

Studnetz, eine Applikation für Mobilgeräte zur Vermittlung von Nachhilfeunterricht

Florian Hirtz

11. Oktober 2018

Inhaltsverzeichnis

Abstract	4
Vorwort	6
1 Einleitung	7
1.1 Zielsetzung	7
2 Konzeptionelle Grundlagen	8
2.1 Client-Server Prinzip	8
2.2 Das Modell-View-Presenter Konzept (Passive View)	9
3 Die entwickelte Applikation	10
3.1 Features	10
3.2 Benutzeroberfläche und Benutzung	11
4 Die Server	19
4.1 Webserver	19
4.1.1 Apache Server	21
4.1.2 PHP	21
4.1.3 JSON	21
4.2 Datenbanken	21
4.2.1 Die MySQL Datenbank	23
4.2.2 Firebase	24
4.2.2.1 Die Firebase Echtzeitdatenbank	24
4.3 Sicherheit der Passwörter in einer Datenbank	25
4.3.1 Hashen	25
4.3.2 Salzen	26
4.3.3 bcrypt	27
4.4 Implementation in der entwickelten Applikation	27
4.4.1 Die verwendeten Server	27

Inhaltsverzeichnis

4.4.2	Datenbankstrukturen	28
4.4.2.1	MySQL Datenbankstruktur	28
4.4.2.2	Firebase Datenbankstruktur	30
4.4.3	PHP Skripte	33
4.4.3.1	Login	33
4.4.3.2	Registrierung	35
4.4.3.3	Änderung der Informationen via Einstellungen des Clients	36
4.4.3.4	Suchanfrage	37
4.4.3.5	Profilbilder	38
5	Der Client	40
5.1	Betriebssystem und Programmiersprache	40
5.1.1	Android	40
5.1.2	Java	41
5.1.3	XML	41
5.1.4	Android Studio	42
5.2	Klassenübersicht des entwickelten Clients	42
5.2.1	Activities	43
5.2.2	Fragments	44
5.2.3	Models	46
5.2.3.1	Das UserModel	47
5.2.3.2	Das ProfilePictureModel	48
5.2.3.3	Das ChatModel	50
5.2.4	Auflistungen mithilfe von Recyclers	51
5.2.4.1	Das XML-Layout eines Elementes	51
5.2.4.2	Das Model eines Elements	52
5.2.4.3	Die ViewHolder-Klassen	52
5.2.4.4	Die Adapter-Klassen	53
5.2.5	Utility-Klassen	56
5.2.5.1	Die JSONToInfo-Klasse	56
5.2.5.2	Die TempFileGenerator-Klasse	56
5.2.5.3	Die ProfilePictureLoader-Klasse	57
5.2.5.4	Die MyReader-Klasse und die SchoolMapper-Klasse	57
5.2.6	Profilbilder und <i>Image Cropping</i>	59
5.2.7	Interaktion mit dem Webserver (MySQL Datenbank) . .	60
5.2.7.1	Formulieren einer Anfrage	60
5.2.7.2	Das Verarbeitung einer Antwort	63
5.2.8	Interaktion mit der Firebase Echtzeitdatenbank . . .	64

Inhaltsverzeichnis

5.2.8.1	Verbindung	64
5.2.8.2	Zugriff vom Client auf die Echtzeitdatenbank	64
6	Zusammenfassung und Diskussion	67
6.1	Herunterladen des Projektes und der Applikation	67
6.2	Verbesserungspotenzial	68
6.3	Fazit	69
A	Entwicklungsprozess	70
A.1	Vorbereitung	70
A.2	Erstellen eines ersten Prototypen (Dezember - Januar)	70
A.3	Ausbauen des Prototypen (Februar - April)	71
A.4	Profilbilder und Passworthashing (Juni - Juli)	72
A.5	Refactoring und Bugfixes (August - Oktober)	73

Abstract

Die Arbeit „Studnetz, eine Applikation für Mobilgeräte zur Vermittlung von Nachhilfeunterricht“ befasst sich mit der Entwicklung einer Applikation für Android-Mobilgeräte. Es wird vertieft auf die Funktion der einzelnen Komponenten der Applikation eingegangen. Es werden mithilfe von konkreten Beispielen aus der Applikation die umgesetzten Lösungsansätze für auftretende Herausforderungen genauer beschrieben um so die genaue Funktionsweise der Applikation ersichtlich zu machen.

Die Applikation „Studnetz“ ist ein Prototyp für eine potentiell marktfähige Applikation für Mobilgeräte mit dem Betriebssystem Android. Das Ziel der Applikation ist die Vermittlung von Nachhilfe unter Schülerinnen und Schülern zu vereinfachen. In der Applikation ist es Schülerinnen und Schülern möglich, Fächer anzugeben, in welchen sie andere unterstützen können. Des Weiteren können sie nach Schülerinnen und Schülern suchen, die ihnen in bestimmten Fächern helfen können. Dabei kann nach Klasse, Schule und Fächer gefiltert werden. Durch einen Chat wird eine Kommunikation zwischen den Schülern und Schülerinnen ermöglicht, wo genauere Daten wie Datum, Zeit und Ort ausgetauscht werden können.

Die Applikation basiert auf einem Client-Server-Konzept. Der Client wurde dabei in der Programmiersprache *Java* und der Auszeichnungssprache *XML* programmiert. Auf der Seite der Server findet sich ein Webserver mit einer installierten *MySQL*-Datenbank und einer Reihe von Skripten in der Programmiersprache *PHP*, welcher für das Speichern der einzelnen Benutzer verantwortlich ist. Des Weiteren wird ein *Firebase*-Server verwendet, welcher eine Chatfunktion in Echtzeit ermöglicht.

Vorwort

Diese Arbeit befasst sich mit der Entwicklung einer Applikation für Mobiltelefone. Die Applikation soll die Vermittlung von Nachhilfeunterricht unter Schülerinnen und Schülern einer Schule vereinfachen. Dabei ist anzumerken, dass die Entwicklung einer vollständig markttauglichen Applikation den Rahmen dieser Arbeit sprengen würde. Das Ziel ist vielmehr das Legen eines Grundsteins, aus welchem ein solches Produkt entstehen könnte.

Zu mir

Ich möchte an dieser Stelle die Gelegenheit ergreifen, ein paar Worte zu mir, dem Autor und Entwickler dieser Arbeit, zu verlieren.

Zum Zeitpunkt, an welchem ich begann die Studnetz-Applikation zu entwickeln, hatte ich zuvor noch nie eine Applikation für Mobiltelefone entwickelt. Bis zu diesem Zeitpunkt habe ich mich hauptsächlich auf die Entwicklung kleinerer Programme für Computer beschränkt. Das dazu nötige Wissen habe ich mir grösstenteils selber angeeignet. So hatte ich mich vor diesem Projekt noch nie mit Datenbanken und Webdevelopment befasst. Ich habe mir dieses Projekt als Herausforderung genommen, um etwas Neues zu lernen und mich als Entwickler weiter zu entwickeln. Dieser Lernprozess hat definitiv stattgefunden. Dies erklärt auch, weshalb teilweise gleiche Probleme an verschiedenen Orten verschieden gelöst wurden oder warum gewisse verwendete Lösungen von anerkannten Praktiken abweichen.

Motivation

Die Motivation für die Entwicklung der Studnetz-Applikation schöpfte ich hauptsächlich aus zwei Quellen. Zum einen war es der Wille, neue Teilgebiete des Programmierens kennenzulernen und zu verstehen, weshalb die Wahl sehr schnell auf eine Applikation für Mobiltelefone fiel. Immerhin sind Mo-

Inhaltsverzeichnis

biltelefone in der heutigen Welt kaum mehr wegzudenken und die Fähigkeit, Programme für sie zu entwickeln hat mich schon lange fasziniert. Die zweite Motivation ist dann aus einer Idee von Herrn Bättig entstanden, der während der einführenden Präsentation in die Maturarbeit eine Applikation zur Vermittlung für Nachhilfe erwähnte. Zuerst bin ich skeptisch gewesen, doch schnell habe ich im Gespräch mit anderen festgestellt, dass eine gut entwickelte Applikation in diesem Bereich durchaus Verwendung und Beliebtheit finden könnte. Dies ist dann der Anstoß gewesen, ein solches Projekt in Angriff zu nehmen.

Danksagungen

Ich möchte mich hier bei den Personen bedanken, die mich während dieser Arbeit unterstützt haben. Zuerst einmal möchte ich meiner betreuenden Lehrperson Herrn Andreas Umbach danken. Er stellte mir nicht nur einen Platz auf der Datenbank des Ergänzungsfaches zur Verfügung, sondern half mir stets bei allfälligen Fragen und Unklarheiten weiter. Zudem möchte ich ein Dankeschön an meinen Onkel Peter Arrenbrecht richten, der mir mit seinem Hinweis auf die Firebase API komplett neue Möglichkeiten eröffnet hat. Als letztes gilt noch ein gigantisches Dankeschön an meinen Vater, der mich während dieser ganzen Arbeit als Ratgeber und Korrekturleser immer unterstützte und immer ein offenes Ohr für meine Probleme zeigte.

Kapitel 1

Einleitung

1.1 Zielsetzung

Das Ziel dieser Arbeit ist das Entwickeln einer Applikation für Android Geräte, die die Vermittlung von Nachhilfeunterricht unter Schülerinnen und Schülern vereinfachen soll. Die Benutzerinnen und Benutzer der Applikation sollen in der Lage sein, sich einen Account innerhalb der Applikation zu erstellen und sich darin einzuloggen. Sie sollen angeben können, in welchen Fächern sie in der Lage sind, anderen Nachhilfe zu geben. Weiter soll eine Funktion vorhanden sein, mit welcher nach anderen Benutzern gesucht werden kann, die in den gewünschten Fächern Nachhilfe anbieten. So sollen Benutzerinnen und Benutzer bei Bedarf gezielt nach Nachhilfelehrern/Nachhilfelehrerinnen suchen können, die ihren Bedürfnissen entsprechen. Weiter soll es ihnen möglich sein, über eine Chatfunktion andere Benutzer zu kontaktieren, wo genauere Informationen wie Zeit, Ort und eventuell Preis ausgetauscht werden können.

Kapitel 2

Konzeptionelle Grundlagen

Im folgenden Kapitel werden die für diese Arbeit wichtigen Grundkonzepte etwas genauer beschrieben.

2.1 Client-Server Prinzip

Das *Client-Server Prinzip* ist ein weit verbreitetes Konzept, um die Aufgaben innerhalb eines Netzwerkes effizient aufzuteilen. Dabei werden die Aufgaben auf zwei im Netzwerk agierende Programme aufgeteilt. Diese Programme werden im Allgemeinen als Client und Server bezeichnet.

Der *Server* hat die Aufgabe, verschiedene Dienste zur Verfügung zu stellen, welche auf Anfrage ausgeführt werden können. Ein solcher Dienst kann zum Beispiel das Versenden einer Nachricht oder das Aufrufen einer Webseite sein. Der Server selbst ist meist passiv, was bedeutet, dass der Server einzig auf Anfragen reagiert und nicht selber Anfragen stellt. Ein Server sollte immer in der Lage sein, Anfragen entgegenzunehmen und zu verarbeiten.

Der *Client* selber ist die aktive Komponente des Systems und ist meist die Komponente, mit welcher ein Benutzer/eine Benutzerin direkt interagiert. Der Client ist in der Lage, Anfragen an den Server zu stellen und von dessen Diensten Gebrauch zu machen. Grundsätzlich gibt es in einem solchen Netzwerk nur einen Server, jedoch kann es durchaus mehrere Clients geben. Ein guter Server sollte also auch darauf vorbereitet sein, mehrere Anfragen von verschiedenen Clients parallel zu bearbeiten. [12]

2.2 Das Modell-View-Presenter Konzept (Passive View)

Das *Modell-View-Presenter* (MVP) Konzept wie auch das sehr ähnliche *Modell-View-Controller* (MVC) Konzept, sind beide für das Entwickeln von Software entworfen worden. Ihre Idee ist es, die Aufgabenbereiche innerhalb einer Applikation strikt voneinander zu trennen. Dabei wird zwischen drei Typen von Aufgabenbereichen unterschieden:

- Die *View* ist die sichtbare Benutzeroberfläche. Sie hat die Aufgabe, dem Benutzer ein bedienbares Interface zu bieten und soll auf Anfrage den Status seiner einzelnen Komponenten weitergeben. Sie kennt das Model nicht.
- Das *Model* ist der Datenspeicher einer Applikation, der gebraucht wird um die View korrekt darzustellen. Es soll auf Anfrage hin Daten ausgeben können. In diesem Falle kennt das Model weder die View noch den Presenter. Je nach Auslegung des Konzepts hat das Model jedoch auch die Aufgabe, falls sich Datensätze ändern, den Presenter davon zu unterrichten. Das Model kennt in diesem Falle zwar das Model, jedoch nicht die View.
- Der *Presenter* oder *Controller* ist sozusagen die Mittelperson der beiden anderen Komponenten. Er ist in der Lage Daten aus dem Model anzufordern und kontrolliert anschliessend was mit diesen Daten geschieht. Er hat ebenfalls die Möglichkeit, die angezeigte View zu ändern und deren Status abzufragen. Der Presenter ist in der Lage, sowohl die View als auch das Model zu manipulieren.

Wichtig bei dem MVP Konzept mit einem passiven View ist es, dass nur der Presenter die Möglichkeit hat, auf die beiden anderen Komponenten zuzugreifen. Der View und das Model sollen unter keinen Umständen direkt miteinander kommunizieren. Sämtlicher benötigter Informationsaustausch soll stets vom Presenter kontrolliert werden. [31] Ein grosser Vorteil einer nach diesen Regeln entwickelter Applikation ist es, dass die einzelnen Komponenten weitgehend unabhängig von einander sind. Somit kann zum Beispiel der View komplett neu gestaltet werden, ohne dass der Presenter oder das Model geändert werden müssen, damit die Applikation weiterhin funktioniert.

Kapitel 3

Die entwickelte Applikation

Die im Rahmen dieser Arbeit entwickelte Applikation ist für Mobiltelefone mit einer Version des Betriebssystems Android mit einem API Level von 17 und höher entwickelt worden [3]. Sie kann darauf installiert und anschliessend ausgeführt werden. Für die Verwendung der Applikation ist eine Verbindung zum Internet notwendig.

3.1 Features

Damit die Applikation auch ihren Zweck erfüllen kann, besitzt sie eine Reihe von Features von welchen Benutzer/Benutzerinnen Gebrauch machen können.

- Das *Account-Feature*: Die Applikation bietet den Benutzern/ Benutzerinnen die Möglichkeit, sich einen persönlichen Account zu erstellen und ihn zu personalisieren. Es ist ihnen möglich, auf ihrem Profil ihren Namen, ihre besuchte Schule und ihr Geburtsjahr anzugeben. Weiter können sie auch eine kurze Beschreibung von sich verfassen und sie haben die Möglichkeit, falls sie selber Nachhilfe anbieten wollen, Fächer auszuwählen, in welchen sie das tun möchten. Das Profil ist für andere Benutzer/Benutzerinnen einsehbar. Die meisten Accountdetails können innerhalb der Applikation auch nach der Registrierung bearbeitet werden.
- Das *Such-Feature*: Es ist registrierten Benutzern/Benutzerinnen möglich, über eine Suchfunktion nach anderen registrierten Benutzern/Benutzerinnen zu suchen. Dabei kann nach dem Namen und nach

ausgewählten Fächern gesucht werden. Die gefundenen Profile der verschiedenen Benutzern/ Benutzerinnen können anschliessend angesehen werden.

- Das *Chat-Feature*: Sollte der Benutzer/die Benutzerin einen anderen Benutzer oder Benutzerin gefunden haben, mit welchem/welcher er/sie Kontakt aufnehmen möchte, kann ein neuer Chat via das gefundene Profil geöffnet werden. Darin können Benutzer/ Benutzerinnen in Echtzeit kurze Textnachrichten miteinander austauschen. Sollte ein Chat geöffnet sein, kann sowohl der Sender/die Senderin wie auch der Empfänger/die Empfängerin einfach von ihrem eigenen Profil auf ihn zugreifen.

3.2 Benutzeroberfläche und Benutzung

In diesem Abschnitt soll auf die Applikation aus der Sicht eines Benutzers/einer Benutzerin eingegangen werden. Dazu gehört zum einen die Benutzeroberfläche der einzelnen Features wie auch deren Benutzung. Eine Übersicht über die gesamte Applikation findet sich in Abbildung 3.1.

Login und Registrierung

Beim Ausführen der Applikation auf dem Mobilgerät wird der Benutzer/die Benutzerin vom Login-Screen der Applikation begrüßt (siehe Abbildung 3.2). Der Benutzer/die Benutzerin kann sich nun hier, sollte er/sie bereits einen Account besitzen, mit seinem/ihrem Benutzernamen und Passwort einloggen. Sollte die Person noch keinen Account besitzen, kann sie über einen Schriftzug unterhalb der Logindaten zu einem Registrierungsformular gelangen (siehe Abbildung 3.2). Im Registrierungsformular wird nach sämtlichen Daten gefragt, die benötigt werden, um einen Account zu erstellen. Dazu gehört ein Benutzername, Vor- und Nachname, eine E-Mail Adresse, ein Passwort, die momentan besuchte Schule sowie die aktuelle Klassenstufe der besuchten Schule. Mit Klassenstufe ist dabei die Klassenstufe gemeint, auf welcher man sich an der momentanen Schule befindet. Wenn ein Benutzer/eine Benutzerin beispielsweise das 3. Jahr an der KSA besuchen würde, befände sie sich in der 3. Klasse. Für Schülerinnen und Schüler der gleichen Schule sollte somit einfacher zu erkennen sein, auf welcher Stufe sich ein anderer registrierter Schüler/eine andere registrierte Schülerin befindet. Da nicht alle Schulen die gleiche Anzahl an Stufen besitzen, passen sich die auswählbaren Stufen innerhalb der Applikation jeweils an die ausgewählte

Kapitel 3. Die entwickelte Applikation

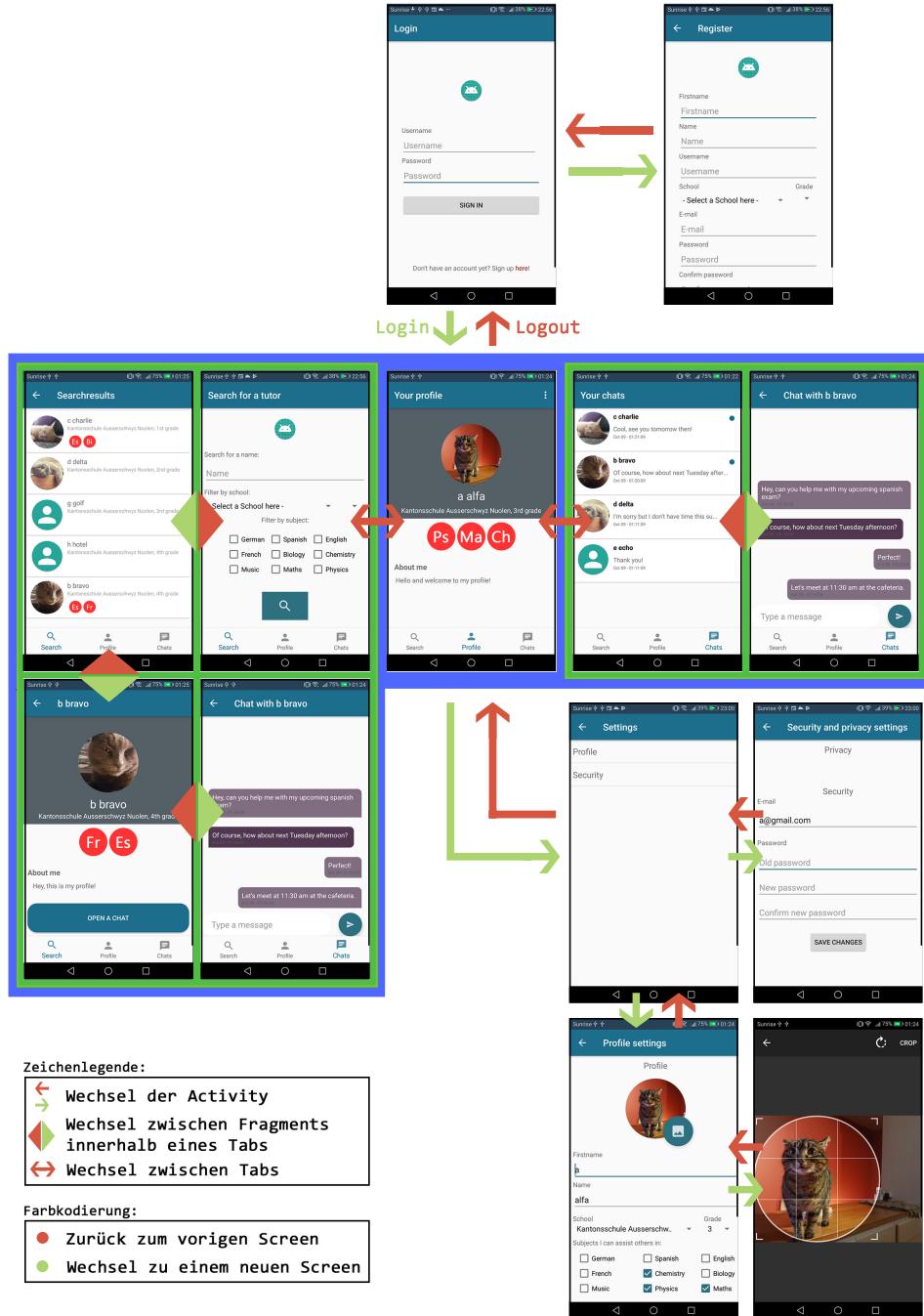


Abbildung 3.1: Übersicht aller Screens der Studnetz-Applikation

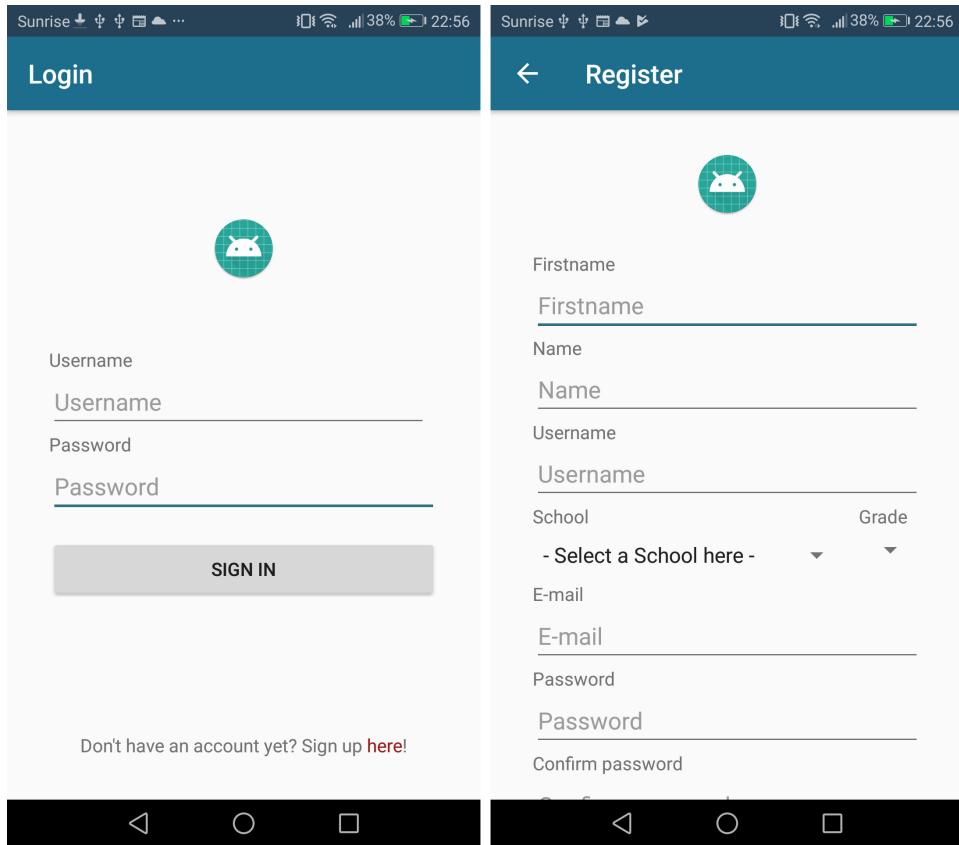


Abbildung 3.2: Links: Loginformular, Rechts: Registrierungsformular

Schule an. So besitzt die KSA beispielsweise vier Stufen, während das Gymnasium Einsiedeln sechs besitzt. Ist die Registrierung erfolgreich, kann sich der Benutzer/die Benutzerin ab sofort in seinen/ihren neuen Account einloggen.

Navigation

Die Navigation innerhalb der Applikation nach einem erfolgreichen Login erfolgt hauptsächlich über eine sogenannte *Bottom Navigation*. Dort können jederzeit bei der Benutzung der Applikation zwischen den wichtigsten Screen hin und her gewechselt werden. Die Studnetz-Applikation besitzt drei solche Screens, welche in den folgenden Paragraphen genauer erläutert werden. Weiter ist es meist möglich, über eine Schaltfläche in der oberen, linken

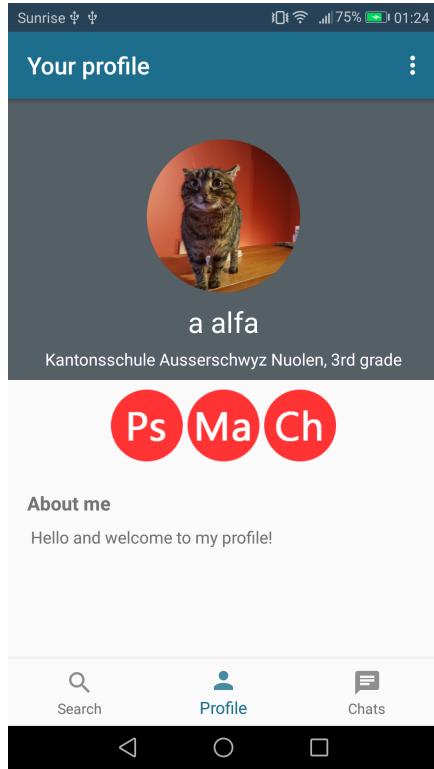


Abbildung 3.3: Hauptseite mit Profil des Benutzers/der Benutzerin

Ecke einen Schritt zurück zu gehen, sollte dies von Bedarf sein.

Benutzerprofil, Einstellungen und Hauptseite

Dieser Screen ist der, welcher nach einem erfolgreichen Login angezeigt wird. Er besteht aus dem Profil des Benutzers/der Benutzerin (siehe Abbildung 3.3). Bei einem neuen Profil findet sich hier ein standardmässiges Profilbild, der volle Name des Benutzers/der Benutzerin, die besuchte Schule sowie die Klassenstufe. Diese und noch weitere Angaben wie angebotene Fächer, eine Beschreibung oder ein persönliches Profilbild können in den Einstellungen, welche über eine Schaltfläche in der oberen rechten Ecke in der Toolbar erreicht werden können, jederzeit konfiguriert werden. Das Profilbild kann dabei entweder direkt mit der Kamera aufgenommen oder aus der Galerie des Mobiltelefons ausgewählt werden. Anschliessend kann der Benutzer/die Benutzerin einen gewünschten Ausschnitt im Bild definieren, auf welchen das Bild dann zugeschnitten wird (siehe Abbildung 3.4). Die ausgewählten

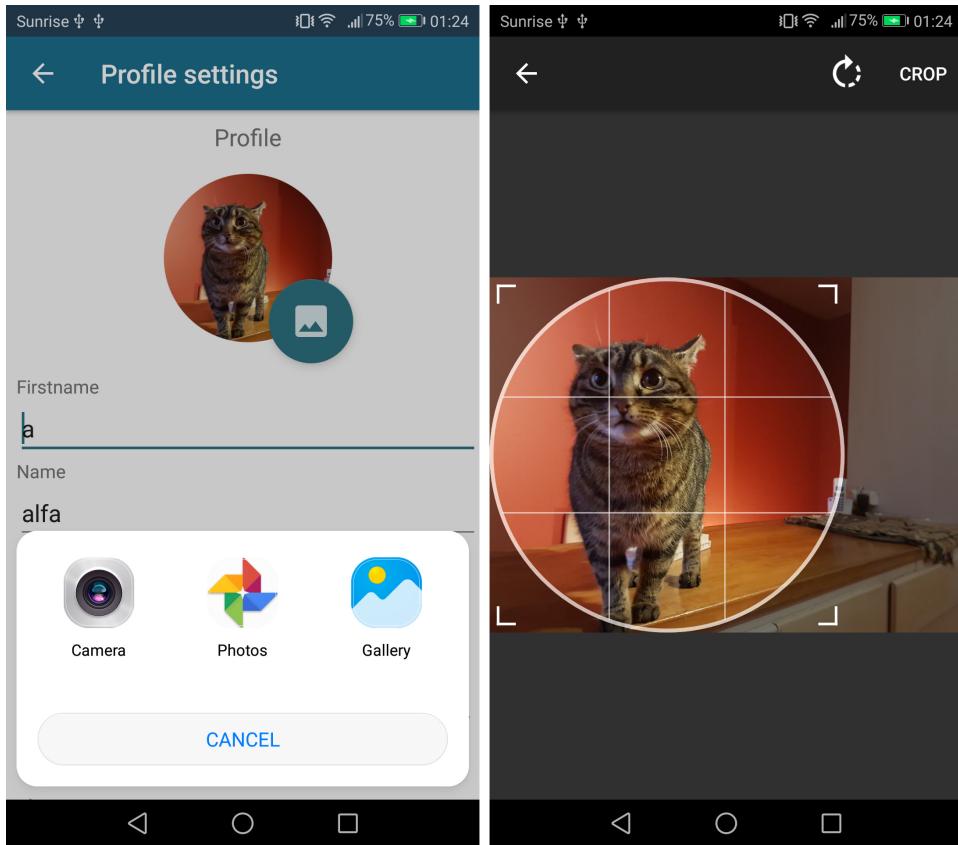


Abbildung 3.4: Links: Auswahl einer Quelle für das Bild, Rechts: Zuschneiden des Bildes

Fächer werden in Form von Emblems auf dem Profil dargestellt.

Suche

Über das Suchfeature der Applikation können Benutzer/Benutzerinnen nach anderen Benutzern/Benutzerinnen suchen, die ihnen bei ihren Problemen helfen können. Die Suche findet sich in Form eines in vier Stufen unterteilten Prozesses. Das Suchfeature kann über die Bottom Navigation erreicht werden. Sollte ein Benutzer/eine Benutzerin mitten in der Suche das Suchfenster verlassen haben, wird der momentane Stand gespeichert und wenn die Suche wieder aufgerufen wird, kann an der Stelle weitergemacht werden, wo zuletzt aufgehört wurde. Die erste Stufe der Suche findet sich dabei in Form

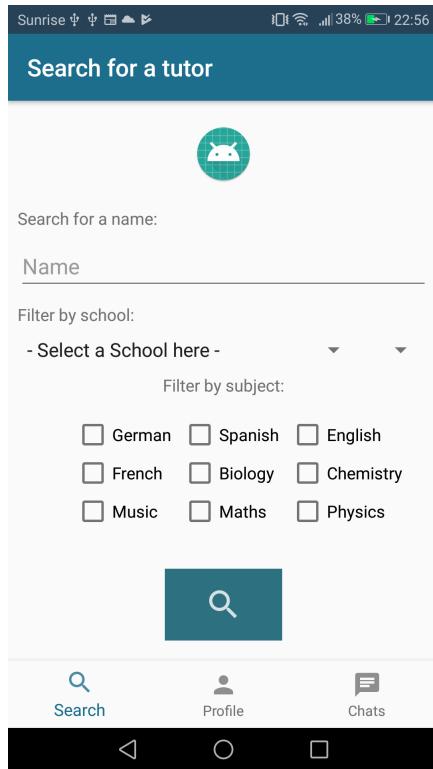


Abbildung 3.5: Filtereinstellungen für die Suche

eines Screens, wo Suchkriterien wie Name, Schule, Stufe und angebotene Fächer definiert werden können, nach welchen gesucht werden soll (siehe Abbildung 3.5). Ist der Benutzer/die Benutzerin zufrieden mit den Konfigurationen der Suche, kann über eine Schaltfläche die Suche ausgeführt werden und der Screen ändert sich zur zweiten Stufe, wo die Suchergebnisse in Form einer Liste dargestellt werden (siehe Abbildung 3.6). Erweckt ein Ergebnis der Suche das Interesse des Benutzers/der Benutzerin, kann über das Auswählen eines Ergebnisses das Profil des gefundenen Benutzers/der gefundenen Benutzerin geöffnet werden, womit man sich auf der dritten Stufe befindet (siehe Abbildung 3.6). Von dort kann ein neuer Chat über eine Schaltfläche eröffnet werden, wo genauere Informationen über die mögliche Zusammenarbeit ausgetauscht werden können. Der Chat bildet dann somit die vierte Stufe der Suche.

Kapitel 3. Die entwickelte Applikation

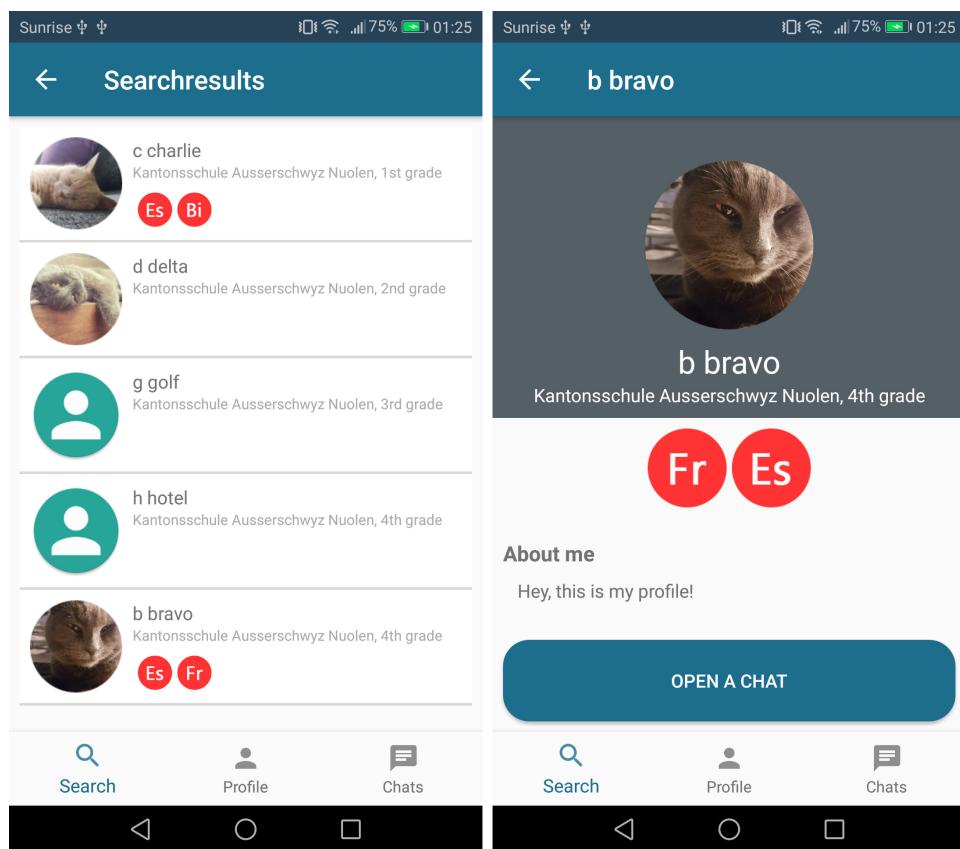


Abbildung 3.6: Links: Auflistung der Suchergebnisse, Rechts: Profil eines Benutzers

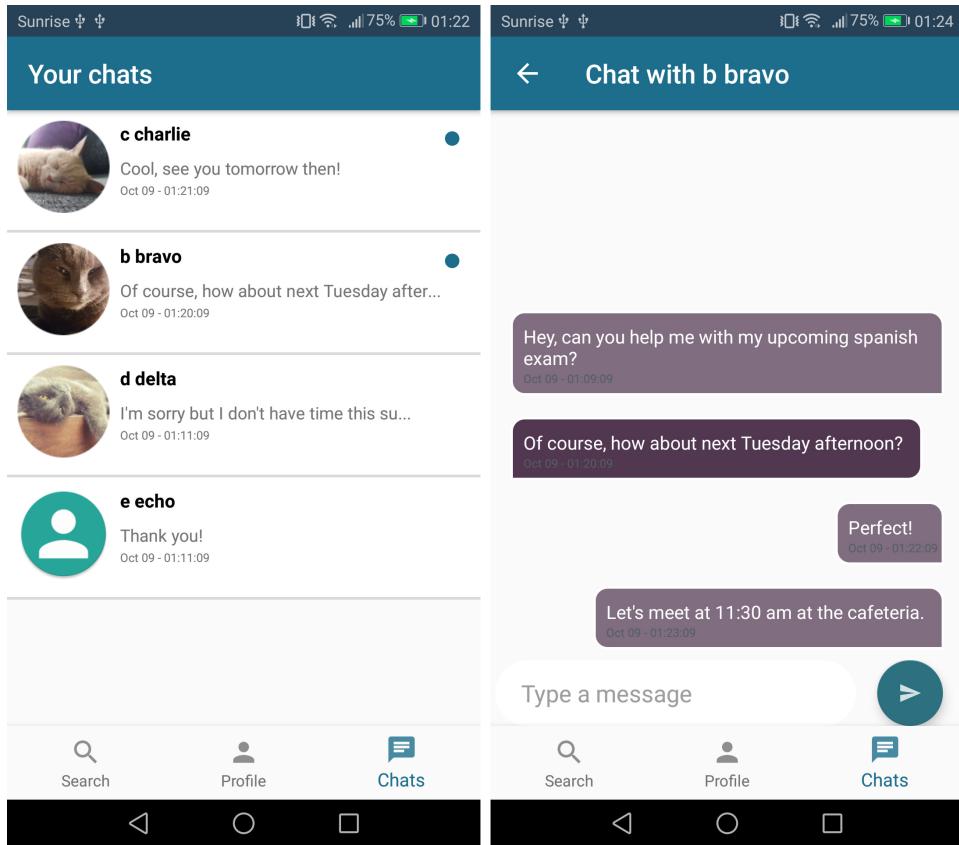


Abbildung 3.7: Links: Auflistung der offenen Chats, Rechts: Offener Chat

Chats

Ebenfalls über die Bottom Navigation kann ein Screen geöffnet werden, wo alle Chats angezeigt werden, in welchen ein Benutzer/eine Benutzerin involviert ist. Die angezeigten Chats in der Übersicht zeigen jeweils die Profilbilder der Chatpartner/ Chatpartnerinnen und die zuletzt gesendete Nachricht an (siehe Abbildung 3.7). Durch das Auswählen eines Elements der Übersicht kann ein gewünschter Chat geöffnet werden (siehe Abbildung 3.7). Die Überlieferungen der Nachrichten im Chat erfolgen in Echtzeit, was eine flüssige Kommunikation zwischen zwei Benutzern/Benutzerinnen ermöglicht.

Kapitel 4

Die Server

Im folgenden Kapitel wird auf die beiden Server eingegangen, welche für die Studnetz Applikation verwendet werden. Zuerst werden die mitwirkenden Komponenten jeweils einzeln erläutert. Die eigentliche Implementation folgt dann in Kapitel 4.4. Ein schematischer Überblick des Verhältnisses der beiden Server zum Client findet sich in Abbildung 4.1.

4.1 Webserver

Bei den beiden in der Applikation verwendeten Server handelt es sich genau genommen um sogenannte Webserver. Ein Webserver beschreibt eine Software, welche das Abrufen von lokalen Diensten (z.B. Programme oder Skripte) und gespeicherten Daten über das Internet ermöglicht. Clients sind dann in der Lage, über die IP-Adresse des Webservers die bereitgestellten Dienste und Daten aufzurufen. Der Webserver ist ebenfalls in der Lage, Antworten auf Anfragen der Clients zurückzusenden.

Skripte, die auf dem Server ausgeführt werden ermöglichen den Zugriff auf eine sich ebenfalls auf dem Server befindende Datenbank. Dafür beliebte Skriptsprachen sind Sprachen wie PHP (siehe Kapitel 4.1.2), Ruby, Python oder Pearl. Diese Skripte sind in der Lage, Anfragen an die Datenbank zu stellen und Antworten zu formulieren, welche dann vom Webserver an den Client zurückgeschickt werden können. Eine direkte Kommunikation zwischen Datenbank und Client findet auf einem Webserver so gut wie nie statt.

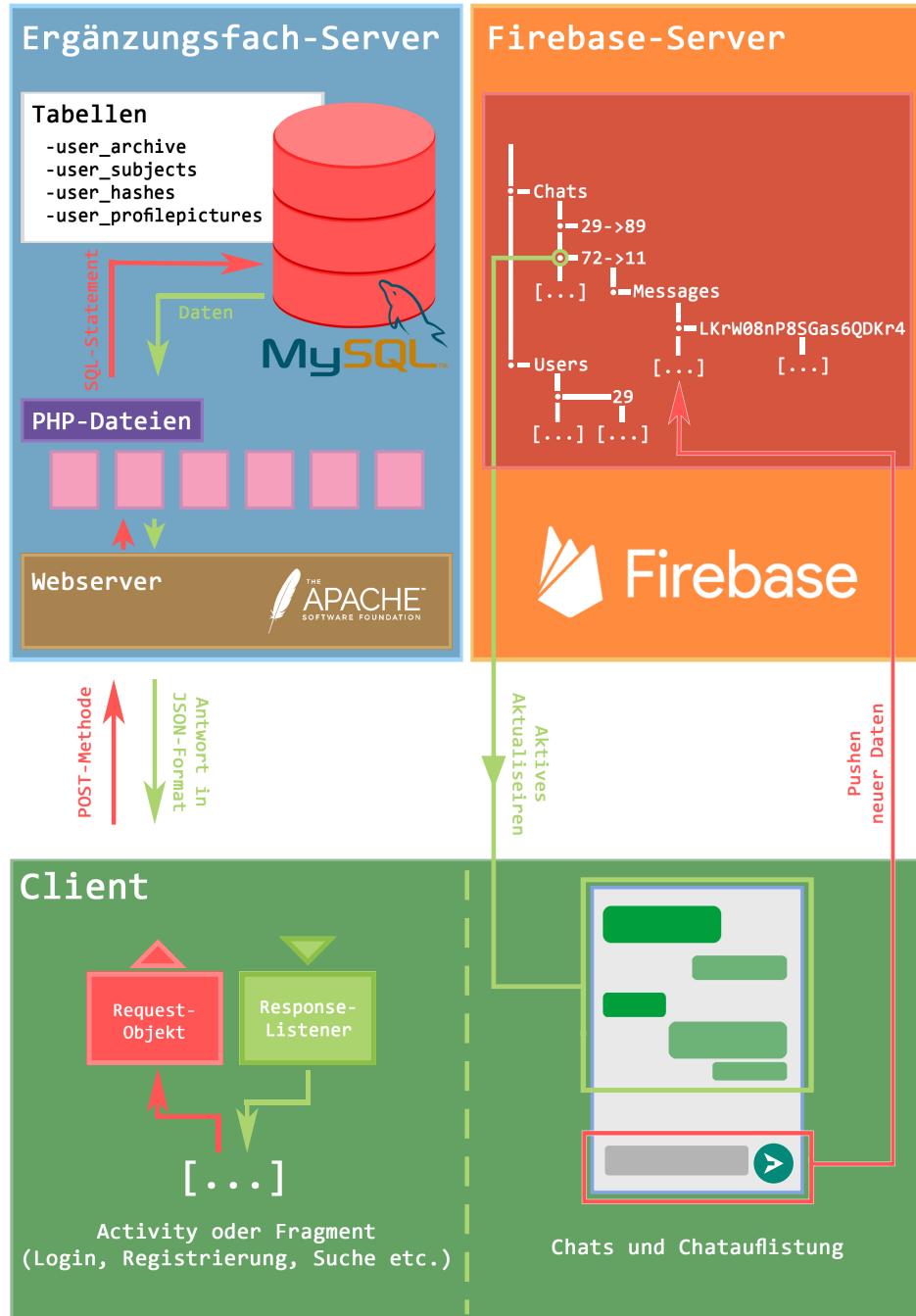


Abbildung 4.1: Schematische Übersicht über die beiden Server und deren Beziehung zum Client

4.1.1 Apache Server

Der *Apache HTTP Server* ist eine Webserver Software. Sie wurde 1995 von der *Apache Group* veröffentlicht und besitzt sich seit 1999 in Besitz der *Apache Software Foundation*. Apache HTTP Server darf kostenlos verwendet werden und ist heute der meist verwendete Webserver weltweit. [30]

4.1.2 PHP

PHP ist eine Open-Source Skriptsprache, welche grosse Beliebtheit in der serverseitigen Webentwicklung findet. Die Abkürzung PHP ist ein rekursives Akronym für *Hyptertext Preprocessor* und wurde speziell für die Webprogrammierung entwickelt. Sich auf Webservern befindende PHP Skripte bieten den grossen Vorteil, dass sie jeweils nur auf dem Server ausgeführt werden. Somit ist es Clients zwar möglich die Skripte via Webadresse auf einem Webserver auszuführen, bekommen jedoch die Codestruktur des PHP Skriptes nie zu Gesicht. [25] Im Skript selber wird eine Antwort formuliert, welche dann vom Webserver an den Client zurückgeschickt werden kann. Die Form der Antwort unterscheidet sich dabei stark von Anwendung zu Anwendung. Antworten können in Sprachen wie HTML, JSON oder JavaScript verfasst sein, wobei jedoch auch Bild- und PDF-Dateien durchaus als Antwortformate in Frage kommen. [1]

4.1.3 JSON

JSON ist eine Abkürzung für *JavaScript Object Notation* und ist eine Datenformat, welches für den Austausch von strukturierten Daten entwickelt wurde. Bei der Entwicklung wurde dabei besonders auf drei Kriterien geachtet: Lesbarkeit für Menschen sowie einfaches Generieren sowie Parsen auf Seiten des Computers. Diese sehr gut umgesetzten Eigenschaften und seine Plattformunabhängigkeit machen JSON zu einem äusserst effizienten Datenaustauschformat für einfachere Datentypen und wird besonders in der Webentwicklung gerne verwendet. [18]

4.2 Datenbanken

Eine der wohl wichtigsten Aufgaben von Computern ist das Speichern, Verwalten und Manipulieren von Informationen. Anwendungen, die sich hauptsächlich mit dieser Aufgabe beschäftigen werden allgemein als *Datenbanken* bezeichnet. Sie haben die Aufgabe, Informationen systematisch

zu ordnen, zu speichern und bei Bedarf zu verändern. Grundsätzlich bezeichnet der Begriff Datenbank gleich zwei Dinge auf einmal. Zum einen wird ein strukturierter Speicher von Informationen als Datenbank bezeichnet und zum anderen auch die Anwendung, die das Verwalten der Daten überhaupt erst ermöglicht. Solche Anwendungen werden als *Database Management System* (DBMS) bezeichnet und sind meist hochkomplex in ihren Funktionsweisen, um die effiziente Verwaltung von selbst riesigen Datensätzen zu ermöglichen. [19] Datenbanken kommen oftmals auf zentralen Servern zum Einsatz, wo zum Beispiel die Benutzer eines Webdienstes oder die Bestellungen einer Firma aufgelistet werden. Sie stellen den dynamischen Speicher eines Servers bzw. Webservers dar.

Tabellenstrukturen

Die Informationen in einer Datenbank werden meist in einer auf Tabellen basierenden Struktur gespeichert. Dabei wird eine einzelne Zeile in der Tabelle als Datensatz oder Eintrag bezeichnet, während die verschiedenen Spalten Felder genannt werden. Beim Erstellen einer solcher Tabelle werden zuerst die verschiedenen Felder bestimmt. Ihnen wird ein Name gegeben, der beschreibt, was darin gespeichert werden soll. Hinzu kommt ein Datentyp, der angibt um was es sich bei diesem Feld handelt (z.B. eine Zahl oder einen kurzen Text). Es ist ebenfalls möglich einem Feld einen *Default*-Wert zuzuschreiben. Dieser wird dann für einen Datensatz verwendet, wenn der Wert des Feldes sonst nicht definiert wurde. Zuletzt ist es noch möglich, einem Feld speziellere Eigenschaften zuzuschreiben. Dazu gehört unter anderem eine Funktion mit dem Namen *auto_increment*. Sie bewirkt, dass diesem Feld bei einem neu eingetragenen Datensatz automatisch ein in der Tabelle einzigartiger Wert des Typen Integer (Datentyp für natürliche Zahlen) zugewiesen wird.

Es ist dabei anzumerken, dass verschiedene Datenbanken oft verschiedene, variierende Datenstrukturen verwenden. Die soeben beschriebene Tabellenstruktur trifft hauptsächlich auf die beiden verbreiteten SQL Datenbanken MySQL und MariaDB zu. Dies schliesst jedoch keineswegs andere Strukturen aus, wie solche, die später in Kapitel 4.2.2.1 thematisiert werden werden.

Datenbanktypen

Datenbanken selber können in verschiedene Typen eingeteilt werden, die alle ihre eigenen Vor- und Nachteile mit sich bringen. Die einfachste Form eines

Datenbanktyps ist wohl die *Einzeltablendatenbank*. Sie besteht aus nur einer Tabelle, in welcher alle Informationen abgespeichert werden. Sie eignet sich gut für kleine, übersichtliche Tabellenstrukturen wie zum Beispiel eine einfache Liste von Adressen. Die Einzeltablendatenbank stösst jedoch spätestens dann an ihrer Grenzen, wenn die Informationen nicht mehr in nur einer, sondern gleich mehreren Tabellen gespeichert werden sollen. Hier tritt ein anderer Datenbanktyp ins Spiel. Eine *relationale Datenbank* ist in der Lage, verschiedene Tabellen logisch miteinander zu verknüpfen und sich darin zu orientieren. Diese logische Verknüpfung ist möglich aufgrund eindeutiger Eigenschaften eines Datensatzes. Dies kann zum Beispiel eine Kundennummer oder ein Name sein. Ein solches Feld wird auch als *Key* bezeichnet. Wichtig dabei ist, dass jeder Key nur einmal in einer Tabelle vorkommt, da ansonsten keine eindeutige Verknüpfungen möglich sind. Die Anwendung zur Verwaltung einer solchen relationalen Datenbank wird *Relational Database Management System* (RDBMS) genannt. [19, S. 745 - 751]

4.2.1 Die MySQL Datenbank

Eines der am weitesten verbreiteten RDBMS ist die MySQL Datenbank. Das System wurde ursprünglich von den drei Gründern Allan Larsson, Michael Widenius und David Axmark 1995 entwickelt, wurde später von *Sun Microsystems* aufgekauft und gelangte schlussendlich in den Besitz des amerikanischen Softwareherstellers *Oracle*. MySQL ist unter einem dualen Lizenzsystem eingetragen, sodass die Software zum einen unter einer *General Public Licence* (GPL) [15], aber auch unter einer proprietären Lizenz [20] gestellt ist. [28] Das MySQL System darf aufgrund der GPL gratis heruntergeladen, installiert und modifiziert werden.

MySQL Datenbanken sind besonders bei der Betreibung von Webdiensten aller Art von grosser Beliebtheit. Sie befinden sich dann, wie bereits in Kapitel 4.2 erwähnt, auf einem zentralen Server. Mithilfe sogenannter *Queries* (Datenbank Abfragen) ist es dem Server möglich mit der Datenbank zu interagieren. Solche Queries sind im Falle einer MySQL Datenbank in der Datenbanksprache *SQL* (Structured Query Language) formuliert. Queries können in vier Arten von Abfragen unterteilt werden: [19, S. 760]

- Auswahlabfragen (*Select Queries*) geben den Inhalt von einem oder mehreren Feldern aus einer oder verschiedenen Tabellen zurück. Dabei kann bei Bedarf nach Kriterien gefiltert werden, um die Suche nach bestimmten Datensätzen einzuschränken. [19, S. 746]

- Einfügeabfragen (*Insert Queries*) fügen einen neuen Datensatz zu einer bestehenden Tabelle hinzu. [19, S. 746]
- Änderungsabfragen (*Update Queries*) ändern bestimmte oder alle Felder eines bestehenden Datensatzes in einer Tabelle. [19, S. 746]
- Löschabfragen (*Delete Queries*) löschen einen Datensatz aus einer Tabelle. [19, S. 746]

4.2.2 Firebase

Firebase ist eine Entwicklungsplattform für Webapplikationen, welche seit 2014 von Google angeboten wird. Firebase entwickelte sich aus dem 2011 gegründete Startup *Envolve* der beiden Gründern James Tamplin und Andrew Lee. Das Ziel von Envolve war es, Kunden eine API (*Application Programming Interface*) zu bieten, mit welcher in Echtzeit synchronisierte Chatfunktionen einfach realisiert werden können. Nachdem jedoch viele Benutzer die API für andere Anwendungen als Chats verwendeten, ja sogar einzelne Spielentwickler sie für eine Synchronisation verschiedener Clients in Echtzeit verwendeten, begann die Entwicklung sich auf das Anbieten einer API für Echtzeitsysteme zu konzentrieren. Der Erfolg war gross und 2014 wurde das Unternehmen von Google aufgekauft. Google entwickelte aus Envolve daraufhin eine Plattform, welche verschiedenste Tools zur Webdienstentwicklung beinhaltet und heute unter dem Namen Firebase vermarktet wird. Firebase bietet sowohl Tools für die Entwicklung neuer Dienste wie Echtzeitdatenbanken, Authentifizierungsfunktionen und Crashanalysen, aber auch für das Unterhalten von bestehenden Diensten. Beispiele für ein solche Tools zur Unterhaltung bestehender Dienste wären zum Beispiel das Senden von Push-Benachrichtigungen an alle Clients oder das Überwachen von geschalteter Werbung innerhalb der Clients. Die Tools dürfen in einem begrenzten Rahmen gratis verwendet werden und sind daher sehr attraktiv für kleinere Entwicklerunternehmen und Lernende, eignen sich aber durchaus auch für grössere Unternehmen. [26]

4.2.2.1 Die Firebase Echtzeitdatenbank

Wie bereits erwähnt, bietet Firebase unter anderem eine sogenannte Echtzeitdatenbank (Realtime Database) an. Diese Echzeitdatenbank ist eine NoSQL Datenbank und unterscheidet sich in ihrer Funktionsweise stark von traditionellen Datenbanken wie MySQL oder MariaDB. Anders als traditionelle Datenbanken auf Webservern agieren Echtzeitdatenbanken nicht nur passiv

auf Anfragen, sondern teilen den Clients aktiv mit, wenn sich ein Datensatz verändert hat (Push-Based Data Access) und halten sie so immer auf dem neusten Stand. Solche Mitteilungen über Datensatzänderungen bezeichnet mal allgemein als *Pushes*. Echtzeitdatenbanken werden besonders dann eingesetzt, wenn sich ein Datensatz häufig oder jederzeit ändern kann und es notwendig ist, dass die Clients ohne grosse Verzögerung davon unterrichtet werden. [35] Bei der Firebase Echtzeitdatenbank ist es nicht einmal nötig, dass die Clients zur Zeit des Pushes online sind. Sie werden beim nächsten Start automatisch dann mit dem Datensatz auf der Datenbank abgeglichen und aktualisiert. Ein weiterer grosser Unterschied der Firebase Echtzeitdatenbank zu einer MySQL Datenbank ist, dass Daten nicht mehr in Tabellenstruktur gespeichert werden. Vielmehr wird ein System verwendet, welches optisch an ein Verzeichnissystem erinnert, wobei es sich jedoch nicht wirklich um Verzeichnisse handelt. Genauer genommen werden die Daten als JSON-Objekte gespeichert und strukturiert. Dabei ist es möglich, mehrere JSON-Objekte jeweils ineinander abzuspeichern, was zu dieser Verzeichnisartigen Struktur führt. Eine solche Struktur nennt man dann auch einen *JSON-Tree*, wobei die einzelnen Elemente als *Nodes* (Knoten) bezeichnet werden. Diese Eigenschaft macht die Firebase Echtzeitdatenbank zu einer äusserst flexiblen Datenbank, da sie dadurch kaum an vorgegebene Strukturen wie Tabellen oder Felder gebunden ist. [14][13]

4.3 Sicherheit der Passwörter in einer Datenbank

Generell gilt, dass Passwörter nie in ihrer reinen Form irgendwo gespeichert werden dürfen, sofern sie vor potentiell bösartigen Angriffen geschützt sein sollen. Um eine Lesbarkeit der Passwörter zu vermeiden wird oft ein sogenannter Prozess des Hashens und des Salzens der Passwörter angewandt.

4.3.1 Hashen

Hashen ist ein mathematisches Verfahren, das im Grunde genommen die Bits einer Eingabe vermischt und zu einem Ergebnis bringt, welches nicht mehr in seine Ursprungsform zurückgerechnet werden kann. Der grosse Vorteil dabei ist, dass jede Eingabe einen absolut einzigartigen Hash bekommt. Es kann keine Kollisionen geben, ausser die Eingabe ist gleich, jedoch dazu etwas später mehr. Weiter sind alle Hashes unabhängig von der Länge der Eingabe immer gleich lang. Es kann also nicht einmal die Passwortlänge aus einem Hash herausgelesen werden. Das Hashen der Passwörter wäre also eine schon deutlich sicherer Methode, um die Passwörter auf dem Server zu

speichern. Somit würde beim Login jeweils das vom Benutzer/der Benutzerin eingegeben Passwort ebenfalls gehasht werden und anschliessend mit dem auf dem Datenbank gespeicherten Hash verglichen werden. Da es keine Kollisionen geben kann, wäre dies eine durchaus sichere Variante, um die Passwörter für einen Menschen unleserlich zu machen.

Doch leider reicht dies noch nicht. Hashes sind mittlerweile gut dokumentiert und ein gehashtes Wort kann einfach in einer Suchmaschine eingegeben werden und mit etwas Glück wird einem bereits in den ersten Ergebnissen das gehashte Wort in Klartext angezeigt. Zudem kommt die Gefahr, dass zwei Benutzer in einer Datenbank zufällig das gleiche Passwort verwenden könnten. Dann wäre ihr gehashtes Passwort ebenfalls identisch. Solche Gegebenheiten können von einem potenziellen Angreifer ausgenutzt werden. [27] [8]

4.3.2 Salzen

Um dieser Problematik vorzubeugen wird ein zweiter Prozess, der allgemein als Salzen bezeichnet wird, angewandt. Salzen ist ein Verfahren, bei welchem jeweils vor oder nach der eigentlich zu hashenden Eingabe noch eine Reihe von zufällig generierten Werte angehängt wird. Diese Werte sind dann das *Salz* des Hashes, da sie den Hash komplett verändern, wie ein Gewürz den Geschmack einer Speise verändert. Die Idee ist, dass jeweils jeder Benutzer sein eigenes zufällig generiertes Salz zu seinem Passwort bekommt. Das Salz wird ebenfalls in der Datenbank gespeichert und muss dabei nicht einmal verschlüsselt werden. Es kann direkt neben dem gehaschten Passwort gespeichert sein. Somit wird jeweils bevor das Passwort gehasht wird das Salz dem Passwort angehängt. Sofern jeder Benutzer ein eigenes Salz besitzt, kann es zu keinen Überschneidungen bei den Passwörtern kommen. Um die Einzigartigkeit der Salze zu gewährleisten wird daher zu Salzen mit mehr als 16 Bytes Länge geraten. Die Chance für eine Überschneidung fällt dann in einen vernachlässigbaren Bereich.

Selbst wenn nun ein Angreifer in Besitz sämtlicher Passwörter und Salze der Datenbank käme, wäre es ihm noch immer nicht möglich, die Passwörter der einzelnen Benutzern herauszufinden, ohne dafür auf einen Bruteforce-Attack (Ausprobieren aller möglicher Zeichenkombinationen) für jedes einzelne Passwort zurückzugreifen. Diese Verfahren gilt daher allgemein als sicher. [27] [8]

4.3.3 bcrypt

Innerhalb der Studnetz Applikation wird für diesen Prozess die kryptologische Hashfunktion *bcrypt* verwendet. Bcrypt ist speziell für das sichere Speichern von Passwörtern entwickelt worden und hasht Eingaben automatisch mit einem Salz. Die Besonderheit an bcrypt ist, dass sie im Gegensatz zu anderen populären Hashfunktionen wie die Algorithmen der *SHA*-Familie (Secure Hash Algorithm) nicht auf Effizienz optimiert worden ist. Das Ziel von bcrypt ist es stattdessen, den Aufwand für das Hashen einer Eingabe möglichst aufwändig zu halten. Der Rechenaufwand kann dabei manuell eingestellt werden. Wenn der Rechenaufwand genug hoch ist (gewöhnlich wird dafür der Wert 10 oder 11 verwendet) macht dies Anwendung von Bruteforce-Attacken zu einem äusserst zeitaufwendigen Prozess, die bei einem einigermassen sicheren Passwort kaum von Erfolg gekrönt werden. [6]

Um dies etwas deutlicher zu machen verweise ich auf einen Benchmarktest von Jeremy M. Gosney, der diverse Hashalgorithmen auf ihre Effizienz verglich. Dabei ist es ihm möglich gewesen, mithilfe der SHA-512 Hashfunktion pro Sekunde ca. 1'075 Millionen Hashes zu generieren, während unter den gleichen Umständen nur ca. 13'100 bcrypt Hashes pro Sekunde mit dem Kostenwert 5 generiert werden konnten. [16]

4.4 Implementation in der entwickelten Applikation

Der folgende Abschnitt des Kapitels befasst sich mit der eigentlichen Implementation der in diesem Kapitel bereits erläuterten Komponenten in der entwickelten Applikation.

4.4.1 Die verwendeten Server

Innerhalb der Studnetz-Applikation werden zwei verschiedene, unabhängige Server verwendet.

Auf der einen Seite findet sich ein herkömmlicher Apache HTTP Webserver, welcher mir freundlicherweise von Herrn Andreas Umbach zur Verfügung gestellt worden ist und sonst für das Ergänzungsfach Informatik verwendet wird. Auf ihm befindet sich eine MySQL Datenbank und diverse PHP-Dateien. Er wird für das Speichern der Accountdaten von den einzelnen Benutzern/Benutzerinnen verwendet.

Auf der anderen Seite findet sich ein mit einem Google Account verbundener Firebase Server mit einer Echtzeitdatenbank. Er wird für die Chats

der Applikation verwendet.

4.4.2 Datenbankstrukturen

Das geschickte Planen und Strukturieren von Datenbanken ist wohl einer der wichtigsten Schritte für die Entwicklung einer Applikation, die von auf einer Datenbank basiert. Oft ist dies auch einer der ersten Schritte in der Planung einer Applikation überhaupt, da die Datenbankstruktur oft massgebend die Funktion der Clients und der angebotenen Dienste bestimmt. Dabei sollte zuerst jeweils einmal festgelegt werden, welche Aufgaben und Ziele die Datenbank erfüllen soll. Dann müssen, im Falle von tabellenbasierten Datenbanken, Tabellen und ihre Felder so strukturiert werden, dass diese Anforderungen erfüllt werden können.

Ein Weg, die geplante Struktur zu visualisieren, kann das Erstellen eines *Entity-Relationship-Models* (ERM) sein. ERMs eignen sich gut für das Darstellen der einzelnen Tabellen und ihren Feldern, wie auch das Verhältnis, in welchem die Tabellen zu einander stehen. Hierzu werden die verknüpften Tabellen mit einander verbunden und dabei das verbindende Feld markiert (s.B. eine Kundennummer oder Bestellnummer). Die Verhältnisse werden dann im Schema (1, 1) über die Verbindung geschrieben. Die erste Ziffer gibt dabei die minimale Anzahl an herrschenden Beziehungen an, die zweite Ziffer die maximale Anzahl. Im Falle von (1, 1) Beziehungen bedeutet dies, dass es immer genau eine solche Beziehung geben darf und auch muss. Sind die Anzahl Beziehungen unlimitiert, stehen anstelle der Ziffern stellvertretend die Variablen m und n . Das für die MySQL Datenbank dieser Arbeit verwendete ERM ist in Abbildung 4.2 dargestellt.

4.4.2.1 MySQL Datenbankstruktur

Die MySQL Datenbank der entwickelten Applikation umfasst vier Tabellen:

- Die *user_archive* Tabelle beinhaltet allgemeine Informationen zu registrierten Benutzern/Benutzerinnen. Dazu gehören Informationen wie Benutzername, Vor- und Nachname, Email, Schule, Klasse und Beschreibung. Zudem kommt noch eine Benutzer ID (*user_id*), welche jedem Benutzer automatisch über die *auto_increment* Funktion beim Registrieren zugewiesen wird und für jeden Benutzer einzigartig ist.
- Die *user_profilepictures* Tabelle besitzt drei Felder. Eines für die *user_id*, und zwei Felder vom Datentyp *Medium BLOB*, in welchen jeweils eine kleine und eine grosse Variante des Profilbildes eines Benutzers/einer

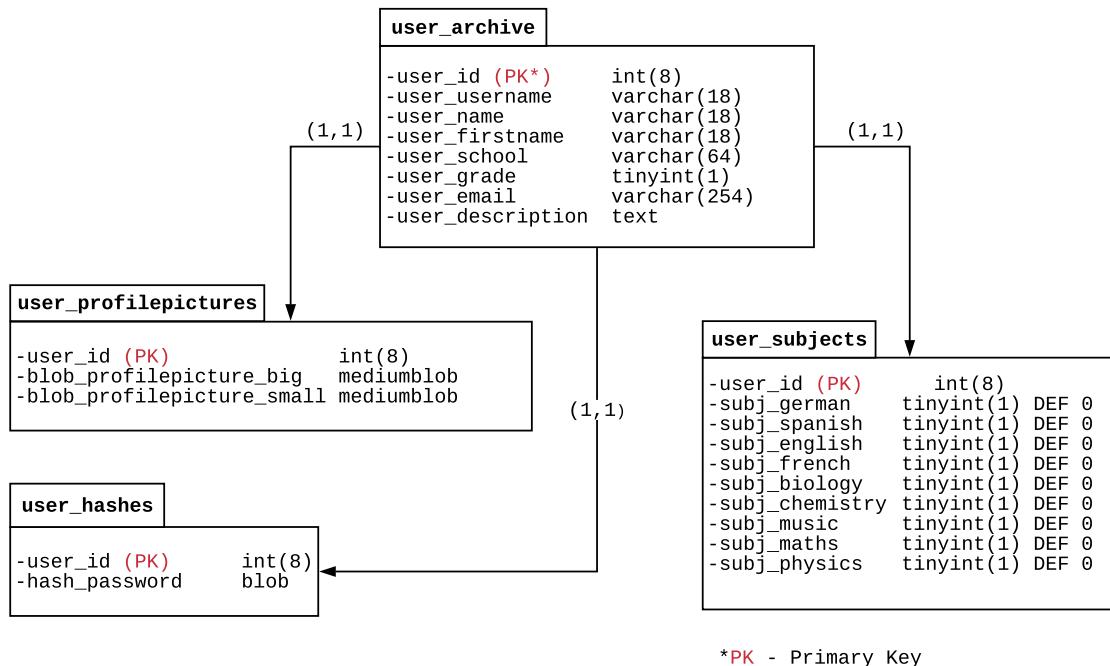


Abbildung 4.2: Das ERM der MySQL Datenbank

Benutzerin gespeichert wird. Die Bildformate sind dabei jeweils in Forme einer *Base64* kodierten Zeichenkette. Die Base64 Kodierung ermöglicht den sicheren Transfer von Bilddateien vom Client zum Server, ohne dass es dabei zu Datenverlust oder Modifikationen aufgrund von Inkompatibilitäten kommt [11]. Das Speichern von grösseren Bilddateien in relationalen Datenbanken ist umstritten, da es die Effizienz der Datenbank negativ beeinflussen kann. Man findet hier verschiedene Ansichten. Im Rahmen dieser Arbeit jedoch sollte diese Datenbankkonfiguration keine grösseren Probleme mit sich bringen, weshalb es dabei belassen wurde.

- Die *user_hashes* Tabelle besitzt zwei Felder. Eines für die *user_id* und eines für den Hash des Passworts.
- Die *user_subjects* Tabelle beinhaltet Felder für alle von der entwickelten Applikation unterstützten Fächer. Mithilfe der Werte 1 und 0 wird angegeben, in welchen Fächern ein Benutzer/eine Benutzerin Nachhilfeunterricht anbietet (1 steht für "Ja", 0 für "Nein").

Die Hauptaufgabe der vier Tabellen ist das Speichern aller statischen Informationen über die verschiedenen registrierten Benutzer/Benutzerinnen. Die einzelnen Tabellen können alle über die *user_id* eines Benutzers miteinander verknüpft werden. Die *user_id* ist zudem in allen Tabellen als ihr jeweiliger *Primary Key* markiert. Für die Datenbank bedeutet dies, dass dieses Feld jeweils für jeden Datensatz einen einzigartigen Wert besitzt und für die Unterscheidung der Datensätze optimiert werden soll. Besonders für grosse Datenbanken ist eine geschickte Handhabung solcher Keys von grosser Bedeutung, da sie die Effizienz einer Datenbank drastisch beeinflussen kann.

4.4.2.2 Firebase Datenbankstruktur

Wie bereits in Kapitel 4.2.2 erwähnt, ist die Firebase Echtzeitdatenbank etwas anders strukturiert, weshalb das dazu gehörende ERM eine baumähnliche Struktur aufweist, wie sie in Abbildung 4.3 dargestellt wird. Auf der Abbildung findet sich ein Schema der für die Studnetz Applikation verwendeten Firebase Datenbankstruktur. Dabei kann zwischen den zwei Hauptknoten (Nodes) unterschieden werden. Hierbei ist anzumerken, dass sich die verwendeten Begriffe *Sender* und *Receiver* aus der Sicht des momentan aufgerufenen Profils verwendet werden. Somit sind bei zwei Chatpart-

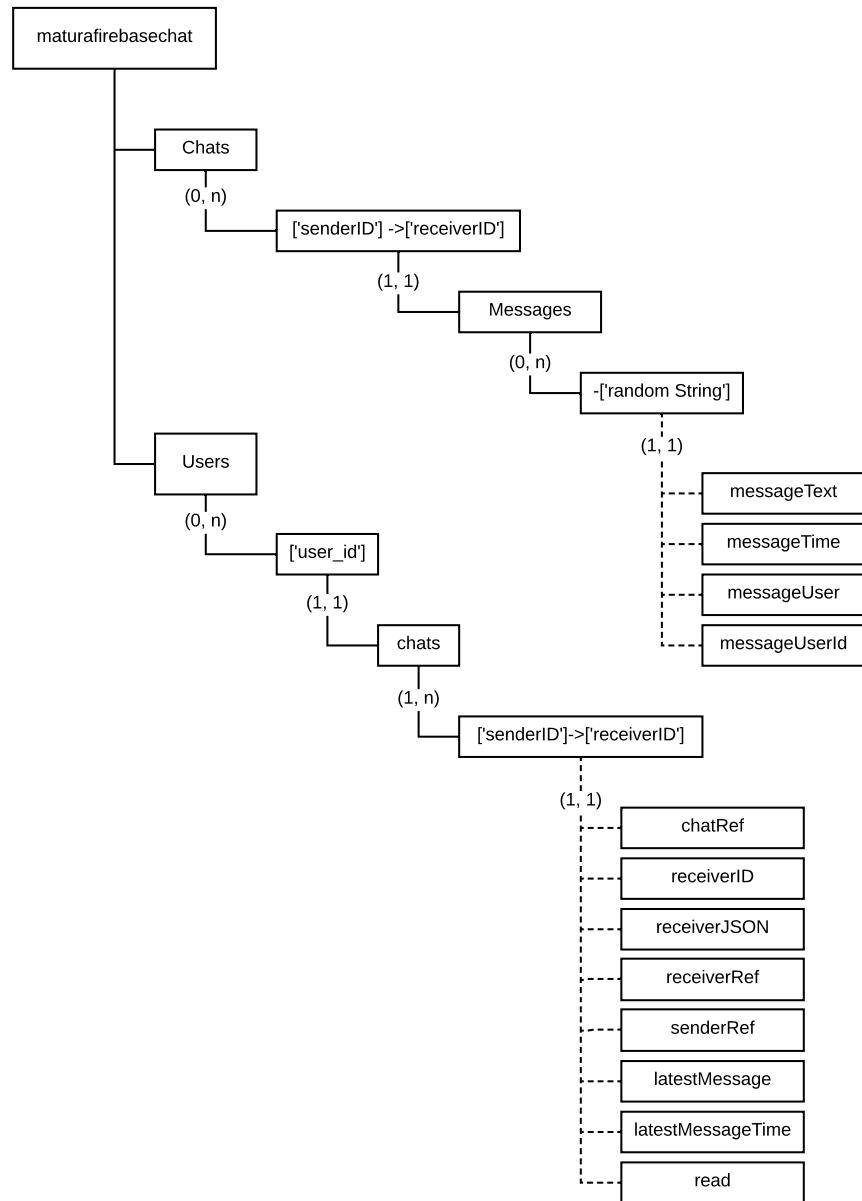


Abbildung 4.3: Darstellung der Firebase Echtzeitdatenbankstruktur

nern/Chatpartnerinnen beide aus der einen Sicht der Sender, während sie von der anderen Seite aus der Receiver sind.

- Der *Users*-Knoten beinhaltet eine Liste aller Benutzer/Benutzerinnen, die je in einem Chat in der Applikation involviert gewesen sind. Die Knoten für die einzelnen Benutzer/Benutzerinnen wird jeweils mit der `user_id` des Benutzers/der Benutzerin aus der MySQL Datenbank gekennzeichnet. Darin findet sich ein Knoten *chats*. Dort werden Knoten für alle Chats aufgeführt, in welchen ein Benutzer/eine Benutzerin involviert ist. Die Knoten sind jeweils so benannt, dass beide `user_ids` der am Chat Benutzer/Benutzerinnen herausgelesen werden können. Der Knoten eines Chats enthält die acht für einen Chat relevanten Werte: die Referenz zum Chat, wo die Nachrichten des Chats aufgeführt sind, ein Wert für die zuletzt gesendete Nachricht, die Zeit, zu welcher sie gesendet wurde, der Status, ob sie gelesen wurde, die Referenzen zu den beiden Benutzern/Benutzerinnen (Sender sowie Receiver), ein JSON-String, in welchem die Benutzerinformationen des Receivers gespeichert sind, und die `user_id` des Receivers.
- Der *Chats*-Knoten beinhaltet Knoten für alle Chats, die in der Applikation geöffnet sind. Darin findet sich ein Knoten mit dem Namen *Messages*. Unter diesem Knoten findet man Knoten für alle im Chat gesendeten Nachrichten. Die Knoten der einzelnen Nachrichten tragen dabei jeweils eine generierte Zeichenkette als Namen. Die Nachrichten selber bestehen aus vier Werten: der Nachricht selber, der Zeit, zu welcher die Nachricht gesendet wurde, dem Benutzernamen des Senders und der `user_id` des Senders.

Es ist an dieser Stelle noch interessant zu erwähnen, weshalb die Firebasestruktur auf den ersten Blick etwas überkompliziert erscheinen mag. Der Grund dafür ist eine spezifische Eigenheit von Echtzeitdatenbanken. Anders als bei relationalen Datenbanken ist es in Echtzeitdatenbanken nur schwer möglich, nach Einträgen zu suchen, bei welchen nicht der gesamte Namen bekannt ist. Somit ist es nicht möglich für die Darstellung der Chats auf der Hauptseite einfach nach Einträgen im Chats Ordner zu suchen, in welchen die `user_id` des Benutzers/der Benutzerin vorkommt. Deshalb musste eine Alternative gefunden werden, welche sich nun in der Form des User Ordners ausdrückt. Dort werden alle Pfade zu den Chats gespeichert, in welchen ein Benutzer/eine Benutzerin vorkommt. Die Pfade werden beim Eröffnen eines neuen Chats dort eingetragen. Auf diese Weise ist es trotzdem möglich, alle

offenen Chats eines Benutzers/einer Benutzerin zu finden, ohne dass eine Suchfunktion benötigt wird.

4.4.3 PHP Skripte

Im folgenden Abschnitt wird nun auf die PHP Skripte eingegangen, die auf dem Webservers der MySQL Datenbank verwendet werden. An dieser Stelle ist anzumerken, dass im folgenden Kapitel für das Hashing des Passwortes von einer PHP Version $\geq 5.3.7$ ausgegangen wird. Da auf dem Ergänzungsfachserver die PHP Version 5.3.29 installiert ist, unterscheiden sich die Namen der Hashfunktionen in der entwickelten Applikation von den in diesem Kapitel beschriebenen Hashfunktionen leicht, die Funktionsweise ist jedoch vergleichbar. Für das Hashing in der entwickelten Applikation musste wegen der PHP Version eine zusätzliche Bibliothek installiert werden (PHP-PasswordLib by Anthony Ferrera [5]). Wichtig zu wissen ist jedoch, dass die mit der Bibliothek generierten Salze bekanntlicherweise Fehler aufweisen können und daher als veraltet gelten. Ein Update wäre also vor einer möglichen Veröffentlichung nötig.

4.4.3.1 Login

Damit ein Client Zugriff auf einen Datensatz eines Benutzers/einer Benutzerin in der Datenbank bekommt, muss sich der Benutzer/die Benutzerin zuerst unter einem bestehenden Account einloggen. Hierzu muss im Client sowohl der Benutzername wie auch das Passwort eines registrierten Accounts eingegeben werden. Danach erfolgt der restliche Login-Prozess auf dem Server.

Hierzu wird vom Client die *login_bcrypt.php*-Datei auf dem Webserver via Webadresse aufgerufen. Dabei werden über die *POST*-Methode der eingegebene Benutzername und das Passwort mitgegeben. Diese Werte können innerhalb der PHP-Skriptes verwendet werden. Auf dem Webserver wird überprüft, ob ein Datensatz in der Datenbank existiert, dessen Benutzername dem eingegebene Benutzernamen entspricht. Dies wird über ein sogenanntes *MySQL Statement* getan. Solche Statements sind in der Lage, Anfragen an die verbundene Datenbank zu stellen. In diesem Falle wäre das nun eine Auswahlabfrage (Select Query), die in der user_archive Tabelle nach einem Datensatz mit dem eingegebenen Benutzername sucht, wobei die Tabelle über einen *INNER JOIN* mit der user_hashes Tabelle verknüpft wird (siehe Abbildung 4.4). Die Fragezeichen stehen stellvertretend für die einzufügenden Werten, die erst später zugewiesen werden. Statements, die

auf diese Weise formuliert werden, werden als sogenannte *Prepared Statements* bezeichnet. Sie werden verwendet, um sogenannten *SQL Injektionen* vorzubeugen [32]. Bei der Auswahlabfrage in Abbildung 4.4 darf jeweils nur genau ein Eintrag gefunden werden, da sonst nicht klar wäre, an welchen Account das Login gerichtet ist. Deshalb muss jeder Benutzername in der Datenbank einzigartig sein.

```
SELECT user_hashes.hash_password FROM user_archive INNER JOIN
    ↵ user_hashes ON user_archive.user_id = user_hashes.user_id
    ↵ WHERE user_archive.user_username = ?
```

Abbildung 4.4: Select Query für das Auslesen des Passwortes eines Benutzers aus der user_archive Tabelle und der user_hashes Tabelle.

Wenn ein Ergebnis gefunden werden konnte, wird der Hash des Passwortes aus der Datenbank ausgelesen und mit dem eingegebenen Passwort verglichen. Dies geschieht über die *password_verify*-Funktion von PHP, welche mit der PHP Version 3.5.7 implementiert wurde (siehe Abbildung 4.5). Wenn das eingegebene Passwort mit dem gehaschten Passwort übereinstimmt, wird fortgefahrene.

```
1 <?php
2     if(password_verify($user_password,
3         ↵ $check['hash_password'])) {
4             [...]
5             //Auslesen des Datensatzes
6         };
7     ?>
```

Abbildung 4.5: Überprüfen eines eingegebenen Passwortes

Nun können alle benötigten Informationen über den Benutzer aus der Datenbank über eine Auswahlabfrage ausgelesen werden und im JSON-Format an den Client geschickt werden. Im JSON wird dem Client gleichzeitig mitgeteilt, dass das Login erfolgreich war.

Sollte jedoch irgendwo in diesem Prozess ein Fehler auftreten, wie zum Beispiel die Verwendung eines nicht existenten Benutzernamen oder die Eingabe eines falschen Passworts, wird dem Client der Fehlschlag mitgeteilt und es werden keine weiteren Informationen aus der Datenbank preisgegeben.

4.4.3.2 Registrierung

Wenn ein Benutzer/eine Benutzerin noch keinen Account hat, soll er/sie die Möglichkeit haben, einen zu erstellen. Hierzu kann er/sie im Client die benötigten Daten eingeben und es wird eine Anfrage an das Skript *register_bcrypt.php* auf dem Server geschickt. Dieses Skript hat die Aufgabe in allen vier Tabellen einen neuen Eintrag für den Benutzer/die Benutzerin zu erstellen, sofern die eingegebenen Daten legitim sind. Zuerst wird dafür eine Auswahlabfrage formuliert, welche nach einem Datensatz mit dem eingegebenen Benutzernamen oder der angegebenen Mailadresse sucht. Nur wenn kein solcher Datensatz vorhanden ist, wird fortgefahren, ansonsten wird frühzeitig eine Antwort an den Client geschickt, die ihm mitteilt, dass die Registrierung erfolglos war, da bereits ein Eintrag mit diesem Benutzernamen oder Mailadresse existiert. Wenn kein bestehender Datensatz mit dem gewählten Benutzernamen gefunden worden ist, wird ein SQL Statement vorbereitet, welches einen neuen Datensatz in die Tabelle *user_archive* einfügt (Insert Query), und bei welchem sogleich alle vom Benutzer/von der Benutzerin angegebenen Daten in die Felder eingefügt werden (siehe Abbildung 4.6).

```
INSERT INTO user_archive(user_username, user_name, [...])
↪ VALUES (?, ?, [...])
```

Abbildung 4.6: SQL Insert Query des register.php Skriptes in die *user_archive* Tabelle

Anschliessend werden nacheinander ähnliche Einfügeabfragen an die drei weiteren Tabellen gestellt, wobei jeweils in allen neuen Datensätzen die gleiche *user_id* verwendet wird. In der *user_subjects*- wie auch in der *user_profile_pictures*-Tabelle werden dabei sämtliche weiteren Werte standardmässig auf 0 gesetzt. Diese Werte sollen erst später über die Profileinstellungen bearbeitet werden können.

Das gewählte Passwort wird auf dem Server gehasht und in der separaten *user_hashes*-Tabelle gespeichert. Das Hashing erfolgt dabei mithilfe der *password_hash*-Funktion von PHP, welche mit der PHP Version 5.3.7 implementiert wurde. Dabei können über die Parameter der Funktion die Eingabe, die zu verwendende Hashfunktion und die genaueren Hashkonfigurationen definiert werden. Ein Beispiel hierfür findet sich in Abbildung 4.7.

```

1 <?php
2     $options = [
3         'cost' => 11
4     ];
5     $hash_password = password_hash($user_password,
6         → PASSWORD_BCRYPT, $options);
7 ?>

```

Abbildung 4.7: Hashen einer Eingabe mithilfe der bcrypt-Hashfunktion und dem Kostenwert 11

Der über diese Funktion generierte *String* (Zeichenkette) enthält alle nötigen Informationen über den Hash wie die verwendete Hashfunktion und das Salz.

Sofern bei diesem Prozess keine weiteren Fehler auftreten wird eine JSON-Antwort geschickt, die ihm die erfolgreiche Registrierung mitteilt.

4.4.3.3 Änderung der Informationen via Einstellungen des Clients

Es soll den Benutzern/Benutzerinnen möglich sein, gewisse Angaben wie Email, Passwort, Profilbild oder die ausgewählten Fächer nachträglich zu verändern. Der Client bietet dafür einen Settings-Screen (Settings Activity) über welchen die Angaben editiert werden können. Wenn die editierten Angaben auf dem Client gespeichert werden, wird eine Anfrage an das Skript *savesettings_bcrypt.php* geschickt. Dieses Skript ist in der Lage bereits bestehende Datensätze in der Datenbank zu verändern und die veränderten Daten zurückzuschicken. Hierzu wird zuerst eine Verbindung mit der Datenbank erstellt. Daraufhin wird überprüft, ob die Anfrage auch die Berechtigung für eine solche Änderungsabfrage (Update Query) hat. Im Falle der entwickelten Applikation ist das sehr simpel gelöst. Es wird ähnlich wie beim *login.php* Skript eine Auswahlabfrage formuliert, die nach einem Datensatz mit einer bestimmten *user_id* fragt und das gehashte Passwort überprüft. Das Passwort muss dabei vom Nutzer nicht manuell eingegeben werden, sondern wird im Hintergrund von der Client Applikation automatisch geregelt. Diese Vorkehrung ist aus Sicherheitsgründen sehr wichtig. Sollte das Passwort nicht überprüft werden, wäre es möglich, dass unberechtigte Personen die Datensätze von Benutzern/Benutzerinnen editieren könnten. Ist die Auswahlabfrage erfolgreich, wird zunächst eine Update Query formuliert, welches den Datensatz des Benutzers/der Benutzerin in der Tabelle

user_archive mit den neu erhaltenen Daten aktualisiert (siehe Abbildung 4.8).

```
UPDATE user_archive SET [...] WHERE user_id=?
```

Abbildung 4.8: SQL Update Query des savesettings.php Skriptes

Anschliessend wird der Datensatz per Auswahlabfrage wieder ausgelesen und die Daten in lokalen Variablen gespeichert. Das Gleiche wird auch mit der user_subjects Tabelle und der user_hashes Tabelle gemacht. Die user_profilepictures Tabelle auf der anderen Seite wird aus Effizienzgründen nur dann aktualisiert, wenn das Profilbild auch wirklich geändert wurde. Sämtliche ausgelesenen Daten, inklusive Profilbild, werden in einen Array gespeichert, welcher anschliessend im JSON Format an den Client zurückgeschickt wird. So wird gewährleistet, dass die lokalen Informationen auf dem Client mit Sicherheit mit denen in der Datenbank übereinstimmen.

4.4.3.4 Suchanfrage

Damit es Benutzern/Benutzerinnen möglich ist, nach anderen Benutzerinnen und Benutzern zu suchen, wird ein PHP Skript benötigt, welches die Datenbank nach Datensätzen durchsucht, welche bestimmten Kriterien erfüllen. Diese Kriterien können vom Benutzer/von der Benutzerin jeweils vor einer Suche definiert werden. Es soll nach Namen, nach Fächern, nach Schulen und Klassenstufen gesucht werden können. Das Skript *search.php* übernimmt diese Aufgabe. Hierzu wird zuerst eine Verbindung mit der Datenbank aufgebaut. Ist das erfolgreich, wird eine Select Query erstellt, die alle den Kriterien entsprechenden Datensätze zurückliefern soll.

Das MySQL Statement wird hierzu in zwei Schritten erstellt.

Im ersten wird ein String erstellt, der gleich zwei Dinge tut.

Zum einen schafft er eine Verknüpfung zwischen der user_archive-Tabelle und der user_subjects-Tabelle. Dies wird durch die SQL Funktion *INNER JOIN* erreicht. Die beiden Tabellen sind somit über die user_id miteinander verknüpft. Nun kann die Select Query in beiden Tabellen gleichzeitig nach mehreren Kriterien suchen und die Ergebnisse direkt zusammenführen (siehe Abbildung 4.9).

```
SELECT user_archive.*, user_subjects.* FROM user_archive INNER
→ JOIN user_subjects ON
→ user_archive.user_id=user_subjects.user_id WHERE
→ user_archive.user_username LIKE ? OR [...] AND
→ user_archive.user_school = ? AND [...]
```

Abbildung 4.9: Etwas abgekürzte Version der SQL Select Query des search.php Skriptes mit einem INNER JOIN der user_archive Tabelle und der user_subjects Tabelle

Zum anderen beinhaltet dieser String das Suchkriterium für die Schule, die Klasse und den Namen. Für den Namen wird die SQL Funktion *LIKE* verwendet. Diese Funktion ermöglicht es, nach Daten zu suchen, die den eingegebenen Wert enthalten, jedoch nicht unbedingt exakt identisch sind. Zum Beispiel würde bei der Suche nach einem „Max“ auch der Datensatz von „Maximilian“ gefunden werden. Dies ermöglicht es Benutzern/Benutzerinnen auch ein gewünschtes Suchresultat zu finden, wenn sie nicht den exakten Namen kennen. Es kann nach Benutzernamen, Vor- sowie Nachnamen gesucht werden.

Im zweiten Schritt werden an den erstellten String die Kriterien für die ausgewählten Fächer angehängt. Dabei werden alle ausgewählten Fächer nacheinander als obligatorische Kriterien an den String angehängt. Das bedeutet, dass wenn nach einer „Magalie“ gesucht wird und gleichzeitig noch Mathematik als ein Kriterium ausgewählt ist, nur die Datensätze ausgewählt werden, bei welchen zum einen der Name „Magalie“ vorkommt, aber auch Mathematik als angebotenes Fach markiert ist.

Die Informationen der Suchergebnisse werden einzeln in eigenen Arrays gespeichert. Nicht enthalten sind dabei jedoch sämtliche Passwortinformationen, Emailadressen und Profilbilder. Die einzelnen Arrays werden anschliessend durchnummieriert und in einem weiteren Array gespeichert. So sind alle Suchergebnisse kompakt und systematisch geordnet und können als ein einzelnes Element im Antwortarray referenziert werden, welcher dann im JSON-Format an den Client zurückgeschickt wird.

4.4.3.5 Profilbilder

Wie bereits in Kapitel 4.4.3.4 erwähnt, werden die Profilbilder nicht gleich bei der Suche abgefragt. Sie werden erst später über das *smallimages.php* Skript aus der Datenbank ausgelesen. Diese Skript hat die Aufgabe, die kleine Version der Profilbilder einer Reihe von Benutzern/Benutzerinnen aus der Datenbank auszulesen und an den Client zurückzugeben. Dieses Skript wird jeweils bei der Anzeige der Suchergebnisse aufgerufen, wo jeweils die Profilbilder von 20 Benutzern geladen werden sollen. Mehr dazu in Kapitel 5.2.6.

Weiter findet sich auch ein Skript mit dem Namen *profilepicture_big.php*, welches ein einzelnes Profilbild in der grossen Version eines Benutzers/einer Benutzerin ausliest. Dies wird für das Öffnen eines Profils benötigt, wo zwar bereits alle Informationen wie Name, Beschreibung etc. bekannt sind, jedoch das Profilbild nur in der kleinen Version geladen ist. Mehr dazu auch in Kapitel 5.2.6.

Kapitel 5

Der Client

Der Client ist das Herzstück der entwickelten Anwendung. Er ist der Teil der Anwendung, welcher von den Endbenutzern/Endbenutzerinnen heruntergeladen und benutzt wird. Er ist das verbindende Glied zwischen den gespeicherten Informationen auf dem Server und den Benutzern/Benutzerinnen. Im folgenden Kapitel wird auf die Funktionsweise des Clients eingegangen und seine Interaktionen mit dem Server beschrieben.

5.1 Betriebssystem und Programmiersprache

5.1.1 Android

Der Client wurde für das Betriebssystem *Android* entwickelt. Es wird geschätzt, dass zwischen 85 und 86 Prozent aller heute verwendeten Mobiltelefone eine Version des Betriebssystems Android installiert haben. Dies macht Android zum mit Abstand meist verwendeten Betriebssystem für Mobiltelefone weltweit. Der Entscheid, die Studnetz-Applikation für Android Geräte zu entwickeln ist dadurch begründet, dass so die grosse Mehrheit der potentiellen Kunden/Kundinnen erreicht werden kann. [7]

Android basiert auf einem stark modifizierten Linux Kernel. Programme werden darauf in einer sogenannten *Android Runtime* Umgebung ausgeführt. Diese Umgebung ist in der Lage, Bytecodes im *Dex-Format* (Dalvik Executable-Format) auszuführen. Diese Dex-Formate werden dabei meist aus für die *Java Virtual Machine* (JVM) kompilierten Bytecodes übersetzt. Die meisten Applikationen für Android wurden deshalb lange in der Programmiersprache Java programmiert, erst vor wenigen Jahren begann ein langsamer Umschwung zur neueren Programmiersprache *Kotlin*, die jedoch auch in Bytecodes für die JVM kompiliert wird. Kotlin gewinnt haupt-

sächlich wegen seiner konzeptionell schönen Syntax an immer grösserer Beliebtheit, wobei insbesondere in der Entwicklung von Applikationen immer mehr auf Kotlin anstelle von Java gesetzt wird. [24]

5.1.2 Java

Für die Entwicklung der Studnetz Applikation wurde die Programmiersprache Java gewählt, da sie sehr gut dokumentiert ist und ich, als Entwickler, bereits vor dieser Arbeit Erfahrungen mit Java gesammelt hatte. Java ist eine Sprache der 3. Generation und gehört zu den objektorientierten Programmiersprachen. Sie wurde erstmals 1995 von Sun Microsystems veröffentlicht und ist seit 2010 in Besitz von Oracle. Java zeichnet sich besonders durch seine Plattformunabhängigkeit aus und eignet sich daher gut für kleinere Applikationen, die auf vielen verschiedenen Geräten funktionieren sollen.

In der entwickelten Applikation wird die Sprache Java hauptsächlich für die logischen Prozesse verwendet, die hinter der optischen Benutzeroberfläche ablaufen.

5.1.3 XML

Für das Layout der Benutzeroberfläche des Clients wird für gewöhnlich nicht Java verwendet. Stattdessen weicht man auf die Auszeichnungssprache *XML* aus. XML steht für *Extensible Markup Language* und kann bis zu einem gewissen Grad mit HTML (*Hypertext Markup Language*) verglichen werden, unterscheiden sich jedoch in einem zentralen Punkt. HTML wurde entwickelt, um zu bestimmen wie Informationen dargestellt werden. XML auf der anderen Seite setzt den Fokus auf die Informationen als solche selber, und weniger auf die Darstellung. Es ist jedoch trotzdem möglich, auch XML für die Definition von Darstellungen zu verwenden, wie es beispielsweise bei Android Applikationen üblich ist. Weiter ist XML deutlich flexibler als HTML, da Entwickler/Entwicklerinnen mehr Möglichkeiten haben, eigene, neue Darstellungsformen zu definieren.

Bei der Verwendung von XML in Android Applikationen wird XML verwendet, um zum einen die Layouts der darzustellenden Screen zu definieren, aber auch um statische Informationen wie beispielsweise zu verwendende Strings und Zahlen zu beschreiben. Somit können in einer Applikation auch relativ einfach verschiedene Sprachpakete integriert werden, da dafür lediglich alle definierten Strings, welche sich in einer Datei befinden, übersetzt werden müssen. Alle XML-Dateien zusammen bilden die *Resources* einer Android Applikation. Diese Resources sind für die Java Klassen zugänglich

und können durch diese manipuliert werden. Dies ermöglicht eine direkte Verknüpfung von Java mit dem Layout und ermöglicht die Entwicklung dynamischer Screens und Darstellungen. [10] [33]

5.1.4 Android Studio

Die Entwicklung der Studnetz Applikation fand hauptsächlich innerhalb der Programmierumgebung *Android Studio* statt. Android Studio ist eine von Google entwickelte Programmierumgebung (IDE) für die Entwicklung von Applikationen für Android Mobilgeräte und unterstützt alle dafür erforderlichen Sprachen. Die Umgebung bietet verschiedenste hilfreiche Werkzeuge für die Entwicklung von Applikationen zu. Dazu gehört beispielsweise ein Visueller Layout Editor, wo die Layouts sehr intuitiv via 'Drag and Drop' gestaltet werden können, wobei der dazu gehörige XML-Code im Hintergrund automatisch generiert wird. Zudem ist es möglich, entwickelte Applikationen sehr einfach auf lokalen Emulatoren zu testen, wobei in der Konsole von Android Studio dann allfällige Fehlermeldungen angezeigt werden. All dies und noch vieles mehr macht Android Studio zu einer sehr starken Programmierumgebung. Das Herunterladen und die Benutzung von Android Studio ist kostenlos. Die Wahl auf die Entwicklungsumgebung für die in dieser Arbeit entwickelten Applikation war sehr schnell gefällt, da es kaum vergleichbare Alternativen zu Android Studio gibt, zumindest was die Entwicklung von Android Applikationen anbelangt.

5.2 Klassenübersicht des entwickelten Clients

Im folgenden Abschnitt wird auf den entwickelten Client auf einer technischen Ebene eingegangen. Es ist an dieser Stelle anzumerken, dass für diesen Abschnitt gewisse Grundkenntnisse des objektorientierten Programmierens und der Programmiersprache Java vorausgesetzt werden. Weiter ist anzufügen, dass die ab hier verwendeten Zeilenangaben innerhalb der Code Beispiele sich jeweils ausschliesslich auf die abgebildeten Ausschnitte des Codes beziehen und nicht die eigentlichen Zeilen im Sourcecode repräsentieren. Ebenfalls sind teilweise Stellen aus dem abgebildeten Sourcecode weggelassen worden. Diese Stellen sind dann durch das Symbol [...] in der Abbildung gekennzeichnet.

5.2.1 Activities

Der Begriff *Activities* ist Android spezifisch und beschreibt einen einzelnen Screen innerhalb einer Applikation. Innerhalb der Studnetz-Applikation werden sechs solche Activities verwendet. Zu jeder Activity gehört jeweils ein XML-Layout und eine Java-Klasse. Das XML-Layout bildet dabei die View-Komponenten, während die Java-Klasse die Presenter-Komponente bildet (siehe Kapitel 2.2). Auf die Struktur einer solchen XML-Layout-Datei wird hier nicht weiter eingegangen. Screenshots der fertigen Layouts finden sich in Kapitel 3, wie die dazugehörigen XML-Dateien angesehen werden können wird in Kapitel 6.1 beschrieben. Stattdessen wird der Fokus auf die Java-Klasse gelegt. Ein beispielhaftes Grundgerüst einer solchen Activity findet sich in Abbildung 5.1.

```
1 public class ActivityName extends AppCompatActivity {  
2  
3     @Override  
4     protected void onCreate(Bundle savedInstanceState) {  
5         super.onCreate(savedInstanceState);  
6         setContentView(R.layout.ActivityLayoutName);  
7  
8         [...]  
9     }  
10  
11    @Override  
12    protected void onStart() {  
13        super.onStart();  
14  
15        [...]  
16    }  
17  
18    @Override  
19    public boolean onOptionsItemSelected(MenuItem item) {  
20  
21        [...]  
22        return super.onOptionsItemSelected(item);  
23    }  
24}
```

Abbildung 5.1: Grundgerüst einer Activity

Alle dargestellten Funktionen sind optional und müssen deshalb nicht zwangsweise in einer Activity vorkommen. Sie werden jedoch in der Studnetz-Applikation so gut wie immer benutzt.

Die onCreate-Methode Die *onCreate*-Methode in den Zeilen drei bis acht ist die Kernfunktion einer jeden Activity. Sie ist die erste Funktion die in einem Lebenszyklus einer Activity ausgeführt wird und definiert das zu verwendende XML-Layout (Zeile 7). Anschliessend folgen meist die Instantiierungen der einzelnen, für die Activity relevanten Objekte. Dazu gehören zum einen rein Java seitige Objekte wie zum Beispiel ein Objekt der Klasse *UserModel* (siehe Kapitel 5.2.3) aber es werden auch bestimmten Komponenten der View wie beispielsweise Schaltflächen passende Listener-Klassen zugewiesen. Diese Objekte erlauben es, die dazugehörige View zu manipulieren und ihnen neue Eigenschaften zuzusprechen.

Die onStart-Methode Die *onStart*-Methode in Zeile 12 bis 15 ist die zweite Methode, die im Lebenszyklus einer Activity ausgeführt wird. Zum Zeitpunkt, bei welcher sie ausgeführt wird, ist der Screen für den Benutzer/die Benutzerin bereits sichtbar. In ihr werden meist diese Prozesse ausgeführt, die ansonsten zu grösseren Verzögerungen beim Laden einer Activity führen würden. Ein Beispiel hierfür wäre das Laden eines Profilbildes oder andere mit dem Server verknüpfte Prozesse.

Die onOptionsItemSelected-Methode Die *onOptionsItemSelected*-Methode in den Zeilen von 19 bis 23 wird dann ausgeführt, wenn eine Schaltfläche in der Toolbar eines Screens betätigt wurde. In ihr wird dann jeweils definiert, was bei der Betätigung einer Schaltfläche geschehen soll. Dies kann zum Beispiel ein Wechsel zu einer neuen Activity sein oder das Ändern zu einem anderen Fragment.

5.2.2 Fragments

Fragments sind ebenfalls Android spezifische Elemente und sind mit Activities vergleichbar. Gleich wie Activities bestehen sie sowohl aus einer Java-Presenter wie einer XML-View. Der grosse Unterschied ist jedoch, das Fragments im Gegensatz zu Activities keine eigenständige Screens sind, sondern jeweils innerhalb einer Activity eingebettet werden. Sie besitzen dann dort einen vorgegebenen Rahmen in welchem ihr Layout angezeigt wird. Die zum Layout gehörenden Methoden finden sich in der dazugehörigen Java-Klasse.

Eine Abbildung eines Grundgerüstes eines solchen Fragments findet sich in Abbildung 5.2.

```
1 public class FragmentName extends Fragment {  
2  
3     @Override  
4     public void onAttach(Context context) {  
5         super.onAttach(context);  
6         this.mActivity = (ParentActivityName) context;  
7  
8         [...]  
9     }  
10  
11    @Override  
12    public View onCreateView(LayoutInflater inflater,  
13        ViewGroup container, Bundle savedInstanceState) {  
14  
15        View view =  
16            inflater.inflate(R.layout.FragmentLayoutName,  
17            container, false);  
18  
19        [...]  
20  
21        return view;  
22    }  
23  
24    @Override  
25    public void onStart() {  
26        super.onStart();  
27    }  
28  
29    @Override  
30    public boolean onOptionsItemSelected(MenuItem item) {  
31  
32        [...]  
33        return super.onOptionsItemSelected(item);  
34    }  
35}
```

Abbildung 5.2: Grundgerüst eines Fragments

Die onAttach-Methode Die *onAttach*-Methode (Zeile 3 bis 10) ist die erste Funktion im Lebenszyklus eines Fragments. In der entwickelten Applikation wird sie verwendet, um dem Fragment eine Referenz zur momentanen Applikationsumgebung (*Context*) zu geben, in welcher sich die Parent-Activity befindet (Zeile 6). Dieses Objekt wird für diverse Prozesse benötigt, die auf Ressourcen der Applikation zugreifen müssen oder anderweitig mit dem Gerät interagieren. [4]

Die onCreateView-Methode Die *onCreateView*-Methode in den Zeilen 13 bis 20 ist nach der onAttach- und der onCreate-Methode die dritte Methode im Lebenszyklus eines Fragments. Die onCreateView-Methode wird in den Fragments der Studnetz selten verwendet, weshalb sie hier nicht selber aufgeführt ist. Stattdessen übernimmt die onCreateView-Methode die Aufgabe des Definierens der View (Layout) und der wichtigen Objekte. Dabei wird zuerst in Zeile 15 ein neues Layout der Klasse *View* instantiiert. Über dieses Objekt kann dann auf die einzelnen View-Komponenten des Layouts zugegriffen werden. Schlussendlich wird die fertig konfigurierte View in Zeile 19 zurückgegeben, wo sie dann für das Fragment als Layout verwendet wird.

Die onStart-Methode Die *onStart*-Methode eines Fragments ist direkt vergleichbar mit der onStart-Methode einer Activity. Gleich wie in einer Activity wird die onStart-Methode ausgeführt, wenn das Layout des Fragments bereits sichtbar ist. Auch sie wird für vom Server abhängige Prozesse verwendet, bei welchen mit Verzögerungen zu rechnen ist.

Die onOptionsItemSelected-Methode Auch bei der *onOptionsItemSelected*-Methode kann auf die äquivalente Methode der Activities in Kapitel 5.2.1 verwiesen werden, da die Hauptaufgabe der Methode das Definieren der Aktionen in der Toolbar ist.

5.2.3 Models

Der Begriff *Models* beschreibt eine Reihe von Klassen, die hauptsächlich für das strukturierte Speichern von Informationen innerhalb des Clients verwendet werden und bilden somit die Model-Komponenten des Systems (siehe Kapitel 2.2). Die Models der Studnetz Applikation können generell vom Aufbau her in vier Abschnitte unterteilt werden:

- Die *Felder* eines Models: Die Felder, auch *Member Variables* genannt,

sind der Datenspeicher eines Models. Sie werden meist als erstes innerhalb der Klasse in Forme einer Reihe von Variablen definiert. [23]

- Die *Konstruktoren*: Die Konstruktoren befinden sich meist an zweiter Stelle. Über sie wird das Model initialisiert. Dabei kann es durchaus mehrere verschiedene Konstruktoren innerhalb eines Models geben. Ihnen werden meist über die Parameter Daten mitgegeben, die für die Felder benötigt werden.
- Die *Getter-* und die *Setter*-Methoden: Die Getter- und die Setter-Methoden finden sich als Letzte in einem Model und sind die Methoden, die anderen Klassen den Zugriff auf die Felder des Models erlauben. Dabei können über die Setter-Methoden, auch *Mutator Methods* genannt, die Felder nachträglich verändert werden, während über die Getter-Methoden, auch *Accessor Methods* genannt, der Inhalt eines bestimmten Feldes abgerufen werden kann. [22]

5.2.3.1 Das UserModel

Das UserModel-Model ist das Model, in welchem die Informationen eines einzelnen Benutzerprofils gespeichert werden. Hierzu findet sich ein Feld für alle dafür relevanten Werte. Bei der Initialisierung wird unterschieden, ob das Model das Model des Clientbenutzers/der Clientbenutzerin selber ist oder ob es sich um ein fremdes Profil handelt. Entsprechend sind Informationen wie Passwort oder Email nicht immer innerhalb des Models gesetzt. Eine Abbildung eines Konstruktors des Models findet sich in Abbildung 5.3.

```

1
2
3  public userInfo(int id, String username, String name, String
   ↳ firstname, [...], String temp_profilepicture_path, String
   ↳ JSON) {
4      this.id = id;
5      this.JSON = JSON;
6
7      this.username = username;
8      this.name = name;
9      this.firstname = firstname;
10     this.school = school;
11     this.grade = grade
12     this.description = description;

```

```

13
14     this.french = french;
15     this.spanish = spanish;
16     this.english = english;
17     this.music = music;
18     this.chemistry = chemistry;
19     this.biology = biology;
20     this.maths = maths;
21     this.physics = physics;
22     this.german = german;
23
24     this.passwordHash = passwordHash;
25     this.salt = salt;
26
27     this.temp_profilepicture_path = temp_profilepicture_path;
28 }

```

Abbildung 5.3: Konstruktor der UserModel-Klasse

Das UserModel ist das zentralste Model der entwickelten Applikation. Es wird in allen Activities und Fragmenten mit Ausnahme des Logins und der Registrierung verwendet. Da jedoch das UserModel-Objekt selber nicht zwischen den Activities weitergegeben werden kann, wird hierzu ein JSON-String verwendet, der im Konstruktor auf Zeile 3 gesetzt wird. Der dabei verwendete JSON-String ist der gleiche, der der Ergänzungsfach-Server verwendet, wenn er Benutzerinformationen an den Client schickt.

5.2.3.2 Das ProfilePictureModel

Für das Darstellen und Speichern von Profilbild auf dem Client wird das ProfilePictureModel-Model verwendet. Das Model besitzt Felder für die drei verschiedenen Formate, in welchen ein Profilbild innerhalb der Applikation auftreten kann.

- Das *Bitmap*-Format: Das Bitmap-Format ist das Format, welches von Android für die Darstellung von Bildern verwendet wird. Es wird in der Applikation verwendet, um das anzuzeigende Bild der *ImageViews* (XML-Objekt für das Darstellen von Bildern) zu definieren.
- Das *Base64*-Format: Base64 ist das Bild-Format welches für den Transfer einer Bilddatei zwischen Server und Client verwendet wird. Wie be-

reits in Kapitel 4.4.2.1 erwähnt ist es ebenfalls das Format, in welchem die Bilder in der Datenbank gespeichert werden.

- Die temporäre *Cache*-Datei: Die temporäre Cache-Datei ist eine Datei im lokalen Cache-Speicher des Gerätes. Cache-Speicher wird verwendet um temporär benötigte Dateien zu speichern, damit sie nicht immer wieder erneut geladen werden müssen. Jede Applikation auf einem Gerät besitzt dabei ihr eigenes Cache-Verzeichnis. Innerhalb der entwickelten Applikation werden jeweils für geladene Profilbilder Cache-Dateien erstellt. Ab dann werden die Bilder jeweils nur noch über den Pfad zur Cache-Datei referenziert. Dies macht einen einfachen Transfer der Bilder von Activity zu Activity möglich. [2]

Das ProfilePictureModel-Modell kann dabei über jedes der drei Formate initialisiert werden, weshalb es jeweils einen Konstruktor für jedes Format besitzt. Anschliessend wird das gegebene Format in die anderen Formate übersetzt. Hierzu finden sich eine Reihe von Methoden in der ProfilePictureModel-Klasse. Hinzu kommt ein vierter Konstruktor, der das Objekt mithilfe eines Parameters der Klasse *Uri* initialisiert. Uri (Uniform Resource Identifier) ist eine Form der Dateireferenzierung. Sie wird bei der Auswahl eines neuen Profilbildes verwendet und kann ebenfalls in die anderen Formate umgewandelt werden. Eine Abbildung der Felder und der Konstruktoren findet sich in Abbildung 5.4.

```

1  public class ProfilePictureModel {
2
3      private static final int MAX_QUALITY = 100;
4
5      private boolean success = false;
6      private Context mContext;
7      private Bitmap imageBitmap;
8      private String BASE64, path;
9      private File tempFile;
10
11     public ProfilePictureModel(Context context, String
12         ↳ BASE64) {
13         this.mContext = context;
14         this.BASE64 = BASE64;
15         this.imageBitmap = decodeBASE64(this.BASE64);
16         ↳ //Methode für das Umwandeln von Base64 zu Bitmap

```

```

15         createTemp(this.imageBitmap, MAX_QUALITY);
16         ↪ //Erstellen einer neuen Bild-Datei im Cache
17     this.success = true;
18 }
19
20 public ProfilePictureModel(Context context, Bitmap
21   ↪ imageBitmap) {
22   [...]
23 }
24
25 public ProfilePictureModel(Context context, Uri imageUri,
26   ↪ int quality) {
27   [...]
28 }
29
30 [...]
31 }
32 }
```

Abbildung 5.4: Felder und Konstruktoren der ProfilePictureModel-Klasse

Eine weitere Aufgabe des ProfilePictureModel-Models ist die Regulierung der Bildqualität und Auflösung. Dies ermöglicht, dass egal was für ein Bild ein Benutzer/Benutzer als Profilbild ausgewählt hat, alle Bilddateien ungefähr gleich gross sind und der gleichen Auflösung entsprechen. Zudem werden die Bitmaps innerhalb der Applikation jeweils in einer runden Form dargestellt. Diese wird ihnen innerhalb des ProfilePictureModel-Models gegeben, wobei hierzu jedoch auf ein bereits existierendes Github-Repository von Arthur Teplitzki [29] zurückgegriffen wurde, welches verschiedene Tools für das Zuschneiden von Bildern bietet. Dieses Repository wird in Kapitel 5.2.6 im Detail thematisiert.

5.2.3.3 Das ChatModel

Das *ChatModel*-Model ist das simpelste Model der Studnetz Applikation und wird für die Chats verwendet. Im Model befinden sich Felder für die

wichtigsten Kernangaben eines Chats wie die UserModel-Models von beiden Chatparteien, eine Firebase-Referenz zu den Firebase-Profilen beider Chatparteien sowie die Firebase-Referenz zum Chat selber. Eine Abbildung des einzigen Konstruktors sowie den Felder findet sich in Abbildung 5.5.

```
1 public class ChatModel {  
2  
3     private UserModel mMainprofileModel, mUserprofileModel;  
4     private DatabaseReference mMainprofileRef,  
5         mUserprofileRef, mChatRef;  
6  
7     public ChatModel(UserModel mMainprofileModel, UserModel  
8         mUserprofileModel, DatabaseReference mMainprofileRef,  
9         DatabaseReference mUserprofileRef, DatabaseReference  
10        mChatRef) {  
11         this.mMainprofileModel = mMainprofileModel;  
12         this.mUserprofileModel = mUserprofileModel;  
13         this.mMainprofileRef = mMainprofileRef;  
14         this.mUserprofileRef = mUserprofileRef;  
15         this.mChatRef = mChatRef;  
16     }  
17 }  
18 }
```

Abbildung 5.5: Felder und Konstruktor der ChatModel-Klasse

5.2.4 Auflistungen mithilfe von RecyclerViews

In der Studnetz Applikation wird an mehreren Orten eine Form einer Auflistung von Elementen benötigt. Dies ist beispielsweise bei der Anzeige der gefundenen Suchergebnissen der Fall oder wenn die offenen Chats aufgelistet werden. Für das Darstellen einer solchen Auflistung werden gleich mehrere verschiedene Klassen benötigt.

5.2.4.1 Das XML-Layout eines Elementes

Standardmäßig verwenden Listen für die einzelnen Elemente ein extrem simples Layout für die einzelnen Elemente. Oft reicht dies jedoch nicht aus sondern die Elemente sollen angepasster an ihre spezifische Aufgabe sein. Hierzu wird ein XML-Layout definiert, welches jeweils für ein Element der Liste verwendet werden soll. Ein Beispiel eines solchen Elementes findet sich in Abbildung 5.6.



Abbildung 5.6: Einzelnes Element wie es bei der Auflistung der Suchergebnisse verwendet wird

5.2.4.2 Das Model eines Elements

Um den verschiedenen Elementen der Auflistung Informationen zuzuschreiben zu können, kommen Models zum Einsatz. Jedem Element wird ein Model zugeschrieben, welches die für das Element wichtigen Informationen beinhaltet. Wie solche Models aufgebaut sind wurde bereits in Kapitel 5.2.3 beschrieben, weshalb hier nicht näher darauf eingegangen wird.

5.2.4.3 Die ViewHolder-Klassen

ViewHolder-Klassen beschreiben jeweils ein einzelnes Element der Auflistung. Innerhalb eines ViewHolders sind Dinge wie das zu verwendende Layout und das zum Element gehörende Model referenziert. Es ist teilweise umstritten, wie weit der Aufgabenbereich der ViewHolder-Klasse einer Liste gehen soll. In der entwickelten Applikation hat der ViewHolder jeweils neben den bereits erwähnten Aufgaben auch die Aufgabe, das XML-Layout des Elementes auf die Informationen des Models anzupassen. Hierzu findet sich meist eine *validate*-Methode, welche diese Modifikation der View ermöglicht. Ein Beispiel für eine solche ViewHolder-Klasse findet sich in Abbildung 5.7.

```

1 class ResultViewHolder extends RecyclerView.ViewHolder{
2
3     private UserModel model;
4     private View view;
5
6     public ResultViewHolder(View view) {
7         super(view);
8         this.view = view;
9     }

```

```

10
11     public void validate(UserModel model) {
12         this.model = model
13
14         TextView name_tv =
15             ↵ view.findViewById(R.id.result_name_textview);
16         TextView school_tv =
17             ↵ view.findViewById(R.id.result_school_textview);
18
19         [...]
20
21         name_tv.setText(this.model.getFirstname() + " " +
22             ↵ this.model.getName());
23         school_tv.setText(this.model.getSchool() +
24             ↵ this.model.getStringGrade());
25     }
26
27     [...]
28
29 }

```

Abbildung 5.7: ViewHolder-Klasse für die Auflistung der Suchergebnisse

5.2.4.4 Die Adapter-Klassen

Adapter-Klassen haben die Aufgabe, die Elemente und somit auch die ViewHolder-Objekte einer Liste zu verwalten. Sie bestimmen, welcher Datensatz welchem Element zukommt und welche View für ein Element verwendet werden soll. In der entwickelten Applikation wird hierzu für die Auflistung der Suchergebnisse und der offenen Chats eine Subklasse der *RecyclerView.Adapter*-Klasse verwendet, während für die Chats selber eine Subklasse der *FirebaseRecyclerAdapter*-Klasse verwendet wird. Grundsätzlich ist der Aufbau beider Adapterarten vergleichbar. Der grosse Unterschied ist jedoch, dass *FirebaseRecyclerAdapter* über eine Referenz mit einer Firebase Echtzeitdatenbank verknüpft sind. Der Datensatz der Auflistung wird jeweils in Echtzeit mit der Datenbank synchronisiert. Auf den *FirebaseRecyclerAdapter* soll hier nicht weiter eingegangen werden. Stattdessen wird der Fokus auf den normalen *RecyclerAdapter* gelegt. Ein Beispiel eines solchen Adapters findet sich in Abbildung 5.8.

```

1  class ChatoverviewAdapter extends
2      RecyclerView.Adapter<OpenChatViewHolder>{
3
4      private ChatoverviewFragment mFragment;
5      private Map<Integer, OpenChatModel> mDataset;
6
7      public ChatoverviewAdapter(ChatoverviewFragment
8          ↪ mFragment) {
9          this.mFragment = mFragment;
10         this.mDataset = mFragment.getDataset();
11     }
12
13     @Override
14     public OpenChatViewHolder onCreateViewHolder(ViewGroup
15         ↪ parent, int viewType) {
16         View view =
17             ↪ LayoutInflater.from(parent.getContext()).inflate(R.layout.openchat,
18                 ↪ parent, false);
19         OpenChatViewHolder viewHolder = new
20             ↪ OpenChatViewHolder(view);
21         return viewHolder;
22     }
23
24     @Override
25     public void onBindViewHolder(OpenChatViewHolder holder,
26         ↪ int position) {
27         OpenChatModel model =
28             ↪ mDataset.get(mDataset.keySet().toArray()[position]);
29         holder.validate(model);
30         holder.getView().setOnClickListener(new
31             ↪ OnOpenChatListener(mFragment, model));
32         holder.setProfilePicture(new
33             ↪ ProfilePictureModel(mFragment.getActivity().getBaseContext(),
34                 ↪ new
35                 ↪ File(model.getUserModel().getTempProfilePicturePath())));
36     }
37
38     @Override
39     public int getItemCount() {
40         return mDataset.size();
41     }

```

```

29      }
30  }
```

Abbildung 5.8: RecyclerAdapter-Klasse für die Auflistung der offenen Chats

Typisch für einen Adapter ist ein Feld für den darzustellenden Datensatz, welcher sich hier auf Zeile 4 in Form einer Map mit einem Integer als Key und einem OpenChatModel-Model als Wert findet.

Das OpenChatModel-Model ist wiederum ein sehr simples Model, welches nur für diese Auflistung verwendet wird. Es beinhaltet Felder für die benötigten Firebase-Referenzen und das Profil des Chatpartners in Form eines UserModel-Models. Zudem kommt noch einen String für die zuletzt im Chat geschickte Nachricht.

Der Datensatz des Adapters wird jeweils bereits bei der Initialisierung des Adapters über die Parameter des Konstruktors auf Zeile 6 referenziert. Hier ist es wichtig zu unterscheiden, dass es sich um eine Referenz und nicht um einen Klon des Datensatzes handelt. Dies bedeutet, dass wenn sich der Datensatz ändern sollte, die Liste über eine notifyDataSetChanged-Methode aktualisiert werden kann, ohne dass der Datensatz dabei erneut übergeben werden muss.

In den Zeilen 12 bis 24 finden sich die beiden Kernmethoden eines RecyclerAdapters.

Die onCreateViewHolder-Methode Die *onCreateViewHolder*-Methode hat die Aufgabe, das Layout eines einzelnen Elements zu laden (Zeile 13) und ein neues ViewHolder-Objekt zu instantiiieren (Zeile 14). Diese Methode wird jeweils so oft aufgerufen, wie Elemente auf dem Bildschirm des Gerätes maximal sichtbar sein können und geschieht deshalb gleich beim Start der Auflistung. Zu diesem Zeitpunkt unterscheiden sich die verschiedenen Elemente jedoch noch nicht und sehen alle gleich aus. Die Methode wird nach dem Start der Auflistung nicht mehr aufgerufen, selbst wenn die Liste durchgesehen wird und neue Elemente sichtbar werden müssen.

Die onBindViewHolder-Methode Die *onBindViewHolder*-Methode wird jeweils für jeden ViewHolder der sichtbaren Elemente aufgerufen. Dabei wird den einzelnen ViewHolder-Objekten ihr Model über die *validate*-Methode zugewiesen (Zeile 21). Das zugewiesene Model wird dabei über die Position des Elementes innerhalb der RecyclerView bestimmt (Zeile 20). Im Falle der ChatOverviewAdapter-Klasse findet sich hier noch eine weitere

Methode der ViewHolder-Klasse, die das Definieren eines zu verwendenden Profilbildes ermöglicht (Zeile 22).

Die onBindViewHolder-Methode wird nach dem Start der View aufgerufen, wenn ein Element vom Bildschirm verschwindet und ein neues erscheinen muss. Es wird dann der ViewHolder des verschwundenen Elements genommen und sein Model wird über die validate-Methode durch ein neues Model ersetzt. Damit kann erreicht werden, dass mit einer nur geringen Anzahl an gleichzeitig geladener Layouts trotzdem beliebig grosse Datensätze effizient dargestellt werden können. Dieser Prozess der Wiederverwertung von nicht mehr sichtbarer Elementen wird als *recycling* bezeichnet, was auch den Namen der beiden Superklassen RecyclerView.ViewHolder und RecyclerView.Adapter erklärt.

5.2.5 Utility-Klassen

5.2.5.1 Die JSONtoInfo-Klasse

Die *JSONtoInfo*-Klasse ist eine in der Applikation sehr oft verwendete Klasse. Sie ist in der Lage über eine *createNewItem*-Methode aus einem JSON-Objekt, wie es vom Server erhalten wird, ein UserModel-Model zu instantiiieren. Diese Datenkonversion ist von grossem Vorteil, da UserModel-Objekte nicht zwischen Activities weitergegeben werden können, jedoch der String eines JSON-Objektes schon. Somit wird zu Beginn einer Activity jeweils aus den Extras der String des JSON-Objektes entzogen, dieser zu einem JSON-Objekt gemacht und dieses wiederum zu einem UserModel-Objekt konvertiert. Ein Beispiel für eine solche Anwendung wird in Abbildung 5.9 dargestellt.

```
1 mainprofileModel = new
  ↳ JSONtoInfo(getApplicationContext()).createNewItem(new
  ↳ JSONObject(extras.getString("clientInfo")));
```

Abbildung 5.9: Instantiierung eines UserModel-Objektes aus den Extras

5.2.5.2 Die TempFileGenerator-Klasse

Die *TempFileGenerator*-Klasse wird verwendet, um aus einem Base64 String eines Bildes eine entsprechende Datei im Cache zu erstellen. Hierzu wird die *getTempFilePath*-Methode verwendet. Sollte der Base64 String jedoch dem Wert 0 entsprechen oder nicht gesetzt sein (null), wird ein vordefiniertes

Platzhalter Bild an Stelle einer Datei im Cache verwendet. Der Pfad zur Bilddatei wird dann jeweils über den Rückgabewert zurückgegeben.

5.2.5.3 Die ProfilePictureLoader-Klasse

Die *ProfilePictureLoader*-Klasse wird verwendet, um mehrere Profilbilder gleichzeitig zu Laden. Dies wird bei der Darstellung von Auflistungen benötigt, wo mehrere Profile mit verschiedenen Profilbildern gezeigt werden. Dabei werden jeweils beim Start der Auflistungen die Profilbilder der ersten 20 Datensatzeinträge mithilfe der ProfilePictureLoader-Klasse jeweils geladen. Dann werden wenn noch ungeladene Profile gezeigt werden über die ProfilePictureLoader-Klasse die nächsten 20 Profilbilder geladen, bis schlussendlich, alle Profilbilder geladen sind.

5.2.5.4 Die MyReader-Klasse und die SchoolMapper-Klasse

Diese beide Klassen sind für die Auflistung der Schulen und der dazugehörigen Schulstufen verantwortlich. Dabei werden die von der Applikation unterstützten Schulen im einen Spinner angezeigt. Wenn eine Auswahl getroffen wurde, werden die Schulstufen der ausgewählten Schule im zweiten Spinner angezeigt. Die unterstützten Schulen sind dabei in einem Textdokument gespeichert und mit einer Zahl versehen, die die Schulstufen der Schule bestimmt. Es ist die MyReader-Klasse, die dieses Dokument zu lesen vermag und anschliessend die gelesenen Zeile als Liste zurückgibt. Die SchoolMapper-Klasse übernimmt das Handling der beiden Spinner. Eine Abbildung der SchoolMapper-Klasse findet sich in Abbildung 5.10.

```
1 public class SchoolMapper {  
2  
3     private Spinner schoolSpinner, gradeSpinner;  
4     Map<String, Integer> mMap;  
5     private Context mContext;  
6  
7     public SchoolMapper(Context mContext, Spinner  
8             ↪ schoolSpinner, Spinner gradeSpinner) {  
9         this.mContext = mContext;  
10        this.schoolSpinner = schoolSpinner;  
11        this.gradeSpinner = gradeSpinner;  
12    }  
13  
14    public Map<String, Integer> map(List<String> mData) {
```

```

14         Map<String, Integer> mOut = new LinkedHashMap<>();
15
16     for(int n = 0; n < mData.size(); n++) {
17         String[] tempData = mData.get(n).split(":");
18         ↳ //Schulen sind jeweils durch ein ":" von
19         ↳ ihrerer Schulstufe getrennt, zum Beispiel:
20         ↳ "Kantonsschule Ausserschwyz:6"
21         mOut.put(tempData[0],
22             ↳ Integer.parseInt(tempData[1]));
23     }
24
25     return mOut;
26 }
27
28 public void startDisplay(String path){
29
30     MyReader mReader = new MyReader(mContext);
31
32     mMap = map(mReader.read(path));
33
34     String[] keySet = mMap.keySet().toArray(new
35     ↳ String[mMap.size()]);
36
37     ArrayList<String> mSchools = new ArrayList<>();
38
39     for(int n = 0; n < mMap.size(); n++) {
40         mSchools.add(keySet[n]);
41     }
42
43     ArrayAdapter<String> schoolAdapter = new
44     ↳ ArrayAdapter<String>([...], mSchools);
45     schoolAdapter.setDropDownViewResource(
46     ↳ android.R.layout.simple_spinner_dropdown_item);
47     schoolSpinner.setAdapter(schoolAdapter);
48     schoolSpinner.setOnItemSelectedListener(new
49     ↳ AdapterView.OnItemSelectedListener() { [...] })
50 }
51 }
```

Abbildung 5.10: Instantiierung eines UserModel-Objektes aus den Extras

Beim Initialisieren der Klasse werden die Referenzen zu den beiden Spinern über die Parameter des Konstruktors der Klasse überreicht (Zeile 7). Wenn die *startDisplay*-Methode (Zeile 25) aufgerufen wird, werden die beiden Spinner konfiguriert.

Hierzu wird zuerst über die *map*-Methode (Zeile 13) das Textdokument mithilfe der MyReader-Klasse gelesen und die gelesenen Schulen (Zeile 29) gemeinsam mit der zugehörigen Schulstufenzahl in eine *LinkedHashMap* gespeichert (Zeile 14). Eine LinkedHashMap ist fast identisch zu einer normalen HashMap, wo auch jeweils ein Key gemeinsam mit einem Wert gespeichert werden. Der einzige Unterschied ist, dass sich die Reihenfolge, in welcher die Elemente in die HashMap eingefügt worden sind, gespeichert wird. Diese HashMap kann dann als Datensatz für den Schulspinner verwendet werden (Zeile 39).

In den Schulspinner wird ein Listener implementiert, welcher bei der Auswahl einer Schule dann auch den Stufenspinner der ausgewählten Schule entsprechend konfiguriert (Zeile 43).

5.2.6 Profilbilder und *Image Cropping*

Kein Profil ist heutzutage ein richtiges Profil ohne ein passendes Profilbild. Ein Profilbild hilft den Benutzern/Benutzerinnen zum einen sich von einer möglichst guten Seite zu zeigen und zum anderen um schnell einen ersten Eindruck von anderen Benutzern/Benutzerinnen zu bekommen. Auch in der entwickelten Applikation ist es möglich, ein Profilbild zu bestimmen. Dabei ist es Benutzern/Benutzerinnen möglich, vom Speicher des Gerätes ein Bild auszuwählen und es auf das gewünschte Format zuzuschneiden oder direkt mit der Kamera ein neues Bild aufzunehmen. Für die Implementierung dieser Features wurde auf zwei öffentliche Github Repositories zurückgegriffen, da die eigene Programmierung dieser Funktionen den Rahmen dieser Arbeit gesprengt hätte. Die Lizenzen für beide Repositories finden sich im Anhang.

Das erste Repository läuft unter dem Namen *EasyImage* und ermöglicht ein einfaches Zugreifen und Auslesen von Bildern aus der Galerie, wie auch direkt von der Kamera [21]. Jedoch wären die Bilder an diesem Punkt einfach in der Grösse und Form, wie sie auf dem Gerät gerade gespeichert sind. Dies ist für ein Profilbild höchst ungeeignet. Hier kommt das zweite Repository ins Spiel. Dieses trägt den Namen *Android Image Cropper* und basiert auf dem 2013 erschienenen *Cropper* von Edmodo Inc. [29] [9]. Dieses Repository bringt eine Reihe an Funktionen für das Zuschneiden und

Anpassen von Bildern mit sich. Dieser Prozess des Zuschneidens wird auch *Cropping* genannt, was auch den Namen des Repositories erklärt. Wählt ein Benutzer/eine Benutzerin ein Bild aus, wird dieser Cropper gestartet. Ein Screenshot davon findet sich in Abbildung 5.11. Der Benutzer/Die Benutzerin kann daraufhin den Bereich des Bildes auswählen, welcher er/sie als Profilbild verwenden möchte. Danach kann die Auswahl bestätigt werden und das Bild wird daraufhin zugeschnitten. Dabei wird auch gleich die Qualität und Grösse des Bildes auf ein einheitliches Mass reduziert. So ist jedes Bild in der Datenbank ungefähr ähnlich gross. Zudem wird jedes Profilbild in einer zweiten, deutlich kleineren Version gespeichert. Dieses wird für die Suchergebnisse und die Aufgelisteten Chats verwendet, wo teilweise gleich 20 Bilder gleichzeitig geladen werden müssen. Um dabei Leistung zu sparen gibt es demnach von jedem Profilbild eine Version, die zwar sehr klein in Grösse und niedrig in Qualität ist, jedoch für die kleinen Icons bei der Suche komplett ausreicht. Beide Bilder werden in der Datenbank in der user_profilepictures Tabelle gespeichert (siehe auch Kapitel 4.4.2.1).

5.2.7 Interaktion mit dem Webserver (MySQL Datenbank)

Für Interaktionen mit dem Webserver, auf welchem sich auch die MySQL Datenbank befindet, wird auf die *Volley*-Bibliothek zurückgegriffen. Die *Volley*-Bibliothek bietet diverse Tools für das effiziente Arbeiten mit Netzwerken und eignet sich für das Kontrollieren der Kommunikation mit dem Webserver auf Seiten des Clients. Die Kommunikation mit dem Webserver läuft dabei in zwei Stufen ab: der Formulierung einer Anfrage und der Verarbeitung der Antwort.

5.2.7.1 Formulieren einer Anfrage

Die Request-Klassen Eine *Request*-Klasse definiert die Webadresse, an welche eine Anfrage gerichtet wird, sowie die dabei mitzugebenden Werte. Man kann sich als ein adressiertes Postpaket vorstellen, welches verschickt werden kann. Sie ist eine Subklasse der *StringRequest*-Klasse aus der *Volley Library*. Für jede Art der Anfrage muss jeweils eine eigene Request-Klasse erstellt werden, sie sind jedoch vom Aufbau her alle identisch. Ein Beispiel für eine solche Klasse findet sich in Abbildung 5.12.

```
1 public class BigProfilePictureRequest extends StringRequest {  
2  
3     private static final String URL = "http:// [...]  
   ↳ /profilepicture_big.php.php";
```

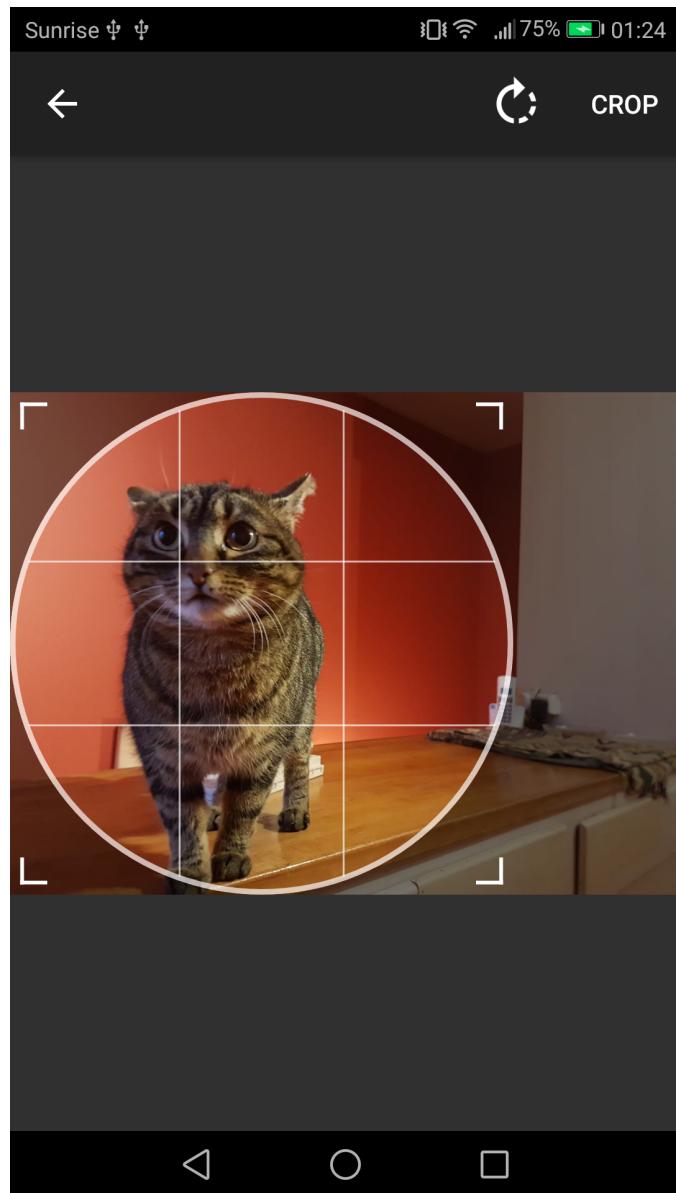


Abbildung 5.11: Screenshot des Croppers beim Zuschneiden

```

4     private Map<String, String> params;
5
6     public BigProfilePictureRequest(int id,
7         ↳ Response.Listener<String> listener) {
8         super(Method.POST, URL, listener, null);
9         params = new HashMap<>();
10        params.put("user_id", id + "");
11    }
12    [...]
13}
14

```

Abbildung 5.12: Request-Klasse für die Anfrage nach einem Profilbild eines Benutzers

Die Webadresse, an welche die Anfrage gehen soll, wird in einem der beiden Felder definiert (Zeile 3). Dabei wird angegeben, welches PHP-Skript aufgerufen werden soll. Das zweite Feld ist eine HashMap (Zeile 4). Die HashMap ist der Datencontainer für die zu übermittelnden Werte. Dabei sind im Beispiel der Key sowie der Wert der HashMap Strings. Die HashMap kann, nachdem sie erfolgreich an den Server übermittelt worden ist, verwendet werden, um die mitgegebenen Werte über die Keys wieder auszulesen.

Der Konstruktor benötigt im Beispiel nur zwei Parameter: eine user_id und ein Objekt der Klasse *Response.Listener<String>* (Zeile 6). Daraufhin wird die Superklasse aufgerufen und die Anfragemethode, die URL, der ResponseListener (siehe Kapitel 5.2.7.2) und ErrorListener als Parameter referenziert (Zeile 7). In der entwickelten Applikation wird immer die *POST*-Methode für die Kommunikation zwischen Webserver und Client verwendet. Dann wird die user_id als Wert in die HashMap eingefügt (Zeile 9).

Das Stellen der Anfrage Nachdem erfolgreich ein Objekt einer Request-Klasse instantiiert worden ist, kann die Request-Klasse über die sogenannte *RequestQueue* aus der Volley Library als Anfrage an den Webserver gesendet werden. Ein Beispiel findet sich in Abbildung 5.13

```

1 BigProfilePictureRequest request = new
  ↳ BigProfilePictureRequest(mUserModel.getId(), new
  ↳ OnBigPictureResponseListener(this));
  ↳ //Instantiiieren der Request-Klasse
2 RequestQueue queue = Volley.newRequestQueue(mActivity);
  ↳ //Instantiiieren einer neuen RequestQueue
3 queue.add(request); //Das Stellen der Request-Klasse als
  ↳ Anfrage an den Server

```

Abbildung 5.13: Request-Klasse für die Anfrage nach einem Profilbild eines Benutzers

5.2.7.2 Das Verarbeitung einer Antwort

OnResponseListener-Klasse Die *OnResponseListener*-Klasse ist einer Subklasse der Response.Listener<String>-Klasse der Volley-Bibliothek. Diese Listener-Klasse wird aufgerufen, wenn eine Antwort vom Server auf eine Anfrage empfangen wird. Die Antwort wird dabei in Form eines Strings übermittelt, der dann in ein JSON-Objekt umgewandelt werden kann. Ein Beispiel einer solchen Klasse findet sich in Abbildung 5.14.

```

1 class OnBigPictureResponseListener implements
  ↳ Response.Listener<String> {
2
3     private OpenprofileFragment mFragment;
4     private MainActivity mActivity;
5
6     [...] //Konstruktor
7
8     @Override
9     public void onResponse(String response) {
10
11         [...]
12
13         JSONObject jsn = new JSONObject(response);
14         boolean success = jsn.getBoolean("success");
15
16         if (success) {
17             [...]
18         }

```

```
19      [...]
20  }
21 }
```

Abbildung 5.14: OnBigPictureResponseListener-Klasse

Bei einer empfangenen Antwort wird als Erstes überprüft, ob die Aufgabe des PHP-Skriptes erfolgreich ausgeführt worden ist (Zeile 14 und 16). Je nachdem wird dann die Antwort weiter verarbeitet (Zeile 17) oder der Fehlschlag dem Benutzer/der Benutzerin mitgeteilt, sofern dies nötig ist (Zeile 19).

5.2.8 Interaktion mit der Firebase Echtzeitdatenbank

Die Interaktionen vom Client mit der Firebase Echtzeitdatenbank unterscheidet sich stark von der Interaktion mit dem Webserver. Anders als bei einem herkömmlichen Webserver ist das Implementieren von serverseitiger Datenverarbeitung in Form von Skripten sehr aufwändig und Firebase bietet hierzu kaum Tools. Deshalb wird bei der Arbeit mit Firebase Echtzeitdatenbanken oft sehr viel der Datenverarbeitung auf Seiten des Client gemacht. Dies bringt eine Menge potentieller Risiken in Bereichen Sicherheit mit sich, die jedoch im Rahmen dieser Arbeit in Kauf genommen worden sind.

5.2.8.1 Verbindung

Die Verbindung zum Server wird wie beim Webserver über eine Webadresse erstellt. Die Adresse wird dabei in einer separaten Datei definiert. In dieser Datei werden auch die gesamten für die Authentifizierung benötigten Werte gespeichert sind. Die Authentifizierung wird dabei von Firebase selber geregelt und läuft über ein Google Konto. Hierbei ist jedoch anzumerken, dass die Authentifizierung für die Studnetz-Applikation ausgeschaltet wurde. Sie müsste noch vor einer Veröffentlichung implementiert werden.

5.2.8.2 Zugriff vom Client auf die Echtzeitdatenbank

Innerhalb des Clients wird über sogenannte *Referenzen* auf den Server zugegriffen. Diese Referenzen sind im wesentlichen die Webadressen der verschiedenen JSON-Objekte. Über sie können neue Werte in die Datenbank eingefügt werden oder ausgelesen werden. Dieser Vorgang soll anhand zweier Beispiele demonstriert werden.

Einfügen neuer Werte in die Firebase Echtzeitdatenbank Das Einfügen neuer Werte in die Echtzeitdatenbank ist eher simpel. Ein Beispiel hierfür findet sich in Abbildung 5.15 wo ein Ausschnitt des Einfügens einer neuen Nachricht in einem Chat dargestellt ist.

```
1 final DatabaseReference newPost =
  ↵   mChatModel.getChatRef().child("Messages").push();
2 newPost.child("messageUser").setValue(messageUser);
```

Abbildung 5.15: Einfügen von neuen Werten in die Firebase Echtzeitdatenbank

Hierzu wird die zu verwendende Referenz definiert (Zeile 1). Über die *getChatRef*-Methode des ChatModel-Models kann die Referenz zum Chat selber geholt werden. Über die *child*-Methode wird zu einem JSON-Objekt mit dem Namen *Messages* navigiert. Die *push*-Methode bewirkt dann, dass ein neues JSON-Objekt mit einem zufällig generierten Namen erstellt wird. Diese nun konfigurierte Referenz kann verwendet werden, um neue Werte in die Datenbank einzufügen. Hierzu wird über die *child*-Methode ein neues JSON-Objekt erstellt, wo der einzufügende Wert über die *setValue*-Methode eingefügt werden kann (Zeile 2).

Auslesen von Werten aus der Firebase Echtzeitdatenbank Das Auslesen aus einer Firebase Echtzeitdatenbank ist im Gegensatz zum Einfügen neuer Werte deutlich komplexer. Jedoch bietet Firebase für häufige Verwendungen wie zum Beispiel das Darstellen von Listen schon bestehende Interfaces und Klassen an, die diese Aufgabe weitgehend übernehmen. In Abbildung 5.16 wird ein Code Ausschnitt dargestellt, der das Auslesen von Daten weitgehend selber bestimmt. Hierzu wird eine Klasse verwendet, in welche das *ValueEventListener*-Interface implementiert wird. Diese Klasse wird für die Darstellung der offenen Chats benötigt, wo eine genauere Kontrolle über diesen Ladevorgang erforderlich ist.

```
1 class ChatsValueEventListener implements ValueEventListener {
2
3     [...] //Konstruktor
4
5     @Override
6     public void onDataChange(DataSnapshot dataSnapshot) {
7         [...]
```

```
8      }
9
10     [...]
11 }
```

Abbildung 5.16: ValueEventListener-Klasse für das Auslesen von Daten aus der Firebase Echtzeitdatenbank

```
1 [Firebase Reference].addValueEventListener(new
  ↳ ChatsValueEventListener(this));
```

Abbildung 5.17: Zuweisung eines ValueEventListeners zu einer Referenz

Eine Klasse mit einem implementierten ValueEventListener-Interface kann jeweils einer oder mehreren Referenzen zugewiesen werden (siehe Abbildung 5.17). Hierzu wird innerhalb der Klasse bei der Zuweisung zu einer neuen Referenz und anschliessend immer wenn sich der Datensatz einer Referenz ändert die *onDataChange*-Methode aufgerufen (Abbildung 5.16, Zeile 6). Diese bekommt über ihre Parameter einen sogenannten *DataSnapshot*-Objekt der Referenz. Ein solches DataSnapshot-Objekt ist eine Momentaufnahme aller im JSON-Objekt einer Referenz gespeicherten Werte. Diese Werte können dann innerhalb der onDataChange-Methode ausgelesen und weiter verarbeitet werden.

Der grosse Vorteil dieser Art des Auslesens ist es, dass die Daten auf dem Client immer mit den Daten in der Firebase Echtzeitdatenbank synchronisiert werden.

Kapitel 6

Zusammenfassung und Diskussion

6.1 Herunterladen des Projektes und der Applikation

Das gesamte in dieser Arbeit beschriebene Projekt befindet sich auf GitHub. GitHub ist eine Plattform für Entwickler und Entwicklerinnen, wo einfach Projekte der Öffentlichkeit zugänglich gemacht werden können. Das gesamte Projekt kann über die Webadresse <https://github.com/florianmatura/Studnetz> aufgerufen werden. Dort finden sich drei Verzeichnisse.

- Das *AndroidStudioProject*-Verzeichnis enthält das Projekt für Android Studio. Das heruntergeladene Verzeichnis kann als Android Studio Projekt importiert werden (*File ->New Project ->Import Project*). Gradle (das von Android Studio verwendet *Build-Management-System*) sollte dann automatisch mit dem Build des Projektes beginnen. Ansonsten kann der Build auch über *Build ->Rebuild Project manuell* ausgeführt werden.
- Das *release*-Verzeichnis beinhaltet die Applikation selber. Die *Studnetz.apk* Datei kann auf Geräte mit einem installierten Android Betriebssystem mit einer Version ≥ 4.2 geladen und dann ausgeführt werden.
- Das *PHP_Files*-Verzeichnis beinhaltet alle Dateien, welche auf dem Webserver verwendet werden, sowie eine Text-Datei für die in der MySQL-Datenbank benötigen SQL-Tabellenstrukturen.

Das Projekt ist bereits verbunden mit einem Firebase-Server. Dieser läuft über ein Google Konto mit der Email hirtzflorianmatura@gmail.com. Sollte das Bedürfnis bestehen, Einsicht in diesen Server zu erlangen oder es sind anderweitige Probleme bezüglich der Einsicht in das Projekt aufgetreten, kann man mich gerne über dieselbe Email kontaktieren.

6.2 Verbesserungspotenzial

In dem Stand, in welchem sich die entwickelte Applikation zur Zeit befindet, ist noch nicht an eine Veröffentlichung zu denken. Mögliche Mängel und nötige nächste Schritte sollen in diesem Kapitel etwas genauer diskutiert und betrachtet werden.

Datenschutz

Damit eine Applikation heutzutage auf dem Markt angeboten werden darf, gelten eine Reihe von Gesetzen, die die Nutzer/Nutzerinnen einer Applikation schützen sollen. Dieses Thema wurde mit der neuen DSGVO (Datenschutz Grundverordnung) geregelt, welche in der EU seit 2018 gilt. Zwar gilt die DSGVO nicht für Schweizer und Schweizerinnen, jedoch für sämtliche EU-Bürger/EU-Bürgerinnen, deren Daten von der Applikation erfasst werden. Hinzu kommt, dass sogar in der Schweiz in naher Zukunft Anpassungen im Datenschutz Gesetz (DSG) zu erwarten sind. Damit die Applikation regelkonform wäre, fehlt es ihr noch an einer Reihe von Features wie zum Beispiel die Funktion, einen Account vollständig löschen zu können. Vor einem rechtlich abgesicherten Release müssten man sich noch vertiefter mit diesem Thema befassen und es müssten wahrscheinlich noch eine Reihe von Änderungen und Erweiterungen vorgenommen werden. [34]

Sicherheit

Die entwickelte Studnetz Applikation verfügt über sehr rudimentäre und teilweise stark unzureichende Sicherheitsvorkehrungen. Diese müssten vor einem Release unbedingt behoben werden. Dazu gehört die noch unverschlüsselte Kommunikation zwischen Server und Client, die veraltete Hashfunktion und die fehlende Authentifizierung beim Zugriff auf den Firebase-Server. Besonders bezüglich des Firebase-Servers ist in noch viel zu tun, was im Rahmen dieser Arbeit keinen Platz mehr gefunden hat. Dazu gehört nicht nur die fehlende Authentifizierung sondern auch das ausstatten der einzelnen Felder mit Datentypen und das markieren von Key-Feldern.

Betatesting und Kompatibilität

Die Applikation ist für einen Release noch unzureichend getestet. Dazu gehören sowohl Tests bezüglich ihrer Verwendung auf verschiedenen Geräten und Android Versionen wie auch Tests bezüglich ihrer Anwendbarkeit, wozu auch ein Betatest mit einer Gruppe von ausgewählten Benutzern gehören würde. Solche Betatests können durch Feedback von Benutzern einer Applikation helfen, da mit ihnen oft Fehler und Mängel entdeckt werden, die den Entwicklern/Entwicklerinnen ansonsten entgangen wären.

Benutzeroberfläche

Die Benutzeroberfläche ist das verbindende Glied zwischen Funktion und Benutzer/Benutzerin. Wenn die Benutzeroberfläche einer Applikation die Benutzer/die Benutzerinnen nicht anspricht, werden sie auch die Funktionen einer Applikation kaum benutzen. Deshalb sind gut durchdachte Benutzeroberflächen heutzutage eine mindestens so wichtige Komponente einer Applikation wie ihre Funktionalität.

Über die Qualität der Benutzeroberfläche der Studnetz Applikation lässt sich streiten. Sie ist definitiv nicht perfekt, erfüllt jedoch ihren Zweck. Aus meiner Sicht gibt es Teile der Applikation, welche ein visuelles Upgrade ertragen würden. Gewisse Screens wirken zu überfüllt und die Icons für die Fächer wirken in ihrer Farbe etwas unpassend.

6.3 Fazit

Das Ziel dieser Arbeit war es, eine Applikation zu entwickeln, die in ihren Grundelementen die Vermittlung von Nachhilfe unter Schülerinnen und Schülern vereinfachen soll. Dieses Ziel konnte zu einem grossen Teil erreicht werden. Die Applikation verfügt über alle dazu nötigen Features und bildet ein solides Fundament für die Entwicklung einer markttauglichen Version.

Weiter habe ich mir als Entwickler während dieser Arbeit sehr viel neues Wissen aneignen können. Ich habe viel gelernt über die Planung und Umsetzung von grösseren Programmierprojekten. Hinzu kommt das ganze technische Wissen, welches nötig gewesen ist, um dieses Projekt zu realisieren, welches ich mir nun erarbeitet habe.

Anhang A

Entwicklungsprozess

A.1 Vorbereitung

Bevor mit dem Programmieren der Applikation begonnen werden konnte, mussten eine Reihe von Vorbereitungen getroffen werden. Dazu gehörte in erster Linie eine klare Zielsetzung. Über diverse Skizzen und Ideensammlungen wurden die Anforderungen an die Applikation ermittelt und mögliche Lösungsansätze konzipiert. Die Planung belief sich dabei hauptsächlich auf die Applikation aus der Sicht eines Benutzers/einer Benutzerin. Am Ende waren die grundlegenden Features der Applikation definiert worden und es waren erste Skizzen über mögliche Layouts angefertigt worden.

Nun ging es darum, die nötigen Hilfsmittel für dieses Projekt zu ermitteln und aufzusetzen. Hierzu informierte ich mich hauptsächlich mithilfe von Online Blogs und Videos. Für die Entwicklung der Applikation wurde die Android Studio Umgebung und Notepad++ verwendet. Weitere verwendete Hilfsmittel waren FileZilla, Postman und Git.

A.2 Erstellen eines ersten Prototypen (Dezember - Januar)

Nachdem die grundlegenden Vorbereitungen getroffen waren, galt es, einen ersten Grundbaustein für die Applikation zu legen. Hierzu sollte eine erste Version eines Clients entwickelt werden, der in der Lage ist, mit einem Server zu kommunizieren.

Layouts Ich begann zuallererst die bereits gesammelten Layoutideen für den Client innerhalb der Entwicklungsumgebung umzusetzen. Dazu gehörte

vorerst ein Login-Screen, ein Registrierungs-Screen, ein Screen für die Accounteinstellungen und eine Hauptseite der Applikation, die nach dem Login erreicht werden sollte. Diese vier Layouts haben sich seither nur kaum verändert und werden auch in der fertigen Applikation noch verwendet. Diese Layouts gaben mir eine erste Struktur, an welcher ich mich orientieren konnte.

Webserver und Datenbank Für den Server verwendete ich zu Beginn dieser Arbeit einen kostenlosen Webserver von 000webhost mit einer darauf installierten MySQL Datenbank [17]. Später stieg ich jedoch von diesem Server um auf den Webserver des Ergänzungsfaches Informatik.

In der ersten Phase der Entwicklung entwarf ich ein sehr simples Konzept für die Datenbankstruktur, die vorerst nur auf das Login- und Registrierungs-Feature der Applikation ausgelegt war.

Erste Features Das erste grosse Ziel war es, Benutzern/Benutzerinnen das Erstellen eines Accounts zu ermöglichen, über welchen sie auf die Dienste der Applikation zugreifen können. Zu diesem Feature gehört eine Registrierungsfunktion, eine Loginfunktion und eine Bearbeitungsfunktion für die Accountdetails. Die drei zu diesem Zeitpunkt entwickelten Funktionen waren in ihrer Funktionsweise noch sehr einfach gehalten. Es ging vorerst hauptsächlich um die Funktionalität und das Sammeln von Erfahrungen anstelle von Perfektion. Es gelang mir, bis Ende Januar alle drei Funktionen erfolgreich zu implementieren. Die drei Funktionen mussten im späteren Verlauf der Arbeit zwar nochmals grundlegend überarbeitet werden, dienten mir jedoch als ein solider Grundstein auf welchem ich aufbauen konnte.

A.3 Ausbauen des Prototypen (Februar - April)

Nachdem ein grobes Fundament gelegt war, galt es die restlichen nötigen Features einzubauen. Dazu gehörte die Suchfunktion und ein Feature, über welches mit einem anderen Benutzer Kontakt aufgenommen werden konnte.

Suchfunktion Für die Entwicklung der Suchfunktion konzentrierte ich mich vorerst nur auf die Suche selber und nicht die Darstellung der Suchergebnisse. Hierzu begann ich mit einer Reihe von SQL Queries zu experimentieren, die mir dies ermöglichen sollten. Ich entwickelte einen Weg, wie ich eine Suche mithilfe von angegebenen Kriterien ermöglichen konnte. Die gefundenen Ergebnisse lagen anschliessend in Form eines Strings vor.

Im zweiten Schritt wurde dann eine Visualisierung für die Suchergebnisse entwickelt.

Optimieren des ursprünglichen Prototypen Während der Entwicklung der Suchfunktion stieß ich auf einen neuen, effizienteren Weg wie ich die Informationen einzelner Benutzer verwalten konnte. So entstand das UserInfo-Model. Ich beschloss es auch in die anderen Bereichen der Applikation zu implementieren, was eine erste Überarbeitung der Loginfunktion und der Einstellungen zur Folge hatte.

Chatfunktion Für die Kontaktaufnahme mit anderen Benutzern beschloss ich eine Chatfunktion zu implementieren. Bei der Planung dieser Funktion stieß ich jedoch schnell auf eine Reihe von Problemen, zu welchen ich keine wirkliche Antwort hatte. Erst der Hinweis meines Onkels Peter Arrenbrecht, dass ich mir Firebase mal etwas genauer anschauen könnte, eröffnete mir neue Lösungswege für diese Probleme. Schlussendlich war es mir möglich, eine Chatfunktion zu implementieren, die eine Kommunikation in Echtzeit ermöglichte und somit den Anforderungen gerecht wurde.

A.4 Profilbilder und Passworthashing (Juni - Juli)

Das Profilbild-Feature und das Passworthashing-Feature waren die letzten beiden grossen Features, welche es zu implementieren galt.

Passworthashing Für das Passworthashing habe ich zu diesem Zeitpunkt ein nicht all zu effizientes und sicheres System entwickelt. Es basierte auf der Annahme, dass die Kommunikation zwischen Server und Client unverschlüsselt abläuft. Das entwickelte System vermied das Verschicken eines Passwortes in Klartext. Ein Angreifer wäre somit nur in Besitz des Passworthashes und des Salzes gekommen. Doch dieses System war nicht sicher, da das Bruteforcen des Passwortes noch immer deutlich zu einfach ausfiel, weshalb ich es später nochmals überarbeitet habe.

Profilbilder Das Implementieren der Profilbilder war ein eher aufwändiges Unterfangen.

Die erste Schwierigkeit bereitete mir das Auswählen und Zuschneiden der Bilddateien. Nach diversen Versuchen und verschiedenen Ansätzen gelang es mir eine Lösung dafür zu finden, die jedoch einen kleinen Schönheitsfehler hatte: beim Zuschneiden des Bildes benötigte die Applikation zu lange um

das Bild umzuformatieren (manchmal bis zu einer ganzen Minute), was ein Benutzer/einer Benutzerin als einen Absturz der Applikation interpretieren könnte. Ich nahm dies jedoch in Kauf und es gelang mir auch später nicht, diese Ineffizienz zu beseitigen.

Die zweite Schwierigkeit zeigte sich dann bei der Darstellung der Profilbilder innerhalb der Listen, wo gleich eine ganze Reihe von Profilbildern geladen werden mussten. Hierzu musste ich ein Verfahren entwickeln, welches dieses Problem effizient zu lösen wusste, was mir schlussendlich auch gelang.

A.5 Refactoring und Bugfixes (August - Oktober)

Zu diesem Zeitpunkt besass die Applikation alle nötigen Features, jedoch war ich noch sehr unzufrieden mit der Bedienung und der Benutzeroberfläche. Zudem war die Applikation noch weitgehend ungetestet.

Implementieren des BottomNavigationBars Ich entschloss mich für die Navigation innerhalb der Applikation auf einen Navigationsbalken (BottomNavigationBar) am unteren Ende des Bildschirms umzusteigen. Dies erforderte jedoch eine drastische Umstrukturierung des Codes, da von den bisher verwendeten Activities nun teilweise auf Fragments umgestellt werden musste. Das Endresultat war jedoch ein voller Erfolg. Nicht nur sah die Applikation besser aus und war einfacher zu bedienen, sie war auch bezüglich des Quellcodes deutlich schöner und systematischer geworden.

Bugfixes Mit einer voll funktionsfähigen Applikation begann ich nun gemeinsam mit einem Freund die Applikation zu testen. Dabei war es uns möglich, eine Reihe von Fehlern ausfindig zu machen und ich war in der Lage, viele davon zu eliminieren.

Umstieg auf serverseitiges Hashen mit bcrypt Der letzte Schritt war der Umstieg meines eigens entworfenen Hashverfahrens auf ein konventionelles Verfahren auf Seiten des Servers mit dem Hashalgorithmus bcrypt. Ich habe bewusst über den Schönheitsfehler der noch immer unverschlüsselten Kommunikation hinweg gesehen, doch sie wäre theoretisch einfach zu implementieren.

Sobald eine solche Verschlüsselung implementiert wird, darf die Applikation als genug sicher betrachtet werden, was das Handling der Passwörter anbelangt.

Literaturverzeichnis

- [1] Alpen-Adria-Gymnasiums Völkermarkt. *Wie funktioniert PHP?* http://www.gym1.at/schulinformatik/aus-fortbildung/fachdidaktik/vo-01/php/wie_funktioniert_php.htm. Abgerufen am: 03.05.2018.
- [2] Android Developers. *Data and file storage overview*. <https://developer.android.com/guide/topics/data/data-storage>, 2018. Abgerufen am: 15.09.2018.
- [3] Android Developers. *uses-sdk (What is API Level?)*. <https://developer.android.com/guide/topics/manifest/uses-sdk-element>, 2018. Abgerufen am: 27.08.2018.
- [4] Android Studio. *Context*. <https://developer.android.com/reference/android/content/Context>, 2018. Abgerufen am: 04.09.2018.
- [5] Anthony Ferrera. *PHP-PasswordLib*. <https://github.com/ircmaxell/PHP-PasswordLib>, 2013. Abgerufen am: 09.10.2018.
- [6] D. Boswell. *Storing User Passwords Securely: hashing, salting, and Bcrypt*. <http://dustwell.com/how-to-handle-passwords-bcrypt.html>, 2012. Abgerufen am: 23.09.2018.
- [7] J. Callaham. *The history of Android OS: its name, origin and more*. <https://www.androidauthority.com/history-android-os-name-789433/>, 2018. Abgerufen am: 07.05.2018.
- [8] P. Ducklin. *Serious Security: How to store your users' password safely*. <https://nakedsecurity.sophos.com/2013/11/20/>

- serious-security-how-to-store-your-users-passwords-safely/, 2013. Abgerufen am: 12.05.2018.
- [9] Edmodo Inc. *Cropper*. <https://github.com/edmodo/cropper>, 2013. Abgerufen am: 31.07.2018.
- [10] Epicodus. *Introduction to XML and Android Layouts*. <https://www.learnhowtoprogram.com/android/introduction-to-android/introduction-to-xml-and-android-layouts>, 2018. Abgerufen am: 03.09.2018.
- [11] Erutuon. *Base64 encoding and decoding*. https://developer.mozilla.org/en-US/docs/Web/API/Window/Base64/Base64_encoding_and_decoding, 2018. Abgerufen am: 01.09.2018.
- [12] fachadmin.de. *Server-Client Prinzip*. https://www.fachadmin.de/index.php?title=Client-Server_Prinzip&oldid=3425, 2011. Abgerufen am: 01.05.2018.
- [13] Firebase. *Firebase Realtime Database*. <https://firebase.google.com/docs/database/>, 2018. Abgerufen am: 05.05.2018.
- [14] Firebase. *Structure Your Database*. <https://firebase.google.com/docs/database/android/structure-data>, 2018. Abgerufen am: 31.08.2018.
- [15] Free Software Foundation. *GNU General Public License, Version 3, 29 June 2007*. <https://www.gnu.org/licenses/gpl-3.0.de.html>, 2007. Abgerufen am: 28.08.2018.
- [16] J. M. Gosney. *8x Nvidia GTX 1080 Hashcat Benchmarks*. <https://gist.github.com/epixoip/a83d38f412b4737e99bbef804a270c40>, 2016. Abgerufen am: 23.09.2018.
- [17] Hostinger. *000Webhost*. <https://www.000webhost.com/>, Unbekannt. Abgerufen am: 02.10.2018.
- [18] json.org. *Introducing JSON*. <https://www.json.org/json-de.html>. Abgerufen am: 03.05.2018.
- [19] S. Kersken. *IT-Handbuch für Fachinformatiker*, volume 8., aktualisierte Auflage. Rheinwerk Verlag GmbH, 2017.

- [20] T. Kreutzer. *Proprietäre Softwarelizenzen: Standard für kommerzielle Programme.* <https://irights.info/artikel/standard-fr-kommerzielle-programme/5028>, 2005. Abgerufen am: 28.08.2018.
- [21] J. Kwiecień. *EasyImage*.
<https://github.com/jkwiecien/EasyImage>, 2015. Abgerufen am: 31.07.2018.
- [22] P. Leahy. *Accessors and Mutators*.
<https://www.thoughtco.com/accessors-and-mutators-2034335>, 2018. Abgerufen am: 06.09.2018.
- [23] Oracle. *Declaring Member Variables*. <https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html>, 2018. Abgerufen am: 06.09.2018.
- [24] T. Patrick. *The Relationship Between Android and Java*.
<https://theiconic.tech/android-java-fdbd55aadc51>, 2018.
Abgerufen am: 02.09.2018.
- [25] php.net. *What is PHP?*
<http://php.net/manual/de/intro-whatis.php>. Abgerufen am: 03.05.2018.
- [26] R. S. *Introduction to Firebase*. <https://hackernoon.com/introduction-to-firebase-218a23186cd7>.
Abgerufen am: 05.05.2018.
- [27] D. Security. *Salted Password Hashing - Doing it Right*.
<https://crackstation.net/hashing-security.htm>. Abgerufen am: 12.05.2018.
- [28] tecmint.com. *MySQL and MariaDB*. <https://www.tecmint.com/the-story-behind-acquisition-of-mysql-and-the-rise-of-mariadb/>, 2017. Abgerufen am: 03.05.2018.
- [29] A. Teplitzki. *Android Image Cropper*.
<https://github.com/ArthurHub/Android-Image-Cropper>, 2016.
Abgerufen am: 31.07.2018.
- [30] The Apache Software Foundation. *How Apache Came to Be*.
http://httpd.apache.org/ABOUT_APACHE.html, Unbekannt.
Abgerufen am: 02.10.2018.

Literaturverzeichnis

- [31] Unbekannter Author. *Presentation Patterns : MVC, MVP, PM, MVVM*. <https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/>, 2012. Abgerufen am: 10.05.2018.
- [32] W3Schools. *PHP Prepared Statements*. https://www.w3schools.com/php/php_mysql_prepared_statements.asp. Abgerufen am: 26.05.2018.
- [33] W3Schools. *Introduction to XML*.
https://www.w3schools.com/xml/xml_whatis.asp, 2018.
Abgerufen am: 03.09.2018.
- [34] D. Windelband. *Die Schweiz und die Datenschutzgrundverordnung*.
<https://www.datenschutz-notizen.de/die-schweiz-und-die-datenschutzgrundverordnung-0020134/>, 2018. Abgerufen am: 01.08.2018.
- [35] W. Wingerath. *Real-Time Databases Explained*.
<https://www.youtube.com/watch?v=HiQgQ88AdYo>. Abgerufen am: 05.05.2018.