DUMMY: Entwicklung einer Applikation fr Mobiltelefone zur Vermittlung von Nachhilfe

Florian Hirtz

11. Mai 2018

Inhaltsverzeichnis

Kapitel 1

Vorwort

DummyText...

1.1 Motivation

DummyText...

1.2 Danksagungen

DummyText...

Kapitel 2

Einleitung

2.1 Zielsetzung

Das Ziel der 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 wären, anderen Nachhilfe zu geben. Weiter soll eine Funktion vorhanden sein, mit welche nach anderen Benutzern gesucht werden kann die gewünschten Kriterien Entsprechen. 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. [5]

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

Das *Modell-View-Presenter* (MVP) und auch das sehr ähnliche *Modell-View-Controller* (MVC) sind beides Konzepte, die für das entwickeln von Software entworfen wurden. Ihre Idee ist es, die Aufgaben innerhalb einer Applikation strikt voneinander zu trennen. Dabei wird zwischen drei Typen unterschieden:

- Der *View* ist die sichtbare Benutzeroberfläche. Er hat die Aufgabe, dem Benutzer ein bedienbares Interface zu bieten und soll auf Anfrage den Status seiner einzelnen Komponenten Weitergeben können. Es

kennt das Model nicht.

- Das *Model* ist der Datenspeicher einer Applikation, der gebraucht wird um das View korrekt darzustellen. Es soll auf anfrage hin Daten ausgeben können und hat auch die Aufgabe, falls sich Datensätze ändern, den Presenter davon zu unterrichten. Das Model kennt weder den Presenter noch das View.

- 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, das angezeigte View zu ändern und dessen Status abzufragen. Der Presenter ist in der Lage, sowohl den 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.[2] Ein grosser Vorteil einer nach den 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

Projektentwicklung

Im nächsten Teil wird mehr auf die Entwickelte Applikation eingegangen. Die einzelnen in der Applikation verwendeten Elemente sollen etwas näher gebracht werden und ihre Rolle in der entwickelten Applikation ersichtlich werden.

4.1 Systemüberblick

Die Applikation wurde nach dem Client-Server Prinzip entworfen und entspricht weitgehend dem MVP Konzept (siehe Kapitel 3). Der Presenter und der View finden sich beide auf Seite des Clients (siehe Kapitel 4.4) während das Model sich in Form der Server wiederfindet (siehe Kapitel 4.2 und 4.3).

4.2 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 *Datanbanken* 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. [9]

Datenbanken selbst wiederum 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. 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. [9, S. 745 - 751]

4.2.1 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. [12] 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* formuliert. Queries können in vier Arten von Abfragen unterteilt werden: [9, 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.[9, S. 746]
- Einfügeabfragen (*Insert Queries*) fügen einen neuen Datensatz zu einer bestehenden Tabelle hinzu.[9, S. 746]
- Änderungsabfragen (*Update Queries*) ändern bestimmte oder alle Felder eines bestehenden Datensatzes in einer Tabelle.[9, 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 aude!
☐	2	Geronimo	Jenni	Shemon		2001	[redacted]		A user description.

Abbildung 4.1: Die user_archive Tabelle.

- Löschabfragen (*Delete Queries*) löschen einen Datensatz aus einer Tabelle. [9, S. 746]

4.2.2 MySQL Datenbankstruktur

In der entwickelten Applikation wird genau ein solches MySQL Datanbanksystem in Kombination mit PHP (siehe Kapitel 4.2.3) 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 selber umfasst zwei miteinander verknüpfte Tabellen. Die eine läuft unter dem Name *user_archive* (siehe Abbildung 4.1) und beinhaltet die essentiellen Accountdetails wie Name, Email, Passwort etc. Zu erwähnen ist hier das erste Feld *user_id*. Es wird bei einem neuen Eintrag in die Tabelle automatisch generiert (*auto_increment*) und gewährt so, dass sämtliche Einträge eindeutig unterschieden werden können. Ähnlich verhält sich das Feld user_username. Es ist als *unique* markiert und soll ebenfalls verhindern, dass es beim Login zu Mehrdeutigkeiten kommt. Sie sind die Keys der Tabelle. Die zweite Tabelle der trägt den Namen *user_subjects* (siehe Abbildung 4.2). Sie umfasst nebst einem Feld für die user_id sämtliche momentan von der Applikation unterstützten Fächer. In den Feldern wird nun mithilfe von 1 und 0 angegeben, welche Fächer von einem Benutzer/ einer Benutzerin angewählt wurden, also in welchen er/sie andere unterstützen könnte. Die Inhalte wurden bewusst von einander getrennt, um die Übersichtlichkeit in der user_archive Tabelle zu wahren. Das user_id Feld ist auch hier ein Key und ist für jeden Benutzer gleich wie in der user_archive Tabelle. Diese Verknüpfung der Tabellen kann sehr übersichtlich durch ein *Entity-Relationship-Modell* (ERM) dargestellt werden, da das Modell lediglich zwei Tabellen (Im falle des ERM

	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 4.2: Die user_subjects Tabelle.

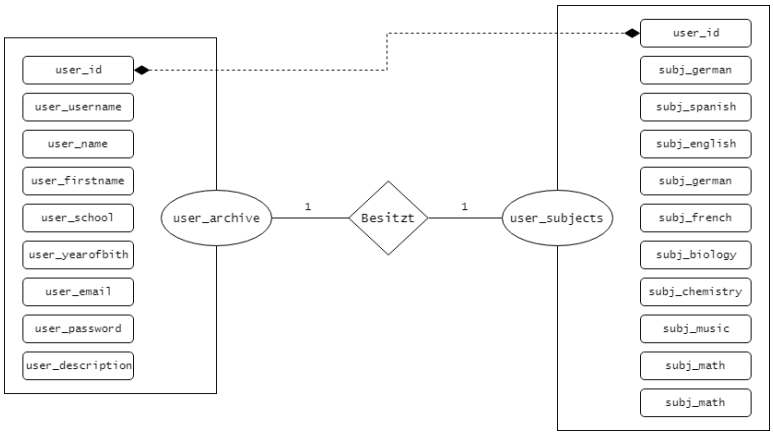

Abbildung 4.3: Das ERM der entwickelten Applikation.

Entities) umfasst. ERMs eignen sich besonders für die Darstellung von Datenbankstrukturen und deren Verknüpfungen. Sie sind oft der erste Schritt, wenn es um die Planung einer neuen Datenbank geht. Die Beziehungen werden durch hauptsächlich zwei Variablen beschrieben, welche im Schema (x, y) dargestellt werden. Die Variable x gibt das Minimum an herrschenden Beziehungen mit anderen Entities vor. Die Variable y das Maximum. Die Variablen können entweder eine bestimmte Zahl wie 1 oder 0 sein oder sie können durch m/n verkörpert werden, stellvertretend für beliebig viele Beziehungen. Im falle des ERMs der entwickelten Applikation (siehe Abbildung 4.3) ist das eine (1,1) : (1,1) Beziehung. Das bedeutet, dass es in beiden Tabellen jeweils genau einen Eintrag für einen Benutzer gibt. [9, S. 750][3][7]

In der entwickelten Applikation wird genau ein solches MySQL Datanbanksystem in Kombination mit PHP (siehe Kapitel 4.2.3) 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 4.1). Die andere Tabelle beinhaltet Felder für alle von der entwickelten Applikation unterstützten Fächer (siehe Abbildung 4.2). 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 4.3 visualisiert. Die Verhältnisse 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

4.2.3 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.[10] 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.[13] Die Website *json.org* ([8]) beschreibt JSON wie folgt:

> "JSON (JavaScript Object Notation) ist ein schlankes Datenaustauschformat, 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."[8]

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.

4.2.3.1 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 4.4.3.1). 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).

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

Wenn die Suche erfolgreich ist, wird auf das Statement mit dem Inhalt eines Datensatzes geantwortet, welcher dann vom PHP Skript wiederum lokalen Variablen zugewiesen wird. Nun ist auch die user_id des Benutzers bekannt und es kann über ein zweites Statement den entsprechenden Datensatz aus der user_subjects Tabelle ausgelesen werden. Wieder werden

zurückgegebenen Werte in lokalen Variablen gespeichert. Die Werte aller gespeicherten Variablen werden nun in ein Array mit dem Namen *response* übertragen und ihnen wird jeweils einen *Key* (Zeichen, unter welchem der Wert im Array gespeichert ist) zugewiesen. Ebenfalls wird eine zusätzliche Variable unter dem Key `success` mit dem Wert `true` in den Array gespeichert. Die Variable `success` Teilt dem Client mit, dass die Anfrage erfolgreich war. Ist jedoch die Suche nach einem registrierten Benutzer mit eingegebenen Daten in der Datenbank erfolglos, verfasst das PHP Skript ein JSON Datei mit der Variable `response["success"]=false` und fügt keine weiteren Daten mehr an. Schlussendlich wird der Antwortarray mit den gespeicherten Daten in einem JSON Format gespeichert und das erstellte JSON File wird an den Client zurückgeschickt. Damit es beim Login zu keinen Mehrdeutigkeiten kommt ist es von Wichtigkeit, dass jeder Benutzername nur einmal in der Tabelle vorkommt (siehe Kapitel 4.2.2).

4.2.3.2 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 4.4.3.2) 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 4.2.3.1). 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. Die Fragezeichen stehen stellvertretend für die einzufügenden Werten, die erst später zugewiesen werden. Mehr dazu in Kapitel (**??**).

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

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 zuerst 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.

4.2.3.3 Änderungen via Einstellungen vom Client

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 4.4.3.4) 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.

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

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 Variablen auf dem Client mit Sicherheit mit denen in der Datenbank übereinstimmen.

4.2.3.4 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 4.4.3.5). 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 erfolgreich 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.

```
'SELECT user_archive.*, user_subjects.*
FROM user_archive INNER JOIN user_subjects ON
user_archive.user_id = user_subjects=user_id WHE-
RE user_archive.user_username LIKE ? OR [...]'
```

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 zuerst einzeln in eigenen Arrays gespeichert. Nun 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. Nun wird noch der Wert `true` mit dem Key `"success"` im Antwortarray gespeichert und das ganze als JSON Datei dem Client zurückgeschickt.

4.3 Firebase

Firebase ist eine Entwicklungsplattform für Webapplikationen, welche seit 2014 von Google angeboten wird. Firebase entwickelte sich aus dem 2011 ge-

gründete 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.[11]

4.3.1 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.[14] 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.[6]

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 4.4.3.8).

4.3.2 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 4.3.2.1). 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.

4.3.2.1 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.

4.4 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.

4.4.1 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.[4]

4.4.2 Architektur des Clients

Der Client der Applikation besteht aus insgesamt zehn sogenannten *Activities*. Activities sind sozusagen die verschiedenen Fenster die in einer Applikation geöffnet werden können. Eine Übersicht über alle Activites findet sich in Abbildung **??**. Activities können grundsätzlich in zwei Teile aufgeteilt: 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).

4.4.2.1 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. 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* 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.[1]

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.

Die userInfo und JSONtoInfo Klasse Damit die einzelnen Activities nicht immer wieder von neuem Anfragen an den Server senden müssen, werden die 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 einer Linie jeweils 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. Dazu gehört Namen, Fächer, Passwort etc. 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 userInfo Klasse instantiiert, die nachher für den Zugriff auf die Accountinformationen gebraucht werden kann.

4.4.3 Module

4.4.3.1 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 4.4.3.2). Wenn auf den Login Knopf gedrückt wird, wird ein login_request mit eingegebenem Benutzernamen und Passwort generiert und das login.php Skript auf dem Server aufgerufen (siehe Kapitel 4.2.3.1). 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 4.4.3.3).

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 den Login Knopf 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.

4.4.3.2 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 einen Knopf 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 den Registrier-Knopf, 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.

4.4.3.3 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 eines Knopfes im rechten oberen Ecken in sie Einstellungen gelangen (siehe Kapitel 4.4.3.4). Weiter kann über den Knopf im unteren rechten Bereich des Bildschirms die Suchfunktion der Applikation ausgeführt werden. Der Knopf 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 noch die beiden Listener Klassen für sowohl den Suchknopf wie auch für den Einstellungen Knopf. Dazu kommt noch ein weiterer Listener, der beschreibt, wie das Bottomsheet ausgefahren werden soll. Zuletzt kommt noch eine onStart und eine View Holder Klasse Methode. Die View

Holder Klasse beschreibt lediglich, wie die einzelnen Viewkomponenten des Layouts des offenen Chats ausgefüllt 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 die View Holder Klasse mit. Somit kann das Bottomsheet mit allen offenen Chats ausgefüllt werden. Die einzelnen offenen Chats bekommen ihre Listener Klassen vom Firebase Adapter und nich von der Mainpage Activty.

4.4.3.4 Settings Activity

Die *Settings Activity* teilt sich eigentlich in ganze 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 Eintsellunsactivites. 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 savaestetting_pw.php eine Anfrage geschickt. Bei Erfolg ändert der Screen automatisch zurück zu Übersicht.

4.4.3.5 Filter Activity

Die Filter Activity wird durch von der Hauptseite aus erreicht und ist dazu da, damit der Benutzer/die Benutzerin ihre Suchkriterien eingeben kann. Dazu gehört Name und Fach. Der Name kann optional in einem Textfeld eingegeben werden während die Fächer über Checkboxen ausgewählt werden können. Darunter findet sich ein Knopf, der die effektive Suche startet. Es wurde bewusst auf eine Dynamische Suchfunktion verzichtet, da sonst das einbauen des Filtern nach Fächern etwas unübersichtlich geworden wäre. Mithilfe diese Filter Activity können sich Benutzer/Benutzerinnen zuerst in ruhe ihre Suchkriterien auswählen und sich anschliessend in die Ergebnisse evaluieren. Beim drücken des Suchknopfes unter den Kriterien wird ein serach_request erstellt, der eine Anfrage an das search.php File schickt. Wenn die Suche Ergebnisse liefert, wird die Searchresults Activity gestartet.

Die Presenter-Klasse 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 ResponseListe-

ner.

4.4.3.6 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 durch scrollen gleich alle wichtigsten Informationen herausgelesen werden können. Dazu gehören Namen, Geburtsjahr, Fächer und Profilbild. In die List einegefüllt 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 4.2.3.4) und somit können die im Array gespeicherten Informationen über den Index an die Userprofile Activity weitergegeben werden.

4.4.3.7 Userprofile Activty

Die *Userprofile Activity* Stellt das Profil eines anderen Benutzers dar. Vom Layout her ist sie fast identisch zur Mainpage Activity, nur das sowohl der Settings- wie auch der Suchknopf fehlt. Dafür findet sich gross und auffällig ein Knopf für das öffnen eines neuen Chats. Es soll für den Benutzer/die Benutzerin sofort klar sein, wie er/sie die Möglichkeit hat, mit dem anderen Benutzer/der anderen Benutzerin Kontakt aufzunehmen. Wird dieser Knopf gedrückt, öffnet sich ein neuer Chat (siehe Kapitel 4.4.3.8).

Der Presenter der Userprofile Activity besteht aus insgesamt drei Komponenten: der onCreate Methode, der Methode für den Pfeil zurück und noch dem Listener für den Knopf, der 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 Nach-

richten empfangen kann.

4.4.3.8 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 wohl 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 Name 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 4.3.1).

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.

Literaturverzeichnis

[1] Android. *Volley overview.* https://developer.android.com/training/volley/, 2018. Abgerufen am: 07.05.18.

[2] 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.

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

[4] 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.

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

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

[7] U. 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.

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

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

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

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

[12] 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.

[13] A.-A.-G. Völkermarkt. *Wie funktioniert PHP?* http://www.gym1.at/schulinformatik/aus-fortbildung/fachdidaktik/vo-01/php/wie_funktioniert_php.htm. Abgerufen am: 03.05.18.

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