ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Realisierung einer mobilen Anwendung zur Steuerung einer Ambient Light Infrastruktur

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Mario Weber
mario.weber@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Marc Schickler

2015

Fassung 24. Oktober 2015

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Im stetigen Wandel der Zeit werden immer mehr Geräte mit den unterschiedlichsten Funktionen entwickelt. Fast jeder trägt ein Smartphone bei sich, das Zuhause ist mit einer Hochgeschwindigkeitsleitung mit dem Internet verbunden und immer mehr Geräte werden miteinander vernetzt. So können beispielsweise Kühlschränke Milch nachbestellen, falls diese zur Neige geht, und das Navigationssystem des Autos lädt sich über das verbundene Smartphone stets aktuelle Verkehrsdaten aus dem Internet. Selbst der Fernseher wird seit Jahrzehnten bequem über das Sofa ferngesteuert. Warum besitzt dann fast jeder Haushalt noch normale Lichtschalter, bei denen der Nutzer extra aufstehen muss, um sie zu bedienen? Und auch dann gibt es meist nur zwei Wahlmöglichkeiten: An und Aus. Ziel dieser Bachelorarbeit ist es mit bereits bekannten Geräten - einem *Apple* Smartphone, dem Minicomputer *Raspberry Pi* und einer Lichtleiste des Möbelhauses *IKEA* - eine Infrastruktur zu entwickeln, die es dem Nutzer ermöglicht ohne Konfigurationsaufwand die Beleuchtung über ein Apple-Smartphone zu bedienen und zu erweitern. So soll das Licht gedimmt, dessen Farbe verändert, und falls gewünscht, über das Internet ferngesteuert werden können.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei meiner Bachelorarbeit unterstützt und motiviert haben.

Insbesondere bedanke ich mich bei meinem Betreuer Dipl.-Inf. Marc Schickler, der mir stets seine Hilfe angeboten hat und mich mit Tipps und kritischen Fragen unterstützte. Des Weiteren möchte ich mich dafür bedanken, dass er sich als Betreuer anbot, obwohl die Idee dieser Arbeit von mir stammt. Zu jeder Zeit genoss ich sein vollstes Vertrauen.

Außerdem danke ich Herrn Prof. Dr. Manfred Reichert des Instituts für Datenbanken und Informationssysteme, für die Ermöglichung dieser Abschlussarbeit. Schon während des Studiums besuchte ich viele seiner Veranstaltungen mit Begeisterung.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Zielsetzung 2

1.2 Aufbau der Arbeit 2

2 Grundlagen **4**

2.1 Home Automation 4

2.2 Apple iOS 6

2.3 Raspberry Pi 7

2.4 IKEA Dioder 8

3 Anforderungsanalyse **10**

3.1 Allgemeine Anforderungen an das System 10

3.1.1 Automatische Konfiguration 10

3.1.2 Erweiterbarkeit 11

3.1.3 Anpassung an die Umgebung 11

3.2 Anforderungen 12

3.2.1 Funktionale Anforderungen 12

3.2.2 Nicht-Funktionale Anforderungen 14

4 Konzeption und Realisierung **16**

4.1 Architektur 16

4.1.1 Verbindung - Client zu Lichtleiste 17

4.1.2 Verbindung - Router und Client 18

4.1.3 Verbindung - Router und Server 18

4.1.4 Verbindung - Bedienoberfläche und Router 18

4.1.5 optional: Verbindung - Bedienoberfläche und Router via Internet 18

4.2 Hardware 19

4.2.1 Lichtleiste an Client 19

4.3 Software 22

4.3.1 Client 22

4.3.1.1 Automatische Anmeldung beim Server 22

4.3.1.2 Steuerung der Lichtleiste 23

4.3.1.3 Webserver 25

4.3.2 Server 27
4.3.2.1 Webserver 28
4.3.3 Mobile Anwendung 31
4.3.3.1 Software-Architektur 32
4.3.3.2 Apple-Developer-Program 33
4.3.3.3 Entwicklung mit XCode 33
4.3.3.4 Layout 34

5 Anforderungsabgleich **40**
5.1 Funktionale Anforderungen 40
5.2 Nicht-Funktionale Anforderungen 44

6 Zusammenfassung und Ausblick **47**
6.1 Zusammenfassung 47
6.2 Ausblick 48
6.3 Abschließende Bemerkungen 48

Literaturverzeichnis **50**

1

Einleitung

Unser Alltag wird immer mehr durch Technik unterstützt [SSPR15]. Dabei können wir uns ein Leben ganz ohne diese Helfer kaum mehr vorstellen. Wir sind ständig über unser Mobiltelefon erreichbar, weite Strecken werden problemlos mit dem Navigationsgerät des Autos bewältigt und unser Smartphone unterstützt unseren Alltag mit modernen *Augmented Reality* Technologien [SPSR15]. Auch das Zuhause wird zunehmend technisch weiterentwickelt: Moderne Lichtsteuerungen mit indirekter Beleuchtung und Farbwahl passend zur aktuellen Stimmung sind gefragter denn je. Des Weiteren wird die Haustechnik immer mehr miteinander vernetzt um ständigen Zugriff auf alle Geräte zu erlangen.

“Home Automation“ ist in aller Munde. Hiermit ist die Gesamtheit aller Überwachungs- und Steuerungswerkzeuge gemeint. Dabei sind den möglichen Anwendungsszenarien keine Grenzen gesetzt. Beispielsweise könnte der Nutzer einen Videoanruf auf sein Smartphone bekommen, wenn dieser nicht zu Hause ist, und jemand an der Tür klingelt. Er könnte so beispielsweise dem Paketdienst mitteilen, dieser dürfe das Päckchen vor die Tür legen, um die Sendung dennoch zu zustellen. Ein anderer denkbarer Anwendungsfall wäre das Garagentor mit dem Smartphone zu öffnen, um sich die zusätzliche Fernbedienung im Auto zu sparen.

Bemerkenswert ist jedoch, dass die nötige Hardware oft bereits vorhanden ist. Immer mehr Geräte können mit dem Internet verbunden werden und sind so von überall aus erreichbar. Außerdem werden Smartphones immer leistungsfähiger und günstiger. Nicht ohne Grund sind diese sehr gefragt. Werden diese miteinander vernetzt, bieten Hersteller schon jetzt einige Kom-

fortfunktionen. Beispielsweise senden aktuelle Waschmaschinen eine Nachricht auf das Handy, sobald ein Waschgang abgeschlossen wurde.

Es gibt bereits fertige Lösungen auf dem Markt. Diese sind jedoch immer als geschlossenes System zu betrachten und bieten keine Erweiterungsmöglichkeiten. So ist man immer an den jeweiligen Hersteller gebunden und muss damit auf die Weiterentwicklung der Produkte hoffen. Außerdem decken diese Infrastrukturen meist nur einen einzigen Anwendungsfall, wie beispielsweise die Lichtsteuerung, ab. Eine Erweiterung um eine Heizungssteuerung, Fernbedienung für den Fernseher et cetera ist nicht möglich. Des Weiteren ist die Hardware hier meist sehr kostspielig. Möchte der Nutzer weitere Anwendungsszenarien abdecken, wird der Kostenfaktor noch verstärkt, da er beispielsweise mehrere Steuerungszentralen pro Szenario benötigt, obwohl dies technisch meist mit einer Einzigen realisierbar wäre. Das entwickelte System dieser Arbeit wird keines dieser Probleme aufweisen und erheblich günstiger sein.

1.1 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll eine Infrastruktur entwickelt und realisiert werden, mit der der Nutzer die Möglichkeit haben soll das Licht in seiner Umgebung ("*Ambient Light*") [GLRS14] passend zur aktuellen Stimmung einzustellen [KRRH11]. Die Bedienung der Helligkeit und der Farbwerte des Lichts sollen schnell und einfach mittels einer *iOS*-Applikation erfolgen. Hierbei wird besonderer Wert darauf gelegt, dass nur Hardware verwendet wird, die es bereits auf dem Markt gibt. Außerdem soll der Nutzer nur wenig Konfigurationsaufwand haben. So soll gewährleistet werden, dass auch Anwender ohne technische Kenntnisse das System in Betrieb nehmen und verwenden können. Des Weiteren sollen Erweiterungen des Systems, wie beispielsweise der bereits genannte Garagentoröffner, einfach zu realisieren und anzuschließen sein. Um dies sicher zu stellen werden stets offene Schnittstellen zur Kommunikation verwendet.

1.2 Aufbau der Arbeit

Nachdem nun die Einleitung sowie die Motivation und Zielsetzung der Arbeit aufgezeigt wurden, werden im zweiten Kapitel die Grundlagen der benötigten Technologien erklärt. Es wird zunächst ein Einblick gegeben, was *Home Automation* überhaupt ist und warum dies eine sehr spannende und interessante Thematik darstellt. Es wird der Werdegang von *Apples iOS* aufgezeigt, und warum dieses als Betriebssystem für die mobile Anwendung ausgewählt wurde. Darauf folgt eine kurze Erklärung der verwendeten Hardware-Komponenten.

Das dritte Kapitel beschäftigt sich mit den Anforderungen, welche das zu entwickelnde System erfüllen soll. Diese werden in drei Blöcke eingeteilt: Die Grundbedingungen, auf denen alle anderen Anforderungen aufbauen. Sie beschreiben zunächst, mit welcher Problemstellung

sich die Arbeit befasst. Die restlichen und deutlich detaillierteren Anforderungen werden dann in funktionale- und nicht-funktionale-Anforderungen aufgeteilt und erklärt.

Das vierte Kapitel der Arbeit beschäftigt sich mit der tatsächlichen Entwicklung der Infrastruktur und welche Schritte dafür notwendig waren. Es wird zunächst erklärt, wie die einzelnen Komponenten zueinander in Verbindung stehen, um miteinander kommunizieren zu können. Im Anschluss werden die Soft- und Hardwarebausteine aufgezeigt. Der Softwareteil ist hierbei aufgeteilt in Client, Server und Smartphone-Applikation.

Kapitel fünf widmet sich dem Abgleich der zuvor festgelegten Anforderungen. Es wird kritisch betrachtet, welche Anforderungen erfüllt wurden, und welche nicht. Außerdem werden sie mittels Schulnotensystem bewertet. Um eine bessere Einsicht in die Bewertung zu erlangen, wird für jede einzelne Anforderung erläutert, wie es zu diesem Ergebnis kam. Auch hier wird wieder zwischen funktionalen- und nicht-funktionalen-Anforderungen unterschieden.

Im letzten Teil wird ein Fazit über das entwickelte System gezogen. Des Weiteren wird ein möglicher Ausblick und der Mehrwert der Infrastruktur aufgezeigt. Zum Schluss werden noch ein paar abschließende Bemerkungen gemacht.

2

Grundlagen

In diesem Kapitel wird auf die einzelnen Bestandteile der Arbeit eingegangen. Diese werden kurz erläutert und beschrieben. Außerdem wird erklärt, warum genau diese Komponenten zum Einsatz kommen.

2.1 Home Automation

Home Automation bezeichnet die Steuerung und Überwachung aller möglichen Geräte und Gegebenheiten im eigenen Zuhause. Diese können sowohl lokal im eigenen Netzwerk vor Ort, oder auch über das Internet bedient werden [Hil15].

Der Aufbau einer Home-Automation-Infrastruktur ist hierbei abhängig von den gewünschten Funktionen des Anwenders [BH15]. Die grundlegende Architektur unterscheidet sich jedoch kaum. Abbildung 2.1 zeigt eine mögliche Konstellation der einzelnen Bestandteile.

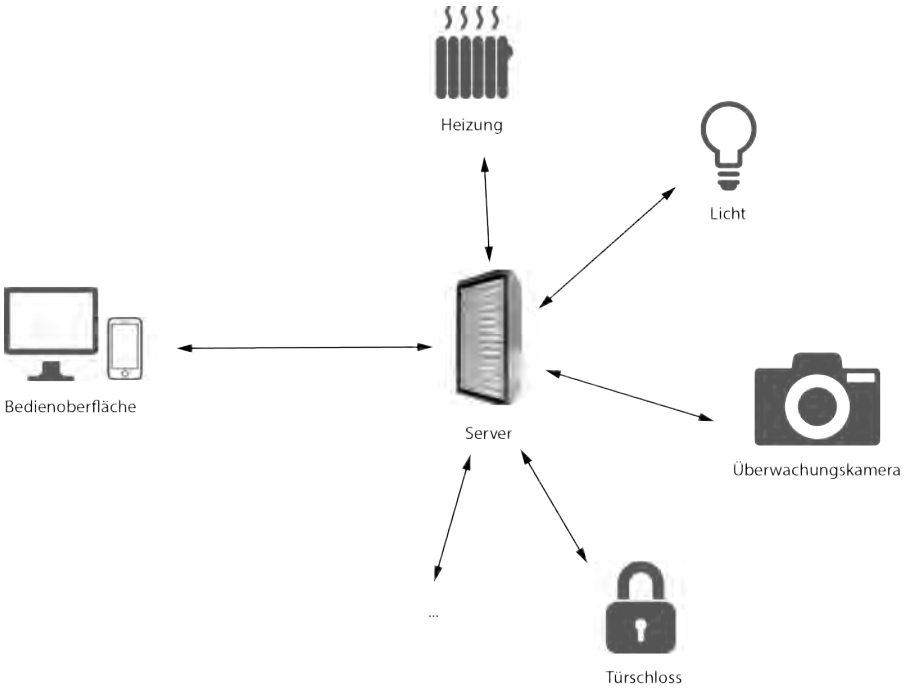

Abbildung 2.1: Home-Automation-Architektur

Es wird immer eine zentrale Steuerungseinheit benötigt. Sie ist dafür zuständig die einzelnen Sensoren und Aktoren zu verwalten [Tad15]. Auch die Nutzerinteraktionen werden von ihr zunächst aufgearbeitet, bevor sie an die einzelnen Endgeräte weitergeleitet werden. Die Steuerungszentrale (auch "*Server*") wird in diesem Projekt mit dem Minicomputer *Raspberry Pi* realisiert.

Um einen manuellen Eingriff in die Infrastruktur zu ermöglichen, wird außerdem eine Bedienoberfläche benötigt. Diese kann sehr unterschiedlich umgesetzt werden. Üblich sind mobile Anwendungen auf dem Smartphone, Bildschirme mit *Touch*-Funktion, die in die Wand eingearbeitet sind, oder einfache Fernbedienungen, die via Infrarotlicht oder Funksender Interaktionen an den Server übermitteln. Da heutzutage fast jeder ein Smartphone bei sich trägt, wird in dieser Arbeit eine mobile Anwendung für *iOS*-Geräte zur Steuerung entwickelt. Sollte dieses einmal nicht zur Verfügung stehen, wird dem Anwender außerdem eine Weboberfläche angeboten, die über alle Browser erreichbar ist.

Zuletzt fehlen noch die einzelnen Endpunkte. Sie stellen Sensoren, Aktoren oder auch beides zur Verfügung. Jeder dieser Endpunkte, oder auch *Clients*, übernimmt eine Aufgabe. Diese könnte beispielsweise das Messen der Temperatur und Feuchtigkeitswerte, oder die Ansteuerung des elektrischen Rollladens sein. In dieser Arbeit soll mittels Client das Umgebungslicht gesteuert werden. Auch hier wird wieder ein *Raspberry Pi* als Controller verwendet. Das Licht selbst wird über die LED-Lichtleiste "*Dioder*" der Möbelhauskette *IKEA* realisiert.

Bereits im Jahr 1999 weckte der amerikanische Film "Smart House" den Wunsch der Zuschauer ein voll automatisiertes Zuhause zu besitzen. Dieses Haus wird durch einen Hochleistungscom-

puter gesteuert, der die Bewohner rund um die Uhr analysiert und all' deren Wünsche erfüllt. Dabei ist der Computer außerordentlich lernfähig. Er kocht ihnen beispielsweise Mahlzeiten, die perfekt auf deren jeweilige Ernährungsbedürfnisse abgestimmt ist.

Der Film zeigt, dass die denkbaren Anwendungsfälle grenzenlos sind: Das Licht wird nur in den Räumen eingeschaltet, die man betritt, und beim Verlassen wieder ausgeschaltet. Mit dem Starten des Autos auf dem Weg zur Arbeit wird das Garagentor geöffnet und alle Fenster geschlossen. Sobald der Anwender das Büro wieder verlässt, wird zu Hause die Heizung eingeschaltet, so dass es schön warm ist, wenn die Bewohner nach Hause kommen. Abends werden die Rollläden heruntergefahren und das Licht für eine angenehme Atmosphäre gedimmt. So oder so ähnlich könnte ein normaler Tagesablauf mit einem automatisierten zu Hause aussehen.

Heutzutage sind immer mehr dieser Wünsche erfüllbar. Die Technik ist auf einem Stand, der all dies kostengünstig ermöglicht [BLM+11]. Dennoch besitzt kaum jemand ein voll automatisiertes Zuhause. Die Produkte auf dem Markt haben meist noch viele Schwächen, die von einem Kauf abhalten. Oftmals sind diese nur mit Geräten bestimmter Hersteller kompatibel oder noch sehr teuer in der Anschaffung. Auch die Bedienbarkeit ist oft sehr eingeschränkt und für die Einrichtung wird ein Techniker mit speziellem Fachwissen benötigt. Außerdem gibt es nur wenige Komplettlösungen. Dennoch wird ständig weiter an neuen Systemen entwickelt.

Teilweise sind potentielle Nutzer außerdem sehr skeptisch dem Konzept gegenüber. Wohin gehen all' die Daten meiner Bewegungsabläufe? Werden Verhaltensmuster erstellt, die dann analysiert und zu kommerziellen Zwecken verwendet werden? Hier besteht noch einiges an Aufklärungsbedarf.

2.2 Apple iOS

Apples Betriebssystem für seine mobilen Endgeräte *iPhone*, *iPad*, *iPod* und *Apple TV* nennt sich *iOS* und wird nur für deren eigenen Geräte lizenziert. Es basiert auf einem *OS X*-Kern und geht letztendlich auf einen *Unix*-Kern zurück. Die Entwicklung begann 2005 für das zukünftige *iPhone* der ersten Generation, welches zusammen mit *iOS* im Jahr 2007 vorgestellt wurde [Wik15].

Erst im März 2008 veröffentlichte *Apple* das *iOS* Software-Development-Kit (SDK). Dieses ermöglichte es erst, Drittanbieter-Applikationen (Apps) zu entwickeln. Um deren Anwendungen zu veröffentlichen, wurde außerdem der *App-Store* eingeführt. Dieser gilt als zentrale und einzige Bezugsquelle aller Applikationen für das mobile Betriebssystem. Sämtliche Anwendungen werden von *Apple* überprüft. Dies bedeutet Sicherheit für den Nutzer, resultiert aber auch oft in Einschränkungen der Anwendungsmöglichkeiten.

Ursprünglich wurde die Programmiersprache *Objective-C* [App15a] benötigt, um Anwendungen für *iOS* zu entwickeln. Sie erweitert die Sprache *C* um Funktionen der Objektorientierung. Seit *iOS* Version 8 kann *Apples* neue Programmiersprache *Swift* [App15c] für die Entwicklung verwendet werden. Diese ist ebenfalls objektorientiert und verbindet Ideen von *Objective-C*, *Haskel*,

Python, *C#* und anderen. Außerdem kann innerhalb *Swift* ein Brücke zu *Objective-C*-Code aufgebaut werden um bestehende Lösungen in die neue Sprache einzubinden. Im Juni 2015 stellte *Apple Swift 2* vor. Neben einigen sprachlichen Neuerungen ist vor allem die Änderung in eine Open Source Lizenz zu beachten.

Mittlerweile wurde *iOS* in der Version 9 veröffentlicht. Dem Nutzer werden immer mehr Funktionen geboten. Die integrierten Applikationen und das Design wurden grundlegend verbessert. Dennoch werden diese stetig weiterentwickelt. Auch immer mehr Drittanbieter entwickeln Anwendungen mit modernsten Funktionen [SSP+13]. *iOS 9* gehört zu den schnellsten und stabilsten Betriebssystemen für mobile Endgeräte [Bre15].

2.3 Raspberry Pi

Der *Raspberry Pi* gehört zu den kleinsten Computern der Welt. Er wurde von der britischen *Raspberry Pi Foundation* entwickelt und kam Anfang 2012 auf den Markt [Fou15]. Dabei zeichnet er sich vor allem durch seinen geringen Stromverbreich und die kompakte Bauweise aus. Die komplette Hardware ist auf der Größe einer Scheckkarte verbaut. Mittlerweile gibt es verschiedene Modelle und Revisionen des Minicomputers. In diesem Projekt wurde das derzeit aktuellste und günstigste *Raspberry Pi rev. 2.0 Model B* verwendet (Abbildung 2.2).

Abbildung 2.2: Raspberry Pi

Das Besondere an diesem Einplatinen-Computer sind die vielen Anschlüsse. Es steht ein LAN-Port zur Verfügung, mit dem der Computer mit dem Netzwerk / Internet verbunden wird. Des Weiteren gibt es die Möglichkeit über einen der beiden USB-Ports einen WLAN-Adapter oder andere Geräte nachzurüsten. An die GPIO-Pins werden später die entsprechenden Kontakte der LED-Leiste angeschlossen. Die Pins können einzeln angesteuert und mit einer logischen 1

oder 0 belegt werden. Die anderen Anschlüsse wie HDMI, Video Out, Audio, ... werden nicht benötigt. Strom bekommt die Platine über einen gewöhnlichen Micro-USB-Anschluss mit circa 700 Milliampere - je nach Last.

Es können die verschiedensten Betriebssysteme installiert werden. Dazu muss dieses auf eine SD-Karte gespielt werden, die dann in den dafür vorgesehenen Platz des *Raspberry Pis* gesteckt wird. In dieser Arbeit wurde das auf *Debian* basierende *Raspbian* verwendet, das speziell auf die Bedürfnisse des Minicomputers Wert legt. Es werden viele Werkzeuge und Packages mit installiert, um eine möglichst gute Hardwareunterstützung zu gewährleisten.

Raspberry Pi rev. 2.0 Model B	
Chipsatz	ARM11
CPU	ARM1176JZF-S 700 MHz 1 Kern
GPU	Broadcom Dual Core VideoCore IV, Full HD 1080p30
Arbeitsspeicher (RAM)	512 MB
LAN-Port	10/100 Mbit/s Full Duplex
USB 2.0 Anschlüsse	2
Tonausgabe	HDMI (digital); 3,5-mm-Klinkenstecker (analog)
GPIO-Pins	17
Leistungsaufnahme	3,5 W (700 mA)
Stromversorgung	5,0 V Micro-USB-Anschluss
Länge x Breite x Höhe	93,0 mm x 63,5 mm x 20,0 mm
Gewicht	45 g

Tabelle 2.1: Technische Daten Raspberry Pi rev. 2.0 Model B

2.4 IKEA Dioder

Als Leuchtmittel werden LED-Leisten des Möbelhauses *IKEA* verwendet. Diese wurden ausgewählt, da das Produkt günstig im Einkauf ist und die Möbelkette bundesweit vertreten ist. Der Artikel ist in mehreren Ausführungen verfügbar. Verwendet wurde eine vierteilige Lichtleiste mit LEDs für die drei Farben Rot, Grün und Blau. Hieraus lassen sich alle erdenklichen Farbkombinationen mischen - inklusive weiß. Die einzelnen Leuchten können aneinandergereiht oder aber auch sternförmig angeordnet werden. Außerdem wird das Produkt in rund und mit flexibler Lichtleiste verkauft. Hier kann der Nutzer selbst wählen welches Produkt besser zu seinen Anforderungen und Gegebenheiten passt. Alle Modelle sind mit dem entwickelten System kompatibel.

Abbildung 2.3: IKEA Dioder

Die Lichtleiste bietet eine eigene Steuerung, mit der die Helligkeit und Farbe eingestellt werden können. Diese musste geöffnet werden, um die entsprechenden Kontakte mit den PINs des *Raspberry Pis* zu verbinden. Sie wird im weiteren Verlauf der Arbeit genauer betrachtet. Außerdem wird die Lichtleiste mit Netzteil geliefert. Dieses wird direkt in einen normale Steckdose mit 230 Volt eingesteckt.

IKEA Dioder Lichtleiste 4-tlg., LED, bunt	
Kabellänge Stecker zu Transformator	2,5 m
Kabellänge Transformator zu Leuchte	1 m
Energieverbrauch	6,0 W
LED Brenndauer	ca. 20.000 Std.
Lichtstrom	90 lm
Länge pro Leuchte	25 cm

Tabelle 2.2: Technische Daten IKEA Dioder Lichtleiste 4-tlg., LED, bunt

3

Anforderungsanalyse

Das Ziel dieser Arbeit ist es eine komplette Infrastruktur einer Lichtsteuerung zu entwickeln. Hierfür sind sowohl eine Steuerungszentrale und eine beliebige Anzahl an Clients notwendig, an die die Lichtleisten angeschlossen werden. Um das System zu bedienen, wird außerdem eine *iOS*-Applikation für *Apples iPhone* entwickelt.

Zunächst müssen alle Anforderungen an das System klar definiert werden. Hierfür werden erst die allgemeinen Anforderungen an das System festgelegt, welche es dann gilt in funktionale und nicht-funktionale Anforderungen einzuteilen. Im Nachfolgenden werden diese aufgeführt und erklärt.

3.1 Allgemeine Anforderungen an das System

Die allgemeinen Anforderungen an das System sind die Grundlage dieses Projektes. Sie stellen hierbei die Verbesserung der bereits bestehenden Infrastrukturen dar.

3.1.1 Automatische Konfiguration

Jeder soll dazu in der Lage sein, das System in Betrieb zu nehmen und bedienen zu können. Vom Nutzer darf kein technisches Grundverständnis gefordert werden. Hierzu ist es erforderlich,

dass alle Gegebenheiten vom System selbständig analysiert und ausgewertet werden. Dies gilt sowohl für Server und Client, als auch für die Smartphone-Anwendung.

Der Anwender soll die Möglichkeit haben das System in einer Grundkonfiguration zu beziehen. Sobald die Geräte mit Strom versorgt und mit dem Netzwerk verbunden werden, sollen sich diese autonom konfigurieren und miteinander kommunizieren. Schon jetzt soll der Nutzer die Möglichkeit haben die Lichtleisten per Webinterface zu bedienen. Zusätzlich soll er mit seinem Smartphone die passende Anwendung aus dem *App Store* herunterladen können. Auch die mobile Applikation soll automatisch nach entsprechenden Geräten im eigenen Netzwerk suchen. Werden diese gefunden, sollen sie dem Nutzer mit den jeweiligen Bedienelementen dargestellt werden.

Entscheidet sich der Nutzer weitere Lichtleisten hinzuzufügen, sollen diese ebenfalls nur in Betrieb genommen werden müssen und das System bindet diese dann vollständig in die bestehende Infrastruktur ein. Ein weiterer Konfigurationsbedarf soll nicht notwendig sein. Gleiches gilt für das Entfernen von Lichtleisten.

3.1.2 Erweiterbarkeit

Idealerweise übernimmt das System die Steuerung eines kompletten Zuhauses. Dabei spielt es keine Rolle, welche Szenarien abgedeckt werden sollen oder welche Gegebenheiten zu beachten sind. Denkbar wäre beispielsweise das System um einen Garagentoröffner zu erweitern, der mit einer *Android*-Applikation bedient werden kann. Oder aber eine Kaffeemaschine, die via Weboberfläche ein- und ausgeschaltet werden kann.

Hierfür sind offene Schnittstellen und eine klare Programmierung notwendig. Es muss gewährleistet werden, dass die verschiedenen Bausteine der Infrastruktur über standardisierte Kanäle miteinander kommunizieren. So können weitere Funktionen entwickelt werden, die allerdings über den Umfang dieser Arbeit hinausreichen würden.

3.1.3 Anpassung an die Umgebung

Der Anwender hat sein Smartphone in der Regel überall dabei. Wenn er das Haus verlässt bestehen andere Bedingungen als in der Wohnung. Beispielsweise ist meist keine *WiFi*-Verbindung mehr vorhanden.

Die mobile Anwendung muss dazu in der Lage sein dies zu unterscheiden. Ist das Smartphone mit dem mobilen Internet verbunden, soll beispielsweise nicht mehr nach Geräten im lokalen Netzwerk gesucht werden.

Auch Server und Client müssen sich an die Umgebung anpassen. Fast jedes Netzwerk ist anders konfiguriert. Meist werden Grundkonfigurationen der jeweiligen Router-Hersteller verwendet. Hierbei gibt es nur wenige Standards. Aber auch in professionell administrierten Netzwerken

muss das System voll funktionsfähig sein. Es ist notwendig, dass Server und Client die Umgebung analysieren und sich nahtlos in die bestehende Infrastruktur einbinden.

3.2 Anforderungen

Nachdem nun die grundlegenden Bedingungen an das System definiert sind, müssen diese konkretisiert werden. Hierbei wird zwischen funktionalen und nicht-funktionalen Anforderungen unterschieden. Nachdem das System entwickelt wurde, wird im Anforderungsabgleich geprüft, ob und in welchem Maße diese erfüllt wurden bzw. überhaupt erfüllbar sind.

3.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen (FA) legen die Funktionen der jeweiligen Bausteine fest. Nutzerfreundlichkeit o.Ä. fließen hier nicht mit ein.

Mobile Anwendung

FA #1: Autonome Konfiguration

Die Smartphone-Anwendung soll den Server im Netzwerk selbständig suchen. Wurde dieser gefunden, soll die Applikation die Informationen des Servers abrufen und dem Nutzer zugänglich machen.

FA #2: Anpassung an Umgebung

Je nachdem, ob der Anwender sich im WLAN befindet oder mit dem mobilen Internet verbunden ist, soll die Applikation nach Servern suchen, oder auch nicht.

FA #3: Mit Server via Internet verbinden

Sofern der Nutzer den Zugriff auf das System via Internet freigeschaltet hat, muss er die Möglichkeit haben die entsprechende URL in der mobilen Anwendung hinterlegen zu können.

FA #4: Speicherung der Konfiguration in Datenbank

Werden vom Nutzer Adressen zu Servern hinterlegt, soll die Applikation diese in einer Datenbank speichern. Diese wird bei jedem Start der Anwendung ausgelesen. So kann auch beispielsweise nach einem Neustart des Smartphones die Konfiguration weiter verwendet werden.

FA #5: Manuelle Aktualisierung

Der Nutzer soll die Möglichkeit haben die Suche nach Servern / Clients manuell starten zu können.

Server

FA #6: Selbständige Einbindung ins Netzwerk

Der Server soll sich selbständig in ein bestehendes Netzwerk integrieren können ohne andere Geräte zu behindern.

FA #7: Plug & Play

Sobald der Server mit dem Netzwerk verbunden und an den Strom angeschlossen wird, sollen automatisch alle notwendigen Dienste gestartet werden. Es soll zu keiner Zeit eine manuelle Konfiguration notwendig sein.

FA #8: REST-Schnittstelle und JSON

Die kompletten Funktionen sollen mittels einer REST-konformen Schnittstelle verfügbar sein. Informationen sollen im JSON-Format übermittelt werden. So können beispielsweise alle verbunden Clients mit ihren jeweiligen Informationen mittels einer bestimmten URL aufgelistet werden.

FA #9: Optionaler Zugriff via Internet

Optional soll der Nutzer die Möglichkeit haben die Lichtleisten via Internet zu steuern. So kann er beispielsweise im Urlaub zum Schutz vor Einbrechern das Licht ein- und ausschalten. Hierfür werden dann aber zusätzliche Konfigurationen im Menü des Routers notwendig auf die das System keinen Einfluss hat.

FA #10: Webinterface

Besitzt der Anwender kein Smartphone oder ist dessen Akku leer, hat dieser außerdem die Möglichkeit das System mittels Webinterface zu steuern. Dieses soll die gleichen Funktionen wie die Smartphone-Anwendung bieten.

Client

FA #11: Selbständige Einbindung ins Netzwerk

Wie der Server auch, soll sich der Client automatisch mit dem Netzwerk verbinden ohne andere Geräte zu stören.

FA #12: Eigenständiges Finden des Servers

Der Client soll nach dem Starten einen Server im Netzwerk suchen. Sobald dieser gefunden wurde, soll er sich bei diesem anmelden und ihm seine Informationen zur Verfügung stellen.

FA #13: Plug & Play

Auch der Client soll nach dem Verbinden ohne weitere Konfigurationen zur Verfügung stehen.

FA #14: REST-Schnittstelle und JSON

Die kompletten Funktionen des Clients sollen ebenfalls mittels einer REST-konformen Schnittstelle zur Verfügung stehen. Beispielsweise soll der gewünschte Farbwert in der aufgerufenen URL angegeben werden. Informationen sollen auch hier im JSON-Format übertragen werden.

3.2.2 Nicht-Funktionale Anforderungen

Die nicht-funktionalen Anforderungen (NFA) legen fest, welche Eigenschaften die Infrastruktur haben soll. Der Nutzer erkennt diese oft nicht direkt, da sie definieren, wie gut bzw. wie qualitativ das System sein soll. Dennoch sind diese Anforderungen sehr wichtig. Sie stehen in direkter Relation zum Nutzungsvergnügen und der Effizienz der Bedienung. Aber auch die Zuverlässigkeit, Wartbarkeit und Korrektheit werden in den nicht-funktionalen Anforderungen klassifiziert.

Mobile Anwendung

NFA #1: Benutzbarkeit und Verständlichkeit

Die mobile Anwendung soll schlank und leicht verständlich gehalten werden. Der Nutzer soll keine lange Einarbeitungsphase benötigen und die Bedienelemente intuitiv nutzen können [Flo15].

NFA #2: Performanz und Effizienz

Der Start der Applikation darf nicht zu lange dauern. Außerdem soll sie sich entsprechend des aktuellen Szenarios optimieren, um maximale Performanz zu erreichen. Die Bedienung muss ohne Verzögerungen zu bewerkstelligen sein.

NFA #3: Wartbarkeit und Erweiterbarkeit

Die Anwendung soll um zukünftige Szenarien ohne hohen Entwicklungsaufwand erweiterbar sein.

NFA #4: Skalierbarkeit

Je nach System und Konfiguration will der Anwender mehrere Server und Clients bedienen. Die Anzahl ist hierbei variabel. Die Anwendung soll alle denkbaren Varianten unterstützen.

Server

NFA #5: Benutzbarkeit und Verständlichkeit

Das Webinterface soll analog zur mobilen Anwendung schlank und intuitiv bedienbar sein.

NFA #6: Performanz und Effizienz

Sobald der Server mit Energie versorgt wird, soll dieser möglichst schnell booten und alle notwendigen Dienste starten. Außerdem sollen Benutzerinteraktionen schnell an die entsprechenden Clients weitergereicht werden.

NFA #7: Wartbarkeit und Erweiterbarkeit

Der Server soll, wie auch die mobile Anwendung, ohne hohen Entwicklungsaufwand zukünftige Szenarien abdecken können. Außerdem sollen auch andere Bedienoberflächen, beispielsweise eine *Android*-Applikation, unterstützt werden.

NFA #8: Skalierbarkeit

Es sollen beliebig viele Clients mit dem Server verbunden und gesteuert werden können.

Client

NFA #9: Zuverlässigkeit

Nimmt der Benutzer Änderungen an der Helligkeit oder Farbe der Lichtsteuerung vor, sollen diese Werte auch zuverlässig übernommen werden.

NFA #10: Performanz und Effizienz

Ähnlich zum Server soll auch der Client, sobald er mit Energie versorgt wird, schnell booten und bedienbar sein. Änderungen an den Lichtwerten sollen möglichst schnell und ohne erkennbare Verzögerung übernommen werden.

4

Konzeption und Realisierung

Um die Entwicklung der Infrastruktur aufzuzeigen, wird diese in drei Kapitel aufgeteilt. Zunächst wird das System als Ganzes erklärt. Es wird gezeigt welche Komponenten es gibt und wie diese ineinander greifen. Außerdem wird dargelegt, wie sie miteinander kommunizieren und welche Kanäle hierfür verwendet werden.

Danach wird aufgezeigt, wie die Hardwarekomponenten *Raspberry Pi* und Lichtsteuerung physisch miteinander verbunden sind.

Zuletzt wird die entwickelte Software für die einzelnen Funktionen erklärt.

4.1 Architektur

Die Architektur zeigt die Komponenten der Infrastruktur mit deren Beziehungen zueinander auf. Diese müssen alle verbunden sein, um miteinander kommunizieren zu können. Dabei gilt es einige Dinge zu beachten. Manche Verbindungen können sich ändern oder sind kabellos. Andere werden von Sicherheitseinstellungen geblockt.

Außerdem wird ein Überblick darüber gegeben, welche Funktionen die einzelnen Bausteine erfüllen müssen. Stehen diese fest, können die Komponenten miteinander verbunden, und so das System aufgebaut werden.

In Abbildung 4.1 werden die Abhängigkeiten der Komponenten grafisch dargestellt.

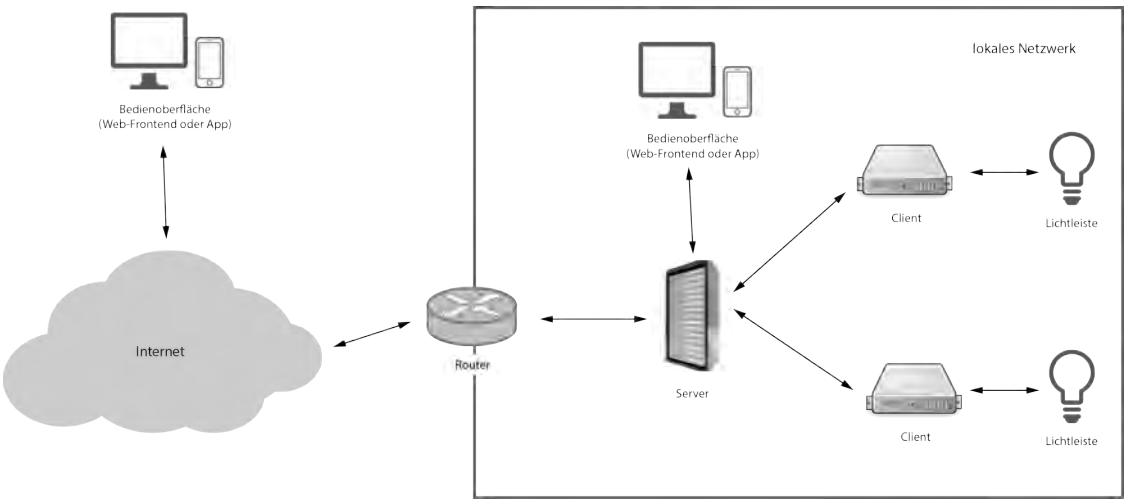

Abbildung 4.1: Architektur

Um die Lichtleiste steuern zu können, ist ein Controller, der mit der Leiste verbunden ist, elementar. Dieser übernimmt die direkte Änderung der Farb- und Helligkeitswerte. Er empfängt die vom Anwender gewünschten Werte, interpretiert sie, und gibt diese in einem für die Lichtleiste verständlichen Format aus. Es kann beliebig viele dieser Clients in der Infrastruktur geben.

Damit die Clients Daten empfangen können, müssen diese zunächst von einem Server versendet werden. Der Server stellt damit die Kommunikation der Bedienoberfläche mit den Clients zur Verfügung. Alle vom Benutzer eingestellten Werte werden erst an den Server übermittelt, der dann den richtigen Client auswählt, an den er die Daten weitergibt. Dies ist notwendig, da der Anwender sonst eine eigene Bedienoberfläche für jeden Client benötigen würde und diese jeweils einzeln aufrufen und wechseln müsste.

Um die Werte der Lichtleisten verändern zu können, hat der Anwender zwei Möglichkeiten zur Auswahl: die mobile Anwendung und das Web-Frontend. Die Weboberfläche stellt der Server selbst zur Verfügung. Er bietet einen eigenen Webserver an, der über jeden beliebigen Webbrowser erreichbar ist - auch mit dem Smartphone. Die mobile Anwendung ist jedoch um einiges komfortabler. Sie muss zunächst aus dem *App-Store* auf das Smartphone heruntergeladen werden. Hat der Nutzer dies erledigt, werden ihm die Bedienelemente aller Clients dargestellt. Ebenso wie der Browser muss auch die Smartphone-Anwendung mit dem Server kommunizieren können. Diese Verbindung kann entweder lokal im gleichen Netzwerk oder über das Internet hergestellt werden. In beiden Fällen findet die Kommunikation zwingend über den eigenen Router statt.

4.1.1 Verbindung - Client zu Lichtleiste

Die Lichtleiste verfügt über eine eigene Steuerung. Sie übernimmt die hardwarenahe Bedienung der einzelnen LEDs innerhalb der Leiste. An ihr befindet sich ein Drehrad mit dem der Anwender die Farbwerte händisch einstellen kann. Die Steuerung wurde geöffnet, um eine Verbindung

zum Client herzustellen. Dieser kann nun das Verändern der Werte vornehmen, ohne dass der Benutzer die Steuerung in die Hand nehmen muss. Die genaue Vorgehensweise und die Verkabelung der Komponenten werden im Hardware-Teil noch detaillierter erläutert (siehe Kapitel 4.2.1).

4.1.2 Verbindung - Router und Client

Außerdem muss der Client mit dem Router verbunden werden. Hierfür gibt es verschiedene Möglichkeiten. Die einfachste Methode ist das direkte Verbinden mittels eines Patch-Kabels. Der Client bekommt so via DHCP eine IP-Adresse vom Router zugeteilt und ist hierüber sofort erreichbar. Der Nachteil ist allerdings, dass ein eigenes Kabel für die Verbindung notwendig ist.

Eine andere Möglichkeit stellt die Verbindung über WLAN dar. Hierfür muss der Client allerdings konfiguriert werden. Er benötigt den Namen des Netzwerks, mit dem er sich verbinden soll, und eventuelle Sicherheitseinstellungen wie das Kennwort, mit dem die Verbindung verschlüsselt ist. Diese Methode ist für Laien also nicht zu empfehlen.

Der beste Mittelweg könnte die Verbindung mittels dLAN-Adapter sein. Es ist keine Konfiguration notwendig und der Anwender spart sich die direkte Verbindung mit dem Router über ein separates Kabel. Mit einer Steckdose muss der Client auf jeden Fall verbunden werden.

4.1.3 Verbindung - Router und Server

Ebenso wie der Client muss auch der Server mit dem Router verbunden werden. Hierbei stehen dem Nutzer die selben Möglichkeiten zur Verfügung. Allerdings ist der Server nicht ortsgebunden, da keine direkte Verbindung zu den Lichtleisten notwendig ist. Es könnte sich also, je nach Aufbau der Geräte im Netzwerk des Nutzers, anbieten, den Server direkt mit einem Patch-Kabel am Router anzuschließen.

4.1.4 Verbindung - Bedienoberfläche und Router

Auch die Bedienoberfläche des Anwenders muss mit dem Router verbunden sein. Die Verbindungseinstellungen übernimmt hierbei das jeweilige Endgerät. In der Regel befinden sich Computer, Laptop oder Smartphone bereits im lokalen Netzwerk. Diese Einstellungen werden auch von der mobilen Anwendung beziehungsweise dem Web-Browser verwendet. Es spielt keine Rolle welche Verbindungsmethode hierbei verwendet wird.

4.1.5 optional: Verbindung - Bedienoberfläche und Router via Internet

Auch über das Internet können die Geräte miteinander kommunizieren. So kann der Anwender auch außerhalb des lokalen Netzwerkes den Server, und damit die Clients, bedienen. Um eine

einwandfreie Verbindung zu gewährleisten sind allerdings einige Konfigurationen und Sicherheitseinstellungen notwendig.

Zunächst benötigt die Benutzeroberfläche die nach außen sichtbare Adresse des Routers. Über diese wird die Verbindung hergestellt. Je nach Anschluss beim Internet Service Provider ändert sich die Adresse allerdings täglich. Um die ständige Erreichbarkeit über den immer gleichen Namen zu gewährleisten muss der Router bei einem DynDNS-Dienst registriert werden. Der Dienst fungiert als Wörterbuch, bei dem die aktuelle Adresse unter einem bestimmten, sich nicht ändernden Namen, erfragt werden kann. Sobald der Router eine neue Adresse zugewiesen bekommt unterrichtet er den DynDNS-Dienst und teilt diesem die neue Adresse mit. Ruft ein Anwender nun den festgelegten Namen auf, wird dieser vom DynDNS-Dienst in die aktuell gültige IP-Adresse übersetzt, auf die der Router reagieren kann.

Da sich im lokalen Netzwerk unter Umständen mehrere Endgeräte befinden, muss dem Router noch mitgeteilt werden, welches Gerät, je nach Anfrage, angesprochen werden soll. Hierfür kann das Port-Forwarding im Backend des Routers konfiguriert werden. Der Nutzer legt fest, an welche lokale IP-Adresse und welchen Port die Daten weitergeleitet werden sollen.

4.2 Hardware

Nachdem nun der Aufbau der Infrastruktur klar ist, werden die Hardwarekomponenten betrachtet. Es wird gezeigt, wie die Lichtleiste mit den Clients verbunden werden und welche Schritte hierfür notwendig sind. Außerdem wird erklärt, warum genau diese Herangehensweise gewählt wurde.

4.2.1 Lichtleiste an Client

Die Lichtleiste bietet, neben Netzteil und Verteiler, auch eine integrierte Steuerung. Diese ist dafür zuständig die gewünschten Leuchtdioden in der Lichtleiste mit ausreichend Strom zu versorgen. An ihr kann der Anwender von Hand mittels einem Farbwähler Änderungen vornehmen.

Diese Steuerung musste geöffnet werden, um mögliche Verbindungsstellen für den Client zu finden. Auf der Platine befinden sich, wie in Abbildung 4.2 zu sehen ist, drei passende Kontaktflächen, die für die Bestimmung der Farbe zuständig sind.

Abbildung 4.2: Controller des IKEA Dioder

Neben den drei Kontaktflächen für die Grundfarben Rot, Blau und Grün befindet sich dort außerdem eine Anschlussmöglichkeit für die *GND*. Werden beispielsweise auf Rot *+5 Volt* angelegt, und die *GND* verbunden, liegt dort eine logische *1* an. Wenn nun noch die anderen Farben mit logisch *0* versorgt werden, leuchten nur die roten Leuchtdioden der Lichtleiste. Gleiches gilt für die anderen Farben.

Mischungen sind ebenfalls möglich. Werden alle Dioden eingeschaltet wirkt das Licht für das menschliche Auge weiß.

Die Helligkeit der einzelnen Farben zu steuern ist dagegen etwas komplizierter. Es darf nicht einfach nur weniger Spannung angelegt werden, da sonst die Leuchtdioden aus gehen. Sie benötigen eine Mindestspannung um überhaupt zu leuchten. Deshalb muss die Helligkeitsänderung mittels schnellem ein- und ausschalten hintereinander realisiert werden. Dieses Verfahren nennt sich *Pulsweitenmodulation* (kurz: *PWM*). Je nachdem zu welchem Anteil die Dioden ein- bzw. ausgeschaltet werden, wird das Licht als heller bzw. dunkler empfunden. Das Diagramm 4.3 stellt dies grafisch dar.

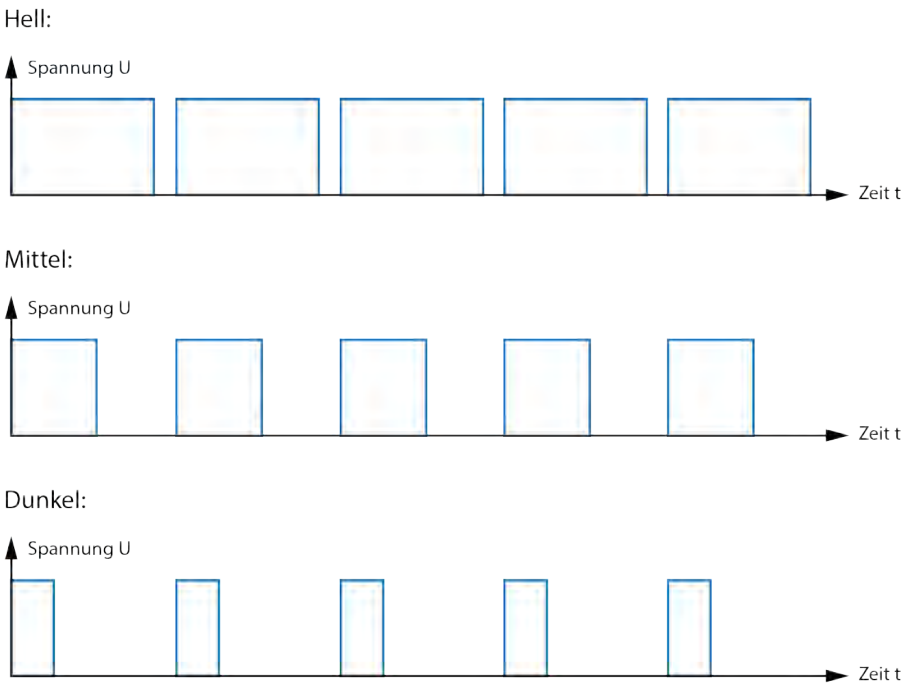

Abbildung 4.3: Pulsweitenmodulation

Die Realisierung der *Pulsweitenmodulation* übernimmt die entwickelte Software. Hierauf wird im Kapitel 4.3.1.2 noch genauer eingegangen.

Nun müssen die vier Kontaktflächen noch mit dem Client verbunden werden. Hierfür bietet der *Raspberry Pi* eine spezielle Schnittstelle. Diese GPIO-PINs (**G**eneral **P**urpose **I**nput **O**utput) bilden die zentrale Verbindungsmöglichkeit des *Raspberry Pis* mit externer Hardware. Insgesamt stehen 26 dieser PINs zur Verfügung, welche teilweise bereits vorbelegt sind. Wie die PINs mit den Kontaktflächen der Steuerung genau verbunden sind kann an Grafik 4.4 abgelesen werden.

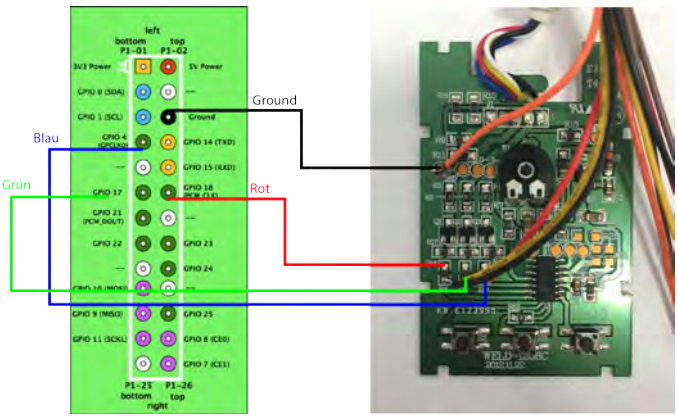

Abbildung 4.4: Pin-Belegung des Clients

4.3 Software

Die Architektur steht fest, die Hardware ist miteinander verbunden - nun müssen noch die Softwarebausteine entwickelt werden. Es werden insgesamt drei Komponenten benötigt. Betrachtet werden zunächst die beiden *Raspberry Pis*. Einer dient als Client, der andere als Server. Zuletzt wird auf die mobile Anwendung eingegangen.

Es wird aufgezeigt, wie die Komponenten von Softwareseite aus miteinander kommunizieren und welche Schritte notwendig sind, um die Farbe der Lichtleiste zu verändern. Außerdem werden verwendete Bibliotheken und Standards erläutert.

4.3.1 Client

Der Client ist zuständig für das Verändern der Farbwerte direkt an der Lichtleiste. Er empfängt vom Server die vom Anwender gewünschten Werte. Diese müssen aber erst interpretiert und in ein für die Steuerung der Lichtleiste verständliches Format übersetzt werden. Hierfür ist unter anderem die Umwandlung in *Pulsweitenmodulation* notwendig.

Außerdem bietet der Client einen Webserver mit REST-Schnittstelle an, um in einem standardisierten Format Informationen zu senden und zu empfangen.

4.3.1.1 Automatische Anmeldung beim Server

Sobald der Client gestartet wurde, sucht dieser nach einem Server im Netzwerk. Wurde der Server gefunden, meldet sich der Client bei ihm mit seinen spezifischen Informationen an. Von nun an weiß der Server, unter welcher Adresse und welchem Namen der Client erreichbar ist, und kann dies an die mobile Anwendung oder das Web-Frontend weiterleiten. In regelmäßigen Abständen prüft der Server, ob die angemeldeten Clients noch zur Verfügung stehen, und entfernt diese bei Bedarf. Das Sequenzdiagramm 4.5 stellt den Vorgang grafisch dar.

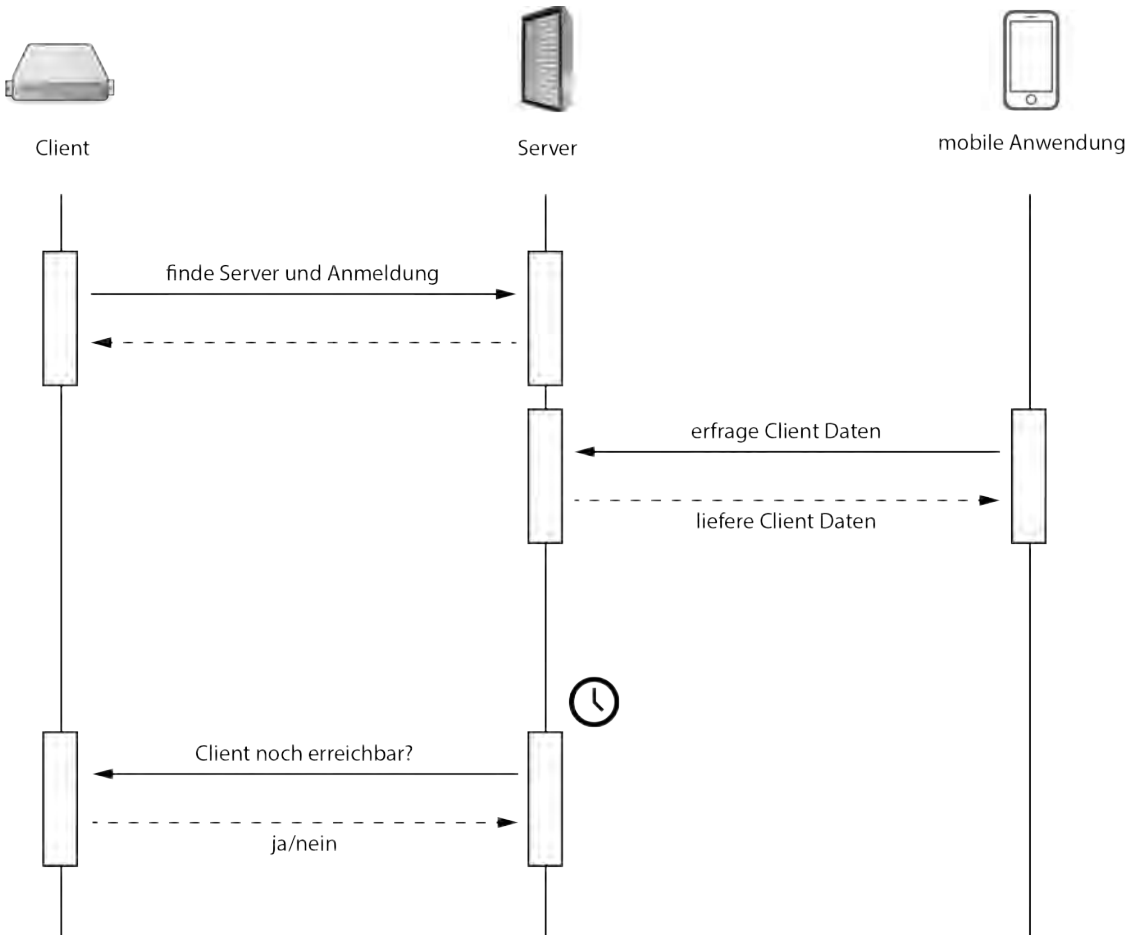

Abbildung 4.5: Sequenzdiagramm: Automatische Anmeldung des Clients am Server

4.3.1.2 Steuerung der Lichtleiste

Nachdem die Lichtleiste nun fest mit dem Client verbunden ist, müssen noch die notwendigen Informationen über die Ausgabe-PINs des *Raspberry Pis* an die Steuerung übertragen werden.

Entwickler haben die Möglichkeit über die Kommandozeile Werte wie logisch *0* oder logisch *1* an die verschiedenen GPIO-PINs anzulegen. Bestimmte PINs können jedoch nicht verändert werden. Sie sind vom Hersteller bereits vorbelegt.

Tabelle 4.1 zeigt, wie die PINs belegt wurden.

Pin-Belegung des Clients	
Blau	PIN 4
Grün	PIN 17
Rot	PIN 18
Ground	GND

Tabelle 4.1: Pin-Belegung des Clients

Der Client ist also bereits jetzt fähig die einzelnen Farben ein- bzw. auszuschalten. Das Verändern der Helligkeit ist allerdings noch nicht möglich. Hierfür ist ein schneller Wechsel zwischen *an* und *aus* in einem bestimmten Takt notwendig. Es ist wichtig, dass dieser Takt konstant bleibt, da die LEDs sonst flimmern würden.

pi-blaster

Um die *Pulsweitenmodulation* zu realisieren wurde die Bibliothek *pi-blaster* [SH13] verwendet. Sie stellt sicher, dass die CPU des *Raspberry Pis* nur gering belastet wird und der Ausgabepuls sehr stabil anliegt. Mit ihr kann über einen Befehl auf der Konsole an einen bestimmten Ausgabepin ein regelmäßiger Puls angelegt werden. Das Umschalten der *0* und *1* Werte geschieht dann automatisch und wird nicht verändert, bis ein anderer Puls angelegt wird.

Abbildung 4.6: Pulsweitenmodulation am Client

Folgende Konfiguration stellte sich für die Steuerung der Dioder als passend heraus:

```
Number of channels: 8
PWM frequency: 100 Hz
PWM steps: 1000
Maximum period (100 %) : 10000us
Minimum period (0.100%): 10us
```

Um einen solchen Puls festzulegen bietet *pi-blaster* einen Befehl der folgenden Struktur an (Listing: 4.1):

Listing 4.1: Aufbau eines Befehls für pi-blaster

```
1 echo "<Pin>=<Wert>" >/dev/pi-blaster
```

<Pin> steht hierbei für die Nummer des gewünschten PINs. Der *<Wert>* stellt eine Dezimalzahl im Wertebereich *[0,1]* dar. Zuletzt wird noch der Pfad zur Bibliothek angegeben, welcher sich nach Installation in *"/dev/pi-blaster"* befindet.

Beispiele für konkrete Werte dieser Infrastruktur wären also (Listing: 4.2):

Listing 4.2: Mögliche Befehle für pi-blaster

```
// Blau auf 100%
echo "4=1" >/dev/pi-blaster

// Gruen auf 0%
echo "17=0" >/dev/pi-blaster

// Rot auf 25%
echo "18=0.25" >/dev/pi-blaster
```

4.3.1.3 Webserver

Nachdem nun die gewünschten Farbwerte am Client manuell eingestellt werden können, stellt die Programmierung des Webservers den nächsten Schritt dar. Er nimmt die eingestellten Werte vom Server über seine REST-Schnittstelle entgegen, wandelt sie um, und gibt sie dann an die *pi-blaster*-Bibliothek weiter. Er muss ständig erreichbar sein ohne lange Ladezeiten aufzuweisen.

Der Webserver wurde in der Programmiersprache *Python* entwickelt und verwendet das Microframework *Flask* [Ric15]. Neben den Template-Dateien steckt die volle Funktionalität in der implementierten Datei *pyDioder.py*. Sie wird ständig im Hintergrund ausgeführt und nimmt Anfragen entgegen. Sobald eine solche Anfrage empfangen wird, werden die gewünschten Daten extrahiert und verarbeitet. Die Farbwerte werden berechnet und an die jeweiligen Ausgabepins weitergeleitet.

Der Server ist unter der automatisch zugeteilten System-IP und dem Webserver-Port *80* erreichbar. Eine Konfiguration des Anwenders wird hierfür nicht benötigt.

Flask

Flask kommuniziert über eine WSGI-Schnittstelle zwischen Webserver und Webanwendung [Ron15]. Es stellt grundlegende Serverfunktionalitäten zur Verfügung und bietet ein breites Spektrum an Erweiterungen. Eine Besonderheit von *Flask* stellt außerdem die gute Dokumentation dar.

Des Weiteren bietet *Flask* eine REST-Schnittstelle an, die im nächsten Abschnitt genauer erklärt wird.

Die entwickelten Webserverkomponenten bauen komplett auf *Flask* auf, wodurch dieses als Grundlage der Serverkomponenten angesehen werden kann.

REST-Schnittstelle

Über die REST-Schnittstelle können in einem weitgehend standardisierten Format Daten zwischen Anfragendem und Server ausgetauscht werden. Hierbei werden verschiedene Methoden zur Verfügung gestellt. Dazu gehören *GET*- und *POST*-Anfragen. Über die URL wird dann die

gewünschte Funktion aufgerufen.

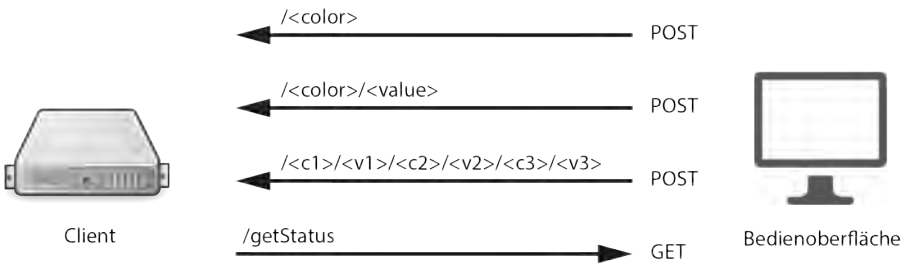

Abbildung 4.7: REST-Schnittstelle des Client

Der Client implementiert folgende Anfragen mit den entsprechenden Funktionalitäten. Alle Aufrufe verwenden hierbei das HTT-Protokoll.

URL/<color>:

```
Mögliche Werte:
<color>: white/black
```

Alle Farben werden entweder mit voller Helligkeit ein- oder ganz ausgeschaltet. Hierzu werden die Werte auf 100% bzw. 0% gesetzt.

URL/<color>/<value>:

```
Mögliche Werte:
<color>: red/green/blue
<value>: [0,255]
```

Eine der drei Farben Rot, Blau oder Grün wird auf den gewünschten Wert zwischen 0 und 255 gesetzt. Der Wert wird hierfür zunächst auf den jeweiligen Prozentsatz umgerechnet, wobei 255 voller Leuchtkraft (100%) entspricht.

URL/<c1>/<v1>/<c2>/<v2>/<c3>/<v3>:

```
Mögliche Werte:
<c1>/<c2>/<c3>: red/green/blue
<v1>/<v2>/<v3>: [0,255]
```

Alle drei Farben werden mit nur einem Seitenaufruf gleichzeitig gesetzt. Hierbei bekommt die Farbe *<c1>* den Wert *<v1>*. Analog dazu werden die anderen Farbwerte eingestellt. Die Reihenfolge der Farben spielt dabei keine Rolle. Da nicht alle drei Ausgabepins gleichzeitig gesetzt werden können, werden die Befehle vom Webserver hintereinander ausgeführt. Die dadurch

entstehende Verzögerung ist so gering, dass diese nicht wahrgenommen werden kann.

URL/getStatus:

```
Mögliche Werte:
keine
```

Hiermit kann die Erreichbarkeit des Clients überprüft werden. Sofern dieser reagiert, wird ein “OK“-Status zurückgegeben. Erfolgt keine Antwort des Clients kann der Server diesen aus der Liste seiner gespeicherten Lichtleisten entfernen.

4.3.2 Server

Der Server dient als zentraler Knotenpunkt zwischen den Clients und der Benutzeroberfläche beim Anwender. Er kommuniziert laufend mit den Clients und stellt deren Verfügbarkeit sicher - ähnlich der *HeartBeat*-Technologie. Außerdem verteilt er die vom Benutzer eingestellten Werte an die jeweiligen Endpunkte.

Gleichzeitig stellt er ein Web-Frontend zur Verfügung durch das der Benutzer Änderungen an den Farbwerten vornehmen kann. Dieses fasst alle vorhandenen Clients zusammen und zeigt sie auf einer einzigen Seite übersichtlich an.

Des Weiteren bietet der Server im Gegensatz zum Client Informationen im JSON-Format an. Diese werden von der mobilen Anwendung benötigt, um die verfügbaren Clientdaten mit deren jeweiligen Einstellungsmöglichkeiten zu erhalten.

Ein Synchronisationsmodul bildet die gespeicherten Daten auf der Oberfläche ab. Auch Änderungen des Anwenders werden hierdurch im internen Speicher hinterlegt.

Die Abbildung 4.8 zeigt die Softwarearchitektur des Servers nach dem MVC-Prinzip.

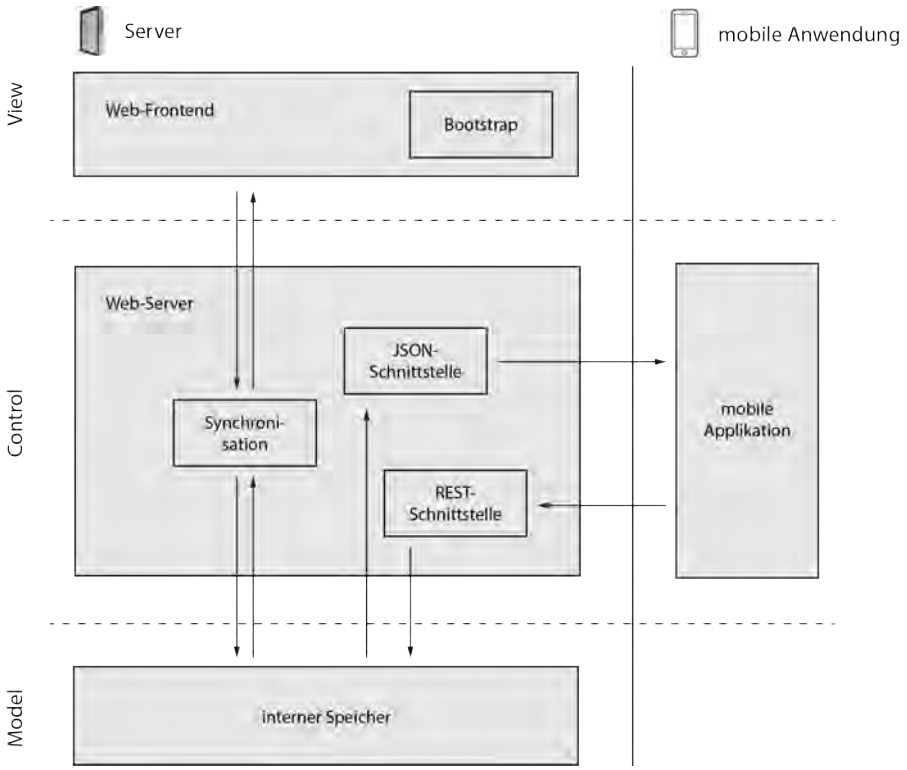

Abbildung 4.8: MVC-Modell des Servers

4.3.2.1 Webserver

Wie der Client auch, stellt der Server einen Webserver zur Verfügung. Die verwendeten Bibliotheken und Schnittstellen sind die selben. Auch hier wurde *Python* als Programmiersprache verwendet und eine REST-Schnittstelle sendet und empfängt Daten. Die Erreichbarkeit wird ebenfalls auf Port Nummer *80* unter der System-IP gewährleistet.

Die Webserver-Komponente des Knotenpunkts muss allerdings andere Aufgaben erfüllen als die eines Clients. Daher sehen sowohl die *pyDioder.py*-Datei, als auch die Template-Dateien anders aus. In der *pyDioder.py* steckt auch hier die Funktionalität. Sie nimmt Anfragen entgegen und reagiert entsprechend. Deshalb ist die ständige Erreichbarkeit von großer Bedeutung. Die Template-Dateien sorgen für die korrekte und benutzerfreundliche Darstellung der angebotenen Funktionen auf der Weboberfläche.

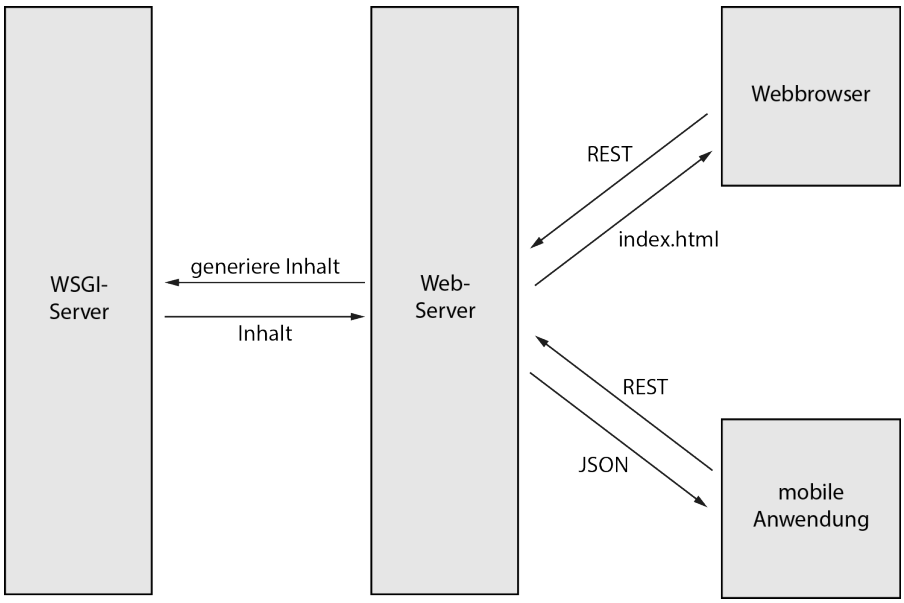

Abbildung 4.9: Web-Server des Servers

Flask

Neben den bereits bekannten Funktionen von *Flask* bietet der Server zusätzlich eine JSON-Komponente. Diese ist dafür zuständig in einem standardisierten Format Informationen anzubieten. Sie werden nach einem bestimmten Muster formatiert und sind daher auch für den Menschen einfach lesbar.

JSON

Damit die mobile Anwendung und eventuell auch andere Benutzeroberflächen wissen, welche Clients zur Verfügung stehen und welche Funktionen diese bieten, stellt der Server diese Informationen im JSON-Format unter einer bestimmten Adresse zur Verfügung. Die Bedienoberfläche kann diese Datei parsen und in ihre eigene Datenhaltung integrieren.

REST-Schnittstelle

Der Server bietet folgende Einstiegspunkte für den Datenaustausch an. Auch hier wird das HTT-Protokoll verwendet.

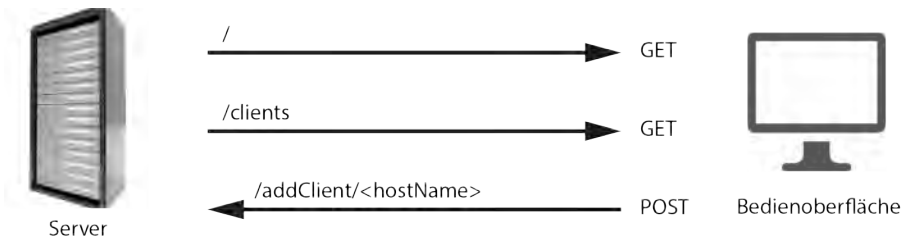

Abbildung 4.10: REST-Schnittstelle des Servers

URL/:

```
Mögliche Werte:
keine
```

Sobald die Startseite des Servers aufgerufen wird, werden die Template-Dateien generiert. Die hinterlegten Informationen zu den Clients werden aufbereitet und an die HTML-Datei übertragen. Diese stellt dann die vorhandenen Funktionen und deren Bedienelemente übersichtlich dar. Für die mobile Anwendung wird diese Seite nicht benötigt.

URL/clients:

```
Mögliche Werte:
keine
```

Die Informationen zu den Clients werden in das JSON-Format umgewandelt und an die anfragende Anwendung übermittelt.

URL/addClient/<hostName>:

```
Mögliche Werte:
<hostName>: Name des jeweiligen Hosts
```

Über diese Adresse können sich Clients mit ihren Hostnamen beim Server registrieren. Die IP-Adresse des Clients bezieht der Server über die Anfrage selbst.

Web-Frontend

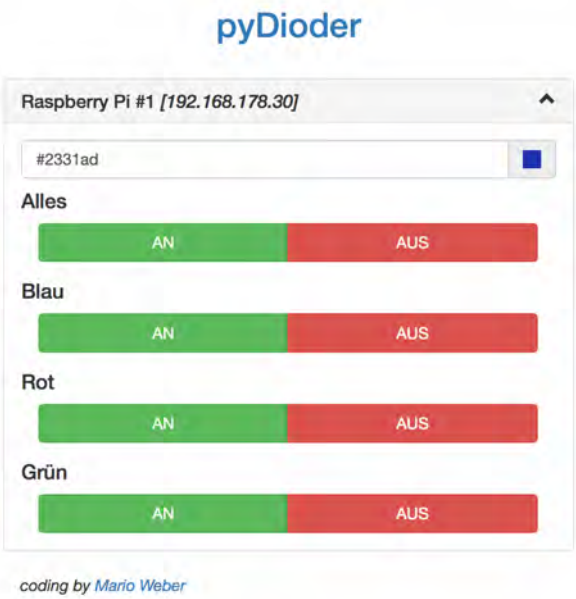

Abbildung 4.11: Screenshot: Web-Frontend des Servers

Im Gegensatz zum Client bietet der Server zusätzlich ein Web-Frontend an. Mit Hilfe dieser können die Lichtleisten gesteuert werden, ohne dass die mobile Anwendung benötigt wird. Sie wurde ebenfalls entwickelt, da nicht jeder Anwender ein Smartphone besitzt. Außerdem kann es vorkommen, dass dessen Akku leer ist. Die Oberfläche kann mit jedem Web-Browser geöffnet werden. Sie passt sich in der Darstellung an das jeweilige Endgerät an, um ein besseres Nutzungserlebnis zu bieten. Hierfür wurde das Responsive-Framework *Bootstrap* [Boo15a] integriert.

Die Bedienelemente werden nach Client gruppiert dargestellt. Diese Gruppen können zugeklappt werden, um eventuell nicht benötigte Funktionen auszublenden.

Da beispielsweise Wisch-Gesten über den Browser nicht möglich sind, stellt das Web-Frontend eine deutlich funktionsärmere Alternative zur mobilen Anwendung dar.

Bootstrap

Bootstrap ist das meist gefragte Framework um “Mobile-First-Projekte“ zu realisieren [Boo15a]. Es ist Open-Source und für jeden kostenlos herunterzuladen. Bereits Millionen von Webseiten verwenden *Bootstrap*. Die Gründe hierfür sind naheliegend: Es ist äußerst effizient, einfach zu integrieren und bietet eine Anpassung der Website auf das jeweilige Endgerät ohne dabei mehrere Versionen der Seiten erstellen zu müssen.

Des Weiteren wird dem Entwickler eine Vielzahl an Designvorlagen angeboten, mit denen die Website deutlich schöner dargestellt wird. Hierfür müssen den Elementen nur die entsprechenden CSS-Klassen zugewiesen werden.

Auch optionale Java-Script-Erweiterungen, um beispielsweise auf Events reagieren zu können, bietet *Bootstrap* an.

4.3.3 Mobile Anwendung

Um das System bedienen zu können, wurde neben dem Web-Frontend auch eine *iOS*-Applikation entwickelt. Sie steht in Kontakt mit dem Server und bildet damit die Benutzeroberfläche des Systems. Alle Änderungen, die der Nutzer an den Farbwerten vornimmt, werden von der mobilen Anwendung an den Server übertragen, der diese dann an den richtigen Client weitergibt. Außerdem bezieht die Applikation alle Informationen der verfügbaren Clients über die JSON-Schnittstelle des Servers.

Da die mobile Applikation als eigentliche Benutzeroberfläche dient und das Web-Frontend lediglich eine Notlösung darstellen soll, falls die mobile Anwendung einmal nicht verwendet werden kann, liegt hier die Aufmerksamkeit vor allem bei der Bedienbarkeit und Performanz [RW13]. Aus diesen Gründen wurde die Anwendung nativ, also nur für *Apples* mobiles Betriebssystem *iOS*, entwickelt. Hybride Applikationen bieten zwar den Vorteil der plattformübergreifenden Verwendungsmöglichkeit, können dafür aber kaum mit der Geschwindigkeit und Funktionsunterstützung nativ entwickelter Software mithalten.

Dieser Abschnitt beschreibt zunächst die Architektur der Anwendung, gefolgt von den Besonderheiten und der Funktionsweise der *iOS*-Entwicklung. Es wird auf das *Apple-Developer-Program* und die Programmiersprache *Swift* eingegangen. Darauf folgt eine Erklärung der einzelnen Bildschirme und deren Zweck.

4.3.3.1 Software-Architektur

Abbildung 4.12 zeigt den Aufbau der einzelnen Softwarebausteine der mobilen Anwendung.

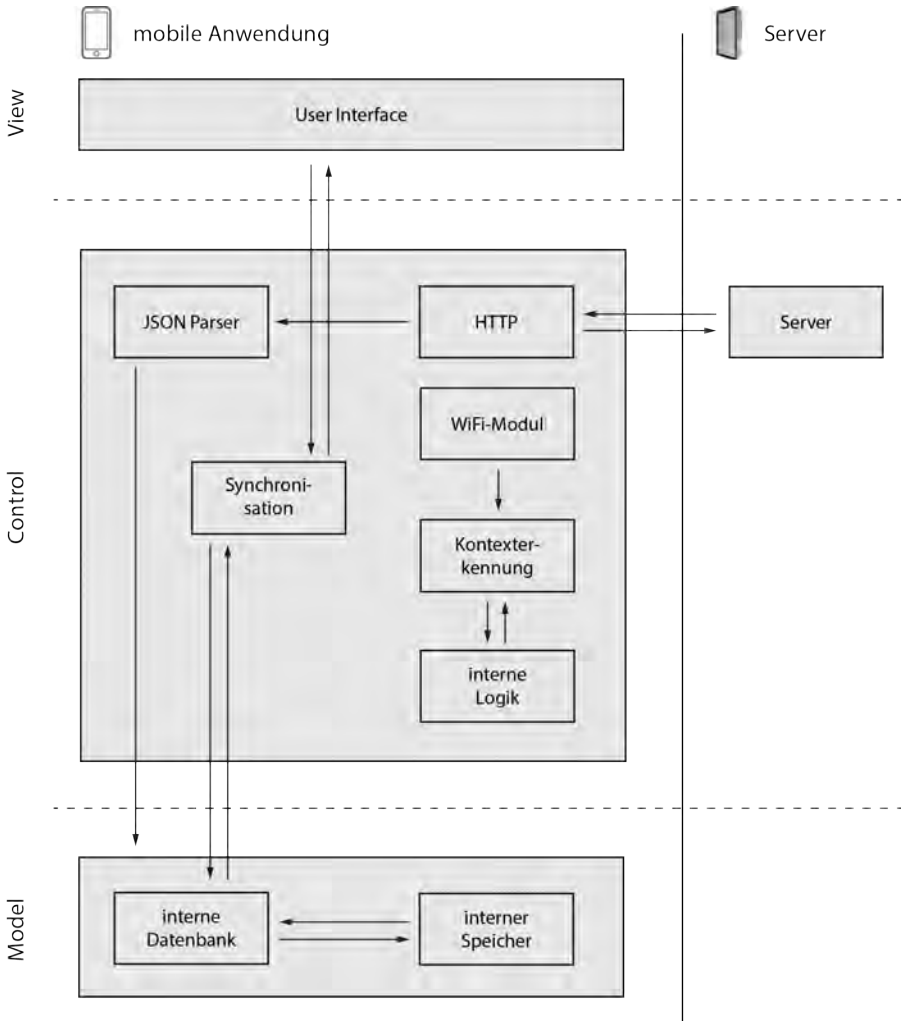

Abbildung 4.12: MVC-Modell der mobilen Anwendung

Mittels HTT-Protokoll empfängt die Anwendung Daten vom Server. Diese werden im JSON-Format übermittelt und müssen zunächst vom integrierten Parser interpretiert werden. Die Datenhaltung speichert die Werte dann in einem integrierten Zwischenspeicher, auf den schnell zugegriffen werden kann. Dadurch wird eine flüssige Bedienung der Anwendung sichergestellt. Da die Daten nach einem Neustart des Smartphones nicht mehr im Zwischenspeicher liegen,

müssen diese außerdem in einer Datenbank hinterlegt werden. Datenbank und Zwischenspeicher werden laufend untereinander abgeglichen. Ein Synchronisationsmodul bildet die hinterlegten Informationen der Datenhaltung auf der Oberfläche ab. Außerdem werden Änderungen auf dem User Interface, wie das Löschen eines hinterlegten Servers, in den Speicher übernommen.

Die interne Logik der Applikation ist unter Anderem zuständig für das Finden von Servern im gleichen Netzwerk. Sie ist eng mit einer Kontexterkennung verbunden, die auf Umgebungseigenschaften reagiert. So haben beispielsweise Änderungen der WiFi-Konnektivität Auswirkungen auf die Funktionsweise der Anwendung. Dadurch kann die Performanz und Bedienbarkeit erheblich verbessert werden.

4.3.3.2 Apple-Developer-Program

Um Applikationen für Geräte der Marke *Apple* zu entwickeln und später im jeweiligen Store zu veröffentlichen, wird eine Mitgliedschaft im *Apple-Developer-Program* vorausgesetzt [App15b]. Dazu muss sich der Entwickler registrieren und wird dann nach einiger Zeit freigeschaltet.

Mitglieder des *Developer-Programs* genießen einige Vorteile wie beispielsweise den Zugriff auf Beta-Software. So können sie schon vor der Markteinführung neuer Technologien und Funktionen die Anwendungen dafür optimieren und auf ihren Geräten testen. Aber auch komplexe Analyse-Werkzeuge, wie Statistiken über die Downloads der Anwendungen werden den Entwicklern geboten. Außerdem werden keine Hosting-Gebühren für das Speichern der Applikationen im Store berechnet. Wird die entwickelte Software kostenpflichtig angeboten, bekommt der Entwickler 70% des Verkaufserlöses. Auch reine B2B-Anwendungen (*Business-to-Business*) sind möglich.

Entwickelt werden kann in den beiden Programmiersprachen *Objective-C* und *Swift*, wobei letztere die neuere Variante darstellt. Hierfür muss *XCode*, welches nur für *Apple*-Betriebssysteme angeboten wird, als Entwicklungsumgebung verwendet werden.

Neben den selbst implementierten Funktionen der Anwendung haben die Entwickler Zugriff auf einige Schnittstellen, die *Apple* anbietet. So kann mit wenigen Zeilen Code eine Verbindung zu *Apple-Pay*, *Wallet*, *Game-Center* und vielen mehr hergestellt werden. Auch In-App-Käufe über den Store und Speicherung von Daten in *iCloud* sind möglich. Dies ist besonders hilfreich, wenn Daten der Anwendung auf mehreren Geräten synchronisiert werden sollen. Die Entwickler müssen sich dann nicht erst um einen eigenen Cloud-Server bemühen, da die meisten Nutzer ohnehin *iCloud*, welche bis zu einem bestimmten Speichervolumen kostenlos angeboten wird, verwenden.

4.3.3.3 Entwicklung mit XCode

Um die entwickelte Anwendung nicht nur im Simulator von *XCode*, sondern auch auf dem eigenen Gerät testen zu können, wird zunächst ein Entwicklungszertifikat benötigt. Dieses ist

notwendig, um Bereitstellungsprofile erzeugen zu können, mit denen die Anwendungen dann zertifiziert und auf dem Gerät installiert werden können. Das Entwicklungszertifikat wird über ein entsprechendes Dokument, welches innerhalb der *Schlüsselbundverwaltung* auf dem *Mac* erzeugt wird, mittels Zertifikatsanfrage, angefordert. Wurde die erzeugte Datei im *Provisioning Portal* hochgeladen und akzeptiert, kann das Zertifikat aus dem Entwicklerportal heruntergeladen werden. Dieses ist dann ein Jahr gültig und muss in den Anmeldeschlüsselbund auf dem *Mac* importiert werden.

Nun muss das jeweilige Endgerät in *XCode* für die Entwicklung angemeldet werden. Dieses wird dann im Portal registriert und ein *Team Provisioning Profile* wird erstellt. Danach kann die Applikation auf dem Gerät ausgeführt werden.

Beim Start der Anwendung via *XCode* wird diese im Debug-Modus ausgeführt. Der Entwickler hat so die Möglichkeit Fehlercodes einzusehen, Ausgaben auf der Konsole zu erstellen und Breakpoints, also Punkte, an denen die Anwendung stoppen soll, zu setzen. Hierfür muss das Gerät mit dem *Mac* via USB-Kabel verbunden sein. Außerdem können auf dem Gerät hardwareseitige Funktionen, wie die Kamera oder Gyro-Sensoren getestet werden.

Wurden diese Hürden überwunden, kann mit der eigentlichen Programmierung begonnen werden.

4.3.3.4 Layout

Um eine einfache und intuitive Bedienung zu gewährleisten, wurde die mobile Anwendung vollständig mit *Apples* Bordwerkzeugen umgesetzt. Alle Bedienelemente, die verwendet wurden, können auch in anderen Applikationen gefunden werden. Die Funktionen dieser Elemente sollten sich im Idealfall kaum unterscheiden und nur für den aktuell benötigten Zweck angepasst werden. Es ist beispielsweise nicht ratsam die Navigation durch die Applikation mittels eigener Buttons zu realisieren. Hierfür werden eigene Elemente in der *Top-Bar* angeboten. Halten sich alle Entwickler an diese Standards, finden sich Nutzer in einer neuen Applikation sofort zurecht und benötigen keine weitere Einarbeitungsphase.

Abbildung 4.13 zeigt das Storyboard. Anhand diesem kann abgelesen werden, welche Bildschirme mit welchen Schritten erreicht werden können. Um besser erkennen zu können, welche Informationen auf den einzelnen Seiten angezeigt werden, wurde die Applikation mit einigen Testwerten gefüllt.

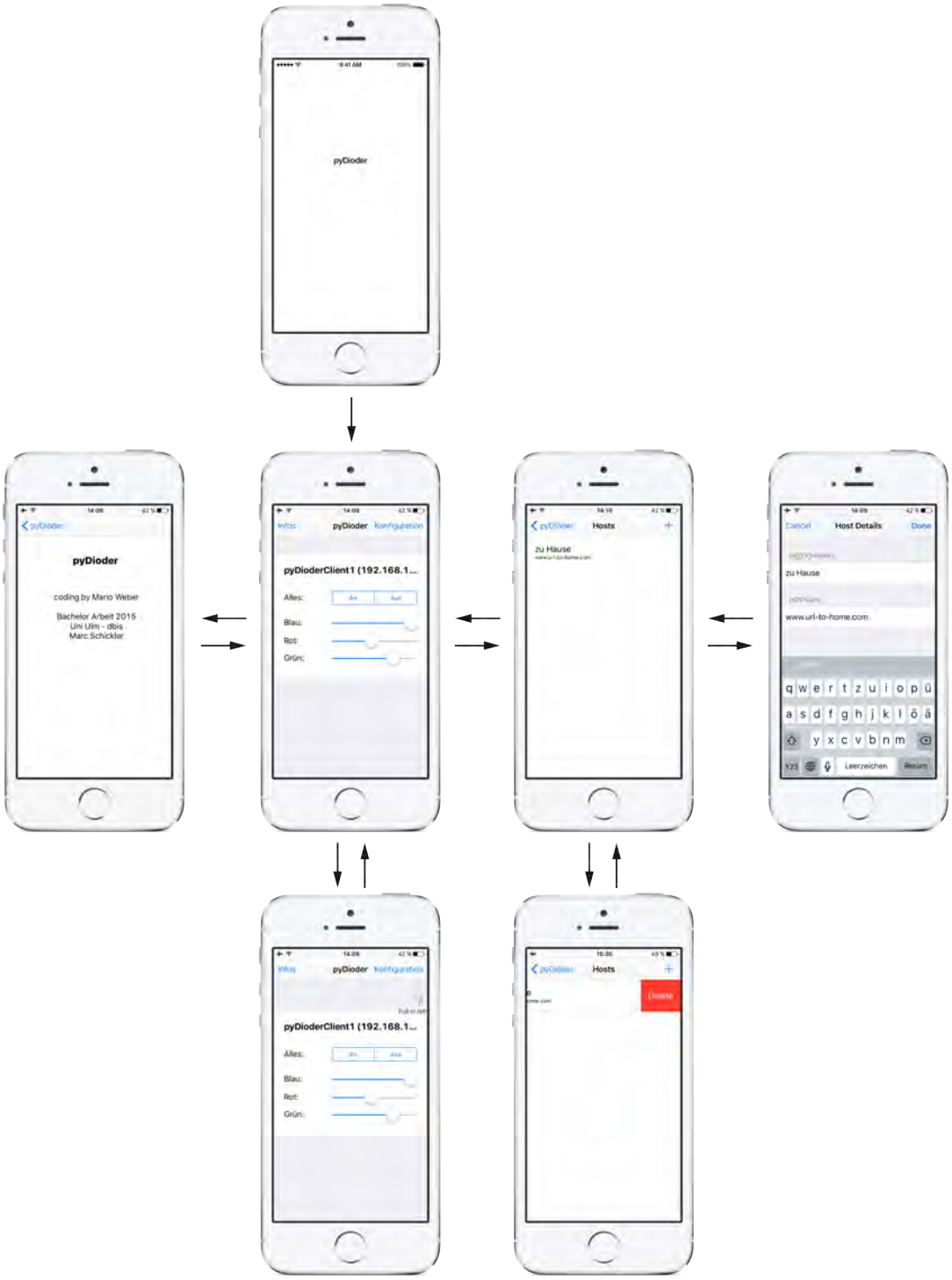

Abbildung 4.13: Storyboard der mobilen Anwendung

Ladebildschirm und Infobildschirm

Abbildung 4.14: Screenshot: Ladebildschirm und Infobildschirm

Abbildung 4.14 zeigt den Bildschirm, sobald die Applikation gestartet wird. Er enthält den Namen der Anwendung. Während dieser Bildschirm dargestellt wird, werden alle Elemente der Applikation in den Arbeitsspeicher geladen, sodass danach eine zügige Bedienung ohne lange Wartezeiten möglich ist.

Außerdem werden im Hintergrund alle Server kontaktiert, deren Adressen in den Einstellungen hinterlegt wurden. Die Informationen der erreichbaren Clients werden via JSON-Schnittstelle heruntergeladen und in einen Zwischenspeicher der Anwendung importiert. Befindet sich der Nutzer im WLAN, werden zusätzlich Server innerhalb des gleichen Netzwerks gesucht - auch wenn diese nicht konfiguriert wurden. Deren Client-Informationen werden dann ebenfalls heruntergeladen und gespeichert.

Sobald alle Elemente geladen wurden, wird zum Startbildschirm (Abbildung 4.15) gesprungen.

Vom Startbildschirm aus kann zum Infobildschirm gelangt werden. Dieser Bildschirm dient lediglich der Information, um welche Anwendung es sich handelt, und zu welchem Zweck sie entworfen wurde. Des Weiteren enthält er einen Verweis auf diese Arbeit und das DBIS-Institut der Universität Ulm.

Startbildschirm

Abbildung 4.15: Screenshot: Startbildschirm

Der Startbildschirm (Abbildung 4.15) zeigt alle Informationen zu den verfügbaren Clients in einer Tabelle an. Jeder Client wird als einzelner Eintrag dargestellt. Innerhalb der jeweiligen Zelle werden Name, Adresse und die verfügbaren Bedienelemente gezeigt. Der Nutzer hat die Möglichkeit alle Farben mit voller Leuchtkraft ein-, oder komplett auszuschalten. Dies wird über eine sogenannte *Segmented-Control* realisiert. Außerdem können die drei Grundfarben Rot, Grün und Blau einzeln in ihrer Leuchtkraft über einen Schieberegler verstellt werden. Jede Tabellenzeile wird dynamisch während der Laufzeit generiert und ist abhängig von den verfügbaren Clients und deren jeweiligen Bedienelementen.

In der *Top-Bar* befinden sich außerdem zwei Navigationsbuttons. Der Linke führt zum Informationsbildschirm, der Rechte zu den gespeicherten Servern.

Durch eine *Pull*-Geste, also ein “Ziehen“ der Tabelleneinträge von oben nach unten, werden die Zellen aktualisiert. Ähnlich dem Ladebildschirm werden alle Server erneut kontaktiert und deren Clientinformationen heruntergeladen. Clients, die nicht mehr verfügbar sind, werden aus der Tabelle entfernt.

Werden keine verfügbaren Server gefunden, beispielsweise weil keine Geräte in den Einstellungen konfiguriert wurden und sich das Smartphone nicht im WLAN befindet, bekommt der Nutzer eine Message-Box mit einem Hinweis angezeigt. Er hat dann die Möglichkeiten erneut nach Servern zu suchen, oder die Box zu schließen und damit auf den Startbildschirm ohne Tabelleneinträge zu gelangen.

Liste der gespeicherten Server

Abbildung 4.16: Screenshot: Liste der gespeicherten Server

Über den *Konfigurieren*-Button in der *Top-Bar* des Startbildschirms gelangt der Nutzer zu dieser Ansicht (Abbildung 4.16). Hier werden ihm alle Server angezeigt, die er zuvor gespeichert hat. Zur schnellen Identifikation, um welche Server es sich hierbei handelt, werden deren selbst vergebener Name und Adresse dargestellt. Jede Tabellenzelle entspricht einem Server.

In der *Top-Bar* stehen dem Nutzer ebenfalls zwei Navigationsbuttons zur Verfügung. Tippt der Anwender auf “pyDioder“, gelangt er zurück zum Startbildschirm. Mit dem “+“-Button kann er neue Server in die Liste aufnehmen. Hierfür wird ihm der Bildschirm “Hinzufügen“ angezeigt.

Mittels einer Wisch-Geste auf einem Tabelleneintrag von rechts nach links, kann dieser gelöscht werden. Er wird dann auch aus der internen Datenbank entfernt. Zur Sicherheit muss der Nutzer die Geste mittels dem erscheinenden “Delete“-Button bestätigen.

Server hinzufügen

Abbildung 4.17: Screenshot: Server hinzufügen

Möchte der Anwender Clients via Internet bedienen, müssen deren Server zunächst in der Applikation angelegt werden (Abbildung 4.17). Dazu ist zunächst die Adresse, unter welcher der Server kontaktiert werden kann, einzugeben. Der Nutzer hat die Wahl entweder eine sprechende URL oder direkt die nach außen sichtbare IP-Adresse zu speichern. Auch eine lokale IP-Adresse wäre möglich, ist aber nicht notwendig, da Server, die sich im selben Netzwerk befinden, ohnehin ohne Konfiguration gefunden werden.

Um die gespeicherten Server besser identifizieren zu können, hat der Anwender außerdem die Möglichkeit eine beliebige Bezeichnung festzulegen. Diese wird dann auch im Bildschirm “Liste der gespeicherten Server“ (Abbildung: 4.16) angezeigt.

In der *Top-Bar* kann der Vorgang abgebrochen, oder die Konfiguration übernommen werden. Die eingegebenen Werte werden dann in einer internen Datenbank gespeichert, sodass diese auch nach einem Systemabsturz oder dem vollständigen Schließen der Anwendung verfügbar sind. Danach gelangt der Nutzer wieder zur “Liste der gespeicherten Server“, in welcher die zuvor vorgenommenen Änderungen aufgenommen werden.

5

Anforderungsabgleich

Die nachfolgenden Tabellen zeigen, wie gut die Anforderungen umgesetzt wurden. “1“ bedeutet hierbei “sehr gut“ und “6“ “ungenügend“ - ähnlich dem Schulnotensystem. Um die Bewertungen besser nachvollziehen zu können, werden die einzelnen Punkte nochmals genauer erläutert.

5.1 Funktionale Anforderungen

Die funktionalen Anforderungen wurden alle erfüllt. Zum Teil weisen diese allerdings kleine Schwächen auf. Sie werden im Folgenden genauer erklärt.

Mobile Anwendung

Mobile Anwendung	**1**	**2**	**3**	**4**	**5**	**6**
Autonome Konfiguration		✓				
Anpassung an Umgebung	✓					
Mit Server via Internet verbinden	✓					
Speicherung der Konfiguration in Datenbank	✓					
Manuelle Aktualisierung		✓				

Tabelle 5.1: Funktionale Anforderungen der mobilen Anwendung

FA #1: Autonome Konfiguration

Die Applikation ist in der Lage den Server im Netzwerk selbständig zu finden, sofern es einen gibt. Allerdings kommt es in einzelnen Fällen bei bestimmten Routern zu Problemen. Dort sind Sicherheitseinstellungen in der Firmware hinterlegt, die einen solchen Zugriff nicht gestatten. Dieses Problem ist nur schwer bis gar nicht lösbar.

Bewertung: gut

FA #2: Anpassung an Umgebung

Diese Funktion wurde vollständig implementiert und funktioniert ohne Einschränkungen. Sie ist daher mit der Bestnote zu versehen.

Bewertung: sehr gut

FA #3: Mit Server via Internet verbinden

Der Nutzer hat die Möglichkeit beliebig viele Server in den Einstellungen zu hinterlegen, um diese via Internet steuern zu können. Das Übernehmen der vorgenommenen Änderungen funktioniert ohne erkennbare Verzögerung.

Bewertung: sehr gut

FA #4: Speicherung der Konfiguration in Datenbank

Sobald der Nutzer Einstellungen hinterlegt, werden diese in einer internen Datenbank von *iOS* hinterlegt. Wird die Anwendung also unerwartet beendet oder manuell geschlossen, stehen dem Anwender die vorgenommenen Konfigurationen weiterhin zur Verfügung.

Bewertung: sehr gut

FA #5: Manuelle Aktualisierung

Die Anwendung bietet im Startfenster eine “pull-to-refresh“-Funktionalität. Der Nutzer kann also mit einem “Ziehen“ der Tabelle von oben nach unten eine manuelle Aktualisierung der Einträge erzwingen. Hierbei werden allerdings keine neuen Clients im Netzwerk gesucht, sondern die beim Server hinterlegten Clients erneut abgerufen. Außerdem werden die in den Einstellungen gespeicherten Server erneut kontaktiert.

Bewertung: gut

Server

Server	1	2	3	4	5	6
Selbständige Einbindung ins Netzwerk		✓				
Plug & Play		✓				
REST-Schnittstelle und JSON	✓					
Optionaler Zugriff via Internet		✓				
Webinterface		✓				

Tabelle 5.2: Funktionale Anforderungen des Servers

FA #6: Selbständige Einbindung ins Netzwerk

Sofern der Server per Kabel oder über dLAN mit dem Router verbunden wird, kann sich dieser vollständig selbst konfigurieren. Für eine Verbindung via WLAN sind weitere Sicherheitseinstellungen Voraussetzung für die Einbindung ins Netzwerk.

Sollten sich im Netzwerk mehrere Server befinden, könnten sich diese gegenseitig behindern. Dies ist jedoch zu vernachlässigen, da mehr als ein Server nicht benötigt wird und daher unnötig ist.

Bewertung: gut

FA #7: Plug & Play

Sobald der Server eingesteckt und mit dem Netzwerk verbunden ist, wird das Betriebssystem gestartet. Ist dieses hochgefahren, werden alle Dienste gestartet, die für die Verwendung als zentraler Knotenpunkt benötigt werden.

Unter sehr speziellen Voraussetzungen kann es vorkommen, dass der Server neu gestartet werden muss, um wieder fehlerfrei zu funktionieren. Hierfür reicht ein einfaches Ein- und Ausstecken.

Bewertung: gut

FA #8: REST-Schnittstelle und JSON

Die komplette Steuerung des Servers findet über eine REST-Schnittstelle statt. Informationen werden im JSON-Format angeboten. Die Anforderung wurde also erfüllt.

Bewertung: sehr gut

FA #9: Optionaler Zugriff via Internet

Sofern die benötigten Einstellungen im Konfigurationsmenü des Routers vorgenommen wurden, ist der Zugriff über das Internet möglich. Um diese Funktion noch weiter zu optimieren, wäre eine Sicherheitsabfrage mit Login denkbar.

Bewertung: gut

FA #10: Webinterface

Der Nutzer hat die Möglichkeit die Lichtsteuerung auch via Webinterface zu bedienen. Dieses bietet aber nur eingeschränkte Funktionen im Gegensatz zur mobilen Anwendung, da hier beispielsweise keine Gesten möglich sind. Allerdings ist die Weboberfläche auch nur als Notlösung gedacht. Im Vordergrund steht weiterhin die Bedienung per Smartphone-Applikation.

Bewertung: gut

Client

Client	1	2	3	4	5	6
Selbständige Einbindung ins Netzwerk		✓				
Eigenständiges Finden des Servers		✓				
Plug & Play		✓				
REST-Schnittstelle und JSON	✓					

Tabelle 5.3: Funktionale Anforderungen des Clients

FA #11: Selbständige Einbindung ins Netzwerk

Für den Client gelten die gleichen Aspekte bezüglich der selbständigen Einbindung ins Netzwerk wie für den Server.

Eine gegenseitige Behinderung mehrerer Clients ist nicht vorhanden.

Bewertung: gut

FA #12: Eigenständiges Finden des Servers

Sobald der Client gestartet wurde, meldet er sich bei einem Server in der Umgebung an. Dazu muss dieser zunächst im Netzwerk gesucht werden. In einzelnen Fällen kann es vorkommen, dass kein Server gefunden wird, und der Client neu gestartet werden muss.

Bewertung: gut

FA #13: Plug & Play

Der Client fährt nach dem Einstecken selbständig hoch und startet alle benötigten Dienste.

Wie auch beim Server kann es vorkommen, dass der Client neu gestartet werden muss.

Bewertung: gut

FA #14: REST-Schnittstelle und JSON

Diese Anforderung wurde, wie auch beim Server, vollständig erfüllt.

Bewertung: sehr gut

5.2 Nicht-Funktionale Anforderungen

Auch die nicht-funktionalen Anforderungen wurden alle umgesetzt. Bezüglich der Performanz und Skalierbarkeit gibt es kleine Verbesserungsmöglichkeiten. Die Benutzbarkeit und Wartbarkeit wurden dagegen besonders zufriedenstellend realisiert.

Mobile Anwendung

Mobile Anwendung	1	2	3	4	5	6
Benutzbarkeit und Verständlichkeit	✓					
Performanz und Effizienz		✓				
Wartbarkeit und Erweiterbarkeit	✓					
Skalierbarkeit		✓				

Tabelle 5.4: Nicht-funktionale Anforderungen der mobilen Anwendung

NFA #1: Benutzbarkeit und Verständlichkeit

Die mobile Anwendung ist nicht überladen und leicht verständlich. Es ist keine große Einarbeitungszeit notwendig bis der Anwender die Funktionsweise versteht. Um dies zu gewährleisten, wurden *Apples* Bedienelemente stets wie vorgesehen eingesetzt.

Bewertung: sehr gut

NFA #2: Performanz und Effizienz

Die Applikation startet schnell und ist auf das aktuelle Umfeld optimiert. Befindet sich der Nutzer beispielsweise nicht im WLAN, werden keine Server gesucht. Lediglich die Geräte mit Internetzugriff werden konnektiert. Wird die Anwendung mit WLAN-Verbindung gestartet, dauert es einen kurzen Moment bis der Server im Netzwerk, falls vorhanden, gefunden wird. Dies ist allerdings kaum zu vermeiden, da keine genaue Adresse vorliegt. Die Applikation muss daher alle Geräte im Netzwerk mit einem potentiellen Server abgleichen.

Alle Oberflächen sind schnell und ohne Verzögerung erreichbar.

Bewertung: gut

NFA #3: Wartbarkeit und Erweiterbarkeit

Sollen weitere Clients mit anderen Funktionen eingesetzt werden, kann die Anwendung einfach mit deren Bedienelementen ausgestattet werden. Der grundlegende Aufbau der Applikation muss hierfür nicht geändert werden.

Bewertung: sehr gut

NFA #4: Skalierbarkeit

Der Anwender kann ohne Probleme beliebig viele Server anlegen. Bei einer großen Anzahl Clients kann aber die Übersichtlichkeit darunter leiden.

Bewertung: gut

Server

Server	1	2	3	4	5	6
Benutzbarkeit und Verständlichkeit	✓					
Performanz und Effizienz	✓					
Wartbarkeit und Erweiterbarkeit	✓					
Skalierbarkeit	✓					

Tabelle 5.5: Nicht-funktionale Anforderungen des Servers

NFA #5: Benutzbarkeit und Verständlichkeit

Die Weboberfläche des Servers ist sehr einfach gehalten und daher leicht verständlich. Es ist keine komplizierte Navigation notwendig, da alle Elemente auf der Startseite Platz finden.

Bewertung: sehr gut

NFA #6: Performanz und Effizienz

Sobald der Server mit Strom versorgt wird, dauert es nur wenige Sekunden bis das System hochgefahren und alle Dienste gestartet sind.

Nimmt der Nutzer Änderungen an den eingestellten Lichtwerten vor, werden diese ohne wahrnehmbare Verzögerung an den jeweiligen Client weitergeleitet.

Bewertung: sehr gut

NFA #7: Wartbarkeit und Erweiterbarkeit

Um Clients mit anderen Funktionen aufnehmen zu können, müssten der Quellcode und die Template-Dateien etwas angepasst werden.

Durch die REST- und JSON-Schnittstelle ist es kein Problem auch andere Bedienoberflächen, wie beispielsweise eine *Android*-Applikation, zu entwickeln und an das System zu adaptieren.

Bewertung: sehr gut

NFA #8: Skalierbarkeit

Der Server bietet die Funktion, beliebig viele Clients zu hinterlegen. Erst bei sehr vielen Geräten kann es zu Einschränkungen kommen, falls der Adressbereich des jeweiligen Subnetzes zu klein ist. Der Nutzer müsste dann eingreifen und die Routerkonfiguration ändern.

Bewertung: sehr gut

Client

Client	1	2	3	4	5	6
Zuverlässigkeit		✓				
Performanz und Effizienz	✓					

Tabelle 5.6: Nicht-funktionale Anforderungen des Clients

NFA #9: Zuverlässigkeit

Wenn sehr schnell hintereinander Änderungen an den Farbwerten vorgenommen werden, kann es vorkommen, dass nicht alle übernommen werden. Der Client benötigt eine kurze Zeit, um die Werte auf die Ausgangspins zu legen und neue Werte empfangen zu können. Hierbei handelt es sich allerdings um Bruchteile einer Sekunde.

Bewertung: gut

NFA #10: Performanz und Effizienz

Wie der Server auch, startet der Client sofort nach dem Einstecken. Innerhalb weniger Sekunden steht dem Nutzer der volle Funktionsumfang zur Verfügung.

Bewertung: sehr gut

6

Zusammenfassung und Ausblick

In diesem letzten Teil der Arbeit soll das Ergebnis nochmals zusammengefasst und kritisch hinterfragt werden. Es werden abschließende Bemerkungen zum Projekt gemacht und ein Ausblick auf mögliche Anwendungsfälle und Erweiterungen aufgezeigt.

6.1 Zusammenfassung

Es galt eine Infrastruktur zu entwickeln, mit der es möglich ist, eine handelsübliche Lichtleiste kabellos via Smartphone-Applikation zu bedienen. Hierbei mussten einige Hürden überwunden werden. Es offenbarten sich einige Problemstellungen für die es verschiedene Lösungsansätze gab. Diese mussten mittels verschiedenster Kriterien, wie der Nutzerfreundlichkeit, abgewägt werden, um das bestmögliche Ergebnis zu erlangen.

Es wurden diverse Hardware-Komponenten verwendet, die verknüpft werden mussten, um miteinander interagieren zu können. Dabei haben die einzelnen Bausteine nur wenig gemeinsam und wurden nicht für ein derartiges Zusammenspiel konzipiert. Dennoch ist es gelungen eine solche Infrastruktur aufzubauen und die jeweiligen Komponenten miteinander zu verbinden.

Besonders die automatische Konfiguration der einzelnen Komponenten stellte sich als komplexes Vorhaben heraus. Es muss auf sehr viele verschiedene Faktoren unterschiedlicher Gegebenheiten reagiert werden. So ist jedes Netzwerk anders aufgebaut und entspricht keinem einheitlichem Muster. Die Infrastruktur muss sich darin dennoch selbständig einbinden, ohne

andere Geräte in der Kommunikation zu behindern. Des Weiteren musste auf sehr tiefe Systemkomponenten des Smartphones zugegriffen werden, um Server im Netzwerk selbständig zu finden. Erst nach mehreren Anläufen gelang es, diese Funktionen nutzen zu können, ohne von integrierten Sicherheitseinstellungen abgewehrt zu werden.

Auch die Erweiterbarkeit um beliebige Komponenten unterschiedlicher Funktionen offenbarte einige Hürden. Es gelang, dass die Geräte nur mittels offener Schnittstellen miteinander kommunizieren und erforderliche Daten selbständig voneinander beziehen. Dennoch müssen, je nach Anwendungsfall, kleine Änderungen am Code vorgenommen werden, um das System zu erweitern. Soll beispielsweise eine Heizungssteuerung in die Infrastruktur eingebunden werden, müssen erst die benötigten Bedienungsfelder auf der Oberfläche der mobilen Anwendung platziert werden. Um alle erdenklichen Szenarien abzudecken, wäre deutlich mehr Zeit für die Entwicklung nötig.

6.2 Ausblick

Die größten Vorteile des entwickelten Systems stellen die Verwendung bereits vorhandener Hardware, die vollständig autonome Konfiguration und die Erweiterbarkeit dar.

Dadurch, dass ausschließlich Hardware namhafter Hersteller verwendet wurde, kann davon ausgegangen werden, dass diese auch später leicht zu beziehen ist. Interessenten können so das Projekt einfach nachbauen oder aber auch die Komponenten erwerben, die entsprechenden Images der *Raspberry Pis* auf die Speicherkarte kopieren, und sofort loslegen. Auch der Verkauf als fertige Pakete ist, so wie das System entwickelt wurde, problemlos möglich.

Andere Entwickler, oder gar Hersteller von Hardwarekomponenten, könnten diese Infrastruktur verwenden, um sie beliebig zu erweitern. Ein *Android*-Entwickler müsste nur von den vorhandenen Schnittstellen Gebrauch machen um eine Anwendung für Googles mobiles Betriebssystem zu entwickeln. Ein Hersteller von Heizkörperthermostaten hingegen könnte einen neuen Client entwerfen, der dann einfach in das System mit eingebunden wird.

Die denkbaren Anwendungsfälle sind grenzenlos. Die entwickelte Infrastruktur ist daher entweder als vollständiges und abgeschlossenes System, oder aber als Grundlage vieler Erweiterungen anzusehen.

6.3 Abschließende Bemerkungen

Es war spannend die originale Fernbedienung der Lichtleiste auseinanderzubauen und auf mögliche Kontaktstellen mit dem *Raspberry Pi* zu untersuchen. Diese dann tatsächlich zu verbinden und die softwareseitige Ansteuerung mit den benötigten Übertragungsinformationen zu entwickeln erfüllte mich mit großer Begeisterung.

Die Hard- und Software der entwickelten Infrastruktur sind eng miteinander verwoben. Es wurden verschiedenste Programmiersprachen verwendet, in die ich mich zum Teil zunächst einarbeiten musste. Aber auch bestehendes Wissen konnte durch die Entwicklung dieser Infrastruktur erweitert und verbessert werden.

Unter anderem die *iOS*-Entwicklung war neues Terrain für mich. Bisher befasste ich mich lediglich mit der Programmierung mobiler Applikationen für das *Android*-Betriebssystem. *Apples iOS* ist damit kaum vergleichbar. Dennoch freute ich mich darauf, mich mit dessen Eigenheiten und Funktionen zu befassen. Wie sich später herausstellte, wurde ich nicht enttäuscht.

Abschließend lässt sich sagen, dass das Projekt ein voller Erfolg war. Es wurde ein sehr intuitiv bedienbares System entwickelt, das nebenbei noch deutlich kostengünstiger ist, als vergleichbare Produkte auf dem Markt.

Alle Funktionen und Anforderungen wurden implementiert und erfüllt. Aus einer Vision mit zunächst komplexen Problemstellungen wurde eine funktionsfähige Infrastruktur, die auch Laien problemlos in Betrieb nehmen können.

Zusätzliche Geräte, wie Fernbedienungen, werden nicht mehr benötigt, da alle Geräte im Smartphone des Nutzers integriert sind.

All dies könnte dazu führen, dass die Thematik *Home Automation* weiterhin von Bedeutung bleibt und immer mehr Anwender Gefallen daran finden. Je mehr Interesse aus Kundensicht besteht, desto mehr Produkte werden entwickelt werden, die am Ende auch deutlich günstiger auf den Markt kommen können als bisher.

Ich freue mich dieses Projekt zum Abschluss gebracht zu haben, und hoffe damit eine Grundlage für eine Vielzahl von Erweiterungen für andere Entwickler zu bieten.

Literaturverzeichnis

[App15a] APPLE: *About Objective-C*. https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html, 2015. – (zuletzt besucht am 28.09.2015)

[App15b] APPLE: *Apple Developer Program - Apple Developer*. https://developer.apple.com/programs/, 2015. – (zuletzt besucht am 06.08.2015)

[App15c] APPLE: *Swift - Resources - Apple Developer*. https://developer.apple.com/swift/resources/, 2015. – (zuletzt besucht am 23.08.2015)

[BH15] BACHFELD, Daniel ; HANSEN, Sven: *Bequeme Steck- und Funklösungen im Einsatz | c't*. http://www.heise.de/ct/ausgabe/2013-9-Bequeme-Steck-und-Funkloesungen-im-Einsatz-2324487.html, 2015. – (zuletzt besucht am 11.06.2015)

[BLM+11] BRUSH, A.J. B. ; LEE, Bongshin ; MAHAJAN, Ratul ; AGARWAL, Sharad ; SAROIU, Stefan ; DIXON, Colin: *Home Automation in the Wild: Challenges and Opportunities*. http://research.microsoft.com/en-us/people/sagarwal/chi2011.pdf, 2011. – (zuletzt besucht am 02.09.2015)

[Boo15a] BOOTSTRAP: *Bootstrap Blog*. http://blog.getbootstrap.com/, 2015. – (zuletzt besucht am 23.09.2015)

[Boo15b] BOOTSTRAP: *Bootstrap · The world's most popular mobile-first and responsive front-end framework*. http://getbootstrap.com/, 2015. – (zuletzt besucht am 03.07.2015)

[Bre15] BREMMER, Manfred: *Kantar Worldpanel: Konvertierende Android-Nutzer bescheren iOS mehr Smartphone-Marktanteil - computerwoche.de*. http://www.computerwoche.de/a/konvertierende-android-nutzer-bescheren-ios-mehr-smartphone-\marktanteil,3098582, 2015. – (zuletzt besucht am 12.05.2015)

[Flo15] FLORIN, Alexander: *User Interface Design - Einstieg und Praxisratgeber*. Books on Demand, 2015

[Fou15] FOUNDATION, Raspberry P.: *Raspberry Pi - Teach, Learn, and Make with Raspberry Pi*. https://www.raspberrypi.org/, 2015. – (zuletzt besucht am 13.08.2015)

[GLRS14] GOVÉN, Tommy ; LAIKE, Thorbjörn ; RAYNHAM, Peter ; SANSAL, Eren: *Influence of ambient light on the performance, mood, endocrine systems and other factors of school children*. http://www.livingdaylights.nl/wp-content/uploads/2014/11/The-influence-of-ambient-light-on-school-children.pdf, 2014. – (zuletzt besucht am 03.09.2015)

[Hil15] HILGEFORT, Ulrich: *ct wissen Smart Home: Heimautomation - so klappts | heise online*. http://www.heise.de/newsticker/meldung/c-t-wissen-Smart-Home-Heimautomation-so-klappts-2468263.html, 2015. – (zuletzt besucht am 06.06.2015)

[KRRH11] KUIJSTERS, Andre ; REDI, Judith ; RUYTER, Boris de ; HEYNDERICKX, Ingrid: *Improving the mood of elderly with coloured lighting*. http://mmi.tudelft.nl/sites/default/files/2011_Lighting_Ami_position_paper_AK_camera_ready.pdf, 2011. – (zuletzt besucht am 03.09.2015)

[Ric15] RICHARDSON, Matt: *Serving Raspberry Pi with Flask - Matt Richardson, Creative Technologist*. http://mattrichardson.com/Raspberry-Pi-Flask/, 2015. – (zuletzt besucht am 12.09.2015)

[Ron15] RONACHER, Armin: *Flask (A Python Microframework)*. http://flask.pocoo.org/, 2015. – (zuletzt besucht am 01.10.2015)

[RW13] RODEWIG, Klaus M. ; WAGNER, Clemens: *Apps entwickeln für iPhone und iPad*. Rheinwerk Computing, 2013

[SH13] SARLANDIE, Thomas ; HIRST, Richard: *sarfata/pi-blaster · GitHub*. https://github.com/sarfata/pi-blaster, 2013. – (zuletzt besucht am 24.08.2015)

[SPSR15] SCHICKLER, Marc ; PRYSS, Rüdiger ; SCHOBEL, Johannes ; REICHERT Manfred: An Engine Enabling Location-based Mobile Augmented Reality Applications. Version: 2015. http://dbis.eprints.uni-ulm.de/1137/. In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, April 3-5, 2014, Revised Selected Papers*. Springer, 2015 (LNBIP)

[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

[SSPR15] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; REICHERT, Manfred: Process-Driven Data Collection with Smart Mobile Devices. Version: 2015. http://dbis.eprints.uni-ulm.de/1136/. In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, Revised Selected Papers*. Springer, 2015 (LNBIP)

[Tad15] TADO: *Für Heizung & Klimaanlage – Intelligent Climate Control | tado°*. https://www.tado.com/de/, 2015. – (zuletzt besucht am 23.08.2015)

[Wik15] WIKIPEDIA: *Apple iOS – Wikipedia*. https://de.wikipedia.org/wiki/Apple_iOS, 2015. – (zuletzt besucht am 17.09.2015)

Abbildungsverzeichnis

2.1 Home-Automation-Architektur 5
2.2 Raspberry Pi 7
2.3 IKEA Dioder 9

4.1 Architektur 17
4.2 Controller des IKEA Dioder 20
4.3 Pulsweitenmodulation 21
4.4 Pin-Belegung des Clients 21
4.5 Sequenzdiagramm: Automatische Anmeldung des Clients am Server 23
4.6 Pulsweitenmodulation am Client 24
4.7 REST-Schnittstelle des Client 26
4.8 MVC-Modell des Servers 28
4.9 Web-Server des Servers 29
4.10 REST-Schnittstelle des Servers 29
4.11 Screenshot: Web-Frontend des Servers 30
4.12 MVC-Modell der mobilen Anwendung 32
4.13 Storyboard der mobilen Anwendung 35
4.14 Screenshot: Ladebildschirm und Infobildschirm 36
4.15 Screenshot: Startbildschirm 37
4.16 Screenshot: Liste der gespeicherten Server 38
4.17 Screenshot: Server hinzufügen 39

Tabellenverzeichnis

2.1 Technische Daten Raspberry Pi rev. 2.0 Model B 8
2.2 Technische Daten IKEA Dioder Lichtleiste 4-tlg., LED, bunt 9

4.1 Pin-Belegung des Clients 23

5.1 Funktionale Anforderungen der mobilen Anwendung 40
5.2 Funktionale Anforderungen des Servers 42
5.3 Funktionale Anforderungen des Clients 43
5.4 Nicht-funktionale Anforderungen der mobilen Anwendung 44
5.5 Nicht-funktionale Anforderungen des Servers 45
5.6 Nicht-funktionale Anforderungen des Clients 46

Listings

4.1 Aufbau eines Befehls für pi-blaster 24
4.2 Mögliche Befehle für pi-blaster 25

Name: Mario Weber Matrikelnummer: 751905

Erklärung

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

Ulm, den .....

Mario Weber