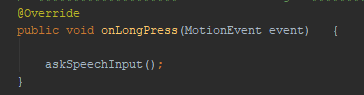


Die Methode „onPause“ pausiert die TTS engine in dem Fall, dass eine neue *Activity* gestartet werden sollte.

Ebenso kann die TTS engine pausiert werden, indem der Benutzer auf den Bildschirm tippt. Dafür wurde ein onTouchEvent implementiert, welches auf das Tippen des Benutzers reagiert.

### STT

In Android Studio ist die Spracherkennungs-API bereits standartmäßig eingebunden. Die Funktionalität der Spracherkennung wurde über die Vater-Klasse „DeviceInteraction“ implementiert. In der Vaterklasse wurde ein *MotionEvent-Listener* eingefügt, der bei jeder aktuellen *Activity* darauf achtet ob ein Nutzer eine bestimmte Geste nutzt. Im Falle der Spracherkennung wird auf das lange drücken auf dem Bildschirm geachtet.



In der Methode „askSpeechInput“ wird ein Intent ausgeführt, der eine neue Activity in der App öffnet. Diese Activity nutzt die Speechrecognition API der Google Server um die, wenn erfolgreiche, Spracheingabe des Nutzers zu verarbeiten. Das Ergebnis ist eine Array Liste in der die interpretierte Spracheingabe als String gespeichert sind. Vom Wahrscheinlichsten bis zum Unwahrscheinlichsten.



Das Resultat wird dann von der Methode „doTaskFromResult“ genutzt. In der Methode wurde eine einfache While-Schleife genutzt, die den Inhalt der Array Liste nutzt um zu erkennen ob ein Stichwort genutzt wurde. Wenn eines der Stichwörter erkannt wird, wird ein Intent mit der entsprechenden Activity gestartet. Hierbei wurde auch darauf geachtet, dass durch die Platzierung der if-Abfragen eine gewisse Priorität entsteht. Wie man am oberen Teilcode erkennen kann, wird bei einem Satz wie: „Ich benötige Hilfe beim Scannen“, die „Hilfe“-Eingabe als erstes erkannt. Die While-Schleife läuft so lange weiter, bis vier Strings abgefragt wurden, eine If-Abfrage ein Intent startet oder kein weiteres Element in der Array Liste verfügbar ist.

Die genutzten Stichwörter und dessen Aktion wurden hier festgehalten:

|  |  |
| --- | --- |
| **Stichwort Tabelle der TextToSpeech App abfallend nach ihrer Priorität** | |
| „…hilfe…“ | Startet die Guide Activity |
| „…scannen…“ | Startet den Kamera Modus |
| „…kamera…“ | Startet den Kamera Modus |
| „…galerie…“ | Startet die Bildauswahl |
| „…schließ…“ | Schließt die App |
| „…einstell…“ | Öffnet die App-Einstellung |
| „…sprache…“ | Öffnet die Spracheinstellungen der App |

## Sprachen

Wie bereits oben erwähnt, sollte die App in verschiedenen Sprachen anwendbar sein und am besten noch mehr Sprachen erkennen und übersetzen können. Bisher wurde sich auf zwei Sprachen beschränkt. Zum einen deutsch, zum anderen natürlich englisch, da es die meist-gesprochene Sprache der Welt ist. Weitere Sprachen sollen eingebunden werden genauso wie weitere Alphabete und Schriftzeichen wie z.B. chinesisch, japanisch oder griechisch, um die App für Touristen attraktiv zu machen.

# Ausblick

Die „TextToSpeech“ App bietet noch viele Möglichkeiten der Erweiterung. Zu einem könnte man eine eigene Kamera Applikation implementieren und diese in der „TextToSpeech“ Applikation einbinden, hiermit kann man die Funktionalität abbilden, dass man die Kamera nur noch auf einen Text hält und diesen nicht fotografieren muss. Eine weitere Erweiterung ist, dass der erkannte Text auch in eine gewünschte Sprache übersetzt und ausgegeben wird. Dies ist eine Funktionalität von der die Zielgruppe Touristen sehr profitieren würden. Ein weiterer wichtiger und nicht zu vernachlässigender Aspekt ist die Usability der Anwendung, diese sollte natürlich immer nach den Wünschen der Nutzer angepasst und verbessert werden. Hierzu sollte die App an Nutzer aus jeweils allen Zielgruppen verteilt werden, diese können dann Feedback geben. Mit dem Feedback kann man Erfahren ob die Bedienung den Zielgruppen entsprechend einfach und verständlich ist und ob man noch Verbesserungen in irgendeiner Art und Weise vornehmen kann. Um natürlich so viele Nutzer wie möglich zu erreichen soll die Anwendung auch in den Google Play Store zur Verfügung gestellt werden. Hierfür müssen die Richtlinien und Regeln die gelten um Android Anwendungen in den Google Play Store zu stellen noch überprüft und angewendet werden. Im gleichen Zuge um mehr Nutzer zu erreichen ist eine Portierung auf iOS Plattform geplant, da die Geräte die iOS benutzen auch einen großen Marktanteil haben.

# Fazit

Viel war geplant und fast genauso viel wurde am Ende auch umgesetzt. Erwartet wurde eine Applikation, welche Texte in Echtzeit per Kamera erkennt, verarbeitet, die Sprache erkennt, den Text zurückgibt und diesen dann noch auditiv ausgibt. Umgesetzt wurde das insofern, dass ein Bild per Geräte eigener Kamera geschossen werden kann und dieser dann verarbeitet wird und auditiv ausgegeben. Die Erkennung der Textsprache wurde nicht umgesetzt, genauso wenig wie das Übersetzen jenes Textes in eine andere Sprache. Weiterhin wurde eine einfache Handhabung gewünscht, welche durch einfache Wisch-Gesten umgesetzt wurde. Auch wurde die geforderte Spracheingabe umgesetzt und kann auch problemlos verwendet werden.

Es wurde also eine funktionierende App entwickelt, welche Menschen mit Sehschwäche Helfen kann und deren Alltag erleichtert.

Wir konnten im gesamten Projekt viel mitnehmen. Angefangen bei der Anwendung der agilen Entwicklungsmethode Scrum, was uns im kommenden Praxissemester bestimmt wiederbegegnen wird, bis hin zur Entwicklung von Smartphone Apps mit Android Studio, was in dieser Zeit auch sehr gefragt ist.

# Quellenverzeichnis

Kanban Board: Screenshot KanbanFlow

Yusuf: https://pixabay.com/de/fl%C3%BCchtling-portr%C3%A4t-mann-alte-1532326/

Tayo: https://pixabay.com/de/menschen-v%C3%B6lker-obdachlose-m%C3%A4nnlich-844212/

Sabine: <https://pixabay.com/de/buch-m%C3%A4dchen-lesen-bildung-urlaub-1524956/>

Use Case App: Erstellt

Informationsarchitektur: Erstellt

Farbpalette: Erstellt

Touchevent: <https://www.iconfinder.com/icons/446301/finger_gesture_hand_interactive_scroll_swipe_tap_icon#size=128>

Mockup Anwendung: Erstellt

Logo: Erstellt

Icon: Erstellt

OCR Engine: Erstellt

TTS: Erstellt

STT: Erstellt

Screen Accessibility: Erstellt

Landscape Hochformat: Erstellt

Landscape Querformat: Erstellt

MVC Architektur: https://commons.wikimedia.org/wiki/File:ModelViewControllerDiagram2.svg#/media/File:ModelViewControllerDiagram2.svg

Anwendung Architektur: Erstellt