LUDUR

MARIO MEILI, SEBASTIAN BOCK

Eine iOS Spieleplattform mit iBeacon
Bachelorarbeit, Frühjahrssemester 2015

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

BETREUER:
Prof. Dr. Josef M. Joller

EXPERTE:
Dipl. Ing. Matthias Lips

UNIVERSITÄT:
HSR Hochschule für Technik Rapperswil

ABTEILUNG:
Abteilung Informatik

INSTITUT:
ITA Institut für Internet-Technologien und -Anwendungen

ORT:
Rapperswil

SEMESTER:
Frühjahrssemester 2015

KURZFASSUNG

Spiele für mobile Geräte können meist nur gegen eine künstliche Intelligenz oder ausgewählte Freunde gespielt werden. Das ist bedauerlich, weil jeder Träger eines Smartphones ein potentieller Gegenspieler ist. Mit den Technologien, die in den modernen Smartphones verbaut werden, sollte es doch möglich sein, eine Verbindung zwischen Geräten in einem bestimmten Radius herzustellen. So könnten auch bisher unbekannte Personen in einem Spiel gegeneinander antreten.

Die Studienarbeit "Ludur, ein iOS Spiel mit iBeacons"[1] beweist die Realisierbarkeit einer neuartigen Spieleplattform, die das Auffinden von Gegenspielern mit Hilfe der iBeacon-Technologie erlaubt. Es wurde ein Prototyp in Form einer App für iOS8 erarbeitet, der ebendieses Verhalten erzielt. Dieser Prototyp ist aber noch arm an Funktionalität und das GUI entspricht nicht den von Apple gesetzten Standards.

Das Ziel dieser Arbeit ist es deshalb, diesen Prototyp zu überarbeiten und um Funktionalität zu erweitern. Das beinhaltet ein komplettes Refactoring des geschriebenen Codes, die Erweiterung der Service-Logik und eine komplette Überarbeitung des GUI-Designs. Ausserdem soll die App um eine ganze Reihe neuer Minigames erweitert werden, die die verschiedenen Sensoren des Gerätes ausreizen.

Das Ergebnis ist eine in sich geschlossene und ansprechende iOS App, die bereits dem Review von Apple für das Beta-Testing standgehalten hat. Spieler finden sich gegenseitig sofort und können sich in kürzester Zeit zu einem Spiel herausfordern.

Wird die App von den Beta-Testern positiv aufgenommen, soll sie im offiziellen Apple App Store veröffentlicht werden.

1 http://eprints.hsr.ch/405/1/Ludur%20-%20ein%20iOS%20Spiel%20mit%20iBeacons.pdf[1]

DANKSAGUNG

Als erstes möchten wir uns bei Prof. Dr. Josef M. Joller bedanken, der es uns ermöglichte, diese Bachelorarbeit zu machen und uns während der Durchführung jederzeit unterstützte.

Ein weiteres Dankeschön geht an Dipl. Ing. Matthias Lips, der uns als Experte bei der Zielsetzung eine grosse Hilfe war.

Ein spezielles Dankeschön geht an unseren Kollegen Fabio Pigagnelli, der uns bei der Gestaltung des App Logos unter die Arme griff.

Auch möchten wir all unseren Kollegen und Familienmitgliedern danken, die uns bei der Ideenfindung für neue Minispiele unterstützten und immer mal wieder für Versuche hergehalten haben.

Schliesslich möchten wir uns für das *classicthesis* LaTeX-Template ganz herzlich bei Prof. Dr.-Ing. André Miede bedanken. Dieses haben wir zur Erstellung und Bearbeitung dieser Dokumentation verwendet.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

ABKÜRZUNGEN

HSR Hochschule für Technik Rapperswil

iOS Mobiles Betriebssystem von Apple für Smartphones und Tablets

App Kurzform für Applikation, meist verwendet für Applikationen, welche auf dem iOS Betriebssystem laufen

SDK Software Development Kit

GUI Graphical User Interface

DAL Data Access Layer

DB Dezibel

GMT Greenwich Mean Time

G Gravitationskonstante = $9.81 m/s^2$

Teil I

EINFÜHRUNG

1 MOTIVATION

Seit der Erfindung des Smartphones sind Applikationen aus unserem Alltag nicht mehr wegzudenken. Zu fast jeder erdenkbaren Aktivität gibt es Applikationen, die den Benutzer unterstützen, unterhalten und informieren. Immer mehr ist es auch der Fall, dass Applikationen nicht nur für sich stehen, sondern Anwender untereinander verknüpfen. Beispiele dafür sind die grossen Social Media-Vertreter Facebook, Twitter, LinkedIn usw. Je mehr Benutzer eine solche Applikation hat, desto grösser ist der Anreiz, sich ebenfalls anzumelden.
Es ist deshalb auch nicht erstaunlich, dass der App Store von Apple bereits mehr als 1.4 Millionen Applikationen zur Verfügung stellt[1]. Erstaunlicher ist die Tatsache, dass Spiele-Applikationen den grössten Anteil, nämlich 21.45%, des App Stores ausmachen[2].

In der Studienarbeit "Ludur, ein iOS Spiel mit iBeacons“ (im Folgenden als Studienarbeit bezeichnet) wurde gezeigt, dass es möglich ist, mithilfe der iBeacon-Technologie Benutzer einer Applikation darüber zu informieren, dass sie sich in der Nähe des jeweils Anderen befinden. Ein Prototyp, der diese Funktionalität beinhaltet, wurde im Rahmen der Studienarbeit bereits implementiert. Eingesetzt wurde diese Funktionalität in einer Spieleplattform, bei der Spieler eine Reihe von Minigames gegeneinander durchführen können, ohne sich vorher zu kennen.

Die grosse Beliebtheit von Spiele-Applikationen und das Vorhandensein einer Basis für eine solche Applikation motivieren dazu, den in der Studienarbeit entwickelten Prototyp bis zur Marktreife zu überarbeiten.

1.1 SMARTPHONES & IHRE SENSOREN

Ein Trend in der Ausstattung von modernen Smartphones ist die grosse Anzahl unterschiedlicher Sensoren. So enthält das iPhone 6 neben dem herkömmlichen Mikrofon, GPS und der Kamera auch noch einen Gyrosensor, einen Sensor für Beschleunigung und ein Barometer[3]. Apple stellt den Entwicklern von Applikationen Frameworks zur Verfügung, die das Ansprechen dieser Sensoren erlauben. Die

1 http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store[2]
2 http://www.statista.com/statistics/270291/popular-categories-in-the-app-store[3]
3 https://www.apple.com/de/iphone-6/technology[4]

Benutzung dieser Frameworks zu erlernen und die Genauigkeit bzw. Brauchbarkeit der von den Sensoren generierten Daten zu untersuchen, ist nicht nur interessant, sondern regt auch dazu an, neue Use Cases für die vorhandenen Sensoren auszuarbeiten.

1.2 EIN FERTIGES PRODUKT

Eine Applikation in einem kleinen Team zu entwickeln ist eine grosse Herausforderung. Typischerweise erfordert eine moderne Applikation eine komplexe Business-Logik, die sich über verschiedene Tiers verteilt. Hinzu kommt, dass besonders bei iOS-Applikationen die Erwartungen an die grafische Oberfläche sehr hoch sind. Das erfordert von den Teammitgliedern, sich mit allen genannten Aspekten auseinanderzusetzen, unabhängig von den Vorlieben und den Stärken des jeweiligen Entwicklers. Die Entwicklung einer Applikation erfordert deshalb Disziplin, Organisation und vor allem eine funktionierende Zusammenarbeit.

2 VISION

Dieses Kapitel beschreibt die Entstehung der Vision und die festgesetzten Ziele dieser Bachelorarbeit.

2.1 ANALYSE & NEUORIENTIERUNG

Schon bald nach der Abgabe der Studienarbeit wurde klar, dass die Arbeit in einer Bachelorarbeit fortgesetzt werden soll. Der Prototyp bewies zwar bereits die Machbarkeit der angestrebten Spieleplattform, implementierte aber nur grundlegende Funktionalitäten. Erweiterungsmöglichkeiten wurden in der Studienarbeit in einem eigenen Kapitel abgehandelt. Natürlich war es vorerst das Ziel, möglichst viele dieser Erweiterungen umzusetzen.

In Absprache mit Prof. Dr. Josef M. Joller und Dipl. Ing. Matthias Lips wurden die meisten erarbeiteten Erweiterungsmöglichkeiten verworfen und entschieden, sich auf die Verbesserung der Plattform zu fokussieren und weitere Minigames zu entwickeln. Diese Entscheidung wird damit begründet, dass es als sinnvoller erachtet wird, eine kleine, dafür qualitativ akzeptable Plattform mit vielen interessanten Minigames zu entwickeln, als einen feature-reichen Prototyp. Fände die Applikation bei Benutzern Anklang, könnten noch weitere Features step by step integriert werden.

2.2 ZIELE

Die im Abschnitt 2.1 beschriebene Neuorientierung erforderte eine Überarbeitung der in der Studienarbeit definierten Ziele. Nachfolgend sind diese Ziele zusammengefasst.

2.2.1 *Funktionalität*

- Das Resultat der Bachelorarbeit soll eine voll funktionsfähige, marktreife Spieleplattform in Form einer iOS-Applikation sein.
- Es soll möglich sein, sich für die Plattform zu registrieren und die angelegten Userdaten zu verwalten.
- Die Plattform soll es ermöglichen, Spieler in der näheren Umgebung der Benutzer ausfindig zu machen, gegen welche man Spiele durchführen kann.

- Als Belohnung für einen Sieg soll der Benutzer die Spielkontakte seiner Gegner erhalten, gegen welche er dann aus jeder Entfernung antreten kann.
- Die Plattform soll auf iPhones ab Betriebssystem iOS8 funktionsfähig sein.

2.2.2 *Zuverlässigkeit*

- Die Applikation soll ohne reproduzierbare Abstürze bedienbar sein. Dies ist eine Voraussetzung für die Akzeptanz der Benutzer.

2.2.3 *Benutzbarkeit*

- Der Einsteig in die Applikation soll für den Benutzer möglichst einfach gestaltet werden. Innerhalb von 15 Minuten soll ein neuer Benutzer die Grundlagen der Plattform verstanden haben.
- Es soll dem Benutzer möglich sein, einfach und schnell neue Spiele starten und durchführen zu können.
- Die grafische Oberfläche soll die von Apple zur Veröffentlichung geforderten Guidelines erfüllen[1]. Dies erhöht wiederum die Akzeptanz der Benutzer.
- Die grafische Oberfläche soll wenn möglich die neu mit iOS8 eingeführten Komponenten verwenden, um sich nahtlos in das Betriebssystem des Benutzers zu integrieren.

2.2.4 *Wartbarkeit & Änderbarkeit*

- Es soll ein striktes Layering in mehrere Schichten vorgenommen werden, um eine übersichtliche Struktur des Codes zu garantieren und die Testbarkeit des Codes zu gewährleisten.
- Es soll möglich sein, für die Applikation weitere Spiele zu entwickeln und diese mit möglichst wenigen Änderungen im bestehenden Code einzubinden. Dies soll durch eine einfache und klar definierte Schnittstelle realisiert werden.
- Das Backend der Applikation soll stets austauschbar bleiben. Dies soll ebenfalls durch eine einfache und klar definierte Schnittstelle ermöglicht werden.

1 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG[5]

- Die Applikation soll mit einem automatischen Build System erstellt werden. Dabei sollen auch alle Tests ausgeführt und ein Code Coverage-Report generiert werden.
- Fehlerbehebungen und Änderungen für neue Betriebssystemversionen sollen durch ein automatisiertes Continous Delivery System einfach deployed werden.

2.2.5 *Übertragbarkeit*

- Die Installation der Applikation soll über den Apple App Store möglich sein.

3 USE CASES

Dieses Kapitel zeigt die Use Cases auf, die im Rahmen dieser Bachelorarbeit umzusetzen sind. Das zugehörige Use Case Diagramm ist im Appendix im Abschnitt A.1 zu finden.

3.1 CRUD USER

Um die Applikation benutzen zu können, muss sich ein Benutzer vorgängig registrieren. Er muss dabei zwingend die nachfolgenden Informationen geben:

- Benutzername
- Vorname
- Nachname
- E-Mail-Adresse
- Passwort
- Benutzerfoto

Der Benutzername muss eindeutig sein. Er darf demzufolge nicht bereits von einem anderen Benutzer verwendet werden.

Ist der Benutzer eingeloggt, kann er seine Daten editieren. Dies gilt für den Vor- und Nachnamen, das Passwort und das Benutzerfoto. Der Benutzername und die E-Mail-Adresse können für ein angelegtes Profil nicht mehr geändert werden. Das Löschen des Profils ist dem Benutzer nicht erlaubt.

3.2 SCAN FOR OPPONENTS

Hat der Benutzer Bluetooth eingeschaltet und der Applikation die benötigte Autorisierung zur Verwendung von Bluetooth gegeben, kann er über das System den Location-Service beauftragen, seine Umgebung nach potenziellen Gegenspielern zu scannen.

Der Location-Service informiert das System sogleich über gefundene Spieler. Das System zeigt dem Benutzer daraufhin ausgewählte Informationen über diese Spieler an. Befindet sich die Applikation zu dieser Zeit im Hintergrund, wird der Benutzer vom System mittels einer Push-Benachrichtigung informiert.

3.3 CHECK OPPONENT SKILLS

Der Benutzer hat die Möglichkeit, statistische Angaben über die Spielausgänge anderer Spieler einzusehen. Das ist aber nur möglich, wenn der gewünschte Spieler dem Benutzer vom System angezeigt wird. Die Zeitspanne, über welche dem Benutzer Auskunft gegeben wird, ist beschränkt.

3.4 INVITE USER

Der Benutzer kann jedem Spieler, der ihm vom System angezeigt wird, eine Einladung zukommen lassen. Allerdings muss dafür eine ganze Reihe von Bedingungen erfüllt sein:

- Es darf kein offenes Spiel zwischen den beiden Benutzern bestehen.
- Der Benutzer darf nur eine begrenzte Anzahl an offenen Spielen haben.
- Der einzuladende Benutzer darf nur eine begrenzte Anzahl an offenen Einladungen haben.

Ist dies alles erfüllt, wird der einzuladende Spieler mittels Push-Benachrichtigung über die Einladung informiert.

3.5 INVITE USERS

Der Benutzer kann mehrere Spieler zu einem Mehrspieler-Spiel einladen. Diese Spieler müssen dem Benutzer vom System angezeigt werden, damit sie angewählt werden können. Folgende Bedingungen müssen erfüllt sein, damit die Einladung erfolgreich ist:

- Der Benutzer darf nur eine begrenzte Anzahl an offenen Spielen haben.
- Jeder einzuladende Benutzer darf nur eine begrenzte Anzahl an offenen Einladungen haben. Ansonsten wird er von den einzuladenden Spielern ausgeschlossen.

Ist dies alles erfüllt, werden die einzuladenden Spieler mittels Push-Benachrichtigung über die Einladung informiert.

3.6 RESPOND INVITATION

Offene Einladungen können vom Benutzer der Applikation angenommen oder abgelehnt werden. Um eine Einladung annehmen zu können, darf er aber nur eine begrenzte Anzahl an offenen Spielen haben. Sobald die Annahme erfolgt ist, ist das Spiel offen und kann gespielt werden.

3.7 PLAY GAME

Ist ein Spiel offen, kann der Benutzer alle seine n spielbaren Runden abhandeln. Pro Runde wird bei der Erstellung des Spiels vom System ein zufälliger Spieletyp ausgewählt. Alle Teilnehmer eines Spiels spielen in der Runde x das Spiel y. Haben alle Spieler ihre n Runden abgehandelt, evaluiert das System den Gesamtsieger des Spiels.

Alle Teilnehmer werden über den Ausgang des Spiels mittels Push-Benachrichtigung informiert.

Teil II

ENTWICKLUNGSPROZESS

4 ARCHITEKTUR

In diesem Kapitel wird auf die Architektur der Applikation eingegangen. Im ersten Teil werden die wichtigsten Erkenntnisse aus einem grossen Architekturreview zusammengefasst. Anschliessend wird die Umsetzung dieser Erkenntnisse und die daraus resultierende Architektur beschrieben. Konkrete Details zur Implementierung sind nicht Bestandteil dieses Kapitels und werden im Kapitel 9 ausgeführt.

4.1 REVIEW & REFACTORING

Zu Beginn des Projektes wurde ein Review durchgeführt, um eventuelle Schwächen in der bestehenden Architektur zu finden und diese mit einem Refactoring zu beheben. Nur so kann eine vernünftige Grundlage geschaffen werden, welche durch neue Komponenten erweitert werden kann. Die Erkenntnisse aus dem Review sind nachfolgend aufgelistet:

- **Review**: Das Layering des Prototyp war zu wenig strikt umgesetzt. Dies führte dazu, dass die Architektur mit zunehmender Komplexität unübersichtlicher wurde und deshalb eine hohe Kopplung zwischen den drei Layern bestand.
  Refactoring: Durch die Einführung eines strikten Layerings wurde eine übersichtliche Struktur in der Architektur geschaffen und die Kopplung reduziert. Die einzige Ausnahme des strikten Layerings bilden die Modelobjekte des Data Access-Layer (DAL). Grund dafür ist, dass die verschiedenen Layer mit der jetzigen Komplexität der Domäne keine eigenen Modelobjekte benötigen.

- **Review**: Die Modelobjekte des DAL enthielten allesamt domänenspezifische Logik. Eine Neuimplementierung des DAL (zum Beispiel für eine Anbindung an einen anderen Backend as a Service-Provider) müsste diese Logik erneut implementieren. Eine Austauschbarkeit des untersten Layers war somit unmöglich. Diese Domänenlogik wurde auf der falschen Seite der Schnittstelle implementiert.
  Refactoring: Sämtliche domänenspezifische Logik wurde von den Modelobjekten entfernt und auf die andere Seite der Schnittstelle in den Service-Layer oder ins Backend verlagert. Modelobjekte enthalten nur noch Properties. Sind die Properties virtuell oder werden Akzessoren überschrieben, geschieht dies nur, um mit den Möglichkeiten des Backend SDK die vom Service-Layer

vorgegebenen Schnittstelle zu erfüllen[1]. Ein einfaches Beispiel hierfür sind überschriebene Akzessoren, um die Backend-Daten in die von der Schnittstelle geforderte Form zu konvertieren.

- **Review**: Die Methoden des DAL waren stark spezialisiert. Der Service-Layer delegierte an vielen Orten nur die Anfrage des übergeordneten Layers weiter und nahm somit die Funktion eines Wrappers ein. Nur an vereinzelten Stellen führte der Service-Layer Domänenlogik aus.
  Refactoring: Durch die Verschiebung der Domänenlogik aus dem DAL in den Service-Layer übernehmen beide Layer ihre eigentlichen Aufgaben. Die Methoden des DAL werden generisch gehalten und ermöglichen so die Wiederverwendung durch den darüberliegenden Layer. Dies hat auch Auswirkungen auf die Testbarkeit der beiden Layer: Der DAL wird einfacher testbar und eine hohe Testabdeckung des Service-Layer wird sinnvoll.

- **Review**: Im Sinne der Remote Data-Pattern angetriebenen Architektur wurde sämtliche Logik auf dem Client-Tier implementiert. Aus Performancegründen ist die Ausführung gewisser Logik auf dem Backend jedoch sinnvoller.
  Refactoring: Parse als Backend-Provider bietet die Möglichkeit, Funktionen mit einer beschränkten Laufzeit im Backend zu implementieren. Wenig Domänenlogik wurde aus oben genanntem Grund in das Backend verschoben. Wenn möglich wird die Domänenlogik jedoch im Service-Layer implementiert, um das Remote Data-Pattern einzuhalten und eine verstreute Domänenlogik zu verhindern.

- **Review**: Die Domänenlogik der Modelobjekte im DAL benutzte vom Parse SDK vererbte Methoden, um eigene Daten nachzuladen und zu speichern[2]. Die Repositories des DAL werden dadurch umgangen und die Zuständigkeiten sind nicht klar aufgeteilt.
  Refactoring: Durch die Entfernung der Domänenlogik enthalten nur noch die Repositories Logik, um Daten aus dem Backend nachzuladen, beziehungsweise darin zu speichern.

- **Review**: Die bestehenden Views waren statisch. Dies war sinnvoll, weil sie für Spiele mit nur einem Gegner konzipiert wurden.
  Refactoring: Durch die Überarbeitung der Views mittels dem Einsatz von Komponenten, die eine dynamischere Darstellung erlauben, ist es nun möglich, Daten mehrerer Gegner anzuzeigen.

1 http://de.wikipedia.org/wiki/Adapter_(Entwurfsmuster) [6]
2 http://www.martinfowler.com/eaaCatalog/activeRecord.html [7]

4.2 RESULTIERENDE ARCHITEKTUR

Die endgültige Architektur der entwickelten Applikation baut auf der festgelegten Spezifikation der Studienarbeit auf. Neben den in Abschnitt 4.1 beschriebenen Veränderungen sind zusätzliche Neuerungen realisiert worden. Diese werden im folgenden Abschnitt näher erläutert. Die der Applikation zugrundeliegende Problemdomäne wurde mit dem Folgeprojekt nicht verändert. Der Vollständigkeit halber ist das Domain Modell im Abschnitt A.2 zu finden.

Durch die Umsetzung neuer Use Cases erhöhte sich die Komplexität der Window-Hierarchie. Dies erforderte neue View Controller und neue Service-Klassen, um die Controller mit Daten versorgen zu können. Auch die Zahl der Interaktionen zwischen den oberen zwei Layern hat dadurch zugenommen. Für einige Services wurden zusätzlich neue Modelobjekte mit den dazugehörigen Repositories im untersten Layer eingeführt.

Zuletzt entstanden weitere Abhängigkeiten zu Core-Frameworks von Apple und externen Libraries. Erstere sind auf die neuen Minigames zurückzuführen. Letztere auf die hohen Ansprüche an die grafische Oberfläche, welche mithilfe von bestehenden Libraries einfacher zu erfüllen sind.

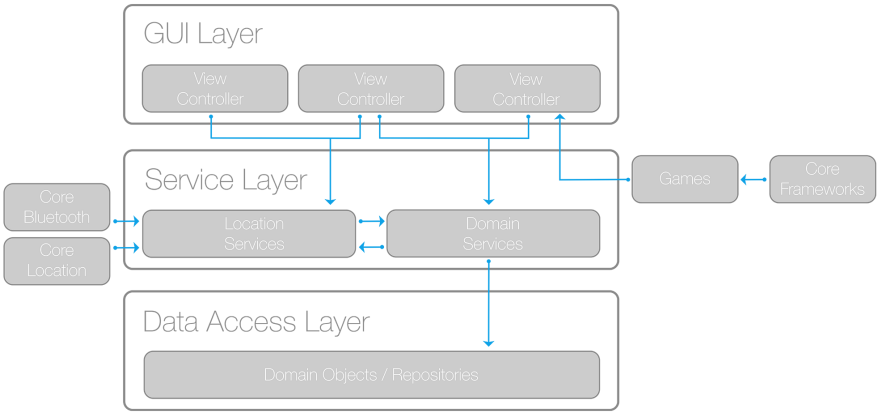


Abbildung 1: Architektur

4.3 ENTSCHEIDUNGEN

4.3.1 *Limitierungen vs. Performance*

Die in Ludur eingesetzte Architektur benötigt Limitierungen, um bei einer steigenden Nutzerzahl akzeptable Antwortzeiten ermöglichen zu können. Beispiele für Limitierungen sind die Anzahl der offe-

nen Spiele, der verschickten Einladungen oder der persistenten Gegner. Im ersten Moment erscheinen diese Limitierungen als klare Einschränkung der Skalierbarkeit. Aus Sicht der Autoren können Limitierungen die Skalierung aber auch beschleunigen. Sie verhindern beispielsweise unbeendete Spiele, Einladungen an inaktive Spieler und Spam-Einladungen. Leider führen in einer Remote Data-Architektur Limitierungen zwingend zu zusätzlichen Anfragen an das Backend, was sich negativ auf dessen Performance auswirkt. Ziel ist es, eine Balance zwischen den Grenzwerten der Limitierungen und der Antwortzeit des Clients zu finden. Die vorgesehene Beta-Testphase eignet sich optimal, um die Applikation zu beobachten und an den Limitierungen zu tunen.

4.3.2 *Promises & Callback Hell*

Die in der Studienarbeit beschriebene Threading-Strategie wurde weitergeführt. Es werden Blöcke verwendet, um asynchrone Methodenaufrufe zu ermöglichen. Anfangs dieses Projektes wurde ein Einsatz von Promises mit PromiseKit[3] diskutiert, um eine Callback Hell[4] zu verhindern. Die PromiseKit-Erweiterung für Parse[5] war zu diesem Zeitpunkt schon über längere Zeit nicht mehr aktualisiert worden. Auch die Reaktionszeit auf Änderungen im Parse SDK ist bei dieser Erweiterung fragwürdig. Ein Einsatz erschien deshalb zu riskant. Resulat dieser Entscheidung ist komplexer, teilweise schwer lesbarer Code und mühsames Mocking[6].

4.3.3 *Virtuelle Properties*

Das Parse SDK erlaubt 1:n Verbindungen nur vom nten Objekt in Richtung 1-Objekt. Oft wird jedoch im GUI eine Darstellung des 1-Objekt mit aggregierten Informationen der n-Objekte benötigt. Des Weiteren ist es nützlich, wenn das 1-Objekt als eine Art Data Transfer Objekt[7] agieren kann und so Daten zwischen Views einfacher verschoben werden können. Um dies unter Einsatz des Parse SDK zu ermöglichen, bieten gewisse Repositories des Data Access-Layer die Möglichkeit, ein virtuelles Property zu generieren, um damit vom 1-Objekt in Richtung n-Objekte navigieren zu können.

3 http://promisekit.org [8]
4 http://callbackhell.com [9]
5 https://github.com/hathway/Parse-PromiseKit
6 http://en.wikipedia.org/wiki/Mock_object [10]
7 http://martinfowler.com/eaaCatalog/dataTransferObject.html [11]

5 MACHBARKEITSSTUDIEN

Im Rahmen dieser Arbeit sind eine Reihe neuer Minigames entstanden, die nicht selten auf Sensoren des iPhones zugreifen. Das setzte voraus, dass die Sensoren vorerst auf ihre Tauglichkeit für das gewünschte Spielverhalten geprüft werden mussten. Diese kleineren Machbarkeitsstudien sind in diesem Kapitel beschrieben. Entwickler, die Applikationen mit einem ähnlichen Kontext entwickeln, können die Code-Beispiele in den referenzierten Github[1] Repositories studieren.

5.1 SCREAMER GAME

Für das Screamer-Spiel wird das Mikrofon des iOS-Device verwendet. Wie der Name schon verrät, geht es bei dem Spiel darum, so laut wie möglich zu schreien. Um abschätzen zu können, wie das Mikrofon auf unterschiedliche Lautstärken reagiert, mussten zuerst dessen Inputwerte untersucht werden. Das Github Repository[2] enthält den zur Studie gehörenden Code.

Die Klasse AVAudioRecorder[3] kann verwendet werden, um Daten des Mikrofons über eine gewünschte Zeitspanne aufzunehmen. Die Methode averagePowerForChannel liefert den aktuellen Dezibelwert für jede gegebene Zeit. Gemäss der Dokumentation von Apple liegen diese Werte zwischen -160 (absolute Stille) und 0 (maximale Lautstärke). Diese Werte eignen sich schlecht, um als Punktezahl für das Spiel verwendet zu werden, weil Spieler nicht mit negativen Punktezahlen rechnen. Deshalb musste eine Formel aufgestellt werden, die die Werte so umrechnet, dass sie für den Benutzer aussagekräftig sind.

Bei einem Test wurden die rohen Rückgabewerte der Methode averagePowerForChannel untersucht. Das führte zu erstaunlichen Ergebnissen:

- -120 stellte sich als Ausgangswert heraus, entgegen der in der Apple-Dokumentation beschriebenen -160.
- Der dB-Wert in einem Wohnzimmer ohne Reden betrug zwischen -67 und -58. Diese Werte sind gerundet.

1 https://github.com [12]
2 https://github.com/meibo/ScreamerTest
3 https://developer.apple.com/library/mac/documentation/AVFoundation/Reference/AVAudioRecorder_ClassReference [13]

- Schreit man ins Mikrofon, erreicht man schliesslich den erwarteten Wert 0.

Um diese Werte in einen positiven Bereich zu bringen, wurde zuerst mit 120 addiert. Da die Werte immer noch logarithmisch verteilt waren, wurde anschliessend die Formel für das Umrechnen von dB in Volt[4] angewendet. Die daraus resultierende Formel für eine Punktezahl zwischen 0 und 1000 setzt sich wie folgt zusammen:

```
CGFloat decibelLevel = ([self.recorder averagePowerForChannel:0] +
                        120.0f) / 1.92869f;
CGFloat voltageLevel = 0.7746f * powf(10.0f, decibelLevel/20.0f);
self.peakLevel = MAX(self.peakLevel, voltageLevel);
```

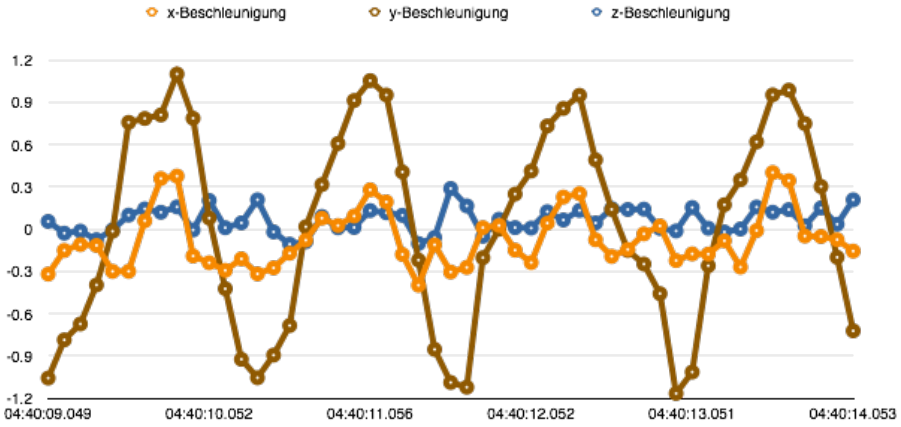
Diese Berechnung ist für das Spiel vorerst einsetzbar. Ganz fair ist sie aber nicht, weil sich die Mikrofone der unterschiedlichen Gerätemodelle in der Sensibilität unterscheiden. Man kann dieses Problem beheben, indem eine Kalibrierung des Mikrofons durchgeführt wird. Weil der Gerätetyp jedes Benutzers herausgefunden werden kann, ist es aber denkbar, diese Kalibrierung automatisch aufgrund dieses Typs vorzunehmen. Dazu müssen die Mikrofone der unterschiedlichen Geräte aber erst über eine längere Zeit getestet werden.

5.2 SHAKER GAME

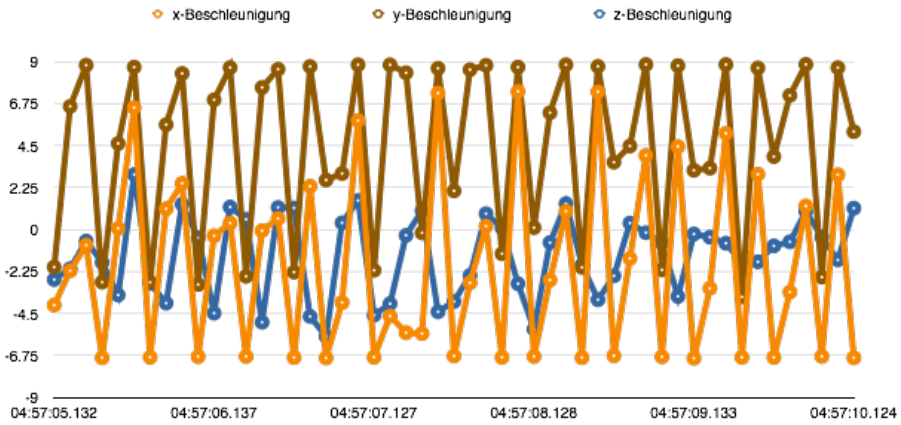
Das Shaker-Spiel verwendet den Accelerometer des iOS-Device. Es betrachtet die Beschleunigung entlang aller Achsen. Sobald ein bestimmter Threshold überschritten wird, erhält der Spieler Punkte. Das Github Repository[5] enthält den zur Studie gehörenden Code. Das Spiel wurde mithilfe eines Threads auf Stackoverflow[6] und einer Anleitung von NSHipster[7] erstellt.

In der Studie wurde untersucht, ob die Beschleunigungsdaten genau genug sind, um ein schwaches, langsames Schütteln von einem intensiven, schnellen Schütteln zu unterscheiden. Zudem musste ein optimaler Threshold gefunden werden, um dem Spiel seinen Reiz zu verleihen. Die folgenden zwei Diagramme zeigen, dass die Daten genau sind und ein optimaler Threshold zwischen drei und fünf G liegt.

4 http://www.sengpielaudio.com/calculator-db.htm [14]
5 https://github.com/meibo/ShakerTest
6 http://stackoverflow.com/questions/19131957/ios-motion-detection-motion-detection-sensitivity-levels [15]
7 http://nshipster.com/cmdevicemotion [16]



(a)



(b)

Abbildung 2: Langsames (a) & schnelles Schütteln (b)

6 ENTWICKLUNGSUMGEBUNG

Dieses Kapitel gewährt einen Einblick in die verwendete Entwicklungsumgebung.

6.1 TESTING

In der Studienarbeit wurde auf die Wichtigkeit eines ausführlichen Testings des Grundgerüstes hingewiesen. Dies wurde im Rahmen dieser Arbeit beibehalten, um die Stabilität in den unteren Layern erneut sicherzustellen. Wie im Abschnitt 4.1 beschrieben, wurde durch ein Refactoring die Domänenlogik in den Service-Layer verschoben. Die Tests des Data Access-Layer wurden dadurch vereinfacht. Diese wurden erneut als Integrationstests umgesetzt. Diese Umstellung führte dazu, dass zusätzlich Tests für den Service-Layer nötig wurden. Es war sinnvoll, diese unabhängig vom unteren Layer zu realisieren. Dadurch wurden die Tests performanter. Der untere Layer musste jedoch komplett durch Mock-Objekte ersetzt werden. Bei vielen asynchronen Methoden mit Blocks im Data Access-Layer gestaltet sich dieser Prozess als besonders umständlich. Im Abschnitt 9.2 wird näher auf die Umsetzung der Mock-Objekte eingegangen.

6.2 DEPLOYMENT

Aus zwei Gründen konnte das in der Studienarbeit eingesetzte Deployment nicht mehr verwendet werden:

1. Travis CI ist kostenpflichtig für Projekte mit mehr als 100 Build-Durchgängen.
2. Testflight wurde von Apple übernommen[1] und in iTunes Connect integriert.

Um die enormen Kosten von Travis CI zu umgehen, wurde ein eigener Build-Server mit Jenkins aufgesetzt. Die Shell-Skripts, die den Build-Ablauf bestimmten, wurden durch die Toolsuite Fastlane ersetzt. Fastlane enthält ein Tool, das den automatischen Upload des gebuildeten iOS App Package in iTunes Connect erlaubt. Mithilfe von iTunes Connect kann dann die Applikation nach einem Review durch Apple an bis zu 1000 externe Tester verteilt werden.

1 https://testflightapp.com [17]

6.3 TOOLS

- Xcode IDE - Entwicklungsumgebung von Apple für iOS-Applikationen
  https://developer.apple.com/xcode/ide
- xctool - Buildsystem
  https://github.com/facebook/xctool
- Cocoapods - Dependency Management System
  http://cocoapods.org
- Eigene Mac-Hardware zum Betrieb eines Jenkins CI-Servers
  https://jenkins-ci.org
- Coveralls - Automatische Code Coverage-Auswertung
  https://coveralls.io
- cpp-coveralls - Python Tool für Code Coverage-Upload an Coveralls
  https://github.com/eddyxu/cpp-coveralls
- iTunes Connect - Plattform für Beta-Testing
  https://itunesconnect.apple.com
- Dash - Dokumentations-Browser und Snippet-Manager
  http://kapeli.com/dash
- PyYAML - YAML-Implementation in Python, die von cpp-coveralls benötigt wird, um Coveralls-YAML-Konfigurationen zu parsen
  http://pyyaml.org

6.4 VERWENDETE FRAMEWORKS, LIBRARIES & SDKS

- Parse SDK - SDK für Anbindung von Parse als Backend
  https://parse.com/docs/downloads
- FXForms - Library zur einfachen Erstellung von tabellenbasierten Formularen
  https://github.com/nicklockwood/FXForms
- SDWebImage - Library, um Bilder asynchron herunterladen und cachen zu können
  https://github.com/rs/SDWebImage
- PNChart - Library zur Erstellung von Charts
  https://github.com/kevinzhow/PNChart
- MZFormSheetController - Eine Alternative zum UIModalPresentationFormSheet
  https://github.com/m1entus/MZFormSheetController

- Waver - Siri-ähnlicher Welleneffekt
  https://github.com/kevinzhow/Waver
- BEMSimpleLineGraph - Library zur Erstellung von Charts
  https://github.com/Boris-Em/BEMSimpleLineGraph
- IHKeyboardAvoiding - Lösung für das einfach Ein- und Ausblenden der Tastatur
  https://github.com/IdleHandsApps/IHKeyboardAvoiding
- OCMock - Framework für das Mocking
  http://ocmock.org
- PixelLove - iOS 8 Icons
  http://www.pixellove.com

Teil III

DIE SPIELEPLATTFORM

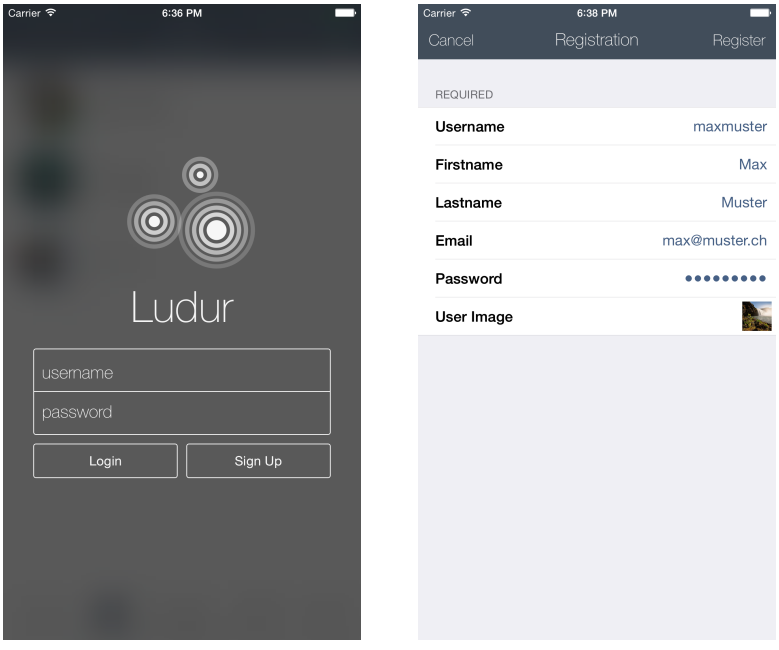
7 FEATURES

Dieses Kapitel umfasst alle Features, die im Rahmen dieser Arbeit implementiert wurden. Die zugrundeliegenden Use Cases wurden im Kapitel 3 definiert. Ziel dieser Beschreibung ist, dass der Leser ohne weitere Instruktionen im Stande ist, die Applikation zu bedienen. Lesern der Studienarbeit wird auffallen, dass viele dieser Features bereits beschrieben wurden. Da viele Features erweitert oder verbessert wurden, wird es aber für sinnvoll erachtet, alle nochmals aufzuführen.

7.1 BENUTZERVERWALTUNG

7.1.1 *Benutzer anlegen*

Beim erstmaligen Start der Applikation muss der Benutzer ein Profil anlegen. Dazu muss er in der Registrierungsansicht eine Reihe von Angaben machen. Verläuft das Speichern dieser Angaben erfolgreich, ist das Benutzerprofil angelegt und der Benutzer kann sich in der Loginansicht anmelden.

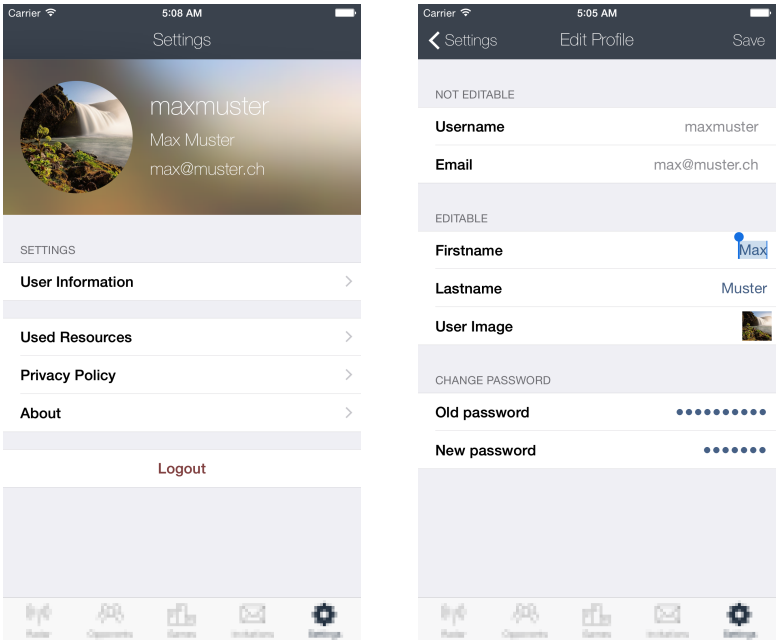


(a) (b)

Abbildung 3: Login- (a) & Registrierungsansicht (b)

7.1.2 *Benutzerdaten editieren*

Der Benutzer kann seine Profildaten (bis auf den Benutzernamen und die E-Mail-Adresse) in der Profilbearbeitungsansicht ändern. Diese Ansicht kann er über eine Verlinkung in der Einstellungsansicht erreichen.



(a) (b)

Abbildung 4: Einstellungs- (a) & Profilbearbeitungsansicht (b)

7.1.3 *Benutzer löschen*

Bei der Benutzung der Applikation werden Benutzer auf Spielen und den dazugehörigen Einladungen und Spielrunden referenziert. Das Löschen des eigenen Benutzerprofils ist deshalb nicht möglich. So wird Inkonsistenz im Backend verhindert.

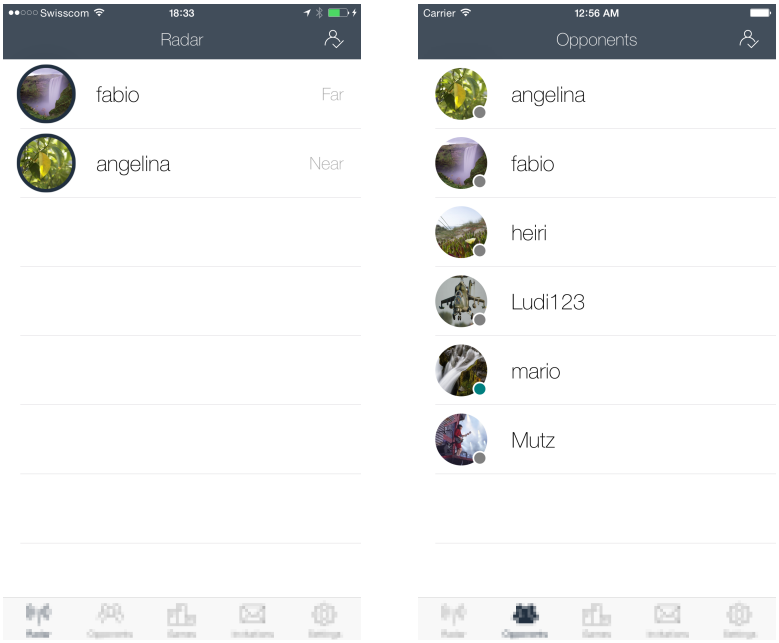
7.2 BENUTZER FINDEN

7.2.1 *Benutzer in der näheren Umgebung*

Weil jeder Benutzer der Applikation als iBeacon-Transmitter fungiert, ermöglicht die Applikation, andere Benutzer in der nähreren Umgebung zu finden. Benutzer, die auf diese Weise entdeckt worden sind, werden in der Radaransicht angezeigt. Sobald sie sich zu weit entfernen und ihr Signal nicht mehr empfangen wird, verschwinden sie wieder aus dieser Ansicht.

7.2.2 *Freigespielte Benutzer*

Im Abschnitt 7.3.4 wird beschrieben, wie ein Benutzer andere Benutzer der Applikation freispielen kann. Diese freigespielten Benutzer werden in der Gegneransicht angezeigt. Persistent freigespielte Benutzer sind mit einem türkisfarbenen Punkt am unteren rechten Rand des Benutzerfotos markiert. Bei nicht persistenten Benutzern weist der Punkt eine graue Färbung auf.



(a) (b)

Abbildung 5: Radar- (a) & Gegneransicht (b)

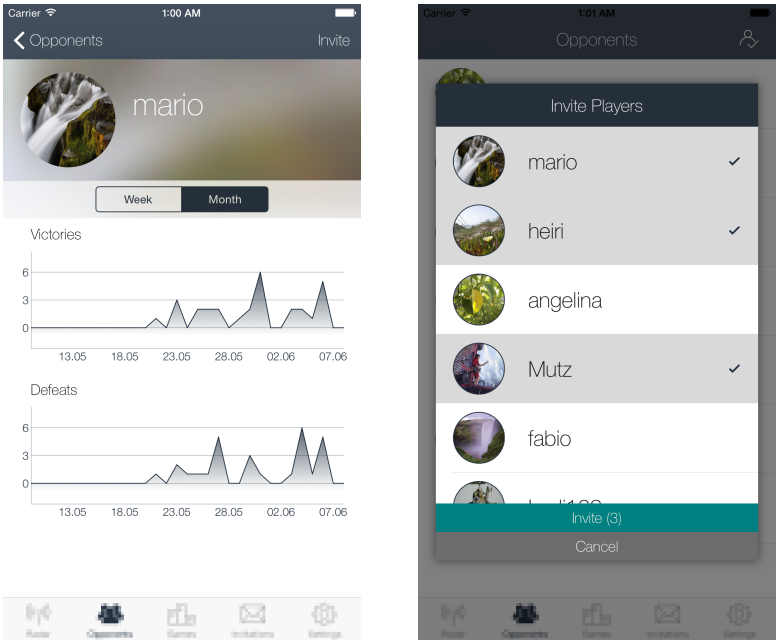
7.3 SPIEL ABSOLVIEREN

7.3.1 *Benutzer einladen*

Um einen oder mehrere Benutzer für ein Spiel einzuladen, gibt es zwei Möglichkeiten:

- **Variante 1**: Ein in der Radar- oder Gegneransicht angezeigter Benutzer kann selektiert werden, um eine Benutzerdetailansicht zu öffnen. Dort kann Einsicht in die Statistiken des Benutzers genommen und der Benutzer zu einem Spiel eingeladen werden.
- **Variante 2**: In der Radar- und Gegneransicht kann die Benutzerwahlansicht geöffnet werden. Diese erlaubt es, mehrere Benutzer gleichzeitig zu selektieren und zu einem Spiel einzuladen.

Benutzer, die mit einer der beiden Varianten eingeladen wurden, werden mittels Push-Benachrichtigung informiert.

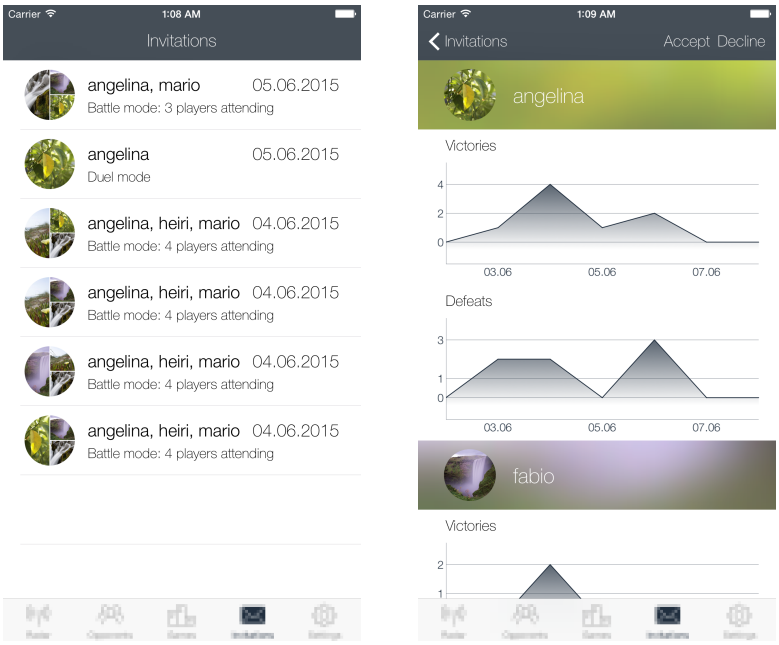


(a) (b)

Abbildung 6: Benutzerdetail- (a) & Benutzerwahlansicht (b)

7.3.2 *Einladung beantworten*

Offene Einladungen werden in der Einladungsansicht angezeigt. Der Benutzer kann sie dort selektieren, um in die Einladungsdetailansicht zu gelangen. Dort kann er Einsicht in die Statistiken der Gegenspieler nehmen und die Einladung annehmen bzw. ablehnen.



(a) (b)

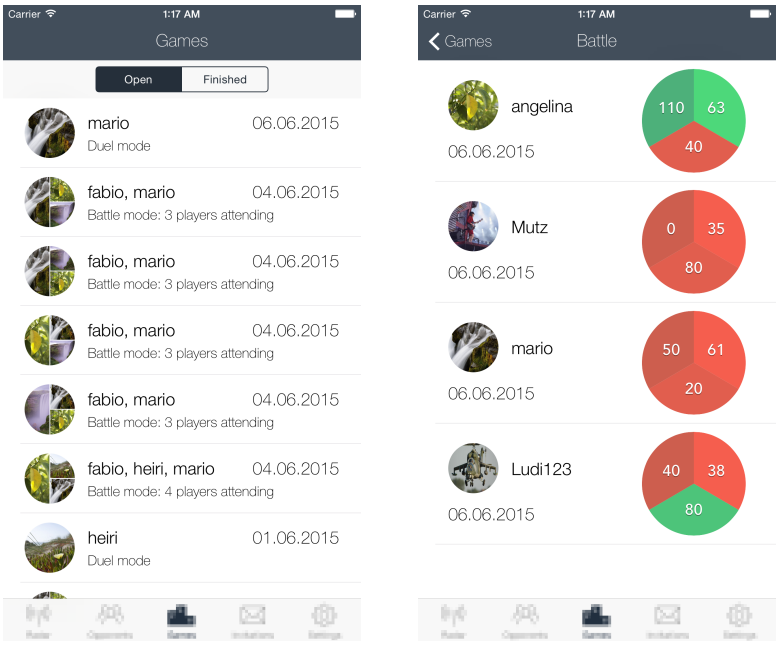
Abbildung 7: Einladungs- (a) & Einladungsdetailansicht (b)

7.3.3 *Spiel durchführen*

Spiele werden in der Spieleansicht angezeigt. Der Benutzer kann auswählen, ob er offene oder bereits fertig gespielte Spiele sehen will. Selektiert der Benutzer eines der Spiele, wird er in die Spielstatusansicht weitergeleitet. Pro Teilnehmer zeigt ein Kuchendiagramm alle Informationen über seine Spielrunden an. Nachfolgend sind die Bedeutung der Färbungen und Anzeigen der einzelnen Abschnitte des Diagramms aufgeführt:

- **Grau**: Die Spielrunde wurde noch nicht gespielt.
- **Blau**: Die Spielrunde wurde gespielt. Es gibt noch offene Spielrunden.
- **Rot**: Die Spielrunde wurde verloren.
- **Grün**: Die Spielrunde wurde gewonnen.
- **Zahl**: Gibt die erspielten Punkte der Spielrunde an.

Hat der Benutzer noch eigene offene Spielrunden, kann er in dieser Ansicht die nächste dieser Runden starten. Beendet der letzte Teilnehmer seine letzte Runde, wird die Auswertung des Spiels angestossen. Diese informiert anschliessend jeden Teilnehmer mittels Push-Benachrichtigung über den Ausgang des Spiels.



(a) (b)

Abbildung 8: Spiele- (a) & Spielstatusansicht (b)

7.3.4 *Belohnungssystem*

Ziel der Applikation ist es, möglichst viele andere Benutzer freizuspielen. Freigespielte Benutzer können persistent oder nicht persistent sein. Persistent bedeutet hier, dass der freigespielte Spieler nicht mehr verloren gehen kann. Das Belohnungssystem unterscheidet sich je nach Art des gespielten Spiels:

- **Einzelspieler**: Dem Gewinner des Spiels werden alle freigespielten Benutzer seines Gegners zu seinen eigenen hinzugefügt. Der Verlierer verliert alle freigespielten Benutzer, die er mit seinem Gegner gemeinsam hat. Verloren gehen können dabei aber nur Benutzer, die nicht persistent sind. Unabhängig vom Ausgang des Spiels wird der jeweilige Gegner neu unter den freigespielten, persistenten Benutzern aufgelistet.
- **Mehrspieler**: Der Gewinner erhält alle freigespielten Benutzer aller anderen Teilnehmer zu seinen eigenen (persistent) hinzu. Je mehr Spieler teilnehmen, desto weniger kann jeder Teilnehmer verlieren.

Dieses Belohnungssystem hat zum Ziel, den Benutzer zu ermutigen, möglichst viele Spiele zu absolvieren.

8 ERWEITERUNGSMÖGLICHKEITEN

Die im Rahmen dieser Arbeit entwickelte Applikation ist in sich abgeschlossen und voll funktionsfähig. Trotzdem wurden im Entwicklungsprozess einige Ideen für Erweiterungen diskutiert und festgehalten. Diese Erweiterungsmöglichkeiten sind neben den umgesetzten Use Cases ebenfalls im Abschnitt A.1 aufgeführt. Auch das Domain Modell im Abschnitt A.2 enhält Objekte , die zu einem Teil der Erweiterungsmöglichkeiten gehören. Diese Erweiterungsmöglichkeiten wurden nicht umgesetzt, um einen Feature Creep[1]. zu verhindern.

In diesem Kapitel werden die oben erwähnten Erweiterungsmöglichkeiten in einer Kurzfassung aufgelistet und ihre Auswirkung auf die Spieldynamik diskutiert.

8.1 MULTIPLAYER LOCATIONS

Diese Erweiterung sieht die Nutzung von stationären iBeacon-Transmitter als Lokationen für Mehrspieler-Turniere vor. Sobald ein Benutzer der Applikation sich nahe genug an einer dieser Lokationen aufhält, wird er vom Backend über mögliche Turnierteilnahmen aufmerksam gemacht. Es ist denkbar, an jeder Lokation verschiedene Turniere anzubieten. So könnten zum Beispiel Tages-, Wochen- und Monatsturniere angeboten werden. Meldet sich der Benutzer für eines der Turniere an, kann er seine Spielrunden absolvieren. Der Sieger jedes Turniers wird nach Ablauf der zeitlichen Frist vom Backend bestimmt. Er erhält als Belohnung die freigespielten Benutzer aller Turnierteilnehmer. Interessant werden diese Mehrspieler-Lokationen vor allem dann, wenn sie an hoch frequentierten Orten platziert werden. Bahnhöfe, Flughäfen, Einkaufszentren und Parks sind Beispiele dafür. Diese Erweiterung verspricht eine positive Auswirkung auf die Spieldynamik, weil es für den Benutzer der Applikation interessant ist, sich mit einer sehr grossen Menge an Gegenspielern zu messen.

8.2 GEGENSPIELER AUSTAUSCHEN

Eine mögliche Erweiterung ist es, das Austauschen von freigespielten Gegnern zu ermöglichen. Dies ermöglicht es dem Benutzer der Applikation, bei der Gegnerauswahl strategischer vorzugehen und die Anzahl seiner freigespielten Gegner schneller zu vergrössern. Das wirkt sich positiv auf die Vernetzung aller Anwender aus und fördert

1 http://en.wikipedia.org/wiki/Feature_creep [18]

dadurch die Skalierbarkeit. Voraussetzung für diese Erweiterung ist, dass für jeden Benutzer die Einsicht in die Liste seiner freigespielten Gegner ermöglicht wird. Diese durchzusehen ist für den Benutzer der Applikation auch dann interessant, wenn anschliessend kein Austausch zustande kommt. So oder so wirkt sich diese Erweiterungsmöglichkeit positiv auf die Spieldynamik aus.

8.3 AUSFÜHRLICHE SPIELSTATISTIKEN

Der Use Case aus Abschnitt 3.3 zeigt, dass die Applikation in ihrem jetzigen Zustand bereits eine Einsicht in die Spielstatistiken der sichtbaren Gegner jedes Anwenders erlaubt. Diese Statistiken werden in Form von Koordinatensystemen dargestellt, wobei die x-Achse die Zeit und die y-Achse die Anzahl der gewonnenen bzw. verlorenen Spiele anzeigt. Zusätzliche Statistiken, die auch sinnvoll für den Anwender sind, sind nachfolgend aufgelistet:

- Die Anzahl der gewonnen bzw. verlorenen Spiele seit der Registrierung soll als Zahl angegeben werden.
- Die Anzahl freigespielter Benutzer soll als Rang angegeben werden.
- Das Land, aus welchem die meisten freigespielten Gegner des Benutzers stammen, soll mit Wappen und Namen angezeigt werden. Das erfordert, dass jeder Benutzer bei der Registrierung zusätzlich sein Heimatland angibt.
- Eine Übersicht über die Rekordhalter jedes Spiels soll integriert werden.

Diese Informationen ermöglichen es dem Anwender der Applikation, bei der Auswahl der Gegenspieler strategischer vorzugehen. Das fördert die Skalierbarkeit der Vernetzung unter den Anwendern und wirkt sich deshalb positiv auf die Spieldynamik aus.

8.4 AUSWAHL AN SPIELEN

Die Anzahl der integrierten Minigames ist stark an den Abwechslungsreichtum der Applikation gekoppelt. Bei der Veröffentlichung von neuen Versionen der Applikation macht es deshalb Sinn, immer auch das Angebot der Minigames zu erweitern. Obwohl diese Erweiterungsmöglichkeit offensichtlich erscheint, darf ihre Auswirkung auf die Spieldynamik nicht unterschätzt werden.

8.5 APPLE WATCH

Eine mögliche Erweiterung ist eine Apple Watch-Integration. Benutzer, die in der näheren Umgebung aufgespürt werden, lösen eine

Push-Benachrichtigung aus, die dann auf der Apple Watch angezeigt werden kann. Des weiteren bietet die Apple Watch zusätzliche Sensoren, welche für neue Minigames verwendet werden können.

8.6 PRIVATSPHÄRE

Diese Erweiterung ermöglicht dem Anwender, seine Privatsphäre zu schützen. Aufdringliche Benutzer können geblockt werden und werden nicht mehr informiert, wenn sich der Anwender in ihrer Nähe bewegt. Ausserdem kann der Anwender entscheiden, ob er beim Gebrauch der Applikation im Hintergrund darauf verzichten will, über Benutzer in seiner Umgebung informiert zu werden. Diese Optionen wirken sich negativ auf die Skalierbarkeit der Applikation aus. Es ist aber denkbar, dass sich durch den Einsatz dieser Erweiterungen zusätzliche Personen für den Gebrauch der Applikation entscheiden.

8.7 RENTABILITÄT

Um eine steigende Nutzerzahl handhaben zu können, werden wahrscheinlich kostenpflichtige Services oder neue Hardware benötigt. Diese Kosten können teilweise mithilfe folgender Erweiterungen gedeckt werden:

- Werbung kann gegen Bezahlung entfernt werden.
- Die Applikation enthält gewisse Einschränkungen, die nur gegen Bezahlung aufgehoben werden können (Freemium Modell).
- Die Applikation kann kostenpflichtig angeboten werden.

Teil IV

IMPLEMENTIERUNG

9 IMPLEMENTIERUNGSDETAILS

Dieses Kapitel gibt Einsicht in Implementierungsdetails, welche interessante Lösungen darstellen. Das soll unter anderem eine Hilfe für Entwickler sein, die bei eigenen Projekten auf die gleichen oder ähnliche Probleme stossen. Sollte Ludur ausserhalb des Rahmens dieser Arbeit weitergeführt werden, ist es relevant, dass die bisher wichtigsten Probleme und Lösungsansätze dokumentiert sind.

9.1 BEZIERKURVEN

Das Logo von Ludur ist vollständig in Code realisiert. Dafür wurde eine wiederverwendbare Komponente geschaffen, die mithilfe von Bezierkurven eine variable Anzahl Ringe zeichnen kann. Die Vorteile gegenüber einer simplen Grafikdatei sind im folgenden aufgelistet:

- Anpassungen für neue Auflösungen (zum Beispiel Retina HD mit der Einführung des iPhone 6 Plus[1]) erfordern keine Neuerstellung einer Grafikdatei.
- Die Bezierkurven lassen sich einfach animieren.
- Der Code der Bezierkurven benötigt weniger Platz als Grafikdateien.

1 https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS8.html [19]

```
// Center circle
CGFloat centerCircleRadius = self.centerCircleRadius;
CGPoint center = CGPointMake(
      CGRectGetMidX(frame)-centerCircleRadius/2,
      CGRectGetMidY(frame)-centerCircleRadius/2);
UIBezierPath* centerCircle = [UIBezierPath bezierPathWithOvalInRect:
      CGRectMake(center.x, center.y, centerCircleRadius,
      centerCircleRadius)];
[[UIColor whiteColor] setFill];
[[UIColor whiteColor] setStroke];
centerCircle.lineWidth = 3;
[centerCircle stroke];
[centerCircle fill];

// sorrounding circles
int numberOfCircles = self.numberOfCircles;
for(int i = 1; i < numberOfCircles; i++){
    CGFloat round1Radius = centerCircleRadius + i*16;
    CGPoint centerRound1 = CGPointMake(CGRectGetMidX(
          centerCircle.bounds)-round1Radius/2, CGRectGetMidY(
          centerCircle.bounds)-round1Radius/2);
    UIBezierPath* round1 = [UIBezierPath bezierPathWithOvalInRect:
          CGRectMake(centerRound1.x, centerRound1.y, round1Radius,
                round1Radius)];
    round1.lineWidth = 5;
    float delta = 1.0/(numberOfCircles);
    float alpha = (1.0-i*delta);
    [[UIColor colorWithRed:1.0 green:1.0 blue:1.0 alpha:alpha]
           setStroke];
    [round1 stroke];
}
```

9.2 MOCKING VON ASYNCHRONEN METHODEN

Wie schon in Abschnitt 6.1 beschrieben, musste der Data Access-Layer durch Mock-Objekte ersetzt werden, um den Service-Layer unabhängig zu testen. Das Mocking betraf vor allem asynchron aufgerufene Repository-Methoden, die Daten vom oder an das Backend liefern. Aufgrund der aktuell noch fehlenden Möglichkeit, Block-Methoden mit void-Rückgabetyp zu mocken[2], wurde das Framework OCMock dem Framework OCMockito vorgezogen. OCMock bietet die Möglichkeit, die Parameter eines Blockes, der als Parameter an die zu mockende Methode übergeben wurde, im Voraus festzulegen. Dazu kann der Block aus einem NSInvocation-Objekt in eine Variable mit dem Typ der Blocksignatur abgelegt und mit eigenen Parametern ausgeführt werden. Die Idee für diesen Lösungsansatz stammt aus einem Stackoverflow-Thread[3].

2 https://github.com/jonreid/OCMockito/pull/93
3 http://stackoverflow.com/questions/17907987/nsinvocation-getargumentatindex-confusion-while-testing-blocks-with-ocmock [20]

```
-(void)test_mockGetGameSessionsByPlayer{
    typedef void (^gameSessionsResultBlock)(NSSet* gameSessions,
          NSError* error);
    id<GameSessionRepository> gameSessionRepository = [DAFactory
          createGameSessionRepository];
    id gameSessionRepositoryMock = OCMPartialMock(
          gameSessionRepository);
    id<GameSession> gameSession = [gameSessionRepository
          createGameSession];

    // mocking of void block parameter method
    // -(void)getGameSessionsByPlayer:(id<User>)player block:(void(^)(
    // NSSet* gameSessions, NSError* error))gameSessionResultBlock;
    [[[gameSessionRepositoryMock expect] andDo:^(NSInvocation
          *invocation) {
        // local variable with block signature to save passed block
        gameSessionsResultBlock resultBlock = nil;
        // load passed block from NSInvocation
        [invocation getArgument:&resultBlock atIndex:3];
        // execute passed block with predefined parameter
        resultBlock([NSSet setWithObject:gameSession], nil);
    }] getGameSessionsByPlayer:[OCMArg any] block:[OCMArg any]];

    // execution of mocked method
    XCTestExpectation *testExpectation = [self
          expectationWithDescription:@"example"];
    __block id<GameSession> capturedGameSession = nil;
    [gameSessionRepositoryMock getGameSessionsByPlayer:nil block:^(
          NSSet *gameSessions, NSError *error) {
        capturedGameSession = [[gameSessions allObjects] firstObject];
        [testExpectation fulfill];
    }];

    // test if mocking object returns predefined parameter
    [self waitForExpectationsWithTimeout:5.0 handler:^(NSError* error)
    {
        XCTAssertEqual(gameSession, capturedGameSession);
    }];
}
```

Des weiteren können Parameter von gemockten Methoden aufgezeichnet und später ausgewertet werden. Dies ist nützlich, um zu überprüfen, welche Daten an das Backend zur Speicherung übergeben wurden.

```
__block NSArray* savedGameRounds = nil;
[[[self.gameRoundRepositoryMock expect] andDo:^(NSInvocation*
      invocation) {
    booleanResultBlock resultBlock = nil;
    [invocation getArgument:&resultBlock atIndex:3];
    resultBlock(YES, nil);
}] saveGameRounds:[OCMArg checkWithBlock:^BOOL(id obj) {
    // capture passed parameter for deferred evaluation
    savedGameRounds = obj;
    return YES;
}] block:[OCMArg any]];
```

9.3 TIMEZONE FEHLER

In verschiedenen Ansichten der Applikation werden Statistiken über abgeschlossene Spiele von Benutzer angezeigt. Um dies zu ermöglichen, müssen die Daten der Spiele vom Backend heruntergeladen und anschliessend nach Datum sortiert werden. Das Problem der NSDate-Klasse[4] ist, dass nur in der GMT gerechnet werden kann. Erst wenn das Datum im GUI angezeigt wird, wird es auf die gerätelokale Zeitzone formatiert. Werden Spiele nun pro Datum in Gruppen sortiert, werden alle NSDate-Objekte, die in GMT zwischen 22:00 und 24:00 angesiedelt sind, der falschen Gruppe zugeordnet, nämlich relativ zur Zeitzone GMT+2. Dort gehören diese Objekte bereits zum darauffolgenden Tag.

Gelöst werden kann das Problem, indem Objekte der Klasse NSDateComponents[5] verwendet werden. So ist es möglich, bereits unterhalb des GUI-Layers mit lokalisierten Daten zu rechnen. Das Datum eines Spiels kann wie folgt ausgelesen werden:

```
NSCalendar* calendar = [NSCalendar currentCalendar];
NSDateComponents *components = [calendar components:
                                    NSCalendarUnitYear
                                    | NSCalendarUnitMonth
                                    | NSCalendarUnitDay
                                    fromDate:gameSession.updatedAt];
```

4 https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSDate_Class [21]

5 https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/Foundation/Classes/NSDateComponents_Class/index.html [22]

10 PROBLEME MIT PARSE

Dieses Kapitel zeigt Probleme und Limitierungen des Backend-Providers Parse auf. Die Probleme konnten während der Implementierung nur bedingt gelöst werden. Bei der Entscheidung über den Einsatz von Parse können diese für zukünftige Projekte massgebend sein. Es ist anzumerken, dass die Probleme nicht unlösbar sind, jedoch im gegebenen Rahmen dieser Arbeit keine bessere Lösung gefunden wurde.

10.1 LIMITIERUNG DER REQUESTS

Das Backend hat ein Request-Limit von 1800 Requests pro Minute[1]. Bei einer Remote Data-Architektur muss bedacht werden, dass auch Überprüfungsanfragen (zum Beispiel die Überprüfung, ob bereits ein offenes Spiel besteht) einen Request auslösen. Überprüfungslogik liesse sich in sogenannten Cloud Code auf das Backend verschieben. Dieser Cloud Code unterliegt jedoch auch dem Request-Limit. Cloud Code ist nichts weiter als ein Javascript-Client, der auf dem Parse-Backend ausgeführt wird.

10.2 LIMITIERUNG DER DATENSÄTZE

Bei Datenabfragen lassen sich keine Where-Contraints auf Foreignkey-Objekte anwenden. Auch mit einer Subquery lässt sich dies nur bedingt lösen, da Parse ein Query-Limit von 1000 Datensätzen hat, das auch für Subqueries gilt.

10.3 FEHLENDE AGGREGATSFUNKTIONEN

Ausser der Count-Funktion sind keine Aggregatsfunktionen vorhanden. Für Übersichten wie die Statistikansicht müssen alle Daten zuerst geladen und anschliessend sortiert werden. Die Count-Funktion ist nur beschränkt verwendbar, da Count-Anfragen limitiert sind und bei über 1000 Datensätzen "inaccurate" (ungenau) werden[2].

1 https://parse.com/plans/faq [23]
2 https://www.parse.com/docs/ios/guide#queries-counting-objects [24]

10.4 TIME-OUT BEI TRIGGER

Die von Parse angebotenen Speicher- und Delete-Trigger versagen oft bei hoher Last. In Ludur wurde ein Speichertrigger auf dem Userobjekt verwendet, um jedem User eine eindeutige ID zuzuweisen (Sequence). Implementiert wurde dies, indem die aktuell höchste ID abgefragt und dem neuen User die inkrementierte ID zugewiesen wird. In Integrationstests, in denen mehrere User nacheinander angelegt wurden, schlugen diese einfachen Abfragen und Zuweisungen häufig mit der Meldung "timed out" fehl.

10.5 TRANSAKTIONALITÄT

Parse bietet keine Transaktionalität an. Die einzig atomar ausführbare Funktion ist incrementKey:key byAmount:amount. Dadurch liesse sich vielleicht mit hohem Aufwand eine Transaktionalität implementieren. Ein manueller Rollback oder Compensation Actions sind mögliche Workarounds.

10.6 PERFORMANCEEINBUSSE DURCH TLS

Parse verwendet mehrheitlich TLS[3]. Diese Security beeinträchtigt die Netzwerkperformance. Da das Parse SDK auf mobile Betriebssysteme ausgelegt ist, wäre eine mögliche Deaktivierung wünschenswert.

3 http://de.wikipedia.org/wiki/Transport_Layer_Security [25]

Teil V

APPENDIX

APPENDIX

A.1 USE CASE DIAGRAMM

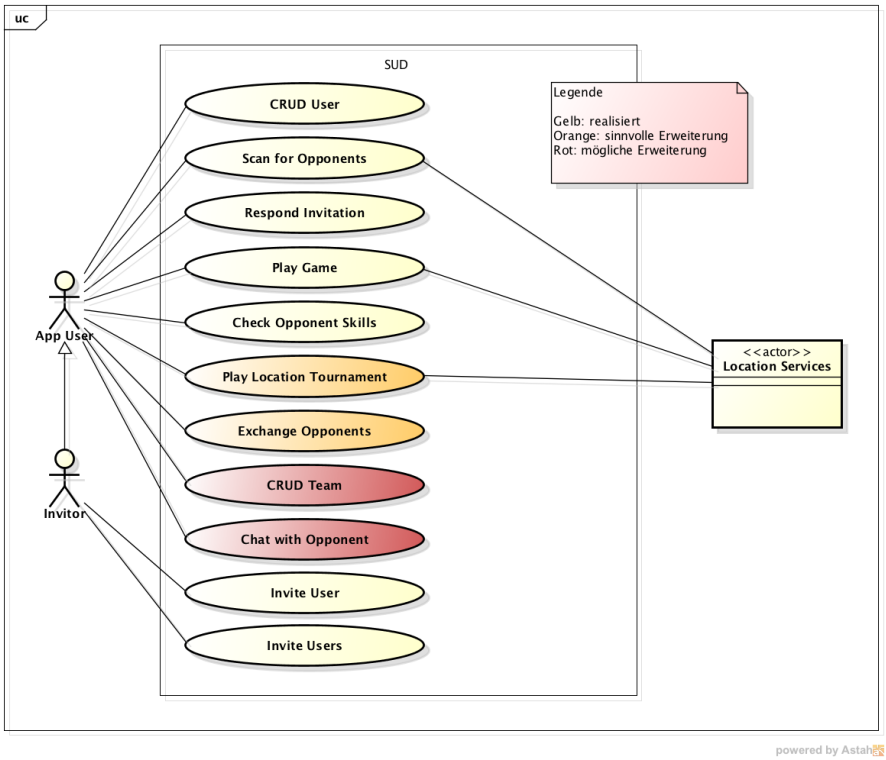


Abbildung 9: Use Case Diagramm

A.2 DOMAIN MODELL

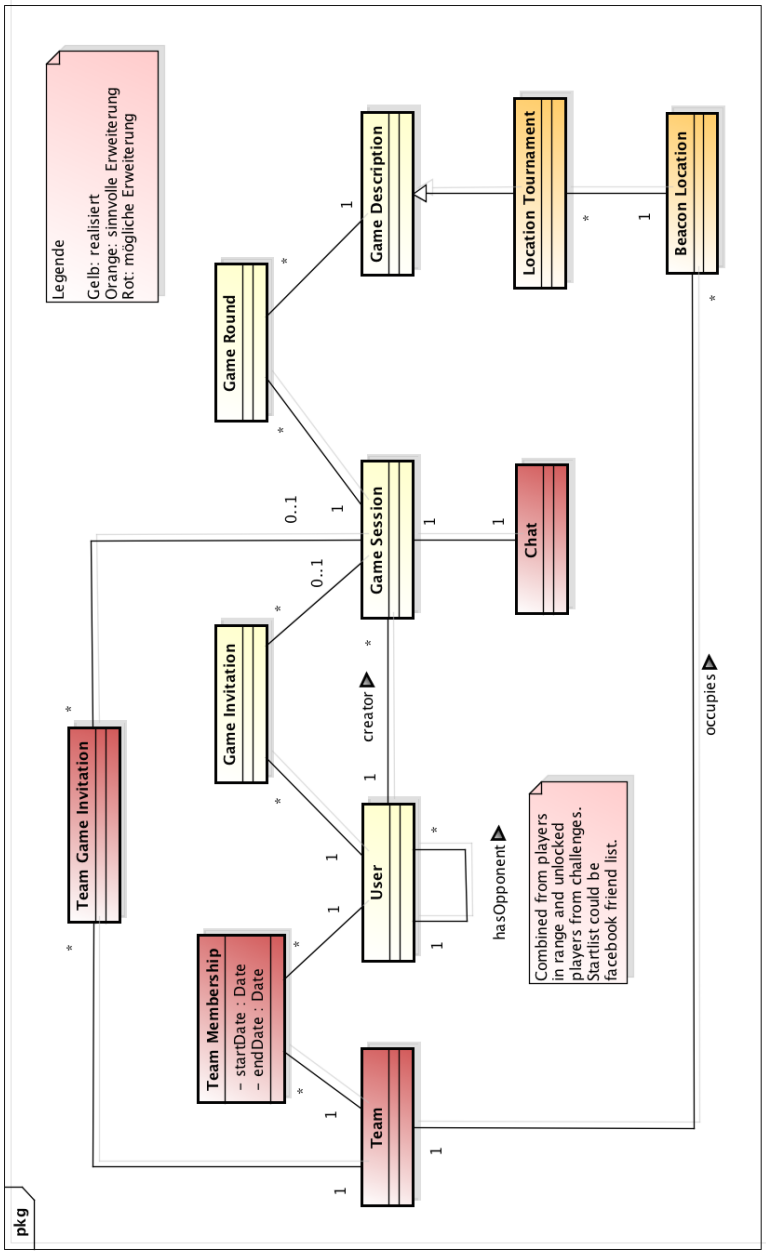


Abbildung 10: Domain Modell

LITERATURVERZEICHNIS

[1] Sebastian Bock and Mario Meili. Studienarbeit - Ludur, ein iOS Spiel mit iBeacons, 2014. URL http://eprints.hsr.ch/405/1/Ludur%20-%20ein%20iOS%20Spiel%20mit%20iBeacons.pdf. [Online, letzter Zugriff 26.05.2015].

[2] Statista. Number of available apps in the Apple App Store from July 2008 to January 2015, 2015. URL http://www.statista.com/statistics/263795/number-of-available-apps-in-the-apple-app-store. [Online, letzter Zugriff 26.05.2015].

[3] Statista. Most popular Apple App Store categories in March 2015, by share of available apps, 2015. URL http://www.statista.com/statistics/270291/popular-categories-in-the-app-store. [Online, letzter Zugriff 26.05.2015].

[4] Apple Inc. Technologie - Extreme Leistung. Extrem effizient., 2015. URL http://www.apple.com/de/iphone-6/technology. [Online, letzter Zugriff 26.05.2015].

[5] Apple Inc. Designing for iOS, 2015. URL https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG. [Online, letzter Zugriff 26.05.2015].

[6] Wikipedia. Adapter (Entwurfsmuster), 2015. URL http://de.wikipedia.org/wiki/Adapter_(Entwurfsmuster). [Online, letzter Zugriff 28.05.2015].

[7] Martin Fowler. Active Record, 2015. URL http://www.martinfowler.com/eaaCatalog/activeRecord.html. [Online, letzter Zugriff 28.05.2015].

[8] PromiseKit. PromiseKit, 2015. URL http://promisekit.org. [Online, letzter Zugriff 07.06.2015].

[9] Callback Hell. Callback Hell, 2015. URL http://callbackhell.com. [Online, letzter Zugriff 07.06.2015].

[10] Wikipedia. Mock object, 2015. URL http://en.wikipedia.org/wiki/Mock_object. [Online, letzter Zugriff 07.06.2015].

[11] Martin Fowler. Data Transfer Object, 2015. URL http://martinfowler.com/eaaCatalog/dataTransferObject.html. [Online, letzter Zugriff 07.06.2015].

[12] Github. Github: Tool für Code Management, 2015. URL https://github.com. [Online, letzter Zugriff 06.06.2015].

[13] Apple Inc. AVAudioRecorder, 2014. URL https://developer.apple.com/library/mac/documentation/AVFoundation/Reference/AVAudioRecorder_ClassReference. [Online, letzter Zugriff 07.06.2015].

[14] sengpielaudio.com. Tontechnik-Rechner, 2015. URL http://www.sengpielaudio.com/calculator-db.htm. [Online, letzter Zugriff 07.06.2015].

[15] Nate Cook. iOS Motion Detection: Motion Detection Sensitivity Levels, 2013. URL http://stackoverflow.com/questions/19131957/ios-motion-detection-motion-detection-sensitivity-levels. [Online, letzter Zugriff 08.06.2015].

[16] zic10. CMDeviceMotion, 2014. URL http://nshipster.com/cmdevicemotion. [Online, letzter Zugriff 08.06.2015].

[17] Apple Inc. TestFlight has moved., 2015. URL https://testflightapp.com. [Online, letzter Zugriff 07.06.2015].

[18] Wikipedia. Feature creep, 2015. URL http://en.wikipedia.org/wiki/Feature_creep. [Online, letzter Zugriff 07.06.2015].

[19] Apple Inc. What's New in iOS, 2015. URL https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS8.html. [Online, letzter Zugriff 07.06.2015].

[20] Kaan Dedeoglu. NSinvocation getArgument:atIndex: confusion while testing blocks with OCMock, 2013. URL http://stackoverflow.com/questions/17907987/nsinvocation-getargumentatindex-confusion-while-testing-blocks-with-ocmock. [Online, letzter Zugriff 08.06.2015].

[21] Apple Inc. NSDate, 2013. URL https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSDate_Class. [Online, letzter Zugriff 07.06.2015].

[22] Apple Inc. NSDateComponents, 2012. URL https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/Foundation/Classes/NSDateComponents_Class/index.html. [Online, letzter Zugriff 07.06.2015].

[23] Parse. Parse Core, 2015. URL https://parse.com/plans/faq. [Online, letzter Zugriff 10.06.2015].

[24] Parse. Counting Objects, 2015. URL https://www.parse.com/docs/ios/guide#queries-counting-objects. [Online, letzter Zugriff 10.06.2015].

[25] Wikipedia. Transport Layer Security, 2015. URL http://de.wikipedia.org/wiki/Transport_Layer_Security. [Online, letzter Zugriff 10.06.2015].

EIGENSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe und
- dass ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Rapperswil, Frühjahrssemester 2015

Mario Meili, 11. Juni 2015

Sebastian Bock, 11. Juni 2015