Mobile App für Foto-Überweisungen

Inhalt

[1 Einleitung 4](#_Toc514412918)

[2 Entwicklung Mobiler Applikationen 4](#_Toc514412919)

[2.1 Anforderungsmodellierung 4](#_Toc514412920)

[2.2 Modellierung 5](#_Toc514412921)

[2.3 Testen 5](#_Toc514412922)

[3 Mobile Development 7](#_Toc514412923)

[3.1 Untersuchungskriterien 7](#_Toc514412924)

[3.2 Java 8](#_Toc514412925)

[3.3 Kotlin 8](#_Toc514412926)

[3.4 Android NDK 9](#_Toc514412927)

[3.5 Xamarin Android 9](#_Toc514412928)

[3.6 Corona 10](#_Toc514412929)

[3.7 PhoneGap 10](#_Toc514412930)

[3.8 Auswahl 10](#_Toc514412931)

[4 Texterkennung 11](#_Toc514412932)

[4.1 Klassischer Ablauf 11](#_Toc514412933)

[4.2 Vorverarbeitung 13](#_Toc514412934)

[4.3 Bestimmen von Merkmalen 15](#_Toc514412935)

[4.4 Texterkennung mit maschinellem Lernen 15](#_Toc514412936)

[4.5 Auswahl einer OCR-Bibliothek 15](#_Toc514412937)

[4.5.1 Google Mobile Vision 16](#_Toc514412938)

[4.5.2 Tesseract 17](#_Toc514412939)

[5 Android Entwicklung 17](#_Toc514412940)

[5.1 Aktivitäten 17](#_Toc514412941)

[5.2 Fragmente 19](#_Toc514412942)

[5.3 Permission 22](#_Toc514412943)

[5.4 Intents 22](#_Toc514412944)

[5.5 Navigation 22](#_Toc514412945)

[5.5.1 Laterale Navigation 22](#_Toc514412946)

[5.5.2 Aufwärtsnavigation 23](#_Toc514412947)

[6 Implementierung 24](#_Toc514412948)

[6.1 Ladebildschirm 24](#_Toc514412949)

[6.2 Kamera 25](#_Toc514412950)

[6.3 Analyse 25](#_Toc514412951)

[6.4 Datenbank 25](#_Toc514412952)

[6.5 Archiv und Favoriten 26](#_Toc514412953)

[6.6 About 26](#_Toc514412954)

[6.7 Navigation 26](#_Toc514412955)

[7 Fazit und Ausblick 27](#_Toc514412956)

[8 Literaturverzeichnis 27](#_Toc514412957)

[9 Abbildungsverzeichnis 27](#_Toc514412958)

# Einleitung

Das Scannen von Dokumenten erfolgt meist als Bilddatei, was dazu führt, dass der digitalisierte Text weder kopiert, noch bearbeitet werden kann. Um dies zu ermöglichen kann der Text als Buchstaben erkannt und ausgegeben werden. Diese Methode der Texterkennung wird als Optical Character Recognition, kurz OCR, bezeichnet.

Moderne Banking Applikationen, beispielsweise die der Sparkasse, bieten die Möglichkeit an aus Fotos Überweisungsdaten wie IBAN, Betrag oder Verwendungszweck zu erkennen um den Überweisungsprozess zu vereinfachen.

Ziel dieses Projekts ist eine Applikation für Mobilgeräte zu entwickeln, die anhand eines Fotos Überweisungsdaten erkennt und diese auf dem Bildschirm ausgibt.

# Entwicklung Mobiler Applikationen

Der Entwicklungsprozess von Applikationen für mobile Geräte lehnt sich an die konventionelle Art des Softwareentwicklungsprozesses an, besitzt jedoch wichtige Unterschiede in beispielsweise Benutzeroberflächendesign oder der Testphase.

Der Aufbau des Projekts folgt dem V-Modell. In der Anforderungsspezifikation werden die Anforderungen des Kunden an die Software notiert. (SE Pressman, S.42)

(Bild) (SE Pressman S.43)

## Anforderungsmodellierung

Die Anforderungsmodellierung dient zur Kommunikation der Kundenanforderungen an den Softwareentwickler in Form von Text, sowie Diagrammen, sodass diese einfach zu verstehen sind. (SE Pressman, S.166)

Das V-Modell zeigt, dass es oft schwierig ist alle Anforderungen von Anfang an zu spezifizieren, weshalb durch verschiedene Testphasen die Anforderungen möglicherweise angepasst werden müssen. (SE Pressman, S.42)

Für dieses Projekt werden Szenariobasierte-, (Verhaltens-?) und Klassenmodelle verwendet, um die Anforderungen darzustellen.

Ein Beispiel für ein Szenariobasiertes Modell ist das Use-Case-Diagramm:

(unser UseCase)

Ein Beispiel für ein Verhaltensdiagramm ist das Aktivitätsdiagramm. Es soll die schon im Use-Case-Diagramm vorhandenen Informationen auf kompaktere Weise darstellen, indem es einen Interaktionsfluss zeigt. (SE Pressman, S.99f)

(unser Aktivitätsdiagramm)

Folgendes Klassendiagramm soll das Klassenmodell darstellen:

(unser Klassendiagramm)

## Modellierung

(Unsere Anforderungen)

## Testen

Das Testen einer Mobilen Applikation kann wie bei konventioneller Software durch Unit-, Integration- und Systemtests durchgeführt werden (SE Pressman, S.483). Die Unittests sollen möglichst automatisiert durchgeführt werden und Fehler in einzelnen Funktionen oder Modulen aufdecken (SE Pressman, S.473). Durch Integrationstest wird geprüft, ob einzelne Komponenten korrekt miteinander zusammenarbeiten. Beispielsweise ob die Kameraansicht der Applikation das aufgenommene Foto korrekt an das Texterkennungsmodul weiterleitet (SE Pressman, S.475f).

Da die Entwickler der Software unbewusst dazu tendieren Tests durchzuführen, die zeigen, dass die Software fehlerfrei funktioniert ist ein weiterer wichtiger Bestandteil die Applikation von Nutzern testen zu lassen und deren Feedback bezüglich Benutzerfreundlichkeit und Navigation in den Softwareentwicklungsprozess einfließen zu lassen.

Da mobile Geräte Eigenschaften aufweisen, die auf Desktop Computer nicht zutreffen, beispielsweise viele verschiedene Formfaktoren, unterschiedliche Betriebssystemversionen und limitierte Akkukapazität und Speicherplatz, ist es notwendig spezifische Tests in Bezug auf Gerätekompatibilität, Performance und Netzwerkverfügbarkeit durchzuführen. (SE Pressman, S.483)

Das Android Developers Team schlägt vor die Testphase in 70% Unit-, 20% Integrations- und 10% UI-Tests zu unterteilen, wie aus der „Testing Pyramid“ in Abbildung X zu erkennen ist. Weiterhin ist in der Grafik zu sehen, dass die die Tests nach oben hin komplexer werden, aber im Gegenzug auch repräsentativer für die einwandfreie Funktion der Applikation sind. (https://developer.android.com/training/testing/fundamentals)

(https://developer.android.com/images/training/testing/pyramid\_2x.png)

Abbildung - Testing Pyramide

# Mobile Development

Die erste Entscheidung die zur Entwicklung mobiler Applikationen getroffen werden muss, ist welche Geräte und Plattformen unterstützt werden sollen. Die in diesem Projekt zu erstellende Applikation, soll auf Smartphones und Tablets lauffähig sein. Aufgrund der limitierten verfügbaren Hardware, wird als Plattform nur Android API-Level XX und höher unterstützt, also Android 6, Marshmallow, und höher.

Um eine Applikation für die Android Plattform zu entwickeln werden zahlreiche Möglichkeiten geboten. Folgend sollen diverse bekannte und weniger bekannte Methoden analysiert und verglichen werden um eine Auswahl zu treffen.

## Untersuchungskriterien

Pressman definiert sieben Kriterien um eine für mobile Applikationsentwicklung geeignete Entwicklungsumgebung und Programmiersprache zu wählen.

1. Die Entwicklungsumgebung sollte Editierung, Projektmanagement, Debugging, Architekturdesign, Dokumentation und Unit-Testing unterstützen.
2. Die Möglichkeit externe (Third-party) Bibliotheken (API/SDK) einzubinden sollte gegeben sein um in nur einer Entwicklungsumgebung arbeiten zu können.
3. Der Quellcode sollte von der Entwicklungsumgebung für bestimmte Endgeräte optimiert werden.
4. Das Testen der Applikation soll in der Entwicklungsumgebung möglich sein.
5. ???
6. Extensive Dokumentation und Tutorials sollten frei verfügbar und einfach verständlich sein.
7. Die Entwicklungsumgebung verfügt über die Funktion Benutzeroberflächen grafisch anzeigen und bearbeiten zu können.

## Java

Die wahrscheinlich bekannteste Methode zur Entwicklung einer Android Applikation ist die Nutzung der Programmiersprache Java in der Entwicklungsumgebung Android Studio.

Android Studio unterstützt Code-Editierung mit autocompletion, zeigt also Vorschläge an, wie man ein angefangenes Wort zu Ende schreiben würde. Dies reduziert Fehler, da man nicht versehentlich einen Variablennamen falsch schreiben kann. Zudem sind Debugging und Unit-Test Features, sowie die Möglichkeit externe Bibliotheken einzubinden implementiert. Die Applikation kann extensiv in der Entwicklungsumgebung getestet und auf verschiedenen Endgeräten emuliert werden.

Aufgrund der großen Beliebtheit der Programmiersprache Java sind viele Features schon implementiert und gut dokumentiert. Es existieren also viele Tutorials zur Android Programmierung in Java.

## Kotlin

Am 17. Mai 2017 hat das Android Team Kotlin als offizielle Android Programmiersprache anerkannt (<https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html>). Kotlin Programmierung wird, wie auch Java, in Android Studio umgesetzt. Die Funktionalitäten der Entwicklungsumgebung bleiben daher gleich. Der Vorteil von Kotlin ist, dass man durch weniger Code, dasselbe Ergebnis wie mit Java erreichen kann und dabei der Code sogar sicherer gegen beispielsweise NullPointerExceptions ist. Kotlin erweitert die Funktionalitäten von Java ohne dessen Vorteile zu verlieren, da Java Code in Kotlin verwendbar ist. Da jedoch Kotlin in Relation zu Java recht neu ist, sind Dokumentationen und Tutorials nicht so extensiv verfügbar wie für Java. (https://developer.android.com/kotlin/index.html)

Abbildung - Kotlin Logo (https://pbs.twimg.com/profile\_images/699217734492647428/pCfEzr6L\_400x400.png)

## Android NDK

Obwohl Android auf dem Linux Kernel basiert, welcher hauptsächlich aus C-Code besteht, hat sich Java aufgrund der großen Beliebtheit als Programmiersprache für Android Applikationen durchgesetzt. Android Studio unterstützt jedoch durch das Android Native Development Kit (NDK) die Programmierung in C und C++. Dies kann zum einen hilfreich sein, wenn man versucht die Applikation so performant wie möglich zu gestalten, oder wenn man Bibliotheken verwenden will, die auf C oder C++ basieren. (https://developer.android.com/ndk/guides/index.html)

## Xamarin Android

Eine Grundlegend andere Art der Android Programmierung bietet Xamarin Android. Die Programmiersprache ist in diesem Fall C# und die Entwicklungsumgebung Visual Studio. Ähnlich zu Android Studio, bietet Visual Studio Tools zur Editierung mit intelligenter autocompletion und Debugging. Die Applikation kann ebenso in der Entwicklungsumgebung getestet und emuliert werden. Ein grafischer Editor zur für visuelle Elemente ist ebenso verfügbar. (https://docs.microsoft.com/de-de/xamarin/android/get-started/)

Abbildung 3 - Xamarin Logo (https://avatars2.githubusercontent.com/u/790012?s=200&v=4)

## Corona

Corona ist ein Framework zur Entwicklung von Applikationen, das auf der Skriptsprache Lua basiert. Der Texteditor ist frei wählbar und es ist möglich in Echtzeit auf einem emulierten Gerät zu sehen, wie sich die Applikation verhält. Zudem enthält Corona Performanceoptimisierungen bei der Compilierung und native Bibliotheken, ob Java, C oder C++, können jederzeit aufgerufen werden. (https://coronalabs.com/product/)

Abbildung 4 - Corona Labs Logo (https://coronalabs.com/wordpress/wp-content/uploads/2016/08/corona\_logo-150x150.png)

## PhoneGap

Ein weiteres Framework zur Entwicklung mobiler Applikationen ist PhoneGap, welches den Ansatz der Web-Entwicklung nutzt. Durch HTML, CSS und Javascript können Android Applikationen erstellt werden. Sollten native Bibkiotheken notwendig sein, können diese ebenso verwendet werden. (<https://phonegap.com/products/>)

Abbildung - PhoneGap Logo (https://pbs.twimg.com/profile\_images/596058283699347456/NgaEDjHt\_400x400.jpg)

## Auswahl

Aufgrund der Beschränkung auf Android Geräte zur Entwicklung der Applikation erscheint es sinnvoll eine Entwicklungsumgebung und Sprache zu wählen, die ausschließlich zur Applikationsentwicklung auf Android Betriebssystemen optimiert ist. Obwohl Xamarin eine ansprechende Option ist, wenn die zu entstehende Applikation auf verschiedenen Plattformen verfügbar sein soll, wird im Rahmen dieses Projektes Android Studio als Entwicklungsumgebung verwendet.

Die zur Auswahl stehenden Programmiersprachen grenzen sich somit auf Java, Kotlin und das Android NDK, welches sich aus C und C++ zusammensetzt, ein. Obwohl Kotlin vergleichsweise neu ist und daher im Vergleich zu älteren Programmiersprachen weniger Beispiele zur Verfügung stehen, besteht in der Nutzung von Kotlin kein Nachteil gegenüber zu Java, da die hundertprozentige Interoperabilität sicherstellt, dass jederzeit zu Java Code zurückgefallen werden kann. Android Studio bietet zudem die Möglichkeit automatisch Java Code in Kotlin Code umzuwandeln, was die Einsteigerfreundlichkeit erheblich steigert.

Aufgrund der wachsenden Popularität und der einfachen Implementierbarkeit soll daher Kotlin in Verbindung mit Android Studio die Entwicklungsgrundlage des Projektes bieten.

# Texterkennung

Das Ziel dieser Studienarbeit ist es eine Rechnung mit einem mobilen Endgerät zu erkennen und vorhandene Kontoinformationen auszulesen. Dafür muss ein Dokument zunächst digitalisiert vorliegen. Dies geschieht beim Mobiltelefon mithilfe einer eingebauten Kamera. Das Bild liegt als Matrix von Pixelinformationen vor, in dem für jeden Pixel ein Farbwert gespeichert ist. Für die Texterkennung wird diese Matrix weiterverarbeitet.

## Klassischer Ablauf

Zunächst wird der theoretische Ablauf einer klassischen Texterkennung an einem Bild untersucht. Dazu werden die fünf folgenden Schritte beschrieben: (OCR,Chapter,2 Page 6)

* Binarisierung
* Identifizierung der Form
* Layout-Analyse
* Vorverarbeitung
* Charaktererkennung

Nachdem ein Bild aufgenommen wurde, wird dieses zunächst binarisiert.

Ein farbiges Bild wird dafür in Graustufen umgewandelt, sodass ein Threshold -Algorithmus angewandt werden kann. Durch die Binarisierung werden Informationen entfernt, die für die Mustererkennung von Buchstaben nicht nötig sind. Die Funktion eines Threshold -Algorithmus soll an folgendem Beispiel erläutert werden. Es wird ein Schwellwert w = 127 festgelegt. Alle Grauwerte kleiner/gleich diesem Wert werden im Ausgangsbild zu einer 0, alle Werte größer werden zu einer 1. Mit der geschickten Wahl und Parametrisierung eines Threshold -Algorithmus kann das Bild für die Erkennung von Text optimiert werden. So kann beispielsweise eine unvorteilhafte Belichtung eliminiert werden. In Abbildung 6 sind das beschriebene Beispiel und Optimierungen des Threshold-Algorithmus dargestellt.



Abbildung 6 Thresholding (https://docs.opencv.org/3.4/d7/d4d/tutorial\_py\_thresholding.html)

Außerdem können weitere Algorithmen angewandt werden um Störungen wie Rauschen zu eliminieren und die Qualität der Zeichen zu verbessern(4.2 Vorverarbeitung).

Im nächsten Schritt wird das Layout eines Dokuments erkannt und eventuell einem Datenbankmodell zugeordnet (OCR,Kap2,Seite6). Dafür werden Struktur-Elemente wie Zeilenumbrüche, Trennlinien, Kästen oder ausfüllbare Felder erkannt und interpretiert. Ein Datenbankmodell kann es dann vereinfachen das Dokument zu verstehen und zu interpretieren. So kann schnell festgelegt werden, ob es sich zum Beispiel um eine bestimmte Art von Antrag oder Rechnung handelt. Dies hilft später bei Texterkennung auch bestimmte Felder auf Plausibilität zu prüfen, da der Kontext bekannt ist.

Abbildung Beispielhaftes Überweisungsformular

Verdeutlicht werden soll dies mit dem Beispiel aus Abbildung 7. Es gibt das Datenbankmodell für Überweisung, wodurch bekannt ist, dass im grünen Feld eine Kontonummer stehen muss.

Wird nun beispielsweise 1234567B9 erkannt, kann die Anwendung sofort schlussfolgern, dass ein Fehler vorliegt. Das B könnte dann durch eine 8 ersetzt werden, oder es könnte dem Anwender ein Fehler ausgegeben werden. So kann auch überprüft werden, ob in einem Email-Feld wirklich eine E-Mail-Adresse steht.

Die erkannten Layout-Informationen und der enthaltene Text werden dann entsprechend dem Datenbankmodell in Feldern und Tabellen gespeichert.

Für jeden logischen Abschnitt wird dann separiert die Texterkennung durchgeführt. Google Mobile Vision teilt den Text zum Beispiel in Blöcke Linien und Wörter auf.(https://developers.google.com/vision/android/text-overview)



Abbildung Textstruktur https://developers.google.com/vision/android/text-overview

Ein Abschnitt kann zunächst anhand der Formelemente, wie zum Beispiel begrenzende Linien ausgerichtet werden. Dafür wird untersucht, ob Linien ungerade sind und das Feld wird dementsprechend gedreht. Dann werden die Formelemente entfernt und der Text in einzelne Zeichen aufgeteilt

Für die möglichen Zeichen gibt es eine Datenbank mit Mustern. Diese Datenbank kann unterschiedliche Sprachen und Zeichensätze unterstützen. Jedes Zeichen wird dann Pixel für Pixel mit den Mustern verglichen und es wird eine Ähnlichkeit berechnet. Nach dem die einzelnen Zeichen erkannt wurden folgt meist die Datenvalidierung und Syntaxanalyse, was vorhin beschrieben wurde. Dafür werden zum Beispiel auch Wörterdatenbanken für unterschiedliche Sprachen eingesetzt um Erkennungsfehler zu eliminieren. So kann das erkannte Wort „8aum“ durch das Wort „Baum“ ersetzt werden, weil das Letztere in der Datenbank zu finden ist.

## Vorverarbeitung

In diesem Kapitel werden Methoden beschrieben um die Qualität eines Bildes vor der Texterkennung zu verbessern.

In der Computer-Bildverarbeitung gibt es Algorithmen für das Glätten und Verschwämmen von Bildern. Dadurch werden die Zeichen für die Maschine besser lesbar. Im Folgenden werden zwei Beispiele dafür vorgestellt. In Abbildung 9 wird ein Gaus-Filter zur Kantenglättung eingesetzt. Die im linken Bild im Schriftzug „OpenCV“ sehr deutlich erkennbaren Störungen werden dadurch rechts beinahe vollständig eliminiert.



Abbildung Gaussian-Blur (<https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>)

Mit dem Median-Verschwämmen aus Abbildung 10 kann das sogenannte Salz-und-Pfeffer-Rauschen gut verkleinert werden. Dazu werden die Werte einer bestimmten Umgebung mit dem Median-Wert ersetzt.



Abbildung Median-Blur https://docs.opencv.org/3.1.0/d4/d13/tutorial\_py\_filtering.html

Um den Text für die Maschine besser lesbar zu machen, können Neigungen korrigiert werden. So kann das gesamte Dokument anhand der Textlinien analysiert werden und ein Offset-Winkel errechnet werden mit dem korrigiert wird, wie in Abbildung 11 zu sehen.



Abbildung Eliminierung der Textneigung

## Bestimmen von Merkmalen

## Texterkennung mit maschinellem Lernen

## Auswahl einer OCR-Bibliothek

In diesem Kapitel soll beschrieben werden, welche Möglichkeiten es gibt optische Texterkennung auf einem mobilen Endgerät zu implementieren. Eine naheliegende Methode wäre es, die in den vorherigen Kapiteln beschrieben technischen Aspekte in einer eigenen Bibliothek zu implementieren. Dafür werden jedoch größere Entwicklungsressourcen benötigt, als sie für dieses Projekt gegeben sind. Um effektive Texterkennung zu ermöglichen, wird eine große Menge an Daten benötigt. Diese Daten setzten sich aus unterschiedlichen Zeichensätzen und unterschiedlichen Schriftarten zusammen. Derartige Daten zu sammeln ist zeitintensiv. Zum anderen wäre die Entwicklung der vorher beschrieben Algorithmen zeitaufwändig. Wenn dieses Risiko aufgenommen wird, kann in abschätzbarer Zeit eine OCR-Bibliothek entwickelt werden, diese wäre aber funktional begrenzt und ungenau. Fertige OCR-Bibliotheken haben dahingegen eine größere Funktionalität und eine höhere Genauigkeit, aufgrund der größeren Entwicklungszeit und -Ressourcen. Außerdem werden viele OCR-Bibliotheken aktiv weiterentwickelt, was einen weiteren Vorteil bietet. Zum anderen wäre die Entwicklung der vorher beschrieben Algorithmen zeitaufwändig. Wenn dieses Risiko aufgenommen wird, kann in abschätzbarer Zeit eine OCR-Bibliothek entwickelt werden, diese wäre aber funktional begrenzt und ungenau. Für kommerzielle Anwendungen werden meist Lizenzen von proprietäre SDKs eingekauft, da diese meist genauer sind. Eine Lizenz für die Aspire OCR SDK kostet aber zum Beispiel in der Lite-Version für Java schon 5000 Dollar pro Entwickler [2]. Aufgrund der hohen

Wie in den vorherigen Kapiteln beschrieben ist die Ausführung von Texterkennungs-Algorithmen rechenintensiv. Wenn die Texterkennung direkt auf den mobilen Endgeräten ausgeführt wird kann die begrenzte Rechenkraft ein Hindernis darstellen. Ein Weg um mögliche Performance-Probleme zu umgehen, wäre die Texterkennung in eine Cloud-Lösung auslagern. Die Anwendung würde also ein Foto aufnehmen und dann für die optische Charaktererkennung einen Webservice nutzen. Die Anwendung lädt also das aufgenommene Foto hoch, dieses wird auf Online-Servern von OCR-Software analysiert und der erkannte Text wird wieder heruntergeladen. Mit diesem Ansatz braucht die Anwendung jedoch eine aktive Internetverbindung. Da ein Bild hochgeladen werden muss, würde dies meist nur über eine aktive WLAN-Verbindung funktionieren, da sonst die Geschwindigkeit nicht ausreichend ist. Außerdem kann nicht garantiert werden, dass der OCR-Server immer erreichbar und nicht überlastet ist. Bei einer Offline-Lösung würde die Anwendung zwar mehr Rechenkraft benötigten, der Engpass über die Internetverbindung würde aber wegfallen.

### Google Mobile Vision

Google Mobile Vision ist ein Framework, welches bildverarbeitende Funktionen für mobile Endgeräte bereitstellt. Mit diesem Framework können Gesichter erkannt, Barcodes gescannt oder Text erkannt werden. Dieses Framework kann ohne aktive Internetverbindung genutzt werden und eignet sich für Echtzeit-Anwendungen(<https://android-developers.googleblog.com/2016/06/android-mobile-vision-text-api.html>)

Die Texterkennungs-API erkennt jede Sprache, die auf dem römischen Alphabet basiert. Der auf einem Bild erkannte Text wird in Blöcke, Linien und Worte aufgeteilt und bietet damit auch relative Positionsinformationen der erkannten Wörter, die später zur Dokumentanalyse genutzt werden können.

(<https://developers.google.com/vision/android/text-overview>)

TESTERGEBNISSE

Genauigkeit

Schnelligkeit

Repräsentation der Daten

### Tesseract

Tesseract ist eine OCR-Engine, die ursprünglich von HP entwickelt wurde und seit 2006 von Google weiterentwickelt wird [3]. Tesseract ist in C++ entwickelt, es gibt auf GitHub jedoch Bibliotheken für unterschiedliche Programmiersprachen, die ein Schnittstelle bieten. Tesseract kann über 100 unterschiedliche Sprachen erkennen und unterstützt eine Menge an Ausgabe- und Eingabeformaten. Mit einer Wertung von aktuell 17700 Stars ist Tesseract die beliebteste freie OCR-Bibliothek auf GitHub. Es gibt auf GitHub einen Fork des Tesseract-Repositories für Android (<https://github.com/rmtheis/tess-two>)

# Erkennung von Rechnungsdaten

## Aufbau einer Rechnung

## Reguläre Ausdrücke

## String-Ähnlichkeit

# Android Entwicklung

## Aktivitäten

Ein essentieller Bestandteil von Android Applikationen sind Aktivitäten. Android Applikationen werden nicht, wie bei Desktop Applikationen üblich, durch eine Main-Methode gestartet, sondern können auf viele verschiedene weisen aufgerufen werden, wofür eine Aktivität entsprechende Callback-Methoden zur Verfügung stellt. (<https://developer.android.com/guide/components/activities/intro-activities>)

Wenn man beispielsweise eine E-Mail-Applikation vom Home Screen aus öffnet, wird der Maileingang als Liste angezeigt. Eine weitere Möglichkeit dieselbe E-Mail-Applikation zu starten wäre zum Beispiel in einer Social-Media-Applikation eine E-Mail anzuklicken. Dies würde die E-Mail-Applikation direkt in „E-Mail verfassen“ starten.

Android Applikationen können durch den Aufruf anderer Applikationen in den Hintergrund geraten. Wenn sie dann wieder in den Vordergrund kommen wird dies als Hot Start bezeichnet. Wenn der Prozess einer Hintergrundapplikation beendet wird um für eine Vordergrundapplikation mehr Speicher freizugeben, dann wird der nächste Applikationsstart als Warm Start bezeichnet. (<https://developer.android.com/topic/performance/vitals/launch-time>)



Abbildung - Aktivitätslebenszyklus

(https://developer.android.com/guide/components/images/activity\_lifecycle.png)

## Fragmente

Ein Fragment ist ein modularer Teil einer Aktivität, welcher einen eigenen Lebenszyklus besistzt. Eine Aktivität kann somit aus mehreren Fragmenten aufgebaut sein und ein Fragment kann in mehreren Aktivitäten wiederverwendet werden. Der Lebenszyklus des Fragments ist zusätzlich noch vom Lebenszyklus der übergeordneten Aktivität abhängig. (https://developer.android.com/guide/components/fragments)



Abbildung - Fragmentlebenszyklus

(https://developer.android.com/images/fragment\_lifecycle.png)

## Permission

Fill with Permission

## Intents

Fill with Intent

## Navigation

Einer der ersten Schritte des Android Applikation Designs ist festzulegen auf welche Daten der Nutzer zugreifen kann und wie dieser von und zu bestimmten Inhalten navigieren kann. Dies führt zu einer Navigationshierarchie, durch die abgeleitet werden kann, welche Navigationselemente am besten an welcher Stelle benutzt werden. Hierzu sollen nun einige Begriffe erklärt werden, die im Rahmen der Implementierung verwendet werden um die Applikation zu beschreiben. (https://developer.android.com/training/design-navigation/)

### Laterale Navigation

Navigation zwischen Elementen auf der gleichen Hierarchieebene, wird als laterale Navigation bezeichnet. Wenn diese Art von Navigation in der höchsten Hierarchieebene umgesetzt werden soll, kann dies durch einen Navigation Drawer realisiert werden. Dieser nimmt in den meisten Fällen die komplette Bildschirmhöhe ein und wird in Leserichtung geöffnet. Ein Navigation Drawer ist aufgeteilt in einen optionalen Header und einer Liste an Zielen, welche wahlweise mit einem Icon versehen werden können. Weitere Formen der lateralen Navigation sind möglich, werden jedoch hier nicht weiter behandelt. (<https://material.io/design/navigation/understanding-navigation.html#lateral-navigation>)



Abbildung - Navigation Drawer Vorlage

(<https://material.io/design/assets/10UGo-uwu3HViW96viJ-EVtV2PSBAF99c/anatomy.png>)

### Aufwärtsnavigation

Es gibt grundlegend zwei Möglichkeiten für den Nutzer in einer Navigation chronologisch aufwärts zu navigieren, zum einen den von Android zur Verfügung gestellten „Back button“ in der Navigationsleiste, zum anderen den „Up button“, der in der Applikationsleiste zu finden ist. (https://developer.android.com/training/design-navigation/ancestral-temporal)



Abbildung 15 - Back button und Up button

(https://material.io/design/assets/1NU5KbnRPZLngt5GnK45eUOaMtyysc5mn/reverse-chronological.png)

# Implementierung

Durch die Implementierungsphase wird die definierte Spezifikation in die Praxis umgesetzt. Dabei ist aufgrund des gewählten inkrementellen Entwicklungsmodells darauf zu achten, dass die Spezifikation durch die Implementierung verändert und verbessert werden kann. (Sommerville S.38)  
Bild (Sommerville S.63)

## Ladebildschirm

Lange Ladezeiten beim Applikationsstart ohne Feedback für den Nutzer, sind für diesen verwirrend, da er sich fragen muss ob die Applikation ordnungsgemäß funktioniert. Dies kann zum einen durch viel overhead in der onCreate Methode der MainActivity oder auch durch ein Gerät mit langsamer Hardware verursacht werden.

(Abbildung selber Link)

Ein individueller Ladebildschirm kann hierbei Abhilfe verschaffen, indem dem Nutzer signalisiert wird, dass die Applikation gestartet wurde aber noch Zeit zum Laden braucht. (<https://developer.android.com/topic/performance/vitals/launch-time>)

Abbildung unser Ladebildschirm

## Kamera

Nachdem die Applikation geladen ist, startet sie in der Kameraansicht, von der aus ein Bild zur Analyse aufgenommen oder aus der Galerie ausgewählt werden kann.

Bild von Kameraansicht und Galerieauswahl

Die Bildaufnahme erfolgt durch Tippen auf den weißen Ring. Auch hier wird dem Nutzer beim Tippen durch eine Animation Feedback geliefert, womit signalisiert wird, dass die Schaltfläche funktioniert. (Animation kommt, wenn der Nutzer sein Daumen auf dem Button lässt)

(vllt bild vorher nachher?)

## Analyse

Sobald ein Bild für die Analyse aufgenommen oder ausgewählt wurde, wird die Texterkennung der Bankdaten ausgeführt und der Analysebildschirm geöffnet. Hier kann der Nutzer einsehen welche Bankdaten erkannt wurden und diese gegebenenfalls modifizieren. Sobald der Nutzer zufrieden mit der erkannten Rechnung ist, kann er diese in die Datenbank speichern. Diese Aktion führt den Nutzer zur Archivansicht.

Sollte der Nutzer mit dem Bild oder dem erkannten Text nicht zufrieden sein, kann der Eintrag durch einen Zurück-Befehl verworfen werden.

## Datenbank

Bitte mit Room Database Umsetzung füllen

## Archiv und Favoriten

Sowohl die Archivansicht, als auch die Favoritenansicht sind als Listen von Einträgen aufgebaut, die aus der Datenbank generiert werden. In der Archivansicht werden alle gespeicherten Einträge angezeigt, während in der Favoritenansicht nur die favorisierten Rechnungen angezeigt werden. Die Liste ist jeweils vom neuestem zu ältestem Eintrag chronologisch sortiert.

Jeder Eintrag zeigt ein kleines Vorschaubild sowie die wichtigsten Informationen auf einen Blick. Durch einen Stern in der unteren linken Ecke kann der Eintrag favorisiert beziehungsweise entfavorisiert werden. Einen Eintrag anzutippen öffnet dessen Detailansicht, die an die Analyseansicht erinnert.

Aus der Detailansicht kann der Eintrag favorisiert, entfavorisiert oder gelöscht werden. Zusätzlich kann der Bearbeitungsmodus aktiviert werden um den Eintrag zu modifizieren und zu speichern.

## About

In der Aboutansicht sind einige Informationen über die Applikation zu sehen, wie das Logo, die Authoren und die Version. Der Nutzer kann hier keine Aktion ausführen.

## Navigation

Die vier wichtigsten Ansichten der Applikation, Kamera, Archiv, Favoriten und About, sind über einen Navigation Drawer zu erreichen. Dieser kann entweder durch Tippen auf das Menüsymbol oder indem vom linken Rand nach rechts gewischt wird aufgerufen werden.

Ansichten die in der Hierarchie eine Ebene tiefer liegen haben keinen Zugriff zum Navigation Drawer. Statt dem Menüsymbol in der oberen linken Ecke befindet sich bei diesen Ansichten ein Up-Button, welcher zur jeweils vorherigen Ansicht zurückwechselt.

Der Back-Button verhält sich wenn eine Up-Navigation verfügbar ist wie der Up-Button. In allen anderen Fällen bringt der Back-Button die Kameraansicht hervor.

# Fazit und Ausblick

Wo gab es Probleme? Was hat gut geklappt, was könnte man besser machen? Was könnte in Zukunft noch umgesetzt werden?

# Literaturverzeichnis

[1] Buch, Author, Jahr – Titel (Abk.)

[2] <https://asprise.com/royalty-free-library/java-ocr-source-code-open-order.html>

[3] <https://github.com/tesseract-ocr/tesseract>

[4] <https://github.com/tmbdev/ocropy>

[5] <https://developers.google.com/vision/android/text-overview>

[6] <https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>

CHERIET, M. *Character Recognition Systems : A Guide for Students and Practitioners*. Hoboken, N.J. : Wiley-Interscience, 2007. ISBN: 9780471415701.

<https://pdfs.semanticscholar.org/74d6/68256131f379d63a3d484ccff513f5bbb6d3.pdf>

# Abbildungsverzeichnis

[Abbildung 1 - Testing Pyramide 6](#_Toc514412784)

[Abbildung 2 - Kotlin Logo (https://pbs.twimg.com/profile\_images/699217734492647428/pCfEzr6L\_400x400.png) 8](file:///C:\Users\gsansone\Desktop\Studium\Studienjahr3\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc514412785)

[Abbildung 3 - Xamarin Logo (https://avatars2.githubusercontent.com/u/790012?s=200&v=4) 9](file:///C:\Users\gsansone\Desktop\Studium\Studienjahr3\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc514412786)

[Abbildung 4 - Corona Labs Logo (https://coronalabs.com/wordpress/wp-content/uploads/2016/08/corona\_logo-150x150.png) 10](file:///C:\Users\gsansone\Desktop\Studium\Studienjahr3\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc514412787)

[Abbildung 5 - PhoneGap Logo (https://pbs.twimg.com/profile\_images/596058283699347456/NgaEDjHt\_400x400.jpg) 10](file:///C:\Users\gsansone\Desktop\Studium\Studienjahr3\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc514412788)

[Abbildung 6 - Aktivitätslebenszyklus 19](#_Toc514412789)

[Abbildung 7 - Fragmentlebenszyklus 21](#_Toc514412790)

[Abbildung 8 - Navigation Drawer Vorlage 23](#_Toc514412791)

[Abbildung 9 - Back button und Up button 24](#_Toc514412792)