Studnetz, eine Mobilapplikation zur Vermittlung von Nachhilfeunterricht

Florian Hirtz

20. September 2018

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 Webdevelopement 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 haupsächlich aus zwei Quellen. Zum einen war es der Wille, neue Teilgebiete des Programmierens kennenzulernen und zu verstehen, weshalb die Wahl dann sehr schnell auf eine Applikation für Mobiltelefone fiel. Immerhin sind Mobiltelefone 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 betreudenen Lehrperson Andreas Umbach danken. Er stellte mir nicht nur einen Platz auf der Datanbank des Ergänzungsfaches zur Verfügugn, 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.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Zielsetzung

Das Ziel dieser Arbeit ist das Entwickeln einer Applikation für Android Geräte, die die Vermittlung von Nachhilfeunterricht unter Schülerinnern 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 dann 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, verschiedenste 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 soviel bedeute wie, 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. [10]

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

Das *Modell-View-Presenter* (MVP) Konzept wie auch 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 die View nicht.

- Der *Presenter* oder *Controller* ist sozusagen der Mittelmann 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.[27] Ein grosser Vorteil einer nach diesen Regeln entwickelter Applikation ist, 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 Modell 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 darüber 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 noch 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 ver-

schiedenen 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 ganz einfach von ihrem eigenen Profil auf ihn zugreifen.

3.2 Die Benutzeroberfläche und Benutzung

In diesem Abschnitt soll nun etwas genauer 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.

Login und Registrierung

Beim ausführen der Applikation auf dem Mobilgerät wird der Benutzer/die Benutzerin vom Login-Screen der Applikation begrüsst. 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 **??**). 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 einfach 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 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 etwas genauer erläutert werden. Weiter ist es meist möglich, über eine Schaltfläche in der oberen, linken 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 hauptsächlich aus dem Profil des Benutzers/der Benutzerin (siehe Abbildung **??**). Standardmässig 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 **??** und **??**). Die ausgewählten Fächer werden in Form von Emblems auf dem Profil dargestellt.

Suche

Das Suchfeature der Applikation findet sich in Form eines in vier Stufen unterteilten Prozess. Das Suchfeature kann über die Bottom Navigation erreicht werden. Sollte ein Benutzer/eine Benutzerin mitten in der Suche das Suchfenster verlassen habe, 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 Forme eines Screens, wo Suchkriterien wie Name, Schule, Stufe und angebotene Fächer definiert werden können, nach welchen gesucht werden

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

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 Chats der Übersicht zeigen jeweils die Profilbilder der Chatpartner/ Chatpartnerinnnen und die Zuletzt gesendete Nachricht an (siehe Abbildung **??**). Durch ein auswählen eines Elements der Übersicht kann ein gewünschter Chat geöffnet werden (siehe Abbildung **??**). Die Überlieferungen der Nachrichten im Chat erfolgen in Echtzeit, was eine flüssige Kommunikation zischen 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 Komponente jeweils einzeln etwas genauer erläutert. Die eigentliche Implementation folgt dann in Kapitel 4.3. Ein schematischer Überblick über das Verhältnis der beiden Server zum Client findet sich in Abbildung **??**.

4.1 Webserver

Bei den beiden in der Applikation verwendeten Server handelt es sich genau genommen und 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.

Es sind genau solche Skripte, die einen Zugriff auf eine sich ebenfalls auf dem Server befindende Datenbank ermöglichen. Dafür beliebte Skriptsprachen sind Sprachen wie PHP (siehe Kapitel 4.1.1), 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.

4.1.1 PHP

PHP ist eine Open-Source Skriptsprache, welche grosse Beliebtheit in der Server seitigen 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 dabei nie zu Gesicht. [21] Im Skript selber wird dann oftmals 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]

4.1.2 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: Leserlichkeit 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. [14]

4.2 Datenbanken

Eine der wohl wichtigsten Aufgaben von Computern ist das Speichern, Verwalten und auch 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 jedoch auch die Anwendung, die das Verwalten der Daten überhaupt erst ermöglicht. Solche Anwendungen werden genauer als *Database Management System* (DBMS) bezeichnet und sind meist hochkomplex in ihren Funktionsweisen, um die effiziente Verwaltung von selbst riesigen

Datenmengen zu ermöglichen. [15] 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.

Tabellen Strukturen

Die Informationen in einer Datenbank werden meist in einer auf Tabellen basierenden Struktur gespeichert. Dabei wird eine einzelnen 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 das Feld 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 im ihr zugeschriebenen Feld 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 es auch noch später in Kapitel 4.2.3 thematisiert werden wird.

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 *Einzeltabellendatenbank*. 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 Einzeltabellendatenbank 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 ein *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. [15, 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 Microsytems* 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) [13], aber auch unter eine proprietäre Lizenz [16] gestellt ist. [25] 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: [15, 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 einzugrenzen.[15, S. 746]
- Einfügeabfragen (*Insert Queries*) fügen einen neuen Datensatz zu einer bestehenden Tabelle hinzu.[15, S. 746]
- Änderungsabfragen (*Update Queries*) ändern bestimmte oder alle Felder eines bestehenden Datensatzes in einer Tabelle.[15, S. 746]
- Löschabfragen (*Delete Queries*) löschen einen Datensatz aus einer Tabelle. [15, S. 746]

4.2.2 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 diese Sicherheitslücke der Passwörter zu vermeiden wird oft ein sogenannter Prozess des Hashens und des Salzens der Passwörter angewandt.

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 mit ruhigem Gewissen 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 ja 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.

Salzen Um dem 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 an zufällig generierter Werten 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 gehashten Passwort gespeichert sein. Somit wird dann 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 Bytes 16 in Länge geraten. Die Chance fällt dann für eine Überschneidung 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. [23] [6]

4.2.3 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 Realtime Synchronisation verschiedener Clients 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.[22]

Die Firebase Echtzeitdatenbank

Wie bereits erwähnt, bietet Firebase unter anderem auch 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.[30] 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 Verzeichnis artigen 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. [12][11]

4.3 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.

4.3.1 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 somit 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 damit erfüllt werden können.

Ein Weg, die geplante Struktur zu Visualisieren, kann das Erstellen eines *Entitiy-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 **??** dargestellt.

4.3.1.1 MySQL Datenbankstruktur

Die MySQL Datenbank der entwickelten Applikation umfasst insgesamt 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. Eine Darstellung der user_archive Tabelle findet sich in Abbildung **??**.

- 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 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 [9]. 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 dann dabei belassen wurde.

- Die *user_hashes* Tabelle besitzt drei Felder. Eines für die user_id, eines für das gehashte Passwort und eines für das Salz eines Benutzers/einer Benutzerin. Mehr dazu in Kapitel **??**.

- Die *user_subjects* Tabelle beinhaltet Felder für alle von der entwickelten Applikation unterstützten Fächer (siehe Abbildung **??**). 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 hauptsächlich das Speichern aller statischen Informationen über die verschiedenen registrierten Benutzer/Benutzerinnen. Das dazugehörige ERM ist in Abbildung **??** dargestellt. 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.

4.3.1.2 Firebase Datenbankstruktur

Wie bereits in Kapitel 4.2.3 erwähnt wurde, ist die Firebase Echtzeitdatenbank etwas anders Strukturiert, weshalb sie auch nur schwer von einem ERM wirklich dargestellt werden kann. Viel besser eignen sich Baum ähnliche Darstellungen, wie sie es in Abbildung **??** 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 kurz anzumerken, dass sich die verwendeten Begriffe *Sender* und *Receiver* immer aus der Sicht des momentan aufgerufenen Profils beziehen. Somit sind bei zwei Chatpart-

nern/Chatpartnerinnen beide aus der einen Sicht der Sender, währen 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 des MySQL Datenbank gekennzeichnet. Darin findet sich dann ein Knoten *chats*. Dort werden dann Knoten für alle Chats aufgeführt, in welchen ein Benutzer/eine Benutzerin involviert ist. Die Knoten sind jeileis so benannt, sodass beide user_ids der am Chat Benutzer/Benutzerinnen daraus herausgelesen werden können. Der Knoten eines Chats enthält dann die sechs für einen Chat relevanten Werte: die Referenz zum Chat, wo die Nachrichten des Chats aufgeführt sind, ein Wert für die zuletzt gesendete Nachricht, 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 dann ein Knoten mit dem Namen *Messages*. Unter diesem Knoten findet man dann Knoten für alle im Chat gesendeten Nachrichten. Die Knoten der einzelnen Nachrichten tragen dabei jeweils einen generierte Zeichenkette als Namen. Die Nachrichten selber bestehen dann 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 nämlich 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 trotz-

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

4.3.2 PHP Skripte

Im folgenden Abschnitt wird nun genauer auf die PHP Skripte eingegangen, die auf dem Webservers der MySQL Datenbank verwendet werden.

4.3.2.1 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 das Login in zwei Schritten.

Im ersten Schritt wird vom Client das *salt.php*-File auf dem Server vie Webadresse aufgerufen, wobei dem PHP File mithilfe der POST Methode der eingegebene Benutzername übergeben wird. Dort wird dann anschliessend ein sogenanntes *MySQL Statement* erstellt. 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 4.1). 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 sind dazu da, um sogenannten *SQL Injektionen* vorzubeugen [28]. Bei der Auswahlabfrage in Abbildung 4.1 darf jeweils nur genau ein Eintrag gefunden werden, da es sonst zu Problemen beim Passwort kommen kann. Deshalb muss jeder Benutzername in der Datenbank einzigartig sein. Wird von der Auswahlabfrage ein Datensatz mit dem eingegebenen Benutzernamen gefunden, wird dem Client das Salz des Benutzers/der Benutzerin in JSON-Format zurückgegeben.

```
SELECT user_hashes.hash_salt FROM user_archive INNER JOIN
↪  user_hashes ON user_archive.user_id = user_hashes.user_id
↪  WHERE user_archive.user_username = ?
```

Abbildung 4.1: Select Query für das Auslesen des Salzes eines Benutzers aus der user_archive Tabelle und der user_hashes Tabelle.

Im zweiten Schritt wird auf Seiten des Clients das eingegebene Passwort zusammen mit dem Salz gehasht. Das Salz ist essentiell nichts anderes als ein für jeden Benutzer/jede Benutzerin zufällig generierter Wert, der Grund dafür wird in Kapitel **??** genauer Erläutert. Der erhaltene Hash wird anschliessend an das *login.php*-File auf dem Server übergeben. Dieses vergleicht nun den erhaltenen Hash mit dem in der Datenbank gespeicherten Hash. Sind beide identisch, wurde das korrekte Passwort eingegeben. 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 auch 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.

4.3.2.2 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_php_v2.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 (siehe Kapitel **??**). 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 4.2).

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

Abbildung 4.2: 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_profilepictures Tabelle werden dabei sämtliche weiteren Werte standardmässig auf 0 gesetzt. Diese Werte sollen erst später über die Profileinstellungen bearbeitet werden können. In der user_hashes Tabelle werden die vom Client erhaltenen Werte für das gehashte Passwort und das Salz eingefügt. Sofern dabei keine weiteren Fehler auftreten wird schlussendlich eine JSON-Antwort geschickt, die ihm die erfolgreiche Registrierung mitteilt.

4.3.2.3 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 Activtiy) mit welchem die Angaben editiert werden können. Wenn die editierten Angaben auf dem Client gespeichert werden, wird eine Anfrage an das Skript *savesettings.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 4.3).

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

Abbildung 4.3: 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 immer 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.

4.3.2.4 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* (Zeichenkette) 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 4.4).

```
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 4.4: Etwas abgekürzte Version der SQL Select Query des search.php Skriptes mit einem Inner Join der user_archive Tabelle und der user_subjects Tabelle

Ebenfalls ist in diesem String bereits 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 komplett 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 noch 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 durchnummeriert 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.

4.3.2.5 Profilbilder

Wie bereits in Kapitel 4.3.2.4 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 **??**.

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 **??**.

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 genauer 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. Die Entwicklung der Studnetz Applikation für Android Geräte macht also nur Sinn, da somit eine grosse Menge an Benutzern/Benutzerinnen erreicht werden kann. [5]

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 [24] 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 dann ein langsamer ein Umschwung zur neueren Programmiersprache *Kotlin*, die jedoch essentiell auch in Bytecodes für die JVM Kompiliert wird.

Kotlin gewinnt hauptsächlich wegen seiner vorteilhaften Syntax an immer grösseren Beliebtheit, wobei insbesondere in der Entwicklung von Applikationen immer mehr auf Kotlin anstelle von Java gesetzt wird. [20]

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 trozdem möglich, auch XML für die Definition von Darstellungen zu verwenden, wie es beispielsweise bei Android Applikationen üblich ist. Zudem kommt das XML deutlich flexibler als HTML ist, 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 Integer. Somit können in einer Applikation auch relativ einfach verschiedene Sprachpackete entwickelt werden, da dafür lediglich alle definierten Strings, welche sich alle in einer Datei befinden, übersetzt werden müssen. Alle XML-Dateien zusammen bilden die *Resources* einer Android Applikation. Auf diese Resources sind für die Java Klassen zugänglich und

können manipuliert werden. Diese ermöglicht eine direkte Verknüpfung von Java mit dem Layout und ermöglicht die Entwicklung dynamischer Screens und Darstellungen. [8] [29]

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 unzählige an hilfreichen Werkzeugen für die Entwicklung von Applikationen. 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, die allen, von Amateur bis hin zu Profi, etwas bietet. Das Herunterladen und die komplette Benutzung von Android Studio ist kostenlos. Die Wahl für die Entwicklungsumgebung für die in dieser Arbeit entwickelten Applikation war also 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 der Arbeit wird auf den entwickelten Client auf einer technischeren Ebene eingegangen. Es ist an dieser Stelle anzumerken, dass für diesen Abschnitt gewisse Grundkenntnisse des objektorientierten Programmierens und der Programmiersprache Java vorausgesetzt werden, da das Erläutern der Grundlagen von dem eigentlich entwickelten Client zu sehr ablenken würde. Weiter ist noch 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 ausgeschnitten 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 soll jetzt hier jedoch nicht weiter eingegangen werden, können jedoch mitsamt Screenshots aus der Applikation im Anhang nachgeschlagen werden. 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
23         return super.onOptionsItemSelected(item);
24     }
25 }
```

Abbildung 5.1: Grundgerüst einer Activity

Alle dargestellten Funktionen sind theoretisch optional und müssen nicht zwangsweise in einer Activity vorkommen, 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 Activty 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 Java-Objekte der passenden Klasse 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 Funktion, die beim 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 24 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 neunen 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 eine 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 Acitvity eingebettet werden. Sie besitzen dann dort einen vorgegebenen Rahmen in welchem ihr Layout dann angezeigt wird. Die zum Layout gehörenden Methoden finden sich dann 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,
       ↪  ViewGroup container, Bundle savedInstanceState) {
13
14         View view =
           ↪  inflater.inflate(R.layout.FragmentLayoutName,
           ↪  container, false);
15
16         [...]
17
18         return view;
19     }
20
21     @Override
22     public void onStart() {
23         super.onStart();
24
25     }
26
27     @Override
28     public boolean onOptionsItemSelected(MenuItem item) {
29
30         [...]
31
32         return super.onOptionsItemSelected(item);
33     }
34 }
```

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 wir sie hauptsächlich 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 irgendwie auf Ressourcen der Applikation zugreifen müssen und es gibt in der Androidentwicklung nur wenige Prozesse, in welchen der Kontext nicht irgendwann benötigt wird. [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 onCreate-Methode wird in den Fragments der Studnetz eher weniger verwendet, weshalb sie hier nicht selber aufgeführt ist. Stattdessen übernimmt die onCreateView-Methode die Aufgabe des Definieren 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 meist für vom Server abhängige Prozesse verwendet, bei welchen mit Verzögerungen zu rechnen ist.

Die onOptionItemSelected-Methode Auch hier bei der *onOptionItemSelected*-Methode kann auf die äquivalente Methode der Activites in Kapitel 5.2.1 verwiesen werden, da auch hier 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 (sie-

he Kapitel 2.2). Die Models der Studnetz Applikation können generell vom Aufbau her in 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. [19]
- 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 meist schon alle Daten mitgegeben, die für die Felder benötigt werden.
- Die *Getter*- und die *Setter*-Methoden: Die Getter- und die Setter-Methoden finden sich als letztes 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. [18]

5.2.3.1 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 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 wohl zentralste Model der entwickelten Applikation. Es wird in allen Activities und Fragements mit Ausnahme des Logins und der Registrierung verwendet. Da jedoch das UserModel-Objekt selber nicht zwischen den Acitivties 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, wie der Webserver verwendet, wenn er Benutzerinformationen an den Client schickt.

5.2.3.2 ProfilePictureModel

Für das Darstellen und Speichern von Profilbild auf dem Client wird das ProfilePictureModel-Model verwendet. Das Model besitzt Felder für die drei verschiedenen Formen 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 bereits in Kapitel 4.3.1.1 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 jeweils 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-Model kann dabei über alle drei der Formate initialisiert werden, weshalb sie jeweils einen Konstruktor für jedes Format besitzt. Anschliessend wird das gegebene Format jeweils in die anderen Formate übersetzt. Hierzu finden sich eine Reihe von Methoden in der ProfilePictureModel-Klasse, die dies ermöglichen. Hinzu kommt dann noch ein vierter Konstruktor, der das Objekt mithilfe einem Parameter der Klasse *Uri* initialisiert wird. Uri (Uniform Resource Identifier) ist eine Form der Dateireferenzierung. Sie wird bei der Auswahl eines neuen Profilbildes verwendet und kann dann 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
        ↪ BASE64) {
12          this.mContext = context;
13          this.BASE64 = BASE64;
14          this.imageBitmap = decodeBASE64(this.BASE64);
            ↪ //Methode für das Umwandeln von Base64 zu Bitmap
15          createTemp(this.imageBitmap, MAX_QUALITY);
            ↪ //Erstellen einer neuen Bild-Datei im Cache
16          this.success = true;
17      }
18
19      public ProfilePictureModel(Context context, Bitmap
        ↪ imageBitmap) {
20          [...]
21      }
22
23      public ProfilePictureModel(Context context, Uri imageUri,
        ↪ int quality) {
24          [...]
25      }
26
27      public ProfilePictureModel(Context context, File
        ↪ tempFile) {
28          [...]
29      }
30
31      [...]
32  }
```

Abbildung 5.4: Felder und Konstruktoren der ProfilePictureModel-Klasse

Eine weitere Aufgabe, die ebenfalls in die Hände des ProfilePictureModel-Models fällt, ist die Regulierung der Bildqualität und Auflösung. Dies ermöglicht, dass egal was für ein Bild ein Benutzer/Benutzer als Profilbild ausgewählt hat, alle Bilder schlussendlich ungefähr gleich gross in der Auflösung sind. Zudem werden die Bitmaps innerhalb der Applikation jeweils in einer Runden Form dargestellt. Diese wird ihnen auch innerhalb des ProfilePictureModel-Models gegeben, wobei hierzu jedoch auf ein bereits existierendes Github-Repository von Arthur Teplitzki zurückgegriffen wurde, welches verschie-

dene Tools für das zuschneiden von Bildern bietet. Dieses Repository wird noch etwas mehr in Kapitel 5.2.6 thematisiert.

5.2.3.3 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,
       ↪  mUserprofileRef, mChatRef;
5
6      public ChatModel(UserModel mMainprofileModel, UserModel
       ↪  mUserprofileModel, DatabaseReference mMainprofileRef,
       ↪  DatabaseReference mUserprofileRef, DatabaseReference
       ↪  mChatRef) {
7          this.mMainprofileModel = mMainprofileModel;
8          this.mUserprofileModel = mUserprofileModel;
9          this.mMainprofileRef = mMainprofileRef;
10         this.mUserprofileRef = mUserprofileRef;
11         this.mChatRef = mChatRef;
12     }
13 }
```

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 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 **??**.

5.2.4.2 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. Die für die verwendeten Models wurden bereits alle in Kapitel 5.2.3 thematisiert, weshalb hier nicht mehr näher darauf eingegangen wird.

5.2.4.3 ViewHolder

Die ViewHolder-Klasse beschreibt jeweils ein einzelnes Element der Auflistung. Innerhalb eines ViewHolders sind Dinge wie das zu verwendende Layout wie auch das zum Element gehörende Model referenziert. Es ist teilweise umstritten, wie weit sich der Aufgabenbereich der ViewHolder-Klasse einer Liste streckt. 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 5.6.

```
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 =
            ↪ view.findViewById(R.id.result_name_textview);
15          TextView school_tv =
            ↪ view.findViewById(R.id.result_school_textview);
16
17          [...]
18
19          name_tv.setText(this.model.getFirstname() + " " +
            ↪ this.model.getName());
20          school_tv.setText(this.model.getSchool() +
            ↪ this.model.getStringGrade());
21      }
22
23      [...]
24
25  }
```

Abbildung 5.6: ViewHolder-Klasse für die Auflistung der Suchergebnisse

5.2.4.4 Adapter

Die 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 werden 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 dann jeweils in Echtzeit mit der Datenbank synchronisiert. Auf den FirebaseRecyclerAdapter soll jedoch hier nicht weiter eingegangen werden, sonder der Fokus wird stattdessen auf den normalen RecyclerAdapter gelegt. Ein Beispiel eines solchen Adapters findet sich in Abbildung 5.7.

```
1   class ChatoverviewAdapter extends
    ↪ RecyclerView.Adapter<OpenChatViewHolder>{
2
```

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

Abbildung 5.7: 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 Forme 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 Forme 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, kann die Liste über eine notifyDatasetChanged-Methode Aktualisiert werden, ohne das der Datensatz dabei erneut übergeben werden muss.

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

Die onCreateViewHolder-Methode Die *onCreateViewHolder*-Methode wird hat die Aufgabe, das Layout eines einzelnen Elements zu laden (Zeile 13) und dann ein neues ViewHolder-Objekt zu instantiieren (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 meist 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 weiter Methode der ViewHolder-Klasse, die das definieren eines zu verwendenden Profilbildes ermöglicht (Zeile 22).

Die onBindViewHolder-Methode wird nach dem Start der View immer dann 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. Somit kann erreicht werden, dass mit einer nur geringen Anzahl an auf einmal geladener Layouts trotzdem beliebig grosse Datensätze effizient dargestellt werden können. Dieser Prozess der Wiederverwertung von nicht mehr sichtbarer Elemente 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 instantiieren. 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 5.8 dargestellt.

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

Abbildung 5.8: 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 um Cache zu erstellen. Hierzu wird die *getTempFilePath*-Methode verwendet. Sollte der Base64 String jedoch dem Wert "0entsprechen oder nicht gesetzt sein (null), wird ein vordefiniertes Platzhalter Bild anstelle 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 gleich mehrere Profilbilder auf einmal zu Laden. Dies wird jeweils bei der Darstellung von Auflistungen benötigt, wo gleichzeitig mehrere Profile mit einem Profilbild sichtbar werden. Dabei werden jeweils beim Start der Auflistungen die Profilbilder der ersten 20 Datensatzeinträge mithilfe der ProfilePictureLoader-Klasse geladen. Dann werden jeweils immer wenn noch ungeladene Profile sichtbar werden über die ProfilePictureLoader-Klasse die nächsten 20 Profilbilder geladen, bis schlussendlich, wenn alle Ergebnisse durchgesehen worden sind, 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 jeweils die von der Applikation unterstützten Schulen im einen Spinner angezeigt und 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 dann die MyReader-Klasse, die dieses Dokument zu lesen vermag und anschliessend die gelesenen Zeile als Liste zurückgibt. Es ist dann die SchoolMapper-Klasse, welche das Handling der beiden Spinner übernimmt. Eine Abbildung der SchoolMapper-Klasse findet sich in Abbildung 5.9.

```
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
       ↪  schoolSpinner, Spinner gradeSpinner) {
8          this.mContext = mContext;
9          this.schoolSpinner = schoolSpinner;
10         this.gradeSpinner = gradeSpinner;
11     }
12
13     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(":");
                     ↪  //Schulen sind jeweils durch ein ":" von
                     ↪  ihrerer Schulstufe getrennt, zum Beispiel:
                     ↪  "Kantonsschule Ausserschwyz:6"
18                   mOut.put(tempData[0],
                     ↪  Integer.parseInt(tempData[1]));
19              }
20
21              return mOut;
22         }
23
24         public void startDisplay(String path){
25
26              MyReader mReader = new MyReader(mContext);
27
28              mMap = map(mReader.read(path));
29
30              String[] keySet = mMap.keySet().toArray(new
                ↪  String[mMap.size()]);
31
32              ArrayList<String> mSchools = new ArrayList<>();
33
34              for(int n = 0; n < mMap.size(); n++) {
35                   mSchools.add(keySet[n]);
36              }
37
38              ArrayAdapter<String> schoolAdapter = new
                ↪  ArrayAdapter<String>([...], mSchools);
39              schoolAdapter.setDropDownViewResource(
                ↪  android.R.layout.simple_spinner_dropdown_item);
40              schoolSpinner.setAdapter(schoolAdapter);
41              schoolSpinner.setOnItemSelectedListener(new
                ↪  AdapterView.OnItemSelectedListener() { [...] }
42         }
43    }
```

Abbildung 5.9: Instantiierung eines UserModel-Objektes aus den Extras

Beim Initialisieren der Klasse werden dabei die Referenzen zu den beiden Spinnern über die Parameter des Konstruktors der Klasse überreicht (Zeile 7). Wenn dann 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 Schulstufenanzahl in eine *LinkedHashMap* gespeichert (Zeile 14). Eine LinkedHashMap ist fast identisch zu einer normalen HashMap, wo auch jeweils einen Key gemeinsam mit einem Wert gespeichert werden. Der einzige Unterschied findet sich hier, 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 jeweils 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 um schnell einen ersten Eindruck von anderen Benutzern/Benutzerinnen zu bekommen. Auch in der entwickelten Applikation ist es möglich, sich 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 jedoch auf zwei öffentliche Github Repositories zurückgegriffen, da die eigene Programmierung dieser Funktionen wahrscheinlich 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 [17]. 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. [26] [7]. Dieses Repository bring 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 nun also ein Benutzer/eine Benutzerin ein Bild aus, wird dieser Cropper gestartet, ein Screenshot davon findet sich in Abbildung **??**. Der Benutzer/Die Benutzerin kann daraufhin den Bereich des Bildes Auswählen, welcher er/sie als Profilbild verwenden möchte. Ist dies getan, kann die Auswahl bestätigt werden und das Bild wird daraufhin zugeschnitten. Dabei wird auch noch gleich die Qualität und Grösse des Bildes auf ein einheitliches Mass gestellt. So ist jedes Bild in der Datenbank ungefähr ähnlich gross. Zudem wird jedes Profilbild gleich noch in einer zweiten, deutlich kleineren Version gespeichert. Dieses wird dann für die Suchergebnisse und die Aufgelisteten Chats verwendet, wo teilweise gleich 20 Bilder gleichzeitig geladen werden müssen. Um dabei Leistung zu sparen gibt es also 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 4.3.1.1).

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 daher gut für das Kontrollieren der Kommunikation mit dem Webserver auf Seiten des Clients. Die Kommunikation mit dem Webserver läuft dabei hauptsächlich 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 sie also als ein adressiertes Postpaket vorstellen, welches dann 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 5.10.

```
1 public class BigProfilePictureRequest extends StringRequest {
2
3     private static final String URL = "http:// [...]
      ↪  /profilepicture_big_php.php";
4     private Map<String, String> params;
```

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

Abbildung 5.10: Request-Klasse für die Anfrage nach einem Profilbild eines Benutzers

Die Webadresse, an welche die Anfrage gehen soll, wird dabei in einem der beiden Felder definiert (Zeile 3). Dabei wird genau 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 Falle des Beispiels der Key sowie der Wert der HashMap Strings. Die HashMap kann nachdem sie erfolgreich an den Server übermittelt worden ist dann verwendet werden, um die mitgegebenen Werte über die Keys wieder auszulesen.

Der Konstruktor benötigt im Falle des Beispiels 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 eine sogenannte *RequestQueue* aus der Volley Library als Anfrage an den Webserver gesendet werden. Ein Beispiel findet sich in Abbildung 5.11

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

Abbildung 5.11: 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 Forme eines Strings übermittelt, der dann in ein JSON-Objekt umgewandelt werden kann. Ein Beispiel einer solchen Klasse findet sich in Abbildung 5.12.

```
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 5.12: OnBigPictureResponseListener-Klasse

Bei einer empfangenen Antwort wird immer 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 sehr stark von der Interaktion mit dem Webserver. Anders als bei einem herkömmlichen Webserver ist das implementieren von serverseitiger Datenverarbeitung in Forme 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 (Wird erweitert sobald Sicherheit geklärt ist!)

Die Verbindung zum Server wird wie beim Webserver auch hier über eine Webadresse erstellt. Die Adresse wird dabei in einer separaten Datei definiert. Es ist auch in dieser Datei, wo die gesamten für die Authentifizierung benötigten Werte gespeichert sind. Die Authentifizierung wird dabei von Freibase selber geregelte und läuft über ein Google Konto.

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 Für das Einfügen neuer Werte in die Echtzeitdatenbank ist eher simpel. Ein Beispiel hierfür findet sich in Abbildung 5.13 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 5.13: Einfügen von neuen Werten in die Firebase Echtzeitdatenbank

Hierzu wird zuerst 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 dann zu einem JSON-Objekt einer Stufe niedriger navigiert mit dem Namen *Messages*. Die *push*-Methode bewirkt dann, dass ein neues JSON-Objekt mit einem zufällig generierten Namen erstellt wird. Diese nun konfigurierte Referenz kann nun verwendet werden, um neue Werte in die Datenbank einzufügen. Hierzu wird zuerst über die child-Methode ein neues JSON-Objekt erstellt, wo dann der einzufügende Wert über die *setValue*-Methode eingefügt werden kann (Zeile 2). Ein visualisierte Darstellung über diesen Vorgang findet sich in Abbildung **??**.

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 die diese Aufgabe weitgehend übernehmen. In Abbildung 5.14 wird jedoch 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 5.14: ValueEventListener-Klasse für das Auslesen von Daten aus der Firebase Echtzeitdatenbank

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

Abbildung 5.15: Zuweisung eines ValueEventListners zu einer Referenz

Eine Klasse mit einem Implementierten ValueEventListener-Interface kann jeweils einer oder mehren Referenzen zugewiesen werden (siehe Abbildung 5.15). Hierzu wird innerhalb der Klasse jeweils bei der Zuweisung zu einer neuen Referenz und anschliessend immer wenn sich der Datensatz einer Referenz ändert die *onDataChange*-Methode aufgerufen (Abbildung 5.14, Zeile 6). Diese bekommt über ihre Parameter einen sogenanntes *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 verarbeitete werden.

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: 03.05.18.

[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] J. Callaham. *The history of Android OS: its name, origin and more*. `https://www.androidauthority.com/history-android-os-name-789433/`, 2018. Abgerufen am: 07.05.18.

[6] 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: 12.05.18.

[7] Edmodo Inc. *Cropper*. `https://github.com/edmodo/cropper`, 2013. Abgerufen am: 31.07.18.

[8] 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.

[9] Erutuon. *Base64 encoding and decoding*. `https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/Base64_encoding_and_decoding`, 2018. Abgerufen am: 01.09.2018.

[10] fachadmin.de. *Server-Client Prinzip*. `https://www.fachadmin.de/index.php?title=Client-Server_Prinzip&oldid=3425`, 2011. Abgerufen am: 01.05.18.

[11] Firebase. *Firebase Realtime Database*. `https://firebase.google.com/docs/database/`, 2018. Abgerufen am: 05.05.18.

[12] Firebase. *Structure Your Database*. `https://firebase.google.com/docs/database/android/structure-data`, 2018. Abgerufen am: 31.08.2018.

[13] 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.

[14] json.org. *Introducing JSON*. `https://www.json.org/json-de.html`. Abgerufen am: 03.05.18.

[15] S. Kersken. *IT-Handbuch für Fachinformatiker*, volume 8., aktualisierte Auflage. Rheinwerk Verlag GmbH, 2017.

[16] T. Kreutzer. *Proprietäre Softwarelizenzen: Standard für kommerzielle Programme*. `https://irights.info/artikel/standard-fr-kommerzielle-programme/5028`, 2005. Abgerufen am: 28.08.2018.

[17] J. Kwiecień. *EasyImage*. `https://github.com/jkwiecien/EasyImage`, 2015. Abgerufen am: 31.07.18.

[18] P. Leahy. *Accessors and Mutators*. `https://www.thoughtco.com/accessors-and-mutators-2034335`, 2018. Abgerufen am: 06.09.2018.

[19] Oracle. *Declaring Member Variables*. `https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html`, 2018. Abgerufen am: 06.09.2018.

[20] T. Patrick. *The Relationship Between Android and Java*. https://theiconic.tech/android-java-fdbd55aadc51, 2018. Abgerufen am: 02.09.2018.

[21] php.net. *What is PHP?* http://php.net/manual/de/intro-whatis.php. Abgerufen am: 03.05.18.

[22] R. S. *Introduction to Firebase*. https://hackernoon.com/introduction-to-firebase-218a23186cd7. Abgerufen am: 05.05.18.

[23] D. Security. *Salted Password Hashing - Doing it Right*. https://crackstation.net/hashing-security.htm. Abgerufen am: 12.05.18.

[24] Tech Terms. *Bytecode*. https://techterms.com/definition/bytecode, 2018. Abgerufen am: 03.09.2018.

[25] tecmint.com. *MySQL and MariaDB*. https://www.tecmint.com/the-story-behind-acquisition-of-mysql-and-the-rise-of-mariadb/, 2017. Abgerufen am: 03.05.18.

[26] A. Teplitzki. *Android Image Cropper*. https://github.com/ArthurHub/Android-Image-Cropper, 2016. Abgerufen am: 31.07.18.

[27] Unbekannter Author. *Presentation Patterns : MVC, MVP, PM, MVVM*. https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/, 2012. Abgerufen am: 10.05.18.

[28] W3Schools. *PHP Prepared Statements*. https://www.w3schools.com/php/php_mysql_prepared_statements.asp. Abgerufen am: 26.05.18.

[29] W3Schools. *Introduction to XML*. https://www.w3schools.com/xml/xml_whatis.asp, 2018. Abgerufen am: 03.09.2018.

[30] W. Wingerath. *Real-Time Databases Explained*. https://www.youtube.com/watch?v=HiQgQ88AdYo. Abgerufen am: 05.05.18.