DUMMY: Entwicklung einer Applikation fr Mobiltelefone zur Vermittlung von Nachhilfe

Florian Hirtz

7. Mai 2018

Inhaltsverzeichnis

Kapitel 1

Vorwort

DummyText...

1.1 Motivation

DummyText...

1.2 Danksagungen

DummyText...

Kapitel 2

Einleitung

2.1 Zielsetzung

DummyText...

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 agierenden Programme aufgeteilt. Diese Programme werden allgemein 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 Website 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 durchaus mehrere Clients. Ein guter Server sollte also auch darauf vorbereitet sein, mehrere Anfragen von verschiedenen Clients parallel zu bearbeiten. [4]

Kapitel 4

Projektentwicklung

DummyText...

4.1 Systemüberblick

DummyText...

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. [8]

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. Die *relationale Datenbank*. Sie 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, ansonsten kann es Probleme bei der Verknüpfung kommen.[8, S. 745 - 751]

4.2.1 MySQL Datenbank

Ein Beispiel für ein solches *Relational Database Management System* (RDBMS) ist die weit verbreitete MySQL Datenbank. Das System wurde ursprünglich von den drei Gründern Allan Larsson, Michael Widenius und David Axmark 1995 entwickelt und wurde später von *Sun Microsytems* aufgekauft und gelangte schlussendlich in den Besitz von dem amerikanischen Softwarehersteller *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. [11] 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 Datenspeichersystemen für Webservice alles Art. Im Falle eines solchen Webservices 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: [8, 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.[8, S. 746]
- Einfügeabfragen (*Insert Queries*) fügen einen neuen Datensatz zu einer bestehenden Tabelle hinzu.[8, S. 746]
- Änderungsabfragen (*Update Queries*) ändern bestimmte oder alle Felder eines bereit bestehenden Datensatzes in einer Tabelle.[8, S. 746]
- Löschabfragen (*Delete Queries*) löschen einen Datensatz aus einer Tabelle. [8, 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 Daten-

	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 4.1: Die user_archive Tabelle.

	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.

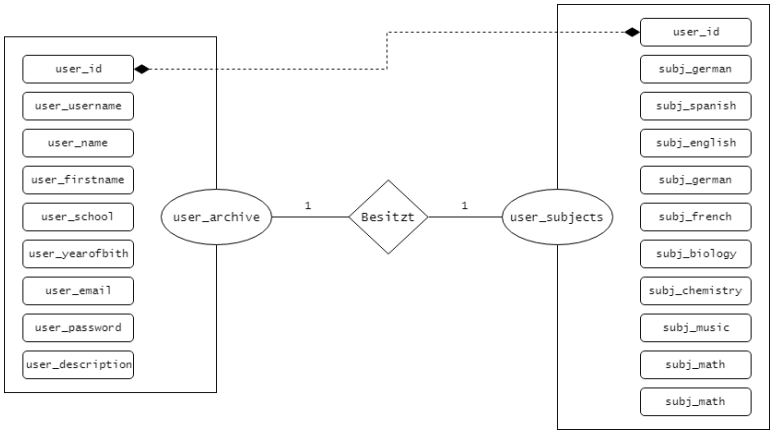

Abbildung 4.3: Das ERM der entwickelten Applikation.

bank 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 *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. [8, S. 750][2][6]

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 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 [9]. Dies ist möglich, da dar 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 oder sogar Bildformaten sein.[12]

Die PHP-Skripte der Entwickelten Applikation verfassen alle ihre Antworten im Datenformat JSON (*JavaScript Object Notation*).

> "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." [7]

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 *login.php* Skript via Webadresse auf. Das aufgerufene Loginskript erstellt zuallererst eine Verbindung zur lokalen MySQL Datenbank. Daraufhin nimmt sie den Mitgeschickten Benutzernamen und das Passwort entgegen und speichert sie in lokalen Variablen. Als nächstes wird versucht ein 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, sich 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 *register.php* File auf dem Server geschickt. Das register.php Skript hat die Aufgabe, dem Benutzer einen Datensatz in beiden der Tabellen zu erstellen, sofern die Daten legitim sind. Zuerst wird dafür eine Verbindung mit der lokalen Datenbank aufgenommen. War dies Erfolgreich, wird daraufhin ein Auswahlabfrage formuliert, welche nach einem Datensatz mit dem eingegebenen Benutzernahmen fordert. 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(siehe Kapitel 4.2.3.1). Wenn kein solcher Datensatz gefunden wurde, wird als nächstes ein 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.

```
'INSERT INTO user_archive([...]) VALUES ([...])'
```

Als nächstes soll noch einen Datensatz in der user_subjects Tabelle eingetragen 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 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 per Default keine Fächer ausgewählt haben sollen. Schlussendlich, wird eine JSON Response 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 *save-settings_pw.php* File geschickt. Diese Skript soll die bereits bestehenden Datensätze auf der Datenbank verändern und die veränderten Daten zurückschicken. 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 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. Nun wird das gleiche mit der user_subjects Tabelle gemacht. Zuerst eine Änderungsabfrage und daraufhin eine Auswahlabfrage. Sämtliche gespeicherten Daten werden in einen Array gespeichert und diese anschliessend im JSON Format an den Client zurückgeschickt.

4.2.3.4 Suchanfrage

Damit es Benutzern/Benutzerinnen möglich ist, nach anderen Benutzern und Benutzerinnen zu suche, wird ein PHP Skript benötigt, welches die Datenbank nach Datensätzen durchsucht, welche bestimmten Kriterien unterliegen. 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 search.php File übernimmt diese Aufgabe. Hierzu wird zuerst eine Verbindung mit der Datenbank aufgebaut. Ist das erfolgreich wird begonnen ein Statement zu formulieren, welches später als eine Select Query gestellt werden soll und 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 [...]'
```

Zum anderen 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 ganz genauen Namen kennen.

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, werden nur die Datensätze ausgewählt, bei welchen zum einen der Name „Magalie“ vorkommt, aber auch Mathematik als ein Fach angegeben haben.

Sind diese beiden Schritte erledigt, wird der String als eine Query ausgeführt und es werden die Datensätze zurückgegeben, die den Suchkriterien unterliegen. Nun werden die Informationen der Suchergebnisse mit Ausnahme des Passwortes zuerst einzeln in eigene Arrays gespeichert. Diese Arrays stellen sozusagen die einzelnen Suchergebnisse dar. 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 Entwickelt wird. Firebse entwickelte sich aus dem 2011 gegründete Startup *Envolve* von den 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 der Benutzer die API für noch viel mehr als nur Chats verwendeten, ja sogar einzelne Spielentwickler sie für eine Realtime Synchronisation verschiedener Clients verwendeten, begann die Entwicklung sich viel mehr 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 Webserviceentwicklung anbietet und heute unter Namen Firebase bekannt ist. Firebase bietet sowohl Tools für die Entwicklung neuer Service wie Realtime Datenbanken, Authentifizierungsfunktionen und Crashanalysen, aber auch für das Unterhalten von bestehenden Servicen. Die Tools dürfen in einem begrenzten Rahmen gratis verwendet werden und sind daher sehr attraktiv für kleinere Entwicklerunternehmen.[10]

4.3.1 Firebase Realtime Datenbank

Wie bereits erwähnt, bietet Firebase unter anderem auch Realtime Datenbankfunktionen. 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, 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 von nöten ist, dass die Clients ohne grosse Verzögerung davon unterrichtet werden.[13] 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.[5]

Die Chatfunktion in der entwickelten Applikation ist ein genau solcher Service, der einen solche Datenbank benötigt. So können Clients in Echtzeit via untereinander Kommunizieren ohne das sie entweder permanent online

sein müssen, oder konstant die Applikation neu laden müssen. 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 per default die Firebase API, weshalb das Einbinden einer Firebase RTDB in eine Applikation sehr simpel 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 Tabelle gespeichert sondern kann vielmehr mit der Struktur eines Ordnersystems verglichen werden. Die Informationen werden unter einer Form von Ordnern gespeichert und in einem System geordnet. Der Ort, wo die Informationen gespeichert sind, wird in Pfaden angegeben, die mit denen von einem Filesystem eines Computers zu vergleichen sind. Dabei gibt es einen Hauptordner, der Sozusagen die gesamte Datenbank umfasst. Diese wiederum hat Unterordner, welcher allgemein als *Children* bezeichnet werden.

Die Struktur der verwendeten Datenbank kann grundsätzlich in zwei Ordner aufgeteilt werden. Der *User*-Ordner und der *Chats*-Ordner. Im User Ordner sind alle Benutzer verzeichnet, die einen oder mehrere offene Chats besitzen. Der Ordner ist hauptsächlich dazu da, um die offenen Chats eine Benutzers auf der Hauptseite des Clients darstellen und öffnen zu können. Die Children des Verzeichnis sind unter den IDs der Benutzer gespeichert. Für jeden offenen Chat findet sich in einem Solcher Child nun ein weiteres Child für jeden offenen Chat des Benutzers/der Benutzerin. Dieses enthält nun 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 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 der Absender/die Absenderin.

4.3.2.1 Warum genau diese Struktur?

Es ist an dieser Stelle vielleicht noch interessant zu erwähnen, weshalb die Firebasestruktur auf den ersten Blick etwas überkompliziert erscheinen mag. Der Grund dafür ist nämlich eine 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 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 das eine Suchfunktion dafür benötigt wird.

4.4 Client

Der Client ist das Herzstück des Service. Er ist der, der schlussendlich bei 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 ein von Google entwickelter IDE für die Entwicklung von Java Applikationen für Android Mobilgeräte. Der IDE 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 verfasst. Java ist eine Sprache der 3. Generation und gehört zu den objektorientierten Programmiersprachen. Sie wurde erstmals 1995 von Sun Microsystems veröffentlicht und ist nun 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 gleich funktionieren sollen. Das Betriebssystem Android, für welche die Applikation entwickelt worden ist, ist das Weltweit meist verwendete OS 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.[3]

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 auf der nächsten Seite. Activities sind grundsätzlich in zwei Komponenten aufgeteilt. Zum einen die *View*-Komponente welche in der Layoutsprache XML geschrieben ist und das Userinterface der Activtiy bestimmt. Die zweite Komponente ist die dazugehörende *Controller*-Klasse, welche in Java ist programmiert ist.

4.4.2.1 Klassenübersicht

Nebst den Activities und ihren jeweiligen Controller 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 Children von der Klasse *Stringrequest* und erstellen aus den zu versendenden Daten eine lokale Hashmap. Die Request Klasse beinhaltet ebenfalls die URL des PHP-Files, an welches die Anfrage gesendet werden soll. Die Request-Klassen können da sie Children der Stringrequest Klasse sind in sogenannten *Request Queues* dann 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 die 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 Child der Klasse *Base Adapter* aus der *Widget* Library verwendet. Die anderen beiden Adapter sind Children 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, geben sie sich jeweils die Informationen untereinander weiter. So muss nur einmal zu beginn beim Login eine Anfrage statt finden. Die Informationen werden unter den Activitis 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 für den Gebrauch 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 also in jeder Activity zuerst aus dem JSON-String eine userInfo Klasse instantiiert, die nachher als 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, der ein Benutzer/eine Benutzerin zu Gesicht bekommt, wenn er/sie die Applikation startet. Er fordert den Benutzer/die Benutzerin auf, sein/ihr Benutzernamen und Passwort einzugeben. Das Design ist sehr allgemein gehalten und ist Vergleichbar mit dem vieler anderer Login Screens von diversen Applikationen. Über der Eingabe ist das Logo der Applikation zu sehen. Es soll das sonst sehr allgemeine Login etwas 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. Sollte ein Benutzer/eine Benutzerin auf den Link klicken, 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 File auf dem Server aufgerufen (siehe Kapitel 4.2.3.1). Ist die Anfrage erfolgreich, wird der JSON String in in die *Extras* (Datencontainer, mit welchem Werte zwischen Activities weitergegeben werden können) gespeichert und die Hauptseite gestartet(siehe Kapitel 4.4.3.3).

Der Controller des Logins besteht im wesentlichen aus 4 Komponenten. Der standard Methode *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 sowohl den Login Knopf wie auch 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 zuletzt eine Checkbox für die AGBs 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 File 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 Controllerklasse finden sich die 5 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

DummyText...

4.4.3.4 Settings Activity

DummyText...

4.4.3.5 Filter Activity

DummyText...

4.4.3.6 Searchresults Activty

DummyText...

4.4.3.7 Userprofile Activty

DummyText...

4.4.3.8 Chat Activity

Literaturverzeichnis

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

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

[3] J. Callaham. *The history of Android OS: its name, origin and more.* https://www.androidauthority.com/history-android-os-name-789433/, 2018. Abgerufen am: 06.05.18.

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

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

[6] 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.

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

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

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

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

[11] 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.

[12] 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.

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