HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

EasyPay

Studienarbeit

Abteilung Informatik

Hochschule für Technik Rapperswil

Herbstsemester 2015

Autoren:	Marino Melchiori
	Dominic Mülhaupt
Betreuer:	Prof. Dr. Farhad Mehta
Projektpartner:	Institut für Software (IFS)

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- das wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Ort, Datum:

Rapperswil, Dezember 2015

Namen, Unterschriften:

Marino Melchiori

Dominic Mülhaupt

Inhaltsverzeichnis

1 Abstract

1.1 Ausgangslage

Obwohl in der Schweiz die meisten Rechnungen per E-Banking bezahlt werden, müssen die Zahlungsdaten normalerweise manuell eingetippt werden. Es gibt zwar Lesegeräte um die erforderlichen Informationen einzuscannen, diese sind aber für die meisten Privathaushalte und kleineren Unternehmer zu teuer. Da heutzutage fast jeder über ein Smartphone verfügt, eignet sich eine mobile App als Lösung.

Dazu ist im Rahmen einer vorhergehenden Studienarbeit bereits eine Machbarkeitsanalyse durchgeführt worden, in der untersucht wurde, welche Möglichkeiten es gibt, diesen Prozess zu automatisieren. Dabei wurde ein technischer Prototyp geliefert, der grundsätzlich in der Lage ist, Daten einzuscannen und auf den PC zu übertragen.

Unsere Aufgabe war es, ausgehend vom technischen Prototyp, eine benutzerfreundliche App für die beiden Plattformen Android und iOS zu entwickeln. Des Weiteren soll die App den neuen, 2018 eingeführten Einzahlungsschein, unterstützen.

1.2 Vorgehen/Technologien

Die Schwerpunkte der Arbeit waren, eine Marktanalyse durchzuführen, das User Interface zu definieren, die Scanzeit zu verringern, sowie die Realisierung der beiden Apps.

Um eine ansprechende und intuitive Benutzeroberfläche zu entwickeln wurden Methoden des User-Centered Design Prozesses eingesetzt.

Für die Erfassung der Informationen auf einem Einzahlungsschein wurde Optical Character Recognition (OCR) eingesetzt, welche mithilfe von Training optimiert werden konnte.

Die jeweiligen Apps wurden native in Android Studio, respektive Xcode entwickelt.

1.3 Ergebnis

Mithilfe der entwickelten Apps können nun sowohl die aktuellen Einzahlungsscheine mit Codezeile, sowie die neuen Einzahlungsscheine mit QR-Code erfasst werden. Am PC kann via Browser auf die gespeicherten Einzahlungsscheine auf dem Smartphone zugegriffen werden. Die erfassten Werte können dann bequem ins E-Banking kopiert werden.

2 Management Summary

2.1 Ausgangslage

Obwohl in der Schweiz die meisten Rechnungen per E-Banking bezahlt werden, müssen die Zahlungsdaten normalerweise manuell eingetippt werden. Es gibt zwar Lesegeräte um die erforderlichen Informationen einzuscannen, diese sind aber für die meisten Privathaushalte und kleineren Unternehmer zu teuer. Da heutzutage fast jeder über ein Smartphone verfügt, eignet sich eine mobile App als Lösung.

Es bestand vor dem Projekt bereits ein technischer Prototyp in Form einer Android App. Für diese soll eine neue, benutzerfreundliche Version erstellt werden. Des Weiteren wird eine weitere App für die iOS Plattform zur Verfügung gestellt.

2.2 Funktionsweise

Um einen Einzahlungsschein zu erfassen und im E-Banking zu bezahlen, muss der Einzahlungsschein erst mit der Smartphone App gescannt werden. Dazu muss die Codierzeile unten rechts auf dem Einzahlungsschein erfasst werden. Ab 2018 werden neue Einzahlungsscheine, die einen QR-Code enthalten, eingeführt. Diese können ebenfalls gescannt werden.

Nach der Erfassung mit dem Smartphone kann der Benutzer am PC über den Browser auf die erfassten Einzahlungsscheine zugreifen. Werte eines Einzahlungsscheins, wie Kontonummer oder Rechnungsbetrag, können per Mausklick in die Zwischenablage kopiert werden um sie im E-Banking einzufügen.

2.3 Ergebnisse

Ergebnis des Projekts ist je eine App für Android und iOS. Damit ist es möglich, in der oben beschriebenen Art und Weise Einzahlungen zu erledigen. Leider bestehen noch einzelne Unannehmlichkeiten, welche in der Nutzung der App noch störend sind. Bevor diese noch beseitigt werden, ist es nicht zu empfehlen, die Apps zu publizieren.

3 Einleitung und Übersicht

Die Aufgabenstellung bestand aus einer Analyse vom technischen Prototyp. Dieser ist in einer Machbarkeitsstudie, Studienarbeit von Michael Burri und Benjamin Wilhelm (Mai 2015), entstanden.

Die bereits implementierte OCR-Bibliothek im technischen Prototyp sollte optimiert werden. Potentielle Benutzer würden die App nicht nutzen, wenn das Erfassen zu lange dauert. Da ab 2018 Einzahlungsscheine mit QR-Codes eingeführt werden, müssen diese auch erfasst werden können. Zu den Zielen gehörte auch die Entwicklung einer iOS-Version in Xcode.

Die übrigen Teile der Abgabe bestehen aus den Usability Tests, der GUI-Entwicklung (User-Centered Design) für die beiden Apps und für die Webseite.

Die Webseite kann vom Browser am Computer, der als Client fungiert, aufgerufen werden. Sie läuft auf einem embedded Webserver auf dem Smartphone. Dort werden die gescannten Einzahlungsscheine angezeigt. Die erfassten Werte eines Einzahlungsscheins kopiert der Benutzer mit einem Klick in die Zwischenablage und fügt Sie im E-Banking ein. Dieser automatische Prozess verhindert Tippfehler und spart Zeit.

Die Aufgaben wurden so verteilt, dass Dominic Mülhaupt sich primär um die Entwicklung der iOS App kümmert während Marino Melchiori den bestehenden Android Prototyp neu gestaltet und sich ausserdem mit der Optimierung der OCR, dem Einsatz von QR-Scannern und weiteren anfallenden Themen auseinandersetzt.

4 Ergebnisse

Die Optimierung vom OCR-Scanner war dank dem Schriftartentraining von Tesseract erfolgreich. QR-Code Unterstützung wurde ebenfalls umgesetzt. Bei schlechten Lichtverhältnissen kann nun der Kamerablitz verwendet werden.

Damit der Benutzer entscheiden kann, wann er die Einzahlungsscheine effektiv bezahlt, werden alle erfassten Daten auf dem Smartphone gespeichert. Die Daten eines Einzahlungsscheins sind für die Persistenz auf dem internen Speicher abgelegt. Alle gespeicherten Einzahlungsscheine werden dem Benutzer in einer Liste angezeigt. Er hat die Möglichkeit die Details eines Eintrages zu sehen, oder ihn zu löschen.

Der Webserver und die Webseite vom technischen Prototyp wurden erweitert. Der Web Client empfängt nun alle gespeicherten Einzahlungsscheine.

Die Installationsanleitungen für die jeweiligen Apps finden sich im Anhang unter *Installationsanleitungen*.

5 Anforderungsanalyse

Bei der Konzipierung wurden die Prinzipien des User-Centered Designs angewendet. Das heisst, dass das Projekt vom User Interface aus gedacht und gesteuert wird und der User von Anfang an in den Entwicklungsprozess einbezogen wird (Frontend first).

Eine ausführliche Beschreibung des User-Centered Design Prozesses findet sich im Dokument *User-Centered* Design im Anhang im Ordner *User-Centered Design*.

5.1 Marktanalyse

Eine Marktanalyse wurde in der Machbarkeitsanalyse schon durchgeführt. In dieser Studienarbeit ergänzen wir sie. Bereits jetzt sind schon sehr gute Lösungen vorhanden. Sie beinhalten Funktionen, wie Export-Files, welche direkt im E-Banking eingefügt werden können. Somit wird die Rechnung automatisch ausgeführt und bezahlt. Die Übertragung per E-Mail oder das Senden an einen Server, der als Desktop Applikation realisiert ist, werden auch schon geboten. Die Apps auf dem Markt können auch mehrere Einzahlungsscheine gleichzeitig übertragen. Bekannte Beispiele sind SmartID und smoothscan. Beide sind gratis, nur SmartID bietet zusätzliche Funktionen, die gekauft werden können.

Die Vorteile von EasyPay sind folgende:

- Keine Zusatzsoftware nötig (einzige Alternative: E-Mail)
- Simple Umsetzung mit nur zwei Screens (intuitive Bedienbarkeit umso wichtiger)
- Chance: Sollte die erste App mit QR-Code Unterstützung sein
- Gratis

Mögliche Nachteile von EasyPay sind:

- Langsamerer Scan
- Kein Scan der Adresse
- Es muss jedes Mal die IP in den Browser eingegeben werden

Unsere Open-Source-Variante Tesseract kann leider nicht mit proprietärer, kostenpflichtiger OCR-Software verglichen werden, da diese doch viel schneller arbeiten.

Die Adresse wird mit dem Codierzeilen-Scanner nicht erfasst. Allerdings kann die Adresse bei anderen Apps auch nicht gescannt werden.

5.2 Use Cases

5.2.1 UC1: Einzahlungsschein erfassen

Der Benutzer startet die App und kann gleich beginnen, den Einzahlungsschein zu erfassen.

Szenario a: vor 2018 (mit Code)

Mit dem dafür vorgesehenen Feld auf dem Bildschirm fokussiert er den Code auf dem Einzahlungsschein.

Szenario b: nach 2020 (mit QR Code)

Der Benutzer fokussiert den QR Code auf dem neuen Einzahlungsschein.

Szenario c: zwischen 2018 und 2020 (mit Toggle)

Dem Benutzer stehen beide Optionen zur Verfügung und er kann direkt im Capture Screen dazwischen umschalten.

Nachdem der Code erkannt wurde, wechselt der Screen und der Benutzer sieht die erfassten Einzahlungsscheine in einer Liste.

5.2.2 UC2: Einzahlungsschein bezahlen

Der Benutzer befindet sich mit seinem Smartphone auf dem Pay Screen. Er sieht die IP-Adresse vom Webserver und die zuvor erfassten Einzahlungsscheine. Er öffnet den Browser auf seinem PC und gibt die angezeigte IP-Adresse an. Nun wird die Webseite mit den Daten der Einzahlungsscheine geladen. Er sieht einen abgebildeten Einzahlungsschein, ausgefüllt mit den gescannten Werten. Unter dem Einzahlungsschein erkennt er die restlichen Daten dargestellt in einer Liste.

Der Benutzer loggt sich in seinem E-Banking ein und erfasst eine neue Zahlung. Er wechselt zurück zum Tab oder Browser. Die Referenznummer kopiert er in die Zwischenablage, indem er auf sie drauf klickt. Der Benutzer wechselt nun zum E-Banking zurück und fügt dort die Referenznummer, die in der Zwischenablage gespeichert ist, ein.

Dies wiederholt er mit allen gewünschten Werten und füllt den Rest noch von Hand aus.

5.2.3 UC3: Einzahlungsschein-Eintrag löschen

Der Benutzer hat die Rechnung bezahlt und möchte den erfassten Einzahlungsschein nicht mehr in der App sehen. Er tippt auf den Löschen Button in der App um den Eintrag zu löschen. Der gelöschte Eintrag wird von der Liste entfernt.

5.3 Funktionale Anforderungen

5.3.1 Allgemein

1	Anforderung	Priorität	Erfüllt
1.1	Internationalisierung	**Muss**	**Teilweise**
1.1.1	Standardmässig wird die Sprache vom Betriebssystem gewählt. Wenn diese Sprache nicht unterstützt ist, wird Englisch gewählt.	**Muss**	**Ja**
1.2	Verbindungsherstellung von Browser zu Webserver wird erklärt	**Muss**	**Ja**

1.3	Benutzer erhält Warnmeldung wenn er nicht mit dem WLAN verbunden ist.	**Muss**	**Teilweise**

Geplant war die Internationalisierung in Deutsch, Italienisch, Französisch und Englisch. Aus zeitlichen Gründen haben wird italienisch und französisch weggelassen. Für den Web Client wurde keine Mehrsprachigkeit umgesetzt.

Der Benutzer wird nicht explizit darauf hingewiesen wenn er nicht mit dem WLAN verbunden ist. Die Funktion wurde in der Android App nicht behandelt, in der iOS App wird statt der IP Adresse ein entsprechender Hinweis angezeigt. Es wurde ein kleines Tutorial implementiert, welches über einen Hilfe-Button aufgerufen wird und das Vorgehen erklärt. Zusätzlich weist es darauf hin, dass sich das Smartphone im gleichen WLAN wie der Computer befinden muss.

5.3.2 Texterkennung und QR-Code Scanner

2	**Anforderung**	**Priorität**	**Erfüllt**
2.1	Es können Einzahlungsscheine per QR-Code Scan erfasst werden.	**Muss**	**Ja**
2.2	Beim Scan der Codezeile wird gleichzeitig ein Bild der Adresse gemacht. Dieses Bild wird bei der Detailansicht eines Eizahlungsscheines auf dem Smartphone und auf der Webseite angezeigt.	**Soll**	**Nein**

Das Erfassen der Adresse beim Scannen der Codezeile wurde aus zeitlichen Gründen nicht umgesetzt. Weiterhin hätte es ein Problem mit dem Übertragen vom Bild an den Webserver gegeben. Der Webserver unterstützt nur Textnachrichten, die er automatisch auf der Webseite einblenden kann. Wenn ein QR-Code gescannt wurde, wird die Adresse extrahiert und auf dem Smartphone sowie auch auf der Webseite angezeigt. Die Adresse lässt sich dann auch kopieren. Sonst ist die Unterscheidung der Einzahlungsscheine immer noch durch den Betrag möglich.

5.3.3 Web

3	**Anforderung**	**Priorität**	**Erfüllt**
3.1	Die Webseite ermöglicht den Use Case “Später Bezahlen”. (Der User kann mehrere Einzahlungsscheine scannen und diese in einer Liste auf der Webseite ansehen.)	**Muss**	**Ja**
3.1.1	Es gibt die Möglichkeit einzelne Einzahlungsscheine zum Löschen zu markieren. In der Nächsten Session werden die Daten automatisch gelöscht.	**Muss**	**Nein**

3.2	Neu gescannte Einzahlungsscheine erscheinen direkt auf der Webseite. Gelöschte werden automatisch aus der Webseite entfernt werden.	**Soll**	**Teilweise**

Das Umsetzen einer Löschfunktion auf der Webseite wurde aus zeitlichen Gründen aufgegeben. Es gab ausserdem keine Möglichkeit vom Webserver eine Antwort zu senden, um die Daten dann effektiv zu löschen.

Die Android Version erfüllt die funktionale Anforderung 3.2, dass Updates automatisch auf der Webseite angezeigt werden. Die iOS Version hat einen Aktualisieren Button, den der Benutzer bei geänderten Daten betätigen muss. Das liegt an dem iOS Webserver Framework, das keine automatischen Updates ermöglicht.

Die Auswertung der funktionalen Anforderungen zeigt, dass der Aufwand für die Umsetzung unterschätzt wurde. Die Möglichkeiten der Bibliotheken wurden zu wenig genau untersucht. Trotzdem konnte die Grundfunktionalität mit einer App die Daten speichern und löschen kann und einem Webserver, der alle Daten in einer Liste anzeigen kann, erreicht.

5.3.4 Abgewiesene Features

Input Stick

Der Input Stick ist zur Übertragung aufgrund von Preis und Aufwand eher uninteressant für Privatpersonen. Gespräche mit betroffenen Personen aus KMUs haben ergeben, dass der externe Kauf eines Input Sticks eher nicht in Frage kommt. Gründe hierfür waren:

- Für ca. 150 CHF statt 40 CHF können auch gleich direkt bewährte Scanner gekauft werden.
- Viele Zahlungsvorgänge werden bereits mit E-Rechnungen gelöst.
- Die Beschreibung, wie ein solcher Input Stick zum Einsatz käme, wurde nicht verstanden. Eine Erklärung innerhalb der App – statt im Dialog – wäre noch schwerer verständlich.

Ein Fallbeispiel, wo eine vergleichbare App ausprobiert wurde, führte zu einer zu komplizierten Konfiguration. Die Verwendung der App wurde zu umständlich. Das Bestellen eines Input Sticks und dann darauf zu warten, wirkt abschreckend.

Für den wohl grösseren Anteil an Benutzern, welche nur den Webserver zur Verbindungsherstellung nutzen wollen, wirkt das Konzept der App verwirrend. Die Überlegung, dass unter Umständen etwas dazu gekauft werden müsste, schreckt ab. Die Entwicklung einer möglichst komfortablen Lösung via Webserver und eine ansprechende Gestaltung stand deshalb für uns im Vordergrund.

5.4 Nicht-funktionale Anforderungen

Die nicht-funktionalen Anforderungen wurden nach ISO/IEC 9126 entwickelt.

5.4.1 Funktionalität

1	**Anforderung**	**Priorität**	**Erfüllt**

1.1	**Richtigkeit**	**Muss**	**Ja**
	Die Anwendung sollte bei einem abgeschlossenen Scan keine falschen Ergebnisse liefern. Dies wird mit Hilfe der Prüfsumme von ESRs und entsprechenden Unit Tests sichergestellt.		
1.2	**Sicherheit**		
	Der einzige relevante Sicherheitsaspekt betrifft die HTTP-Verbindung zwischen Browser und Webserver innerhalb der Applikation. Über diese Verbindung werden die erfassten Werte der Einzahlungsscheine übermittelt. Diese Informationen könnten abgefangen werden wenn man sich im selben WLAN-Netzwerk befindet. Aufgrund der Kosten für ein entsprechendes SSL Zertifikat – um die HTTPS-Verbindung zu ermöglichen – und dem Umstand, dass Rechnungen normalerweise keine sicherheitsrelevanten Informationen enthalten, ist eine HTTP-Verbindung ausreichend. So oder so kann auf die gescannten Daten zugegriffen werden wenn man sich im selben WLAN-Netzwerk befindet *und* die lokale IP des mobilen Geräts kennt.		
1.3	**Ordnungsmässigkeit**	**Muss**	**Ja**
	Die Richtlinien für die jeweiligen App Stores – Apple App Store und Googles Play Store – werden eingehalten. Die Richtlinien finden sich im Anhang unter *Externe Spezifikationen/Richtlinien der Stores*. Für iOS: *App Store Richtlinien.pdf* Für Android: *Google Play Richtlinien.pdf* Die für die EasyPay relevanten Richtlinien wurden im Dokument *Übersicht der Richtlinien.pdf* in Form einer Checkliste zusammengefasst.		
1.4	**Konformität**	**Muss**	**Ja**
	Die Standards – sowohl für die aktuellen als auch die neuen Einzahlungsscheine – finden sich im Anhang unter *Externe Spezifikationen/Einzahlungsscheine*. Für die Standards der aktuellen, orangen Einzahlungsscheine halten wir uns an die hier definierten Vorgaben: *Manual ISR.pdf*.		

Für die Standards des 2018 neu eingeführten Einzahlungsscheins halten wir uns an die hier definierten Vorgaben: *Mass- und Gestaltungsmuster ES.pdf*, sowie den hier definierten Fahrplan: *Fahrplan – Migration ES.pdf*.

5.4.2 Zuverlässigkeit

2	Anforderung	Priorität	Erfüllt
2.1	**Reife** Die App darf in keinem der definierten Tests abstürzen und auch sonst keine Bugs aufweisen.	**Muss**	**Teilweise**
2.2	**Fehlertoleranz** Fehlerhafte Daten können nur beim Scanprozess entstehen. Unter schlechten Bedingungen wie schwachem Licht werden bei der OCR mehr Bilder falsch „gelesen“. Es wird allerdings bei jedem Scan anhand der Prüfsumme der Codierzeile überprüft, ob der Einzahlungsschein gültig. Nur falls der Einzahlungsschein gültig ist, wird er gespeichert.	**Muss**	**Ja**

Die beiden Apps haben die Anforderung der Reife noch nicht erfüllt da noch diverse Bugs vorhanden sind. Diese sind im Kapitel *Schlussfolgerungen* aufgeführt. Abstürze konnten bei den durchlaufenen Tests allerdings keine verzeichnet werden.

5.4.3 Benutzbarkeit

3	Anforderung	Priorität	Erfüllt
3.1	**Erlernbarkeit** Die App unterstützt den Benutzer beim Verständnis der Übertragungsmöglichkeiten indem einmalig Hilfestellungen eingeblendet werden sobald das entsprechende Verständnis benötigt wird (im Gegensatz zu einem einzelnen Textblock, der alles bereits vor der Nutzung der App erklärt). Die Hilfestellungen sollen auch noch nach dem ersten Lesen verfügbar für den Benutzer sein.	**Soll**	**Ja**
3.1.1	**Konformität**	**Muss**	**Ja**

Das Design der Apps wird konform zu den Design Guidelines der jeweiligen Plattformen entwickelt.

iOS:
https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/

Android: https://developer.android.com/design/index.html

5.4.4 Effizienz

4	Anforderung	Priorität	Erfüllt
4.1	**Zeitverhalten** Der Scanprozess soll bei optimalen Bedingungen (leistungsfähige Kamera, gute Lichtverhältnisse) nicht länger als zwei Sekunde dauern. Bei sehr schlechten Bedingungen (schwache Kamera, schlechte – aber noch akzeptable - Lichtverhältnisse, Smartphone nicht senkrecht über Einzahlungsschein) darf der Scanprozess nicht länger als fünf Sekunden dauern. Die Zeitdauer des Scanprozesses beginnt, sobald die Codierzeile fokussiert ist und endet sobald die Codierzeile erkannt wurde. Die Testergebnisse sind im Kapitel *Testing* dokumentiert.	**Muss**	**Ja**

5.4.5 Wartbarkeit

5	Anforderung	Priorität	Erfüllt
5.1	**Analysierbarkeit** Die App speichert detaillierte Log-Einträge, welche zu Debug-Zwecken verwendet werden können.	**Soll**	**Nein**
5.2	**Testbarkeit** Unit Tests beschränken sich auf die *PaymentSlip* Klassen. Da die Funktionstüchtigkeit der App stark von äusseren Umständen (Hardware, Umgebung) abhängig ist, wird verstärkt auf funktionale und nicht-funktionale Tests gesetzt. Weitere	**Muss**	**Ja**

Informationen sind in der Testspezifikation zu finden.

5.4.6 Übertragbarkeit

6	Anforderung	Priorität	Erfüllt
6.1	**Installierbarkeit** Der Installationsprozess wird durch die App Stores der jeweiligen Apps vorgenommen. Ansonsten muss nur ein Browser auf dem PC des Benutzers installiert sein.	**Kann**	**Nein**
6.2	**iOS** Es wird iOS 8.0 vorausgesetzt. Die App wird mit Swift programmiert, welches mit iOS 8 eingeführt wurde. Um auch tiefere iOS-Versionen zu unterstützen müssen für verschiedene Funktionen Spezialbehandlungen eingesetzt werden. Stand 14.12.15 verwenden nur 8% der App Store User eine tiefere iOS-Version als iOS 8 (https://developer.apple.com/support/app-store/). Ausserdem konnten keine Testgeräte mit einer tieferen iOS-Version organisiert werden.	**Muss**	**Ja**
6.3	**Android** Vorausgesetzt wird API Level 15 (Android 4.0.3 Ice Cream Sandwich). Für diese Mindestversion sind vier verschiedene Testgeräte, mit unterschiedlicher CPU Leistung, vorhanden. Nur 4% aller Geräte, die den Google Play Store nutzen haben einen tieferen API Level als Level 15.	**Muss**	**Ja**

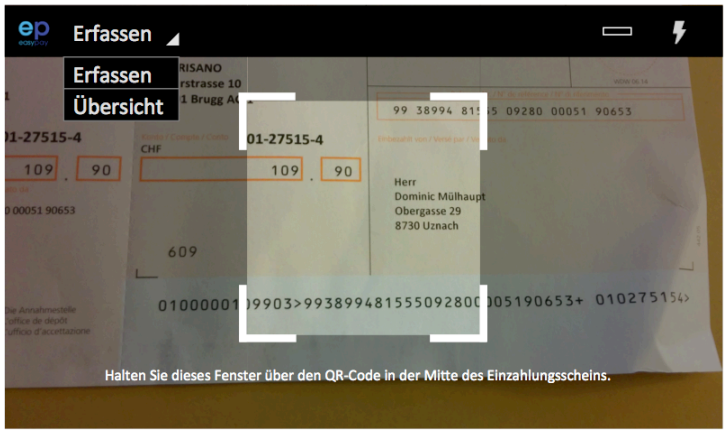
Das Deployment über die App Stores war aus organisatorischen Gründen nicht möglich, hätte aber auch zu einer weiteren Zeitbelastung geführt.

5.5 GUI

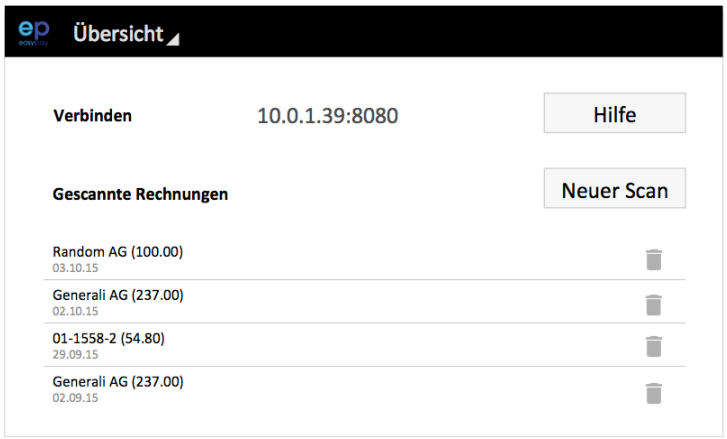
Ein Schwerpunkt des Projekts war die Entwicklung einer ansprechenden Benutzerführung. Im Anhang befindet sich im Ordner *User-Centered Design* eine Dokumentation zum Entstehungsprozess der GUI. Im Dokument *GUI final* findet sich die aktuelle Version der GUI.

Hier ein grober Überblick über die einzelnen Elemente.

5.5.1 Android

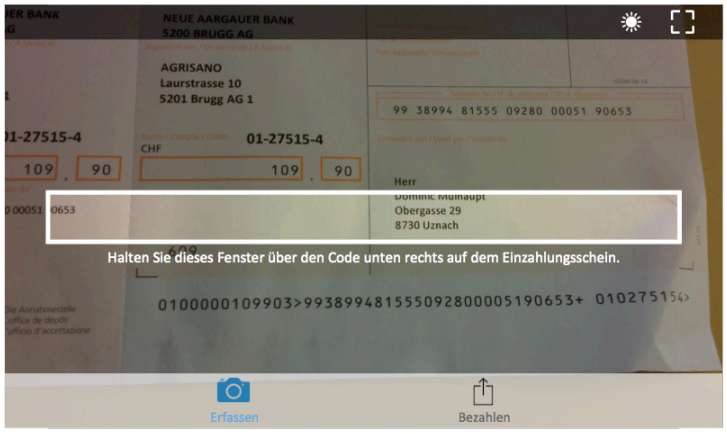


Capture Screen

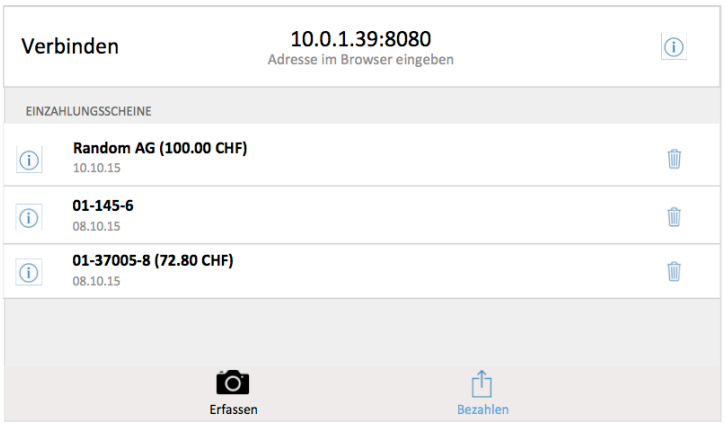


Pay Screen

5.5.2 iOS



Capture Screen



Pay Screen

5.5.3 Web Client

Ansicht im Browser

6 Technische Aspekte

6.1 Risikomanagement

6.1.1 Risikoanalyse der Elaboration-Phase

Unsere Risikoanalyse vom 30.09.2015.

Risiko	Massnahme	Eintritts-wahrscheinlichkeit	Schaden
1. In gewissen Fällen zu langsame Scans.	Optimierung mit OpenCV - Bildgrösse anpassen, Bild zuschneiden, Schwellenwertverfahren, Schrifttraining.	Wahrscheinlich	Hoch

Um dieses Risiko zu beseitigen wurde sehr viel Zeit aufgewendet. Es gibt viele Möglichkeiten dieses Risiko anzugehen. Der Aufwand war schwer einzuschätzen und stimmte nicht immer mit dem Ertrag überein. Zum Beispiel wurde eine Optimierung der Bilder durch OpenCV, mit einem Schwellenwertverfahren, in Betracht gezogen. Da Tesseract selber einen solchen Filter einsetzt, wäre das Ergebnis nicht mit Sicherheit besser gewesen. Um dies zu testen war schlicht keine Zeit vorhanden. Das Risiko der zu langsamen Scans konnte dann erfolgreich mit dem Schrifttraining für Tesseract behoben werden. Details hierzu sind im Kapitel *OCR* beschrieben.

2. Fehlplanung aufgrund der Einarbeitung in iOS Technologien	Lernen vor der Entwicklung	Wahrscheinlich	Gross

Ein grosses Risiko war die ganze Planung einzuhalten, ohne richtig einschätzen zu können, wie viel Zeit die Tasks für die Umsetzung brauchen. Das hat die Planungsphase erheblich erschwert. Es war nie sicher, ob unsere Ziele wirklich realistisch sind. Und das hat sich im ganzen Projekt bemerkbar gemacht. Die iOS Technologien mussten während dem Projekt erlernt werden.

3. Fehlerhafte Scans.	Test der Korrektheit der Scans durchführen.	Gering	Hoch

Das Risiko zu den fehlerhaften Scans konnte ebenfalls geklärt werden. Mit dem repetitivem Testen der App in der Entwicklungsphase, wurden die Zeichen stets korrekt erkannt.

6.1.2 Risikoanalyse der Construction-Phase

Risiko	Massnahme	Eintritts-wahrscheinlichkeit	Schaden
1. Tesseract Bug kann nicht rechtzeitig behoben werden.	Funktionen der App kürzen, damit sie rechtzeitig veröffentlicht werden kann	Wahrscheinlich	Gross

Während der Entwicklung ist unglücklicherweise ein Problem mit der Tesseract Bibliothek aufgetreten. Die App liess sich nicht mehr starten und der Grund war lange nicht klar. Mit grossem Aufwand konnten wir den Fehler und das Risiko beheben.

2. Die Webserver für Android und iOS sind zu verschieden, was zu einer höheren Entwicklungszeit führt.	Keine.	Wahrscheinlich	Mittel

Hier ist das Problem eingetroffen und führte zu einem weiteren Risiko (Nummer 6). Der iOS Webserver unterstützte keine direkten Updates der Webseite. Unsere Lösung war es, bei der Webseite der iOS App einen Aktualisieren Button einzubauen.

3. Webserver kann nicht rechtzeitig fertiggestellt werden	Löschfunktion der Webseite streichen. Eine Liste mit Links anstatt einer aufklappbaren Liste mit Einzahlungsscheinen darstellen.	Wahrscheinlich	Gering

Aus Zeitlichen Gründen haben wir die Liste mit den Links umgesetzt. Die Löschfunktion wird nur noch in der App angeboten. Von der Funktionalität her hat sich aber nicht viel geändert. Die Webseite ist immer noch sehr übersichtlich. Es gibt ein Einzahlungsschein der seine Werte ändern, sobald auf einen Link geklickt wurde. Die Liste mit den Links wird nur angezeigt, wenn mehr als ein Einzahlungsschein vorhanden ist.

4. Die Apps werden nicht rechtzeitig im Store freigeschaltet. Die Überprüfung dauert viel zu lange.	Store-Richtlinien genau überprüfen und umsetzen. Fokus auf die iOS App legen – da die Veröffentlichung auf dem Applestore viel länger dauert.	Wahrscheinlich	Gross

Für die Zeit vom Hochladen bis zur effektiven Veröffentlichung haben wir zwei Wochen eingeplant. Um dieses Risiko zu kontrollieren mussten wir unsere Zeitressourcen umverteilen. Die Veröffentlichung der App war ein klares Ziel dieser Studienarbeit. Aus diesen Gründen wollten wir die Entwicklung iOS App so schnell wie möglich beenden und zusammen am Webserver und der Webseite arbeiten. Dies hat auch geklappt, jedoch stellte sich im späteren Verlauf des Projektes heraus, dass die Veröffentlichung auf den Stores organisatorisch nicht so einfach möglich ist. Die Herausgabe über die HSR nicht möglich. Es lag an den Anforderungen von Apple und an den Kosten. Somit fiel das Ziel die Apps zu veröffentlichen weg.

6.2 Cross-Platform

6.2.1 Ursprünglicher Plan

Unser erster Gedanke war es die Apps separat zu entwickeln. Trotzdem suchten wir nach einer einfacheren Möglichkeit und entdeckten dann Xamarin Studio. Der Vorteil lag darin, dass sogar die GUI mit Xamarin.Forms nur einmal entworfen werden müsste. Gleichzeitig kamen aber grössere Probleme dazu.

Xamarin ist kostenpflichtig. Falls kein Support mehr angeboten wird, käme es zu einem Problem. Denn die App sollte erweiterbar und gut zu Warten sein. Es müssten neue Frameworks für C# gefunden werden. Und der bestehende technische Prototyp müsste portiert werden. Das macht vom Aufwand her keinen grossen Unterschied zu einer Entwicklung einer zusätzlichen iOS App.

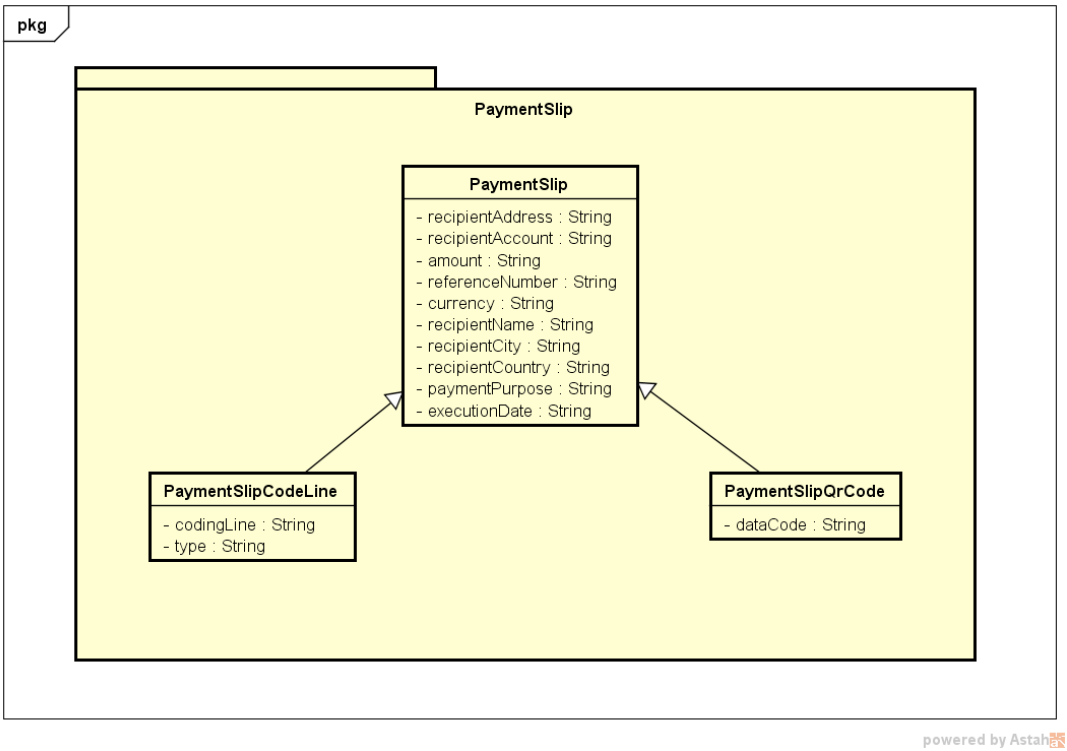
6.2.2 Entscheid

Aus den Problemen, die mit Xamarin entstanden wären, haben wir uns doch für einzelne native Apps (Android und iOS) entschieden. Die Frameworks können so grösstenteils übernommen und optimiert werden.

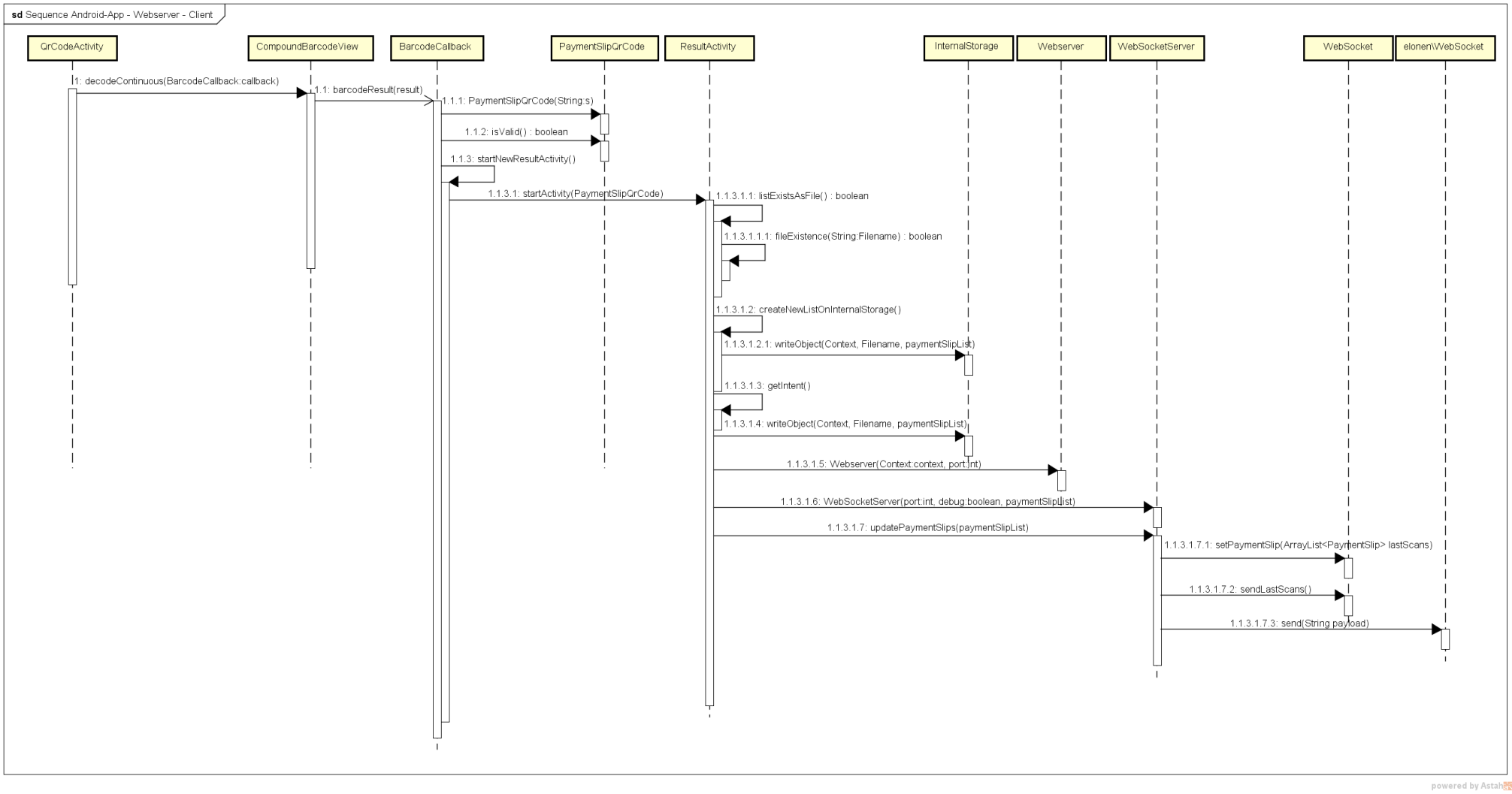
6.3 Architektur

6.3.1 Domainmodell

Das Domainmodell für die Einzahlungsscheine sieht folgendermassen aus. Die Struktur ist einfach aufgebaut. Der jeweilige Scanner erzeugt das passende PaymentSlip-Objekt.



6.3.2 Scanprozess und Webserver Kommunikation



6.4 OCR

6.4.1 Ausgangslage

Uns ist aufgefallen, dass die Zeit für das Scannen, vom technischen Prototyp, sehr unterschiedlich ist. Teilweise dauern Scans gerade mal 2 Sekunden - manchmal dauert es aber auch bis zu 8 Sekunden.

Als klare erste Priorität wurde die Scangeschwindigkeit vom Tesseract[1] OCR gesetzt. Unser Ziel war es also die Scanzeiten zu verbessern und konstant zu halten. Dafür mussten wir abschätzen, was in der vorhandenen Zeit überhaupt umsetzbar ist und ob es schlussendlich die Scan Zeit verringert.

Es gab viele verschiedene Möglichkeiten. Zum Beispiel wäre OpenCV[2], eine Open-Source-Bildbearbeitungssoftware, eine Option gewesen. Beim Recherchieren nach Beispielprojekten wurde OpenCV oft in Verbindung mit Tesseract verwendet. Nur war es nicht sicher, ob der grosse Aufwand auch zu einem guten Ergebnis führen würde. Tesseract hat sich weiterentwickelt und nutzt nun selber Schwellenwertverfahren (Otsu thresholding) um Binärbilder zu erstellen.

Trotzdem haben wir aber simplere und weniger aufwendigere Bildbearbeitungen, wie Graustufenfilter, ausgetestet. Es gab aber keine Unterschiede in den Scanzeiten.

6.4.2 Massnahmen

Als letzte Möglichkeit bot sich das Schrifttraining mit der Codezeilen-Schrift für Tesseract an. Und in der Machbarkeitsanalyse wurde das Schrifttraining als sehr aufwendig eingestuft. Glücklicherweise hat aber nur das Einlesen, wie gute Trainingsdaten erstellt werden können, Zeit gebraucht. Das Training selber war schnell erledigt und brachte auch eine Verbesserung der Scanzeiten. Es wurden hat auch die vorher fehlgeschlagenen Scans behoben. Denn selten wurden Codierzeilen gar nicht erkannt.

Um das Training mit der Codierzeilen-Schrift durchzuführen wurde die Schriftart OCR-B1 verwendet. Mit dem jTessBoxEditor[3] wurde das Boxfile erzeugt. Diese Datei legt den rechteckigen Rahmen, der gerade das ganze Zeichen enthält und dessen Wert fest. Für gute Ergebnisse sollte jeder Wert eines Zeichens mindestens fünf Mal, in zufälliger Reihenfolge, zum entsprechenden optischen Zeichen gemappt werden. Also haben wir neun Codierzeilen in einer Textdatei mit der OCR-B1 Schrift aufgelistet und das Boxfile erstellt. Trainingsdaten, die aus einer Textdatei mit mehr als neun Codierzeilen erstellt wurden, brachten keine bessere Performance. Aus dem Boxfile konnten wir die gesamten Trainingsdaten mit dem Serak Tesseract Trainer[4] V0.4 erstellen. Diese Software erzeugt einen Container. Die Spracheinstellung von Tesseract muss nur noch auf den Dateinamen des Containers gesetzt werden.

6.4.3 Technischer Prototyp verglichen mit SmartID Pay

Die genauen Testergebnisse befinden sich im Anhang unter *Testing/OCR.*

[1] https://code.google.com/p/tesseract-ocr/
[2] http://opencv.org/
[3] http://vietocr.sourceforge.net/training.html
[4] https://code.google.com/p/serak-tesseract-trainer/

Testergebnisse bei Tageslicht

Open Source tess-two	Open Source tess-two	OCR von SmartID Pay
EasyPay – technischer Prototyp **Scanzeiten in Sekunden**	**EasyPay mit OCR-Trainingsdaten** **Scanzeiten in Sekunden**	**SmartID Pay** **Scanzeiten in Sekunden**
Durchschnitt **2.77**	Durchschnitt **1.78**	Durchschnitt **0.4**

Testergebnisse bei schlechtem, künstlichem Licht

Open Source tess-two	Open Source tess-two	OCR von SmartID Pay
EasyPay – technischer Prototyp **Scanzeiten in Sekunden**	**EasyPay mit OCR-Trainingsdaten** **Scanzeiten in Sekunden**	**SmartID Pay** **Scanzeiten in Sekunden**
Durchschnitt **3.94**	Durchschnitt **2.55**	Durchschnitt **1.07**

Jede Zeile repräsentiert einen Einzahlungsschein mit einer einzigartigen Codierzeile. Die Erkennungszeiten liegen jetzt meistens zwischen 1.5 bis 2 Sekunden. Dies erfüllt unsere nichtfunktionalen Anforderungen. Gemessen wurde die Zeit wie es im Dokument im Anhang, unter *Testing/Nichtfunktionale Tests*, beschrieben wurde.

6.5 Frameworks

6.5.1 Android

6.5.1.1 OCR

Die Texterkennung wird wie vom technischen Prototyp durch das Framework tess-two[5] erledigt. Es ist eine Tesseract-Portierung für Android.

6.5.1.2 Web Server

Der technische Prototyp wurde mit NanoHTTPD[6] umgesetzt. Diese Implementation wurde fortgesetzt und auf unsere Anforderungen angepasst.

6.5.1.3 QR-Code

Zur Auswahl standen ZBar[7] und ZXing[8]. Zwei bekannte Projekte, die unseren Anforderungen entsprechen.

[5] https://github.com/rmtheis/tess-two
[6] https://github.com/NanoHttpd/nanohttpd

Entscheidung

Wir haben uns zuerst für ZXing entschieden, obwohl die Implementation eher aufwendig ist und es keine gleiche Lösung, wie für die iOS Variante gab. ZBar ist ausserdem veraltet.

Der Vorteil von ZXing ist, dass es sich direkt in die App integrieren lässt. Ohne, dass der Benutzer eine App installieren muss um diese dann aufzurufen. Weitere Vorteile waren, dass der Scanner kontinuierlich verbessert wurde und die Dokumentation ausführlich vorhanden ist.

Im Verlauf vom Projekt stellte sich heraus, dass die Integration etwas zu viel Zeit benötigte. Da sind wir auf ZXing embedded[9] gestossen. Dieses Framework liess sich durch das Anpassen vom Buildfile ganz einfach integrieren.

6.5.2 iOS

6.5.2.1 OCR

Für die OCR wurde das Framework Tesseract-OCR-iOS[10] verwendet, welches ebenfalls eine Portierung von Tesseract ist. Die Library ist in Objective-C geschrieben, kann aber auch mit Swift verwendet werden. Es handelt sich hierbei um die einzige noch aktuelle Portierung von Tesseract für iOS.

6.5.2.2 Web Server

Es wurden drei Frameworks untersucht: GCDWebServer[11], CocoaHTTPServer[12] und Swifter[13].

CocoaHTTPServer ist allerdings veraltet und wird auch nicht mehr aktualisiert.

GCDWebServer sieht ansprechend aus, ist aber in Objective-C geschrieben und war schwierig zu interpretieren.

Swifter ist in Swift geschrieben, sehr simpel gehalten und hat eine aktive Community.

Schlussendlich haben wir uns für Swifter entschieden da es einfacher zu integrieren war als GCDWebServer. Leider wurde erst zu spät bemerkt, dass Swifter die gängige Socket-Kommunikation nicht unterstützt und der Client nicht vom Webserver aus aktualisiert werden kann. Könnten wir uns nochmals entscheiden, würde GCDWebServer gewählt.

6.5.2.3 QR-Code

QR-Code Erkennung ist in iOS bereits integriert (AVFoundation[14]). Es musste kein externes Framework gesucht werden.

---

[7] https://github.com/ZBar/ZBar
[8] https://github.com/zxing/zxing
[9] https://github.com/journeyapps/zxing-android-embedded
[10] https://github.com/gali8/Tesseract-OCR-iOS
[11] https://github.com/swisspol/GCDWebServer
[12] https://github.com/robbiehanson/CocoaHTTPServer
[13] https://github.com/glock45/swifter

7 Testing

7.1 Funktionale Tests

Um die Funktionalität zu testen wird ein Testablauf spezifiziert, der sich aus den Use Cases ableitet. Die definierten Testläufe werden mit allen zur Verfügung stehenden Testgeräten durchlaufen. Jeder Test wird mit *Bestanden*, *Bestanden mit Einschränkungen*, oder *Fehlgeschlagen* bewertet. Wenn bei einem Test unerwartetes Verhalten auftritt, wird dieses als Kommentar erfasst.

Die Testspezifikation ist folgendermassen aufgebaut.

7.1.1 Starten der App

Beschreibung	Die App wird unter verschiedenen Startbedingungen gestartet.
Vorbedingungen	Der Benutzer befindet sich auf dem Home Screen. Die aktuelle App Version ist installiert. Die App wurde noch nie gestartet.
Schritte	1. Auf App Icon tippen 2. Zu Capture Modus QR-Code wechseln 3. App schliessen 4. Auf App Icon tippen 5. Zu Capture Modus Codierzeile und wieder zurück zu QR-Code wechseln
Erwartetes Resultat	Während dem Laden zeigt die App einen Launch Screen. Nach dem Laden zeigt die App den Capture Screen. Nach dem 1. Schritt befindet sich der Capture Screen im Codierzeilen-Modus. Nach dem 2. Schritt wird dem Benutzer ein Hinweis zum QR-Code angezeigt. Nach dem 4. Schritt befindet sich der Capture Screen im QR-Code-Modus. Nach dem 5. Schritt wird kein Hinweis zum QR-Code mehr angezeigt.

7.1.2 Scannen mit Codierzeile

Beschreibung	Ein ESR mit Codierzeile wird erfasst.
Vorbedingungen	Der Capture Screen befindet sich im Codierzeilen-Modus.

[14] https://developer.apple.com/av-foundation/

Schritte	1. Die Codierzeile mit dem transparenten Fenster im Capture Screen fokussieren
Erwartetes Resultat	Die Codierzeile wird innerhalb kurzer Zeit, nachdem der Kamerafokus ausgerichtet wurde, erkannt. Die App wechselt zum Pay Screen. Der erfasste Einzahlungsschein wird in der Liste angezeigt und wurde gespeichert.

7.1.3 Scannen mit QR-Code

Beschreibung	Ein Einzahlungsschein mit QR-Code wird erfasst.
Vorbedingungen	Der Capture Screen befindet sich im QR-Code-Modus.
Schritte	1. Die Kamera über den QR-Code bewegen
Erwartetes Resultat	Der QR-Code wird innerhalb kurzer Zeit, nachdem der Kamerafokus ausgerichtet wurde, erkannt. Die App wechselt zum Pay Screen. Der erfasste Einzahlungsschein wird in der Liste angezeigt und wurde gespeichert.

7.1.4 Details eines Einzahlungsscheins ansehen

Beschreibung	Details eines Einzahlungsscheins ansehen
Vorbedingungen	Die App zeigt den Pay Screen. Es wurde mindestens ein Einzahlungsschein erfasst.
Schritte	1. Auf Info Button im Listeneintrag eines Einzahlungsscheins tippen
Erwartetes Resultat	Die App zeigt dem Benutzer alle über den Einzahlungsschein erfassten Informationen an

7.1.5 Manuelles Wechseln zwischen Screens

7.1.5.1 iOS

Beschreibung	Über die Tab Bar wird zwischen Capture und Pay Screen gewechselt.
Vorbedingungen	Die App zeigt den Capture Screen.
Schritte	1. In der Tab Bar auf Bezahlen tippen 2. In der Tab Bar auf Erfassen tippen

Erwartetes Resultat	Nach dem 1. Schritt zeigt die App den Pay Screen an. Nach dem 2. Schritt zeigt die App den Capture Screen an.

7.1.5.2 Android

Beschreibung	Über den Navigation Spinner in der Action Bar wird zwischen Capture und Pay Screen gewechselt.
Vorbedingungen	Die App zeigt den Capture Screen.
Schritte	1. Im Navigation Spinner auf Übersicht tippen 2. Im Navigation Spinner auf Erfassen tippen
Erwartetes Resultat	Nach dem 1. Schritt zeigt die App den Pay Screen an. Nach dem 2. Schritt zeigt die App den Capture Screen an.

7.1.6 Mit Browser auf Webserver zugreifen

Beschreibung	Mit Browser auf Webserver zugreifen
Vorbedingungen	Smartphone und PC Client sind zum selben WLAN-Netzwerk verbunden. Die App zeigt den Pay Screen. Browser auf PC Client läuft. Es wurden mindestens zwei Einzahlungsscheine erfasst.
Schritte	1. Auf Info Button, der neben der IP Adresse angezeigt wird, tippen 2. Erklärung wegtippen 3. Über den Browser auf die im Pay Screen angezeigte IP Adresse zugreifen
Erwartetes Resultat	Nach dem 1. Schritt wird eine Erklärung angezeigt, die die Verbindungsherstellung zwischen Client und App beschreibt. Nach dem 3. Schritt wird im Browser die Client View angezeigt. Dabei wurde die Ansicht des Einzahlungsscheins mit den Werten des zuletzt gescannten Einzahlungsscheins aufgefüllt. Unterhalb dieser Ansicht findet man eine Liste mit allen gespeicherten Einzahlungsscheinen. Der 3. Schritt kann auch ausgeführt werden wenn die App sich im Capture Screen befindet.

7.1.7 Austausch zwischen App und Client

7.1.7.1 iOS

Beschreibung	Mit der App werden neue Einzahlungsscheine erfasst und bestehende gelöscht. Der Client soll dabei aktuell gehalten werden.
Vorbedingungen	Die Verbindung zwischen App und Client ist hergestellt. Die App zeigt den Capture Screen.
Schritte	1. Einzahlungsschein mit der App erfassen 2. Im Browser auf Aktualisieren klicken 3. Den vorherig erfassten Einzahlungsschein in der App löschen 4. Im Browser auf Aktualisieren klicken
Erwartetes Resultat	Nach dem 2. Schritt stellt die Ansicht des Einzahlungsscheins im Browser den soeben erfassten Einzahlungsschein dar. Ausserdem wurde die Liste um den entsprechenden Einzahlungsschein erweitert. Nach dem 4. Schritt ist der im 1. Schritt erfasste Einzahlungsschein weder in der Liste noch in der Ansicht des Einzahlungsscheins zu finden.

7.1.7.2 Android

Beschreibung	Mit der App werden neue Einzahlungsscheine erfasst und bestehende gelöscht. Der Client soll dabei aktuell gehalten werden.
Vorbedingungen	Die Verbindung zwischen App und Client ist hergestellt. Die App zeigt den Capture Screen.
Schritte	1. Einzahlungsschein mit der App erfassen 2. Den vorherig erfassten Einzahlungsschein in der App löschen
Erwartetes Resultat	Nach dem 1. Schritt stellt die Ansicht des Einzahlungsscheins im Browser den soeben erfassten Einzahlungsschein dar. Ausserdem wurde die Liste um den entsprechenden Einzahlungsschein erweitert. Nach dem 2. Schritt ist der im 1. Schritt erfasste Einzahlungsschein weder in der Liste noch in der Ansicht des Einzahlungsscheins zu finden.

7.1.8 Verwendung des Clients

Beschreibung	Die Funktionen im Browser des Clients werden verwendet.

Vorbedingungen	Die Verbindung zwischen App und Client ist hergestellt. Es wurden mindestens zwei Einzahlungsscheine erfasst. Der zweitoberste Eintrag der Liste der Einzahlungsscheine enthält einen (neuen) Einzahlungsschein mit allen erfassbaren Werten.
Schritte	1. Den zweitobersten Listeneintrag unter „Erfasste Einzahlungsscheine“ anklicken 2. Für alle erfassten Werte wiederholen: a. Auf den erfassten Wert klicken b. Überprüfen, ob die Zwischenablage mit dem erfassten Wert übereinstimmt
Erwartetes Resultat	Nach dem 1. Schritt werden sämtliche Werte des zweiten Einzahlungsscheins in der Ansicht des Einzahlungsscheins dargestellt. Bei 2.b. stimmen die beiden Werte immer überein.

7.1.9 Testergebnisse

7.1.9.1 iOS

Folgende Probleme sind aufgetreten:

- Scannen mit Codierzeile
    - Die Codierzeile wird nur gescannt, wenn manuell in den Codierzeilen-Modus gewechselt wird. Bei insgesamt sechs Tests mit iOS-Geräten haben wir festgestellt, dass dieses Problem nur bei den drei Geräten unter iOS 9 auftritt.
    - Das Geräusch des Kameraauslösers wird immer abgespielt. Das Gerät muss auf lautlos eingestellt sein.
- Mit Browser auf Webserver zugreifen
    - Im HSR-Netzwerk hat die Verbindungsherstellung nicht geklappt.

7.1.9.2 Android

Folgende Probleme sind aufgetreten:

- Starten der App
    - Der Letzte Scan-Modus wird mit dem Schliessen der App nicht gespeichert. Die App startet immer mit dem Codierzeilen-Scanner.
- Details eines Einzahlungsscheins ansehen
    - Der Fall, dass der Einzahlungsschein keinen Betrag liefert, wurde nicht behandelt.

Die vollständigen Ergebnisse, mit einer genaueren Beschreibung der Probleme, befinden sich im Dokument *Funktionale Tests – Bericht* im Anhang unter *Testing*.

7.2 Nichtfunktionale Tests

Bei den Tests wurde die geprüft ob die Anforderung der Scanzeit eingehalten wird.

Geräte	Durchschnittszeit bei Tageslicht	Durchschnittszeit bei schwachem Licht
iPhone 5	1.5	2.6
Sony Xperia T3	1.5	2.5
LG Nexus 5	1.4	2.5
Samsung Galaxy S4	1.5	2.4
Samsung Galaxy Nexus	1.8	2.7

Die ausführlichen Tests befinden sich im Anhang unter *Testing/Nichtfunktionale Tests.*

7.3 Unit-Tests

Unit Tests sind nur in der der *PaymentSlip* Klasse nötig. Da werden die Checksumme, die Validation von Werten und Währungstypen, das Parsing und die Formatierung von Werten für alle erforderlichen Typen von Einzahlungsscheinen geprüft.

8 Schlussfolgerungen

Verglichen mit anderen Codierzeilen-Scannern auf dem Markt, sind unsere Scanzeiten noch zu langsam. Wie schon in der Machbarkeitsanalyse festgestellt, müsste dazu eine bessere OCR-Lösung verwendet werden. Mögliche OCR-Alternativen sind in der Machbarkeitsanalyse evaluiert worden.

Was wir auf jeden Fall erreicht haben ist eine intuitive und benutzerfreundliche GUI. Vergleichbare Apps sind durch die vielen Informationen und Funktionen unübersichtlich.

Internationalisierung wurde auf Deutsch und Englisch umgesetzt. Die App SmartID Pay bietet zusätzlich noch Französisch an. Ein QR-Code Scanner wird hingegen noch von keiner App auf dem Markt angeboten.

Die Übertragung unserer Daten mit dem Webserver ist viel eleganter, als die Übertragung per E-Mail. Nur die Bezahlung per Datentransfer (nächstes Kapitel) wird nicht unterstützt. Diese Funktion einzubinden wäre für das weitere Vorgehen geplant.

Ändern würden wir die Lösung des Webservers. Leider konnten für iOS und Android nicht die gleichen Frameworks verwendet werden. Dadurch unterscheiden sich auch die Funktionen und wie die Webseite zu bedienen ist. Es müsste eine einheitliche Lösung gefunden werden.

8.1 Noch zu behandeln

8.1.1 Bestehende Bugs

Die Apps – sowohl für Android als auch iOS – enthalten noch Bugs, welche behoben werden müssten. Eine Auflistung dieser Bugs finden sich in den Testergebnissen der funktionalen Tests.

8.1.2 Future Work

8.1.2.1 Zahlung per Datentransfer (DTA und OPAE)

Im E-Banking lassen sich Rechnungen direkt bezahlen, wenn sie als Datei im DTA- oder OPAE-Format hochgeladen werden.

Vorteile:

- Praktisch, auch für mehrere Rechnungen
- Apps auf dem Markt bieten dies ebenfalls an. Die Konkurrenz löst das Problem der Dateiübertragung per E-Mail.
- Wird von allen bekannten Banken unterstützt

Nachteile:

- Weitere Daten müssen auf Smartphone statt im E-Banking erfasst werden (Kontoinhaber: Vorname, Name, PLZ, Ort, IBAN / Empfänger: Name, Adresse, PLZ, Ort)
- Relativ unbekannt – dem Benutzer muss geholfen werden.

Glossar

Apple App Store	Apples Vertriebsplattform für iOS Apps (u.a.)
Capture Screen	View innerhalb der App, in der neue Einzahlungsscheine erfasst werden können
ESR	Einzahlungsschein mit Referenznummer
Google Play Store	Googles Vertriebsplattform für Android Apps (u.a.)
IFS	Institut für Software der HSR
OCR	Optical Character Recognition
Pay Screen	View innerhalb der App, welche die Übersicht über alle erfassten Einzahlungsscheine anzeigt