Party Popper

Leon Hornig
leon.hornig@fh-bielefeld.de

Christian Krebel
christian.krebel@fh-bielefeld.de

Joyce Rafflenbeul
joyce_marvin.rafflenbeul@fh-bielefeld.de

Mobile Applikationen
Prof. Dr.-Ing. Martin Hoffmann

FH Bielefeld
University of
Applied Sciences

Inhaltsverzeichnis

1 Einleitung

Die folgende Projektdokumentation befasst sich mit der Planung, Entwicklung und möglichen Publizierung der in der Projektarbeit des Wahlmodules "**Mobile Applikationen**" entwickelten Android App **Party Popper**, welche in Zusammenarbeit der Studierenden Joyce Marvin Rafflenbeul, Christian Krebel und Leon Hornig im Zeitraum von Oktober 2019 bis Januar 2020 entstanden ist.

Bei möglichen Rückfragen zu der Dokumentation oder zu der Applikation, bitten wir Sie, uns umgehend über eine der genannten E-Mail-Adressen zu kontaktieren.

1.1 Idee

Die Idee der entwickelten Applikation entstand im Oktober 2019 während eines Meetings aller Gruppenmitglieder. Nach einiger Zeit der Überlegungen, ereilte uns die Idee einer mobilen Applikation, welche sich speziell auf die Erstellung und Verbreitung von vornehmlich öffentlichen Veranstaltungen durch Clubs, Bars & anderen Veranstaltungsorganisatoren bezieht.
Nutzern sollen nahe gelegene und besonders beliebte Veranstaltungen priorisiert angezeigt werden, außerdem sollen Nutzer über Aktivitäten anderer, befreundeter Nutzer benachrichtigt werden können.
Nutzer können an Veranstaltungen teilnehmen, sodass diese Benachrichtigungen über den aktuellen Status der Veranstaltungen oder der Teilnahme eines Freundes erhalten.
Des Weiteren erhalten Nutzer die Möglichkeit, anderen Nutzern zu folgen und Organisatoren zusätzlich auch zu bewerten und zu blockieren.

1.2 Herleitung

Die Idee einer Applikation für die Publizierung von Veranstaltungen ist nicht neu, hierbei käme dem allgemeinen Nutzer mit höchster Wahrscheinlichkeit zuerst die Veranstaltungsverwaltung des sozialen Netzwerks "Facebook" in den Sinn, besonders weil diese sich an einer großen Beliebtheit unter den Nutzern erfreut.

Die besondere Frage an dieser Stelle lautet also: “warum sollte man etwas neu erfinden, was bereits in gewisser Weise erfolgreich existiert?”.

Die Antwort: Auch wenn Facebook ihre Veranstaltungsverwaltung schon seit Jahren in ihre Systeme integriert hat, bietet die allgemeine Idee der Veranstaltungsverwaltung weitaus mehr Möglichkeiten der Implementierung von Funktionalitäten und Ausschöpfung von innovativen Ideen, als bis heute von Facebook und anderen Unternehmen ausgenutzt und umgesetzt wurde.

Es haben sich in den letzten Jahren insbesondere in den sozialen Netzwerken viele verschiedene Trends entwickelt, welche auch in vielen Fällen in besonderer Form kombiniert werden können. Viele Applikationen wie Reddit, Stack Overflow oder die deutsche Applikation Jodel nutzen zuweilen ein *Push-System*, welches der Nutzergemeinschaft erlaubt, besonders beliebte oder hilfreiche Posts oder Thread-Einträge anderen Nutzern schnell zugänglich zu machen, indem diese in der Applikation "*ganz oben*" angezeigt werden, wohingegen weniger hilfreiche oder nutzlose Posts sehr weit unten oder dem Nutzer gar nicht mehr angezeigt werden.

Diese Funktion lässt sich in unserer angedachten Applikation sehr gut auf die Relevanz von Veranstaltungen übertragen; beliebte Veranstaltungen sollen also durch ein Push-System noch weiter verbreitet werden, wohingegen unbeliebte Veranstaltungen nach einer Weile dem Nutzer mit nur sehr niedriger Relevanz, also in der Applikation außerhalb des ersten Sichtbereichs angezeigt werden.

Es soll an dieser Stelle aber keinesfalls passieren, dass sich besonders beliebte Veranstaltungen auch immer über *neu erstellte* Veranstaltungen mit anfangs sehr geringer Reichweite platzieren, sodass die Filteroptionen neben einem Filter der beliebtesten Veranstaltungen auch einen Filter der neuesten Veranstaltungen beinhalten soll.

2 Stand der Technik

Es gibt viele verschiedene Mobile Applikationen, welche ihren primären Fokus auf das Finden von nahegelegenen Veranstaltungen legen. Einige legen Wert auf soziale Veranstaltungen und das Interagieren mit neuen Menschen mit z.B. gleichen Interessen. Diese Veranstaltungen können in solchen Applikationen von jedem angemeldeten Nutzer erstellt werden.
Die Applikation Meetup ermöglicht die Erstellung von Veranstaltungen durch jeden Nutzer über Themen wie Kochen und Fahrradfahren [vgl. 2.1].
Andere Applikationen hingegen publizieren nur kommerzielle Veranstaltungen, welche eine Eintrittsgebühr veranschlagen.
Ein Beispiel hierfür ist die Applikation The Move, welche sich auf den Verkauf von Tickets und das Scannen dieser konzentriert [vgl. 2.2].
Party Hunt: Goa enthält sogar verschiedene Ansichten um Veranstaltungen besser entdecken zu können: eine “Posterwand” und eine Kartenansicht [vgl. 2.3]. Weiterhin kann in dieser Applikation ein In-App Rabatt für Veranstaltungen eingelöst werden.
Die nächste Einheit von veranstaltungsorientierten Applikationen beinhalten eine klare Differenzierung zwischen normalen Nutzern und geschäftlichen Nutzern bzw. Organisatoren, da diese nur offizielle Veranstaltungen publizieren möchten.
Eventbrite hat genau aus diesem Grund eine zweite Applikation nur für geschäftliche Nutzer bzw. Organisatoren entwickelt [vgl. 2.4].
Eine letzte Applikation, welche auf Veranstaltungen abzielt, ist Ticketmaster, die ihre Spezialisierung nur auf Veranstaltungen wie Live Shows und Live Performances legt [vgl. 2.5].

2.1 Bekannte Funktionalitäten

Die oben genannten Applikationen haben zusammengefasst folgende Funktionalitäten:

- Veranstaltungen, Parties und Shows werden in einer Listenansicht und/oder in einer Kartenansicht angezeigt
- Veranstaltungs-, Organisator- und Teilnehmerinformationen sind standardmäßig öffentlich einsehbar
- es gibt die Option, Tickets direkt (mit einem Rabatt) in der Applikation zu kaufen
- soziale Interaktion:
    - Bewertung von Veranstaltung und Organisatoren
    - Kommentarfunktionen
    - Folgen von Freunden, Organisatoren, Themen, Gruppen und Künstlern
    - Teilen von Organisatoren und Veranstaltungen

- Funktionen zur Empfehlung von Organisatoren und Veranstaltungen
- Die Option zum Suchen und Filtern
- Die Option eine Veranstaltung zu erstellen
    - als Nutzer
    - und als Organisator
- Die Möglichkeit den Eintritt zu einer Veranstaltung zu vereinfachen (z.B. VIP-Ticket durch In-App Kauf)
- einsehen von öffentlichen Statistiken zu Veranstaltungen und Organisatoren

Viele der oben genannten Applikationen leiden unter toten Servern, geografischen Einschränkungen oder zu vielen bzw. zu wenigen Funktionen. Einen richtigen Durchbruch hat es bis jetzt mit keiner dieser Applikationen gegeben.

3 Anforderungsdokumentation

Im folgenden Kapitel gehen wir auf die funktionalen Anforderungen unserer Anwendung ein, außerdem erstellen wir eine Meilensteinplanung mit festgelegten Zeiten, wann diese fertig sein sollen.

3.1 Funktionale Anforderungen

- **Must have:**
  - Benutzerregistrierung & -anmeldung
  - Organisator-/Unternehmensregistrierung & -anmeldung
  - Hinzufügen von Veranstaltungen durch Organisatoren/Unternehmen
  - Liste aller aktuellen Events, welche in unmittelbarer Nähe des jeweiligen Nutzers liegen
- **Should have:**
  - Events zusagen
  - Organisatoren/Unternehmen bewerten
  - Organisatoren/Unternehmen folgen oder blockieren
  - Veranstaltungen filtern
  - Veranstaltungen suchen
- **Could have:**
  - anderen Nutzern folgen
  - Anzeigen von Veranstaltungen auf einer Karte
  - Hinzufügen von privaten Veranstaltungen durch alle Nutzer
- **Won’t have:**
  - auf Nutzer zugeschnittene Ansichten (z.B. “das könnte dir auch gefallen”)
  - QR-Code Promotions
  - gesponserte Inhalte

3.2 Meilensteine

Meilenstein	**fertiggestellt bis**
Mockups	25. Oktober 2019
Prototypen	01. November 2019
Implementierung der Main Layouts, Activities, Views	15. November 2019
Integration der Google Firebase	20. November 2019
Implementierung zur Erstellung von Veranstaltungen	27. November 2019
Implementierung der restlichen Should-Haves	11. Dezember 2019
Implementierung der Could-Haves	31. Dezember 2019
Projektdokumentation	02. Januar 2020
Projektpräsentation	08. Januar 2020

4 Architekturbeschreibung

Im folgenden Kapitel gehen wir auf die Architekturen in unserer Anwendung ein und veranschaulichen diese mit Diagrammen, damit man diese schneller verstehen kann. Abschnitt 4.1 Client-Server-Architektur zeigt, wie unsere App und die Backend-Services miteinander kommunizieren. Im Abschnitt 4.2 Client Architektur gehen wir genauer auf die einzelnen Activities und deren wichtigsten Funktionalitäten ein.
4.3 Datenbank Architektur beschreibt die Datenbank Modellierung und wieso wir uns für genau diese Architektur entschieden haben.

4.1 Client-Server-Architektur

Als Client nutzen wir ausschließlich eine Android App, aber unsere Server-Architektur ermöglicht es, weitere Clients wie eine Webseite oder weitere Apps zu erstellen und unserem bestehenden System hinzuzufügen. Wie in Abbildung 4.1 gezeigt, besteht unser Backend aus mehreren Firebase Services, die mit der App und untereinander kommunizieren. Die App kann direkt auf die einzelnen Services zugreifen.
So ist es zum Beispiel möglich, clientseitig Datenbank-Queries zu erstellen und diese direkt der Datenbank zu schicken, ohne das dort eine Middleware nötig ist.
Ohne Middleware kommt natürlich die Frage auf, ob ein solcher Ansatz auch sicher ist oder der Nutzer einfach Zugriff auf alle Daten haben soll. Um dieses Problem zu beseitigen gibt es die Firebase Security Rules [4.1], mit welchen man jeden einzelnen Service, jede Anfrage und auch jedes Dokument in der Datenbank und im Storage absichern kann.
In 4.1.1 Authentifizierung wird gezeigt, wie jede dieser Anfragen mithilfe eines JSON Web Tokens authentifiziert wird.

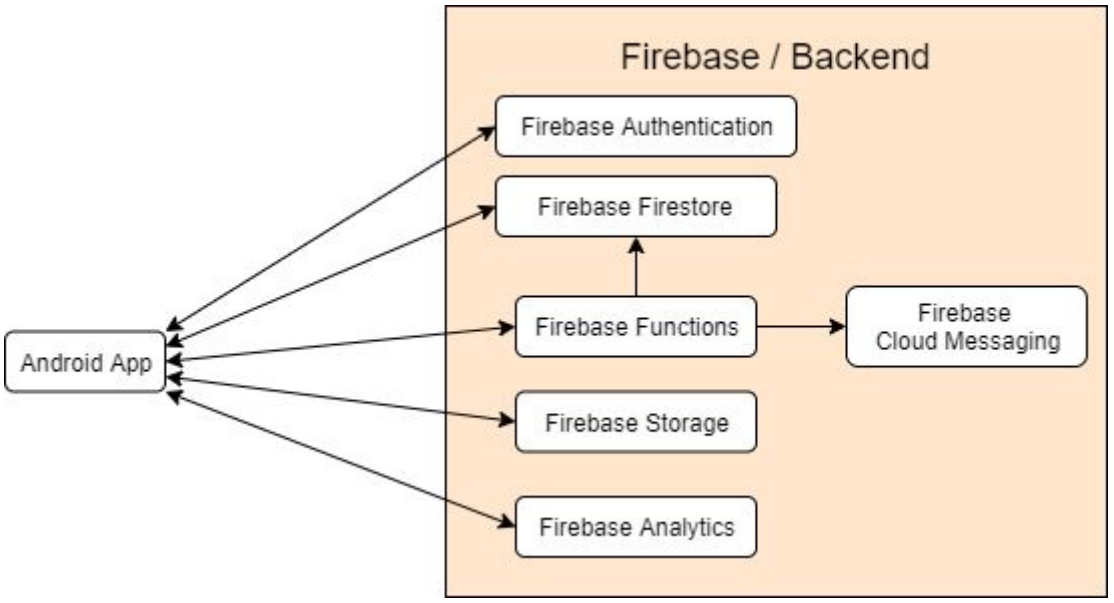


Abbildung 4.1: Zeigt die Kommunikation zwischen der Android App und den Firebase Services.

4.1.1 Authentifizierung

Für die Authentifizierung nutzen wir den Firebase Authentication Service. Dieser Service unterstützt die neuesten OAuth- 2.0 Industrie Standards und ermöglicht uns so eine leichte Integration des Google Authentifizierungsservers, welche ein Login mit einem bereits bestehenden Google Account ermöglicht [4.2]. Dies sorgt bei den Benutzern für mehr Vertrauen, weil diese nicht darauf angewiesen sind, einen neuen Account in unserer App anzulegen und wir somit kein Passwort von den Nutzern speichern müssen. Trotzdem bieten wir als zweite Möglichkeit die Anmeldung per E-Mail Adresse und Passwort an, damit wir auch Nutzern ohne einen Google Account die Nutzung unserer App ermöglichen können.

In der Abbildung 4.2 sieht man den Ablauf einer Authentifizierung. Zunächst sendet die Applikation ein Authentication Request an unseren Firebase Authentication Service. Als Antwort erhält die Applikation einen JSON Web Token (JWT) mit welchem man auf unsere Backend Services zugreifen kann. Der JWT enthält neben den essentiellen Informationen zu dem Benutzer auch einen Timestamp, wann dieser abläuft. Sollte dies der Fall sein, aktualisiert die Applikation diesen automatisch mit dem im JWT enthaltenen Refresh Token. Dies ermöglicht, dass der Benutzer sich auf seinem Gerät nicht nach jedem Neustart neu anmelden muss und erhöht die Sicherheit. Mithilfe des JWT ist es möglich Backend-Services einzubinden, die unabhängig vom Authentication Server arbeiten können, da diese die Anfrage selbst über den JWT autorisieren können. So ist es auch möglich, eigene Backend Services unabhängig von Firebase zu integrieren.

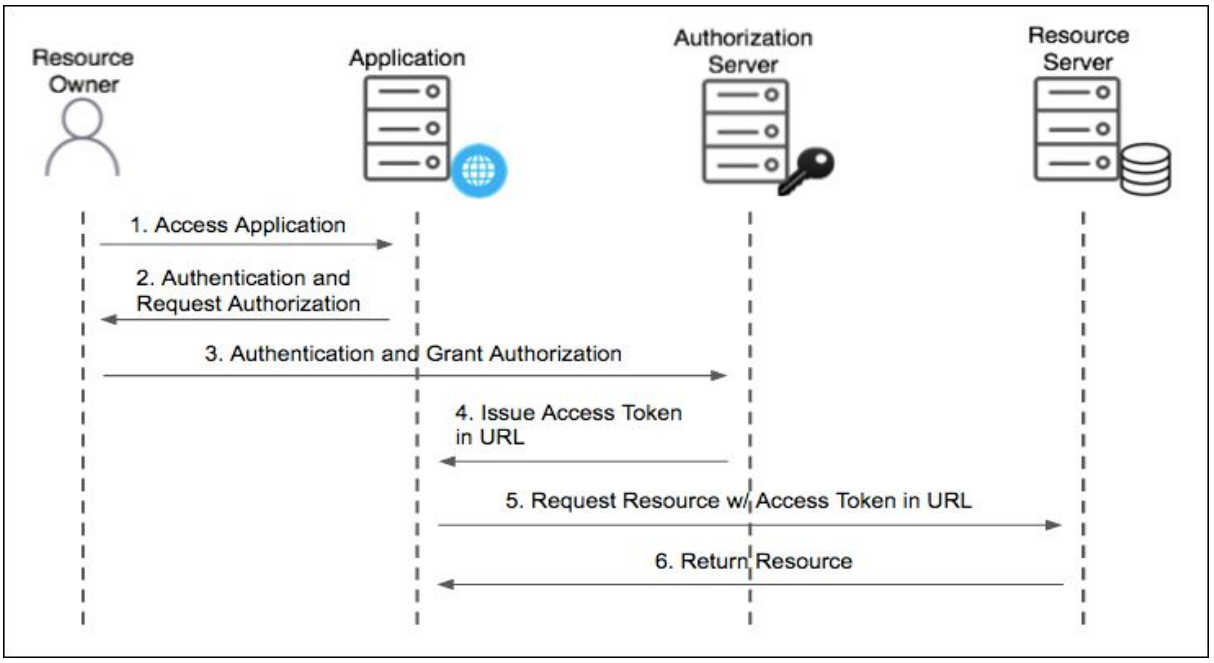


Abbildung 4.2: Zeigt die Authentifizierung um Backend-Services nutzen zu können.

4.2 Client Architektur

Unsere clientseitige Architektur umfasst vier grundlegende Pakete:

1. database
2. features
3. service
4. utils

In den folgenden Teilabschnitten 4.2.1 bis 4.2.5 gehen wir expliziter auf die genaue Umsetzung der wichtigsten Activities des **features-Paketes** ein. Weitere Informationen zu den Paketen ***database***, ***service*** und ***utils*** können aus anderen Abschnitten der Dokumentation entnommen werden.

Das feature-Paket unterteilt sich in die Unterpakete **authentication**, **business**, **dashboard**, **eventDetail**, **events**, **organizer**, **publishEvent** und **splash**.
Diese Pakete enthalten die jeweiligen Activity-Klassen und gegebenenfalls weitere Helfer-Klassen.

Alle Activity-Klassen erben von der Klasse `BaseActivity`.

Abbildung 4.3: zeigt clientseitige Paketstruktur

4.2.1 BaseActivity

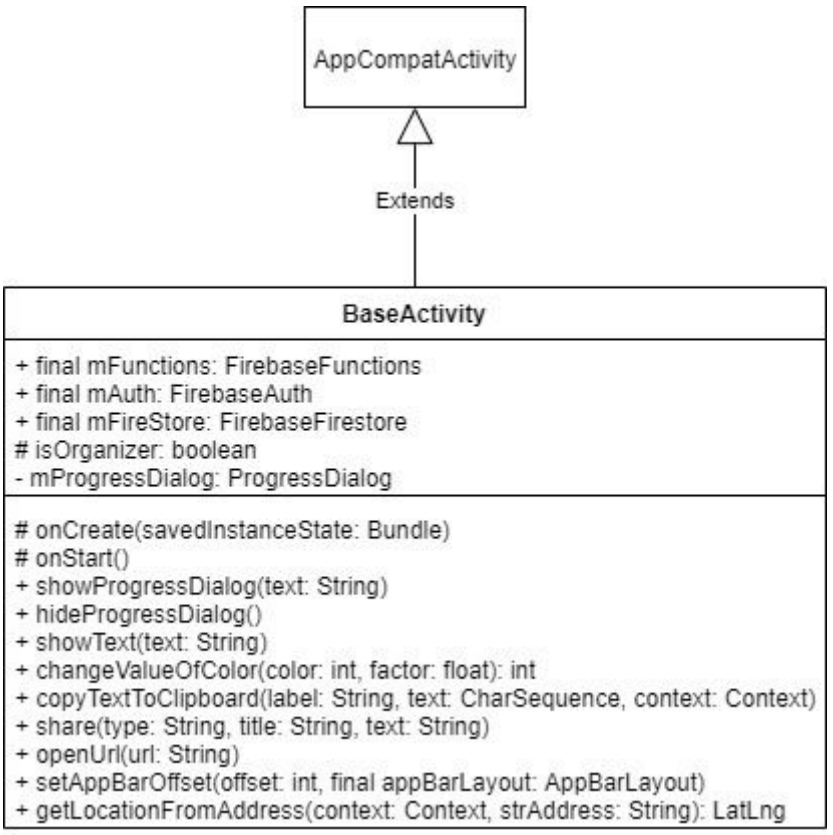


Abbildung 4.4: zeigt UML-Darstellung der Klasse `BaseActivity`

Die Klasse `BaseActivity` umfasst klassischerweise die grundlegendsten Attribute und Methoden, wie zum Beispiel Methoden für das Anzeigen einer Snackbar oder eines Toasts, sowie Metainformationen zur Authentifizierung.

4.2.2 AuthenticationAcitvity

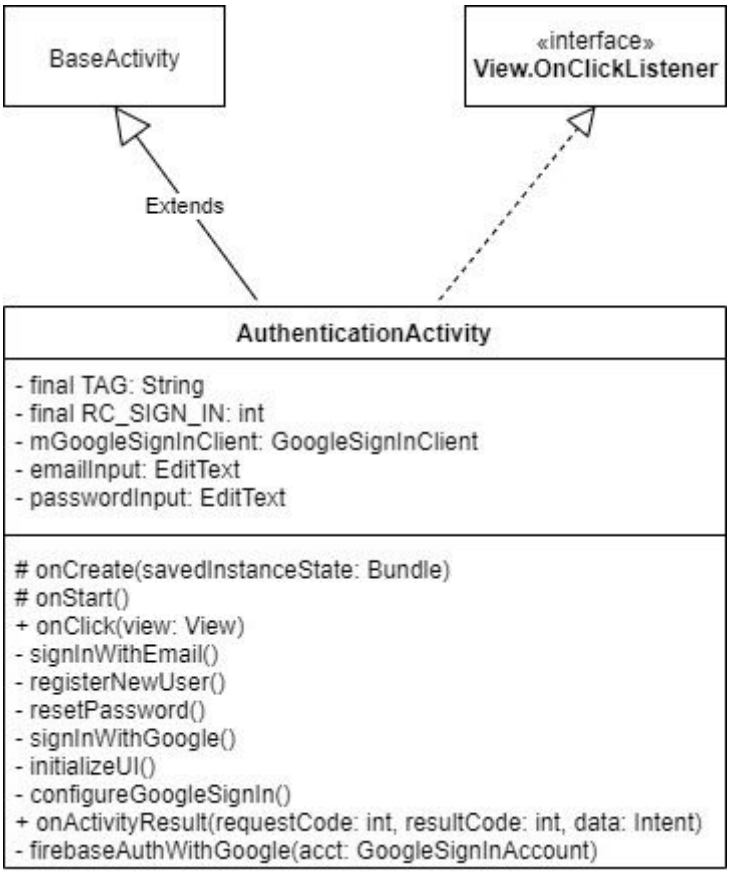


Abbildung 4.5: zeigt UML-Darstellung der Klasse `AuthenticationActivity`

Die Klasse `AuthenticationActivity` dient zur Authentifizierung eines bereits registrierten Nutzers oder zur Registrierung eines dem System noch unbekannten Nutzers.
Bereits registrierte Nutzer werden mit ihrer E-Mail-Adresse und ihrem Passwort über die Methode `signInWithEmail` eingeloggt, die Firebase API stellt hier die Methode `signInWithEmailAndPassword` bereit, welche in Kombination mit einem `onCompleteListener` die Nutzereingaben mit der Datenbank abgleicht.

Die Methode `registerNewUser` erstellt und registriert ein neues Nutzerkonto, die Eingaben des Nutzers müssen auch hier wieder eine gültige E-Mail-Adresse und ein Passwort sein. Die Methode `createUserWithEmailAndPassword` übernimmt auch hier in Kombination mit einem `onCompleteListener` die Registrierung in der Datenbank.

Ein besonderes Highlight, welches sich durch die Nutzung der Google Firebase API ergibt, ist die sehr leichte Implementierung des Logins mit einem existierenden Google-Konto. Hierbei wird ein Intent-Objekt erzeugt und mit dem Rückgabewert der Methode `getSignInIntent` des Objektes `GoogleSignInClient` belegt, anschließend wird dieses Intent-Objekt der Methode `startActivityForResult` mit der Konstante *`RC_SIGN_IN`* übergeben, sodass eine Google Anmeldemaske in einer neuen Activity geöffnet wird. Der Nutzer kann sich an dieser Stelle mit einem gültigen Google-Konto registrieren.

4.2.3 SplashActivity

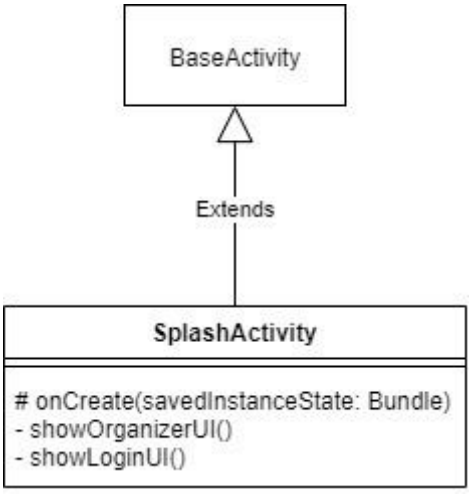


Abbildung 4.6: zeigt UML-Darstellung der Klasse `SplashActivity`

Die Klasse `SplashActivity` kapselt die Funktionalitäten zur Entscheidungsfindung, welcher der beiden Nutzertypen **User** und **Organizer** zu welcher Version der `DashboardActivity` weitergeleitet werden soll.

In der Methode on`Create` wird mit Hilfe des *Google Firebase Registrationtokens* und eines on`SuccessListener` geprüft, ob es sich beim Nutzer um einen Organisator oder um einen normalen Nutzer handelt.

Die Methode show`OrganizerUI` leitet den Nutzer hier zu einer Version des Dashboards weiter, welche eine für Organisatoren angepasste Benutzeroberfläche enthält. Hierfür wird der Methode `setOrganizer`, enthalten in der Klasse `DashboardActivity`, der Wahrheitswert `true` übergeben.

4.2.4 DashboardActivity

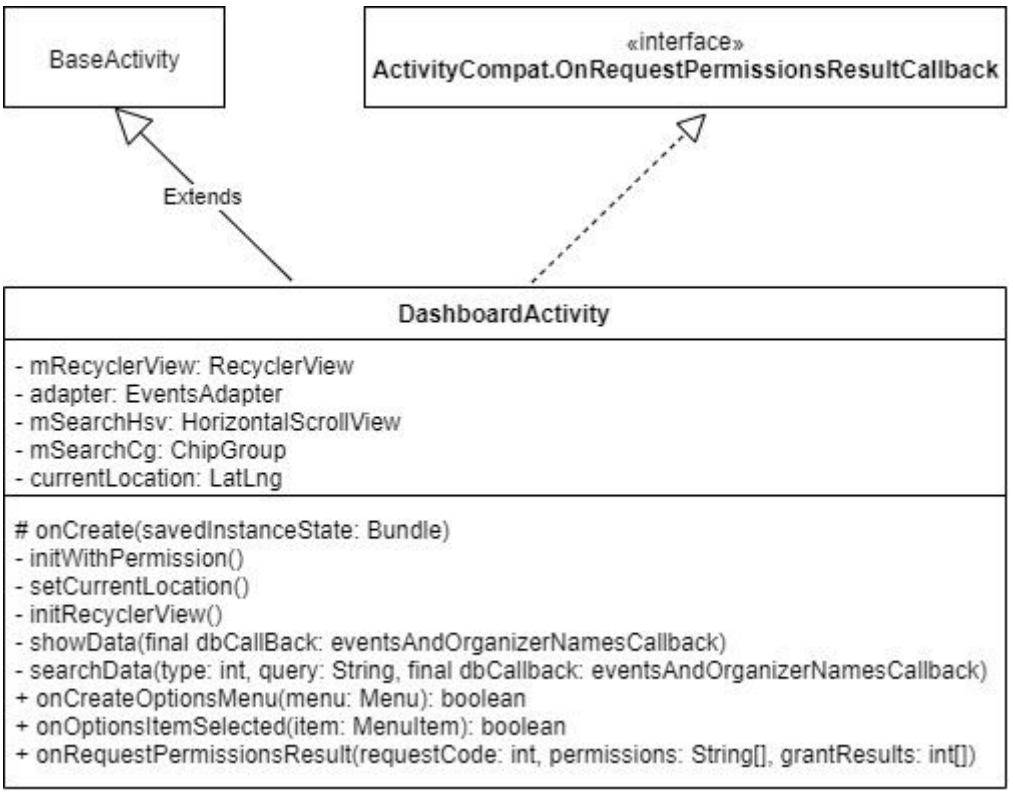


Abbildung 4.7: zeigt UML-Darstellung der Klasse `DashboardActivity`

Die Klasse zeigt dem angemeldeten Nutzer eine Liste von Veranstaltungen.
Die Methode `initWithPermission` prüft, ob der Zugriff auf den Standort eines Nutzers freigegeben wurde, da die Klasse `DashboardActivity` zur Auflistung der Veranstaltungen in der Nähe des Nutzers auf diesen Standort gezwungenermaßen zugreifen muss. Falls eine solche Erlaubnis an das System durch den Nutzer noch nicht erteilt wurde, öffnet sich ein Fenster, welches dem Nutzer (erneut) um die entsprechende Erlaubnis bittet.

Die Methode `showData` kommuniziert mit der Datenbank und befüllt die `RecyclerView` mit den erhaltenen Daten aus der Datenbank für jede Veranstaltung.

`searchData` ermöglicht die Suche nach Veranstaltungen, hierfür erwartet die Methode eine vom Nutzer eingegebene `query`. Die Methode `searchByName` aus der Klasse `EventsRepository` gibt dann innerhalb der Methode `searchData` eine Liste aller zu der Query passenden Veranstaltungen zurück.

4.2.5 Organisator relevante Activities

4.2.5.1 BusinessActivity

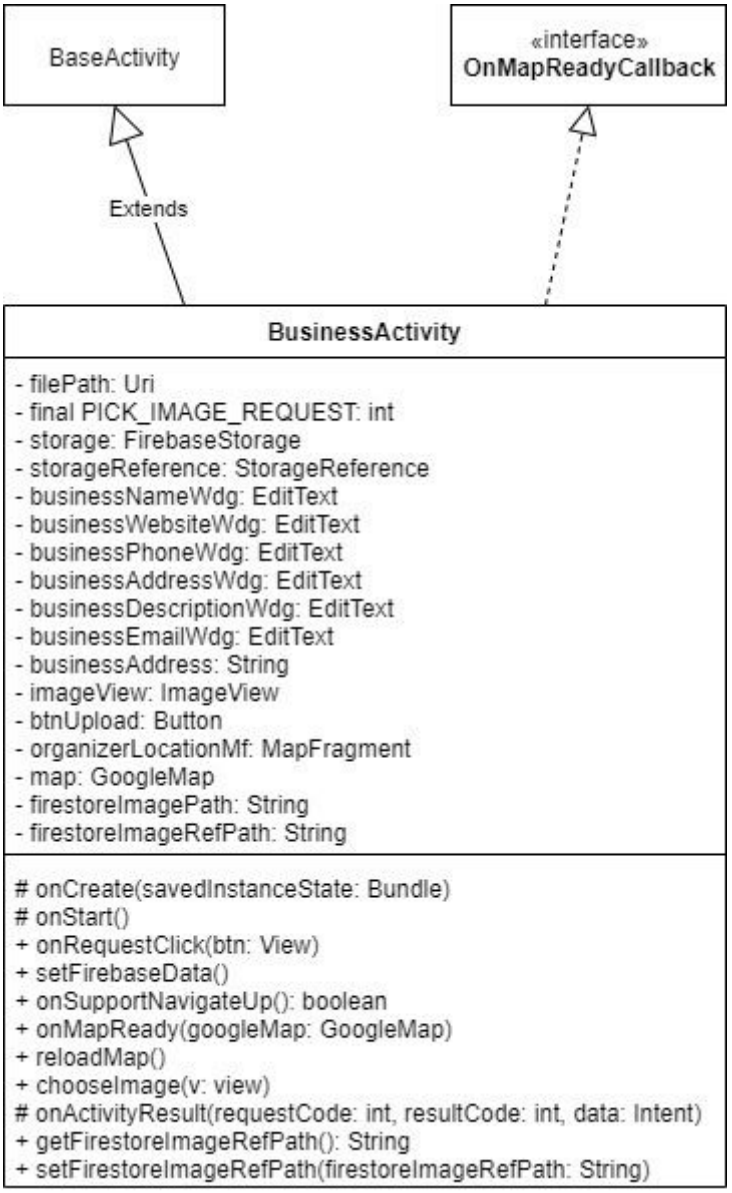


Abbildung 4.8: zeigt UML-Darstellung der Klasse `BusinessActivity`

Die Klasse `BusinessActivity` dient zur Aufwertung eines normalen Nutzerkontos zu einem Organisatorenkonto, außerdem implementiert Sie das Interface `OnMapReadyCallback` zur Konfiguration einer Google Map.

Der Nutzer muss für ein Organisatorenkonto einige Informationen angeben, neben den üblichen Informationen wie Geschäftsname, E-Mail-Adresse und Telefonnummer, muss der Nutzer hier außerdem eine Adresse angeben und kann bei Bedarf ein Profilbild hochladen.
Wenn der Nutzer eine Adresse in das dafür vorgegebene Textfeld eingibt, wird eine eingebundene Google Map [vgl. 4.4] automatisch auf Grundlage der Benutzereingabe aktualisiert. Hierfür werden die Methoden `reloadMap` für das ständige Aktualisieren der Google Map und `onMapReady` für das platzieren eines Markers und der Einstellung eines geeigneten Zooms genutzt.

`onRequestClick` wird ausgelöst, wenn der Nutzer den **Button** mit der Bezeichnung “request” betätigt. In der Methode wird das Profilbild mit Hilfe des FirebaseStorage-Objektes der Google Firebase API in den Firestore Storage [vgl. 4.3] hochgeladen und es wird zugleich eine passende URL zur Referenzierung des Bildes im Firestore Storage für den neuen Organisator in der Firebase erzeugt, welche in der Methode `setFirebaseData` weiter verwertet wird.

Die Methode `setFirebaseData` wertet alle Eingabefelder der Activity aus und kapselt die daraus gewonnen Daten inklusive der URL des Profilbildes in einem Organizer-Objekt, welches mit der Methode `signUpOrganizer` aus der Klasse `OrganizerRepository` in die Datenbank eingepflegt wird.

4.2.5.2 PublishEventActivity

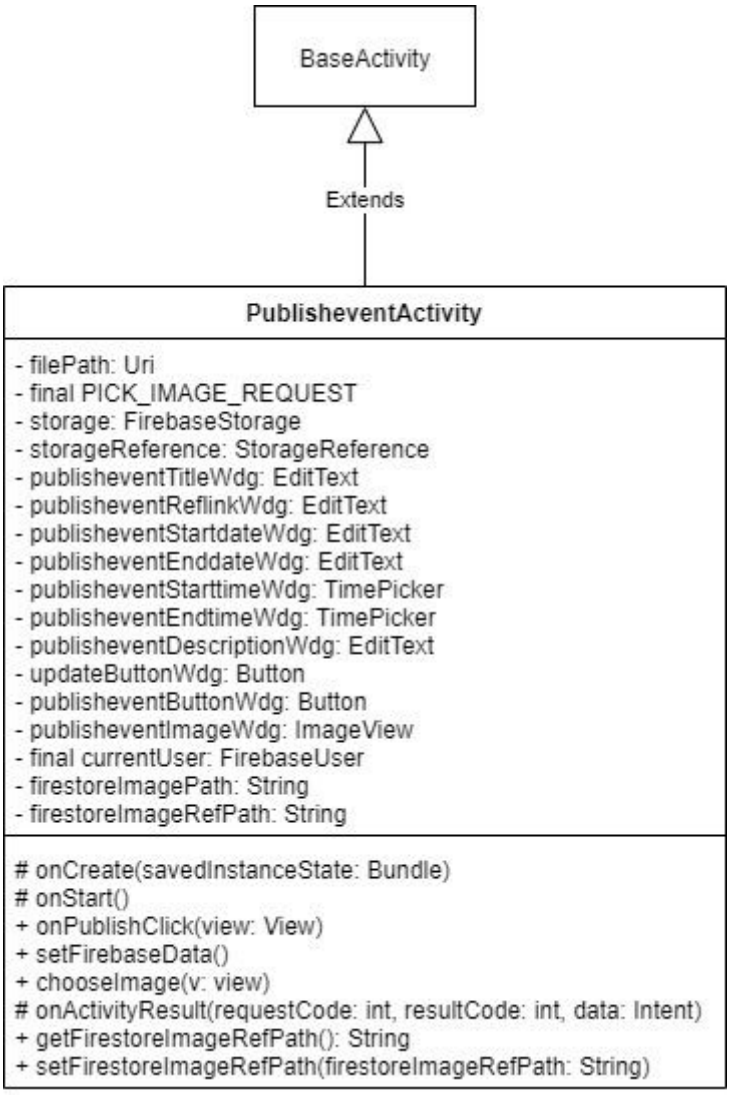


Abbildung 4.9: zeigt UML-Darstellung der Klasse `PublishEventActivity`

Die Veröffentlichung einer Veranstaltung in der Applikation wird durch die Klasse `PublishEventActivity` vorgenommen.
Ähnlich zur Klasse `BusinessActivity` sorgt hier die Methode `setFirebaseData` für die Kapselung aller aus den Eingabefeldern ausgelesenen Daten inklusive der URL des Veranstaltungsbildes (siehe `onPublishClick`) in ein Objekt vom Typ `Event`.

Die Methode `onPublishClick` lädt auch hier ähnlich zur Klasse `BusinessActivity` das gewählte Veranstaltungsbild des Organisators in den Firestore Storage hoch und erzeugt gleichzeitig eine passende Referenz-URL, welche wie oben erwähnt von dem Event-Objekt in der Methode `setFirebaseData` gekapselt wird.

4.3 Datenbank Architektur

Als Datenbank nutzen wir Cloud Firestore von den Firebase Services. Dies ist eine flexible und skalierbare NoSQL Datenbank Lösung [4.3], ähnlich zu MongoDB. Diese arbeitet mit Collections und Dokumenten, welche die Daten enthalten. Cloud Firestore ist so gestaltet, dass es Millionen von Dokumenten innerhalb einer Collection problemlos durchsuchen kann. Man muss aber beachten, dass ein Dokument nicht größer als 1 MiB (1,048,576 bytes) sein darf und man für jedes Dokument, welches an den Client gesendet wird, Kosten auftreten. Da es für die Darstellung einer NoSQL Datenbank keine richtige Norm gibt, haben wir in Abbildung 4.3 versucht, alle Aspekte unserer Datenbank mit einem ER-Diagramm ähnlichen Modell darzustellen. In einer NoSQL Datenarchitektur ist es üblich, redundante Daten in mehreren Dokumenten zu haben, weil es keine Relationen gibt. Dies führt zu dem in einer NoSQL Datenbank vorhandenen Geschwindigkeitsvorteil. Damit wir eine effektive, aber auch kostengünstige Datenstruktur haben, haben wir zwei gute Lösungen gefunden, die wir in den folgenden drei Abschnitten beschreiben. In NoSQL Datenbanken gibt es nicht die perfekte Lösung, sondern man muss sich für jeden Anwendungsfall Gedanken machen, auf welchen Faktor man diese optimieren möchte.

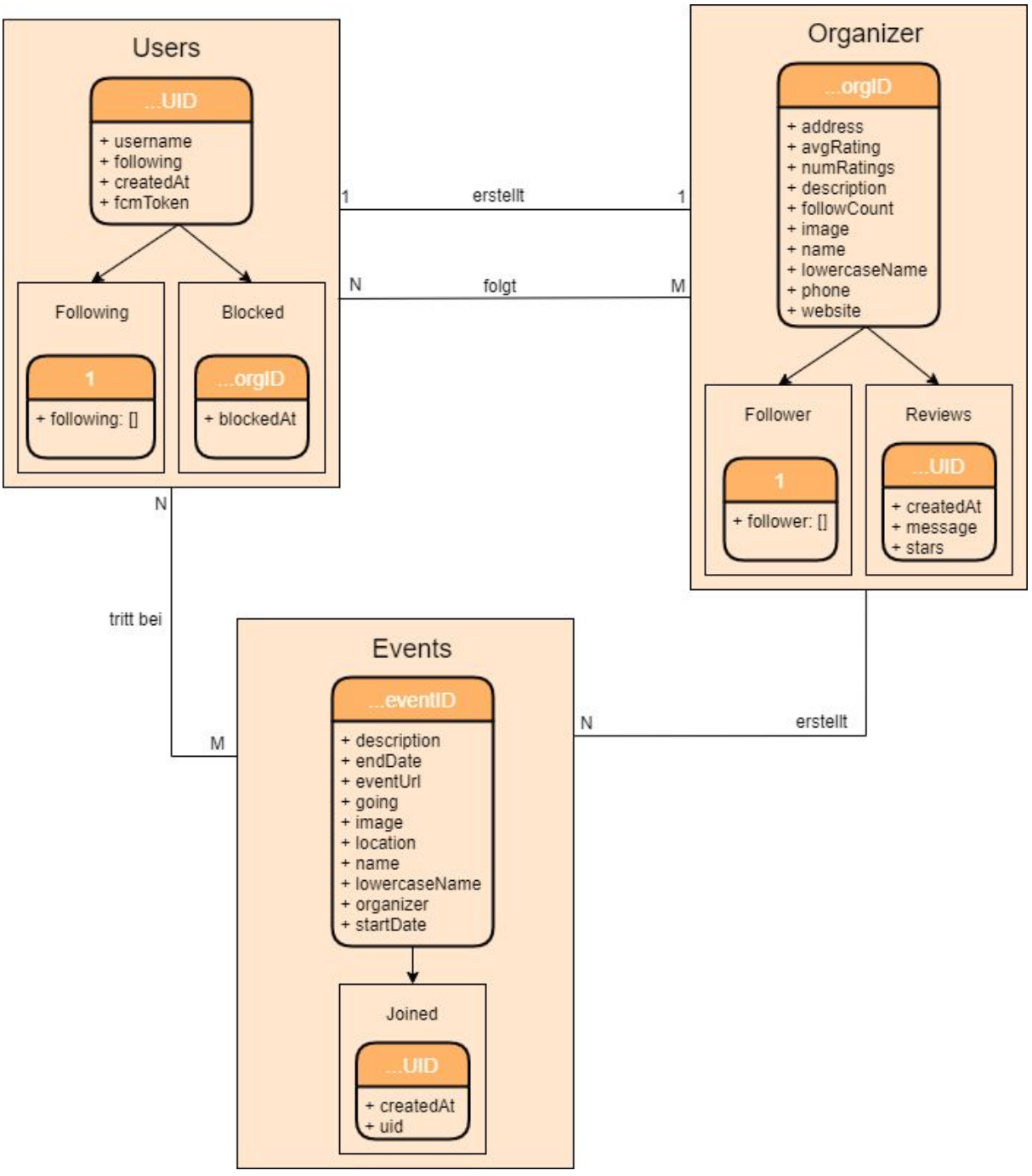


Abbildung 4.10: Zeigt den Aufbau unserer Datenbank mit einem selbst entworfenen Modell. Die großen hellroten Kästen sind Collections und die Kästen mit einem orangenen Kopf sind Dokumente. Wenn in dem Kopf eines Dokuments drei Punkte vor dem Namen stehen, bedeutet dies, dass dieses Dokument häufiger enthalten mit jeweils einem anderen Namen ist. Befindet sich eine Collection in einer größeren Collection, so ist dies eine Subcollection von der oberen. Das bedeutet, dass jedes Dokument in der oberen Collection eine eigene Subcollection enthält, die dem jeweiligen Dokument zugeordnet ist.

4.3.1 Organizer folgen

Zum folgen von Veranstaltern haben wir uns ein komplexes, aber effektives System ausgedacht. Und zwar hat jeder Nutzer eine Subcollection, in welcher nur ein Dokument enthalten ist mit dem Namen „1“ und einem Array von Organizer-IDs von Veranstaltern, welchen er folgt. Und jeder Veranstalter hat eine Subcollection mit jeweils auch nur einem Dokument mit dem Namen „1“ und einem Array von User-IDs. Dies bietet den Vorteil, dass der Veranstalter und auch der Benutzer all seine Follower/Followings in einem Dokument haben und so nur einen einzelnen Lesevorgang benötigen, um diese aufzulisten. Der größte Nachteil ist, dass die Dokumente von der Größe begrenzt sind und somit nicht beliebig viele Follower abgespeichert werden können. Ein Dokument kann maximal 1 MiB (1.048.576 bytes) Daten enthalten. Eine User-ID ist 28 UTF-8 Zeichen oder besser gesagt 28 bytes groß. So ergibt sich eine maximale Menge von 37.449 Followern. Dies ist aber nicht die genaue Zahl, weil das Array noch benannt ist und jeder String jeweils mit Anführungszeichen anfängt und endet. So kommen wir zu einem Limit von etwa 35.000 Followern pro Dokument. Sollte man nun eine größere Menge an Followern benötigen, gibt es die Möglichkeit mehrere Dokumente mit einem next-Attribut aneinander zu hängen. Dies haben wir aber für unseren jetzigen Stand nicht für notwendig gehalten. Sollte es mal zu dem Punkt kommen, dass unsere App so viele Nutzer hat, ist es möglich dies in wenigen Stunden mit hinzuzufügen.

Vorteile

- Nur ein Lesevorgang, um die Follower aufzulisten
- User und Organizer können ihre Follower leicht auflisten

Nachteile

- Array muss vor Asynchronen Aufrufen geschützt werden (Performance Verlust)
- Follower limitiert, solange die Dokumente nicht verbunden werden
- Benötigt eine Backend Funktion aufgrund der Sicherheit, weil der Benutzer sonst die Möglichkeit hat, das ganze Array zu manipulieren.

4.3.2 Organizer bewerten

Zum Bewerten eines Organizers hat jedes Dokument in der Organizer Collection eine Subcollection mit dem Namen Reviews. In dieser Subcollection wird für jeden Benutzer der diesen bewertet hat, ein Dokument mit der jeweiligen User-ID angelegt. In diesem Dokument werden die Daten zum jeweiligen Review abgespeichert; wie und wann wurde es erstellt, wie viele Sterne gibt das Review und eine personalisierte Nachricht. Das besondere an diesem Ansatz ist, dass jeder Organizer so beliebig

viele Reviews erhalten kann. Trotzdem benötigt dieser Ansatz auch ein wenig weitere Implementierung für die Berechnung der Reviews. Auch dazu haben wir mehr in 5.2 Backend geschrieben.

Vorteile

- beliebig viele Reviews
- schnelle Suche nach Nutzer Reviews
- Sicherheit, jeder Nutzer kann nur sein Dokument editieren

Nachteile

- Als Liste anzeigen sind viele Reads (hohe Kosten)
- Schwierig dem Nutzer all seine Reviews zu zeigen

4.3.3 Organizer blockieren

Für das Blockieren der Organizer haben wir denselben Ansatz wie in 4.3.2 Organizer bewerten genutzt. Jedes User Dokument enthält eine Subcollection blocked, in welcher jede Organisation, die von dem Nutzer geblockt worden ist, ein Dokument angelegt wird. Der Organizer weiß in diesem Falle nicht, von welchen Benutzern er blockiert worden ist. Die Vor- und Nachteile sind dieselben und das größte Problem ist, dass eine Darstellung einer Liste sehr hohe Kosten verursachen kann, falls sich ein Benutzer seine tausende blockierten Organizer anschauen möchte.

4.3.4 Organizer erstellen

Jeder Organizer ist auch erstmal ein User und weil jeder User eine einzigartige User-ID hat, wird diese auch verwendet beim Erstellen eines neues Organizer Dokumentes. Somit ergibt sich, dass jeder Organizer auch weiterhin ein User unserer App bleibt und auch die Funktionen wie folgen, bewerten und Events anzeigen besitzt.

4.3.5 Events erstellen

Nur Organizern ist es möglich ein Event zu erstellen. Für die Events gibt es eine eigene Collection unabhängig von den Organizern. Aber sobald ein Organizer ein Event erstellt, wird dessen ID als Wert im Event Dokument angelegt und somit eine Relation zwischen dem Organizer und dem Event erzeugt.

5 Implementierung

Im Folgenden wird auf die wichtigsten bzw. erwähnenswertesten Merkmale der Implementierung eingegangen. Dabei beziehen wir uns zunächst auf das Frontend und danach auf das Backend. Es wird erwähnt, wie vorgegangen wurde, welche Methodiken genutzt wurden und was dabei zu beachten war.

5.1 Frontend

Das Frontend „bezeichnet […] die Presentation Layer, also den Teil eines IT-Systems, der näher am Anwender ist“ [5.1]. Bei Android Applikationen besteht dies zum einen aus den Ressource-Dateien wie XML-Layouts für das GUI[1] und zum anderen aus dem Java-Quellcode, der die Ressourcen nutzt und bindet. Außerdem enthält er die Geschäftslogik, weswegen wir diese ebenfalls unter dem Punkt Frontend betrachten, auch wenn sie als Zwischenglied für Backend und Frontend dient.

5.1.1 Theming

Es ist äußerst wichtig, dass das GUI ansprechend und intuitiv gestaltet ist, damit der Benutzer sich wohl fühlt und die Applikation weiterhin benutzt. Dabei sollen Richtlinien wie die Material Guidelines von Google helfen [vgl. 5.2]. Um diese schnell und einfach implementieren zu können, gibt Google bereits Styles vor, die man als Theme oder direkt als Style nutzen kann. Das „AppTheme“ ist auch ein Style und kann von `Theme.MaterialComponents` erben, wie man im Quellcode 5.1 sehen kann, wodurch die ganze Applikation die neuesten Standards der Guidelines befolgt [vgl. 5.3].

```
<!-- Base application theme. -->
<style name="AppTheme"
parent="Theme.MaterialComponents.NoActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    ...
</style>
```

Quellcode 5.1: Das Theme, das für die gesamte Applikation genutzt wird, erbt von einem Material Theme von Google.

[1] Graphical User Interface, dt. Benutzeroberfläche

Damit dieses Theme gefunden werden kann, ist es noch nötig, Googles Maven Repository zur Projekt-build-Datei hinzuzufügen und die Bibliothek `com.google.android.material:material` in der Modul-build-Datei zu implementieren.
Das Theme regelt die meisten Styles, jedoch werden vereinzelt weitere Styles benötigt, die man dann per style-Attribut einem View zuweist. Ein Beispiel wäre `Widget.MaterialComponents.Button.OutlinedButton`, welches einem Button das Aussehen verleiht, keinen Hintergrund, aber Außenlinien bzw. Kanten zu besitzen.
Einige weitere Views können zudem aber auch gänzlich ersetzt werden. Zum Beispiel haben wir anstelle von den Android üblichen EditText-Views ein `com.google.android.material.textfield.TextInputEditText` genutzt, welches das direkte Kind eines `com.google.android.material.textfield.TextInputLayout` sein muss. In Abbildung 5.1 kann man den Unterschied gut erkennen, es gibt dazu verschiedene Arten zur Auswahl von Google.

Abbildung 5.1: Das Standard-EditText-View von Android (links) im Vergleich mit den neuen Varianten von Google (rechts).

5.1.2 Events in einem RecyclerView

An einigen Stellen in der Applikation werden Events aufgelistet. Seien es solche aus der Nähe oder nur die neuesten Events eines spezifischen Veranstalters. Für performante Listen nutzt man heutzutage das RecyclerView, welches Einträge dynamisch laden und entladen kann, was ein Vorteil gegenüber eines ListViews ist. Um ein RecyclerView mit Einträgen zu befüllen, wird eine Variante des Model-View-Controller-Konzepts genutzt. Im Folgenden wird sich dem Verständnis wegen oft auf Abbildung 5.2 bezogen, welches ein Klassendiagramm für einen kleinen Teil der Applikation zeigt.

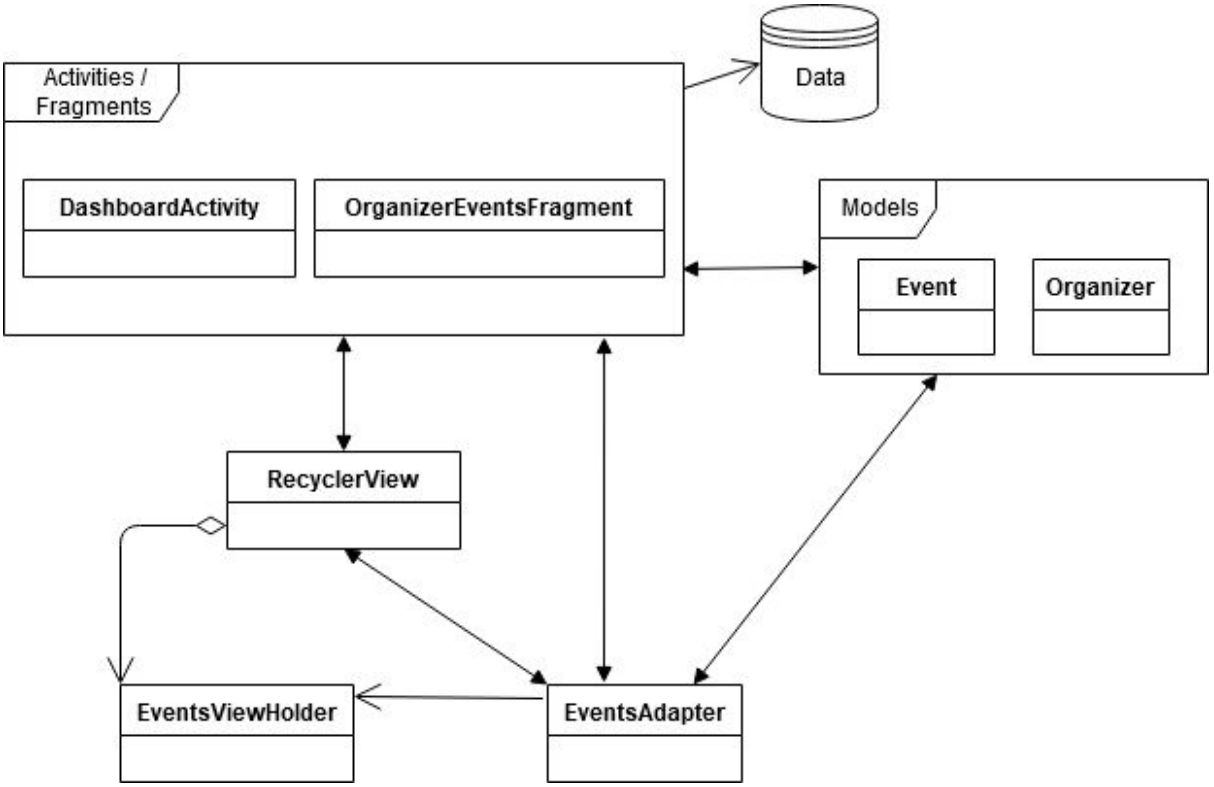


Abbildung 5.2: Klassendiagramm, das die Beziehungen zeigt, wie Daten gebunden und in einem RecyclerView angezeigt werden. In diesem Fall sind es Events.

Bevor man jedoch genauer darauf eingeht, wie die Daten gebunden und geladen werden, sollte man verstehen, dass hier zwei MVC-Konzepte ineinander geschachtelt vorliegen.
Zum einen gibt es die Model-Klassen `Event` und `Organizer`, welche Objektklassen sind. D.h. sie können instanziiert werden und besitzen genau die gleichen Attribute, wie in der Datenbank vorgegeben. Für diese Attribute gibt es Getter- und Setter-Methoden. Zur Simplifizierung nicht in Abbildung 5.2 angezeigt, werden Datenbankanfragen per Repository-Klassen durchgeführt. Sie vereinfachen diese und bilden die Daten direkt auf Objektklassen ab. Eine Activity oder auch ein Fragment kommt dadurch also schnell und einfach an eine Liste von Events. Dabei fungieren die Activity bzw. das Fragment und die Repository-Klassen als Controller, da sie steuern, was von wo und wie in die Model-Klassen konvertiert wird. Der `EventsAdapter` entspricht dann dem View-Element, er entscheidet wie die Model-Daten angezeigt werden [vgl. 5.4].
Zum anderen kann man auch den Adapter und das Konzept drumherum als eine Art MVC-Konzept ansehen. Wenn wir nun also eine Liste der Events instanziiert haben, können wir diese als Argument bei der Instanziierung eines Adapters übergeben.

Der Adapter erstellt dann genug `EventsViewHolder` passend für jedes Event und entscheidet dabei auch, welches Layout für ein Event-Element im RecyclerView genutzt werden soll. Ein ViewHolder hat die Aufgabe, die einzelnen Views eines RecyclerView-Eintrags wie ImageView, TextView etc. zu instanziieren, indem er die passenden IDs aus den XML-Layout-Dateien findet. Ein Interface namens `ClickListener` definiert er ebenfalls und instanziiert es als Attribut, so dass der Adapter beim erstellen von ViewHoldern neues Verhalten für das Klicken auf ein Event-Element im RecyclerView implementieren kann. Sonst würde dies im EventHandler geschehen.
Nach dem Erstellen bindet der Adapter die ViewHolder. Dies bedeutet, dass zu jeder Position im RecyclerView nun auch jeweils ein ViewHolder und ein Model existieren. Die Views des ViewHolders werden dabei mit den Daten des Modells beschrieben. Nun ist alles nötige getan, um die Einträge im RecyclerView anzuzeigen und diese auch auf Klicke reagieren zu lassen. Um das Dashboard von allen weiteren Activities bzw. Fragmenten zu unterscheiden, da es die zentrale Haupt-Activity ist, wird dabei ein anderes Layout für die Einträge genutzt wie man anhand von Abbildung 5.3 erkennen kann.

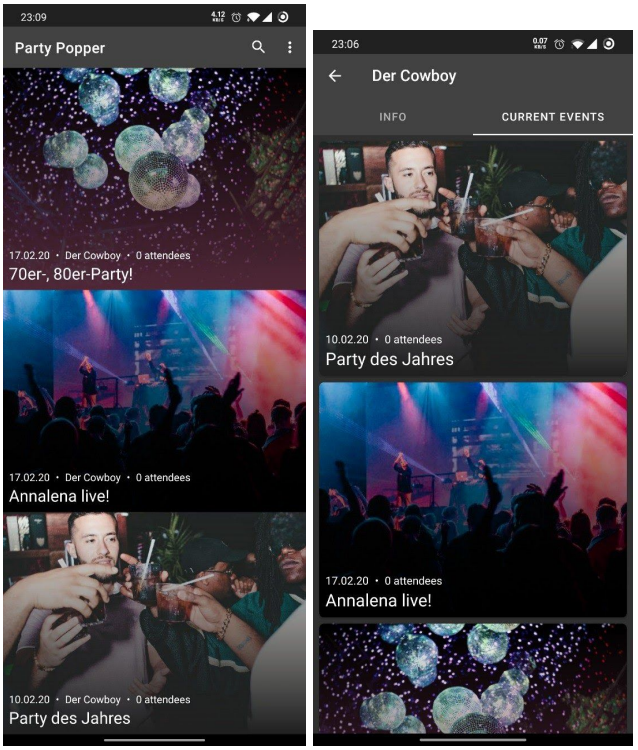


Abbildung 5.3: RecyclerView befüllt mit Event-Einträgen auf der `DashboardActivity` (links) und auf dem `OrganizerEventsFragment` (rechts). Das Dashboard beinhaltet die Events der Umgebung und die Karten besitzen keinen Abstand, während die Liste beim Veranstalter-Profil die neuesten Events des Veranstalters beinhaltet und die Karten einen Abstand besitzen.

5.1.2.1 Probleme und Schwierigkeiten

Vor dem Implementieren des RecyclerViews und seiner notwendigen Klassen kommt einem die Frage auf, welche Struktur man genau nutzen möchte. Zum Beispiel könnte man den Adapter, der von der open-source Bibliothek FirebaseUI vorgegeben wird, nutzen [vgl. 5.5]. Dann wäre man jedoch eingeschränkt im Bezug auf objektorientierter Programmierung und der Wiedernutzung des Adapters, weswegen wir uns für die flexiblere Variante der eigenen Implementierung entschieden haben, die auch die ClickListener beinhaltet.

5.1.3 Fragmente und Tabbed Activity

Es kann dazu kommen, dass mehrere Activities denselben Inhalt teilen möchten. Um redundanten Code zu vermeiden, kann man hierfür Fragmente nutzen. Was Fragmente sind und wie man sie bei einer Activity mit Tabs nutzt, wird folgend genauer erläutert.

5.1.3.1 Fragmente

Fragmente sind keine Activities und erben auch nicht von ihnen. Sie repräsentieren Verhalten oder einen Teil der Benutzeroberfläche [vgl. 5.6]. Man kann sie dynamisch zur Laufzeit einer Activity hinzufügen oder entfernen. Außerdem helfen sie dabei, das Layout passend zu gestalten, wenn man den Landscape-Modus unterstützen möchte, was wir bei unserer Applikation jedoch nicht implementiert haben. Fragmente kann man programmatisch als auch per Layout nutzen. In dem Fall der Wiedernutzung ist das MapFragment ein gutes Beispiel [vgl. 5.7]. Dieses nutzen wir bei der `EventDetailActivity`, die Detailansicht eines Events, und dem `OrganizerInfoFragment`, das Informationstab auf dem Veranstalterprofil, da dort eine Karte angezeigt werden soll. Google gibt dafür eine Klasse vor, die ein normales Fragment nutzen muss.

5.1.3.2 Tabbed Activity

Das Implementieren von eigenen Fragmenten und die Nutzung dieser im programmatischen Sinne ist komplexer als Fragmente im Layout zu definieren. Die Activity `OrganizerActivity`, welches das Profil eines Veranstalters zeigt, besitzt die zwei Tabs „Info“ und „Current Events“. Um diese mit unterschiedlichen Inhalt zu befüllen, kann man Fragmente mit einem FragmentPagerAdapter steuern. Dieser weiß, welches Fragment für welches Tab instanziiert und zurückgegeben werden muss. In unserem Fall gibt es ein Fragment für alle Informationen des Veranstalters und eines für die Auflistung seiner Events. Um Daten zwischen diesen und der Activity zu teilen, könnte man dazu ein ViewModel implementieren. Wir haben uns dagegen entschieden, da die Fragmente sehr unterschiedlich sind und die wenigen

Daten, die die Activity den Fragmenten übergeben möchte, per Argument an das jeweilige Fragment übergeben werden, während der FragmentPagerAdapter sie instanziiert. Damit nur eine Instanz der jeweiligen Fragmente vorherrscht, implementieren diese das Factory-Method-Pattern.
Views der `OrganizerActivity` werden in dieser initialisiert, Views der Fragmente wiederum innerhalb der Fragmente. Diese Aufteilung ist nicht immer vorteilhaft, jedoch wird das jeweilige Layout nur dort gesetzt. Das sorgt dafür, dass onClick-Methoden programmatisch umgesetzt werden müssen, da das onClick-Layout-Attribut eine Methode in einer Activity benötigt.

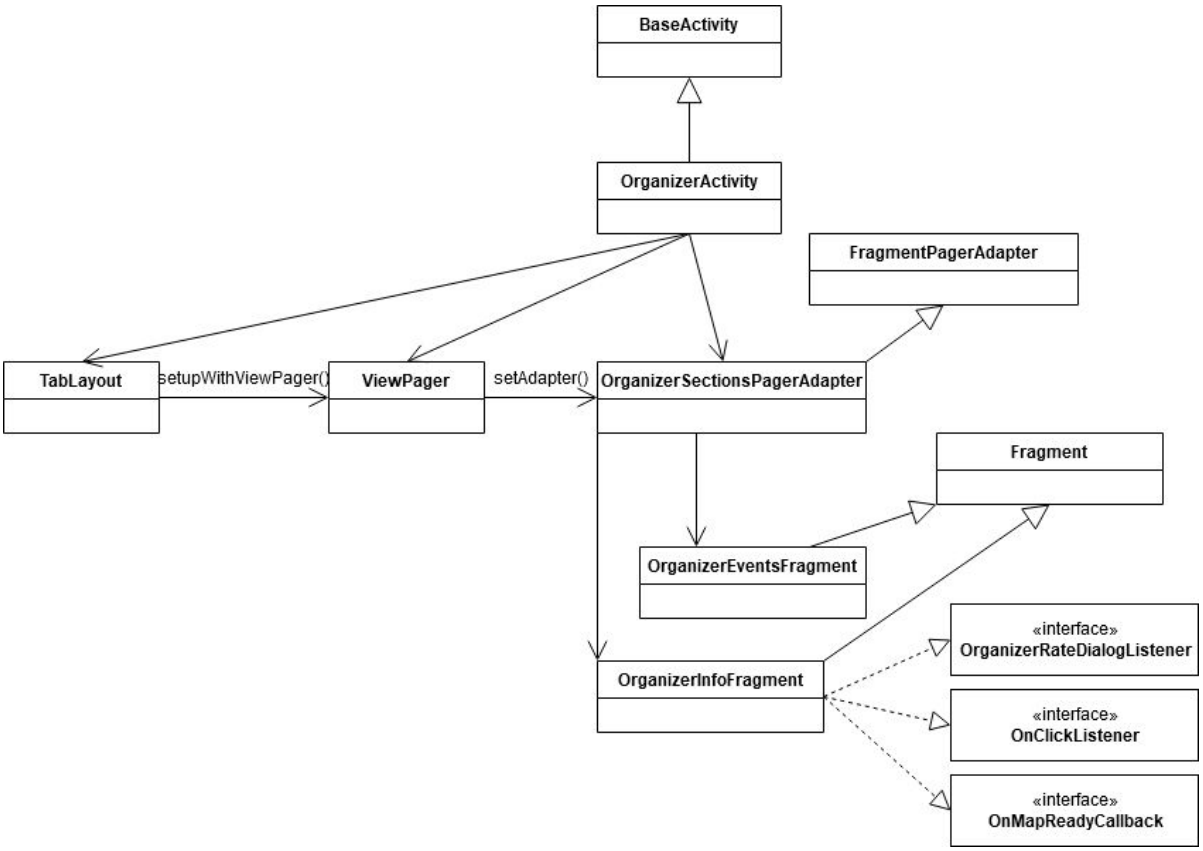


Abbildung 5.4: Klassendiagramm für die `OrganizerActivity` und ihre Fragmente. Damit Tabs angezeigt werden, wird ein TabLayout benötigt. Dieser nutzt einen ViewPager, mit dem man mit einer Wischgeste zwischen Fragmenten wechseln kann. Dieser weiß, welche Fragmente beachtet werden sollen durch den `OrganizerSectionsPagerAdapter`. Da das `OrganizerInfoFragment` ein MapFragment anzeigt, implementiert es den `OnMapReadyCallback`, auf Klicke reagieren möchte, den `OnClickListener` und die Daten eines Dialogs zum Bewerten eines Veranstalters zurückbekommen möchte, den `OrganizerRateDialogListener`.

5.1.4 onResume()

Im Lebenszyklus einer Activity gibt es viele verschiedene Stadien. Die `onResume()`-Methode wird z.B. aufgerufen, wenn die Activity wieder in die Sicht des Nutzers kommt [s. Abbildung 5.5]. Für das Frontend kommt dies zum Vorteil, da einige Views wie RatingBars oder Buttons beim Zurücknavigieren nicht mehr aktuell sein könnten, weil sie sich auf Daten der Datenbank verlassen. Es könnte also dazu kommen, dass sich die Bewertung eines Veranstalters oder der Zustand, ob man dem Veranstalter folgt, verändert hat. Damit der Nutzer sich nicht beim Navigieren wundert, rufen wir Methoden, die nur den Stand einer View aktualisieren sollen, in der `onResume()`-Methode auf.

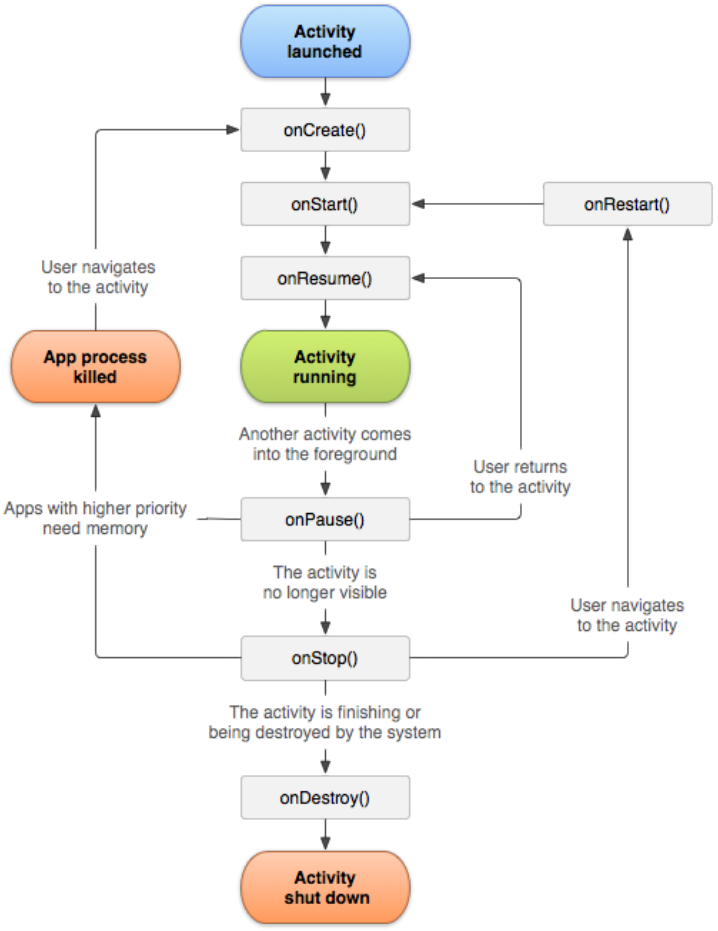


Abbildung 5.5: Der Lebenszyklus einer Activity.

5.2 Backend

Das Backend setzt sich, wie in Abbildung 4.1 bereits gezeigt, aus mehreren Firebase Services zusammen. Dazu gehören Authentication, Firestore, Functions, Storage, Analytics und Cloud Messaging. Diese arbeiten alle gemeinsam, um dem Client ein gut strukturiertes Backend zu bieten. Dies ermöglicht eine große Skalierung der Anwender, weil jede Funktionalität ein eigener Service ist und somit in der Google Cloud laufen kann. In den folgenden Abschnitten gehen wir mehr auf die einzelnen Services ein und wie diese funktionieren.

5.2.1 Authentication Service

Mithilfe des Authentication Services ist es möglich gewesen, innerhalb von wenigen Stunden eine komplett funktionierende OAuth 2.0 Authentifizierung zu erstellen und in unserer App zu integrieren. Serverseitig mussten wir bei dem Service nur Google Authentication aktivieren und schon war es uns möglich uns mit ein paar Zeilen Code im Client mit Google zu verbinden [5.8]. Des Weiteren nutzen wir den Authentication Service, um dem Nutzer mithilfe von Functions einen Custom Claim hinzuzufügen, falls dieser ein Organizer ist [5.9]. Dieser wird dann bei jeder Anfrage mithilfe eines JWT mitgesendet und kann dann in den Services unabhängig abgefragt werden.

5.2.2 Firestore

Als Datenbank nutzen wir die NoSQL-Lösung Firestore. Auf diese Datenbank können wir von unseren Functions und auch von unserem Client aus direkt zugreifen. Als nächsten Schritt müssten wir die Datenbank für Client Zugriffe absichern, mit sogenannten Security Rules. Dies haben wir bei unseren jetzigen Stand aber für noch nicht nötig gehalten, weil wir vor der Veröffentlichung noch einige Features hinzufügen würden. Die serverseitigen Functions haben Adminrechte beim Zugriff auf die Datenbank und können beliebige Daten ändern. Clientseitig haben wir für den Zugriff auf die Datenbank Repository-Klassen angelegt, welche die verschiedenen Queries beinhalten. Da gibt es zunächst die Klasse `FirestoreRepository`, welche die Implementierung von 6 Standard Methoden (`exists`, `get`, `getAll`, `create`, `update`, `delete`) beinhaltet. Die Klasse wurder generisch implementiert und kann ein beliebiges `Model` erhalten, welches unsere Klasse `Identifiable` für die Dokument ID einbindet. Des Weiteren gibt es noch spezialisierte Repositories, welche für die einzelnen Anwendungsfälle konzipiert wurden, darauf wird am Ende weiter eingegangen.

Die einzelnen Query Antworten (JSON) werden von unserem generischen `SimpleMapper` in ein POJO geparst.

`EventsRepository`
Dieses Repository ist für die Auflistung, Suche und Beitreten von Events da.

`FollowRepository`
Dieses Repository ist für das Folgen, Blockieren und Bewerten von Organizern zuständig.

`OrganizerRepository`
Dieses Repository ist für die Verwaltung von Events und Registrierung eines neuen Organizers zuständig, weil man für diese Funktionalitäten eine gültige Organizer ID benötigt bzw. bekommt.

5.2.3 Functions

Für die Logik unseres Backends nutzen wir Firebase Functions. Die Funktionen können von der App direkt aufgerufen werden und bekommen als Antwort einen asynchronen Task, welchen man mit einem `OnCompleteListener` und `OnFailureListener` bearbeiten kann. Die Functions kann man per Namen aufrufen und es ist möglich eine Map mit Daten zu übergeben. Die Functions werden in JavaScript geschrieben und anschließend mit dem Befehl `firebase deploy` an die Google Cloud gesendet. Zu Testzwecken ist es einfacher diese per `firebase serve --only functions` lokal auszuführen und per Postman zu testen, weil der Deploy in die Google Cloud längere Zeit benötigt. Zur Übersichtlichkeit haben wir die Funktionen auf mehrere Module aufgeteilt, die alle anschließend im `index.js` zusammengeführt werden.

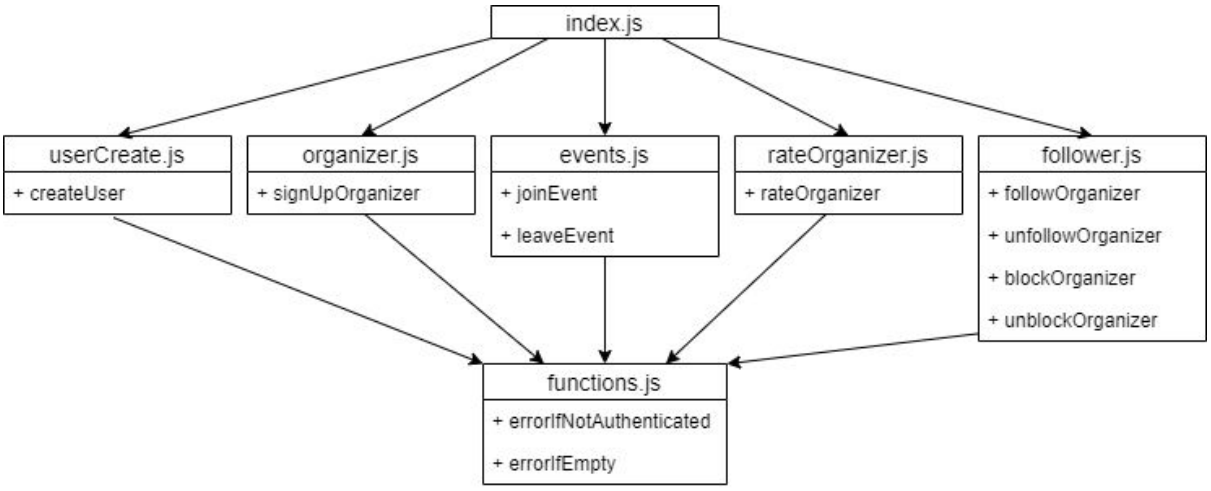


Abbildung 5.6: Zeigt die Struktur unserer Module.

5.2.3.1 createUser

Die `createUser` Funktion wird automatisch ausgeführt, sobald sich ein neuer Benutzer registriert. Dies hat den Sinn, dass wir neben den Standarddaten auch weitere Daten zu dem Benutzer speichern können, weil diese Funktionalität vom Authentication Service nicht direkt unterstützt wird.

5.2.3.2 signUpOrganizer

Die `signUpOrganizer` Funktion kann von jedem authentifizierten Benutzer aufgerufen werden mit einigen Extra-Informationen zum Organizer. Dies erstellt anschließend automatisch ein Organizer Dokument mit seiner dazugehörenden Subcollection für Follower. Falls wir nun aber möchten, dass nicht jeder Organizer werden kann, könnten wir einen weiteren Wert hinzufügen, dass dieser Organizer noch nicht von uns bestätigt worden ist.

5.2.3.3 joinEvent/leaveEvent

Diese Methoden können nur von authentifizierten Benutzern aufgerufen werden und wurden als Funktion geschrieben, weil neben dem Erstellen des User Dokuments in der Joined Subcollection noch der going-Wert im Event Dokument erhöht werden muss. Eine schönere Variante wäre gewesen, das User Dokument in der Joined-Collection Clientseitig zu erstellen und eine Funktion zu haben, die auf ein Datenbank Update in dieser Collection wartet und so den going-Wert erhöht. Mit unserem Ansatz ist es aber auch problemlos möglich, beansprucht jedoch nur einige Millisekunden mehr an Rechenleistung. Beim Verlassen eines Events wird das Dokument gelöscht und der going-Wert wird wieder runtergezählt.

5.2.3.4 rateOrganizer

Das Bewerten eines Organizers ist für jeden unserer authentifizierten Benutzer möglich und wurde in einer Funktion geschrieben wegen der komplexen Berechnung. Zunächst wird eine Transaktion gestartet, weil wir aus dem Organizer Dokument Daten zum berechnen brauchen und sich diese während der Berechnung nicht verändert dürfen. In der Transaktion prüfen wir zunächst, ob der Benutzer den Organizer bereits bewertet hat. Ist dies der Fall, wird das alte Dokument geladen und später vom neuen Review überschrieben. Anschließend berechnen wir mit der `Durchschnittsbewertung * Anzahl Bewertungen` die gesamte Zahl der vergebenen Sterne. Diese Zahl addieren wir jetzt mit der Anzahl des neuen Reviews

und dividieren durch die Anzahl der Reviews, um einen neuen Durchschnitt zu erzeugen. Sollte der Benutzer den Organizer bereits bewertet haben, erhöhen wir die Anzahl der Reviews nicht um eins. Außerdem rechnen wir die neue Anzahl an Sternen minus die alte Anzahl an Sternen um entweder Sterne hinzuzufügen oder vom alten Review zu entfernen. Anschließend werden die neuen Werte im Organizer Dokument gespeichert.

5.2.3.5 follow/unfollowOrganizer

Als angemeldeter Benutzer ist es möglich, Organizern zu folgen und so die aktuellsten Nachrichten von diesen zu bekommen. Hier tritt das in 4.3.1 Organizer folgen bereits erwähnte Problem auf, dass ein Array normalerweise nicht für asynchrone Aufrufe geeignet ist und wir dieses absichern müssen. Dies ist mit der im firebase-Admin enthaltenen Funktion `admin.firestore.FieldValue.arrayUnion(WERT)` möglich. So wird beim Folgen eines Organizers alles gleichzeitig ausgeführt. Die UID wird in der Subcollection des Organizers in das Dokument 1 mit arrayUnion hinzugefügt und gleichzeitig der followCount des Organizers mit `admin.firestore.FieldValue.increment(1)` erhöht. Genau dasselbe findet zeitgleich in der Subcollection des Users statt, nur mit der Organizer-ID. Sein following-Count wird ebenfalls erhöht. Zum Entfolgen eines Organizers werden die IDs in beiden Subcollections mit arrayRemove entfernt und die jeweiligen Zählerwerte mit -1 inkrementiert.

5.2.3.6 block/unblockOrganizer

Auch das Blocken eines Organizers kann man nur als angemeldeter Benutzer ausführen. Dort wird als erstes versucht dem Organizer zu entfolgen, weil wenn man einem Organizer folgt und blockiert, kommt es in den Notifications zu fehlern. Anschließend wird die Organizer-ID als einzelnes Dokument in die blocked Subcollection des Users hinzugefügt. Zum Entblocken eines Organizers wird dieses Dokument einfach gelöscht.

5.2.4 Storage

Der Firestore Storage [vgl. 4.3] dient zur Speicherung von Dateien.
Um Dateien in den Firestore Storage aus der Applikation heraus hochladen zu können, werden Objekte vom Typ `FirebaseStorage` & `StorageReference` benötigt.
Mithilfe dieser Objekte kann die Verbindung zu einem (Unter-)Ordner des Firestore Storages folgendermaßen hergestellt werden:

```
FirebaseStorage storage = FirebaseStorage.getInstance();
StorageReference storageReference =
storage.getInstance().getReference().child(“path”);
```

Quellcode 5.2: Initialisierung eines FirebaseStorage- und eines StorageReference-Objekts

Mithilfe eines `OnSuccessListener` kann ein Bild wie folgt hochgeladen werden:

```
storageReference.putFile(“path”)
        .addOnSuccessListener(new
OnSuccessListener<UploadTask.TaskSnapshot>() { // Code };
```

Quellcode 5.3: Nutzung der Methode `putFile`

In der Methode `onSuccess` wird eine Referenzierungs-URL erzeugt, sodass wir das jeweilige Bild später einer Veranstaltung oder einem Organisator zuordnen können:

```
ref.getDownloadUrl().addOnSuccessListener(new
OnSuccessListener<Uri>() {
    @Override
    public void onSuccess(Uri uri) {
        String rawFirestoreImagePath =
"https://firebasestorage.googleapis.com" + uri.getPath() + "?" +
uri.getEncodedQuery();
        firestoreImagePath = rawFirestoreImagePath.replace("path/",
"path%2F");
        setFirebaseData(); // setzt Attribut refPath
    }
});
```

Quellcode 5.4: Zusammensetzung der Referenzierungs-URL

Die nun erzeugte Referenzierungs-URL kann nun ganz einfach der aktuellen Veranstaltung oder dem aktuellen Organisator übergeben werden:

```
organizer.setImage(getFirestoreImagePath());
```

Quellcode 5.5: Übergabe der Referenzierungs-URL an Organizer-Objekt

5.2.5 Analytics

Für das Einbinden der Analytics mussten wir nichts machen, außer unsere App mit Firebase zu verbinden. Anschließend sammelt Firebase automatisch Daten, wie zum Beispiel die Anzahl der aktuellen Nutzer, deren Regionen und auch welche Versionen sie nutzen.

5.2.6 Cloud Messaging

Zum Senden von Push Notifications nutzen wir Firebase Cloud Messaging. Da Benutzer sich auf mehreren verschiedenen Endgeräten anmelden können, benötigen wir einen extra FCM-Token zu der User-ID. Dieses Token speichern wir, falls dieser sich ändert, in der Datenbank in dem User Dokument ab und sind so in der Lage, dem Endgerät eine Notification zu senden. Zum Senden einer Notification wartet eine Backend Funktion auf das Erstellen eines neuen Events in der Datenbank. Sollte dies der Fall sein, wird jedem Follower des Organizers über das FCM-Token eine Push Notification gesendet.

6 Test und Usability

6.1 Aufbau und Design

Beim Aussehen der Applikation haben wir uns wie in 5.1.1 bereits kurz erwähnt auf Googles Material Design gestützt. Die Intention dahinter ist, dass wir eine moderne Oberfläche erzeugen möchten, die jedoch einen eigenen Stil verfolgt. So nutzen wir durchgehend den Darkmode, da die Events meist am Abend oder in der Nacht stattfinden und wir uns daran orientieren [s. Abbildung 6.1]. Der Nutzer hat nicht die Möglichkeit, ein helleres Theme auszuwählen.
Außerdem legen wir großen Wert auf Bilder, da sie Events oder Veranstalter repräsentieren. Sie sollen großflächig und sichtbar dargestellt oder auch ausgeklappt werden können. Ist ein Bild für ein Event im Hochkantformat, so wird dies beim Aufrufen der Activity komplett angezeigt und dann zur Hälfte hochgescrollt, da dieses sonst zu viel Platz einnehmen würde. Das Scrollen soll demonstrieren, dass dies der Fall ist und die Möglichkeit besteht, es trotzdem komplett angucken zu können.
Um Events noch einzigartiger darzustellen und von anderen abzuheben, werden an mehreren Stellen Farbcodes dem Bild entnommen und in Views genutzt. Dies erkennt man bei den Einträgen in der Liste von Events. Dort ist der sog. Scrim, ein Fading-Effekt um Schriften und Icons lesbarer zu halten, eingefärbt. Ersichtlicher wird der Effekt bei der Activity für ein spezifisches Event. Dort werden die Statusleiste, der Scrim, die Actionbar, der Zusagen-Button und Links in TextViews eingefärbt. Wird etwas nicht eingefärbt, behält es den Standardakzentfarbton, ein leuchtendes Orange.
Die Applikation ist simpel aufgebaut. Man kann sich entweder per E-Mail und Passwort einloggen oder direkt durch Google. Wie man anhand von Abbildung 6.1 erkennen kann, wird man dann zum Dashboard weitergeleitet, welches die Events der Umgebung anzeigt. Dort kann man diese auch durchsuchen. Normalerweise ist man zu Beginn kein Veranstalter, also würde man den Floating-Action-Button unten rechts zum Erstellen von Events nicht sehen können. Möchte man Events erstellen können, kann man sich oben rechts im Menü unter einem Menüeintrag als Veranstalter bewerben. In unserer Implementation wird man sofort Veranstalter, bei Release würde man hier jedoch eine manuelle Bestätigung durchführen, weswegen auch eine private E-Mail-Adresse benötigt wird. Schaut man sich ein spezifisches Event an, kann man ebenfalls Informationen zum Veranstalter sehen. Diese kann man aufklappen und sind ebenfalls auf der Profilseite des Veranstalters sichtbar. So kann man Redundanz der Layout-Dateien verhindern. Auf dem Profil werden dazu noch weitere Informationen angezeigt wie z.B. eine Beschreibung und eine Liste seiner Events. Außerdem kann man den Veranstalter bewerten und Events auch teilen.

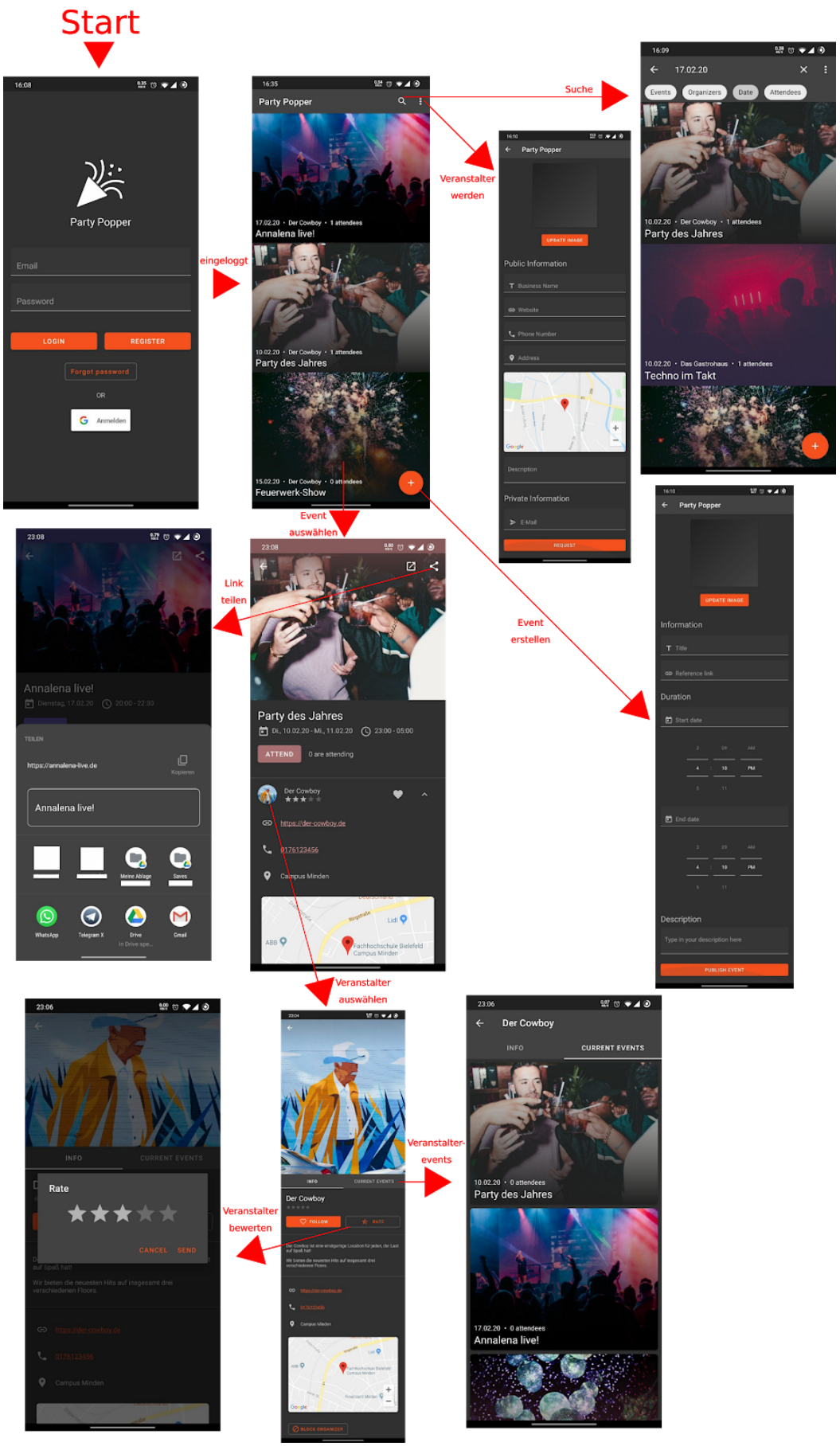


Abbildung 6.1: Übersicht der Activities und Aktionen.

6.3 Usability-Test

Um die Usability der Applikation zu testen, haben wir Probanden (n = 6) die App erkunden und anschließend ein Formular ausfüllen lassen. Die Personen bestanden aus Informatikern als auch aus Beschäftigten anderer Gebiete, damit wir ein buntes Ergebnis erhalten. Im Folgenden werden die Fragestellungen und die Anzahl an Antworten angegeben.

Wie ansprechend fanden Sie das generelle Design der Applikation?

		1	2	3

Nicht ansprechend — Sehr ansprechend

Wie empfanden Sie die Navigation (Mussten Sie lange nachdenken, wie Sie etwas ausführen)?

		1	1	4

Negativ — Positiv

Sind alle Beschriftungen / Icons ausschlaggebend?

			3	3

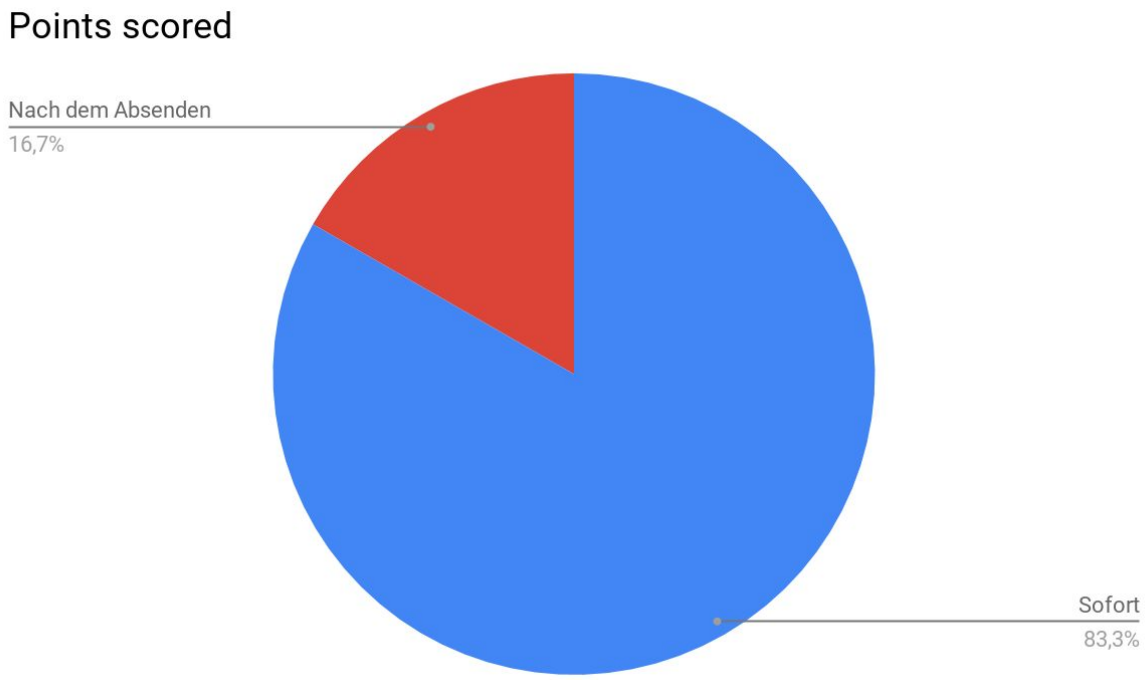
Nicht ausschlaggebend — Sehr ausschlaggebend

Wie hilfreich fanden Sie die Suche?

	1		3	2

Nicht hilfreich — Sehr hilfreich

Sollen Suchergebnisse sofort beim Tippen angezeigt werden oder erst nach dem Absenden?



Wie empfanden Sie es, ein Veranstalter zu werden?

		1	1	4

Kompliziert Einfach

Wie empfanden Sie es, ein Event zu erstellen?

			2	4

Kompliziert Einfach

Wie informativ ist die Ansicht eines Events gestaltet?

			4	2

Nicht informativ Sehr informativ

Wie informativ ist die Ansicht eines Veranstalters gestaltet?

			4	2

Nicht informativ Sehr informativ

6.3.1 Auswertung

Wenn man die Resultate anguckt, kann man ein weitaus positives Ergebnis erkennen. Es ist nicht perfekt und es weist darauf hin, dass man einige Sektoren der Applikation verbessern könnte, jedoch sollte man bedenken, dass bei solchen Tests nie ein perfektes Ergebnis herauskommt.
Interessant waren jedoch Ausreißer. Jemand empfand die Suche als nicht sehr hilfreich während die meisten das Gegenteil sahen. Die Mehrheit bevorzugt dazu eine Anzeige der Ergebnisse beim Tippen und nicht erst nach dem Absenden.
Würde die Menge an Probanden größer sein, könnte man Ausreißer weniger Beachtung schenken.

7 Zusammenfassung

Nach den Abschnitten der Ideenfindung & -herleitung, dem Recherchieren des aktuellen Standes der Technik, der Beschreibung unserer Anforderungen, der Beschreibung unserer Software-Architektur, der eigentlichen Implementierung unserer Applikation und des Testens dieser kommen wir in diesem Schlussteil zum Ende unserer Projektdokumentation und wollen einen kritischen Blick auf den Soll- und Ist-Zustand der Applikation werfen.

Unser ursprüngliches Ziel uns von anderen bereits publizierten mobilen Applikationen im Rahmen des Funktionsumfanges abzuheben, konnten wir leider nicht erreichen. Hauptgrund an dieser Stelle ist nicht fehlendes technisches Know-How, sondern viel eher der begrenzte Zeitraum, in der die Applikation entwickelt und fertiggestellt werden sollte.
Trotzdem ist es uns gelungen, alle unsere selbstgesteckten Ziele abzudecken.

- **Must have:**
  - Benutzerregistrierung & -anmeldung ✓
  - Organisator-/Unternehmensregistrierung & -anmeldung ✓
  - Hinzufügen von Veranstaltungen durch Organisatoren/Unternehmen ✓
  - Liste aller aktuellen Events, welche in unmittelbarer Nähe des jeweiligen Nutzers liegen ✓
- **Should have:**
  - Events zusagen ✓
  - Organisatoren/Unternehmen bewerten ✓
  - Organisatoren/Unternehmen folgen oder blockieren ✓
  - Veranstaltungen filtern ✓
  - Veranstaltungen suchen ✓
- **Could have:**
  - ~~anderen Nutzern folgen~~
  - ~~anzeigen von Veranstaltungen auf einer Karte~~
  - ~~Hinzufügen von privaten Veranstaltungen durch alle Nutzer~~
- **Won't have:**
  - ~~auf Nutzer zugeschnittene Ansichten (z.B. "das könnte dir auch gefallen")~~
  - ~~QR-Code Promotions~~
  - ~~gesponserte Inhalte~~

Die Nutzung von Google Firebase [7.1] erwies sich als wahrer Gewinn für unsere Applikation und konnte unseren Entwicklungsprozess weitaus effizienter gestalten, aufwändige Implementierungen und der damit verbundene zeitliche Aufwand von zum Beispiel Authentifizierungsverfahren oder der Datenbankanbindung konnten somit weitestgehend umgangen oder zumindest minimiert werden.

Schlussendlich lässt sich sagen, dass es uns innerhalb des gegebenen, begrenzten Zeitraumes trotzdem gut gelungen ist, eine optisch sowie technisch sehr ansprechende mobile Applikation zu entwickeln, welche mit etwas mehr zeitlichen Spielraum sicherlich einer Veröffentlichung im Google Play Store würdig ist.

Literaturverzeichnis

2

[2.1] Google Playstore, „Meetup“: https://play.google.com/store/apps/details?id=com.meetup (abgerufen am 07.01.2020)

[2.2] Google Playstore, „The Move“: https://play.google.com/store/apps/details?id=com.themove (abgerufen am 07.01.2020)

[2.3] Google Playstore, „Party Hunt: Goa“: https://play.google.com/store/apps/details?id=com.samsara.partyhunt (abgerufen am 08.01.2020)

[2.4] Google Playstore, „Eventbrite“: https://play.google.com/store/apps/details?id=com.eventbrite.attendee (abgerufen am 08.01.2020)

[2.5] Google Playstore, „Ticketmaster“: https://play.google.com/store/apps/details?id=com.ticketmaster.tickets.international (abgerufen am 08.01.2020)

4

[4.1] Google Firebase, „Firebase Security Rules“: https://firebase.google.com/docs/rules (abgerufen am 08.01.2020)

[4.2] Google Firebase, „Firebase Authentication“: https://firebase.google.com/docs/auth (abgerufen am 08.01.2020)

[4.3] Google Firebase, „Cloud Firestore“: https://firebase.google.com/docs/firestore (abgerufen am 09.01.2020)

[4.4] Google Maps, „Google Maps Platform“: https://cloud.google.com/maps-platform (abgerufen am 10.01.2020)

5

[5.1] SoftSelect, „Definition und Erklärung Frontend“: http://www.softselect.de/business-software-glossar/frontend (abgerufen am 08.01.2020)

[5.2] Google, „Material Design“: https://material.io/ (abgerufen am 08.01.2020)

[5.3] Google, „Material Components for Android“: https://material.io/develop/android/docs/getting-started/ (abgerufen am 08.01.2020)

[5.4] faruk und Aakash Anuj, „What is model, view and controller in Android RecyclerView?“: https://stackoverflow.com/questions/37662322/what-is-model-view-and-controller-in-android-recyclerview (abgerufen am 08.01.2020)

[5.5] Firebase, „FirebaseUI for Android — UI Bindings for Firebase“: https://github.com/firebase/FirebaseUI-Android (abgerufen am 09.01.2020)

[5.6] Android Developers, „Fragments“: https://developer.android.com/guide/components/fragments (abgerufen am 09.01.2020)

[5.7] Google Developers, „MapFragment | APIs for Android“: https://developers.google.com/android/reference/com/google/android/gms/maps/MapFragment (abgerufen am 09.01.2020)

[5.8] Google Firebase, „Authenticate Using Google Sign-In on Android“: https://firebase.google.com/docs/auth/android/google-signin (abgerufen am 09.01.2020)

[5.9] Google Firebase, „Create Custom Tokens“: https://firebase.google.com/docs/auth/admin/create-custom-tokens (abgerufen am 09.01.2020)

7

[7.1] Google, „Google Firebase”: https://firebase.google.com/ (abgerufen am 08.01.2020)

Abbildungsverzeichnis

Quellcodeverzeichnis