Mobile App für Foto-Überweisungen

Inhalt

[1 Einleitung 5](#_Toc515451592)

[2 Entwicklung Mobiler Applikationen 6](#_Toc515451593)

[2.1 Anforderungsmodellierung 6](#_Toc515451594)

[2.2 Anforderungsspezifikation 10](#_Toc515451595)

[2.2.1 Funktional 10](#_Toc515451596)

[2.2.2 Nicht-Funktional 12](#_Toc515451597)

[2.3 Testen 12](#_Toc515451598)

[3 Mobile Development 14](#_Toc515451599)

[3.1 Untersuchungskriterien 14](#_Toc515451600)

[3.2 Java 15](#_Toc515451601)

[3.3 Kotlin 15](#_Toc515451602)

[3.4 Android NDK 16](#_Toc515451603)

[3.5 Xamarin Android 16](#_Toc515451604)

[3.6 Corona 16](#_Toc515451605)

[3.7 PhoneGap 17](#_Toc515451606)

[3.8 Auswahl 17](#_Toc515451607)

[4 Texterkennung 19](#_Toc515451608)

[4.1 Einleitung 19](#_Toc515451609)

[4.2 Funktionsweise 19](#_Toc515451610)

[4.2.1 Klassischer Ablauf 19](#_Toc515451611)

[4.2.2 Vorverarbeitung 23](#_Toc515451612)

[4.2.3 Bestimmen von Merkmalen 25](#_Toc515451613)

[4.2.4 Klassifizierung der Muster 27](#_Toc515451614)

[4.3 Auswahl einer OCR-Bibliothek 28](#_Toc515451615)

[4.3.1 Google Mobile Vision 30](#_Toc515451616)

[4.3.2 Tesseract 30](#_Toc515451617)

[4.4 Vergleich: Tesseract – Google Mobile Vision Text 31](#_Toc515451618)

[4.4.1 Kriterien 31](#_Toc515451619)

[4.4.2 Beispiel: Getränke-Rechnung 32](#_Toc515451620)

[4.4.3 Performance 35](#_Toc515451621)

[4.4.4 Datenrepräsentation 36](#_Toc515451622)

[4.4.5 Komplexität 37](#_Toc515451623)

[4.4.6 Genauigkeit 38](#_Toc515451624)

[4.4.7 Fazit 39](#_Toc515451625)

[5 Android Entwicklung 41](#_Toc515451626)

[5.1 Applikations-Manifest 41](#_Toc515451627)

[5.2 Aktivitäten 41](#_Toc515451628)

[5.3 Fragmente 43](#_Toc515451629)

[5.4 Berechtigungen 46](#_Toc515451630)

[5.5 Intents 46](#_Toc515451631)

[5.6 Navigation 47](#_Toc515451632)

[5.6.1 Laterale Navigation 47](#_Toc515451633)

[5.6.2 Aufwärtsnavigation 48](#_Toc515451634)

[6 Implementierung 50](#_Toc515451635)

[6.1 Aufbau der Applikation 51](#_Toc515451636)

[6.1.1 Kamera 52](#_Toc515451637)

[6.1.2 Analyse 52](#_Toc515451638)

[6.1.3 Archiv und Favoriten 52](#_Toc515451639)

[6.1.4 About 53](#_Toc515451640)

[6.2 Datenbank 53](#_Toc515451641)

[6.3 Navigation 55](#_Toc515451642)

[6.4 Filtern der Rechnungsdaten 56](#_Toc515451643)

[7 Fazit und Ausblick 60](#_Toc515451644)

[8 Literaturverzeichnis 61](#_Toc515451645)

[9 Abbildungsverzeichnis 63](#_Toc515451646)

# 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. Im folgenden Bericht wird der Entwicklungsprozess von der Modellierung, über die Aufarbeitung der technischen Grundlagen, bis zur Implementierung und dem entstandenen Ergebnis beschrieben.

# 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)

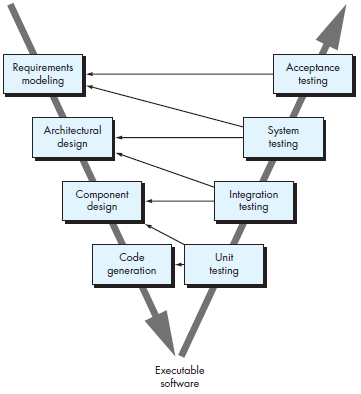


Abbildung 1 - V-Modell

(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)

Zu Beginn des Projektes wurde das Klassendiagramm aus Abbildung 2 erstellt um die Anwendungsstruktur zu modellieren.

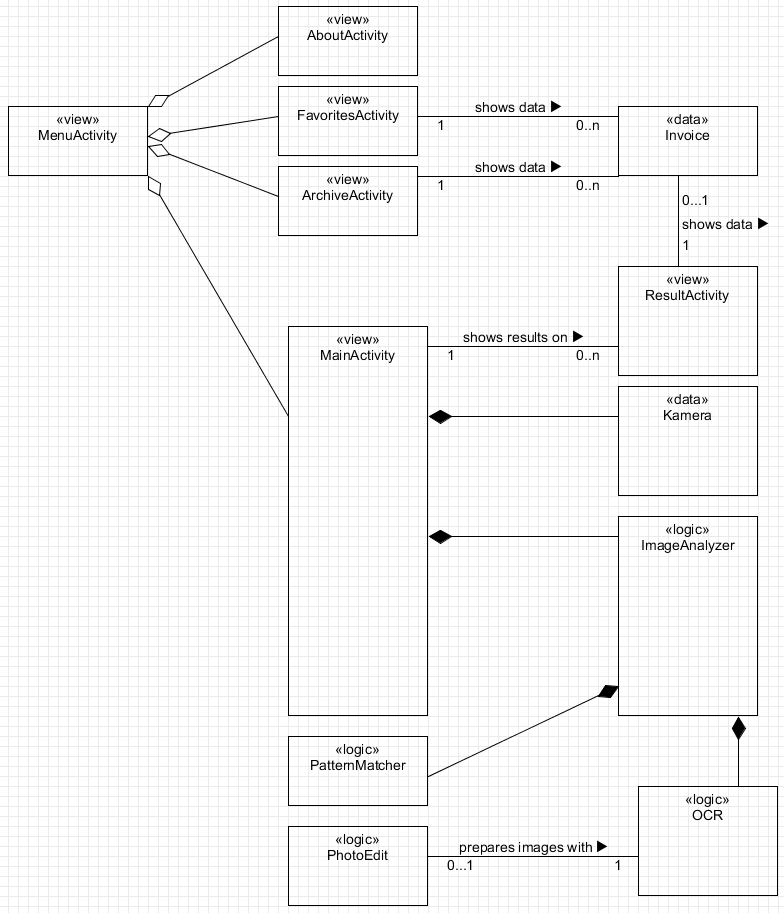


Abbildung 2 - Modellierung\_Klassen

Die Klassen sind in drei Stereotypen aufgeteilt, welche in nachfolgender Tabelle beschrieben werden.

|  |  |
| --- | --- |
| <<view>> | Dieser Stereotyp beschreibt Klassen, die mit dem Benutzer interagieren. Sie sollen nur vorhandene Daten visualisieren und Benutzereingaben verarbeiten. |
| <<data>> | Dieser Stereotyp beschreibt Klassen, die die Daten eines Objekts speichern |
| <<logic>> | Dieser Stereotyp beschreibt Klassen, die logische Aufgaben, wie Berechnungen, erfüllen |

Es gibt sechs <<view>>-Klassen. Diese spiegeln die sechs Ansichten wieder, die die Applikation haben soll. Das Menü bietet die Möglichkeit, zwischen vier der Ansichten zu wechseln und setzt sich somit aus den vier Aktivitäts-Klassen (vgl. 5.2 „Aktivitäten“) zusammen.

Aus dem Menü sind folgende Ansichten zu erreichen:

* About
* Archiv
* Favoriten
* Main

Die About-Aktivität stellt dem Benutzer allgemeine Informationen zur Applikation und den Entwicklern dar. Das Archiv stellt chronologisch alle gespeicherten Aufnahmen (Rechnungen) an. Diese Archiv-Aufnahmen lassen sich favorisieren und werden dann in der Favoriten-Aktivität angezeigt. Die Daten einer Aufnahme werden in der <<data>>-Klasse Invoice gespeichert. Die Detail-Ansicht einer Rechnung wird in der <<view>>-Klasse Result-Activity angezeigt. Die Main-Ansicht stellt die Kameraansicht dar. In dieser Ansicht können Aufnahmen gemacht werden, die dann analysiert und in der Result-Activity angezeigt werden. Die Kameradaten werden aus der <<data>>-Klasse Kamera geliefert. Außerdem benötigt die Main-Aktivität eine Instanz der <<logic>>-Klasse ImageAnalyzer, welche eine Aufnahme analysiert. Die Analyse setzt sich aus der OCR-Klasse und der PatternMatcher-Klasse zusammen. Die OCR-Klasse filtert den Text aus einer Aufnahme. Um eine Aufnahme vorher noch für OCR zu optimieren hat sie dafür eine Instanz der Klasse PhotoEdit. Die PatternMatcher-Klasse ist dafür verantwortlich aus dem erkannten Text, die konkret für eine Rechnung benötigten Daten zu filtern.

## Anforderungsspezifikation

Um nach der Implementierung gegen die Anforderungen zu testen, werden in diesem Kapitel die Funktionen spezifiziert, welche die Applikation später erfüllen soll. Dabei wird zwischen funktionalen und nicht-funktionalen Anforderungen entschieden.

### Funktional

Foto aufnehmen:

Über einen zentralen Button kann in der Kameraansicht ein Bild aufgenommen werden. Dieses Bild kann entweder auf Überweisungsdaten untersucht oder verworfen werden.

Text erkennen:

Ein aufgenommenes Bild kann mit Texterkennung analysiert werden. Dafür werden erst der rohe Text mit OCR-Methoden erkannt und dann mit String Analyse und Zuordnungsbibliotheken auf Überweisungsdaten untersucht.

Aufnahme speichern:

Eine Aufnahme soll nach der Analyse gespeichert oder verworfen werden können.

Menüleiste öffnen:

Es soll in jeder Ansicht der Applikation das Menu geöffnet werden können. Über das Menü können alle anderen Ansichten erreicht werden.

About-Informationen öffnen:

In der Menüansicht können die About-Informationen geöffnet werden. Hier sind Informationen zu den Entwicklern und der Version zu finden.

Archivansicht öffnen:

Über das Menü kann das Archiv geöffnet werden. Hier werden alle Aufnahmen in einer Liste gespeichert, bestehend aus Titel und Datum.

Favoritenansicht öffnen:

Über das Menü kann die Favoritenansicht angezeigt werden.

Kameraansicht öffnen:

Über das Menü kann die Favoritenansicht angezeigt werden.

Listenpunkt löschen:

Im Archiv können Aufnahmen gelöscht werden

Listenpunkt favorisieren/ entfavorisieren:

Eine Aufnahme/Archivlistenpunkt kann favorisiert werden oder entfavorisiert werden.

Listenpunkt bearbeiten:

Die erkannten Daten eines Listenpunktes, wie zum Beispiel Titel oder IBAN, können bearbeitet werden, sollten Fehler bei der Erkennung aufgetreten sein.

Aufnahme eines Listenpunkts vergrößern:

Eine Aufnahme soll auf Vollbild vergrößert werden können.

Galeriebild öffnen:

Statt eine neue Aufnahme zu machen, soll eine Aufnahme auch aus der Galerie geladen werden können.

Aufnahme in externen Speicher speichern:

Die Aufnahmen der Applikation sollen in einem eigenen Ordner in der Galerie gespeichert werden.

### Nicht-Funktional

Datenbank

Die Datenbank soll keine redundanten Daten enthalten. (3.te Normalform)

Performance

Die Reaktionszeiten der interaktiven Elemente sollen unter zwei Sekunden liegen.

Usability

Die Anwendung soll überschaubar sein und intuitiv sein.

Es soll eine flache Hierarchie implementiert werden.

## 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 3 - 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 4 - 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 5 - 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 6 - 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 7 - 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

## Einleitung

In diesem Kapitel wird zunächst die Funktionsweise von optischer Charaktererkennung erklärt werden um ein grundlegendes Verständnis zu erlangen. Daraufhin werden die Möglichkeiten beschrieben, wie OCR im Projekt implementiert werden kann. Zuletzt wird erörtert, was der beste Weg ist OCR für die Zwecke einer Rechnungserkennung auf einem mobilen Endgerät einzusetzen. Dafür werden unterschiedliche Bibliotheken implementiert und mit verschiedenen Daten getestet.

## Funktionsweise

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: (Cheriet, 2007, S. 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 v = 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 mit dem Adaptive Mean Thresholding oder dem Adaptive Gaussian Thresholding eine unvorteilhafte Belichtung eliminiert werden, da hier auch die Umgebung der Pixel beachtet wird, statt nur der Pixelwerte. Diese drei Beispiele sind in Abbildung 8 dargestellt.



Abbildung 8 - Thresholding (OpenCV-Dokumentation, Image Thresholding, 2018)

Außerdem können weitere Algorithmen angewandt werden um Störungen wie Rauschen zu eliminieren und die Qualität der Zeichen zu verbessern (vgl. 4.2.2 „Vorverarbeitung“).

Im nächsten Schritt wird das Layout eines Dokuments erkannt und kann einem Modell aus einer Datenbank zugeordnet werden (Cheriet, 2007, S. 6). Zu diesem Zweck werden Struktur-Elemente wie Zeilenumbrüche, Trennlinien, Kästen oder ausfüllbare Felder erkannt und interpretiert. Ein Modell aus einer Datenbank kann dabei helfen 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 9 - Beispielhaftes Überweisungsformular

Verdeutlicht werden soll dies mit dem Beispiel aus Abbildung 9. Es gibt das Modell 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, da eine Kontonummer nur aus zahlen besteht. Das B kann durch eine 8 ersetzt werden, oder es kann dem Anwender ein Fehler ausgegeben werden. Mithilfe des Modells kann auch sichergestellt werden, dass 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 (Google-Developers, 2017).



Abbildung 10 - Textstruktur GMV (Google-Developers, 2017)

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 konventionelle 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, da kleine Lücken gefüllt und kleine Störungen entfernt werden. (Cheriet, 2007, S. 30). Im Folgenden werden zwei Beispiele dafür vorgestellt. In Abbildung 11 wird ein Gaus-Filter zur Kantenglättung eingesetzt. Die im linken Bild im Schriftzug „OpenCV“ sehr deutlich erkennbaren Erhebungen an den Buchstabenrändern sind rechts beinahe vollständig eliminiert.



Abbildung 11 - Gaussian-Blur (OpenCV-Dokumentation, Smoothing Images, 2015)

Mit dem Median-Verschwämmen aus Abbildung 12 kann das sogenannte Salz-und-Pfeffer-Rauschen gut verkleinert werden. Dazu werden die Werte einer bestimmten Umgebung mit dem Median-Wert ersetzt.



Abbildung 12 - Median-Blur (OpenCV-Dokumentation, Smoothing Images, 2015)

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 13 zu sehen. Dies ist wichtig bei eingescannten Dokumenten, da diese oft nicht genau rechtwinklig auf den Scanner gelegt werden. Auch bei der Aufnahme von Fotos mit einer Handykamera treten meist Neigungen des gesamten Dokuments auf, die korrigiert werden sollen.



Abbildung 13 Eliminierung der Textneigung (Cheriet, 2007, S. 33)

Bei handgeschriebenen Texten kann für jedes Zeichen separat die Neigung korrigiert werden. Für gedruckte Texte ist dies jedoch nicht nötig, da die Textlinien gerade sind.

Der wichtigste Vorverarbeitungs-Prozess in der Texterkennung ist die Charakter-Normalisierung (Cheriet, 2007, S. 36). Das Ziel der Charakter-Normalisierung ist es alle Zeichen zu vereinheitlichen. Alle Zeichen eines Textes müssen auf ein quadratisches Feld mit einer festgelegten Größe (z.Bsp 32x32) abgebildet werden. Dafür muss ein Zeichen solange vergrößert oder verkleinert werden bis eine Seite gleich wie die vordefinierte Größe ist und das Zeichen vollständig im Feld enthalten ist. Das Seitenverhältnis eines Zeichens ist ein Merkmal, das für die Zeichenerkennung nützlich ist. Es können außerdem die Momente eines Zeichens normalisiert werden. Ein Moment ist ein Merkmal, welches die Lage und Orientierung eines Objektes beschreibt (Herbert Süße, 2014, S. 515).

### Bestimmen von Merkmalen

Nach der Optimierung der Zeichen werden Merkmale bestimmt, die relevant für die Klassifizierung sind. Unter Klassifizierung versteht man die Zuordnung eines Zeichens zu einem Buchstaben (Cheriet, 2007, S. 129). Dazu gehören die Aufgaben Merkmale zu bestimmen, die relevanten Merkmale zu filtern und das beste Merkmal für eine gegebene Klassifizierungsaufgabe zu finden (Cheriet, 2007, S. 54) . Im folgenden Kapitel sollen Merkmale beschrieben werden um ein Verständnis zu gewinnen, wie die Texterkennung funktioniert.

Ein oft genutztes Merkmal um 2D-Bilder auf Muster zu untersuchen sind Momente (Cheriet, 2007, S. 55). Momente beschreiben Eigenschaften, wie die Fläche einer Kontur, den Schwerpunkt und die Ausrichtung eines Bildes oder die Trägheit. Momente sind gewichtete Mittelwerte, die sich aus den Intensitäten der Bildpunkte und einer Parameterfunktionen berechnen lassen (Cheriet, 2007, S. 55).

Ein weiteres Merkmal ist das Amplituden-Histogramm, beispielhaft in Abbildung 14 - Image Histogramm zu sehen. Es zeigt die Verteilung der Grauwerte aller Pixel eines Grauwertbildes. Gibt es über die gesamte Amplitudenweite verteilte Grauwerte, dann hat das Bild einen hohen Kontrast. Liegen die Grauwerte eines Bildes nur in einem kleinen Grauwertbereich dann hat das Bild einen geringen Kontrast.



Abbildung 14 - Image Histogramm (Sun-Microsystems, 1999)

Des Weiteren können zum Beispiel mit der Hough Transformation verschiedene Formen aus einem Bild herausgefiltert werden. Mit entsprechenden Gleichungen als Parameter können so gerade Linien, Kurven oder andere parametrisierbare geometrische Formen gefiltert werden (Cheriet, 2007, S. 68). Bei der Hough-Transformation wird das ursprüngliche Bild aus seinem Bildraum mit einer Zuordnungsformel in den Parameterraum transformiert. Dafür wird jede Linie im Bildraum einem Punkt im Parameterraum zugeordnet. Die Koordinaten des Punktes im Parameterraum sind dabei die Parameter der Linie im Bildraum (OCR Seite 69). In Abbildung 15 - Hough-Transformation ist links das ursprüngliche Bild und rechts der Parameterraum zu sehen. Die hellen Stellen im Parameterraum weisen darauf hin, dass mit den zugehörigen Parametern im Originalbild eine Linie zu finden ist. Im kartesischen Koordinatensystem sind die Parameter einer Linie die Steigung und der Achsenabschnitt. Im Polarkoordinatenraum lässt sich eine Gerade durch den Winkel und die Länge der Normalen beschreiben, die durch den Ursprung geht. Polarkoordinaten sind hier geeignet, da senkrechte Gerade im kartesischen Koordinatensystem schwer zu beschreiben sind. (Steigung geht gegen unendlich)



Abbildung 15 - Hough-Transformation (OpenCV-Dokumentation, Hough Line Transform, 2017)

Es gibt eine Vielzahl weiterer Algorithmen um Merkmale zu beschreiben, die hier nicht weiter behandelt werden (Cheriet, 2007, S. 54-86). In jeder OCR-Software werden unterschiedliche Algorithmen eingesetzt um die Zeichen zu analysieren und relevante Merkmale festzustellen, die dann später zur Klassifizierung dienen. Je mehr Merkmale bestimmt werden, desto länger dauert die Texterkennung, umso genauer wird jedoch auch das Ergebnis.

### Klassifizierung der Muster

Im letzten Schritt der Texterkennung wird ein Zeichen mit seinen Merkmalen einem Label zuzuordnen, oder es wird ein Wert bestimmt, der die Ähnlichkeit zu einem vordefinierten Muster wiedergibt. Dafür wird in diesem Kapitel die statistische Klassifikation beschrieben. Es gibt weitere Arten um Zeichen zu klassifizieren, diese können auch über Voting-Systeme kombiniert werden um eine höhere Genauigkeit zu erreichen. Es werden zum Beispiel vermehrt künstliche neuronale Netzwerke werden für die Klassifikation eingesetzt und versprechen immer genauer werdende Texterkennung. Vor allem für Handschriften und komplexere Alphabete versprechen neuronale Netze gute Erkennbarkeit, (Computerwoche, 1992) während die klassische Texterkennung bei gut lesbaren, gedruckten Texten aus dem römischen Alphabet bereits sehr ausgereift ist.

Für ein grobes Verständnis wird die statistische Klassifikation veranschaulicht. Diese basiert auf der Bayes-Entscheidungsregel. Das soll mit einem Beispiel veranschaulicht werden. Dafür werden für ein Zeichen Merkmale bestimmt. Daraus ergibt sich der - dimensionale Vektor . soll einer Klasse aus der Menge zugeordnet werden. Mit den A-priori-Wahrscheinlichkeiten und den klassen-abhängigen Wahrscheinlichkeitsverteilungen ergeben sich die A-posteriori-Wahrscheinlichkeiten aus der Formel 1 Bayes-Formel“) (Cheriet, 2007, S. 131).

Formel 1 Bayes-Formel



Die Klassifizierung erfolgt über die höchste A-posteriori-Wahrscheinlichkeit mit der zu einer Klasse aus zugeordnet werden kann. M wäre damit beispielsweise eine Datenbank wie in Kapitel 4.2.1 („Klassischer Ablauf“) beschrieben, die für jeden Buchstaben die Merkmale und ihre Werte speichert, während der aktuell zu untersuchende Buchstabe mit allen seinen Merkmalen ist. Mit der Wahrscheinlichkeit gehört zur Klasse , die einen Buchstaben repräsentiert.

## 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. Aus den vorherigen Kapiteln wird ersichtlich, dass für eine effektive Texterkennung viele Daten gesammelt werden müssen. Diese Daten setzten sich aus Datenbanken für Dokumente, Schriftarten, Zeichensätzen, Sprachen und Klassifikationsmodellen zusammen. Derartige Daten zu sammeln ist zeitintensiv. Zum anderen wäre die Entwicklung der vorher beschrieben Algorithmen zeitaufwändig. Wenn die beschrieben Risiken eingegangen werden, 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 und damit stetig verbessert. Für kommerzielle Anwendungen werden meist Lizenzen von proprietären 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 (Asprise, 2018). Aufgrund der hohen Kosten wird in dieser Applikation eine freie Bibliothek benutzt.

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 es die Texterkennung in eine Cloud-Lösung auszulagern. Die Anwendung würde ein Foto aufnehmen und dann für die optische Charaktererkennung einen Webservice nutzen. Die Anwendung lädt 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, funktioniert dies meist nur über eine aktive WLAN-Verbindung, da sonst die Geschwindigkeit und die Kapazität nicht ausreichend sind. Außerdem kann nicht garantiert werden, dass der OCR-Server immer erreichbar und nicht überlastet ist. Bei einer Offline-Lösung benötigt die Anwendung zwar mehr Rechenkraft auf dem Gerät, der Engpass über die Internetverbindung fällt aber weg. In den folgenden Kapiteln werden zwei unterschiedliche freie Offline-OCR-Bibliotheken für Android untersucht.

### 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 (Sipe, 2016).

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 (Google-Developers, 2017).

### Tesseract

Tesseract ist eine OCR-Engine, die ursprünglich von HP entwickelt wurde und seit 2006 von Google weiterentwickelt wird (tesseract-ocr). Tesseract ist in C++ entwickelt, es gibt auf GitHub jedoch Bibliotheken für unterschiedliche Programmiersprachen, die eine Schnittstelle bieten. Tesseract kann über 100 unterschiedliche Sprachen erkennen und unterstützt eine Menge an Ausgabe- und Eingabeformaten. Mit einer Wertung von aktuell 18700 Stars ist Tesseract die beliebteste freie OCR-Bibliothek auf GitHub. Es gibt auf GitHub einen Fork des Tesseract-Repositories für Android (tess-two).

## Vergleich: Tesseract – Google Mobile Vision Text

### Kriterien

Um festzustellen, welche der beiden Bibliotheken sich für den Zweck dieser Anwendung besser eignet sollen beide Bibliotheken implementiert und mit Testdaten untersucht werden. Dafür werden folgende Kriterien untersucht:

* Performance
* Datenrepräsentation
* Komplexität
* Genauigkeit

Um die Performance zu untersuchen, soll geprüft werden wieviel Zeit auf einem Testgerät für die Texterkennung benötigt wird. Aufgrund der geringen verfügbaren Ressourcen wurde nicht mit Geräten unterschiedlicher Bauweise getestet. Das Testgerät ist ein Huawei P8 Lite 2017.

Das Kriterium „Genauigkeit“ beschreibt die Übereinstimmung eines erkannten Textes mit dem tatsächlichen Text, wie ihn ein Nutzer liest.

Die Datenrepräsentation beschreibt die Art und Weise, wie die von der API erkannten Daten dem Programm zur Verfügung stehen.

Die Komplexität beschreibt den Aufwand für einen Entwickler, die Funktionsweise der API zu verstehen und sie zu benutzen. Zum anderen soll hier auch der Funktionsumfang beschrieben werden, den die API bietet.

### Beispiel: Getränke-Rechnung

Es wird anhand der beispielhaften Getränke-Rechnung aus Abbildung 16 - Getränke-Rechnung und den Kriterien des vorherigen Kapitels ein API-Test durchgeführt.

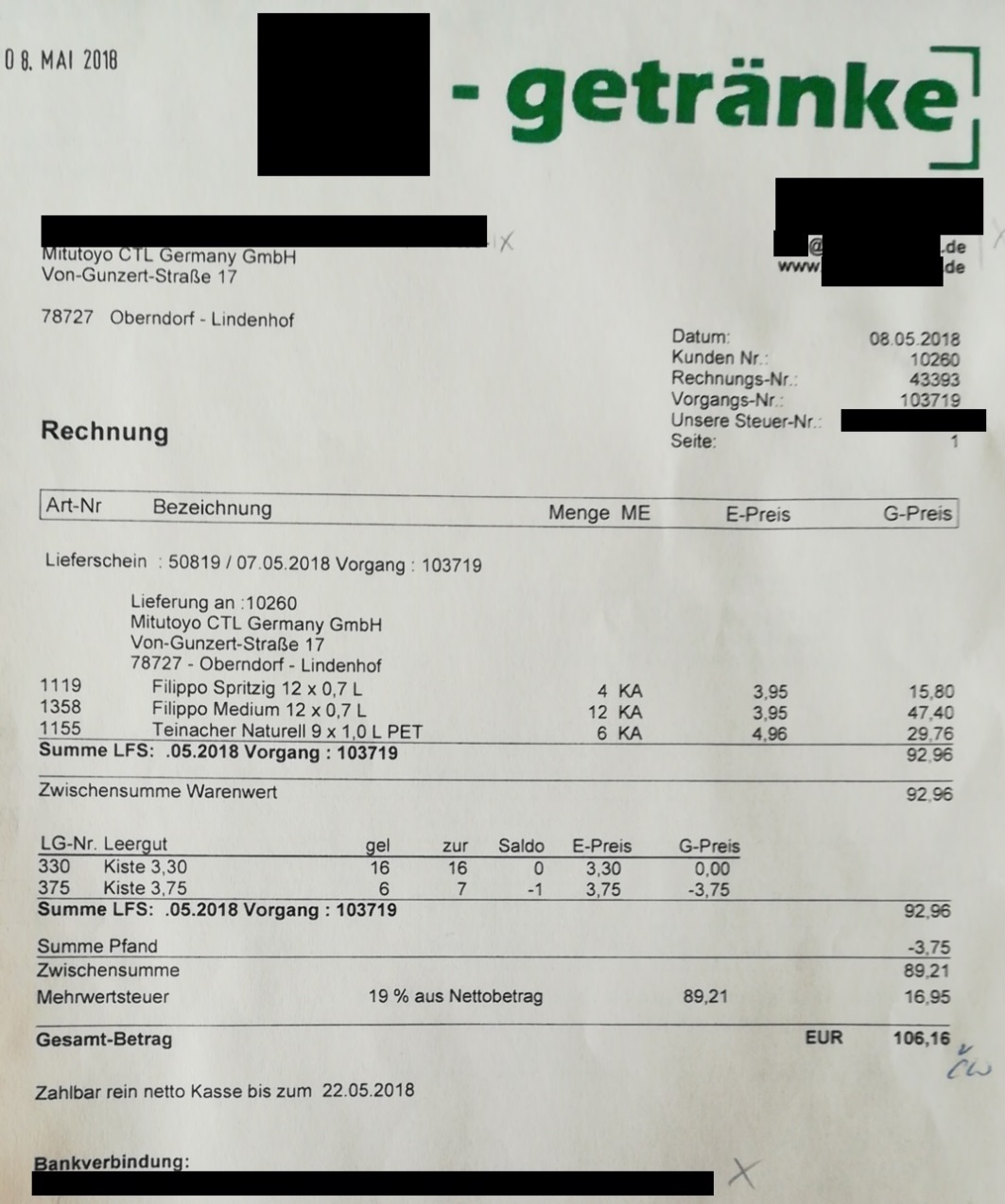


Abbildung 16 - Getränke-Rechnung

Der Text der Rechnung entspricht dabei etwa dem nachfolgenden Text. Einige Stellen sind aufgrund von Datenschutz geschwärzt.

08. MAI 2018 **XX**-getränke

**Absender-Anschrift** **Telefonnummer**

Mitutoyo CTL Germany GmbH **Faxnummer**

Von-Gunzert-Straße 17 **Email**

**Website**

78727 Oberndorf - Lindenhof

Datum: 08.05.2018

Kunden Nr.: 10260

Rechnungs-Nr.: 43393

Vorgangs-Nr.: 103719

Unsere Steuer-Nr.: **XXXXX** / **XXXXX**

Rechnung Seite: 1

Art-Nr Bezeichnung Menge ME E-Preis G-Preis

Lieferschein : 50819 / 07.05.2018 Vorgang : 103719

Lieferung an : 10260

Mitutoyo CTL Germany GmbH

Von-Gunzert-Straße 17

78727 - Oberndorf - Lindenhof

1119 Filippo Spritzig 12 x 0,7 L 4 KA 3,95 15,80

1358 Filippo Medium 12 x 0,7 L 12 KA 3,95 47,40

1155 Teinacher Naturell 9 x 1,0 L PET 6 KA 4,96 29,76

Summe LFS: .05.2018 Vorgang : 103719 92,96

Zwischensumme Warenwert 92,96

LG-Nr. Leergut gel zur Saldo E-Preis G-Preis

330 Kiste 3,30 16 16 0 3,30 0,00

375 Kiste 3,75 6 7 -1 3,75 -3,75

Summe LFS: .05.2018 Vorgang : 103719 92,96

Summe Pfand -3,75

Zwischensumme 89,21

Mehrwertsteuer 19 % aus Nettobetrag 89,21 16,95

Gesamt-Betrag EUR 106,16

Zahlbar rein netto Kasse bis zum 22.05.2018

Bankverbindung:

**IBAN: DE1234567891011121314 BIC: ABC2DE1XXX Beispielbank**

Bei der Analyse des Bildes mit Google Mobile Vision ergibt sich der nachfolgende Text:

08.05.2018 10260 43393 103719 **XXXXX/XXXXX** Datum: Kunden Nr Rechnungs-N

Vorgangs-Nr Unsere Steuer-Nr. Seite Bankverbindung: IBAN: **DEXXXX XXXX XXXX XXXX XXXX** BIC: **XXXXXX1XXX** **Beispielbank** EUR 89,21 19 % aus Nettobetrag 3,30 3,75 Zur 16 G-Preis 0.00 -

3,75 Saldo E-Preis 3,95 3,95 4.96 4 KA 12 KA 6 KA G-Preis E-Preis Menge ME Bezeichnung

Rechnung **Absender-Anschrift** **Telefonnummer** **Faxnummer** [xxxx@xx-getraenke.de](mailto:xxxx@xx-getraenke.de)

**w.xxxxxxxx.de** O8. MAI 2018 **xx** –getränke

106,16 Mehrwertsteuer Gesamt-Betrag Zahlbar rein netto Kasse bis zum 22.05.2018 92.96 -3.75

89.21 16.95 92.96 Teinacher Naturell 9 x 1,0 L PET 15.80 47.40 29.76 92.96 Lieferung an:10260

Von-Gunzert-Straße 17 Art-N

Bei der Analyse mit Tesseract wird der folgende Text als Ergebnis ausgegeben:

\/

l] 8. MAI 2018 "

‚'**XX** - g e ra n e '

**Telefonnummer**

**XX** gelak Amis H h RbenB ns rs 12 F"“°"“’”°“°

. , „ e r 35 emnc s 0 » osc - ua & ‚721725u11» ‚. **XXXX**@**XX**-getraenke.de

Mllutoyo CTL Germany GmbH w n ke.de

Von»Gunzen-Straße 17 '" -gmen

78727 Oberndorf - L nd h 1

' °" ° Datum ua 05 2018

Kunden Nr 10280

Rechnungs«Nr 43393

Vorgangs-Nr 103715

Unsere Steuer-Nr **XXXXX** / **XXXXX** '

Rech n ung Seite: 1

Af!?f f‚Efeziié??yng \_, » Wénge ME E-Prens G-Preis

Lieferschem 50819 / 07 05 2018 Vorgang ' 103719

L1e1erung an “10260

Mltuloyo CTL Germany GmbH

Von-Gunzen-Straße 17

78727 - Oberndorf - Lindenhof

1119 Filippo SpritZig 12 x 0.7 L 4 KA 3.95 15.80

1358 Fliipp0 Medium 12 x 0.7 L 12 KA 3,95 47.40

1155 ‚ ‚ TemacherNalure|19 x1.0 L PET 6 KA 4.% 29 76

Summe LFS: ‚05.2018 Vorgang : 109719 92 96

Zwßche113umme Warenwert 92 96

LG-Nr‚ Leergul gel zur Saldo E-Preis G-Preis

330 Kiste 3,30 16 16 0 3,30 0.00

375 Kiste 3.75 6 7 —1 3.75 -3.75

Summe LFS: .05.2018 Vorgang : 103719 92.96

Summe Pfand -3.75

Zwischensumme 89.21

Mehrwertsteuer 19 % aus Nencbetrag 89.21 16.95

WF\_—”Tmsns „

[m

Zahlbar rein netto Kasse bis zum 22.05.2018

Bankverbindung:

IBAN: **DEXXXX XXXX XXXX XXXX XXXX** BIC: **XXXXXX1XXX** **Beispielbank**

‘a«. >

ITU-.? \_\_

4 . ) ' v

. .„ "“ .—i,\_ ’ .

.), } ‘\_\_ . li“.

. \_ ‘

In den folgenden Kapiteln werden anhand dieser Beispiel-Rechnung die beiden Bibliotheken auf die Kriterien aus Kapitel 4.4.1 („Kriterien“) untersucht.

### Performance

Um die Performance zu messen wird mit der Funktion System.nanoTime() die Zeit gemessen bevor die Texterkennung beginnt (Android-Developers, System, 2018). Diese Funktion liefert die vergangene Zeit seit einem fixen, aber beliebigen Zeitpunkt. Dieser Zeitpunkt ist für alle Instanzen der aktuellen Java Virtual Machine gleich.

Durch die hohe Auflösung und den fixen Bezugspunkt eignet sich die Funktion für die Perfomance-Messung. Eine weitere Messung wird durchgeführt, nachdem die Texterkennung abgeschlossen ist. Die gesamte Dauer der Texterkennung errechnet sich aus der Differenz der beiden gemessenen Zeitpunkte.

Für diese Aufnahme ergaben sich folgende Werte:

Tesseract: 23,820 Sekunden

GMV: 1.430 Sekunden

Auch bei weiteren Tests und anderen Dokumenten benötigt Tesseract 15- bis 25-mal mehr Zeit.

### Datenrepräsentation

Bei Google Mobile Vision wird der erkannte Text in einem Array zurückgegeben. Jedes Arrayelement beschreibt dabei einen Textblock, wie in Abbildung 10 zu sehen. Diese Textblöcke bestehen wiederum aus Zeilen, welche sich in Wörter aufspalten.

Für das Verständnis der Struktur wurde eine Aufnahme und die darin erkannten Textblöcke in Abbildung 17 visualisiert. Bei dieser Aufnahme wurde der gesamte Text erkannt, d.h. alle Textabschnitte sind in einem Block. GMV hatte jedoch auch hin und wieder das Problem, dass wie in Kapitel 4.4.2 („Beispiel: Getränke-Rechnung“) nicht der gesamte Text erkannt wurde. Aus der Abbildung wird ersichtlich, dass GMV Text das Dokument in logische Blöcke aufteilt und dann einzeln den Text analysiert. Diese Blöcke sind nicht sortiert, sondern werden zufällig dem Array angehängt. Es werden jedoch auch die Koordinaten der Blöcke gespeichert, sodass man die Textblöcke mit eigener Programmlogik sortieren könnte, um den Text logisch anzuordnen.

Tesseract analysiert ein Bild, sobald die API mit einer Sprach-Bibliothek initialisiert wurde und das Bild als Parameter übergeben wird. Tesseract kann den erkannten Text als UTF8-Text, aber auch in anderen Formaten zurückgeben. Außerdem können die erkannten Regionen, Textlinien oder Wörter, sowie einige weitere Layout-Informationen ausgegeben werden.

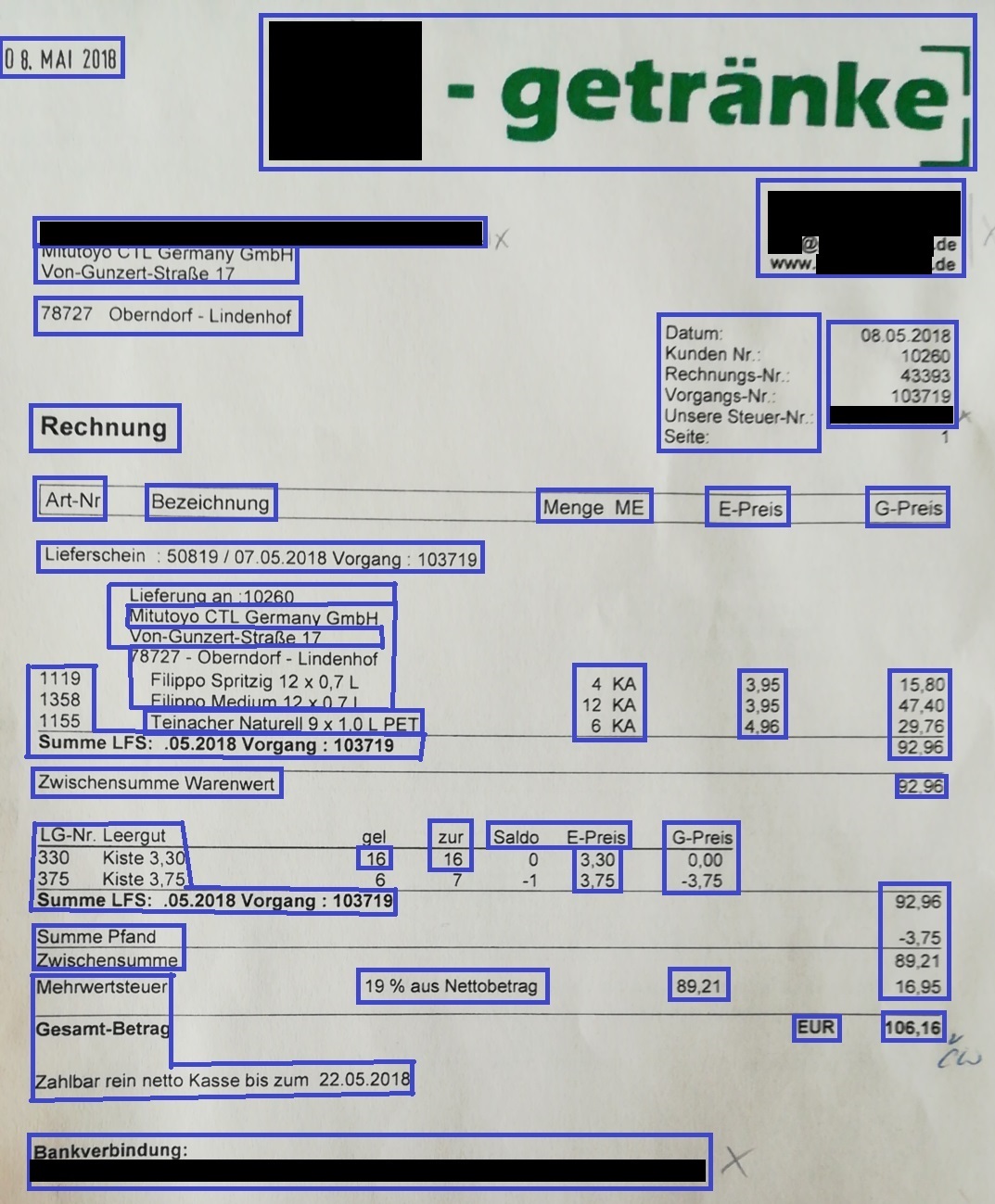


Abbildung 17 - Dokument von GMV Text analysiert

### Komplexität

Google Mobile Vision Text und Tesseract können einfach mit Gradle (Build-Managment-Tool) dem Projekt hinzugefügt werden. GMV Text benötigt nur ein Bild und einen Kontext und kann dann mit wenigen Zeilen Code implementiert werden. Mit der detect()- Funktion wird das Bild analysiert und der Text zurückgegeben. Wie in Kapitel 4.4.4 („Datenrepräsentation“) beschrieben, wird das Array aus Textblöcken zurückgegeben. Für die erkannten Textblöcke sind die Eckpunkte angegeben und es wird eine vorherrschende Sprache für jeden Textblock angegeben. Es werden aber sonst keine weiteren Funktionen oder Daten zur Verfügung gestellt.

Die Tesseract-API muss mit einer Sprachebibliothek initialisiert werden, die nicht durch die Gradle-Installation gegeben ist. Für die deutsche Sprache muss überprüft werden, ob die Datei „/tessdata/deu.traineddata“ in den Applikations-Dateien existiert. Ist dies nicht der Fall kann die Datei aus den Assets hinzugefügt werden. Außer diesem zusätzlichen Entwicklungsaufwand funktioniert die API ähnlich wie GMV Text. Tesseract kann die Daten in verschiedenen Formaten ausgeben, zum Beispiel hOCR oder UTF8. Die API besitzt über 60 Funktionen.

### Genauigkeit

Um den Unterschied zweier Zeichenketten zu bestimmen kann die Levenshtein Distanz berechnet werden. Diese gibt die Anzahl der Aktionen ‚Löschen‘,‘Einfügen‘ und ‚Ersetzen‘ um eine Zeichenkette in eine andere zu transformieren. Für die gesamten erkannten Texte aus Kapitel 4.4.2 („Beispiel: Getränke-Rechnung“) ergeben sich folgende Distanzen:

GMV: 950

Tesseract: 397

Dies ist jedoch wenig aussagekräftig, da wie in Kapitel 4.4.4 („Datenrepräsentation“) beschrieben, die Textfelder bei GMV zufällig ausgegeben werden. Deshalb untersuchen wir einzelne Zeilen um die generelle Genauigkeit zu bestimmen.

Wie in Tabelle 1 - Vergleich einzelner Zeichenketten zu sehen macht GMV bei erkannten Zeichenketten kaum Fehler, während dies bei Tesseract häufig vorkommt. Das Problem bei GMV ist wie in Kapitel 4.4.4 („Datenrepräsentation“) beschrieben, dass einige Textfelder nicht gefunden werden, während Tesseract alle Zeilen findet. Dies schlägt sich auch in der gesamten Distanz der Analysedaten nieder, die bei GMV fast dreimal höher ist als bei Tesseract

Tabelle 1 - Vergleich einzelner Zeichenketten

|  |  |  |
| --- | --- | --- |
| **Zeichenkette** | **Distanz GMV** | **Distanz Tesseract** |
| „Teinacher Naturell 9 x 1,0 L PET“ | 0 | 9 |
| „Von-Gunzert-Straße 17“ | 0 | 4 |
| Die Bankverbindung aus der Rechnung | 0 | 4 |
| „08. MAI 2018“ | 1 | 5 |

### Fazit

Im Folgenden werden die erfassten Kriterien nochmal tabellarisch zusammengefasst. In dieser Arbeit wurde nur eine Beispielrechnung beschrieben, andere Rechnungen lieferten jedoch ähnliche Ergebnisse.

Tabelle 2 Zusammenfassung API Vergleich

|  |  |  |
| --- | --- | --- |
|  | GMV | Tesseract |
| Performance | Gut | Schlecht |
| Datenrepräsentation | Mittel | Gut |
| Komplexität | Gut | Mittel |
| Funktionsumfang | Mittel | Gut |
| Genauigkeit | Gut | Mittel |

Die Performance ist bei GMV Text um einiges besser als bei Tesseract. Die lange Analysezeit von Tesseract weist darauf hin, dass die Bibliothek möglicherweise nicht für die Nutzung auf mobilen Geräten optimiert ist. GMV ist in der Entwicklung für die Anwendung auf mobilen Geräten konzipiert worden und lässt vermutlich besonders rechenaufwändige Algorithmen weg. Eine Wartezeit von 20-40 Sekunden (4.4.3 „Performance“) ist für den Benutzer nicht tragbar.

Tesseract liefert mehr Funktionen und auch die erkannten Textlinien sind gut sortiert und in einer sinnvollen Reihenfolge, während der Text bei GMV zufällig durchgemischt ist. Die Aufteilung in Textblöcke bringt bei Rechnungen relativ wenig. Tesseract ist komplexer in der Implementierung, da die Sprachbibliotheken selbst hinzugefügt werden müssen, dies macht jedoch für den Nutzer keinen Unterschied.

Zur Genauigkeit lässt sich sagen, dass Tesseract alle Textzeilen erkennt, während GMV manche Textblöcke weglässt. Dafür macht Tesseract viel mehr Fehler, während GMV kaum Fehler macht, wie in Kapitel 4.4.6 („Genauigkeit“) untersucht.

Trotz einer weniger geeigneten Datenstruktur eignet sich Google Mobile Vision Text durch die Kombination aus Performance, Einfachheit und relativ genauer Erkennung für die Texterkennung auf mobilen Endgeräten mehr als Tesseract.

# Android Entwicklung

## Applikations-Manifest

Jede Android-App muss eine AndroidManifest.xml Datei haben (Android-Developers, App Manifest Overview, 2018). Diese Datei beschreibt essentielle Informationen über die Applikation, wie zum Beispiel das Android-Betriebssystem oder die Build-Tools. Hier muss auch der Paketname festgelegt werden. Des Weiteren werden hier die App-Komponenten, wie Intents und Aktivitäten (vgl. 5.5 „Intents“) beschrieben. Auch muss hier angegeben werden, welcher Hardwarekomponenten und Berechtigungen die Anwendung benötigt (vgl. 5.4 „Berechtigungen“).

## 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. In dieser Beispielapplikation wären sowohl der Maileingang, als auch das Verfassen einer neuen E-Mail, jeweils eine eigene Aktivität, mit ihrem eigenen Lebenszyklus, der in Abbildung 18**Fehler! Verweisquelle konnte nicht gefunden werden.** zu sehen ist.

Android Applikationen können durch den Aufruf anderer Applikationen in den Hintergrund geraten. Dies muss nicht zwangsweise willentlich geschehen, wie es zum Beispiel beim Aufruf von Intents der Fall ist (vgl. 5.5 Intents). Schon ein eingehender Anruf kann die laufende Applikation in den Hintergrund verschieben. 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 18 - Aktivitätslebenszyklus

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

## Fragmente

Ein Fragment ist ein modularer Teil einer Aktivität, welcher einen eigenen Lebenszyklus besitzt (vgl. Abbildung 19). 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 19 - Fragmentlebenszyklus

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

## Berechtigungen

Android setzt Berechtigungen ein um die Privatsphäre von Android-Nutzern zu sichern (Android-Developers, Permissions Overview, 2018). Benötigt eine Applikation eine Berechtigung für sensible Nutzerdaten, wie zum Beispiel die Kontakte, muss diese explizit vom Nutzer erstattet werden. Dasselbe gilt für bestimmte System-Funktionen wie das Internet oder die Kamera. Die Berechtigungen müssen im Applikations-Manifest angegeben werden. Damit kann verhindert werden, dass eine Applikation, die zwingend eine Kamera benötigt auf einem Mobiltelefon ohne Kamera ausgeführt werden kann. Wenn eine Berechtigung zur Laufzeit benötigt wird, wird systemabhängig ein Standard-Dialog angezeigt, wie in Abbildung 20 zu sehen. Dieser Dialog kann vom Entwickler auch erweitert werden, um dem Anwender zu erklären, warum genau eine Berechtigung benötigt wird.



Abbildung 20- Permission-Request

## Intents

Ein Intent ist ein Nachrichtenobjekt mit dem eine Aktion von einer anderen Applikations-Komponente angefordert werden kann (Android-Developers, Intents, 2018). Es gibt die drei folgenden Anwendungsfälle für Intents (Android-Developers, Application Fundamentals, 2018):

* Eine Aktivität starten ( 5.2 „Aktivitäten“)
* Einen Service starte
* Einen Broadcast an eine andere Anwendung liefern

Ein Service ist ein Dienst, der im Hintergrund ohne Benutzeroberfläche läuft. Mit einem Broadcast kann einer Applikation signalisiert werden, dass ein Event eingetreten ist, zum Beispiel, dass das Gerät angefangen hat zu laden.

Für diese Anwendung ist der erste Anwendungsfall interessant, da zum Beispiel die systemabhängige Kamera-Anwendung gestartet werden könnte um ein Bild aufzunehmen. Auch die Galerie kann mit einem Intent geöffnet werden.

Für Aktivitäts-Intents kann eine Rücksprungfunktion im Programm implementiert werden, um das Ergebnis weiterzuverarbeiten.

## 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 (vgl. Abbildung 21). Weitere Formen der lateralen Navigation sind möglich, werden jedoch hier nicht weiter behandelt. (<https://material.io/design/navigation/understanding-navigation.html#lateral-navigation>)



Abbildung 21 - 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 (links), zum anderen den „Up button“ (rechts), der in der Applikationsleiste zu finden ist (vgl. Abbildung 22). (https://developer.android.com/training/design-navigation/ancestral-temporal)



Abbildung 22 - Back button und Up button

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

# Implementierung

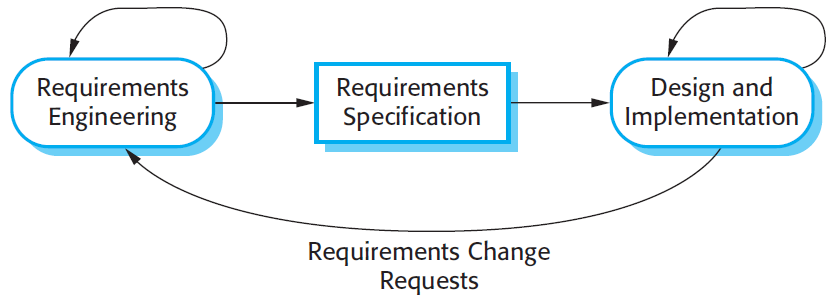
Durch die Implementierungsphase wird die definierte Spezifikation in die Praxis umgesetzt. Dabei ist aufgrund des gewählten inkrementellen Entwicklungsmodells darauf zu achten, dass die Spezifikation durch die Implementierung verändert und verbessert werden kann. (Sommerville S.38) 

Abbildung 23 - Plan-Based Development

(Sommerville S.63)

## Aufbau der Applikation

Wie in Kapitel 5.6 erläutert, ist vor der Implementierung im Code eine Navigationshierarchie zu erstellen, damit festgelegt ist wie der Nutzer in der Applikation zwischen zwei Ansichten navigieren kann. Die in diesem Projekt entstandene Applikation „InvoiceScanner“ ist aus einer einzigen Aktivität aufgebaut. Der Nutzer navigiert nur zwischen Fragmenten, die jeweils den kompletten Bildschirm einnehmen. Wie in Abbildung 24 zu sehen ist, ist die Navigationshierarchie zwei Ebenen tief, wobei die obere grün und die untere blau gekennzeichnet ist. Die grünen Pfeile demonstrieren die laterale Navigation zwischen den Hauptelementen durch das Menü, während die blauen Pfeile Aktionen darstellen, die den Nutzer zwischen den Ebenen navigieren lässt.

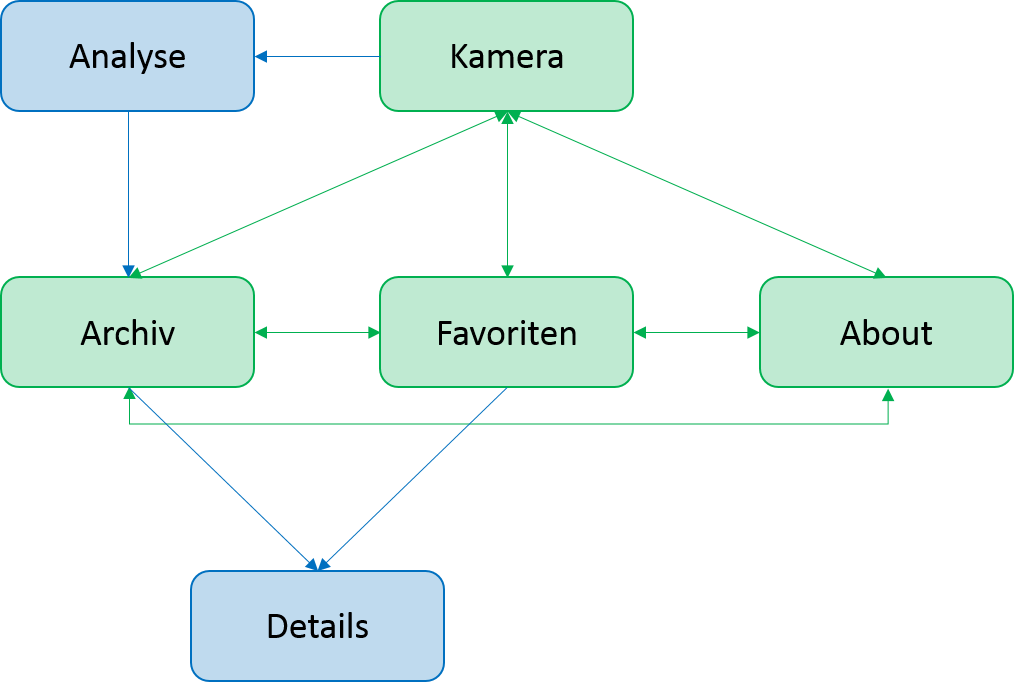


Abbildung 24 - Abwärts- und Lateralnavigation

### 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.

### 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.

## Datenbank

Um Rechnungsdaten zu speichern, wird eine Datenbank benötigt. Dies wurde mit der Room Persistance Library umgesetzt, die auf SQLite aufsetzt. Diese Bibliothek bietet eine Abstraktionsschicht um Datenbankzugriffe zu vereinfachen und trotzdem den vollständigen Funktionsumfang von SQLite auszunutzen. Um eine Rechnung zu speichern, wurde die Entität aus Abbildung 25 entworfen. Jede Rechnung hat eine eindeutige automatisch inkrementierende Identifikationsnummer. Eine Rechnung muss mindestens eine IBAN, den Betrag und einen Empfänger haben. Optional können der Pfad zum Bild der Rechnung, die BIC und ein Verwendungszweck angegeben werden. Außerdem gibt die Boolean-Variable „isFavorite“ an, ob ein Archiveintrag favorisiert wurde.



Abbildung 25 - Invoice Entität

Für den Zugriff auf Daten aus der Datenbank werden Zugriffs-Objekte (Data Access Object) definiert. Diese beschreiben, welche Operationen für bestimmte Daten möglich sind. Für Rechnungen wurden die Funktionen aus Abbildung 26 implementiert.

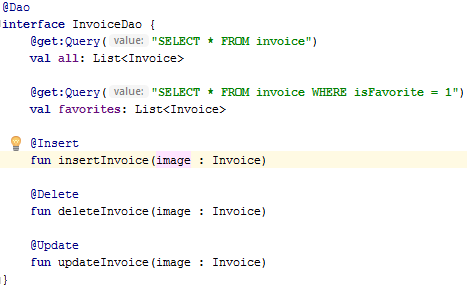


Abbildung 26 - Invoice Data Access Object

Rechnungen dürfen hinzugefügt, gelöscht und aktualisiert werden. Mit dem Funktionsaufruf database.invoiceDao().all werden alle Rechnungen in einer Liste zurückgegeben. Es kann mit dem Aufruf database.invoiceDao().favorites auch nur die Liste der favorisierten Rechnungen zurückgegeben werden

## 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. Dieses Verhalten wird durch Abbildung 27 verdeutlicht. Navigation durch den Back-Button ist hier durch einen roten Pfeil verdeutlicht, während Navigation durch den Up-Button gelb dargestellt ist.

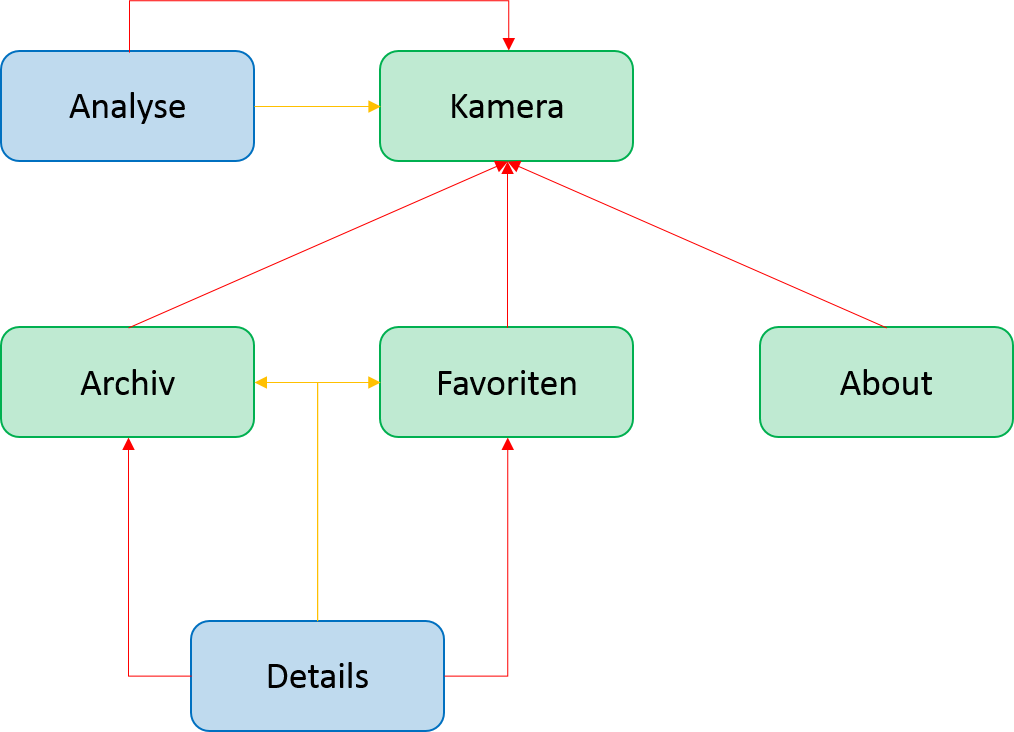


Abbildung 27 - Aufwärtsnavigation

## Filtern der Rechnungsdaten

Es werden folgende Rechnungsdaten für eine Überweisung benötigt:

* IBAN
* BIC
* Betrag
* Empfänger
* Verwendungszweck

Um diese Daten zu filtern, wurde mit regulären Ausdrücken gearbeitet. Ein regulärer Ausdruck ist eine Zeichenkette, die zur Suche nach einem bestimmten Muster in einer Zeichenkette dient.

Die Struktur einer IBAN ist durch den ISO 13636-1 Standard definiert und besteht aus einem Ländercode von zwei Buchstaben und bis zu 30 alphanumerischen Charakteren (iban swift reg). Die Länge der IBAN ist länderabhängig und die ersten beiden Charaktere nach dem Ländercode sind eine individuelle Prüfsumme. Eine deutsche IBAN wird wie in Abbildung 28 - Deutsche IBAN

gebildet. Nach dem Ländercode und der Prüfsumme werden die 8-stellige Bankleitzahl und die 10-stellige Kontonummer angehängt.



Abbildung 28 - Deutsche IBAN

Der reguläre Ausdruck für die IBAN-Implementierung ist in Abbildung 29 visualisiert.

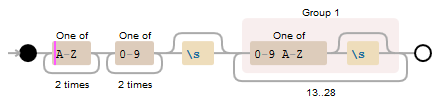


Abbildung 29 - IBAN-Regex grafisch

C:\Users\andy\Pictures\Screenpresso\2018-05-25_16h45_57.png

Abbildung 30 - IBAN Regex code

Mit dem Ausdruck [A-Z]{2} wird nach zwei Großbuchstaben in einem Text gesucht. Gefolgt wird diese Zeichenkette von zwei Zahlen aus der Menge [0-9]. Danach kann ein Leerzeichen folgen, oder es geht mit dem nächsten Teil des Ausdrucks weiter. Dieser definiert, dass nun ein Zeichen aus der Menge [0-9 A-Z] folgt. Danach kann wieder kein oder ein Leerzeichen folgen. Wiederholt wird das 13-28-mal. Die kürzeste IBAN hat Norwegen mit 15 Zeichen(<http://www.sepaforcorporates.com/single-euro-payments-area/iban-number-format-sepa-country/>).

Die IBAN hat für jedes Land einen individuellen Aufbau, eine Überprüfung wurde nicht implementiert. Die meisten Fehler werden durch die Überprüfung der Checksumme gefunden. Dies geschieht mit dem Modulo97 – Algorithmus (https://www.sparkonto.org/manuelles-berechnen-der-iban-pruefziffer-sepa/). Dafür wird der Ländercode aufgefüllt mit zwei Nullen hinten an die IBAN angehängt. Die Prüfsumme wird entfernt. Die Buchstaben werden mit Zahlen substituiert und zuletzt wird Modulo 97 gerechnet. Wenn das Ergebnis mit der Prüfsumme übereinstimmt ist die IBAN gültig.

Außerdem wird überprüft, ob der Ländercode gültig ist. Dafür gibt es ein Textdokument, in dem alle Ländercodes enthalten sind. Mit diesem wird der Ländercode der IBAN verglichen.

Die Struktur der BIC ist im Standard ISO 9362 definiert.

(<https://de.wikipedia.org/wiki/ISO_9362>)

Eine BIC besteht aus acht oder aus elf Zeichen. Die ersten vier Zeichen sind Buchstaben und vom Geldinstitut frei wählbar. Die nächsten zwei Zeichen sind der Ländercode. Darauf folgt eine zweistellige Codierung des Ortes aus Buchstaben, oder Zahlen. Die letzten drei Zeichen sind optional und entsprechen der Filiale oder Abteilung. Umgesetzt wurde dies mit dem Pattern aus Abbildung 31 und Abbildung 32.

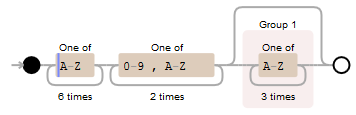


Abbildung 31 - BIC Pattern grafisch

C:\Users\andy\Pictures\Screenpresso\2018-05-25_17h26_50.png

Abbildung 32 - BIC Pattern code

Auch der Betrag wird mit einem Regex-Pattern gefiltert. Generell kann davon ausgegangen werden, dass die höchste Kommazahl auf einer Rechnung dem Betrag entspricht. Alle anderen Zahlen sind meist Teilbeträge, oder keine Kommazahlen. Es wird nach einem Komma gesucht, dass davor und danach unendliche viele Zahlen enthalten kann. Alle gefunden Kommazahlen werden in einer Liste gespeichert und nach der Suche wird die größte gesucht.

C:\Users\andy\Pictures\Screenpresso\2018-05-25_17h30_52.png

Abbildung 33 - RegexPattern Kommazahl

C:\Users\andy\Pictures\Screenpresso\2018-05-25_17h30_27.png

Abbildung 34 - RegexPattern Kommazahl code

Um den Verwendungszweck zu finden, wird in jedem Textblock (vgl. 4.4.4 „Datenrepräsentation“) nach dem String „Verwendungszweck:“ gesucht. Wird dieser gefunden, wird der restliche Text dieses Blocks als Verwendungszweck gespeichert. Auch die Erkennung des Empfängers funktioniert auf die gleiche Art und Weise.

# Fazit und Ausblick

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

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

# Literaturverzeichnis

Android-Developers. (08. 05 2018). *App Manifest Overview*. Abgerufen am 30. 05 2018 von https://developer.android.com/guide/topics/manifest/manifest-intro

Android-Developers. (22. 05 2018). *Permissions Overview*. Abgerufen am 30. 05 2018 von https://developer.android.com/guide/topics/permissions/overview#permissions

Android-Developers. (08. 05 2018). *System*. Abgerufen am 30. 05 2018 von https://developer.android.com/reference/java/lang/System

Asprise. (2018). *Java OCR and Barcode Recognition*. Abgerufen am 30. 05 2018 von https://asprise.com/royalty-free-library/java-ocr-source-code-open-order.html

Cheriet, M. (2007). *Character Recognition Systems : A Guide for Students and Practitioners.* Hoboken, N.J.: Wiley-Interscience.

Cleron, M. (17. Mai 2017). *Android Announces Support for Kotlin*. Abgerufen am 18. Februar 2018 von https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html

Computerwoche. (10. 07 1992). *Neuronale Netze auch für anspruchsvolle OCR-Aufgaben*. Abgerufen am 30. 05 2018 von (https://www.computerwoche.de/a/neuronale-netze-auch-fuer-anspruchsvolle-ocr-aufgaben,1135000),

Coronalabs. (kein Datum). *The 2D Game Engine*. Abgerufen am 13. März 2018 von https://coronalabs.com/product/

Developers, A. (8. Mai 2018). *App Startup Time*. Abgerufen am 10. Mai 2018 von https://developer.android.com/topic/performance/vitals/launch-time

Developers, A. (17. April 2018). *Designing Back and Up navigation*. Abgerufen am 25. April 2018 von https://developer.android.com/training/design-navigation/ancestral-temporal

Developers, A. (25. April 2018). *Designing effective Navigation*. Abgerufen am 1. Mai 2018 von https://developer.android.com/training/design-navigation/

Developers, A. (8. Mai 2018). *Fragments*. Abgerufen am 5. Mai 2018 von https://developer.android.com/guide/components/fragments

Developers, A. (17. April 2018). *Introduction to Activities*. Abgerufen am 18. April 2018 von https://developer.android.com/guide/components/activities/intro-activities

Developers, A. (kein Datum). *Getting Started with the NDK*. Abgerufen am 13. März 2018 von https://developer.android.com/ndk/guides/

Developers, A. (kein Datum). *Kotlin and Android*. Abgerufen am 13. März 2018 von https://developer.android.com/kotlin/index.html

Google-Developers. (20. Dezember 2017). *Text Recognition API Overview*. Abgerufen am 30. 05 2018 von https://developers.google.com/vision/android/text-overview

Herbert Süße, E. R. (2014). *Bildverarbeitung und Objekterkennung.* Wiesbaden: Springer Vieweg.

Material. (kein Datum). *Understanding Navigation*. Abgerufen am 2. Mai 2018 von https://material.io/design/navigation/understanding-navigation.html#lateral-navigation

McLemore, M., & olprod. (04. 25 2018). *Erste Schritte*. Abgerufen am 13. März 2018 von https://docs.microsoft.com/de-de/xamarin/android/get-started/

OpenCV-Dokumentation. (18. 12 2015). *Smoothing Images*. Abgerufen am 30. 05 2018 von https://docs.opencv.org/3.1.0/d4/d13/tutorial\_py\_filtering.html

OpenCV-Dokumentation. (24. 10 2017). *Hough Line Transform*. Abgerufen am 30. 05 2018 von https://docs.opencv.org/3.3.1/d3/de6/tutorial\_js\_houghlines.html

OpenCV-Dokumentation. (30. 05 2018). *Image Thresholding*. Abgerufen am 30. 05 2018 von https://docs.opencv.org/3.4/d7/d4d/tutorial\_py\_thresholding.html

PhoneGap, A. (kein Datum). *Products*. Abgerufen am 13. März 2018 von https://phonegap.com/products/

Pressman, R., & Maxim, B. (2015). *Software Engineering - A practitioner's approach.* New York: McGraw-Hill Education.

Sipe, M. (27. 06 2016). *Android Mobile Vision restores operation and adds Text API*. Abgerufen am 30. 05 2018 von https://android-developers.googleblog.com/2016/06/android-mobile-vision-text-api.html

Sommerville, I. (2011). *Software Engineering.* Boston: Pearson Education.

tesseract-ocr. (kein Datum). *tesseract*. Abgerufen am 30. 05 2018 von https://github.com/tesseract-ocr/tesseract

tess-two. (kein Datum). *Github tess-two*. Abgerufen am 30. 05 2018 von https://github.com/rmtheis/tess-two

# Abbildungsverzeichnis

[Abbildung 1 - V-Modell 6](#_Toc515451647)

[Abbildung 2 - Modellierung\_Klassen 8](#_Toc515451648)

[Abbildung 3 - Testing Pyramide 13](#_Toc515451649)

[Abbildung 4 - Kotlin Logo (https://pbs.twimg.com/profile\_images/699217734492647428/pCfEzr6L\_400x400.png) 15](file:///D:\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc515451650)

[Abbildung 5 - Xamarin Logo (https://avatars2.githubusercontent.com/u/790012?s=200&v=4) 16](file:///D:\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc515451651)

[Abbildung 6 - Corona Labs Logo (https://coronalabs.com/wordpress/wp-content/uploads/2016/08/corona\_logo-150x150.png) 16](file:///D:\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc515451652)

[Abbildung 7 - PhoneGap Logo (https://pbs.twimg.com/profile\_images/596058283699347456/NgaEDjHt\_400x400.jpg) 17](file:///D:\Studienarbeit\Studienarbeit_Mobile_App_Für_Foto_Überweisungen.docx#_Toc515451653)

[Abbildung 8 - Thresholding (OpenCV-Dokumentation, Image Thresholding, 2018) 20](#_Toc515451654)

[Abbildung 9 - Beispielhaftes Überweisungsformular 21](#_Toc515451655)

[Abbildung 10 - Textstruktur GMV (Google-Developers, 2017) 22](#_Toc515451656)

[Abbildung 11 - Gaussian-Blur (OpenCV-Dokumentation, Smoothing Images, 2015) 23](#_Toc515451657)

[Abbildung 12 - Median-Blur (OpenCV-Dokumentation, Smoothing Images, 2015) 24](#_Toc515451658)

[Abbildung 13 Eliminierung der Textneigung (Cheriet, 2007, S. 33) 24](#_Toc515451659)

[Abbildung 14 - Image Histogramm 26](#_Toc515451660)

[Abbildung 15 - Hough-Transformation (OpenCV-Dokumentation, Hough Line Transform, 2017) 27](#_Toc515451661)

[Abbildung 16 - Getränke-Rechnung 32](#_Toc515451662)

[Abbildung 17 - Dokument von GMV Text analysiert 37](#_Toc515451663)

[Abbildung 18 - Aktivitätslebenszyklus 43](#_Toc515451664)

[Abbildung 19 - Fragmentlebenszyklus 45](#_Toc515451665)

[Abbildung 20- Permission-Request 46](#_Toc515451666)

[Abbildung 21 - Navigation Drawer Vorlage 48](#_Toc515451667)

[Abbildung 22 - Back button und Up button 49](#_Toc515451668)

[Abbildung 23 - Plan-Based Development 50](#_Toc515451669)

[Abbildung 24 - Abwärts- und Lateralnavigation 51](#_Toc515451670)

[Abbildung 25 - Invoice Entität 54](#_Toc515451671)

[Abbildung 26 - Invoice Data Access Object 54](#_Toc515451672)

[Abbildung 27 - Aufwärtsnavigation 56](#_Toc515451673)

[Abbildung 28 - Deutsche IBAN 57](#_Toc515451674)

[Abbildung 29 - IBAN-Regex grafisch 57](#_Toc515451675)

[Abbildung 30 - IBAN Regex code 57](#_Toc515451676)

[Abbildung 31 - BIC Pattern grafisch 59](#_Toc515451677)

[Abbildung 32 - BIC Pattern code 59](#_Toc515451678)

[Abbildung 33 - RegexPattern Kommazahl 59](#_Toc515451679)

[Abbildung 34 RegexPattern Kommazahl code 59](#_Toc515451680)