
PRAKTIKUMSBERICHT

Herr
Hannes Steiner

Entwicklung einer Dokumenten-Scanner-App

**Digitalisierung der Verwaltung an der Hochschule
Mittweida**

2020

Fakultät **Angewandte Computer- und
Biowissenschaften**

PRAKTIKUMSBERICHT

Entwicklung einer Dokumenten-Scanner-App

**Digitalisierung der Verwaltung an der Hochschule
Mittweida**

Autor:
Hannes Steiner

Studiengang:
Softwareentwicklung

Seminargruppe:
IF17wS-B

Matrikelnummer:
46540

Erstprüfer:
Prof. Dr. Marc Ritter

Zweitprüfer:
Dipl.-Inf. Holger Langner

Mittweida, Mai 2020

Bibliografische Angaben

Steiner, Hannes: Entwicklung einer Dokumenten-Scanner-App, Digitalisierung der Verwaltung an der Hochschule Mittweida, 47 Seiten, 23 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Praktikumsbericht, 2020

Dieses Werk ist urheberrechtlich geschützt.

Referat

Im Rahmen eines zwölfwöchigen Forschungspraktikums an der Hochschule Mittweida arbeiteten der Student Tobias Kallauke und der Verfasser des Berichts gemeinsam an einem Projekt mit dem Hintergrund der Digitalisierung der Verwaltung von Lehr- und Forschungseinrichtung. Dabei entwickelten sie einen Anschauungsprototyp, mit dessen Hilfe kontrollierte Klausuren eingescannt und digitalisiert werden können.

I. Inhaltsverzeichnis

Inhaltsverzeichnis	I
Abbildungsverzeichnis	II
Abkürzungsverzeichnis	III
1 Einleitung	1
2 Hochschule Mittweida	2
3 Problemstellung	3
3.1 Digitalisieren der Klausur-Daten	3
3.2 Klausuren-Vorlage	3
3.3 Klausuren-Vorlage verbessern	4
4 Anforderungen	5
5 Konzept der Dokumenten-Scanner-App	7
5.1 Scan-Vorlage erstellen und speichern	7
5.2 Scan-Vorlage verwenden	9
5.3 Erweiterung des Konzepts	10
6 Entwicklung des ersten Prototyps	12
6.1 Anforderungsplanung	12
6.2 Analyse und Definition	12
6.3 Grundlagen	13
6.4 Entwurf und Design	15
6.5 Implementierung	17
6.5.1 Registrieren und Anmelden	18
6.5.2 Scan-Vorlagen erstellen und speichern	18
6.5.3 Scan-Vorlage verwenden	22
7 Fazit	26
7.1 Stand des Prototyps	26
7.2 Probleme und Grenzen der App	26
7.2.1 Probleme beim Erkennen von Dokumenten	27
7.2.2 Probleme der Klausur-Vorlage beim Scannen	27

7.2.3 Weitere Probleme der App	28
7.3 Verbesserung der Klausuren-Vorlage	28
8 Ausblick	29
A Wasserfall-Modell	31
B Workflow	32
C Weitere Funktionalität	35
D Installation der iOS App	38
E Tätigkeitsbericht	39
F Tätigkeitstabelle	42
Literaturverzeichnis	46

II. Abbildungsverzeichnis

3.1 Umsetzung der Klausuren-Vorlage der Fakultät CB	4
5.1 Schematisches Schaubild zum Verständnis des Schablonen-Konzepts	7
5.2 Kompaktes Flussdiagramm zum Erstellen einer Scan-Vorlage	7
5.3 Schematische Darstellung des Vorgangs Scan-Vorlage erstellen und auf dem Server speichern	8
5.4 Schematischer Ablauf beim Verwenden einer Scan-Vorlage	9
5.5 Kompaktes Flussdiagramm zum Verwenden einer Scan-Vorlage	10
5.6 Schematische Darstellung des Vorgangs beim Verwenden einer Scan-Vorlage	10
6.1 Schematische Darstellung des Model-View-ViewModel Konzepts	14
6.2 Architektur Schemata	16
6.3 Willkommen- Registrier- und Anmelde-View	19
6.4 Die ersten Views zur Erstellung einer Scan-Vorlage	20
6.5 Views zur Erstellung von Regionen	21
6.6 Views zur Erstellung von Kontrollmechanismen	21
6.7 Dialogfenster zur Auswahl der OCR-Engine	22
6.8 Listen- und Detail-Ansicht von Scan-Vorlagen	25
A.1 Das Wasserfall-Model nach Winston W. Royce	31
B.1 Gekürztes Flussdiagramm zum Erstellen einer Scan-Vorlage	33
B.2 Gekürztes Flussdiagramm zum Verwenden einer Scan-Vorlage	34
C.1 Vorlagen-Detailansicht auf einem iPad mit Barrierefreiheit Beispiel	35
C.2 Beispiel-Fehlermeldungen und View beim Löschen eines angelegten Kontrollmecha- nismuses	36
C.3 Seiten-Vorschau-View	36
C.4 Seiten-Vorschau im Light- und Dark-Mode	37
C.5 Barrierefreiheit und Dark-Mode Beispiele	37

III. Abkürzungsverzeichnis

API	application programming interface, deutsch: Programmierschnittstelle, Seite 17
CRUD	Das Akronym CRUD steht für Create/Erstellen, Read/Lesen, Update/Aktualisieren und Delete/Löschen und umfasst die vier grundlegenden Operationen persistenter Speicher, Seite 5
Fakultät CB	Fakultät für angewandte Computer- und Biowissenschaften, Seite 2
HSMW	Hochschule Mittweida - university of applied science, Seite 2
IDE	integrated development environment, deutsch: Integrierte Entwicklungsumgebung, Seite 17
MVVM	Model-View-ViewModel, Seite 13
OCR	optical character recognition, deutsch: optische Zeichenerkennung und im deutschen Synonym für Texterkennung, Seite 9

1 Einleitung

Digitalisierung wird gängig als Integration von digitaler Technologie in den Alltag verstanden und soll helfen Zeit einzusparen. Mit diesem Gedanken initiierten die Mitarbeiter Holger Langner und Falk Schmidsberger der Hochschule Mittweida das Projekt Memo Space. Im Zuge dessen sollen kleinere Forschungsergebnisse entstehen, die richtungsweisend für die Digitalisierung der Verwaltung von Lehr- und Forschungseinrichtung sind.

Im Rahmen eines zwölfwöchigen Forschungspraktikums an der Hochschule Mittweida arbeiteten der Student Tobias Kallauke und der Verfasser des Berichts gemeinsam an einem Forschungsprojekt von Memo Space. Dabei entwickelten sie einen Anschauungsprototyp zum Einstellen und Digitalisieren von kontrollierten Klausuren.

2 Hochschule Mittweida

Die Hochschule Mittweida - university of applied science (HSMW) wurde vor über 150 Jahren gegründet. Heute lehrt und forscht sie mit ca. 6000 Studenten in fünf Fakultäten und vier Forschungsschwerpunkten [5]. Einer der Schwerpunkte ist die Angewandte Informatik, in dem Memo Space angesiedelt ist.

Nach eigener Einschätzung schreibt jeder Student der HSMW ca. 5 Prüfungen pro Semester. Das bedeutet, dass im Jahr um die 60.000 Klausuren kontrolliert werden. Dazu kommt, dass die Zensuren in das Verwaltungs-Intranet der HSMW eingetragen werden müssen. Dieser Vorgang ist ebenfalls aufwendig, repetitiv und deshalb kognitiv belastend.

Da die Mitarbeiter der Fakultät Angewandte Computer- und Biowissenschaften (Fakultät CB) Holger Langner und Falk Schmidsberger selbst Klausuren kontrollieren und die Problematik genau kennen, entstand hier eine der ersten Ideen für Memo Space.

3 Problemstellung

An der Kontrolle von Klausuren arbeiten zum Ende eines Semesters Hochschulmitarbeiter über Tage. Diese Aufgabe muss stets mit hoher Konzentration erledigt werden. Auch bei der anschließenden Eingabe der Noten in ein Webformular des Verwaltungs-Intranets der HSMW ist ein kognitiv belastender Navigationsaufwand notwendig. Denn um die Zensuren einzutragen, muss für jeden Studenten zunächst die jeweilige Seminargruppe und anschließend der Studenten-Name herausgesucht werden. Die unübersichtlichen Listen und die hohe Anzahl an Klicks sind für die Hochschulmitarbeiter belastend. Zusätzlich wird die Suche nach der Seminargruppe durch die ähnlich lautenden Bezeichnungen erschwert. Das führt beim Danebenklicken in der Seminargruppen-Liste dazu, dass man sich eventuell in der Navigation verirrt und von vorne beginnen muss.

3.1 Digitalisieren der Klausur-Daten

Für genau diesen Vorgang wird nach einer effizienteren Lösung gesucht. Die Prüfer sollen effektiv und möglichst zeitsparend diese Aufgabe verrichten, ohne die Überlastung ihrer Aufmerksamkeitsspanne. Zudem ist angedacht, alle relevanten Klausuren-Daten zu digitalisieren und in ein geeignetes Format zu bringen. Darüber hinaus empfiehlt es sich, digitale Kopien der Klausuren abzuspeichern, um sie nicht nur analog zu archivieren.

3.2 Klausuren-Vorlage

Eine Klausuren-Vorlage bzw. ein Gestaltungsleitfaden für Klausuren der Fakultät Angewandte Computer- und Biowissenschaften bietet außerdem die Möglichkeit der Kontrolle des Prüfers. Durch das vorgegebene Layout der Klausur ist es möglich, Fehler des Prüfers zu erkennen. Die Klausur in Abbildung 3.1 ist nach dieser Klausuren-Vorlage angefertigt worden. Auf dem Deckblatt (3.1a) der Prüfung ist eine Tabelle mit drei Zeilen und jeweils einer Spalte pro Klausur-Aufgabe. In der ersten Zeile befinden sich die Nummern der Aufgaben, in der zweiten die zu erreichenden Punkte der Aufgabe. In der dritten Zeile trägt der Prüfer die erbrachten Punkte des Studenten ein. Unter der Tabelle befindet sich ein Feld für die erreichte Gesamtpunktzahl sowie ein weiteres für die aus den Punkten resultierende Note. Es soll nun überprüft werden, ob die Summe der erreichten Punkte mit der Gesamtpunktzahl übereinstimmt. Zudem wird auch die Note errechnet und mit dem Ergebnis des Prüfers verglichen werden. Ein weiteres Merkmal der Klausuren-Vorlage ist ein Feld, für die vom Studenten erreichten Punkte über jeder Aufgabenstellung auf den nachfolgenden Seiten. Zu sehen sind solche Felder in Abbildung 3.1b am rechten Seitenrand. Die dortige Angabe sollte mit der Angabe in der

Tabelle auf dem Deckblatt (3.1a) übereinstimmen und bietet somit noch eine weitere Möglichkeit der Kontrolle.

**Klausur
Grundlagen
Informations-
technologie**

Wintersemester
2019/2020

Professur
Medieninformatik
Prof. Dr. rer. nat.
Marc Ritter
Dipl.-Inf.
Falk Schmidberger
Dipl.-Inf.
Robert Manthey

Name: _____
Matrikel #: _____
Seminargruppe: _____

Aufgabe	1	2	3	4	5	6	7	8	9	10	11	12
Punkte	10	5	5	9	3	7	5	9	8	9	2	9
Score												

Punkte: / 81 Note: _____

Allgemeine Hinweise

Schreiben Sie Ihren Namen, Vornamen und Ihre Matrikelnummer leserlich auf alle Blätter, die Sie für Ihre Lösung verwenden - bevor Sie mit der Bearbeitung beginnen! Blätter ohne diese Angaben können nicht gewertet werden!

Geöffnete blaue Unterlagen (Spirale, Vorlesungsmitschriften, etc.) zugelassen. Sie werden als Tauschgegenmittel gewertet. Dessen abweichend wird bei Ihnen jeweils ein A4-Blatt mit ausschließlich handschriftlichen Notizen zugelassen!

Der Versuch der Täuschung hat die Wertung der Klausur mit "nicht bestanden" zur Folge.

Benutzen Sie keine Bleistiele, keine rotschreibenden Stifte und kein Tipp-Ex (oder ähnliche Produkte).

Zum Bestehen der Klausur mindestens die Hälfte der erreichbaren Punkte notwendig.

Bei mehreren Aufgabenbearbeitungen müssen Sie sich Hinweise und Hilfestellungen werden, falls erforderlich, offiziell für den zuständigen Hörsaal durchsetzen.

Geben Sie keine mehrdeutigen (oder mehrere) Lösungen an. In solchen Fällen wird statt die Lösung mit der geringeren Punktzahl gewertet (Eine richtige und eine falsche Lösung zu einer Aufgabe ergeben also 0 Punkte).

NUTZEN SIE DIE SATZFORM BEI DEN BEGRIFFEN ERLÄUTERN UND BESCHREIBEN!

NUTZEN SIE FÜR IHRE LÖSUNGEN AUCH DIE RÜCKSEITEN. GEBEN SIE STETS DIE ZUGEHÖRIGE AUFGABENNUMMER AN.

Rechnerarchitektur

Aufgabe 1
Vervollständigen Sie die Begriffe in der folgenden Skizze der von Neumann-Architektur eines Personal Computers. (/10 Punkte)

Betriebssysteme

Aufgabe 2
Welche 5 Hauptaufgaben hat ein Betriebssystem? (/5 Punkte)

Verschlüsselung

Aufgabe 3
Erläutern Sie das Verfahren der symmetrischen kryptografischen Verschlüsselung und nennen Sie Vor- und Nachteil. (/5 Punkte)

(a) Deckblatt der Klausur

(b) Erste Seite der Klausur mit drei Aufgaben

Abbildung 3.1: Umsetzung der Klausuren-Vorlage der Fakultät CB

Die Bilder sind beim Erstellen einer Scan-Vorlage mit der App entstanden.

3.3 Klausuren-Vorlage verbessern

Nachdem ein erster Prototyp zum Digitalisieren der Klausur-Daten entwickelt wurde, sollen außerdem Klausuren-Vorlagen entstehen, die für die Digitalisierung optimiert sind. Dabei sollten Probleme, die beim Einscannen der aktuellen Vorlage erkannt wurden, behoben werden.

4 Anforderungen

In diesem Kapitel werden die Anforderungen an das Software-System dargestellt.

Es soll eine App für das Betriebssystem iOS entstehen, mit der Klausuren digitalisiert werden können. Dabei müssen die, für die Notenfreigabe relevanten Daten der Klausur, in ein tabellarisches Format gebracht werden. Ferner muss die iOS-Anwendung Bilder von ausgefüllten Prüfungen aufnehmen können und den Inhalt mithilfe von Texterkennung digitalisieren.

Auch bei unterschiedlicher Seitengestaltung der Klausuren, soll die App in der Lage sein, alle relevanten Informationen, wie z. B. Name, Matrikelnummer und Note richtig zu lokalisieren und anschließend zu digitalisieren. Dafür muss der Benutzer digitale Vorlagen, sogenannte Scan-Vorlagen erstellen. Mit deren Hilfe können, die Informationen sofort gefunden und schneller digitalisiert werden, ohne die gesamte Klausuren-Seite zu verarbeiten.

Zusätzlich soll es dem Benutzer möglich sein, Kontrollmechanismen festzulegen. Wie im Abschnitt 3.2 beschrieben, kann überprüft werden, ob die Punkte auf dem Deckblatt mit den Punkten auf den nachfolgenden Seiten übereinstimmen. Des Weiteren wird kontrolliert, ob die Gesamtpunktzahl korrekt summiert sowie die daraus resultierende Note richtig errechnet wurde. Ein Kontrollmechanismus soll so gestaltet sein, dass diese Überprüfungsmöglichkeiten vom Benutzer bestimmt werden können. Bei Nichtübereinstimmung der genannten Kriterien erhält der Benutzer eine entsprechende Fehlermeldung.

Um alle relevanten Daten zu erfassen werden die kontrollierten Klausuren zunächst fotografiert. Bei der anschließenden Digitalisierung aller Schriftzeichen durch die App findet optische Zeichen- bzw. Texterkennung Anwendung. Für die Texterkennung auf externen Servern müssen die entsprechenden Server-Schnittstellen zur Kommunikation implementiert werden.

Die durch die Texterkennung digitalisierten Daten, die Klausuren-Fotos und die digitalen Scan-Vorlagen sollen außerhalb der App auf einem Server gespeichert werden. Somit können alle Benutzer der App die bereits erstellten Scan-Vorlagen verwenden. Wie bereits erwähnt, werden die digitalisierten Daten zudem in ein geeignetes Format gebracht.

Damit Synchronität der Daten zwischen allen Apps und dem Server gewährleistet werden kann, sind auf beiden Seiten Schnittstellen notwendig. Diese sollten den standardmäßigen CRUD-Operationen entsprechen. Für genauere Details zu den Sicherheitsmechanismen und den CRUD-Operationen zum Server siehe im Praktikumsbericht von

Tobias Kallauke.

5 Konzept der Dokumenten-Scanner-App

Dieses Kapitel beschreibt, wie das Softwaresystem die Anforderungen umsetzt.

5.1 Scan-Vorlage erstellen und speichern

Im folgenden Abschnitt wird zuerst verkürzt und anschließend detailliert geschildert, wie eine Scan-Vorlage zu erstellen ist.

Um wirklich nur relevante Daten zu digitalisieren, benötigt es die sogenannten Scan-Vorlagen, die in der App durch den Benutzer angelegt werden können. Eine Scan-Vorlage beinhaltet mehrere Schablonen. Solch eine Schablone wird beispielsweise auf das Deckblatt gelegt, sodass nur noch die relevanten Informationen, die digitalisiert werden sollen, zu sehen sind. Alles andere, was nicht von Interesse ist, wird von der Schablone überdeckt. In der Abbildung 5.1 ist dieses Konzept schemenhaft zu sehen. Der Benutzer kann in zwei groben Schritten eine digitale Schablone anfertigen. Zuerst muss ein Foto von der Seite aufgenommen werden. Anschließend werden die Regionen auf diesem Foto manuell markiert, die digitalisiert werden sollen. Eine Scan-Vorlage ist somit eine Sammlung aller digitalen Schablonen zu einer Klausur, denn jede Seite benötigt eine eigene individuelle Schablone.

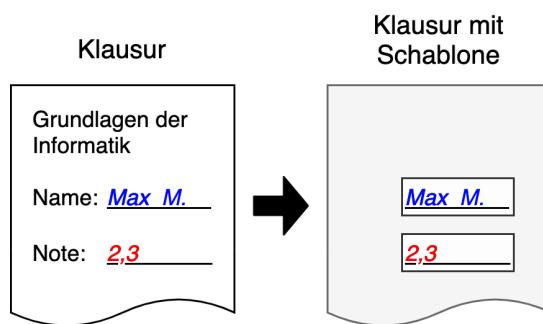


Abbildung 5.1: Schematisches Schaubild zum Verständnis des Schablonen-Konzepts

Nachfolgend wird detailliert das Erstellen einer Scan-Vorlage beschrieben. Der Vorgang ist in Abbildung 5.2 als kompaktes Flussdiagramm und in Abbildung 5.3 schematisch dargestellt.

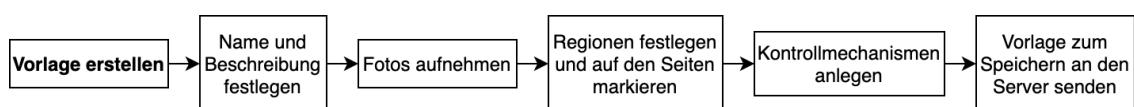


Abbildung 5.2: Kompaktes Flussdiagramm zum Erstellen einer Scan-Vorlage

1. Zu Beginn wird der Scan-Vorlage ein Name und eine Beschreibung gegeben. Anschließend muss jede Seite der Klausur fotografiert werden. Dabei wird in jedem

Bild das Dokument erkannt, vom Hintergrund getrennt bzw. ausgeschnitten und perfekt ausgerichtet. Zudem wird der Kontrast erhöht, sodass die Schrift leichter lesbar ist. Die Abbildung 3.1 resultiert aus dem Prozess (siehe auch Abbildung 6.4b).

2. Anschließend markiert der Benutzer diejenigen Regionen auf jeder Seite, die zu digitalisieren und/oder zu kontrollieren sind. In dem Schema 5.3 sind die Regionen blau hinterlegt (siehe auch Abbildung C.3b). Zusätzlich muss jeder Region eine Bezeichnung und einer der folgenden Datentypen zugeordnet werden: Unbekannt, Vorname, Nachname, Matrikelnummer, Seminargruppe, Punkte und Note (siehe auch Abbildung C.3a und C.3b). Die Angabe des Datentyps ist wichtig, da dadurch die Texterkennung optimiert wird und die Ergebnisse der Texterkennung in das geeignete Format gebracht werden können.
3. Zuletzt wird die Vorlage zum Speichern an einen Server gesendet, wodurch alle Benutzer der App die bereits erstellten Scan-Vorlagen verwenden können.

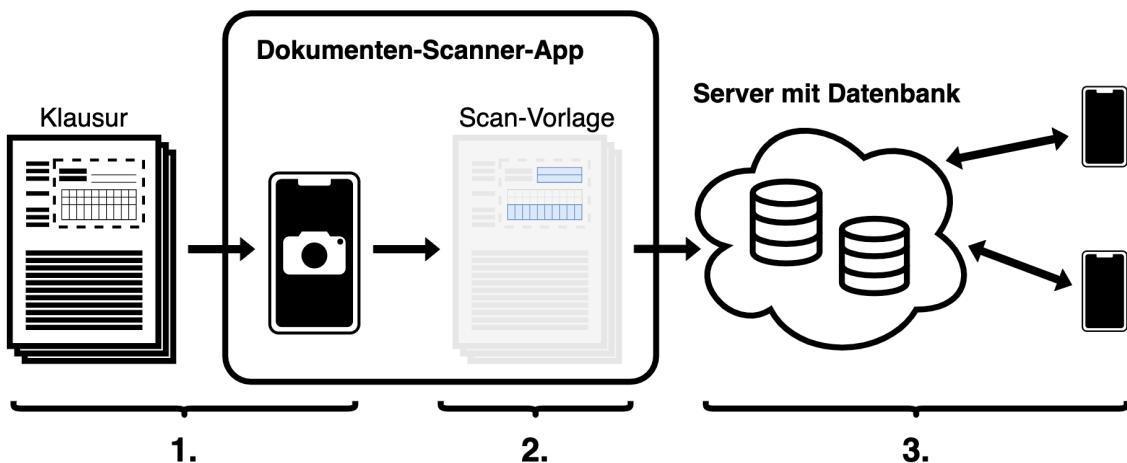


Abbildung 5.3: Schematische Darstellung des Vorgangs Scan-Vorlage erstellen und auf dem Server speichern

Zusätzlich können in der Phase der Erstellung einer Scan-Vorlage die in Kapitel 4 beschriebenen Kontrollmechanismen festgelegt werden. Dies geschieht nach Punkt 2, also nachdem alle Regionen angelegt wurden. Diese Kontrollmechanismen beruhen auf zwei simplen Konzepten.

Das erste Konzept ist der Vergleich. Zur Erinnerung: Es sollen die eingetragenen Punkte auf dem Deckblatt mit den Punkten neben der Aufgabenstellung auf den nachfolgenden Seiten auf Gleichheit überprüft werden. Das zweite Konzept baut auf dem ersten auf. Statt zwei Regionen bzw. Komponenten zu vergleichen, wird eine oder werden beide Komponenten zuvor berechnet. Beispielsweise muss die Summe der Punkte pro Aufgabe mit der Gesamtpunktzahl übereinstimmen. Das bedeutet, es wird erst die Summe berechnet und anschließend mit der vom Prüfer eingetragenen Punktzahl verglichen.

Oder aus der Gesamtpunktzahl wird die Note ermittelt und mit der vom Prüfer eingetragenen Note verglichen.

Beim Festlegen der Kontrollmechanismen wird wie folgt vorgegangen:

1. Der Benutzer wählt einen Kontrollmechanismus aus. Zur Auswahl stehen:
 - der Vergleich zweier Regionen,
 - die Gesamtpunktzahl berechnen und mit der eingetragenen Gesamtpunktzahl vergleichen,
 - oder die Note aus der Gesamtpunktzahl berechnen und mit der eingetragenen Zensur vergleichen.
2. Aus den zuvor angelegten Regionen müssen nun die passenden Regionen ausgewählt werden. Abhängig von dem gewählten Kontrollmechanismus wird dem Benutzer Auskunft darüber gegeben, welche und wie viele Regionen auszuwählen sind.
3. Der Kontrollmechanismus kann gespeichert werden, sobald in jeder zu vergleichenden Komponente die Mindestanzahl an auszuwählenden Regionen erreicht wurde. Diese werden in der Scan-Vorlage hinterlegt und mit an den Server zum Speichern gesendet. So besitzen alle Benutzer dieselben Kontrollmechanismen zu den jeweiligen Scan-Vorlagen.

5.2 Scan-Vorlage verwenden

Im folgenden Abschnitt wird zuerst verkürzt und anschließend detailliert die Verwendung und Funktionsweise einer erstellten Scan-Vorlage beschrieben.

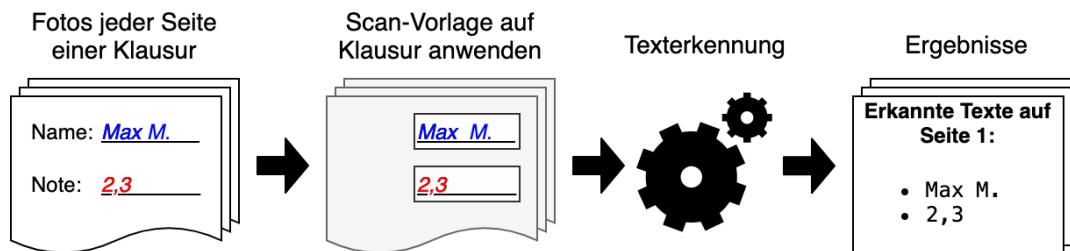


Abbildung 5.4: Schematischer Ablauf beim Verwenden einer Scan-Vorlage

Nach dem Erstellen einer Scan-Vorlage kann diese auf bewerteten Klausuren angewandt werden. In Abbildung 5.4 wird der Ablauf schematisch dargestellt. Zuerst werden die Seiten einer Klausur fotografiert. Anschließend stellen die digitalen Schablonen die relevanten Informationen jeder Seite für den nächsten Schritt zur Verfügung. Optical character recognition (OCR), auf deutsch optische Zeichenerkennung, wird dazu benutzt, die Daten aus den übrig gebliebenen Regionen zu digitalisieren.

Nachfolgend wird detailliert das Verwenden einer Scan-Vorlage beschrieben. Der Vorgang ist in Abbildung 5.5 als kompaktes Flussdiagramm und in Abbildung 5.6 schematisch

tisch dargestellt.

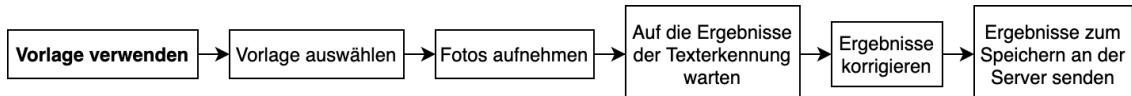


Abbildung 5.5: Kompaktes Flussdiagramm zum Verwenden einer Scan-Vorlage

1. Die Seiten der Klausur müssen dazu in derselben Reihenfolge, wie in der Scan-Vorlage fotografiert werden. Anschließend werden die Regionen der Scan-Vorlage auf das jeweilige Bild angewendet. Bei dem Prozess entstehen Bildausschnitte, in denen die relevanten Informationen enthalten sind.
2. Die entstandenen Bildausschnitte werden seitenweise in die Texterkennung überführt.
3. Die Inhalte der Ausschnitte werden mithilfe von OCR digitalisiert. Der Datentyp der zugehörigen Region aus der Scan-Vorlage nimmt nun Einfluss auf das Ergebnis. Durch ihn wird während der Worterkennungsphase in der Texterkennung eine Liste an vordefinierten möglichen Ergebnissen eingespeist. Diese Liste hat Vorrang vor dem sogenannten Standard-Wörterbuch und verbessert dadurch die Ergebnisse. Ein Beispiel für solch eine Liste können alle möglichen Noten sein. Es ist auch vorstellbar, dass alle Seminargruppen oder Namen von Personen, die an der Klausur teilgenommen haben, dort verwendet werden.
4. Die Ergebnisse werden im Anschluss an den Server gesendet. Zuvor kann der Benutzer jedoch die Ergebnisse anpassen bzw. korrigieren, falls es bei der Texterkennung zu Fehlern kam. Auch die Kontrollmechanismen finden vor dem Hochladen zum Server Anwendung und geben dem Nutzer durch Fehlermeldungen Hinweise.

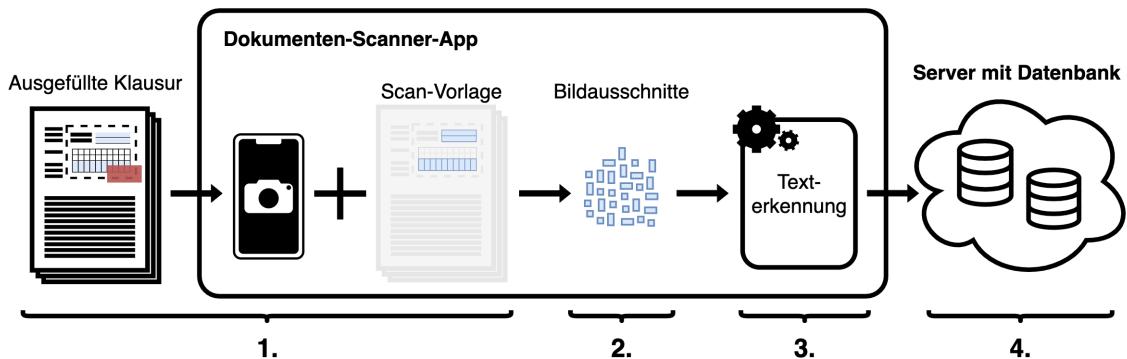


Abbildung 5.6: Schematische Darstellung des Vorgangs beim Verwenden einer Scan-Vorlage

5.3 Erweiterung des Konzepts

Weiter ist eine zusätzliche Korrektur angedacht. Angenommen ein Student verschreibt sich bei seiner Seminargruppe und vergisst ein Zeichen. Die Texterkennung erkennt

zwar jedes Zeichen, jedoch nicht die korrekte Bezeichnung der Seminargruppe. Deshalb sollte nach der Texterkennung die falsche Bezeichnung durch diejenige Seminargruppe ersetzt werden, die die größte Übereinstimmung mit der erkannten Bezeichnung hat. Bei diesem Vorgehen ist zu beachten, dass nur die Seminargruppen verwendet werden sollten, die tatsächlich die Klausur geschrieben haben. Ähnliches ist auch für die Namen oder Matrikelnummern der Studenten möglich.

Allerdings entstehen auch durch das OCR Fehler. Aus diesem Grund wird neben jedem Ergebnis die sogenannte confidence angezeigt. Diese sagt aus, wie sicher die Texterkennung des Ergebnisses ist. Jedoch kann es auch hier falsche Positiv-Ergebnisse geben. Deshalb ist auch die confidence kein perfekter Indikator für die Richtigkeit des Ergebnisses. Deswegen gibt es, wie in Punkt 4 beschrieben, die Möglichkeit, die Ergebnisse noch einmal zu überprüfen und zu korrigieren, bevor sie an den Server zum Abspeichern gesendet werden. Auch die Kontrollmechanismen reagieren auf die Änderungen der Ergebnisse, sodass Fehlermeldungen behoben werden können.

6 Entwicklung des ersten Prototyps

Dieses Kapitel beschreibt einzelne Phasen der Entwicklung der iOS-App in einer ähnlichen Reihenfolge, wie im Wasserfall-Model zur Verwaltung der Entwicklung großer Softwaresysteme nach Winston W. Royce [7] (siehe Abbildung A.1).

6.1 Anforderungsplanung

Vor Beginn des Praktikums wurde diskutiert und kalkuliert, welches Thema aus dem Projekt Memo Space für ein zwölfwöchiges Praktikum geeignet ist. Dabei entstand eine Art Durchführbarkeits- / Machbarkeitsstudie, welche zur Entscheidung führte, das Dokumenten-Scanner-Softwaresystem zu entwickeln. Tobias Kallauke soll aufgrund seiner Kenntnisse und Erfahrungen ein Backend mit entsprechenden Schnittstellen und eine Android-App erstellen, während der Verfasser eine iOS-App programmiert. Die iOS-Anwendung soll den Anforderungen, die im gleichnamigen Kapitel 4 zu finden sind, erfüllen. Weitere Details über den Server, dessen Architektur und die Android-App sind im Praktikumsbericht von Tobias Kallauke beschrieben.

6.2 Analyse und Definition

Die Aufgaben bzw. Ziele dieser Phase sind:

1. die Auseinandersetzung mit der Problemstellung und den Anforderungen,
2. die Durchführung einer Problemanalyse,
3. die Entwicklung von Ideen und eines genauen Konzepts der iOS-App sowie
4. das Entwickeln eines ersten minimalen Prototyps als Machbarkeitsnachweis.

Bei der Entstehung des Konzepts, welches im Kapitel 5 zu finden ist, spielen die Dokumentationen der Frameworks Vision¹, VisionKit² und PhotoKit³ von Apple eine entscheidende Rolle. Aus ihnen geht hervor, welche Funktionen der App noch zu entwickeln und welche in Frameworks schon vorhanden sind. Z. B. ist das Erkennen und das Geradeziehen von Dokumenten in Echtzeit sowie die Texterkennung auf Bildern in Vision und VisionKit enthalten. Bei der Entwicklung des ersten minimalen Prototyps half eine Beispiel-App von Apple [1], die das Erkennen von Objekten in Standbildern mithilfe der genannten Frameworks umsetzt.

Durch die Auseinandersetzung mit den Bibliotheken konnte festgestellt werden, dass

¹ Dokumentation von Vision - <https://developer.apple.com/documentation/vision>

² Dokumentation von VisionKit - <https://developer.apple.com/documentation/visionkit>

³ Dokumentation von PhotoKit - <https://developer.apple.com/documentation/photokit>

die zu entwickelnde App nur Geräte ab der Betriebssystem-Version iOS 13.0 unterstützen wird. Grund dafür sind die Apple Frameworks SwiftUI und VisionKit.

SwiftUI bietet die Möglichkeit, Benutzeroberflächen für alle Apple-Plattformen in der Programmiersprache Swift zu erstellen. Die deklarative Swift-Syntax ist einfach zu lesen und schnell zu schreiben, sodass es möglich ist, die App in wenigen Wochen für iPhone und iPad zu entwickeln. Als Alternative gibt es UIKit⁴, welches unter allen iOS Versionen funktioniert. Dieses Framework ist allerdings nicht deklarativ, sodass Views sowohl im Code, als auch in Interface-Dateien getrennt voneinander erstellt und konfiguriert werden müssen [8]. Dadurch dauert die Entwicklung einer App im Gegensatz zu SwiftUI deutlich länger, wie man auch in dem Video⁵ von Paul Hudson, einem in der Swift-Community sehr bekannten Programmierer und Autor, sieht. Sehr ähnliche Erfahrung hat der Verfasser vor dem Praktikum in seiner Freizeit mit den beiden Frameworks gesammelt.

VisionKit dagegen ist das Framework zum Scannen der Dokumente. Auch hierfür existiert eine Alternative, welche auch auf älteren iOS Versionen funktioniert. Allerdings unterstützt das Framework WeScan⁶ noch kein stapelweises Scannen. Das bedeutet, dass man immer nur ein Foto machen kann, welches erst abgespeichert werden muss, bevor man das nächste aufnehmen kann. Das ist für den Benutzer umständlich und zeitintensiv. Zu Informationen anderer verwendeter Frameworks siehe im Kapitel 6 Abschnitt 6.5.

Zur Projektplanung und zum Projektmanagement wurde das Scrum-Konzept⁷, welches sich für agile Softwareentwicklung anbietet, verwendet. Als Versionsverwaltung der App-Software wurde Git⁸ in Kombination mit GitHub Issues⁹ und GitHub Project Boards als Planungstools benutzt. Dadurch konnte der Verfasser jedem sogenannten Sprint selbstständig Aufgaben zuordnen und den Fortschritt nachvollziehen. Informationen über den Server und der Android-App zu diesem Thema sind im Praktikumsbericht von Tobias Kallauke zu finden.

6.3 Grundlagen

In den folgenden zwei Absätzen werden wichtige Konzepte erläutert und Grundwissen vermittelt, die im nachfolgendem Abschnitt 6.4 nochmals thematisiert werden.

Model-View-ViewModel Model-View-ViewModel (MVVM) ist ein Software-Architekturmuster. Unter einem Software-Architekturmuster oder auch Entwurfsmuster ist eine bewährte

⁴ Dokumentation von UIKit - <https://developer.apple.com/documentation/uikit>

⁵ SwiftUI vs UIKit – Comparison of building the same app in each framework - <https://www.youtube.com/watch?v=qk2y-TiLDZo>

⁶ WeScan GitHub Repository - <https://github.com/WeTransfer/WeScan>

⁷ Der Scrum Guide™ - <https://www.scrumguides.org/scrum-guide.html>

⁸ Git Internetseite - <https://git-scm.com/>

⁹ Mastering Issues - <https://guides.github.com/features/issues/>

Lösungsvorlage für wiederkehrende Entwurfsprobleme zu verstehen. MVVM erleichtert die Trennung der Entwicklung der grafischen Benutzeroberfläche (der View in Abbildung 6.1) von der Entwicklung der Geschäftslogik oder der Back-End-Logik (dem Model). So ist die View nicht von einer bestimmten Model-Plattform abhängig. MVVM verwendet das Konzept eines Schichtmodells¹⁰ und ist eine abstrakte Darstellung einer Benutzeroberfläche in Form einer Datenstruktur. Diese Struktur enthält Daten, die auf der Benutzeroberfläche angezeigt werden sollen und Anweisungen, die auf der Benutzeroberfläche aufgerufen werden können. Dieses sogenannte ViewModel besitzt keine direkte Verbindung zur View, wie es sonst bei anderen Entwurfsmustern üblich ist, um Daten auf der Benutzeroberfläche anzuzeigen. Stattdessen verwendet eine MVVM-View eine Bindungsfunktion (data binging) zur bidirektionalen Zuordnung von Daten aus dem ViewModel zu den jeweiligen Objekten auf der View. Z. B. bildet eine Liste von Zahlen im ViewModel die Einträge in einem Dropdown-Menü auf der View ab. Aber auch das Binden von Daten aus dem Model an Benutzereingaben durch Maus, Tastatur oder Touch-Screens ist möglich. Beispielsweise kann ein Mausklick eine Anweisung in dem ViewModel auslösen. Die Anweisung besorgt Daten aus dem Model, wodurch die View durch data binding aktualisiert wird. [6] [4] [3]

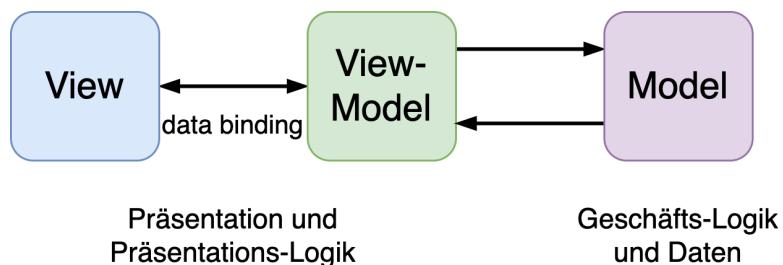


Abbildung 6.1: Schematische Darstellung des Model-View-ViewModel Konzepts

Redux.js Redux.js ist eine Bibliothek¹¹ für die Programmiersprache JavaScript und stellt einen sogenannten vorhersehbaren Zustandscontainer¹² für JavaScript-Anwendungen bereit. In diesem Container wird der Zustand der gesamten Anwendung in einem sogenannten Objektbaum gespeichert. Dieser Baum ist mit der Ordner-Struktur von Windows vergleichbar. Es gibt einen Wurzel-Ordner, in dem wiederum Ordner oder aber auch Dateien enthalten sind. Im Zustandscontainer bezeichnet man die Ordner als States bzw. auf deutsch Zustände. Die Dateien sind einfache Daten, die unter anderem auf der Benutzeroberfläche angezeigt werden sollen. Diese Aufteilung in die States ist dazu gedacht, jeder View nur die wirklich benötigten Daten bereitzustellen. Diese Struktur erleichtert das Testen oder Untersuchen der Anwendung und ermöglicht es, durch Hinzufügen eines neuen States den aktuellen Entwicklungsstand der Anwendung beizubehalten und dadurch den Entwicklungsprozess zu beschleunigen.

¹⁰ Geschichtete Anwendung - [https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650258\(v=3dpanp.10\)](https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650258(v=3dpanp.10))

¹¹ Redux.js Github Repository - <https://github.com/reduxjs/redux>

¹² Redux Core Concepts - <https://redux.js.org/introduction/core-concepts>

Ein weiteres wichtiges Prinzip¹³ von Redux ist, dass die Daten in den States schreibgeschützt sind. Die einzige Möglichkeit den Zustand bzw. die Daten zu ändern, besteht darin, eine Aktion auszulösen. Die Aktion beschreibt, wie der State und dessen Daten sich ändern. Dadurch wird sichergestellt, dass weder die Views noch Netzwerk-Rückrufe jemals direkt den Zustand ändern können. Stattdessen bringen sie nur die Absicht zum Ausdruck, den State zu verändern und lösen eine Aktion aus, die die Änderung für sie vornimmt. Da alle Aktionen zentralisiert sind und eine nach der anderen in einer strikten Reihenfolge erfolgen, gibt es weniger Programmfehlerquellen.

6.4 Entwurf und Design

In der Entwurfsphase wird normalerweise über die Datenhaltung, die Verteilung des Software-Systems im Netz und die Benutzeroberfläche entschieden¹⁴. Jedoch standen diese Grundsatzentscheidungen schon zu Beginn des Praktikums durch die Kenntnisse von Tobias Kallauke und des Verfassers fest. Zur Datenspeicherung verwendet der Server PostgreSQL¹⁵, eine relationale Datenbank. Zum Speichern der Bilder wird die Server-Verzeichnisstruktur genutzt. In der App hingegen werden keine Daten persistent gespeichert. Durch die Aufteilung von App und Server handelt es sich um eine sogenannte Client-Server-Architektur und für die Benutzeroberfläche der iOS-App wird, wie schon erwähnt, das deklarative SwiftUI verwendet. Weitere Informationen zum Entwurf der Server-Architektur und der Android-App befinden sich im Praktikumsbericht von Tobias Kallauke.

Zur Definition einzelner Teil-Workflows, die zusammenhängende Aufgaben umfassen, wurden Flussdiagramme und schriftliche Zustandsautomaten angefertigt (siehe Anhang B). Diese Modelle halfen dabei Views zu entwickeln und deren Design festzulegen. Das Aussehen der App sollte sich jedoch im Laufe der Zeit noch ändern, da erst einmal die Umsetzung des Konzepts im Vordergrund stand.

Ausgehend vom Design und den Teil-Workflows entstand ein grobes Konzept zur Handhabung der Daten innerhalb der App. Da SwiftUI das Entwurfsmuster MVVM umsetzt, wird eine Struktur für das ViewModel und für den Datenfluss der asynchronen Server-Rückrufe benötigt. Die Struktur-Entwicklung erwies sich bereits während des ersten Prototyps als problematisch. Denn auch bei steigender Komplexität sollte das ViewModel noch übersichtlich bleiben, um eine schnelle Weiterentwicklung zu gewährleisten. Bei der Suche nach einer besseren Lösung wurde die JavaScript-Bibliothek Redux.js zur Verwaltung von Zustandsinformationen in Webanwendungen gefunden.

¹³ Redux Three Principles - <https://redux.js.org/introduction/three-principles>

¹⁴ Vorlesung 9, Folie 27 aus Softwaretechnik Grundlagen von Prof. Dr.-Ing. Wilfried Schubert (2019) - https://www.staff.hs-mittweida.de/~wschub/intranet/ss19/Fach_SWT/Fach_SWT_Zeitplan.htm

¹⁵ PostgreSQL Internetseite - <https://www.postgresql.org/>

Das Konzept von Redux hilft komplexe Views mit vielen Daten schnell und einfach zu implementieren. Somit werden Server-Antworten und zwischengespeicherte Daten sowie lokal erstellte Daten, die noch nicht auf dem Server gespeichert wurden, strukturiert und zentral abgelegt. Das erleichtert nicht nur das Wiederverwenden von Daten, sondern reduziert die Wiederholung von Server-Aufrufen. Zudem ermöglicht Redux durch die modulare Aufteilung des State-Containers eine schnelle Weiterentwicklung der App, trotz steigender Komplexität. Zudem stellten die Vorteile hinsichtlich des Testens und Untersuchens der App, das schnelle Auffinden von Fehlern in Aussicht. Es erschien nun möglich, trotz begrenzter Praktikumszeit, den Prototyp möglichst fehlerfrei und weit zu entwickeln.

Daher erschien es sinnvoll die wesentlichen Konzepte von Redux umzusetzen und einen Redux ähnlichen State-Container, als ViewModel zu implementieren. Aus der Definition von Teil-Workflows sollte der Container oder auch Store genannt mindestens 5 States haben:

- Für Authentifizierung sowie Registrierung,
- für das Anlegen von Scan-Vorlagen und Speichern der Zwischenergebnisse,
- für das Ausführen von Server-Aufrufen zum Speichern und Abrufen von Vorlagen,
- für die Steuerung des Workflows bzw. den stellvertretenden Views sowie
- für sonstige Daten, die sehr häufig verwendet werden.

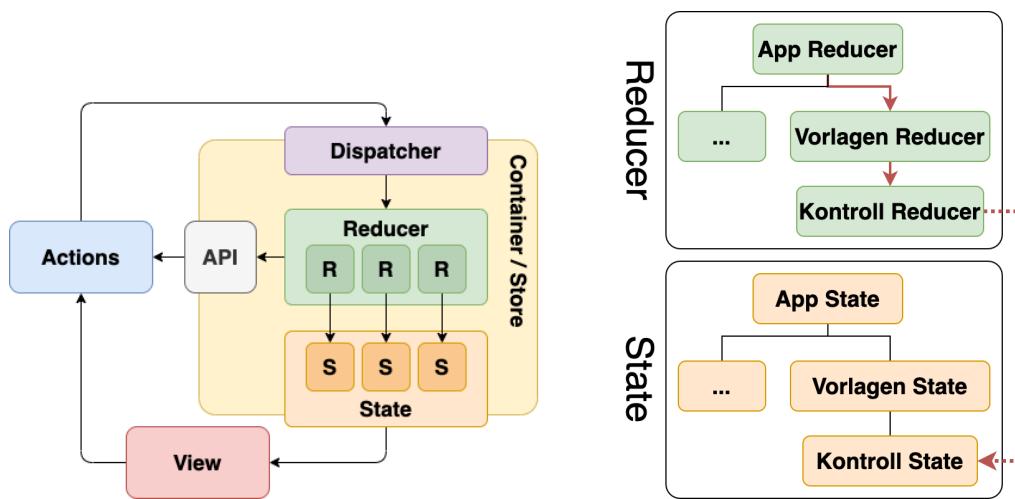


Abbildung 6.2: Architektur Schemata

Das Schema 6.2a stellt die in der App umgesetzte Struktur dar. Über eine View (rot) können Aktionen (blau) aufgerufen werden. Diese gelangen zuerst in einen sogenannten Dispatcher (lila), welcher als Verteiler dient. Anhand der Art einer Aktion, wird der entsprechende Reducer (grün) vom Dispatcher beauftragt, die Aktion auszuführen. Je-weils ein Reducer ist nur für eine Aktion-Art zuständig. In Abbildung 6.2b ist eine Schachtelung von Reducern (grün) zu sehen. Ebenfalls können Aktion-Arten ineinander geschachtelt werden. Das hat den Hintergrund, dass die States wie ein Objekt-

baum aufgebaut sind. So hat jeder State seinen eigenen Reducer und eigene Aktionen, was für die oben erwähnte Modularität sorgt. Muss beispielsweise eine Änderung im Kontroll-Mechanismen-State (siehe Kontroll-State in 6.2b) vorgenommen werden, wird die Haupt-Aktion in einer Kontroll-Mechanismen-Reducer-Aktion geschachtelt und diese wiederum in einer Vorlagen-Reducer-Aktion. Zum Ausführen der Haupt-Aktion, werden zuvor die entsprechenden Reducer die Schachtelung von außen nach innen auflösen. Nachdem die Aktion ausgeführt und eine State-Änderung herbeigeführt wurde, aktualisiert sich durch data binding von MVVM die View. Speziell werden alle Views, die eine Bindung zu dem jeweiligen Datum im State haben, über die Änderung benachrichtigt und daraufhin aktualisieren diese ihre Benutzeroberfläche.

Ein weiterer wichtiger Bestandteil der Architektur sind die Server-Aufrufe. Für diese gibt es einen eigenen Reducer und eigene Schnittstellen. Diese Schnittstellen sind in der Abbildung 6.2a grau markiert und mit API beschriftet. Ihre Besonderheit besteht darin, selbst Aktionen an den Container zu senden. Beispielsweise löst der Benutzer die Aktion aus, alle Vorlagen vom Server zu laden. Diese Aktion gelangt über den vorgesehenen Reducer zu der API-Schnittstelle. Dort wird ein entsprechender Server-Aufruf gestartet und auf den Server-Rückruf gewartet. Sobald die Antwort des Servers angekommen ist, wird diese mit Hilfe einer Aktion zurück zum Container gesendet. Dort verarbeitet ein entsprechender Reducer eventuelle Fehler oder die heruntergeladenen Vorlagen.

6.5 Implementierung

Zur Realisierung der entworfenen Systemkomponenten wurde ausschließlich, die von Apple entwickelte IDE Xcode¹⁶, verwendet. Diese stellt virtuelle iOS oder iPadOS Geräte zur Verfügung, auf denen die App getestet werden kann. Diese Simulatoren bieten unter anderem auch die Möglichkeit, die Anwendung schnell auf verschiedenen Betriebssystem-Versionen zu testen. Auch ist das Testen der App auf echten Geräten durch Xcode möglich, was bei der Entwicklung unumgänglich war. Grund dafür ist die Benutzung der Kamera, die bei den Simulatoren zu gewollten Abstürzen führt, da diese keinen Zugriff auf ein Kamerasystem besitzen.

Die App ist ausschließlich in der Programmiersprache Swift geschrieben. Neben den Frameworks SwiftUI, Vision und VisionKit von Apple wurde das Framework Kingfisher¹⁷, zum Downloaden und Cachen von Bildern verwendet. Das Vision-Framework führt Gesichts-, Text- und Barcode-Erkennung, Bildregistrierung und allgemeine Merkmalsverfolgung durch. Jedoch wurde in der App nur die integrierte Texterkennung benutzt. Es ist aber anzunehmen, dass zur Dokumenten-Erkennung Algorithmen aus Vision von VisionKit verwendet werden. Für mehr Informationen über die Frameworks SwiftUI und VisionKit siehe in Abschnitt 6.2.

¹⁶ Xcode Internetseite - <https://developer.apple.com/xcode/>

¹⁷ Kingfisher GitHub Repository - <https://github.com/onevcat/Kingfisher>

Zum Einrichten und Testen des Backends wurde außerdem Visual Studio Community bzw. Visual Studio Code¹⁸ mit dem REST Client Plugin¹⁹ verwendet. Das Backend konnte als lokaler Server auf dem Entwicklungscomputer gestartet und im lokalen Netzwerk aufgerufen werden. Für die Verwaltung der PostgreSQL-Datenbank bot sich PgAdmin²⁰ an. Damit ist es möglich, einzelne Einträge oder auch ganze Tabellen der Datenbank zu bearbeiten.

6.5.1 Registrieren und Anmelden

Die E-Mail-Textfelder besitzen, wie auch das Passwortfeld der Anmeldung eine AutoFill-Funktion. Dadurch kann die App Registrier- und Anmelddaten vorschlagen und automatisch in die entsprechenden Felder einsetzen. Im dritten Bild 6.3c ist das anhand des „Passwörter“-Knopfes über der Tastatur zu sehen. Außerdem sind alle Passwortfelder gesichert, wodurch der Inhalt zum Schutz der Privatsphäre standardmäßig ausgeblendet und mit Punkten ersetzt wird. Aufgrund der Sicherheitsstandards von iOS sind die Punkte auf der Bildschirmaufnahme 6.3b nicht zu sehen. Durch Berühren des Auges neben dem Textfeld kann der Inhalt ein- und ausgeblendet werden. Zusätzlich validieren alle Textfelder ihren Inhalt mithilfe von regulären Ausdrücken. Beispielsweise muss ein Passwort aus 8 Zeichen bestehen und zwei von den drei folgenden Eigenschaften erfüllen:

1. Es ist mindestens ein Sonderzeichen enthalten.
2. Es ist mindestens ein Großbuchstabe enthalten.
3. Es ist mindestens eine Zahl enthalten.

Die Validierung ist jedoch nicht standardmäßig, sondern wurde selbst entwickelt.

6.5.2 Scan-Vorlagen erstellen und speichern

Bei der Implementierung des ersten Arbeitsschrittes der Scan-Vorlagen wurde das Flussdiagramm B.1, welches im Anhang B zu finden sind, benutzt.

Scan-Vorlage erstellen

Zu Beginn sind ein Name und eine Beschreibung der Vorlage anzugeben (siehe 6.4a), um im Anschluss die Fotos aufzunehmen. Bei der Kamera-View (siehe 6.4b) handelt es sich um die Scan-View des Frameworks VisionKit. Mithilfe von Kantenerkennung und anderen Algorithmen, die Tobias Kallauke in seinem Bericht beschreibt, wird das Dokument, sobald es erkannt ist, automatisch fotografiert. Das Dokument wird in Echtzeit aus dem Bild ausgeschnitten und gerade gezogen. Ein manuelles Auslösen des Fotos

¹⁸ Visual Studio Internetseite - <https://visualstudio.microsoft.com/de/>

¹⁹ REST Client Repository - <https://github.com/Huachao/vscode-restclient>

²⁰ PgAdmin Internetseite - <https://www.pgadmin.org/>

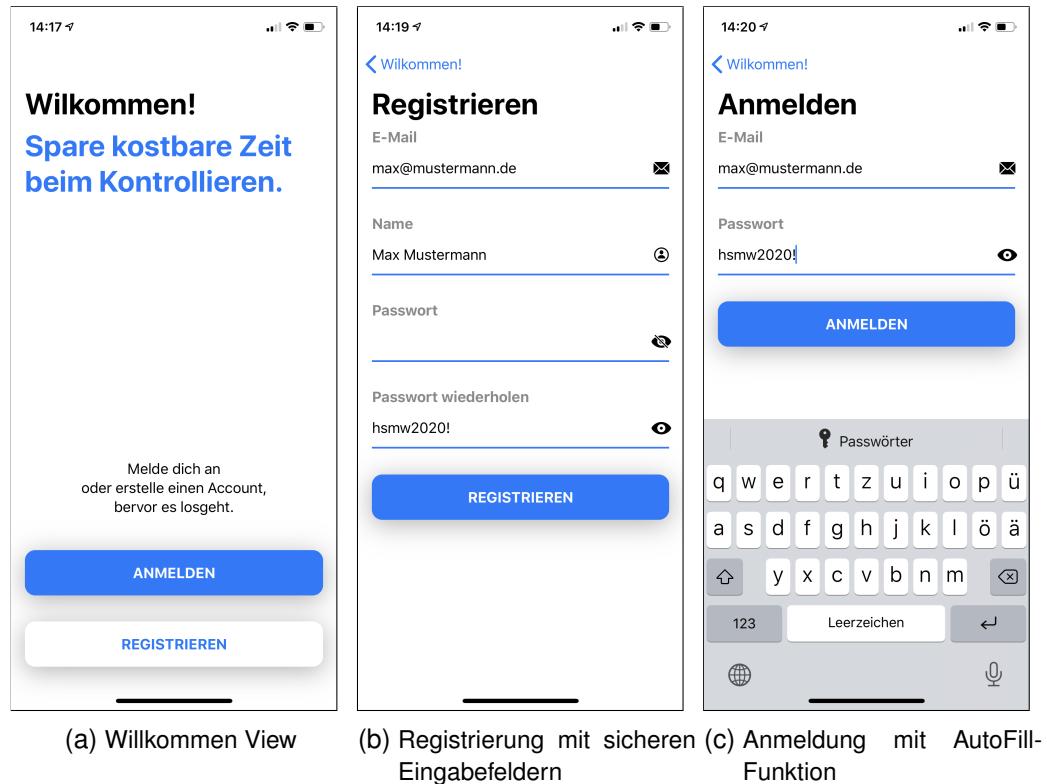


Abbildung 6.3: Willkommen-, Registrier- und Anmelde-View

und Anpassen der Dokumentenkanten im Bild, ist ebenfalls möglich. Zugleich werden alle erstellten Bilder in einer Gruppe gesammelt. Diese können vor dem Abspeichern angeschaut und nochmals bearbeitet werden. Die Abbildung 3.1 entstand durch diesen Prozess.

Regionen erstellen

Im Anschluss sind die Regionen auf den Dokumentenseiten zu markieren, deren Inhalt beim Einstellen digitalisiert werden soll. Dafür wird zunächst die gewünschte Seite aus einer Übersicht (siehe 6.4c) ausgewählt. Anschließend ist eine Vorschau der Seite mit allen eingetragenen Regionen zu sehen (siehe C.3a bzw. C.3b). Über einen Button können weitere Regionen hinzugefügt werden. Dazu muss ein Name (siehe 6.5a) und ein Datentyp (siehe 6.5b) festgelegt werden. Sobald die Eigenschaften der Region definiert sind, muss diese Region auf dem Bild markiert werden. Dazu zieht man mit einem Finger ein Rechteck über die gewünschte Region (siehe 6.5c). Zudem kann an die Seite heran- oder herausgezoomt werden, um beispielsweise kleine Regionen zu markieren. Dieses Vorgehen muss anschließend für alle benötigten Regionen auf den jeweiligen Seiten wiederholt werden.

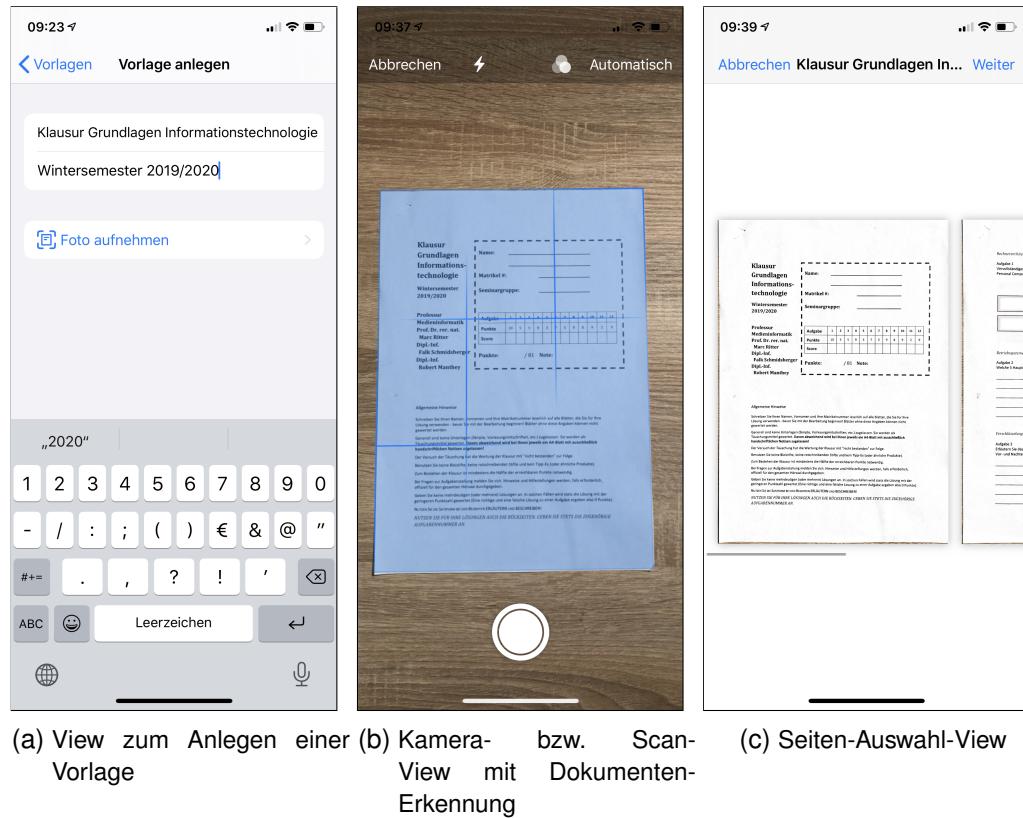


Abbildung 6.4: Die ersten Views zur Erstellung einer Scan-Vorlage

Kontrollmechanismen erstellen

Schließlich können die Kontrollmechanismen erstellt werden. Diese Funktion ist allerdings noch nicht sehr weit fortgeschritten und beinhaltet aktuell nur den Kontrollmechanismen-Typ zum Vergleichen von Regionen, wie im Kapitel 5 beschrieben ist. Beim Erstellen einer Kontrolle dieses Typs müssen zwei Regionen aus gewählt werden (siehe Abbildung 6.6b), deren Inhalt nach der Texterkennung auf Gleichheit überprüft wird. Bei der Wahl der zwei Regionen stehen alle vorher angelegten Regionen, wie in Abbildung 6.6c zu sehen ist, zur Verfügung. In Abbildung 6.6a sind die bereits angelegten Kontrollmechanismen dieser Scan-Vorlage zu sehen sowie ein Button zum Speichern der Vorlage.

Scan-Vorlage speichern

Die Speicherung erfolgt in mehreren Schritten. Dazu werden die entsprechenden Server-Schnittstellen in einer bestimmten Reihenfolge aufgerufen. Genauere Details zur API des Servers sind im Praktikumsbericht von Tobias Kallauke zu finden.

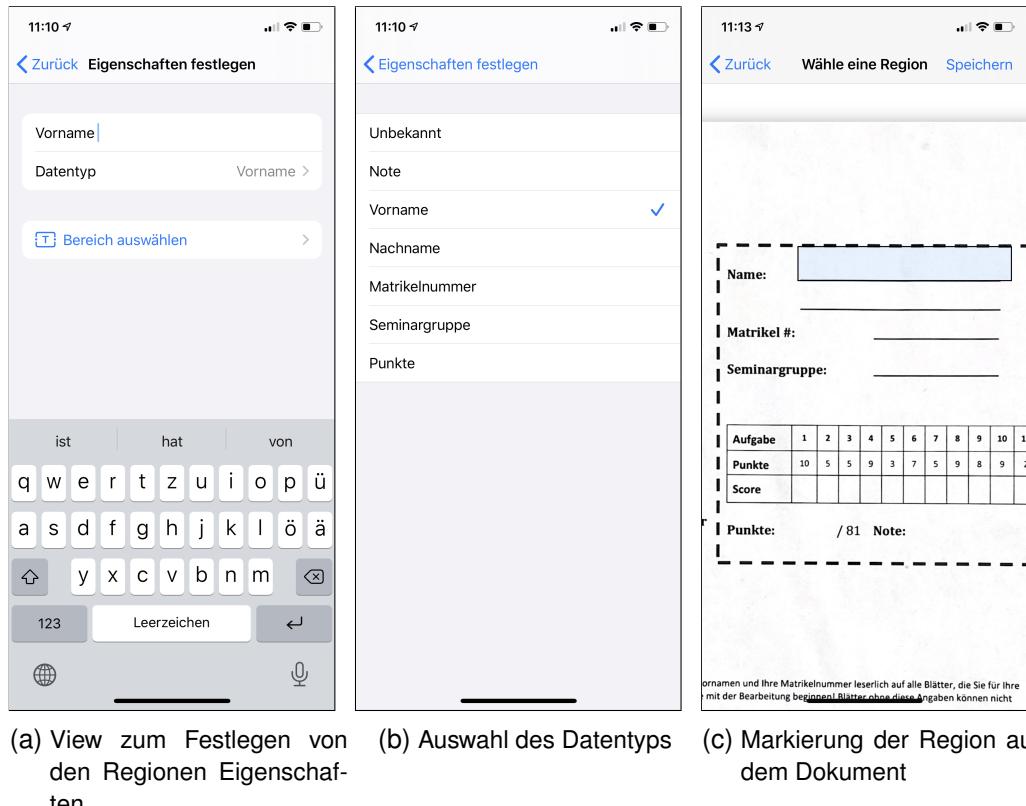


Abbildung 6.5: Views zur Erstellung von Regionen

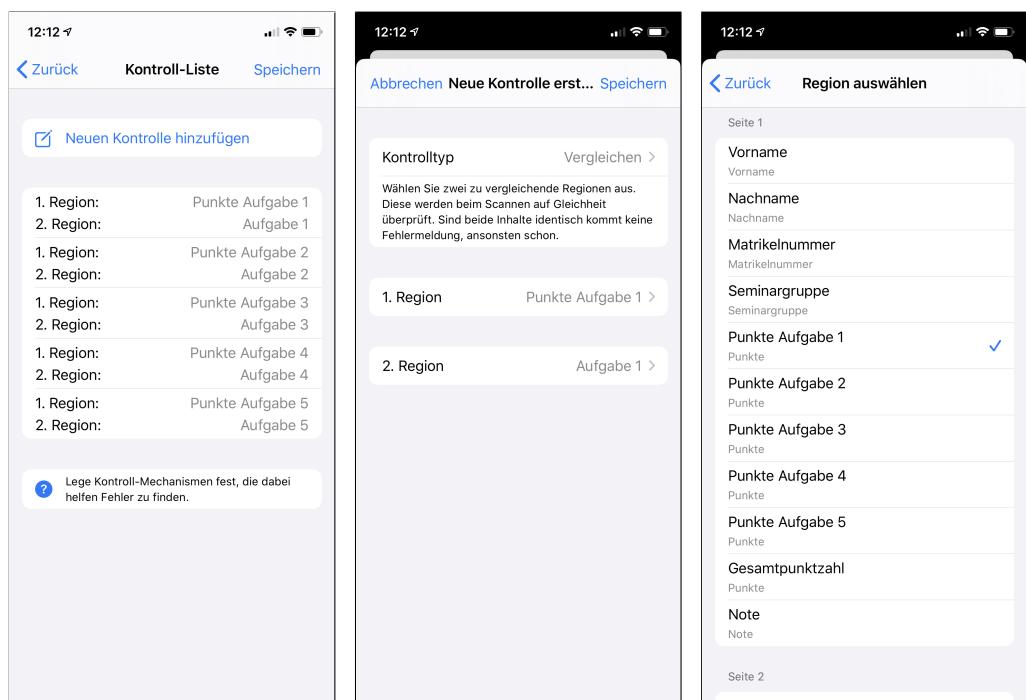


Abbildung 6.6: Views zur Erstellung von Kontrollmechanismen

6.5.3 Scan-Vorlage verwenden

Bei der Implementierung dieses Arbeitsschrittes wurde das Flussdiagramm B.2, welches im Anhang B zu finden ist, benutzt.

Um eine Klausur zu digitalisieren, muss die vorher erstellte Scan-Vorlage ausgewählt werden (6.7a). Danach ist das Einscannen einer Klausur über einen Button möglich (6.7b). Bevor die Bilder aufgenommen werden können, erscheint ein Dialogfenster (6.7c). Der Benutzer muss entscheiden, welche OCR-Engine benutzt werden soll. Aktuell gibt es zwei Möglichkeiten:

- Die OCR-Engine des Vision-Frameworks, welche direkt auf dem Gerät und ohne eine Internetverbindung die Texterkennung durchführt,
- oder die OCR-Engine Tesseract, die über eine Schnittstelle des Servers aufzurufen ist.

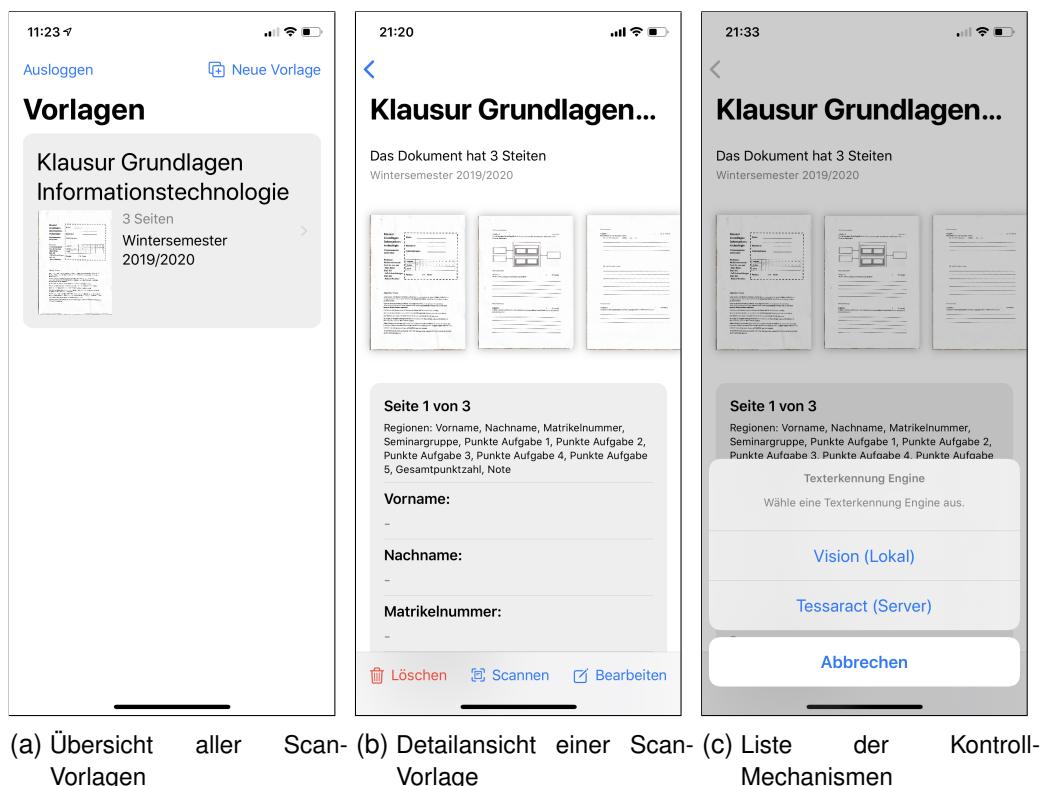


Abbildung 6.7: Dialogfenster zur Auswahl der OCR-Engine

Nun erscheint die Scan-View und die Bilder können aufgenommen werden. Wichtig dabei ist, dass die Seiten beim Fotografieren dieselbe Reihenfolge, wie in der Scan-Vorlage haben. Auch muss die Anzahl der eingescannten Seiten mit denen in der Vorlage übereinstimmen. Mithilfe der Scan-View wird die Seite aus dem Bild ausgeschnitten, geglättet und der Kontrast des Bildes so verändert, dass Schriftzeichen leichter zu erkennen sind und Schatten bzw. Falten verschwinden.

Nach dem Einscannen der Klausurseiten beginnt der Prozess der Texterkennung. Durch

Aktivitätsanzeigen wird dem Benutzer mitgeteilt, dass die Anwendung gerade arbeitet. Die Abläufe der online OCR-Engine und des Vision Frameworks unterscheiden sich deutlich.

Ablauf mit Vision

Bei der Verwendung von Vision gibt es folgende grobe Schritte:

1. Die markierten Regionen werden in der Vorlage auf die neuen Fotos angewendet.
2. Mithilfe der Regionen werden Bildausschnitte aus den Fotos erstellt.
3. Die Ausschnitte werden in die ausgewählte OCR-Engine überführt.
4. Darstellung der Texterkennungsergebnisse auf der View.

Zu 1.: Die markierten Regionen werden in der Vorlage auf die neuen Fotos angewendet. Nun wird jedes Foto des neuen Scans mit den Regionen der jeweiligen Seite der Scan-Vorlage verarbeitet. Wie in den Abbildungen 3.1a und 3.1b zu sehen ist, haben beide, durch die App entstandenen Bilder, unterschiedliche Maße. Die Abbildung 3.1a ist größer und etwas breiter als die Abbildung 3.1b. Das liegt daran, dass sich der Winkel und der Abstand der Kamera zum Dokument zwischen den Aufnahmen geändert hat. Wenn die Bilder in der Vorlage nicht genau so groß sind, wie die Bilder des neuen Scans, können wichtige Informationen durch die absoluten Positionen der Regionen abgeschnitten werden. Aus diesem Grund müssen nun die absoluten in relative Positionen umgerechnet werden. Gleiches gilt für die Höhe und Breite der Regionen.

Zu 2.: Mithilfe der Regionen werden Bildausschnitte aus den Fotos erstellt. Nachdem die neue Position und die Maße berechnet wurden, entstehen daraus Bildausschnitte für die Texterkennung. Bei falscher Erstellung der Vorlagen oder bei falschem Einscannen der Klausur kann es trotzdem passieren, dass Teile der Regionen wegfallen. Dazu im Unterabschnitt 7.2.1 mehr.

Zu 3.: Die Ausschnitte werden in die ausgewählte OCR-Engine überführt. Zu jedem Bildausschnitt wird eine sogenannte Texterkennungsanfrage konfiguriert. Hierfür wird weiterhin die zugehörige Region benötigt. In der Konfiguration werden an Hand des Region-Datentyps (6.5b) unterschiedliche Einstellungen getroffen. Beispielsweise wird beim Datentyp Note eine sogenannte costum words-Liste verwendet. Sie beinhaltet alle zu erwartende Ergebnisse bzw. Noten. Die Liste hat in der Worterkennungsphase Vorrang vor dem Standard-Wörterbuch, wie schon im Kapitel 5 erwähnt und sorgt für bessere Ergebnisse²¹. Standardmäßig werden bei der Bearbeitung einer Texterkennungs-

²¹ Vision *costumWords* Dokumentation <https://developer.apple.com/documentation/vision/vnrecognizetextrequest/3152640-customwords>

anfrage zunächst alle Zeichen im Eingabebild lokalisiert und dann jede Zeichenfolge analysiert²². Anschließend wird eine Korrektur der erkannten Wörter auf der Grundlage eines Wörterbuchs und anderer Wahrscheinlichkeitsheuristiken für Zeichenpaare durchgeführt [2]. Der genaue Ablauf der Texterkennung bzw. die verwendeten Algorithmen von Vision sind allerdings nicht bekannt.

Nach der Texterkennung durch Vision erfolgt in einigen Fällen noch eine selbst entwickelte Korrektur, wie sie in Kapitel 5 Abschnitt 5.3 beschrieben ist. Wieder ist der Regionen-Datentyp (6.5b) ausschlaggebend. Beispielsweise wird der Text einer vermeintlichen Seminargruppe mit allen in der App hinterlegten Seminargruppen verglichen. Das erkannte Wort wird durch den Seminargruppennamen mit der größten Übereinstimmung ersetzt und die confidence gegebenenfalls angepasst.

Zuletzt werden die erkannten Texte bzw. Worte und deren confidence an den State-Container der App gesendet. Für die Ergebnisse der Texterkennung ist ein eigener State vorgesehen, in dem auch die Serverergebnisse von Tesseract gesichert werden. Somit ist das Anzeigen der Ergebnisse bei beiden OCR-Engines gleich.

Zu 4.: Darstellung der Texterkennungsergebnisse auf der View. Die Ergebnisse der Texterkennung und die jeweilige confidence werden auf der Benutzeroberfläche, nach Seiten sortiert, angezeigt (6.8a). Alle OCR-Ergebnisse können zudem noch angepasst werden. Abhängig vom Datentyp der Region passt sich die Tastatur beim Ändern der Texte an (6.8b). Beispielsweise sind die Tastaturen beim Anpassen von Noten und Punkten auf Zahlen und Dezimalpunkte spezialisiert. Zudem kann die confidence farblich (siehe Abbildung 6.8a), aber auch als Zahlenwert (siehe Abbildung 6.8b) angezeigt werden. Wurde kein Text erkannt, ist dementsprechend ein Warnsymbol an der Stelle der confidence zu sehen.

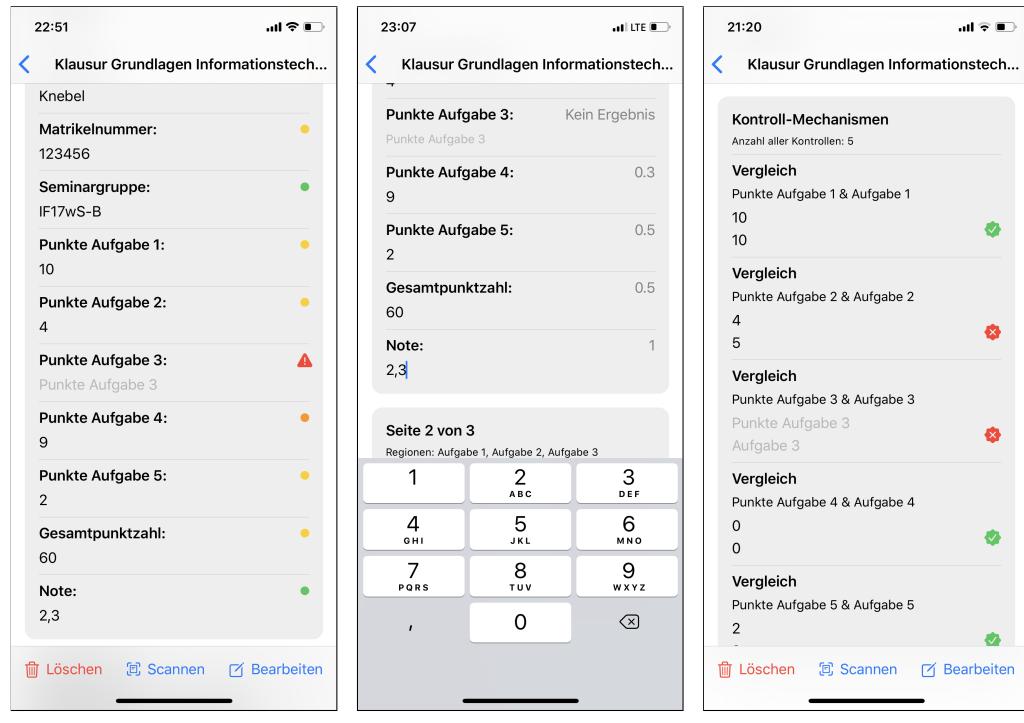
Ablauf mit Tesseract

Die Texterkennung mit Tesseract folgenden Ablauf:

1. Bilder an den Server senden.
2. Auftrag zur Texterkennung an den Server senden.
3. Serverseitige Texterkennung durchführen.
4. Texterkennungsergebnisse empfangen und darstellen.

Zu 1.: Bilder an den Server senden. Nachdem das Dokument fotografiert wurde, werden die Aufnahmen an den Server gesendet. Dieser gibt die Speicherpfade der

²² Quelle: *VNRecognizeTextRequest* Dokumentation <https://developer.apple.com/documentation/vision/vnrecognizetextrequest>



(a) Übersicht nach der Texterkennung mit farblicher confidence
 (b) Änderung der Note mit spezieller Tastatur für Dezimalzahlen und confidence als Zahlenwert
 (c) Dialogfenster zur Auswahl der OCR-Engine

Abbildung 6.8: Listen- und Detail-Ansicht von Scan-Vorlagen

Bilder als Antwort zurück.

Zu 2.: Auftrag zur Texterkennung an den Server senden. Anschließend wird für jeden Pfad ein Auftrag zur Texterkennung an die OCR-Schnittstelle des Servers ausgelöst. In der Anfrage wird der Pfad des Bildes und die passende Seiten-ID der ausgewählten Vorlage gesendet. Nun übernimmt der Server die Arbeit der Texterkennung.

Zu 3.: Serverseitige Texterkennung durchführen. Mithilfe der Seiten-ID, die in der Datenbank des Servers hinterlegt ist, werden die Regionen der Seite auf das Bild, welches sich an dem gesendeten Pfad befindet, angewendet. Für mehr Details siehe im Praktikumsbericht von Tobias Kallauke.

Zu 4.: Texterkennungsergebnisse empfangen und darstellen. Nach der Texterkennung sendet der Server die Ergebnisse der Seiten als Antwort zurück. Wie bei Vision werden diese im selben State des State-Containers gesichert. Die Ergebnisse bestehen aus der Region-ID zur Zuordnung, dem erkannten Text sowie der confidence des OCR zum jeweiligen Text. Ähnlich, wie bei Vision.

7 Fazit

Dieses Kapitel reflektiert den aktuellen Stand der App und zieht einen Vergleich mit den Anforderungen aus Kapitel 4. Im Anhang Anhang C sind dazu einige weitere Screenshots abgebildet.

7.1 Stand des Prototyps

Während der Entwicklung konnten die meisten angedachten Funktionen des Anschauungsprototyps aus Kapitel 4 umgesetzt werden. Jedoch fehlt zum Zeitpunkt der Abgabe ein essentieller Bestandteil der App. Auf Grund mangelnder Zeit war es nicht mehr möglich, die Funktion zum Speichern der digitalisierten Daten in Front- und Backend zu implementieren. Auch fehlt die Überführung der Daten in ein geeignetes Format.

Außerdem sind nicht alle verfügbaren Serverschnittstellen implementiert. Die Views zum Ändern einer Scan-Vorlage sind schon integriert, da die Views zum Erstellen einer Scan-Vorlage hier wieder verwendet werden konnten. Es fehlen lediglich die interne Logik sowie Schnittstellen zum Server.

Weiterhin konnte nur einer der Kontrollmechanismen umgesetzt werden, da zu wenig Zeit in die Planung investiert wurde. Grundsätzlich ließen sich die fehlenden Mechanismen recht schnell implementieren, jedoch ist die aktuelle Datenstruktur nicht zum Speichern in der Datenbank geeignet. Sobald Änderungen an der Datenstruktur vorgenommen werden, kann es zu Fehlern kommen. Es existiert aber schon eine Idee zur Verbesserung. Dafür müssten allerdings neben der App auch der Server und die Datenbank angepasst werden. Auch müsste in jeder Scan-Vorlage zur Berechnung der Note ein Notenspiegel bzw. eine Berechnungsformel hinterlegt werden.

Des Weiteren wurden Fehlermeldungen und/oder -behandlungen bezüglich des Servers nur sporadisch implementiert. Aber auch hierfür sind die Grundlagen schon gelegt, da Fehler stets aufgefangen und zentral abgelegt werden.

7.2 Probleme und Grenzen der App

In diesem Abschnitt werden eventuell auftretende Probleme benannt und Lösungsvorschläge gegeben.

7.2.1 Probleme beim Erkennen von Dokumenten

Beim Fotografieren der Klausuren können infolge mangelnder Belichtung, schlechtem Hintergrund oder durch umgeknickte bzw. verdeckte Ecken, die Seitenränder und/oder die Ecken nicht ausreichend erkannt werden. Die Segmentier-Algorithmen stoßen bei der Bildverarbeitung hier an ihre Grenzen. Auf die Erklärung der genauen Hintergründe wird in diesem Bericht verzichtet. Weiterhin kommt es bei schnellen Bewegungen der Kamera oder zu flachem Winkel zum Dokument dazu, dass die App einen falschen Bildausschnitt wählt.

- Die erste Problemlösung besteht darin, einen Blitz zu verwenden. Meistens schaltet sich bei zu wenig Licht die zusätzliche Belichtung automatisch ein. Grundsätzlich empfiehlt es sich aber, immer einen Blitz zu verwenden, da dadurch das Dokument gleichmäßig belichtet wird. Bei einer zu starken Grundbelichtung verschlechtert ein Blitz jedoch das Ergebnis, da Text und Kanten unkenntlich gemacht werden. Um eine gute Qualität zu erzielen, ist es am günstigsten die Dokumente bei Tageslicht und mit Blitz aufzunehmen.
- Weiter ist es möglich die Dokumentenseite noch einmal zu fotografieren oder diese im Bild manuell auszurichten. Dafür stellt VisionKit eine besondere View zur Verfügung, die es dem Benutzer ermöglicht, die Ecken des Dokumentes per Hand auszuwählen.

7.2.2 Probleme der Klausur-Vorlage beim Scannen

Dieser Abschnitt bezieht sich ausschließlich auf bestehende Probleme mit den Klausurseiten, welche in Abbildung 3.1 zu sehen sind und während des Praktikums genutzt wurden. Sie stehen stellvertretend für Klausuren-Vorlagen der Fakultät CB. Im Abschnitt 7.3 werden mögliche Änderungen und Lösungen zu den hier aufgeführten Problemen diskutiert.

Wie in der Abbildung 3.1a zu sehen ist, wird nicht klar zwischen einem Feld für den Vornamen und Nachnamen differenziert. Zudem ist ein einzelner Strich als Feldbegrenzung ungeeignet. Die Punktefelder in der Tabelle dagegen sind besser abgegrenzt, jedoch zu klein. Die geringe Größe verleitet dazu, das Feld komplett auszunutzen und die Ziffer möglichst groß reinzuschreiben. Allerdings sollte eine gewisser Abstand zur Region begrenzung für optimale Texterkennung gelassen werden. Bei dem Notenfeld dagegen existiert keine Begrenzung. Für das Markieren der Region ist das ebenfalls ungeeignet (C.3b). Denn hinter der Note wird die Unterschrift des Prüfers gesetzt, welche bei der Digitalisierung nicht mit erscheinen darf.

7.2.3 Weitere Probleme der App

Obwohl SwiftUI die Entwicklung des Prototyps erst möglich gemacht hat, bringt das Framework von Juni 2019 einige Schwierigkeiten mit sich. Seit der Veröffentlichung existiert eine sehr kleine und unvollständige Dokumentation der SwiftUI-Schnittstellen. Auch ist der geringe Umfang von SwiftUI nicht mit dem von UIKit, welches seit Jahren in der iOS-Entwicklung als Standard benutzt wird, zu vergleichen. Das führt immer wieder zu einigen provisorischen Lösungen und macht das Auffinden von Fehlern besonders schwierig. Es kann daher bei der Benutzung der App zu unerwarteten Abstürzen oder Aufhängern kommen. Des Weiteren sind die verwendeten Texterkennung-Engines nur für gedruckte Schrift ausgelegt. Druckschrift wird daher häufiger erkannt als Schreibschrift. Auch bei bestimmten Ziffern, wie 1, 4 und 7 kommt es zu Fehlern bzw. Verwechslungen.

7.3 Verbesserung der Klausuren-Vorlage

Basierend auf der gesammelten Erfahrung mit der Anwendung beschäftigt sich dieser Abschnitt mit Verbesserungs- und Änderungsvorschlägen der Klausur in Abbildung 3.1. Sie steht, wie auch zu vor, stellvertretend für die Klausuren-Vorlagen der Fakultät CB. Alle folgenden Vorschläge haben den Hintergrund, die App bei der Digitalisierung zu unterstützen und beziehen sich auf die in Abschnitt 7.2 beschriebenen Probleme.

Der erste Vorschlag bezieht sich auf die große Bedeutung der Ecken und Kanten bei der Erkennung des Dokuments. Diese wichtigen Referenzpunkte werden aber unter bestimmten Bedingungen, wie im Unterabschnitt 7.2.1 erklärt, nicht immer erkannt. Deshalb empfiehlt es sich, eigene Referenzpunkte auf dem Dokument anzubringen. Beispielsweise durch einen Rahmen, wie es auf der Klausur (3.1a) zu sehen ist. oder durch ein QR-Code ähnliches Muster. Ein Vorteil bei eigenen Referenzpunkten ist, dass sich ihre Gestalt immer vom Dokument abheben kann. Somit ist die Dokumentenerkennung nur noch vom Licht abhängig. Dadurch kann die Kamera näher an den Text heran, wodurch sich die Bildqualität verbessert. Jedoch kann nun die Erkennung nicht mehr von Frameworks übernommen werden. Es müssten künstliche neuronale Netze trainiert werden, welche diese Referenzpunkte in Bildern erkennen.

Der zweite Vorschlag bezieht sich auf den Unterabschnitt 7.2.2. Dementsprechend wäre es sinnvoll alle Felder ausreichend zu beschriften und mit einer dünnen Linie zu begrenzen. Mithilfe von Bildverarbeitungsalgorithmen könnten die feinen Rahmen ohne Verlust der wichtigen Daten heraus gerechnet werden.

Abschließend ist zu erwähnen, dass laut der Problemstellung aus Kapitel 3 eine Klausuren-Vorlage entwickelt werden sollte. Auf Grund mangelnder Zeit war das jedoch nicht möglich.

8 Ausblick

In den folgenden Abschnitten werden einige Ideen erläutert, wie die App und das gesamte Software-System verbessert werden können.

Verbesserung der Benutzerfreundlichkeit und des Workflows Angesichts des Prototyp-Status der App sollten bei einer Weiterentwicklung zu allererst Systemkomponenten implementiert werden, die die Benutzung verbessern. Beispielsweise werden Login-Daten und schon heruntergeladene Scan-Vorlagen nicht dauerhaft gespeichert. Auch muss zu jeder Region ein Name vergeben werden. Bei genauerer Betrachtung gibt aber auch der Datentyp (6.5b) der Region in den meisten Fällen Auskunft darüber, wie die Region benannt werden sollte (siehe dazu Abbildung 6.5a). Handelt es sich um den Vor- bzw. Nachnamen, die Matrikelnummer, Seminargruppe oder Note, taucht diese Region auch nur genau einmal in einer Scan-Vorlage auf. Ausschließlich der Datentyp Punkte soll mehrmals vergeben werden können. Daher ist es sinnvoll den Workflow beim Erstellen von Regionen entsprechend anzupassen.

Weitere Dokument- und Datentypen Auf dem Abschnitt Verbesserung der Benutzerfreundlichkeit und des Workflows aufbauend, ist folgende Idee entstanden. Durch die generischen Vorlagen ist es möglich, neben Klausuren auch andere Dokumente nach diesem Schema zu digitalisieren. Beispielsweise Krankenscheine, Urlaubsanträge oder Arbeitsverträge. Dafür könnten spezielle Workflows zum Erstellen der jeweiligen Vorlage und weitere Regionen-Datentypen implementiert werden, wie z. B. Datum, IBAN-Nummer, E-Mailadresse und Telefonnummer.

Erweiterung der Texterkennung Wie im Abschnitt 7.2 erwähnt unterstützen Vision und Tesseract nicht das Erkennen von Schreibschrift. Jedoch wäre es möglich eigene neuronale Netze zu trainieren und implementieren, die diese Aufgabe übernehmen. Beispielsweise existiert ein Datensatz an handgeschriebenen Ziffern²³, mit dem ein einfaches Netz für das Erkennen von Ziffern, trainiert werden kann. Außerdem bietet die Serverarchitektur die Möglichkeit, andere oder mehr OCR-Engines zu implementieren. Für mehr Details siehe im Praktikumsbericht von Tobias Kallauke.

Klausuren-Einsicht Die archivierten Klausurenbilder könnten über ein Web-Portal dazu genutzt werden, eine Online-Einsicht der Prüfungen zu ermöglichen. Jedoch bringt

²³ MNIST-Datensatz - <http://yann.lecun.com/exdb/mnist/>

solch eine Plattform auch ein Problem mit sich. Durch sie ist die Verbreitung von Klausuren noch einfacher als zuvor, wodurch die Professoren dazu angehalten werden, sich immer wieder neue Aufgaben auszudenken. Aber auch dafür könnte es in Zukunft eine Lösung geben.

Weitere Plattformen Neben der iOS-App versuchte Tobias Kallauke dieselbe für die Android-Plattform zu entwickeln. Jedoch existierten zum Zeitpunkt des Praktikums keine passenden Frameworks, die das Erkennen von Dokumenten in Echtzeit übernehmen. Deshalb entwickelte Tobias einen Prototyp, der sich ausschließlich mit diesem Problem beschäftigt. Jedoch könnte das Software-System auf noch mehr Plattformen Anwendung finden. Neben einem Programm für Linux, MacOS und Windows ist auch eine Web-Plattform erdenklich. Alle Plattformen könnten den selben Funktionsumfang besitzen. Lediglich das Erstellen bzw. Aufnehmen der Fotos erfolgt beispielsweise über Scan- oder Kopiergeräte.

Intelligente Referenzpunkte In Abschnitt 7.3 wurde erwähnt, dass eigene Referenzpunkte auf den Klausuren Vorteile mit sich bringen. Möglich wäre auch, dass die Referenzpunkte ähnlich wie QR-Codes aufgebaut sind. Diese enthalten die nötigen Informationen zu den Regionen auf der Seite und ermöglichen das Digitalisieren ohne die Auswahl der richtigen Vorlage. Daran angelehnt könnten ein L^AT_EX-Paket oder Word-PlugIn entwickelt werden, wodurch die QR-Codes automatisch für jede Seite und deren Regionen generiert werden.

Klausuren-Vorlage entwickeln Auf Abschnitt 7.3 und den intelligenten Referenzpunkten aufbauend könnten eigene Klausuren-Vorlagen umgesetzt und getestet werden. Eine Anpassung der App ist erforderlich, sodass die Referenzpunkte erkannt werden, wie im Abschnitt 7.3 beschrieben.

Benutzerakzeptanzstudie Aktuell ist unklar, ob die App tatsächlich Zeit einspart. Aus diesem Grund ist es sinnvoll, eine umfangreiche Benutzerakzeptanzstudie durchzuführen. Außerdem könnten so weitere Probleme entdeckt und Lösungen entwickelt werden, bevor die App benutzt werden kann.

Anhang A: Wasserfall-Modell

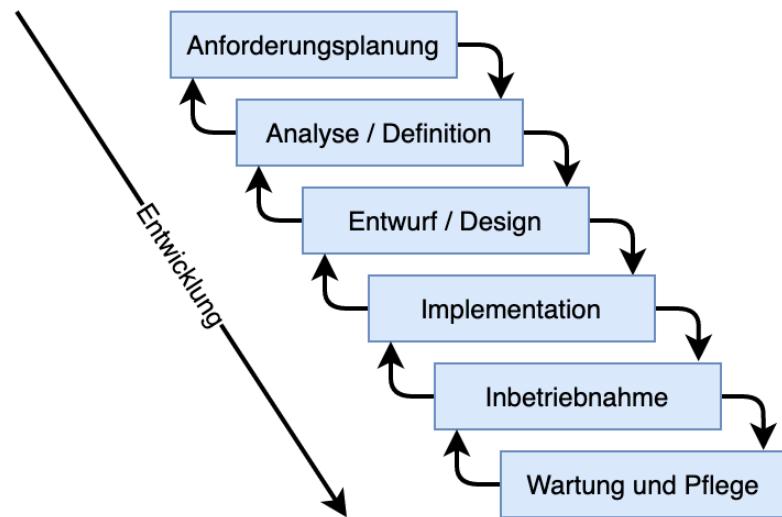


Abbildung A.1: Das Wasserfall-Modell nach Winston W. Royce

Anhang B: Workflow

Scan-Vorlage erstellen

1. "Neue Vorlage erstellen"
2. Foto machen
3. Frage: Ist Foto gut?
 - a) Ja: gehe zu 4.
 - b) Nein: gehe zu 2.
4. Neues Attribut hinzufügen
5. Bereich auf Bild auswählen
6. Frage: Ist Bereich gut?
 - a) Ja: gehe zu 7.
 - b) Nein: gehe zu 5.
7. Name für Attribut festlegen
8. Datentyp für Attribut festlegen (Name, Matrikelnummer, Note, ...)
9. Frage: Sind alle Attribute vorhanden?
 - a) Ja: gehe zu 10.
 - b) Nein: gehe zu 4.
10. Fertig
11. Vorlage an Server senden

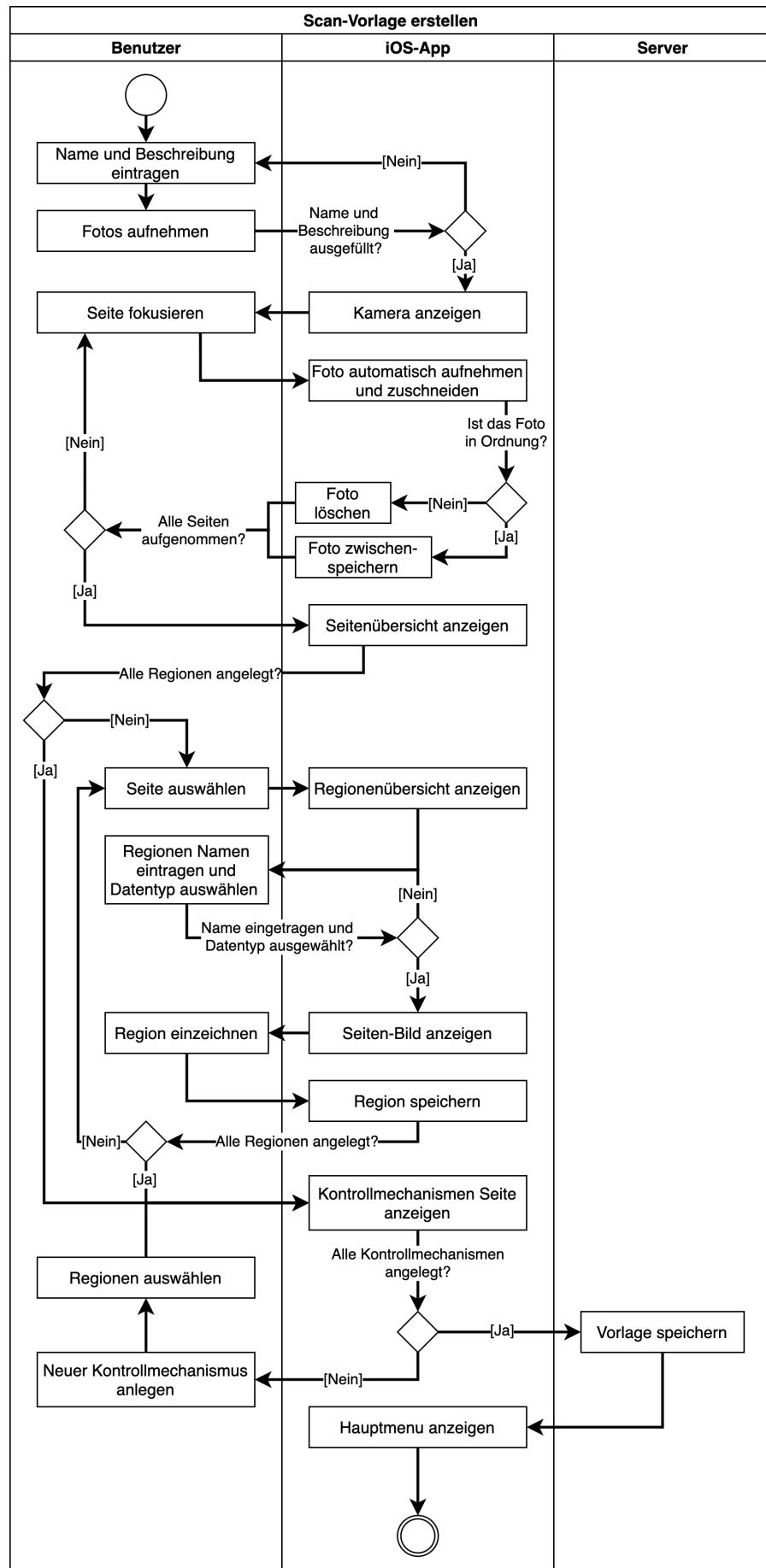


Abbildung B.1: Gekürztes Flussdiagramm zum Erstellen einer Scan-Vorlage

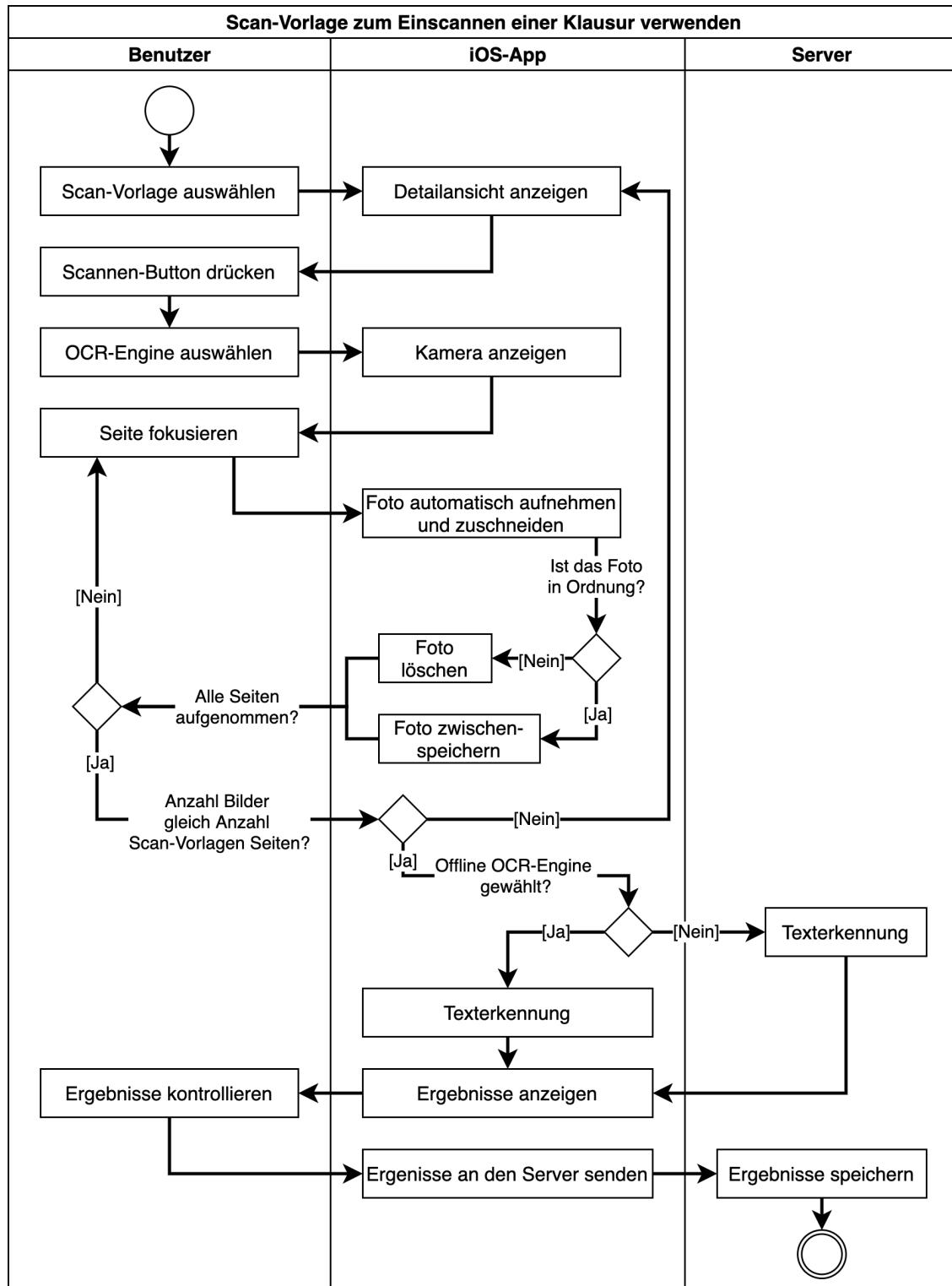
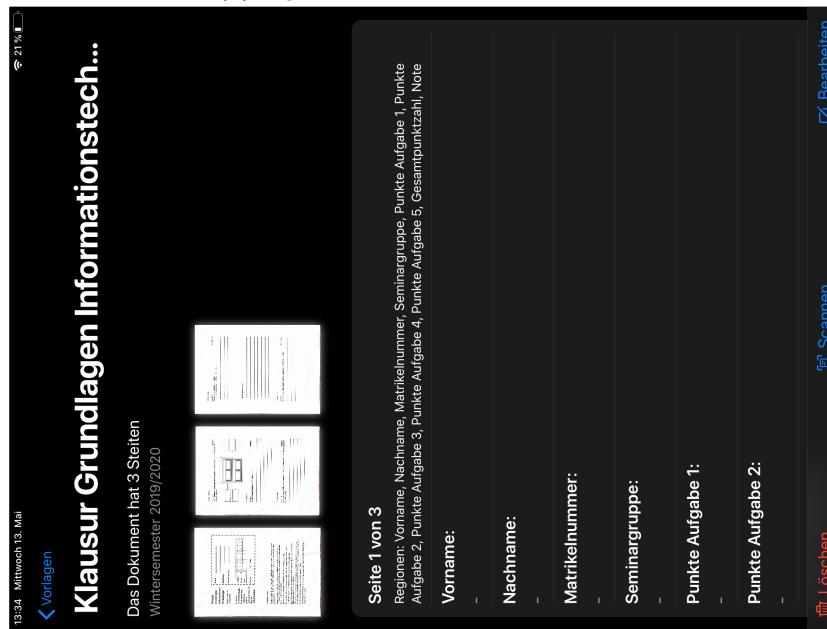


Abbildung B.2: Gekürztes Flussdiagramm zum Verwenden einer Scan-Vorlage

Anhang C: Weitere Funktionalität

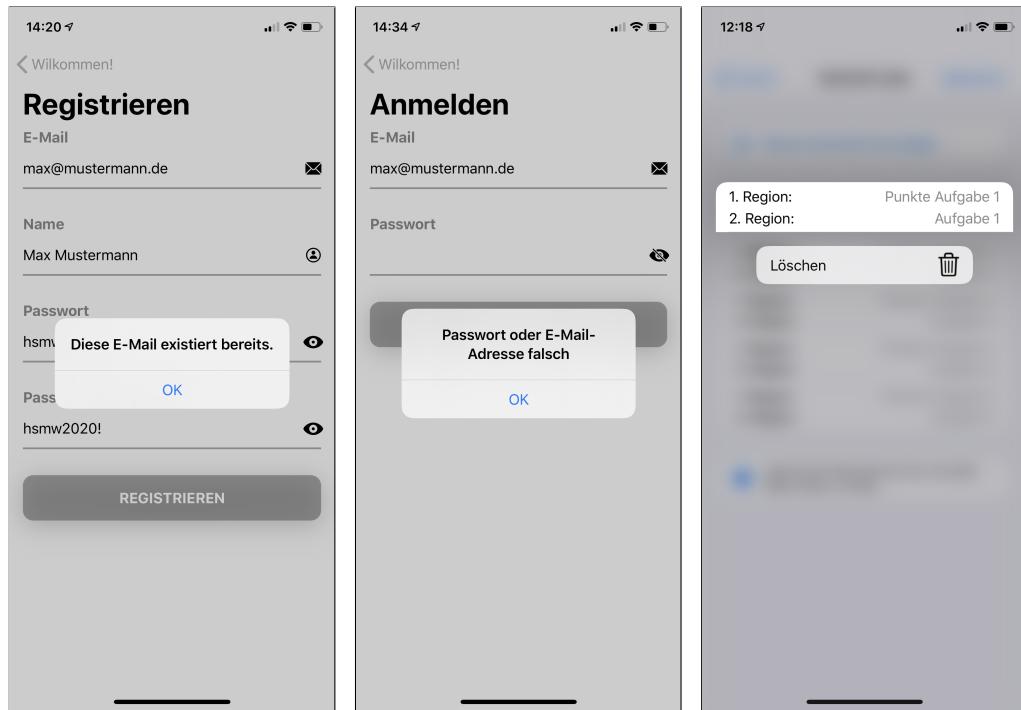


(a) Ergebnisse ansehen auf dem iPad



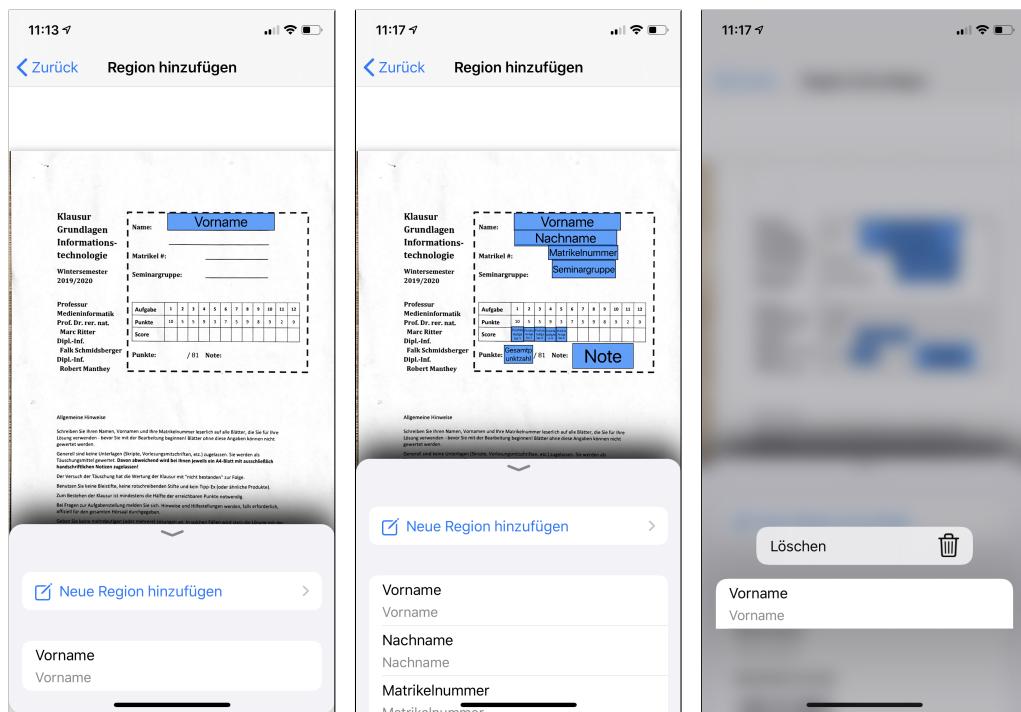
(b) Vorlagen-Detailansicht mit Dark-Mode und der größten einstellbaren Textgröße

Abbildung C.1: Vorlagen-Detailansicht auf einem iPad mit Barrierefreiheit Beispiel



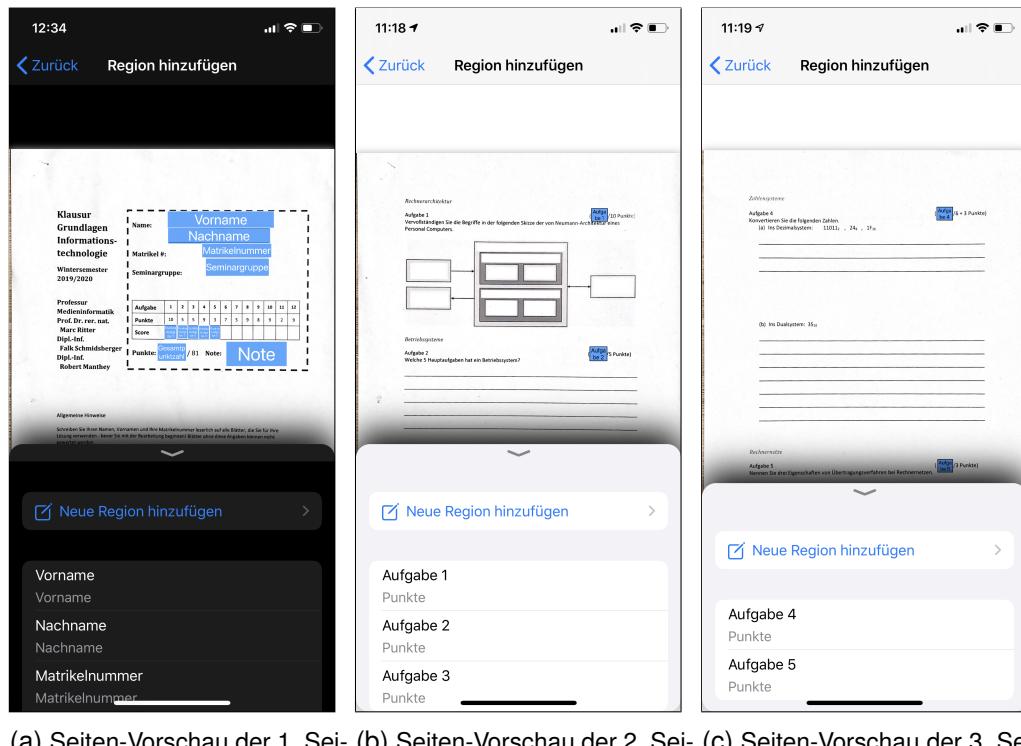
- (a) Eine der integrierten Fehlermeldungen beim Registrieren
 (b) Eine der integrierten Fehlermeldungen beim Anmelden
 (c) Löschen eines angelegten Kontrollmechanismuses

Abbildung C.2: Beispiel-Fehlermeldungen und View beim Löschen eines angelegten Kontrollmechanismuses



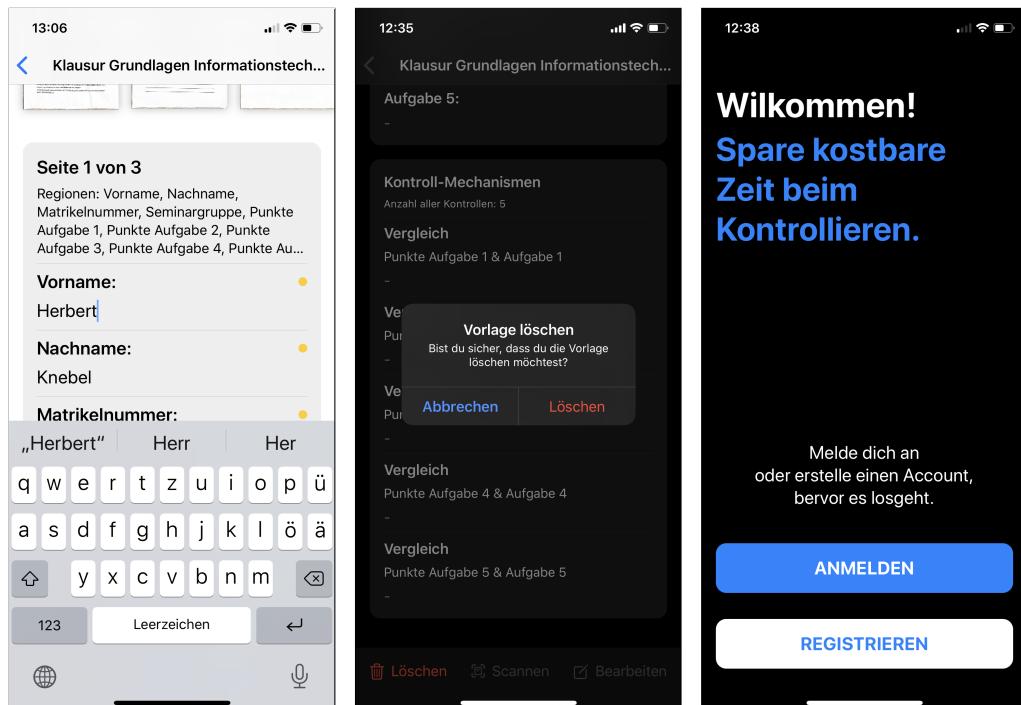
- (a) Seiten-Vorschau mit der ersten Region
 (b) Seiten-Vorschau mit allen Regionen
 (c) Löschen einer angelegten Region

Abbildung C.3: Seiten-Vorschau-View



(a) Seiten-Vorschau der 1. Seite mit allen Regionen im Dark-Mode
 (b) Seiten-Vorschau der 2. Seite mit allen Regionen
 (c) Seiten-Vorschau der 3. Seite mit allen Regionen

Abbildung C.4: Seiten-Vorschau im Light- und Dark-Mode



(a) Ergebnis bearbeiten mit der größten einstellbaren Textgröße
 (b) Löschen einer Vorlage im Dark-Mode
 (c) Willkommens-View mit der größten einstellbaren Textgröße

Abbildung C.5: Barrierefreiheit und Dark-Mode Beispiele

Anhang D: Installation der iOS App

Im folgenden Kapitel ist beschrieben, wie die App auf einem iPhone oder iPad zu installieren ist.

Voraussetzungen:

- macOS ab Version 10.15.4
- Xcode ab Version 11.4.1
- iPhone ab Version iOS 13.0 und/oder iPad ab Version iPadOS 13.4
- Internetzugang
- Zusatz: SwiftLint (<https://github.com/realm/SwiftLint>)

Installation: Stelle sicher, dass Xcode installiert und das iPhone/iPad mit dem Mac über ein USB-Kabel verbunden ist.

1. Lade das Projekt herunter und entpacke gegebenenfalls die .zip-Datei.
2. Öffne die Datei *DokumentenScanner.xcodeproj* mit Xcode.
3. Klicke in der Ordnerübersicht bzw. im Navigator ganz oben auf das Xcode-Projekt *DocumentenScanner*.
4. Wähle im Editor-Fenster unter *Targets* das Projekt *DocumentenScanner* aus. (Entweder links oben als Dropdown-Menü oder im geöffneten Menü links innerhalb des Editor-Fensters.)
5. Wähle den Reiter *Signing & Capabilities* aus. Es erscheint im Editor-Fenster ein Unterpunkt *Signing*.
6. Wähle ein passendes Team aus oder lege eins an.
7. Ändere den Bundle Identifier passend zum ausgewählten Team. Z. B. com.<Teamname>.DokumentenScanner. Es taucht ein Button auf, mit dem die App signiert werden muss. Falls kein Button erscheint, gehe zum nächsten Schritt und komme bei einer auftretenden Fehlermeldung bezüglich der Signierung hierher zurück. (Die Internetverbindung wird benötigt.)
8. Wähle nun einen Simulator oder das angeschlossene Gerät in der Toolbar aus den *Active Devices* aus.
9. Drücke anschließend den Start bzw. Run Button der *Build Controls* oder die Tastenkombination **cmd + R**.
10. Folge den Anweisungen von Xcode und auf dem iPhone oder iPad. Beim Verwenden eines Simulators tauchen keine weiteren Anweisungen auf. Die App startet automatisch.

Anhang E: Tätigkeitsbericht

24.02. - 01.03. Zunächst habe ich mich mit der Problemstellung auseinander gesetzt, Ideen gesammelt, Problemanalyse betrieben und einen kleinen Prototypen entwickelt. Dazu erstellte ich eine minimale Projektplanung, arbeitete mich in die Frameworks Vision und VisionKit ein und setzte eine Versionsverwaltung auf. Zusätzlich suchte ich nach einer passenden App-Architektur, die für das deklarative GUI-Framework SwiftUI sowie für asynchrone Aufgaben, wie z. B. API-Aufrufe geeignet ist. Dabei stieß ich auf Cleancode Architecture und Redux.

02.03. - 08.03. In dieser Woche habe ich die Texterkennung auf den berechneten Regionen eines neuen Fotos implementiert, den Workflow sowie viele andere Kleinigkeiten in der App verbessert und alle Fehler der letzten Woche behoben, sodass ich neue Dinge implementieren konnte. Zudem probierte ich CI sowie Lint für das Projekt aus. Da CI für eine iOS-App mit GitHub Actions schwer aufzusetzen war und ab April kostenpflichtig wurde, verwarf ich meine Pläne. Des Weiteren pflegte ich das Projekt Management durch Issues und Project Boards in GitHub. Anschließend programmierte ich den App-Workflow so um, dass nun mehr als eine Seite aufgenommen und analysiert werden konnte.

Abgesehen von neuen Quellcode begann ich mit dem Schreiben des Praktikumsberichts und arbeitete mich dazu in \LaTeX und die Bachelorarbeit-Vorlage für \LaTeX der Hochschule Mittweida ein.

09.03. - 15.03. Zu Beginn der dritten Woche schaute ich mir Möglichkeiten für serverseitiges OCR an. Dabei sammelte ich Informationen zu dem Framework Vapor und Swift unter Linux. Da die Frameworks Vision und CoreML von Apple unter Linux nicht funktionierten, stellte sich IronOCR als beste Option herausstellte. Mithilfe der in der App verwendeten Datentypen entwickelte ich ein Datenbankmodell und erstellte dazu noch eine JSON-Struktur, die später für die APIs verwendet werden könnte. Außerdem gab es ein Meeting, in dem Tobias Kallauke und ich unseren aktuellen Stand präsentierten, um weitere Schritte und Aufgaben zu planen. Bis zum Ende der Woche arbeitete ich fortlaufend an meinem Beleg und schrieb den Datenfluss in der App um. Nun ähnelte er sehr dem Redux-Model.

16.03. - 22.03. Anfangs schrieb ich meinen Praktikumsbericht weiter, bearbeitete alte Issues und fügte neue dem Project Board hinzu. Außerdem gepflegte ich die Dokumentation, um anschließend Kontrollmechanismen hinzuzufügen. Dabei entstanden neue Views. Der Redux-Store musste dadurch angepasst werden. Es kam eine Erweiterung für die Texterkennung hinzu, so dass man durch die Auswahl eines Datentyps, das Resultat der Erkennung verbessern konnte. Des Weiteren habe ich bis zum Ende der Woche den Kontroll-Typ Vergleich vollständig implementiert und die App auf Fehler und

Abstürze kontrolliert sowie den Beleg um einige Kapitel erweitert.

23.03. - 29.03. Ich begann den Workflow und die Navigation in der App zu verbessern und vereinfachen. Dabei beseitigte ich einigen Quellcode des Prototyps, erweiterte die Dokumentation und behob einige Fehler. Anschließend überarbeitete ich einige Views, sodass sie übersichtlicher und einfacher zu benutzen sind. Nach dem iOS 13.4 Update Mitte der Woche funktionierte ein Teil der App nicht, da sich das Verhalten von Views geändert hatte. Ich behob die Fehler, testete ausgiebig die App und fügte iPad Unterstützung hinzu. Des Weiteren entstand eine neue verbesserte permanente Scan-Vorlage und ich schrieb einen großen Teil des Berichts.

30.03. - 05.04. Das Backend für die App war soweit, dass ich es aufsetzen und die API-Schnittstellen implementieren konnte. Dazu erstellte ich Views für Registrieren und Anmelden, die mithilfe von regulären Ausdrücken, die Eingaben überprüfen. Außerdem entwickelte ich für die anfallenden asynchronen Aufgaben einen Schicht im App-Store. Dabei las ich mich in das Framework Combine ein und überlegte mir einen geeigneten Aufbau. Da der Ansatz von Combine sehr neu für mich war, dauerte es zwei Tage, bis ein erster API-Service mit Fehler-Handling funktionierte. Zum Ende der Woche waren alle der Create-Schnittstellen implementiert, getestet und dokumentiert. Nebenbei erstellte ein paar Issues für das Backend und sprach mich mit Tobias über OCR auf dem Server ab.

06.04. - 12.04. Diese Woche startete mit dem Umschreiben der Kontroll-Mechanismen und deren Analyse. Anschließend integrierte ich die Neuerungen vom Backend und erstellte die API für den Upload von Bildern. Danach fügte ich die APIs zusammen, um Vorlagen vollständig auf dem Server zu speichern und ab zu rufen. Dazu schrieb ich eine eigene JSON-Decoder-Funktion, um die App internen Datentypen zu unterstützen. Zusätzlich wurde die App etwas benutzerfreundlicher und ein Problem mit dem Start der iPad Version wurde behoben. Nach einem Meeting folgten noch weitere Absprachen mit Tobias und ich arbeitete weiter an dem Bericht.

14.04. - 19.04. Ein kritisches Problem mit den Sessions der vorherigen Woche konnte in dieser endlich gelöst werden. Dazu konnte ich die Anwendung etwas optimieren und einen sehr großen Teil des Beleges fertig stellen. Dabei half auch die Beantwortung vieler Fragen während eines Meetings mit dem Betreuer. Allerdings entstanden Probleme mit der Datenbank, die die Funktionalität der App einschränkten.

20.04. - 26.04. Ziel dieser Woche war es, so viel wie möglich des Belegs zu schreiben. Dafür fertigte ich einige Schemata und Bilder an. Zusätzlich konnte ich einige Fehler der App analysieren und beheben, sodass die Performance verbessert und Server-Aufrufe eingespart wurden. Außerdem kam die Online-Texterkennung über den Server hinzu sowie die zweite Korrektur in der Texterkennung mit Vision. Jedoch konnte die Online-Texterkennung auf Grund von Server-Problemen nicht getestet werden. Auch in

dieser Woche konnten einige Fragen zum Aufbau des Berichts in einem Meeting geklärt werden.

27.04. - 03.05. Im Mittelpunkt der Woche stand der Bericht, an dem ich täglich arbeitete. Ich erstellte einige Bilder für ihn und den Vortrag. Auch gab es diese Woche wieder ein Meeting.

04.05. - 10.05. Auch diese Woche arbeitete ich hauptsächlich am Beleg und stellte ihn so weit fertig, dass nur noch wenige Dinge zu erledigen sind. Zudem gelang es Tobias die Fehler der online OCR-Engine Tesseract zu beseitigen. Dadurch konnte ich die letzten fehlenden Schritte für den Umgang mit den OCR-Schnittstellen implementieren und testen.

11.05. - 15.05. In der letzten Woche bereitete ich alle Dokumente und Ordner für die Abgabe vor. Dabei erhielt ich nützliches Feedback der Betreuer.

Anhang F: Tätigkeitstabelle

Datum	Stunden	Tätigkeit
24.02.20	6,5	Ideen gesammelt, Dokumentationen angeschaut, Workflows erstellt, Projektplanung betrieben
25.02.20	6	Ideen gesammelt, minimalen Prototyp mit Texterkennung erstellt
26.02.20	8,5	Ideen gesammelt, weiteren Prototyp in SwiftUI zum Einzeichnen und Verschieben von Regionen entwickelt
27.02.20	9	Projekt mit Git aufgesetzt, Workflow zum Erstellen einer Region implementiert
28.02.20	6,5	Mock-Scan-Vorlage zum Testen erstellt, Fehlersuche bei Regionen-Problemen
29.02.20	1,5	Ideen gesammelt
01.03.20		
02.03.20	5	Ideen gesammelt, Texterkennung welche Regionen verwendet implementiert, Workflow verbessert
03.03.20	6	Ideen gesammelt, Continues Integration ausprobiert, viele Fehler behoben
04.03.20	6,5	Regionen Einzeichnen fertiggestellt (Einzeichnen, Bewegen, Zoom), Seiten- und Regionenvorschau implementiert, JSON Datenstruktur für Server-Kommunikation entwickelt
05.03.20	8	Überarbeitung des Workflows für beliebig viele Seiten
06.03.20	4	Swiftlint aufgesetzt, Fehler behoben, GitHub Issues erstellt
07.03.20	5	Einarbeitung in LaTeX, Ideen für den Bericht gesammelt, Beleg auf Github und Overleaf überführt
08.03.20	1	Ideen gesammelt, Beleg bearbeitet (Stichpunkte)
09.03.20	2	Beleg bearbeitet (Stichpunkte)
10.03.20	1	Ideen gesammelt, JSON Datenstruktur für Server-Kommunikation entwickelt
11.03.20	2	Iron OCR und CoreML für serverseitige OCR-Engines angeschaut, Datenbankstruktur entwickelt
12.03.20	6	Datenbankstruktur bearbeitet, Beleg bearbeitet (Stichpunkte)
13.03.20	6,5	Beleg bearbeitet (Kap. 1, 2)
14.03.20	2,5	Beleg bearbeitet (Kap. 1, 2)
15.03.20	7	Server Side Swift für OCR angeschaut, Redux Container umgeschrieben
16.03.20	7,5	Ausführliche Dokumentation, Projektmanagement mit GitHub Issues, Beleg bearbeitet

Datum	Stunden	Tätigkeit
17.03.20	8	Vorbereitung für Kontrollmechanismen, Kontrollmechanismen Anlegen implementiert
18.03.20	5	Kontrollmechanismen Anlegen implementiert
19.03.20	7	Kontrollmechanismen Anlegen implementiert, App ausführlich getestet
20.03.20	7	Fehler mit den Kontrollmechanismen behoben, weitere Issues erstellt, Beleg bearbeitet
21.03.20	2	Beleg bearbeitet (Kapitel 3), Ideen für Kontrollmechanismen gesammelt
22.03.20	5	Beleg bearbeitet (Kap. 3, 5, 6, Tätigkeitsbericht)
23.03.20	9	Dokumentation, Fehlerbehebung, Verbesserung der Navigation und einiger Views
24.03.20	8,5	Probleme behoben, Beleg bearbeitet (Kap. 4, 5, 6)
25.03.20	10	Probleme mit neuer iOS-Version gesucht und behoben, neue Mock-Scan-Vorlage
26.03.20	9	Mock-Scan-Vorlage fertiggestellt, Design der App verbessert, App für iPad angepasst, Beleg (Kap. 4)
27.03.20	6	Beleg bearbeitet (Kap. 4, 5), Fragen zum Meeting aufgeschrieben
28.03.20		
29.03.20		
30.03.20	9,5	Backend aufgesetzt, Anmelde-, Registrier- und Willkommensansicht entwickelt
31.03.20	9,5	Einarbeitung in Combine-Framework, Vorbereitung Authentifikation, App State angepasst, Network Service geschrieben
01.04.20	7,5	Authentifikation-State überarbeitet, Absprache mit Tobias über OCR, Login-API implementiert
02.04.20	6,5	Fertigstellung Authentifikation, Vorbereitung für Vorlagen-Network-Service, erste API fertig
03.04.20	7	Weitere Vorlagen-API fertiggestellt und getestet,
04.04.20	1,5	Dokumentation geschrieben, Ideen zum Ausführung der Kontrollmechanismen gesammelt
05.04.20		
06.04.20	8	Weitere Ideen zum Ausführen der Kontrollmechanismen gesammelt, weitere APIs implementiert, Beleg bearbeitet (Tätigkeitsbericht)
07.04.20	7	Upload APIs fertig, Ablauf zum gesamten Upload fertig
08.04.20	7,5	Eigenen speziellen JSON-Decoder für Download programmiert, Vorlagen-Download-API fertiggestellt, mit Pull-To-Refresh

Datum	Stunden	Tätigkeit
09.04.20	6	iPad-Problem behoben, Meeting, Absprache mit Tobias, Beleg bearbeitet (Kap. 6.3)
10.04.20	5,5	Beleg bearbeitet (Kap. 6.3), JWT-Session-Problem probiert zu lösen
11.04.20		
12.04.20		
13.04.20		
14.04.20	8	JWT-Session-Problem probiert zu lösen, Performance-Verbesserungen, Absprache mit Tobias, Beleg bearbeitet (Kap. 6)
15.04.20	8,5	Beleg bearbeitet (Kap. 6), eigene .bst-Datei erstellt
16.04.20	8	Beleg bearbeitet (Kap. 6.5), Commits für Backend erstellt, Lösung für JWT-Session gefunden
17.04.20	8	Beleg bearbeitet (Kap. 6.4, 6.5), viele Screenshots erstellt, Update für Backend aufgesetzt
18.04.20		
19.04.20		
20.04.20	8,5	Beleg bearbeitet (Kap. 5, 6.5.2), viele Screenshots erstellt und eingefügt
21.04.20	8	Beleg bearbeitet (Kap. 6.5.1, 6.5.3), Meeting
22.04.20	9	OCR-API angefangen zu implementieren, Workflow in View und Store angepasst
23.04.20	8	OCR-API-Grundlagen fertig implementiert, Beleg bearbeitet (Kap. 7.2)
24.04.20	6,5	offline OCR Verbesserung, Beleg (Kap. 1, 2)
25.04.20		
26.04.20		
27.04.20	8	Beleg bearbeitet (Kap. 7.2, 8)
28.04.20	7	Beleg bearbeitet (Kap. 8), Meeting
29.04.20	6,5	Beleg bearbeitet (Kap. 8), neue Bilder/Screenshots
30.04.20	7	Beleg bearbeitet (Kap. 4, 5)
01.05.20	4	Beleg bearbeitet (Kap. 4, 5)
02.05.20		
03.05.20		
04.05.20	8,5	Beleg bearbeitet (Kap. 4, 5)
05.05.20	7,5	Beleg bearbeitet (Kap. 7, 8, Tätigkeitsbericht), App verbessert, Absprache mit Tobias
06.05.20	10	Beleg korrigiert, viele kleine App-Probleme behoben, Absprache mit Tobias

Datum	Stunden	Tätigkeit
07.05.20	6,5	Beleg bearbeitet (Installation, Tätigkeitsbericht, Fazit) und korrigiert, Meeting
08.05.20	8,5	Beleg korrigiert, Flussdiagramme angefertigt
09.05.20	2,5	Beleg korrigiert
10.05.20	6	Beleg korrigiert und neue Bilder erstellt und eingefügt
11.05.20	8	Beleg korrigiert und neue Bilder erstellt und eingefügt
12.05.20	8,5	Beleg korrigiert, Layout verbessert
13.05.20	8,5	Beleg korrigiert und neue Bilder erstellt und eingefügt
14.05.20	6	Beleg korrigiert, Tätigkeitstabelle erstellt und in LaTeX eingefügt
15.05.20	5	Beleg korrigiert, Tätigkeitstabelle vervollständigt

Literaturverzeichnis

- [1] APPLE. Detecting Objects in Still Images | Apple Developer Documentation, Internetseite (2019). Verfügbar unter: https://developer.apple.com/documentation/vision/detecting_objects_in_still_images.
- [2] APPLE. Text Recognition in Vision Framework - WWDC 2019 - Videos, Internetseite (2019). Verfügbar unter: <https://developer.apple.com/videos/play/wwdc2019/234/>.
- [3] BRAGGE, M. *Model-View-Controller architectural pattern and its evolution in graphical user interface frameworks*. Bachelorarbeit (2013). Verfügbar unter: <https://lutpub.lut.fi/handle/10024/92156>.
- [4] FREEMAN, A. *Pro ASP.NET Core MVC 2*. Expert's Voice in .NET. Apress, Buch, 2. (30. Juni 2010) Edition (2017). ISBN 978-1-4842-3150-0. Verfügbar unter: <https://www.apress.com/gp/book/9781484231494>.
- [5] MITTWEIDA, H. Hochschule Mittweida: Portrait, Internetseite. Verfügbar unter: <https://www.hs-mittweida.de/hochschule/portrait.html>.
- [6] PAPA, J. Fundamental MVVM, Internetseite (2011). Verfügbar unter: <https://visualstudiomagazine.com/articles/2011/08/15/fundamental-mvvm.aspx>.
- [7] ROYCE, D. W. W. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS. Artikel, (1970), 11. Verfügbar unter: <http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf>.
- [8] SILLMANN, T. Einstieg in SwiftUI, Internetseite (2019). Verfügbar unter: <https://www.heise.de/developer/artikel/Einstieg-in-SwiftUI-4594018.html>.

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 15.05.2020