Squash Manager for Android

Seminararbeit AndroidApp

Raphael Marques

10. Juni 2014

Inhaltsverzeichnis

1 Inhalt der Arbeit

1.1 Motivation

Racket-Sportarten sind cool, sie haben ein Problem, man muss die richtige Zeit und der richtige Partner finden. Die Applikation welche ich für die Seminar-Arbeit erstelle sollte dieses Problem lösen. Durch ein "Doodle-Style Termin und Spieler finder sowie Liga-Verwaltung werden die Teilnehmer angespornt, öfters zu Squashen.

1.2 Abgrenzung und Inhalt

Diese Arbeit konzentriert sich auf die API und die Android App.

1.3 Vorgehen

Folgendermassen wird Vorgegangen: - Erstellen einer REST-API um Daten der Applikation zur Verfügung zu stellen - Erstellen der Android App - Dokumentieren

1.4 Zielsetzung der Arbeit

Folgende Ziele sollten erreicht werden: - REST API um die Aktuellen Liga, Match, Spieler Daten von dem Server zu bekommen - Android App mit folgender Funktionalität: - Erstellen einer Liga - Registrieren neuer User - Andere Spieler für einen Match herausfordern - Termin Finden "Doodle-Style Protokollierung der gespielten Matches - Liga-System mit Rangliste aller Spieler in einer Liga - Dokumentation der App und der REST-API

2 Planung und Features

2.1 Features

2.1.1 Verabredung für ein Spiel

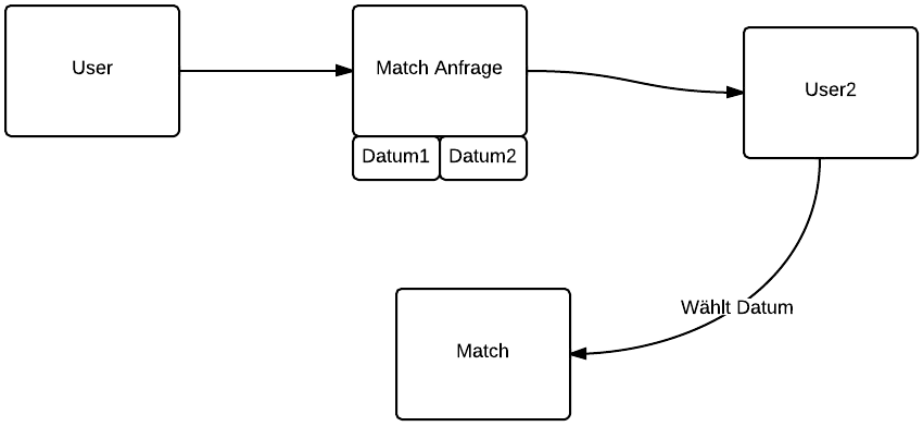

Abbildung 2.1: Grafische Darstellung wie sich User verabreden

Der erste Use Case zeigt wie ein User eine Herausforderung an einen anderen User senden kann. Der Empfänger der Herausforderung kann anschliessend zwischen mehreren vorgeschlagenen Daten wählen und die Herausforderung annehmen.

2.1.2 Erstellen und Beitreten einer Liga

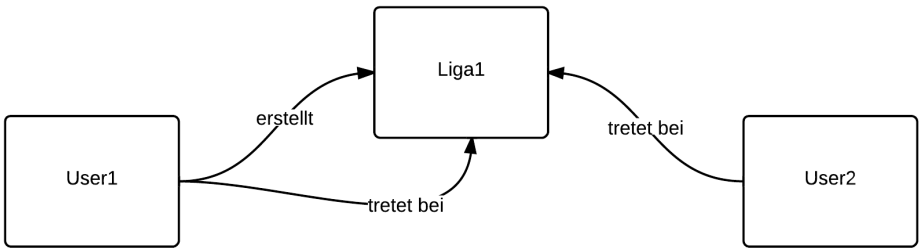

Abbildung 2.2: Liga erstellen und beitreten

Ein User kann selber eine Liga erstellen. Die Liga beinhaltet ein Ranking über alle User, welche daran teilnehmen. User können der Liga beitreten, um in das Ranking aufgenommen zu werden und Spiele gegen andere User der gleichen Liga zu spielen.

2.1.3 Spielen eines Matches

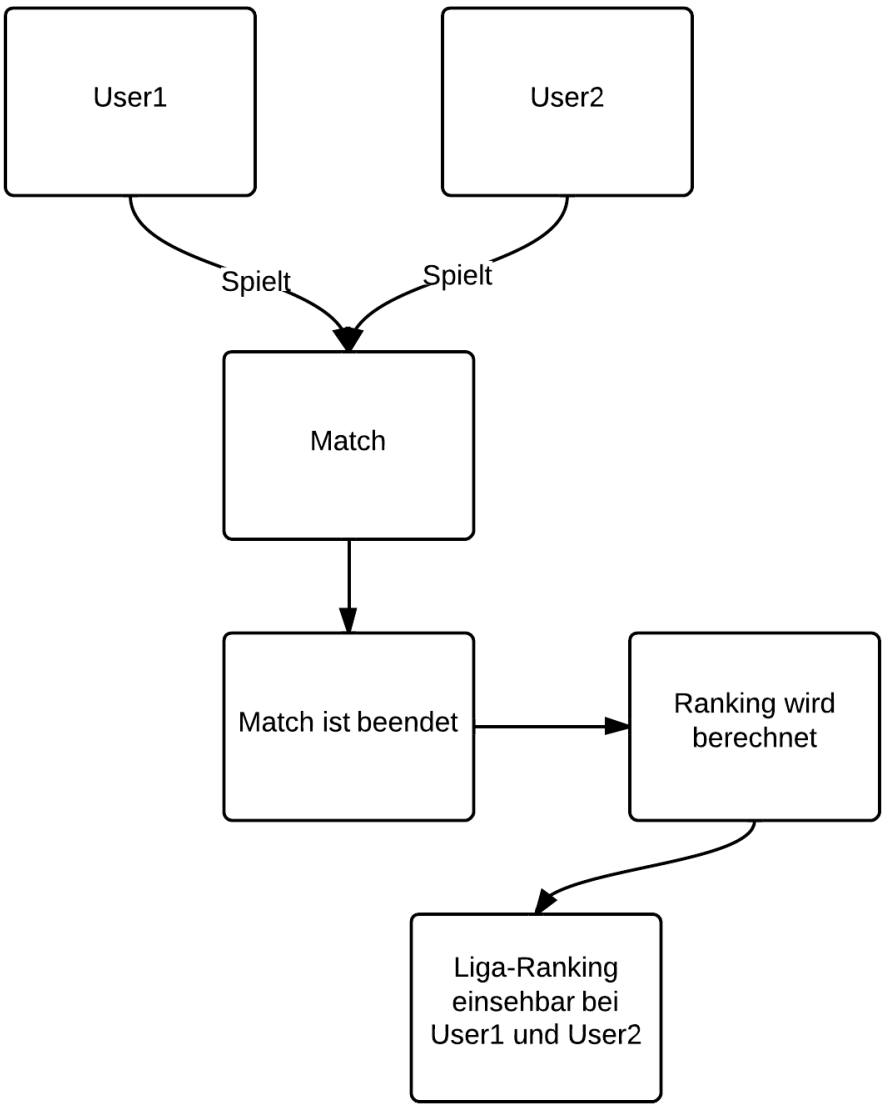

Abbildung 2.3: Spielen und Abschliessen eines Spiels

Sobald die User das Spiel vereinbart haben, können sie das Spiel spielen und ihre Resultate in der App festhalten. Sobald die User fertig sind mit spielen können sie das Spiel abschliessen. Nach Abschluss kann das Resultat nicht verändert werden und das Resultat wird im Ranking den einzelnen Spielern zugeschrieben.

2.2 Planung

Folgendermassen will ich vorgehen:

1. Erstellen einer Datenbank, in welche die Daten gespeichert werden

2. Erstellen einer REST-API, welche die Daten der APP zur Verfügung stellt

3. Erstellen einer App, welche der Benutzer bedienen kann.

In der Implementation findet man Details zu den einzelnen Punkten und wie ich die Problemstellungen gelöst und implementiert habe.

3 Implementation

3.1 Technologien

Folgende Technologien wurden verwendet:

- Spring und Hibernate/JPA um eine Rest-API zu erstellen
- Android Studio um die Android App zu erstellen

3.2 Kommunikation zwischen Server und Client

Die Androd App kommuniziert live mit der REST API. Sie hat keinen eigenen Cache oder

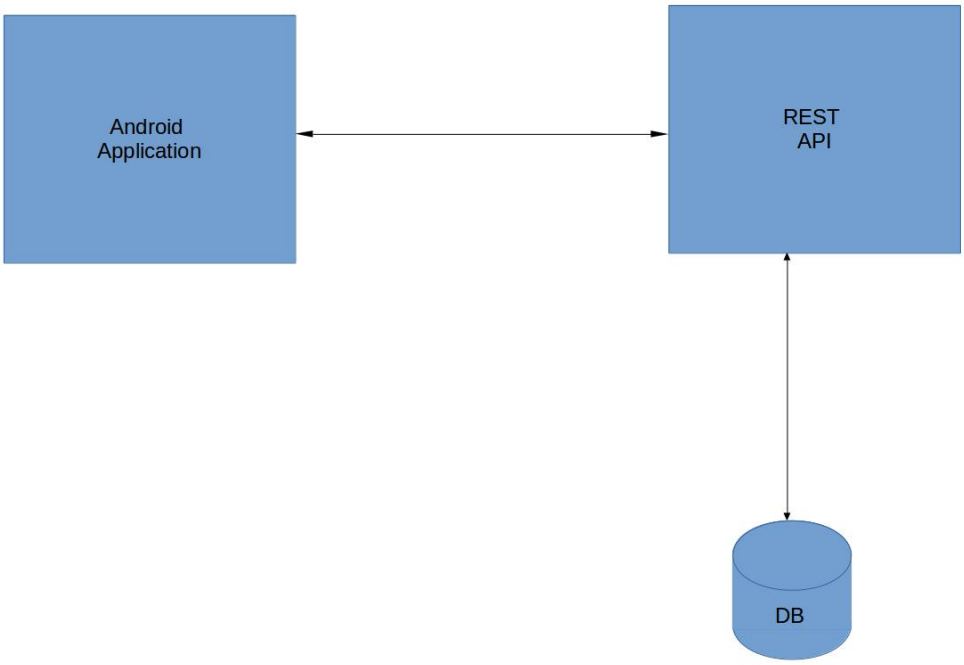

Abbildung 3.1: Übersicht wie Daten fliessen

3.3 DB

Da ich am Anfang des Projektes einen etwas grösseren Scope hatte, beinhaltet die Datenbank Daten, welche von der App nicht verwendet werden. Diese sind Rot im folgenden Diagramm

markiert:

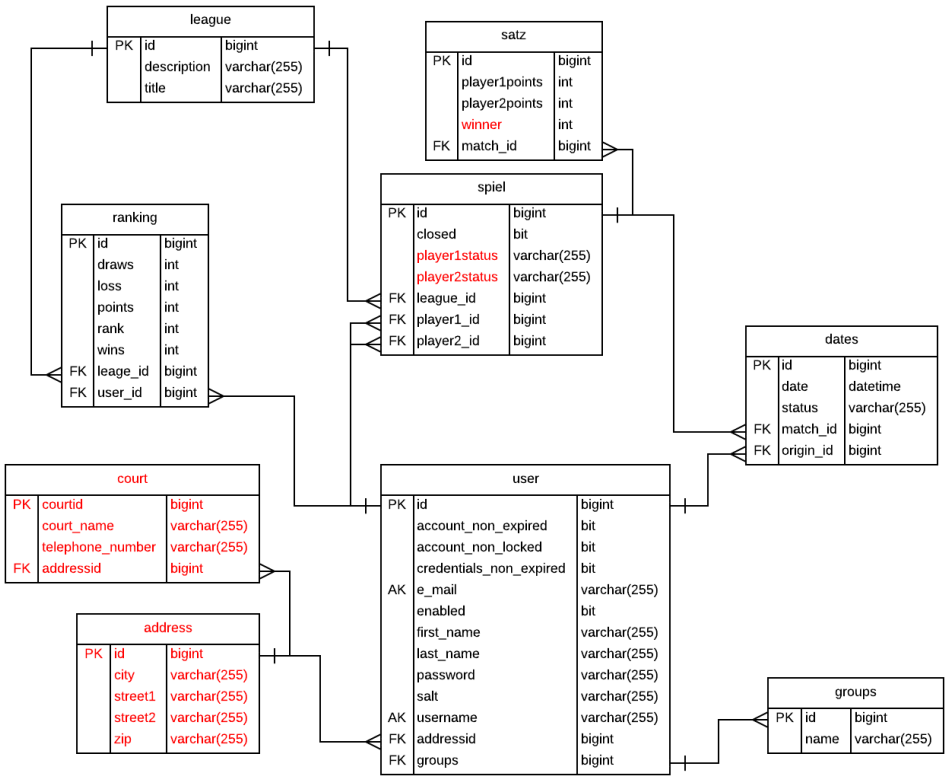

Abbildung 3.2: ERD Datenbank Squash

Wie man im oberen Layout sieht gibt es 3 zentrale Ressorucen:

- User: Verwaltet Benutzer, welche Squash spielen wollen, dient auch zur Authentisierung und Autorisierung der Benutzer

- Spiel: Jedes Spiel wird in der Tabelle Spiel festgehalten. Hier ist anzumerken, dass das Spiel eine Referenz zu verschiedenen Daten und verschiedenen Sätzen hat. So kann eine 1:M Relation erfüllt werden. Dies ist Pflicht, wenn man wie bei Doodle mehrere Daten zur Auswahl stellen will.

- League: Die Liga beinhaltet Spiele und Rankings. Der Server berechnet nach jedem Spielabschluss, welcher Spieler für welche Liga wie viele Punkte bekommen hat. Darum ist auch ein Spiel immer mit einer Liga referenziert. Jeder Eintrag im Ranking ist genau einer Liga und einem User zugeordnet. So kann auf alle Rankings einer Liga, sowie auf alle Rankings eines Users zugegriffen werden.

Diese Ressourcen bilden der Kern der Applikation und speichern alle relevanten Daten, welche von der Android Appliktaion über eine REST-API gebraucht wird.

Folgende Ressourcen habe ich bei dem Descoping (siehe Kapitel Reflexion) nicht mehr gebraucht:

- Address: Speichert Addressen von Benutzern und Courts. Diese Daten sind in diesem Stadium der App nicht relevant. Einige Features wären jedoch denkbar, wie z.B. Vorschläge welcher Court am nächsten zum Benutzer ist usw..

- Court: Für ein Spiel braucht es eine Zeit, einen Partner und einen Ort. Da dies normalerweise Relativ klar ist, hatte diese Variable jedoch tiefe Priorität und wurde descoped (siehe Kapitel Reflexion).

3.4 API

Als API habe ich die Datenbank über REST/JSON zu Verfügung gestellt. Dies hat den Vorteil das ich mehr Logik in die App einbauen konnte. Im Verlauf des Projektes stellte es sich jedoch als Hindernis heraus, da Fehler und Inkonsistenzen auftreten können, wenn man der grossteils der Logik im Client anstatt im Server implementiert.

Folgende Syntax habe ich für die API verwendet: ¡host¿/api/¡ressource¿[/id]

Mit GET request auf die Ressource empfängt man ein JSON Array aus allen verfügbaren Ressourcen auf dem Server.

Vorteile:

- Der Client kann visualisieren was nötig ist.

- Der Client hat den ganzen Serverkontext.

Nachteile:

- Der Client sieht evtl. zu viel, Sicherheitstechnisch nicht OK.

- Der Client muss selber Logik einbauen um das richtige Resultat anzuzeigen (was in dieser Arbeit erwünscht ist, da der Fokus auf Android liegen sollte.)

Als Rest API wird das MVC Framework Spring verwendet. Folgendermassen funktioniert die Kommunikation vom Controller, zur Datenbank und zurück zur View:

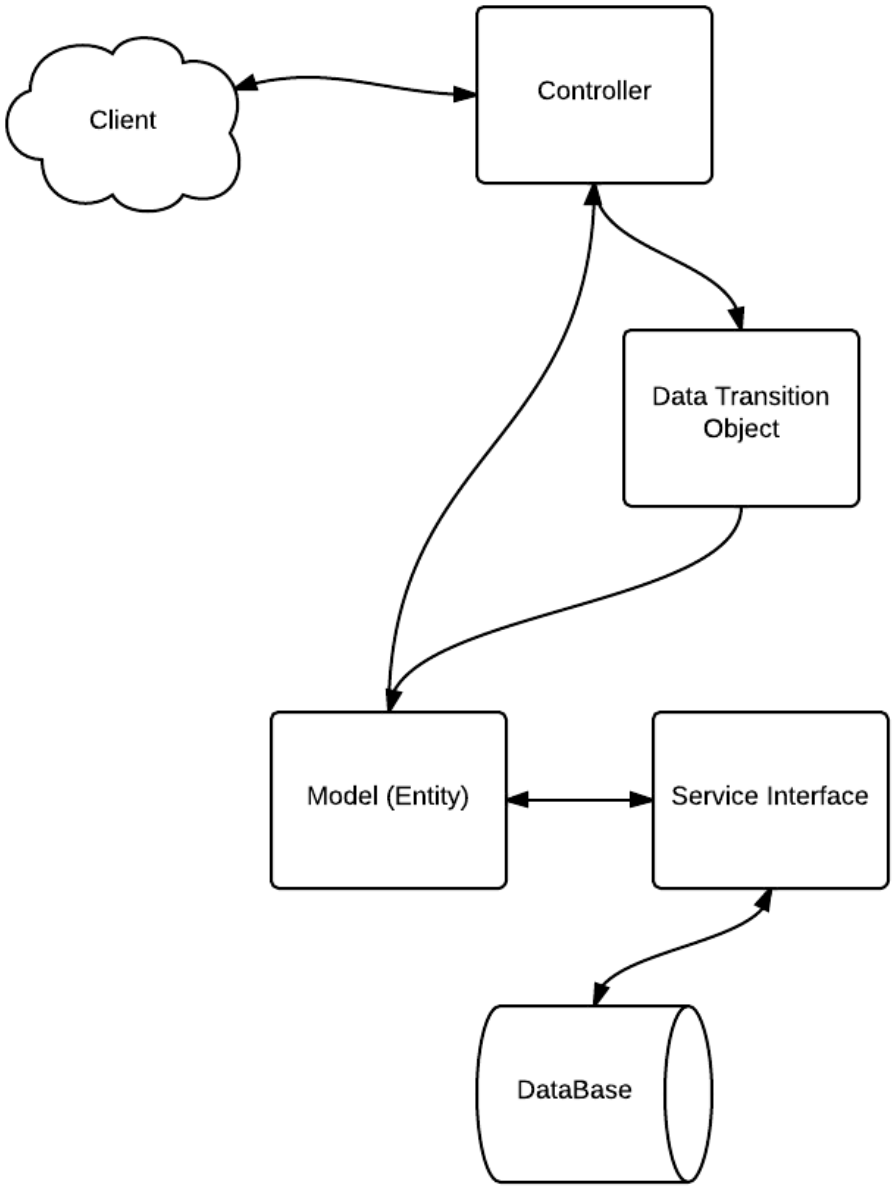

Abbildung 3.3: API Architecture

Wenn ein Request (JSON format oder GET) die API aufruft, wird sie vom Controller in ein Model-Objekt gespeichert. Die (geänderten) Werte des Model-Objektes werden mit einem Data Transition Objekt in das tatsächliche Datenbank-Model übertragen. Über ein Service Interface (DAO) wird das Model in die Datenbank gespeichert und abgerufen.

3.5 Android App

Folgendes Bild verdeutlicht die Architektur innerhalb der Android App.

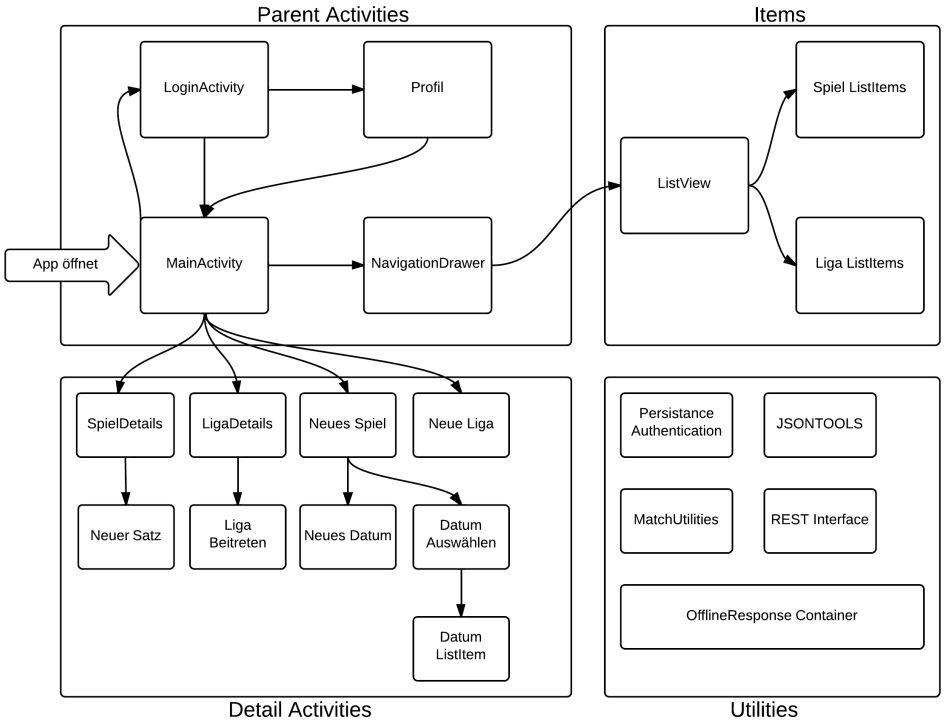

Abbildung 3.4: Android App Architecture

Der Entrypoint einer Android App ist immer die MainActivity. Diese Aktivität zeichnet ein NavigationDrawer und eine ListView, über welche man die verschieden Spiele bzw. Ligen sehen kann. Falls in dem Persistance Authentication Objekt kein Username oder Passwort hinterlegt ist, wird direkt zur Login-Activity weitergeleitet, welche eine Eingabemaske für Login oder Registrierung anzeigt.

Falls eine noch nicht bekannte E-Mail Adresse eingegeben wurde, versucht die APP, die neue e-Mail Adresse direkt zu registrieren. Falls die Registrierung erfolgreich durchgeführt werden kann, wird auf das Profil weitergeleitet um mögliche Änderungen vorzunehmen (z.B. wechseln des Benutzernamens). Das Profil ist zusätzlich über die MainActivity immer erreichbar.

Von der Main Activity können nun alle Operationen durchgeführt werden. Der Übersicht halber wird in diesem Dokument von zwei Teilen der Applikation gesprochen. Das Liga- und das Spielmanagement.

Nahezu alle Aktivitäten benutzen Utilities. Diese Utilities sind meist statische Methoden, welche typische und häufig wiederholte Funktionen zusammenfasst. Als Beispiel ist hier das REST Interface, welches HTTP Gets und Post ausführt bei Entgegennahme der URL und der zu sendenden Daten. Gleiches gibt es für JSON Operationen, welche nicht von der Standard JSON.org API abgedeckt sind.

3.5.1 Liga Management

Das Liga-Management ist nicht sehr komplex. In der ListView sieht man die Übersicht aller Ligen. Je nachdem ob der Spieler einer Liga beigetreten ist oder nicht, bestimmt was die Detailansicht ist. Wenn der Spieler der Liga noch nicht beigetreten ist und auf diese Klickt, erscheint eine Anfrage ob er der Liga beitreten will oder nicht. Sobald er beigetreten ist, sieht er die Details der Liga und das Ranking der einzelnen Spieler.

Als nächste Funktion kann man eine neue Liga hinzufügen. Hier gibt es eine simple Eingabe Maske.

3.5.2 Spiel Management

Das Spiel Management ist erheblich komplexer, da verschiedene Daten vorgeschlagen werden können, welche der Gegner anschliessend auswählen kann. Auch hier wird eine ListView mit der Übersicht der einzelnen Matches geladen. Falls der Spieler auf die Antwort des Gegners wartet, das Spiel schon vereinbart ist oder das Spiel aktuell stattfindet, so wird der Spieler auf die Detail Ansicht des Spieles weitergeleitet. Falls der Spieler von einem Gegner herausgefordert wurde, so bekommt er eine Auswahl von Daten und er kann auswählen wann das Spiel stattfindet.

Sobald das Spiel vereinbart wurde ist es möglich Sätze hinzuzufügen. Wurde ein Satz hinzugefügt, wird davon ausgegangen, dass das Spiel läuft. Man kann so lange Sätze eintragen, bis man das Spiel beendet. Mit der Beendigung des Spieles wird das Spiel gewertet und der Gewinner bekommt Punkte in das Ranking.

Um ein neues Spiel zu erstellen erscheint eine Eingabemaske, bei dem man Gegner, Liga und Daten auswählen kann. Zusätzlich kann man neue Daten erstellen (welches eine neue View öffnet).

4 Reflexion

4.1 Planung und Tests

Da ich nicht angenommen habe, dass eine Android App zu programmieren so komplex ist. Und zusätzlich die API und die Android App das erste Projekt dieser beiden Klassen waren, habe ich einige Fehleinschätzungen gemacht.

Nichts desto trotz habe ich die Minimalziele, welche in der Aufgabenstellung beschrieben wurden erreicht. Durch den Zeitmangel habe ich grosse Abstriche im Testing hinnehmen müssen, und habe die Appikationen nur von Hand getestet. Ich schätze die Stabilität der Applikation als Alpha Version ein.

4.2 Funktionalität

Ich habe viel zu spät bemerkt, dass ich viel zu viel Funktionalität umsetzen wollte, welche ich aus Zeitgründen gar nicht konnte:

- Ein HTML5 Webinterface neben der API
- Bessere Sicherheit und User-Based Filtering
- Ort des Matches berücksichtigen
- Automatische Herausforderungen der Liga (z.b. alle 2 Wochen)
- Remove-Endpoints für alle Ressourcen

Diese Funktionalitäten wurden darum Descoped und würden in einem nächsten Release der Applikation (inklusive Tests) umgesetzt werden. Die API sowie die Applikation würde eine Weiterentwicklung problemlos unterstützen. Ich habe zusätzlich darauf geachtet das die Applikation in Zukunft gut erweitert werden kann.

4.3 Hibernate-Bug und Workaround

Wärend der Arbeit trat ein Hibernate Bug auf. Die Funktion findOne(id) gab ein falsches Resultat zurück. Ich konnte ein Workaround herstellen, indem ich findAll() aufrufte und mein Objekt aus dem Array aus Objekten herauslas.

Abbildungsverzeichnis