

MOBILE ANWENDUNGEN MIT
ANDROID – **ZWINDER**,
EINE DATING APP

HOCHSCHULE KAISERSLAUTERN,
KARIM STOCK, 868184

INHALT

Einleitung	3
Zieldefinition	3
Services und Plugins	4
Übersicht an Genutzten Plugins	4
Firebase Authentication	4
Firebase Database	5
Firebase Storage	5
Butterknife – View Binding Library	6
SwipeCards	6
Glide – Image Loading Library	7
Design – Sourcey’s Materiallogindemo	7
Pageflow	8
Login-Vorgang und Registrierung	8
Registrierung zu Datenbank	9
Validierung von Daten	9
MainActivity und zugehörige settings	10
Swipen in MainActivity	10
UserSettingsActivity	11
Probleme	12
Hauptproblem: Query	12
Die Query	12
Lösungen	13
1. User, welche existieren	13
2. Den Userinteressen entspricht	13
3. Der potenzielle User auch Interesse an diesem User hat	14
4. Der potenzielle User noch nicht geswiped wurde	15
5. Der potenzielle User nicht der eigene Ausgangsuser ist	15
Vollständige Query in MainActivity	15
Fazit	16
Literaturverzeichnis	17

EINLEITUNG

Inspiziert von Tinder [1] und ähnlichen modernen Dating Apps [2], welche nur darauf basieren, potenziellen zukünftigen Partner mit einem *like* oder einem *dislike* zu bewerten, beruht auch diese Dating App, **Zwinder**, dieses simple Bewertungssystem, um festzustellen, ob sich zwei User gegenseitig mögen.

Der Vorgang wird generell als **Swipen** bezeichnet und auch im Verlauf dieser Dokumentation so genannt, da es in Tinder und ähnlichen Apps üblich ist, Userprofile als Kärtchen darzustellen, welche auf die rechte oder linke Hälfte des Bildschirmes *geswiped* werden können. Typischerweise bedeutet nach rechts *swipen*, dass man den anderen User mag, wobei nach links *swipen* das Desinteresse widerspiegelt.

ZIELDEFINITION

Ziel dieser Applikation war es, eine nutzbare Dating App zu realisieren, welche von Erstellung eines Accounts, über Hochladen von Bildern, sowie Einstellungen zu Geschlecht und Präferenzen verfügt, um entsprechende Partner vorgeschlagen zu bekommen.

User sollten sich gegenseitig *liken* oder *disliken* können, indem sie die jeweiligen animierten Kärtchen eines potenziellen Partners vorgeschlagen bekommen und diese entsprechend *swipen* können.

Kommt es hierbei zu einem sogenannten **Match**, soll heißen, beide User haben sich gegenseitig *geliked*, werden beide benachrichtigt und es bietet sich die Möglichkeit, mit dem anderen User Nachrichten auszutauschen.

SERVICES UND PLUGINS

ÜBERSICHT AN GENUTZTEN PLUGINS

ANFORDERUNGEN AN PLUGIN	MEGA FLOW PLUGIN
USER AUTHENTIFIZIERUNG	Firebase Authenticationg
ECHTZEIT DATENBANK	Firebase Database
CLOUD STORAGE (PROFILBILDER)	Firebase Storage
VIEW BINDING LIBRARY	Butterknife
SWIPE CARD ANIMATIONS	SwipeCards
IMAGE LOADING LIBRARY (DOWNLOADEN VON PROFILBILDERN)	Glide

FIREBASE AUTHENTICATION

Zum Identifizieren einzelner User und Erfassen derer Daten war eine Form der Authentifikation vorausgesetzt. Firebase Authentication [3] bietet mit seinem SDK unterschiedliche Möglichkeiten, User zu authentifizieren (z.B. über Google, Facebook und Twitter Accounts). Für diese Anwendung wurde lediglich das herkömmliche Registrierungsverfahren eines Accounts über E-Mail Adressen realisiert.



Durch das hinzufügen und initialisieren von **FirebaseAuth** konnten zusätzlich Listener hinzugefügt werden, die in einzelnen Activities den Zustandswechsel der Firebase Authentifikation abfragen und so z.B. bei der **LoginActivity** feststellen können, dass der User noch eingeloggt ist und diesen direkt in die **MainActivity** weiterleiten können.

```
//Falls User authentifiziert -> weiterleiten an onSignupSuccess() -> MainActivity
firebaseAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        final FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
        if (user != null) {
            onSignupSuccess();
        }
    }
};
```

FIREBASE DATABASE

Da grade bei einer umfangreichen Dating App viele Informationen zu einzelnen Usern in deren Profilen gespeichert – und von anderen Usern wieder abgerufen – werden können, war es wichtig, diese schnell teilen und zwischen einzelnen Clients synchronisieren zu können. Firebase Realtime Database [4] bietet eine cloud-hosted Datenbank, welche Daten in JSON speichert und in Echtzeit mit allen verbundenen Clients synchronisiert, ein weiterer Standard in Android Apps.

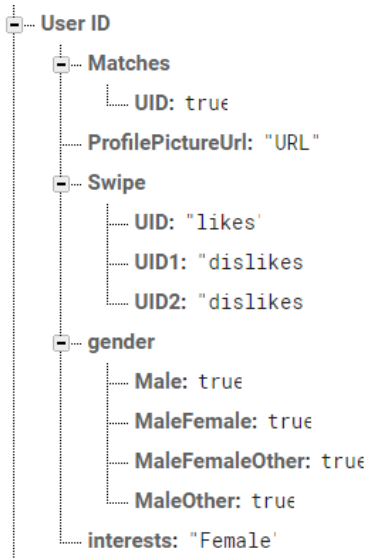


Figure 1 zeigt die Struktur, in welcher mittels Firebase Realtime Database Userprofile zu ihrer jeweiligen ID angelegt werden. Dabei bildet ein nichtabgebildetes Objekt „UIDs“ den übergeordneten Zweig zu allen angelegten Userprofilen. Die UserID ergibt sich aus der, durch Firebase Authentication erstellten ID, **Gender** und **Interests** stellen Geschlecht und Sexualität dar und werden beim Anlegen des Accounts gewählt.

Das Profilbild erst später über Firebase Storage hochgeladen und eine Download URL wird zum zugehörigen User in der Datenbank, als Wert von **ProfilePictureUrl** eingetragen. **Swipe** und **Matches** ergeben sich aus den jeweiligen Entscheidungen des Users in der **MainActivity**.

Figure 1 Datenbankstruktur

FIREBASE STORAGE

Da in Firebase Database lediglich Eintragungen einzelner Strings in einer JSON Struktur stattfinden, müssen Dateien – wie in diesem Fall die Profilbilder – auf einen weiteren Service hochgeladen werden. Hierfür wurde der Einfachheit entsprechend ein weiterer Firebase Service genutzt: Firebase Storage [5]. Firebase Storage ist ein effektives Mittel, Objekte sicher in Googles Cloud Storage zu speichern und durch das SDK in der Android App hoch- und runterzuladen.



gs://zwinder-32ccf.appspot.com > profilePictures

Datei hochladen




<input type="checkbox"/>	Name	Größe	Typ	Zuletzt geändert
<input type="checkbox"/>	 2pq6FPmLsETgTwhrT5QQkNW2rAu2.jpg	20,7 KB	image/jpeg	01.07.2018
<input type="checkbox"/>	 3nvDxnUqZ9aihuDRVB4OkyVlvFy2.jpg	5,26 KB	image/jpeg	01.07.2018
<input type="checkbox"/>	 3yYN4uqNhxMZ37f5OfOQgl9f4TH2.jpg	10,59 KB	image/jpeg	02.07.2018

Figure 2 Firebase Storage Hierarchie

Figure 2 zeigt die Firebase Storage Hierarchie mit abgelegten User Profilbildern im **profilePicture** Ordner.

BUTTERKNIFE – VIEW BINDING LIBRARY

Ein Tool, das sich beim Arbeiten mit Views als besonders hilfreich herausgestellt hat, ist Butterknife [6]. Butterknife ist ein View Binding Library von Jake Wharton, welches eine simple und saubere Möglichkeit bietet, Views, Felder und Methoden zu *binden* [7], um sie später vereinfacht wieder abrufen zu können (statt jedes Mal auf *findViewById(R.id.view)*) zurückzugreifen [8].



```
35 //Bindviews mit Butterknife
36 @BindView(R.id.input_name) EditText _nameText;
37 @BindView(R.id.input_email) EditText _emailText;
38 @BindView(R.id.input_password) EditText _passwordText;
```

So können z.B. die Werte der einzelnen Felder aus Abbildung X. im weiteren Verlauf der jeweiligen Activity vereinfacht abgerufen werden (siehe Abbildung X).

```
//Name, E-Mail, Passwort
final String name = _nameText.getText().toString();
String email = _emailText.getText().toString();
String password = _passwordText.getText().toString();
```

SWIPECARDS



Ein weiteres Library, welches Platz in dieser App gefunden hat, ist Swipecards [9]. Da die Dating App stark von der erfolgreichen Dating App *Tinder* inspiriert ist und eine der Hauptmerkmale von *Tinder* das berühmte *Swipen* und Evaluieren einzelner Profilbilder innerhalb weniger Sekunden ist, wurde auch hier eine Möglichkeit realisiert, Userkärtchen nach Präferenz rechts oder links zu *Swipen*.

Das *Swipen* gibt die jeweilige Präferenz dann an Firebase Database weiter und legt den entsprechenden Eintrag in der Datenbank fest.

Entwickelt wurde Swipecards[9] von *Dionysis Lorentzos* und ist ebenfalls auf Github einsehbar.

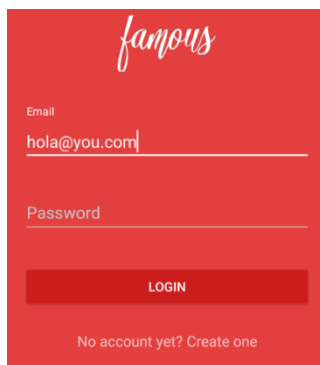
GLIDE – IMAGE LOADING LIBRARY

Die letzte Bibliothek, die in diesem Projekt Anwendung gefunden hat, ist Glide [10]. Glide bietet ein schnelleres - und vor allen Dingen schlaueres - Mittel zum Laden von Bildern während die App ausgeführt wird [11].



Probleme vor dem inkludieren von Glide waren unter anderem ein kurzes einfrieren der App beim laden einzelner Bilder und überschreiben aller Bilder, die einer jeweiligen geteilten Ressource entsprungen sind, z.B. überschreiben aller Profilbilder der geladenen Userkärtchen der **MainActivity**.

DESIGN – SOURCEY'S MATERIAL LOGIN DEMO



Als Vorlage für das Design wurde Sourcey's Material Login Demo verwendet, welche von Entwickler *Sourcey* Open Source auf GitHub [12] gestellt wurde. Ein zusätzlicher Blogeintrag [13] dokumentiert die Erstellung seines Login- und Sign-Up Designs umfangreich.

Übernommen wurden hiervon die beiden Activities, welche mit zusätzlichen Code für weitere Validation ausgestattet wurde und um die eigenen Firebase Authentication und Database Services erweitert wurden.

Die Wahl der Farben wurde für das weitere Design in den XML Dateien der App übernommen.

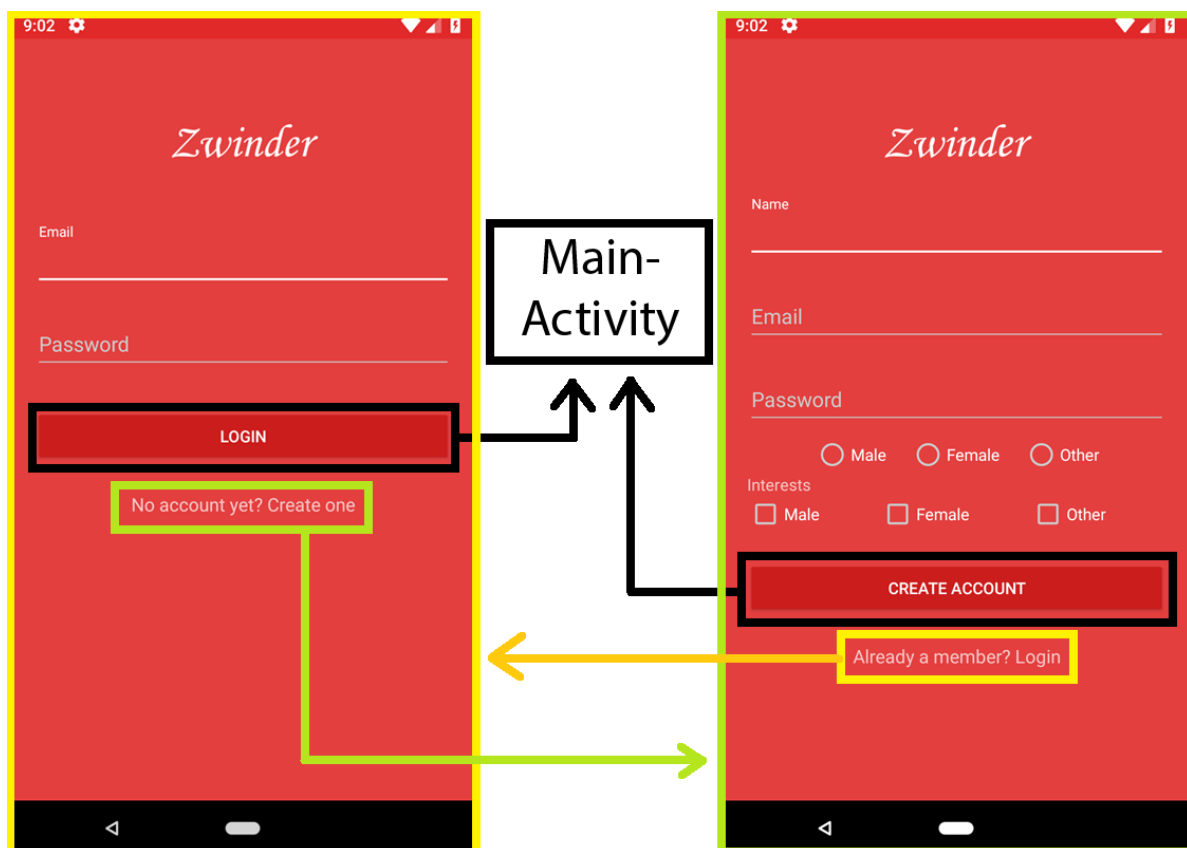
PAGEFLOW

In diesem Kapitel werden die Vorgänge und das Ausmaß einzelner Vorgänge beschrieben und abgebildet, um auch ohne Code darzustellen, welche Folgen einzelne Interaktionen haben und welche Activities sich jeweils von anderen Activities ansteuern lassen.

LOGIN-VORGANG UND REGISTRIERUNG

Die erste Activity, die beim Starten der Applikation aufgerufen wird ist die **LoginActivity**. Diese überprüft zuallererst über Firebase Authentication, ob der User bereits authentifiziert ist und leitet in diesem Fall zur MainActivity weiter.

Ist dies nicht der Fall, kann der User sich hier mit seiner E-Mail Adresse und dem Passwort einloggen, oder auf den Registrierungsvorgang zugreifen, um über die **SignupActivity** einen neuen Account anzulegen.



Beim Anlegen eines Accounts angegebene Daten ergeben dann zugehörige Einträge in der Firebase Database.

REGISTRIERUNG ZU DATENBANK

Figure 3 beschreibt den Vorgang der Registrierung zum abspeichern der Einträge in der Datenbank als JSON. Der obige Knoten beschreibt hierbei die UserID. Einträge für **Name**, **Gender** und **Interests** bilden sich aus den angegebenen Werten der **SignupActivity**.

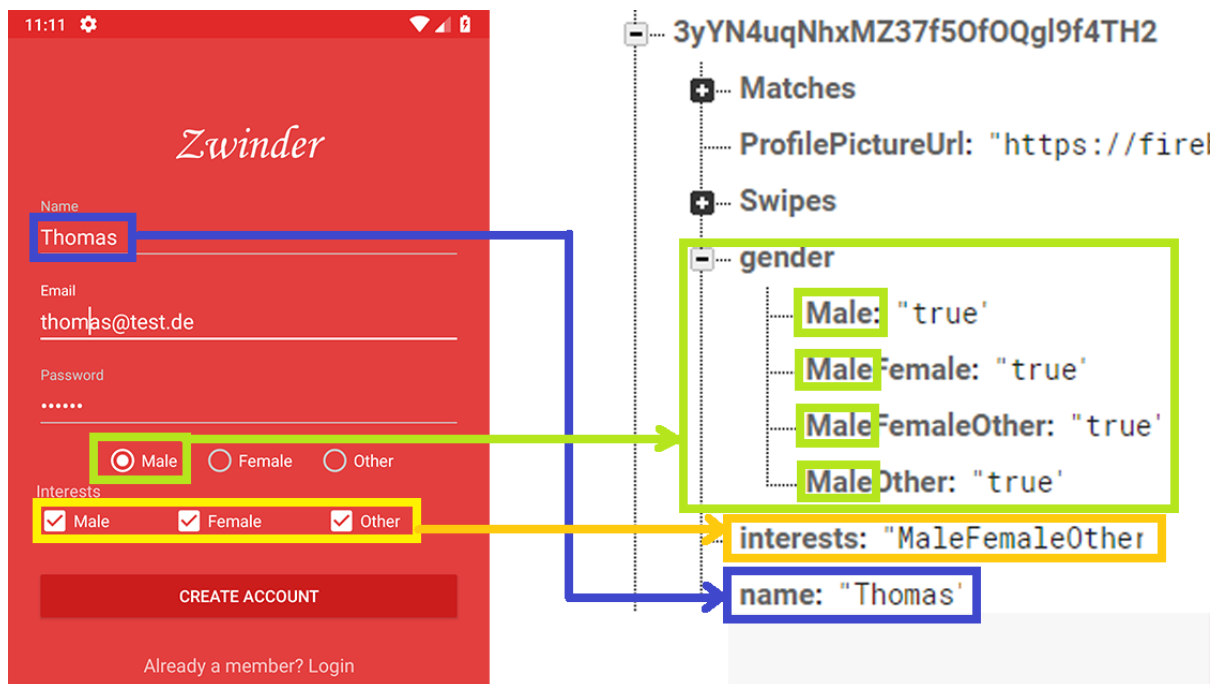


Figure 3 Registriervorgang vs. Datenbank

Nach der Registrierung ist der User authentifiziert und wird in die **MainActivity** geleitet.

VALIDIERUNG VON DATEN

Die meisten Daten werden beim Registrieren bei der **SignupActivity** entgegengenommen. Diese werden vorm Absenden allerdings validiert.

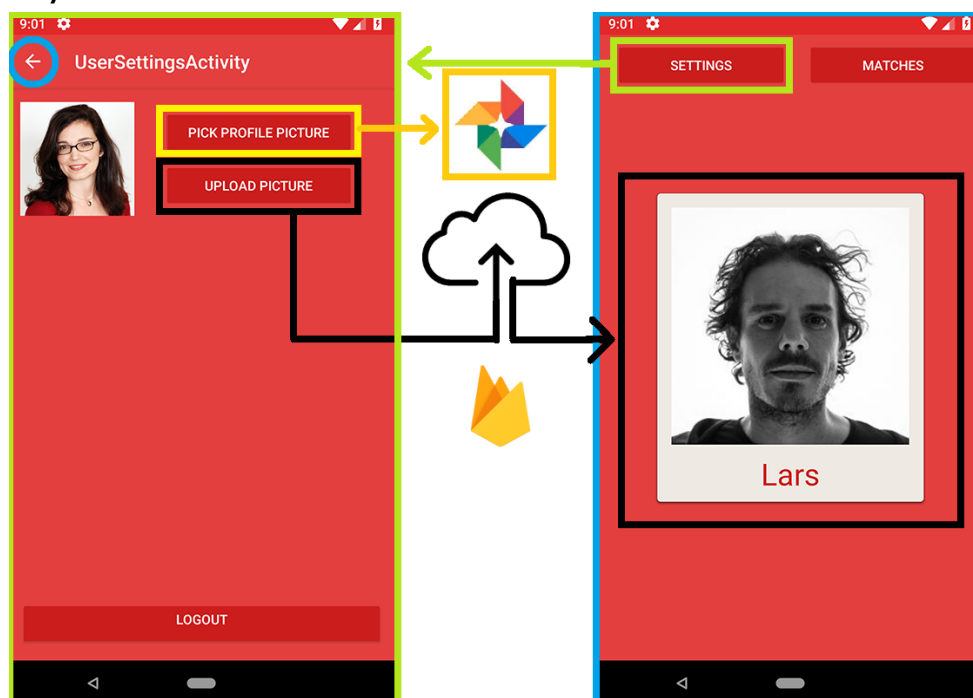
Die Eingaben umfassen: **Name**, **Passwort**, **E-Mail**, Geschlecht und Interessen. Es gelten folgende Einschränkungen für diese Werte, die es zu erfüllen gilt. Da das Geschlecht eine RadioGroup ist, konnte hier einfach ein Default gesetzt werden, statt die Eingaben überprüfen zu müssen. Die E-Mail Eingabe wird über ein Android Pattern Match [14] geprüft.

WERTE ZU VALIDIEREN	FORM DER EINSCHRÄNKUNG
NAME	Minimum 3 Zeichen
PASSWORT	Zwischen 4-10 alphanumerische Zeichen
E-MAIL ADRESSE	Android Pattern Match EMAIL_ADRESS
GESCHLECHT	RadioGroup, default ist Male, kann nicht leer sein
INTERESSEN	Eine von 3 Checkboxes muss gesetzt sein

MAINACTIVITY UND ZUGEHÖRIGE SETTINGS

Die **MainActivity** beinhaltet die Hauptanwendung der „Dating App“, nämlich das *Swipen* von potentiellen Partnern. Links trägt einen „*dislike*“ in die Datenbank ein, nach rechts *swipen* trägt einen „*like*“ in die Datenbank ein.

Zusätzlich können von hier aus die **UserSettingsActivity** und die **MatchesActivity** angesteuert werden, welche das Hochladen und Updaten des Profilbildes ermöglichen, sowie anzeigen der „*Matches*“ mit anderen Usern, die einen ebenfalls nach rechts *geswiped* haben. Über die **Back** Buttons kommt man von der jeweiligen Activity zurück in die **MainActivity**.



SWIPEN IN MAINACTIVITY

Das *Swipen* in der **MainActivity** wird durch die bereits genannte Library *Swipecards* ermöglicht. Ein **ArrayAdapter** für die Klasse der Karten – ähnlich eines **RecyclerViews** – bildet hierbei die Basis der einzelnen Karten, um Informationen jeweiliger User aus der Datenbank zu erfragen.

Swipecards selbst bietet von Haus aus die Animationen und die jeweiligen **EventListeners**, um festzustellen, in welche Richtung *geswiped* wurde, oder ob Karten entfernt werden. Eine zusätzlich verknüpfte XML bietet für die Werte der *Swipecard* Klasse vom **ArrayAdapter** letztendlich das abgebildete Design.



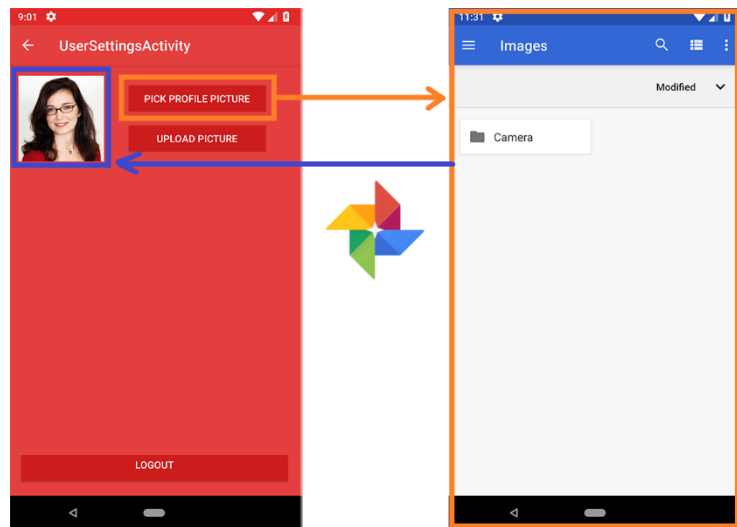
USERSETTINGSACTIVITY

In der **UserSettingsActivity** können Profilbilder aus der eigenen Galerie gewählt werden und über den Button „*Upload Picture*“ letztendlich in den Firebase Storage hochgeladen werden.

```
Intent intent = new Intent();
intent.setType("image/*");
intent.setAction(Intent.ACTION_GET_CONTENT);
```

In diesem Fall wird der Typ des Intents für das klicken des „*Pick Profile Picture*“ Buttons gewählt, sodass nur Dateien des Typ *image* ausgewählt werden können. Das Klicken des „*Upload Picture*“-Buttons lädt schließlich das Profilbild in den angelegten Profilbildordner im Cloud Storage, unter dem Namen „*UserID.jpg*“.

Zusätzlich wird in der **uploadImage()** Methode die URL des hochgeladenen Bildes abgefragt und als *Child* Objekt in den Firebase Database Eintrag des jeweiligen Users abgespeichert, sodass andere User nurnoch die URL abfragen müssen, statt durch Firebase Storage jeweils eine Neue URL zu erzeugen oder die Firebase Storage File Requests auszuführen.



```
firebaseAuth.signOut();
Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
startActivity(intent);
finish();
```

Der Logout Button führt die FirebaseAuth Methode **signOut()** aus und leitet zurück zur **LoginActivity**, sodass der User sich unter anderem Account einloggen kann.

Der „**Back**“-Button in der Actiontoolbar schickt den User zurück in die **MainActivity**.

PROBLEME

HAUPTPROBLEM: QUERY

Das Hauptproblem beim Realisieren dieser App waren die Queries. Beim Suchen nach potenziellen Usern, die vom Geschlecht und den Interessen her den Präferenzen des eigenen Users entsprechen, stößt man an die Grenzen von Eigenschaften, nach denen man mit Firebase Database filtern kann.

Hauptproblem kurzgesagt: NoSQL besitzt keine so hohen Standards beim komplexen Filtern von Daten, wie SQL [15]. Es musste also zum vorbereiten der Queries, dass sie den Anforderungen entsprechen viel rumprobiert, umgedacht und Datenstrukturen angepasst werden.

Wo mit SQL in kürze festgestellt werden kann, wo zutreffende Datensätze liegen, musste hier mühsam nach einzelnen Eigenschaften gefiltert werden, wieder und wieder, was zur Folge hatte, dass die meisten Tage und Stunden der Realisierung sich der Vorarbeit und Ausarbeit der Queries widmeten.

DIE QUERY

Da sich das Resultat auch nur begrenzt in Teilabschnitten in SQL formulieren lässt, wird die Query hier nun in Prosa beschrieben, sodass der Leser sich nicht mit den Namen der Datensätze auseinandersetzen muss. Auf den ersten Blick scheinen diese ebenfalls selbstverständlich.

In Prosa:

Dieser User möchte alle User, welche:

1. existieren
2. seinen Interessen entsprechen
3. ebenfalls Interesse an seinem Geschlecht haben
4. noch nicht von ihm *geswiped* wurden
5. nicht er sind

Für jede dieser Voraussetzungen wurde eine Schicht der Query hinzugefügt und letztendlich wird auch nach all diesen Voraussetzungen gefiltert, teilweise über Umwege, teilweise über Umstrukturierung.

LÖSUNGEN

Hier werden nun die Lösungen der einzelnen Einschränkungen der Query präsentiert.

1. USER, WELCHE EXISTIEREN

Letztendlich wird von Firebase Database beim Entgegennehmen eines Snapshots zusätzlich gefragt, **if dataSnapshot.exists()**, um festzustellen, ob dieser Eintrag existiert.

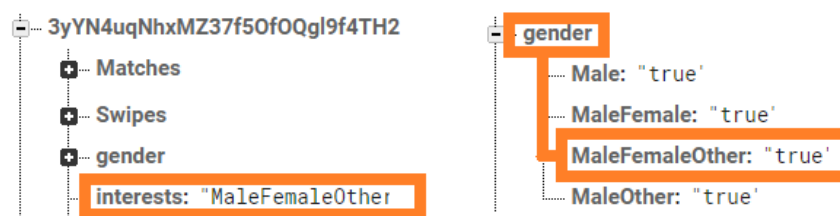
2. DEN USERINTERESSEN ENTSPRICHT

Da es nicht möglich ist, auf direktem Wege nach mehreren Eigenschaften gleichzeitig zu suchen, und ein User Interesse an 3 unterschiedlichen Geschlechtern, **Male**, **Female** und **Others** haben kann, wurden die unterschiedlichen möglichen Interessen als binäre Aufzählung zusammengefasst.

Der User kann als Interesse also **000b** bis **111b** verschiedene Interessen haben, dadurch wurden aus den drei Möglichkeiten 8 finale Interessen definiert, wobei der Registrierungsvorgang die erste Möglichkeit **000b** ausgeschlossen hat, nämlich, dass der User asexuell ist, also kein Interesse an einem der drei Geschlechter hat. Dabei haben sich folgende 7 Restinteressen ergeben:

- **Male**
- **MaleFemale**
- **MaleFemaleOther**
- **Female**
- **FemaleOther**
- **Other**
- **MaleFemaleOther**

Ist ein User nun z.B. interessiert an **FemaleOther**, bedeutet das, er hat Interesse an **Female** und **Other**, aber nicht Male. Beim Auflisten der dataSnapshot von der Datenbank wird nun zusätzlich erfolgreich abgefragt (die eigenen Interessen werden in der **MainActivity onCreate()** abgerufen und in einem private String zwischengespeichert)



```
if (dataSnapshot.child("gender").hasChild(myInterests))
```

3. DER POTENZIELLE USER AUCH INTERESSE AN DIESEM USER HAT

Wie bei der vorigen Abfrage, muss nun eben andersrum festgestellt werden, ob der User denn nun auch den Interessen des potenziellen Partners entspricht. Im Grunde genommen eine umgekehrte Abfrage des vorhergehenden. Allerdings können innerhalb dieser Queries keine anderen, nicht untergeordneten Werte abgefragt werden und diese bezieht sich lediglich auf die Werte des potenziellen Partner Users. (Weshalb auch vorher die eigenen Interessen zwischengespeichert wurden.

An dieser Stelle ist es wichtig, die Geschlechterrollen zu erklären.

Dadurch, dass die Interessen in 7 finale Möglichkeiten unterteilt wurden, wurden auch die Geschlechter so eingeteilt. Statt ein einfaches Geschlecht direkt **Male, Female, Other** zu beschreiben, wurde jedes Geschlecht in alle Gruppen unterteilt, welche das eigene Geschlecht beinhalten. Mit dieser Umstrukturierung der Datensätze wurden weitere detaillierte Queries vermieden.

- gender

- Male: "true"
- MaleFemale: "true"
- MaleFemaleOther: "true"
- MaleOther: "true"

Beim Registrieren resultiert also die Angabe des eigenen Geschlechtes in einem Datenbankeintrag von mehreren Geschlechtergruppen, welche das tatsächliche eigene Geschlecht beinhalten, um dann genau das Schlagwort der vorher definierten Interessen erfüllen zu können.

```
public void setupGenderQuery(DatabaseReference db, String gender){  
    switch (gender){  
        case "Male":  
            db.child("Male").setValue("true");  
            db.child("MaleFemale").setValue("true");  
            db.child("MaleOther").setValue("true");  
            db.child("MaleFemaleOther").setValue("true");  
    }
```

In der **MainActivity** wird also zusätzlich zu dem String der eigenen Interessen auch eine Liste an Geschlechtergruppen **myGender** initialisiert, welche alle eigenen Gruppen des Users umfasst, um dann einfach überprüfen zu können, ob das der User auch dem Interesse des potenziellen Zielusers entspricht.

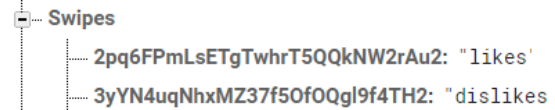
```
if(myGender.contains(dataSnapshot.child("interests").getValue().toString()))
```

Damit konnte bewerkstelligt werden, dass User sich nur gegenseitig vorgeschlagen bekommen, wenn sie auch den jeweiligen Präferenzen des anderen entsprechen.

4. DER POTENZIELLE USER NOCH NICHT GESWIPED WURDE

Das nächste Kriterium, was User erfüllen müssen, um als potenzielle Partner vorgeschlagen zu werden ist, dass sie vom Ausgangsuser in der Vergangenheit nicht bereits *geswiped* bzw. bewertet worden sind.

Dies war einfach, denn beim *Swipen* in der **MainActivity** werden die *swipenden* UserIDs bereits eingetragen, mitsamt der Information, ob sie gefallen am potenziellen Partner haben, oder eben nicht. Dadurch kann beim potenziellen Zieluser einfach überprüft werden, ob der Ausgangsuser bereits in „Swipes“ eingetragen ist.



```
if(!dataSnapshot.child("Swipes").hasChild(myUID))
```

5. DER POTENZIELLE USER NICHT DER EIGENE AUSGANGSUSER IST

Zum Schluss die letzte Lösung im Zusammenhang mit der Query, anstelle einer weiteren Abfrage oder eines Kriteriums, das zu erfüllen galt, *wwipen* sich neue User beim Registrieren automatisch selbst und *disliken* sich, um jegliche Art von Vorschlag oder Match mit sich selbst zu vermeiden.

```
//Sich selbst "Dislike" Swipen, nimmt arbeit für späteres Filtern ab  
dbUserSwipes.child(userId).setValue("dislikes");
```

VOLLSTÄNDIGE QUERY IN MAINACTIVITY

```
public void checkUser() {  
    Query potentialUser = FirebaseDatabase.getInstance().getReference("UIDs");  
    potentialUser.addChildEventListener(new ChildEventListener() {  
        @Override  
        public void onChildAdded(DataSnapshot dataSnapshot, String s) {  
            if(dataSnapshot.exists() && dataSnapshot.child("gender").hasChild(myInterests)){  
                if (myGender.contains(dataSnapshot.child("interests").getValue().toString())){  
                    if(!dataSnapshot.child("Swipes").hasChild(myUID)) {  
                        cards item = new cards(dataSnapshot.getKey(),  
dataSnapshot.child("name").getValue().toString(),  
dataSnapshot.child("ProfilePictureUrl").getValue().toString());  
                        rowItems.add(item);  
                        arrayAdapter.notifyDataSetChanged();  
                    }  
                }  
            }  
        }  
    })  
}
```

Nach vollständiger Erklärung aller Aspekte der Query um Uservorschläge zu erbringen, ist der Block Code in der **MainActivity** hoffentlich verständlich.

FAZIT

Es wurde das Open-Source Projekt, der Dating App „**Zwinder**“ realisiert, welches ebenfalls im Laufe der Entwicklung auf GitHub [16] veröffentlicht und durchkommentiert wurde.

Schnell wurde festgestellt, dass die Entwicklung dem Umfang der Zieldefinition entsprechend nicht ohne hinzufügen zahlreicher Libraries zur Unterstützung gerecht werden konnte. Trotz hinzufügen Dieser, wurde die Zieldefinition jedoch für einen einzelnen Entwickler als zu ambitiös festgestellt und nicht alle Features, welche ursprünglich geplant waren, konnten in Zeit und Umfang dieses Projektes implementiert werden.

So werden „*Matches*“ z.B. angezeigt und eine **MatchActivity** wurde angelegt, aber die nächsten Schritte zum erzeugen einer Benachrichtigung der User sowie anzeigen der „*Matches*“ und Austausch von Nachrichten konnten nicht mehr durchgeführt werden.

Die erarbeiteten Ansätze hierfür wären, über die Firebase Cloud Messaging Api eine Benachrichtigung für den jeweiligen Match zu erzeugen und in der **MatchActivity** die bereits abgespeicherten „*Matches*“ über einen RecyclerView aufzulisten.

Letztendlich konnten jedoch alle Mittel umgesetzt werden, um Useraccounts zu erstellen, diese nach mehreren unterschiedlichen Geschlechtern zu Filtern und ähnlich der Darstellung von Tinder anzuzeigen. Damit bildet „**Zwinder**“ die Basis einer Dating App, welche durch bereits implementierte Mittel weitere Eigenschaften (wie etwa Alter, Location, usw.) der Query hinzufügen könnte, um den Umfang zu erweitern.

LITERATURVERZEICHNIS

- [1] Tinder, Dating Website, <https://tinder.com/?lang=en> (zuletzt geprüft 03.07.2018)
- [2] OkCupid, Dating Website, <https://www.okcupid.com> (zuletzt geprüft 03.07.2018)
- [3] FirebaseAuth, <https://firebase.google.com/docs/auth/> (zuletzt geprüft 03.07.2018)
- [4] Firebase Database, <https://firebase.google.com/docs/database/> (zuletzt geprüft 03.07.2018)
- [5] Firebase Storage, <https://firebase.google.com/docs/storage/> (zuletzt geprüft 03.07.2018)
- [6] Butterknife, Bind Viewing Library, GitHub, <https://github.com/JakeWharton/butterknife> (zuletzt geprüft 03.07.2018)
- [7] Butterknife Website, <http://jakewharton.github.io/butterknife/> (zuletzt geprüft 03.07.2018)
- [8] Butterknife, Viewbinding Library for Android Guide, <https://medium.com/@pranaypatel/butterknife-a-viewbinding-library-for-android-beginner-guide-fd92caf8e505> (zuletzt geprüft 03.07.2018)
- [9] Swipecards, GitHub, <https://github.com/Diolor/Swipecards> (zuletzt geprüft 03.07.2018)
- [10] Glide, Image Loading Library, <https://bumptech.github.io/glide/> (zuletzt geprüft 03.07.2018)
- [11] Glide, GitHub, <https://github.com/bumptech/glide> (zuletzt geprüft 03.07.2018)
- [12] Sourcey's Material Login Demo, GitHub, <https://github.com/sourcey/materiallogindemo> (zuletzt geprüft 03.07.2018)
- [13] Sourcey's Material Login Design Blogeintrag, <https://sourcey.com/beautiful-android-login-and-signup-screens-with-material-design/> (zuletzt geprüft 03.07.2018)
- [14] Android Pattern Matches, <https://developer.android.com/reference/android/util/Patterns> (zuletzt geprüft 03.07.2018)
- [15] SQL vs. NoSQL, TheGeekStuff, https://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/?utm_source=tuicool (zuletzt geprüft 03.07.2018)
- [16] Zwinder App, GitHub, <https://github.com/khstock/zwinder-app> (zuletzt geprüft 03.07.2018)