DUMMY: Entwicklung einer Applikation fr Mobiltelefone zur Vermittlung von Nachhilfe

Florian Hirtz

3. Juni 2018

Inhaltsverzeichnis

Kapitel 1

Vorwort

DummyText...

1.1 Motivation

DummyText...

1.2 Danksagungen

DummyText...

Kapitel 2

Einleitung

2.1 Zielsetzung

Das Ziel dieser Arbeit ist das Entwickeln einer Applikation für Android Geräte, die die Vermittlung von Nachhilfe 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 suchen können, die ihren Bedürfnissen entsprechen. Weiter soll es ihnen möglich sein, über eine Chatfunktion andere Benutzer zu kontaktieren.

Kapitel 3

Konzeptionelle Grundlagen

3.1 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 passiv. Ein Server sollte immer in der Lage sein, Anfragen entgegenzunehmen und zu verarbeiten.

Der *Client* selber ist die aktive Komponente des Systems. Er 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. [7]

3.2 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 zu manipulieren, als auch das Model.

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.[3] 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 4

Die entwickelte Applikation

Die im Rahmen dieser Arbeit entwickelte Applikation wurde für Mobiltelefone mit einer Version des Betriebssystems Android entwickelt. Sie kann darauf installiert und anschliessend ausgeführt werden. Für den Gebrauch der Applikation ist eine Verbindung zum Internet notwendig.

4.1 Features

Damit die Applikation auch die aufgegebenen Aufgaben 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 selber schreiben und sie haben die Möglichkeit, falls sie selber Nachhilfe anbieten wollen, Fächer auszuwählen, in welchen sie das tun möchten. Das Profil selber ist für andere Benutzer/Benutzerinnen einsehbar. Die meisten Accountdetails können über die 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 oder nach ausgewählten Fächern gesucht werden. Die gefundenen Profile der verschiedenen Benutzern/Benutzerinnen können anschliessend angesehen werden.

- Das *Chat-Feature*: Sollte der Benutzer/die Benutzerin einen anderen Benutzer oder Benutzerin gefunden haben, mit welchem/welcher er/sie

Kontakt aufnehmen möchte, so 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.

4.2 Benutzung

Wenn die Applikation auf dem Mobilgerät ausgeführt wird, öffnet sich der Login-Screen (siehe Abbildung **??**). 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, ein Geburtsdatum sowie die momentan besuchte Schule. Ist die Registrierung erfolgreich, kann der Benutzer/die Benutzerin ab sofort sich in seinen/ihren neuen Account einloggen. Nach dem erfolgreichem Einloggen öffnet sich die Hauptseite der Applikation. Sie besteht hauptsächlich aus dem Profil des Benutzers/der Benutzerin (siehe Abbildung **??**). Standardmässig findet sich hier ein Profilbild, der Name des Benutzers/der Benutzerin, sein/ihre Geburtsjahr und die besuchte Schule. Weitere Angaben wie angebotene Fächer und eine Beschreibung können in den Einstellungen konfiguriert werden, welche über eine Schaltfläche in der oberen rechten Ecke erreicht werden können. Die ausgewählten Fächer werden in Form von Emblems auf dem Profil dargestellt. Ebenfalls auf der Hauptseite findet sich ein ausfahrbares *BottomSheet* am unteren Ende des Bildschirms. Es kann über eine runde Schaltfläche in der Mitte oder ein einfaches Streichen nach oben ausgefahren werden. Im BottomSheet selber finden sich Einträge für alle offenen Chats des Benutzers/der Benutzerin(siehe Abbildung **??**). Über einen einfachen Klick können sie geöffnet werden. Neben dem BottomSheet findet sich ebenfalls im linken unteren Ende eine runde Schaltfläche mit einer Lupe. Über sie kann auf das Such-Feature zugegriffen werden und sie führt einen auf einen neuen Screen, auf welchem die Kriterien für die Suche definiert werden können (siehe Abbildung **??**). Hier könne Dinge wie Name und Fächer angegeben werden, nach welchen gesucht werden soll. Ist der Benutzer/die Benutzerin zufrieden mit den Einstellungen kann über eine Schaltfläche die Suche ausgeführt werden. Alle Suchergebnisse werden in Form einer Liste daraufhin im Client aufgeführt. Sollte den Benutzer/die Benutzerin Interesse an einem der Ergebnisse haben, kann durch einfaches Auswählen des Eintrags das Profil des/der gefun-

denen Benutzer/Benutzerin aufgerufen werden (siehe Abbildung **??**). Ist der Benutzer/die Benutzerin interessiert, Kontakt mit dem/der gefundenen Benutzer/Benutzerin aufzunehmen, kann über eine Schaltfläche auf dem Profil ein neuer Chat geöffnet werden (siehe Abbildung **??**). Dort können genauere Informationen über eine mögliche Zusammenarbeit ausgetauscht werden.

Kapitel 5

Applikationskomponenten

In diesem Kapitel wird verwendete Komponenten innerhalb der Applikation eingegangen. Die einzelnen Komponenten sollen erklärt werden und ihre Rolle in der entwickelten Applikation ersichtlich werden.

5.1 Systemüberblick

Die Applikation basiert auf dem Client-Server Prinzip. Der Client findet sich in Form der entwickelten Anwendung, die auf Mobiltelefonen von den Benutzern/Benutzerinnen installiert werden kann. Der Server wird von den beiden Datenbanken gebildet. Die Interaktionen der einzelnen Komponenten entsprechen weitgehend dem MVP Konzept (siehe Kapitel 3). Der Presenter und die View finden sich beide auf Seiten des Clients (siehe Kapitel 5.4) während das Model sich in Form der beiden Server wiederfindet (siehe Kapitel 5.2 und 5.3).

5.2 Datenbank Server

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 auch als *Database Management System* (DBMS) bezeichnet und sind meist hochkomplex in ihren Funktionsweisen. [10]

Datenbanken selbst wiederum können in verschiedene Typen eingeteilt werden, die alle ihre eigenen Vor- und Nachteile mit sich bringen. Die ein-

fachste 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. 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 eindeutigen Eigenschaften eines Eintrags. 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. [10, S. 745 - 751]

5.2.1 MySQL Datenbank

Ein Beispiel für ein solches RDBMS ist die weit verbreitete 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), aber auch unter eine proprietäre Lizenz gestellt ist. [14] Das MySQL System darf aufgrund der GPL gratis heruntergeladen, installiert und modifiziert werden.

In Kombination mit der Programmiersprache PHP bildet sie eines der meist verwendeten Datenspeichersysteme für Webdienste aller Art. Im Falle eines solchen Webdienstes befindet sich die Datenbank meist auf einem zentralen Server, auf welchem sich ebenfalls die benötigten PHP-Skripte befinden. Die PHP-Skripte haben die Aufgabe, Abfragen von den Clients entgegenzunehmen und über sogenannte *Queries* (Datenbank Abfragen) auf die Informationen in der Datenbank zuzugreifen. Im Falle einer MySQL Datenbank werden solche Queries in der Datenbanksprache *SQL* (Structured Query Language) formuliert. Queries können in vier Arten von Abfragen unterteilt werden: [10, 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.[10, S. 746]

	Field	Type	Collation	Attributes	Null	Default	Extra	Action
	user_id	int(8)			No		auto_increment	
	user_username	varchar(18)	utf8_general_ci		No			
	user_name	varchar(18)	utf8_general_ci		No			
	user_firstname	varchar(18)	utf8_general_ci		No			
	user_school	varchar(30)	utf8_general_ci		No			
	user_yearofbirth	int(4)			No			
	user_email	varchar(30)	utf8_general_ci		No			
	user_password	varchar(28)	utf8_general_ci		No			
	user_description	text	utf8_general_ci		No			

Check All / Uncheck All With selected:

	user_id	user_username	user_name	user_firstname	user_school	user_yearofbirth	user_email	user_password	user_description
	1	hoffmann	Hoffmann	Ernst T. A.		1772	eta@gmail.com	123	Sapere audel
	2	Geronimo	Jenni	Shemon		2001	[redacted]		A user description

Abbildung 5.1: Die user_archive Tabelle, aufgenommen aus dem phpMyAdmin Interface

- Einfügeabfragen (*Insert Queries*) fügen einen neuen Datensatz zu einer bestehenden Tabelle hinzu.[10, S. 746]
- Änderungsabfragen (*Update Queries*) ändern bestimmte oder alle Felder eines bestehenden Datensatzes in einer Tabelle.[10, S. 746]
- Löschabfragen (*Delete Queries*) löschen einen Datensatz aus einer Tabelle. [10, S. 746]

5.2.2 MySQL Datenbankstruktur

In der entwickelten Applikation wird genau ein solches MySQL Datanbanksystem in Kombination mit PHP (siehe Kapitel 5.2.3) benutzt. Die Datenbank wird verwendet, um die Accountdaten der einzelnen Benutzer zu speichern und den Clients zur Verfügung zu stellen. Sie befindet sich auf dem Schulserver vom Ergänzungsfach. Die Datenbank umfasst 2 Tabellen: die *user_archive* Tabelle und die *user_subjects* Tabelle. In der user_archive Tabelle findet sich ein Eintrag für jeden registrierten Benutzer mit Informationen wie Benutzername, Passwort und Jahrgang (siehe Abbildung 5.1). Die andere Tabelle beinhaltet Felder für alle von der entwickelten Applikation unterstützten Fächer (siehe Abbildung 5.2). Mithilfe der Werte 1 und 0 wird angegeben, in welchen Fächern ein Benutzer/eine Benutzerin Unterstützung anbietet. Die Struktur der Datenbank kann durch ein *Entitiy-Relationship-Modell* (ERM) dargestellt werden. ERMs sollen visualisieren, in welcher Beziehung verschiedene Bestandteile von Datenstrukturen (*Enities*) zu einander stehen. ERMs sind oftmals der erste Schritt wenn es um die Planung einer neuen Datenbank geht. Das ERM der im Rahmen dieser Arbeit benutzen Datenbank ist in Abbildung 5.3 visualisiert. Die Verhältnisse

	Field	Type	Collation	Attributes	Null	Default	Extra	Action
	user_id	int(8)			No			
	subj_german	tinyint(1)			No			
	subj_spanish	tinyint(1)			No			
	subj_english	tinyint(1)			No			
	subj_french	tinyint(1)			No			
	subj_biology	tinyint(1)			No			
	subj_chemistry	tinyint(1)			No			
	subj_music	tinyint(1)			No			
	subj_maths	tinyint(1)			No			
	subj_physics	tinyint(1)			No			

Check All / Uncheck All With selected:

	user_id	subj_german	subj_spanish	subj_english	subj_french	subj_biology	subj_chemistry	subj_music	subj_maths	subj_physics
	1	1	0	0	0	0	0	1	0	1
	2	0	0	0	0	0	0	1	0	1

Abbildung 5.2: Die user_subjects Tabelle, aufgenommen aus dem phpMyAdmin Interface

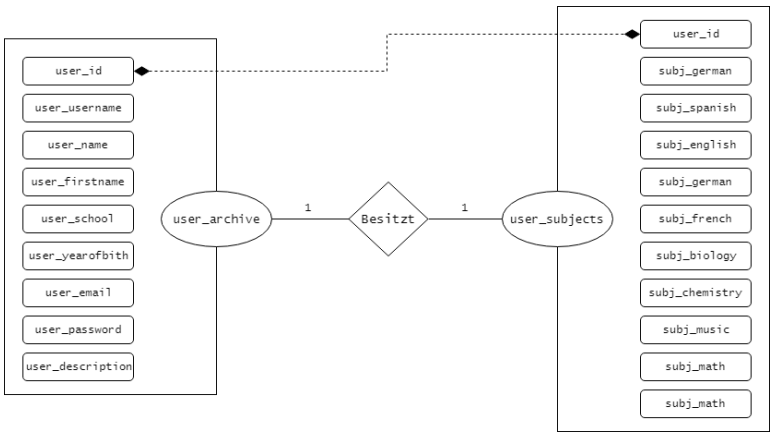

Abbildung 5.3: Das ERM der entwickelten Applikation.

werden im Schema (1, 1) geschrieben. Die erste Ziffer gibt 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, steten anstelle der Ziffern stellvertretend die Variablen m oder n. Die Tabellen sind über den Key user_id miteinander Verknüpfbar.[10, S. 750] [4] [15]

Abbildungen von den beiden Tabellen findet sich in den Abbildungen 5.1 und 5.2. Im oberen Teil der Abbildungen sieht man die Konfiguration der einzelnen Felder. In der Spalte *Field* ist der Name der Felder eingetragen. In der Spalte *Type* wird der zugelassene Datentyp des Feldes definiert. Ist der Datentyp eine Form von Text, wird in der Spalte *Collation* die Kodierung der Unicode-Zeichen angegeben. In der Spalte *Extras* können noch eine Reihe von speziellen Eigenschaften den Feldern zugewiesen werden. Eine davon ist die im Feld *user_id* verwendete *auto_increment* Funktion. Sie führt dazu, dass neue Einträge in der Tabelle in diesem Feld automatisch durchnummeriert werden. Das Feld user_id ist in beiden Tabellen als Key markiert. Das heisst, dass das die user_id von der Datenbank zur eindeutigen Verknüpfung der beiden Tabellen benutzt werden soll. Ebenfalls zu erwähnen ist das Feld *user_username*. Es ist als *unique* markiert, was heisst, dass es in der Tabelle keine identische Benutzernamen geben darf (siehe Kapitel 5.2.3.1).

Im unteren Teil der Abbildungen findet sich ein beispielhafter Ausschnitt, wie eine solche Tabelle mit ausgefüllten Feldern Aussehen könnte.

5.2.3 PHP Skripte

Nebst dem Datenbanksystem selbst befinden sich ebenfalls diverse PHP Skripte auf dem Server. Sie sind in der Open-Source Skriptsprache PHP geschrieben. PHP ist ein rekursives Akronym für *Hypertext Preprocessor* und ist eine speziell für die Webprogrammierung entwickelte Programmiersprache, welche zu den objektorientierten Programmiersprachen gehört. PHP wird hauptsächlich für die Programmierung innerhalb von Webservern verwendet und bietet den grossen Vorteil, dass Clientbenutzer zwar durchaus die Skripte auf dem Server ausführen, jedoch nicht ansehen können. Sie erhalten lediglich eine Antwort ohne je die Codestruktur des PHP Skriptes zu Gesicht bekommen zu haben.[11] Dies ist möglich, da der PHP-Code ausschliesslich serverseitig ausgeführt wird. Ein *Interpreter* führt ihn auf dem Server aus und erst die vom Interpreter generierte Ausgabe wird dem Client als Antwort zugeschickt. Die Antwort ist meist in der Auszeichnungssprache HTML (*Hypertext Markup Language*) verfasst, können jedoch durchaus auch in PDF, JavaScript, JSON oder sogar Bildformaten sein.[1] Die Website *json.org* ([9]) beschreibt JSON wie folgt:

> "JSON (JavaScript Object Notation) ist ein schlankes Da-

```
SELECT * FROM user_archive WHERE user_username=? AND
↪  user_password=?
```

Abbildung 5.4: SQL Select Query des login.php Skriptes an die user_archive Tabelle

> tenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen (Analysieren von Datenstrukturen) und zu generieren ist.[...]Bei JSON handelt es sich um ein Textformat, das komplett unabhängig von Programmiersprachen ist[...]. Diese Eigenschaften machen JSON zum idealen Format für Datenaustausch."[9]

Die im Rahmen dieser Arbeit entwickelte Applikation benutzt für ihre Kommunikation von MySQL Datenbank zu Client ausschliesslich JSON-Formate. Dieses Format eignet sich perfekt für den Austausch von Daten wie sie in der Datenbank gespeichert sind.

5.2.3.1 Login

Damit ein Client Zugriff auf einen Datensatz in der Datenbank bekommt, muss sich der Benutzer/ die Benutzerin zuerst unter einem bestehenden Account einloggen. Dabei wird auf dem Client Benutzername (Username) und Passwort eingegeben (siehe Kapitel 5.4.4.1). Daraufhin ruft der Client das Skript *login.php* via Webadresse auf und übergibt ihm Benutzername und Passwort. Das aufgerufene Loginskript erstellt eine Verbindung zur lokalen MySQL Datenbank. Als nächstes wird versucht einen Eintrag in der user_archive Tabelle zu finden, bei welchem die Felder user_username und user_password mit den empfangenen Werten übereinstimmen. Dies wird über ein sogenanntes *MySQL Statement* getan. In diesem Statement wird in der Datenbank nachgefragt, ob es einen Eintrag mit den bestimmten Angaben gibt (Select Query)(siehe Abbildung 5.4). Wenn die Suche erfolgreich ist, wird dem Statement mit dem Inhalt eines Datensatzes geantwortet. Es ist wichtig, dass jeweils nur maximal ein Eintrag dabei gefunden werden kann. Deshalb muss der Benutzername aller Benutzer/Benutzerinnen einzigartig sein (siehe Kapitel 5.2.2). Mit diesen Informationen kann nun über die user_id ebenfalls der Datensatz des Benutzers/der Benutzerin aus der Tabelle user_subject ausgelesen werden. Alle nun erhaltenen Daten aus der Datenbank werden anschliessen im JSON-Format an den Client geantwortet und teilt gleichzeitig auch noch dem Client mit, dass das Login erfolgreich war. Hat die Suche jedoch keine Ergebnisse geliefert, wird dem Client über ein JSON-Format geantwortet, dass das Login fehlgeschlagen ist.

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

Abbildung 5.5: SQL Insert Query des register.php Skriptes in die user_archive Tabelle

5.2.3.2 Registrierung

Wenn ein Benutzer/eine Benutzerin noch keinen Account hat, soll er/sie die Möglichkeit haben, einen zu erstellen. Hierzu kann er/sie auf dem Client die benötigten Daten eingeben (siehe Kapitel 5.4.4.2) und es wird eine Anfrage an das Skript *register.php* auf dem Server geschickt. Dieses Skript hat die Aufgabe in beiden Tabellen einen neuen Eintrag für den Benutzer/die Benutzerin zu erstellen, sofern die Daten legitim sind. Zuerst wird dafür eine Verbindung mit der lokalen Datenbank aufgenommen. War dies erfolgreich, wird ein Auswahlabfrage formuliert, welche nach einem Datensatz mit dem eingegebenen Benutzernamen sucht. Nur wenn kein solcher Datensatz gefunden werden kann, 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 existiert (siehe Kapitel 5.2.3.1). 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 5.5). 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 der wohl effektivste Weg, um sogenannten *SQL Injections* vorzubeugen.[16] Als nächstes soll noch ein Eintrag in der user_subjects Tabelle erstellt werden, welcher die gleiche user_id hat, wie der soeben neu erfasste Datensatz in der user_archive Tabelle. Hierzu wird über ein Statement die von der Datenbank automatisch zugewiesene user_id aus der user_archive Tabelle ausgelesen. Daraufhin wird eine weitere Insert Query mit der ausgelesenen user_id und dem Wert 0 für alle Fächer in die user_subjects Tabelle gestellt. Der Wert 0 steht in den Feldern stellvertretend für den Wert `false`, da neu erstellte Accounts standardmässig keine Fächer ausgewählt haben sollen. Schlussendlich, wird eine JSON Antwort verfasst, welche dem Client eine erfolgreiche Registrierung mitteilt.

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

Abbildung 5.6: SQL Update Query des savesettings_pw.php Skriptes

5.2.3.3 Änderungen via Einstellungen des Clients

Es soll den Benutzern/Benutzerinnen möglich sein, gewisse Angaben wie Email, Passwort oder die ausgewählten Fächer nachträglich zu verändern. Der Client bietet dafür eine Settings Activity (siehe Kapitel 5.4.4.4) mit welcher die Angaben editiert werden können. Wenn die editierten Angaben auf dem Client gespeichert werden, wird eine Anfragen an das Skript *savesettings_pw.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 aufgenommen. 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 und Passwort fragt. Das Passwort muss 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 in der Tabelle user_archive durch den neuen mit den Aktualisierten Daten ersetzt (siehe Abbildung 5.6). Anschliessend wird der Datensatz per Auswahlabfrage wieder Ausgelesen und die Daten in lokalen Variablen gespeichert. Das gleiche mit der user_subjects Tabelle gemacht. Sämtliche ausgelesenen Daten werden in einen Array gespeichert und dieser wird anschliessend im JSON Format an den Client zurückgeschickt. So wird gewährleistet, dass die lokalen Informationen auf dem Client mit Sicherheit mit denen in der Datenbank übereinstimmen.

5.2.3.4 Suchanfrage

Damit es Benutzern/Benutzerinnen möglich ist, nach anderen Benutzern und Benutzerinnen zu suchen, wird ein PHP Skript benötigt, welches die Datenbank nach Datensätzen durchsucht, welche bestimmten Kriterien erfüllen. Diese Kriterien kann der Benutzer/die Benutzerinnen auf dem Client angeben (siehe Kapitel 5.4.4.5). Es soll nach Namen und nach Fächern gesucht werden können. Das Skript serach.php übernimmt diese Aufgabe. Hierzu wird zuerst eine Verbindung mit der Datenbank aufgebaut. Ist das erfolg-

```
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
↪  user_archive.user_name LIKE ? OR [...]
```

Abbildung 5.7: SQL Select Query des search.php Skriptes mit einem Inner Join der user_archive Tabelle und er user_subjects Tabelle (Zur Übersichtlichkeit etwas gekürzt)

reich wird Select Query erstellt, die alle den Kriterien entsprechenden Datensätze zurückliefern soll.

Das MySQL Statement wird in zwei Schritten erstellt.Im ersten wird ein *String* erstellt der gleich zwei Dinge tut. Zum einen schafft er eine Verknüpfung zwischen den beiden Tabellen user_archive und user_subjects. Dies wird durch die SQL Funktion *INNER JOIN* erreicht. Diese Funktion ermöglicht es, die beiden Tabellen über die user_id (Key) zu verbinden. Nun kann die Select Query in beiden Tabellen gleichzeitig nach mehreren Kriterien suchen und die Ergebnisse direkt zusammenführen (siehe Abbildung 5.7). Weiter ist im String auch bereits das Suchkriterium bezüglich des Namens enthalten. Dazu wird die SQL Funktion *LIKE* verwendet. Diese Funktion ermöglicht es, das nach Daten gesucht wird, 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 ein obligatorisches Kriterium 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 mit Ausnahme des Passwortes werden einzeln in eigenen Arrays gespeichert. Anschliessend werden alle diese Arrays durchnummeriert in einen weiteren Array gespeichert. So sind alle Suchergebnisse kompakt und systematisch geordnet und können als ein einzelnes Element im Antwortarray referenziert werden, welcher denn an den Client zurückgeschickt wird.

5.3 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 Realtime 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 Realtime Service zu konzentrieren. Der Erfolg war gross und 2014 wurde Google darauf aufmerksam und kaufte das Unternehmen auf. 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 Realtime Datenbanken, Authentifizierungsfunktionen und Crashanalysen, aber auch für das Unterhalten von bestehenden Servicen. 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, jedoch durchaus auch für grössere Unternehmen.[12]

5.3.1 Firebase Realtime Datenbank

Wie bereits erwähnt, bietet Firebase unter anderem auch Realtime Datenbankfunktionen an. Die Firebase Realtime Datenbank (RTDB) ist eine NoSQL Datenbank und unterscheidet sich in ihrer Funktionsweise stark von relationalen Datenbanken. Anders als relationale Datenbanken agieren Echtzeitdatenbanken nicht nur Passiv auf Anfrage sondern teilen den Clients aktiv mit, wenn sich ein Datensatz verändert hat (Push-Based Data Access) und halten ihn so immer auf dem neusten Stand. 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.[17] Es ist dann nicht einmal nötig, das die Clients zur Zeit des Pushes online sind. Sie werden bei Start automatisch mit dem Datensatz auf der Datenbank abgeglichen und aktualisiert.[8]

Die Chatfunktion in der entwickelten Applikation ist ein genau solcher Dienst, der eine solche Datenbank benötigt. So können Clients in Echtzeit miteinander Kommunizieren, ohne dass sie permanent online sein müssen, oder die Applikation konstant Anfragen an den Server nach neu empfangenen Nachrichten schicken muss. Die Wahl für die Firebase Plattform ist dann

einfach gewesen. Der Client ist im ebenfalls von Google entwickelten Android Studio entwickelt worden. Android Studio Unterstützt standardmässig die Firebase API, weshalb das Einbinden einer Firebase RTDB in eine Applikation sehr einfach ist (siehe Kapitel 5.4.4.8).

5.3.2 Firebasestruktur

Firebase Echtzeitdatenbanken haben eine andere Struktur als relationale Datenbanken. Die Informationen sind nicht in Form von Tabellen gespeichert sondern können vielmehr mit der Struktur eines Ordnersystems verglichen werden. Der Ort, wo die Informationen gespeichert sind, wird in Pfaden angegeben, die mit denen eines Filesystems eines Computers vergleichbar sind. Dabei gibt es einen Hauptordner, der sozusagen die gesamte Datenbank umfasst. Diese wiederum hat Unterordner, welche im Allgemeinen als *Children* bezeichnet werden.

Die Struktur der im Rahmen diese Arbeit benützen Datenbank besteht aus zwei Ordnern, dem *User*-Ordner und dem *Chats*-Ordner. Im User Ordner sind alle Benutzer verzeichnet, die einen oder mehrere offene Chats besitzen. Der Ordner dient dazu, offene Chats eines Benutzers auf der Hauptseite des Clients darzustellen und öffnen zu können. Die Children des Verzeichnis sind unter den user_ids der Benutzer gespeichert. Für jeden offenen Chat findet sich in einem solchen Child nun ein weiteres Child für jeden offenen Chat des Benutzers/der Benutzerin. Dieses enthält die eigentlichen Informationen, die für das Öffnen und Darstellen des Chats benötigt werden. Ebenfalls enthalten sie auch den Pfad des eigentlichen Chats, wo auch die Nachrichten gespeichert sind (siehe auch Kapitel 5.3.2.1). Die offenen Chats findet man alle im Chats Verzeichnis. Dort sind als Children alle Chats zu finden. Der Name wird ihnen durch die beteiligten Chatpartner/Chatpartnerinnen gegeben. Das heisst, ein Chat zwischen der Benutzer A mit der ID 9 und einer Benutzerin B mit der user_id 21 würde unter dem Namen 9>>21 zu finden sein. Im Chatverzeichnis selber befinden sich alle gesendeten Nachrichten. Der Name der Nachrichten wird von Firebase selber generiert. Die Nachrichten enthalten nun die drei eigentlich versendeten Werte: die Nachricht, das Sendedatum und Uhrzeit und den Absender/die Absenderin.

5.3.2.1 Warum genau diese Struktur?

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 trotzdem möglich, alle offenen Chats eines Benutzers/einer Benutzerin anzuzeigen, ohne dass eine Suchfunktion benötigt wird.

5.4 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. Der Client, welcher im Rahmen dieser Arbeit entwickelt wurde, ist für Android Geräte programmiert worden. Er wurde in der Programmierumgebung *Android Studio* entwickelt. Android Studio ist eine von Google entwickelte Programmierumgebung (IDE) für die Entwicklung von Java Applikationen für Android Mobilgeräte. Die Umgebung bietet diverse hilfreiche Tools für Entwickler wie das Erstellen von virtuellen Geräten für das sichere Testen der Applikationen oder eine Umgebung für das Gestalten von User Interfaces. Somit viel die Wahl für die Entwicklung diese Applikation sehr schnell auf Android Studio, da die Umgebung so gut wie alles besitzt, was für die Entwicklung einer solchen Applikation gebraucht wird.

5.4.1 Programmiersprache

Die Applikation wurde vorwiegend in der Programmiersprache Java entwickelt. 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 besonders gut für kleinere Applikationen, die auf vielen verschiedenen Geräten funktionieren sollen.

Das Betriebssystem Android, für welche die Applikation entwickelt worden ist, ist das weltweit meist verwendete Betriebssystem für Mobiltelefone und unterstützt standardmässig die auf Java basierte Laufzeitumgebung *Android Runtime*. Somit werden fast alle Applikationen für Android in der Sprache Java entwickelt, weshalb auch der Client dieser Arbeit in Java geschrieben wurde, um eine möglichst grosse Menge an Benutzern/Benutzerinnen zu erreichen.[5]

5.4.2 Architektur des Clients

Der Client der Applikation besteht aus insgesamt zehn sogenannten *Activities*. Activities entsprechen den verschiedenen Fenstern die in einer Applikation geöffnet werden können. Eine Übersicht über alle Activites findet sich in Abbildung **??**. Activities können grundsätzlich in zwei Teile aufgeteilt werden: eine *View*, welche in der Layoutsprache XML geschrieben ist und das Userinterface der Activtiy definiert und die in Java programmierte *Presenter*-Klasse (siehe Kapitel 3.2). Ein Beispiel, wie eine solche Activity genau aufgebaut ist, findet sich in Kapitel **??**. An dieser Stelle ist anzumerken, dass die ab hier verwendeten Zeilenangaben 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.

5.4.2.1 Klassenübersicht

Nebst den Views und ihren jeweiligen Presenter sind auch noch eine Reihe von anderen Klassen verwendet worden.

Die Request Klassen Für die Anfragen an den Server werden Request Klassen verwendet. Sie sind Instanzen der Klasse *StringRequest* und erstellen aus den zu versendenden Daten eine lokale Hashmap auf dem Client. Hashmaps sind eine effiziente Lösung, um Reihen von Daten systematisch in einem Element zu speichern. Die Request Klassen beinhalten ebenfalls die URL der PHP-Files, an welches die Anfrage gesendet werden. Die Request Klassen können, da sie von der StringRequest Klasse abgeleitet sind, in sogenannten *Request Queues* als Anfragen an den Server ausgeführt werden. Die für diese Aufgabe verwendeten Funktionen finden sich alle in der *Volley* Library. Volley bietet diverse Tools für das effiziente Arbeiten mit Netzwerken und eignet sich daher perfekt für diese Aufgabe.[2]

Die Adapter Klassen Adapter Klassen sind bestimmend für das Verhalten von Auflistungen. Es gibt einen Adapter für die Anzeige der Suchergebnisse, den Chat und dem Bottomsheet von der Haupseite, wo die offenen Chats angezeigt werden. Adapter geben der Applikation vor, wie die verschiedenen Elemente einer Liste anzuzeigen sind. Für die Suchergebnisse wird dafür eine Instanz der Klasse *Base Adapter* aus der *Widget* Library verwendet. Die anderen beiden Adapter sind Instanzen der *Firebase Recycler Adapter* Klasse aus der Firebase Library. Sie sind für das Auflisten von Objekten in der Firebase Datenbank verantwortlich und haben jeweils eine Referenz zu einem Verzeichnis in der Firebase Datenbank, dessen Inhalt sie auflisten.

```
1  public userInfo(int id, String username, String name, String
   ↪  firstname, int yearofbirth, String description, boolean
   ↪  french, boolean spanish, boolean music, boolean english,
   ↪  boolean chemistry, boolean biology, boolean maths, boolean
   ↪  german, boolean physics) {
2      this.id = id;
3      this.username = username;
4      this.name = name;
5      this.firstname = firstname;
6      this.yearofbirth = yearofbirth;
7      this.description = description;
8      this.french = french;
9      this.spanish = spanish;
10     this.english = english;
11     this.music = music;
12     this.chemistry = chemistry;
13     this.biology = biology;
14     this.maths = maths;
15     this.physics = physics;
16     this.german = german;
17 }
```

Abbildung 5.8: Konstruktor der userInfo Klasse

Die userInfo und JSONtoInfo Klasse Damit die einzelnen Activities nicht immer wieder von Neuem Anfragen an den Server senden müssen, werden die benötigten Informationen jeweils an die nächste Activity weitergegeben. So muss nur einmal zu Beginn beim Login eine Anfrage an die Datenbank stattfinden. Die Informationen werden zwischen den Activities im selben JSON String weitergegeben, wie sie vom Server empfangen wurden. Der String ist kompakt und kann in nur einem String weitergegeben werden. Er eignet sich jedoch nicht zur Verwendung in den Activities selber. Hierzu gibt es die userInfo Klasse. Sie fungiert als eine Art „Variabelcontainer“ und besitzt sämtliche Eigenschaften, die ein Benutzerprofil besitzt (siehe Abbildung 5.8). Dazu gehört Namen, Fächer, Passwort etc., wobei das Passwort jedoch nicht direkt über den Konstruktor gesetzt wird, da die Klasse auch für die Informationen anderer Benutzer benutzt wird. Diese gespeicherten Variablen können dann in den Activities an den angeforderten Stellen über sogenannte *Getter*-Funktionen ausgelesen werden. Die JSONtoInfo Klasse wiederum besteht aus nur einer Funktion, die in der Lage ist, den JSON-String in eine Instanz der userInfo Klasse zu speichern. So wird in jeder Activity zuerst aus dem JSON-String eine Objekt der userInfo Klasse

```
1 userInfo clientInfo = new JSONtoInfo().createNewItem(new
  ↪  JSONObject(extrasBundle.getString("clientInfo")));
```

Abbildung 5.9: Instantiierung eines userInfo Objektes aus dem JSON String aus den Extras

instantiiert, welches nachher für den Zugriff auf die Accountinformationen gebraucht werden kann (siehe Abbildung 5.9).

5.4.3 Die Mainpage Activity

Die Mainpage Activity ist der zentrale Screen der Applikation. Nachdem ein Benutzer/eine Benutzerin sich erfolgreich eingeloggt hat, wird er/sie auf diesen Screen weitergeleitet. Von hier aus können alle Features der Applikation erreicht werden. Die Activity selber besteht aus den standardmässigen zwei Hauptkomponenten, der XML Datei, die das Layout definiert, und dem Java Sourcecode, der das Verhalten des Screens bestimmt. Auf die XML-Datei soll hier nicht gross eingegangen werden, das fertige Layout findet sich in Abbildung **??**, der dazugehörige XML-Sourcecode befindet sich im Anhang. Der Java Sourcecode wiederum besteht im Wesentlichen aus drei Teilgebieten: der onCreate Methode, dem Listener für die verschiedenen Schaltflächen und aus einer Reihe von Komponenten, die das Auflisten und Öffnen der Chats via des Bottomsheets ermöglichen.

5.4.3.1 Die onCreate Methode

Die onCreate Method ist der Kern einer Activity. Sie wird aufgerufen, wenn eine neue Activity gestartet wird und wir in der entwickelten Applikation hauptsächlich dazu verwendet, das Layout zu definieren und bestimmte Variablen zu instantiieren (siehe Abbildung 5.10). Im folgenden Abschnitt wird direkt auf den abgebildeten Codeausschnitt eingegangen.

Zu Beginn der onCreate Methode wird in Zeile 4 auf das zu verwendete XML-Layout Dokument verwiesen, welches verwendet für den Screen werden soll. In Zeile 6 wird dann auf den zu erscheinenden Titel des Screens verwiesen. Dieser ist in einem separaten XML Dokument gespeichert und beinhaltet hier den Text "Your Profile". In Zeile 7 werden die vom Login erhaltenen Extras mit dem JSON String in einer lokalen Variable referenziert. Diese Referenz kann dann auf Zeile 10 benutzt werden, um ein userInfo Objekt daraus zu instantiieren (siehe Kapitel 5.4.2.1). In den Zeilen 14 und 15 werden dann die Referenzen zu den im einzelnen im Layout verwendeten Elementen erstellt. Hier sind das beispielhaft zwei Textfelder für den Namen des Benutzers/der Benutzerin und die Beschreibung. Über diese Referenzen kann dann später in den Zeilen 19 und 20 der Text, der in den Textfeldern

erscheinen soll, bestimmt werden. Der Text wird dazu aus dem userInfo Objekt über eine sogenannte *Getter* Methode ausgelesen.

5.4.3.2 Die Listener

Damit die entwickelte Applikation in der Lage ist, auf bestimmte Aktionen des Benutzers/der Benutzerin zu reagieren, werden sogenannte Listener verwendet. In ihnen wird bestimmt, was geschehen soll, wenn ein bestimmtes Ereignis, wie zum Beispiel ein Tippen auf eine Schaltfläche, auftritt. Die Listener werden in separaten Klassen definiert und implementieren alle das Interface *View.OnClickListener*. Durch das Implementieren dieses Interfaces kann innerhalb der Klasse in Zeile 4 eine *onClick* Methode definiert werden. Sie wird ausgeführt, wenn auf die View, welchem der Listener zugeschrieben worden ist, geklickt wird. Eine solche View kann zum Beispiel ein Textfeld oder eine Schaltfläche sein. Ein Beispiel für einen solchen Listener findet sich auf Abbildung 5.11.

Die Listener Klasse in Abbildung 5.11 ist die Listenerklasse der Schaltfläche, die den Benutzer/die Benutzerin zu den Einstellungen der Applikation bringt. Dazu wird beim Klicken auf die Schaltfläche eine neuer sogenannter *Intent* gestartet. Der Intent selber wird in Zeile 5 instantiiert und beschreibt einen Wechsel von der momentanen Mainpage Activity zur SettingsOverview Activity. Dem Intent werden in Zeile 6 die nötigen Extras mitgegeben, die für den Start der Activity benötigt werden. In Zeile 7 wird dann der Intent ausgeführt.

5.4.3.3 Auflistung der offenen Chats

Damit auf der Mainpage Activity im Bottomsheet die offenen Chats eines Benutzers/einer Benutzerin aufgelistet und geöffnet werden können, werden eine Reihe von Methoden und Klassen verwendet. Dazu gehört eine Firebase Adapter Klasse, die OpenChat Klasse, eine ViewHolder Klasse und eine onStart Methode.

Die OpenChat Klasse Die OpenChat Klasse ist der Datencontainer einzelner Elemente der Auflistung. Sie hat die Aufgabe, die für die Auflistung und das Öffnen des Chats benötigten Variablen und Werte zu speichern und wiedergeben zu können. Ihr Aufbau ist vergleichbar mit der clientInfo Klasse. Der Konstruktor der Klasse ist in Abbildung 5.12 dargestellt. Wichtig ist, dass die Namen der einzelnen Variablen in Zeile 1 genau mit den Namen der einzelnen Verzeichnisse in der Firebase Datenbank übereinstimmen. Dies ist eine Notwendigkeit, da die Klasse von der Firebase API als ein Model für die Firebase Datenbank verwendet wird und Firebase selber die OpenChat Klasse mithilfe der in der Datenbank gespeicherten initialisiert.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_mainpage_activity);
     getSupportActionBar().setHomeButtonEnabled(false);
     getSupportActionBar().setTitle(R.string.clientprofile_title);
     extrasBundle = getIntent().getExtras();

     try {
          clientInfo = new JSONtoInfo().createNewItem(new
          ↪  JSONObject(extrasBundle.getString("clientInfo")));

          [...]

          final TextView nameandfirstname_tv =
          ↪  findViewById(R.id.userprofileact_name_textview);
          final TextView description_tv =
          ↪  findViewById(R.id.userprofileact_description_textview);

          [...]

          nameandfirstname_tv.setText(clientInfo.getFirstname()
          ↪  + " " + clientInfo.getName());
          description_tv.setText(clientInfo.getDescription());

          [...]
}
```

Abbildung 5.10: Ausschnitt aus der onCreate Methode der Mainpage Activity

```
class onSettingsListener implements View.OnClickListener  {

     @Override
     public void onClick(View view) {
          Intent settings_intent = new
          ↪  Intent(mainpage_activity.this,
          ↪  settingsOverview.class);
          settings_intent.putExtra("clientInfo",
          ↪  extrasBundle.getString("clientInfo"));
          startActivity(settings_intent);
     }
}
```

Abbildung 5.11: onSettingsListener Klasse der Mainpage Activity

```
public OpenChat(String subject, String receiverName, String
↪  latestMessage, String chatPath, String receiverRef, String
↪  senderRef, int receiverID) {
     this.subject = subject;
     this.receiverName = receiverName;
     this.receiverID = receiverID;
     this.latestMessage = latestMessage;
     this.chatPath = chatPath;
     this.senderRef = senderRef;
     this.receiverRef = receiverRef;
}
```

Abbildung 5.12: Der Konstruktor der OpenChat Klasse

```
1  public static class OpenChatViewHolder extends
   ↪  RecyclerView.ViewHolder {
2
3       View view;
4
5       public OpenChatViewHolder(View itemView) {
6            super(itemView);
7            view = itemView;
8       }
9
10      public void setChat(String receiverName, String
        ↪  latestMessage) {
11           TextView cName_tv = (TextView)
             ↪  view.findViewById(R.id.openchat_name_textview);
12           cName_tv.setText(receiverName);
13           TextView cLatest_tv = (TextView)
             ↪  view.findViewById(R.id.openchat_latest_textview);
14           cLatest_tv.setText(latestMessage);
15      }
16 }
```

Abbildung 5.13: OpenChatViewHolder Klasse der Mainpage Activity

Die ViewHolder Klasse Die *OpenChatViewHolder* Klasse ist ein abgeleitet Klasse der Klasse *RecyclerView.ViewHolder*. Sie besitzt einen Konstruktor und eine setChat Funktion. Ihre Aufgabe ist es, das Layout der einzelnen offenen Chats zu konfigurieren. Sie selber nimmt jeweils bei der Instanziierung eine Referenz zu einer View entgegen, die in der Liste erscheinen soll. Wird dann anschliessend die setChat Methode aufgerufen, setzt der ViewHolder die ihm mitgegebenen Variablen in die vom Konstruktor referenzierte View ein. Im Falle des in Abbildung 5.13 dargestellten ViewHolders sind das der Name des Empfängers und die zuletzt empfangene Nachricht.

Die Firebase Adapter Klasse Die Firebase Adapter Klasse ist das Herzstück der Auflistung. Sie ist von der *FirebaseRecyclerAdapter* abgeleitet und ist sozusagen die Schnittstelle der Firebase Datenbank und der Auflistung im Bottomsheet des Clients. Beim Initialisieren der Klasse werden alle dazu benötigten Objekte entgegengenommen. Eine Abbildung des Konstruktors findet sich in Abbildung 5.14. Zu den für die Initialisierung benötigen Objekten gehört zum einen die OpenChat Klasse. Sie wird in der Liste als Model für die Daten auf der Firebase Datenbank verwendet. Ebenfalls referenziert wird die ViewHolder Klasse und das für für die einzel-

```
1 public BottomsheetFirebaseAdapter(Class<OpenChat> modelClass,
  ↪ int modelLayout,
  ↪ Class<mainpage_activity.OpenChatViewHolder>
  ↪ viewHolderClass, Query ref, Context context, String
  ↪ clientName, String clientInfo) {
2
3       this.mContext = context;
4       this.clientName = clientName;
5       this.clientInfo = clientInfo;
6       super(modelClass, modelLayout, viewHolderClass, ref);
7 }
```

Abbildung 5.14: Konstruktor der Firebase Adapter Klasse

nen Items zu verwendende Layout. Das Layout ist im XML Format verfasst und wird innerhalb der Liste für jeden offenen Chat einzeln generiert und ein ViewHolder dafür instantiiert, der das Layout an das Model anpasst. Die vier letzten übergebenen Werte sind zum einen eine Referenz zu der zu verwendenden Datenbank, der Kontext, um einen neuen Chat öffnen zu können, der volle Name des Benutzers/der Benutzerin und der JSON String mit den Client Informationen. Sie werden dann in den Zeilen 4 bis 5 den entsprechenden Variablen in der Klasse zugewiesen. In Zeile 6 wird dann der von der Firebase API bereitgestellte eigentliche FirebaseRecylclerAdapter aufgerufen, der dann auch die einzelnen Elemente in die Liste einfüllt.

Die zweite Methode in der Firebase Adapter Klasse ist die *populateViewHolder* Methode. Sie ist in Abbildung 5.15 dargestellt und wird jedes Mal nachdem ein neuer Eintrag in die Liste aufgenommen worden ist aufgerufen. Sie hat die Aufgabe, die setChat Funktion der ViewHolder Klasse des Elements aufzurufen und dem Element einen Listener zuzuschreiben, der bei einem Klick auf das Element den dazugehörigen Chat öffnet. Die setChat Funktion wird in Zeile 4 aufgerufen, der Listenner wird dem Objekt dann in Zeile 5 zugewiesen. Der Listener wird dann in den folgenden Zeilen 6 bis 13 definiert.

Die onStart Methode Die onStart Methode ist das letzte Puzzle Teil für die Darstellung der offenen Chats. Sie wir jeweils beim starten der Activity ausgeführt und hat die Aufgabe, den Firebase Adapter zu initialisieren und der RecyclerView im Bottomsheet zuzuweisen. Sie ist in Abbildung 5.16 abgebildet. Dabei werden in den Zeilen 6 bis 12 alle mitzugebenden Objekte aufgeführt. Die meisten davon sind bereits schon in der onCreate Methode definiert worden. Zuletzt wird dann in Zeile 15 der Firebase Adapter dem

```
1  @Override
2  protected void populateViewHolder
   ↪  (mainpage_activity.OpenChatViewHolder viewHolder, final
   ↪  OpenChat model, int position) {
3
4       viewHolder.setChat(model.getReceiverName(),
        ↪  model.getLatestMessage());
5       viewHolder.view.setOnClickListener(new
        ↪  View.OnClickListener() {
6
7            @Override
8            public void onClick(View view) {
9
10           [...]
11
12           }
13      });
14 }
```

Abbildung 5.15: populateViewHolder Methode der Firebase Adapter Klasse

RecyclerView im Bottomsheet zugewiesen.

5.4.4 Module

5.4.4.1 Login Activity

Der Login Screen ist der erste Screen, den ein Benutzer/eine Benutzerin zu Gesicht bekommt, wenn er/sie die Applikation startet. Er fordert den Benutzer/die Benutzerin auf, seinen/ihren Benutzernamen und das Passwort einzugeben. Das Design ist sehr allgemein gehalten und ist vergleichbar mit dem vieler anderer Login Screens anderer Applikationen. Über der Eingabe ist das Logo der Applikation zu sehen. Es soll das sonst sehr generische Login einzigartiger machen, um mit einem guten Logo dem Benutzer/der Benutzerin gleich in Erinnerung zu bleiben. Unter dem auffälligen Login Knopf findet sich der Link zum Registrierungsformular. Es ist bewusst unauffällig gestaltet, da jeder Benutzer es voraussichtlich nur einmal brauchen wird und es sonst das Bild stören würde. Das Design und der Text des Links sind inspiriert von anderen Loginformularen populärer sozialer Netzwerke wie Instagram oder Twitter. Wenn ein Benutzer/eine Benutzerin auf den Link klickt, öffnet sich die Register Activity (siehe Kapitel 5.4.4.2). Wenn die Login Schaltfläche betätigt wird, wird ein login_request mit eingegebenem Benutzernamen und Passwort generiert und das login.php Skript auf

```
1      @Override
2  protected void onStart() {
3       super.onStart();
4
5       FirebaseRecyclerAdapter FCBA = new
        ↪  BottomsheetFirebaseAdapter(
6            OpenChat.class,
7            R.layout.openchat,
8            OpenChatViewHolder.class,
9            databaseReference,
10           mainpage_activity.this,
11           String.format("%s %s",clientInfo.getFirstname(),
             ↪  clientInfo.getName()),
12           extrasBundle.getString("clientInfo")
13      );
14
15      recView.setAdapter(FCBA);
16  }
```

Abbildung 5.16: Konstruktor der Firebase Adapter Klasse

dem Server aufgerufen (siehe Kapitel 5.2.3.1). Ist die Anfrage erfolgreich, wird der JSON String in die *Extras* (Datencontainer, mit welchem Werte zwischen Activities weitergegeben werden können) gespeichert und die Hauptseite gestartet(siehe Kapitel 5.4.4.3).

Der Presenter des Logins besteht aus vier Komponenten. Die erste Methode ist Standardmethode *onCreate*, die das XML Layout des Logins aufruft und gewisse Variablen instantiiert. Diese Methode existiert bei allen Activities und wird später nicht mehr speziell erwähnt. Weiter gibt es noch die beiden *Listenerklassen* für die Login Schaltfläche und das Label für die Registrierung. Sie bestimmen, was geschehen soll wenn auf diese Elemente gedrückt wird. Zuletzt gibt es noch die *Responselistener* Klasse. Sie ist für das Verarbeiten der vom Server empfangenen Antwort verantwortlich.

5.4.4.2 Register Activity

Das Registrierungsformular fragt den Benutzer/die Benutzerin nach allen Informationen, die benötigt werden, um ihm/ihr einen eigenen Account zu erstellen, unter welchem sie sich einloggen kann. Dazu wird nach einem Benutzernamen, einem Vor- und Nachnamen, dem Geburtsjahr, der aktuell besuchten Schule, eine Email und einem Passwort gefragt. Zudem gibt es noch eine Checkbox zum Bestätigen, dass die AGBs akzeptiert werden und

eine Schaltfläche um sich schliesslich zu registrieren. Wenn diese angewählt wird, wird zuerst geprüft, dass auch alle Felder richtig ausgefüllt sind. Ist das der Fall, wird ein register_request vorbereitet und eine Anfrage an das register.php Skript auf dem Server geschickt. Teilt die Antwort einen Erfolg beim Erstellen eines neuen Accounts mit, so wechselt der Bildschirm automatisch zurück zum Login Screen. Sollte ein Benutzer/eine Benutzerin trotz keiner erfolgreichen Registration zurück zum Login Screen wollen, so kann sie den Pfeil im linken oberen Ecken verwenden, welcher sie zurück auf die Login Seite bringt. Diese Pfeile sind mittlerweile Standard geworden innerhalb von Android wie aber auch IOS Applikationen. Da sich die Benutzer wahrscheinlich bereits an diese Funktion gewöhnt haben, wird sie auch innerhalb der Applikation mehrmals verwendet, um die Bedienung der Applikation möglichst natürlich und intuitiv zu gestalten.

In der Presenter-Klasse finden sich die fünf Hauptkomponenten der Activity: der Listener für die Registrier-Schaltfläche, der Responselistener für die Registrierung, die onCreate Methode, eine Methode für den Zurück-Knopf und noch eine Methode, die die letzen 100 Jahre in den Spinner einfügt, der benötigt wird, um den Jahrgang anzugeben. Die Methode für den Spinner ermöglicht es, dass die Applikation automatisch neue mögliche Jahrgänge einfügt und nicht manuell jedes neues Jahr eingetragen werden muss.

5.4.4.3 Mainpage Activity

Die *Mainpage Activity* ist sozusagen das Herz der Applikation. Sie wird direkt nach erfolgreichem Login geöffnet und ist sozusagen das eigene Profil des Benutzers/der Benutzerin. Von hier aus können verschiedene Dinge gemacht werden. Zum einen kann man mithilfe einer Schaltfläche im rechten oberen Ecken in die Einstellungen gelangen (siehe Kapitel 5.4.4.4). Weiter kann über die Schaltfläche im unteren rechten Bereich des Bildschirms die Suchfunktion der Applikation ausgeführt werden. Die Schaltfläche soll möglichst offensichtlich platziert sein, damit der Benutzer/die Benutzerin ohne Probleme die Suchfunktion findet. Ebenfalls kann durch ein ausfahrbares Bottomsheet auf offenen Chats zugegriffen werden. Das Profil selber besteht aus den ausgewählten Fächern, welche ähnlich wie Medaillen dargestellt werden, einem später setzbaren Profilbild mit Name des Benutzers/der Benutzerin und einer Beschreibung.

Der Presenter der Mainpage beinhaltet die sechs Komponenten der Activity. Zum einen enhält sie wieder die onCreate Methode, die hier hauptsächlich die Informationen, welche vom Server empfangen wurden in die Felder einfügt. Weiter gibt es die beiden Listener für sowohl die Schaltfläche für die Suche wie auch für die Schaltfläche zu den Einstellungen. Dazu kommt noch ein weiterer Listener, der beschreibt, wie das Bottomsheet ausgefahren werden soll. Zuletzt kommt noch eine onStart Methode und einen View Holder. Der View Holder beschreibt lediglich, wie die einzelnen Viewkomponenten des

Layouts des offenen Chats gefüllt werden sollen. Die onStart Methode ist ähnlich wie die onCreate Methode, ist hier jedoch vor für das Bottomsheet von Bedeutung. Sie weist dem Bottomsheet einen der bereits erwähnten Firebase Adapter zu und gibt den View Holder mit. Somit kann das Bottomsheet mit allen offenen Chats ausgefüllt werden. Die einzelnen offenen Chats bekommen ihren Listener vom Firebase Adapter und nicht von der Mainpage Activty.

5.4.4.4 Settings Activity

Die *Settings Activity* teilt sich in drei Activites auf: der *settingsOverview Activity*, der *profileSettings* Activity und der *securitySettings* Activity. Die settingsOverview Activity ist lediglich ein Navigator. In ihr finden sich die beiden Verweise zu den anderen Einstellunsactivites. Dies ist vor allem bei Mobilapplikationen weit verbreitet. Da der Screen nur sehr beschränkten Platz zur Verfügung hat, werden volle Formulare schnell unübersichtlich. Dann lohnt es sich das Formular in verschiedene kleinere Formulare aufzuteilen, wie es auch hier gemacht worden ist. In der securitySettings Activty können Dinge wie Passwort oder Email bearbeitet und anschliessend gespeichert werden. In der profileSettings Activity wiederum sind die Einstellungen bezüglich des Profils wie Name, Fächer und Schule zu finden. Wenn die Einstellungen beider Activies gespeichert werden wird ein savesettings_pw_request erstellt und an das savestetting_pw.php eine Anfrage geschickt. Bei Erfolg ändert sich der Screen automatisch zurück zu Übersicht.

5.4.4.5 Filter Activity

Die Filter Activity wird von der Hauptseite aus erreicht und ermöglicht dem Benutzer/der Benutzerin ihre Suchkriterien einzugeben. Dazu gehören Name und Fach. Der Name kann optional in ein Textfeld eingegeben werden während die Fächer über Checkboxen ausgewählt werden können. Darunter findet sich eine Schaltfläche, der die effektive Suche startet. Es wurde bewusst auf eine dynamische Suchfunktion verzichtet, da sonst das Einbauen des Filtern nach Fächern unübersichtlich geworden wäre. Mithilfe dieser Filter Activity können sich Benutzer/Benutzerinnen zuerst ihre Suchkriterien auswählen und sich anschliessend die Ergebnisse evaluieren. Beim Betätigen der Such-Schaltfläche unter den Kriterien wird ein search_request erstellt, der eine Anfrage an das search.php File schickt. Wenn die Suche Ergebnisse liefert, wird die Searchresults Activity gestartet.

Der Presenter der Filter Activity besteht aus lediglich vier Komponenten: der onCreate Methode, der Methode für den Pfeil zurück zur Mainpage, eine Listener für den Suchknopf und noch einem ResponseListener.

5.4.4.6 Searchresults Activty

Die *Searchresults Activity* hat die Aufgabe, die vom Server gefundenen Suchergebnisse anzuzeigen. Das Layout der Activity selber besteht lediglich aus einer Liste. Spannender ist da das Layout der einzelnen Komponenten der Liste. Jede Komponente repräsentiert ein Suchergebnis und somit ein Profil eines Nutzers/einer Nutzerin. Die einzelnen Elemente sind so gestaltet, dass beim Durchscrollen gleich alle wichtigsten Informationen herausgelesen werden können. Dazu gehören Namen, Geburtsjahr, Fächer und Profilbild. In die Liste eingefüllt werden die einzelnen Komponenten von einem Adapter, der jeweils das Layout bestimmt und den Informationen des Profils anpasst. Damit die einzelnen Elemente auch angewählt werden können, wird noch ein Listener benötigt, der den jeweiligen Elementen auch ein bestimmtes Profil zuweist und die Userprofile Activity starten kann. Diese wird in der Searchresults Activity selbst als ein Child der Klasse *onItemSelectedListener* bestimmt. Wenn nun ein Element der Liste ausgewählt, so holt sich der Listener die Indexnummer, an welcher das Element steht. Dieser Index entspricht dem Index der Profilinformationen im erhaltenen Antwortarray (siehe Kapitel 5.2.3.4) und somit können die im Array gespeicherten Informationen über den Index an die Userprofile Activity weitergegeben werden.

5.4.4.7 Userprofile Activty

Die *Userprofile Activity* stellt das Profil eines anderen Benutzers dar. Vom Layout her ist sie fast identisch zur Mainpage Activity, nur dass sowohl die Settings-Schaltfläche wie auch die Suchschaltfläche fehlt. Dafür findet sich gross und auffällig eine Schaltfläche für das Öffnen eines neuen Chats. Es soll für den Benutzer/die Benutzerin sofort klar sein, wie er/sie die Möglichkeit hat, mit diesem anderen Benutzer/der anderen Benutzerin Kontakt aufzunehmen. Wird diese Schaltfläche betätigt, öffnet sich ein neuer Chat (siehe Kapitel 5.4.4.8).

Der Presenter der Userprofile Activity besteht aus insgesamt drei Komponenten: der onCreate Methode, der Methode für den Pfeil zurück und noch dem Listener für die Schaltfläche, die den Chat öffnet. Dieser Listener hat jedoch nicht nur die Aufgabe, die Chat Activity zu öffnen, sondern erstellt dem beiden Benutzern/Benutzerinnen einen Wert in der Firebase Datenbank. Hier wird für einen Benutzer/eine Benutzerin, wenn das sein/ihr erster Chat ist, ein Profil im *Users* Verzeichnis in der Datenbank angelegt. Darin wird auch gleich der neu geöffnete Chat eingetragen, um ihn dann später auf der Mainpage anzeigen zu können. Hat der Benutzer/die Benutzerin bereits einen Profil, wird lediglich der neu geöffnete Chat darin eingetragen. Dies geschieht gleich bei beiden Chatpartnern, was gewährleistet, dass auch der Empfänger/die Empfängerin den neu geöffneten Chat sieht und die Nachrichten empfangen kann.

5.4.4.8 Chat Activity

Die *Chat Activtiy* ist die Activity, die die Kommunikation zwischen verschidenen Benutzerinnen und Benutzern ermöglicht. Sie ist in der Lage, Chatverläufe zwischen zwei Accounts darzustellen. Das Layout ist dabei reduziert auf eine List View, in welcher die Nachrichten dargestellt werden, eine Texteingabe und einen Sendeknopf. Das Layout ist stark inspiriert von anderen Chatfunktionen von Applikationen wie Whatapp oder der normalen SMS Messaging Funktion auf Mobiltelefonen. Auch hier soll sich der Benutzer/die Benutzerin direkt vertraut fühlen und wissen, wie es funktioniert. Die Nachrichten werden in zwei verschiedenen Layouts dargestellt. Das eine ist für empfangene Nachrichten, das andere für gesendete. Dieses Layout wird den Nachrichten vom Firebase Adapter zugewiesen, welcher anhand des Namens des Senders entscheidet, ob eine Nachricht selber geschickt wurde, oder vom Client empfangen wurde.

Aufgrund der Tatsache, dass der Chat auf einer Echtzeitdatenbank basiert, ist es sogar möglich, dass beide Chatpartner/Chatpartnerinnen gleichzeitig Nachrichten verfassen können und beide sofort bei erfolgreichem Senden die jeweilig empfangene Nachricht sehen können (siehe Kapitel 5.3.1).

Die Presenter der Chat Activity beinhaltet fünf Komponenten: die onCreate Methode, die hier alle wichtigen Variablen instantiiert, der onStart Methode für das zuweisen des Firebase Adapters, einer Viewholderklasse für den Adapter, einem Listener für den Sendeknopf und zu Schluss noch der Methode für den Pfeil zurück. Die Methode für den Pfeil fällt hier etwas komplexer aus, da ein Chat sowohl von der Mainpage, wie auch von einem Benutzerprofil geöffnet werden kann. Also muss hier zuerst unterschieden werden, von wo ein Benutzer/eine Benutzerin überhaupt in den Chat gekommen ist. Je nach dem wird anschliessen dann die passende Activity gestartet. Der Listener des Sendeknopfes hat die Aufgabe, einen neuen Eintrag für die gesendete Nachricht im Firebase Datanbanksystem zu erstellen. Dafür *pusht* er einen neuen Eintrag in das Verzeichnis des Chats. Dieser Eintrag beinhaltet Sendername, Sendedatum und Zeit sowie die gesendete Nachricht.

5.5 Kritik und Verbesserungspotenzial

Das Ziel der Arbeit war, eine funktionsfähige Applikation zu entwickeln, die die Vermittlung von Nachhilfe vereinfachen sollte. Dieses Ziel ist erreicht worden. Jedoch fehlt es der Applikation noch an einer Reihe essentieller Features, bevor sie öffentlich angeboten werden kann.

5.5.1 Passwortsicherheit

Die Applikation ist in Sachen Sicherheit noch unvollständig. Ein Beispiel dafür ist die Handhabung der Passwörter innerhalb der Applikation. Die Passwörter werden unverschlüsselt zwischen Server und Client hin und her geschickt und ebenfalls unverschlüsselt in der Datenbank gespeichert. Generell sollten Passwörter nie in ihrer reinen Form irgendwo gespeichert oder verschickt werden. Ein Lösungsansatz für das sichere Speichern von Passwörtern wäre das *Hashen* und *Salzen* der Passwörter. 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. Doch, um zu vermeiden, dass zwei Benutzer innerhalb einer Tabelle den gleichen Hash aufweisen, da sie zufällig das gleiche Passwort haben, gibt es ein zweites Verfahren: das Salzen von Hasheingaben. Salzen ist ein Verfahren, bei welchem jeweils vor oder nach der eigentlich zu hasenden Eingabe noch eine Reihe an zufällig generierter Werte angehängt wird. Diese Werte sind dann das *Salz* des Hashes, da sie den Hash komplett verändern. Die Idee ist, dass jeweils jeder Benutzer sein eigenes zufällig generiertes Salz zu seinem Passwort bekommt, bevor es gehasht und gespeichert wird. Das Salz wird ebenfalls in der Datenbank gespeichert und muss dabei nicht einmal verschlüsselt werden. Es kann direkt neben dem gehashten Passwort sein.

Dieses Verfahren des Hashen und Salzen von Passwörtern ist eine sehr weit verbreitete Methode und wird als Sicher betrachtet, solange es richtig gemacht wird. Mögliche Fehlerquellen dabei ist das benutzen des gleichen Salzes für mehrere Benutzer oder das generieren von zu kurzen Salzen, was ebenfalls zu Überschneidungen gleicher Salze führen kann. Im allgemeinen gelten Salze mit einer Länge von über 16 Bytes als sicher. [13] [6]

5.5.2 Firebase Zugriffsrechte

Die Firebase Datenbank hat momentan noch keine aktivierte Zugriffsüberwachung aktiviert. So ist es theoretisch jedem möglich, auf die Datenbank zuzugreifen und Daten auszulesen oder zu modifizieren. Firebase bietet für die

Lösung dieses Problems diverse Möglichkeiten um den Zugriff von Benutzern/Benutzerinnen und deren Zugriffsrechte zu kontrollieren. Weiter kann dabei sogar noch in Betracht gezogen werden, den Benutzern/Benutzerinnen die Möglichkeit zu bieten, sich via Google Login in der Applikation zu Registrieren. Dies ist ebenfalls ein Feature, das Firebase Entwicklern zur Verfügung stellt. Es würde den Nutzern/Nutzerinnen die Registrierung weitgehend ersparen, da viele Android Nutzer bereits einen Google Account besitzen.

5.5.3 IOS Unterstützung

Nebst des weitverbreiteten Android Betriebssystem ist ebenfalls das von Apple Produkten verwendete Betriebssystem IOS besonders in Ländern wie der Schweiz von bedeutender Beliebtheit. Damit die Applikation also an Schulen gut angewendet werden kann und möglichst viele Leute erreicht werden können, wäre ein für IOS entwickelter Client von grossem Vorteil. Dieser Client müsste dann jedoch komplett neu in der Sprache Swift programmiert 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. *Volley overview.* https://developer.android.com/training/volley/, 2018. Abgerufen am: 07.05.18.

[3] U. 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.

[4] M. Begerow. *Entity-Relationship-Modell (ER-Modell / ERM).* http://www.datenbanken-verstehen.de/datenmodellierung/entity-relationship-modell//. Abgerufen am: 03.05.18.

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

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

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

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

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

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

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

[14] 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.

[15] Universität Göttingen. *Datenmodellierung mit dem Entity Relationship-Modell.* http://www.winfoline.uni-goettingen.de/gast/eas/html/text/5/2/text_3.htm. Abgerufen am: 03.05.18.

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

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