LUDUR

MARIO MEILI, SEBASTIAN BOCK

Ein iOS Spiel mit iBeacons
Studienarbeit, Herbstsemester 2014

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

BETREUER:
Prof. Dr. Josef M. Joller

UNIVERSITÄT:
HSR Hochschule für Technik Rapperswil

ABTEILUNG:
Abteilung Informatik

INSTITUT:
ITA Institut für Internet Technologien und -Anwendungen

ORT:
Rapperswil

SEMESTER:
Herbstsemester 2014

KURZFASSUNG

Bekannte Anwendungen, die die iBeacon Technologie verwenden, beschränken sich oft auf den gleichen Use Case: kontextbasierte Informationsbenachrichtigung. Klar ist jedoch, dass sich der iBeacon Standard auch für interessantere Use Cases nutzen lässt. In dieser Arbeit wird untersucht, ob sich die Technologie auch für ein Spiel nutzen lässt.

Entstanden ist Ludur (lat. ludus, das Spiel), eine Spielplattform, mit welcher man verschiedene Spiele mit Personen aus der unmittelbaren Umgebung spielen kann. Implementiert ist Ludur in Objective-C und ist somit geeignet für alle mobilen Geräte, die iOS8 unterstützen.

Das Ziel der Arbeit ist die Überprüfung der Machbarkeit aller Kernfunktionalitäten, die die angestrebte Spielplattform benötigt. Dies beinhaltet die Verwaltung von Benutzern, ein ansprechendes GUI Design und die Vernetzung von Benutzern mithilfe der iBeacon Technologie. Ausserdem sollen Erweiterungsmöglichkeiten aufgezeigt werden, die bei einer Fortführung des Projektes implementiert werden können. Das Produkt der Arbeit kann als ein erster Prototyp der Plattform angesehen werden.

Ludur zeigt, dass der Einsatz der iBeacon Technologie in einer Spielplattform nicht nur möglich ist, sondern besonders gut dafür geeignet ist. Benutzer in der Nähe können schnell und zuverlässig ausgemacht und identifiziert werden.

Da sich Ludur noch in der Entwicklungsphase befindet, wurde es bisher nicht veröffentlicht. Eine Beta Testphase ist aber bereits in Planung.

DANKSAGUNG

Als erstes möchten wir uns bei Prof. Dr. Josef M. Joller bedanken, der es uns ermöglichte, dieses offene Projekt durchzuführen und der uns während der Arbeit jederzeit unterstützte.

Ein ganz spezielles Dankeschön geht an unseren wehrten Kommilitonen Fabio Pigagnelli, der uns tatkräftig bei der Gestaltung des App-Logos unter die Arme griff.

Auch möchten wir all unseren Kollegen danken, die uns bei der Ideenfindung zu Beginn des Projektes zu Seite standen.

Zum Schluss möchten wir uns ganz herzlich bei Prof. Dr.-Ing. André Mied für das *classicthesis* LaTeX-Template bedanken, welches wir zur Erstellung und Bearbeitung dieser Dokumentation verwendet haben.

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 Betriebsystem laufen
UUID	Universally Unique Identifier
SDK	Software Development Kit
GUI	Graphical User Interface
DAL	Data Access Layer
GB	Gigabyte

Teil I

EINFÜHRUNG

1 MOTIVATION

Seit das erste iPhone im Jahr 2007 die Mobilgeräte Welt revolutioniert hat, werden täglich neue Apps im App Store publiziert. Hinter vielen der erfolgreichsten Apps stehen grosse Entwicklerteams. Es gibt aber auch immer wieder Ausnahmebeispiele, die zeigen, dass auch mit wenigen Ressourcen eine App bekannt werden kann. Um mit einer selbst entwickelten App eine zumindest Campus weite Verbreitung zu erreichen, muss untersucht werden, welche Eigenschaften zum Erfolg einer App führen können.

Vielen der erfolgreichsten Apps liegt eine grosse Vernetzung der Anwender zugrunde. Beispiele dafür sind Facebook, Twitter, LinkedIn, usw. Faszinierend dabei ist die Skalierbarkeit dieser Netze und die unzähligen Anwendungen, die mit einer solchen Vernetzung möglich werden. Diese Eigenschaft in eine eigene App einzubauen und dabei gleichzeitig die iBeacon Technologie einzusetzen, ist eine spannende und vielseitige Herausforderung.

1.1 iOS8 und Objective-C

Objective-C belegt gemäss dem TIOBE Index den dritten Rang unter den beliebtesten Programmiersprachen[1]. Da Objective-C nicht an der HSR unterrichtet wird, entsteht der Reiz, diese Lücke in einem eigenen, grösseren Projekt zu füllen. Gleichzeitig ist es interessant zu sehen, was das erst kürzlich erschienene iOS8 alles an Neuigkeiten zu bieten hat.

1.2 iBeacon

iBeacon wurde der breiten Öffentlichkeit zum ersten Mal bei der Präsentation von iOS7[2] vorgestellt. Seither gibt es verschiedene Hersteller von iBeacon Transmittern sowie Apps, die die iBeacon Technologie verwenden. Selbst Apple verwendet in ihren App Stores iBeacon Transmitter, welche mit der Apple Store App kommunizieren. Die meisten der im App Store veröffentlichten Apps beschränken sich jedoch auf den gleichen Use Case, nämlich den Check-in bei einem iBeacon mit anschliessender kontextbasierten Informationsbenachrichti-

1 http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html [1]

2 https://developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS7.html#//apple_ref/doc/uid/TP40013162-SW1.url [2]

gung. Typische Beispiele für diesen Use Case sind Museums- und Einkaufsapps. Es kann jedoch nicht sein, dass dies die einzigen realisierbaren Use Cases sind. Auch ein iPhone oder iPad kann beispielsweise als iBeacon Transmitter verwendet werden, was einfache Transmitter überflüssig macht und somit eine Peer-to-Peer Kommunikation erlaubt.

2 VISION

Dieses Kapitel beschreibt den Prozess unserer Ideenfindung und fasst unsere gesetzten Ziele für das Gesamtprojekt zusammen.

2.1 IDEENFINDUNG

Die grosse Herausforderung jedes offenen Projektes ist die Ideenfindung, da kaum Vorgaben bestehen, in welche Richtung das Projekt gehen soll. Dieser Abschnitt dient dazu, unsere Herangehensweise an dieses Problem zu beschreiben.

In einem ersten Schritt haben wir hauptsächlich Brainstorming betrieben. Dies geschah über die Zeitdauer einer Woche und im Austausch mit unserem Betreuer und unseren Kommilitonen. Wichtig dabei war, dass keine Annahmen über die technische Realisierbarkeit der Ideen gemacht wurden, weil diese den Denkprozess eingeschränkt hätten. Die Ideen wurden niedergeschrieben, damit keiner der erarbeiteten Ansätze vergessen gehen konnte.

Drei der Ideen, welchen wir über längere Zeit nachgegangen sind und zu welchen wir uns genauere Überlegungen gemacht haben, sind nachfolgend mit der Begründung, weshalb wir sie verworfen, beziehungsweise angenommen haben, beschrieben. Interessant dabei ist es, dass es sich bei allen drei Ideen um Spielkonzepte handelt, obwohl wir zu Beginn der Brainstorming Phase komplett andere Ansätze verfolgt haben. Uns haben es diese Ideen deshalb so angetan, weil uns noch kein Spiel bekannt ist, welches die iBeacon Technologie einsetzt und sich Spiele zudem mit einer hohen Skalierbarkeit ausgezeichnet kombinieren lassen.

2.1.1 *Fangen spielen*

Beim Fangen spielen wird jeder Spieler mit einem iPhone und einem iBeacon Transmitter ausgerüstet. Der iBeacon Transmitter muss von jedem Spieler sichtbar getragen werden, optimalerweise im Bereich des Rückens. Ziel des Spieles ist es, den gegnerischen Spieler von seinem Transmitter zu trennen beziehungsweise ihm diesen zu entreissen. Wird der Transmitter weit genug vom gegnerischen Spieler entfernen, merkt das das iPhone des Gegners und meldet diesem, dass er ausgeschieden ist. Unzählige Features und Erweiterungen für dieses Spiel wären denkbar. Zum Beispiel könnten zusätzliche Spielmodi

wie "Capture the flag'[1] realisiert werden, wobei wieder ein iBeacon Transmitter die Flagge repräsentieren würde.

Schliesslich haben wir diese Idee wieder verworfen, da der Aufwand, dieses Spiel zu spielen, für ein Smartphone Spiel vergleichsweise hoch ist. Auch kann das Spiel nicht alleine gespielt werden und eine globale Vernetzung der Spieler ist nur schwer zu realisieren. Hinzu kommt, dass zusätzliche Hardware (iBeacon Transmitter) erforderlich ist. Aus diesen Gründen nehmen wir an, dass das Spiel nur schlecht skaliert und somit unseren Ansprüchen nicht genügt.

2.1.2 *Risiko*

Ähnlich dem Brettspiel Risiko[2] wird um Landflächen gekämpft. Die gesamte Welt ist die Spielfläche. Einzelne Landflächen werden durch WLAN Access Points, iBeacon Transmitter und Points of interest[3] unterteilt, die die Grenzen zwischen den Landflächen ziehen. Spieler formieren sich in Teams und versuchen, möglichst viele Grenzpunkte einzunehmen. Wird die totale Grenze einer Landfläche von einem Team beherrscht, gehört ihm die dazugehörige Landfläche.

Im Gegensatz zum Fangen spielen denken wir, dass dieses Spiel um einiges besser skalieren würde. Aufgrund der komplexen Spiellogik ist die Hürde, an diesem Spiel teilzunehmen, jedoch sehr hoch. Hinzu kommt der sehr hohe Implementationsaufwand, der die Komplexität des Spiels mit sich bringt. Ein Spiel zu implementieren, welches nur bei einem kleinen Publikum Anklang findet und zudem einen so erheblichen Aufwand für die technische Realisierung nach sich zieht, empfanden wir als keine gute Idee für eine Semesterarbeit. Deshalb haben wir diese Idee verworfen.

2.1.3 *Quizduell*

Zu den in letzter Zeit erfolgreichsten Spiele für das Smartphone gehört sicherlich Quizduell[4]. Einen grossen Nachteil hat dieses Spiel aber: Man kann nur gegen bekannte Speiler aus einer Freundesliste spielen. Das Spiel könnte aus unserer Sicht noch besser skalieren, wenn auch gegen fremde Spieler, zum Beispiel aus der unmittelbaren Umgebung, gespielt werden könnte. Unsere Idee begrenzte sich daher anfänglich auf ein Quizduell ähnliches Wissensspiel, welches gegen Spieler in der eigenen Umgebung gespielt werden kann. Als Belohnung für eine gewonnene Spielrunde soll es nicht nur Punkte

1 http://de.wikipedia.org/wiki/Capture_the_Flag [3]
2 http://de.wikipedia.org/wiki/Risiko_%28Spiel%29 [4]
3 https://de.wikipedia.org/wiki/Point_of_Interest [5]
4 http://www.quizduell-game.de [6]

geben, sondern zusätzlich eine Liste an neuen Gegnern, gegen welche es anzutreten gilt. Dies sollte unserer Meinung nach die Skalierbarkeit des Spieles erheblich erhöhen.

Eine einfache Kopie von Quizduell war uns dann aber doch zu wenig und wir erweiterten die Idee um einige zusätzliche Konzepte. So soll es zum Beispiel möglich sein, an Multiplayer Spielen an vorgegebenen geographischen Lokationen teilzunehmen. Wie beim Spiel Risiko in Abschnitt 2.1.2 sind diese Punkte durch WLAN Access Points, iBeacon Transmitter und Points of interest realisiert. Ausserdem wird bei jeder Spielrunde eines von mehreren Mini Games ausgewählt, damit der Spieler genug Abwechslung erfährt. Die Anzahl der Spiele kann durch Updates der App erweitert werden.

Aus der obigen Beschreibung kristallisiert sich heraus, dass die Idee aus einem Kern für die einfache Eins gegen Eins Spiellogik und aus weiterer komplexerer Zusatzfunktionalität besteht. Dies lädt dazu ein, ebendiesen Kern als eine Machbarkeitsstudie in der Studienarbeit abzuhandeln und bei Erfolg gegebenenfalls den so erhaltenen Prototypen in der Bachelorarbeit um die zusätzliche Funktionalität zu erweitern. In Absprache mit dem Betreuer haben wir uns schlussendlich für diese modifizierte Variante des Quizduell Spiels entschieden.

2.2 ZIELE

Die Ziele dieser Studienarbeit sind im folgenden zusammengefasst.

2.2.1 *Funktionalität*

- Das Resultat der Semesterarbeit soll ein voll funktionsfähiges Spiel sein, das die Kernidee aus Abschnitt 2.1.3 verwirklicht.
- Es soll möglich sein, sich für das Spiel zu registrieren und die so angelegten Userdaten zu verwalten.
- Das Spiel soll Spieler in der näheren Umgebung des Benutzers ausfindig machen können, gegen welche man dann ein Eins gegen Eins Spiel durchführen kann.
- Als Belohnung für einen Sieg soll der Benutzer die Spielkontakte seines Gegners erhalten, gegen welche er dann aus jeder Entfernung antreten kann.
- Das Spiel soll auf iPhones und iPads ab Betriebssystem iOS8 lauffähig sein.

2.2.2 *Zuverlässigkeit*

- Das Spiel soll ohne reproduzierbare Abstürze spielbar sein. Nur so kann eine gewisse Akzeptanz bei Beta Testern erreicht werden.

2.2.3 *Benutzbarkeit*

- Der Einsteig in das Spiel soll dem Benutzer möglichst einfach gestaltet werden. Innerhalb von Minuten soll ein Beta Tester die Grundlagen des Spieles mit unserer Hilfe verstanden haben.
- Das Spiel soll über eine möglichst grafisch attraktive Oberfläche verfügen, da diese von iOS Usern gefordert wird und die Akzeptanz bei Testern weiter fördert.

2.2.4 *Wartbarkeit & Änderbarkeit*

- Auch wenn nur ein Proof of concept implementiert wird, soll trotzdem ein Layering in verschiedene Schichten vorgenommen werden, um eine übersichtliche Struktur des Codes zu garantieren. Nur so können Teile der Logik in einer weiterführenden Bachelorarbeit wiederverwendet werden.
- Die Applikation soll mit einem automatischen Build System erstellt werden. Dabei sollen auch alle Tests ausgeführt werden und ein Test Coverage Report generiert werden.

2.2.5 *Übertragbarkeit*

- Die Installation der Applikation soll, nach einer Registrierung als Beta Tester, aus dem Testflight App Store[5] möglich sein.

2.2.6 *Erweiterbarkeit*

- 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 erlangt werden.
- Die Relationen zwischen den Modell Klassen sollen so angelegt werden, dass eine Multiplayer Funktionalität ohne Anpassungen an ihnen erforderlich ist.

5 https://www.testflightapp.com [7]

3 USE CASES

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

3.1 CRUD USER

Jeder Benutzer muss sich als erstes bei der Applikation registrieren. Dazu ist mindestens ein Benutzername, eine Mailadresse und ein Passwort nötig. Im eingeloggten Zustand kann der Benutzer jederzeit seine Benutzerdaten abrufen und bis auf den Benutzernamen und die Mailadresse alle seine Daten ändern. Das Löschen des eigenen Benutzerkontos ist dem Benutzer nicht erlaubt. Der Grund dafür ist in Abschnitt 7.1.3 beschrieben.

3.2 INVITE USER

Nach dem erfolgreichen Einloggen hat der Benutzer die Möglichkeit, andere Spieler in der unmittelbaren Umgebung oder über eine Freundesliste zu einem Spiel einzuladen.

3.3 RESPOND INVITATION

Der Benutzer wird über eine Einladung notifiziert und hat die Möglichkeit, diese anzunehmen oder abzulehnen. Das System evaluiert bei einer Annahme der Einladung einen für beide Spieler verfügbaren Spieletyp.

3.4 PLAY GAMEROUND

Wurde eine Einladung von einem Benutzer angenommen, kann der Eingeladene sowie der Einladende die erste Runde des Spiels spielen. Der Spieletyp der n spielbaren Runden wurde bei der Annahme der Einladung bereits vom System evaluiert. Anhand des evaluierten Spieletyps lädt das System die für das Spiel erforderlichen Daten und präsentiert dem jeweiligen Benutzer das Spiel. Der Benutzer muss je nach Spieletyp das Spiel in einer vorgegebenen Zeit beenden. Das System speichert die Resultate jeder Runde und evaluiert nach der letzten Spielrunde einen Sieger. Der Sieger wie auch der Verlierer werden dann über den Ausgang des Spieles notifiziert. Beendet ein

Spieler seine Runde nicht in der vorgegebenen Zeit, bemerkt dies das System und der Spieler verliert diese Runde.

Teil II

ENTWICKLUNGSPROZESS

4 MACHBARKEITSSTUDIEN

Wie in Abschnitt 2.1 beschrieben, haben wir uns während der Brainstorming Phase bewusst keine Gedanken über die technische Umsetzbarkeit möglicher Projektideen gemacht. Dies führte aber dazu, dass wir vor Beginn der Entwurfsphase einige Machbarkeitsstudien durchführen mussten, die die Realisierbarkeit einiger unserer Features aufzeigen sollten. Die wichtigsten dieser Machbarkeitsstudien werden in diesem Kapitel erläutert und die daraus gezogenen Erkentnisse aufgezeigt. Diese sollen unter anderem auch anderen Entwicklern helfen, welche Apps mit einem ähnlichen Kontext entwickeln möchten. An dieser Stelle sei anzumerken, dass die referenzierten Code Beispiele keine Erklärungen enthalten und nur nach Einarbeitung in die verwendeten Frameworks gut genug interpretiert werden können.

4.1 BENUTZER FINDEN & GEFUNDEN WERDEN

Diese erste Machbarkeitsstudie sollte uns aufzeigen, ob es möglich ist, als iBeacon Transmitter zu agieren und gleichzeitig andere iBeacon Transmitter zu erkennen. Ausserdem sollte eruiert werden, wie genau die Distanz zwischen iBeacon Transmittern bestimmt werden kann.

Der Code zu dieser Machbarkeitsstudie kann auf Github[1] in unserem BeaconCatchMe Repository[2] eingesehen werden. Der Code muss für einen Funktionstest auf zwei iOS8 fähigen Geräten installiert werden.

Die Studie hat aufgezeigt, dass es möglich ist, als iBeacon Transmitter zu fungieren und gleichzeitig Signale anderer iBeacon Transmitter zu empfangen, solange sich beide Applikationen im Vordergrund befinden. Das Empfangen ist aber auch beim Betrieb der Applikation im Hintergrund möglich. Das Aussenden eines Signals ist beim Betrieb der Applikation im Hintergrund möglich, jedoch kann dieses unter Verwendung des CoreLocation Frameworks[3] nicht detektiert werden. Die Distanz zwischen zwei iBeacon Transmittern zu bestimmen, gestaltet sich sehr einfach, solange man sich auf die vier vom Framework vorgeschriebenen Stufen (unknown, far, near, immediate) beschränkt. Es ist auch möglich, aus der Signalstärke des Senders eine Distanz in Meter zu berechnen. Diese Methode ist aber gemä-

1 https://github.com [8]
2 https://github.com/pipeaesac/BeaconCatchMe
3 https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework [9]

ss unseren Messungen ziemlich ungenau, vor allem dann, wenn sich Hindernisse zwischen Sender und Empfänger befinden.

4.2 BENUTZER FINDEN IM HINTERGRUND

Aus den Ergebnissen der ersten Studie haben sich sofort neue Fragestellungen ergeben. So wollten wir wissen, ob es möglich ist, Signale, die beim Betrieb der Applikation im Hintergrund ausgesendet werden, unter Verwendung des CoreBluetooth Framework[4] zu detektieren.

Der Code zu dieser Machbarkeitsstudie kann auf Github in unserem BeaconCatchmeCB Repository[5] eingesehen werden. Auch hier mus der Code für einen Funktionstest auf zwei iOS8 fähigen Geräten installiert werden.

Nach vielen Versuchen kamen wir zum Schluss, dass das Detektieren der Signale möglich ist. Allerdings enthalten diese Signale zu wenig Information, um damit einen Benutzer der Applikation eindeutig zu bestimmen. Im Vordergrund ausgesendete Signale enthalten eine UUID, über die das Signal unserer Applikation zugeordnet werden kann. Jedoch weist das Signal keine Information zur eindeutigen Bestimmung des Benutzers auf. Der Gerätename, der ebenfalls zum Inhalt des Signals gehört, kann nicht zur Benutzeridentifikation verwendet werden, weil er nicht eindeutig ist. Das Empfangen von Signale unter Verwendung des CoreBluetooth Frameworks eignet sich deshalb nicht für unsere Applikation.

Das beschriebene Signal ist in der folgenden Abbildung ersichtlich. Die erste UUID wurde nicht mit dem Signal mitgeliefert, sondern vom Empfänger des Signals generiert (lokale UUID). Die zweite UUID entspricht der vom Sender in der Applikation registrierten UUID:

```
2014-12-04 16:42:46.916 BeaconCatchMe[822:60370]
did discover peripheral with ID: "C491FA7A-3DFF-990B-75A9-F0C20632B6E6",
name: Max Musters iPhone6, data: {
    kCBAdvDataIsConnectable = 1;
    kCBAdvDataLocalName = myPeripheral;
    kCBAdvDataServiceUUIDs = (
    "F9F9B32A-7C1E-4170-B9B8-66A12F65A9CD"
    );
}, -51.00
```

Die nachfolgende Abbidlung zeigt ein Signal, das beim Betrieb der

4 https://developer.apple.com/Library/ios/documentation/CoreBluetooth/Reference/CoreBluetooth_Framework [10]

5 https://github.com/cakl/BeaconCatchmeCB

Applikation im Hintergrund ausgesendet wird. Die vom Sender in der Applikation registrierte UUID ist hier nicht mehr ersichtlich:

```
2014-12-04 16:43:27.271 BeaconCatchMe[830:60728]
did discover peripheral with ID: "C491FA7A-3DFF-990B-75A9-F0C20632B6E6",
name: Max Musters iPhone6, data: {
    kCBAdvDataIsConnectable = 1;
}, -56.00
```

4.3 ENTDECKEN OHNE UUID

Bei einigen möglichen Features spielten wir mit dem Gedanken, bestehende iBeacon Transmitter, welche in anderen Applikationen verwendet werden, für unsere Applikation zu nutzen. Wir wollten testen, ob es machbar ist, einen iBeacon Transmitter als solchen zu erkennen, ohne dass dessen UUID im Voraus bekannt ist.

Für diese Machbarkeitsstudie wurde wieder der Code aus den Github Repositories BeaconCatchme und BeaconCatchmeCB verwendet.

Das CoreLocation Framework benötigt immer eine UUID, damit iBeacon Transmitter als solche entdeckt werden können. Zudem können pro Applikation nur 20 verschiedene UUIDs gleichzeitig registriert werden. Deshalb ist das CoreLocation Framwork für die oben genannte Aufgabe nicht brauchbar. Mit dem CoreBluetooth Framework ist es möglich, Signale jeglicher Geräte zu empfangen. Hierbei gilt jedoch die Einschränkung, dass nur Signale, welche auch mit CoreBluetooth ausgesendet wurden, empfangen werden können. Es ist nicht möglich, iBeacon Transmitter, welche mittels des CoreLocation Frameworks ihre Signale aussenden, mithilfe des CoreBluetooth Frameworks zu empfangen. Es ist aus unserer Sicht also nicht möglich, bestehende iBeacon Transmitter für unsere eigene Applikation zu nutzen.

4.4 BACKEND AS A SERVICE

Damit wir uns vollumfänglich auf die Entwicklung der iOS Applikation konzentrieren konnten, war es uns wichtig, ein Backend as a Service einzusetzen. Wir entschieden uns für diesen Zweck, Parse[6] sowie CloudKit[7] genauer zu betrachten. Das Backend soll es möglich machen, n:m Relationen zwischen Domänenobjekten zu realiseren, Push Notifikationen zu versenden und letztlich auch Server Side Logik ausführen zu können.

6 https://www.parse.com [11]
7 https://developer.apple.com/library/ios/documentation/CloudKit/Reference/CloudKit_Framework_Reference [12]

Schon nach wenigen einfachen Beispielprogrammen war klar, dass Parse diese Bedingungen nicht nur erfüllt, sondern auch weitere Features, wie zum Beispiel eine sehr einfache Userverwaltung, bereitstellt. Zudem ist Parse sehr intuitiv gestaltet und eine rege Community, sowie eine aktuelle Dokumentation machen die Einbettung von Parse in die eigene Applikation sehr einfach. CloudKit hingegen wurde erst mit iOS8 vorgestellt, weshalb sich nur sehr wenige Anleitungen dazu finden lassen. Hinzu kommt, dass bei einem so jungen Framework oft noch viele Fehler vorhanden sind. Deswegen entschieden wir uns gegen CloudKit.

Es sei an dieser Stelle zu erwähnen, dass Parse nur limitiert kostenfrei verfügbar ist. Die Grenze befindet sich bei 30 Anfragen pro Sekunde über alle Benutzer hinweg gesehen und einem Speicherplatz von 20GB. Gegen monatliche Bezahlung kann diese Begrenzung nach oben verschoben werden. Sollte der Fall auftreten, dass unsere Benutzeranzahl diese Grenzen überschreitet, sind im Abschnitt 8.5 Ideen aufgelistet, wie wir die Rentabilität unserer Applikation sicherstellen können.

4.5 POINTS OF INTEREST

Die letzte Machbarkeitsstudie sollte aufzeigen, ob es ein Framework gibt, das auf einfache Weise Points of interest in der unmittelbaren Umgebung des Benutzers aufzeigt. Die Idee dabei ist, mögliche Locations zu bestimmen, an welchen dann Multiplayer Spiele ausgetragen werden können.

Der Code zu dieser Machbarkeitsstudie kann auf unserem Github Repository PointsOfInterestTest[8] eingesehen werden.

Das MapKit[9] Framework von Apple bietet die Möglichkeit, solche Points of interest abzufragen. Unsere Tests zeigten jedoch auf, dass nur wenige Points verfügbar sind und die Kategorisierung (Restaurant, Lebensmittelladen, Museum, usw.) zudem sehr heterogen ist. Trotzdem reichen die Daten aus, um zu erkennen, ob sich ein User in der Nähe eines Points of interest befindet, an welchem dann Multiplayer Spiele gespielt werden könnten.

8 https://github.com/cakl/PointsOfInterestTest

9 https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference [13]

5 ARCHITEKTUR

In diesem Kapitel gehen wir auf die Architektur unserer Applikation ein. Es wird kurz das Layering beschrieben und wichtige Entscheidungen bei der Architekturfindung aufgezeigt. Interessante Details zur Implementierung sind im Kapitel 9 zu finden.

5.1 LAYERING

Obwohl es sich bei der Applikation Ludur nur um einen Proof of concept handelt und somit die Strukturierung des Codes nicht im Vordergrund steht, ist dieser dennoch in drei Layer aufgeteilt. Diese Aufteilung wurde vorgenommen, um stets den Überblick über das Projekt zu behalten. In den folgenden Abschnitten werden die drei Layer genauer erklärt.

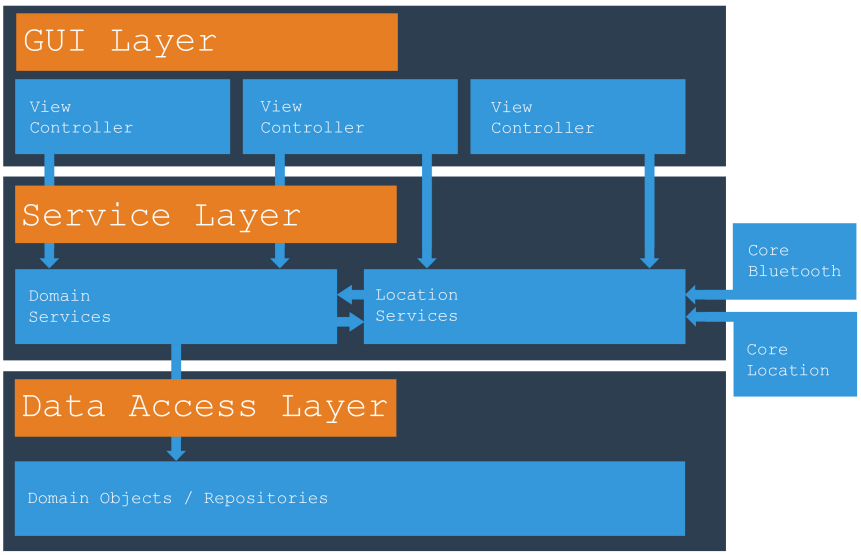


Abbildung 1: Layering

5.1.1 *Data Access Layer (DAL)*

Dieser Layer ist zuständig für den Zugriff auf den Backend as a service Provider Parse. Das Parse SDK bietet eigene Klassen, welche sehr einfach synchron oder asynchron im Backend gespeichert beziehungsweise gelesen werden können. Im Backend werden diese Klassen in Tabellen abgebildet und sind über eine Weboberfläche einsehbar. Der DAL besteht deshalb mehrheitlich aus Domänen Klassen

(in Abschnitt A.2 genauer beschrieben), welche Parse Klassen vererben und zusätzlich eine domänenspezifische Logik in Methoden implementieren. Hinzu kommen einige Klassen, welche das Repository Pattern[1] sowie das Factory Pattern[2] implementieren. Diese Klassen werden benötigt, um Domänen Klassen für darüberliegende Layer zur Verfügung zu stellen. Um eine gewisse Austauschbarkeit des Data Access Layer zu garantieren, sind alle oben genannten Klassen gegen ein Interface programmiert und die darüberliegenden Layer haben keine Abhängigkeiten in das Parse SDK. Ein Austausch des Backend as a service Provider soll so ermöglicht werden.

5.1.2 *Service Layer*

Der Service Layer unterteilt sich in zwei Teile. Der erste Teil enthält Service Klassen, welche Domänen Objekte an den darüberliegeden Layer weiterreichen und zustandslose Operationen auf den Domänen Objekten ausführen. Oft wird in diesen zustandslosen Operationen domänenspezifische Logik aggregiert, um so den darüberliegenden Layer zu vereinfachen beziehungsweise die Komplexität zu reduzieren. Der zweite Teil enthält Service Klassen, welche mithilfe der Frameworks CoreBluetooth und CoreLocation ein Beacon Signal aussenden beziehungsweise empfangen können. Der darüberliegende Layer wird mithilfe eines Delegates (näher beschrieben im Abschnitt **??**) über empfagene Signale benachrichtigt. Weiterhin enthalten diese Service Klassen Logik, um zu überprüfen, ob die benötigten Systemdienste (Lokationsdienste, Bluetooth) über ausreichende Rechte verfügen und fordern gegebenfalls eine neue Autorisierung beim Benutzer an.

5.1.3 *GUI Layer*

Der GUI Layer wird, wie vom UIKit vorgeschrieben, klassisch mit dem MVC-Pattern implementiert. Die verschiedenen Views wurden in einer Storyboard Datei[3] deklariert. Ausnahmen sind die Views der Spiele, welche in Zukunft für dieses Projekt erstellt werden. Genauere Details zu dieser Trennung sind unter Architekturentscheidungen 5.2.2 beschrieben. Zu jeder View gehört eine Controller Klasse, welche, wenn nötig, das Model beim Service Layer laden sowie Interaktionen des Users verarbeiten kann. Änderungen an den mit dem Model geladenen Objekten können dann wiederum an den jeweiligen Service weitergereicht werden.

---

1 http://de.wikipedia.org/wiki/Repository_%28Entwurfsmuster%29 [14]
2 http://de.wikipedia.org/wiki/Fabrikmethode [15]
3 https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIStoryboard_Class/index.html [16]

5.2 ARCHITEKTURENTSCHEIDUNGEN

5.2.1 *Threading*

Das Parse SDK bietet bei allen Methoden, welche eine Verbindung zum Backend aufbauen, eine synchrone und eine asynchrone Variante an. Da eine Verbindung zum Backend einige Zeit benötigt und auf eine Antwort des Backends gewartet werden muss, sollte dies asynchron zum Main Thread geschehen. Es gibt zwei Möglichkeiten dies zu bewerkstelligen:

1. Man erstellt selbst einen neuen Thread und ruft Methoden, welche eine Verbindung aufbauen, in diesem Thread synchron auf. Der Nachteil daran ist, dass man sich selbst um die Threaderstellung und Threadsynchronistation mit dem Main Thread kümmern muss.

2. Alle Methodenaufrufe, welche eine Verbindung aufbauen, werden asynchron aufgerufen. Oft werden jedoch mehrere Methoden, welche eine Verbindung aufbauen, hintereinander aufgerufen. Falls die Threads nicht recycled werden, wird für jede solche Methode ein neuer Thread erstellt. Dies kann zu einem grossen Overhead führen. Hinzu kommt, dass der Code mit vielen verschachtelten Blöcken[4] unleserlich und unnötig komplex wird.

Trotzdem haben wir uns für die zweite Variante entschieden, da wir beide noch nicht viel Erfahrung mit dem Grand Central Dispatch[5] haben und wir uns deshalb nicht selbst um das Threading kümmern wollten. Die Usability der Applikation wird durch die vermeintlich schlechtere Performance aus unsererer Sicht nicht zu stark beeinträchtigt. Da Ludur nur ein Proof of concept ist, können wir auch den stellenweise unschönen Code akzeptieren. Dies bietet natürlich noch Potential für Verbesserungen. Die Sauberkeit und die Komplexität des Codes im Data Access Layer würde somit stark gesteigert. Auch die Möglichkeit, Aufrufe an das Backend zu sammeln und erst dann auszuführen, wenn wirklich nötig, wurde in der aktuellen Lösung zu wenig betrachtet. Dadurch könnte eine Minimiering der Backend Aufrufe erzielt werden. Da aber oft Abhängigkeiten zwischen den Anfragen an das Backend bestehen, könnte diese Möglichkeit schwer zu implementieren sein.

4 https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithBlocks/WorkingwithBlocks.html [17]

5 https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/index.html [18]

5.2.2 *Spielschnittstelle*

Schon zu einem frühen Zeitpunkt hatten wir uns entschieden, dass unsere Applikation verschiedene Spiele, für jedes Duell zwischen zwei Benutzern, auswählen kann. Es ist uns deshalb wichtig, dass man bei Updates der Applikation zusätzliche Spiele hinzufügen oder bestehende Spiele mit neueren Versionen ersetzen kann. Dabei soll die restliche Applikationslogik nicht angepasst werden müssen. Da jedoch alle Views der Applikation in der Storyboard Datei deklariert werden, müsste man die Views von neuen oder aktualisierten Spielen in die Storyboard Datei einfügen. Dies ist nur über mühsames Copy-und-Paste oder über das Einfügen der Views im XML Source der Storyboard Datei möglich. Eine Alternative zur Definition der Views im Storyboard File sind Xcode Interface Builder Files (xib). Diese bieten die Möglichkeit, Views unabhängig vom Storyboard File zu deklarieren. Views der Spiele müssen aus diesem Grund in einem eigenen xib File deklariert werden. Zusätzlich muss ein Spiel ein von uns vorgegebenes Interface implementieren. Dieses Interface beinhaltet lediglich eine Methode, um das Spiel zu starten. Durch diese Bedingungen gelingt eine für uns zufriedenstellende Entkopplung der Spiele und dem Rest der Applikationslogik.

Theoretisch wäre es dadurch möglich, dass andere Spieleentwickler Ludur mit ihren Spielen erweitern. Wir sehen dies aber nicht als realistisch, da der ganze Code von uns kompiliert werden muss. Andere Spieleentwickler müssten uns also ihren Code offenlegen. Hinzu kommen die rechtlichen Probleme, die dadurch entstehen würden. Vielleicht wäre dieses Problem mit einer Lösung basierend auf den in iOS8 eingeführten Extensions[6] zu umgehen. Aus zeitlichen Gründen wurde dies nicht mehr weiter geprüft.

5.2.3 *Verzicht auf Parse ViewController*

Das Parse SDK bietet dem Entwickler die Möglichkeit, bereits vorgefertigte ViewController für fast alle denkbaren Szenarien in den Code zu integrieren. Deshalb sind wir bei unserer Recherche auf viele Beispiele gestossen, die diese Parse ViewController direkt einsetzen und auch die von Queries erhaltenen Parse Objekte bis in den GUI Layer mit hochgeben. Dies bedeutet aber aus Architektursicht eine extrem hohe Abhängigkeit der eigenen Applikation zum Parse SDK.

Diese Abhängigkeit ist uns aber entschieden zu streng. Parse ist zwar unsere erste Wahl als Backend as a Service Provider, jedoch behielten wir auch immer CloudKit als Alternative im Auge. Sollten wir uns

6 https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/index.html [19]

also zu einem beliebigen Zeitpunkt entscheiden, von Parse SDK zu CloudKit zu wechseln, so sollte dies unserer Meinung nach möglich sein, ohne dass die ganze Applikation umegschrieben werden muss.

Wir haben uns deshalb entschieden, komplett auf die von Parse SDK bereitgestellten ViewController zu verzichten und auch keine Parse Objekte über die Grenzen des Data Access Layers hinaus zu verwenden. Diesen Schnitt zu ziehen, ist mit erheblich mehr Programmierarbeit verbunden, muss aber unserer Meinung nach gemacht werden.

6 ENTWICKLUNGSUMGEBUNG

Dieses Kapitel gibt einen Überblick über die von uns verwendete Entwicklungsumgebung.

6.1 TESTING

Bei einem Proof of concept Projekt eine sehr hohe Test Coverage anzustreben, macht nicht viel Sinn. Deshalb wird nur die wichtigste Funktionalität im untersten Layer getestet. Im untersten Layer darum, weil mit diesem begonnen wurde und dieser am Anfang des Projektes ein stabiles Grundgerüst für die oberen Layer darstellen sollte. Die somit erreichte Test Coverage beträgt 20%. Die Ausführung der Tests geschieht vollautomatisiert auf dem Build Server. Beim Ausführen werden gcov[1] Dateien erstellt und anschliessend vom Build Server an Coveralls[2] übertragen. Coveralls erstellt daraus eine Test Coverage Statistik, welche über eine Weboberfläche eingesehen werden kann.

6.2 ARCHITEKTURENTSCHEIDUNGEN

Das Ziel ist es, dass das Deployment des auf dem Build Server erstellten Paketes automatisch erfolgt. Das ist aber nicht so einfach möglich, wenn das Paket vor dem Deployment noch signiert werden muss, was bei iOS Applikationen der Fall ist. Hinzu kommt, dass kein Entwickler seinen zum Signieren verwendeten Private Key gerne in das Repository eincheckt. Mit dem Build Server von Travis Ci ist es möglich, dem Build Server Passwörter mitzuteilen. Somit kann der Private Key verschlüsselt im Repository abgelegt werden. Der Build Server entschlüsselt den Private Key und kann am Ende des Build Durchgangs das Paket mit dem Key signieren. Anschliessend wird das signierte Paket auf die Testflight Plattform hochgeladen und die Beta Tester mit einem Downloadlink über das neue Release informiert. Die Konfiguration des Build Servers und des Deployments wurde mithilfe einer Anleitung von objc.io[3] erstellt. Eine genaue Übersicht über den Build Verlauf lässt sich in den Implementierungsdetails im Kapitel 9 finden.

1 https://gcc.gnu.org/onlinedocs/gcc/Gcov.html [20]
2 https://coveralls.io[21]
3 http://www.objc.io/issue-6/travis-ci.html [22]

6.3 TOOLS

- Xcode IDE - Entwicklungsumgebung von Apple für iOS Applikationen
  https://developer.apple.com/xcode/ide
- xctool - Build System
  https://github.com/facebook/xctool
- Cocoapods - Dependency Management System
  http://cocoapods.org
- Travis-ci Build Server und Continous Integration System
  https://travis-ci.com
- Coveralls - Automatische Test Coverage Auswertung
  https://coveralls.io
- cpp-coveralls - Python Tool für Code Coverage Upload an Coveralls
  https://github.com/eddyxu/cpp-coveralls
- Testflight - Beta Testing Platform
  https://www.testflightapp.com
- Dash - Dokumentations Browser und Snippet Manager
  http://kapeli.com/dash
- PyYAML - YAML Implementation in Python, wird von cpp-coveralls benötigt um Coveralls YAML Konfiguration 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 einfach Erstellung von tabellenbasierten Formularen
  https://github.com/nicklockwood/FXForms
- SDWebImage - Library um Bilder asynchron herunterladen und cachen zu können
  https://github.com/rs/SDWebImage

Teil III

DAS SPIEL

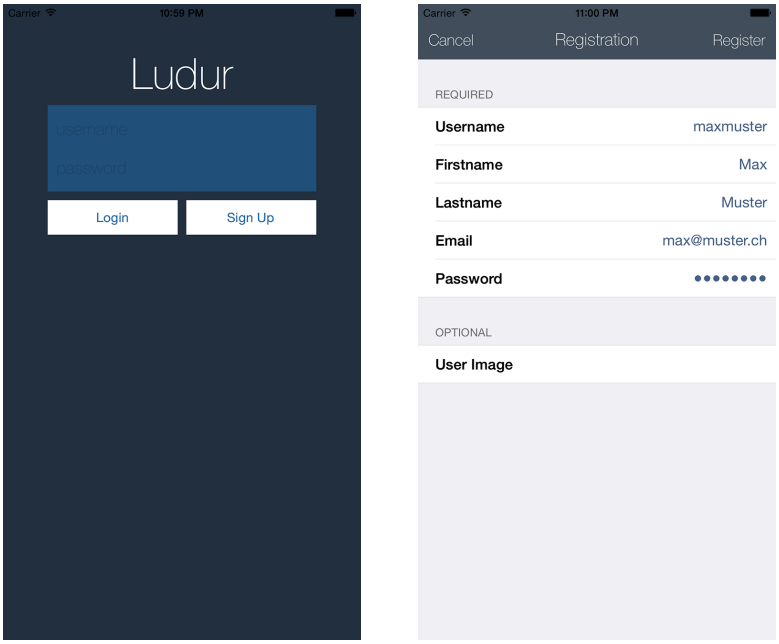
7 FEATURES

Dieses Kapitel gibt Einblick in die von uns im Rahmen dieser Arbeit implementierten Features. Die nachfolgenden Erläuterungen sollen dem Leser ermöglichen, die Applikation ohne weitere Erklärungen bedienen zu können. Die den Features zugrunde liegenden Use Cases wurden in Kapitel 3 definiert.

7.1 BENUTZERVERWALTUNG

7.1.1 *Benutzer anlegen*

Wird die Applikation gestartet, findet sich der Benutzer auf einem Login Bildschirm wieder. Bei der erstmaligen Benutzung muss der Benutzer zuerst auf dem im Loginbildschirm verlinkten Registrierungsbildschirm seine gewünschten Benutzerdaten einfügen. Ein Benutzername, der Vor- und Nachname des Benutzers, eine Mailadresse und ein Passwort ist dabei zwingend.



(a) (b)

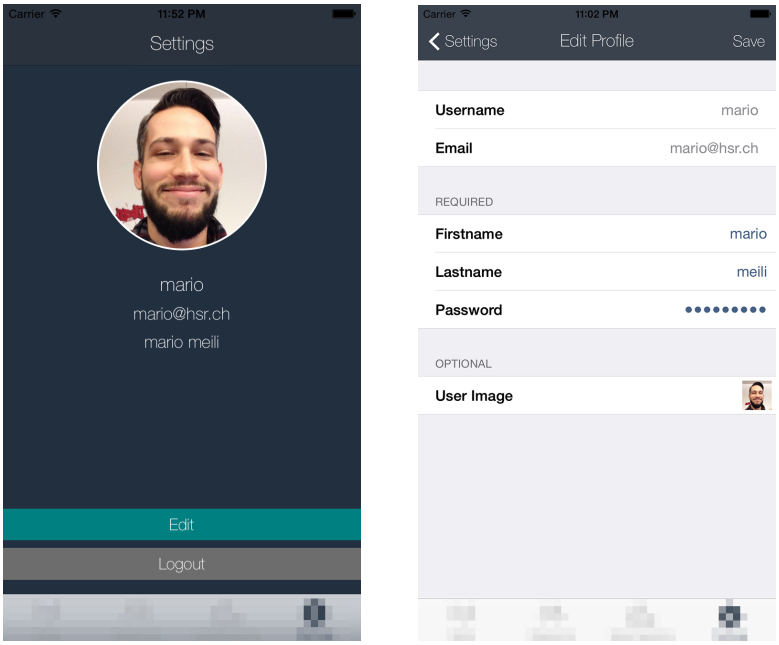
Abbildung 2: Login- (a) und Registrierungsbildschirm (b)

Optional kann bereits zu diesem Zeitpunkt ein Benutzerfoto aufgenommen oder aus der Gallerie des Benutzers ausgewählt werden.

Beim Klick auf den Registrierungsbutton werden die Daten auf dem Backend mit einer eindeutigen Identifikationsnummer versehen und persistiert. Der Benutzer wird zurück auf den Login Bildschirm geleitet. Durch Angabe seines Benutzernamens und seines Passwortes kann sich der Benutzer nun in die Applikation einloggen.

7.1.2 *Benutzerdaten editieren*

Ist der Benutzer eingeloggt, kann der Benutzer in der Ansicht "Einstellungen" seine Benutzerdaten nach seinen Wünschen anpassen. Davon ausgenommen sind die Mailadresse und der Benutzername, da diese einen Wiedererkennungswert unter den anderen Applikationsbenutzern haben. Falls der Benutzer bei der Registrierung kein Benutzerfoto definiert hat, kann er das jederzeit in dieser Ansicht nachgeholt werden.



(a) (b)

Abbildung 3: Settings- (a) und Editierbildschirm (b)

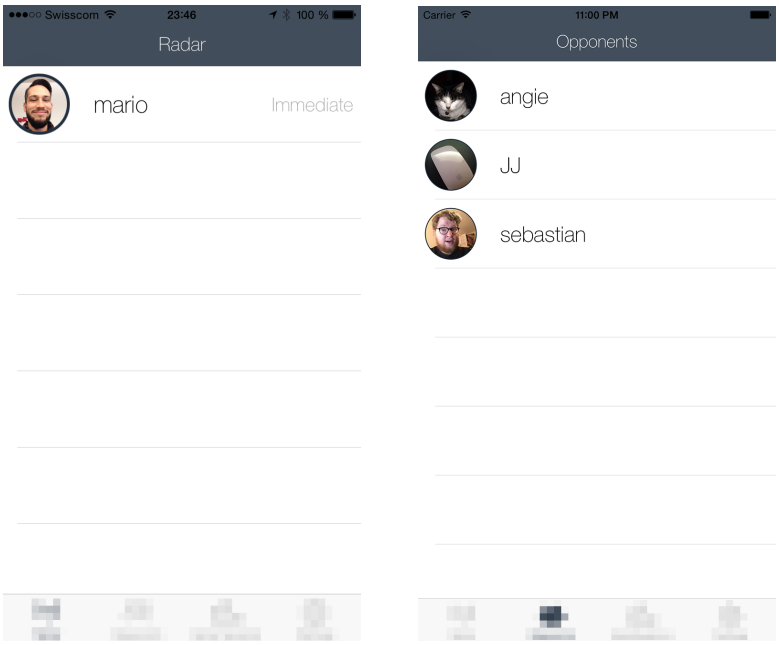
7.1.3 *Benutzer löschen*

Bei der Benutzung der Applikation wird der Benutzer auf allen absolvierten Spielsessionen und den dazugehörigen Spieleinladungen referenziert. Um Inkonsistenzen im Backend zu verhindern, haben wir uns deshalb entschieden, dass das Löschen des eigenen Benutzerkontos nicht möglich ist. Der Delete User Use Case wurde deshalb verworfen.

7.2 BENUTZER FINDEN

7.2.1 *Benutzer in der näheren Umgebung*

Da jeder Benutzer der Applikation als iBeacon Transmitter fungiert, ist es möglich, Benutzer in der näheren Umgebung zu finden. Benutzer, die ihre Applikation im Vordergrund ausführen, geben ihre Anwesenheit mit einem iBeacon Signal bekannt. Befindet sich die Applikation eines anderen Benutzers beim Empfang eines solchen Signals im Hintergrund, so erhält der empfangende Benutzer eine Push-Nachricht. Diese zeigt, dass sich jemand in seiner Nähe befindet. Die so gefundenen Benutzer werden in der Ansicht "Radar" mit Benutzerfoto und Benutzernamen angezeigt. Ausserdem pulsiert das Foto des Benutzers, um zu verdeutlichen, dass sich der Benutzer in der unmittelbaren Umgebung befindet. Entfernt sich ein Benutzer aus der unmittelbaren Umgebung, verschwindet er wieder vom Bildschirm.



(a) (b)

Abbildung 4: Radar- (a) und Gegnerbildschirm (b)

7.2.2 *Freigespielte Benutzer*

Wie in Abschnitt 7.3.4 beschrieben, kann ein Benutzer bei jeder Spielsession andere Benutzer freispielen. Diese Benutzer werden in der Ansicht "Gegner" angezeigt. Befinden sich diese Benutzer gleichzeitig in der unmittelbaren Umgebung, pulsiert deren Benutzerfoto.

7.3 SPIEL ABSOLVIEREN

7.3.1 *Benutzer einladen*

In den Ansichten "Radar" 7.2.1 und "Gegner" 7.2.2 können Benutzer eingeladen werden. In beiden dieser Ansichten kann ein angezeigter Benutzer angewählt werden, worauf sich ein neuer Bildschirm öffnet. Auf diesem wird das Benutzerfoto und der Benutzername, sowie ein Button zum Versenden einer Einladung des ausgewählten Benutzers angezeigt. Durch das Betätigen dieses Buttons wird die Einladung im Backend persistiert und eine Push-Nachricht an den Benutzer verschickt, die ihn über diese Einladung informiert.

7.3.2 *Einladung beantworten*

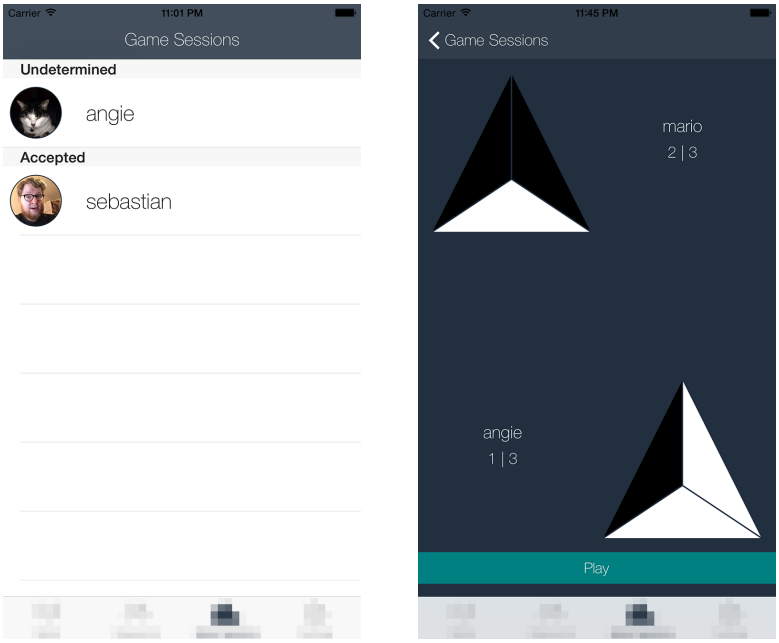
Unbeantwortete Einladungen kann der Benutzer in der Ansicht "Spielsessionen" finden. Die dort dargestellte Tabelle kann bis zu drei Sektionen beinhalten. Die Einladungen werden in der obersten dieser Sektionen angezeigt. Der Benutzer hat nun die Möglichkeit, eine Einladung anzunehmen oder abzulehnen. Dies kann er erreichen, indem er entweder von rechts in die gewünschte Tabellenzeile hinein swiped (so werden die Optionen Annehmen und Ablehnen auf der Zeile eingeblendet) oder die Tabellenzeile anwählt. Bei der zweiten Variante öffnet sich ein neuer Bildschirm, auf dem das Benutzerfoto und der Benutzername des Gegners, sowie je ein Button für das Annehmen oder Ablehnen der Einladung zu finden ist.

Lehnt der Benutzer die Einladung ab, wird der zugehörige Eintrag aus der Tabelle gelöscht. Im Backend wird der Status der Einladung auf abgelehnt gesetzt, anstatt die Einladung zu löschen. Dies könnte zu einem späteren Zeitpunkt, zum Beispiel für Auswertungen, nützlich sein. Nimmt der Benutzer die Einladung an, wird der zugehörige Tabelleneintrag ebenfalls entfernt. Der Status der Einladung wird im Backend aber auf angenommen gesetzt. Gleichzeitig wird eine Spielsession generiert und persistiert. Diese Spielsession wird nun im zweit obersten Abschnitt der Tabelle angezeigt, nämlich unter den offenen Spielsessionen.

7.3.3 *Spielsessionen durchführen*

Offene Spielsessionen sind, wie in Abschnitt 7.3.2 beschrieben, in der Ansicht "Spielsessionen" in der zweiten Sektion der Tabelle aufgelistet. Dies ist bei beiden Benutzern, die an der Spielsession beteiligt sind, der Fall. Beide Benutzer können nun zu der Detailansicht der Spielsession gelangen, indem sie die zugehörige Tabellenzeile anwählen.

Die Benutzer können nun abwechselnd ihre drei Runden absolvieren, die es benötigt, um eine komplette Spielsession zu beenden. Bereits absolvierte Runden werden in diesem Bildschirm durch Dreiecksstücke visualisiert, wobei für jede absolvierte Runde eines dieser Stücke gefüllt wird. Es wird eine Progressionsdynamik erzeugt. Das heisst, der Benutzer wird ermutigt, alle Runden zu absolvieren, weil sich dadurch seine Fortschrittsanzeige immer weiter vervollständigt.



(a) (b)

Abbildung 5: Spielrunden- (a) und Spielbildschirm (b)

Erst wenn alle Runden absolviert sind, wird für jede einzelne dieser Runden der Sieger ausgewertet und die Dreiecksstücke neu eingefärbt. Dabei erhalten gewonnene Runden eine andere Farbe als verlorene Runden. So ist der Endstand der Spielsession sofort für beide Benutzer ersichtlich. Zuletzt wird auf der Spielsession der Gewinner gesetzt. Im Backend wird die Spielsession dann als absolviert markiert.

7.3.4 *Belohnungs- & Punktesystem*

Belohnt wird der Benutzer immer dann, wenn er eine Spielsession als Sieger hinter sich gebracht hat. Er erhält dann alle freigespielten Benutzer seines Gegenspielers, gegen welche er dann wiederum eine neue Spielrunde spielen kann. Zusätzlich hat der Benutzer seinen Gegenspieler freigespielt und behält diesen persistent. Verliert der Benutzer eine Spielsession, so verliert er auch alle freigespielten Benut-

zer, die er mit seinem Gegenspieler gemeinsam hat. Dies ist nicht der Fall, wenn der freigespielte Benutzer persistent ist. Die in verschiedenen Ansichten angezeigte Punktezahl eines Benutzers ist gleich der Anzahl seiner freigespielten Gegenspielern.

Mit einer höheren Punktezahl wächst das Ansehen des Benutzers in der Gemeinschaft der Applikationsbenutzer. Dies hat meistens einen positiven Einfluss auf die gesamte Spieldynamik, denn der Benutzer hat nun einen Grund, immer wieder neue Spielsessionen durchzuführen.

8 ERWEITERUNGSMÖGLICHKEITEN

Das Use Case Diagramm im Appendix im Abschnitt A.1 zeigt neben den von uns im Rahmen dieser Arbeit umgesetzten Use Cases auch noch weitere Use Cases auf. Diese wurden als Bestandteil der Applikation festgelegt. Allerdings war uns klar, dass die Zeit für eine Umsetzung dieser Use Cases nicht reichen würde und wir entschieden uns, nur den in Kapitel 3 definierten Kern an Funktionalität umzusetzen.

Die nicht umgesetzten Use Cases sind in diesem Kapitel als Erweiterungsmöglichkeiten der Applikation aufgelistet und würden bei einer allfälligen Fortsetzung der Arbeit einen wesentlichen Beitrag zur Optimierung der Spieldynamik leisten.

8.1 MULTIPLAYER SPIELE

Im umgesetzten Kern der Applikation werden bisher nur die Geräte der Benutzer als iBeacon Transmitter eingesetzt. Eine mögliche Erweiterung der Applikation verwendet stationäre iBeacon Transmitter, um Lokationen für Multiplayer Spiele anzuzeigen. An diesen Spielen soll nur teilgenommen werden können, wenn sich ein Benutzer in der unmittelbaren Nähe des zugehörigen iBeacon Transmitters befindet. Eine Einladung zu diesem Spiel erfolgt durch das Backend, sobald sich eine gewisse Anzahl Benutzer in der Nähe des Transmitters befinden. Diese Anzahl wird bestimmt durch die durchschnittliche Besucherzahl der Lokation. Um die Besucherzahl zu bestimmen, muss sich jeder Benutzer an der Lokation einchecken, unabhängig davon, ob er an einem Multiplayer Spiel teilnimmt oder nicht. Benutzer können von sich aus keine Multiplayer Spiele auslösen. Der Sieger des Spiels erhält die freigespielten Benutzer aller Spielteilnehmer. So kann man Benutzer dazu bewegen, sich zu bestimmten Zeiten an bestimmten Lokationen einzufinden, was sich wiederum positiv auf die Spieldynamik auswirkt.

8.2 TEAMBILDUNG

Die in unseren Augen interessanteste Erweiterungsmöglichkeit ist, den Benutzern der Applikation das Bilden von Teams zu erlauben. Benutzer können so von den Erfolgen anderer Gruppenmitgliedern profitieren und ihren persönlichen Punktestand schneller und einfacher verbessern. Da der Beitritt zu einem Team aber nur unter Zustim-

mung der bestehenden Teammitglieder erfolgen kann, ist es im Interesse des Benutzers, sein persönliches Ansehen stetig zu erhöhen. Es ist sogar denkbar, dass man innerhalb eines Teams verschiedene Zuständigkeiten oder Positionen einnehmen kann, um somit zusätzliche Punkte zu erhalten. Zuletzt sollte es möglich sein, andere Teams zu einem direkten Duell (mit dem eigenen Team) herauszufordern. Von einem Sieg würden dann nicht nur die Repräsentanten des Teams, die am Spiel teilnehmen, belohnt, sondern das ganze Team.

Es ist zu vermuten, dass durch die Einführung dieses Use Cases die Vernetzung von Benutzern der Applikation erheblich schneller und in grösserem Ausmass geschehen würde als beim bisher umgesetzten Einzelspieler Use Case.

8.3 PRIVATSPHÄRE

Eine weitere mögliche Erweiterung der Applikation ist die Verwaltung der Privatsphäre. Dabei sollte es Benutzern zwar möglich sein, sich vor anderen Benutzern versteckt zu halten, jedoch nur soweit, dass die Skalierbarkeit der Applikation nicht zu sehr darunter leidet. So wäre es zum Beispiel denkbar, dass aufdringliche Benutzer gezielt blockiert werden können. So würde dann bei beiden Benutzern keine Benachrichtigung verschickt, wenn sich der jeweils andere Benutzer in der näheren Umgebung befindet. Auch in den Ansichten "Radar" und "Gegner" dürften diese Benutzer gegenseitig nicht mehr erscheinen.

8.4 CHAT

Um die Interaktion zwischen Benutzern zu fördern, war unsere Idee, einen Chat einzuführen, der während jeder Spielsession geführt werden könnte. So könnte man zwischen den einzelnen Runden Nachrichten austauschen. Dadurch wäre es möglich, sich an einem Punkt zu treffen und seine Identität offenzulegen, wenn man sich sowieso schon in unmittelbarer Nähe befindet. Man könnte den Chat aber auch unabhängig von den einzelnen Spielsessionen implementieren, sodass der alte Stand der Konversation beibehalten bleibt.

8.5 RENTABILITÄT

Wie in Abschnitt 4.4 beschrieben, kann es sein, dass die Benutzung des Backends bei hoher Beanspruchung kostenpflichtig wird. Um diese Kosten auszugleichen, wären folgende Erweiterungen der Applikation möglich:

- Werbung innerhalb der Applikation einbinden.
- Einkäufe innerhalb der Applikation, die es einem zum Beispiel erlauben, die oben genannte Werbung wieder zu entfernen.
- Die Applikation kostenpflichtig anbieten.
- Grundfunktionalität gratis anbieten und zusätzliche Funktionalität kostenpflichtig anbieten.

Teil IV

IMPLEMENTIERUNGSDETAILS

9 IMPLEMENTIERUNGSDETAILS

Dieses Kapitel beschreibt einige Details zu Implementierungen, die aus unserer Sicht spezielle Lösungen auf komplexe Probleme darstellen und somit auch für weitere oder weiterführende Projekte von Interesse sein könnten.

9.1 MAJOR- & MINOR-NUMBERING

Das Spiel Ludur setzt voraus, dass andere Benutzer in der unmittelbaren Umgebung zu finden sind. Um dies möglich zu machen, muss jeder Benutzer eindeutig identifizierbar sein. Zugleich muss diese eindeutige Identifizierung mithilfe von iBeacon Transmittern ausgesendet werden können. Der iBeacon Standard bietet hierfür ein Advertisement Paket[1], in welchem eine UUID (16 Bytes), eine Major- und eine Minor-Zahl (je zwei Bytes) Platz haben. Die naheliegendste Lösung wäre, auf jedem Gerät die gerätespezifische UUID ohne Major- und Minor-Zahl auszusenden. Diese Lösung hat jedoch zwei Nachteile:

1. Dem zum Empfangen verwendeten CoreLocation Framework müssen die UUIDs im Voraus bekannt sein, damit das Framework sie entdecken kann. Maximal können dem Framework im Voraus 20 UUIDs bekannt gegeben werden[2].

2. Wird die gerätespezifische UUID verwendet, ist der Benutzer an dieses Gerät gebunden.

Der erste Nachteil der Lösung kann behoben werden, indem immer die gleiche UUID ausgesendet wird. Somit bleiben noch vier Bytes Platz für die Benutzeridentifikation. Das reicht für unsere Applikation aus. Durch das Ablegen der Benutzeridentifikation im Backend, lässt sich auch der zweite Nachteil umgehen. Registriert sich ein neuer Benutzer in der Applikation, muss für ihn im Backend eine eindeutige Major- und Minor-Zahl generiert werden. Das Parse SDK ist jedoch stark darauf ausgelegt, dass alle Logik im Client ausgeführt wird und nur Daten im Backend abgelegt werden. Eine Auto-Increment/Sequence[3] ähnliche Funktion bietet Parse leider nicht an.

1 https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf [23]
2 https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CLLocationManager_Class/index.html#//apple_ref/occ/instm/CLLocationManager/startMonitoringForRegion: [24]
3 http://dev.mysql.com/doc/refman/5.0/en/example-auto-increment.html [25]

Es lässt sich jedoch Cloud Code im Parse Backend ausführen, welcher in Javascript geschrieben werden muss. Damit kann eine Auto-Increment/Sequence nachgebildet werden. Der Code dafür wurde grösstenteils aus einem Beitrag aus dem Parse Forum[4] übernommen.

```
function getSequence(callback) {
    var Test = Parse.Object.extend("Sequence");
    var query = new Parse.Query(Test);
    //console.log('Getting the Sequence object');
    query.get("AnrNDqv9bz", {
        success: function(object) {
            object.increment('sequence');
            object.save(null,{
                success:function(object) {
                    //console.log('In success from getSequence save');
                    callback(object.get('sequence'));
                },
                error:function(object,error) {
                    //console.log('In error from getSequence save');
                    //console.log(error);
                    callback(undefined);
                }
            });
        }, error: function (error) {
            //console.log('In error from getSeq get');
            console.log(error);
            callback(undefined);
        }
    });
}

Parse.Cloud.beforeSave(Parse.User, function (request, response) {
    if (request.object.isNew()) {
        getSequence(function(sequence) {
            if (sequence) {
                request.object.set("major", sequence);
                response.success();
            } else {
                response.error('Could not get a sequence.')
            }
        });
    } else {
        response.success();
    }
});
```

Einfachheitshalber wird im Backend nur eine Identitätszahl pro registriertem User generiert. Diese Identitätszahl wird auf der Client Seite noch in eine Major- und Minor-Zahl aufgeteilgt und danach mithilfe von CoreBluetooth ausgesendet.

4 https://www.parse.com/questions/can-i-have-something-similar-to-auto_increment-field-for-a-pfobject [26]

```
-(NSNumber*)major{
    if([[self objectForKey:@"major"] longValue] > (pow(2,16)-1) ) {
        return [NSNumber numberWithLong:(pow(2,16)-1)];
    } else {
        return [self objectForKey:@"major"];
    }
}

-(NSNumber*)minor{
    NSNumber* returnValue = [NSNumber numberWithInt:0];
    if([[self major] longValue] > (pow(2,16)-1) ){
        returnValue = [NSNumber numberWithLong:([[self major] longValue]
            % (long)(pow(2,16)-1))];
    }
    return returnValue;
}
```

9.2 GUI UPDATES DURCH SERVICE LAYER

Wie schon in Abschnitt 5.1.2 zum Service Layer beschrieben, wird der GUI Layer mithilfe eines Delegates über empfangene iBeacon Signale informiert. Jeder ViewController des GUI Layer, der Updates (wie zum Beispiel iBeacon Signale) vom LocationManagerService des darunterliegenden Service Layers erhalten will, muss das Protokoll LocationManagerServiceDelegate implementieren.

```
@protocol LocationManagerServiceDelegate <NSObject>

-(void)locationManagerServiceDelegate:(LocationManagerService*)
            service didUpdateUsersInRegion:(NSSet*)usersInRegion;

//...

@end
```

Das Protokoll enthält die Callback Methoden, welche der LocationManagerService bei Updates aufruft und so an den ViewController delegiert.

```
@interface LocationManagerService ()

//...
@property (nonatomic, strong) LocationManager* locationManager;

@end

@implementation LocationManagerService

-(void)notifyDelegate{
    [self.delegate locationManagerServiceDelegate:
        self didUpdateUsersInRegion:self.currentNeighbors];
}

//...

@end
```

Der ViewController registriert sich in der viewWillAppear Methode, die ausgeführt wird, kurz bevor die View als aktueller Delegate des LocationManagerService angezeigt wird.

```
@implementation RadarViewViewController

-(void) viewWillAppear:(BOOL)animated{
    self.locationManagerService.delegate = self;
    // ...
}

-(void)locationManagerServiceDelegate:(LocationManagerService*)
            service didUpdateUsersInRegion:(NSSet*)usersInRegion{
    self.currentNeighbors = [usersInRegion allObjects];
    [self.tableView reloadData];
    //...
}

//...

@end
```

Somit ist immer der ViewController, der aktuell angezeigten View, der Delegate des LocationManagerService. Diese Implementierung des Delegate Patterns wurde einem Observer vorgezogen, da wir es als einfacher und übersichtlicher betrachten. Des Weiteren ist beim Delegate Pattern die Kupplung klar ersichtlich, was beim Observer Pattern nicht der Fall ist. Weitere Informationen zum Delegate Pattern sind in den "Concepts in Objective-C Programming"[5] zu finden.

5 https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/DelegatesandDataSources/DelegatesandDataSources.html#//apple_ref/doc/uid/TP40010810-CH11-SW3 [27]

Teil V

APPENDIX

APPENDIX

A.1 USE CASE DIAGRAMM

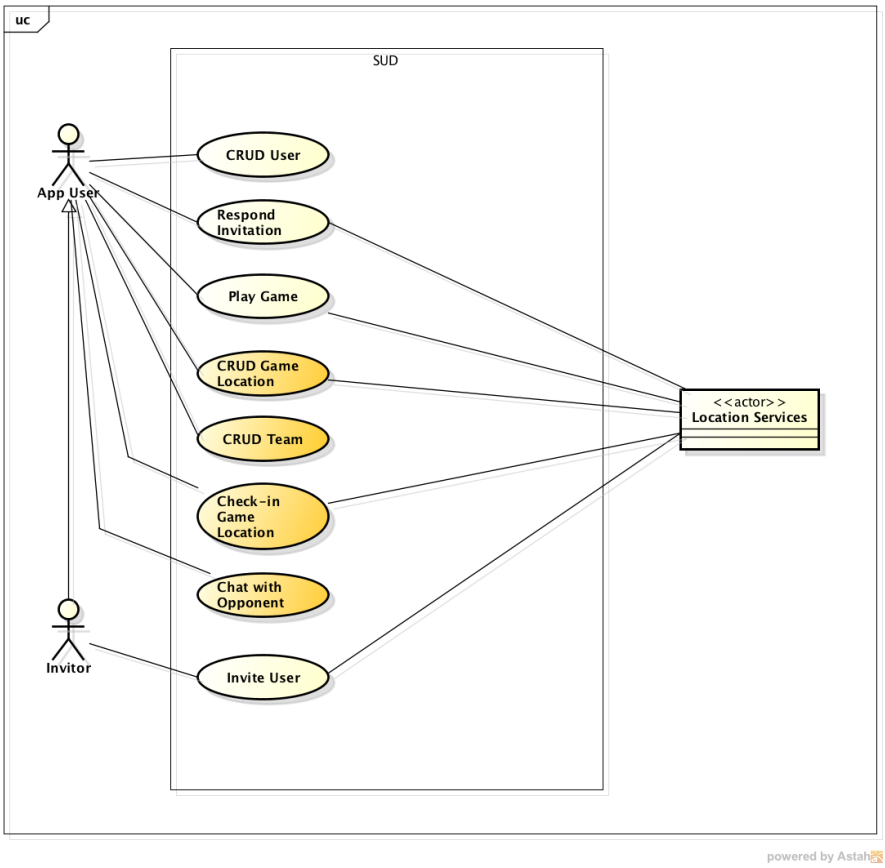


Abbildung 6: Use Case Diagramm

A.2 DOMAIN MODELL

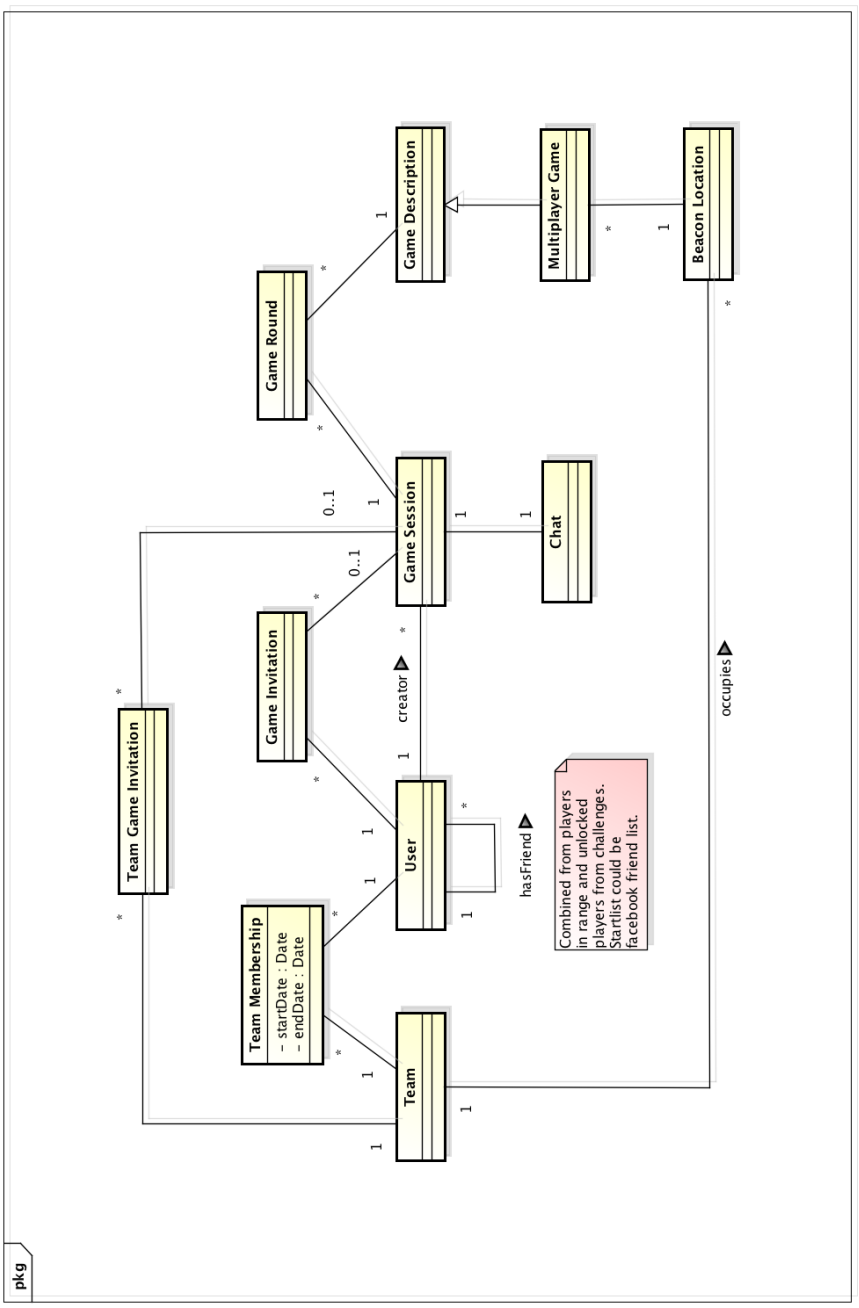


Abbildung 7: Domain Modell

[1] Tiobe. TIOBE Index for December 2014: Statistik zur Popularität von Programmiersprachen, 2014. URL http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Online, letzter Zugriff 17.12.2014].

[2] iOS Developer Library. What's New in iOS7: Auflistung der Neuerungen in iOS7, 2014. URL https://developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS7.html#//apple_ref/doc/uid/TP40013162-SW1.url. [Online, letzter Zugriff 17.12.2014].

[3] Wikipedia. Capture the Flag —Wikipedia, The Free Encyclopedia, 2014. URL http://de.wikipedia.org/wiki/Capture_the_Flag. [Online, letzter Zugriff 17.12.2014].

[4] Wikipedia. Risiko (Spiel) —Wikipedia, The Free Encyclopedia, 2014. URL http://de.wikipedia.org/wiki/Risiko_%28Spiel%29. [Online, letzter Zugriff 17.12.2014].

[5] Wikipedia. Point of Interest —Wikipedia, The Free Encyclopedia, 2014. URL https://de.wikipedia.org/wiki/Point_of_Interest. [Online, letzter Zugriff 17.12.2014].

[6] Quizduell. Quizduell: Spiel mit Quizfragen unter Freunden, 2013. URL http://www.quizduell-game.de. [Online, letzter Zugriff 17.12.2014].

[7] TestFlight. TestFlight: Deployment Plattform für Testing, 2014. URL https://www.testflightapp.com. [Online, letzter Zugriff 17.12.2014].

[8] Github. Github: Tool für Code Management, 2014. URL https://github.com. [Online, letzter Zugriff 17.12.2014].

[9] iOS Developer Library. Core Location Framework Reference: Referenzseite zum Core Location Framework, 2013. URL https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CoreLocation_Framework. [Online, letzter Zugriff 17.12.2014].

[10] iOS Developer Library. Core Bluetooth Framework Reference: Referenzseite zum Core Bluetooth Framework, 2014. URL https://developer.apple.com/library/ios/documentation/CoreBluetooth/Reference/CoreBluetooth_Framework. [Online, letzter Zugriff 17.12.2014].

[11] Parse. Parse: Homepage des Parse Backend Service, 2014. URL https://www.parse.com. [Online, letzter Zugriff 17.12.2014].

[12] iOS Developer Library. CloudKit Framework Reference: Referenzseite zum CloudKit Framework, 2014. URL https://developer.apple.com/library/ios/documentation/CloudKit/Reference/CloudKit_Framework_Reference. [Online, letzter Zugriff 17.12.2014].

[13] iOS Developer Library. MapKit Framework Reference: Referenzseite zum MapKit Framework, 2013. URL https://developer.apple.com/library/ios/documentation/MapKit/Reference/MapKit_Framework_Reference. [Online, letzter Zugriff 17.12.2014].

[14] Wikipedia. Repository (Entwurfsmuster) —Wikipedia, The Free Encyclopedia, 2014. URL http://de.wikipedia.org/wiki/Repository_%28Entwurfsmuster%29. [Online, letzter Zugriff 17.12.2014].

[15] Wikipedia. Fabrikmethode —Wikipedia, The Free Encyclopedia, 2014. URL http://de.wikipedia.org/wiki/Fabrikmethode. [Online, letzter Zugriff 17.12.2014].

[16] iOS Developer Library. UIStoryboard: Einfühtung in das Storyboard, 2014. URL https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIStoryboard_Class/index.html. [Online, letzter Zugriff 17.12.2014].

[17] iOS Developer Library. Working with Blocks: Einführung in Blocks, 2014. URL https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/WorkingwithBlocks/WorkingwithBlocks.html. [Online, letzter Zugriff 17.12.2014].

[18] iOS Developer Library. Grand Central Dispatch (GCD) Reference: Einführung in GCD, 2014. URL https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/index.html. [Online, letzter Zugriff 17.12.2014].

[19] iOS Developer Library. App Extensions Icrease Your Impact: Einführung in Etensions, 2014. URL https://developer.apple.com/library/ios/documentation/General/Conceptual/ExtensibilityPG/index.html. [Online, letzter Zugriff 17.12.2014].

[20] GCOV. 10 gcov - a Test Coverage Program: Erzeugt benötigte Files um Test Coverage zu berechnen. URL https://gcc.gnu.org/onlinedocs/gcc/Gcov.html. [Online, letzter Zugriff 17.12.2014].

[21] Coveralls. Code Coverage History and Stats: Test Coverage automatisch berechnen lassen mit Coveralls, 2014. URL https://coveralls.io/. [Online, letzter Zugriff 17.12.2014].

[22] Mattes Groeger. Travis CI for iOS: Deployment mit Travis Konfigurationsanleitung, 2013. URL http://www.objc.io/issue-6/travis-ci.html. [Online, letzter Zugriff 17.12.2014].

[23] iOS Developer Program. Getting Started with iBeacon: Einführung in die iBeacon Technologie, 2014. URL https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf. [Online, letzter Zugriff 17.12.2014].

[24] iOS Developer Library. startMonitoringForRegion: Methodenbeschreibung. URL https://developer.apple.com/library/ios/documentation/CoreLocation/Reference/CLLocationManager_Class/index.html#//apple_ref/occ/instm/CLLocationManager/startMonitoringForRegion:. [Online, letzter Zugriff 17.12.2014].

[25] MySQL. Using AUTO_INCREMENT: Wie ein Autoincrement funktioniert. URL http://dev.mysql.com/doc/refman/5.0/en/example-auto-increment.html. [Online, letzter Zugriff 17.12.2014].

[26] Fosco Marotto. Can I have something similar to auto_increment field for a PFObject: Beispielcode für Autoincrement, 2014. URL https://www.parse.com/questions/can-i-have-something-similar-to-auto_increment-field-for-a-pfobject. [Online, letzter Zugriff 17.12.2014].

[27] iOS Developer Library. Delegates and Data Sources: Erklärung der Funktionsweise von Delegates, 2014. URL https://developer.apple.com/library/ios/documentation/General/Conceptual/CocoaEncyclopedia/DelegatesandDataSources/DelegatesandDataSources.html#//apple_ref/doc/uid/TP40010810-CH11-SW3. [Online, letzter Zugriff 17.12.2014].

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, Herbstsemester 2014

Mario Meili,
18. Dezember 2014

Sebastian Bock,
18. Dezember 2014