

# **Mobile Application Development**

**Eventure Dokumentation**

Alexander Rasputin, Maximilian Karthan, Jona Ruof

13. Oktober 2017

# **Inhaltsverzeichnis**

<b>1 Einleitung</b>	<b>3</b>
1.1 Motivation . . . . .	3
1.2 Ziel . . . . .	3
1.3 Aufbau . . . . .	4
<b>2 Anforderungsanalyse</b>	<b>5</b>
2.1 Funktionale Anforderungen . . . . .	5
2.2 Nicht Funktionale Anforderungen . . . . .	6
<b>3 Architektur</b>	<b>8</b>
3.1 Libraries und Frameworks . . . . .	8
3.2 Klassenstruktur . . . . .	9
<b>4 Konzept und Entwurf</b>	<b>11</b>
4.1 Mockups . . . . .	11
<b>5 Implementierung</b>	<b>13</b>
5.1 Übersicht . . . . .	13
5.2 Kartenansicht . . . . .	16
5.3 Event Erstellung . . . . .	18
5.4 Server . . . . .	20
5.4.1 Verwendete Technologien . . . . .	20
5.4.2 Benutzerverwaltung . . . . .	20
5.4.3 Events . . . . .	20
<b>6 Anforderungsabgleich</b>	<b>21</b>
6.1 Funktionale Anforderungen . . . . .	21
6.2 Nicht Funktionale Anforderungen . . . . .	23
<b>7 Zusammenfassung und Ausblick</b>	<b>23</b>

# 1 Einleitung

## 1.1 Motivation

Um die Kenntnisse im Bereich der mobile Anwendungsentwicklung zu vertiefen, sollte im zweiten Teil des Mobile Application Projects eine eigene Applikation entwickelt werden. Die Anwendung sollte dabei eine Problemstellung aus einem frei wählbaren Themengebiet lösen. Die hier entwickelte *Eventure* App vereint viele verschiedenen Aspekte der mobile Anwendungsentwicklung und stellt somit eine ausreichend komplexe Aufgabe dar.

Prägend für das Design von *Eventure* ist der Begriff des *Events*. Als Event wird hier jede Art des lokalen, zeitlich begrenzten, sozialen Ereignisses verstanden. So werden z.B. eine Theatervorstellung, ein Volleyballspiel im Park oder eine Geburtstagsfeier als Events bezeichnet. Events bieten Möglichkeiten zur sozialen Interaktion mit Freunden und Fremden und sind damit relevant für das Leben aller Menschen. Gerade für ortsfremde Personen ist es oftmal kompliziert an die relevanten Eckdaten (Zeit, Ort, etc. ...) eines Events zu kommen. Da keine zentralisierte Sammelstelle für solche Information existiert, sind diese meist in verschiedensten Medien zu finden oder werden nur von Person zu Person weitergegeben. Daher scheint die Notwendigkeit einer Applikation für das Entdecken von Events gegeben.

Es existieren bereits einige solcher Apps, welche allerdings (subjektiv betrachtet) nicht einfach und intuitiv genug bedient werden können. So sind insbesondere die Abläufe für das Finden und Erstellen von Events zu kompliziert gestaltet. Zahlreiche Einstellungs- und Suchoptionen verschlechtern die Bedienbarkeit zusätzlich.

## 1.2 Ziel

Das Ziel des Mobile Application Developments ist die Entwicklung einer Android Applikation die das Entdecken und Erstellen von Events ermöglicht. Dabei sollte die App besonders einfach zu bedienen sein. Die Probleme der bereits bekannten Event Apps sollten gelöst oder zumindest entscheidend verbessert werden. Außerdem sollen die Kenntnisse in der Android Programmierung weiter vertieft werden, so dass am Ende des Projekts alle grundlegenden und einige weiterführende Konzepte der mobilen Anwendungsentwicklung bekannt sind.

### **1.3 Aufbau**

In der folgenden Dokumentation werden in der Anforderungsanalyse zunächst die funktionalen und nicht funktionalen Anforderungen der Applikation präsentiert. Ebenfalls werden verwendeten Frameworks und die Architektur des Gesamtsystems beschrieben. Danach wird das Konzept und der Entwurf der App anhand von Mockups erläutert. Im nächsten Abschnitt werden Details der Implementierung und Besonderheiten der App präsentiert. Zuletzt werden die gestellten Anforderungen mit der tatsächlichen Implementierung der App verglichen es wird festgestellt welche Anforderungen bis zu welchem Grad erfüllt wurden.

## **2 Anforderungsanalyse**

### **2.1 Funktionale Anforderungen**

#### **FA1 Accountmanagement**

Zur Verwendung der App wird ein Benutzeraccount benötigt. Dieser kann in der App selbst erstellt werden. Wenn möglich, sollte auch eine OAuth Authentifizierungs implementiert werden, die es Benutzer ermöglicht vorhandene Accounts von Google, Facebook etc. zur Anmeldung zu verwenden. Für einen Account werden als Informationen mindestens die E-Mail Adresse des Benutzers ein Passwort und ein Name benötigt. Optional sollte auch ein persönliches Benutzerbild eingefügt werden können. Es sollte außerdem möglich sein Accountdetails wie den Namen oder das Passwort zu bearbeiten oder den Account vollständig zu löschen.

#### **FA2 Eventmanagement**

Der Benutzer muss die Möglichkeit haben selbst eigene Events in der entwickelten App zu erstellen. Dabei besteht ein Event aus einem Bild, einem Titel, einem Ort, einer Startzeit, einer maximalen Anzahl an Teilnehmern, einer Kategorie und einem optionalen Kommentar. All diese Details sollten auch nach dem Erstellen eines Events jederzeit wieder geändert werden können. Außerdem muss der Benutzer ein selbst erstelltes Event jederzeit löschen können. Ein Benutzer kann einem Event beitreten, insofern die maximale Anzahl an Event-Teilnehmern noch nicht erreicht wurde. Zu jedem Zeitpunkt kann eine Benutzer, der nicht der Ersteller des Events ist, ein Event verlassen. Der Ersteller eines Events kann diese nur löschen, nicht aber verlassen.

#### **FA3 Eventübersicht**

Um die verschiedenen Events dem Benutzer zu präsentieren muss die App eine Übersicht über die verschiedenen Events bieten. Dabei sollte vor allem zwischen Events an welchen der Benutzer beteiligt und an welchen der Benutzer nicht beteiligt ist unterschieden werden.

#### **FA4 Filter**

Die in der Eventübersicht angezeigten Events müssen nach verschiedenen Kriterien gefiltert werden können. So kann der Benutzer z.B. einen Suchradius für Events konfigurieren. Alle Events, deren Entfernung zum Standort des Benutzer kleiner als dieser

Radius ist, sollen nicht gefiltert werden. Alle Events die außerhalb des Radius liegen werden gefiltert, d.h. nicht angezeigt. Außerdem ist es möglich die Events anhand ihrer Kategorien zu filtern.

#### **FA5 Gruppenchat**

Ist der Benutzer einem Event beigetreten hat dieser die Möglichkeit sich mit anderen Teilnehmern des Events in einem Gruppenchat auszutauschen. Dabei wird für jedes Event eine eigener Chat erstellt zu dem nur die Teilnehmer eines Events zugelassen sind.

#### **FA6 Chatübersicht**

Der Benutzer hat die Möglichkeit durch eine Übersicht alle Chats anzeigen zu lassen an welchen dieser teilnimmt. Von dieser Übersicht aus kann ein beliebiger Chat betreten werden.

#### **FA7 Kartenansicht**

Der Benutzer kann sich alle momentan verfügbaren Events in einer Kartenansicht anzeigen lassen. Dabei wird eine Kartenstapel simuliert, wobei immer nur die oberste Karte für den Benutzer sichtbar ist. Durch eine Wischgeste wird die oberste Karte des Stapels entfernt und die nächste Karte sichtbar. So kann der Benutzer nacheinander durch die verfügbaren Events blättern.

## **2.2 Nicht Funktionale Anforderungen**

#### **NFA1 Robustheit**

Die App soll eine gewisse Robustheit aufweisen, also auf fehlerhafte Eingaben oder unvorhergesehene Ereignisse angemessen reagieren. Im Falle eines Absturzes sollte die Applikation ohne Verlust des Spielfortschritts neu gestartet werden können.

#### **NFA2 Erweiterbarkeit**

Die Programmstruktur der Applikation sollte derart gestaltet sein, dass spätere Erweiterungen möglichst einfach vorgenommen werden können.

#### **NFA3 Responsiveness**

Die Oberfläche sollte stets innerhalb einer sehr kurzen Zeit auf Benutzereingaben reagieren. Bei längeren Wartezeiten, etwa während eines Downloads, sollte der Benutzer

permanent, durch den Einsatz von entsprechenden GUI-Elementen, über den Fortschritt der Operation informiert werden.

#### **NFA4 Usability**

Der Benutzer sollte die App, nach einer kurzen Einführung, durch eine intuitive und benutzerfreundliche Oberfläche ohne weitere Anleitung bedienen können.

## 3 Architektur

### 3.1 Libraries und Frameworks

Um einige Funktionen der Applikation nicht selbst implementieren zu müssen wurden folgende zusätzliche Programmzbibliotheken und Frameworks verwendet:

#### **Volley, <https://github.com/google/volley>**

Volley ist eine Netzwerkbibliothek welche die Kommunikation mit Webservices erheblich erleichtert. Alle requests und responses werden asynchron behandelt und als abstrakte Objekte repräsentiert. Diverse Datenformate (wie JSON) werden direkt in einer Objektrepräsentation ausgegeben und sind somit im Programm ohne weitere Umwandlung zugänglich. Auch das direkte Laden von Bildern in Widgets (z.B. ImageView) und das dazugehörige Caching wird von Volley übernommen. Volley wurde verwendet um die Kommunikation mit dem Event-Server der Eventure App zu implementieren.

#### **Gson, <https://github.com/google/gson>**

Gson ist eine Bibliothek, welche die Serialisierung und Deserialisierung von JSON Objekten in Java Objekte implementiert. Dabei können JSON Objekte auch direkt in Instanzen einer Java Klasse umgewandelt werden, was einen besonders komfortablen Umgang mit JSON ermöglicht. Gson wurde verwendet um den Austausch von JSON-Nachrichten zwischen Server und App zu erleichtern.

#### **CircleImageView, <https://github.com/hdodenhof/CircleImageView>**

Diese Bibliothek beinhaltet ein modifiziertes ImageView Widget, welches statt einem Rechteck einen Kreis als Grundfläche besitzt. Mit Hilfe von CircleImageView wurde die Anzeige des Profilbilds in der Chatübersicht realisiert.

#### **PlaceHolderView, <https://github.com/janishar/PlaceHolderView>**

Die PlaceHolderView Bibliothek basiert auf der RecyclerView des Android Frameworks und erweitert diese um Animationen und zusätzliche Funktionen. Mit Hilfe dieser Library wurde die Kartenansicht der Events erstellt.

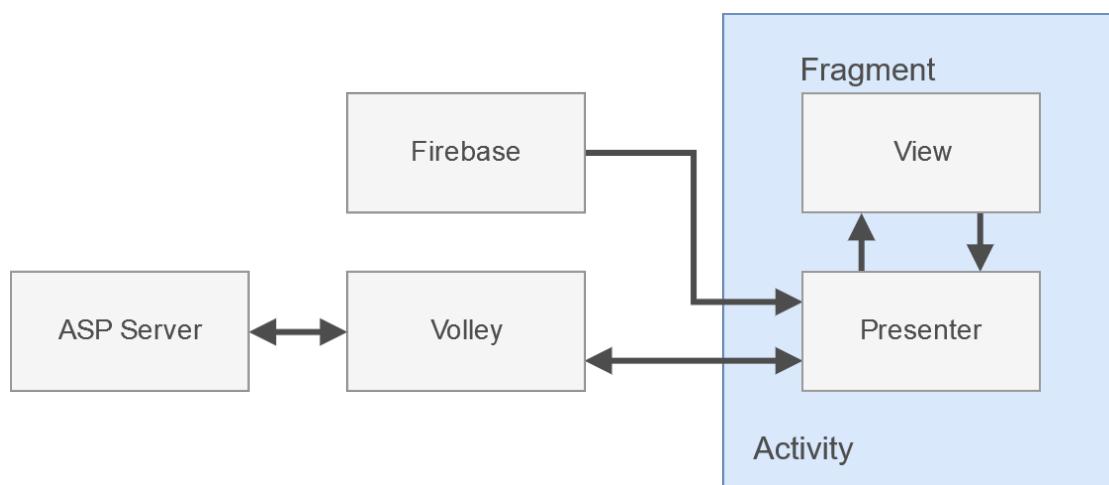
#### **Google Location API, <https://developer.android.com/training/location/index.html>**

Dieses API ermöglicht die Lokalisierung des Android Gerätes über die Verwendung des eingebauten GPS, der momentanen Funkzelle oder der verfügbaren WLAN Netzwerke. Das Location API wurde verwendet um den Standort des Benutzer für die Filterung der Events zu bestimmen.

## Firebase, <https://firebase.google.com>

Firebase ist eine online Datenbank auf JSON Basis. Die Datenbank kann durch einfache Queries manipuliert werden und Clients können sich über Änderungen in der Datenbank in Echtzeit informieren lassen. Firebase wurde verwendet um den Gruppenchat der einzelnen Events zu realisieren.

### 3.2 Klassenstruktur



Um die Applikation zu strukturieren wurde hier das Model-View-Presenter Pattern angewendet. Mit diesem Pattern wird jede Activity in Model-, View- und Presenter- Komponenten zerlegt. Die Model Komponenten beinhalten die Zugriffsschicht auf den Event Server mit allen zugehörigen Klassen. In der hier vorgestellten App wird das Model durch eine „Network“ Singleton Klasse, welche mit Volley realisiert wurde, und durch entsprechende Entitätsklassen dargestellt. Ebenfalls werden alle Firebase-Komponenten, die für die Realisierung des Chat zuständig sind, zum Model gezählt. Man beachte, dass hier keine klassische Datenbank als Model verwendet wurde. Stattdessen sind alle Ressourcen nur online zugänglich. Da die hier benötigten Daten design-bedingt sehr kurzlebig sind, hätte eine lokale Datenbank auf dem Gerät die App lediglich verkompliziert. Somit stellen der Event- und Firebase-Server das eigentliche Model dar.

Auf das Model kann nur vom Presenter aus zugegriffen werden. Dieser verwaltet die eigentlich Logik einer Activity. Dabei ist der Presenter selbst eine einfache Java Klasse ohne direktes Wissen über das Android System oder die Datenbank. Events der View werden an den Presenter weitergeleitet und dort bearbeitet. Der Presenter kann zur

Bearbeitung von Events sowohl auf das Model als auch auf die View Komponenten zugreifen. Diese Komponenten beinhaltet die Anzeigeschicht und verwaltet die einzelnen GUI-Elemente. In unserer App wird die View von einer Fragment Klasse implementiert. Direkt kommuniziert die View nur mit dem Presenter und greift nicht auf das Model zu. In der Praxis gibt es dabei allerdings einige Ausnahmen.

Um z.B. das Anzeigen von Listen in Android effizient zu realisieren werden sogenannte Adapter verwendet. Diese greifen meist direkt auf die Datenbank zu und stellen die erhaltenen Daten direkt in einer ListView da. Damit können diese Klassen nicht eindeutig den Model oder View Komponenten zugeordnet werden. Daher implementieren diese meist sowohl die Eigenschaften des Models als auch der View.

Durch die klare und immer gleichförmige Strukturierung in Model, View und Presenter, wird das Erstellen von neuen Activities vereinfacht, da somit ein klarer Arbeitsablauf gegeben ist. Auch das lesen und editieren von fremden Code wird erleichtert, da von vornherein anhand der Benennung der einzelnen Klassen klar ist welche Aufgaben diese Komponenten übernehmen.

## 4 Konzept und Entwurf

### 4.1 Mockups

Vor dem eigentlichen Implementieren der App wurden Mockups erstellt. Diese sollten einen ersten Eindruck vom User Interface geben und dienten als Orientierung während der eigentlichen Implementierung. Wobei sich nach einigen Versuchen manche Konzepte aus den Mockups als weniger geeignet herausstellten und entsprechend verändert wurden.

Als Einstiegspunkt in die App wollten wir eine TabView implementieren, die auf der ersten Seite die Events in der Nähe anzeigt. Abbildung 1 (links) zeigt das Konzept dieser View.

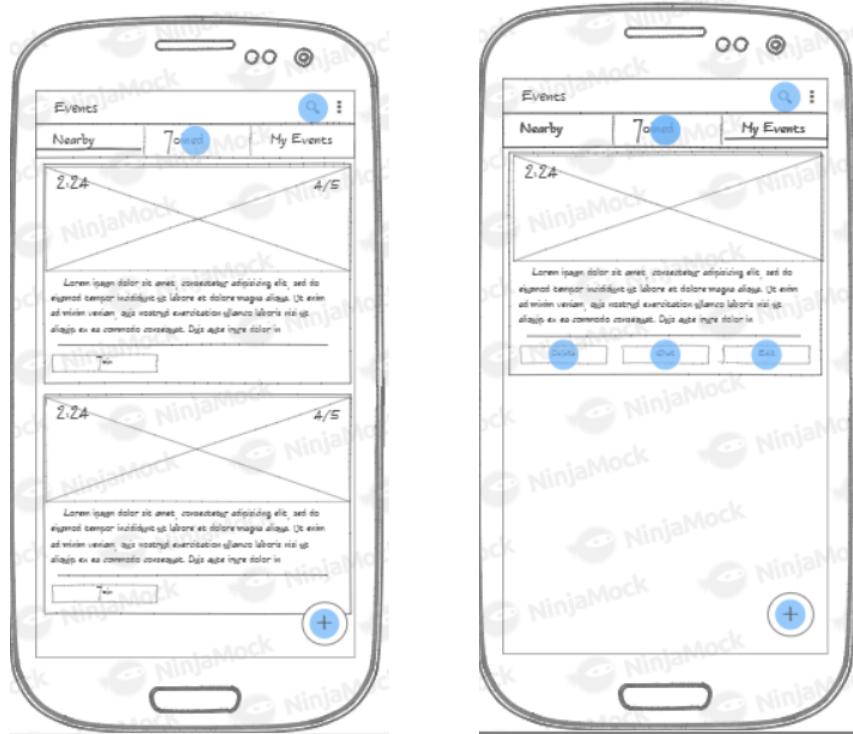


Abbildung 1: Konzept der Übersicht(links: Events in der Umgebung, rechts: selbst erstellte Events)

Abbildung 1 (rechts) skizziert die Übersichtsseite der vom Benutzer selbst erstellten Events. Sie sollte ähnlich aufgebaut sein, allerdings mit entsprechend anderen Funktionalitäten wie der Möglichkeit ein Event zu löschen.

Das Konzept der TabView wurde in der tatsächlichen Anwendung schließlich doch nicht mehr genutzt. Stattdessen kam die „Bottom navigation“, eine Komponente aus Googles Material Design Guidelines, zum Einsatz. Nach einigen Tests kamen wir zu dem Schluss, dass diese sich besser „anfühlt“.

Bei einem Klick auf ein eigenes Event oder einem, dem der Benutzer bereits beigegetreten ist soll dieser zu einem Chat gelangen. Von dort aus ist auch eine Ansicht aller Teilnehmer des Events zugänglich diese beiden Views zeigt Abbildung 2.

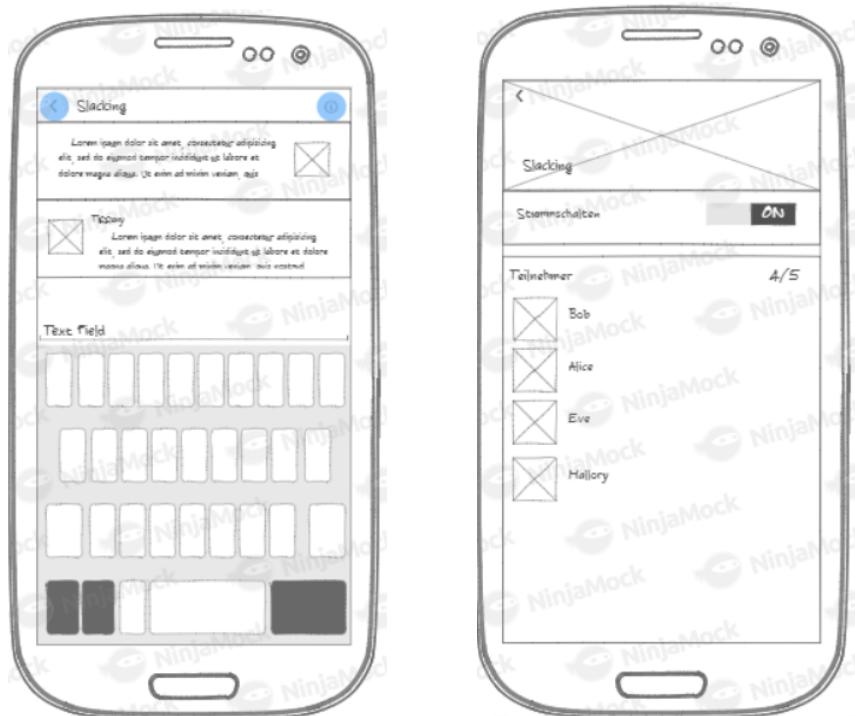


Abbildung 2: Konzept der Chatview und der Teilnehmerliste

Außerdem haben wir Mockups erstellt die eine View zeigen um Events zu erstellen diese View zeigt Abbildung 3

Wir haben darauf verzichtet Mockups für Activities zu machen die zum Beispiel eine Karte für die Navigation zeigen oder nur die Auswahl einer Kategorie und ein Suchfeld beinhalten. Im Laufe der Entwicklung sind weitere Activities hinzugekommen. Zum Beispiel die "TinderAnsicht. Diese Views werden im nächsten Abschnitt behandelt.



Abbildung 3: Konzept der Eventerstellung

## 5 Implementierung

### 5.1 Übersicht

Im Startbildschirm erhält der Benutzer eine Übersicht über alle momentan verfügbaren Events und Chats. In der ActionBar dieser Activity können die Sucheinstellungen für Events angepasst, neue Events hinzugefügt, und das Benutzerprofil aufgerufen werden. Durch eine Leiste am unteren Bildschirmrand kann zwischen in der Nähe befindlichen („Nearby“), beigetretenen Events („My Events“) und den zugehörigen Chats („Chat“) ausgewählt werden. Dabei werden die Events als eine Liste aus Karten dargestellt die der Benutzer per Wischgeste durchsuchen kann. Eine Karte zeigt dabei jeweils das Bild des Events, den Titel, die Beschreibung, den Startzeitpunkt und die Anzahl der noch freien Plätze. Außerdem werden im unteren Teil der Karte verschiedene Aktionen als Buttons angezeigt.

In der „Nearby“ Liste kann der Benutzer lediglich Events beitreten. Wird einem Event beigetreten wird eine entsprechende Anfrage an den Event-Server gesendet. Meldet der

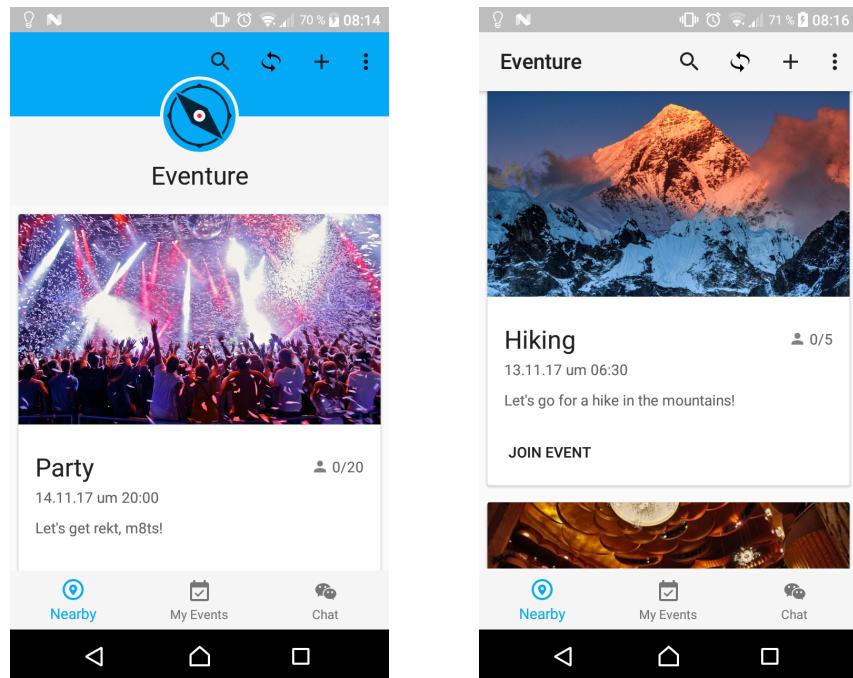


Abbildung 4: „Nearby“ Tab der Übersicht

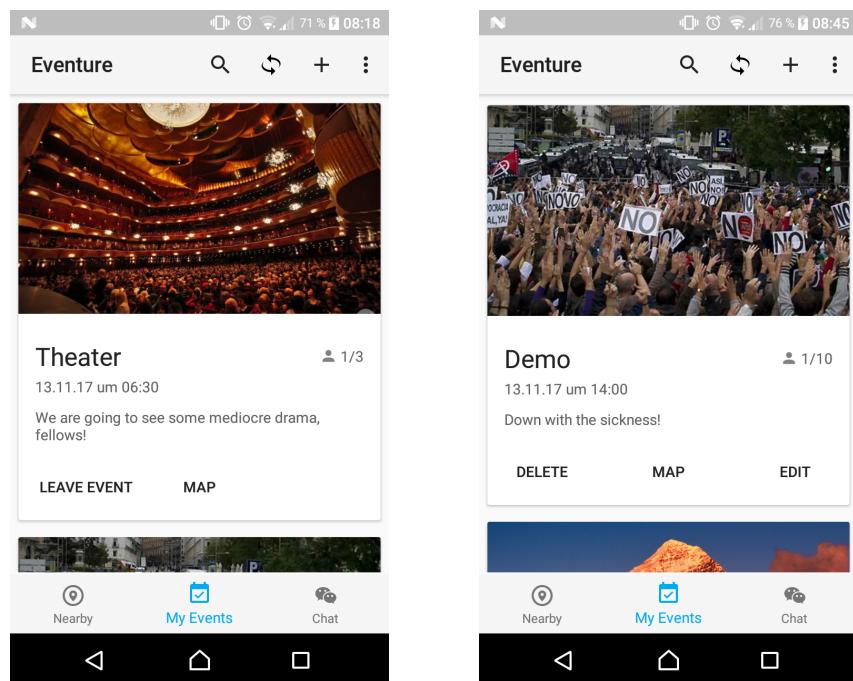


Abbildung 5: „My Events“ Tab der Übersicht

Server einen erfolgreichen Beitritt wird daraufhin die „Nearby“-Liste aktualisiert, und das Event in die „My Events“ Liste eingefügt. Events in dieser Liste stellen dem Benutzer entweder die Aktionen „Map“ und „Leave“ oder „Delete“, „Map“ und „Edit“ bereit. Dabei werden Letztere nur für diejenigen Events angezeigt, welche der Benutzer selbst erstellt hat. Ist der Benutzer einem Event begetreten, hat dieses aber nicht selbst erstellt so wird statt „Delete“ der Button „Leave“ angezeigt. Tippt der Benutzer auf den „Delete“ Button wird eine Anfrage an den Event-Server und an die Firebase-Datenbank gesendet, die auf dem Event-Server das Event und in Firebase den zugehörigen Chat aus der Datenbank löscht. Wird der „Leave“ Button betätigt wird nur der Benutzer aus dem Event entfernt. Das Event selbst bleibt aber bestehen und ist wieder in der „Nearby“-Liste auffindbar. Ein Event hat somit immer mindestens den erstellenden Benutzer als Teilnehmer. Der außerdem verfügbare „Map“ Button öffnet über einen speziell formatierten *Intent* die Google Maps App und zeigt den Ort des Events an. So können Benutzer schnell zum Ort des Events navigieren. Über den „Edit“ Button kann der erstellende Benutzer nachträglich die Details eines Events bearbeiten.

Alle diese Events werden asynchron direkt vom Event-Server geladen und nicht zwischengespeichert. Somit wird garantiert, dass immer nur aktuelle Events angezeigt werden. Als Übertragungsformat wird hier JSON verwendet. Ein JSON-Objekt repräsentiert dabei ein komplettes Event, wobei das zugehörige Bild über eine separate URL vom Server geladen wird. Auch die Liste an verfügbaren Chats wird asynchron geladen und nicht zwischengespeichert. Die Daten werden dabei allerdings von Firebase geladen, wobei das bereitgestellte Firebase-API verwendet wurde. Dieses liefert die angeforderten Objekte als *DataSnapshot* Instanzen zurück, die mittels Reflection in Objekte einer passenden Java Klasse umgewandelt werden können. Außerdem wird die App über die Firebase-API über Änderungen an der Chat Liste benachrichtigt. Anhand dieser Änderungen wird die dargestellte Liste aktualisiert, womit diese immer aktuell gehalten wird.

Die verschiedenen Listen des Startbildschirms werden über drei verschiedene Fragments realisiert. Dabei werden alle drei Fragments beim Erstellen der zugehörigen Activity instanziert und während des Bestehens der Activity permanent im Speicher gehalten. Die Activity selbst beinhaltet neben der unteren Navigationsleiste lediglich ein Frame Layout, in welches die Fragments platzierte werden können. Wird eine Liste ausgewählt wird einfach das zugehörige Fragment in das Frame Layout geladen.

## 5.2 Kartenansicht

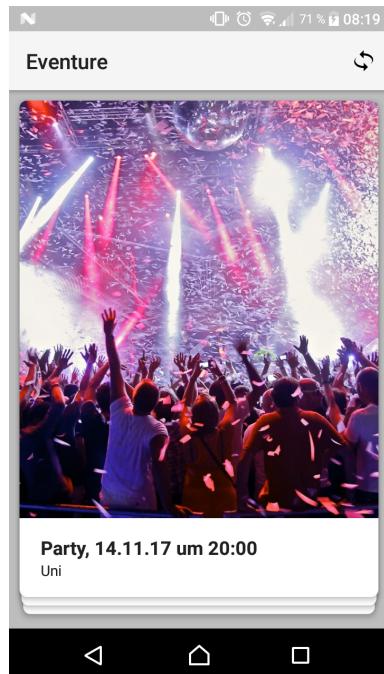


Abbildung 6: Kartenansicht der verfügbaren Events

In dieser Ansicht werden alle Events angezeigt die auch in der „Nearby“ Liste in der Übersicht angezeigt werden. Dabei werden die Events hier allerdings nicht als eine kontinuierliche Liste präsentiert sondern als ein simulierter Kartenstapel. Jede Karte beinhaltet nur minimale Information und bietet hauptsächlich Platz für das Bild des Events. So soll eine spontane Entscheidung über das Gefallen oder Nicht-Gefallen des Events unterstützt werden.

Wird die Karten nach rechts geschoben tritt der Benutzer dem Event bei. Wird die Karte dagegen nach links geschoben tritt der Benutzer dem Event nicht bei und die Karte wird wieder in den virtuellen Kartenstapel eingeordnet. Dieser Stapel wird hier durch die *PlaceHolderView* realisiert. Die anzugeigenden Events werden dabei ebenfalls direkt vom Event-Server und nicht aus einer lokalen Datenbank geladen.

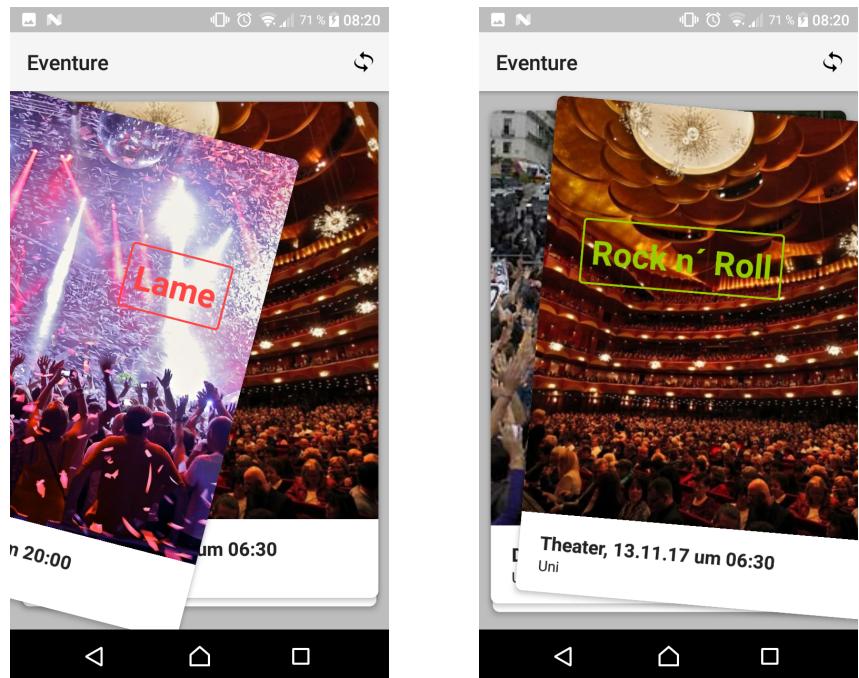


Abbildung 7: Event wird abgelehnt oder dem Event wird beigetreten

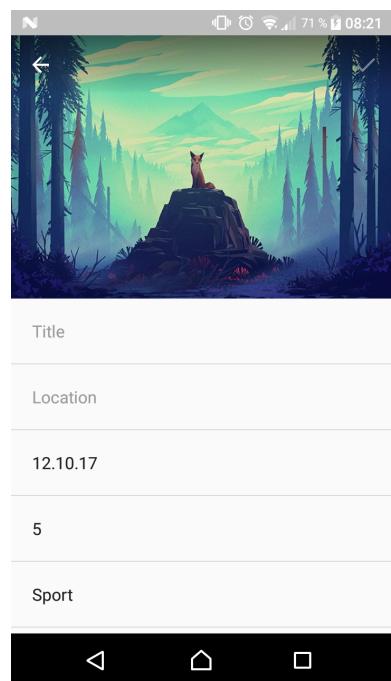


Abbildung 8: View zur Event Erstellung

### 5.3 Event Erstellung

In dieser Activity kann ein Benutzer ein Event anlegen oder ein bereits bestehendes Event bearbeiten. Die dafür nötige Eingabemaske wurde dabei so einfach und klar wie möglich strukturiert. Den Großteil der vorhandenen Fläche nimmt eine ImageView ein, in welcher das zugehörige Bild des Events angezeigt wird. Um die Fläche für dieses Bild zusätzlich zu maximieren wurde hier die ActionBar der Activity transparent dargestellt. Damit der weiße „Zurück“-Button der Activity immer noch sichtbar bleibt wird die ActionBar aber nicht komplett transparent sondern mit einem leichten Farbverlauf gezeichnet. Dieser Farbverlauf ist über dem Bild der ImageView kaum sichtbar, aber erhöht dennoch merklich die Sichtbarkeit des „Zurück“-Buttons. Klick der Benutzer auf das Bild kann er ein beliebiges auf dem Gerät vorhandenes Bild auswählen, das dann in der ImageView angezeigt wird. Dabei wird das Bild asynchron im Hintergrund, dass heißt abseits des Haupt-Threads geladen.

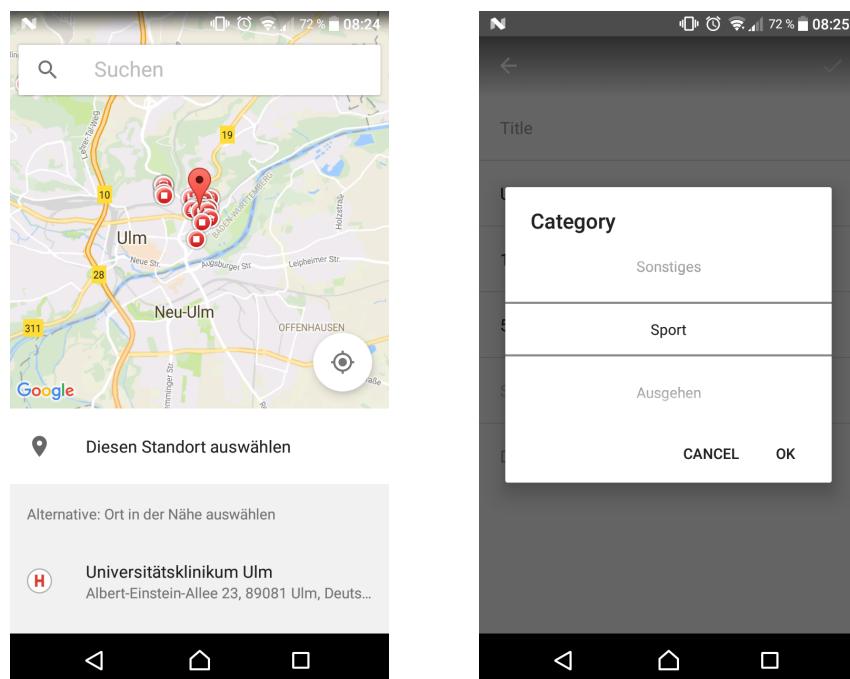


Abbildung 9: Picker zur Orts- und Kategorieauswahl

In den dann folgenden Textfeldern können die weiteren Parameter des Events konfiguriert werden. Dabei wurde das Design der Textfelder so angepasst, dass diese die gesamte Bildschirmbreite verwenden. Damit wurde auch auch die klickbare Fläche

der Textfelder vergrößert, sodass das Auswählen eines einzelnen Textfelds einfacher möglich ist. Obwohl als Anzeige Textfelder verwendet werden sind diese nur bedingt für die Auswahl einiger Informationen geeignet. Klickt der Benutzer z.B. auf das „Location“ Textfeld wird über das Google Places API eine Place Picker Activity gestartet die eine benutzerfreundliche Auswahl eines Ortes ermöglicht. Der ausgewählte Ort wird intern gespeichert und eine textuelle Repräsentation des Ortes im zugehörigen Textfeld angezeigt. Um die Startzeit auszuwählen wurde ein Time-Picker-Dialog erstellt, der geöffnet wird, wenn der Benutzer auf das „Start“ Textfeld tippt. So kann eine Startzeit benutzerfreundlich eingegeben werden. Auch werden so Formatierungsfehler vermieden, die bei der Eingabe von Daten in ein Textfeld auftreten können. Ebenfalls wurden Picker-Dialoge für die Felder „Max participants“ und „Category“ erstellt.

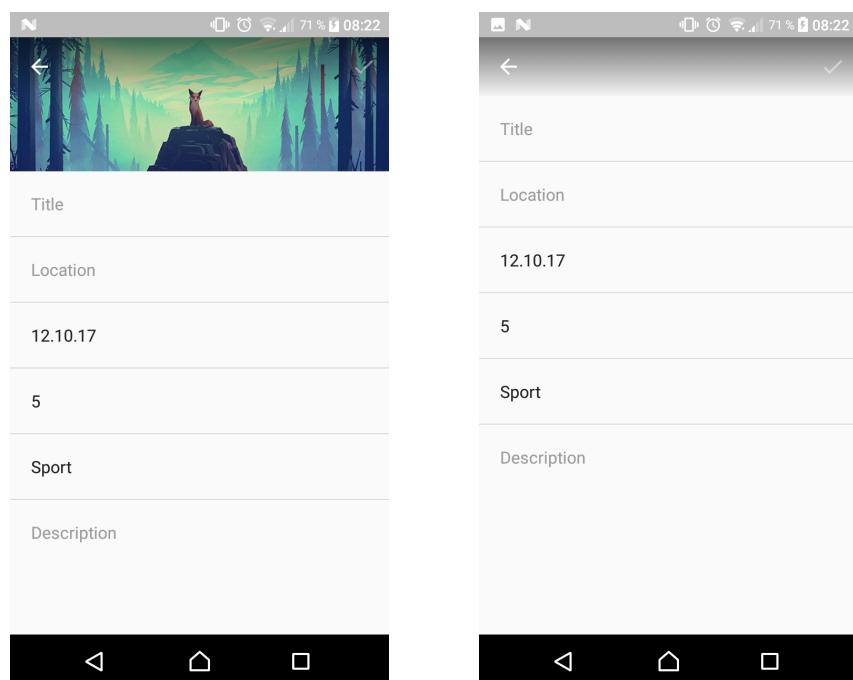


Abbildung 10: Scroll-Verhalten mit *NestedScrollView* und *AppBarLayout*

Aufgrund der relative großen Darstellung des Bilds können nicht alle benötigten Felder auf einer Bildschirmlänge dargestellt werden. Daher musste die Parameterliste scrollbar implementiert werden. Um von jeder Scroll-Position die Activity wieder verlassen zu können muss der „Zurück“-Button, dass heißt die ActionBar der Activity, immer zugänglich bleiben. Allerdings wurde diese transparent gezeichnet und ist somit auf den Textfeldern schlecht lesbar. Daher wurde hier eine *NestedScrollView* in Kombination

mit einem *AppBarLayout* verwendet. Wird auf der Parameterliste gescrollt schiebt sich diese über das Bild des Events und der Hintergrund der AppBar wird so transformiert, dass dieser nun nicht mehr transparent ist. Somit kann nun der „Zurück“-Button immer angezeigt und die Parameterliste gescrollt werden.

## 5.4 Server

### 5.4.1 Verwendete Technologien

Als Server wurde ein Windows Server benutzt mit dem Aktuellen Windows Server 2016 und einem IIS 10.0 (Internet Information Services, Microsofts Web server für Windows Server). Programmiert wurde die Serverlogik in C# mit ASP.NET Core. Als Datenbank kam SQL Server 2016, ebenfalls von Microsoft, zum Einsatz. Der Zugriff auf die Datenbank wurde mit Microsofts EntityFramework realisiert.

Wir haben uns dazu entschieden auf eine „Thin Client“ Lösung zuzetzen. Die gesamte Datenverarbeitung findet folglich auf dem Server statt. Dies erschien uns als die Sinnvollere Möglichkeit da unsere Zielgeräte eben nur beschränkt Leistung zur Verfügung haben.

### 5.4.2 Benutzerverwaltung

Zur Verifizierung der Benutzer kommen JWT (JSON Web Token) zum Einsatz. Da bei diesem Standard sowohl Passwort als auch Username der Benutzer im Klartext übertragen werden findet die komplette Kommunikation zwischen Client und Server natürlich ausschließlich per SSL statt.

### 5.4.3 Events

Die REST-Schnittstelle bietet die Möglichkeit sich alle Events in einer Liste zu holen, dabei können diese nach Kategorie, Koordinaten, Umkreis, Start- und End-Datum gefiltert werden. Außerdem können die Bilder zu den Events einzeln geladen werden um „Lazy loading“ zu unterstützen und eine bessere Parallelisierung beim Laden der Bilder zu ermöglichen. Genau so können natürlich auch einzelne Events, Events denen der

Aktuelle Benutzer beigetreten ist und seine selbst erstellten Events separat angefragt werden. Kurz alle typischen CRUD - Operationen sind für die Events implementiert.

Außerdem läuft auf dem Server ein Skript, dass Events automatisch entfernt sobald diese länger als 24 Stunden in der Vergangenheit liegen. Wobei Events in der „Nearby“ Übersicht nicht mehr angezeigt werden wenn diese bereits beendet sind.

## 6 Anforderungsabgleich

Im Folgenden werden die gegebenen Anforderungen mit der tatsächlichen Implementierung verglichen. Es wird festgestellt ob eine Anforderung ganz, teilweise oder gar nicht erfüllt wurde.

### 6.1 Funktionale Anforderungen

#### **FA1 Accountmanagement - teilweise erfüllt**

Die Möglichkeit sich mit Google und Facebook anzumelden wurde Serverseitig vorbereitet jedoch fehlt noch die Implementierung im Client. Die restlichen Anforderungen an das Accountmanagement wurden erfüllt.

#### **FA2 Eventmanagement - erfüllt**

Das Eventmanagement wurde wie in der Anforderungsanalyse definiert implementiert und erfüllt alle Bedingungen.

#### **FA3 Eventübersicht - erfüllt**

Auch diese wurde wie gefordert implementiert.

#### **FA4 Filter- erfüllt**

In der Anforderungsanalyse werden zwei Filterkriterien beschrieben zum einen die Umkreissuche und zusätzlich ein Filtern nach Kategorien. Beides wurde implementiert und funktioniert wie gefordert.

### **FA5 Gruppenchat - erfüllt**

Der Gruppenchat ist wie gefordert implementiert und funktionsfähig.

### **FA6 Chatübersicht - erfüllt**

### **FA7 Kartenansicht - erfüllt**

Die Kartenansicht wurde implementiert und den geforderten Wischgesten noch die Funktionen des Beitreten bzw. entfernen des Events vom Stapel zugeordnet.

## **6.2 Nicht Funktionale Anforderungen**

### **NFA1 Robustheit - teilweise**

Es wurde versucht Unvorhersehbare und Falsche Eingaben zu erkennen und entsprechend zu reagieren dies funktioniert zum Aktuellen Stand im weitesten ganz gut. Es ist dennoch vereinzelt unvorhersehbares Verhalten im Zusammenhang mit Benutzereingaben aufgetreten. Hier gilt es die entsprechenden Stellen noch auszumachen und zu beseitigen.

### **NFA2 Erweiterbarkeit - erfüllt**

Durch die Verwendung des Model-View-Presenter Patterns ist eine einfache Erweiterbarkeit gegeben.

### **NFA3 Responsiveness - erfüllt**

Durch den permanenten Einsatz von asynchronen Programmiertechniken wurde sicher gestellt, dass die Applikation immer entsprechend schnell reagiert.

### **NFA4 Usability - erfüllt**

Um die Usability sicher zu stellen wurden die Google User Interface Guidelines als Referenz für die Gestaltung der Oberflächen verwendet.

## **7 Zusammenfassung und Ausblick**

In dieser Dokumentation wurde die Planung der Entwicklung einer Android App und die tatsächliche Implementierung dieser vorgestellt. Da die allermeisten der funktionale und nicht funktionalen Anforderungen erfüllt wurden, kann das Projekt als erfolgreich bewertet werden. Interessant wären nun einige Feldversuche mit der Benutzerschnittstelle um herauszufinden wie diese ankommt und ob die App tatsächlich so leicht zu benutzen ist wie wir uns das vorgestellt haben. Außerdem wären Stresstests hinsichtlich der Kommunikation zwischen Client und Server interessant. Ob die Implementierung auch skaliert wenn viele Benutzer gleichzeitig Daten schreiben und lesen.