Technology
Arts Sciences
TH Köln

Controlling Mobile Apps in Social Contexts

Praxisprojekt SoSe17 Dokumentation

ausgearbeitet von

Duc Giang Le - Matrikelnr.: 11089582
und
Vu Phi Hai Dinh - Matrikelnr.: 11090042

Prüfer und Betreuer: Prof. Dr. Matthias Böhmer
Technische Hochschule Köln

Gummersbach, der 6. März 2018

Inhaltsverzeichnis

1 Einleitung

Die folgende Ausarbeitung ist eine Projektdokumentation, die im Rahmen des Moduls Praxisprojekt in Kooperation mit Prof. Dr. Matthias Böhmer entstanden ist. Sie beschreibt Problematiken und Lösungsansätze, die im Projektverlauf aufgetreten sind. Außerdem werden in der Dokumentation wichtige Abwägungen und Entscheidungen, die im Projekt getroffen wurden, festgehalten, damit ein besserer Überblick über den Prozess in der Projektarbeit gegeben wird.

1.1 Nutzungsproblem

Das Smartphone ist in der heutigen Gesellschaft unentbehrlich. Er ist ein ständiger Begleiter, sei es zu Hause, in der Fachhochschule oder auch unter Freunden. Auch wenn das Smartphone durch seine vielfältigen Funktionen viele Vorteile bietet und wie ein mobiler Computer genutzt werden kann, birgt es unter anderem auch Nachteile und negative Auswirkungen auf die Menschen. Oft lassen sich Menschen zu häufig von ihrem Smartphone ablenken. Sie neigen dazu, bei jeder neuen Benachrichtigung ständig auf das Smartphone zu schauen und dies führt zur Behinderung der Kommunikation. Das fördert bspw. Prokrastination und Phubbing. Letzteres soll genauer erklärt werden. Der Begriff Phubbing bezeichnet die Angewohnheit, sich mit dem Smartphone zu beschäftigen, während man die Menschen, mit denen man gerade gesellschaftlich verkehrt, vernachlässigt. Das kann z.B. psychologische Auswirkungen auf seine Mitmenschen haben, weil die Person ihre Aufmerksamkeit mehr auf das Smartphone richtet statt auf seine Mitmenschen. Dadurch kann man sich unwichtig fühlen. Die Problemstellung, die im Praxisprojekt bearbeitet wird, bezieht sich auf den Aspekt des Phubbings.

1.2 Motivation

Die Projektidee entstand von seitens des Betreuers Prof. Dr. Matthias Böhmer. Für das Team war diese Idee interessant, weshalb diese Projektidee im Praxisprojekt ausgearbeitet werden soll. Die persönliche Motivation dieses Projektes durchzuführen ist zum einen, dass das Team neue Erfahrungen mit mobilen Applikationen und Smartphones sammeln will. Hierbei speziell Android und Java und zum anderen besteht das eigene Interesse, dass oben geschilderte Nutzungsproblem zu lösen und dem Phubbing entgegenzuwirken, denn die eigene Erfahrungen in Bezug auf Phubbing ist dem Team schon bekannt. Außerdem will das Team weiterhin weiteres Wissen über neue Technologien sammeln, auf die sie im Laufe des Studiums noch nicht gestoßen sind.

1.3 Zielsetzung

Für das Lösen des geschilderten Nutzungsproblem hat das Team folgende Zielsetzungen definiert, die mithilfe der im Projekt angewandten Methoden erreicht werden können. Damit die Ziele deutlicher werden, hat das Team eine Zielhierarchie aufgestellt und die einzelnen Ziele mithilfe der Verben "muss, soll oder kann"priorisiert.

strategische Ziele:

- Das Phubbing muss in gesellschaftlichen Treffen entgegenwirkt werden.

taktische Ziele:

- Es soll eine mobile Anwendung entwickelt werden, die die Funktionalität des Smartphones vom Gegenüber für die Dauer der gesellschaftlichen Treffen einschränkt.
- Es muss von der Anwendung erkannt werden können, wann sich Nutzer in einer Konversation befinden bzw. wann sich Nutzer miteinander treffen.
- Die Aufmerksamkeit, die der Nutzer dem Smartphone zuwendet, muss bei einem Treffen reduziert werden.

operative Ziele:

- Die Domäne des Nutzungsproblems muss detaillierter analysiert werden.
- Es muss ein Prototyp entwickelt werden, der die zu entwickelnde Anwendung repräsentiert.
- Es muss für das Projekt ein Vorgehen abgewägt und definiert werden.
- Es müssen vorhandene Technologien untersucht und getesteten werden, um die Realisierbarkeit einer solchen Applikation zu prüfen.
- Es sollen erste UI-Lösungen entwickelt werden, die in Form von Wireframes getestet werden können.
- Es sollen ähnliche vorhandene Produkte untersucht werden, die das Nutzungsproblem ganz oder teils versuchen zu lösen.

2 inkrementell-iteratives Vorgehen

Vor dem Beginn der Entwicklung der mobilen Anwendung hat das Team sich tiefer in das Thema eingearbeitet und die Domäne genauer erforscht, damit fundiertes Wissen über das Nutzungsproblem vorhanden ist. Dabei wurden von Prof. Dr. Matthias Böhmer informationsreiche Quellen zur Verfügung gestellt, die vom Team bearbeitet und ausgewertet wurden. Außerdem hat Prof. Dr. Matthias Böhmer schon einige Anforderungen an die Anwendung aufgestellt, da es wie bereits beschrieben seine Projektidee ist.
Für den weiteren Verlauf des Projekts hat das Team in einem Exposé eine erste Version eines möglichen Vorgehens definiert, das Ähnlichkeiten mit dem Wasserfall-Modell hat. Nach Diskussion mit dem Betreuer hat sich herausgestellt, dass dieses Vorgehen in diesem Projekt sehr ungeeignet war, da die Implementierung erst sehr spät erfolgen würde. Da das Projekt am Ende die Realisierbarkeit der Anwendung prüfen sollte, wäre es sinnvoll, so früh wie möglich zu implementieren und frühzeitig die Machbarkeit von einzelnen Funktionen und Technologien zu testen. Deshalb hat der Betreuer dem Team das inkrementell-iterative Vorgehen vorgeschlagen, in dem es darum geht, mehrere kleine Iterationen durchzuführen, in der Methoden wie Szenarien, Anforderungsermittlung, Wireframes, Evaluation usw. dadurch im gesamten Projekt öfters angewandt werden. Außerdem soll in den Iterationen bereits mit der Implementierung und mit dem Testen gestartet werden, so dass nach jeder Iteration die Anwendung weiterentwickelt wird und weiterwächst.
Dieses Vorgehen wurde vom Team übernommen und als Projektvorgehen definiert. Basierend auf diesem Vorgehen wurde anschließend noch der alte Projektplan aktualisiert und es entstanden bis zur Abschlusspräsentation vier Iterationen, die im Folgenden genauer beschrieben werden. Eine weitere Iteration wurde durch die Verschiebung des Termins der Abschlusspräsentation hinzugefügt.
Des Weiteren wird in den Iterationen zwischen dem Team, Phi Hai (Vu Phi Hai Dinh) und Giang (Duc Giang Le) unterschieden, damit deutlich wird, welche Aufgaben das Team gemeinsam erledigt hat und welche Aufgaben die einzelnen Teammitglieder erledigt haben. Außerdem wird im weiteren Verlauf der Name „ShutApps“ erwähnt, der unsere zu entwickelnde Anwendung repräsentiert.

3 Erste Iteration

3.1 Marktrecherche

In der Marktrecherche wurden eine Vielzahl von Anwendungen in den Stores der Plattformen Android und iOS unter die Lupe genommen, um in Erfahrung zu bringen, ob Anwendungen existieren, die dieselben oder ähnliche Funktionen aufweisen. Darüber hinaus werden aus den Erkenntnissen der Marktrecherche, neue Anforderungen für die zu entwickelnde Anwendung ermittelt. Außerdem soll durch die Untersuchung der vorhandenen Produkte gezeigt werden, was zum jetzigen Zeitpunkt realisierbar wäre und welche Plattformen bzgl. der Realisierbarkeit geeignet wären. In den folgenden Tabellen werden die untersuchten Anwendungen aufgelistet.

Tabelle 3.1: Marktrecherche Teil 1

	Freedom - Reduce Distractions	**Unicef Tap Project**
Plattform	iOS	Web
Beschreibung	Die App ermöglicht dem Nutzer, Webseiten zu blockieren, um nicht abgelenkt zu werden.	Mithilfe der App kann man für einen Menschen Trinkwasser spenden, indem man sein Smartphone mit dem Bildschirm nach unten liegen lässt bzw. nicht anfasst.
Eigenschaften	- Die App sorgt dafür, dass Zugang ins Internet für eine vorher festgelegte Zeit blockiert ist. - Es kann während dieser "Freizeit"kein Content von Webseiten geöffnet werden.	- Alle fünf Minuten kriegt man für einen Menschen, einen Tag Trinkwasser - Man kann auch separat Geld spenden - Freunde herausfordern, um deren Zeit zu überbieten (Gamification) - Verzicht der Handynutzung für einen guten Zweck

Tabelle 3.2: Marktrecherche Teil 2

	AppDetox	**Digital Detox Challenge**	**Forest**
Plattform	**Android**	**Android**	**Android/iOS**
Beschreibung	Die App bietet dem Nutzer die Möglichkeit, Regeln für das Blockieren seiner Apps zu definieren. Zum Beispiel will er während seiner Arbeit, Facebook oder Instagram blockieren.	Die App soll dem Nutzer dabei unterstützen, die Handynutzung per Challenges unter Freunden einzuschränken.	Die App ermöglicht dem Nutzer, seine Konzentration aufrechtzuerhalten, indem er virtuelle Bäume pflanzen kann und so gezwungen wird, das Smartphone für eine vorher von ihm festgelegte Zeit ruhen zu lassen. Wenn das Smartphone für diese Zeit nicht genutzt wird, kann der Baum auswachsen. Anderfalls stirbt er ab.
Eigenschaften	1) "Die App nutzt einen Mechanismus des Android-Betriebssystem. Sie schaut, welche anderen Apps gerade benutzt werden." 2) Man kann Regeln definieren. Bsp: Ein Zeitfenster setzen, in dem man bestimmte Apps nicht nutzen will. 3) Man kann eine App auch für immer sperren.	1) Begrenzter Zugriff auf dem Smartphone während der Herausforderung 2)Mehrfache Schwierigkeitsstufen mit built-in accountability 3) Scheduling und Whitelisting-Funktion 4) Errungenschaften und Leaderboard bei Play Games	1) Durch Gamification versucht die App, die Nutzung des Smartphones zu reduzieren. 2) Die anderen Anwendungen können durch die App nicht blockiert werden. Es wird nur erkannt, ob andere Apps geöffnet werden.

Tabelle 3.3: Marktrecherche Teil 3

	Menthal	**Offtime**	**Offtime light**	**Procrastination Punisher**
Plattform	Android	Android	iOS	Android
Beschreibung	Die App misst und überwacht die Nutzung des Smartphones.	Die App soll dem Nutzer dabei unterstützen, die Handynutzung besser zu kontrollieren und einzelne Anwendungen gezielt für einen selbst definierten Zeitraum zu blockieren.	Die App liegt den Fokus auf das Erstellen einer Übersicht der Nutzung des Smartphones.	Die App bietet die Möglichkeit, zu definierten Arbeitszeiten bestimmte Anwendung zu blockieren. Sollte der Nutzer dennoch während dieser Arbeitszeit eine blockierte App öffnen wollen, muss er einen kleinen Geldbetrag an eine Organisation spenden.
Eigenschaften	- Mit den gemessten Daten berechnet die App einen "M Score", der zwischen 0 und 100 liegt. - Je höher die Zahl ist, umso bedenklicher ist der Handygebrauch. - Diese App überwacht die Handynutzung, aber blockierte keine anderen Anwendungen	- automatisches Blockieren und Filtern von Anrufen und Apps - Kalendersynchronisation - persönliche Nutzungsübersicht - Profile für verschiedene Situationen und Umgebung - es können Ausnahmen definiert werden - App benötigt viele Rechte und Zugriffe auf persönliche Daten	- Beobachtet das Nutzungsverhaltens und dich mit anderen - digitale Auszeit nehmen (im Moment nur über Flugmodus/ Stumm)	- Sollte selbstdefinierte Apps blockieren können, jedoch funktioniert diese Funktion laut dem Userfeedback im Play Store nicht

Als Ergebnis stellt sich heraus, dass in Android die Möglichkeit besteht, andere installierte Anwendungen auf einem Smartphone zu blockieren, während dies in iOS nicht zum jetzigen Zeitpunkt nicht funktioniert. Die iOS-Plattform ist zu einschränkend, damit eine Anwendung genug Rechte bekommt, um andere installierte Anwendungen zu blockieren. Viele im Markt vorhandene Anwendungen können lediglich den Zugang zum Internet blockieren beziehungsweise dafür sorgen, dass der Inhalt von Webseiten nicht geladen werden können. Die einzige Möglichkeit bisher, um in iOS den Zugang zu anderen Anwendungen zu blockieren ist es, das Smartphone zu "jailbreaken", um genügend Rechte freizuschalten. Jedoch ist dieser Weg nicht optimal für den Nutzer, weswegen diese Möglichkeit in iOS momentan verfällt.

3.2 Location-Technologien

In diesem Abschnitt geht es um die verfügbaren Technologien am Smartphone, die es ermöglichen, Standorte zu bestimmen oder andere Smartphones in der Umgebung zu erkennen und mit ihnen zu kommunizieren. Hierbei wurden fünf verschiedene Technologien analysiert, um abschließend abzuwägen, welche der Technologien für die Entwicklung der Anwendung am geeignetsten ist. Für die Anwendung spielt die Location eine wichtige Rolle, damit Smartphones, die sich in einer bestimmten Reichweite befinden, eine Verbindung miteinander aufbauen und miteinander kommunizieren können, indem sie z.B. Nachrichten miteinander austauschen. Im Folgenden sollen die analysierten Technologien erläutert werden.

3.2.1 Beacons

Beacon ist ein Sender oder Empfänger, der auf der Bluetooth Low Energy (BLE) oder auch Bluetooth Smart Technologie basiert. Meistens besteht die Verbindung zwischen Smartphone/App und dem Beacon. Zudem kann der Beacon laut Böhme (2015) den Standort eines Smartphones über (UUID) also die ID des Gerätes durch mehrere Sender ermittelt werden. Smartphones können über dem Beacon Daten austauschen bzw. kommunizieren. In der Abbildung 3.1 ist ein Beacon zu sehen, der per Bluetooth Nachrichten über eine App an die jeweiligen Smartphones sendet, wenn die Person sich in der Nähe befindet. Meistens wird der Beacon in Einkaufsläden benutzt, die die dazugehörige App nutzt, um darüber Informationen über Angebote oder ähnliches zu bekommen. Der Nachteil ist, dass sie keine großen Datenmenge transferieren können, weil es zu lange dauern würde.

Abbildung 3.1: Beacon
Quelle: http://www.jwtinside.com/wpcontent/uploads/2016/10/beacondiagram.jpg

3.2.2 GPS (Global Positioning System)

GPS bezeichnet ein System zur Positionsbestimmung und Navigation mit Hilfe von Satelliten. Laut der Seite von MagicMaps sendet jeder Satellit in bestimmten Zeitintervallen Signale mit seiner Position und der exakten Uhrzeit aus. Aus den Signalen von mindestens vier Satelliten kann ein GPS-Empfänger seinen Standort berechnen. Je mehr Satellitensignale empfangen werden, desto genauer wird die Position des Benutzers bestimmt. Das GPS Gerät des Benutzers wertet die Signale der Satelliten aus und berücksichtigt die Korrektursignale von z.B. EGNOS. Dazu benötigt ein Gerät eine Antenne, eine Quarzuhr, etwas Speicher und einen Prozessor zum Rechnen.
In der Abbildung 3.2 wird gezeigt, wie das Signal zustande kommt.

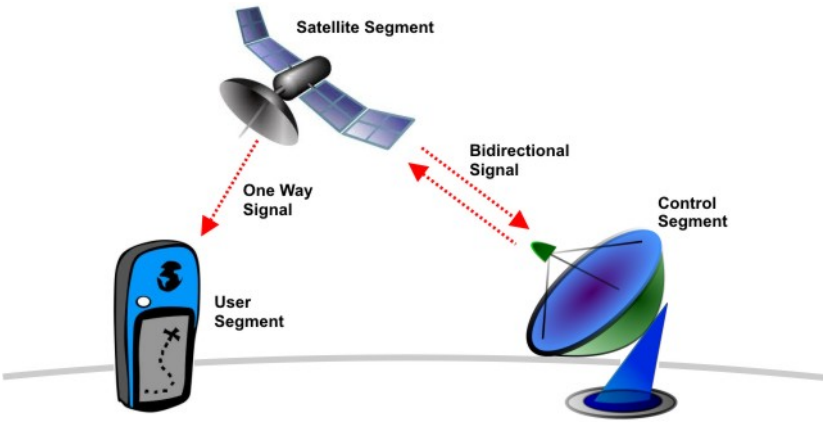


Abbildung 3.2: Global Positioning System
Quelle: http://www.azosensors.com/article.aspx?ArticleID=29

3.2.3 Google Nearby API

Die Google Nearby API ermöglicht es einem Smartphone andere Smartphones in der Nähe zu erkennen und Daten mit ihnen auszutauschen. Durch drei verschiedene Signale (WiFi, Bluetooth und Audio) erkennt Nearby, ob zwei Smartphone nah genug beieinander sind, so dass eine Verbindung zwischen ihnen aufgebaut werden kann. Beide Geräte werden nicht direkt über WiFi verbunden, sondern es wird lediglich die verfügbaren WLAN-Access Points beider Geräte miteinander verglichen. Das Bluetooth sendet einen speziellen Token, die von Geräten mit Nearby gesehen werden können. Außerdem senden die Smartphones Ultraschall, die von den Mikrofonen der Geräte erkannt werden und dadurch können sich beide Geräte gegenseitig "hören". Mittels des Publish-Subscribe Prinzip ist den Smartphones nun möglich, Nachrichten zu publishen und auch veröffentlichte Nachrichten zu empfangen. In der Abbildung 3.3 wird der Aufbau der Kommunikation zwischen zwei naheliegenden Geräten beschrieben.

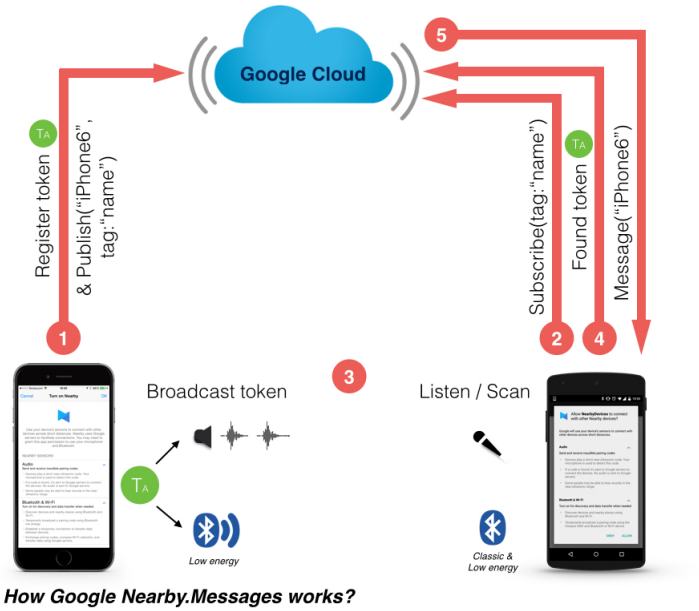


Abbildung 3.3: Google Nearby API
Quelle: http://blog.p2pkit.io/how-google-nearby-really-works-and-what-else-it-does/

3.2.4 WiFi-Basiert mittels der FIND Framework

Das Framework FIND (The Framework for Internal Navigation and Discovery) ermöglicht es, durch das Android Smartphone oder einem WiFi-fähigem Gerät die Position innerhalb eines kleinen Gebäude (Wohnung oder Büro) zu bestimmen. Dabei ist zu bedenken, dass ein Router benötigt wird, um die Verbindung zwischen Geräten aufzubauen. Dadurch ist die Benutzung meistens nur im Indoor möglich. Die Abbildung 3.4 soll die Standortbestimmung durch das Framework verdeutlichen.

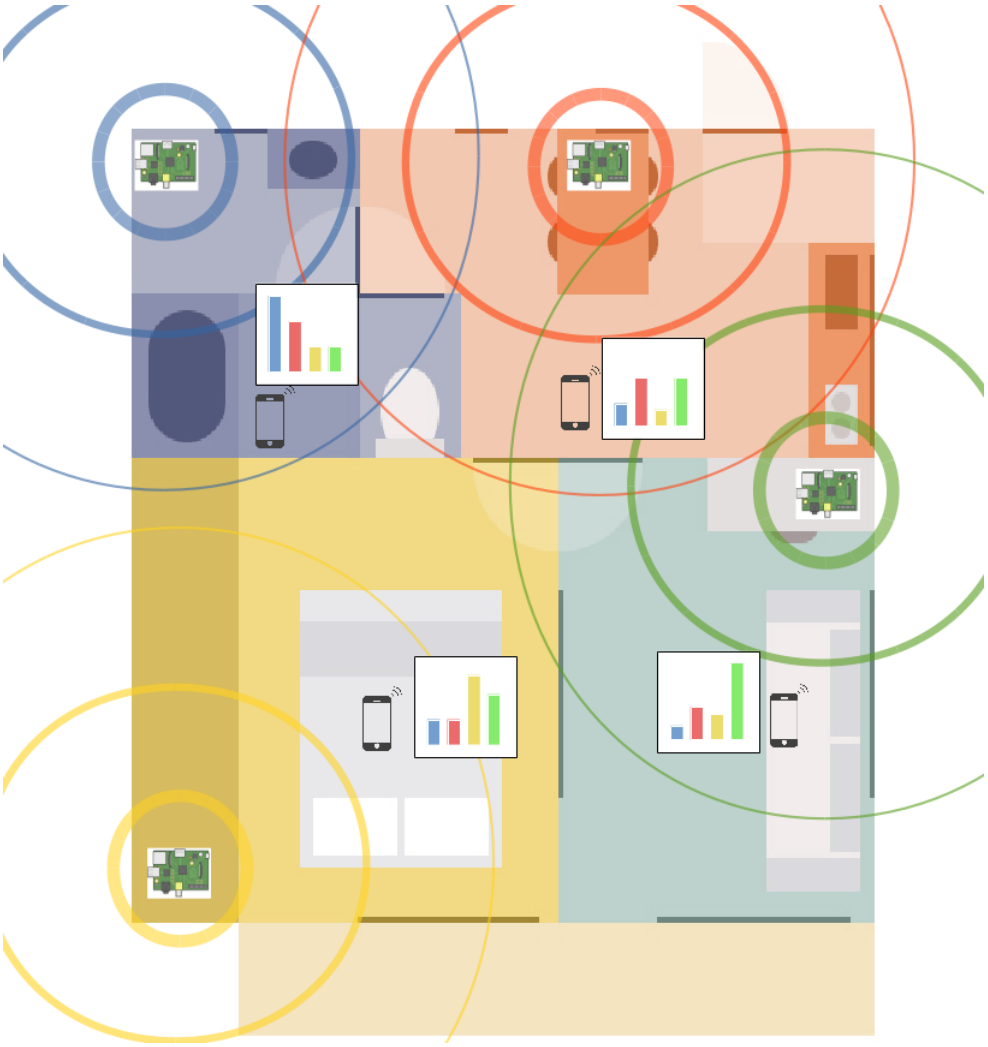


Abbildung 3.4: WiFi-basierte Standortbestimmung
Quelle: https://www.internalpositioning.com/faq/

Durch die Signalstärke zu den einzelnen WLAN-fähigen Geräten ist es für das Framework möglich, den Standort des Smartphones in einer Wohnung zu bestimmen. Je stärker die Signalstärke ist, desto näher befindet sich das Smartphone an dem jeweiligen Gerät.

3.2.5 WiFi-Direct

Wi-Fi Direct ist eine Methode zur Datenübermittlung zwischen zwei oder mehreren Geräten, die auch als WLAN-Endgeräten anzusehen sind. Hierbei benötigt man keinen Router und keine Internetverbindung, sondern es hat die Eigenschaft eines drahtlosen Netzwerkes. Die Geräte werden als Gruppe dargestellt und man kann Bilder, Videos usw. von einem Gerät zum anderen senden, egal wie groß die Datei ist, weil die Bandbreite schneller ist, als beim Bluetooth. Es ist so gesehen laut dem Autor Christian (2011) die schnellere Variante von Bluetooth, aber zugleich auch ein Akkufresser.

Abbildung 3.5: WiFi-Direct
Quelle: http://static.giga.de/wp-content/uploads/2009/10/wifi_direct.jpg

In den folgenden Tabellen wurden vier Kriterien aufgestellt, die für die Anwendung von Bedeutung sind. Hierbei hat das Team sich auf die Genauigkeit, Akkulaufzeit, Reichweite und Verfügbarkeit fokussiert. Bei der Genauigkeit wurde auf die Positionsbestimmung geachtet, um so exakt wie möglich ein Gerät zu orten. Die Akkulaufzeit besagt, wie viel Akku das Smartphone mit der Technologie frisst, denn im Normalfall müsste die Technologie ständig laufen, sobald die Anwendung geöffnet wurde. Je weniger Akku verbraucht wird, umso besser. Mit der Reichweite wird bestimmt, wie weit der Empfang beziehungsweise der Radius zwischen zwei Smartphones sein kann. Unter Verfügbarkeit ist zu verstehen, an welchen Orten beziehungsweise Bereichen die Technologie funktioniert und ob es auf beiden Plattformen verfügbar ist. Obwohl iOS in der Marktrecherche schon ausgeschlossen wurde, wurde dies trotzdem in Betracht gezogen, falls in Zukunft doch die Möglichkeit besteht in iOS andere installierte Anwendungen zu blockieren. Anhand dieser Kriterien wurde im Team abgewägt, welche Technologie für die Entwicklung der Anwendung am geeignetsten wäre.

Tabelle 3.4: Technologien Teil 1

	Beacon	**GPS**	**Google Nearby API**
Genauigkeit	abhängig Reichweite	5-15 Meter	Hoch
Akkulaufzeit	Hoch	je nach Nutzung des Netzwerkstyps	Mittel
Reichweite	10-50 Meter je nach Hersteller	Hoch	ca. 30 Meter
Verfügbarkeit	Indoor/Outdoor Android/iOS	Outdoor Indoor je nach Gebäude Android/iOS	Indoor/Outdoor Android/iOS

Tabelle 3.5: Technologien Teil 2

	WiFi	**Wifi-Direct**
Genauigkeit	abhängig von Postition des WLAN Geräts	Hoch
Akkulaufzeit	Geringer als GPS	Gering
Reichweite	abhängig Anzahl der WLAN Geräten	ca. 200 Meter
Verfügbarkeit	Outdoor Android	Nur selben Hersteller

Nach genauerer Betrachtung aller Technologien sind wir zum Entschluss gekommen, die Google Nearby API für die Umsetzung unserer Anwendung zu nutzen. Die WiFi-basierte Variante ist zu sehr abhängig von WLAN-fähigen Geräten, die sich meistens nur in Gebäuden befinden. Dadurch würde bspw. eine Positionsbestimmung im Freien durch das FIND-Framework nicht funktionieren. Für unsere Anwendung ist die Nutzung jedoch für Indoor und Outdoor wichtig. Das ist für das FIND-Framework ein Ausschlusskriterium.
Die Beacons können genau bestimmen, ob sich ein Gerät innerhalb der Beacon-Reichweite befindet. Dadurch ist es einfach zu bestimmen, ob sich zwei Geräte in der Nähe befinden. Jedoch müsste der Nutzer für die Nutzung unserer App ständig einen Beacon bei sich tragen. Deshalb kommt die Nutzung von Beacons für die App nicht in Frage.
Beim GPS ist die Genauigkeit der Positionsbestimmung nicht immer exakt und abhängig von äußeren Faktoren. Dadurch könnte die Erkennung, ob sich zwei Geräte in der Nähe befinden, problematisch werden. Für unsere App ist es wichtig, dass diese Erkennung einwandfrei funktioniert. Außerdem verbraucht GPS durch das ständige Scannen viel Akku, so dass GPS ebenfalls nicht in Frage kommen wird.
WiFi-Direct wurde ebenfalls ausgeschlossen, weil es momentan nur möglich ist eine Verbindung zwischen den selben Hersteller zu erstellen. Da es viele Herstellern gibt und jeder ein anderes Gerät besitzt, wäre dies nicht geeignet für die Anwendung.

Wie oben bereits erwähnt wird die Google Nearby API für die Umsetzung unserer Anwendung zum Einsatz kommen. Die Nutzung von mehreren Smartphone-Sensoren erlaubt es dem Smartphone genau zu bestimmen, ob sich ein anderes Smartphone in der Nähe befindet. Außerdem ist Nearby durch das exakte Scannen indoor sowie outdoor verfügbar. Des Weiteren wurde Google Nearby speziell dafür entwickelt, dass ein Smartphone andere Smartphone in der Nähe erkennen und mit ihnen kommunizieren kann. Ein Nachteil wäre jedoch, dass die Nutzung der verschiedenen Smartphone-Sensoren mehr Akku im Vergleich zu den anderen Technologien verbrauchen könnte, was wir für den Einsatz der Nearby API in Kauf nehmen müssen. Im weiteren Projektverlauf wollen wir uns erkundigen, ob es möglich ist, den Akkuverbrauch von Nearby so gering wie möglich zu halten oder ob es vielleicht weitere Möglichkeiten gibt, die uns dazu bewegt, Google Nearby auszuschließen.

3.3 Anforderungsermittlung

Auf Grundlage der Marktrecherche und den von Prof. Dr. Matthias Böhmer aufgestellten Anforderungen hat das Team Anforderungen abgeleitet und ermittelt. Diese folgenden Anforderungen dienen zum einen für die Erstellung des Paper Prototyp und zum anderen auch für die Implementation der einzelnen Funktionen der Applikation. Die Formulierung der Anforderungen orientiert sich an der Anforderungsschablone aus der Literatur „Rupp (2014)“.

- F10: Die Anwendung muss in der Lage sein, andere installierte Applikationen zu blockieren.
  - F11: Die Anwendung muss fähig sein, den Zugang zu anderen Anwendungen zu blockieren.
  - F12: Die Anwendung muss fähig sein, bestimmte Anwendungen nur zu blockieren, wenn mindestens ein Freund in der Nähe ist.
- F20: Die Anwendung muss Freunde in der Nähe identifizieren können.
  - F21: Die Anwendung muss fähig sein, andere Smartphones in der Umgebung zu erkennen.
  - F22: Die Smartphones, die diese Anwendung nutzen, müssen untereinander kommunizieren können.
  - F23: Die Anwendung muss fähig sein zu erkennen, welche Personen in der Umgebung Freunde sind.
  - F24: Die Anwendung muss fähig sein, Freunde darzustellen.
- F30: Die Anwendung muss eine Blacklist erstellen können, die besagt welche Anwendungen blockiert werden sollen.
  - F31: Die Anwendung muss dem Nutzer die Möglichkeit bieten, weitere zu blockierende Anwendungen in die Blacklist hinzuzufügen.
  - F32: Die Anwendung muss dem Nutzer die Möglichkeit bieten, sich in der Blacklist befindende Anwendungen zu entfernen.
- F40: Die Anwendung muss die installierten Anwendungen eines Smartphones auslesen können.
  - F41: Die Anwendung soll dem Nutzer darstellen, welche Anwendungen er installiert hat.
  - F42: Die Anwendung muss dem Nutzer ermöglichen, installierte Anwendungen in die Blacklist hinzuzufügen.
- F50: Die Anwendung muss die App-Liste aller beteiligten Nutzer entnehmen und zu einer neuen gemeinsamen Liste vereinigen können.
- F60: Die Anwendung sollte dem Nutzer die Möglichkeit bieten zu sehen, wer seine Anwendungen im Moment blockiert.

Die hier ermittelten Anforderungen sind erste Anforderungen, die im Projektverlauf iterativ weiterentwickelt und aktualisiert werden. Für das Team sind F10 und F20 die beiden wichtigsten Anforderungen für das Projekt, da sie die Grundlage für die Realisierbarkeit der Anwendung darstellen.

3.4 Architekturentwurf

Für die Implementierung der Anwendung ist es wichtig, vorher einen Architekturentwurf zu entwickeln, der zeigt, aus welchen Komponenten das System am Ende besteht. Dabei hat das Team auf Grundlage von Vorwissen aus anderen Modulen des Studiums eine Client-Server Architektur entwickelt, die in Abbildung 3.6 zu sehen ist. Dieser Entwurf ist nach Feedback mit dem Betreuer Prof. Dr. Matthias Böhmer entstanden. Dabei ist es wichtig, dass die Clients miteinander kommunizieren können, was durch die Nutzung der Google Nearby API ermöglicht wird. Mit der Facebook API sollen außerdem die einzelnen Clients personalisiert werden. Diese Clients kommunizieren mit dem Firebase-Service, der folgende Komponenten zur Verfügung stellt: Datenbank, Storage und Authentication. Des Weiteren wurde geplant, den Firebase-Service als Brücke zwischen Client und Server einzusetzen, so dass der Client indirekt immer noch mit dem Server kommunizieren kann. Für den Server ist die Realisierung in node.js vorgesehen, auf dem eine bestimmte Anwendungslogik laufen soll.

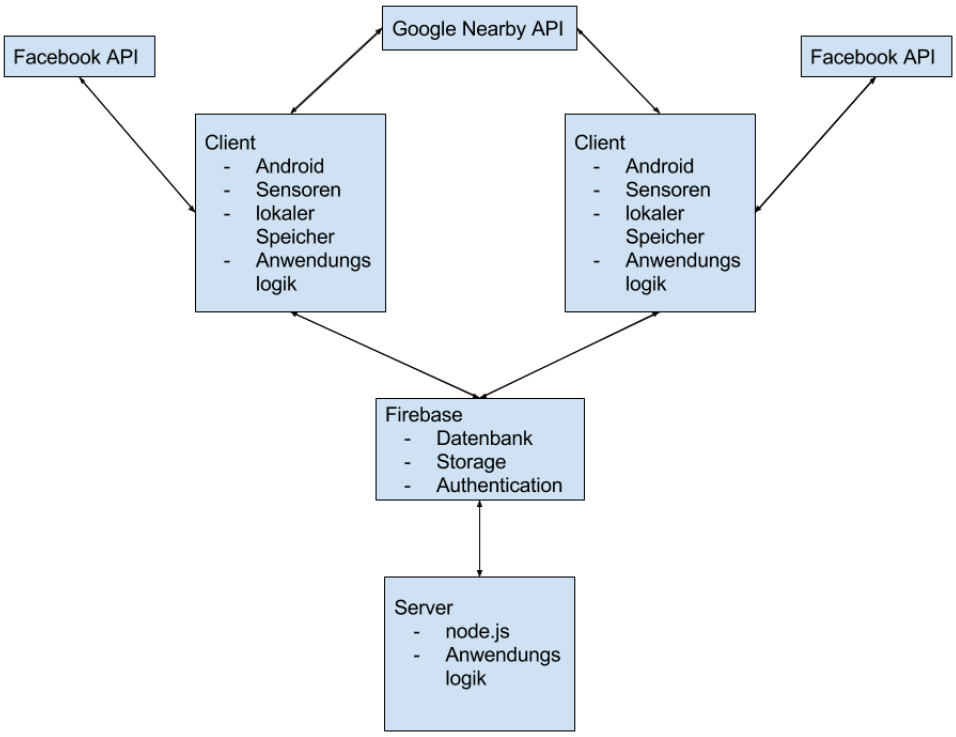


Abbildung 3.6: erste Version des Architekturentwurfs (29.03.2017)

3.5 Paper Prototyp

In dieser Sektion beschäftigen wir uns zunächst mit dem User Interface. Die Anwendung soll schlicht und einfach aussehen und dabei leicht bedienbar sein. Dazu wurde ein Paper Prototyp erstellt, um die erste Vorstellung auf Papier zu bringen. Als Anregung von Prof. Dr. Matthias Böhmer wurden seine und unsere Anforderungen berücksichtigt, worauf die User Interface gestützt ist. Mithilfe der Anforderungen erstellte das Teammitglied Phi Hai den Paper Prototyp, was in den folgenden Abbildungen kurz erläutert wird.

In Abbildung 3.7 ist der Screen abgebildet, wenn die Anwendung gestartet wird. In der Menüleiste sind zwei Kategorien zu sehen. Einmal die App-Liste und die Blacklist. Momentan befindet sich die App in der App-Liste, die darunter angezeigt wird. Die App-Liste ist vordefiniert und mit dem Plus-Button anwählbar. Zwischen der Menüleiste und die App-Liste befindet sich das Profilbild des jeweiligen Nutzers.

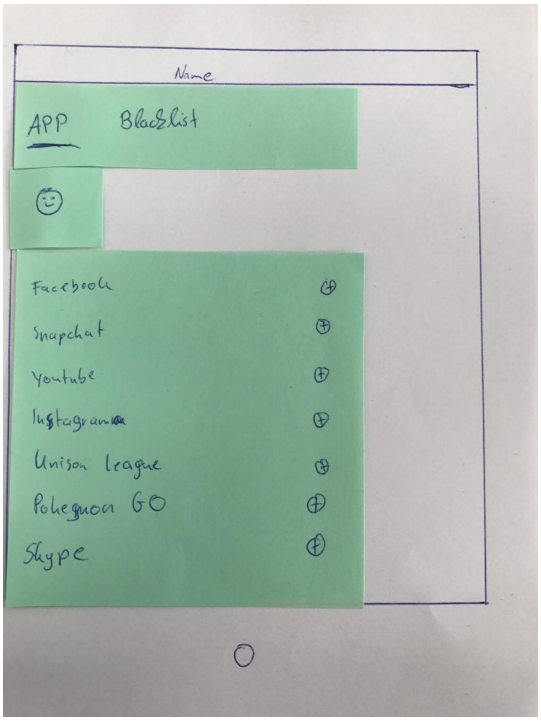


Abbildung 3.7: Paper Prototyping - Alleine

In der darauffolgenden Abbildung 3.8 wird die Blacklist angezeigt mit den geblockten Anwendungen. Die Liste kann sich durch das Auswählen der Anwendungen erweitern. In diesem Fall wird momentan Facebook geblockt, wenn ein Freund sich in Reichweite befindet. Das eigene Gerät wird nicht blockiert.

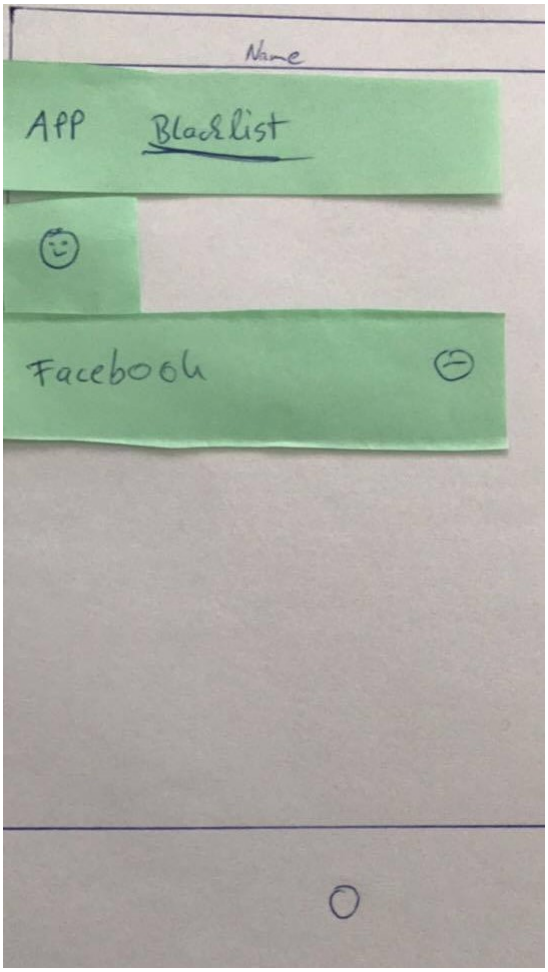


Abbildung 3.8: Paper Prototyping - Alleine mit BL

In diesem Beispiel 3.9 befindet sich die Anwendung in der App-Liste. Diesmal hat die Anwendung, Freunde in der Nähe erkannt und listet alle mit ihrem Profilbilder auf. Jetzt wurde die Voraussetzung geschaffen, dass mindestens zwei Geräten miteinander kommunizieren und die Anwendung dadurch ihre eigentlichen Aufgaben erfüllen kann.

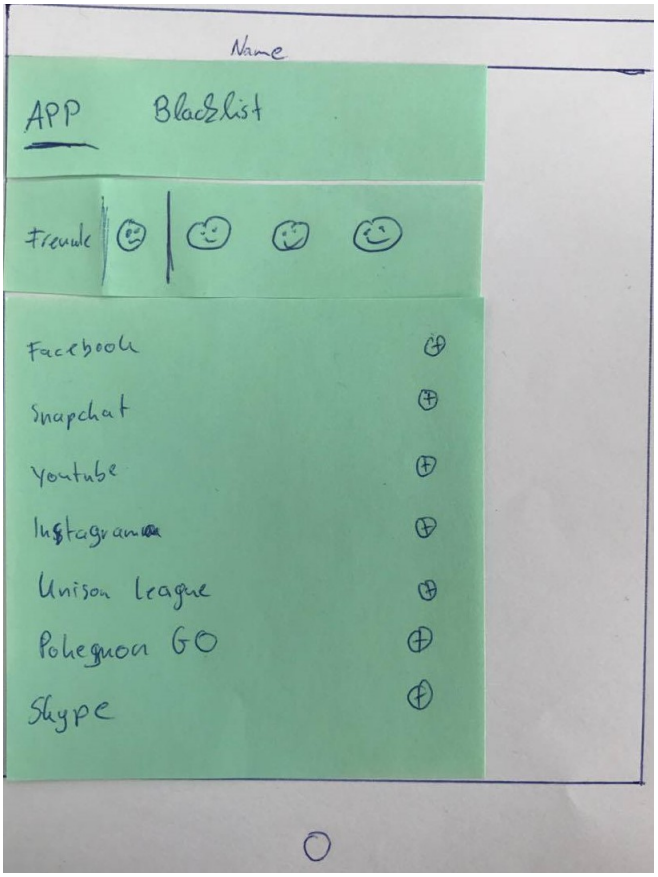


Abbildung 3.9: Paper Prototyping - Freunde

In der letzten Abbildung des Paper Prototyps befindet sich die Anwendung in der Blacklist mit den jeweils blockierten Anwendungen eines Nutzers. Diese Anwendungen sind für die Freunde, die oben aufgelistet sind, nicht mehr zugängig, solange die Freunde sich in der Nähe des Geräts befinden. Gehen die Freunde aus der Reichweite raus, werden diese Anwendungen von deren Geräten wieder entblockt.

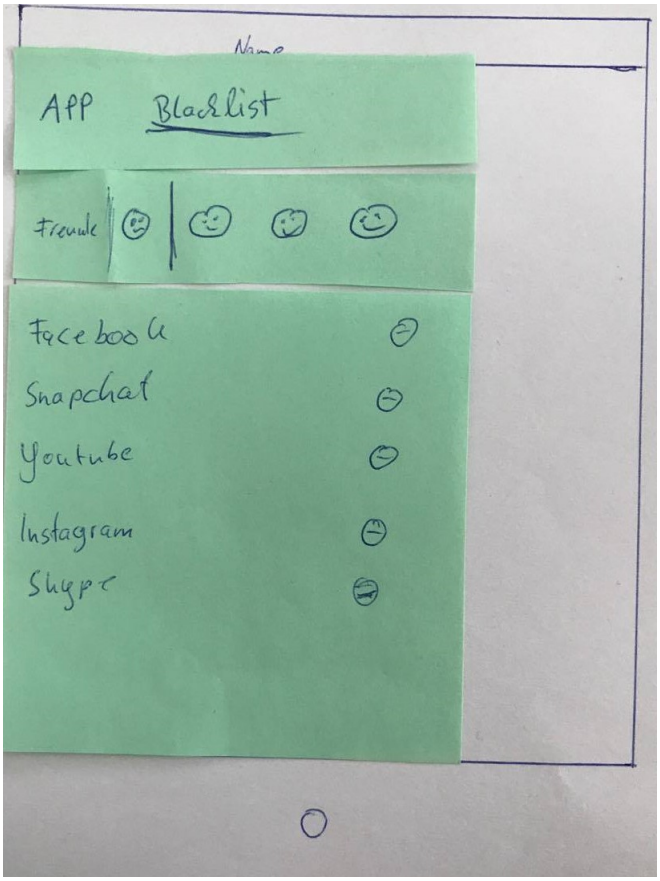


Abbildung 3.10: Paper Prototyping - Freunde mit BL

3.6 Evaluation

Nachdem das Paper Prototyp erstellt wurde, hat Phi Hai zwei Freunde gefragt, ob sie mit ihm eine Evaluation durchführen würden. Beide bejahen die Anfrage und so setzte er sich einzeln mit den Probanden hin, um Probleme und neue Erkenntnisse zu gewinnen. Dazu wurden Aufgaben aufgestellt, die die Probanden bewältigen müssen, um daraufhin ein Feedback zum User Interface geben zu können.

3.6.1 Aufgabenstellung

Die Aufgabenstellung wurden von Phi Hai erstellt, die kurz und knapp formuliert wurden. Nachdem Phi Hai den Probanden die Situation geschildert hat, damit sie den Hintergrund der Anwendung verstehen, wurden das Paper Prototyp vorgelegt und die Aufgaben den Probanden vorgelesen. Insgesamt wurden sechs Aufgaben gestellt, um in Erfahrung zu bringen, wie das User Interface auf den Testpersonen wirkt. Die Testpersonen wurden dazu aufgefordert ihre Gedanken laut auszusprechen, damit Phi Hai diese notieren kann.

1. Wie finden Sie das UI auf dem ersten Blick?
2. Fügen Sie Facebook in die Blacklist.
3. Entfernen Sie Facebook von der Blacklist.
4. Erkennen Sie Ihre Freunde?
5. Wählen Sie mehrere Apps in die Blacklist.
6. Löschen Sie alle blockierten Apps wieder.

Proband 1

1. Proband 1 findet das UI übersichtlich und einfach gehalten.
2. Hat statt dem Plus-Button auf dem Namen der App gedrückt, weil sie dachte, dass der Plus-Button für Informationen über die App selbst steht.
3. Das Entfernen der App ging reibungslos, weil sie den Minus-Button direkt erkannt und draufgedrückt hat.
4. Sie hat die Freunde erkannt und hat draufgedrückt, weil sie dachte, dass weitere Informationen über den Nutzer angezeigt wird.
5. Nachdem Sie erkannt hat, dass der Plus-Button für das Hinzufügen dient, lief die Frage ohne Probleme.
6. Auch hier gab es keine Probleme.

Proband 2

1. Proband 2 findet das UI schlicht und einfach gehalten.
2. Hat direkt erkannt, dass der Plus-Button zum Hinzufügen dient. Deswegen gab es da keine Probleme.
3. Auch hier gab es keine Probleme, da die Testperson zunächst in die Menüleiste Blacklist gegangen ist und die App wieder entblockt hat mit dem Minus-Button.
4. Die Testperson erkennt die Freunde.
5. Beim Auswählen der Apps gab es keine Probleme.
6. Sie fand es mühsam alle Apps abzuwählen und wünscht sich ein alle-auswählen-Button.

3.6.2 Feedback

Nach dem Test mit den Probanden gab es einige Anregungen, die die Anwendung potentiell verbessern bzw. erweitern könnte.
Proband 1 wünscht sich weitere Optionen im Bereich Profilbild, wo man weitere Informationen über die Freunde sehen kann und zudem eventuell auch die blockierten Liste von der Person sieht.
Erwähnt wurde von Proband 2, dass unter den Profilbild der Name des jeweiligen Nutzers stehen soll, damit man direkt erkennt, wer die Person ist. Außerdem wünscht sie sich optional ein alle-auswählen-Button, damit man nicht ständig einzelne Apps aus der Blacklist entfernen muss. Zudem sollte laut Ihrer Aussage die Apps aus der App-Liste, die ausgewählt wurden entweder gekennzeichnet werden mit einer Farbe oder komplett aus der App-Liste verschwinden, solange die Anwendung in der Blacklist ist.

Im Großen und Ganzen finden beide Testpersonen die Anwendung schlicht und einfach gehalten und man behält die Übersicht über jegliche Aktivitäten und würden diese Anwendung im Privaten nutzen wollen. Durch diese Erkenntnisse konnte Phi Hai neue Anforderungen für die Anwendung aufstellen.

3.7 Implementierung

3.7.1 Erkennen der Vordergrund-Apps

Gegen Ende der ersten Iteration wurden schon erste Anforderungen abgeleitet und ermittelt. Dabei haben sich wie bereits beschrieben zwei wichtige Anforderungen herauskristallisiert, die für das Projekt ausschlaggebend sind. Deshalb hat das Team beschlossen, mit der Implementation der Anforderung F10 zu beginnen. Das Team hat dafür anfangs einige UI-Elemente implementiert wie die ListView zur Darstellung der installierten Apps und die TabView zum Wechseln zwischen App-Liste und Blacklist. Es wurde mithilfe des PackageManager in Android die installierten Apps ausgelesen und über die UI mit der ListView dargestellt. Dadurch wurde die Anforderung F40 bereits realisiert.
Nachdem bereits Teile der UI implementiert wurde, hat Giang angefangen, die Anforderung F10 zu programmieren. Dabei hat er versucht, die Apps, die auf dem Smartphone gerade im Vordergrund laufen auszulesen. Es wurden mehrere Vorgehensweise ausprobiert, da eine übliche Methode getRunningTasks aus der ActivityManager API deprecated ist. Außerdem hat Giang das Auslesen der Vordergrund-Apps mit dem UsageStatsManager API getestet. Jedoch konnte mit dieser API die Vordergrund-Apps nicht ausgelesen werden. Deshalb hat das Team nach einem weiteren Lösungsansatz recherchiert, so dass die Library von Rummler (2015) gefunden wurde. Die Library bietet Funktionen an, die die Vordergrund-Apps auslesen können. Im Listing 3.1 wird die Methode dargestellt, die die Vordergrund-Apps untersucht und wiedergibt.

Listing 3.1: Android Processes: getRunningForegroundApps

```
List<AndroidAppProcess> processes =
    AndroidProcesses.getRunningForegroundApps(BlockService.this);
```

3.7.2 Blockieren von Applikationen

Dadurch war es nun möglich, diese Apps mit die im Code definierten Apps zu vergleichen und bei Gleichheit das Blockieren auszuführen. Da es in Android nicht möglich ist, den Start einer Applikation zu verhindern bzw. zu unterbinden, hat Giang eine Alternative entwickelt. Wenn das Blockieren ausgeführt werden sollte, wird das Blockieren so simuliert, dass der Homescreen geöffnet wird, so dass die blockierte App nicht genutzt werden kann. Diese Funktionen wurden in einem Background-Service gepackt, damit unsere Anwendung im Hintergrund die Vordergrund-Apps untersucht und falls festgelegt blockiert.

Diese Methode des Blockierens hat nach dem Testen und Diskussionen im Team einige Nachteile offenbart. Erstens untersucht der Background-Service alle zwei Sekunden die Vordergrund-Apps, wodurch der Akkuverbrauch leicht erhöht wird. Außerdem wurden die zu blockierenden Apps nicht schnell genug unterdrückt, so dass sie sich noch für einige Sekunden nutzen lassen konnten. Diese Probleme will das Team in den späteren Iterationen nochmals angehen und lösen.

4 Zweite Iteration

Nach Abschluss der ersten Iteration wurden vom Team neue Aufgaben und Ziele für die zweite Iteration formuliert. Hier soll die Implementierung der Google Nearby Messages API im Mittelpunkt stehen. Dadurch soll es ermöglicht werden, dass sich Smartphones untereinander in der Umgebung finden. Anschließend müssen die Smartphones erkennen, ob sie mit den anderen Geräten „befreundet“ sind. Des Weiteren stand zum Beginn der zweiten Iteration der Audit an, in der das Team das Projekt vorstellen sollte und so Userfeedback bekommen konnte.

4.1 Audit/Userfeedback

Am Anfang der zweiten Iteration stand nun der Audit des Praxisprojekts-Seminar bevor. In diesem Audit sollte das Team das Thema, das Konzept und die Planung des Projekts anderen Teilnehmern vorstellen. Die Projektidee bekam viel Resonanz von den Teilnehmern und dem Team wurden viele Fragen gestellt und mögliche Probleme und Anregungen aufgezeigt. Beispielsweise wurde ein Feature vorgeschlagen, der es ermöglichen soll, wenn sich bspw. eine Gruppe einigt, dass das Blockieren für eine kurze Zeit abgeschaltet werden könnte, damit man als Gruppe z.B. zusammen ein Foto schießen kann.
Außerdem wurden in der ersten Iteration Anforderungen F40 und F50 ermittelt, die im Audit jedoch negative Resonanz fanden, da manche ihre Apps nicht auflisten wollten, weil es zu privat oder peinlich wäre. Nach einer Diskussion im Team und mit Betreuer Prof. Dr. Matthias Böhmer wurde entschieden, diese Anforderungen zu entfernen bzw. zu ändern. Dadurch sieht die aktualisierte Anforderungsliste wie folgt aus. Dabei steht rot für die entfernten und grün für die neu hinzugefügten Anforderungen.

- F10: Die Anwendung muss in der Lage sein, andere installierte Anwendungen zu blockieren.
  - F11: Die Anwendung muss fähig sein, den Zugang zu anderen Anwendungen zu blockieren.
  - F12: Die Anwendung muss fähig sein, bestimmte Anwendungen nur zu blockieren, wenn mindestens ein Freund in der Nähe ist.

- F20: Die Anwendung muss Freunde in der Nähe identifizieren können.
  - F21: Die Anwendung muss fähig sein, andere Smartphones in der Umgebung zu erkennen.
  - F22: Die Smartphones, die diese Anwendung nutzen, müssen untereinander kommunizieren können.
  - F23: Die Anwendung muss fähig sein zu erkennen, welche Personen in der Umgebung Freunde sind.
  - F24: Die Anwendung muss fähig sein, Freunde darzustellen.
- F30: Die Anwendung muss eine Blacklist erstellen können, die besagt welche Anwendungen blockiert werden sollen.
  - F31: Die Anwendung muss dem Nutzer die Möglichkeit bieten, weitere zu blockierende Anwendungen in die Blacklist hinzuzufügen.
  - F32: Die Anwendung muss dem Nutzer die Möglichkeit bieten, sich in der Blacklist befindende Anwendungen zu entfernen.
- F40: Die Anwendung muss die installierten Anwendungen eines Smartphones auslesen können.
  - F41: Die Anwendung soll dem Nutzer darstellen, welche Anwendungen er installiert hat.
  - F42: Die Applikation muss dem Nutzer ermöglichen, installierte Anwendungen in die Blacklist hinzuzufügen.
- F50: Die Applikation muss die App-Liste aller beteiligten Nutzer entnehmen und zu einer neuen gemeinsamen Liste vereinigen können.
- F40: Die Applikation muss dem Nutzer die Möglichkeit bieten, von uns aus definierte Anwendungen zum Blockieren zur Verfügung zu stellen.
- F50: Die Applikation sollte dem Nutzer die Möglichkeit bieten zu sehen, wer seine Anwendungen im Moment blockiert.

4.2 Architekturentwurf

Aufgrund der Tatsache, dass durch das entstandene Userfeedback im Audit die alte Anforderung F50 entfernt wurde, wurde dadurch die geplante Anwendungslogik auf dem Server ebenfalls entfernt. In einer Diskussion im Team und nach langer Recherche und Überlegungen ist das Team zum Entschluss gekommen, in der Architektur weiterhin die Firebase Entwickler Plattform zu nutzen. Jedoch sollte der node.js-Server, der mit Firebase verbunden war, komplett aus dem Architekturentwurf entfernt werden, da ohne verteilte Anwendungslogik der Server zu diesem Zeitpunkt überflüssig ist. Der daraus resultierende Architekturentwurf wird in Abbildung 4.1 dargestellt.

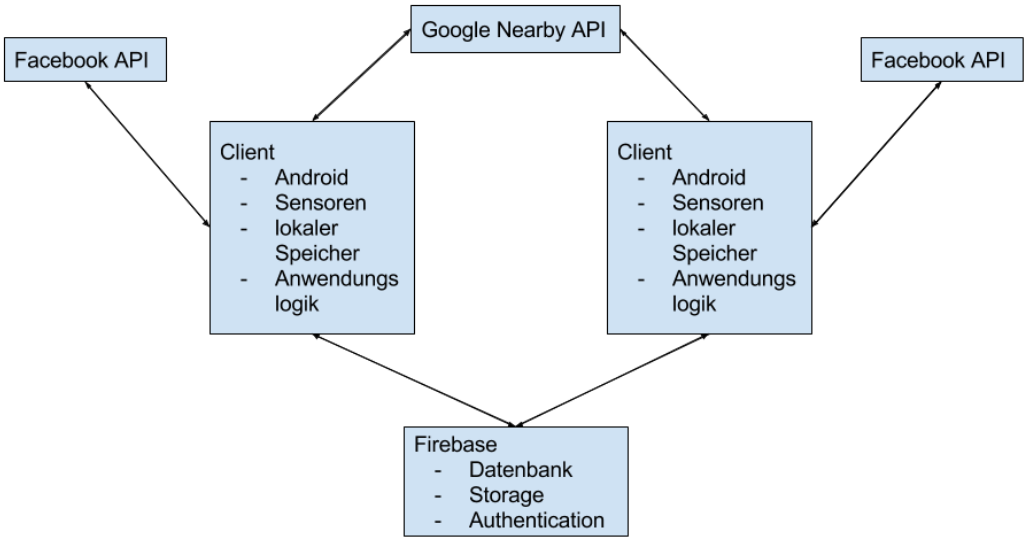


Abbildung 4.1: zweite Version des Architekturentwurfs (14.04.2017)

4.3 Implementierung

4.3.1 Telefonnummer

Das Team muss sich entscheiden, wie die Anwendung eine Unterscheidung zwischen Freunde und Nicht-Freunde erkennen soll. Praktisch wäre eine Abfrage der eigenen Telefonnummer auf alle Geräte in der Umgebung, die ShutApps benutzen. Jeder Nutzer hat normalerweise die Telefonnummern seiner Freunde im Adressbuch. Solange die Telefonnummer im Adressbuch ist, kann man die Nummern mit dem Adressbuch abgleichen. Aufgrund dieser Tatsache wäre es sinnvoll damit zu arbeiten.
Dazu hat Phi Hai zunächst auf der Seite von Android Developer sich über die Methoden in Android informiert. Dabei findet er eine Klasse, die sich ContactsContract.Data nennt und dort gibt es eine Methode, die eine Abfrage auf das Adressbuch macht. Diese Methode wurde ein wenig modifiziert. Mit dem Cursor macht er eine Abfrage auf die Datenbank beziehungsweise den Speicherort der Kontakte auf dem eigenen Geräte und in der While-Schleife soll er solange die nächste Nummer aus der Datenbanken holen, bis es keine mehr gibt und das wird in die Liste allContactNumbers gespeichert. Die Methode sieht man im Listing 4.1.

Listing 4.1: Get Contacts

```
public void getContacts(View view) {
Cursor phones = getContentResolver().query
(ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
   null,null,null, null);
        while (phones.moveToNext())
        {
            String
               name=phones.getString(phones.getColumnIndex
            (ContactsContract.CommonDataKinds.Phone.DISPLAY_NAME));
            String phoneNumber =
               phones.getString(phones.getColumnIndex
            (ContactsContract.CommonDataKinds.Phone.NUMBER))
            .replaceAll(" ", "");
            allContactNumbers.add(phoneNumber);
        }
        phones.close();
    }
```

Dazu muss jeder Nutzer die Erlaubnis erteilen, den Zugriff auf das Adressbuch zu genehmigen. Hierbei wird in der Android Manifest die Permission eingefügt. Siehe Listing 4.2.

Listing 4.2: Contacts Permission

```
<uses-permission
   android:name="android.permission.READ_CONTACTS" />
```

Das Problem auf den Phi Hai bezüglich der Telefonnummer gestoßen ist, ist das die eigene Telefonnummer nicht ausgelesen wird. Theoretisch soll die eigene Telefonnummer als hashCode an andere Geräte übermittelt werden, damit ein Abgleich stattfindet, ob die eigene Telefonnummer in der Adressbuch der Geräte in der Nähe befindet. Dadurch, dass das Team dieses Problem nicht lösen konnte, haben sie sich letztendlich dazu entschieden diesen Lösungsansatz beiseite zu legen und als Alternative die Facebook API zu benutzen, um nicht in Verzug zu kommen.

4.3.2 Facebook API

Als Alternative zu den Telefonnummern gibt die Facebook API. Für den Gebrauch benutzt das Team die Freundesliste aus Facebook. Jeder Facebook User hat eine einmalige Profil ID, mit dem man spezifisch darauf abfragen kann. Dabei kommt die Firebase Authentication zum Einsatz. Wie bereits im Punkt 3.3 Architekturentwurf zu sehen, benutzt das Team die Firebase Entwickler Plattform. Mit Inbegriffen ist auch die Firebase Authentication, womit das Anbinden von Drittanbieter vereinfacht wird. Phi Hai hat sich mit der Firebase Authentication auseinandergesetzt und die Facebook API in die Anwendung eingebunden. Dafür muss er zunächst auf der Firebase Seite ein neues Projekt erstellen und die Verknüpfung zwischen der ShutApps und der Datenbank einrichten. Danach hat er eine Login Activity erstellt, worin ein Login Button auf Facebook referenziert. Durch das Anmelden mit dem Facebook Account, holt die Anwendung über Firebase Authentication die Daten über dem Nutzer.

4.3.3 Google Nearby Messages API

Im Verlauf der zweiten Iteration hat sich das Team mit der zweiten Kernfunktion der Applikation beschäftigt. Hierbei handelt es sich um das Erkennen der Geräte untereinander. Dadurch, dass das Team sich in der Technologie für das Google Nearby API entschieden hat, um die Kommunikation zwischen Smartphones in der Nähe zu ermöglichen, liegt der Fokus nun in der Implementierung der Nearby API.
Dafür hat Giang mithilfe der Google Nearby Documentation, die API in die Anwendung eingebunden. Aufgrund der ausführlichen Dokumentation seitens Google war das Einbinden der API kein Problem. Nach dem Einrichten konnten sich bereits beide Testgeräte finden und nach dem Publish/Subscribe-Prinzip miteinander kommunizieren. Durch den MessageListener der API, der im Listing 4.3 dargestellt ist, werden Callbacks ausgeführt, wenn sich ein Gerät aus der Reichweite entfernt oder in die Reichweite eines anderen Geräts kommt. Zuvor mussten die Geräte ihre definierten Nachrichten publishen und andere Nachrichten subscriben.

Listing 4.3: MessageListener

```
mMessageListener = new MessageListener() {
        @Override
    public void onFound(Message message) {
        String messageAsString = new
           String(message.getContent());
        Log.d(TAG, "Found message: " + messageAsString);
    }
    @Override
    public void onLost(Message message) {
             String messageAsString = new
                String(message.getContent());
        Log.d(TAG, "Lost sight of message: " +
           messageAsString);
    }
};
```

Die onFound-Methode wird ausgeführt, sobald ein Gerät in die Nähe kommt. In der Variable message ist die Nachricht enthalten, die zuvor von einem Gerät veröffentlicht wurde. Dadurch ist es möglich, selbst definierte Nachrichten zu veröffentlichen, so dass andere Geräte in der Nähe diese lesen können.
Die onLost-Methode wird wiederum ausgeführt, wenn sich ein Gerät so weit entfernt, dass es von anderen Gerät nicht mehr gefunden werden kann. Dadurch können andere Geräte die Nachricht nicht mehr lesen.
Auf Basis dieser Erkenntnis ist es nun möglich, vom Team definierte Nachrichten zu publishen und mit diesen Nachrichten zu arbeiten. Als Erstes hat Giang versucht, über die Telefonnummer im HashCode befreundete Geräte zu finden. Da Phi Hai vorher die Telefonnummer der Kontakte auslesen konnte, konnte theoretisch gesehen, die eigene Nummer veröffentlicht werden und die mit der Kontaktliste des anderen Gerätes verglichen werden, um zu sehen, ob diese Geräte in einer Beziehung zueinander stehen. Jedoch war es wie bereits beschrieben nicht möglich gewesen, die eigene Telefonnummer im Gerät auszulesen, so dass eine Alternative gesucht werden musste.
Deshalb wurde mit der Facebook API gearbeitet, um eine Freundes-Beziehung zwischen zwei oder mehreren Geräte mit Nearby herzustellen. Jedoch setzt das voraus, dass der Nutzer ein Facebook-Account besitzt, um ShutApps zu benutzen. Ist man nun mit seinem Facebook-Account in ShutApps eingeloggt, soll ShutApps die Profil ID des Facebook-Accounts abfragen und diese verschlüsselt mit Nearby publishen, so dass andere Geräte in der Nähe diese Nachricht lesen können. Anschließend können sie diese Nachricht entziffern und eine Facebook API Anfrage auf die Freundesliste machen und vergleichen, ob die gelesene Profil ID sich in der Freundesliste befindet. Bei Gleichheit sind beide Geräte miteinander befreundet. Dieser Algorithmus wurde von Giang implementiert und im späteren Verlauf des Projekts weiterentwickelt.

5 Dritte Iteration

Für die dritte Iteration war nun geplant, die Google Nearby Messages API, den Algorithmus zum Erkennen der Freunde in der Nähe und das Blockieren miteinander zu verknüpfen. Außerdem wurden Szenarien geschrieben, die verschiedene Situationen in der ShutApps genutzt wird, beschreiben und dadurch neue Anforderungen abzuleiten, um die Anforderungsliste zu aktualisieren. Darüber hinaus sollten erste Datenbank-Strukturen in Firebase erarbeitet und diskutiert werden, welche Daten in der Datenbank gespeichert werden sollen. Zu guter Letzt wurde in einem Gespräch mit dem Team und Prof. Dr. Matthias Böhmer überlegt, ob es möglich wäre, die Notifications der blockierten Apps zu unterbinden. Dies soll ebenfalls in der dritten Iteration realisiert und getestet werden.

5.1 Konzeptuelle Beschreibung

Um die Anwendung für den Alltag gebräuchlich zu machen, wurden Szenarien aufgeschrieben, die im Alltag auftauchen könnten. Phi Hai hat dazu drei Szenarien aufgeschrieben und dies analysiert, um eventuell neue Anforderungen zu ermitteln, die für die App von Bedeutung sein könnten.

5.1.1 Szenarien

1. An einem regnerischen Mittwochnachmittag ist Franz Bauer unterwegs in die Stadt und will sich mit Alex Schrader auf einem Kaffee treffen. Beide haben sich seit der Hochzeit von Alex vor einem Jahr nicht mehr getroffen und haben deswegen viel zu erzählen. Die beiden Freunde sind auf Facebook befreundet und benutzen die App „ShutApps“ auf ihrem Android Gerät, weil beide sich schnell von ihrem Smartphone ablenken lassen und nicht wollen, dass die Konversation dadurch gestört wird. Franz kommt zuerst am Treffpunkt an, setzt sich an einem freien Tisch hin und öffnet die App, um Facebook, Instagram und Pokemon GO in die Blacklist hinzuzufügen. Als Alex sich auf dem Weg gemacht hat, hat er die App gestartet um Snapchat, Youtube und Whatsapp in seine Blacklist hinzugefügt. Kurz vor seiner Ankunft schaut Alex auf die App, um nochmal nachzuschauen, ob er die Apps in die Blacklist hinzufügt hat und bekommt zufällig im selben Moment sieht er das Profilbild von Franz, der sich in der Nähe befindet und auch Apps blockiert hat. So treffen sich beide in der Bäckerei und unterhalten sich über die verpassten Themen. Zufällig taucht der Abteilungsleiter von Alex auf, mit dem er auch in Facebook befreundet ist und setzt sich zu seinen Freunden, die auf ihn gewartet haben.

Nachdem er sich einen Kaffee und ein Streusel-Kirsch Kuchen bestellt hat, bemerkt der Abteilungsleiter, dass sein Whatsapp blockiert wurde, da er nach seinem Meeting die App noch nicht geschlossen hat. Er schaut in die „ShutApps“ und sieht, dass Alex sich in der Nähe befindet und eine Blacklist erstellt hat, wo Whatsapp, Youtube und Snapchat blockiert wurde. Der Abteilungsleiter hält Ausschau nach Alex und findet ihn auch in der Bäckerei. Er geht direkt zu ihm hin und sagt: „Hallo Herr Schrader, mir ist gerade aufgefallen, dass ich mein Whatsapp nicht mehr nutzen kann. Können Sie mich bitte aus der Freundesliste abwählen?“ Daraufhin fragt Alex: „Oh, sie haben auch gerade die ShutApps auf?“ Und fügt hinzu, dass er ihn aus seiner Freundesliste abwählt. Der Abteilungsleiter bedankt und verabschiedet sich höflich von Alex, schließt zugleich die „ShutApps“, weil es zuvor nicht möglich war und geht wieder zu seinen Freunden, die die App nicht benutzen. Und so unterhalten sich die beiden weiter, bis Franz zu einem Termin gehen muss. Deswegen verabschieden sich die beiden und gehen getrennte Wege. Die App entblockt nach einer kurzen Distanz und beide erhalten jegliche Benachrichtigungen von deren Apps, die sie zuvor blockiert haben.

Analyse

In diesem Szenario ist es wichtig für unsere App, dass die Personen in Facebook befreundet sind. Sonst würde unsere App die Menschen, die die App benutzen nicht zwischen Freunde und nicht Freunde unterscheiden können. Denn der Abteilungsleiter wurde nur von Alex geblockt, aber nicht von Franz. Zudem konnte der Abteilungsleiter die ShutApps nicht schließen, weil er sich in einer Zone befand, wo er von Alex geblockt wurde. Also musste er sich umschauen, ob Alex gerade in Sichtweite ist. Hierbei entsteht eine soziale Interaktion, was dazu führt, dass man mit anderen Freunden in seiner Umgebung interagieren muss. Um die ShutApps schließen zu können, darf kein anderes Gerät in der Nähe sein.

2. Es ist Samstagmittag. Maria und Carolin haben sich verabredet um Shoppen zu gehen, da der Sommer naht und beide neuen Bikinis für ihren Urlaub nach Ibiza brauchen. In der Zeit wollen beide ungestört Shoppen und haben deshalb die ShutApps gestartet, wo Maria Snapchat und Instagram blockiert hat und Carolin Whatsapp und die Kamera App. Im Zara treffen die beiden Mädels zufällig auf eine gemeinsame Freundin namens Anna, die dieselbe Intention hat wie die Maria und Carolin und so entschlossen sie sich gemeinsam auf die Suche nach neuen Bikinis zu begeben. Anna hat von der App „ShutApps“ noch nie zuvor gehört und war total begeistert, als Maria davon erzählt hat und entschloss sich die App sofort herunterzuladen. Sie meldet sich mit ihrem Facebook Account an und sieht direkt die beiden Mädels in der Freundesliste. Anna fragt Carolin: „Die Apps in der „Blacklist“ sind die Apps, die jetzt momentan blockiert sind oder?“ Daraufhin antwortet Carolin: „Ja, das sind die Apps, die Maria und ich ausgewählt haben. Falls du noch was hinzufügen möchtest, dann kannst du es ruhig tun.“ Anna hat dann Facebook noch hinzugefügt. Einige Zeit später wollen die Mädels zusammen ein Bild auf Snapchat machen, um ihre neuen Einkäufe zu teilen. Dabei bemerkt Anna, dass Snapchat ja geblockt wurde. So fragte sie:

„Besteht hier die Möglichkeit, die App wieder zu entblocken?“ Maria erwiderte: „Aber natürlich! Du musst nur auf Snapchat halten und dann erscheint die Anfrage zum Entblocken. Wir bekommen dann eine Benachrichtigung, ob wir es zulassen.“ Und so befolgt Anna die Anweisung und stellt die Anfrage. Maria und Carolin befürworten es und machen gemeinsam paar Bilder in ihrem ausgewählten Bikinis. Nachdem sie fertig waren hat Anna Snapchat wieder in die Blacklist hinzugefügt und will bezahlen gehen. Alle drei haben was für den Sommer gefunden und sind glücklich darüber. Maria und Carolin wollen noch weiter Shoppen und Anna war bedient. Deshalb verabschiedet sich Anna von den beiden und geht wieder nach Hause. Dabei bemerkt sie, dass sie kurz nach dem Abschied alle Benachrichtigungen von den geblockten Apps bekommen hat und wieder alle nutzen kann.

Analyse In dieser Geschichte geht es hauptsächlich darum, dass man in bestimmten Situationen eine App gerne benutzen möchte, die momentan geblockt wurde. Hier wird dann eine Anfrage von der Person gestellt, wo die anderen eine Benachrichtigung bekommen, ob die es zulassen oder nicht.

3. Stefan und Gernold sind gerade auf dem Weg ins Kino und wollen sich einen Film anschauen. Beide haben sich durch eine Freundin kennengelernt und haben damals ihre Nummer ausgetauscht, um mal zusammen was zu unternehmen. Sie unterhalten sich darüber, wie nervig es ist, wenn man mit Freunden unterwegs ist und alle nur auf dem Smartphone schauen. Gernold hat von einem Freund erzählt bekommen, dass es eine App gibt, die andere Apps der Freunde blockieren kann, aber sich selbst zu blockieren ist nicht möglich. Daraufhin sagte Stefan: „Ja, die App „ShutApps“ habe ich auf meinem Smartphone drauf, aber dafür muss man in Facebook befreundet sein.“ Gernold besitzt kein Facebook und hat sich überlegt, ob er sich deswegen in Facebook anmelden soll. Da sie aber schon vor dem Kino stehen hat Gernold es erstmal gelassen und sein Smartphone auf Stumm geschaltet.

Analyse

Hierbei handelt es sich um ein technologisches Problem, weil die App momentan neben Facebook keine andere Möglichkeit anbietet, die App zu nutzen. Als Voraussetzung müsste man eine andere Alternative anbietet wie beispielsweise die Telefonnummer erkennen, um dieses Problem zu lösen. Zwar kann man sein Smartphone auf Stumm oder Flugmodus stellen, aber da besteht die Gefahr, dass man aus Gewohnheit oder aus Neugier wieder sein Smartphone benutzt ohne daran gehindert zu werden.

5.1.2 Anforderungsermittlung

Aus der Analyse der Szenarien entstanden neue Anforderungen bzw. Features, die für die ShutApps interessant sind. Diese können je nachdem, wie viel Zeit übrig bleibt implementiert werden. Die neuen Anforderungen wurden in die aktuelle Anforderungsliste hinzugefügt und farblich markiert.

- F10: Die Anwendung muss in der Lage sein, andere installierte Anwendungen zu blockieren.
  - F11: Die Anwendung muss fähig sein, den Zugang zu anderen Anwendungen zu blockieren.
  - F12: Die Anwendung muss fähig sein, bestimmte Anwendungen nur zu blockieren, wenn mindestens ein Freund in der Nähe ist.
- F20: Die Anwendung muss Freunde in der Nähe identifizieren können.
  - F21: Die Anwendung muss fähig sein, andere Smartphones in der Umgebung zu erkennen.
  - F22: Die Smartphones, die diese Anwendung nutzen, müssen untereinander kommunizieren können.
  - F23: Die Anwendung muss fähig sein zu erkennen, welche Personen in der Umgebung Freunde sind.
  - F24: Die Anwendung muss fähig sein, Freunde darzustellen.
  - F24: Die Anwendung muss fähig sein, Freunde mit Profilbilder und Name darzustellen.
  - F25: Die Anwendung soll fähig sein, die Schließung der Applikation zu verhindern, solange die Person im Radius befindet.
- F30: Die Anwendung muss eine Blacklist erstellen können, die besagt welche Anwendungen blockiert werden sollen.
  - F31: Die Anwendung muss dem Nutzer die Möglichkeit bieten, weitere zu blockierende Anwendung in die Blacklist hinzuzufügen.
  - F32: Die Anwendung muss dem Nutzer die Möglichkeit bieten, sich in der Blacklist befindende Anwendungen zu entfernen.
- F40: Die Anwendung muss dem Nutzer die Möglichkeit bieten, von uns aus definierte Anwendungen zum Blockieren zur Verfügung zu stellen.
- F50: Die Anwendung sollte dem Nutzer die Möglichkeit bieten zu sehen, wer seine Anwendungen im Moment blockiert.
- F60: Die Anwendung soll dem Nutzer die Möglichkeit bieten, Freunde von der Liste abwählen zu können.
- F70: Die Anwendung muss dem Nutzer die Möglichkeit bieten, sich mit Facebook anzumelden.

- F80: Die Anwendung soll fähig sein, für einen kurzen Moment eine andere Anwendung entblocken zu können.

5.1.3 Schematische Beschreibung der Blacklist

In diesem Abschnitt wird verdeutlicht wie sich die Blacklist beziehungsweise die blockierten Apps auf den Geräten verhält, wenn Freunde oder Nicht-Freunde aufeinander treffen. Dafür wurden von Phi Hai fünf verschiedenen Situationen in der Mengenlehre abgebildet, die den Ablauf der Blacklist beschreibt und daraus Erkenntnisse gezogen.

Erste Situation - Person A und B kennen sich
In der Abbildung 5.1 sind Person A und Person B fusioniert. Jede Person beziehungsweise jedes Gerät hat eine eigene Zone um sich, wodurch die Smartphones erkennen können, wer sich in dieser Zone befindet. Die App erkennt alle Freunde, die die App „ShutApps“ benutzen und auf Facebook befreundet sind. Hierbei tretet auch das Blockieren der Apps in Kraft. Person A kann die Apps von Person B blockieren und zugleich kann Person B die Apps von Person A blockieren.

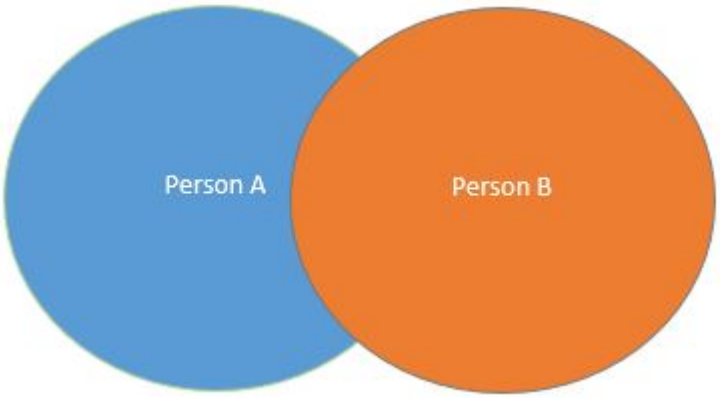


Abbildung 5.1: Erste Situation - Person A und B kennen sich

Zweite Situation - Person A, B und C kennen sich
In dieser Abbilung 5.2 sind Person A, B und C fusioniert. Alle Personen sind miteinander in Facebook verbunden und können von der ShutApps aus Apss blockieren, die für alle gelten. Falls beispielsweise Person C aus der Zone gehen sollte, werden alle Apps die von Person C geblockt wurden bei Person A und B wieder entblockt und gleichzeitig werden die geblockten Apps von Person A und B bei Person C entblockt. zwischen Person A und B bleiben die blockierten Apps bestehen. Falls mehrere Personen zukommen, die mit allen befreundet sind, würde die Personen die Liste von allen anderen bekommen.

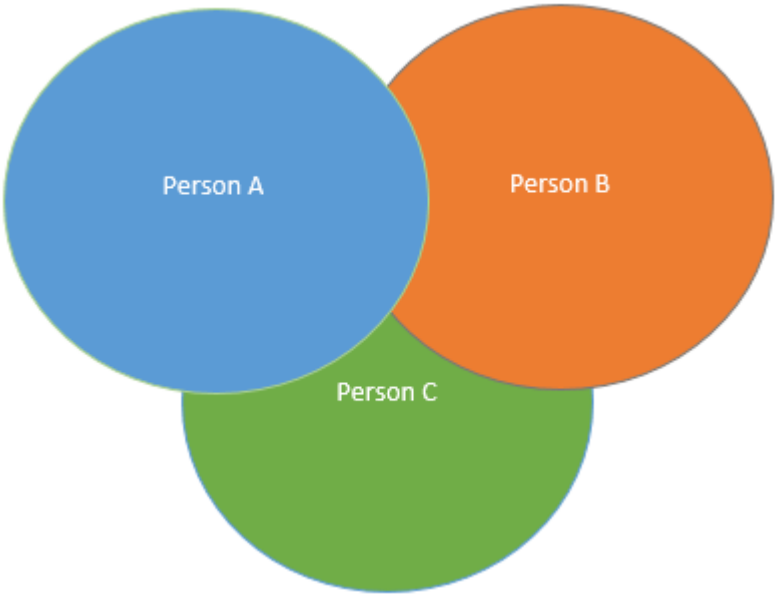


Abbildung 5.2: Zweite Situation - Person A, B und C kennen sich

Dritte Situation - Person A, B und C kennen sich nicht
In der Abbildung 5.3 benutzen Person A, Person B und Person C die ShutApps, aber kennen sich nicht bzw. sind nicht auf Facebook befreundet. Hierbei würde die Anwendung keine Reaktion zeigen, weil die Bedingung, dass sie auf Facebook befreundet sein müssen nicht erfüllt wird. Somit würde der Algorithmus des Blockens nicht laufen.

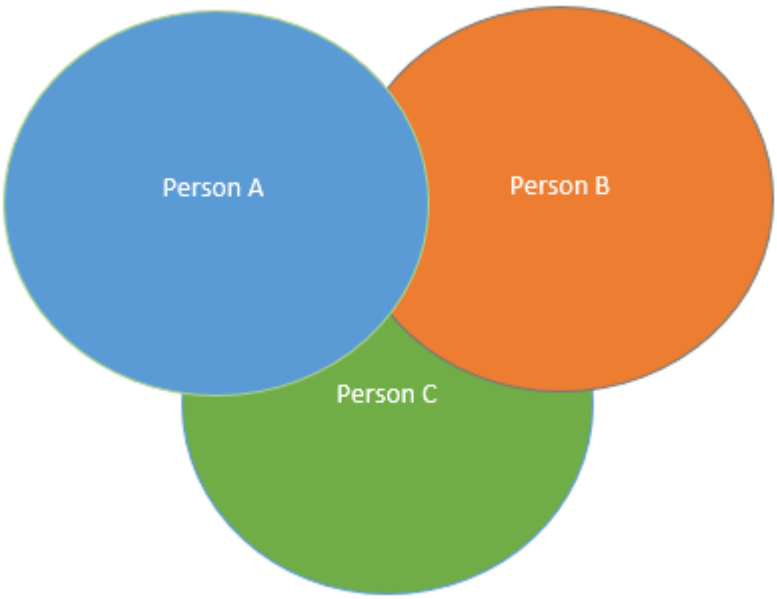


Abbildung 5.3: Dritte Situation - Person A, B und C kennen sich nicht

Vierte Situation - Person A und B benutzen die App, aber Person C nicht
In dieser Abbildung 5.4 nutzen Person A und B die ShutApps, aber Person C nicht. Person C ist aber mit Person A und B befreundet. Deswegen hätte die App keinen Einfluss auf Person C, solange Person C nicht die App installiert hat und mit Person A und Person B in Facebook befreundet ist.

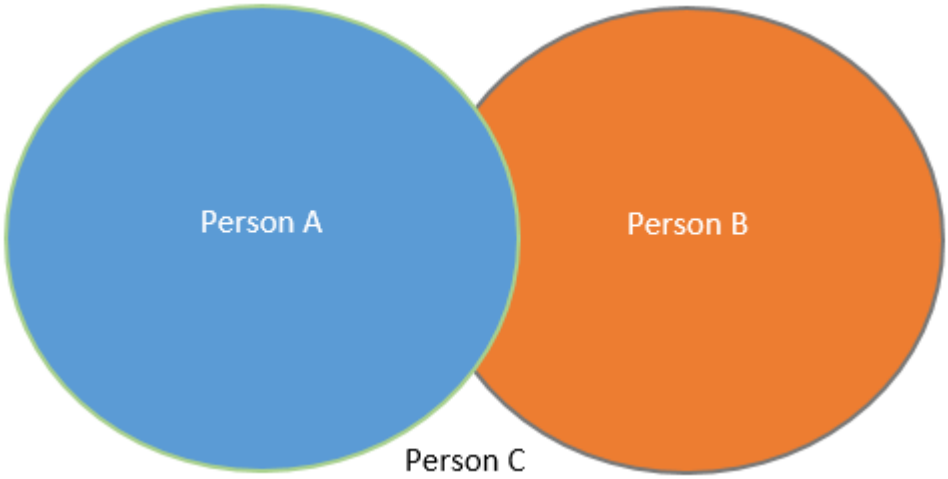


Abbildung 5.4: Vierte Situation - Person A und B benutzen die App, aber Person C nicht

Fünfte Situation - Person A und B kennen sich und Person B und C kennen sich
In der letzen Abbildung 5.5 ist Person A mit Person B befreundet und Person B mit Person C, aber Person A ist nicht mit Person C befreundet. Person C würde die Blacklist von Person B bekommen, aber nicht von Person A, weil sie nicht auf Facebook miteinander befreundet sind. Gleichzeitig würde Person A von Person B die Blacklist erhalten, aber nicht von Person C. Aber Person B würde von Person A und C die Blacklist erhalten.

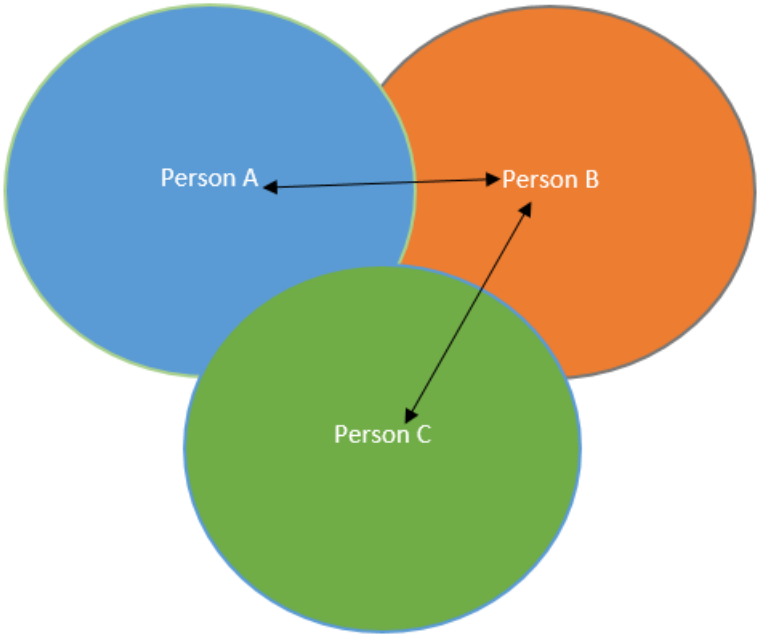


Abbildung 5.5: Fünfte Situation - Person A und B kennen sich und Person B und C

Nachdem die Situationen mit der Blacklist geschildert wurden, wurde die Wichtigkeit des Facebook Accounts für die Anwendung nochmal genauer verdeutlicht. Momentan würde die ShutApps ohne die Informationen des Facebook Accounts nicht funktionieren. Die Anwendung ist abhängig von den Facebook API. Außerdem verdeutlicht die Abbildungen mit den Beschreibungen, das Verhältnis zwischen den blockierten Apps und das Erkennen der Freunde. Jede Situation beschreibt ein mögliches Auftreten in der Realität, was dazu führt, dass diese Situation berücksichtigt werden muss. Schließlich lässt sich noch sagen, dass anhand dieser schematischen Beschreibung Erkenntnisse gewonnen wurden, wie sich die Blacklist in verschiedenen Situationen verändert und verhält.

5.2 Datenstruktur

Es wurde in einer früheren Iteration bereits entschieden, als Datenbank die Realtime-Datenbank der Firebase Entwickler Plattform zu nehmen. Jedoch kam die Datenbank bisher noch nicht zum Einsatz, da sie zu diesem Zeitpunkt noch nicht benötigt wurde. In dieser Iteration wurden vom Team erste Überlegungen angestellt, welche Daten in der Datenbank gespeichert werden müssen und in welcher Form diese festgehalten werden. Dabei wurde vom Team entschieden, die Blacklist der einzelnen Nutzer in der Datenbank zu speichern und mithilfe der Facebook Profil ID zu personalisieren, d.h. dass ShutApps mit der Profil ID auf die richtigen Blacklisten zugreifen kann.

5.3 Implementierung

5.3.1 Freunde in der Nähe blockieren

In der dritten Iteration soll nun die Verknüpfung von Google Nearby Messages API, dem Algorithmus zum Erkennen von Freunden in der Nähe und der Blockier-Algorithmus der Apps stattfinden, damit der teilweise implementierte Prototyp ausführlich getestet werden kann.
Hierfür hat Giang folgende Logik implementiert: Als Erstes kann ShutApps die Facebook Profil ID des eingeloggten Nutzers auslesen und diese verschlüsselt mit Nearby Messages API publishen. Dadurch ist es für andere Geräte in der Nähe, die ebenfalls ShutApps nutzen, möglich, diese verschlüsselte ID zu lesen und diese mithilfe der Facebook API mit der Freundesliste zu vergleichen. Bei Gleichheit kann ShutApps als Nächstes auf dem Gerät die Background-Service starten und dadurch das Blockieren der Apps ausführen. Jedoch sind in dem Background-Service im Code bestimmte Apps festgelegt, die blockiert werden sollen. Das heißt, dass der Nutzer über das User Interface bisher keine Möglichkeit hat, eigene Apps in die Blacklist hinzuzufügen und so zu definieren, welche Apps blockiert werden sollen. Dieses Problem soll im späteren Verlauf bearbeitet und behoben werden. Sobald sich nun beide Geräte nicht mehr finden können, sollen die Background-Services auf beiden Geräte abgeschaltet werden, damit das Blockieren nicht mehr läuft. Mit dieser implementierten Logik soll es nun möglich sein, den Ablauf von ShutApps einigermaßen zu testen.

5.3.2 Testen der bisherigen Version von Shutapps

Nachdem die wichtigsten Funktionsbausteine miteinander verknüpft sind, wollte das Team die bisherige Version von ShutApps gemeinsam testen. Dafür wurden auf zwei Testgeräten ShutApps installiert und ausgeführt. Beide Geräte befanden sich zu der Zeit im selben Raum. Nach der Ausführung von ShutApps mussten sich beide Teammitglieder über Facebook in ShutApps anmelden. Nach der Anmeldung haben sich beide Geräte direkt gefunden und erkannt, dass beide in Facebook miteinander befreundet sind. Dadurch hat die App den Background-Service auf beiden Geräten gestartet, der nun bestimmte Apps blockieren sollte. Nachdem auf einem Gerät in den Homescreen gewechselt wurde, hat das Team jedoch bemerkt, dass sich beide Geräte nicht mehr finden können. Dadurch wurde der Background-Service wieder geschlossen.

Nach kurzer Recherche hat das Team erfahren, dass Google Nearby bisher nur im Vordergrund läuft, d.h. dass ShutApps ständig im Vordergrund sein muss, damit sich die Geräte finden können. Aufgrund dessen konnte in dieser Version von ShutApps nicht getestet werden, ob eine App blockiert ist, wenn ein Freund in der Nähe ist. Dieses Problem musste in diesem Fall so schnell wie möglich gelöst werden.

5.3.3 Ansätze zum Background-Scanning

Damit man nicht ständig in ShutApps sein muss, um Geräte miteinander zu verbinden, hat das Team nach Alternativen recherchiert. Das Team ist dabei auf das Background-Scanning gestoßen. Zum Zeitpunkt dieser Recherche hat Google Nearby Messages API Background-Publishing und -Subscribing jedoch nur für iOS angeboten. Für Android steht lediglich das Background-Subscribing zur Verfügung. Nach längerem Testen des Background-Subscribings hat Giang erkannt, dass im Background-Subscribing nur gescannt wird, wenn der Bildschirm an ist. Für das Testen wäre dies genügend, aber für die alltägliche Nutzung von ShutApps wäre dies unvorteilhaft. Deshalb hat Giang nach weiteren Alternativen gesucht und ist auf die Android Beacon Library von Young (2014) gestoßen. Nach längerem Lesen der Dokumentation der Android Beacon Library wurde erkannt, dass dies eine gute Alternative zu der Google Nearby API wäre, um im Background zu publishen und zu subscriben. Diese Alternative wurde jedoch erst in der nächsten Iteration erarbeitet, da durch das Problem mit der Google Nearby API Zeit verloren gegangen ist.

5.3.4 Notifications unterbinden

In einem Gespräch zwischen dem Team und dem Betreuer Prof. Dr. Matthias Böhmer wurde überlegt, ob es möglich ist, die Notifications der blockierten Apps zu unterbinden. Das Team hat daher beschlossen, dieses Feature zu realisieren und dadurch kommt es als neue Anforderung hinzu. Die Aktualisierung der Anforderungsliste erfolgt im späteren Verlauf der Dokumentation.
Für die Realisierung des Unterbinden von Notifications hat Giang in der Android Developer nach APIs gesucht, die mit den Notifications arbeitet. Dabei ist er auf den NotificationListenerService gestoßen, der die Notifications auf einem Smartphone erhält. Dadurch war es möglich, eine Logik zu implementieren, die nach Erhalt einer bestimmten Notification, diese direkt aus der Benachrichtigungszentrale löscht. Mit dieser Logik konnte dem Nutzer bestimmte Notifications unterbunden werden, so dass sie nach Erhalt nicht gelesen werden konnten. Jedoch sollten die Notifications nicht nur unterbunden werden, sondern nach dem Entblocken für den Nutzer wieder sichtbar gemacht werden, so dass er sehen kann, welche Notifications er in der Zeit des Blockierens verpasst hat. Dieser Punkt sollte jedoch im späteren Verlauf des Projekts nochmal erfasst und erarbeitet werden.

6 Vierte Iteration

Gegen Ende der dritten Iteration standen die meisten Anforderungen fest, so dass noch kaum konzeptuelle Arbeit ansteht. Deshalb fokussierte sich die vierte Iteration auf die Implementation von ShutApps. Dabei stand der Wechsel von Google Nearby Messages zu Android Beacon Library als Alternative bevor. Außerdem wurden Updates an dem User Interface vorgenommen und die Firebase Datenbank integriert, damit es dem Nutzer möglich war, über das User Interface zu blockierende Apps zu festzulegen. Des Weiteren wurde an dem Unterbinden der Notifications weitergearbeitet und versucht, unterbundene Notifications zu speichern und an einem späteren Zeitpunkt dem Nutzer aufzuzeigen.

6.1 Anforderungsermittlung

Im Gespräch zwischen dem Team und dem Betreuer Prof. Dr. Matthias Böhmer entstanden neue Anforderungen in Bezug auf das Unterbinden von Notifications. Wie in Punkt 5.3.4 erwähnt, soll an dieser Stelle die Anforderungsliste final aktualisiert werden, da in dieser und der letzten Iteration der Fokus auf die Implementation mit Testen gelegt werden sollte. Deshalb soll im Folgenden die finale Anforderungsliste dargestellt werden:

- F10: Die Anwendung muss in der Lage sein, andere installierte Anwendungen zu blockieren.
  - F11: Die Anwendung muss fähig sein, den Zugang zu anderen Apps zu blockieren.
  - F12: Die Anwendung muss fähig sein, bestimmte Anwendungen nur zu blockieren, wenn mindestens ein Freund in der Nähe ist.
  - F13: Die Anwendung muss fähig sein, Notifications während der Nutzung zu unterbinden.
  - F14: Die Anwendung soll dem Nutzer die Möglichkeit bieten, die zuvor unterbundenen Notifications nach der Nutzung anzuzeigen.
- F20: Die Anwendung muss Freunde in der Nähe erkennen können.
  - F21: Die Anwendung muss fähig sein, andere Smartphones in der Umgebung zu erkennen.
  - F22: Die Smartphones, die diese Anwendung nutzen, müssen untereinander kommunizieren können.
  - F23: Die Anwendung muss fähig sein zu erkennen, welche Personen in der Umgebung Freunde sind.

  - F24: Die Anwendung muss fähig sein, Freunde mit Profilbilder und Name darzustellen.
  - F25: Die Anwendung soll fähig sein, die Schließung der Anwendung zu verhindern, solange die Person im Radius befindet.

- F30: Die Anwendung muss eine Blacklist erstellen können, die besagt welche Anwendungen blockiert werden sollen.
  - F31: Die Anwendung muss dem Nutzer die Möglichkeit bieten, weitere zu blockierende Anwendungen in die Blacklist hinzuzufügen.
  - F32: Die Anwendung muss dem Nutzer die Möglichkeit bieten, sich in der Blacklist befindende Anwendungen zu entfernen.

- F40: Die Anwendung muss dem Nutzer die Möglichkeit bieten, von uns aus definierte Anwendungen zum Blockieren zur Verfügung zu stellen.

- F50: Die Anwendung sollte dem Nutzer die Möglichkeit bieten zu sehen, wer seine Anwendungen im Moment blockiert.

- F60: Die Anwendung soll dem Nutzer die Möglichkeit bieten, Freunde von der Liste abwählen zu können.

- F70: Die Anwendung muss dem Nutzer die Möglichkeit bieten, sich mit Facebook anzumelden.

- F80: Die Anwendung soll fähig sein, für einen kurzen Moment eine andere Anwendung entblocken zu können.

6.2 Implementierung

6.2.1 Wechsel von Google Nearby zu Android Beacon Library

Aufgrund der Tatsache, dass Google Nearby in Android kein Background-Publishing unterstützt und es bisher keine Möglichkeit gibt, mit Nearby über Umwege im Hintergrund zu publishen bzw. zu scannen, musste wie bereits in der dritten Iteration beschrieben auf eine Alternative zurückgegriffen werden. Dabei hat das Team die untersuchten Technologien nochmals angeschaut und diskutiert, welche dieser Möglichkeiten nach der Google Nearby API am sinnvollsten für ShutApps wäre. Nach langer Diskussion stand fest, dass Beacons als Alternative für ShutApps am geeignetsten wäre, da sie über Bluetooth mit Smartphones in der Nähe kommunizieren können. Jedoch müsste bei der Nutzung von Beacons und ShutApps im Alltag ständig vom Nutzer ein Beacon mitgenommen werden, damit dies funktionieren könnte. Nach weiterer Recherche über Beacons hat das Team erfahren, dass es auf neueren Smartphones möglich ist, einen Beacon zu simulieren und dadurch als falscher Beacon Nachrichten in die Umgebung zu senden.

Mit diesem Wissen ist Giang wie bereits in der dritten Iteration erwähnt auf die Android Beacon Library gestoßen, die das Simulieren von Beacons auf Smartphones ermöglicht. Auch das Finden von Beacons bietet die Library als Funktion an. Aufgrund dessen musste er den Code von der Google Nearby API entfernen und die Library in ShutApps einbinden. Da die Library für Giang noch unbekannt war, musste er durch selbstständiges Testen Erfahrungen sammeln, um das Smartphone als Beacon zu simulieren und mit dem Smartphone Beacon in der Umgebung zu finden. Das Simulieren der Beacons konnte implementiert werden. Das Smartphone hat mithilfe von Bluetooth eine vordefinierte Beacon ID gesendet, die mithilfe der Beacon-App aus dem Play Store auf einem anderen Gerät empfangen werden konnte. Im folgenden Listing wird aufgezeigt, wie ein Beacon Objekt erzeugt wird und mit dem BeaconTransmitter simuliert wird. Dabei sendet das Smartphone Daten in Form des AltBeacon-Standards. Deshalb wird mit setId1() die Beacon ID festgelegt. Das folgende Listing 6.1 soll dies nochmals verdeutlichen:

Listing 6.1: Beacon Transmitter

```
Beacon beacon = new Beacon.Builder()
        .setId1("2f234454-cf6d-4a0f-adf2-f4911ba9ffa6")
        .setId2("1")
        .setId3("2")
        .setManufacturer(0x0118)
        .setTxPower(-59)
        .setDataFields(Arrays.asList(new Long[] {0l}))
        .build();
BeaconParser beaconParser = new BeaconParser()
        .setBeaconLayout(beaconLayout);
BeaconTransmitter beaconTransmitter = new
   BeaconTransmitter(getApplicationContext(),
   beaconParser);
beaconTransmitter.startAdvertising(beacon);
```

Auch war es nach längerem Probieren möglich, in ShutApps die Beacon IDs anderer Beacons zu empfangen und diese so zu finden. Für das Finden der Beacons musste Giang sich mit zwei verschiedenen Möglichkeiten auseinandersetzen. Zum Einen war es möglich mit der Montoring API bestimmte Beacons anhand der ID zu suchen und zu erkennen, wann sie in die Reichweite kommen und wann sie aus dieser Reichweite verschwinden. Diese Funktionsweise ist ähnlich wie die in der Google Nearby Library. Zum anderen gibt es die Ranging API, die anstatt bestimmte Beacons alle Beacons in der Umgebung findet und in kurzen Zeitintervallen scannt. Für den Anwendungsfall in ShutApps müssen alle Smartphones in der Nähe gesucht werden und im Falle, dass beide Geräte miteinander befreundet sind, diese Geräte genauer untersucht werden, ob sie aus der Reichweite raus sind. Aufgrund dessen hat Giang überlegt, beide APIs der Library miteinander zu kombinieren, um diesen Anwendungsfall durchführen zu können. Dabei musste zuerst die Ranging API implementiert werden, die nach Beacons unabhängig ihrer Beacon ID in der Umgebung sucht und im Erfolgsfall wiedergibt. Das folgende Listing 6.2 soll die Funktionsweise des Rangings verdeutlichen:

Listing 6.2: Ranging API

```
public void onBeaconServiceConnect() {
        beaconManager.addRangeNotifier(new RangeNotifier()
           {
            @Override
            public void
               didRangeBeaconsInRegion(Collection<Beacon>
               beacons, Region region) {

               }
            }
});
```

Hierbei wird die Methode didRangeBeaconsInRegion jedes Mal aufgerufen, wenn ein neuer Beacon gefunden werden konnte. Die gefundenen Beacons werden dann in der Collection zwischengespeichert. Nach längeren Testen der Ranging API ist Giang aufgefallen, dass der Akkuverbrauch erhöht war, da diese API in kurzen Zeitintervallen mit dem Bluetooth Scans durchführt. Trotz des erhöhten Akkuverbrauchs muss die Ranging API in ShutApps vorerst zum Einsatz kommen.
Anschließend mussten die gefundenen Geräte auf ihre Beacon IDs untersucht werden, ob sie miteinander befreundet sind. Im Erfolgsfall müssten diese Geräte genauer untersucht werden und zwar wenn sie sich außer Reichweite befinden. Hierbei sollte die Monitoring API zum Einsatz kommen. Dafür hat Giang die Monitoring API in den Callback didRangeBeaconsInRegion integriert, so dass das Monitoring direkt stattfinden kann, wenn befreundete Geräte gefunden worden sind. Das folgende Listing 6.3 soll dies nochmals verdeutlichen:

Listing 6.3: Ranging und Monitoring API

```
public void onBeaconServiceConnect() {
  beaconManager.addRangeNotifier(new RangeNotifier() {
     @Override
     public void
        didRangeBeaconsInRegion(Collection<Beacon>
        beacons, Region region) {
            beaconManager.addMonitorNotifier(new
               MonitorNotifier() {
                @Override
                public void didEnterRegion(Region region) {

                }

                @Override
                public void didExitRegion(Region region) {

                }

               }
            }
});
```

Diese Form der Callbacks ist aus der Google Nearby API bekannt. Sobald ein bestimmter Beacon in die Nähe kommt, wird die Methode didEnterRegion ausgeführt. Entfernt sich der Beacon aus der Reichweite, wird didExitRegion aufgerufen. Dadurch war es Giang möglich, eine Logik zu implementieren, dass im Falle von didEnterRegion der Abgleich mit der Freundesliste aus Facebook stattfindet und bei Gleichheit auf dem Gerät das Blockieren der Apps gestartet wird. Wird nun didExitRegion aufgerufen, wird das Blockieren der Apps beendet.
Nach erfolgreichem Implementieren dieser Alternative wurde diese Version von ShutApps im Team nochmal getestet. Im Test war es dieses Mal möglich, ins Homescreen zu wechseln, ohne dabei das Scanning zu schließen, da es im Hintergrund weiterlief. Befanden sich nun zwei befreundete Geräte in der Nähe, wurden bestimmte Apps blockiert und durch das Starten dieser Apps wurde wieder direkt in den Homescreen gewechselt. Der Anwendungsfall, dass definierte Apps im Falle, dass zwei befreundete Geräte aufeinandertreffen, blockiert werden, konnte somit durchgeführt werden. Jedoch war der Anwendungsfall zu diesem Zeitpunkt nur mit maximal zwei Geräten möglich, da der Code bisher nur auf zwei Geräte abgestimmt war. Deshalb musste ShutApps noch so erweitern werden, dass es mit drei oder mehreren Geräten funktioniert.

6.2.2 Integration der Firebase-Datenbank

In der zweiten Iteration wurde zunächst nur die Firebase Authentication für die Anbindung mit der Facebook API benutzt. Jetzt kommt die Anbindung der Firebase Datenbank, um nun über die ShutApps die Apps in die Blacklist hinzuzufügen und zu blockieren. Weil die Anbindung zwischen der ShutApps und der Firebase schon steht, hat Phi Hai ein FireBaseDatabase Objekt erstellt um darauf eine Referenz zu bekommen. Siehe Listing 6.4.

Listing 6.4: Firebase Object

```
FirebaseDatabase database =
   FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference();
```

Die Datenbank beinhaltet einmal die vordefinierten Apps für die Blacklist siehe Abbildung 6.1. Für die vordefinierten Apps wurden für den Test verschiedene Apps mit ihrem PackageName, Name, Type und Icon in die Datenbank gespeichert, damit jeder die gleiche App-Liste hat. Hier wurde die F40 in der Anforderungsliste berücksichtigt. Zudem wurden die Icons in der Firebase Storage gespeichert, um einen Verweis auf die Datenbank zu bekommen.

Abbildung 6.1: Beispiel: vordefinierte App-Liste

Des Weiteren hat Phi Hai die Facebook-ID der Nutzer in Verbindung mit der Blacklist in die Datenbank eingefügt, um zu erfahren, wer welche Apps blockiert hat. In der Abbildung 6.2 wird für jeden Nutzer, der eine App in die Blacklist hinzufügt, die Facebook-ID, der App-Name und der PackageName in die Datenbank gespeichert. Dadurch, dass die Firebase ein Realtime Database ist, wurde beim Hinzufügen oder beim Entfernen einer App die Datenbank sofort aktualisiert.

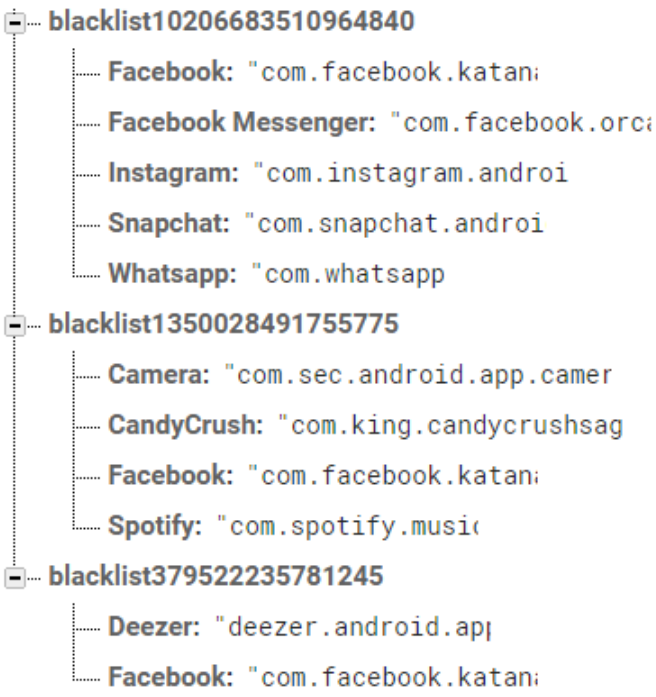
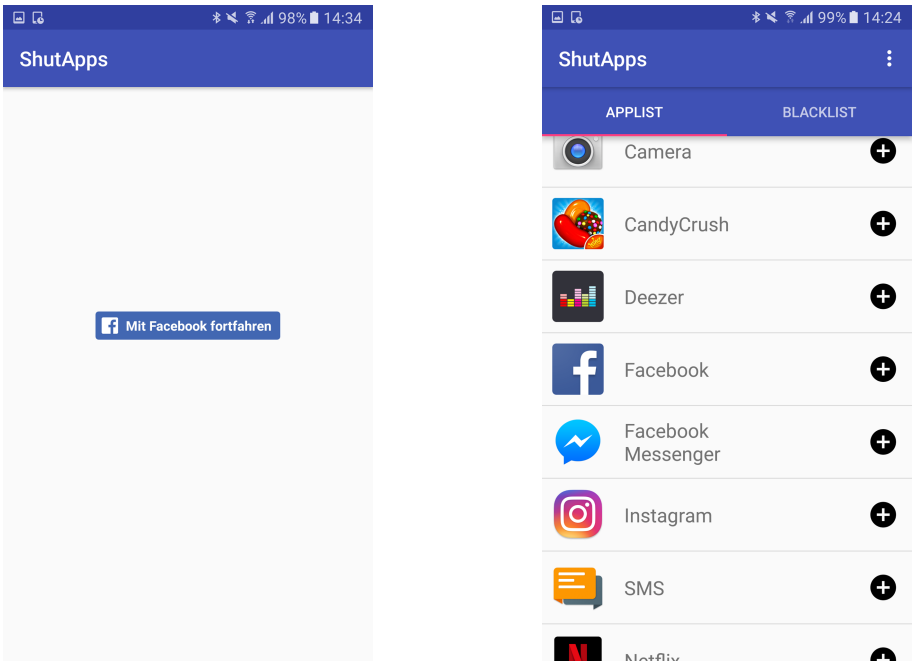


Abbildung 6.2: Blacklist mit der Facebook-ID

6.2.3 Update an dem User Interface

In Punkt 3.5 Paper Prototyp wurde das User Interface nach dem Feedback mit den Probanden beibehalten. Zuvor wurden nur einzelne Bestandteile des UI erstellt, damit das Team eine Übersicht bekommt und darin arbeiten kann. Nun wird das User Interface vervollständigt, damit die letzte Evaluation des Prototypes stattfinden kann. In der Abbildung 6.3 ist die Darstellung der App-Liste und zugleich nach dem Login-Button, die Startseite der Anwendung. Im Reiter sind zwei Menüleiste zu sehen. Und zwar die Applist und die Blacklist. Falls eine App blockiert werden soll, drückt man neben der App-Name auf das Plus-Button.



(a) ShutApps - LogIn mit Facebook

(b) ShutApps - Applist

Abbildung 6.3: Anwendung - ShutApps

Die Apps, die blockiert werden sollen, landen dann in der Menüleiste Blacklist wie in Abbildung 6.4 zu erkennen. Hier besteht die Möglichkeit, die Apps aus der Blacklist wieder zu entfernen. Dazu betätigt man neben der App-Name das Minus-Button.

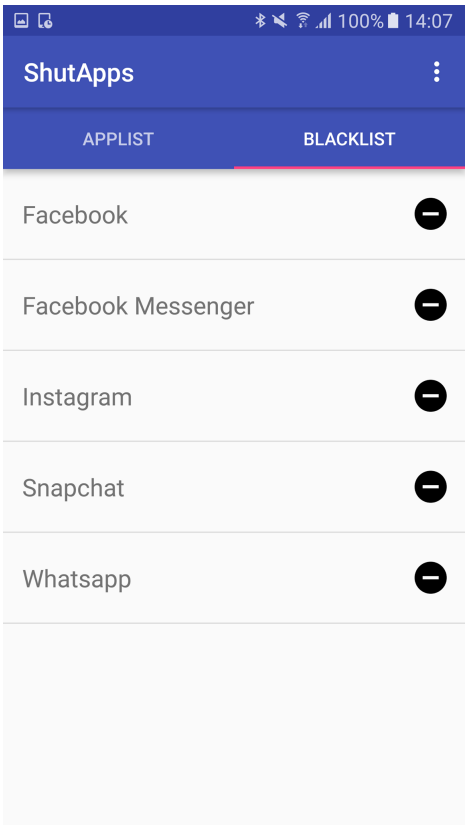
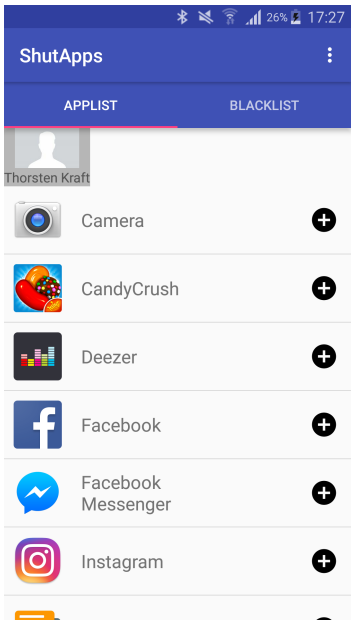
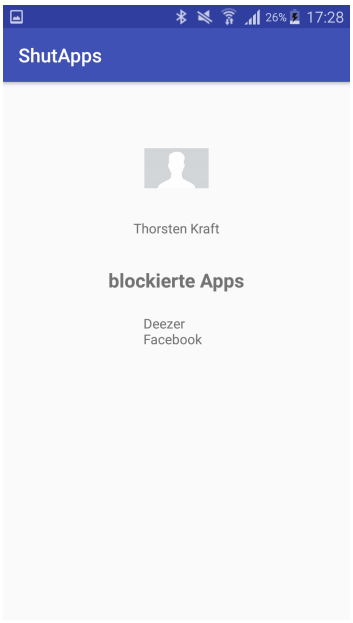


Abbildung 6.4: ShutApps - Blacklist

Wenn Freunde die ShutApps nutzen und in der Nähe sind, werden ihre Profilbilder mit ihren Namen angezeigt. Siehe Abbildung 6.5. Darüber hinaus kann man auf die Profilbilder drücken, um weitere Informationen über die Person zu bekommen. In diesem Fall werden die blockierten Apps von der jeweiligen Person sichtbar, damit der Nutzer weißt, welcher Freund welche Apps blockiert.



(a) ShutApps - Profilbild vom Freund



(b) ShutApps - Profil vom Freund

Abbildung 6.5: Anwendung - ShutApps User Interface

6.2.4 Integration des Unterbinden von Notifications

Die Machbarkeit, dass Notications unterbunden werden können, wurde in der dritten Iteration von Giang bereits geprüft. In dieser Iteration soll diese Anforderung in den Anwendungsfall von ShutApps eingebunden werden. Dadurch soll es möglich werden, dass die Notifications von blockierten Apps nicht gelesen werden können, solange sie blockiert sind. Dafür hat Giang den NotificationListenerService in ShutApps integriert und im Falle des Blockens wird die Blacklist in der Datenbank ausgelesen und die Notifications dieser Apps unterbunden. Das Listing 6.5 soll die Logik des Unterbinden der Notifications verdeutlichen.

Listing 6.5: onNotificationPosted

```
public void onNotificationPosted(StatusBarNotification
    statusBarNotification){
         String packageName =
            statusBarNotification.getPackageName();

     for(Object pkg : blockedNotifications) {
         if(packageName.equals(pkg)) {
                  savedNotifications.add(statusBarNotification)
              cancelNotification(statusBarNotification.getKey());
         }
         }
}
```

Die Methode onNotificationPosted wird immer ausgeführt, wenn eine neue Notification im Smartphone ankommt. Dadurch ist es möglich, einen Algorithmus auszuführen, sobald das Smartphone eine Notification empfängt. Dafür hat Giang den PackageName der empfangenen Notifications mit denen, in der Blacklist abgeglichen und bei Gleichheit wird die Methode cancelNotification ausgeführt, die eine Notifications aus der Benachrichtigungszentrale entfernt. Da diese Methode sofort ausgeführt wird, wenn von einer blockierten App eine Notification empfangen wird, ist gewährleistet, dass der Nutzer vorerst keine Möglichkeit bekommt, Benachrichtigungen von blockierten Apps zu lesen.
Natürlich dürfen diese Benachrichtigungen nicht für immer gelöscht werden. Deshalb müssen diese vorher zwischengespeichert werden, bevor sie gelöscht werden, damit die Nutzer diese an einem späteren Zeitpunkt lesen können. Giang hat das so implementiert, dass die zu löschenden Notifications in eine Liste zwischengespeichert wird, bevor sie gelöscht werden. Auf diese Liste soll zugegriffen werden, wenn beim Nutzer keine Apps mehr blockiert sind bzw. wenn keine Freunde mehr in der Nähe sind. Dadurch kann dem Nutzer später die Benachrichtigungen präsentiert werden, damit er schauen kann, welche Notifications er in der Zwischenzeit verpasst hat. Jedoch bestand aufgrund von Zeitmangel noch nicht die Möglichkeit, zu implementieren, dass ShutApps aus der Liste die Benachrichtigungen neu erzeugt mit ihren Titeln, Texten, Icons und Verweisungen.

6.3 Architekturentwurf

Bei der letzten Aktualisierung des Architekturentwurfs wurde der Server aus dem Entwurf entfernt, da ohne verteilte Anwendungslogik der Server in der Applikation keinen Nutzen hat. In dieser Iteration musste von der Technologie Google Nearby Messages API zu den Beacons bzw. Smartphone als Beacon gewechselt werden, da das Background-Scanning in Nearby ein Problem war. Jetzt kommunizieren die Geräte miteinander über Bluetooth und als Beacons. Dadurch hat sich der Architekturentwurf nochmals verändert, der in der Abbildung 6.6 als finale Version der Architektur dargestellt wird.

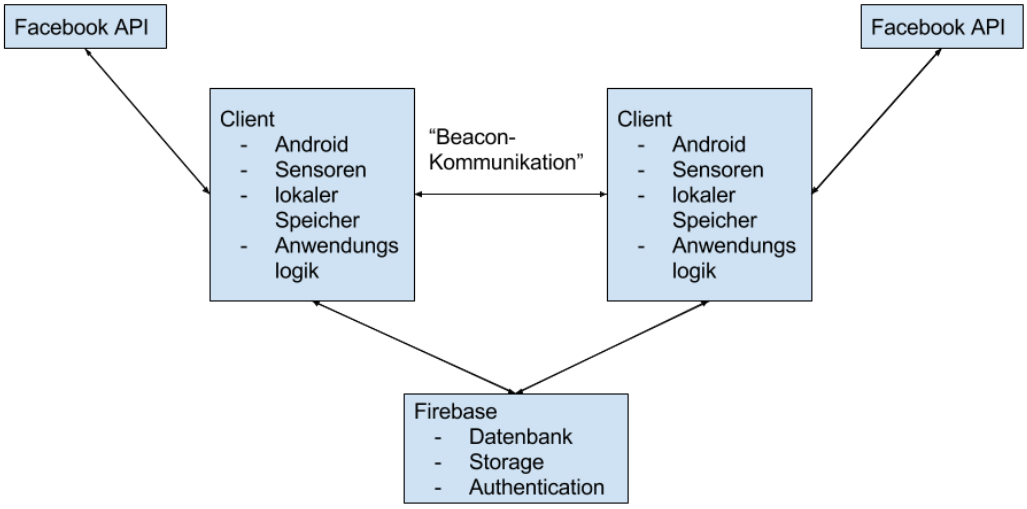


Abbildung 6.6: dritte und finale Version des Architekturentwurfs (19.05.2017)

7 Fünfte Iteration

Die fünfte und letzte Iteration wurde hinzugefügt, da durch die Verschiebung des Termins der Abschlusspräsentation genügend Zeit entstanden ist, um eine weitere Iteration durchzuführen. Diese Iteration fokussiert sich lediglich auf die Implementierung des Anwendungsfall von ShutApps für drei oder mehrere Personen und auf die Optimierung von Funktionen und Codebestandteilen.

7.1 Implementierung

7.1.1 Anwendungsfall für drei oder mehrere Personen

Bisher funktioniert ShutApps nur mit maximal zwei Personen. Für einen alltäglichen Gebrauch muss ShutApps auch mit drei oder mehreren Personen laufen, da ein Treffen nicht zwingend nur aus zwei Personen besteht. Das Problem, warum es nur mit maximal zwei Personen funktioniert, war das Zusammenstellen der Blacklist für den Background-Service bzw. auch für den NotificationListenerService für drei oder mehrere Personen. Durch die Erkenntnisse, die durch die schematische Beschreibung der Blacklist gewonnen wurden sind, konnte Giang den Code so umändern, dass der Anwendungsfall mit drei Geräten durchgeführt werden konnte. Das Testen dieser pre-finalen Version von ShutApps sollte dann einmal im Team, einmal mit dem Betreuer Prof. Dr. Matthias Böhmer und mit weiteren Testpersonen geschehen.

7.1.2 Testen des pre-finalen Prototypen

Das Team hat zusammen den Ablauf der Anwendung nochmal unter die Lupe genommen. Hierfür wurden alle Szenarien nochmal aufgerufen, die im Alltag passieren könnten. Dabei wurde explizit auf die Funktionalitäten geachtet, damit die ShutApps reibungslos läuft. Zunächst fing das Team mit zwei Geräten an. Das Team hat zuerst beide mit ihren Facebook-Account angemeldet. Nach der Anmeldung erkennt die ShutApps die Geräte und zeigt dazu das Profilbild von Facebook an, auch wenn es ein kleine Verzögerung gab. Da die Blacklist leer war, wurden auf beiden Geräten einige Apps in die Blacklist hinzugefügt, um die Funktionalität des Blockens zu überprüfen. Sowohl das Erkennen der Freunde, als auch das Blocken der Apps untereinander funktioniert einwandfrei. Selbst wenn sich ein Gerät aus der Reichweite bewegt, bleibt die Blacklist des Gerätes bestehen.

Nun kommt ein weiteres Gerät dazu, womit das Team die Situation mit drei oder mehrere Personen simulieren kann. Alle Geräte wurden wieder zusammengeführt. In Bezug auf die schematische Beschreibung in Punkt 5.1.3, wurden alle Situationen in der Anwendung durchgeführt. Auch hier funktioniert das Blockieren einwandfrei. Die Profilbilder der Freunde werden auf jedem Gerät angezeigt und selbst die Apps, die auf einem Geräte nicht installiert sind, wird spätestens nach dem Starten der App geblockt. Das einzige Problem, das dem Team aufgefallen ist, ist wenn das dritte Gerät sich aus der Reichweite befindet, löscht die ShutApps das Profilbild vom zweiten Gerät, aber das Blockieren funktioniert weiterhin auf dem zweiten Gerät. Anders gesagt, die Profilbilder entfernen sich nacheinander. Das ist ein User Interface Problem, was bis jetzt noch nicht behoben wurde.

Zwischenzeitlich hat das Team sich mit Prof. Dr. Matthias Böhmer und seinem Labor Mitarbeiter Marco getroffen, um den Test nochmal durchzuführen. Nach anfänglichen Problemen im Code, was zuvor nicht der Fall war, wurden diese Probleme gelöst und den Test mit den Herren durchgeführt. Dabei wurden nochmal alle Szenarien abgespielt, damit Prof. Dr. Matthias Böhmer die Funktionalitäten der ShutApps genauer betrachten konnte. Währenddessen gab es von Marco einige Anregungen bezüglich der Background-Service, was das Team in Betracht gezogen hat.

7.1.3 Vom Background-Service zum Accessibility-Service

Das Blockieren mithilfe des Background-Service ist bisher realisiert. Jedoch waren zwei Probleme, die in der ersten Iteration erkannt worden, noch vorhanden. Zum Einen untersucht der Background-Service alle zwei Sekunden die Vordergrund-Apps, wodurch der Akkuverbrauch leicht erhöht wird. Zum Anderen wurden die zu blockierenden Apps nicht schnell genug unterdrückt, so dass sie sich noch für einige Sekunden nutzen lassen konnten. Beim Testen mit Prof. Dr. Matthias Böhmer und dem Labor-Mitarbeiter Marco kam die Idee auf, den Background-Service mit einem Accessibility-Service zu ersetzen, da dieser nicht alle zwei Sekunden scannen muss, um zu erkennen, welche Apps sich im Vordergrund befinden. Nach einigen Denkanstößen von Marco und Diskussion im Team wurde entschieden, den Background-Service mit der Library AndroidProcesses durch den Accessibility-Service zu ersetzen. Dafür musste das Team sich über Accessibility in Android informieren und recherchieren, wie der Accessibility-Service die Vordergrund-Apps erkennt. Dabei ist das Team auf die Accessibility-Events gestoßen, die bei einem bestimmten Event die Methode `onAccessibilityEvent` ausführt. Das folgende Listing 7.1 soll den neuen Blockier-Algorithmus mit dem Accessibility-Service darstellen.

Listing 7.1: onAccessibilityEvent

```
public void onAccessibilityEvent(AccessibilityEvent
    accessibilityEvent) {
     if(accessibilityEvent.getEventType() ==
         AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED) {
          if(accessibilityEvent.getPackageName() !=
              null) {
               String currentApp =
                   accessibilityEvent.getPackageName().toString();
               if(blockedApps.contains(currentApp)) {
                   //Hier wird die App blockiert, wenn sie
                   //sich in der Blacklist befindet und
                   //gestartet wurde
               }
          }
     }
}
```

Hierbei wird ersichtlich, dass das Accessibility-Event TYPE_WINDOW_STATE_CHANGED für ShutApps von Nöten ist. Mit diesem Event ist es möglich, den PackageName der App zu bestimmen, die neu geöffnet wurde, denn das Event wird jedes Mal aktiviert, wenn im Smartphone ein neues Fenster geöffnet bzw. geschlossen wird. Dabei können es neue Activities, Dialog, Pop-Ups usw. sein.
Nach dem Implementieren des Accessibility-Services und der Integration in ShutApps hat das Team den Blockier-Algorithmus nochmals getestet. Das Testen hat folgende Erkenntnisse gebracht: Der Nutzer muss in den Einstellungen die Eingabehilfe für ShutApps aktivieren. Ansonsten würde der ganze Ablauf von ShutApps nicht funktionieren. Das Aktivieren der Eingabehilfe ist in der App selbst nicht möglich, weshalb der Nutzer in die Einstellungen geschickt werden musst, um die Eingabehilfe selbstständig zu aktivieren. Außerdem ist das Blockieren performanter, d.h. dass die Nutzer keine Möglichkeit haben, etwas in der blockierten App zu machen, wenn diese beim Öffnen blockiert wird. Durch den Accessibility-Service wird die blockierte App sofort unterdrückt. Des Weiteren wurde vom Team erkannt, dass eine Situation existiert, in der die blockierte App nicht unterdrückt wird. Diese Situation besteht dann, wenn auf einem Gerät bspw. Facebook geöffnet und offen gelassen wird. Wenn jemand auf einem anderen Gerät Facebook in die Blacklist hinzufügt, sollte Facebook im Normalfall geschlossen werden. Jedoch ist dies im Test nicht geschehen. Nach Diskussion im Team ist erkannt worden, dass der Accessibility-Service nur die Vordergrund-App erkennt, wenn sich das Fenster verändert. Deshalb war es noch möglich, bspw. in Facebook hoch- und runterzuscrollen. Wenn jedoch ein neues Fenster in Facebook geöffnet wurde, wurde die App sofort unterdrückt. Auch wenn dieses Problem bisher noch besteht, hat das Team entschieden, den Accessibility-Service dem Background-Service mit der Library AndroidProcesses vorzuziehen.

7.2 Abschlusspräsentation

Gegen Ende des Projekts hat das Team die Abschlusspräsentation des Projekts in einem Seminar vorgetragen. Daraus wurden die Erkenntnisse der Abschlusspräsentation mitgenommen, um weitere Anregungen für die Anwendung festzuhalten. Hierbei kam beispielsweise eine Frage von Prof. Christian Noss, ob man die Beacons nicht an bestimmten Orten platzieren könne, so dass das Smartphone nicht mehr als Beacon arbeiten muss. Dadurch soll es z.B. möglich werden, an bestimmten Orten das Blockieren zu starten. Diese Anregung wurde vom Team im Projekt nicht bedacht, da die Nutzung von Beacons nur zum Einsatz kam, weil der jetzige Stand der Google Nearby API für die Realisierung von ShutApps nicht komplett geeignet war.
Des Weiteren wurde eine Anregung von einem Kommilitone gegeben bezüglich der Benachrichtigung, falls ein Freund in Reichweite kommt. Falls die Situation auftreten sollte, dass ein Nutzer gerade eine App benutzt und ein Freund von ihm in Reichweite kommt, der zufällig die App blockiert hat, soll der Nutzer eine Vorwarnung bekommen, dass diese App blockiert wird, bevor es dann tatsächlich der Fall ist. Diese Anregung wurde zur Kenntnis genommen, um im späteren Verlauf als mögliches Feature zu implementieren.
Zudem hat Prof. Dr. Matthias Böhmer eine Frage an das Team gestellt, ob es immer sinnvoll ist, dass die Anwendung andere Anwendungen blockiert. Es existieren Situationen, in denen es wichtig ist, dass bestimmte Apps nicht blockiert werden dürfen. Beispielsweise erwartet jemand einen wichtigen Anruf oder eine App muss bei einem Notfall unbedingt genutzt werden können. Diese Möglichkeiten will das Team für ShutApps noch berücksichtigen.

8 Fazit/Ausblick

Fazit
Mit diesem Projekt haben wir untersucht, ob es möglich ist, durch eine mobile Anwendung, das Phubbing bei gesellschaftlichen Treffen zu unterbinden oder zu reduzieren. Dieses strategische Ziel haben wir zum jetzigen Zeitpunkt des Projekts noch nicht erreicht, da ShutApps noch im Alltag mit verschiedenen Testpersonen getestet werden muss. Durch den Einsatz im Alltag könnte erst beobachtet werden, ob ShutApps dem Phubbing bei einem gesellschaftlichen entgegenwirkt. Jedoch lässt sich sagen, dass das Team alle operativen Ziele erreicht hat, so dass eine mobile App in Android entwickelt wurde, die es dem Nutzer ermöglicht, Apps auf dem Smartphone eines Freundes zu blockieren, wenn er sich in der Nähe befindet. Durch die Realisierung von ShutApps hat das Team außerdem bewiesen, dass es mit dem jetzigen Stand der Technologie möglich ist, solch eine Anwendung in Android zu entwickeln.
Die Projektarbeit an sich ist im Team gut verlaufen, da man direkt am Anfang einen Projektplan erstellt hat, wodurch man sich Meilensteine setzt, in denen das Team bestimmte Dinge zu einem bestimmten Zeitpunkt erledigen muss. Obwohl im Projektverlauf das Problem mit der Telefonnummer und der Google Nearby API aufgetreten ist, hatte man durch den Projektplan noch genügend Zeit für solche Situationen eingeplant. Dadurch waren die Probleme nicht so schwerwiegend gewesen bzgl. des Projekterfolgs.

Ausblick
Im Ausblick möchte das Team weitere Möglichkeiten erläutern, die in ShutApps weiterentwickelt oder optimiert hätte werden können, wenn mehr Zeit zur Verfügung stünde. Als Erstes ist zu sagen, dass die restlichen Anforderungen realisiert werden könnten, damit ShutApps eine aus unserer Sicht komplette Anwendung wird. Des Weiteren lässt sich ShutApps durch weitere Features erweitern. Einige dieser Features sollen im Folgenden erläutert werden:
Ein mögliches Feature wäre z.B. die Gruppierung von einzelnen Apps in der App-Liste. Dadurch soll es möglich sein, Apps in verschiedenen Kategorien aufzulisten, damit der Nutzer statt einer einzelnen Anwendung, alle Anwendungen einer bestimmten Kategorie wie z.B. Social, Entertainment oder Games blockieren kann.
Eine weitere Eigenschaft, die ShutApps bisher noch nicht besitzt, ist es, dass die Anwendung nicht so einfach abgeschaltet werden kann. In der bisherigen Version von ShutApps lässt sich das Blockieren bspw. durch das Deinstallieren von ShutApps, durch das Stoppen des Accessibility-Service oder durch den erzwungenen Stopp von ShutApps abschalten. Dadurch, dass die Anwendung abhängig von dem Bluetooth des Smartphones ist, kann durch das Abschalten des Bluetooth, die Funktionalität von ShutApps lahm gelegt werden. Für den Gebrauch im Alltag sollte ShutApps bzw. das Blockieren der Anwendungen nicht so einfach abgeschaltet werden können.

Im Projekt wurde das Feature realisiert, dass Notifications von blockierten Anwendungen unterbunden werden. Aufgrund von Zeitmangel ist es dem Team nicht möglich gewesen, die unterbundenen Notifications an einem späteren Zeitpunkt wieder anzeigen zu lassen. Das ist nicht unter anderem nicht möglich gewesen, da die Notifications direkt gelöscht werden. Dadurch müsste man die zwischengespeicherten Notifications komplett neu erzeugen oder in einer Übersicht darstellen, damit der Nutzer weiß, welche Notifications er verpasst hat. Jedoch erlaubt die zweite Möglichkeit nur, dass der Nutzer die Notifications lesen kann. Er wird dadurch nicht in die jeweilige Anwendung weitergeleitet. Die Erweiterung des Notification-Features hätte das Team mit ein wenig mehr Zeit gerne realisiert.
Ein weiteres Feature ist wie bereits beschrieben durch das Feedback in der Abschlusspräsentation entstanden. Dabei ging es darum, dass eine zu blockierende Anwendung in ShutApps direkt blockiert wird, wenn ein Freund diese in der Blacklist hat und in die Reichweite kommt. Dadurch würde bspw. WhatsApp sofort blockiert werden, obwohl man z.B. seine Nachricht noch nicht zu Ende geschrieben hat. Der Kommilitone hatte sich gewünscht, in diese Situation durch ShutApps eine Benachrichtigung zu bekommen, dass seine App blockiert wird.
Im Falle, dass zufällig ein Freund in der Nähe ist, aber beide nicht miteinander interagieren, könnte ein Feature in Frage kommen, dass es dem Freund erlaubt, dass sich beide Freunde gegenseitig abwählen können, damit sie sich nicht gegenseitig blockieren.
Gegen Ende des Projekts hat das Team bemerkt, dass die Firebase Entwickler Plattform aktualisiert wurde. Dabei hat Firebase eine weitere Anmeldemöglichkeit in die Firebase Authentication integriert. Hier ist es nun möglich, sich durch die Telefonnummer anzumelden, wodurch ShutApps den Zugriff auf die Telefonnummer hätte und sich dadurch von der Facebook API lösen könnte. In diesem Fall könnte ShutApps die Abhängigkeit zu Facebook lösen, denn nicht jede Person besitzt einen Facebook Account. Außerdem würde ShutApps nicht funktionieren, wenn Facebook nicht mehr funktioniert. Diese Änderung ist ein großer Fortschritt in ShutApps. Aufgrund der Tatsache jedoch, dass das Update von Firebase zu spät kam, lässt sich diese Änderung an ShutApps vorerst nicht durchführen.
Ein letztes Feature, dass im Gespräch zwischen dem Team und dem Betreuer Prof. Dr. Matthias Böhmer entstanden ist, ist die Nutzung des Eddystone URL Standard anstatt des AltBeacon-Standards. Dadurch wäre es z.B. möglich eine eigene URL in die Beacon-Lösung zu integrieren und neben dem Senden einer ID auch diese eigene URL zu senden, wodurch man bspw. Werbung für die eigene Anwendung machen kann.
Abschließend lässt sich noch sagen, dass das Team mit ShutApps noch Alltagstest mit verschiedenen Testpersonen durchgeführt hätte, wenn noch genügend Zeit vorhanden wäre. Außerdem könnte das Team aufgrund der Erkenntnisse aus diesem Projekt sich vorstellen, auf Grundlage dieses Projekts in einer möglichen Bachelorarbeit weiterzuarbeiten.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Böhme 2015] Böhme, Florian: *Beacon: Was ist es und was bringt es?* http://www.wirsindwoar.de/blog/beacon-was-ist-es-und-was-bringt-es, 2015. – [zuletzt zugegriffen: 28.03.2017]

[Christian 2011] Christian: *Was ist eigentlich WiFi-Direct?* https://allaboutsamsung.de/2011/11/was-ist-eigentlich-wifi-direct/, 2011. – [zuletzt zugegriffen: 13.04.2017]

[MagicMaps] MagicMaps: *Wie funktioniert Satellitennavigation?* http://www.magicmaps.de/produktinfo/gps-grundlagen/wie-funktioniert-gps.html, . – [zuletzt zugegriffen: 28.03.2017]

[Rummler 2015] Rummler, Jared: *Android Processes.* https://github.com/jaredrummler/AndroidProcesses, 2015. – [zuletzt zugegriffen: 07.06.2017]

[Rupp 2014] Rupp, Rainer Chris und J. Chris und Joppich: *Requirements-Engineering und -Management.* München : 6. Auflage Carl Hanser Verlag, 2014

[Young 2014] Young, David G.: *Android Beacon Library.* https://altbeacon.github.io/android-beacon-library/index.html, 2014. – [zuletzt zugegriffen: 07.06.2017]

Listingverzeichnis