Racketsports Manager

Semesterarbeit ZHAW

Autor: Raphael Marques
Betreuer: Michael Reiser

4. August 2015

Inhaltsverzeichnis

1 Inhalt der Arbeit

Racket-Sportarten haben ein Problem. Man muss die richtige Zeit und den richtigen Partner finden. Die Applikation, welche ich für die Semester-Arbeit erstelle sollte dieses Problem lösen. Durch einen "Doodle-StyleTermin und Spieler-Finder sowie eine Liga-Verwaltung werden die Teilnehmenden motiviert, öfters zu squashen.

1.1 Ausgangslage

Einen Partner für Racket-Sportarten zu finden, ist nicht immer sehr einfach. Wenn man schliesslich jemanden gefunden hat, ist es immer schwer, einen Termin zu finden. Spielt man regelmässig gegen die gleiche Person, stellt man sich oft auf diese ein und lernt mit der Zeit nur noch wenig dazu.

Diese App sollte diese Probleme lösen, indem man einfach miteinander Termine vereinbaren kann und das Termine vom System für die User automatisch generiert werden.

1.2 Aufgabenstellung

Um die Ziele zu erreichen, muss methodisch vorgegangen werden. In der Dokumentation müssen angewendete Technologien begründet werden sowie Konzepte und Architekturen der Applikation festgehalten werden.

1. Informieren über den momentanen Markt und Organisation rund um Racketsport
    - Andere Applikationen rund um Racketsport identifizieren
    - Zielgruppen- und Anwendungsfallanalyse
2. Anforderungsanalyse, welche Funktionalitäten dem Nutzer einen echten Mehrwert bieten
    - Basierend auf (1) eine Anforderungsanalyse erstellen
    - User Stories erstellen
    - Sprints planen
3. Konzeption der App, Technologien wählen
    - Entscheidungsmatrix, welche Technologien verwendet werden
    - DB Design
    - Klassen-Diagramm
    - GUI-Entwurf
4. Die Applikation umsetzen

   - Applikation programmieren
   - Unit Tests erstellen/ausführen
5. Systemtests durchführen
   - Systemstests konzipieren
   - Systemtests durchführen
   - End-to-End Testing
6. Benutzertests durchführen
   - Benutzer einladen für Alpha-Test
   - Feedback von Benutzer verlangen

1.3 Erwartete Resultat

Folgende Resultate werden erwartet:

- Marktanalyse & Benutzerverhaltensanalyse zur vereinbarung von Spielen
- Anforderungsanalyse
- Recherche für einzusetzende Techniken
- Konzepte und Implementationstechniken
- User Stories und Projektplanung
- Android App und Webapplikation
- Testplan und Umsetzung
- Benutzerfeedback des Alpha-Tests
- Fazit

1.4 Zielsetzung der Arbeit

Ziel der Arbeit ist eine WebApplikation sowie eine Android App mit bestimmten Funktionalitäten zu erstellen. Der Autor der Arbeit soll so Zugang zu neuen Technologien im Webbereich erhalten und Erfahrung in der Webapplikations-Programmierung sammeln.

Aufgabenstellung:

- Dokumentation über Entscheidungen und Implementationstechniken
- Webapplikation mit folgender Funktionalität:
  - RestAPI um alle untenstehenden Funktionalitäten
  - Liga erstellen und löschen
  - Einer Liga beitreten und eine Liga verlassen
  - Spiel vereinbaren
  - Termin für Spiel finden
  - Spielresultat eintragen
  - Rangliste der Liga berechnen
  - Automatische Spielvorschläge innerhalb der Liga (z.B. jeder Spieler
  - spielt jede zweite Woche ein Spiel)
- Android App, welche auf die WebAPI zugreift mit gleicher Funktionalität wie WebApplikation

1.4.1 Grobplanung

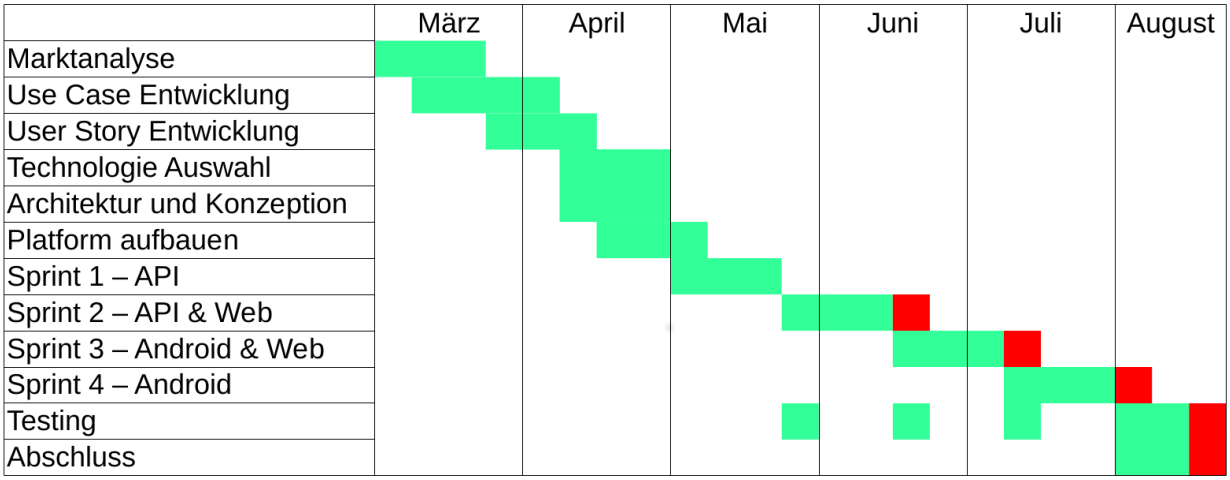


Abbildung 1.1: Grobplanung für Applikation

1.4.2 Termine

2 Analyse

Das Ziel der Software ist, den User in der Terminfindung, Protokollierung und Partnerfindung optimal zu unterstützen. Dieser Teil der Dokumentation dient zur Findung exakter Anforderungen an die Software, die den User optimal unterstützen. Zusätzlich wird Anfangs untersucht wie das Marktumfeld rund um die geplante Applikation aussieht, um einen möglichen Erfolg einer solchen Applikation zu schätzen sowie mögliche Synergieeffekte zu identifizieren.

2.1 Marktumfeld

Um mögliche Synergien oder Wettbewerber zu identifizieren, müssen zuerst die geplanten Basis Funktionalitäten aufgelistet werden:

- Vereinfachung zur Identifizierung von Partnern
- Vereinfachung zur Terminvereinbarung
- Vereinfachung eines Amateur-Liga Management
- Vereinfachung zu Protokollierung eines Spiels

2.1.1 Identifizierung von Partnern

GlobalTennisNetwork.com ist eine Community zur Identifizierung von Tennispartnern. Die Website ist fokussiert auf Tennis. Andere Racket Sportarten werden nicht behandelt. Dadurch sieht der Autor dieser Service nicht als direkte Konkurrenz. Eine Verbindung mit dem Service um eine Grössere Population von Potenziellen Tennispartnern zu erreichen wäre vorstellbar.
Spontacts ist eine Android Applikation, welche Spontane Terminvereinbarungen ermöglicht. Es kann zusätzlich als Identifizierung von Partnern für jegliche Freizeitaktivitäten dienen. Es gibt jedoch keine Fokussierung auf Racket Sport. Nichts desto trotz kann Spontacts als konkurrenz zu der geplanten Applikation angesehen werden, da zwei Funktionalitäten (auch wenn der Fokus nicht auf Sport liegt teilweise abgedeckt werden können. Jedoch wird die Konkurrenz nicht als erheblich eingeschätzt.
sport42.com bietet auch eine Möglichkeit zur Identifizerung eines Portenziellen Sportpartners. Racket Sportarten werden hier auch abgedeckt. Nach Recherchen stellte sich jedoch heraus, das der Fokus der Applikation auf dem US Markt liegt. Bei einer potenziellen Expansion der geplanten Applikation müsse die Konkurrenz neu evaluiert werden. Da der Fokus

jedoch im Moment auf den deutschsprachigen Raum gerichtet ist, wird dieser Service nicht als Konkurrenz eingeschätzt.
sportpartner.com ist ein auf Sport fokussierter Service um Partner zu identifizieren. Der Service deckt auch Racketsportarten wie Tennis, Squash sowie Badminton ab. Bis jetzt gibt es allerdings nur wenig aktive Spieler in Europa.

2.1.2 Terminvereinbarung

Doodle ist der bekannteste Terminvereinbarungs Service. Ein ähnlicher Algorithmus, welcher in der geplanten Applikation verwendet wird, wurde von den Ersteller dieser Applikation entwickelt. Jedoch gibt es bei Doodle keinerlei Fokus auf Inhalt zum Termin. Es ist möglich Business wie auch Freizeit Termine zu erstellen. Die geplante Applikation kann davon Profitieren einen ähnlichen, auf Sport Termine fokussierte Terminvereinbarungs Algorithmus zu entwickeln. Sollte dieser genügend intuitiv Umgesetzt sein, wird Doodle nicht als Konkurrenz betrachtet.
Moreganize.ch wie Doodle benützt auch Moreganize einen Terminvereinbarungs Algorithmus.

2.1.3 Amateur-Liga Management

LeagueManager Wordpress Plugin ist eine Zusatzsoftware für Wordpress, mit welchen man Teams und Matches zusammenstellen kann und so Ligen erstellen kann. Der Fokus liegt hier grösstenteils auf Teamsports.

Excel von Microsoft wird oft als Tracking und Organisation für Ligen verwendet. Die Einfachheit und das breite Skillset spricht für diese Lösung. Jedoch ist Zusammenarbeit und Organisation umständlich.

2.1.4 Protokollierung eines Spiels

Excel von Microsoft wird oft als Tracking und Organisation für Ligen verwendet. Die Einfachheit und das breite Skillset spricht für diese Lösung. Jedoch ist Zusammenarbeit und Organisation umständlich. Als alternative, welche die Zusammenarbeit vereinfacht bietet sich auch **Google Docs** an.

2.2 Benutzergruppen/Ist-Analyse

Die geplante Applikation wird nicht für alle Benutzergruppen interessant sein. Um das Zielklientel zu finden wird erstellt diese Kapitel eine Kategorisierung in verschiedene Benutzergruppen. Anschliessend wird abgeschätzt, wie welche Benutzergruppe von der Applikation profitieren kann.

2.2.1 Gelegenheitsspieler

2.2.1.1 Analyse

Gelegenheitsspieler spielen nicht regelmässig (wöchentlich) Racketsport. Sie vereinbaren mit - meistens wenige Personen in engem oder erweitertem privaten Umfeld - unregelmässig Spiele. Oft pausieren Gelegenheitsspieler mehrere Monate und haben anschliessend intensivere Phasen mit mehreren Spielen innerhalb wenigen Wochen. Der Aufwand um ein Spiel zu vereinbaren und einen passenden Court zu finden ist dementsprechend gross. Viel fehlt auch an einem passenden Partner im privaten Umfeld, insbesondere wenn der Spieler Ambitionen hegt sich zu verbessern. Schlussendlich ist die Motivation zu einer „gewissen Regelmässigkeit “nicht hoch, da die Vereinbarung eines Spieles aufwändig ist, sowie eine Verbesserung des Spiels gegenüber anderen Spielern nicht schnell ersichtlich ist.
Zusammengefasst lassen Sich folgende Charakteristiken zusammenfassen:

- Einzelne oder wenige Partner im privatem Umfeld
- Oft längere Pausen gefolgt von intensiveren Phasen
- Grosse Aufwand um Spiel zu vereinbaren
- Fehlende Partner bei gewollter Verbesserung im Spiel
- Fehlende Vergleichsmöglichkeiten, da wenige Partner und meist keine Protokollierung der Ergebnisse
- Wenig Anreize um nächstes Spiel zu organisieren

2.2.1.2 Use Cases

Gelegenheitsspieler können ausserordentlich von der geplante Applikation profitieren.
Neue **Partner** in der Nähe können unkompliziert mit der Applikation identifiziert werden. So ist es möglich längere Pausen, wenn der originale Partner in den Ferien ist oder schlicht verhindert ist, zu verhindern.
Courts und Spielzeiten können schnell vereinbart werden. Dies verringert den Aufwand um ein Spiel durchzuführen und erhöht somit die Motivation öfters Squash zu spielen. Spiele können **protokolliert** werden und so kann eine Statistik erstellt werden, welche wiederum als Anreiz dienen kann um besser zu werden.
Durch ein **Liga Management** können zusätzliche Anreize, neue Partner sowie eine Regelmässigkeit gefunden werden. Möglicherweise sind ein Grossteil der Gelegenheitsspieler nicht an einer solchen Liga interessiert, jedoch gibt es ein gewisses Potenzial dafür.

2.2.2 Regelmässige Spieler

2.2.2.1 Analyse

Regelmässige Spieler spielen regelmässig zu einem vereinbarten Termin mit einem oder mehreren Partner einen Racketsport. Meist werden genannte vereinbarte Termine mit den gleichen Partnern ausgetragen. Bei den vereinbarten Terminen gibt es gewisse Variationen. So

kann der Zeitpunkt und auch der Wochentag variieren.

Zusätzlich zu den regelmässigen Terminen, kommen meist noch unregelmässige Termine in der Charakteristik der Gelegenheitsspieler hinzu.

Durch die Regelmässigkeit ist diese Spielergruppe meist einiges ambitionierter als Gelegenheitsspieler. Dadurch das Sie regelmässig mit den gleichen Spielern spielen, wollen Sie besser sein als die anderen. Da Sie jedoch nicht in einer Liga spielen gibt es meist kein Protokoll der Resultate oder ein Ranking.

2.2.2.2 Use Cases

Regelmässige Spieler können durch die geplante Applikation sehr unkompliziert ihre Spiele **protokollieren**.
Eine **Liga** kann zwischen mehreren regelmässigen Spielern arrangiert werden. Diese kann durch regelmässige **Terminvereinbarungen vom System** dazu beitragen das es keine Pausen gibt bei Krankheit oder Abwesenheit. Spieler die ausserhalb der eigenen Stammgruppe spielen wollen, können gleichzeitig bei **öffentlichen Ligen** weitere Erfahrungen sammeln.

2.2.3 Clubmitglieder

2.2.3.1 Analyse

Clubmitglieder Spielen regelmässig und haben dafür fix Vereinbarte Clubtrainings. Diese sind nicht flexibel. Zusätzlich besteht auch eine Infrastruktur für Ranking über die Liga sowie eine Protokollierung von Spielen in den meisten Fällen. Clubmitglieder sind somit nicht im Hauptfokus dieser Applikation. Höchstens Ausserhalb des Clublebens ist es gut möglich das ein Clubmitglied diese Applikation braucht um neue Spiele zu vereinbaren oder in einer privaten Liga mitzuspielen.

2.2.3.2 Use Cases

Neue Clubs ohne Infrastruktur könnten diese Applikation für ihre Administration verwenden.

2.3 Anforderungsanalyse

2.3.1 Vision

Erleichterung zur Terminfindung und Administration von Racketsportspieleren

2.3.2 Ziele

Folgende Business Prozesse sollten unterstützt werden:

- Identifizierung von Partnern
- Terminvereinbarung
- Amateur-Liga Management
- Protokollierung eines Spiels

2.3.3 Rahmenbedingungen

2.3.3.1 Allgemein

Die Applikation hat keine konkreten Rahmenbedingungen. Sie ist eine Standalone Applikation und hat keine externen Abhängigkeiten.

2.3.3.2 Technologie

Es gibt keine Einschränkungen, welche Technologie benutzt werden sollte, solange die Anforderungen erfüllt werden.

2.3.3.3 Erweiterbarkeit

Die Applikation soll möglicht einfach erweiterbar sein. Zusätzliche Sicherheit, neue Funktionalitäten sowie skalierbarkeit in Performance sowie Stabilität sollten mit der eingesetzten Technologie möglich sein.

2.3.3.4 Sicherheit

Technologien sollten Stanard Sicherheitsanforderungen im Web umsetzen können. Privacy und Integrity Anforderungen sind in diesem Proof of Concept noch nicht geplant.

2.3.3.5 Stabilität und Performance

Für den Proof of Concept sind keine Stabilitäts- und Performance Anforderungen nötig.

2.3.4 Schnittstellen

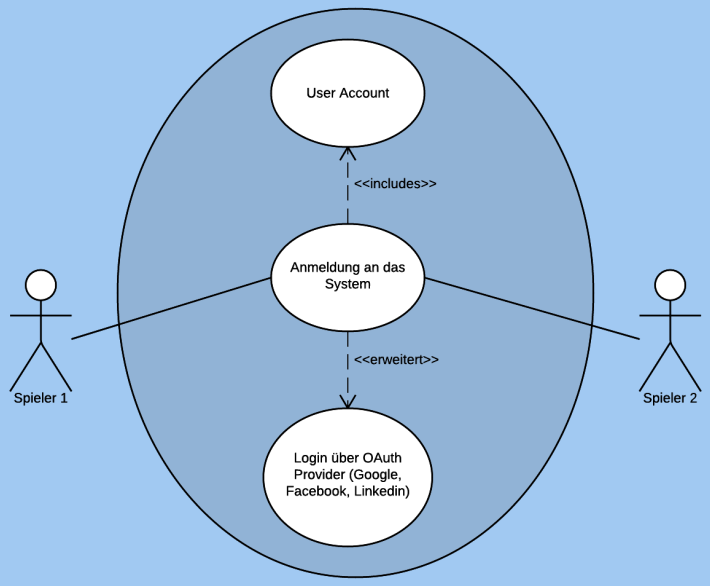
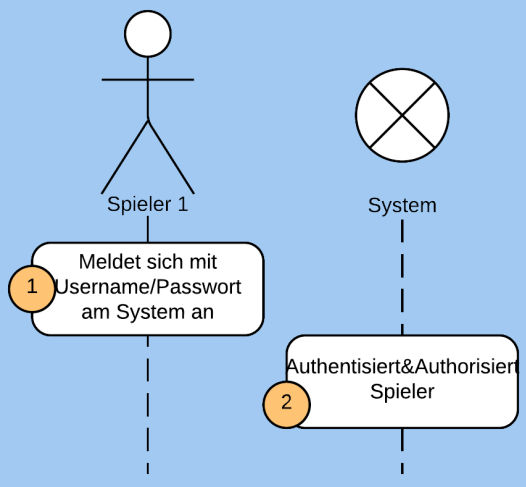
Folgende Tabelle zeigt interne sowie externe Schnittstellen auf:

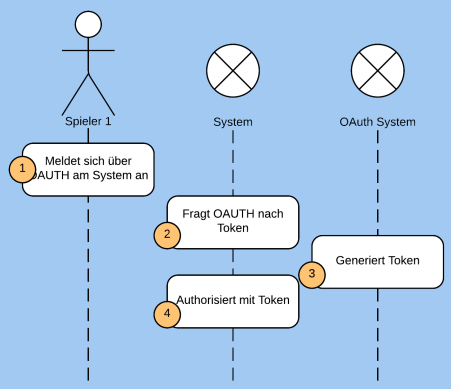
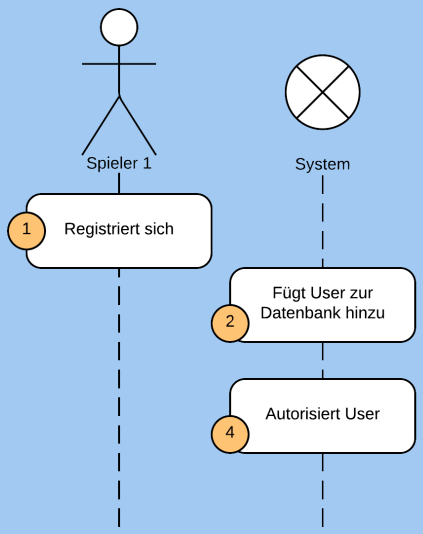
ID	Name	Beschreibung
I1	API ⟶ Client	Der Client greift auf die API zu um die aktuellen Daten dem Benutzers anzuzeigen. API sollte HTTP REST benutzen.
I2	Client ⟶ Browser	Der Client läuft innerhalb eines Browsers. Der Browser führt die Applikation - bestehend aus HTML, CSS und Javascript - aus.
I3	Browser ⟶ Android	Der Browser läuft innerhalb der Android Applikation. Die Android Applikation Emuliert den Browser und zeigt die Applikation als App dem Benutzer an.

2.3.5 Anwendungsfälle

2.3.5.1 UC0 - Login in das System

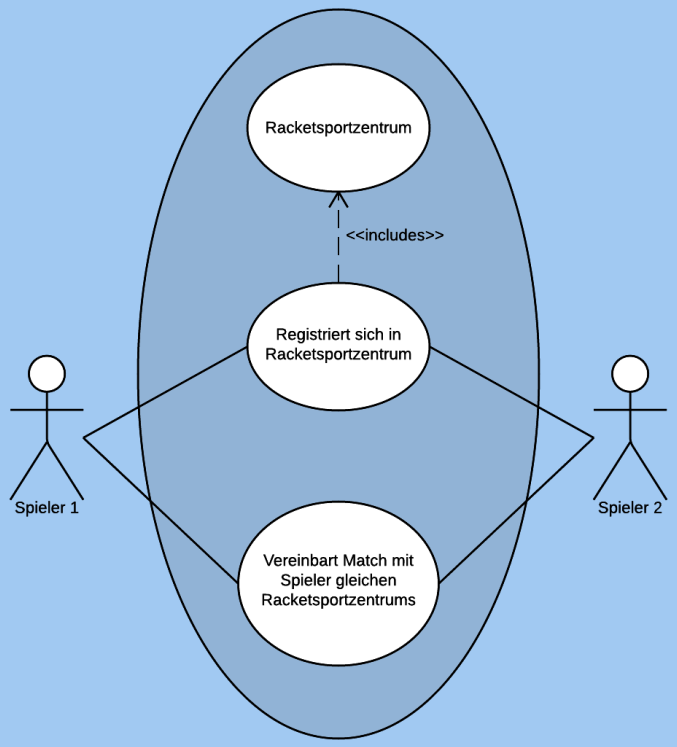
UC0	Login in das System
Beschreibung	Ein User des Systems will sich mit dem System authentisieren. Der Benutzer bekommt durch die Authentisierung Autorisierungen zugesprochen sowie ein Profil und Daten zugeordnet. Es ermöglicht dem User seine eigene Ansicht in die Daten des Systems zu erhalten. Da dieses System eine Applikation individualisiert für den User darstellt, sind alle Ansichten in das System einer Autorisierung zugeordnet und können nur von authentisierten Benutzern benutzt werden.

Diagramm	
Version	1.0
Vorbedingung	-
Anforderungen	REQ 0.1 - Anmeldung an das System REQ 0.2 - Registrierung im System
Testfälle	Test0.1: Anmeldung über Username/Passwort Test0.2: Anmeldung über OAuth Test0.3: Registrierung über Username/Passwort Test0.4: Registrierung über OAuth
Standard Sequenz	

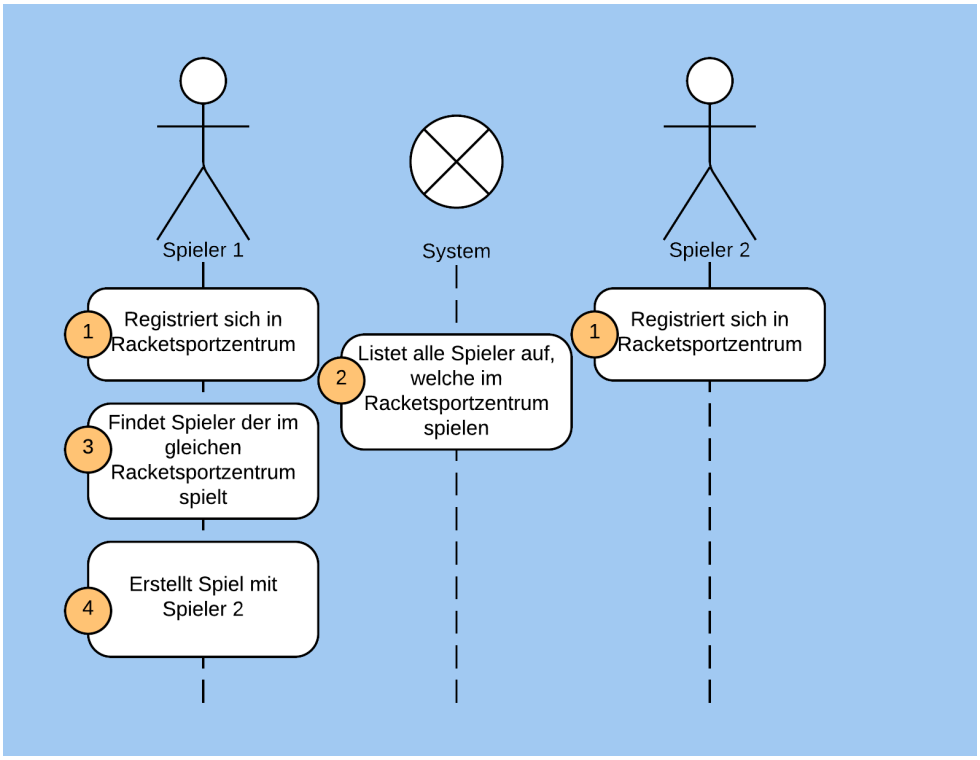
Alternative Sequenzen	 

2.3.5.2 UC1 - Identifizierung von Partnern

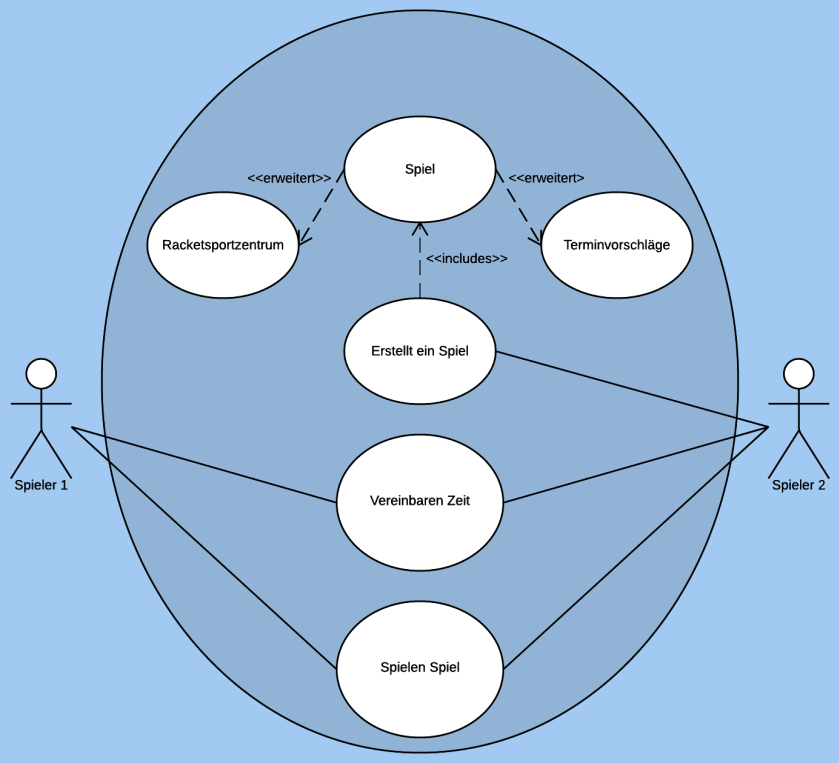
UC1	Identifizierung von Partnern
Beschreibung	Zwei Partner - A und B - wollen einen bestimmten Racketsport in einer bestimmten Region spielen. Der User A oder B kann Spielvorschläge an andere User senden. Dafür kann er nach Region filtern.

Diagramm	
Version	1.0
Vorbedingung	-
Anforderungen	REQ 1.1 - Zuteilung zu Regionen REQ 1.2 - Broadcast Herausforderungen REQ 1.3 - Details zu Spieler REQ 1.4 - Friend System
Testfälle	Test1.1: TBD Test 1.2: TBD

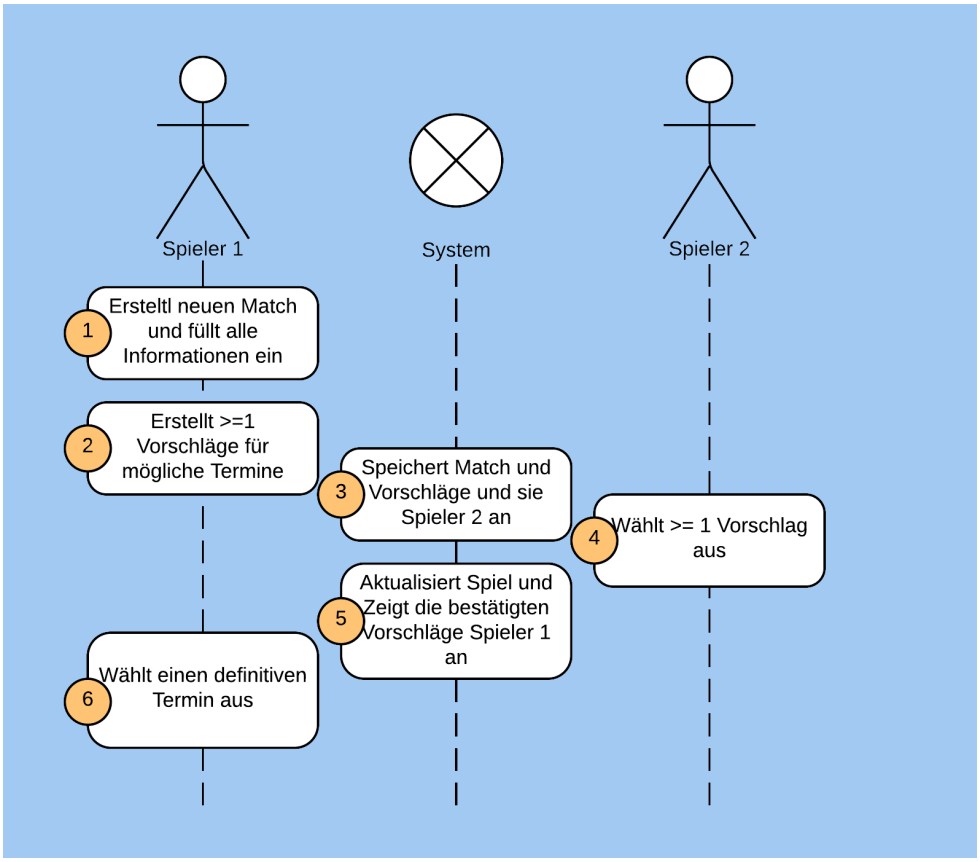
Standard Sequenz



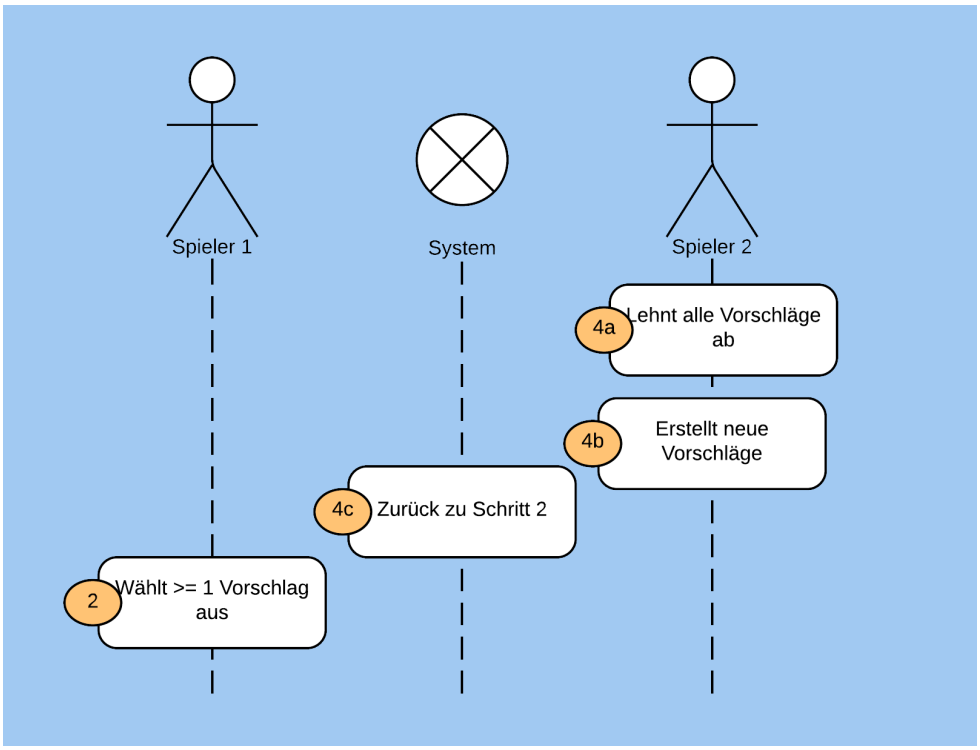
2.3.5.3 UC2 - Terminvereinbarung

UC2	Terminvereinbarung
Beschreibung	Der Spieler A kann einen oder mehrere Spielvorschläge senden. Anschliessend kann der Spieler B einer dieser Spielvorschläge **annehmen**, **ablehnen** oder **neue Spielvorschläge senden**. Sobald beide Nutzer einen Spielvorschlag angenommen hat, gilt der Termin als vereinbart.
Diagramm	
Version	1.0
Vorbedingung	UC0
Anforderungen	REQ 2.1 - Spiel erstellen REQ 2.2: Spielterminvorschläge erstellen REQ 2.3: Spielterminvorschläge annehmen REQ 2.4: Spielterminvorschläge ablehnen REQ 2.5: Regelmässiges Spiel erstellen
Testfälle	Test2.1: TBD Test2.2: TBD

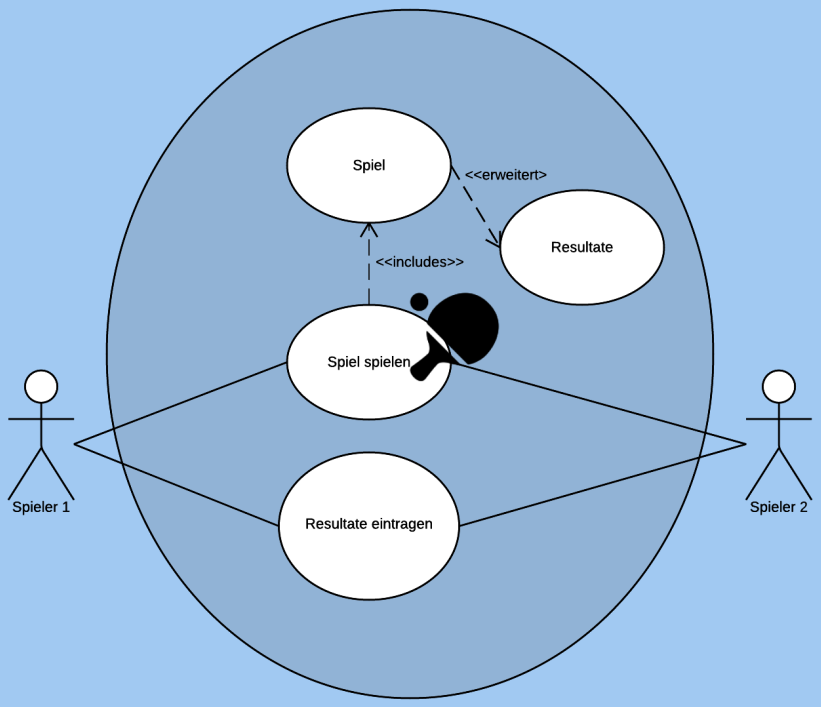
Standard Sequenz



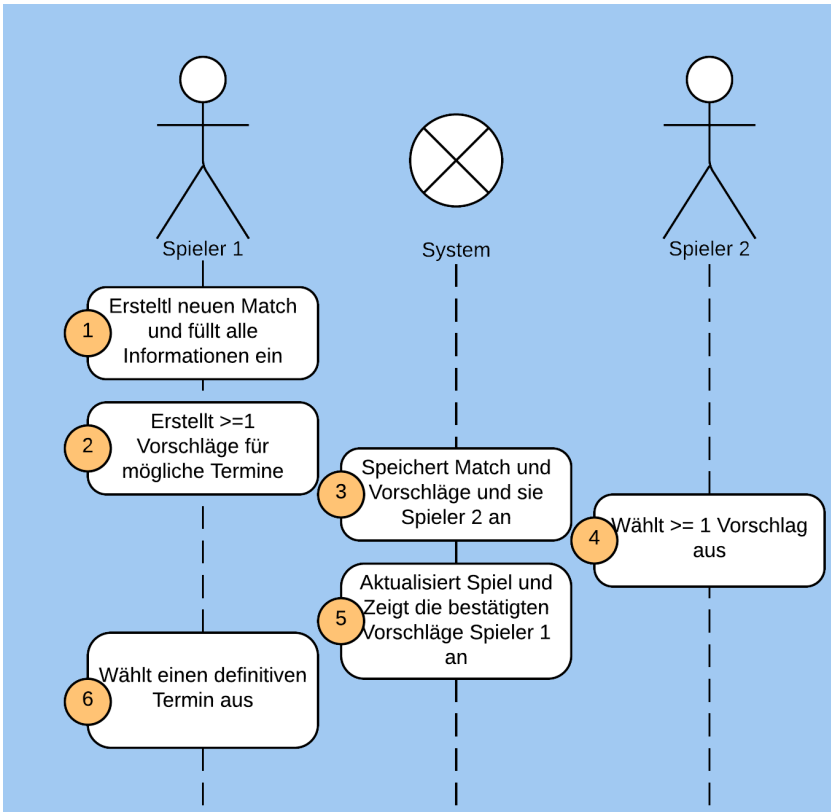
Alternative Sequenzen



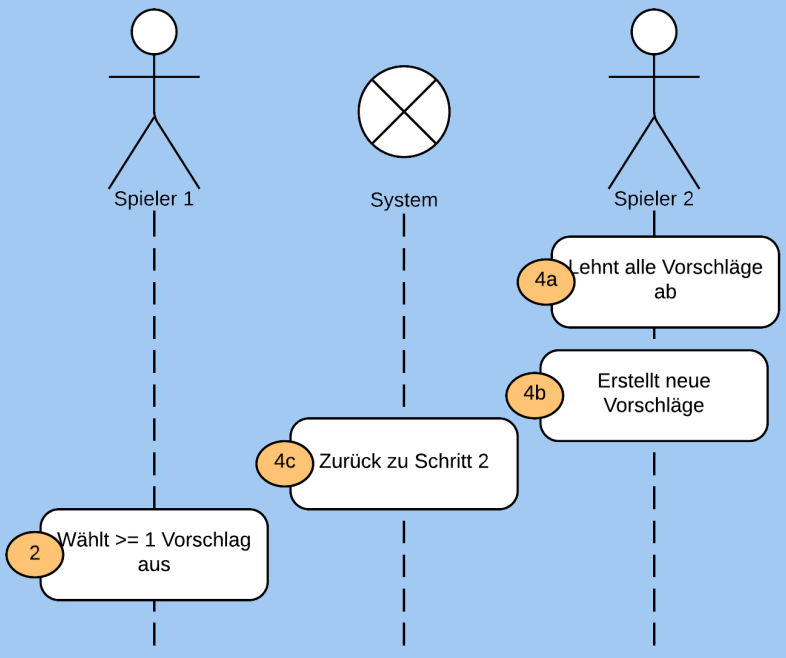
2.3.5.4 UC 3 - Spiel Protokollieren

UC3	Spiel protokollieren
Beschreibung	Nachdem ein Spiel durchgeführt wurde, kann der Spieler B sowie der Spieler A die Scores eintragen oder bestätigen. Anschliessend können beide Spieler die jeweiligen vergangenen Spiele jederzeit in der Applikation nachschlagen. Optional wird das protokollierte Spiel in einer Liga gewertet (falls vor dem Spiel so deklariert.
Diagramm	
Version	1.0
Vorbedingung	UC0 UC1
Anforderungen	REQ 3.1 - Ergebnisse des Spiels eintragen REQ 3.2: Bestätigung der Ergebnisse
Testfälle	Test3.1: TBD Test1.2: TBD

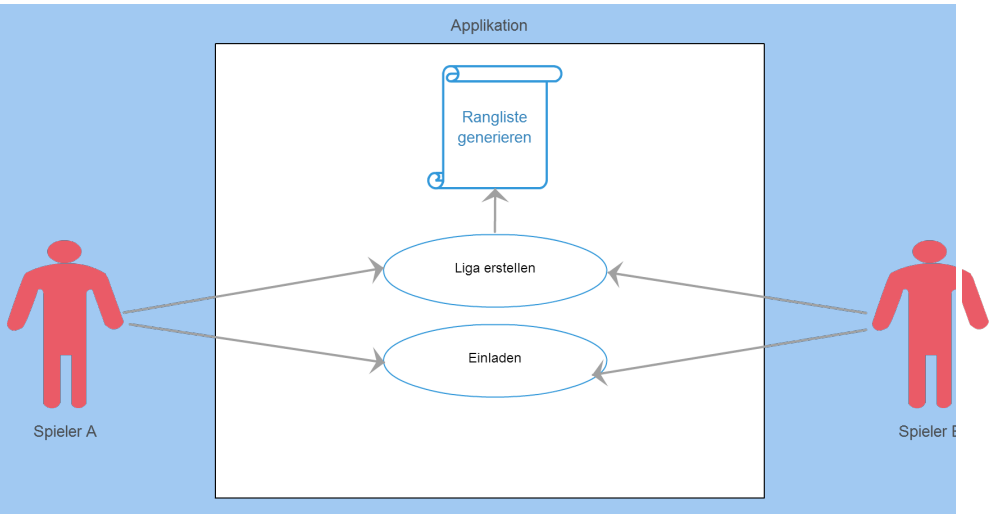
Standard Sequenz



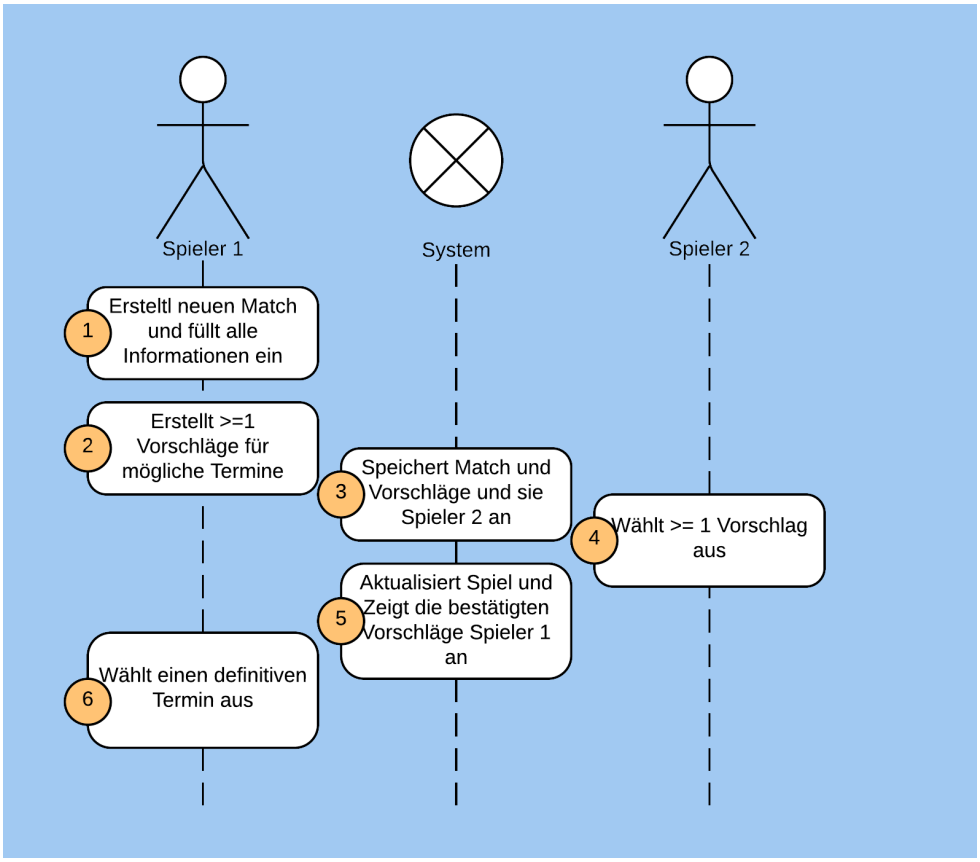
Alternative Sequenzen



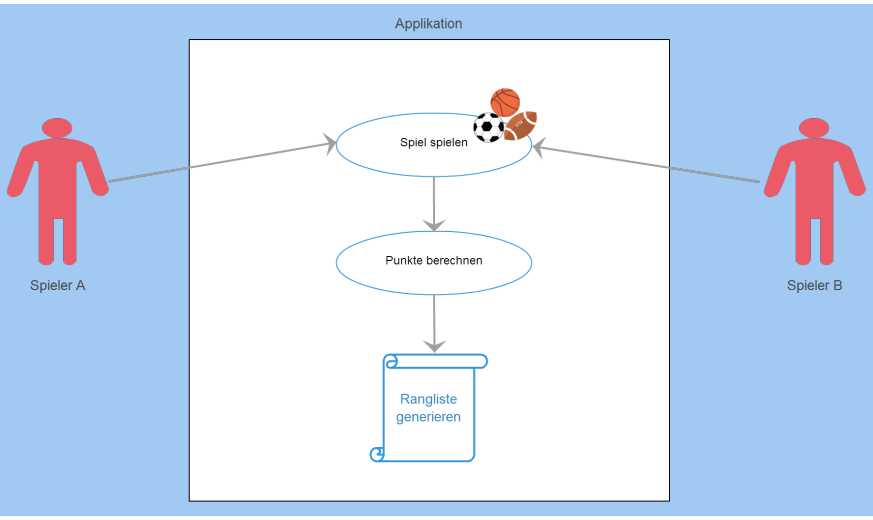
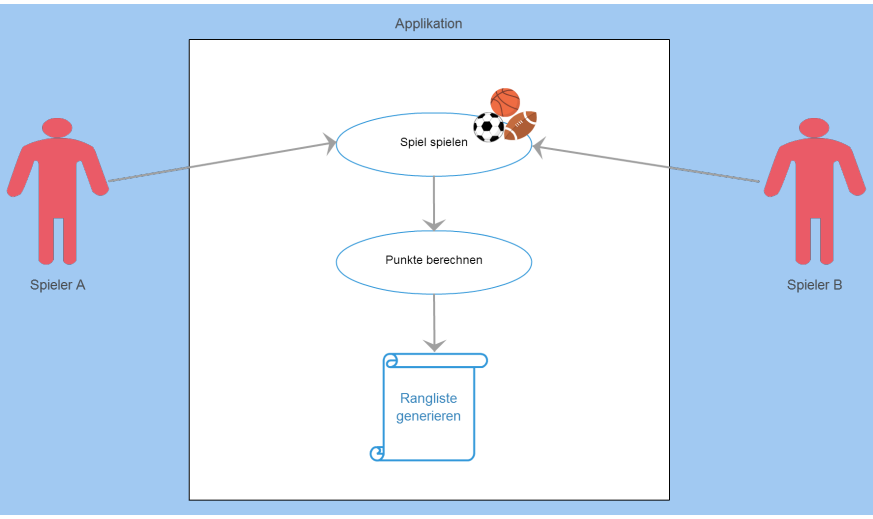
2.3.5.5 UC 4.1 - Liga erstellen

UC4.1	Liga erstellen
Beschreibung	Jeder Spieler kann eine Liga erstellen. Diese Liga generiert eine Rangliste. Der Spieler kann anschliessend andere Spieler in die Liege einladen oder die Spieler treten der Liga bei
Diagramm	
Version	1.0
Vorbedingung	UC0
Anforderungen	REQ 4.1: User in die Liga einladen REQ 4.2: Als User einer Liga beitreten
Testfälle	Test4.1: TBD Test4.2: TBD

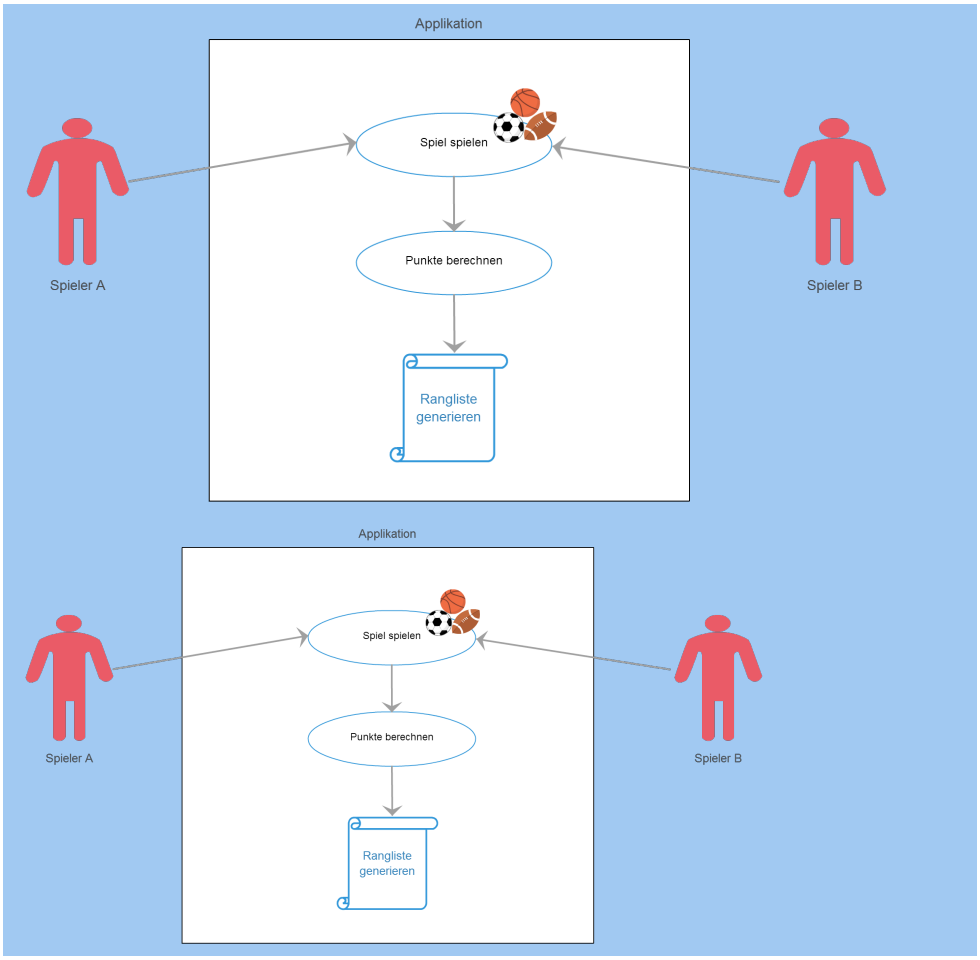
Standard Sequenz



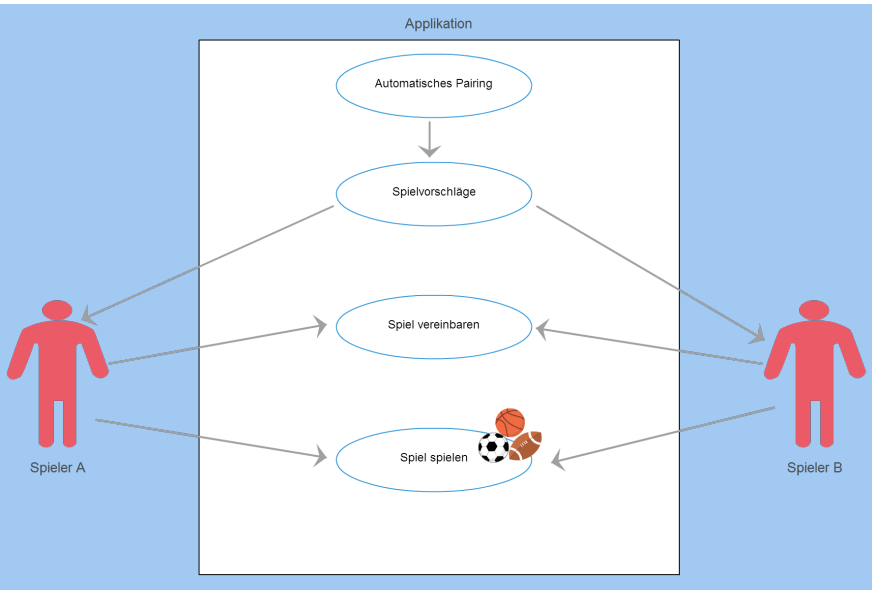
2.3.5.6 UC 4.3 -Rangliste durch Spiele aktualisieren

UC4.2	Rangliste durch Spiele aktualisieren
Beschreibung	Nachdem ein Spiel gespielt ist, können beide Spieler ihre Scores eintragen. Stimmt diese überein oder bestätigt ein Spieler der eingetragene Score des anderen Spielers, wird dieser zur Rangliste hinzugerechnet.)
Diagramm	
Version	1.0
Vorbedingung	UC4.1 - Liga erstellen
Anforderungen	REQ 4.4 - Spiele zu Liga zuordnen REQ 4.5 - Aktualisierung der Ranglist bei gespielten Spielen
Testfälle	Test1.1: Test Test 1.2: Test2
Szenarien	

Alternative Szenarien



2.3.5.7 Spiele automatisch anfordern

UC4.3	Spiele automatisch anfordern
Beschreibung	Bei der Erstellung der Liga kann einen Algorithmus zur automatischen Spielvereinbarung ausgewählt werden. Dieser Algorithmus schlägt jedem Spieler nach der definierten Regelmässigkeit Terminvorschläge vor und stellt so sicher, das alle Spieler regelmässig gegen alle andere Spieler spielen. Die Spieler können sich auf einen der vorgeschlagenen Termine - oder ein anderer individuell vorgeschlagener Termin festlegen.)
Diagramm	
Version	1.0
Vorbedingung	UC4.1 - Liga erstellen
Anforderungen	REQ 4.6 - Automatische Spielaufforderungen
Testfälle	Test1.1: Test Test 1.2: Test2

2.3.6 Anforderungen

Notwendigkeit und Kritikalität

2.3.6.1 Funktionale Anforderungen

2.3.6.2 Nicht funtionale Anforderungen

Die Anforderung in diesem Abschnitt werden direkt in User Stories übertragen in das Tool Yodiz: `https://app.yodiz.com/plan/pages/task-board.vz?cid=14274&pid=1&iid=1`

2.3.6.3 REQ1.1.x - Zuteilung zu Regionen

Der User muss zu einer Region zugeordnet werden können. Regionen sind im Racketsport Zentren, die der Spieler vorzugsweise benutzt. Geographische nähen sind in diesem Use Case viel weniger präzise, da je nach Verkehrsanbindung oder Lebensumstände nicht Racketsport Zentren in der unmittelbaren Geografischen nähe referenziert werden.
REQ1.1.1 - Racketsport Zentren registrieren: Der User der Applikation kann Racketsport Zentren registrieren.
REQ1.1.2 - Zuordnung zu Racketsport Zentren: Der User kann sich zu Racketsport Zentren zuordnen und so anderen Spielern signalisieren, dass er da bevorzugt Spielt.
REQ1.1.3 - Suche nach Spielern per Racketsport Zentrum: User können andere Spieler suchen die sich an ein Racketsport Zentrum zugeordnet haben.

2.3.6.4 REQ1.2 - Broadcast Herausforderung für Region

Der Spieler kann kann für eine Region einen Broadcast Spielvorschlag versenden für eine bestimmte Zeit.

2.3.6.5 REQ1.4 - Details zur Spielerselektion

Der Spieler soll von anderen Spielern verschiedene Merkmale herauslesen können:

- Username
- Spielstärke
- Freiwillige Beschreibung
- Absichten (Regelmässige Spielpartner, Gelegenheitspartner, Spass spiele, Training, ...)
- Sportzeiten (z.b. nur abends, flexibel, nur Montags - Mittwoch)

2.3.6.6 REQ1.5 - Friend System

User können andere User zu ihrer Freundesliste hinzufügen. Privatsphäre Einstellungen können anschliessend drauf abgestimmt werden.

2.3.7 Anforderungen für UC2

2.3.7.1 REQ2.1 - Spiel erstellen

Der User kann ein Spiel erstellen. Darin wird der Gegner, die Spielart sowie Terminvorschläge definiert.

2.3.7.2 REQ2.2 - Spielterminvorschläge erstellen

Der User kann mehrere Terminvorschläge für ein Spiel registrieren (Terminvorschlag besteht aus einer Lokation und einem Datum/Uhrzeit). Diese Terminvorschläge werden einem Spiel zugeordnet.

2.3.7.3 REQ2.3 - Spielterminvorschläge annehmen

Der Empfänger der Terminvorschläge kann einen Spielterminvorschlag annehmen.

2.3.7.4 REQ2.4 - Spielterminvorschläge ablehnen

Der Empfänger der Terminvorschläge kann alle Vorschläge ablehnen und neue Vorschläge erstellen.

2.3.7.5 REQ2.5 - Regelmässige Spiele erstellen

Der User kann ein regelmässiges Spiel mit einem Gegner registrieren. Beide User können Vorschläge zur Abweichung einreichen

2.3.8 Anforderungen für UC3

2.3.8.1 REQ3.1 - Ergebnisse des Spiels eintragen

Der User kann das erzielte Ergebnis des Spiels eintragen. Informationen über Sätze sowie Punkte werden in der App für das jeweilige Spiel eingetragen.

2.3.8.2 REQ3.2 - Bestätigung der Ergebnisse

Der Gegner kann das eingetragene Ergebnis (REQ3.1) bestätigen oder korrigieren. Bei Korrektur wird das Ergebnis dem originalen User zur Bestätigung zurückgesendet.

2.3.9 Anforderungen für UC4

2.3.9.1 REQ4.1 - Liga erstellen

Der User kann eine neue Liga erstellen. Die Liga kann öffentlich (jeder kann sich einschreiben) oder privat (Einschreibung nur per Einladung möglich) sein. Bei der Erstellung der Liga wird eine Rangliste erstellt. Die Rangliste beinhaltet alle User, welche der Liga zugeordnet sind sowie Punkte.

2.3.9.2 REQ4.2 - User zur Liga einladen

Der User kann einen oder mehrere andere User zur Liga einladen. Diese User können die Einladung anschiessend annehmen oder ablehnen.

2.3.9.3 REQ4.3 - User kann Liga beitreten

Der User kann eine Übersicht der Liga abrufen und einer gewünschten Liga beitreten.

2.3.9.4 REQ4.4 - Spiele zu Liga zuordnen

Der User kann ein Spiel- bevor gespielt wird - einer Liga zuordnen.

2.3.9.5 REQ4.5 - Aktualisierung er Rangliste bei gespieltem Spiel

Bei einem beendeten Spiel wird die Rangliste der Liga neu berechnet um die korrekte Punktezahl zu erhalten.

2.3.9.6 REQ4.6 - Automatisch Spielaufforderungen

In einem bestimmten Turnus werden alle Spieler aufgefordert ein Spiel mit einem anderen Spieler aus der Liga zu Spielen

3 Konzeption

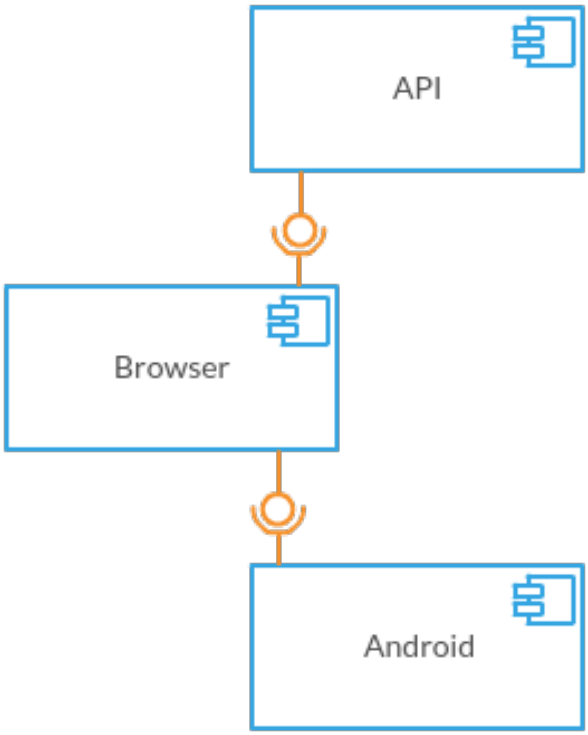


Abbildung 3.1: Grobkonzept für Applikation

Die Applikation besteht aus drei Teilen. Einem Webserver, der eine API und statische Clientfiles zur Verfügung stellt, der Client im Browser, welcher die API konsumiert sowie eine Android Applikation welche die Website lädt.

3.1 Technologiestack

Wie in dem Grobkonzept beschrieben, wird für die Applikation einen Technologiestack gebraucht, welcher eine skalierbare API sowie eine gute Integration der API mit einer Browser Frontend Anwendung bietet. Folgende Anforderungen werden an den Technologiestack gestellt:

- Skalierbare REST API

- Einfacher und schneller Umgang mit AJAX
- Gute Integration zwischen API und HTML/JS Client
- Responsive Design, Integration mit OAUTH für 3rd Party Authentisierung
- Persistance Layer (Datenbankunterstützung)

3.1.1 MVC Frameworks

Komplexe Applikationen werden vorzugsweise mit MVC Frameworks erstellt. Über MVC APIs sind Routing, Logik sowie Präsentationsschicht gut voneinander Abstrahiert. Es ist gut möglich, für Views mit verschiedenen Daten über ein Model zu versehen, oder auch ein Logisches Routing für eine API zu entwickeln. Folgende Graphik zeigt ein MVC Konzept. Der Client sendet einen Request zu dem Server und wird vom Routing zur Logik im System weitergeleitet. Die Logik findet das richtige Model sowie die dazugehörige View. Die View wird mit dem Model gerendert und es entsteht eine Antwort, welche dem Client zurückgesendet wird. In einer REST API ist die View JSON. Das Model wird in JSON umgewandelt und versendet.

Folgendes Code-Beispiel - die Funktion list() - zeigt, wie alle Courts aus dem Persistance Layer selektiert werden und per JSON zum Client gesendet werden. Dies ist ein API Endpunkt zur auflistung von Courts (http://webserver/courts). Gut zu sehen ist, dass die jsonp() Funktion als Renderer gebraucht wird anstatt eine Standard-View.

```
exports.list = function(req, res) {
        Court.find().sort('-created').populate('user')
          .exec(function(err, courts) {
                  if (err) {
                          return res.status(400).send({
                                  message:
                                   errorHandler.getErrorMessage(err)
                          });
                  } else {
                  res.jsonp(courts);
                  }
        });
};
```

Mit einem MVC Framework auf der Server-Seite kann man so gut abstrahierte und ausbaufähige - skalierbare - APIs erstellen. Diese APIs müssen nun jedoch vom Client Browser verarbeitet werden können. Folgende Möglichkeiten bieten sich an:

- Parallel zu der REST API werden Renderer gebaut, welche das Model mit einer View in eine - für den Client statische - Website rendern. Der Client verfügt hier ausschliesslich Logik um die verschiedenen Webseiten abzurufen.

- Ein Website-Skelett mit Logik wird beim ersten Aufruf an den Client verschickt. Der Client bezieht nun Daten aus der REST API und reichert die schon vorhandenen Views mit den Objekten - gesendet über AJAX - selber an.

Eine parallele Implementierung zur Rest API geht entgegen dem Basis Konzept, dass alle Applikationen so gut wie möglich von der REST API profitieren. Zusätzlich würde bei einer parallelen Implementierung jeder Klick in einer Aktualisierung der Applikation resultieren. Dies ist unerwünscht, da sich die Website nicht schnell und intuitiv anfühlt. Man hat bei jedem Klick eine Downtime, da viel Daten übertragen werden müssen, und der Browser den DOM jedes mal neu Aufbauen muss. Bei der zweiten Option wird die Website nur einmal heruntergeladen. Der DOM wird nach dem Download aufgebaut und von der Logik verändert. Klicks lösen einen viel geringeren Aufwand von Server bis Client aus und somit ist die Downtime viel kleiner. Die Applikation fühlt sich schneller und intuitiver an.

Wie bei dem Server, kann man auch bei der Applikation ein MVC Pattern implementieren. Ein Routing definiert, bei welchr URL welche View aufgerufen wird. Bei dem Aufruf einer View ist ein Controller hinterlegt, welcher bei der API das Model und Objekt besorgt. Die View rendert die vom Controller generierten Daten

„Figure Angular JS “

Server und Client MVCs können so miteinander Kombiniert werden und es entsteht eine skalierbare und wartbare Applikationsumgebung.

3.1.1.1 Server MVCs

MVCs für den Server gibt es verschiedene:

- Spring Framework - Java
- ExpressJS - JSON
- Rails - Ruby

3.1.1.2 Client MVCs

MVC für den Server sind ausschliesslich in Javascript geschrieben:

- AngularJS
- ???

3.1.1.3 Stacks

Eine Konfiguration des Client- sowie Server MVCs, damit beide gut miteinander Funktionieren ist zusätzlich wichtig. In der Evaluation wird somit folgende Konfigurationen abgewogen:

- Spring Framework mit AngularJS
- Express Framework mit AngularJS

3.1.2 Spring + AngularJS

Spring ist ein MVC Framework in Java. Man Programmiert in der J2EE Umgebung und bietet eine API zum Client. Gleichzeitig sendet man den AngularJS Stack zum Client, welcher anschliessend die API konsumiert. Als Persistance Layer können Relationale Datenbanken wie MySQL, Oracle oder Sybase verwendet werden. Über Data Access Object wird dieser Layer angesprochen und in Models Emuliert.

Folgende Vor- / Nachteile bring das Spring-Angular Setup mit:

- + reife Technologien, Markt erprobt
- + viel Know-How auf dem Markt in Java/Spring
- + Anbindung an Zahlreiche Persistance Layer Applikationen
- - Viele Daten-Transformationen (Relationale DB ¡-¿ Java Objekt ¡-¿ JSON Object)
- - Kompliziertes Setup

3.1.3 MEAN Stack

Der MEAM Stack besteht aus folgender Produkten:

- M - MongoDB, der Skalierbare Persistance Layer
- E - ExpressJS, ein MVC um APIs zu entwickeln
- A - AngularJS, ein MVC auf dem Client um Sing-Page Applikationen zu erstellen, welche auf die ExpressJS API zugreifen.
- N - NodeJS, JavaScript Applikationsserver, welcher sehr gut skalierbar ist.

Folgende Vor- / Nachteile bring das MEAN Stack Setup mit:

- + Kleine Daten Transformationen (JSON Object wird in MongoDB gespeichert)
- + Gleiche Programmiersprachen (JavaScript, HTML, CSS)
- + Einfaches Setup, grosse Flexibilitäten
- + Innovative Technologien
- - Wenig Markt erprobt

3.1.4 Evaluation

Da das Ergebnis dieser Arbeit ein Proof of Concept darstellt, und die Applikation nicht ausgereift sein wird zum Zeitpunkt der Abgabe, sowie eine produktiver Nutzen nicht das Ziel dieser Aufgabenstellung ist, will der Autor mit möglichst innovativen, flexiblen und einfachen Technologien arbeiten. Der MEAN Stack scheint dadurch der optimale Kandidat für dieses Projekt.

3.2 Architektur

3.2.1 Kontext

Die Applikation ist aufgeteilt auf einen Server, sowie auf einen Client, welcher ein Browser oder eine Android App ist. Auf dem Server sind alle Daten hinterlegt:

- Gespeicherte Objekte in MongoDB
- Server Logik
- Client Daten, welche vom Browser über HTTP abgefragt werden

Im Anfangszustand hat der Client keine Daten. Der Client bekommt die Daten bei dem Abruf der Applikations URL über HTTP. Er baut nun die Logik im Browser Cache auf und startet das JavaScript Programm. Das JavaScript Programm lädt nun die auf dem Server gespeicherten Objekte über HTTP AJAX Abrufe und stellt diese dar.

Die Logik von Server wie auch Client benutzt das M(V)C Pattern. Objecte werden in Models - inklusive Business Logik - gespeichert, der Controller beinhaltet die Applikationslogik, welche das Model sowie die View auswählt. Die View rendert nun das Model in ein bestimmtes Schema (siehe Abbildung 3.2).

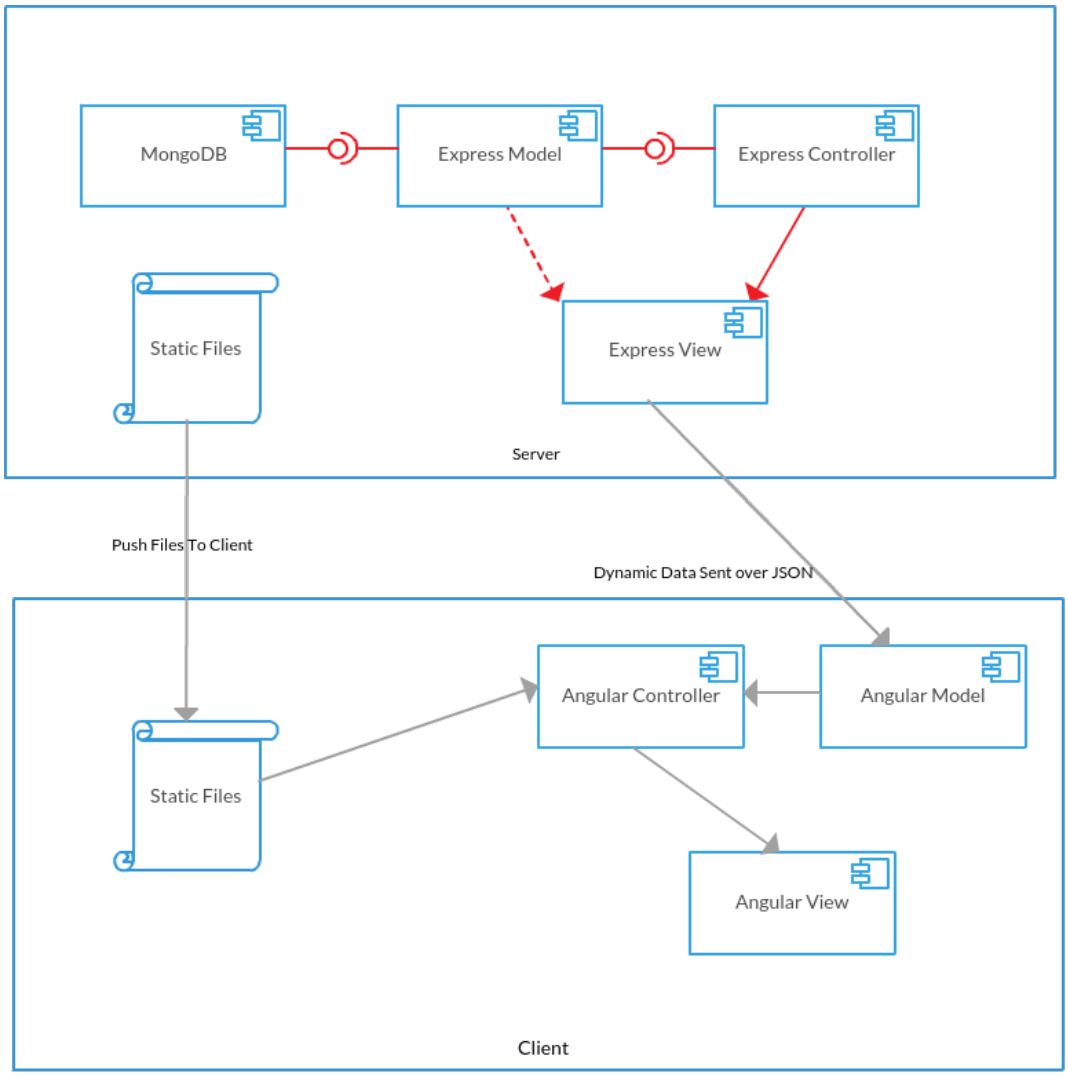


Abbildung 3.2: Detaillierte Applikations Architektur

3.2.1.1 Continues Integration

3.2.2 Businessschicht

3.2.2.1 DB Design

In der Applikation gibt es kein Relationales Datenbankmodel. MongoDB arbeiet mit Dokumenten, sowie Referenzen. Dokumente sind JSON-Objekte in JavasScript, welche in MongoDB als Dokument gespeichert werden. Ein Objekt ist eine Representation von Business Objekten in der Applikation.

Das User-Objekt repräsentiert der User der Applikation. Der User hat einen Namen, einen Usernamen, ein Passwort (encrypted und salted), Berechtigungen und Freunde. Zusätzlich werden Ihm andere Objekte zugeordnet sowie andere User (Repräsentation als Freund).

Das Court-Objekt repräsentiert ein Racketsportzentrum der Applikation. Dieses Objekt wird benötigt um den Physikalischen Austragungsort eines Spieles zu definieren. Das Objekt beinhaltet einen Namen, eine Adresse (inklusive Koordinaten für eine zukünftige Umkreissuche), was für Sportarten gespielt werden können und welche User in diesem Racketsportzentrum spielen wollen.

Das Match-Objekt respäsentiert das Spiel, welches geplant, ausgetragen oder beendet ist. Das Spiel-Modell definiert zwei oder einen Spieler, einen Status, eine Sportart, ein Court, mehrere Datumvorschläge, maximal fixes Datum, eine Punkzahl, sowie ein Gewinner. Hinter dem Match-Objekt existiert ein relativ grosser Business-Workflow, welcher im Kapitel ¡¡¡¡IMPLEMENTATION Matchmaking¿¿¿¿ definiert ist.

Das Liga-Objekt repräsntiert eine Liga. Verschiedene Benutzer können einer Liga beitereten und sind nach Beitritt bestimmten regeln unterworfen. Dafür können die Benutzer spiele für die Liga spielen und so Punkte für einen optionalen Preis sammeln. Die Liga beinhaltet neben einem Namen, einer Sportart, einem Standort (inklusive Koordinaten, für zukünftige Umkreissuche), einem Niveau, Start- und Enddatum, einem Preis und einem Matchmaking Plan (wird später im Dokument erläuert¿¿¿¿¿¿¿REF).

Folgende Grafik zeigt die Beziehung der Verschiedenen Schemas auf, das Datenbankmodell ist nicht Relational, und somit nicht normalisiert.

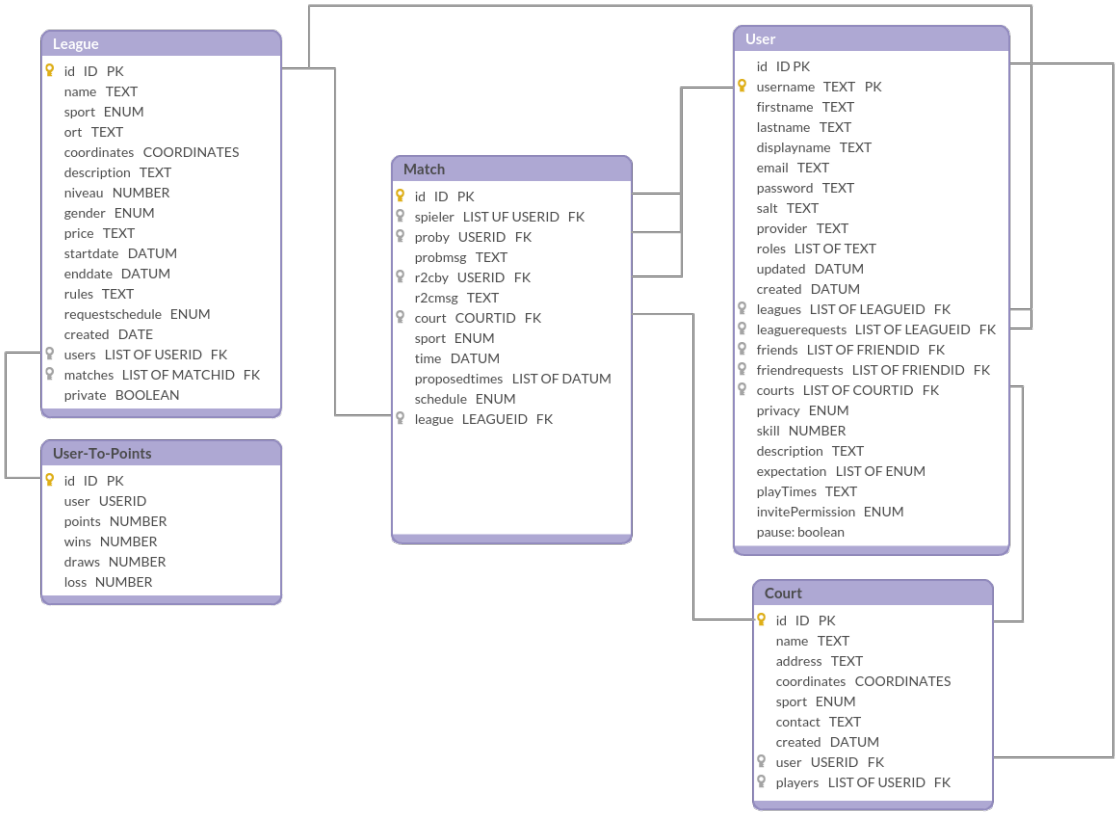


Abbildung 3.3: Datenbank Modell

3.2.3 Präsentationsschicht

Als GUI wird ein standard Bootstrap Design verwendet. Ohne Authentisierung kann nur die Home-Page gesehen werden sowie die Login und Signup Page. Für alle anderen Seiten muss der Benutzer authentisiert sein. Sobald die Authentisierung durchgeführt wurde, gibt es können die Elemente (Nach Datenbank) Benutzer, Liga, Racketsportzentrum sowie Spiele selektiert werden. Innerhalb der einzelnen Menu kann man verschiedene Operationen direkt ansteuern, einige nur über andere Operationen. Folgendes Diagram zeigt die Interaktion durch die verschiedenen Views.

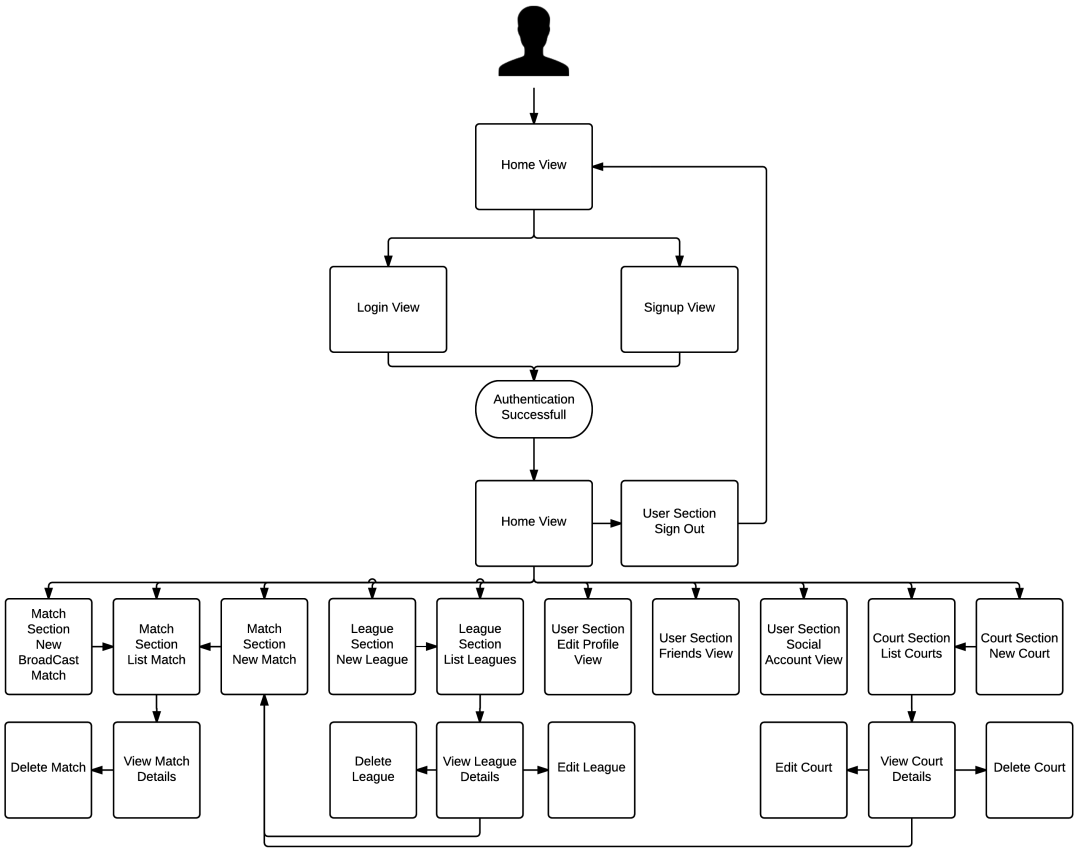


Abbildung 3.4: GUI Interaktions Modell

3.2.3.1 User Section

Die User Section einhaltet drei Views die direkt aus dem Menu erreichbar sind. Die erste View User Profile ermöglicht dem User, Details über sich preiszugeben. Er kann zusätzlich das Passwort ändern. Inder Friends view kann er neue Friendrequests erstellen, und pendente Friendrequests annehmen oder ablehnen. Die Social Account View bietet eine Verknüpfung von Social Accounts mit der Applikation an.

3.2.3.2 Court Section

Die Court Section beinhaltet vier Views sowie eine Aktion. In der new Court View kann ein neues Racketsportzentrum registriert werden. In der List Courts view findet man alle Racketsportzentren und erreicht bei klick auf ein Zentrum die View Court Details View. In dieser View kann man alle Details des Racketsportzentrum anschauen, sowie alle Spieler, welche in diesem Racketsportzentrum spielen. Durch klick auf den Spieler kann in die New Match View gewechselt werden, um einen Spieler herauszufordern. Von der Detail View kann man zusätzlich das Court löschen, sofern man das Court erstellt hat oder ein Admin ist.

3.2.3.3 League Section

Die League Section beinhaltet vier Views sowie eine Aktion. In der New League View kann ein neues Liga registriert werden. In der List League View findet man alle Ligen und erreicht bei klick auf eine Liga die View League Details View. In dieser View kann man alle Details die Liga anschauen, sowie alle Spieler, welche in dieser Liga spielen. Durch klick auf den Spieler kann in die New Match View gewechselt werden, um einen Spieler herauszufordern. Von der Detail View kann man zusätzlich die Liga löschen, sofern man die Liga erstellt hat oder ein Admin ist.

3.2.3.4 Match Section

Die Match Section beinhaltet alle Interaktionen im Match. Drei Views sind direkt aus dem Menu erreichbar. Auf der New Match View kann man ein neues Spiel erstellen. Man kann Court, Spieler in einem Formular auswählen. Über den Menupunkt New Broadcast Match View, wählt man ein Court sowie eine Zeit und alle Spieler, welche in diesem Court spielen werden angefragt für eine spontanes Spiel. In der List Match View werden alle Spiele aufgelistet. Von da kommt man in die View Match Details View, welche den Matchworkflow abdeckt.

3.2.4 Sicherheitssicht

4 Implementation

4.1 REST API

Die REST API besteht aus fünf Endpunkten:

- /matches - Stellt alle Operationen für Matches zur Verfügung
- /leagues - Stellt alle Operationen für Liga Management zur Verfügung
- /courts - Stellt alle Operationen für die Verwaltung von Racketsportzentren zur Verfügung
- /users - Stellt alle Operationen für das Usermanagemnt zur Verfügung
- /core - Stellt Core-Funktionalitäten (Home Seite) zur Verfügung

Die Endpunkte /users und /core waren im MEANJS Stack schon vorhanden. Der User Endpunkt wurde jedoch modifiziert. Die Modifizierungen sind in dem Kapitel dokumentiert, die schon vorhandenen Endpunkte nicht.

4.1.1 Routing

4.1.2 Spiel Endpunkt /matches

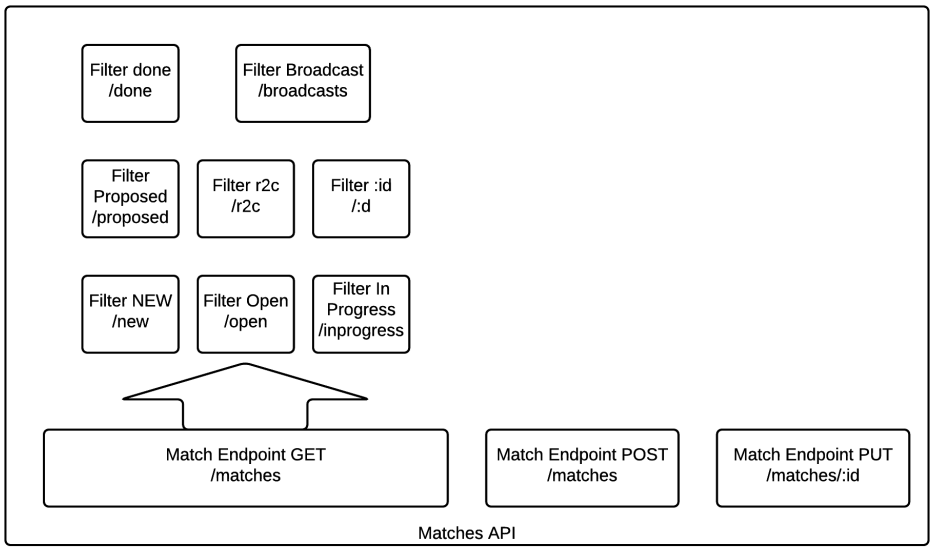


Abbildung 4.1: API für Spiele

Bei allen Match Endpunkten muss der User als Spieler registriert sein, um Informationen über das Spiel zu erhalten. Ausnahme ist, wenn er direkt die ID eingibt und direkt auf Spiel Detauls zugreift.

Mit einem Post fügt man der Datenbank ein Spiel hinzu, mit PUT aktualisiert man das Spiel mit neuen oder geänderten Daten. Hinter dem PUT interface gibt es gewisse Input Validations um Missbrauch zu verhindern. In dieser ersten Version sin ddie Validations jedoch relativ einfach gehalten.

Um eine gute Übersicht aller Spiele auf der List Matches View zu erstelllen, gibt es für jeden Status eines Spieles einen eigenen API Call (/matches/new, /matches/open, /matches/inprogress, /matches/proposed, /matches/r2c, /matches/done).

Der API Endpunkt /matches/broadcast listet zusätzlich alle broadcasting Anfragen auf. Der Conntroller des Enpunkt Korreliert, in welchen Racketsportzentren der User registriert ist und die Matches ohne zweiten Spieler und gibt das Resultat dem Client.

4.1.2.1 Codebeispiel - Filtering

4.1.3 Liga Endpunkt /leagues

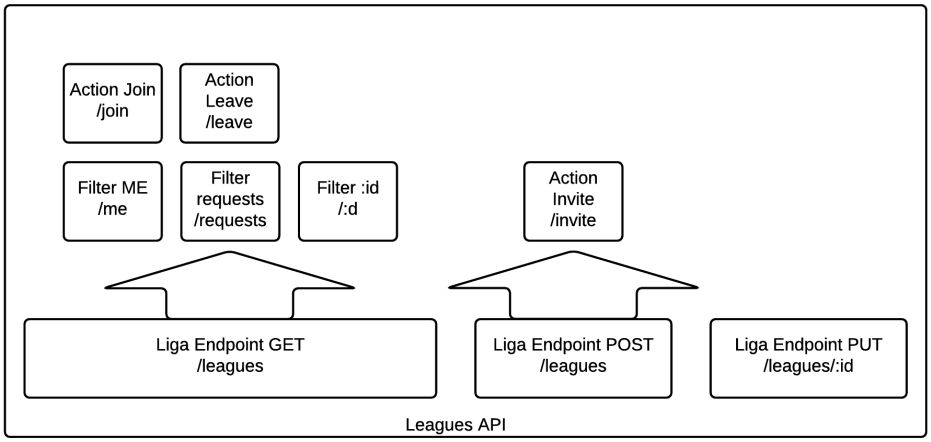


Abbildung 4.2: API für Ligen

Bei der Liga gibt es - wie bei allen Endpunkten - CRUD Endpunkte (/leagues für list all, /leagues/:id für Show Element, POST /leagues für Create League, PUT /leagues/:id für Update League). Zusätzlich gbit es eine Join Action, welche den authentisierten User einer Liga hinzufügt sowie ein Leave Endpunkt um die Registriertung zu löschen.

Ein zusätzlicher Endpunkt ist /leagues/invite, welcher ermöglicht einen User zu einer Liga einzuladen.

4.1.3.1 Codebeispiel - Join/Leave Funktionen

4.1.4 Racketsportzentrum Endpunkt /courts

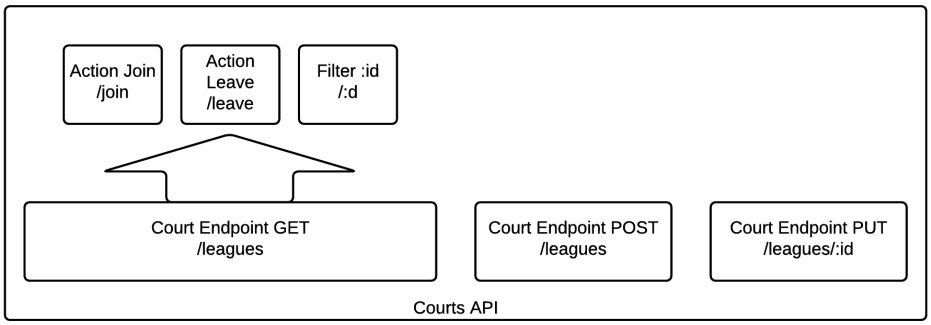


Abbildung 4.3: API für Racketsportzentren

Gleich wie beim Liga Endpunkt gibt es die CRUD Endpunkte sowie ein Join/Leave Endpunkt

4.1.5 Benutzer Endpunkt /users

Neben den üblichen Benutzerverwaltungs Endpunkten (/user/signin, /user/signout, /user signup, /auth/forgot), welche hier nicht Dokumentiert werden, gibt es Endpunkte für das Freunde-System:

- GET /users/friend - Auflistung aller Freunde
- DELETE /users/friend - Löschen eines Freundes
- GET /users/request - Senden eines Freund Requests
- DELETE /users/request - Löschen eines Freund Requests

4.1.5.1 Codebeispiel - OAuth Integration

4.2 Web Applikation

4.2.1 BootStrap/AngularJS

4.2.2 Google Maps Integration

4.3 Android Applikation

Als Grundlage für die Android Applikation wurde eine Applikation von gonative.io generiert. Im Laufe des Projektes - nach erheblicher Überschreitung des vorgeschriebenen Aufwandes - wurde entschieden keine vollständig Native Webapplikation zu erstellen. Stattdessen wird

eine WebView erstellt, welche die Mobile Webseite darstellt. Um alle Funktionalität zu behalten, wird über die WebView und Interception Algorithmen Push-Nachriten ermöglicht. Der einzige Setback ist, das die Website offline nicht verfügbar ist.

4.3.0.1 Codebeispiel - Push Interception

4.4 Workflows

4.5 Allgemeine Workflows

4.5.1 CRUD für Datenobjekte

Alle Datenobjekte haben einen Endpunkt. Jeder Endpunkt stellt CRUD Operationen zur Verfügung:

- C - Neues Objekt erstellen
- R - Ein Objekt anzeigen
- U - Ein Objekt aktualisieren
- D - Ein Objekt löschen

Zusätzlich wird noch einen Endpunkt zur Auflistung aller Objekte angeboten.

4.6 Court

4.6.1 Court Registrierung

Wenn der User den Knopf im User Interface zur Registrierung das Racketsportzentrums drückt, wird im Hintergrund der /courts/join API Call ausgeführt. Dieser Call fügt der User der Anfrage in ein Array - bestehend aus allen registrierten Usern - ein.

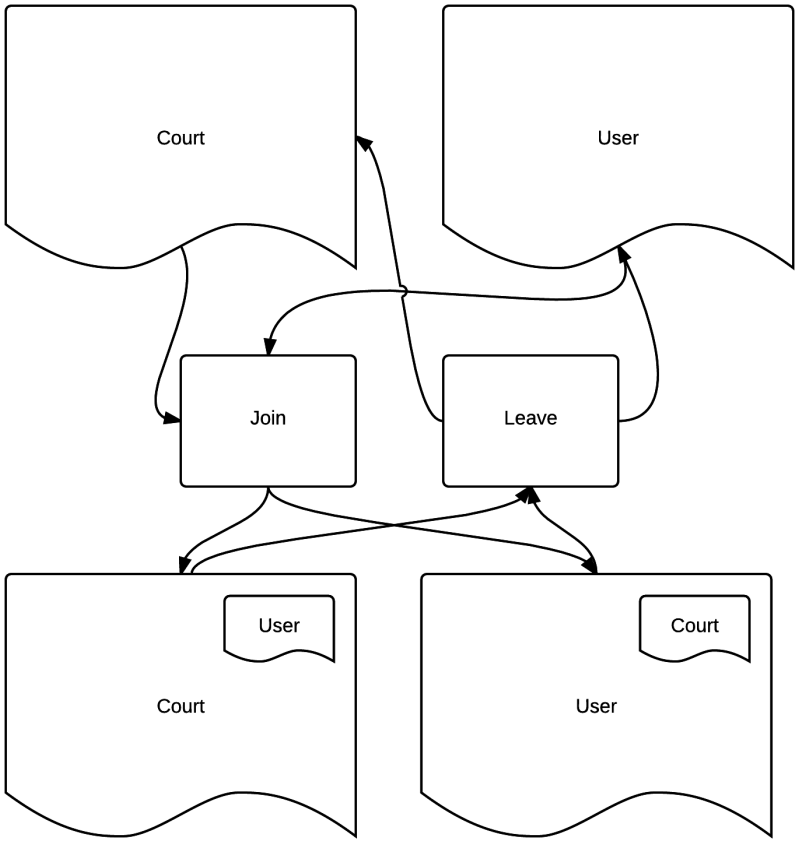


Abbildung 4.4: Racketsportzentrum Workflow

4.7 Liga

4.7.1 Liga Registrierung

Identisch zu der Court Registrierung funktioniert die Liga Registrierung

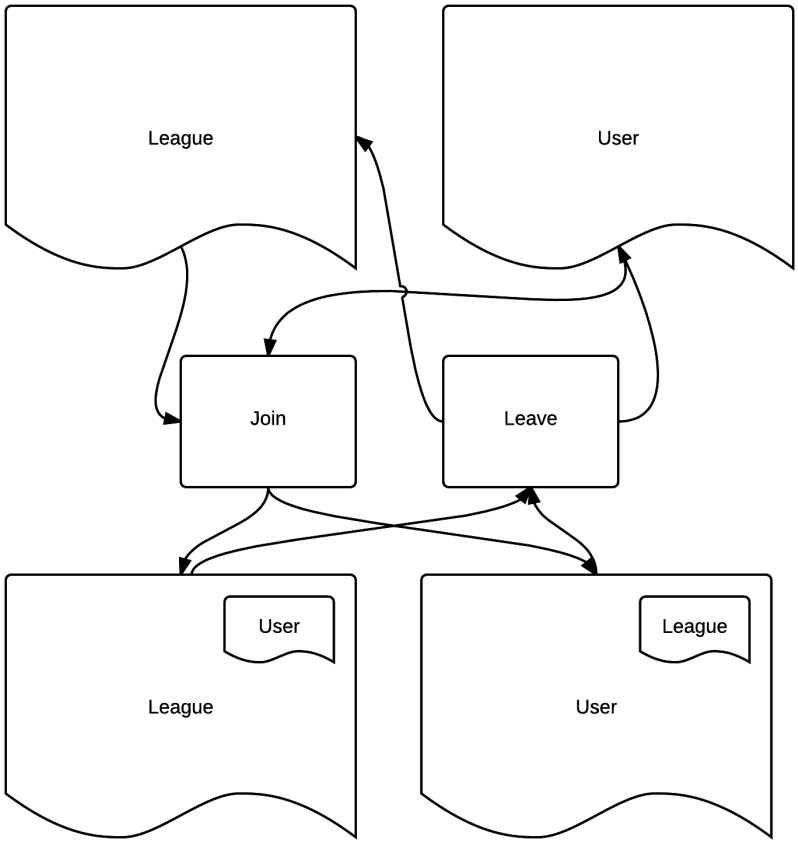


Abbildung 4.5: Liga Workflow

4.7.2 Automatische Herausforderung Liga

Bei der erstellung einer Liga kann ausgewählt werden ob automatische Herausforderungen aktiviert werden sollten. Aktuell gibt es vier verschiedene auswählbare Modi:

- Weeklyall: Wöchentliche Herausforderung, jeder gegen jeder, zufälliger Gegner
- Biweeklyall: Herausforderung alle zwei Wochen, jeder gegen jeder, zufälliger Gegner
- Weekly

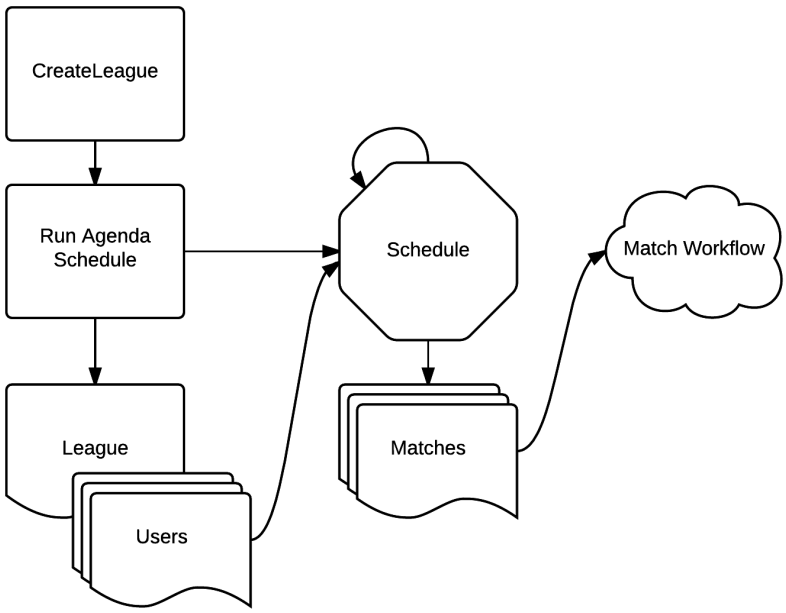


Abbildung 4.6: Schedule Workflow

4.7.2.1 Codebeispiel - Scheduling

4.7.2.2 Random Herausforderungen / Rang Herausforderungen

4.8 Match

4.8.1 Match Workflow

Der Matchworkflow ist das Hauptelement der Applikation. Der Workflow regelt, wie der Match als Business Prozess durchgeführt wird.

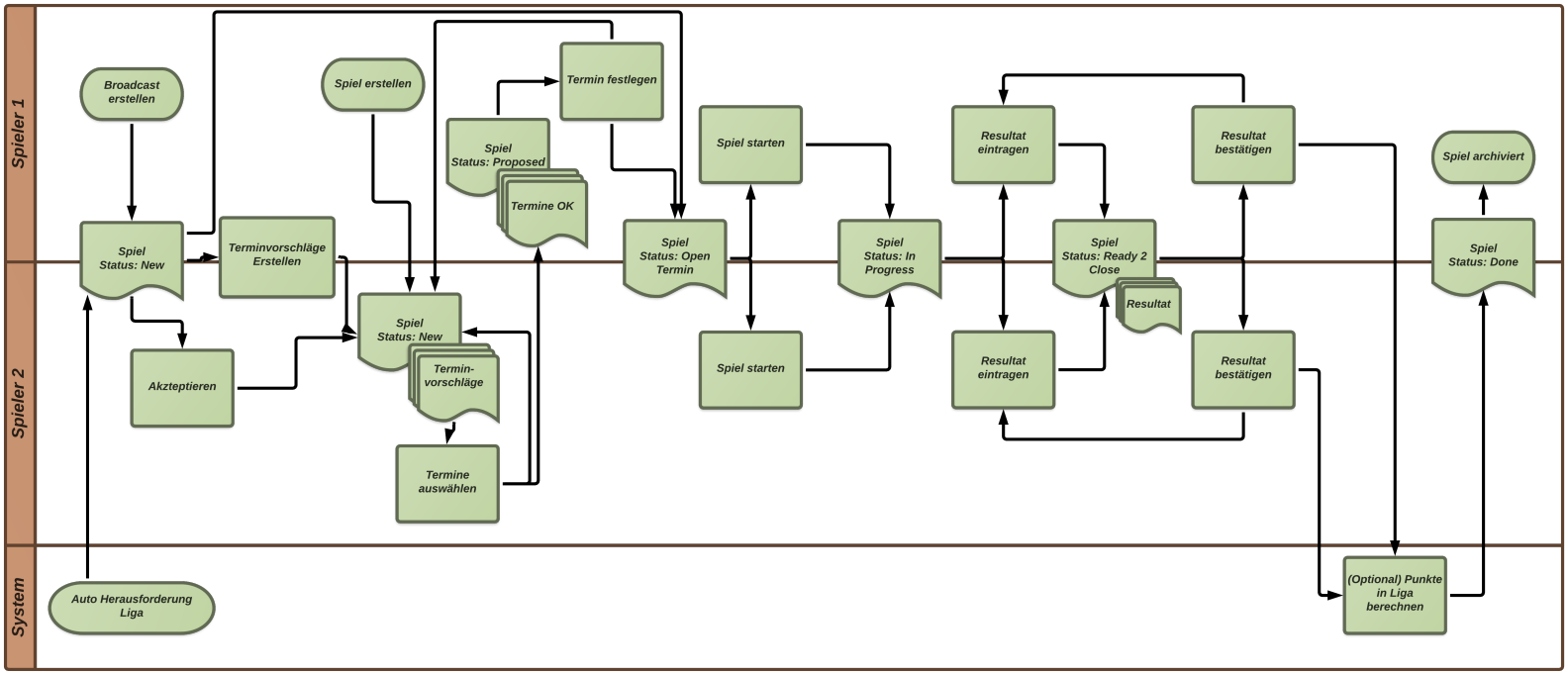


Abbildung 4.7: Spiel Workflow

Der Workflow wird über drei verschiedene Fälle gestartet:

- Das System erstellt Auto-Herausforderungen für die Liga
- Der User erstellt ein Broadcast Spiel
- Der User erstellt ein reguläres spiel.

Wenn der User ein **reguläres Spiel** erstellt, sind beide Spieler, sowie Terminvorschläge schon definiert. Es folgt die Aktion "Termin auswählen".

Erstellt der User ein **broadcast Spiel**, hat das Spiel den Status "New", jedoch noch keinen zweiten Spieler definiert. Zusätzlich werden keine Terminvorschläge ausgefüllt, sondern einen fixen Termin. Akzeptiert jemand den Broadcast wird der zweite Spieler eingetragen und der Status ändert sich direkt auf Open.

Sind User in einer Liga, erstellt die **Liga eine Herausforderung**. Das Spiel enthält kein Court und keine Terminvorschläge. Der User muss nun Terminvorschläge ausfüllen und ein Court definieren.

Anschliessend haben alle Use Cases den gleichen Workflow. Ist das Spiel und der Termin definiert. Geht der Status des Spiels zu "Open". Danach kann von beiden Spielern der Status auf "In progress"gesetzt werden. Beide können ein Resultat eintragen. Der jeweil andere Spieler bestätigt anschliessend das Resultat. Bei der Bestätigung des Resultats wird das Spiel archiviert und optional die Rangliste der Liga aktualisiert.

4.9 User

4.9.1 Freunde System

5 Test

5.1 Unit Tests

5.2 System Tests

5.3 User Acceptance Tests

6 Reflektion

Abbildungsverzeichnis