ulm university universität **u**ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Konzeption und Realisierung einer Home Automation Infrastruktur für mobile Endgeräte am Beispiel von Android

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Moritz Kraus
moritz.kraus@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Dipl.-Inf. Marc Schickler

2013

Fassung 15. Oktober 2013

Satz: PDF-LaTeX 2ε

Kurzfassung

Der Mensch ist stets gewillt, sich das Leben so einfach wie möglich zu gestalten. Für nahezu jedes System gibt es eine technische Lösung, die dem Menschen dessen Bedienung erleichtert. Das einfachste Beispiel sind Fernbedienungen. Wir benutzen sie, um technische Geräte im Haus, wie zum Beispiel Fernseher, HiFi-Anlagen oder DVD-Player zu steuern. Allerdings gibt es noch viel mehr Geräte im Haus, für die eine Fernbedienung sinnvoll wäre. Zum Beispiel könnte man seine Lampen mit einer Fernbedienung bequem vom Sofa aus bedienen oder eine ganze Steckdosenleiste fernsteuern, um alle angeschlossenen Geräte mit einem Tastendruck ein- und auszuschalten. Das Ziel dieser Bachelorarbeit ist es, eine Infrastruktur zu entwickeln und aufzubauen, die es ermöglicht, bestimmte Steckdosen mittels einer Android-Applikation ein- und auszuschalten. So wird das Android-Gerät zur Universalfernbedienung für alles, was an diesen Steckdosen angeschlossen wird.

Danksagung

Ich möchte mich bei verschiedenen Menschen bedanken, ohne die die Bachelorarbeit in dieser Form nicht möglich gewesen wäre. Zum Einen möchte ich mich bei meinem Betreuer, Dipl.-Inf. Marc Schickler, bedanken. Er hatte immer ein offenes Ohr für mich und so konnten Probleme schnell beseitigt werden. Da die Idee für diese Arbeit von mir stammt, bedanke ich mich im Besonderen für sein Vertrauen und dass er sich als Betreuer zur Verfügung gestellt hat.

Weiterhin möchte ich dem Prüfungsausschussvorsitzenden, Herrn Prof. Dr. Manfred Reichert, für die Möglichkeit danken, diese Bachelorarbeit in seinem Institut schreiben zu können. Ich habe bereits während meines Studiums einige interessante Vorlesungen von ihm gehört.

Zuletzt möchte ich noch meiner Familie danken, ohne die das Studium und der bevorstehende Abschluss nicht möglich gewesen wäre. Sie haben mich immer dazu animiert, neue Dinge auszuprobieren. So habe ich schon früh eine Leidenschaft für Computer entwickelt und angefangen, programmieren zu lernen. Damit wurde der Grundstein für mein Studium gelegt. Außerdem möchte ich mich für die finanzielle, kulinarische und moralische Unterstützung während meines Studiums bedanken.

Inhaltsverzeichnis

1 Einleitung

1.1 Einleitung und Motivation

Heutzutage haben die Elektrotechnik und die Informatik gravierende Auswirkungen auf den Alltag. In nahezu allen Bereichen werden Computer eingesetzt, um dem Menschen das Leben zu erleichtern. In der Automobilbranche zum Beispiel werden immer mehr Assistenz-Systeme eingebaut, die Unfälle verhindern oder deren Auswirkungen minimieren können.

Fast jeder trägt ein leistungsstarkes Smartphone bei sich, mit dem sich durch die Vielzahl an Apps die unterschiedlichsten Dinge machen lassen, wie zum Beispiel die visuelle Unterstützung des Benutzers durch Augmented Reality [GPSR13]. Durch die ständige Verfügbarkeit der Smartphones und deren permanenten Anschluss an das Internet eignen sich diese perfekt als Fernbedienung. Zum Beispiel gibt es Apps, die als

Fernbedienung für den Fernseher oder die HiFi-Anlage benutzt werden können, solange diese mit dem Netzwerk verbunden sind. Da zunehmend Hardware entwickelt wird, die mit dem Internet verbunden werden kann, entstehen mehr und mehr Anwendungsfälle für Apps, die diese Hardware kontrollieren und steuern können. Der Kreativität sind keine Grenzen gesetzt.

Im Rahmen dieser Arbeit soll ein System realisiert werden, das es ermöglicht, Steckdosen (und damit die dort angeschlossenen Geräte) im Eigenheim mittels einer Smartphone-App kontrollieren zu können. Die gewünschte Funktionalität kann nur durch das Zusammenwirken mehrerer Soft- und Hardwaremodule bereitgestellt werden, welche gemeinsam die Infrastruktur des Systems bilden.

Die Thematik dieser Arbeit lässt sich in die Rubrik *Home Automation* einordnen, die sich mit der Steuerung und Automatisierung von verschiedenen Elementen im Haus (wie zum Beispiel Lichter, Heizungen, Rollländen usw.) beschäftigt. Es existieren bereits kommerzielle Produkte mit vergleichbarer Funktionalität, allerdings sind diese noch sehr kostspielig. Das Ergebnis dieser Arbeit wird die Kosten auf einen Bruchteil der Kosten eines kommerziellen Systems beschränken.

1.2 Aufbau der Arbeit

Nachdem im vorigen Abschnitt eine einleitende Motivation gegeben wurde, werden im zweiten Kapitel die Grundlagen erklärt, auf denen das Projekt basiert. Es wird kurz das Konzept *Home Automation* vorgestellt und auf die mobile Plattform *Android* eingegangen.

Das dritte Kapitel beinhaltet die Anforderungsanalyse, die sich aus den Anwendungsfällen, die ebenfalls in diesem Kapitel behandelt werden, ergibt. Hier werden alle Anforderungen an das System erarbeitet, um einen Überblick über die zu bewältigenden Probleme und Aufgaben zu bekommen.

Das vierte Kapitel widmet sich der eigentlichen Konzeption und Realisierung des Projekts. Hier wird zunächst die Architektur des Systems vorgestellt, alle technischen Aspekte

des Projekts erläutert und die verwendete Hardware beschrieben. Anschließend wird auf die verwendete und entwickelte Software eingegangen. Zunächst wird die Server-Komponente vorgestellt, danach wird die Client-Komponente (die Android Applikation) skizziert.

Es folgt ein Anforderungsabgleich in Kapitel fünf, in dem überprüft wird, welche Anforderungen erfüllt wurden und welche Anforderungen nicht erfüllt werden konnten. Falls nicht erfüllte Anforderungen existieren, werden diese analysiert und der Grund beschrieben, weshalb die Anforderungen nicht erfüllt werden konnten. Letztendlich folgen eine Zusammenfassung der Arbeit und abschließende Bemerkungen.

2

Grundlagen

Dieses Kapitel beschreibt die Grundlagen, auf denen die Arbeit aufsetzt. Es wird das Konzept *Home Automation* vorgestellt und auf die mobile Plattform *Android* eingegangen.

2.1 Home Automation

Home Automation bezeichnet ein Konzept, dessen erklärte Aufgabe es ist, die Steuerung verschiedener Elemente im Haus (Licht, Heizung, Rollladen, Fenster etc.) zu automatisieren und/oder über das Internet zu kontrollieren.

Schon in den 80er Jahren stellte man sich das Haus der Zukunft so vor, dass es viele tägliche Aufgaben automatisch erledigt, gerade intelligente Licht- und Heizungssteuerung. Mit dem Klingeln des Weckers werden die Rollläden geöffnet und die Heizung im

Bad eingeschaltet, Musik wird automatisch der Stimmung angepasst, die Lichter werden eingeschaltet, wenn man die Wohnung betritt und ausgeschaltet wenn man sie verlässt. Das war in den 80er Jahren noch Zukunftsmusik und nicht realisierbar, stellt mittlerweile aber technisch kein Problem mehr dar. Die folgende Abbildung zeigt ein schematisches Beispiel, wie eine Home Automation Applikation ablaufen könnte.

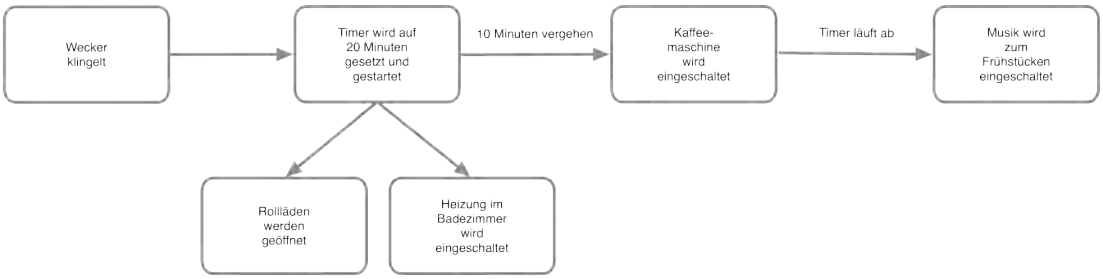

Abbildung 2.1: Beispielhafter Ablauf einer fiktiven Home Automation Applikation

Home Automation steckt noch in den Kinderschuhen. Es existieren zwar inzwischen mehrere kommerzielle Home Automation Systeme auf dem Markt, allerdings sind diese noch sehr teuer und unpopulär. Zu dieser Aussage wurde im Rahmen dieser Arbeit eine empirische Umfrage durchgeführt, in deren Umfang 47 Personen befragt wurden. Abbildung 2.2 visualisiert das Ergebnis der Umfrage.

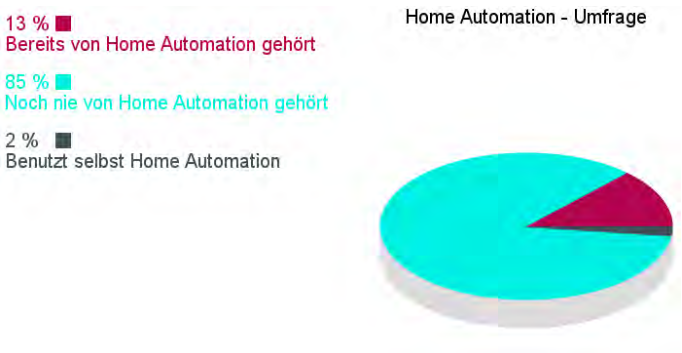

Abbildung 2.2: Ergebnis der Umfrage zu Home Automation

Aufgrund des Ergebnisses dieser Umfrage ist es sinnvoll, in diesem Bereich zu entwickeln. Mit der Zeit wird Home Automation populärer werden und die Preise für Systeme werden sinken. Noch ist die Gesellschaft allerdings nicht reif für die Technologie.

2.2 Android

Android wird von der Firma *Google* entwickelt und weiterentwickelt. Es ist ein Betriebssystem für Smartphones und Tablets. Die Plattform ist *Open Source*, was bedeutet, dass Entwickler selbst Apps programmieren, diese über den hauseigenen App-Store (*Google Play Store*) anbieten und Updates an Benutzer verteilen können. Da Android auf Java basiert, wird auch der logische Teil der Apps in Java programmiert. Die grafischen Elemente werden, im Gegensatz zu herkömmlichen Java-Applikationen, als XML-Struktur abgespeichert. Des Weiteren benötigt jede Android-App die Datei *AndroidManifest.xml*, in der grundlegende Einstellungen für die App festgelegt werden, wie zum Beispiel die minimal benötigte Android-Version und die Berechtigungen, die die App vom System benötigt, um korrekt arbeiten zu können.

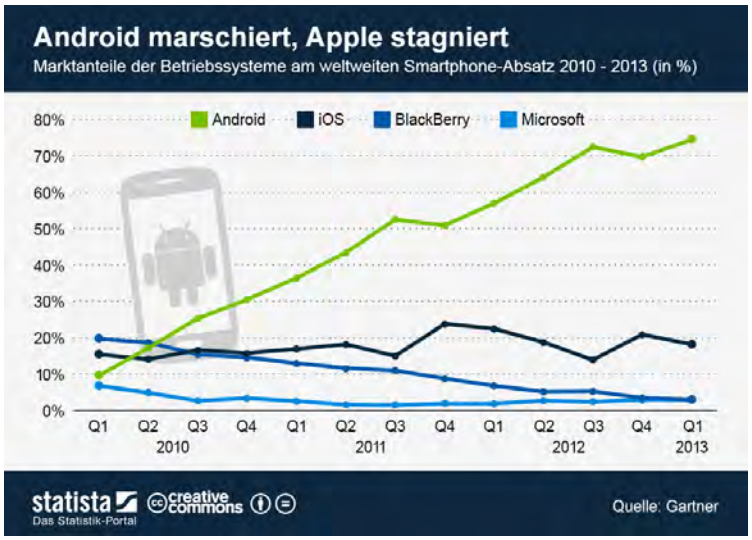

Abbildung 2.3: Entwicklung der Marktanteile von Smartphone-Betriebssystemen weltweit [Bra13]

Die Plattform ist sehr weit verbreitet. Android hat derzeit bei Tablets und Smartphones einen Marktanteil von knapp 80% [Bra13]. Das macht es sehr interessant und lukrativ, Apps für diese Plattform zu entwickeln, da viele kompatible Endgeräte existieren und die App so von sehr vielen Menschen genutzt werden kann. Abbildung 2.3 zeigt die Entwicklung der Marktanteile der verschiedenen mobilen Plattformen über die letzten drei Jahre.

3

Anforderungsanalyse

Das Ziel dieser Arbeit ist es, Funksteckdosen mittels einer Android-App zu kontrollieren. Dafür gibt es verschiedene Szenarien, aus denen sich die Anforderungen (funktionale als auch nicht-funktionale) an das System ableiten lassen. In diesem Kapitel werden zuerst die auftretenden Szenarien dargestellt und anschließend die daraus resultierenden Anforderungen dokumentiert.

3.1 Szenario

Dieses Kapitel widmet sich den verschiedenen Anwendungsfällen, die während der Benutzung des Systems auftreten können. Das folgende Diagramm zeigt alle Fälle, die es zu realisieren gilt.

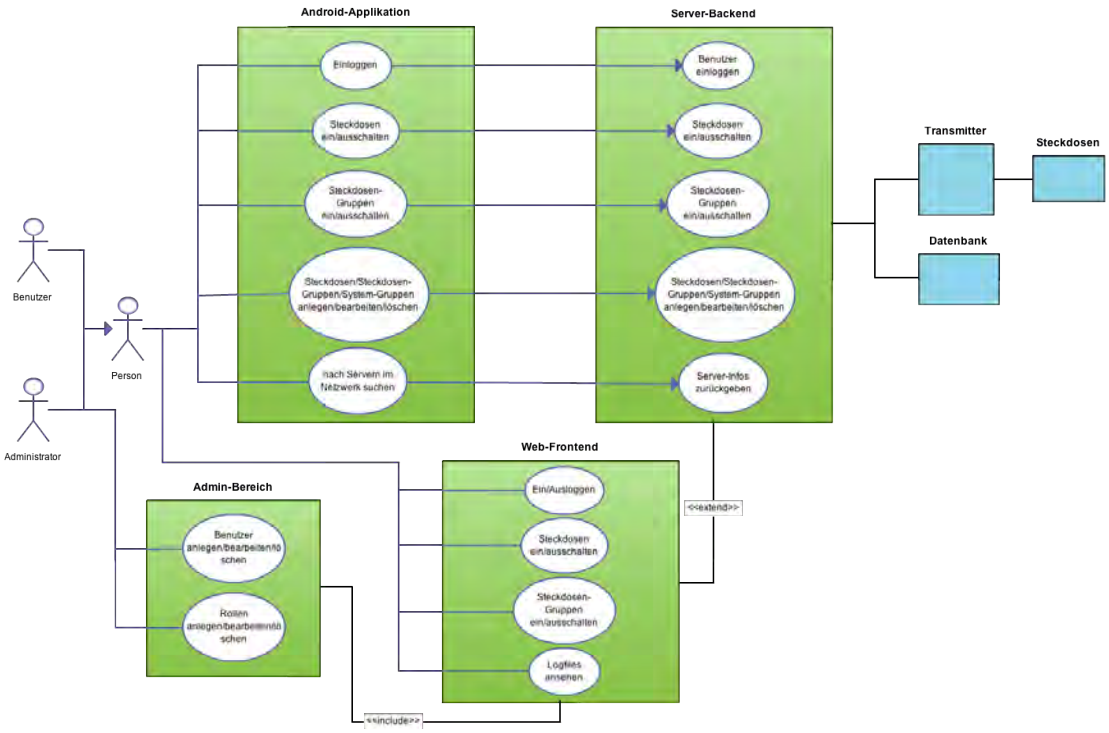

Abbildung 3.1: Anwendungsfalldiagramm des Systems

3.2 Anforderungen

Nachdem die Anwendungsfälle vorgestellt wurden, können daraus nun die Anforderungen an das System abgeleitet werden. Außer den Fällen, die direkt aus dem Anwendungsfalldiagramm ersichtlich sind, fallen noch weitere Anforderungen an. Die folgende Tabelle listet alle Anforderungen an das System auf.

Art	Anforderung	Kommentar
Funktional	Das System soll Funksteckdosen per Android-App steuern können.	Dies ist die Basisanforderung des Systems. Es sollen alle Funksteckdosen unterstützt werden, die auf dem 433.92 MHz-Frequenzband kommunizieren.
Nicht-Funktional	Das System soll jederzeit verfügbar sein.	Da die Steckdosen über das Internet gesteuert werden sollen und Zugang zum Internet eine Grundvoraussetzung für das System ist, kann das System jederzeit verfügbar sein.
Funktional	Das System soll Benutzer und Benutzer-Gruppen verwalten können.	Hierfür wird ein Administrator-Konto benötigt, das dazu berechtigt ist, neue Benutzer anzulegen/Benutzer zu editieren/Benutzer zu löschen.
Funktional	Alle nutzerspezifischen Daten sollen auf einem Server gespeichert werden.	Dies ist notwendig für eine konsistente Benutzerverwaltung. So können sich Benutzer an verschiedenen Endgeräten anmelden und immer auf den selben Datensatz zugreifen.
Funktional	Benutzer sollen nutzerspezifische Daten verwalten können.	Benutzer sollen eigene Steckdosen-Gruppen definieren, anlegen und löschen können
Funktional	Bei Gruppen sollen Start- und Stop-Skripte hinterlegt werden können.	Die Skripte sollen zusammen mit dem Ein/Ausschalten der Gruppe auf dem Server ausgeführt werden. So lässt sich die Basisfunktionalität des Systems erweitern.
Funktional	Die Android-Applikation soll sich automatisch konfigurieren können.	Die App soll das Netzwerk nach Servern durchsuchen und anschließend automatisch die Verbindung zum Server konfigurieren können

Art	Anforderung	Kommentar
Funktional	Die Android-Applikation soll automatisch die beste Verbindungsart zum Server wählen.	Die Android-Applikation soll je nach Situation entscheiden können, ob eine lokale oder globale Verbindung (eine Verbindung über das Internet) zum Server genutzt wird. Die lokale Verbindung soll gewählt werden, wenn sich der Client im selben lokalen Netzwerk wie der Server befindet, anderenfalls soll die globale Verbindung benutzt werden.
Funktional	Die Android-Applikation soll keine falsche Bedienung zulassen.	Beim Anlegen/Ändern/Löschen von Daten soll die Android-Applikation durch das Ausblenden ungültiger Datenkombinationen eine Fehleingabe verhindern.
Nicht-funktional	Die Android-Applikation soll beim Start nach spätestens fünf Sekunden benutzbar sein.	Um Frust beim Benutzer zu vermeiden, muss der Server zeitnah auf Aktionen des Benutzers reagieren.
Funktional	Der Server soll ein Web-Frontend bereitstellen.	Damit die Steckdosen auch kontrolliert werden können wenn gerade kein Smartphone oder Tablet zur Verfügung steht (zum Beispiel durch einen leeren Akku), soll zusätzlich zur Android-Applikation noch die Möglichkeit bestehen, die Steckdosen über eine Webseite zu kontrollieren.
Funktional	Der Server soll auf Veränderungen des Standorts des Benutzers reagieren können.	Eine ausgewählte Gruppe von Steckdosen soll automatisch ein/ausgeschaltet werden wenn Benutzer das Haus betritt oder verlässt.

4

Konzeption und Realisierung

Dieses Kapitel beschreibt den kompletten Entwicklungsprozess des Systems. Da das System aus mehreren Hard- und Softwarekomponenten besteht, wird zuerst ein Überblick über die Gesamtarchitektur gegeben, um zu verdeutlichen wie die einzelnen die Komponenten miteinander interagieren. Anschließend werden diese isoliert betrachtet und deren Funktionalität, Konfiguration und Notwendigkeit erläutert.

4.1 Gesamtarchitektur

Die Gesamtarchitektur stellt alle Komponenten des Systems mit deren Kommunikationswegen und Beziehungen zueinander dar. Bei der Entwicklung der Gesamtarchitektur kann ein Überblick darüber erlangt werden, wie das System funktioniert und welche Funktionen die einzelnen Komponenten erfüllen müssen. Mit diesem elementaren Wis-

sen können die einzelnen Komponenten unabhängig voneinander entwickelt werden. Sind alle Komponenten entwickelt, kann das System zusammengesetzt und getestet werden. Im Folgenden wird die Gesamtarchitektur beschrieben.

Es muss eine Möglichkeit geschaffen werden, die Steckdosen über das Internet ansprechen zu können. Dafür sind mehrere Kommunikationswege elementar. Die Steckdose muss innerhalb des Hauses mittels einer Funkverbindung angesprochen werden. Es wird ein Server im Haus benötigt, der diese Aufgabe übernimmt. Außerdem wird eine Verbindung aus dem Internet zum Server im Haus benötigt. Hier gibt es keine direkte Verbindung, da der Server aus dem Internet nicht erreichbar ist, sondern lediglich der Internet-Router. Somit muss eine Verbindung vom Client zum Router und vom Router zum Server geschaffen werden. Sind alle Verbindungen hergestellt, kann der Client über die Verbindungen mit den Steckdosen kommunizieren. Abbildung 4.1 stellt den Sachverhalt grafisch dar.

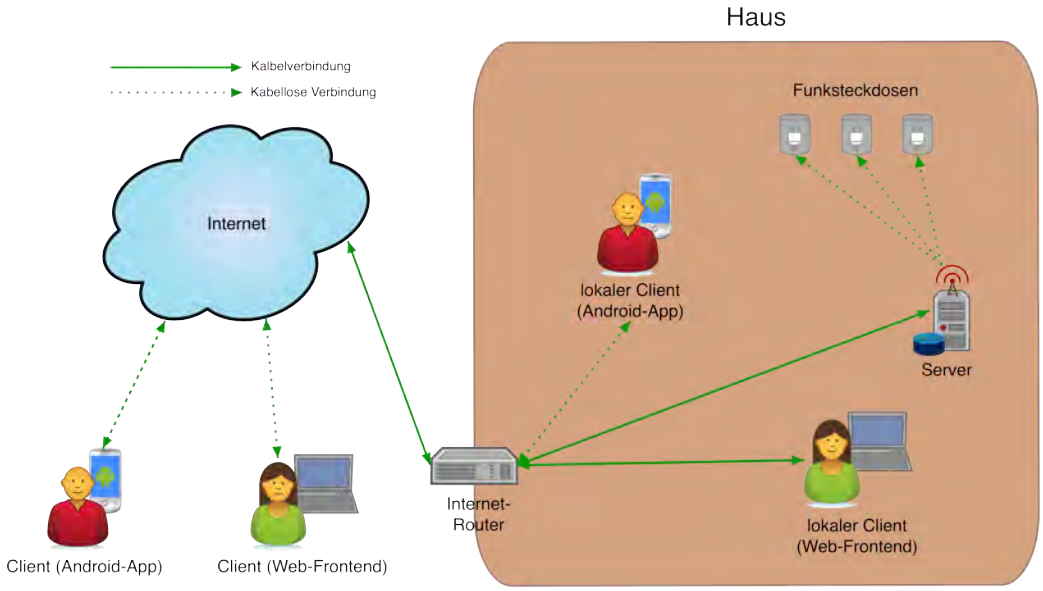

Abbildung 4.1: Gesamte Systemarchitektur

4.1.1 Verbindung vom Client zum Router

Der Router ist vom Internet aus über seine IP-Adresse erreichbar. Diese ändert sich jedoch periodisch. Deshalb muss ein DynDNS-Service registriert werden. Der Service bietet einen statischen Domain-Namen, der immer dynamisch auf die aktuelle IP-Adresse des Routers abbildet. So kann der Router aus dem Internet immer unter dem selben Domain-Namen erreicht werden. Die Login-Daten des DynDNS-Benutzerkontos müssen im Router hinterlegt sein. Außerdem muss im Router ein Port Forward eingerichtet werden, der die Verbindung an die lokale IP-Adresse und den Port des Servers weiterleitet.

4.1.2 Verbindung vom Router zum Server

Da die eigentlichen Nutzdaten nicht für den Router bestimmt sind, sondern von diesem nur an den Server weitergeleitet werden sollen, muss dem Router mitgeteilt werden, an welche lokale IP-Adresse er die Daten weiterleiten soll. Dies wird mithilfe eines Port-Forwardings realisiert. Die Daten werden auf einem vom Client bestimmten Port ankommen. In der Router-Konfiguration hat man die Möglichkeit, Tupel mit Port-Nummern und IP-Adressen zu hinterlegen. Kommen Daten auf einem Port an, der als Tupel gespeichert ist, werden die Daten an die lokale IP-Adresse des Tupels weitergeleitet. So kann der Router die Daten an den richtigen Ziel-Host transferieren.

Wenn sich der Client und der Server im selben Netzwerk befinden, ist die Verbindung über das Internet und den Router nicht notwendig, da sich der Server in diesem Fall auch direkt über seine lokale IP-Adresse erreichen lässt.

4.1.3 Verbindung vom Server zu den Steckdosen

Die Steckdosen kommunizieren mittels eines eingebauten Funkempfängers, der Signale auf der Frequenz 433,92 MHz empfangen kann. Die Steckdosen werden zusammen mit einer Fernbedienung vertrieben, die Signale auf eben dieser Frequenz senden kann. Es muss ein 433 MHz-Transmitter ähnlich dem der Fernbedienung an den Server

angeschlossen werden, damit der Server die Steckdosen mit dessen Hilfe ansprechen kann.

4.2 Verwendete Hardware

Dieses Kaptiel beschreibt die Hardware, die für dieses Projekt verwendet und teilweise auch eigenhändig hergestellt wurde. Ferner wird aufgezeigt, wie diese Komponenten zusammenwirken, um so die gewünschte und geforderte Funktionalität bereitstellen zu können.

4.2.1 Raspberry Pi

Der Raspberry Pi ist ein Einplatinen-Computer. Er misst Kreditkartengröße und gehört damit zu den kleinsten Computern der Welt. Er wurde von der Raspberry Pi Foundation[1] entwickelt und im Februar 2012 veröffentlicht.

Abbildung 4.2: Raspberry Pi rev. 2.0 model B [Ras13]

[1] Ein Zusammenschluss aus sechs Treuhändlern, die zusätzlich von der Universität von Cambridge und Broadcom unterstützt werden

In diesem Projekt wird der Raspberry Pi als zentraler Server genutzt, von dem aus sich die Steckdosen steuern lassen. Er zeichnet sich durch seinen geringen Stromverbrauch und durch die Tatsache aus, dass er eine Schnittstelle für zusätzliche Hardware in Form von Ein- und Ausgabe-Pins (sogenannte GPIO-Pins) besitzt. Diese werden im Laufe des Projekts benötigt. Es wurde ein Raspberry Pi Model B verwendet, der folgende Eckdaten aufweist:

- **CPU**: ARM1176JZFS v6 32Bit Single Core mit mathematischem Koprozessor (VPU) und DSP, 700 MHz
- **GPU**: Videocore IV, Dual Core, 128 KB L2-Cache, 250 MHz
- **Arbeitsspeicher**: 512 MB RAM, 400 MHz
- **5 Status LEDs**: Power, SD-Card Zugriff, LAN 10/100 MBit, LAN Full-Duplex, LAN Link / Zugriff
- **Gewicht**: ca. 45g
- **Maße**: 85,60mm x 53,98mm x 17mm
- **Versorgungsspannung**: 5V via MicroUSB-Anschluss
- **Stromverbrauch**: max. 3,5W (700mA)

4.2.2 Funksteckdosen

Funksteckdosen sind Steckdosen, die sich per Funkverbindung (meist mittels einer Fernbedienung) ein- und ausschalten lassen. Für diese Arbeit wurden Funksteckdosen der Serie Elro AB440 ausgewählt. Diese kommunizieren auf dem 433 MHz-Frequenzband. Grundsätzlich lassen sich auch andere Funksteckdosen verwenden, solange sie auf demselben Frequenzband kommunizieren.

Abbildung 4.3: Elro Funksteckdose der Serie AB440 [Fun13]

Jede Steckdose besitzt zwei DIP-Bänke[2](siehe Abb. 4.4). Eine DIP-Bank ist eine Gruppierung von Schaltern. Durch die Kombination der Schalter lassen sich verschiedene Zahlen binär kodieren.
Mit der ersten DIP-Bank lässt sich ein *System-Code* einstellen. Dieser stellt eine Gruppe dar, in der Steckdosen zusammengefasst werden können. Die DIP-Bank hat fünf Schalter deren Zustände beliebig kombiniert werden können, somit können insgesamt $2^5 = 32$ verschiedene System-Codes (und damit Steckdosen-Gruppen) angelegt werden. Die Schalter der DIP-Bank sind mit den Zahlen 1 bis 5 gekennzeichnet.
Die zweite DIP-Bank besteht ebenfalls aus fünf Schaltern, sie stellt den *Unit-Code* ein. Der Unit-Code identifiziert die Steckdose innerhalb der System-Code-Gruppe. Bei dieser Bank muss immer genau ein Schalter auf 1, und die restlichen Schalter auf 0 stehen. Somit kann man bei dieser DIP-Bank genau fünf verschiedene Codes einstellen. Die Schalter sind mit den Buchstaben A bis E gekennzeichnet.

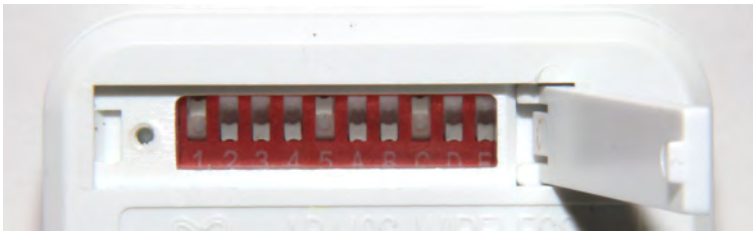

Abbildung 4.4: Einstellungsmöglichkeiten der Steckdose

[2] steht für Dual in-line package

Durch die Kombination verschiedener Werte auf beiden DIP-Bänken können maximal $(2^5 = 32) * 5 = 160$ verschiedene Steckdosen im System eindeutig identifiziert werden. In Abbildung 4.4 wären zum Beispiel der System-Code 10001 (eine dezimale 17) und der Unit-Code 00100 (Schalter C) eingestellt. Mit diesen Informationen lässt sich die Steckdose nun steuern. In der Fernbedienung wird dazu der selbe System-Code, der auch in der Steckdose eingestellt ist, per DIP-Bank eingestellt. Sie besitzt vier An/Aus Knöpfe für die Steckdosen A bis D. Wird beispielsweise der An-Knopf von Steckdose C gedrückt, wird Steckdose C in dem eingestellten System-Code eingeschaltet. Abbildung 4.5 zeigt die Kommunikation zwischen der Fernbedienung und den Steckdosen, wenn diese auf denselben System-Code eingestellt sind. Hier ist schon eine Einschränkung des kommerziellen Produkts ersichtlich. Da die Fernbedienung nur jeweils vier Knöpfe für das Ein- und Ausschalten der Steckdosen besitzt, kann sie auch nur maximal vier verschiedene Steckdosen steuern, wobei diese sich alle in derselben System-Code-Gruppe befinden müssen. Da das System insgesamt 160 verschiedene Steckdosen beinhalten kann, kann die Fernbedienung nur 2,5% der maximal verfügbaren Steckdosen steuern. Die zu entwickelnden Android-Applikation wird diesen Wert auf 100% steigern.

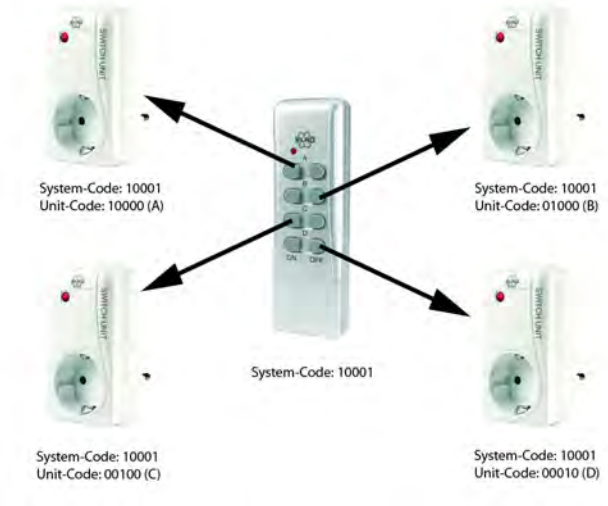

Abbildung 4.5: Kommunikation zwischen der Fernbedienung und mehreren Steckdosen mit demselben System-Code

4.2.3 433 MHz-Transmitter

Da weder ein Smartphone noch der Server (in diesem Fall der Raspberry Pi) die drahtlose Kommunikation auf dem 433 MHz-Frequenzband unterstützen, wird ein Sender (wie er auch in der Fernbedienung zu den Steckdosen verbaut ist) benötigt, der die Steckdosen ansprechen kann und seinerseits vom Server ansprechbar ist. Es wäre möglich, den Sender aus der Fernbedienung auszubauen und an den Server anzuschließen, allerdings wäre die Fernbedienung damit nicht mehr benutzbar. Um die Fernbedienung weiterhin benutzen zu können, wurde hier ein zusätzlicher Sender verwendet.

Aufgrund der Bauform des Raspberry Pi und der Tatsache, dass der Transmitter nicht exklusiv dafür konzipiert wurde, muss ein Adapter angebracht werden, um den Transmitter an den Raspberry Pi anzuschließen. Dieser besteht aus einer Lochmaske, auf die der Transmitter gelötet wird und einem weiblichen Pin-Header, der auf die GPIO-Pins des Raspberry Pi aufgesteckt werden kann.
Der Transmitter besitzt drei Anschlüsse: eine Datenleitung, eine Stromleitung und eine Erdungsleitung. Diese werden mit an der richtigen Stelle auf den Pin-Header gelötet. So werden die Leitungen an die Pins des Raspberry Pi angeschlossen, wenn der Adapter aufgesteckt wird.

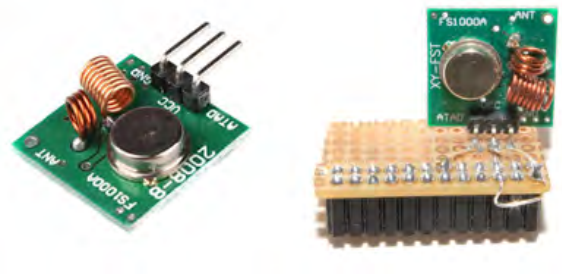

Abbildung 4.6: 433 MHz Transmitter (links) [tra13b], Transmitter mit Adapter (rechts)

Es gibt mehrere Möglichkeiten, den Transmitter an den Raspberry Pi anzuschließen. Die folgende Abbildung zeigt ein Schaubild der Pinbelegung des Raspberry Pi Model B inklusive der gewählten Anschlüsse des Transmitters.

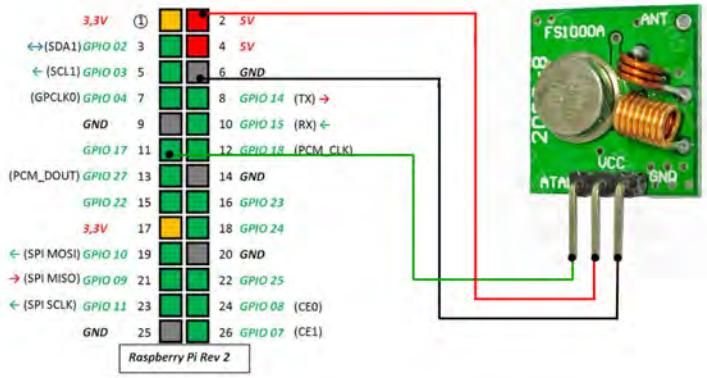

Abbildung 4.7: Anschluss des Transmitters an den Raspberry Pi Model B [tra13a]

Abbildung 4.8 zeigt den Raspberry Pi mit angeschlossenem Transmitter.

Abbildung 4.8: Raspberry Pi mit angeschlossenem Transmitter

4.3 Serverkomponente

Dieses Kaptiel beschreibt den zentralen Server in Hinblick auf dessen Software. Als Hardware dient der bereits beschriebene Raspberry Pi (siehe Kapitel 4.2.1). Der Server ist ein zentraler Bestandteil der Arbeit und besteht aus mehreren Komponenten, auf die im Folgenden im Detail eingegangen wird. In Abbildung 4.9 wird die Architektur des Servers mit allen Komponenten dargestellt.

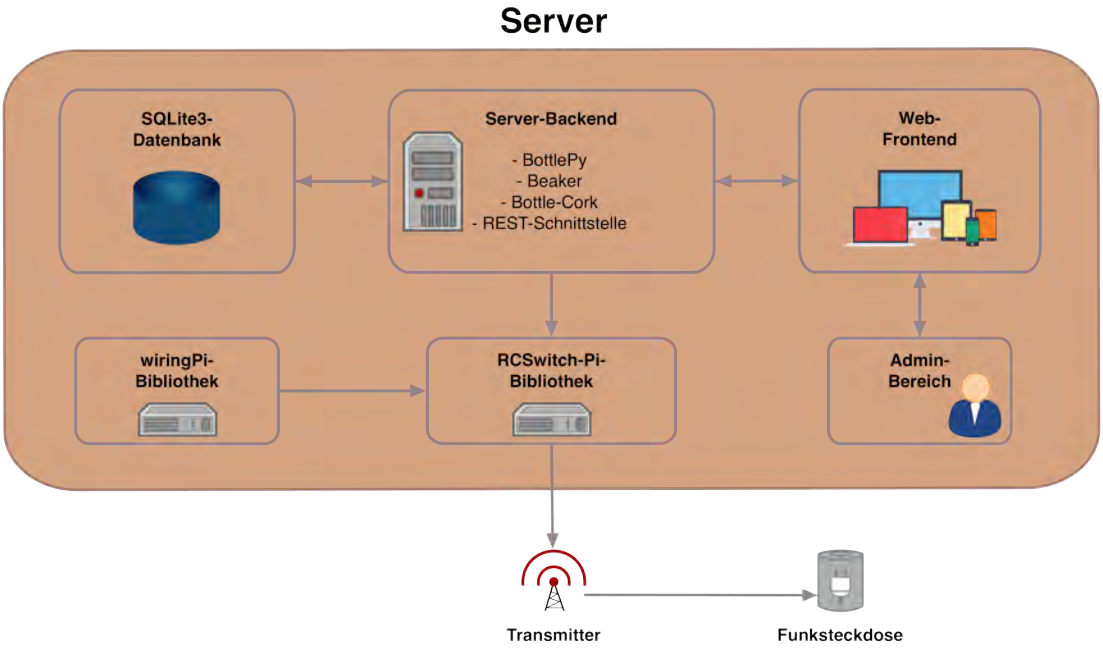

Abbildung 4.9: Architektur des Servers

4.3.1 Steuerung des Transmitters

Nachdem der Transmitter über die GPIO-Pins an den Raspberry Pi angeschlossen werden kann, wird jetzt eine Möglichkeit benötigt, den Transmitter software-seitig ansprechen zu können. Dafür werden zwei Bibliotheken benutzt, die es gemeinsam ermöglichen, den

Transmitter mit den benötigten Informationen zu versorgen und so mit den Steckdosen zu kommunizieren.

4.3.1.1 wiringPi

wiringPi ist eine C Bibliothek. Sie bietet eine kleine und einfach zu nutzende Schnittstelle, über die sich die GPIO-Pins steuern lassen. Das erspart die Arbeit, sich selbst um die korrekte Ansteuerung der Pins kümmern zu müssen. Allerdings hat wiringPi ein weiteres spezielles Pin-Mapping, das beachtet werden muss. Wie in Kapitel 4.2.3 beschrieben, ist die Datenleitung des Transmitters an Pin 11 angeschlossen, welcher GPIO-Pin 17 entspricht. GPIO-Pin 17 entspricht wiederum wiringPi-Pin 0. Über diesen Pin kann der Transmitter mithilfe der wiringPi-Bibliothek nun angesprochen werden.

Die folgende Abbildung zeigt die Tabelle mit den Pin-Mappings. Die relevante Zeile ist hervorgehoben.

Abbildung 4.10: Pin-Mapping der wiringPi Bibliothek [wir13]

4.3.1.2 RCSwitch-Pi

RCSwitch-Pi ist eine C++ Bibliothek. Sie generiert Signale, die über den 433 MHz-Transmitter an die Funktsteckdosen verschickt werden können. Sie implementiert außerdem die Bibliothek wiringPi, um deren Funktionalität zu nutzen. wiringPi wird, außer von

RCSwitch-PI, nicht benutzt. Damit RCSwitch-Pi den Transmitter ansprechen kann, muss im Code der zu benutzende wiringPi-Pin (in diesem Fall Pin 0) hinterlegt werden. Um Signale über den Transmitter an die Steckdosen senden zu können, benötigt RCSwitch-Pi die folgenden drei Parameter:

1. Den System-Code der anzusprechenden Steckdose in binärer Form, zum Beispiel 10001
2. Den Unit-Code der anzusprechenden Steckdose in dezimaler Form, zum Beispiel 3 (entspricht dem Code 00100 und damit der Steckdose C)
3. Den auszuführenden Befehl in binärer Form: 1 um die Steckdose anzuschalten, 0 um die Steckdose auszuschalten

Aus diesen drei Blöcken generiert RCSwitch-Pi das Signal und schickt es an den im Code definierten Wiring-Pi Pin. Der dort angeschlossene Transmitter übermittelt das Signal anschließend an die per System- und Unit-Code definierte Steckdose.

Nun hat man die Möglichkeit, per Kommandozeile Befehle an die Steckdosen zu senden. Ein solcher Befehl sieht zum Beispiel so aus:

```
1 sudo /home/pi/steckdosen/rcswitch-pi/./send 10001 3 1
```

Dieser Befehl würde Steckdose 3 (C) in System 10001 (17) einschalten (1).

4.3.2 Server-Backend

Nachdem eine Möglichkeit gefunden wurde, die Steckdosen mithilfe des Transmitters ansprechen zu können, wird nun ein Server-Backend benötigt, das einen Kommunikationskanal zwischen dem Client (der Android-Applikation) und dem Transmitter herstellt. Dieser Server muss über das Netzwerk von der Android-Applikation aus erreichbar sein und gleichzeitig in der Lage sein, den benötigten Kommandozeilenbefehl zum Senden des Signals auszuführen. Außerdem muss das Backend mit der Datenbank kommunizieren können.

Dieses Backend ist in der Programmiersprache Python geschrieben. Es besteht hauptsächlich aus der Datei server.py, in der die komplette Logik des Servers steckt, und der Datei serverconfig.json, in der die Serverkonfiguration gespeichert ist. Dort können die folgenden Parameter festgelegt werden:

- Der Port, auf dem der Server kommunizieren soll. Diese Angabe ist obligatorisch.
- Die DynDNS-Url. Diese Angabe ist optional. Sie wird benötigt, wenn der Service von außerhalb des eigenen lokalen Netzwerks (folglich aus dem Internet) genutzt werden soll.

Die IP-Adresse, unter der der Server erreichbar sein soll, wird beim Starten des Servers automatisch auf die aktuelle IP des Raspberry PI gesetzt. Hier ist deshalb keine manuelle Konfiguration notwendig.

4.3.2.1 BottlePy als Python-Microframework

BottlePy ist ein WSGI[3] Microframework für Python. Es stellt die grundlegenden Funktionen eines Web-Servers bereit, sodass diese nicht selbst entwickelt werden müssen.
Es implementiert eine REST-Schnittstelle, auf die später genauer eingegangen wird, und stellt außerdem ein Template-System bereit, mit dem statische HTML-Seiten dynamisch mit Inhalt gefüllt und zurückgesendet werden können.

Dieses Framework bildet die Grundlage des Servers. Da das Framework alleine bereits als autonomer Server funktioniert, muss lediglich die eigentliche Funktionalität implementiert werden.

4.3.2.2 Verwendete Bibliotheken

Zusätzlich zu BottlePy wurden zwei weitere Bibliotheken verwendet, um deren Funktionalität zu nutzen.

[3] Web Server Gateway Interface

Beaker Beaker ist eine Bibliothek, die sich um die Session-Verwaltung des Servers kümmert. Sie generiert, sendet und speichert Cookies. Das erlaubt es den Clients, eingeloggt zu bleiben.

Bottle-Cork Bottle-Cork liefert eine Benutzerverwaltung, die über ein Web-Interface (eine Webseite, die auf dem Server läuft) konfiguriert werden kann. Das integrierte Rollensystem erlaubt es, hierarchisch geordnete Rollen anzulegen und Benutzern bestimmte Rollen zuzuweisen. So können verschiedene Rechte für die verschiedenen Gruppen angelegt und normale Benutzerkonten von Administrator-Konten abgegrenzt werden. In diesem Projekt werden die Gruppen *user* und *admin* benutzt, wobei die Gruppe *admin* den höheren Stellenwert besitzt. Außerdem kann jeder Benutzer eigene Steckdosen-Gruppen verwalten.

Des Weiteren wird durch die Benutzerverwaltung verhindert, dass Fremde die Steckdosen steuern. Man muss sich zuerst als Benutzer einloggen, um die Funktionalität des Systems nutzen zu können.

Bottle-Cork stellt außerdem die Möglichkeit bereit, dass sich Nutzer direkt auf der Weboberfläche registrieren können. Sie müssen lediglich ihre Email-Adresse verifizieren. Dieses Feature wurde allerdings für diese Arbeit entfernt, da sich sonst jeder ungefiltert registrieren und die Steckdosen steuern könnte, der die URL kennt. Neue Benutzer anlegen kann nur der Administrator über das Web-Interface.
Die Benutzerpasswörter werden nicht im Klartext, sondern als Hash-Code verschlüsselt abgespeichert, so hat der Administrator auch keinen Einblick in die privaten Passwörter.

4.3.2.3 Datenverwaltung

Es wurde eine sqlite3-Datenbank [sql13] gewählt, um die Daten (bis auf die Benutzerverwaltung, die von Bottle-Cork übernommen wird) zu speichern. Es gibt 4 Tabellen, um die nötigen Datenstrukturen darzustellen:

- dips: Repräsentiert die Daten einer DIP-Gruppe. Das ist der System-Code, der in jeder Steckdose eingestellt werden muss.

- plugs: Stellt die Daten einer Steckdose dar.

- groups: Repräsentiert von Benutzern erstellte Steckdosen-Gruppen, die er so steuern kann. Eine Gruppe kann Steckdosen aus verschiedenen DIP-Gruppen beinhalten. Außerdem kann in jeder Gruppe ein Start- und ein Stop-Skript hinterlegt werden, das ausgeführt wird, wenn die Gruppe ein- oder ausgeschaltet wird.

- plugToGroup: Ist die Zwischentabelle zwischen plugs und groups. Diese ist nötig, da es sich bei den beiden Tabellen um eine m-zu-n-Beziehung handelt. Eine Steckdose kann zu mehreren Gruppen gehören und eine Gruppe kann mehrere Steckdosen beinhalten.

Die Tabellen sind mit verschiedenen Primär- und Fremdschlüsseln belegt. Das erlaubt es, Fremdschlüssel-Beschränkungen zu benutzen. In diesem Fall wurde die Beschränkung *ON DELETE CASCADE* benutzt. Diese sorgt dafür, dass beim Löschen eines Eintrags aus einer Tabelle, die einen Primärschlüssel beinhaltet, alle Einträge in allen anderen Tabellen gelöscht werden, die diesen Primärschlüssel referenzieren. Die Regel kommt zum Beispiel zum Einsatz, wenn eine komplette DIP-Gruppe gelöscht wird. In diesem Fall werden auch alle Steckdosen gelöscht, die sich innerhalb der DIP-Gruppe befinden. Durch das Löschen der Steckdosen wiederum werden die entsprechenden Einträge aus den Steckdosen-Gruppen entfernt.

So verhindert die Fremdschlüssel-Beschränkung die Anhäufung von Datenleichen in der Datenbank und sichert somit die referentielle Integrität der Datenbank.

Abbildung 4.11 zeigt die Tabellen und deren Abhängigkeiten.

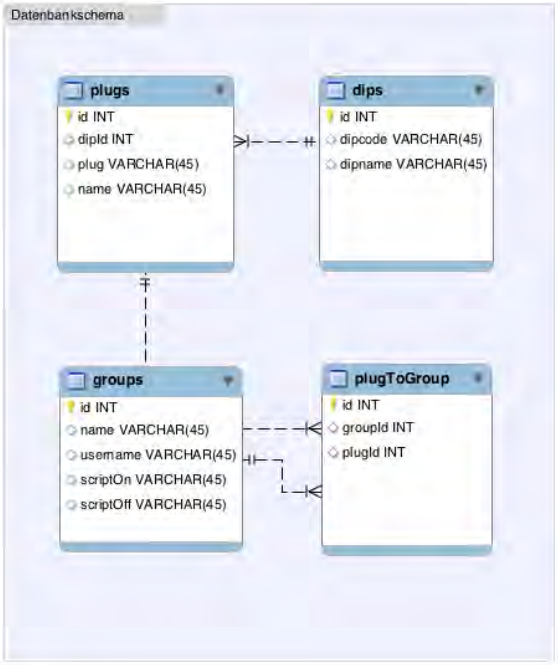

Abbildung 4.11: Datenbankschema des Servers

4.3.2.4 Web-Frontend

Der Server stellt außerdem ein Web-Frontend bereit. Dort hat man unter anderem die Möglichkeit, Steckdosen und Gruppen ein- und auszuschalten und sich allgemeine Informationen über den Server anzeigen lassen. Das Web-Frontend bietet eine funktionsärmere Alternative zur App. Hier können die Steckdosen vom Browser aus gesteuert werden, ohne dass ein Android-Gerät benötigt wird. Zusätzlich können Logfiles eingesehen werden, die der Server generiert. In den Logfiles werden alle Toggle-Aktionen mit Datum, Uhrzeit und Benutzer aufgelistet. Außerdem gibt es einen Admin-Bereich innerhalb des Frontends, der von Bottle-Cork vordefiniert und vorkonfiguriert ist. Dort kann der Administrator Benutzer und Rollen anlegen, bearbeiten und löschen. Dieser Bereich wurde um eine Funktion erweitert, mit der die Datenbank auf ihren initialen Zustand zurückgesetzt werden kann. Abbildung 4.12 zeigt die Homepage des Web-Frontends.

Abbildung 4.12: Web-Frontend

4.3.2.5 Kommunikation mit dem Server

REST-Schnittstelle In diesem Projekt wird die REST[4]-Schnittstelle als einfache Möglichkeit im Backend implementiert, einen Datenaustausch zwischen dem Server und den Clients zu ermöglichen.

REST ist folgendermaßen definiert:

> **Re**presentational **S**tate **T**ransfer (REST) ist ein Architekturstil mit dem Webservices realisiert werden können. Ein anderer ist SOAP. REST, das von Dr. Thomas Roy Fielding in seiner Dissertation beschrieben wurde, benutzt Prinzipien, die in großen, verteilten Anwendungen wie im World Wide Web (WWW) eingesetzt werden. Das REST-Architekturmodell beschreibt die Funktionsweise des WWW und dient als Referenz für zukünftige Erweiterungen und als Anleitung wie Web-Standards Web-gerecht eingesetzt werden können. Es abstrahiert die Architektur-Elemente in einem verteilten Hypermedia-

[4] Representational State Transfer

> System, konzentriert sich aber nicht auf Details wie einzelne Komponenten implementiert oder in welcher Syntax die Protokolle verfasst wurden, sondern es beschäftigt sich mit der Rolle und der Funktion der Komponenten. Ihre Beziehungen und Interaktionen untereinander werden durch ihre Uniform Resource Locator (URL) bestimmt. REST zeigt den fundamentalen Zusammenhang der Komponenten und Daten, die die Basis der Web-Architektur bilden und ihr Verhalten in netzwerkbasierten Anwendungen [RES13].

REST bietet verschiedene vordefinierte Methoden. Die wichtigsten beiden sind GET und POST. GET ist eine einfache Anfrage, die der Server anhand der aufgerufenen URL einfach beantworten kann. Bei der POST-Methode gibt es zusätzlich die Möglichkeit, Formulardaten mit der Anfrage mitzusenden. So kann man dem Server zum Beispiel Login-Daten mitteilen, damit sich der Benutzer einloggen kann.

Im Folgenden werden exemplarisch einige Funktionen und deren HTTP-Methoden aufgelistet. Für sämtliche Methoden wird mindestens die Rolle *user* vorausgesetzt.

- GET /setup: Führt das initiale Setup des Servers durch. Das Setup besteht darin, die Datenbank mit deren Tabellen anzulegen, da diese von Haus aus nicht existieren. Die Methode wurde mit in die REST-Schnittstelle des Servers aufgenommen um die Inbetriebnahme des Servers zu erleichtern. Andernfalls müssten die Datenbank mit deren Tabellen manuell vom Administrator angelegt werden. Berechtigungen, diese Methode aufzurufen, besitzt nur der Administrator.

- GET /info: Liefert allgemeine Informationen zum Server, inklusive der aktuellen Serverkonfiguration.

- POST /plugs: Eine neue Steckdose wird angelegt oder eine bestehende Steckdose wird geändert. Es werden die Daten der Steckdose als Formulardatensatz erwartet. (Analog: /dips, /groups)

- DELETE /dips: Eine DIP-Gruppe wird gelöscht. Es wird die id der DIP-Gruppe als Formulardatensatz erwartet. (Analog: /plugs, /groups)

- GET /content: Liefert den kompletten benutzerspezifischen Datensatz aus der Datenbank als JSON-String zurück. Dazu gehören alle DIP-Gruppen, alle Steckdosen und alle vom Benutzer erstellten Gruppen.

- POST /togglePlug: Schaltet eine Steckdose ein oder aus. Es werden die id der Steckdose und der Auszuführende Befehl (ein/aus) als Formulardatensatz erwartet.

- POST /toggleGroup: Schaltet alle Steckdosen einer Gruppe ein oder aus. Es werden die id der Gruppe und der Auszuführende Befehl (ein/aus) als Formulardatensatz erwartet.

- GET /allOff: Schaltet sämtliche im System registrierten Steckdosen aus.

- GET /: Liefert die Index-Seite des Web-Frontends zurück. Leitet auf die login-Seite weiter wenn der Benutzer nicht eingeloggt ist.

- GET /logs: Liefert alle Logeinträge zurück. Logeinträge werden beim Ein-und Ausschalten von Steckdosen und Gruppen geschrieben.

Die Methoden GET /webTogglePlug und GET /webToggleGroup sind für das Web-Frontend gedacht. Sie verhalten sich analog zu ihren verwandten Methoden POST /togglePlug und POST /toggleGroup, allerdings ist die Parameterübergabe eine andere. Hier werden die erforderlichen Daten an die URL hinten angehängt, bei den anderen Methoden werden sie als Form-Daten in dem POST-Request übergeben. Das ist notwendig, da der Server keine POST-Daten an sich selbst schicken kann. Außerdem unterscheiden sie sich in der HTTP-Antwort des Servers. Wird die Aktion vom Web-Frontend aus ausgeführt, wird ein redirect auf die Homepage des Web-Frontends zurückgeliefert. Wird die Aktion hingegen von der App aus ausgeführt, wird nur ein *success* oder im Falle eines Fehlers die entsprechende Exception zurückgeliefert. Das spart Ressourcen, da so in jedem Fall nur relevante und keine überflüssigen Daten übertragen werden (es wird nicht die Homepage an die App übertragen, die mit den Daten nichts anfangen könnte). In den Abbildungen 4.13 und 4.14 sind die beiden Kommunikationswege dargestellt.

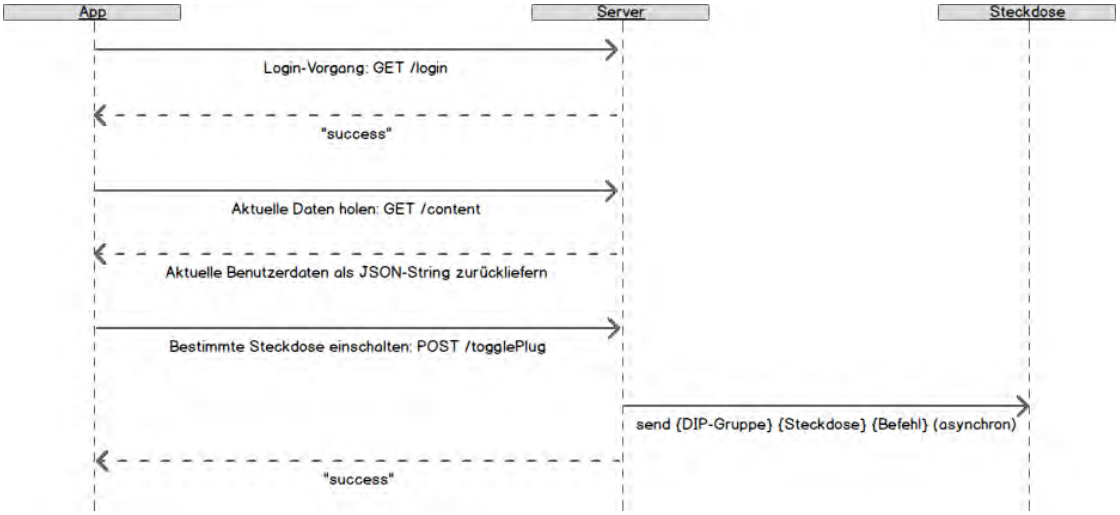

Abbildung 4.13: Sequenzdiagramm der Kommunikation zwischen App, Server und Steckdose

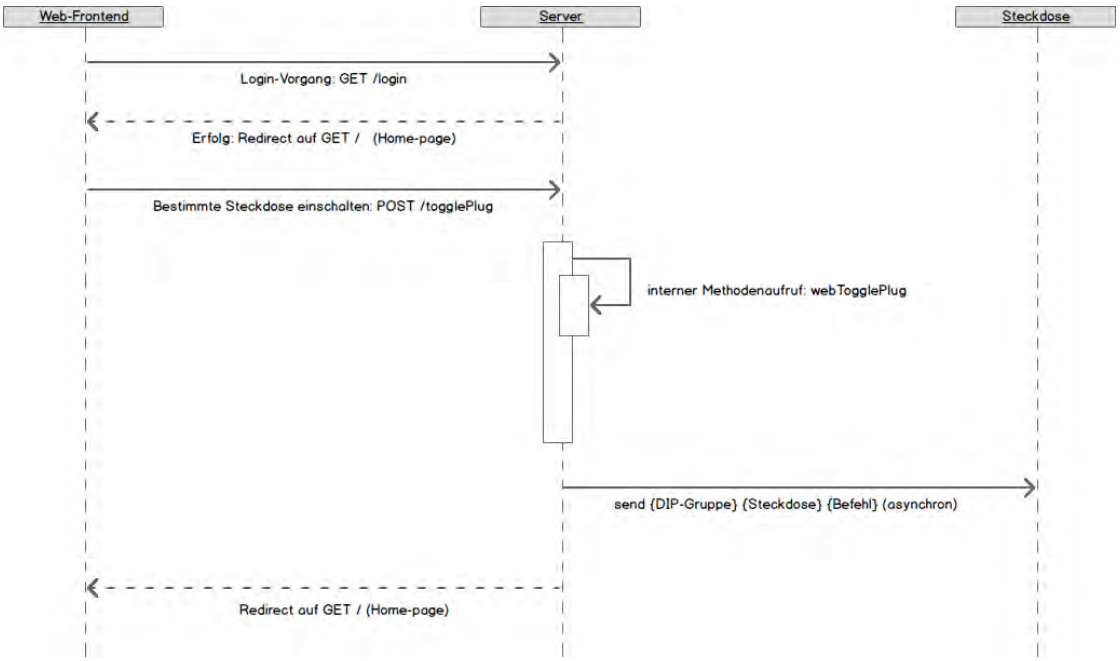

Abbildung 4.14: Sequenzdiagramm der Kommunikation zwischen Web-Frontend, Server und Steckdose

Zusätzlich zu den oben aufgelisteten Methoden gibt es noch einige Methoden, die von BottlePy und Bottle-Cork bereits vordefiniert sind.

4.4 Mobile Anwendung

Nachdem die Serverkomponente implementiert und eingerichtet wurde, fehlt noch ein Client. Hier wurde eine Android-Applikation entwickelt, die mit dem Server kommunizieren kann. Diese wurde aufgrund der besseren Performance und der größeren Unterstützung von Sensoren nativ entwickelt und nicht dem aktuellen Trend entsprechend als hybride, plattformübergreifende Applikation [SSP+13].

Sie ist die mobile Komponente des Systems, die es dem Benutzer erlaubt, die Steckdosen von überall aus per Android-Gerät zu steuern. Dieses Kapitel beschreibt die Entwicklung der Applikation.

4.4.1 Android SDK

Das Android Software Development Kit (SDK) stellt grundlegende Bibliotheken bereit, die benötigt werden, um eine konsistente Android-Applikationen zu entwickeln. Die SDK teilt sich in verschiedene Bereiche auf. Zum einen gibt es die *Android SDK Tools*, diese beinhalten allgemeine Entwicklungs- und Debugging Tools. Zum anderen beinhaltet sie revisionierte APIs (Application Programming Interfaces), die auf die verschiedenen Android-Versionen abbilden. Für dieses Projekt wurde API Level 17 verwendet, die auf die Android-Version 4.2.2 Jelly-Bean abbildet. Als minimale API wurde 16 (Android 4.1.2) ausgewählt. Dies hat zur Folge, dass Android-Geräte mit älteren Versionen die Applikation nicht verwenden können.
Außerdem gibt es noch weitere Bibliotheken, die je nach Bedarf eingebunden werden können. Für dieses Projekt wurde zusätzlich die Bibliothek *Google Play services* verwendet, die unter Anderem die Funktionalität der *Geofences* bereitstellt, die in diesem Projekt verwendet wurde.

4.4.2 Layout-Design

Um die Applikation optisch ansprechend zu gestalten, wurde ein Custom Theme erstellt. Dieses basiert auf dem Standard-Theme von Android 4.0+ (*Holo Dark*), sodass das Design konsistent zum Rest des Betriebssystems ist. Die Farbwahl wurde aufgrund der Inspiration von *Orangium*, einem Custom Theme für den Apex Launcher[5], getroffen. Die nötigen Designs wurden mithilfe der tools *ActionBarGenerator* [Act13] und *Android Holo Colors Generator* [Hol13] erstellt. Die in der Applikation verwendeten Icons sind aus dem kostenlosen Icon-Set *Android General* von iconShock [Ico13].

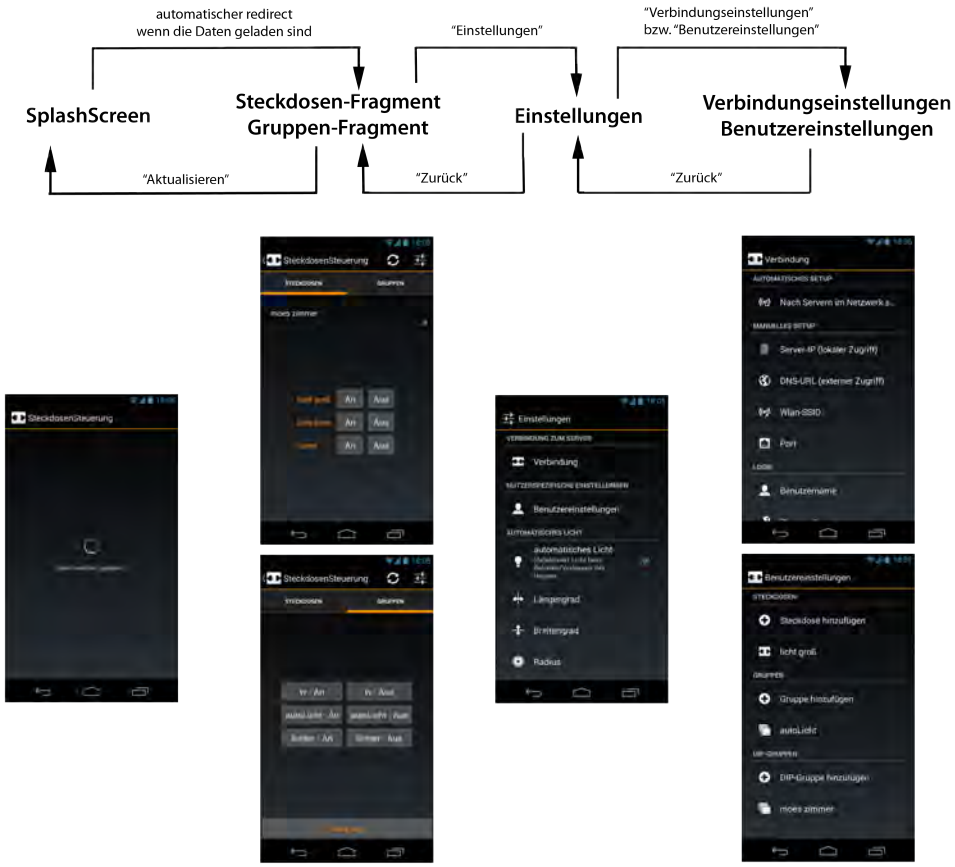

Abbildung 4.15: Storyboard der Applikation

[5]eine alternative Oberfläche für Android 4.0+

Abbildung 4.15 zeigt das Storyboard der Applikation. Es wurden bereits einige Benutzerdaten angelegt, damit die Oberfläche der Applikation im aktiven Zustand gezeigt werden kann.

Abbildung 4.16 zeigt ein Zustandsdiagramm der Applikation als Petrinetz.

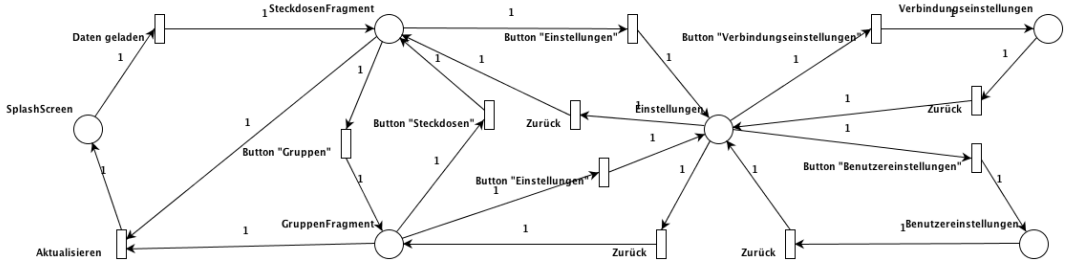

Abbildung 4.16: Zustandsdiagramm der Applikation als Petrinetz

4.4.3 Ladebildschirm: SplashScreenActivity

Die Activity stellt den Ladebildschirm dar. Während er angezeigt wird, wird ein Hintergrundprozess ausgeführt, der den in den Verbindungseinstellungen hinterlegten Benutzer beim Server einloggt und die aktuellen Daten herunterlädt. Ist dieser Prozess beendet, wird der Hauptbildschirm geladen.
Der Hintergrundprozess wird außerdem automatisch ausgeführt sobald sich die Verbindungsart zum Internet ändert, zum Beispiel von WLAN zum Mobilfunknetz oder umgekehrt. Das stellt sicher, dass beim nächsten Befehl an den Server die richtige Verbindungsart gewählt wird. Wenn man sich im lokalen Netzwerk befindet, wird die IP als Host verwendet, wenn man sich außerhalb des lokalen Netzwerks befindet, wird die DynDNS-URL als Host verwendet. Innerhalb des lokalen Netzes ist die Latenz meist deutlich geringer. Ist keine DynDNS-URL hinterlegt, kann die Applikation nur im lokalen Netzwerk verwendet werden.

4.4.4 Hauptbildschirm: MainActivity

Dies ist der Hauptbildschirm. Er besitzt am oberen Rand eine Tab-Leiste, mit der zwischen zwei *Fragments* gewechselt werden kann. Fragments sind Layout-Fragmente, die in einen Container eingebettet werden und dynamisch ausgetauscht werden können. Je nachdem welcher Tab in der Tab-Leiste ausgewählt ist, wird das entsprechende Fragment im Container darunter angezeigt. Die MainActivity hat keine weiteren Aufgaben außer der Bereitstellung der Fragments im Container und der Tabs für deren Auswahl.

4.4.4.1 Steckdosen-Fragment

Dieses Fragment dient dazu, einzelne Steckdosen an- und auszuschalten. Oben befindet sich ein *Spinner*, mit dem man die gewünschte DIP-Gruppe auswählen kann. Ein Spinner ist eine Dropdown-Liste, aus der man einen Eintrag auswählen kann. Darunter werden die Steckdosen der aktuell ausgewählten DIP-Gruppe angezeigt, mit jeweils einem An- und einem Aus-Button. Der DIP-Spinner bekommt keinen erklärenden Begleittext, da der Name der angezeigten DIP-Gruppe (z.B. Wohnzimmer) selbsterklärend ist.

4.4.4.2 Gruppen-Fragment

Das Gruppen-Fragment zeigt alle vom eingeloggten Benutzer angelegten Gruppen mit entsprechenden Bedienelementen an. Die Gruppen sind nicht die DIP-Gruppen, die in den Steckdosen kodiert werden, sondern allgemeine Gruppen, in denen der Benutzer verschiedene Steckdosen aus verschiedenen DIP-Gruppen zusammenfassen kann. Außerdem wird, unabhängig von den angezeigten Gruppen, unten ein *alles aus*-Button angezeigt, mit dem sämtliche registrierten Steckdosen auf einmal ausgeschaltet werden können. Nachfolgend ist ein Design-Entwurf der beiden Fragments, eingebettet in der MainActivity, zu sehen. Anschließend werden die im Design-Entwurf eingezeichneten Callouts erklärt.

Abbildung 4.17: Design-Entwurf des Steckdosen-Fragments (links) und des Gruppen-Fragments (rechts)

Im Folgenden werden die numerierten Elemente beschrieben:

1. Der *Aktualisieren*-Button ruft die SplashScreenActivity auf.
2. Der *Einstellungen*-Button ruft die Einstellungen auf.
3. Mit diesen beiden Tabs kann das aktive Fragment gewechselt werden.
4. Das ist ein DIP-Spinner, um die DIP-Gruppe auszuwählen.
5. Mit diesen Buttons können die entsprechenden Steckdosen-Gruppen ein- und ausgeschaltet werden.
6. Mit diesen Buttons können einzelne Steckdosen der bei 4. ausgewählten DIP-Gruppe ein- und ausgeschaltet werden.
7. Mit dem *Alles Aus*-Button können alle im System registrierten Steckdosen ausgeschaltet werden.

Die Einträge des DIP-Spinners sowie die Buttons zur Steuerung der Steckdosen und Steckdosen-Gruppen werden jeweils dynamisch zur Laufzeit generiert und angezeigt.

4.4.5 Einstellungen

Hier können sämtliche Einstellungen der Applikation vorgenommen werden. Außerdem können die Benutzerdaten verwaltet werden. Aus Gründen der Übersichtlichkeit wurden die Verbindungseinstellungen und die Benutzereinstellungen jeweils in einen separaten Bereich ausgelagert.

4.4.5.1 Verbindungseinstellungen

Hier kann die Verbindung zum Server konfiguriert werden.

Die Einstellungsmöglichkeiten sind:

- Die IP-Adresse des Servers
- Die DynDNS-URL des Servers (optional)
- Die Wlan-SSID des Netzwerks. Sie wird benötigt um zu überprüfen, ob sich das Gerät in dem selben Netzwerk wie der Server befindet. In diesem Fall kann die App über das lokale Netzwerk mit dem Server kommunizieren und nicht über das Internet. Dies verringert die Latenz deutlich. (optional)
- Der Port des Servers
- Der Benutzername und das Passwort, damit sich die App beim Server einloggen kann

Automatische Client-Konfiguration Zusätzlich zur manuellen Konfiguration besteht die Möglichkeit, dass sich die Applikation selbst konfiguriert. Diese Funktion kann nur bei einer bestehenden WLAN-Verbindung genutzt werden. Sie durchsucht das Subnetz, in dem sich das mobile Gerät befindet, auf Port 80 nach Servern (dieser Port ist der Standard-Port für HTML-Webserver). Hierfür wird ein GET-Request an die URL */info* gesendet und die Antwort überprüft. Wird die Antwort als die Info-Seite des Servers identifiziert, weiß die Applikation, dass sie einen Server gefunden hat. Ist der Scan-Vorgang beendet, kann der Benutzer unter allen gefundenen den gewünschten Server

aus einer Liste auswählen und die Applikation konfiguriert sich automatisch mit dem entsprechenden Datensatz.

In den meisten Fällen wird das Subnetz 24 Bit lang sein. Das bedeutet es werden 256 IP-Adressen gescannt. Dies benötigt ca. 30 Sekunden.

Abbildung 4.18 zeigt einen Screenshot der Verbindungseinstellungen.

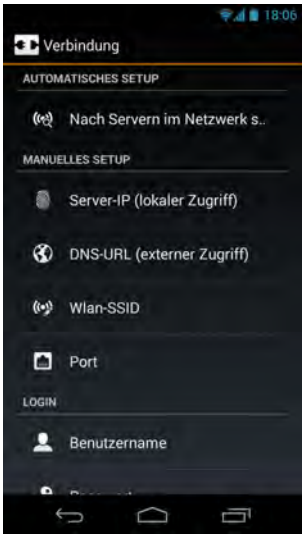

Abbildung 4.18: Screenshot der Verbindungseinstellugnen

4.4.5.2 Benutzereinstellungen

Hier kann der Benutzer DIP-Gruppen, Steckdosen und eigene Gruppen anlegen, ändern und löschen. Dieser Bereich der Einstellungen kann nur aufgerufen werden, wenn die Applikation sich erfolgreich beim Server eingeloggt hat, da die Daten benutzerspezifisch sind.
Der Bereich ist mit verschiedenen Constraints belegt.

- Es kann keine Steckdose angelegt werden, die bereits existiert. Anstatt in diesem Fall eine Fehlermeldung auszugeben, wird dem Benutzer erst gar nicht die Möglichkeit gegeben, eine Steckdose anzulegen, die bereits existiert. Die entsprechenden Kombinationen von DIP-Gruppen und Steckdosennummern werden nicht aufgeführt.
- Beim Anlegen von DIP-Gruppen werden nur nicht-genutzte DIP-Codes angeboten.
- Beim Anlegen von Steckdosen können nur DIP-Gruppen ausgewählt werden, die noch Kapazitäten für weitere Steckdosen besitzen.
- Wird eine DIP-Gruppe gelöscht, werden auch alle darin vorhandenen Steckdosen und damit auch die gelöschten Steckdosen aus den Gruppen gelöscht.
- Wird eine DIP-Gruppe geändert, werden die darin vorhandenen Steckdosen auf die neue DIP-Gruppe portiert.

Diese Constraints lassen ein mögliches Fehlverhalten des Benutzers nicht zu und sind konform zur referentiellen Integrität der Datenbank auf dem Server.

Abbildung 4.19 zeigt einen Screenshot der Benutzereinstellungen.

Abbildung 4.19: Screenshot der Benutzereinstellugnen

4.4.5.3 Automatisches Licht (Geofences)

Dieses Feature ist dazu gedacht, Steckdosen, an denen z.B. Lampen angeschlossen sind, automatisch einzuschalten (auszuschalten), wenn der Benutzer nachhause kommt (das Haus verlässt). Es waren mehrere Ansätze nötig, um letztendlich eine zuverlässig funktionierende Lösung zu finden.

Erster Ansatz Für diesen Ansatz werden *BroadcastReceiver* benötigt. Ein BroadcastReceiver ist ein Service von Android. Wenn Applikationen Aktionen auslösen, wird die Nachricht, dass die Aktion ausgelöst wurde, systemweit per Broadcast verteilt. BroadcastReceiver empfangen diese Nachrichten und können so auf die Aktionen reagieren.

Der erste Gedanke war, in der Applikation eine WLAN-SSID zu hinterlegen und mittels eines BroadcastReceivers auf die Benachrichtigung einer Änderungen der Verbindungsart zu warten. Verbindet sich das Gerät mit dem angegebenen WLAN-Netz, wird die angegebene Gruppe eingeschaltet. Verliert das Gerät die Verbindung, wechselt in ein anderes Netz oder wechselt von WLAN zu GSM, wird die Gruppe ausgeschaltet.
Diese Methode ist leider nicht sehr zuverlässig, da der BroadcastReceiver auch manchmal im Ruhezustand benachrichtigt wird (Hintergrund: das Gerät schaltet die WLAN-Funktion im Standby ab um Energie zu sparen). Dies führt dazu, dass plötzlich eine Aktion ausgeführt wird, obwohl sich an der aktuellen Situation nichts verändert hat. Außerdem wird die Aktion natürlich auch dann ausgelöst, wenn man von Hand das WLAN an- oder abschaltet. Dieses Verhalten ist inakzeptabel.

Zweiter Ansatz Der nächste Versuch war, einen NFC-Tag an die Haustüre zu kleben, der die Aktion auslöst, wenn man an der Tür ist. NFC-Tags können von NFC-fähigen Geräten erkannt werden, wenn diese nahe an den NFC-Tag kommen. Auf diesen Tags können Informationen gespeichert werden, die das Gerät auslesen kann. So könnte eine Information hinterlegt werden, die die Applikation veranlasst eine Gruppe ein- oder auszuschalten.
Dabei existieren zwei Probleme. Erstens müsste man sich sehr nahe an die Tür stellen,

da NFC in der Praxis nur wenige Zentimeter an Reichweite besitzt und zweitens funktioniert die NFC-Erkennung des Smartphones nur bei eingeschaltetem Bildschirm (Dies ist kein Bug, sondern ein Sicherheitsfeature von Android). Also müsste man zuerst das Smartphone hervorholen, es aufwecken, entsperren und dann an den NFC-Tag halten. Dies ist nicht im Sinne einer Automation und ist daher ebenfalls inakzeptabel.

Dritter Ansatz Nicht die Applikation reagiert auf seine eigene geografische Veränderung, sondern der Server überprüft regelmäßig ob sich der Client in der Nähe befindet. Die Applikation registriert sich mit seiner MAC-Adresse oder IP als Teilnehmer des Features beim Server und dieser schaut ob der Teilnehmer das Netzwerk betritt. Wenn das der Fall ist, sendet er das entsprechende Signal an die Steckdosen. Der Vorteil gegenüber dem ersten Ansatz wäre, dass das Handy ein passiver Teilnehmer ist und das Feature so keinen Strom des Smartphones verbraucht. Der Nachteil bleibt leider derselbe, denn wenn man von Hand das WLAN aktiviert/deaktiviert oder das Smartphone in den Standby-Modus geht, wird die Aktion ausgeführt. Damit scheidet auch dieser Ansatz aus.

Vierter Ansatz Der finale Ansatz lautet *Geofences*. Google hat auf seiner *Google I/O 2013* vom 15.5.2013 neue Location APIs vorgestellt, unter anderem Geofences.
Diese definieren einen geographischen Punkt, der aus einem Längen- und einem Breitengrad besteht sowie einen Radius um diesen Punkt. Damit ist ein geographischer Bereich eingegrenzt. Betritt oder verlässt das Smartphone diesen Bereich, verteilt Android systemweit eine entsprechende Nachricht per Broadcast. Die Applikation ist mittels eines BroadcastReceivers in der Lage die Nachricht zu empfangen und zu interpretieren. So kann sie auf das aufgetretene Ereignis reagieren.

Dieser Ansatz funktioniert zuverlässig. Das einzig Wichtige ist, den Radius geschickt zu wählen, da die Ortung des Smartphones neben GPS auch durch das Mobilfunknetz erfolgt.
Die Ortung über das Mobilfunknetz ist sehr viel ungenauer als die Ortung über WLAN. Wird der Radius zu gering gewählt, kann dies bei ausgeschaltetem WLAN dazu führen,

dass sich das Gerät außerhalb des Geofences befindet, obwohl es in der Wohnung ist. So werden beim Aktivieren und Deaktivieren des WLANs die Aktionen ausgeführt, obwohl sich das Android-Gerät die komplette Zeit über in der Wohnung befand (selbes Problem wie bei Ansatz 1 und 3). Wird der Radius hingegen zu groß gewählt, wird die Aktion auch ausgeführt, wenn man sich nur in der Nähe des Hauses aufhält ohne das Haus betreten oder verlassen zu wollen. Der Radius muss also so gewählt werden, dass die Ortung über das Mobilfunknetz immer noch innerhalb des Geofence liegt, wenn man sich im Haus befindet, jedoch nicht wesentlich größer. Außerdem unterliegt die Berechnung des optimalen Radius weiteren bestimmten Gegebenheiten und Voraussetzungen.

- Bei der Berechnung des optimalen Radius wird der Fall des Betretens des Geofence betrachtet, nicht der Fall des Verlassens. Dies hat den Hintergrund, dass es beim Verlassen keine Rolle spielt, ob das Licht etwas früher oder später ausgeschaltet wird. Beim Betreten des Geofence ist jedoch darauf zu achten, dass das Licht eingeschaltet wird, *bevor* die Person das Haus betritt.
- Der aktuelle Standort wird in einem Intervall zwischen zwei und fünf Minuten überprüft. Da die Zuverlässigkeit des Systems und nicht die Energieeffizienz der angeschlossenen Geräte oberste Priorität hat, muss das Intervall der Standortüberprüfung bei der Berechnung mit dem worst case angenommen werden (5 Minuten oder 300 Sekunden).
- Die angenommene durchschnittliche Schrittgeschwindigkeit liegt bei einem Meter pro Sekunde.

Es muss sichergestellt werden, dass das Überprüfungsintervall innerhalb der Zeitspanne liegt, die der Benutzer benötigt, um vom Rand des Geofence zu seiner Wohnung zu gelangen (die Entfernung entspricht dem Radius des Geofence), da es andernfalls vorkommen kann, dass der Benutzer sein Haus betritt bevor das Android-Gerät den Eintritt in den Geofence registriert. Bei einem Intervall von 300 Sekunden und einer Laufgeschwindigkeit von einem Meter pro Sekunde ergibt sich ein Radius von 300 Metern. Dieser Radius ist außerdem groß genug, um das Problem mit der ungenauen Standortbestimmung über das Mobilfunknetz zu umgehen. Er hat sich während den Praxistests des Features bewährt. Allerdings ist die hier angenommene Schrittgeschwindigkeit nur

ein Durchschnittswert. Die tatsächliche Schrittgeschwindigkeit hängt von der jeweiligen Person ab. Um trotzdem eine optimale Funktionsweise des Features zu garantieren, kann der Benutzer den Faktor der Schrittgeschwindigkeit durch eine Feinjustierung des Radius ausgleichen. 300 Meter dienen als Richtwert.

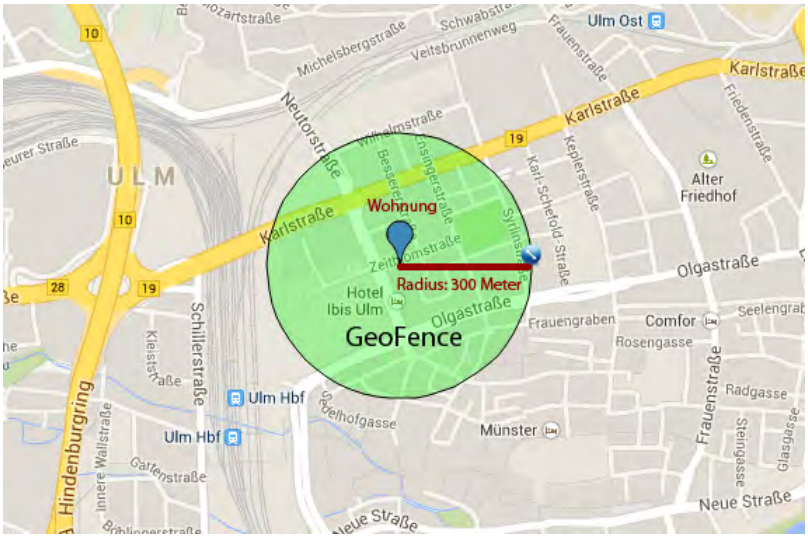

Abbildung 4.20: Visualisierter Geofence, auf *Google Maps*-Karte eingezeichnet

Das Feature *Geofences* gehört zu den *Location API*s, die wiederum zur *Google Play services* Bibliothek gehören. Da diese Bibliothek nicht auf jedem Android-Gerät in der erforderlichen Version installiert ist, wird deren Vorhandensein beim Aktivieren des Features überprüft. Kann die Bibliothek nicht gefunden werden, kann auch das Feature nicht aktiviert werden.
Des Weiteren benötigt die Funktion *Standortzugriff*, um korrekt funktionieren zu können. Dieser muss in den Android-Einstellungen aktiviert sein. Ist das nicht der Fall, wird der Benutzer der Applikation beim Aktivieren der Funktion auf die entsprechende Einstellungsseite weitergeleitet und erhält die Meldung, zuerst die Standorteinstellungen anpassen zu müssen bevor die Funktion genutzt werden kann.

Für den Fall eines eintretenden Ereignisses (dem Betreten oder Verlassen des Geofence) hat der Benutzer verschiedene Optionen. Es kann eine Gruppe festgelegt werden, die beim Betreten/Verlassen des Geofence ein- und ausgeschaltet werden soll. Zusätzlich gibt es die Möglichkeit beim Verlassen des Hauses nicht nur die Steckdosen in der festgelegten Gruppe, sondern alle beim Server registrierten Steckdosen auszuschalten. Außerdem kann die Zeitspanne festgelegt werden, in der der Geofence für das Betreten des Hauses aktiv sein soll. So lässt es sich vermeiden, dass die Lichter eingeschaltet werden, wenn es hell ist.

Abbildung 4.21 zeigt einen Screenshot der Einstellungsmöglichkeiten für das automatische Licht.

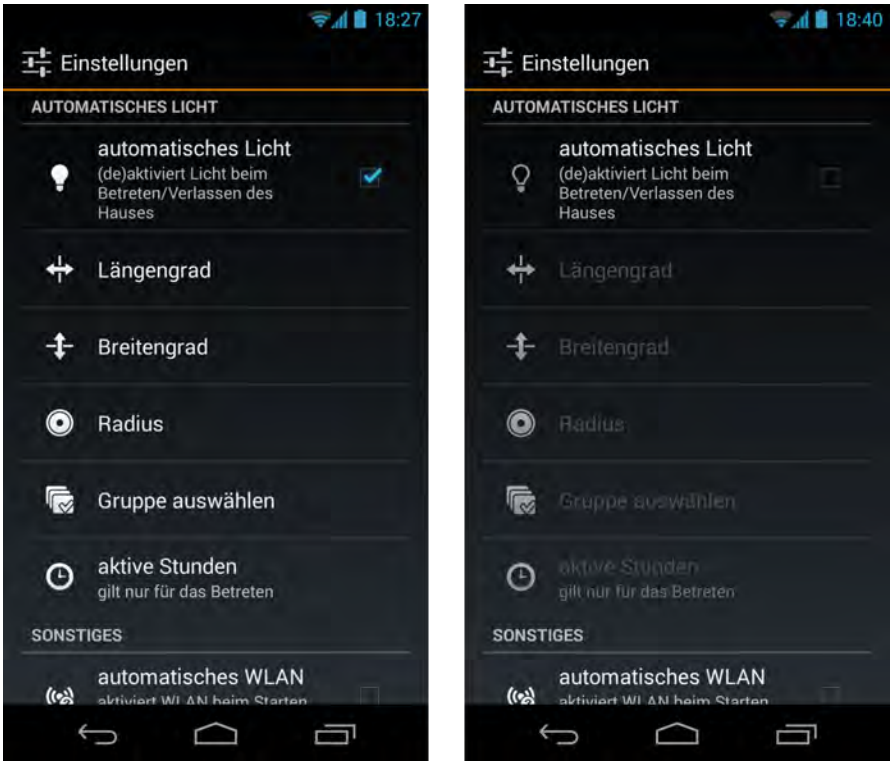

Abbildung 4.21: Screenshot der Einstellungen für das automatische Licht. Links ist das Feature eingeschaltet, rechts ist es ausgeschaltet.

4.4.5.4 Automatisches WLAN

Wenn diese Funktion aktiviert ist, schaltet sich das WLAN automatisch ein sobald die Applikation gestartet wird. Dies ist vor allem dann von Vorteil, wenn das Gerät keinen Zugriff auf das mobile Internet hat. In diesem Fall kann die Applikation nur genutzt werden, wenn das Gerät mit dem selben Netzwerk verbunden ist, in dem auch der Server ist. Dazu ist WLAN notwendig. Allerdings wird der Startvorgang der Applikation dadurch verlangsamt, da sich das Gerät zuerst mit dem WLAN-Netz verbinden muss bevor sich die Applikation beim Server einloggen und die aktuellen Daten empfangen kann.

5

Anforderungsabgleich

In diesem Kapitel werden die in der Anforderungsanalyse erarbeiteten Anforderungen auf deren Umsetzung überprüft. Die nachfolgende Tabelle zeigt eine Übersicht der Anforderungen und die Bewertung deren Umsetzung nach folgendem Schlüssel:

+ Die Anforderung wurde vollständig erfüllt.

o Die Anforderung konnte nur teilweise erfüllt werden.

- Die Anforderung konnte nicht erfüllt werden.

Anforderung	Umsetzung	Kommentar
Das System soll Steckdosen per Android-App steuern können.	+	Das entspricht der Grundfunktionalität des Projekts und wurde deshalb mit oberster Priorität entwickelt.
Das System soll jederzeit verfügbar sein.	+	Solange der Client und der Server eine Internetverbindung haben und der Server läuft, ist das System verfügbar.
Das System soll Benutzer und Benutzer-Gruppen verwalten können.	+	Die Benutzerverwaltung wurde mithilfe der Bibliothek *Bottle-Cork* umgesetzt. Der Administrator kann die Benutzerverwaltung über das Web-Frontend aufrufen.
Alle nutzerspezifischen Daten sollen auf dem Server gespeichert werden.	+	Alle Daten liegen in der Datenbank auf dem Server. Auf sie kann nur zugegriffen werden, wenn der Benutzer mit dem richtigen Account eingeloggt ist.
Benutzer sollen nutzerspezifische Daten verwalten können.	+	Dies kann der Benutzer in den Benutzereinstellungen der Android-Applikation tun.
Bei Gruppen soll ein Start- und Stop-Skript hinterlegt werden können	+	Dies kann der Benutzer ebenfalls bei den Benutzereinstellungen der Android-Applikation tun, wenn er eine neue Steckdosen-Gruppe anlegt oder bearbeitet.
Die Android-Applikation soll sich automatisch konfigurieren können.	+	Die automatische Konfiguration kann unter den Verbindungseinstellungen der Android-Applikation ausgeführt werden.
Die Android-Applikation soll automatisch die beste Verbindungsart zum Server wählen.	+	Das macht die App automatisch im Hintergrund. Wenn sich etwas an der Internetverbindung des Android-Gerätes ändert, reagiert die App automatisch darauf und konfiguriert sich neu.

Anforderung	Umsetzung	Kommentar
Die Android-Applikation soll keine falsche Bedienung zulassen.	+	Constraints verhindern die falsche Bedienung: es werden nur gültige Datenkombinationen angezeigt.
Die Android-Applikation soll beim Start nach spätestens fünf Sekunden nutzbar sein.	o	Die Zeit, die verstreicht bis die App benutzbar ist, hängt maßgeblich von der Zeit ab, die die Kommunikation des Clients mit dem Server benötigt. In den meisten Fällen beträgt diese weniger als zwei Sekunden. In Einzelfällen ist es jedoch möglich, dass die Kommunikation mehr Zeit in Anspruch nimmt, zum Beispiel wenn der Client über das mobile Internet mit dem Server kommuniziert und die Abdeckung des Mobilfunknetzes am Standort des Clients schlecht ist. Somit kann keine maximale Verzögerung der Benutzbarkeit von fünf Sekunden garantiert werden, allerdings liegt die Zeit in den meisten Fällen deutlich darunter.
Der Server soll ein Web-Frontend bereitstellen.	+	Das Frontend kann ganz einfach über das Internet im Browser aufgerufen und benutzt werden.
Der Server soll auf Veränderungen des Standorts des Benutzers reagieren können.	+	Dieses Feature wurde mithilfe von *GeoFences* umgesetzt.

6

Zusammenfassung

Das Ziel der Arbeit war es, ein Home Automation System von Grund auf selbst zu entwickeln. Wie gezeigt wurde ist dies ein komplexes Vorhaben. Das System besteht aus mehreren Software-Modulen und verschiedener Hardware.

Die Software wurde in verschiedenen Programmiersprachen implementiert. Die Android-Applikation wurde in Java und der Server in Python entwickelt. Außerdem wurde die Abfragesprache SQL für die Kommunikation mit der Datenbank verwendet. Um den Transmitter mit dem Raspberry Pi zu verbinden, musste noch ein Adapter gelötet werden, um eine stabile Verbindung zu gewährleisten. Die Grundfunktionalität, Steckdosen per Android-Applikation ein- und ausschalten zu können wurde durch mächtige Funktionen erweitert. Es lassen sich verschiedene Steckdosen gemeinsam ein- und ausschalten, in dem sie zu einer Gruppe zusammengefasst werden. So können die verschiedensten Geräte miteinander kombiniert werden. Diesen Gruppen können Skripte zugewiesen

werden, die beim ein- oder ausschalten der Gruppe auf dem Raspberry Pi ausgeführt werden. Die Schnittstelle zu den Skripten ist sehr mächtig, da sie plattformunabhängig ist und in den Skripten weiterer Code ausgeführt werden kann. Damit ist das System beliebig erweiterbar. So kann das entwickelte System gleichzeitig als abgeschlossenes System oder als Basis-System für weitere Automationen genutzt werden.

6.1 Abschließende Bemerkungen

Dieses Projekt ist eine einfache Umsetzung eines Home Automation Systems. Trotzdem ist die Installation des Systems nicht ganz trivial. Um den Server korrekt aufzusetzen, müssen einige Bibliotheken kompiliert und installiert werden. Für die Kommunikation mit dem Transmitter ist dies ebenfalls notwendig. Hierfür müssen Grundkenntnisse des Systems Linux und dessen Bedienung per Kommandozeile vorliegen. Außerdem muss die Infrastruktur des Netzwerks angepasst werden, dass der Server aus dem Internet erreichbar ist. Hierfür werden Kenntnisse in Rechnernetzwerken vorausgesetzt. Da dies alles von einem Laien kaum zu bewältigen ist, ist das Projekt im aktuellen Zustand nicht marktreif.

Eine Möglichkeit das zu ändern wäre, das System-Image des Raspberry Pi so zu verändern, dass alle benötigten Bibliotheken bei der Installation des Betriebssystems direkt mit installiert werden, inklusive dem Server. Dann könnte man das Projekt als Komplettpaket zusammen mit einem Raspberry Pi anbieten, auf dem die Software bereits installiert ist. Das Netzwerk-Problem allerdings ist komplexer. Wenn man den erforderlichen DynDNS-Service registrieren möchte, kann dies bei verschiedenen Anbietern im Internet geschehen. Manche sind kostenpflichtig, andere sind hingegen kostenlos. Hat man den Service registriert, muss man die erforderlichen Daten in den DynDNS-Client im Hausrouter eintragen. Da es viele verschiedene Router gibt, gibt es auch keine einheitliche Vorgehensweise, um die Daten einzutragen, bei einigen Modellen ist das auch nur bedingt oder gar nicht möglich. In diesem Fall müsste ein DynDNS-Client auf den Router nachinstalliert werden. Aus der Sicht eines Laien ist dieses Problem nicht zu bewältigen.

Des Weiteren ist der Raspberry Pi nicht der leistungsstärkste Computer. Das hier verwendete Raspberry Pi Model B hat einen ARM-Prozessor, der auf 700 MHz getaktet ist. Die Rechengeschwindigkeit reicht aus, um mit dem Transmitter die Signale erfolgreich zu versenden. Werden allerdings zusätzlich Ressourcen von weiterer Software verbraucht (zum Beispiel einem DLNA-Server, auf den vom Netzwerk aus zugegriffen wird), reicht die Rechenleistung nicht mehr aus um die Signale zuverlässig zu übertragen. Dann kommt es zu sporadischen Ausfällen des Systems.

Abschließend lässt sich sagen, dass die Entwicklung des Projekts erfolgreich war. Alle Anforderungen, die an das Projekt gestellt wurden, konnten erfüllt werden, sowohl funktionale als auch nicht-funktionale. Im Zuge der Systemtests ist das System bereits seit über zwei Monaten im Dauereinsatz und funktioniert schnell, stabil und zuverlässig.

6.1.1 Ausblick

Da der Bereich *Home Automation* noch sehr jung ist, sind die Produkte, die in diesem Bereich angeboten werden, noch sehr teuer und der Markt konnte sich noch nicht richtig etablieren. Das wird sich jedoch über die Zeit ändern. Die Produkte werden günstiger werden, eventuell setzt sich auch ein offener Standard durch, der es den Herstellern ermöglicht, günstiger zu produzieren und es den Kunden erlaubt, Produkte verschiedener Anbieter miteinander zu kombinieren. All dies wird dazu führen, dass das öffentliche Interesse für diesen Bereich wächst und sich ein neuer Markt etablieren kann.

Abbildungsverzeichnis

Literaturverzeichnis

[Act13] *Android Action Bar Style Generator*. http://jgilfelt.github.io/android-actionbarstylegenerator/. Version: 2013, Abruf: 10.10.2013

[Bra13] BRANDT, Mathias: *Marktanteile der Smartphone-Betriebssysteme*. http://de.statista.com/themen/581/smartphones/infografik/1097/marktanteile-der-smartphone-betriebssysteme/. Version: 2013, Abruf: 10.10.2013

[Fun13] *ELRO Funksteckdose*. http://g-ec2.images-amazon.com/images/G/03/homeimprovementapplus/49801492/B002QXN7X6_4.jpg. Version: 2013, Abruf: 10.10.2013

[GPSR13] GEIGER, Philip ; PRYSS, Rüdiger ; SCHICKLER, Marc ; REICHERT, Manfred: Engineering an Advanced Location-Based Augmented Reality Engine for Smart Mobile Devices / University of Ulm. Version: October 2013. http://dbis.eprints.uni-ulm.de/972/. Ulm : University of Ulm, October 2013 (UIB-2013-09). – Technical Report

[Hol13] *Android Holo Colors Generator*. http://android-holo-colors.com/. Version: 2013, Abruf: 10.10.2013

[Ico13] *Die in der App verwendeten Icons*. http://www.iconshock.com/android-icons/. Version: 2013, Abruf: 10.10.2013

[Ras13] *Raspberry Pi*. http://microsites.pearl.de/i/07/px9883_0.jpg. Version: 2013, Abruf: 10.10.2013

[RES13] *Definition der REST-Schnittstelle*. `http://www.itwissen.info/definition/lexikon/representational-state-transfer-REST.html`. Version: 2013, Abruf: 10.10.2013

[sql13] *Datenbank sqlite*. `http://www.sqlite.org/`. Version: 2013, Abruf: 10.10.2013

[SSP+13] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; NIENHAUS, Hans ; REICHERT, Manfred: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In: *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, 2013, 509–518

[tra13a] *Anschluss der Pins an den Raspberry Pi*. `http://erik-bartmann.de//download/PiMeUp_ModelB_Rev2.pdf`. Version: 2013, Abruf: 10.10.2013

[tra13b] *Transmitter*. `http://www.miniinthebox.com/de/433mhz-funk-sendemodul-superregenerationsschaltung-fuer-arduino-gruen_p411875.html`. Version: 2013, Abruf: 10.10.2013

[wir13] *Pin-Mapping von wiringPi*. `https://projects.drogon.net/raspberry-pi/wiringpi/pins/`. Version: 2013, Abruf: 10.10.2013

Name: Moritz Kraus Matrikelnummer: 724347

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Moritz Kraus