Technische Universität Braunschweig

ISF

Verifiable Paper Certificates

Android- und Serverapplikation

Software-Entwicklungspraktikum (SEP)
Sommersemester 2015

Technischer Entwurf

Auftraggeber
Technische Universität Braunschweig
Institut für Betriebssysteme und Rechnerverbund
Prof. Dr.-Ing. Lars Wolf
Mühlenpfordtstraße 23
38106 Braunschweig

Betreuer: Björn Gernert, Dominik Schürmann

Auftragnehmer:

Name	E-Mail-Adresse
Osama Alrjoob	y0067649@tu-bs.de
Tim Brandes	y0067685@tu-bs.de
Max-Frederik Eckardt	y0058622@tu-bs.de
Linda Fliss	y0067315@tu-bs.de
Thomas Haas	y0068180@tu-bs.de
Jan-Frederick Musiol	y0067262@tu-bs.de
Jan Schlichter	y0067112@tu-bs.de
Tim Schubert	y0067212@tu-bs.de

Braunschweig, 1. Juli 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

CertCheck ist eine Software, welche zur Validierung von Zeugnissen genutzt werden kann. Dabei besteht das Projekt aus zwei Teilen. Zum einen die Android-Anwendung, mit der der Benutzer Zeugnisse einlesen und überprüfen kann und zum anderen die Serveranwendung, mit der Zeugnisse angelegt, bearbeitet und ausgestellt werden können. Außerdem stellt die Serveranwendung eine Schnittstelle für die Zeugnisüberprüfung durch die Android-Anwendung zur Verfügung.

Dieses Dokument spezifiziert die technische Umsetzung von CertCheck und spezifiziert, wie die im Pflichtenheft beschriebenen Funktionen umgesetzt sind. Dabei werden in den folgenden Kapiteln alle für die Lauffähigkeit von CertCheck nötigen Details genauestens beschrieben. Dies beinhaltet alle implementierten Funktionalitäten, alle Komponenten und ihre Schnittstellen, die Beziehungen zwischen den Schnittstellen und die Verteilungen auf die Android-Anwendung und den Server.

Der Ablauf der Zeugnisüberprüfung und damit die grundsätzliche Benutzung durch den Anwender, der ein Zeugnis überprüfen möchte, wird durch das folgende Aktivitätsdiagramm beschrieben.

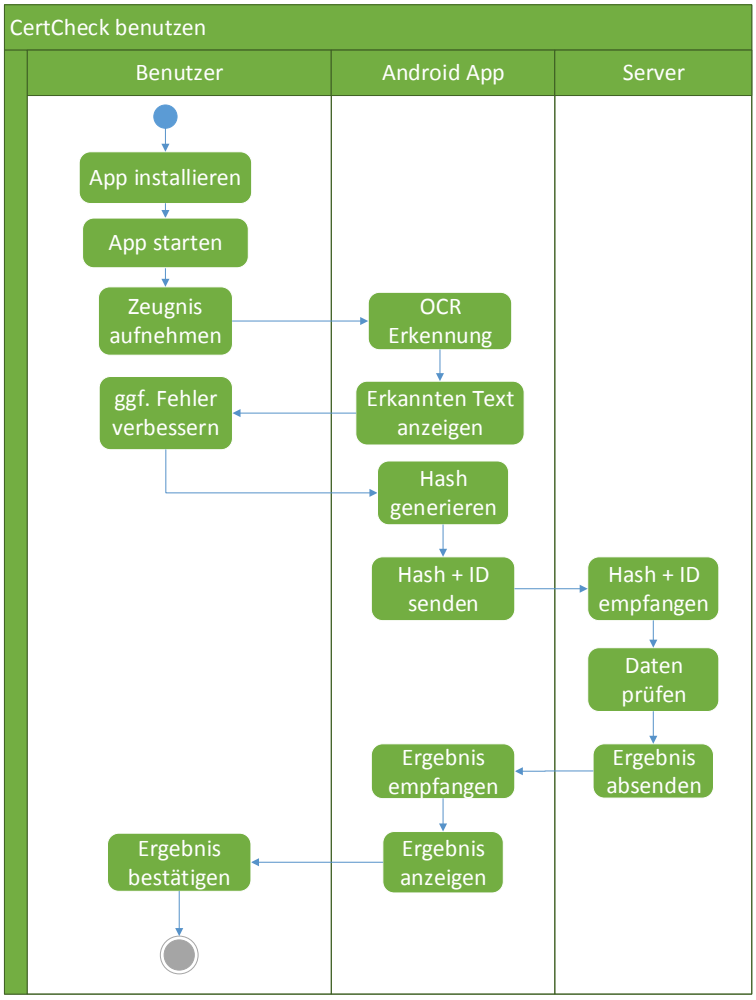

Abbildung 1.1: Aktivitätsdiagramm: Benutzung von CertCheck

Das Aktivitätsdiagramm aus Abbildung 1.1 stellt den durch CertCheck ermöglichten vereinfachten Validierungsvorgang dar. Der Benutzer muss die installierte App starten und kann dann durch die App eine OCR-Erkennung des Zeugnisses einleiten. Nach etwaiger Fehlerkorrektur, da Erkennungsungenauigkeiten zum Beispiel durch schlechte Lichtverhältnisse und ähnliches auftreten können, kann er die Überprüfung durch die App starten. Dabei sendet die App die Daten, die das Zeugnis eindeutig identifizieren an den Server, der durch die angebundene Datenbank die Echtheit eines Zeugnisses überprüfen kann. Zum Abschluss wird die Bestätigung oder Ablehnung des Zeugnisses zurück an die App gesendet die das Ergebnis dem Anwender anzeigt.

1.1 Projektdetails

Im folgenden Abschnitt werden die grundsätzlichen Funktionen der Android- und der Webanwendung genauer beschrieben. Dabei wird die Anwendung exemplarisch mit einem Aktivitätsdiagramm dargestellt.

1.1.1 Android-Anwendung

Hier wird die normale Verwendung der Android-Anwendung gezeigt.

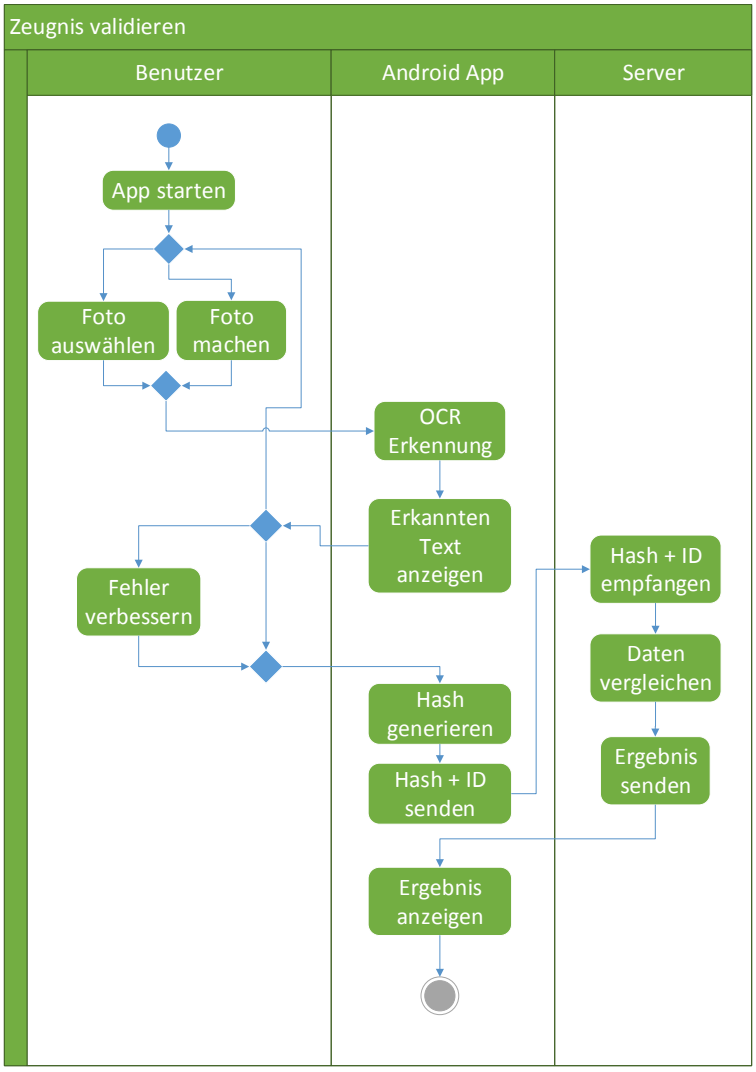

Abbildung 1.2: Aktivitätsdiagramm: Benutzung der Android-Anwendung

In Abbildung 1.2 wird ein klassischer Validierungsablauf durch einen Benutzer dargestellt.

Der Benutzer, der ein Zeugnis validieren möchte, startet die Anwendung auf seinem Android-Gerät und kann entweder ein gespeichertes Bild laden oder ein neues Bild eines Zeugnisses aufnehmen. Nach dem Einlesen der Aufnahme wird die Texterkennung (OCR) der App gestartet. Nach Abschluss der Erkennung wird das Ergebnis auf dem Display präsentiert. Hier kann der Benutzer möglicherweise falsch erkannte Zeichen verbessern und anschließend den Validierungsvorgang starten. Nun generiert die Android-Anwendung aus den ausgelesenen Daten einen Hashwert und sendet diesen mit der SecureID als Identifikationsmerkmale an den zentralen Server. Dieser gleicht die empfangenen Daten mit den Datensätzen in der Datenbank ab und meldet das Validierungsergebnis an die App zurück, die dann das positive oder negative Feedback anzeigt.

In der Abbildung 1.3 wird gezeigt wie der Benutzer der Android-Anwendung durch die verschiedenen Aktivitäten der Anwendung geführt wird.

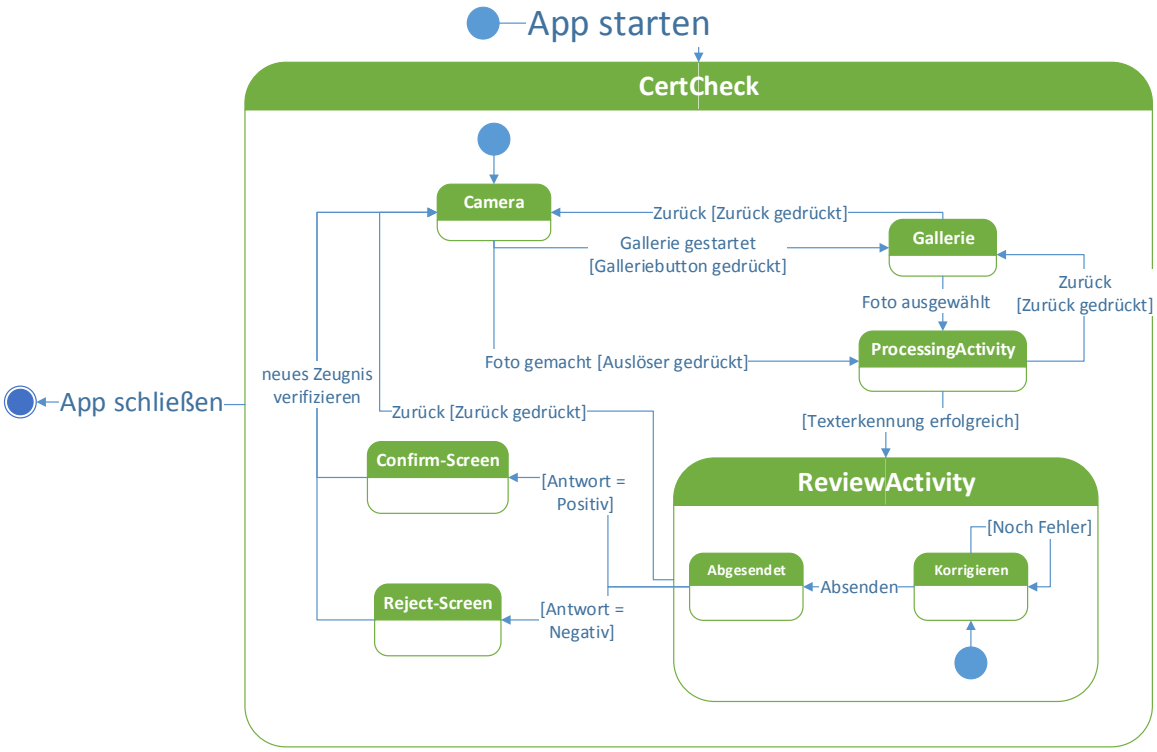

Abbildung 1.3: Aktivitätsdiagramm: Benutzerführung in der Android-Anwendung

1.1.2 Web-Anwendung

Hier wird beispielhaft die Zeugniserfassung über die Web-Anwendung gezeigt.

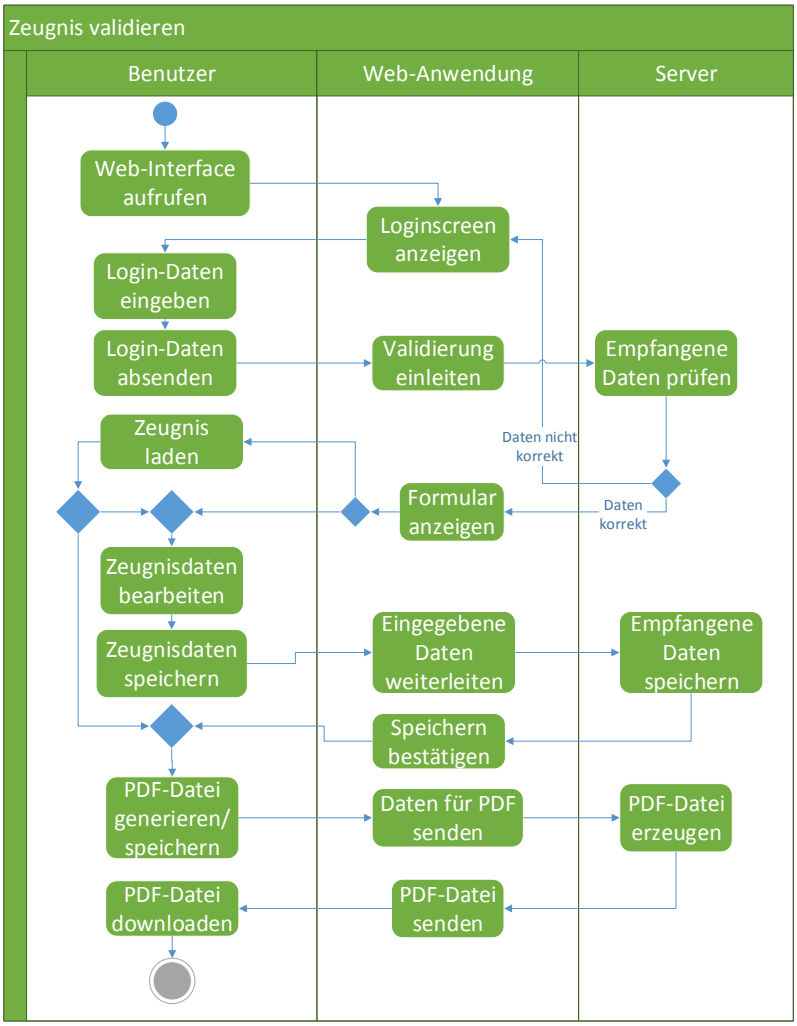

Abbildung 1.4: Aktivitätsdiagramm: Benutzung der Web-Anwendung

In Abbildung 1.4 wird gezeigt, wie neue Zeugnisdaten durch einen Benutzer erfasst werden können. Der Mitarbeiter eines Instituts, der ein Zeugnis erfassen möchte, ruft als erstes die Web-Oberfläche auf. Auf dem erscheinenden Login-Bildschirm authentifiziert er sich mit seinem Benutzernamen und seinem Passwort. Nach erfolgreicher Verifizierung durch die Anwendung wird das Eingabeformular aufgerufen. Hier sind alle Zeugnisfelder, die eine manuelle Eingabe erfordern, aufgeführt. Vom System vergebene Datenfelder werden in der Formularansicht nicht

gezeigt. Nach kompletter Eingabe der Zeugnisdaten kann aus dem Formular eine Zeugnis-PDF generiert werden, die dann vom Benutzer gespeichert werden kann.

2 Analyse der Produktfunktionen

In diesem Kapitel werden die einzelnen Produktfunktionen aus dem Pflichtenheft analysiert und das geplante Verhalten durch Sequenzdiagramme konkretisiert.

2.1 Android-Anwendung

Die im folgenden aufgelisteten Funktionalitäten <F10> bis <F50> beziehen sich hauptsächlich auf die Android-Anwendung.

2.1.1 Analyse von Funktionalität <F10>: Foto machen

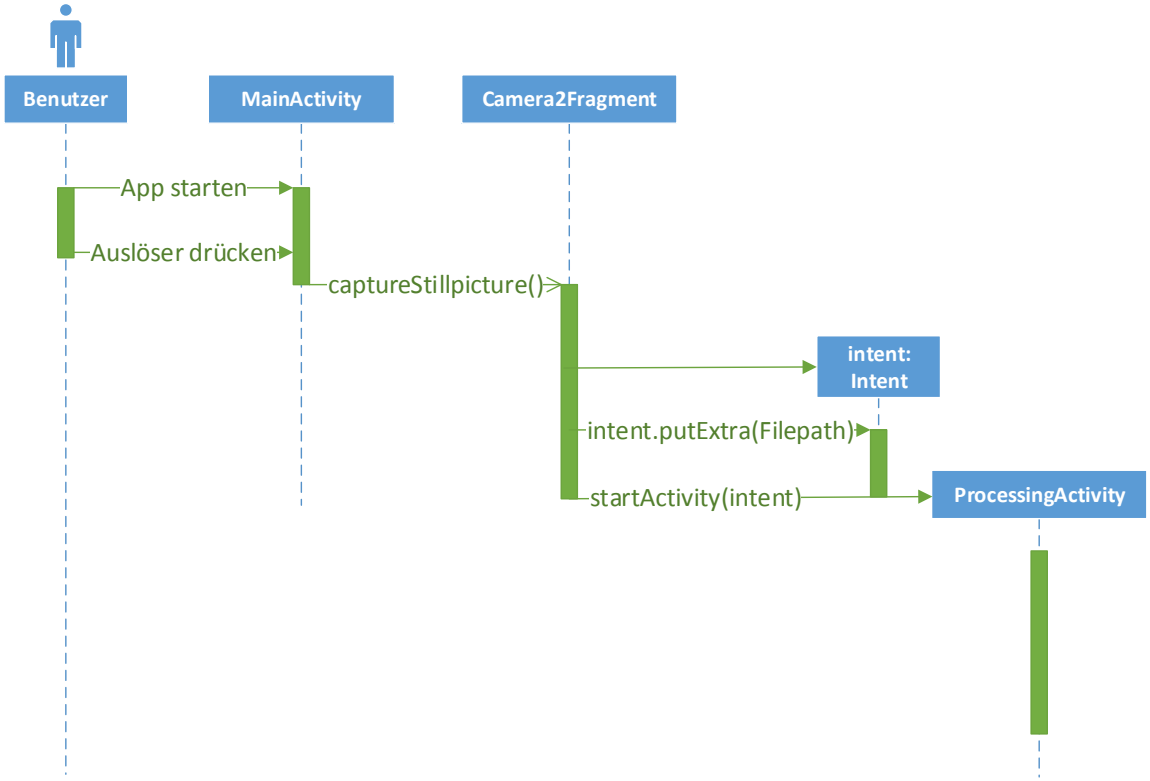

Abbildung 2.1: Sequenzdiagramm: Foto Aufnahme

In Abbildung 2.1 wird die Aufnahme eines Zeugnisausdruckes dargestellt. Beim Öffnen der Applikation gelangt der Benutzer sofort auf die Kameraoberfläche. Nach Drücken des Auslöseknopfes wird ein Bild aufgenommen und in der nächsten Aktivität angezeigt.

2.1.2 Analyse von Funktionalität <F20>: Foto aus der Galerie auswählen

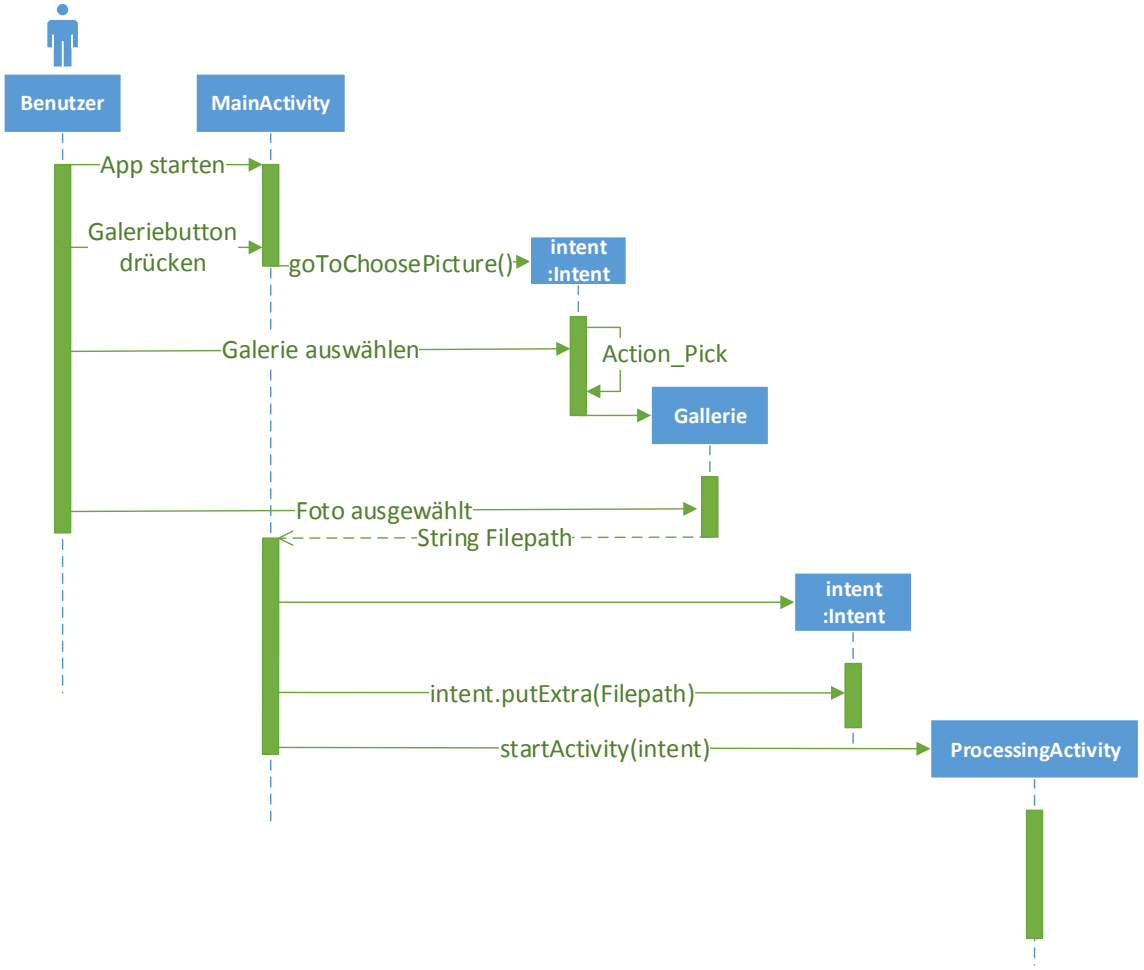

Abbildung 2.2: Sequenzdiagramm: Gallerieauswahl

Die Abbildung 2.2 zeigt, wie der Nutzer auch mittels einer auf seinem Gerät installierten Galerie-Applikation ein Foto auswählen kann. Hierzu wählt der Benutzter, nachdem er den Galerieknopf in der Action Bar gedrückt hat, eine der installierten Galerie-Applikationen aus. Nachdem ein Foto ausgewählt wurde, wird die nächste Aktivität gestartet.

2.1.3 Analyse von Funktionalität <F30>: Text erkennen

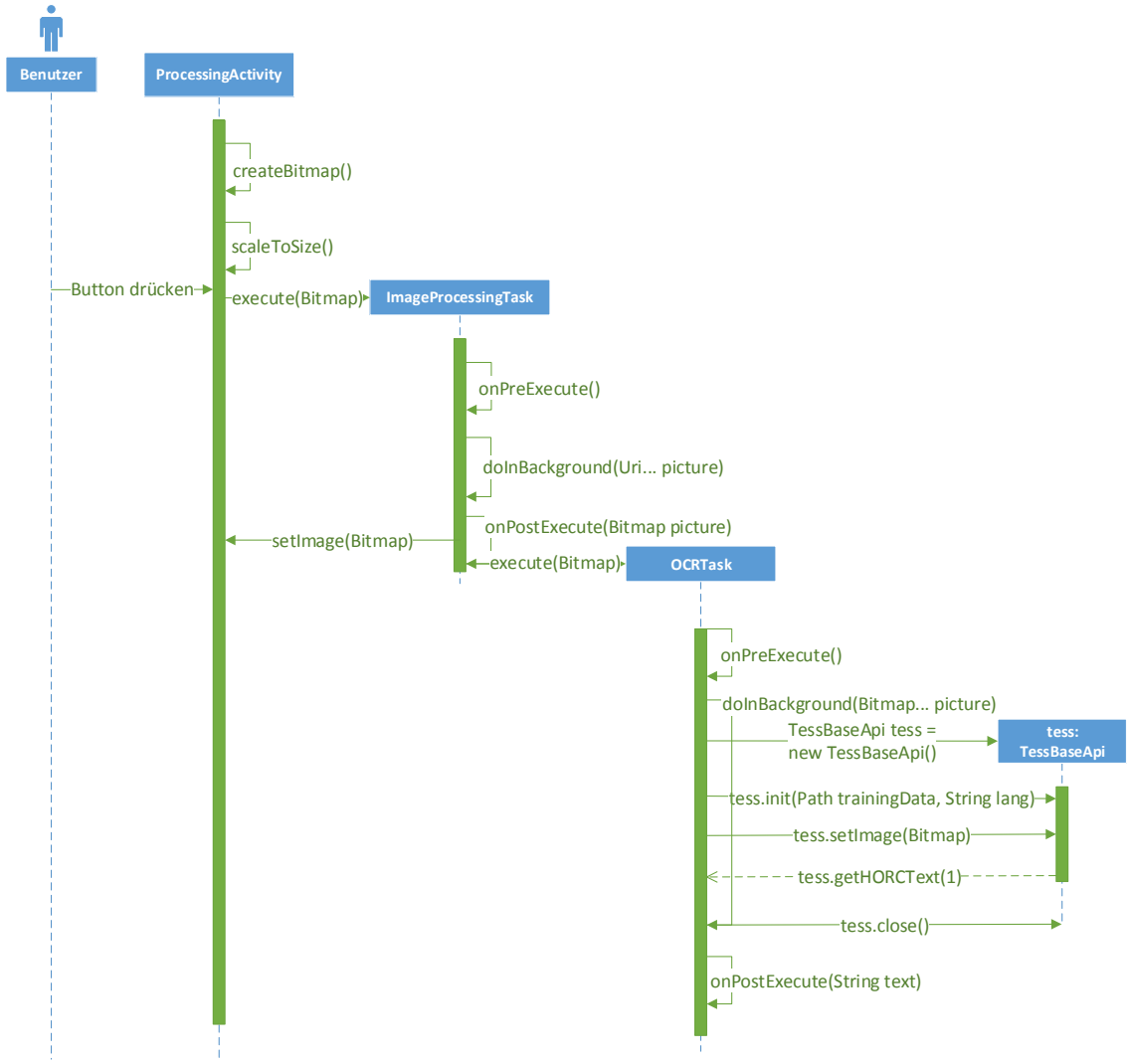

Abbildung 2.3: Sequenzdiagramm: Texterkennung

Die Texterkennung, wie in Abbildung 2.3 beschrieben, geschieht in einem separaten Thread. Zuerst wird in der ProcessingActivity eine Bitmap Datei aus dem zuvor ausgewählten oder gemachten Foto erstellt und anschließend zugeschnitten. Danach wird das Bitmap in einem separaten Thread für die Texterkennung vorbereitet. Hierfür wird die Klasse ImageProcessingTask aufgerufen, welche sich um das Multithreading der Bildbearbeitung kümmert. Wenn die Bildbearbeitung fertig ist, wird ein weiterer Thread mithilfe der Klasse OCRTask gestartet. Dieser kümmert sich um die Texterkennung, indem ein Objekt der TessBaseApi erschaffen wird und anschließend mit den notwendigen Daten über die .init() Methode versorgt wird. Zuletzt gibt man der TessBaseApi eine Bitmap Datei mit der Methode .setImage(Bitmap). Den erkannten Text bekommt man mit der Methode getHOCRText() zurück.

2.1.4 Analyse von Funktionalität <F40>: Text anzeigen

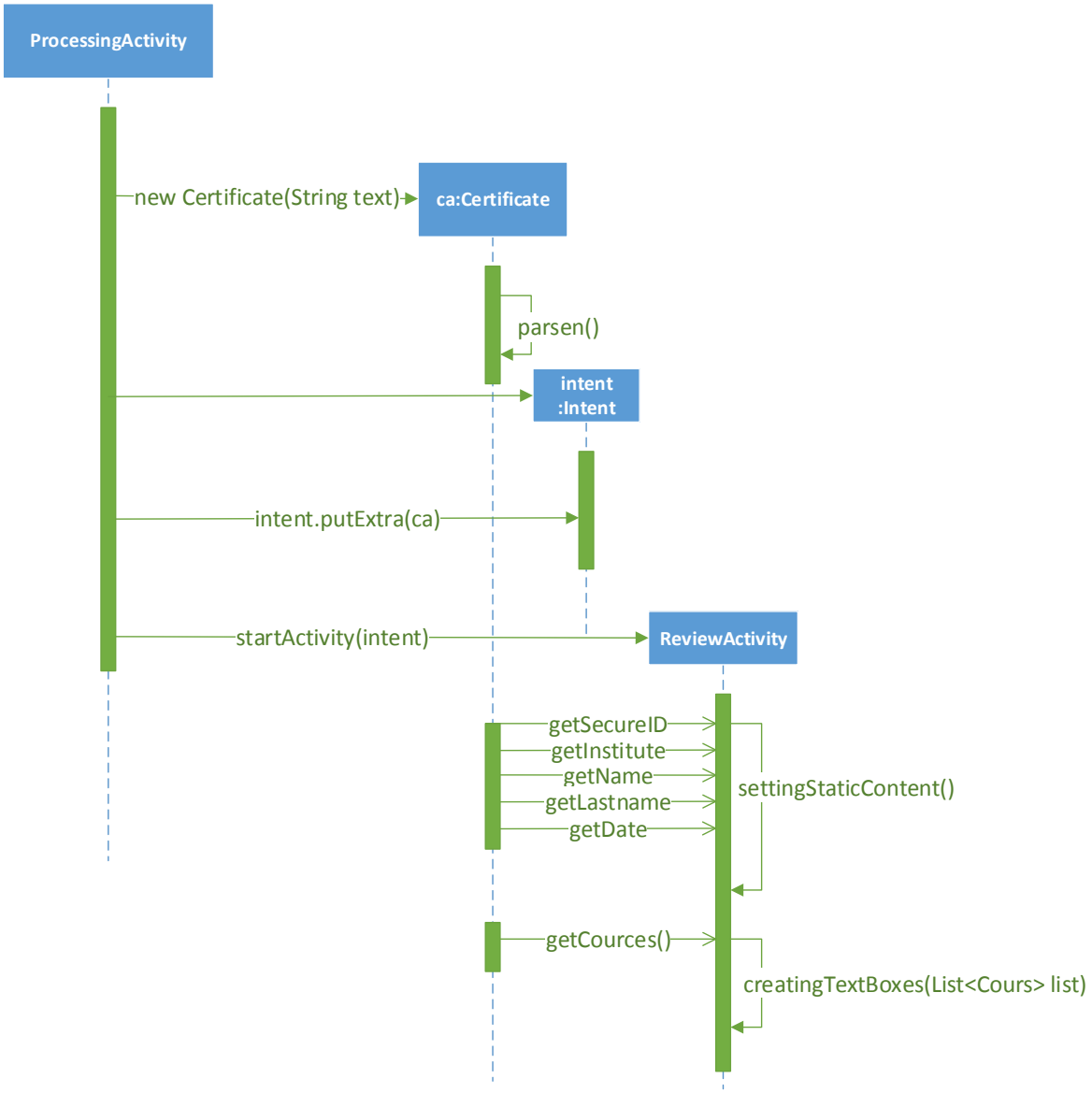

Abbildung 2.4: Sequenzdiagramm: Text anzeigen

Wie in Abbildung 2.4 dargestellt, liest die ReviewActivity die Informationen aus dem Certificate Objekt aus und zeigt sie in Textfeldern an. Da jedes Certificate unterschiedlich viele Kurse haben kann, zeigt die Methode creatingTextBoxes() genau so viel Textfelder an, wie es Kurse im Certificate gibt.

2.1.5 Analyse von Funktionalität <F50>: Hash generieren

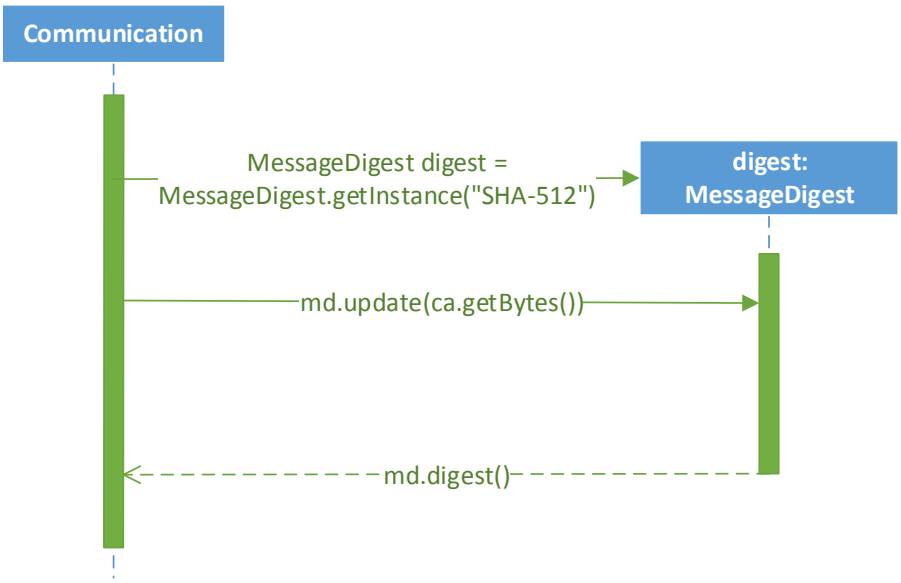

Abbildung 2.5: Sequenzdiagramm: Hash generieren

In Abbildung 2.5 wird gezeigt, wie der Hash aus den erkannten Daten unter Nutzung der Java-Klasse MessageDigest generiert wird. Zuerst instantiieren wir die Klasse und geben an, welchen Hash-Algorithmus wir nutzen wollen. Anschließend übergeben wir die Daten als Bytes. Zum Schluss erhalten wir den Hash mit der Methode .digest().

2.2 Web-Anwendung

Die im folgenden aufgelisteten Funktionalitäten <F60> bis <F110> beziehen sich hauptsächlich auf die Web-Anwendung. Aufgrund der funktionalen Zusammengehörigkeiten sind die Funktionen <F60> und <F70> und auch <F90> und <F100> in einem Sequenzdiagramm zusammengefasst.

Außerdem kommen zu den Funktionen aus dem Pflichtenheft noch im Pflichtenheft nicht aufgeführte Funktionen zu den Anforderungen <RM8> und <RS5> hinzu.

2.2.1 Analyse von Funktionalität <F60>: Daten zum Server senden und <F70>: Daten vom Server empfangen

Zwischen Android-Anwendung und Validierungsserver erfolgt ein Datenaustausch.

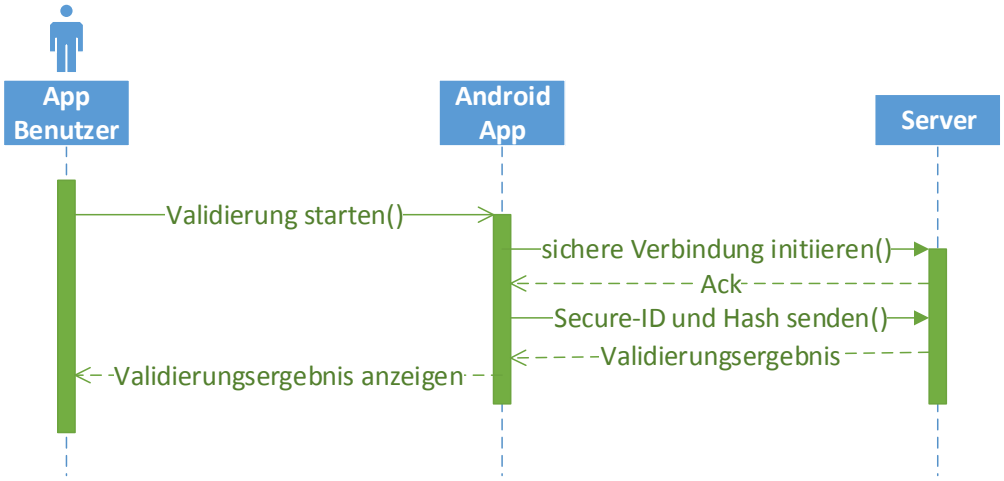

Abbildung 2.6: Sequenzdiagramm: Kommunikation App und Server

Der Benutzer der Android-Anwendung startet durch einen Button den Validierungsvorgang (siehe Abbildung 2.6). Dafür initiiert die App eine verschlüsselte Verbindung mit dem Validierungsserver. Nach einem erfolgreichen Verbindungsaufbau wird dem Server die SecureID des eingelesenen Zeugnisses und der dazugehörige, berechnete Hash über den aufgebauten Übertragungskanal gesendet. Der Server sendet dann das Ergebnis der Überprüfung an die App zurück. Hierbei erfolgt lediglich ein positives oder negatives Feedback, damit Zeugnisfälscher keinerlei Informationen erhalten, an welchen Stellen ein Problem mit den gesendeten Daten entstanden ist. Die App präsentiert dem Benutzer die erfolgreiche oder fehlgeschlagene Validierung auf dem Display.

2.2.2 Analyse von Funktionalität <F80>: Einloggen

Absicherung der Weboberfläche vor unbefugtem Zugriff durch einen separaten Login.

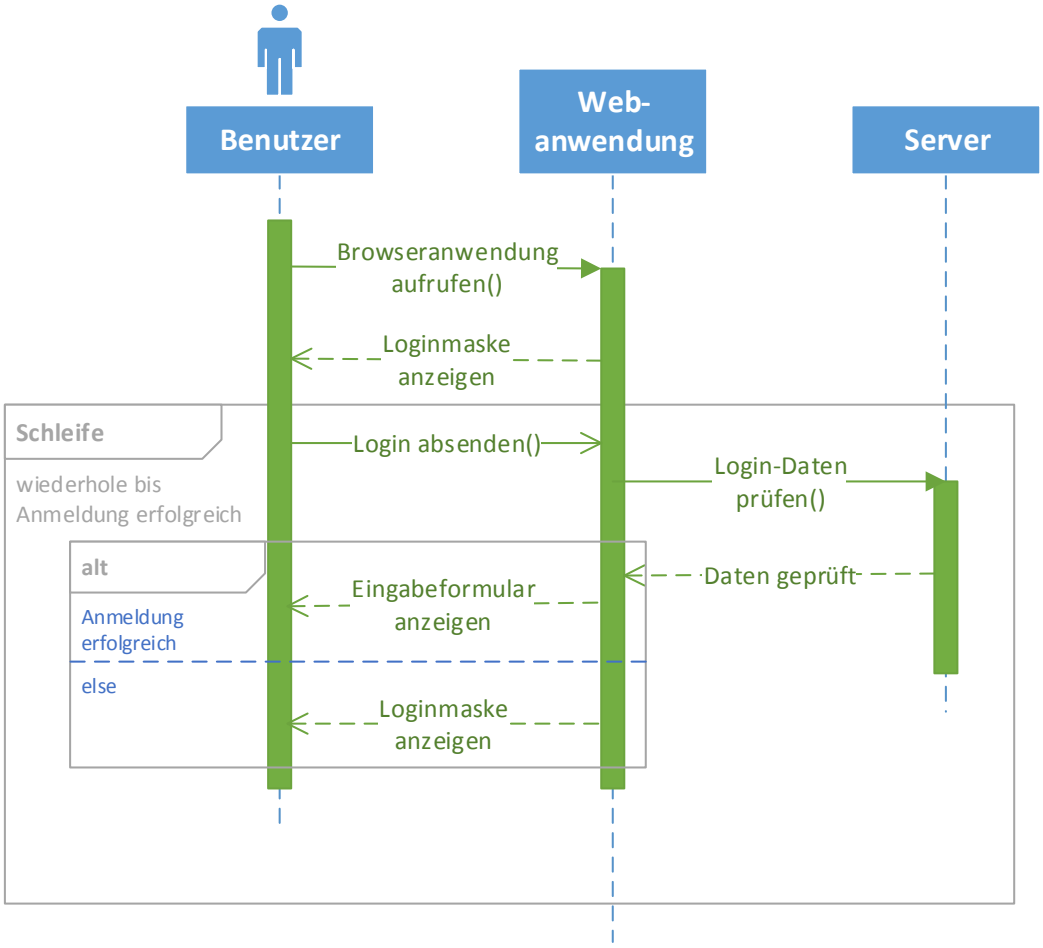

Abbildung 2.7: Sequenzdiagramm: Einloggen an Webinterface

In Abbildung 2.7 ist dargestellt, dass der Benutzer sich an der Weboberfläche für die Zeugnisdatenverwaltung anmelden möchte. Dazu ruft er die URL der Anwendung auf und bekommt eine Loginmaske eingeblendet. Nun muss der Benutzer seinen festgelegten Benutzernamen und sein persönliches Passwort eingeben und die Eingabe absenden.

Die Webanwendung sendet die eingegebenen Daten verschlüsselt an den zentralen Server, der die Gültigkeit der Benutzer-Passwortkombination in seiner Datenbank überprüft. Auch hier wird das Ergebnis der Überprüfung an die Webanwendung zurückgegeben. Waren die Eingaben eine gütige Kombination, wird dem Benutzer das Formular für die Zeugnisdatenverarbeitung angezeigt. War jedoch die Eingabe ungültig, wird dem Benutzer erneut die Loginmaske eingeblendet und der Benutzer kann den Anmeldeprozess von vorne beginnen.

2.2.3 Analyse von Funktionalität <F90>: Zeugnis erstellen und <F100>: PDF downloaden

Ausstellung von Zeugnissen durch berechtigte Mitarbeiter.

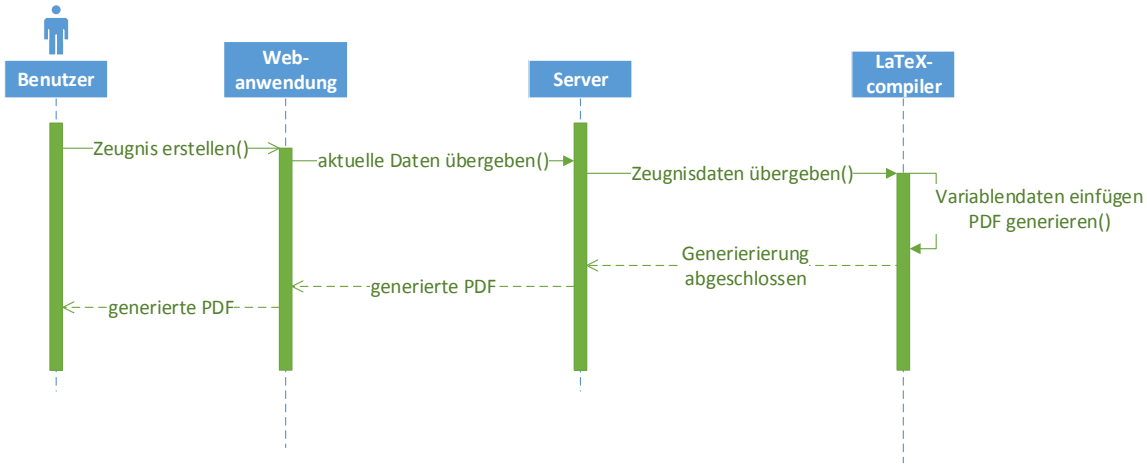

Abbildung 2.8: Sequenzdiagramm: Zeugnis erstellen und downloaden

Abbildung 2.8 zeigt wie der Benutzer ein Zeugnis erstellen kann. Dazu muss er die gewünschten Daten nach der Authentifizierung in das Formular eingeben. Nun kann mittels eines Buttons das Zeugnis generiert werden. Dafür werden alle eingegebenen Daten an den Server übermittelt. Nun generiert der Server mit dem installierten LaTeX-Compiler eine PDF-Datei. Wenn der Generierungsprozess abgeschlossen ist, startet der Download, sodass der Benutzer das Zeugnis ausdrucken kann.

2.2.4 Analyse von Funktionalität <F110>: Hash validieren

Überprüfung von Zeugnissen durch die Verwaltungsanwendung.

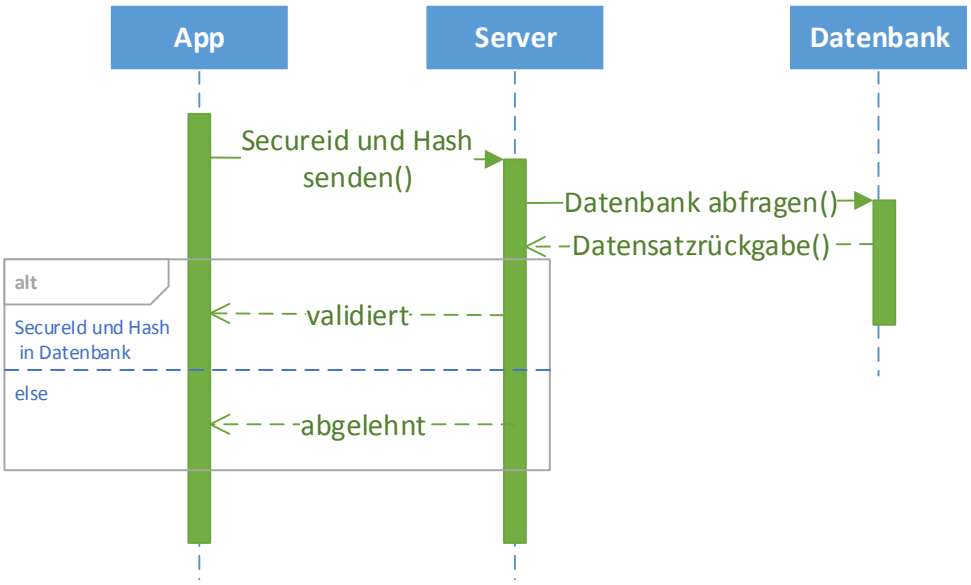

Abbildung 2.9: Sequenzdiagramm: Hash validieren

Um eine empfangene SecureID und Hashwertkombination zu validieren, startet der Server eine Datenbankabfrage und kontrolliert, ob die empfangenen Daten zu einem gültigen Eintrag in der entsprechenden Datenbanktabelle "certificates_certificate" passend sind. Dies ist in 2.9 zu sehen. Gibt es einen passenden und gültigen Datenbankeintrag, wird ein positives Validierungsergebnis an die App gesendet. Ergibt die Kombination keinen gültigen Eintrag oder treten anderweitige Unstimmigkeiten auf, so wird der App ein negatives Feedback gesendet. Auf eine genauere Fehlerbeschreibung muss an dieser Stelle verzichtet werden, da dies potentiellen Zeugnisfälschern eine genauere Fehleranalyse ermöglichen würde.

2.2.5 Analyse von Funktionalität <RM8>: Zeugnisse verwalten

Verwaltungsmöglichkeiten von vorhandenen Zeugnissen.

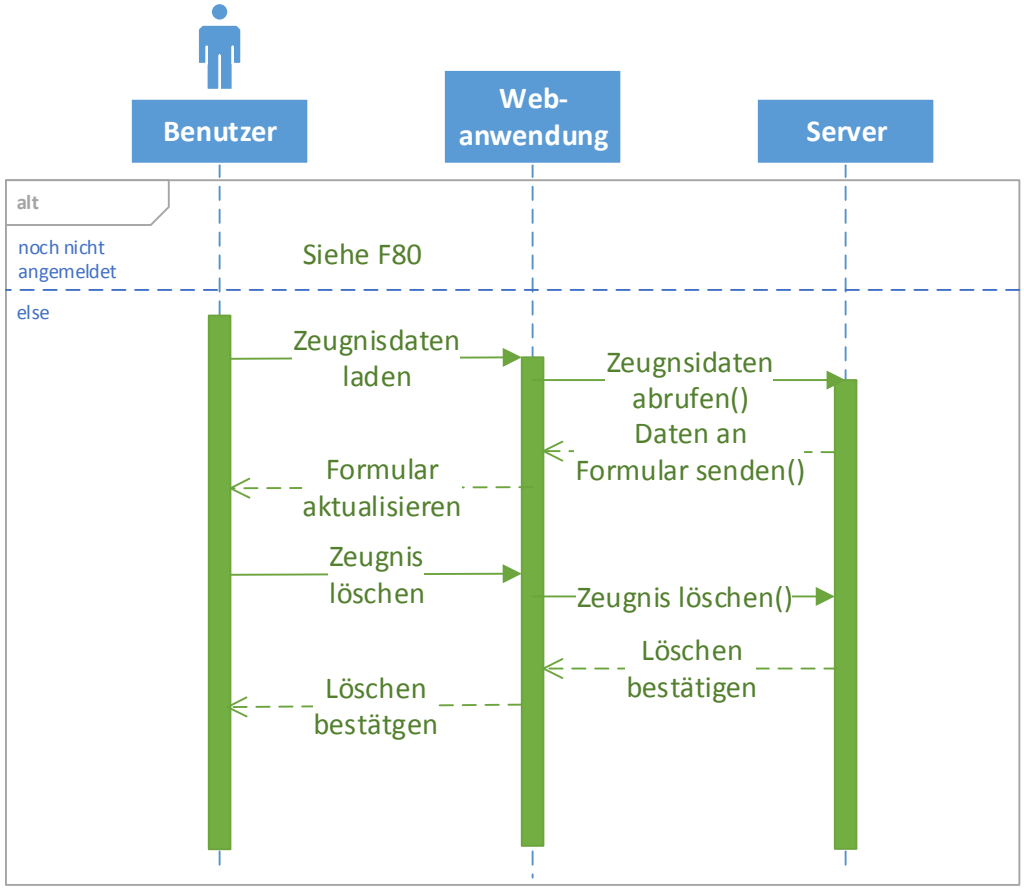

Abbildung 2.10: Sequenzdiagramm: Zeugnisse verwalten

Dem Benutzer wird es, wie in Abbildung 2.10 gezeigt, ermöglicht bereits vorhandene Zeugnisse zu verwalten. Der Benutzer kann nach erfolgreicher Authentifizierung, anstatt neue Zeugnisdaten einzugeben, auch ein bereits vorhandenes Zeugnis über den entsprechenden Button in der Formularanzeige laden. Dazu wählt er über die ihm bekannten Kriterien ein Zeugnis aus, das er laden möchte. Mit diesen Informationen lädt die Anwendung die entsprechenden Daten aus der Serverdatenbank und blendet diese in das Formular ein. Nach dem erfolgreichen Laden, kann der Benutzer das Zeugnis mittels Löschbutton entfernen. Dabei werden die gespeicherten Daten nicht komplett entfernt, sondern lediglich inaktiv gesetzt, sodass sie für keinen externen Zugriff auf die Datenbank mehr verfügbar sind. Nach dem Löschvorgang bekommt der Benutzer eine Bestätigung angezeigt.

2.2.6 Analyse von Funktionalität <RS5>: Zeugnis ändern

Ändern von bereits gespeicherten Daten in der Zeugnisdatenbank.

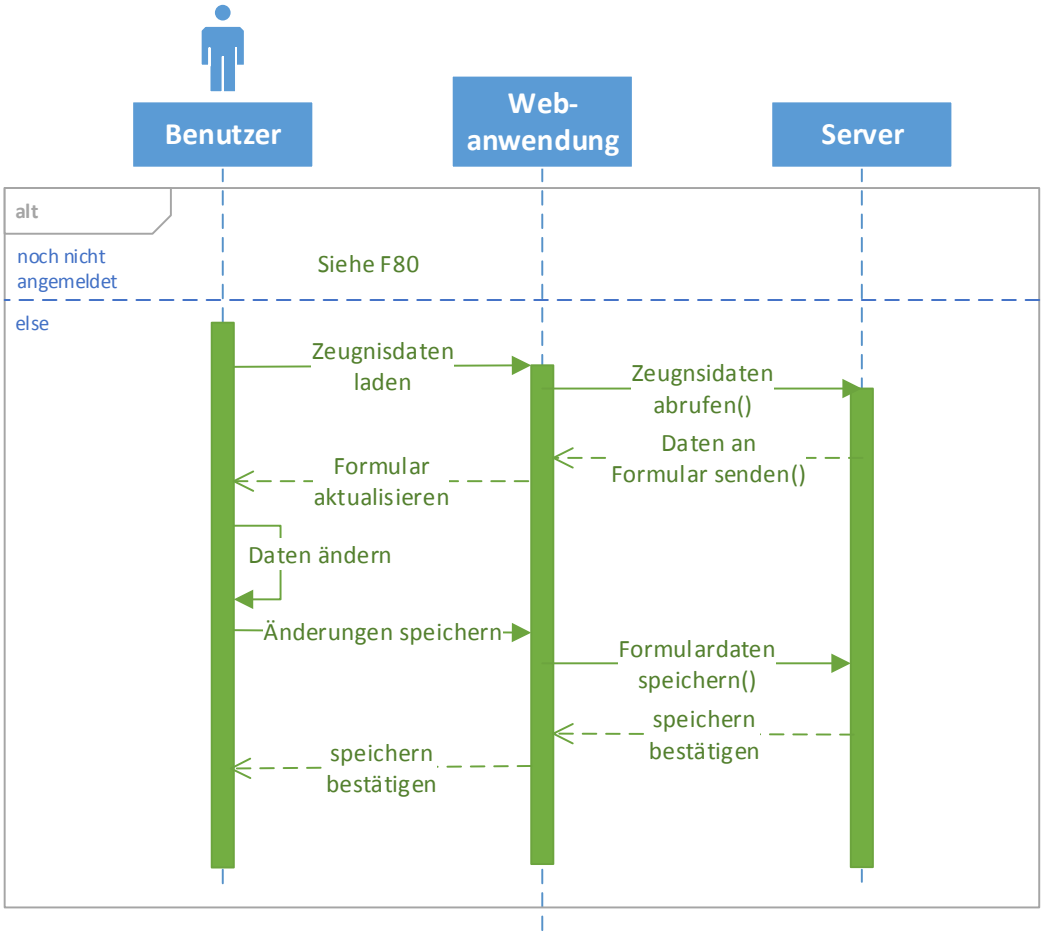

Abbildung 2.11: Sequenzdiagramm: Zeugnisdaten ändern

In Abbildung 2.11 wird ausgeführt, wie der Benutzer ein vorhandenes Zeugnis ändern kann, um es zum Beispiel etappenweise fertig zu stellen. Dazu geht der Benutzer wie beim Löschen eines Zeugnisses vor. Er wählt aus den vorhandenen Zeugnissen eines aus und lädt die dazu gespeicherten Daten in das Eingabeformular.

Nun kann er die Daten wie bei Ersterfassung ändern und ergänzen.

Anschließend müssen die Änderungen über den Speicherbutton bestätigt werden, sodass diese in die Datenbank übertragen werden.

Auch an dieser Stelle bekommt der Anwender Feedback von der Anwendung, ob das Speichern erfolgreich oder nicht erfolgreich war.

2.3 Nichtfunktionale Anforderungen

Die nicht funktionalen Anforderungen aus dem Pflichtenheft werden über die Art der Umsetzung gewährleistet. Dabei sind an dieser Stelle keine genauen technischen Details zu erwähnen, da während des gesamten Entwicklungsprozesses auf die Einhaltung der nicht funktionalen Anforderungen geachtet wird.

3 Resultierende Softwarearchitektur

In diesem Abschnitt wird ein Überblick über die zu entwickelnden Komponenten und Subsysteme geliefert.

3.1 Android-App

Im Folgenden wird die Softwarearchitektur der App erklärt.

3.1.1 Komponentenspezifikation

Aus den Komponenten der App ergibt sich das Diagramm in Abbildung 3.1. Im Folgenden werden die enthaltenen Komponenten im Detail beschrieben.

Komponente $\langle C10 \rangle$**: <Image>**
Lässt den Nutzer ein Bild von einem Zeugnis aufnehmen oder aus dem Telefonspeicher auswählen.

Komponente $\langle C20 \rangle$**: <ImageProcessing>**
Wandelt das erhaltene Bild eines Zeugnisses in den darauf enthaltenen Text um.

Komponente $\langle C30 \rangle$**: <Verification>**
Lässt den Nutzer die erkannten Daten aus dem Zeugnis überprüfen und gleicht diese dann mithilfe der Communication Komponente mit der Datenbank des Servers ab. Anschließend zeigt sie das Ergebnis an.

Komponente $\langle C40 \rangle$**: <Communication>**
Verwaltet auf der Client-Seite die Kommunikation mit dem Server.

Komponente $\langle C50 \rangle$**: <Data>**
Speichert den erkannten Text temporär nach Kontext sortiert ab. Enthält außerdem die globalen Einstellungen der App.

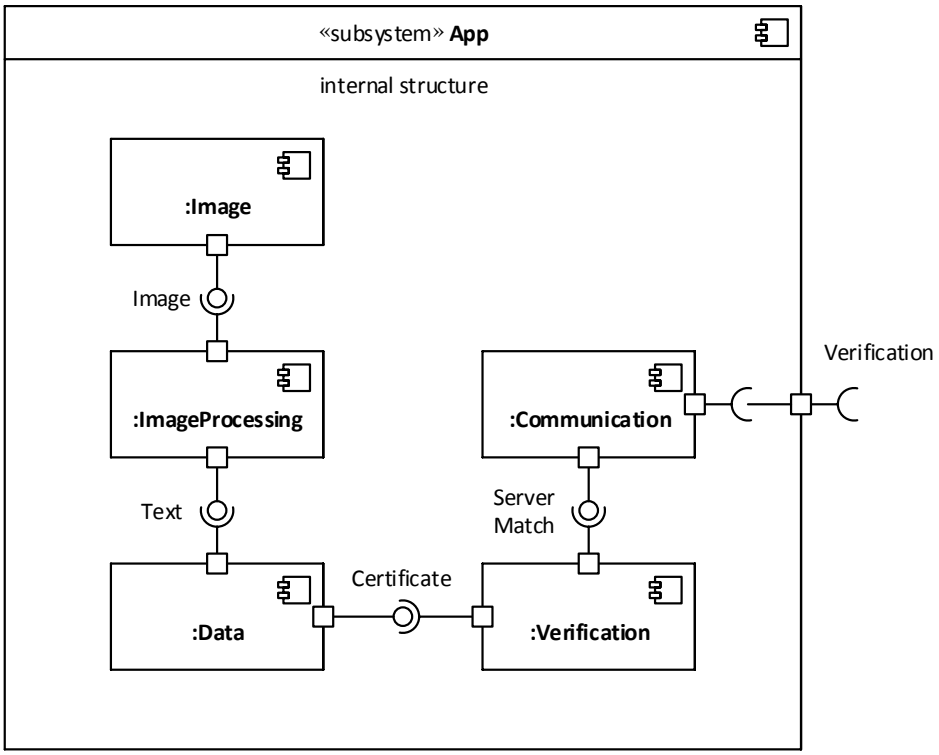

Abbildung 3.1: Komponentendiagramm.

3.1.2 Schnittstellenspezifikation

Im Folgenden werden die einzelnen Schnittstellen der Komponenten aus der Komponentenspezifikation näher erläutert, d.h. die von ihnen zur Verfügung gestellten Operationen werden dokumentiert.

Anmerkung: Die Schnittstellen Image und Certificate sind mithilfe von Intents und Extras implementiert. Hierbei ruft die vorausgehende Komponente die nachfolgende mit `Intent intent = new Intent()` auf und fügt mit `intent.putExtra()` die zu übergebenen Daten hinzu. Die nachfolgende Komponente ruft diese Daten mit `getIntent().getExtras()` wieder ab.

Schnittstelle $\langle I10 \rangle$**: <Image>**

Operation	**Beschreibung**
intent.putExtra("Uri", Uri.parse(picturePath));	Das Bild an der Stelle picturePath im Telefonspeicher wird übergeben.

Schnittstelle $\langle I20 \rangle$: <Text>

Operation	Beschreibung
new Certificate(String ocr)	Hier wird der Constructor des Certificate Objektes mit dem in der Image-Komponente erhaltenen String ocr aufgerufen.

Schnittstelle $\langle I30 \rangle$: <Certificate>

Operation	Beschreibung
intent.putExtra("Cert", cert);	Das Certificate-Objekt cert mit allen Informationen vom Zeugnis wird übergeben.

Schnittstelle $\langle I40 \rangle$: <Server Match>

Operation	Beschreibung
Communication.validateCertificate(String id, String content, Context context)	Hiermit wird geprüft, ob das Zeugnis mit der SecureID id und dem Inhalt content in der Datenbank des Servers hinterlegt ist.

3.1.3 Protokolle für die Benutzung der Komponenten

Außerhalb dieses Projekts ist die Wiederverwendung der erstellten Komponenten nicht sinnvoll möglich, da sie stark auf die gemeinsame Verwendung in Android 5 und zur Lösung einer spezifischen Aufgabe ausgelegt sind.

3.2 Server

Im folgenden wird die Softwarearchitektur des Servers erklärt.

3.2.1 Komponentenspezifikation

Aus den Komponenten des Servers ergibt sich folgendes Diagramm.

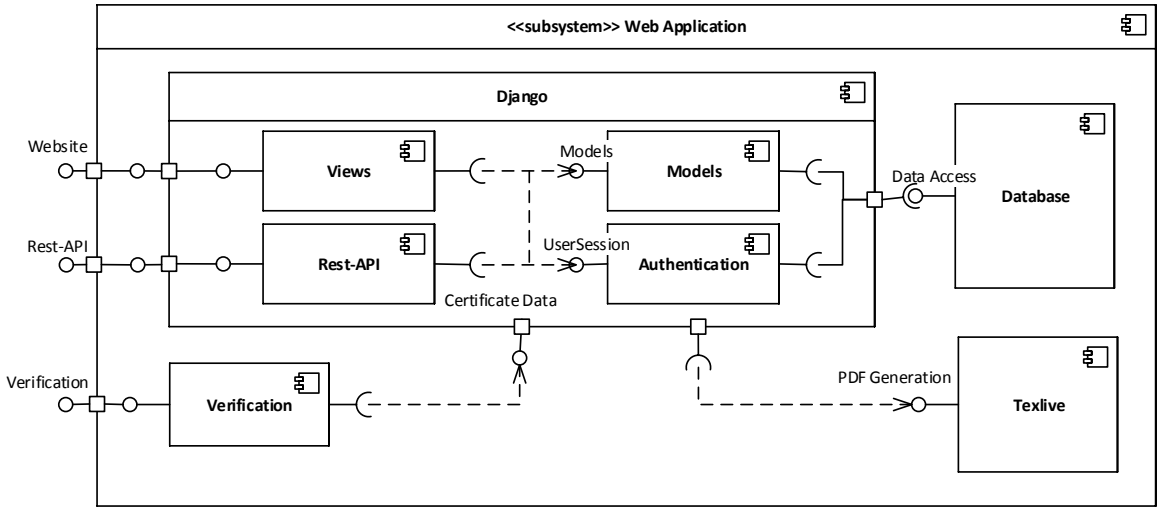

Abbildung 3.2: Serverkomponenten

Im Folgenden werden die Komponenten aus der Abbildung 3.2 beschrieben.

Komponente $\langle C60 \rangle$**: Authentication**
Bietet die Möglichkeit sich als authorisierter Nutzer einzuloggen.

Komponente $\langle C70 \rangle$**: Views**
Bietet eine grafische Oberfläche für authorisierte Nutzer, um Zeugnisse und andere Daten in der Datenbank anzulegen, zu löschen oder zu modifizieren. Weiterhin können über diese Schnittstelle auch Zeugnisse als PDF generiert werden.

Komponente $\langle C80 \rangle$**: Verification**
Ermöglicht das Verifizieren von Zeugnissen, auch für nicht authorisierte Nutzer.

Komponente $\langle C90 \rangle$**: Models**
Verwaltet die Zugriffe auf die Datenbank und bietet in Form von Models eine vereinfachte

Schnittstelle um auf den Daten der Datenbank zu arbeiten.

Komponente ⟨$C100$⟩**: Database**
Enthält die Daten aller Zeugnisse und Nutzer. Diese können verwaltet und ausgelesen werden.

Komponente ⟨$C110$⟩**: Texlive**
Eine externe Komponente, die Zeugnisse als PDF generieren kann.

Komponente ⟨$C120$⟩**: Rest-API**
Stellt eine REST-API bereit mit der die Daten geändert werden können.

Komponente ⟨$C130$⟩**: Django**
Das Django Framework.

3.2.2 Schnittstellenspezifikation

Der überwiegende Teil der Schnittstellen wird von Django und den verwendeten Frameworks bereitgestellt, sodass sich kaum eigene Schnittstellen ergeben. Für eine umfangreiche Dokumentation dieser Schnittstellen ist auf die jeweilige Dokumentation zu verweisen.

Anmerkung: Hier soll <typ>/<id> gleichbedeutend sein mit dem letzten Teil einer URL, unter der die Webseite erreichbar ist. <typ> sei ein Model aus certificates.models und <id> der primäre Schlüssel des betreffenden Objektes.

Schnittstelle ⟨$I80$⟩**: UserSession**

Siehe hierzu die Dokumentation von Django.

Schnittstelle ⟨$I90$⟩**: Website**

Operation	**Beschreibung**
`GET /admin`	Stellt das Webinterface für die Zeugnisverwaltung bereit.

Schnittstelle ⟨$I100$⟩**: Verification**

Operation	**Beschreibung**
`GET certificate/<id>`	Benutzt verify über HTTP.

Schnittstelle ⟨*I*110⟩**: Models**

Operation	Beschreibung
`certificates.models.Certificate.update_hash()`	Aktualisiert den Hash-Wert des Zeugnisses auf dem es aufgerufen wird.
`certificates.models.<some_class>.save()`	Speichert das Objekt in der Datenbank.
`certificates.models.<some_class>()`	Erstellt ein neues Objekt aus dem Datenmodell.

Schnittstelle ⟨*I*120⟩**: Data Access**

Siehe hierzu die Dokumentation von SQLite 3.

Schnittstelle ⟨*I*130⟩**: PDF Generation**

Siehe hierzu die Dokumentation von TexLive.

Schnittstelle ⟨*I*140⟩**: Rest-API**

Operation	Beschreibung
`GET <typ>/<id>`	Ruft ein Objekt aus dem Datenmodell ab.
`GET <typ>/`	Listet alle Objekte aus dem Datenmodell auf.
`POST <typ>/`	Erstellt ein neues Objekt in dem Datenmodell.
`PUT <typ>/<id>`	Ändert ein bestehendes Objekt in dem Datenmodell.
`DELETE <typ>/<id>`	Löscht ein bestehendes Objekt aus dem Datenmodell.

3.2.3 Protokolle für die Benutzung der Komponenten

Außerhalb dieses Projekts ist die Wiederverwendung der erstellten Komponenten nicht sinnvoll möglich, da sie darauf ausgerichtet sind innerhalb von Django verwendet zu werden und speziell auf das verwendete Datenmodell angepasst sind.

4 Verteilungsentwurf

Dieses Kapitel beschäftigt sich mit dem Verteilungsentwurf unseres Softwareprojektes.

4.1 Verteilungsdiagramm

In diesem Unterkapitel wird die Verteilung des Systems beschrieben.

4.2 Erklärung

Im Folgenden wird die Abbildung 4.1 kurz erklärt.

Die App läuft auf einem Smartphone mit dem Betriebssystem Android auf Version 5.0 oder höher. Die darauf laufende Java 7 VM führt alle Komponenten der App aus. Das TessTwo Paket benötigt dabei für das korrekte Erkennen von deutschem Text die Datei `deu.traineddata`. Außerdem wird das Zertifikat `Studentcloud216.scloud.etc.tu-bs.de.cer` für die Kommunikation mit dem Server benötigt.

Diese Kommunikation findet über das TCP/IP Protokoll statt.

Der Server ist eine Serverhardware oder ein VServer mit Verbindung zum Internet. Darauf läuft ein Debian Linux, dass über das Internet mit der App verbunden ist. Darauf läuft ein Apache-Webserver, der beim Start das Modul WSGI lädt und seine Einstellungen unter anderem aus der Datei `"certcheck.conf"` bezieht. WSGI lädt Django und vermittelt die Kommunikation von Django mit der App über den Webserver. Django lädt seine Konfiguration aus der Datei `"settings.py"` und benutzt eine Datenbankdatei, deren Name und Ort im Dateisystem von den Einstellungen in `"settings.py"` abhängen, um Daten permanent zu speichern. Django ruft das Compilerprogramm von TexLive auf, um Zeugnisse zu erstellen. Dabei wird zur Erstellung eines Zeugnisses die Zeugnisvorlage `"certificate_form.tex"` genutzt.

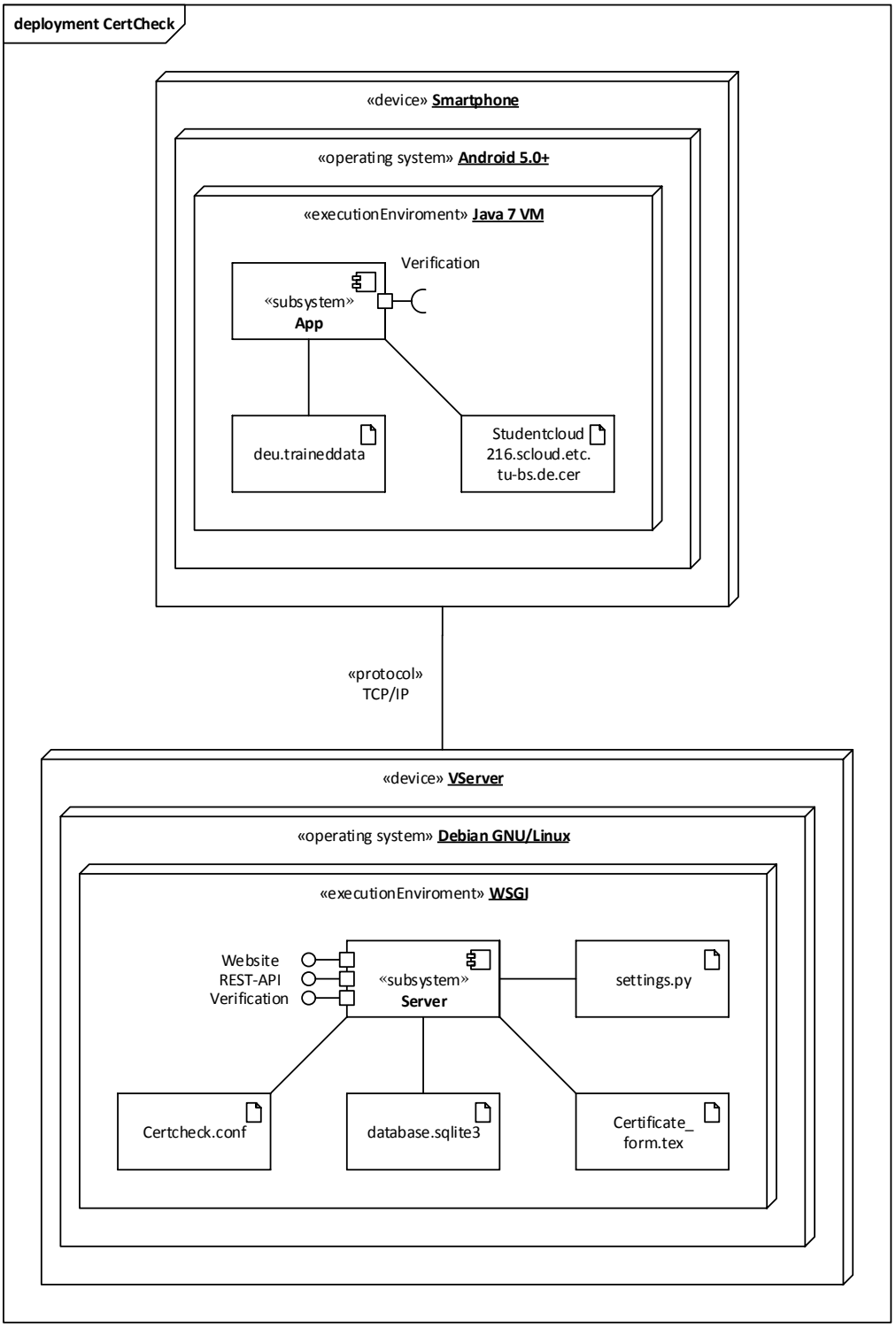

Abbildung 4.1: Verteilungsdiagramm

5 Implementierungsentwurf

In diesem Abschnitt wird die Implementierung des Produkts erläutert. Hierbei wird auf die Implementierung der Komponenten gesondert eingegangen und die erstellten Klassen und Attribute werden genau beschrieben.

5.1 Implementierung der Android-Applikation

Das folgende Diagramm gibt eine allgemeine Übersicht über alle Klassen der Android-Applikation. In den darauf folgenden Unterkapiteln werden dann die einzelnen Komponenten detailliert betrachtet.

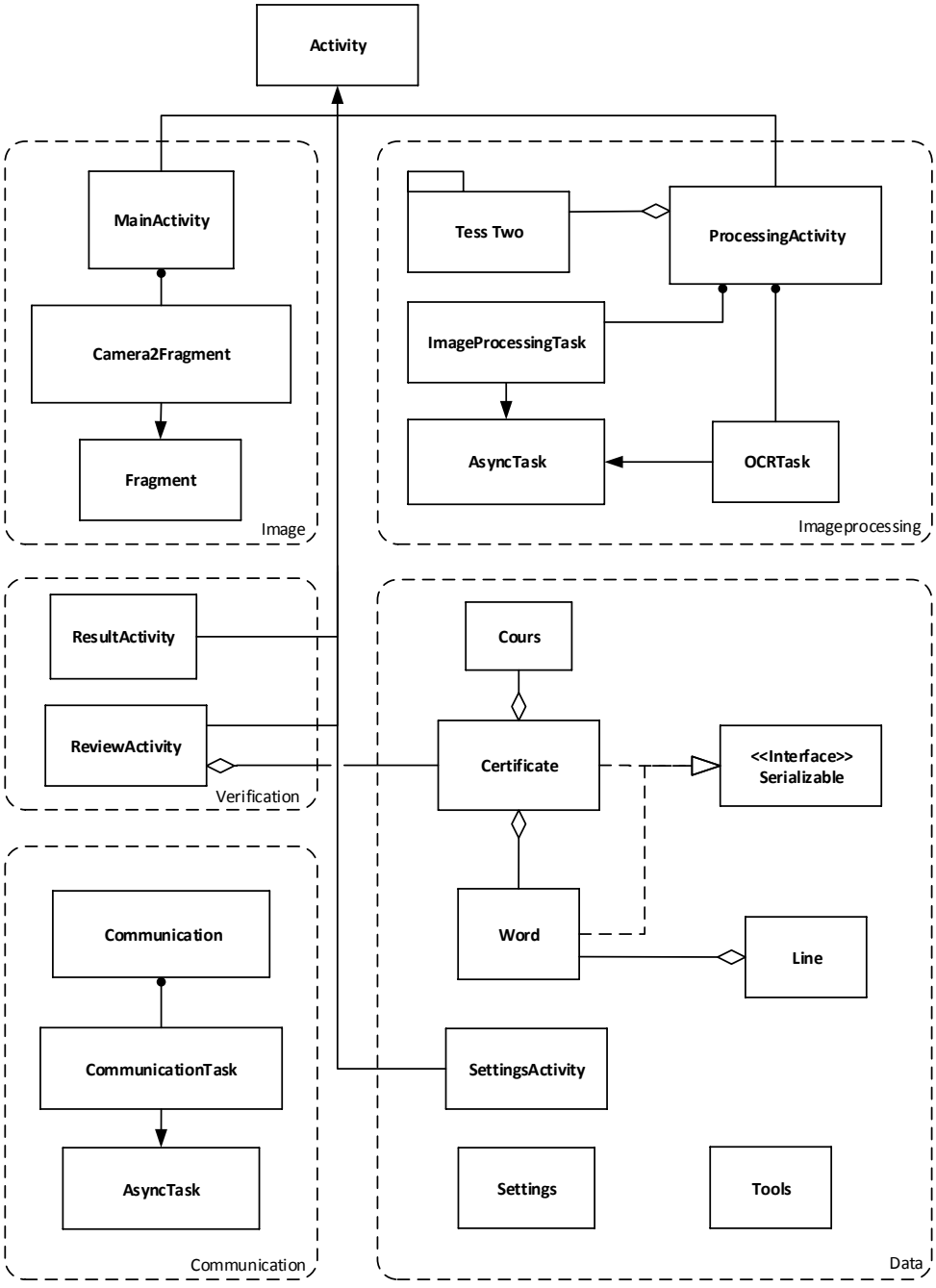

Abbildung 5.1: Klassendiagramm der gesamten Android Applikation

5.1.1 Implementierung von Komponente $\langle C10 \rangle$ Image

Die Komponente Image beinhaltet den Code um ein Bild aufzunehmen oder ein Bild aus der Galerie auszuwählen.

Paket-/Klassendiagramm

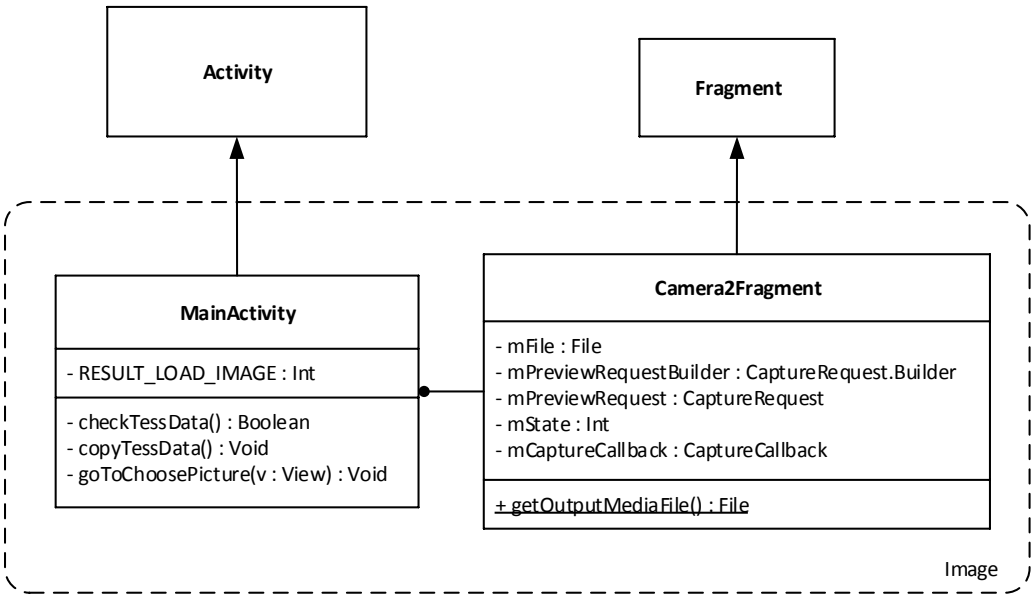

Abbildung 5.2: Klassendiagramm der Komponente Image

Erläuterung

MainActivity⟨$CL10$⟩

Aufgabe

Die MainActivity ist die Startactivity. Sie startet das Kamerainterface und stellt sicher, dass die für die OCR-Bibliothek benötigten Dateien am richtigen Speicherort sind.

Operationen

`Boolean checkTessData` Überprüft, ob die benötigten Dateien an ihrem Speicherort vorhanden sind.

`Void copyTessData()` Kopiert für die OCR-Bibliothek benötigte Dateien an einen bestimmten Speicherort.

Camera2Fragment⟨$CL20$⟩

Aufgabe

Darstellung der Kameraoberfläche.

Aufnahme eines Fotos.

Attribute

`File mFile` Das aufgenommene Foto.

Operationen

`File getOutputMediaFile()` Gibt ein File zurück, in dem ein Foto gespeichert wird.

Kommunikationspartner

ProcessingActivity

5.1.2 Implementierung von Komponente ⟨C20⟩ Imageprocessing

Die Bildbearbeitung und die Texterkennung wird in der Komponente ImageProcessing vollzogen. Für die Texterkennung benutzen wir die Open Source Bibliothek TessTwo.

Paket-/Klassendiagramm

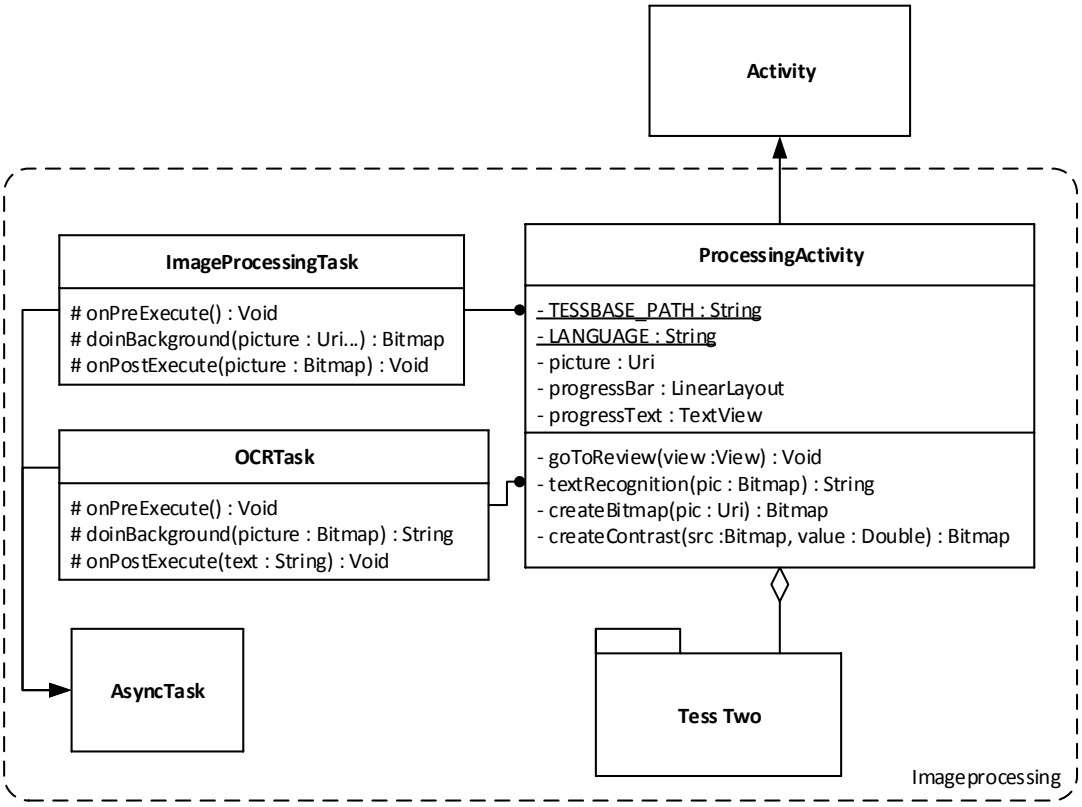

Abbildung 5.3: Klassendiagramm Komponente Imageprocessing

Erläuterung

ProcessingActivity⟨CL30⟩

Aufgabe

Vorbereitung das Bildes für die OCR. Anwendung der OCR Bibliothek TessTwo.

Attribute

`Uri picture` Verweis auf den Speicherort des Fotos.

`String erkannterText` Der von der OCR-Bibliothek erkannte Text.

`String LANGUAGE` Die zu erkennende Sprache.

`String TESSBASE_PATH` Pfad zu einer Trainingsdatei, die die OCR-Bibliothek benötigt.

Operationen

`Bitmap createBitmap(Uri pic)` Erzeugt ein Bitmap.

`String textRecognition(Bitmap pic)` Wendet die OCR Bibliothek TessTwo auf ein Bitmap an.

Kommunikationspartner

TessTwo

ReviewActivity

ImageProcessingTask$\langle CL40 \rangle$

Aufgabe

Vollzieht die Bildbearbeitung in einem separaten Thread.

Operationen

`Void onPreExecute()` Blendet einen Text und eine Progressbar ein.

`Bitmap doInBackground(Uri... picture)` Schneidet das Foto zu und optimiert es für die Texterkennung.

`Void onPostExecute(Bitmap picture)` Blendet die Progressbar aus und startet einen Thread zur Texterkennung.

Kommunikationspartner

OCRTask

OCRTask$\langle CL50 \rangle$

Aufgabe

Vollzieht die Texterkennung in einem separaten Thread.

Operationen

`Void onPreExecute()` Blendet einen Text ein.

`String doInBackground(Bitmap picture)` Wendet die OCR-Bibliothek TessTwo auf picture an und gibt das Ergebnis als String zurück.

`Void onPostExecute(Bitmap picture)` Startet die nächste Activity und übergibt das Ergebnis der Texterkennung.

Kommunikationspartner

ReviewActivity

5.1.3 Implementierung von Komponente $\langle C30 \rangle$ Communication

Die Komponente Communication stellt die Verbindung zu unserem Server her. Sie sendet Anfragen und empfängt die Antworten.

Paket-/Klassendiagramm

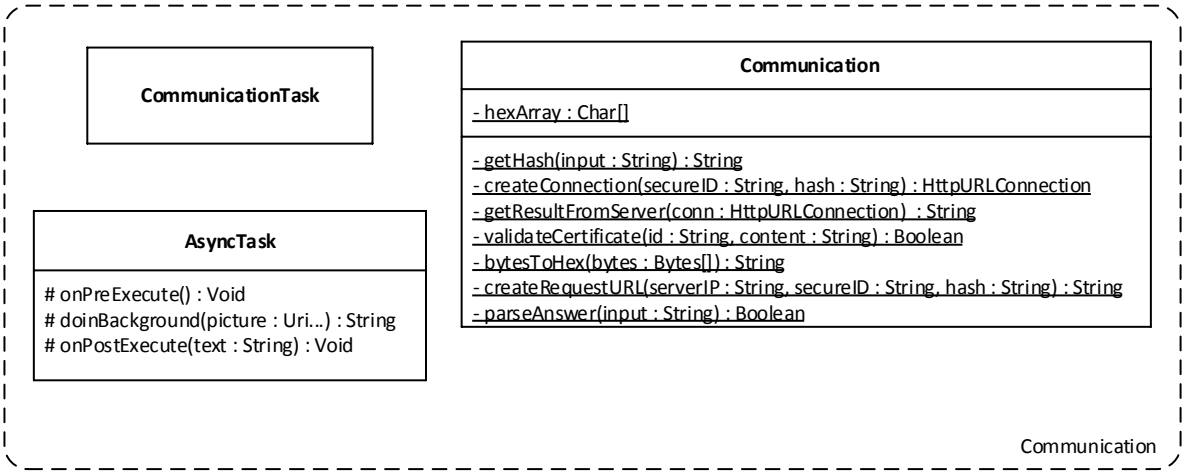

Abbildung 5.4: Klassendiagramm der Komponente Communication

Erläuterung

Communication$\langle CL60 \rangle$

Aufgabe

Verbindung zum Server herstellen.
Daten vom Server Empfangen.

Attribute

`Char[] hexArray` Definiert das HEX-Alphabtes.

Operationen

`String getHash(String input)` Erzeugt einen SHA-512 Hash aus einem String.

`HttpURLConnection createConnection(String secureID, String hash)` Stellt eine Http Verbindung her.

`String getResultFromServer(HttpURLConnection conn)` Liest das Ergebnis aus der Http Verbindung im Argument.

`Boolean validateVertificate(String id, String content)` Gibt das Ergebnis einer Validierung als Boolean wieder.
`String bytesToHex(Bytes[] bytes)` Übersetzt ein Byte Array in ein String aus hexadezimalen Zahlen
`String createRequestURL(String serverIP, String secureID, String Hash)` Erzeugt eine Url für eine Anfrage.
`Boolean parseAnswer(String input)` Übersetzt die Antwort des Servers in ein Boolean.

Kommunikationspartner

Server

CommunicationTask$\langle CL70 \rangle$

Aufgabe

Löst die Kommunikation vom Main-Thread.

Operationen

`Boolean doInBackground(Object... params)` Stellt eine Verbindung zum Server her und gibt das Ergebnis der Validierung zurück.
`Void onPostExecute(Boolean result)` Startet die nächste Activity und übergibt das Ergebnis der Validierung.

Kommunikationspartner

Communication

5.1.4 Implementierung von Komponente $\langle C40 \rangle$ Verification

Die Komponente Verification behandelt die notwendigen Schritte um die hauptsächliche Verifikation durchzuführen. Um mit dem Server zu kommunizieren bedient sie sich der Komponente Kommunikation.

Paket-/Klassendiagramm

Siehe Abbildung 5.5

Erläuterung

ReviewActivity$\langle CL80 \rangle$

Aufgabe

Darstellung des erkannten Textes in Textboxen.
Bereitstellung einer Korrekturmöglichkeit.

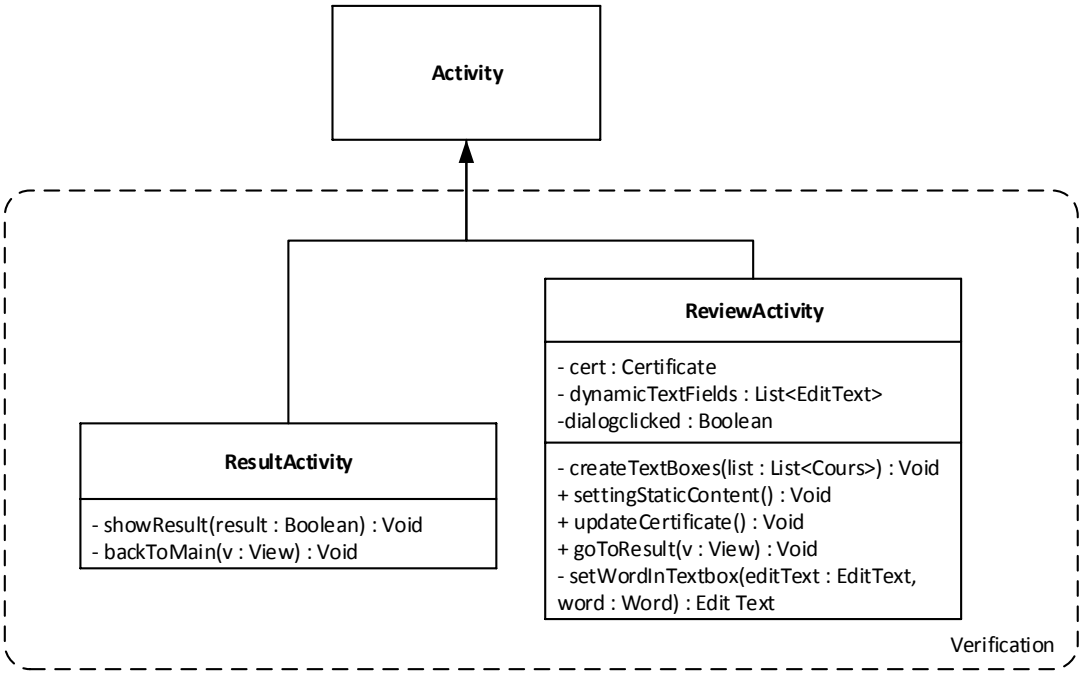

Abbildung 5.5: Klassendiagramm der Komponente Verification

Attribute

`Certificate cert` Das angezeigte Certificate.

`Boolean resultValue` Ergebnis der Serverabfrage.

Operationen

`Void createTextBoxes(List<Cours> list)` Erzeugt für jeden Kurs aus einer Liste ein Textfeld.

`Void settingStaticContent()` Erzeugt Textfelder für die Daten die in jedem Certificate vorhanden sind.

`Void updateCertificate()` Speichert Änderungen des Benutzers wieder im Certificate.

`Void goToResult(View v)` Startet die Activity, die das Ergebnis der Verifikation anzeigt.

ResultActivity$\langle CL90 \rangle$

Aufgabe

Visuelle Darstellung des Verifikationsergebnises.

Operationen

`Void showResult(Boolean result)` Wechselt zur entsprechenden Ansicht abhängig von `result`.

`Void backToMain(View v)` Wechselt die Activity zur MainActivity.

5.1.5 Implementierung der Komponente $\langle C50 \rangle$ Data

Die Komponente Data behandelt die Speicherung und Bearbeitung von Daten.

Paket-/Klassendiagramm

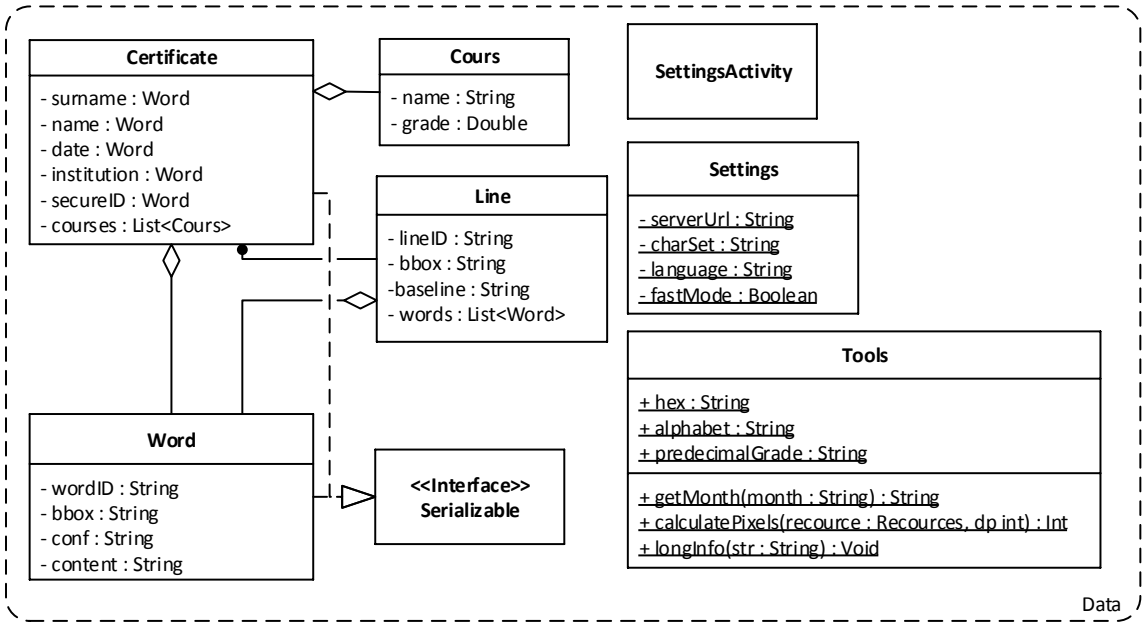

Abbildung 5.6: Klassendiagramm der Komponente Data

Erläuterung

Certificate$\langle CL100 \rangle$

Aufgabe

Praktische Speicherung eines Zeugnisses.

Attribute

`Word surname` Nachname

`Word name` Vorname

`Word date` Datum
`Word institution` Institut
`Word secureID` Eindeutige ID
`ListCours courses` Liste der belegten Kurse

Cours⟨$CL110$⟩

Aufgabe

Praktische Speicherung eines Kurses.

Attribute

`String name` Name des Kurses.
`Double grade` Note

Word⟨$CL120$⟩

Aufgabe

Speicherung eines erkannten Wortes mit zusätzlichen, von Tess Two bereitgestellten, Informationen.

Attribute

`String wordID` Eindeutige ID.
`String bbox` Position des Wortes.
`String conf` Wahrscheinlichkeit, dass das Wort richtig erkannt wurde.
`String content` Erkanntes Wort.

Line⟨$CL130$⟩

Aufgabe

Speicherung einer erkannten Zeile.

Attribute

`String lineID` Eindeutige ID.
`String bbox` Position der Zeile.
`String baseline` Grundlinie der Zeile.
`ListWord` Liste der Worte in der Zeile.

SettingsActivity⟨$CL140$⟩

Aufgabe

Änderungen der Settings.

Settings⟨$CL150$⟩

Aufgabe

Beinhaltet wichtige Parameter zur Konfiguration der App.

Attribute

`String serverUrl` Url des Servers.

`String charSet` Formatierung von Strings.

`String language` Sprache der Applikation.

`Boolean fastmode` Modus, der die ReviewActivity überspringt.

Tools⟨$CL160$⟩

Aufgabe

Ansammlung von nützlichen Methoden.

Attribute

`String hex` Definiert die hexadezimalen Ziffern.

`String alphabet` Definiert das Alphabet.

`String predecimalGrade` Definiert den Zahlenbereich der Noten.

Operationen

`String getMonth(String month)` Gibt die Zahl eines Monats zurück.

`Int calculatePixels(Recources recource, Int dp)` Errechnet die Pixel für einen bestimmten DP-Wert.

`Void longInfo(String str)` Zerteilt lange Nachrichten in mehrere kleine Nachrichten.

5.2 Implementierung des Servers

Im folgenden wird die Implementierung des Servers erklärt.

Zu Komponenten und Klassen, deren Dokumentationen nicht bereits durch bestehende andere Dokumentationen abgedeckt wird, existiert im Quellcode und in `"server/doc"` eine Dokumentation.

Anmerkung: Die Komponenten $\langle C100 \rangle$ Database, $\langle C60 \rangle$ Authentication, $\langle C110 \rangle$ Texlive, und $\langle C130 \rangle$ werden aus bestehenden Softwareprodukten bereitgestellt und sind anderswo dokumentiert.

5.2.1 Implementierung von Komponente $\langle C70 \rangle$: Views

Diese Komponente stellt ein Webinterface zur Interaktion mit dem Benutzer bereit, um Zeugnisse zu verwalten. Sie kann zudem mithilfe von $\langle C110 \rangle$ PDFs für Zeugnisse erstellen und zum Download anbieten.

Paket-/Klassendiagramm

Siehe Abbildung 5.7

Erläuterung

certificates.admin.GradeInLine$\langle CL170 \rangle$

Aufgabe

Darstellung der Zeugnisnoten innerhalb des Zeugnisses

Attribute

`model` die zu verwendende Klasse aus dem Datenmodell

`raw_id_fields` legt fest wie die Noten angezeigt werden sollen

`auutocomplete_lookup_fields` legt fest nach welchem Attribut die Noten autovervollständigt werden sollen

Operationen

keine

Kommunikationspartner

Django, django.contrib.admin

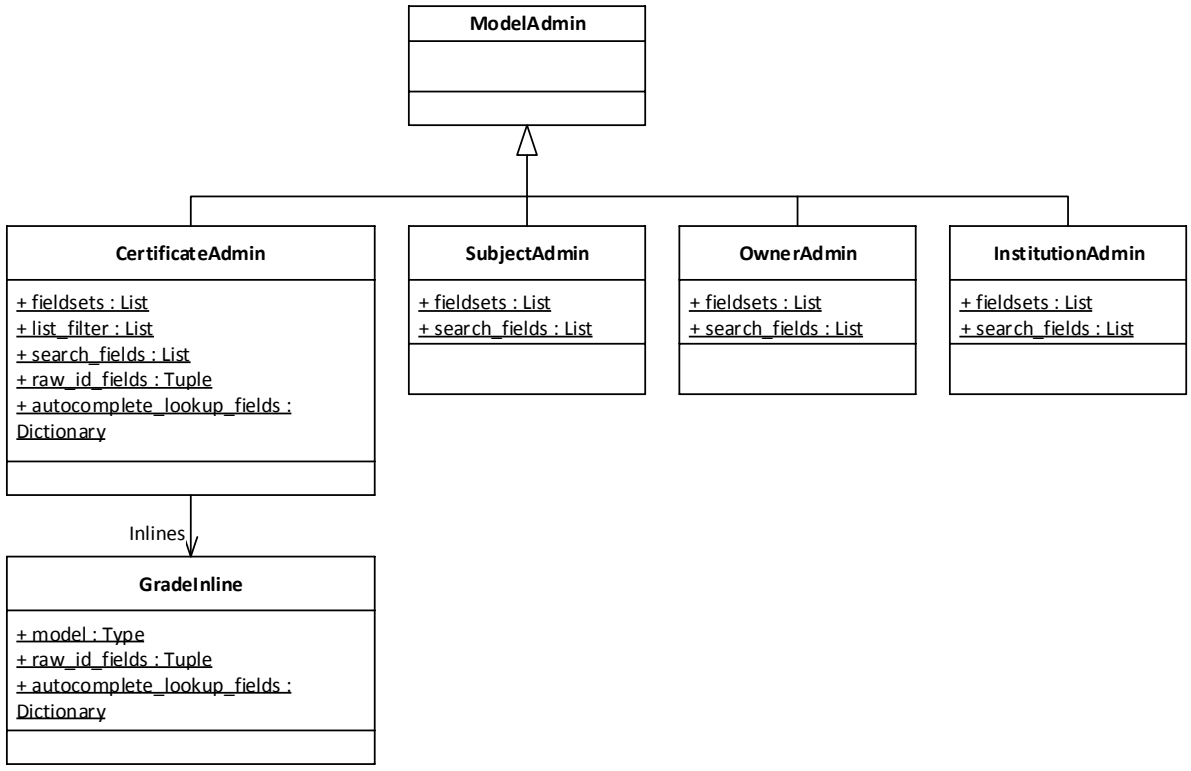

Abbildung 5.7: Klassendiagram der Komponente Views

certificates.admin.CertificateAdmin$\langle CL180 \rangle$

Aufgabe

Darstellung des Formulars zum Verwalten von Zeugnissen

Attribute

`fieldsets` beschreibt die Formularfelder, die angezeigt werden sollen

`inlines` gibt an welche Dinge direkt im Formular angezeigt werden sollen, anstatt in einem gesonderten Formular

`list_filter` legt fest wonach die Zeugnisse sortiert werden, wenn sie in der Auflistung angezeigt werden

`search_fields` legt fest wonach die Zeugnisse durchsuchbar sind

`raw_id_fields` legt fest wonach die Zeugnisse bei der Vervollständigung angezeigt werden sollen

`autocomplete_lookup_fields` legt fest nach welchen Attributen die Zeugnisse bei der Autovervollständigung durchsucht werden sollen

Operationen

keine

Kommunikationspartner

Django

certificates.admin.InstitutionAdmin⟨$CL190$⟩

Aufgabe

Darstellung des Formulars zum Verwalten von Insitutionen

Attribute

`fieldsets` beschreibt die Formularfelder, die angezeigt werden sollen

`search_fields` legt fest wonach die Zeugnisse durchsuchbar sind

Operationen

keine

Kommunikationspartner

Django

certificates.admin.SubjectAdmin⟨$CL200$⟩

Aufgabe

Darstellung des Formulars zum Verwalten von Fächern

Attribute

`fieldsets` beschreibt die Formularfelder, die angezeigt werden sollen

`search_fields` legt fest wonach die Zeugnisse durchsuchbar sind

Operationen

keine

Kommunikationspartner

Django

certificates.admin.OwnerAdmin⟨$CL210$⟩

Aufgabe

Darstellung des Formulars zum Verwalten von Zeugnisinhabern

Attribute

`fieldsets` beschreibt die Formularfelder, die angezeigt werden sollen

`search_fields` legt fest wonach die Zeugnisse durchsuchbar sind

Operationen

keine

Kommunikationspartner

Django

certificates.views.generate_pdf⟨$CL220$⟩

Aufgabe

generiert ein PDF aus den Daten des Zeugnisses, welches die angegebene SecureID hat

Attribute

Operationen

`generate_pdf(request, secure_id)` generiert ein PDF für das Certificate mit der SecureID `secure_id`

Kommunikationspartner

Django, jinja2, latex, certificates.models

5.2.2 Implementierung von Komponente ⟨$C80$⟩: Verification

Diese Komponente stellt die Verifikationsschnittstelle für die App bereit.

Paket-/Klassendiagramm

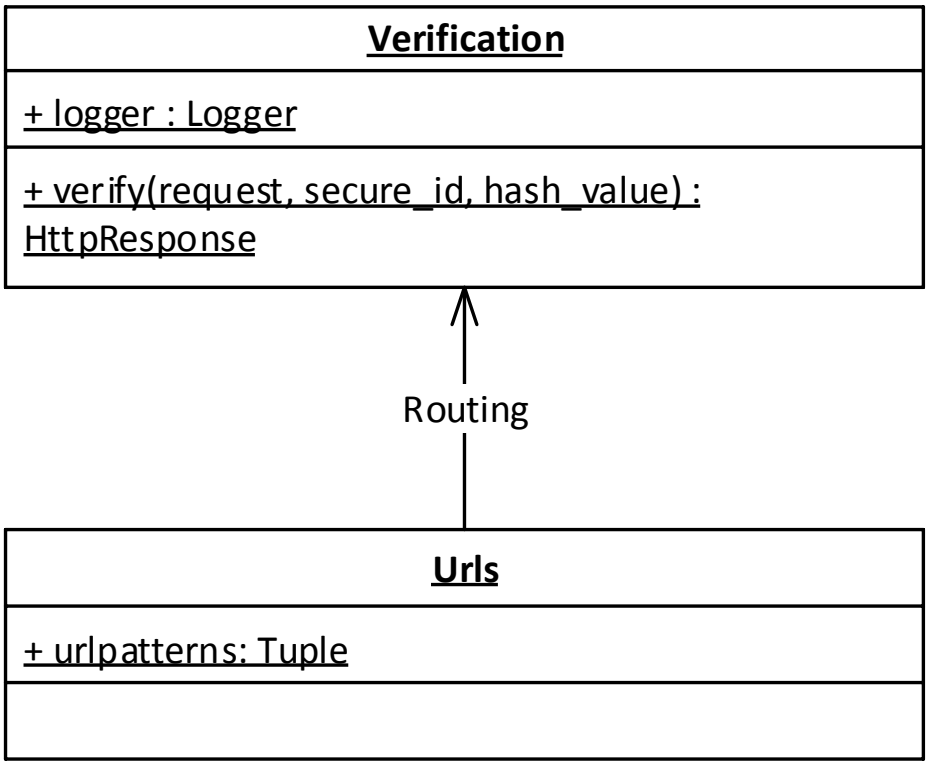

Abbildung 5.8: Klassendiagram der Komponente Verification

Erläuterung

certificates.views.verify$\langle CL230 \rangle$

Aufgabe

Verifikation von Hash-Wert und SecureID Paaren

Attribute

`logger` der in diesem Moul genutzte Logger

Operationen

`verify(request, secure_id, hash_value)` verifiziert ein Certificate anhand des gesendeten Hash-Wertes und der gesendeten SecureID

Kommunikationspartner

Django, json, sqlite3, certificates.models.certificate

5.2.3 Implementierung von Komponente $\langle C90 \rangle$: Models

Diese Komponente stellt das Datenmodell bereit.

Paket-/Klassendiagramm

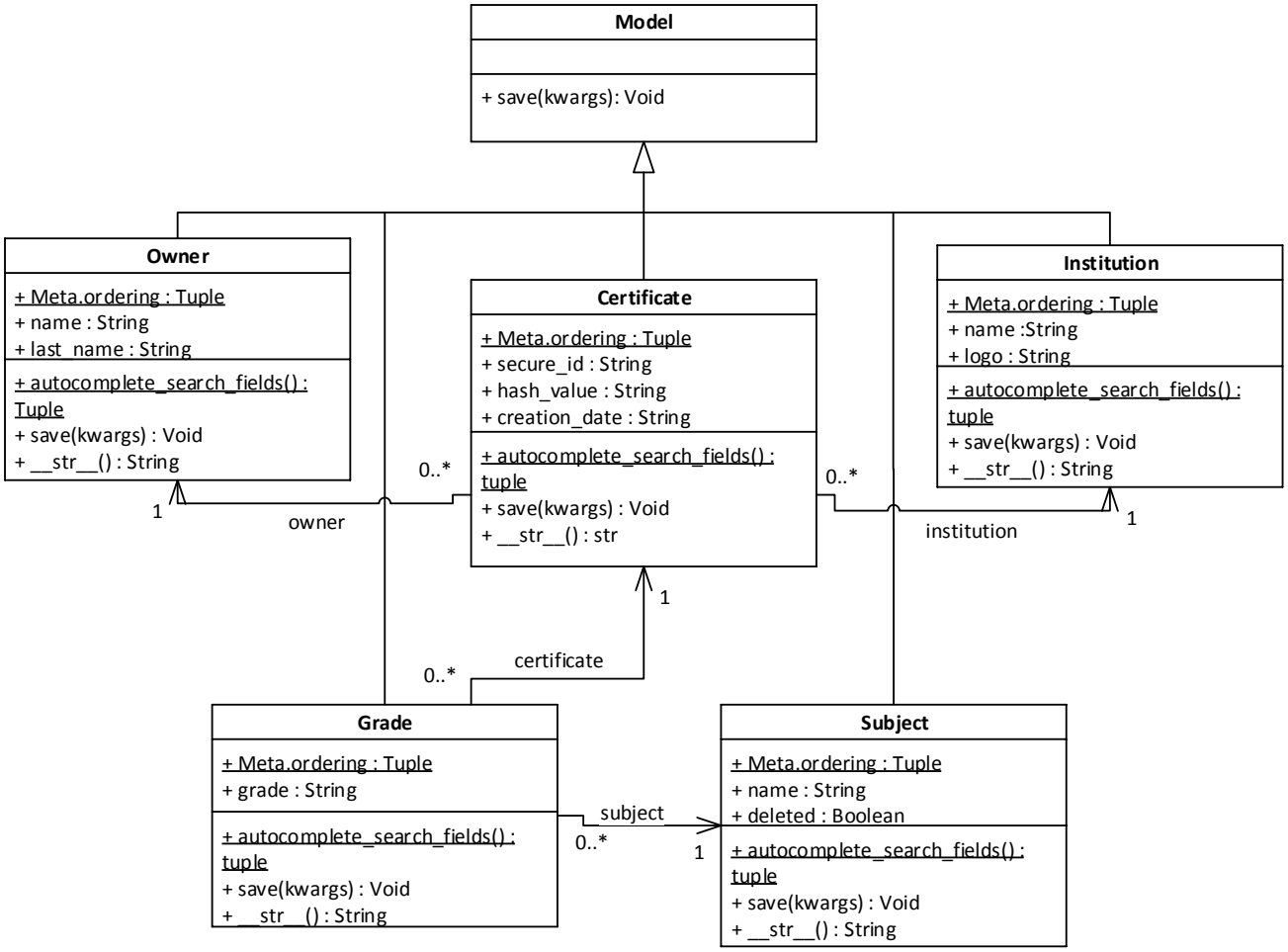

Abbildung 5.9: Klassendiagram der Komponente Models

Erläuterung

certificates.Owner⟨$CL240$⟩

Aufgabe

Stellt das Datenmodell für Zeugnisinhaber bereit

Attribute

`name` der Vorname des Zeugnisinhabers

`last_name` der Nachname des Zeugnisinhabers

`Meta` Die Meta-Klasse enthält Angaben zur Ordnung der Objekte dieser Klasse.

Operationen

`autocomplete_search_fields()` gibt an nach welchen Attributen im Admininterface eine automatische Vervollständigung durchgeführt werden soll

`save(self, **kwargs)` speichert das Owner-Object in die Datenbank

`__str__(self)` erstellt eine String-Representation für das Objekt

Kommunikationspartner

Django, jinja2, certificates.models

certificates.Subject⟨$CL250$⟩

Aufgabe

stellt das Datenmodell für Zeugnisfächer bereit

Attribute

`name` Name des Zeugnisfaches

`deleted` True, falls das Zeugnisfach nicht für neue Zeugnisse verfügbar sein soll, ansonsten False

`Meta` Die Meta-Klasse enthält Angaben zur Ordnung der Objekte dieser Klasse.

Operationen

`autocomplete_search_fields()` gibt an nach welchen Attributen im Admininterface eine automatische Vervollständigung durchgefüjrt werden soll

`save(self, **kwargs)` speichert das Objekt in die Datenbank.

`__str__(self)` erstellt eine String-Representation für das Objekt.

Kommunikationspartner Django, certificates.models.certificate, certificates.models.grade

certificates.Institution⟨$CL260$⟩

Aufgabe

stellt das Datenmodell für Institutionen, die Zeugnisse ausstellen bereit

Attribute

`name` Name der Institution

`logo` das Logo der ausstellenden Institution

`Meta` Die Meta-Klasse enthält Angaben zur Ordnung der Objekte dieser Klasse.

Operationen

`autocomplete_search_fields()` gibt an nach welchen Attributen im Admininterface eine automatische Vervollständigung durchgeführt werden soll

`save(self, **kwargs)` speichert das Objekt in die Datenbank.

`__str__(self)` erstellt eine String-Representation für das Objekt.

Kommunikationspartner

Django, certificates.models.certificate, certificates.models.grade

certificates.models⟨$CL270$⟩

Aufgabe

enthält Klassen für die verwendeten Datenmodelle

Attribute

keine

Operationen

`generate_secureid()` erstellt eine zufällige SecureID

Kommunikationspartner

keine

certificates.models.Certificate⟨*CL*280⟩

Aufgabe

stellt das Datenmodell für Zeugnisse bereit

Attribute

`secure_id` eine SecureID

`hash_value` der Hash-Wert der Zeugnisdaten

`creation_date` das Datum, an dem das Zeugnis ausgestellt beziehungsweise geändert wurde

`institution` die Institution, die dieses Zeugnis ausgestellt hat

`owner` der Inhaber dieses Zeugnis

`Meta` Die Meta-Klasse enthält Angaben zur Ordnung der Objekte dieser Klasse.

Operationen

`save(self, **kwargs)` aktualisiert den Hash-Wert, speichert anschließend das Objekt in die Datenbank.

`update_hash(self)` aktualisiert den Hash-Wert des Zeugnis

`__str__(self)` erstellt eine String-Representation für das Objekt.

Kommunikationspartner

Django, certificates.models.certificate, certificates.models.*

certificates.models.Grade⟨*CL*290⟩

Aufgabe

stellt das Datenmodell für Zeugnisnoten bereit

Attribute

`subject` das Zeugnisfach, für das die Note erteilt wurde

`grade` die erteilte Zeugnisnote

`certificate` das Zeugnis auf dem die Note stehen soll

`Meta` Die Meta-Klasse enthält Angaben zur Ordnung der Objekte dieser Klasse.

Operationen

`save(self, **kwargs)` Aktualisiert den Hash-Wert, speichert anschließend das Objekt in die Datenbank.

`__str__(self)` erstellt eine String-Repräsentation für das Objekt.

`__unicode__(self)` erstellt eine Unicode-Repräsentation für das Objekt.

Kommunikationspartner

Django, certificates.models.certificate, certificates.models.Subject

5.2.4 Implementierung von Komponente ⟨C120⟩: Rest-API

Zur Bereitstellung der REST-API wird das „Django REST Framework“ in Kombination mit Django genutzt.

Paket-/Klassendiagramm

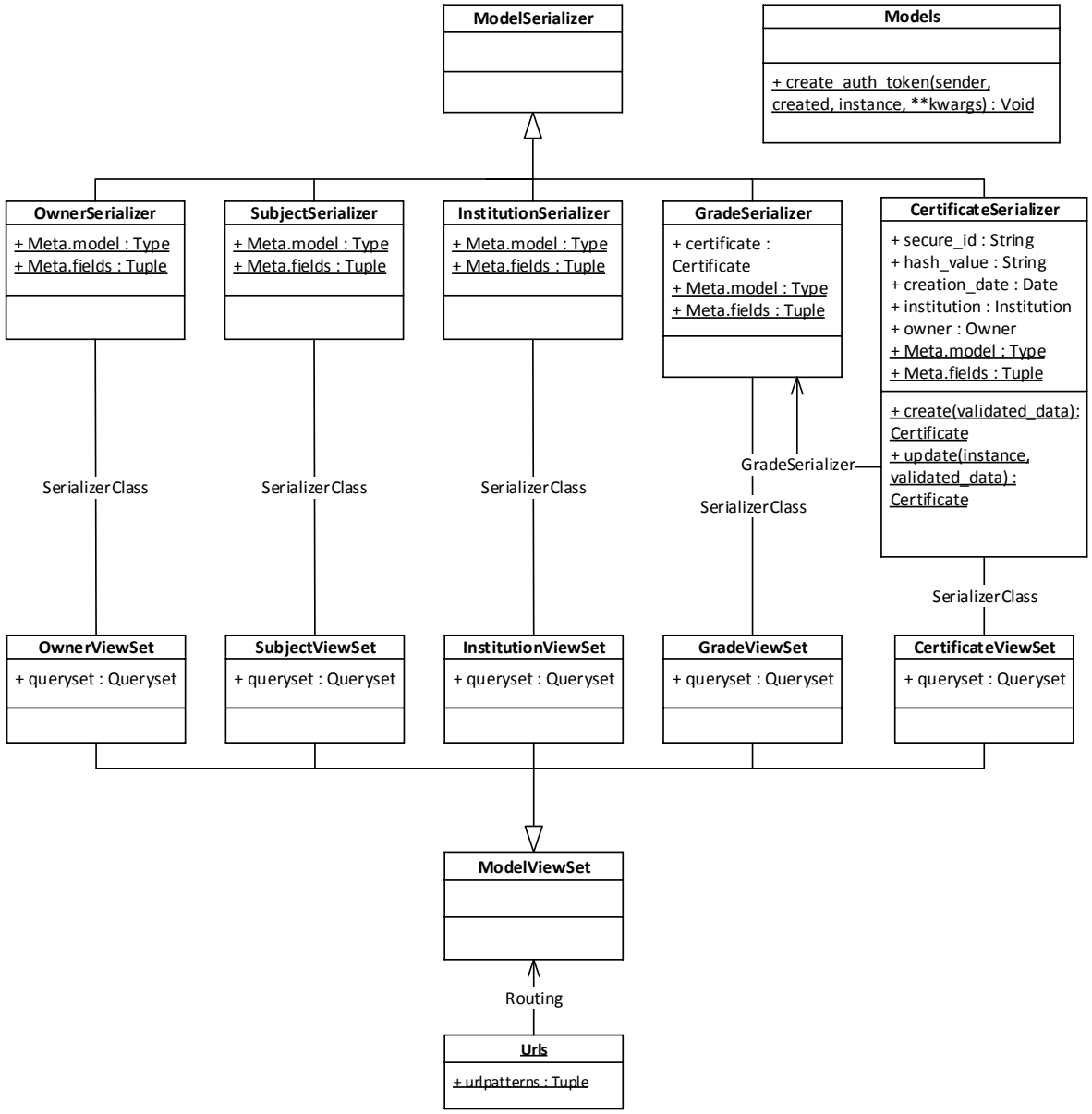

Abbildung 5.10: Klassendiagram der Komponente Rest-API

Erläuterung

certificates.models⟨$CL300$⟩

Aufgabe

Erstellung von neuen Authentifizierungstokens

Attribute

`Meta` Die Meta-Klasse enthält folgende Attribute:

`model` das zu verwendende Datenmodell

`fields` die zu serialisierenden Felder des Datenmodells

Operationen

`create_auth_token(sender, created, instamce, **kwargs)` erstellt neue Authentifizierungstokens

Kommunikationspartner

REST-Framework

certificates.serializers.OwnerSerializer⟨$CL310$⟩

Aufgabe

Serialisierung und Deserialisierung der über die REST-API ausgetauschten Daten für das Datenmodell certificates.serializers.Owner.

Attribute

`Meta` die Meta-Klasse enthält folgende Attribute:

`model` das zu verwendende Datenmodell

`fields` die zu serialisierenden Felder des Datenmodells

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.serializers.SubjectSerializer⟨$CL320$⟩

Aufgabe

Serialisierung und Deserialisierung der über die REST-API ausgetauschten Daten für das Datenmodell certificates.serializers.Subject.

Attribute

`Meta` die Meta-Klasse enthält folgende Attribute:

`model` das zu verwendende Datenmodell

`fields` die zu serialisierenden Felder des Datenmodells

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.serializers.InstitutionSerializer⟨$CL330$⟩

Aufgabe

Serialisierung und Deserialisierung der über die REST-API ausgetauschten Daten für das Datenmodell certificates.serializers.Institution.

Attribute

`Meta` Die Meta-Klasse enthält folgende Attribute:

`model` das zu verwendende Datenmodell

`fields` die zu serialisierenden Felder des Datenmodells.

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.serializers.GradeSerializer⟨$CL340$⟩

Aufgabe

Serialisierung und Deserialisierung der über die REST-API ausgetauschten Daten für das Datenmodell certificates.serializers.Grade.

Attribute

`certificate` der Primärschlüssel des referenzierten certificates.serializers.Certificate

`Meta` Die Meta-Klasse enthält folgende Attribute:

`model` das zu verwendende Datenmodell

`fields` die zu serialisierenden Felder des Datenmodells.

Operationen

keine

Kommunikationspartner

Diese Klasse Dise Klasse wird vom REST-Framework genutzt.

certificates.serializers.CertificateSerializer⟨$CL350$⟩

Aufgabe

Serialisierung und Deserialisierung der über die REST-API ausgetauschten Daten für das Datenmodell certificates.models.Certificate

Attribute

`secure_id` das Feld, welches die SecureID serialisiert

`hash_value` das Feld, welches den Hash-Wert serialisiert

`creation_date` das Feld, welches das Änderungsdatum beziehungsweise das Erstellungsdatum serialisiert

`institution` das Feld, welches die in Relation stehende Institution serialisisert

`owner` das Feld, welches den in Relation stehenden Zeugnisinhaber serialisisert

`grades` das Feld, welches die in Relation stehenden Noten serialisisert

`Meta` Die Meta-Klasse enthält folgende Attribute:

`model` das zu verwendende Datenmodell

`fields` die zu serialisierenden Felder des Datenmodells.

Operationen

`create(self, validated_data)` erstellt ein neues Zeugnis aus den deserialisierten, geprüften Daten

`update(self, instance, validated_data)` aktualisiert ein vorhandenes Zeugnis mit den validierten Daten

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.viewsets.OwnerViewset⟨$CL360$⟩

Aufgabe

Automatisiertes bereitstellen von views für die REST-API für das Owner-Datenmodell

Attribute

`queryset` die Objekte des Datenmodells für die die Views erstellt werden sollen

`serializer_class` der zu nutzende Serializer

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.viewsets.SubjectViewSet⟨$CL370$⟩

Aufgabe

Automatisiertes bereitstellen von views für die REST-API für das Subject-Datenmodell

Attribute

`queryset` die Objekte des Datenmodells für die die Views erstellt werden sollen

`serializer_class` der zu nutzende Serializer

Operationen

`destroy(self, request, *args, **kwargs)` Wenn eine Löschaktion für das mit dem request verbundene Subject-Objekt gesendet wird, setze Subject.deleted auf True.

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.viewsets.InstitutionViewset⟨$CL380$⟩

Aufgabe

Automatisiertes bereitstellen von views für die REST-API für das Institution-Datenmodell

Attribute

`queryset` die Objekte des Datenmodells für die die Views erstellt werden sollen

`serializer_class` der zu nutzende Serializer

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.viewsets.CertificateViewset⟨$CL390$⟩

Aufgabe

Automatisiertes bereitstellen von views für die REST-API für das Certificate-Datenmodell

Attribute

`queryset` die Objekte des Datenmodells für die die Views erstellt werden sollen

`serializer_class` der zu nutzende Serializer

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

certificates.viewsets.Grade⟨$CL400$⟩

Aufgabe

Automatisiertes bereitstellen von views für die REST-API für das Grade-Datenmodell

Attribute

`queryset` die Objekte des Datenmodells für die die Views erstellt werden sollen

`serializer_class` der zu nutzende Serializer

Operationen

keine

Kommunikationspartner

Dise Klasse wird vom REST-Framework genutzt.

5.2.5 Implementierung von Komponente $\langle C130 \rangle$: Django

Um Django zu verwenden müssen verschiedene Einstellungen vorgenommen werden.

Paket-/Klassendiagramm

Abbildung 5.11: Klassendiagram der Komponente Django

Erläuterung

certcheck.settings⟨*CL*410⟩

Aufgabe

Einstellungsoptionen für Django und die verwendeten Bibliotheken.

Attribute

`SECRET_KEY` ein Wert, der zur Sicherung der gespeicherten Daten dient

`DEBUG` wenn True, aktiviert django den Debug-Modus

`BASE_DIR` das Wurzelverzeichnis von Django

`TEMPLATE_DIRS` Dateisystemverzeichnisse, aus denen Django Templates bezieht

`ADMINS` Administratoren und ihre EMail-Adressen

`LOGGING` der Pfad zur Logdatei

`LOGGING` gibt an was wie geloggt wird

`TEMPPLATE_DEBUG` Debug-Modus für die Verwendung von Templates

`INSTALLED_APPS` zu verwendende Bibliotheken

`MIDDLEWARE_CLASSES` zu nutzende Middlewares

`ROOT_URLCONF` die zuerst nach URL-Weiterleitungen zu durchsuchende Datei

`WSGI_APPLICATION` die für die Einbindung in den Webserver zu nutzende Bibliothek

`DATABASES` Zugangsdaten für die Datenbank (Treiber und Pfad)

`LANGUAGE_CODE` die zu verwendende Ländereinstellung

`TIME_ZONE` die zu verwendende Zeitzone

`USE_I18N` kontrolliert die Übersetzung

`USE_L10N` kontrolliert Formatierung

`USE_TZ` aktiviert die Zeitzonenunterstützung

`STATIC_ROOT` gibt den Pfad zu dem Verzeichnis an, dass statische Dateien enthält

`STATIC_URL` gibt den Pfad der URL an, unter dem statische Dateien zu finden sein sollen

`STATICFILES_DIRS` gibt die Orte an, aus denen statische Dateien kopiert werden sollen

`REST_FRAMEWORK` Einstellungen für das REST-Framework

`TEMPLATE_CONTEXT_PROCESSORS` Context verarbeitende Klassen

`AUTHENTICATION_BACKENDS` Backends, die weitere Methoden zur Authentifizierung bereitstellen, wie beispielsweise LDAP

Operationen

keine

Kommunikationspartner

Dise Klasse wird von Django genutzt.

certcheck.urls⟨*CL*420⟩

Aufgabe

Weiterleitung der URL-Aufrufe an die Django-Apps

Attribute

`urlpaterns` enthält reguläre Ausdrücke und Ziele von Weiterleitungen

Operationen

keine

Kommunikationspartner

Django

certificates.urls⟨*CL*430⟩

Aufgabe

Weiterleitung der URL-Aufrufe an die Views

Attribute

`urlpaterns` enthält reguläre Ausdrücke und Ziele von Weiterleitungen

`router` enthält den Router, der für die Weiterleitung der API-Views genutzt wird.

Operationen

keine

Kommunikationspartner

Django

certcheck.wsgi⟨*CL*440⟩

Aufgabe

Enthält Einstellungen für das Modul WSGI

Attribute

`application` verweist auf die Verwendete WSGI-Instanz

Operationen

keine

Kommunikationspartner

Django

6 Datenmodell

Im folgenden werden die bereits im Pflichtenheft aufgeführten Daten, die für den bestimmungsgemäßen Gebrauch der Anwendung gespeichert werden, genauer spezifiziert und der Grund der Speicherung erläutert.

6.1 Diagramm

Das nachfolgende Diagramm beschreibt das Datenmodell des Produktes.

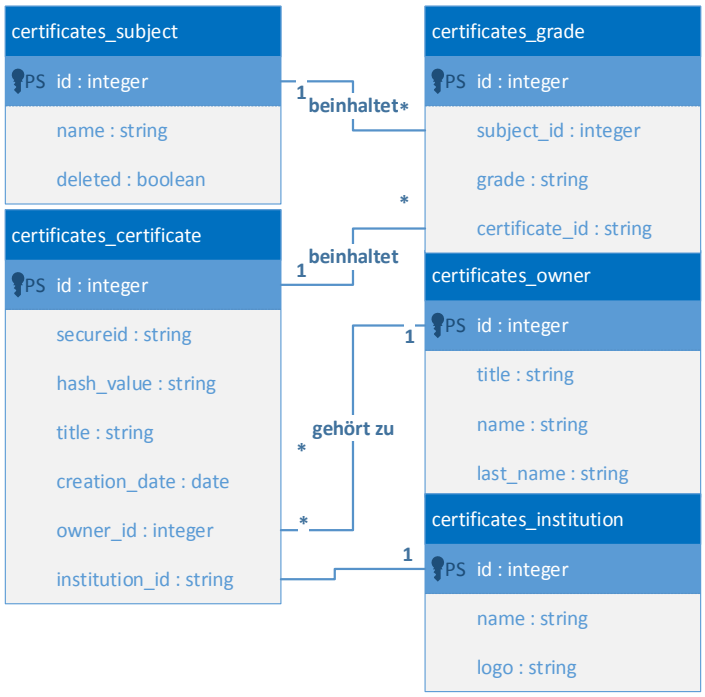

Abbildung 6.1: Klassendiagramm: Datenbankmodell

6.2 Erläuterung

Im folgenden wird das Datenmodell in Abbildung 6.1 kurz erläutert.

Da die durch diese Anwendung gespeicherten Daten auf ein Minimum reduziert wurden, sind bei allen Entitäten die Attribute vollständig aufgeführt. Die jeweiligen Aufteilungen auf die unterschiedlichen Datenbanktabellen wurden vorgenommen um die gesamte Datenbank in die 3. Normalform zu bringen und damit den verbundenen Wartungsaufwand bei etwaigen Änderungen gering zu halten.

Aufgrund dieser nötigen und sinnvollen Erweiterung der Datenbank ergeben sich folgende Abhängigkeiten zwischen den einzelnen Datenbanktabellen.

certificates_subject $\langle E10 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Zeugnisnoten	1 zu *	Min: 1, Max: Gesamtanzahl aller Fächer	beinhaltet die Einzelnoten der auf einem Zeugnis aufgeführten Fächer

certificates_grade $\langle E20 \rangle$

Beziehung	**Kardinalität**	**Erwartete Datenmenge**	**Beschreibung**
Zeugnisfach	* zu 1	Min: 1 pro Zeugnis, Max: 10 pro Zeugnis	beinhaltet das benotete Zeugnisfach
Zeugnis	* zu 1	Min: 0, Max: gesamte Zeugnisanzahl	beinhaltet das Zeugnis zu dem die Note gehört

certificates_certificate ⟨*E*30⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Zeugnisnoten	1 zu *	Min: 1 pro Zeugnis, Max: 10 pro Zeugnis	die zum Zeugnis gehörigen Fächer und Noten
Zeugnisinhaber	* zu 1	Min: 0, Max: Gesamtanzahl derer die ein Zeugnis erhalten haben	die Studenten/ Schüler denen ein Zeugnis ausgestellt wird
Institution	* zu 1	Min: 0, Max: Anzahl der zugriffsberechtigten Institutionen	Institution die dieses Zeugnis ausgestelt hat

certificates_owner ⟨*E*40⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Zeugnis	1 zu *	Min: 0, Max: Gesamtanzahl ausgestellter Zeugnisse	Dem Zeugnis wird der Inhaber zugeordnet

certificates_institution ⟨*E*50⟩

Beziehung	Kardinalität	Erwartete Datenmenge	Beschreibung
Zeugnis	1 zu *	Min: 0, Max: Gesamtanzahl ausgestellter Zeugnisse	Dem Zeugnis wird das ausstellende Institut zugeordnet

7 Konfiguration

Nachfolgend werden die notwendigen Einstellungen an der Produktumgebung beschrieben.

7.1 Voraussetzungen

Der nachfolgende Abschnitt beschreibt die Voraussetzungen zur Verwendung des Produkts in einer Produktionsumgebung.

Um die Android-App zu benutzen, wird ein Android 5 fähiges Endgerät mit einer Kamera und einer Internetverbindung vorausgesetzt. Außerdem benötigt die App eine Konfigurations-Datei für TessTwo und die Zertifikats-Datei des Servers. Durch die Konfigurations-Datei wird sichergestellt, dass die OCR-Erkennung richtig funktioniert, während durch das Zertifikat gewährleistet wird, dass die App nur Antworten vom richtigen Server akzeptiert und somit die Verbindung sicher ist.

Für die Nutzung der Serversoftware muss in der Produktionsumgebung Python 3.4, Django zumindest in der Version 1.7.7, ein Apache Webserver zumindest in der Version 2.4 und eine zum Web-Server und mit der Python-Version kompatible Version von modwsgi installiert sein. Um die PDF-Export Funktion zu nutzen, muss eine aktuelle LaTeX-Umgebung installiert sein. Hier wird die Nutzung von `"Texlive"` empfohlen. Desweiteren werden folgende Python-Pakete benötigt: django-grappelli, djangorestframework, latex, six, Jinja2, django-python3-ldap Optional kann zum Erstellen der Dokumentation aus den Quelldateien, Sphinx als Python-Paket installiert und verwendet werden. Die Verwendung dieses Werkzeugs beschreibt ein anderes Dokument. Siehe dazu die angefügten URLs. Die Anwendung muss als Dateisystemverzeichnis in `"/var/www"` abgelegt sein, oder dorthin verlinkt sein. Unter anderem auch, um die Datenbank schreiben und lesen zu können, muss zumindest die Datenbank durch den Nutzer, unter dem der Webserverprozess läuft schreib und lesbar sein. Die übrigen Dateien und Verzeichnisse müssen durch diesen Nutzer zumindest lesbar und ausführbar sein. Auf dem Webserver muss SSL aktiviert sein. Dies kann, falls nicht bereits geschehen, mit dem Befehl `"a2enmod ssl"` geschehen. In der Konfiguration des Webservers muss ein neuer virtueller Host hinzugefügt werden. Siehe `"certcheck.conf"` `"apache.conf"` muss `"WSGIPythonHashSeed random"` enthalten. Diese Eintragungen sind gegebenenfalls zu verändern. Anstatt des `*` kann die IP-Adresse oder der Domänenname stehen unter dem die Anwendung erreichbar sein soll. `"example.com"` muss jeweils

durch den Namen des Servers ersetzt werden. "SSLCertificateFile" und "SSLCertificate-KeyFile" müssen auf die Pfade zum Server-Zertifikat und dem privaten Schlüssel des Servers angepasst werden. Um die Serversoftware einsetzen zu können, muss in "/etc/django_secret¬_key.txt" ein geheimer Schlüssel erzeugt werden. Hierzu sollte ein guter Algorithmus genutzt werden und der Schlüssel muss 50 Zeichen lang sein. Der DEBUG-Modus darf nicht aktiviert sein. Hierzu muss sichergestellt werden, dass in "certcheck/certcheck/settings.py" "DEBUG = False" ist. In "certcheck/certcheck/settings.py" muss in "ALLOWED_HOSTS" die Hostnamen unter denen die Serversoftware erreichbar sein soll, gesetzt werden. Die Datenbank muss regelmäßig gesichert werden, um Datenverlust vorzubeugen. Hierzu ist eine automatische Sicherung einzurichten. "CSRF_COOKIE_SECURE" und "SESSION_COOKIE_SECURE" müssen beide in "chertcheck/certcheck/settings.py" auf "True" gesetzt werden. In "certcheck/certcheck/settings.py" können in "ADMINS" die EMail-Adressen derer eingetragen werden, die über mögliche gefälschte Zeugnisse informiert werden sollen. In "certcheck/certcheck/settings.py" kann in "LOGFILE" der Ort für die Logdatei eingetragen werden. Die Einstellungen von django-python3-ldap müssen angepasst werden, die Details dazu werden in einem anderen Dokument beschrieben. Weiterführende Informationen können den Dokumentationen der verwendeten Softwareprodukte entnommen werden.

https://docs.djangoproject.com/en/1.7/

https://code.google.com/p/modwsgi/wiki/WhereToGetHelp?tm=6

https://httpd.apache.org/docs/current/index.html

https://github.com/etianen/django-python3-ldap

https://grappelliproject.com/

https://sphinx-doc.org/

https://www.tug.org/texlive/

https://code.google.com/p/modwsgi/

7.2 Installation der Voraussetzungen

Im folgenden werden die Softwarevoraussetzungen der Übersicht wegen kurz zusammengefasst.

Benötigt werden folgende Abhängigkeiten:

```
# apt-get install apache2 python3-pip3 texlive-full
  libapache2-mod-wsgi-py3

# pip3 install django django-grappelli djangorestframework latex
  six Jinja2 django-python3-ldap
```

Optionale Abhängigkeiten für die Dokumentation:

```
# pip3 install sphinx
```

8 Änderungen gegenüber Fachentwurf

Hier werden Änderungen am Server und der Android-Applikation bezüglich des Fachentwurfs beschrieben.

8.1 Allgemeine Änderungen

Hier werden kurz allgemeine Änderungen bezüglich des Fachentwurfs beschrieben.

In der Einleitung wurde ein Aktivitätsdiagramm für das gesamte Softwaresystem als Abbildung 1.1 eingefügt, um den funktionalen Zusammenhang der Subsysteme zu klären. Alle aus dem Fachentwurf übernommenen Abbildungen wurden so verändert, dass sie im Druck gut lesbar sind.

8.2 Änderungen an der App

Hier werden Änderungen an der Android-Anwendung beschrieben.

8.2.1 Komponenten

Grundsätzlich ist bei der App die Funktion des Imageprocessings hinzugekommen, die das Bild bearbeitet bevor die Texterkennung gestartet wird. Da die Funktion aber eine Art Unterfunktion der Texterkennung ist, wird sie nicht in einem eigenen Sequenzdiagramm behandelt, sondern ist in Abbildung 2.3 enthalten. In dieser Abbildung wurden außerdem Tasks eingefügt, die in Android für das Multithreading benutzt werden können. Neben dieser Funktion sind auch die Klassen Word und Line hinzugekommen. Die Klasse Line ist eine innere Klasse in der Certificate Klasse und dient lediglich dazu den von TessTwo zurückgegebenen Output besser parsen zu können. Dies findet in Abbildung 2.4 in dem Funktionsaufruf parse() statt. Die Klasse Word wird dazu verwendet nicht nur die von TessTwo erkannten Wörter zu speichern, sondern auch die Wahrscheinlichkeit mit der sie richtig sind und an welcher Position sie stehen. Damit sind nun alle erkannten Wörter nicht mehr vom Typ String sondern vom Typ Word. Im Sequenzdiagramm wird diese Klasse allerdings nicht weiter erwähnt, da sie lediglich als Datentyp gedacht ist.

Ansonsten wurden nur kleinere Änderungen, wie zum Beispiel das Einfügen von get()-Methoden Aufrufen in Abbildung 2.4 vorgenommen.

8.2.2 Konfiguration

Hier werden jetzt zwei weitere Dateien benötigt, zum einen eine Konfigurationsdatei für TessTwo und zum anderen das Zertifikat des Servers. Beide Dateien müssen allerdings nicht manuell vom Anwender hinzugefügt werden, sondern werden mitgeliefert und automatisch installiert. Lediglich wenn ein anderer Server für die Überprüfung genutzt werden soll, muss das Zertifikat angepasst werden.
Die Konfigurationsdatei für TessTwo, auch "training data" genannt sorgt zum Beispiel dafür, dass TessTwo die richtige Sprache unterstützt. Um TessTwo zu benutzt, wird zwangsläufig eine solche Datei benötigt.
Das Serverzertifikat gewährleistet, dass die App nur mit dem von uns bereitgestellten Validierungsserver kommuniziert und nicht durch "Man-In-The-Middle-Angriffe" oder Ähnliches angegriffen werden kann.

8.2.3 Datenmodell

Da, wie schon im Fachentwurf beschrieben, für die App keine Daten dauerhaft gespeichert werden, hat sich an dem Datenmodell nichts geändert.

8.3 Änderungen am Server

Hier werden Änderungen am Server beschrieben.

8.3.1 Änderungen an der Konfiguration

Wie ursprünglich beabsichtigt wurde die Konfiguration in ein eigenes Kapitel eingefügt. Die Abhängigkeiten und die notwendigen Einstellungen wurden ergänzt. Unter anderem sind hier Grappelli, Sphinx, Texlive, latex und Jinja2 hinzugekommen. Es wurde ein eigenes Unterkapitel eingefügt, um die Schritte zur Installation der Voraussetzungen übersichtlich darzustellen. Auf die Abbildung der Konfigurationsdateien im Text wurde verzichtet, da dies die Lesbarkeit erhöht.

8.3.2 Änderungen am Datenmodell

Der Primärschlüssel für die `certificates_subject` hat sich geändert. Zuvor war dieser die SecureID des Zeugnisses. Nun ist es eine einzigartige Ganzzahl. Diese Änderung war notwendig, da bei der Objekterzeugung mit Django die Funktion zur Erzeugung der SecureID erst nach dem Anlegen des Objektes aufgerufen wurde und dadurch kein Primärschlüssel zum Erzeugungszeitpunkt verfügbar war, was zu einem Fehler führte.

9 Erfüllung der Kriterien

Hier wird erläutert, welche Muss-, Soll-, und Kannkriterien die App beziehungsweise der Server erfüllt. Im speziellen werden die Komponenten angegeben, die die entsprechenden Funktionen implementieren.

9.1 Musskriterien

Die folgenden Kriterien sind unabdingbar und müssen durch das Produkt erfüllt werden:

App:
RM1 Die Möglichkeit ein Bild aufzunehmen wird mittels der camera2-Api von Android in der Komponente $\langle C10 \rangle$ Image implementiert.
RM2 Das Durchführen der "Optical Character Recognition" findet in der Komponente $\langle C20 \rangle$ ImageProcessing statt, wobei neben dem reinen OCR-Scan auch noch das Bild selber bearbeitet wird um ein genaueres Ergebnis zu erhalten.
RM3 Der Hash-Wert, der später an den Server übermittelt wird, wird in der Komponente $\langle C40 \rangle$ Communication berechnet.
RM4 Genau wie die Hash-Wert Berechnung findet auch die Übermittlung der Daten in der $\langle C40 \rangle$ Communication Komponente statt. Ein besonderes Augenmerk wurde dabei darauf gelegt, dass die Kommunikation über SSL stattfindet.
RM5 Die $\langle C30 \rangle$ Verification Komponente setzt das in RM5 geforderte Anzeigen des Ergebnisses der Überprüfung um. Hierbei wurde darauf geachtet, dass, anders als bei der Projektplanung vorgesehen, nur die Ergebnisse "Valides Zeugnisöder "nicht valides Zeugnis"möglich sind. Das sorgt dafür, dass ein Zeugnisfälscher keine weiteren Anhaltspunkte erhält, woran die Überprüfung gescheitert ist.

Server:
RM6 Zeugnisse können mit der Web-Anwendung einfach erstellt werden. Implementiert wird dies durch $\langle C70 \rangle$ Views. Hierzu wurde in Django Grappelli als verbessertes Admin-Interface eingebunden und entsprechend erweitert.
RM7 Zeugnisse können aus dem Admin-Interface vom Nutzer angefordert werden, implementiert wird dies durch $\langle C70 \rangle$ Views. Vom Server werden die Zeugnisse als PDF erzeugt und dem

Nutzer zum Download angeboten werden. Als gut erprobtes Verfahren zur Erstellung von PDFs wird hierbei LaTeX eingesetzt. Dies geschieht durch die Komponente $\langle C110 \rangle$ Texlive.
RM8 Über das Admin-Interface, realisiert durch $\langle C70 \rangle$ Views, können erstellte Zeugnisse eingesehen, gelöscht und auch geändert werden. Hierbei wurde darauf geachtet, dass der Hash-Wert sich bei geänderten Zeugnisdaten unterscheiden muss. Die Datenbank selber wurde durch $\langle C100 \rangle$ Database realisiert und der Zugriff wird über $\langle C90 \rangle$ Models geregelt.
RM9 Die SecureID-Hash Kombination wird durch $\langle C80 \rangle$ Verification vorgenommen und nutzt Djangos Datenzugriffsmethoden, die in $\langle C90 \rangle$ Models implementiert wurden. Dementsprechend wird ein angebrachtes Caching der Zeugnisdaten vorgenommen und das Laufzeitverhalten der Anfragen optimiert.
RM10 Das von $\langle C80 \rangle$ Verification ermittelte Ergebnis wird an die App als JSON zurück übermittelt. Hierbei wurde darauf geachtet, dass der App keine zusätzlichen Informationen zukommen, die nicht zur Überprüfung des Zeugnisses nötig sind, aber beim Fälschen von Zeugnissen hilfreich wären. Wie es zum Beispiel die Information wäre, ob die SecureID vorhanden ist, aber der Hash-Wert nicht passt.

9.2 Sollkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

App:
RS1 Die Unterstützung verschiedener Android-Geräte wird grundsätzlich dadurch erreicht, dass die gesamten GUI-Elemente der $\langle C10 \rangle$ Image-, $\langle C20 \rangle$ ImageProcessing- und $\langle C30 \rangle$ Verfication-Komponenten für verschiedene Bildschirmformate und Auflösungen angepasst sind. Hier ist allerdings die Einschränkung zu machen, dass nur Geräte mit einer ausreichend guten Kamera unterstützt werden.
RS2 Die Android Design Guidlines wurden soweit wie möglich in den GUI-Elementen der $\langle C10 \rangle$ Image, $\langle C20 \rangle$ ImageProcessing und $\langle C30 \rangle$ Verfication Komponente umgesetzt. Wobei hier besonders darauf geachtet wurde, dass die Bedienung der App möglichst einfach ist und der Prozess der Überprüfung möglichst simpel und schnell für den Nutzer gehalten ist.
RS3 Ursprünglich bezog sich die detailierte Ausgabe von Fehlermeldungen darauf, dass bei einer fehlgeschlagenen Überprüfung genau angezeigt wird woran die Überprüfung gescheitert ist. Wie schon bei RM5 erklärt wurde diese Funktion aus sicherheitstechnischen Gründen nicht implementiert. Bei App-internen Fehlern wurde allerdings in allen Komponenten auf verständlich Fehlermeldungen geachtet.

Server:

RS4 Der Nutzer der Web-Anwendung muss sich auf der Admin-Seite authentifizieren, um Zugriff auf die Zeugnisdaten zu haben. Realisiert wird dies durch $\langle C60 \rangle$ Authentication und $\langle C70 \rangle$ Views. Um Zugang zur Schnittstelle zu haben, muss für den Nutzer entweder ein Authentifizierungstoken reserviert sein, oder er muss eine aktive Session haben.
RS5 Zur Änderung der Zeugnisdaten steht in $\langle C120 \rangle$ Rest-API eine RESTful API bereit. Hierzu wurde das Django REST framework genutzt.

9.3 Kannkriterien

Die Erfüllung folgender Kriterien für das abzugebende Produkt wird angestrebt:

App:
RC1 Die Auswahl von verschiedenen Verifikationsservern lässt sich, um Missbrauch zum Beispiel durch das Eintragen eines falschen Servers zu verhindern, nicht vom Benutzer selber durchführen. Allerdings wird durch die $\langle C50 \rangle$ Data Komponente eine Settings Klasse implementiert, in der mit minimalem Aufwand die App für andere Serveradressen angepasst werden kann.
RC2 Die Überprüfung von Zeugnissen aus dem Fotoalbum wird durch die $\langle C10 \rangle$ Image Komponente implementiert.
RC3 Die Review der eingelesenen Daten hat während der Entwicklungsphase immer mehr an Bedeutung gewonnen und wird durch die $\langle C30 \rangle$ Verification Komponente umgesetzt. Die Priorität dieses Kriteriums ist so stark gestiegen, da die Texterkennung sehr von den Lichtverhältnissen abhängig ist und sich diese auch nur bedingt durch Imageprocessing verbessern lassen. Durch die Überprüfung der Daten durch den Nutzer können wir trotzdem eine korrekt funktionierende Überprüfung des Zeugnisses umsetzen.
RC4 Das Auswählen von verschiedenen Sprachen wird nicht umgesetzt, da generell nur sehr wenige sprachliche Elemente in der App vorhanden sind, die außerdem in einfachem Englisch gehalten sind.

Server:
RC5 Fehlgeschlagene Überprüfungsversuche werden durch $\langle C90 \rangle$ Views in einer Log-Datei protokolliert und eingetragene Administratoren werden über diese Vorfälle per EMail informiert.
RC6 Dieses Kriterium wurde bisher nicht implementiert, da es zur Erfüllung der Funktion des Softwaresystems nicht notwendig ist.
RC7 Zeugnisse können über das Admin-Interface ausgewählt werden und erneut als PDF generiert werden. Diese Funktionen sind in $\langle C90 \rangle$ Views implementiert und nutzen $\langle C110 \rangle$ Texlive. Da Zeugnisse nicht dauerhaft als PDF auf dem Server gespeichert bleiben, muss die Generierung für jede solche Anfrage erneut geschehen.

RC8 Diese Funktion wurde bislang nicht implementiert, da die Softwarebibliothek, die zur Bereitstellung dieser Funkion verwendet werden sollte, nicht wie erwartet funktioniert.

10 Glossar

Hier werden Fachbegriffe erklärt.

Bezeichnung	Beschreibung
Action Bar:	Die Action Bar befindet sich in einer Android Applikation am oberen Bildschirmrand. Sie zeigt dem Nutzer seine Lage in der Applikation und bietet verschiedene Aktionen.
OCR:	Optical Character Recognition (Zeichenerkennung) - erkennen von nicht digitalen Zeichen und deren Digitalisierung
TessBaseApi:	Ist eine Java Klasse der Schrifterkennungsbibliothek Tess Two.
Tess Two:	Android optimierter Fork der Open Source Bibliothek Tesseract.
Trainingdaten:	Datenpaket um die OCR-Bibliothek Tess Two auf die Erkennung einer Sprache zu trainieren.
Bitmap:	Rasterfeld im Speicher, das ein Bild aus einzeln ansteuerbaren Pixeln in Vertikal-/Horizontal- Koordinaten aufbaut.
URL:	Uniform Resource Locator - identifiziert und lokalisiert eine Ressource in einem Netzwerk
PDF-Datei:	Portable Document Format - ein Dateiformat welches für die plattformunabhängige Anzeige von Dokumenten benutzt wird
HASH:	Streuwertfunktion, Abbildung beliebiger Daten auf einen Wertebereich fester Größe.
HTTP:	Hypertext Transfer Protocol
SSL:	Secure Sockets Layer
SecureID:	Eine für ein Zeugnis eindeutig vergebene Identifikationsnummer, die gleichzeitig aber noch keine Aussage über den Inhalt des Zeugnis tätigt.

Aktivität:	Repräsentiert die Präsentationsebene einer Android Applikation, z.B. einen Programmbildschirm den der Benutzter sieht. Eine Android Applikation kann aus mehreren Aktivitäten bestehen und sie kann während der Laufzeit zwischen mehreren Aktivitäten wechseln.
parsen:	Maschinenlesbare Daten analysieren, segmentieren und codieren.
GUI:	Graphical User Interface.
WebInterface bzw.Webschnittstelle:	als Webschnittstelle bezeichnet man eine Schnittstelle zu einem System, die über HTTP angesprochen werden kann. Dabei handelt es sich um • eine grafische Benutzeroberfläche (GUI), über die ein Benutzer mit Hilfe eines Webbrowsers mit dem System interagieren kann, oder • einen Webservice, durch den das System anderen Systemen Daten oder Funktionen zur Verfügung stellt.
serverIP:	ist eine Adresse in Computernetzen, die – wie das Internet – auf dem Internetprotokoll (IP) basiert. Sie wird Geräten zugewiesen, die an das Netz angebunden sind, und macht die Geräte so adressierbar und damit erreichbar.
camera2:	Die camera2 Api wurde mit Android 5 eingeführt und ersetzt die nun als deprecated gekennzeichnete camera Api. Sie dient dazu Bilder aufzunehmen.
Man-in-the-Middle-Angriff:	Grob gesagt steht dabei ein Angreifer zwischen zwei Kommunikations-Partnern und gibt beiden vor der jeweils andere zu sein. Dadurch erlangt er volle Kontrolle über alle ausgetauschten Daten.
Multithreading:	Multithreading bedeutet das verschiedene Aufgaben zeitgleich ausgeführt werden.
TCP/IP:	(Transmission Control Protocol/Internet Protocol) ist die grundlegende Kommunikationssprache (Protokoll) des Internets. Es kann außerdem als Kommunikationsprotokoll innerhalb eines privaten Netzwerks (Intranet oder Extranet) verwendet werden.
JSON:	JavaScript Object Notation.
LDAP:	Lightweight Directory Access Protocol.
Django:	ist ein in Python geschriebenes quelloffenes Web Application Framework, das einem Model-View-Controller-Schema folgt. Benannt ist es nach dem Jazz-Gitarristen Django Reinhardt.