

Technische Hochschule Nürnberg
Fakultät Elektrotechnik Feinwerktechnik Informationstechnik
Studiengang Media Engineering

Bachelor-Arbeit von
Felizia Bernutz

Konzeption und prototypische Umsetzung einer Home Automation-Applikation für iOS-Geräte

Wintersemester 2015 / 2016

Erstgutachter: Prof. Dr. Hans-Georg Hopf

Zweitgutachter: Prof. Dr. Matthias Hopf

Betreuer bei adorsys: Steffen Blümm

Schlagworte: iOS, Swift, Home Automation, HomeKit, iBeacons

Abstract

Aufgabenstellung und Zielsetzung dieser Bachelor-Arbeit ist die Konzeption einer iOS-Applikation für Home Automation in Kombination mit iBeacons und dessen anschließender prototypischen Entwicklung in Swift. Dabei wurde HomeKit von Apple als Home Automation-Protokoll verwendet.

Prüfungsrechtliche Erklärung gemäß § 19 Abs. 8 APO

Ich, *Felizia Bernutz*, Matrikel-Nr. 2377559, versichere, dass ich diese Bachelor-Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum

Felizia Bernutz

Inhaltsverzeichnis

1 Einleitung	6
1.1 Motivation	6
1.2 Zielsetzung	8
2 Grundlagen der Home Automation	9
2.1 Anwendungsfälle	9
2.2 Home Automation mit HomeKit	11
2.2.1 Entwicklung von HomeKit.....	11
2.2.2 Möglichkeiten.....	12
2.2.3 HomeKit-Struktur	14
2.3 Lokalisierung in Gebäuden mit iBeacons	16
2.4 Geofencing	16
2.5 Probleme und Lösungen	17
2.5.1 Speichern und Manipulieren von Daten.....	18
2.5.2 Multi-Awareness	19
2.5.3 Kinder-Sicherung	19
2.5.4 Unabsichtliche Pannen	20
3 Produktvision	21
3.1 Namensfindung.....	21
3.2 Zielgruppe	21
3.3 Geräteauswahl.....	22
3.4 Design	24
3.4.1 Farben.....	24
3.4.2 Font	25
3.4.3 App-Icon	26
3.4.4 Icons	26
3.5 Usability	27
4 Klick-Dummy	28
4.1 Aufbau des Klick-Dummies	30
4.2 Erstellung der Screens	31
4.2.1 Skizzen auf Karteikarten	31
4.2.2 Nachbauen der Layouts in Photoshop.....	32
4.3 Erstellung des Klick-Dummies	32
4.4 Verbesserungsvorschläge	33

4.4.1	Erklärungs- und Hilfestellungsseiten	33
4.4.2	Startseite	34
5	Umsetzung	36
5.1	Aufbau der App.....	36
5.2	Trennung der Datenstrukturen.....	37
5.3	Aufbau der Klassen-Struktur.....	38
5.4	HomeKit Service.....	39
5.4.1	Laden der HomeKit-Daten	39
5.4.2	Umwandlung in interne Datenstruktur: Homes und Rooms	41
5.4.3	Laden der bereits verbundenen Accessories.....	42
5.4.4	Umwandlung in interne Datenstruktur: Accessories.....	44
5.4.5	Laden der Characteristics von Accessories	47
5.4.6	Steuerung der Accessories	50
5.4.7	Arbeiten mit dem HomeKit Accessory Simulator	51
5.5	iBeacon Service	52
5.5.1	Autorisierung	52
5.5.2	Start Monitoring	53
5.5.3	Location Manager Delegates	53
5.5.4	Ablauf nach Finden eines iBeacon	54
5.5.5	Verbindung von HomeKit und iBeacons	55
5.6	Finaler Funktionsumfang vom Prototypen	56
5.7	Ausblick	61
6	Zusammenfassung und Fazit.....	64
a.	Literaturverzeichnis	66
b.	Abbildungsverzeichnis.....	69
c.	Anhang	71

1 Einleitung

Nachdem die Motivation und die Zielvision im Folgenden dargestellt wurden, werden im Anschluss die Grundlagen von Home Automation erklärt, damit die darauffolgenden Kapitel verständlich sind. Dabei werden auch Anwendungsfälle eines Smart Homes erläutert, um die Vielfältigkeit der Möglichkeiten zu zeigen. Anschließend wird fokussiert auf die Eigenschaften von HomeKit eingegangen. Es werden Ergänzungsmöglichkeiten aufgezeigt und auch mögliche Risiken und deren Lösungen.

Im Anschluss wird die Produktvision der Applikation Smart Living präsentiert. Es wird die Entwicklung eines Klick-Dummys vorgestellt. Der Fokus liegt auf der Beschreibung der Entwicklungsschritte des Prototyps.

Abgeschlossen wird die Arbeit mit einem Überblick über die bereits implementierten Funktionen, einem Ausblick, was noch ergänzt werden kann und einem Fazit und einer Zusammenfassung über den Arbeitsablauf der Bachelor-Arbeit.

1.1 Motivation

Die modernen Technologien schreiten immer schneller voran. Das Internet gehört zum täglichen Gebrauch. Manchmal wünscht man sich noch mehr Unterstützung durch diese neuen Technologien in alltäglichen Situationen.

Eine dieser alltäglichen Situationen wäre beispielsweise, wenn man nach einem anstrengenden Tag endlich auf dem Sofa sitzt und bemerkt, dass in der Küche noch das Licht brennt. Die Steuerung des Lichtes ist nur über einen Schalter in der Küche möglich. Man muss also erneut aufstehen, um das Licht auszuschalten.

Jeder hat sich schon einmal gewünscht, dass jemand anderes das Licht ausschalten könnte, wie ein Butler oder ähnliches.

Das ist jedoch bereits durch einen Befehl, wie „Schalte das Licht aus“, oder durch die Steuerung über das Smartphone machbar. Die Technologie dahinter nennt sich *Smart Home*¹ oder *Home Automation*².

Eine Umfrage³ der GfK Nürnberg hat ergeben, dass die Smart Home-Technologie nicht nur in Deutschland, sondern auch in Brasilien, USA, Großbritannien und Nordirland sowie Südkorea großes Interesse hervorruft. In diesen Ländern wurde im November 2015 eine internationale Umfrage zum Einfluss der Technologie-Trends auf das eigene Leben durchgeführt. Dabei hat jeder zweite Befragte angegeben, dass die Smart Home-Technologie in den kommenden Jahren den größten Einfluss auf sein Leben haben wird.

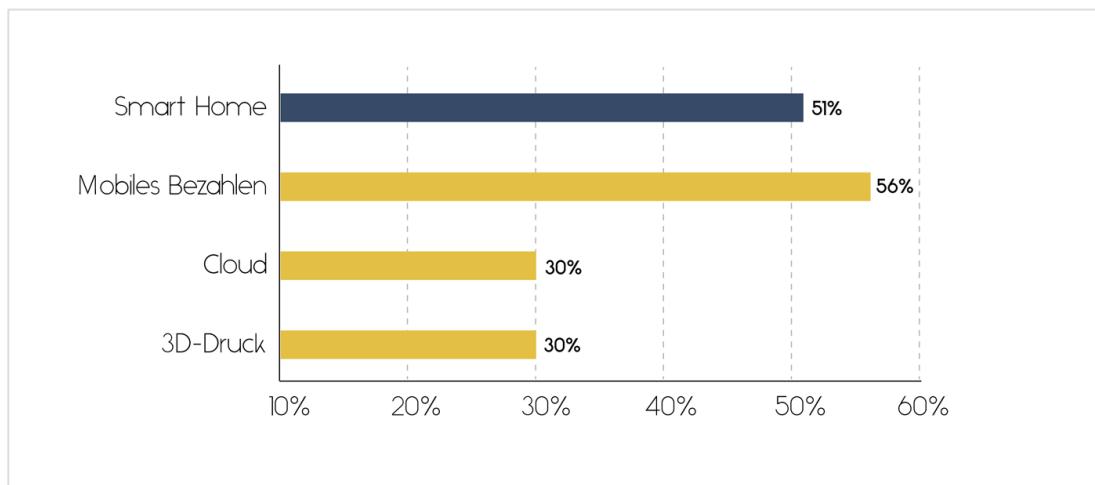


Abbildung 1-1: Umfrage über Interesse an neuen Technologien
(Eigendarstellung mit Daten von GfK)

Ein Jahr vor Beginn dieser Bachelor-Arbeit stellte Apple das erste Mal HomeKit vor. Damit setzte der Technikgigant einen eigenen Standard für die Home Automation.

Da ein großes Potenzial in dem Bereich Home Automation besteht, werden mit dieser Bachelor-Arbeit zwei aktuelle Themen miteinander verbunden, nämlich die Trend-Technologie Smart Home und der neue Standard HomeKit.

Der Markt ist noch jung und deswegen gibt es noch wenig Angebote. Darin sehe ich die Chance eine App zur Steuerung von HomeKit kompatiblen Geräten zu entwickeln, die den eigenen Ansprüchen gerecht wird und diese mit weiteren Technologien zu verbinden. Mit der Verbindung zu *iBeacons* und *Geofencing* beispielsweise, bieten

¹ Engl. für Intelligentes Haus

² Engl. für Heimautomatisierung

³ Smart Home für die Mehrheit der deutschen Befragten noch zu teuer | GfK 2015

sich Anwendungsfälle, bei denen der Nutzer durch geschickte Kombination der Technologien eine hohe Automatisierung erreichen kann.

1.2 Zielsetzung

Zu den Aufgaben meiner Bachelor-Arbeit zählt unter anderem die Erarbeitung eines Konzepts und dessen prototypischer Umsetzung in einer App für iPhones mit der Programmiersprache Swift. Die Arbeitsumgebung ist Xcode.

Eine App kann nur intuitiv benutzt werden, wenn sie fehlerfrei funktioniert und übersichtlich und ansprechend aussieht. Deswegen ist ein Ziel, die App praktikabel aufzubauen, mit besonderem Fokus auf der Startseite, da diese Seite am Meisten genutzt wird. Außerdem soll es mithilfe von Schnellzugriffen die Möglichkeit geben, direkt auf häufig genutzte Aktionen zugreifen zu können.

Zu der prototypischen Umsetzung zählen einerseits die Home Automation Funktionen, wie die Steuerung von HomeKit kompatiblen Geräten, andererseits aber auch die Erkennung von iBeacons in der Nähe des Smartphones und der Reaktion auf die erkannte Präsenz. Zusätzlich sollen Szenarios definiert werden, die mehrere Aktionen hervorrufen können. Somit sind Anwendungsfälle denkbar, wie das Einschalten der Lichter nach Sonnenuntergang oder das Ausschalten aller Geräte beim Verlassen des Hauses. Zusätzlich sollen Szenen über Sprache aktivierbar sein. Mit „*Gute Nacht, Siri*“ werden alle Geräte ausgeschalten und der Wecker für den nächsten Tag gestellt.

Die wichtigsten Ziele der App sind Zeit und Aufwand im Alltag zu sparen. Außerdem können auch Kosten für Strom und Heizung gesenkt werden, indem man die Geräte ideal nach An- und Abwesenheit steuert.

2 Grundlagen der Home Automation

Im Folgenden werden die Grundlagen der Home Automation erklärt. Hierbei geht es einerseits um mögliche Anwendungsfälle in einem Smart Home, andererseits wird spezifiziert was man unter HomeKit versteht. Es wird ebenso auf Risiken in der Home Automation hingewiesen und es werden Lösungen aufgezeigt. Schließlich wird die Kombination von HomeKit, iBeacons und Geofencing spezifiziert, die für die Bachelor-Arbeit von Relevanz ist.

2.1 Anwendungsfälle

Um ein Verständnis für die Möglichkeiten eines Smart Home zu erhalten, werden nachfolgend einige Anwendungsfälle beschrieben. Diese sind keine Zukunftsvisionen mehr, sondern bereits umsetzbar.

Generell kann jedes Haushaltsgerät in ein Smart Home integriert werden, indem man ermöglicht, das Gerät über eine Internetverbindung anzusteuern. Typische Geräte sind unter anderem Glühbirnen, Heizungen, Sensoren, wie beispielsweise Luftqualitäts- und Feuchtigkeitsmessung oder Tür- und Fenster-Sensoren, Garagentore und Türschlösser.

Es folgen nun Anwendungsfälle in Form eines Tagesablaufs.

Aufstehen

Der Tag startet mit dem Klingeln des Weckers. Die Lampen beginnen gedimmt zu leuchten, das Radio startet und ein Kaffee wird in der Küche vorbereitet. Wenn die Außentemperatur unter 0°C beträgt, wird entweder die Standheizung des Autos in das System integriert und das Auto wird vorgeheizt oder man erhält eine Benachrichtigung auf das Smartphone, dass Frostgefahr besteht und man gegebenenfalls das Auto freikratzen muss.

Die für diese Situation verwendeten Geräte sind ein Stromstecker zur Steuerung der Kaffeemaschine, Glühbirnen für die Lichter und ein Wettersensor zum Messen der Temperatur.

Verlassen des Hauses

Kurz bevor man das Haus verlässt, um in die Arbeit zu fahren, wird man benachrichtigt, ob noch Fenster oder wichtige Türen geöffnet sind, ebenso ob noch Geräte, wie Lichter, Radio etc. an sind. Wenn es an diesem Tag noch regnen soll, wird man erinnert, einen Schirm mitzunehmen. Das kann beispielsweise auch durch eine bestimmte Farbe einer Lampe angezeigt werden. Wenn man durch die Haustür gegangen ist, wird automatisch abgeschlossen.

Die hierfür verwendeten Geräte sind ein Wettersensor zum Bestimmen der Regenwahrscheinlichkeit, Tür- und Fenstersensoren, Stromstecker zur Steuerung des Radios und anderer Geräte, Glühbirnen und ein Türschloss, das mithilfe des Smartphones ver- und entsperrt werden kann.

Auf dem Weg nach Hause

Wenn man sich nach der Arbeit auf den Heimweg macht, werden die Heizungen so eingestellt, dass alle Zimmer auf Zimmertemperatur vorgeheizt sind, wenn man ankommt.

Das hierfür verwendete Gerät ist ein Thermostat zur Steuerung der Heizungen.

Nach Hause kommen

Um die Haustür aufzuschließen, benötigt man nur das iPhone und keinen Schlüssel. Nach dem Öffnen der Tür, schalten sich Lichter und das Radio an und man bekommt eine Erinnerung auf das Smartphone, welche Termine oder Aufgaben noch anstehen.

Die hierfür verwendeten Geräte sind ein Stromstecker zur Schaltung des Radios, Glühbirnen und ein Türschloss.

Schlafen gehen

Wenn man abends schlafen gehen möchte, aktiviert man eine Szene mit dem Spruch „*Gute Nacht, Siri*“. Daraufhin wird der Wecker für den nächsten Tag gestellt, alle Lichter werden ausgeschalten und der Strom wird für alle nicht dauerhaft notwendigen Geräte abgestellt. Es gibt eine Warnung, wenn wichtige Türen, wie die Haustür, noch geöffnet sind. Die Heizungen werden auf den Nachtmodus geschaltet, beziehungsweise komplett ausgestellt.

Die hierfür verwendeten Geräte sind Sensoren für Türen und Fenster, die erkennen, ob sie geschlossen oder offen sind, Stromstecker zur Steuerung der Geräte, Glühbirnen und ein Thermostat zur Regulierung der Heizungen.

Schlussfolgerung

Anhand dieser Anwendungsfälle ist zu erkennen, wie viel in der Home Automation möglich ist, um den Bewohner im Alltag zu unterstützen. Bemerkenswert ist, dass das Meiste bereits ohne viel Aufwand realisierbar ist.

2.2 Home Automation mit HomeKit

Es gibt viele Möglichkeiten der Kommunikation in der Home Automation. Es werden Protokolle, wie *ZigBee*, *Z-Wave*, *Enocean*, *Bluetooth*, *Loxone*, *Homee*, *Digitalstrom* und weitere⁴ benutzt, um mit den Geräten zu kommunizieren. Wie in der Einleitung bereits beschrieben wurde, wird in der Bachelor-Arbeit mit *HomeKit* gearbeitet, aufgrund dessen wird nur auf dieses Protokoll eingegangen.

2.2.1 Entwicklung von HomeKit

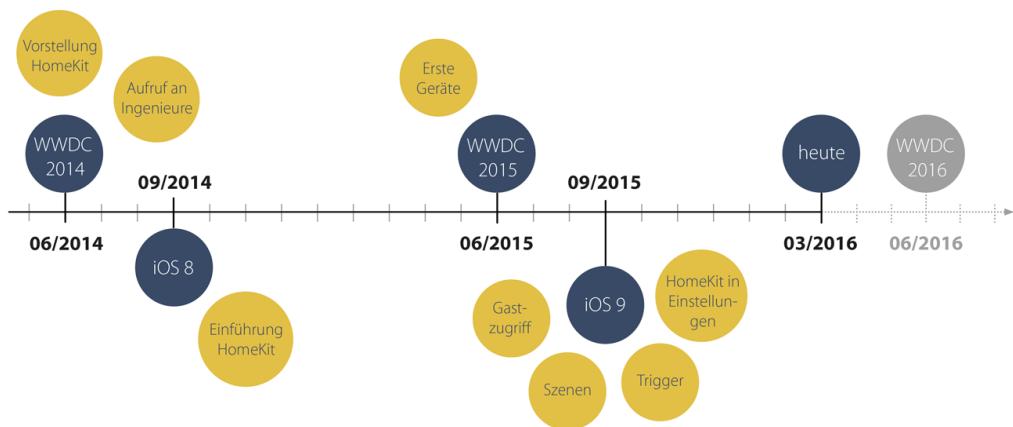


Abbildung 2-1: Entwicklung von HomeKit in zeitlicher Übersicht
(Eigendarstellung)

HomeKit wurde erstmals im Juni 2014 auf der WWDC – Worldwide Developers Conference in San Francisco, USA – von Apple vorgestellt. Dort wurde angekündigt⁵, dass Apple einen eigenen Home Automation Standard für iOS 8 anbieten wird. Apple stellt somit ein Protokoll zur Verfügung, um iOS-Geräten, wie iPhone, iPad und iPod Touch, zu ermöglichen mit Geräten zu kommunizieren. Apple selbst wird keine Geräte für die Kommunikation mit HomeKit entwickeln. Auf der Konferenz wurden Ingenieure dazu aufgerufen, Geräte für den HomeKit-Standard zu entwerfen. Dabei wurde betont, dass alle Möglichkeiten in Betracht gezogen werden können. Über das

⁴ Robert di Marcoberardino 2015, 2015

⁵ Introducing HomeKit - WWDC 2014 - Videos | Apple Developer 2014

MFI-Lizenziungsprogramm⁶ können dann Geräte von Apple autorisiert werden, den HomeKit-Standard zu verwenden.

Drei Monate später wurde iOS 8 eingeführt und somit auch HomeKit. Leider gab es zu diesem Zeitpunkt noch keine Geräte, mit denen man die Home Automation ausüben konnte. Das lag unter anderem daran, dass Apples strenge Sicherheitsrichtlinien die Entwicklung der Geräte erschwerte. Um App-Entwicklern trotzdem das Entwickeln zu ermöglichen, wurde von Apple ein *Home Kit Accessory Simulator* bereitgestellt, der verschiedene Gerätetypen simulieren kann und so die Entwicklung und das Testen auch ohne reale Geräte möglich macht.

Kurz vor Beginn der WWDC im Juni 2015 haben erste Hersteller ihre HomeKit Geräte bekannt gegeben⁷. Auf der Messe selbst wurde HomeKit aber nur wenig thematisiert.

Mit iOS 9, welches im September 2015 veröffentlicht wurde, gab es erstmals die Möglichkeit in den Einstellungen von iOS die eigenen HomeKit-Konfigurationen zu verwalten. Des Weiteren wurden *Trigger* und *Scenes* eingeführt, siehe Kapitel 2.2.2, und die Sprachsteuerung von iOS, namens *Siri*, wurde verbessert. Zusätzlich wurde auch ein *Gastzugriff* eingeführt, mit dem man Familienmitgliedern und Freunden Zugang zu seinen HomeKit-Daten geben kann. Der Zugang wird über die iCloud-ID geregelt, indem die Gäste-ID in den Benutzer-Einstellungen hinzugefügt wird. Nachdem die Einladung per Mitteilung angenommen wurde, haben Gäste nun die Möglichkeit Daten einzusehen und Geräte zu steuern. Sie können jedoch nicht die Konfigurationen ändern und auch nicht den Zugang an Dritte übertragen.⁸

Der Stand Ende Februar 2016 ist, dass es inzwischen mehrere HomeKit-Apps und auch einige HomeKit-Geräte gibt. Auf die Verfügbarkeit der Geräte in Deutschland wird in Kapitel 3.3 noch genauer eingegangen.

2.2.2 Möglichkeiten

In diesem Kapitel werden HomeKit-spezifische Eigenschaften und dessen Möglichkeiten erklärt.

Common Database

Der Begriff *Common Database* beschreibt eine durch das Betriebssystem verwaltete Ressource, die allen Apps zur Verfügung steht – wie das Adressbuch oder der

⁶ MFi Program | Apple Developer 2016

⁷ Angetestet: Elgato Eve (HomeKit) | Macerkopf.de 2015

⁸ Angetestet: Elgato Eve (HomeKit) | Macerkopf.de 2015

Kalender. Darin werden die HomeKit-Daten gespeichert, sodass ein Nutzer mit verschiedenen Apps auf dieselben Daten zugreifen kann.

Für den Nutzer hat das den Vorteil, dass die Synchronisation der Daten gesichert ist. Somit können in Apps HomeKit-Daten verändert werden und die veränderten Daten sind in den anderen HomeKit-Apps auch zu finden.

Sprachsteuerung

HomeKit-Nutzer können Siri benutzen, um Geräte mit Hilfe der Sprache steuern zu können. Siri übernimmt in iOS die Funktion eines persönlichen Assistenten und kann natürlich gesprochene Sprache erkennen und verarbeiten.⁹ Jedoch kann Siri nicht alle Sätze zum Thema HomeKit verstehen und umsetzen.

Anbei einige Beispiele, die Siri verstehen und umsetzen kann.¹⁰

- "Schalte das Licht ein."
- "Dimme das Licht auf 50 %."
- "Schalte die Kaffeemaschine ein."
- "Schalte alle Lampen im Obergeschoss aus."

Steuerung von Unterwegs

Mit HomeKit kann man auch von unterwegs seine Geräte mit Siri-Befehlen steuern, wenn man einen Apple TV (3. Generation oder neuer) besitzt. Dazu muss auf dem iPhone und dem Apple TV dieselbe Apple-ID eingerichtet sein.¹¹

Sicherheit und Privatsphäre

Die Kommunikation zwischen iOS-Gerät und HomeKit-Gerät wird mit einer Ende-zu-Ende-Verschlüsselung gesichert. Somit können Geräte nicht fremdgesteuert werden. Die Privatsphäre der Nutzer ist damit gesichert.

Bei einer Ende-zu-Ende Verschlüsselung werden die Daten vom Sender bis zum Empfänger verschlüsselt. Unbefugte können nicht auf die Daten zugreifen oder sie manipulieren.

Action Sets oder Scenes

Mit HomeKits *Action Sets* kann man durch einen Auslöser zeitgleich mehrere Aktionen hervorrufen. Auslöser können unter anderem eine Statusänderung eines

⁹ Siri (Software) | Wikipedia 2016

¹⁰ HomeKit-kompatible Heimelektronik nutzen | Apple Support 2015

¹¹ HomeKit-kompatible Heimelektronik nutzen | Apple Support 2015

Gerätes sein oder die Uhrzeit, wie die Zeit des Sonnenaufgangs. Auch das Betreten oder Verlassen eines bestimmten Bereichs, das über GPS-Koordinaten oder durch iBeacons, siehe Kapitel 2.3, erkannt wird, kann als Auslöser verwendet werden. Dieses Prinzip wird *Geofence* genannt und genauer in Kapitel 2.4 beschrieben.

Denkbar ist zum Beispiel alle Lichter in den am meisten genutzten Zimmern nach Sonnenuntergang einzuschalten oder beim Verlassen des Hauses alle nicht dauerhaft genutzten Geräte auszuschalten.

Trigger

Trigger werden die Auslöser für Action Sets genannt. Wie in Action Sets schon beschrieben kann das entweder ein Zeitpunkt, ein Standort oder der Zustand eines anderen HomeKit Geräts sein. Das Ausführen der Aktion wird von iOS im Hintergrund übernommen. Das erlaubt dem Nutzer komplexe Aktionen zu definieren.

Anhand des Beispiels werden Trigger und Action Sets noch einmal veranschaulicht. „Wenn die Tür abgeschlossen wird, dann schalte alle Geräte aus.“ Hier ist das Abschließen der Tür der Trigger der Szene, dass alle Geräte ausgeschalten werden.¹²

2.2.3 HomeKit-Struktur

Nun wird die Struktur von HomeKit anhand der Abbildung 2-2¹³ erklärt.

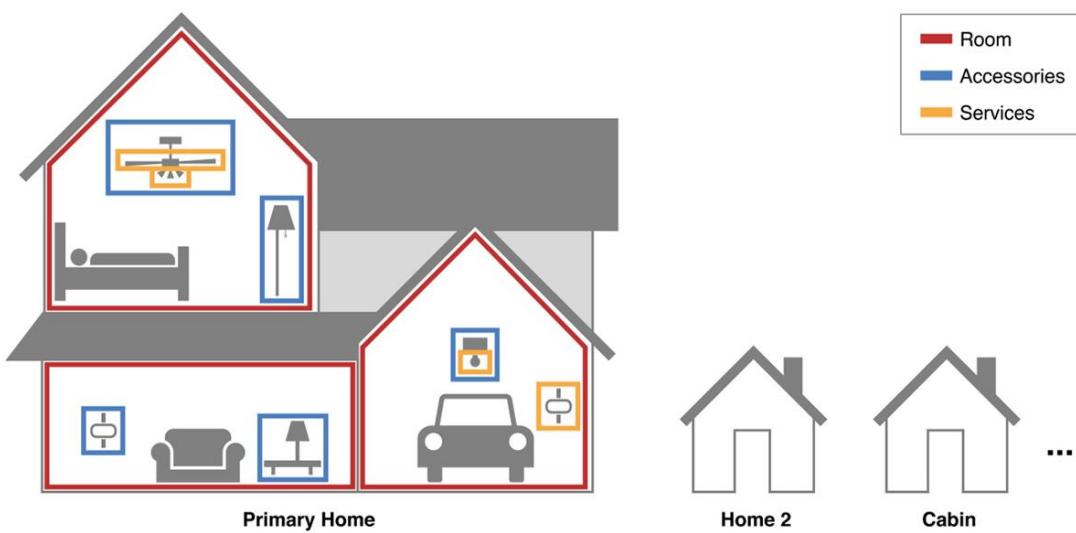


Abbildung 2-2: HomeKit Struktur
(Getting the Home Layout | Apple Developer 2015)

¹² HomeKit Basics: Auslöser verständlich erklärt | HomeKit.tips 2016

¹³ Getting the Home Layout | Apple Developer 2015

Homes

Es gibt *Homes*, die für einzelne Häuser stehen. Man muss mindestens ein Haus haben, welches die Funktion des primären Hauses übernimmt.

Rooms

Ein Home besitzt *Rooms*, die für die verschiedenen Zimmer eines Hauses stehen (in Abbildung 2-2 rot umrandet).

Accessories

Dort können dann verschiedene Geräte verwaltet werden, die von Apple *Accessories* (in Abbildung 2-2 blau umrandet) genannt werden. Ein Accessory wird einem Room zugewiesen und kann zum Beispiel ein Ventilator sein.

Services

Geräte besitzen *Services*, die für eine Funktion des Geräts stehen. Neben dem Information Service, der standardmäßig in jedem Accessory vorhanden ist, kann es mehrere Services (in Abbildung 2-2 gelb umrandet) geben. Der Ventilator hat neben einem *Fan Service*, der für die Lüftung zuständig ist, auch einen *Lightbulb Service*, wenn der Ventilator mit einer Lampe kombiniert wurde.

Es gibt eine Liste an Services, die Apple zur Verfügung stellt, wie die beiden genannten. Trotzdem können auch eigene Services, sogenannte *Custom Services*, erstellt werden, welche von Siri aber nicht verstanden und gesteuert werden können.

Characteristics

Die Eigenschaften der Services werden *Characteristics* genannt. Eine Lampe kann dann unter anderem die Characteristics der Helligkeit oder den Status (An/Aus) haben. Ein Ventilator hat unter anderem die Characteristic, die dafür zuständig ist, wie schnell sich die Rotorblätter drehen.

Zones

Man kann mehrere Räume in Zonen zusammenfassen – wie alle Kinderzimmer oder alle Räume der oberen Etage. So ist es möglich Szenen anzulegen, in der alle Lichter einer Zone ausgeschaltet werden.

2.3 Lokalisierung in Gebäuden mit iBeacons

Wie in Kapitel 2.2.2 schon aufgezeigt wurde, können iBeacons als Auslöser von Action Sets verwendet werden. Es wird nun erklärt, was iBeacons sind und wie man sie in der Home Automation verwenden kann.

iBeacons sind kleine Bluetooth-Sender, welche in regelmäßigen Intervallen, zwischen 20 Millisekunden und 10 Sekunden, Bluetooth-Low-Energy-Signale (BLE-Signale) aussenden. Die Signale können von Geräten, wie Smartphones, empfangen werden. Durch das Empfangen der Signale wird erkannt, ob sich iBeacons in der Nähe befinden. Hierbei kann zwischen unmittelbarer, kleiner und großer Entfernung unterschieden werden.

Ein Datenpaket eines iBeacon-Signals besteht aus einer *ID*, einer *UUID*, der Major und *Minor* und der *TX Power*. Ein Smartphone kann mithilfe der *UUID* nach iBeacons suchen. Die Ergebnis der Suche kann noch mit der Major und Minor weiter eingegrenzt werden. Dieses sind die für die spätere Entwicklung wichtigen Parameter.

iBeacons können gut in der Home Automation angewendet werden. Ein iBeacon steht dann für einen einzelnen Raum. Wenn man nun mit der geöffneten App von Raum zu Raum geht, werden die zum Raum passenden Accessories angezeigt und der Nutzer muss nicht manuell den Raum wechseln. Es sind außerdem Anwendungsfälle vorstellbar, wie das Ausschalten aller Geräte, wenn das Haus und somit die Reichweite des iBeacons verlassen wird.

Besonders die Kombination von HomeKit und iBeacons ist eine große Chance, dem Nutzer die Bedienung der App zu erleichtern.

In der Bachelor-Arbeit wurden iBeacons der Firma *Estimote*¹⁴ verwendet, da mit diesen bereits gute Erfahrungen gemacht werden konnten.

2.4 Geofencing

Genauso wie ein iBeacon kann auch *Geofencing* als Auslöser von Action Sets verwendet werden. Mithilfe von Geofencing können automatisiert Aktionen ausgelöst werden, indem eine Begrenzung in Form von GPS-Daten überschritten wird.¹⁵ Geofencing wird von Apple seit iOS 7 genutzt, um Benachrichtigungen auszulösen, wenn ein vorher festgelegte Bereich betreten oder verlassen wurde.

¹⁴ Real-world context for your apps | Estimote 2016

¹⁵ Geofencing | Wikipedia 2016

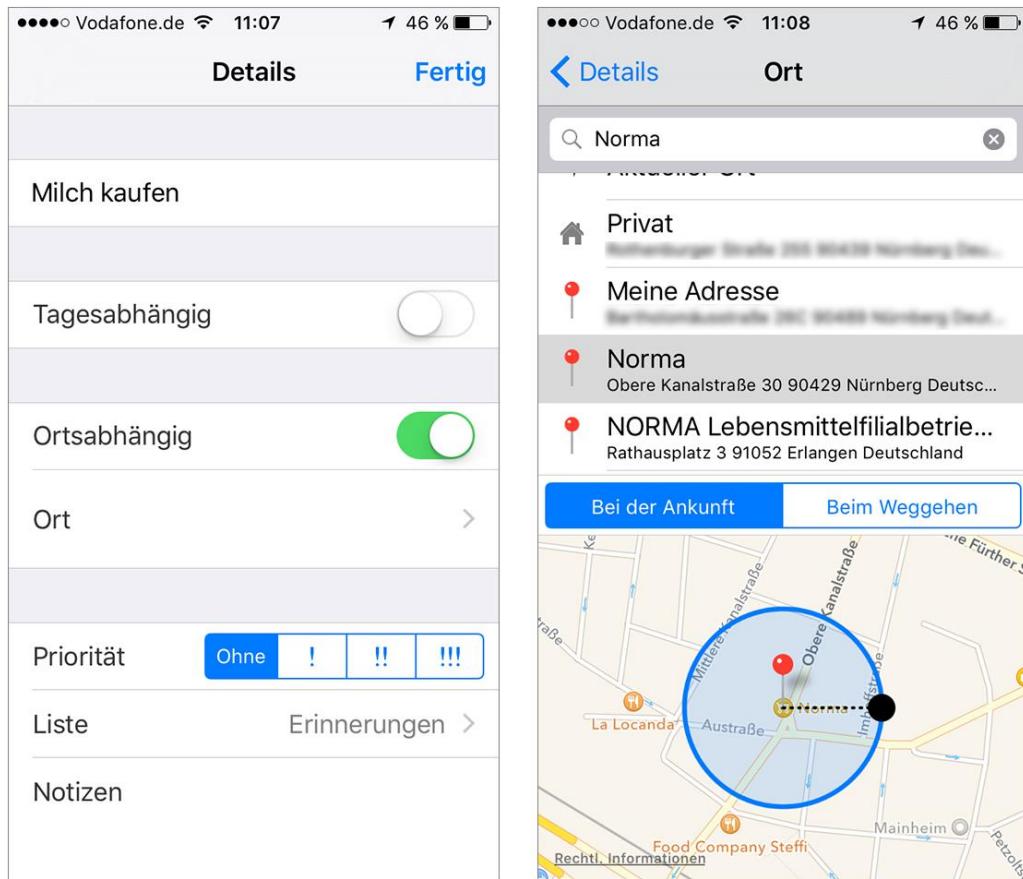


Abbildung 2-3: Ortsabhängige Erinnerung mit Geofencing
(Screenshot von Erinnerungsapp)

Diese Technologie kann man sehr gut in einer Home Automation App gebrauchen, da Anwendungsfälle möglich sind, wie das Aufheizen aller Zimmer auf Zimmertemperatur, wenn der Arbeitsplatz verlassen wurde.

2.5 Probleme und Lösungen

Was viele Menschen noch zweifeln lässt, sind neben Datenschutz- und Sicherheitsbedenken auch die hohen Kosten der Anschaffung. Das hat die GfK in einer Umfrage¹⁶ herausgefunden.

¹⁶ Smart Home für die Mehrheit der deutschen Befragten noch zu teuer | GfK 2015

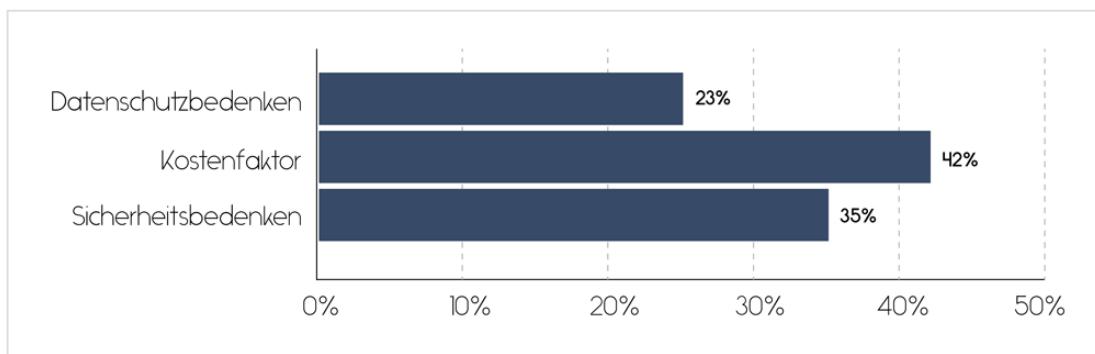


Abbildung 2-4: Umfrage über Bedenken bei Home Automation
(Eigendarstellung mit Daten von GfK)

Im Folgenden wird detailliert auf mögliche Risiken eines Smart Homes hingewiesen. Es werden Lösungen aufgezeigt, die die Sicherheit wieder garantieren können.

2.5.1 Speichern und Manipulieren von Daten

Aufgrund der ununterbrochenen Internet-Verbindung der Geräte kann das Verhalten eines Smart Home Nutzers analysiert und ausgewertet werden.

Das kann vorteilhaft sein, da das intelligente Zuhause dazu lernen kann, wann der Anwender von der Arbeit kommt und wie warm er es dann gerne hat. Das Erstellen eines Nutzerprofils kann beispielsweise auch hilfreich sein, wenn man bestimmten möchte, wie Strom gespart werden kann.

Jedoch könnte das Sammeln der Daten auch schwerwiegende Folgen haben, da aus den Daten beispielsweise ermittelbar wäre, wann sich der Nutzer zuhause aufhält und wann nicht. Das kann die Wahrscheinlichkeit eines erfolgreichen Einbruchs erhöhen. Außerdem stellt sich die Frage, ob beispielsweise Türschlösser eines Smart Homes manipulierbar sind.

Das sind schon zwei Gründe dafür, dass geklärt werden muss, wie man garantieren kann, dass die Daten sicher und nicht manipulierbar sind.

Lösung

Bei HomeKit wird stets eine direkte Verbindung zu den Geräten hergestellt. Seit iOS 9 wird die Verbindung über das *HomeKit Accessory Protocol* – kurz HAP – hergestellt. Hierbei wird eine Ende-zu-Ende Verschlüsselung verwendet, sodass auch Apple nicht mehr auf die Daten zugreifen kann.¹⁷

¹⁷ Apple HomeKit und das Smart Home der Zukunft | meintechblog.de 2015

Das Verlieren eines Smartphones, welches für das Verschließen und Öffnen von Türen benutzt wird, hat weitreichende Folgen. Hier gibt es die Möglichkeit sich über die Internetseite des Anbieters von seinem Account abzumelden, sodass das Steuern des Schlosses nicht mehr möglich ist.¹⁸

2.5.2 Multi-Awareness

Eine weitere Situation, die Probleme mit sich führen kann, entsteht, wenn mehrere Personen in einer Wohnung leben und nur eine Person das Smart Home steuert. Beim Verlassen der Wohnung werden nun Szenen ausgelöst, wie das Ausschalten der Geräte, jedoch ist die Wohnung gar nicht leer. Das kann für die Mitbewohner ärgerlich sein.

Lösung

Es gibt die Möglichkeit Szenen zu deaktivieren, wenn man das vor dem Verlassen der Wohnung beachtet, wird diese Situation vermieden.

Eine andere Lösung wäre, dass man für jeden Bewohner eine Lampe einstellt. Wenn diese Bewohner anwesend sind, stellen sie ihre Lampe auf eine Farbsättigung von beispielsweise 70%. Nur wenn sie abwesend sind, stellen sie die Farbsättigung auf 90%. Wenn man nun ein Action Set „*Ich verlasse das Haus*“ anlegt, muss man vorher überprüfen, ob alle Mitbewohner-Lampen auf 90% stehen, erst dann sollen alle Geräte ausgeschalten werden.¹⁹

2.5.3 Kinder-Sicherung

Des Weiteren sollte es die Möglichkeit geben, anderen Nutzern unterschiedliche Rechte zuzuordnen und Steuerungen zu sperren. Die Kontrolle über das Garagentor oder das Aktivieren des Ofens, sollte Kindern nicht erlaubt werden, wenn sie ein Smartphone zur Steuerung besitzen.

Lösung

Eine Lösung dafür wäre es, verschiedene Benutzergruppen anzulegen und diesen unterschiedliche Rechte zu geben. Kinder dürfen beispielsweise die Daten nur Lesen, aber nicht verändern. Erwachsene dürfen beides, während Gäste auch nur Lesen dürfen.

¹⁸ Lost Phone | august 2016

¹⁹ Anwesenheitserkennung bei mehreren Bewohnern | HomeKit.tips 2016

2.5.4 Unabsichtliche Pannen

Es gibt viele Fälle, die ein Smart Home nicht beachten kann. Wenn man das Zuhause verlässt und sein Smartphone aber vergisst, verschließt sich ein smartes Türschloss nicht. Eine mögliche Erinnerung könnte sein, wenn man sein Auto auch mit dem Smartphone öffnen muss. Dann stellt man fest, dass das Smartphone noch in der Wohnung liegt und da die Tür noch nicht abgeschlossen ist, kommt man wieder in die Wohnung. Wenn man sich allerdings zu Fuß entfernt und sein Smartphone auf dem Weg nicht benutzt, bemerkt man das Fehlen auch nicht und die Wohnung bleibt die gesamte Zeit unabgeschlossen.

Lösung

Eine Möglichkeit wäre, einen zweiten Weg zur Steuerung anzubieten, beispielsweise über einen Webauftritt Steuerungen vorzunehmen, so könnte man die Wohnung noch nachträglich abschließen.

Man muss jedoch auch bedenken, dass das Smart Home eine Erleichterung im Alltag bieten soll. Aber es bedeutet nicht, dass man nicht mehr selbst mitdenken muss.

3 Produktvision

3.1 Namensfindung

Durch den App-Namen soll für den Nutzer erkennbar sein, um was es in der App geht. Da die Applikation den Alltag erleichtern und für den Nutzer mitdenken soll, habe ich mich für den Namen *Smart Living* entschieden. Die Assoziation vom intelligenten Wohnen wird hervorgerufen. Außerdem ist es ein kurzer einprägsamer Name, der leicht lesbar und sprechbar ist.

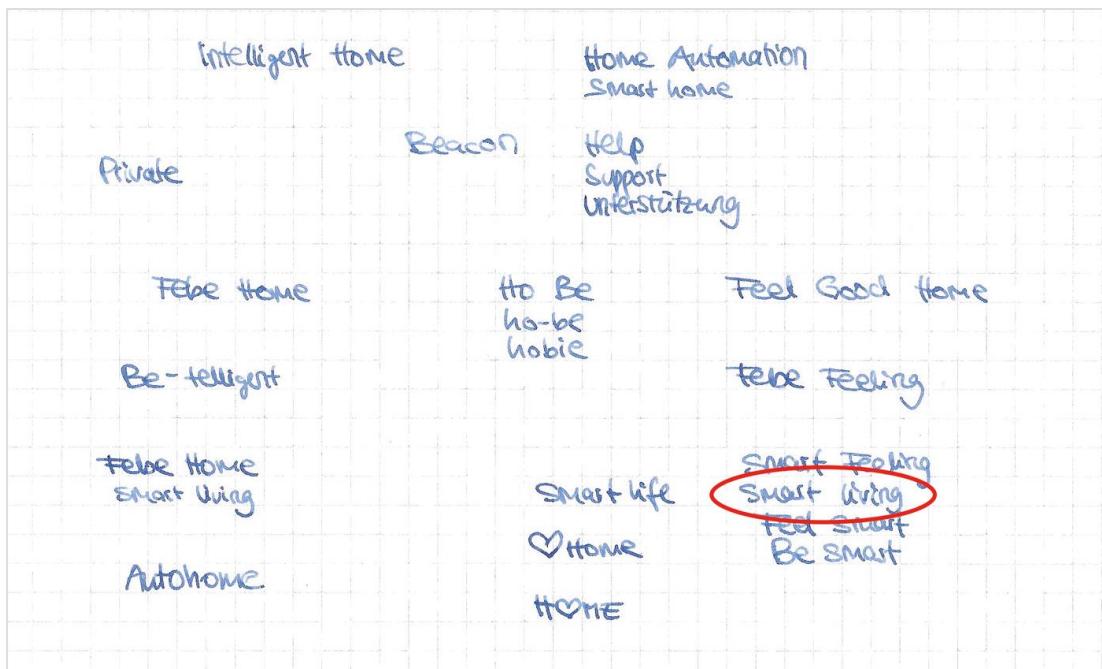


Abbildung 3-1: Brainstorming Namensfindung
(Eigendarstellung)

3.2 Zielgruppe

Die potenziellen Nutzer von Smart Living sind alle Besitzer eines iPhones mit dem Betriebssystem iOS 9.0 oder neuer.

Grundsätzlich gibt es zwei verschiedene Nutzergruppen für die App. Die Einen sind technikbegeistert, offen zu neuen Technologien eingestellt und haben Vertrauen in

die Technologie. Zusätzlich ist ein Smart Home in der Anschaffung teuer, weshalb man finanziell gut gestellt sein sollte. Die Einrichtung mit fünf Geräten beträgt schon mehrere Hundert Euro.

Die andere Art von Nutzern sind ältere oder hilfsbedürftige Menschen, da sie mit der App eine Hilfestellung für den Alltag bekommen würden. Besonders die Sprachsteuerung über Siri kann hilfreich sein, wenn die beeinträchtigte Motorik die Steuerung eines Smartphones behindert. Für die Einrichtung eines Smart Homes sollte zusätzliche Hilfe in Anspruch genommen werden.

3.3 Geräteauswahl

HomeKit kompatibel sind alle Geräte, die mit der folgenden Grafik²⁰ gekennzeichnet sind.



Abbildung 3-2: HomeKit-Zertifikat
(HomeKit-kompatible Heimelektronik nutzen |
Apple Support 2015)

Viele Firmen haben bereits angekündigt HomeKit zu unterstützen, jedoch wurde es technisch noch nicht umgesetzt und die Produkte sind noch nicht erwerblich. Ende Februar 2016 gibt es nur wenige HomeKit kompatible Geräte. Hinzu kommt, dass die meisten Geräte nur in den USA und nicht in Europa in den Handel kommen.

Die erste Firma, die in Deutschland HomeKit Produkte auf den Markt gebracht hat, war die Firma *Elgato*²¹. Im Angebot haben sie einen Wetersensor, den *Eve Weather*, der außerhalb der Wohnung platziert wird und einen Raumsensor, *Eve Room*, der in der Wohnung steht und dort die Luftqualität messen kann. Es gibt einen Sensor zur Erkennung, ob Fenster und Türen geöffnet oder geschlossen sind, der *Eve Door & Window* genannt wird. Schließlich gibt es noch ein steuerbares Gerät, den *Eve Energy*. Dieser kann zur Schaltung einer Steckdose verwendet werden. Mit dem Stecker können Geräte aktiv an- und ausgeschalten werden. Außerdem kann man den verbrauchten Strom messen.

²⁰ HomeKit-kompatible Heimelektronik nutzen | Apple Support 2015

²¹ Eve Landing | elgato.com 2015

Die in Deutschland zur Verfügung stehenden Produkte, werden im Folgenden aufgezählt.²²

Gerät	Einordnung	Veröffentlichung	Preis
Eve Weather	Temperatur Außen/Luft	Anfang 08/2015	49,95€
Eve Room	Temperatur Innen/Luft	Anfang 08/2015	79,95€
Eve Energy	Stecker	Anfang 08/2015	49,95€
Eve Door & Window	Sicherheit	Anfang 08/2015	39,95€
Philips Hue E27	Bridge und drei Glühbirnen	Anfang 11/2015	179,59€
Netatmo	Thermostat	Ende 11/2015	179,95€
Parce One	Stecker	-	59,95€
Command Kit Wireless Smart Light Bulb Adapter	Adapter für Glühbirnen	-	49,99€
Monkey	Haustüröffner	-	79,00€
Nanoleaf Smarter Kit	Glühbirnen	-	89,99€

Abbildung 3-3: Verfügbare Geräte in Deutschland
(HomeKit Produktübersicht | HomeKit.tips 2016)

Für die Entwicklung wurden außer dem Eve Room alle Eve Elgato Produkte von der Firma *adorsys* bereitgestellt. Die Geräte funktionieren zuverlässig. Die einzige Kritik gibt es beim Eve Door & Window, da hier ein durchgehendes Piepen wahrnehmbar ist. Jedoch wurde das von Elgato in einer neuen Produktionsreihe bereits behoben.

Mit diesen Geräten ist unter anderem der folgende Anwendungsfall denkbar. Wenn das Fenster geöffnet ist und die Außentemperatur unter einen bestimmten Wert sinkt, soll eine vorher bestimmte Lampe eingeschalten werden, damit man darauf aufmerksam gemacht wird, das Fenster zu schließen.

²² HomeKit Produktübersicht | HomeKit.tips 2016

3.4 Design

Der Stil der App soll schlicht und übersichtlich sein. Dazu gehören dünne Trennlinien, übersichtliche Überschriften, eine leicht leserliche Schrift und eine klare Struktur. In den nachfolgenden Kapiteln wird näher auf die Wahl der Farben, der Schrift und der Icons eingegangen.

3.4.1 Farben

Auf der Suche nach einem Farbkonzept für Smart Living, fand ich die Webseite colorso.co, bei der man sich zufällige Kombinationen aus fünf Farben ausgeben lassen kann. Anschließend ist es möglich einzelne Farben zu speichern und die dazugehörigen neu zu mischen oder auch einzelne Farben manuell anzupassen. Schließlich wurde die in Abbildung 3-4 sichtbare Farbpalette²³ ausgewählt.

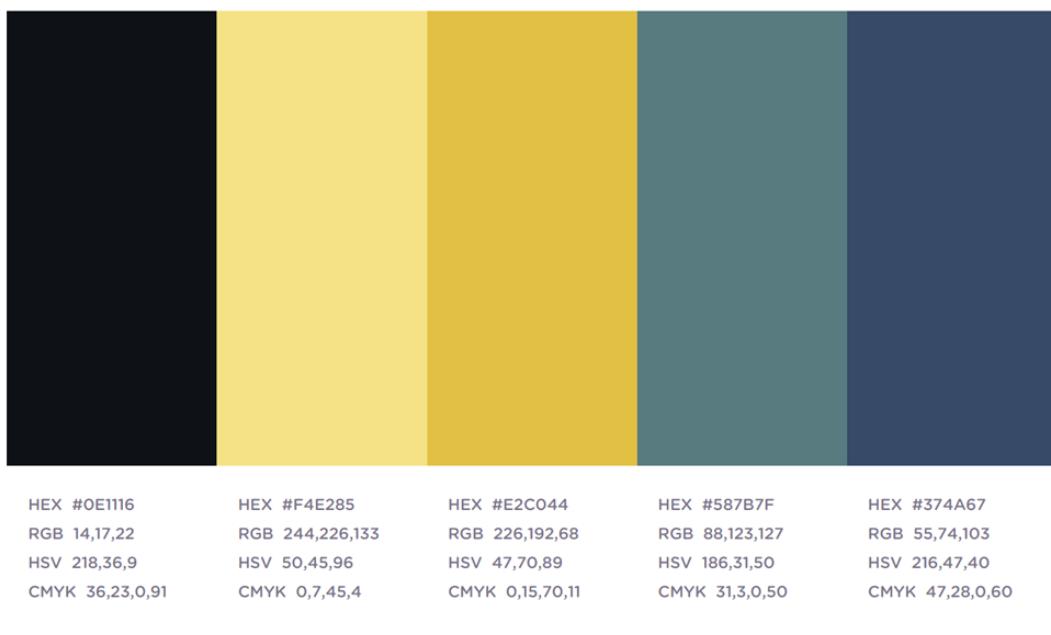


Abbildung 3-4: Ausgewählte Farbpalette
(The super fast color schemes generator | Coloros von Fabrizio Bianchi 2015)

Das Farbkonzept der App wurde auf eine Grundfarbe und eine Kontrastfarbe festgelegt. Der Lauftext wird aufgrund der einfachen Lesbarkeit in schwarz dargestellt. Die primäre Farbe in der App ist das dunkle Blau der Farbpalette, welches ganz rechts sichtbar ist. Die Kontrastfarbe ist das Gelb in der Mitte. Die restlichen Farben der Farbpalette wurden in der App nicht verwendet.

²³ The super fast color schemes generator | Coloros von Fabrizio Bianchi 2015

3.4.2 Font

Für Überschriften und hervorzuhebende Begriffe wird der Font namens *Ikaros*²⁴ verwendet. Die Schriftart ist leicht lesbar, hat jedoch etwas Besonderes, das Überschriften sehr gut von dem restlichen Text abheben lässt. Ikaros ist charakteristisch für die gesamte App und wurde auch im App-Icon verwendet.

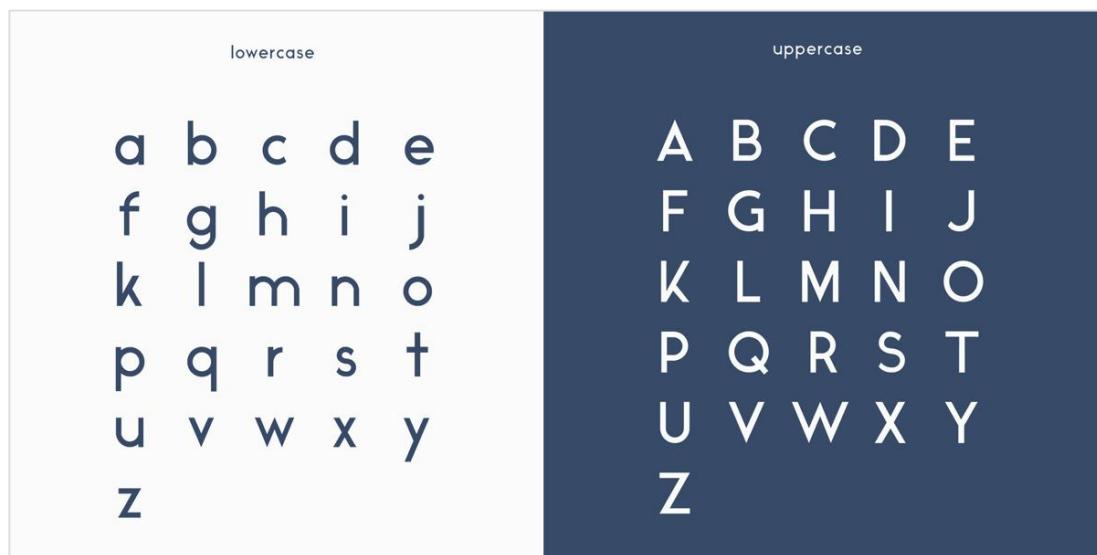


Abbildung 3-5: Ikaros Font
(Ikaros - Free Typeface | Behance 2015)

Einzeln wurde die Schrift in einer Mischung aus der Light und der Regular Version verwendet. So erhält man noch ein ansprechendes Design.



Abbildung 3-6: Verwendung der Schriftart Ikaros
(Eigendarstellung)

Als Kontrast wird im Lauftext die *System Font* in der Thin-Version aus Xcode verwendet. Die System Font ist die *San Francisco* für iOS 9. Diese ist eine schlichte Schrift ohne Serifen, welche es in vielen Schriftgewichtungen gibt und somit vielfältig einsetzbar ist.

²⁴ Ikaros - Free Typeface | Behance 2015

3.4.3 App-Icon



Abbildung 3-7: App-Icon
(Eigendarstellung)

Das App-Icon setzt sich aus der Hauptfarbe, der für Überschriften gewählten Schriftart und einer Grafik zusammen, wobei die Farbe und die Schriftart bereits beschrieben wurden.

Das Logo ist in zwei Ebenen unterteilt. Die erste Ebene ist die Schriftebene, in der der App-Name dargestellt wird. Die zweite Ebene ist die Bildebene. Hier wird die obere von der unteren Hälfte mithilfe einer Linie getrennt. Die untere Hälfte ist die Grundlage, auf die die App sich

bezieht. Da Smart Living dafür steht, den Alltag zu erleichtern, stehen die Wurzeln für verschiedene Aspekte des Alltags. Alle Wurzeln münden in einem Schnittpunkt des oberen und unteren Teils, wobei sich im oberen Teil auf den Nutzer bezogen wird. Diesen Schnittpunkt soll Smart Living darstellen, da die App die Komplexität des Alltags für den Nutzer bündelt.

3.4.4 Icons

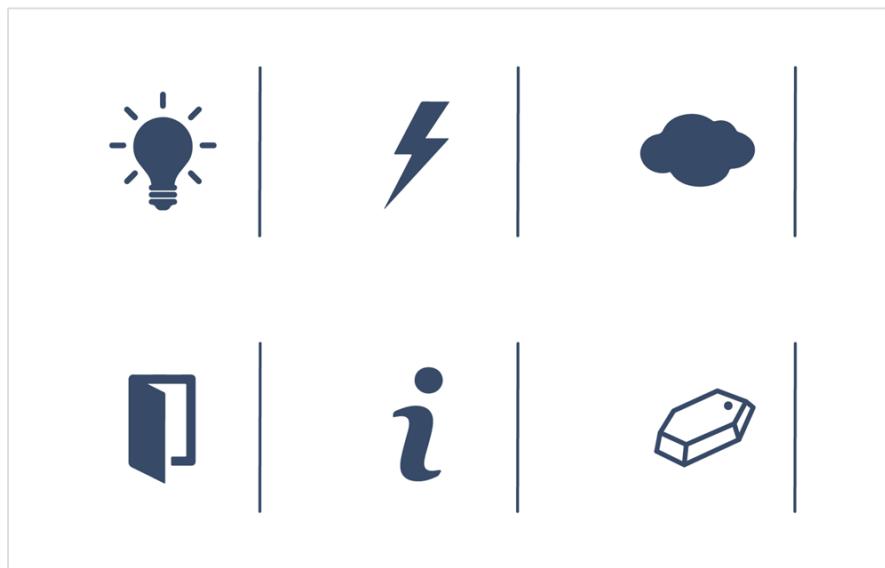


Abbildung 3-8: Icons zur Darstellung der Accessory-Typen
(Eigendarstellung)

Die in Abbildung 3-8 gezeigten Icons stehen für die verschiedenen Accessory-Typen, mit Ausnahme des letzten Icons, welches für ein iBeacon steht. Die Icons wurden in der Hauptfarbe angelegt, um das gewählte Farbkonzept aufzugreifen.

Der Stil der Icons ist schlicht. Es soll aber trotzdem direkt erkennbar sein, welche Kategorie dargestellt wird. Die dünnen Trennlinien stehen für das minimalistische

Design. Diese werden benutzt, um das Icon von dem restlichen Inhalt abzutrennen, siehe spätere Abbildung 5-1.

3.5 Usability

Smart Living soll einfach und selbsterklärend zu benutzen sein.

Besonders die Punkte Individualität und schnelle Erreichbarkeit sind für die App von Vorteil. Dies soll am Beispiel der Startseite erkennbar gemacht werden. Hier wird eine Übersicht über die verbundenen Accessories und deren Characteristics gegeben.

Es soll die Möglichkeit geben, dass sich der Nutzer die Ansicht individuell anpassen kann. Durch die Anpassungen bekommt der Nutzer das Gefühl, dass die Technik seinen Bedürfnissen folgt und er sich nicht an eine neue Technik anpassen muss²⁵. Vor Allem die Anzeige der Informationen soll anpassbar sein. Der Nutzer soll die Möglichkeit bekommen, zu entscheiden, ob er beim Eve Energy nur die Temperatur mit einem kleinen Bild angezeigt bekommen möchte oder ob er weitere Informationen wie Luftfeuchtigkeit und Luftdruck angezeigt haben will.

Ein weiterer Punkt ist, dass Nutzer für das Steuern der Geräte mehrere Schritte durchfolgen müssen. Um das Licht anzumachen, muss man das Smartphone in die Hand nehmen, es entsperren, die App öffnen und dann die Einstellung suchen, mit der man das Licht anschaltet. Mit einer Fernbedienung geht das um einiges schneller, da die Fernbedienung nur in die Hand genommen werden muss und ohne hinzusehen, kann der richtige Knopf gedrückt werden. Die Steuerung über das Smartphone kann also aufwändiger sein, als das Bedienen über eine Fernbedienung. Eine Lösung dafür sind *Schnellzugriffe*²⁶. Die wichtigsten Steuerungen kann man in die Mitteilungszentrale von iOS einbinden, siehe Anhang I. Diese ist mit einer Geste immer erreichbar. Der Schnellzugriff soll auch individuell einzurichten sein.

²⁵ Lars Hinrichs 2014

²⁶ Praxistest Philips hue Lampensystem | MacTechNews 2015

4 Klick-Dummy

Nach den Definitionen zum Design und der Usability, war das Ziel einen Klick-Dummy zu entwerfen bevor mit der Umsetzung begonnen wird. So konnten die festgelegten Kriterien getestet werden und inhaltlich konnte genauer spezifiziert werden, wie die App aufgebaut sein soll.

In diesem Kapitel geht es um die Erstellung eines Prototyps mit Klick-Flächen.

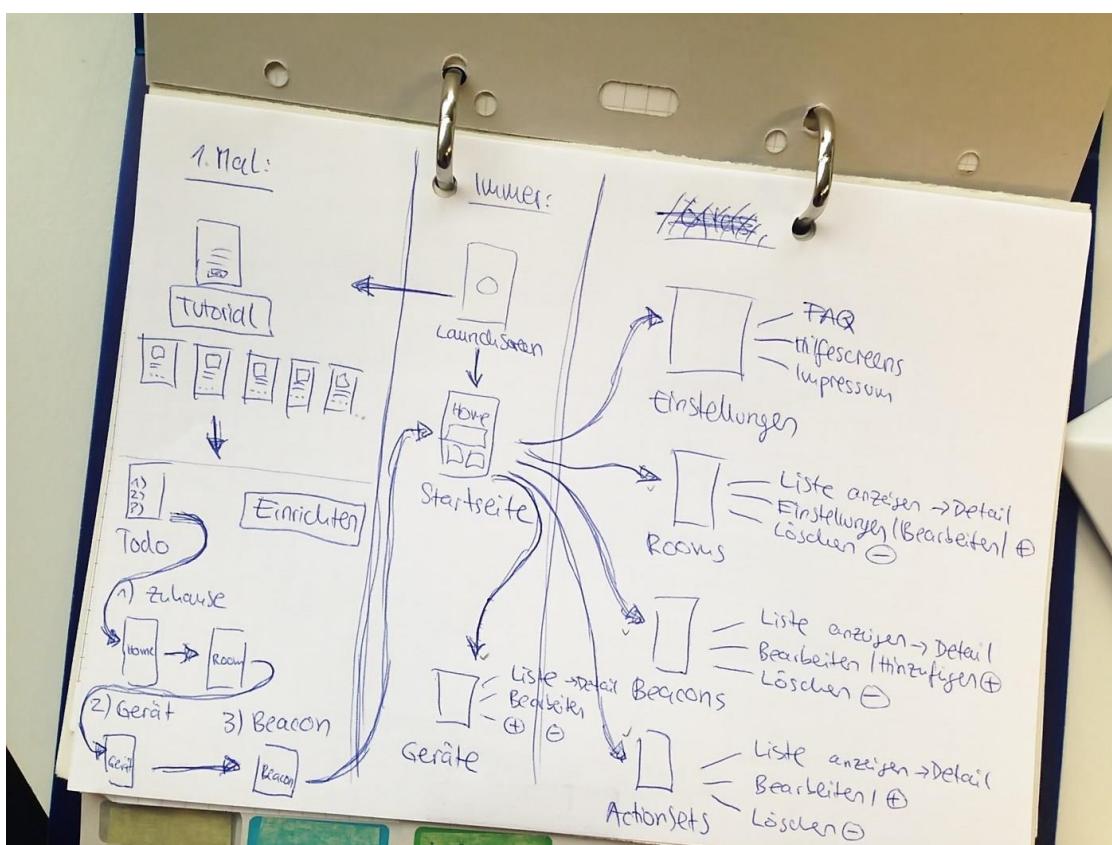


Abbildung 4-1: Skizze vom Aufbau der App
(Eigendarstellung)

Öffnen des Klick-Dummys

Um auf den fertigen Klick-Dummy zugreifen zu können, kann entweder per Link oder über den QR-Code, siehe Abbildung 4-2, der Klick-Dummy auf einem Smartphone installiert werden.

Laden per Link

<https://marvelapp.com/fge766>

Laden per QR-Code



Abbildung 4-2: QR-Code zum
Laden des Klick-Dummys
(Free mobile & web prototyping /
Marvel)

Um anschließend den Prototypen auch ohne *Browser-Leiste* ansehen zu können, kann man sich die Seite über den Teilen-Button auf den Home-Bildschirm speichern, siehe Abbildung 4-3. Wenn man nun vom Home-Bildschirm des Smartphones den Prototypen öffnet, erscheint nach einem kurzen Ladescreen der Klick-Dummy.

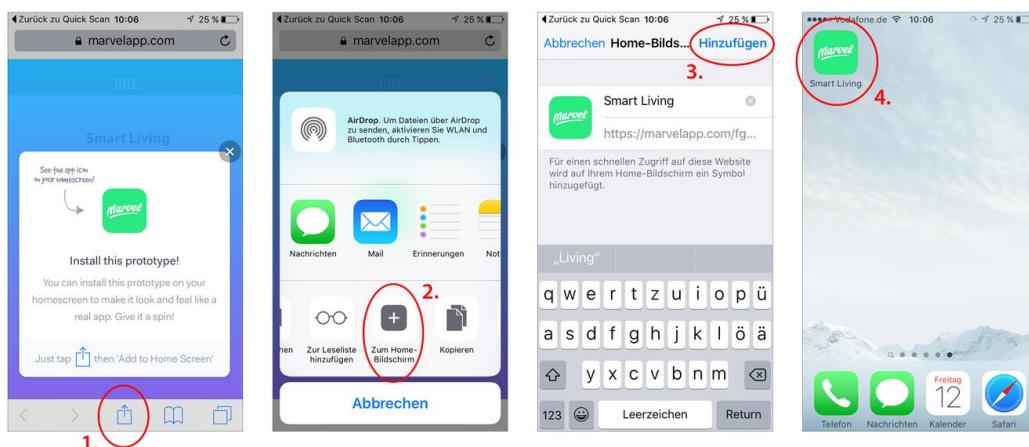


Abbildung 4-3: Hilfestellung zum Installieren des Klick-Dummys
(Screenshots von Safari und Homescreen)

Falls der Klick-Dummy nicht mehr erreichbar sein sollte, sind in Anhang I die wichtigsten Seiten des Klick-Dummys zum Ansehen bereitgestellt.

4.1 Aufbau des Klick-Dummys

Der Klick-Dummy ist folgendermaßen aufgebaut. Nach dem Öffnen der App wird geprüft, ob Daten zuvor eingerichtet wurden. Wenn nicht, dann wird man in ein Einleitungs-Szenario weitergeleitet. Dort werden die Funktionalitäten der App erklärt und es wird Hilfestellung beim Einrichten zu den in Abbildung 4-4 aufgezählten Themen gegeben.

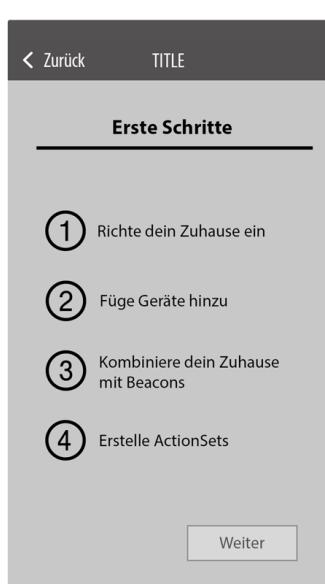


Abbildung 4-4: Hilfe-Themen im Klick-Dummy
(Eigendarstellung)



Abbildung 4-5: Startseite des Klick-Dummys
(Eigendarstellung)

Nach dem Einrichten wird man auf die Startseite geführt, siehe Abbildung 4-5. Dort werden die Räume und Geräte angezeigt. Ebenso die erstellten Aktionen und verknüpften iBeacons. Es wird mit einem Rahmen erkenntlich gemacht, in welchem Raum man sich momentan befindet und es ist direkt erkennbar, welche Geräte aktiv oder inaktiv sind.

Auf der Startseite hat man die Möglichkeit auf zusätzliche Informationen, wie Info-Screens und FAQs, zuzugreifen, ebenso wie das Einsehen des Impressums. Außerdem

gibt es links oben ein Menü, mit dem man die Option erhält, schnell auf bestimmte Kategorien zuzugreifen. Man kann auf alle Homes und Rooms Zugriff erhalten, außerdem auf iBeacons, Geräte und Aktionen. Bei Klick darauf, werden die vorhandenen Elemente in einer Liste angezeigt und man kann weitere Details einsehen.

4.2 Erstellung der Screens

Da der finale Aufbau des Klick-Dummies schon beschrieben wurde, wird nun der Weg bis dorthin beschrieben.

4.2.1 Skizzen auf Karteikarten



Abbildung 4-6: Layout-Skizzen auf Karteikarten
(Eigendarstellung)

Um die Gedanken zum Aufbau einer App und dem Inhalt festzuhalten, sind Skizzen mit Stift und Papier gut geeignet.

Skizzieren auf Papier hat den Vorteil, dass man eine Skizze leichter von Grund auf neu gestaltet. Bei digitalen Skizzen möchte man die Arbeit nicht verwerfen und ist meist eingeschränkt auf das schon erstellte Layout. Des Weiteren sind Skizzen schlicht aufgebaut und besitzen keine Details, somit hat man noch die Möglichkeit weitere Ideen einzubauen.

4.2.2 Nachbauen der Layouts in Photoshop

Die um die 60 skizzierten Layouts wurden anschließend in Photoshop mit iOS-spezifischen Elementen nachgebaut. Im Laufe der Umsetzung entstanden des Öfteren neue Ideen und Verbesserungsmöglichkeiten.

Die daraus resultierenden Screens wurden schließlich für den Klick-Dummy verwendet.

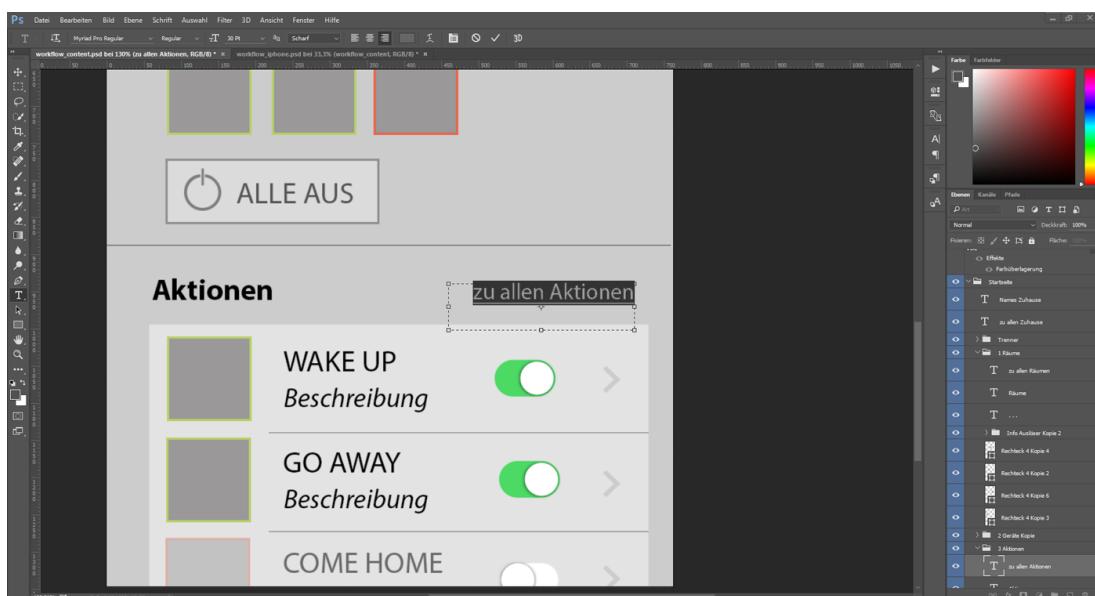


Abbildung 4-7: Arbeiten mit Adobe Photoshop
(Screenshot von Adobe Photoshop)

4.3 Erstellung des Klick-Dummies

Für den Klick-Dummy wurde die Software Marvel²⁷ benutzt. Marvel hat den Vorteil, dass man schon fertige Layouts, als PNG, JPEG und sogar als Photoshop-Datei, hochladen kann.

Auf dem Layout werden dann Schaltflächen erstellt, die mit einem Ziel-Bild verlinkt werden können. Somit kann man die einzelnen Bilder zu einem fortlaufenden Klick-Dummy zusammenbauen. Den Prototypen kann man dann entweder im Browser oder auf einem Smartphone durchklicken. Ein weiterer Vorteil ist, dass man die Bilder über Dropbox synchronisieren lassen kann, während die Verlinkungen erhalten bleiben. Nur, wenn sich die Platzierung eines Buttons im Design ändert, muss man die Schaltfläche noch einmal manuell anpassen.

²⁷ Free mobile & web prototyping | Marvel

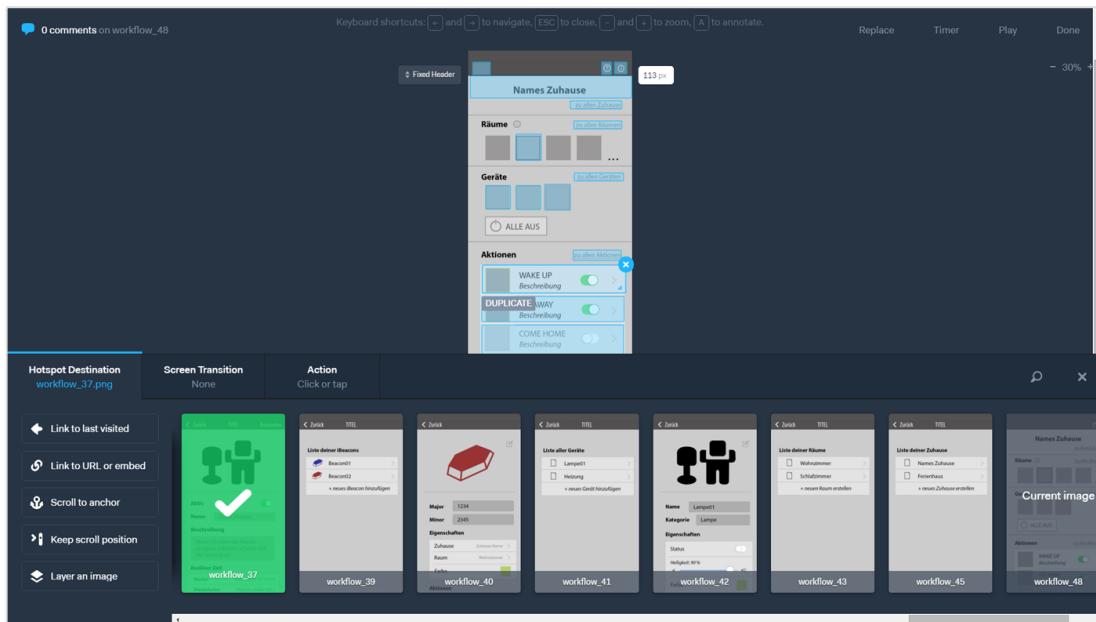


Abbildung 4-8: Arbeiten mit Marvel
(Screenshot von Marvel)

4.4 Verbesserungsvorschläge

Nach der Erstellung des Klick-Dummys wurde Feedback von einigen Personen mit unterschiedlichem Wissensstand über das Thema Home Automation eingeholt. Das Feedback ist im Nachkommenden zusammengefasst.

4.4.1 Erklärungs- und Hilfestellungsseiten

Im Klick-Dummy gibt es eine Hilfestellung für das Einrichten der App. Diese waren in vier Kapitel unterteilt und hatten ungefähr fünf Seiten pro Kapitel. Die Seiten sollten dem Nutzer helfen sein Zuhause einzurichten und Geräte, sowie iBeacons hinzuzufügen. Nach jedem Kapitel gab es eine Bestätigungsseite, mit der man seine eingegebenen Daten überprüfen konnte, um anschließend das Kapitel abzuschließen oder um sie noch einmal zu überarbeiten.

Anmerkungen

Durch die große Anzahl an Unterseiten, verlor der Nutzer den Überblick über den Stand in der Navigation. Es war unklar, wie viele Seiten zu dem Thema noch folgen. Hier wurde angemerkt, möglichst wenig Seitenwechsel einzubauen und eher einen langen Screen zu machen und dort die Punkte untereinander abzufragen. Für die Eingabe wären dann immer ein Pop-Up zu verwenden, wo man den Hintergrund des Screens noch sehen kann und in dem man die gefragten Daten eingeben kann.

Die Bestätigungsseiten sollte man zudem streichen und die Informationen lieber am oberen oder unteren Rand einblenden lassen. Diese verschwinden nach einer gewissen Zeit, wenn man nicht auf sie klickt. So kann man auf eine weitere Seite verzichten.

Eine andere Anmerkung war, dass man eine Unternavigation einbaut, die deutlich macht, wie viele Seiten in dem Kapitel noch kommen. Diese baut sich nach jeder Seite weiter auf, siehe Abbildung 4-9.

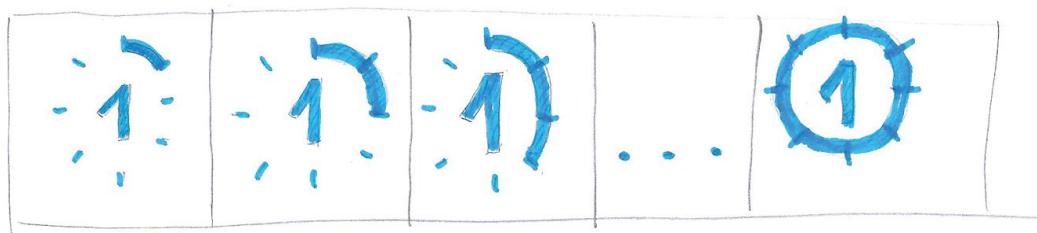


Abbildung 4-9: Aufbau der Navigation im Klick-Dummy
(Eigendarstellung)

Zusätzlich könnte man eine Zeitangabe für jedes Kapitel angeben, wie lange man für diesen Schritt voraussichtlich benötigt. Das hat einerseits den Vorteil, dass der Nutzer weiß was ihn erwartet, aber es kann andererseits abschreckend wirken und den Nutzer die Aktion abbrechen lassen.

Des Weiteren stellte sich die Frage, ob das Thema Action Set überhaupt zum Einrichten der App gehört oder ob es nicht fortgeschrittenere Einstellungen sind und dem Nutzer zu einem späteren Zeitpunkt erklärt werden sollten.

4.4.2 Startseite

Auf der Startseite wurde der Fokus auf die Übersicht über alle Funktionalitäten gelegt. Es gab eine Übersicht über die vorhandenen Räume, Geräte, Aktionen und iBeacons.

Kritik

Bei der Startseite wurde angemerkt, dass der Fokus mehr auf den Geräten und den Räumen liegen soll. Die Geräte sollten direkt auf der Startseite bedienbar sein ohne dass man erst auf die Detail-Seiten der Geräte navigieren muss. Die Umsetzung könnte ähnlich wie bei der Apple Watch aussehen, mit Hilfe der sogenannten *Complications*, siehe Abbildung 4-10. Hier wird aufgrund des geringen Platzes das Layout in festgelegte Muster aufgeteilt und diese werden mit Informationen gefüllt. Dieses Modell bietet dem Nutzer zusätzlich auch mehr Individualität, wenn er sich aussuchen kann, welche Informationen wo und wie groß angezeigt werden sollen.

Die Verbesserungen wurden aufgrund des hohen Aufwandes nicht mehr im Klick-Dummy umgesetzt. Sie wurden aber für die spätere Entwicklung festgehalten.

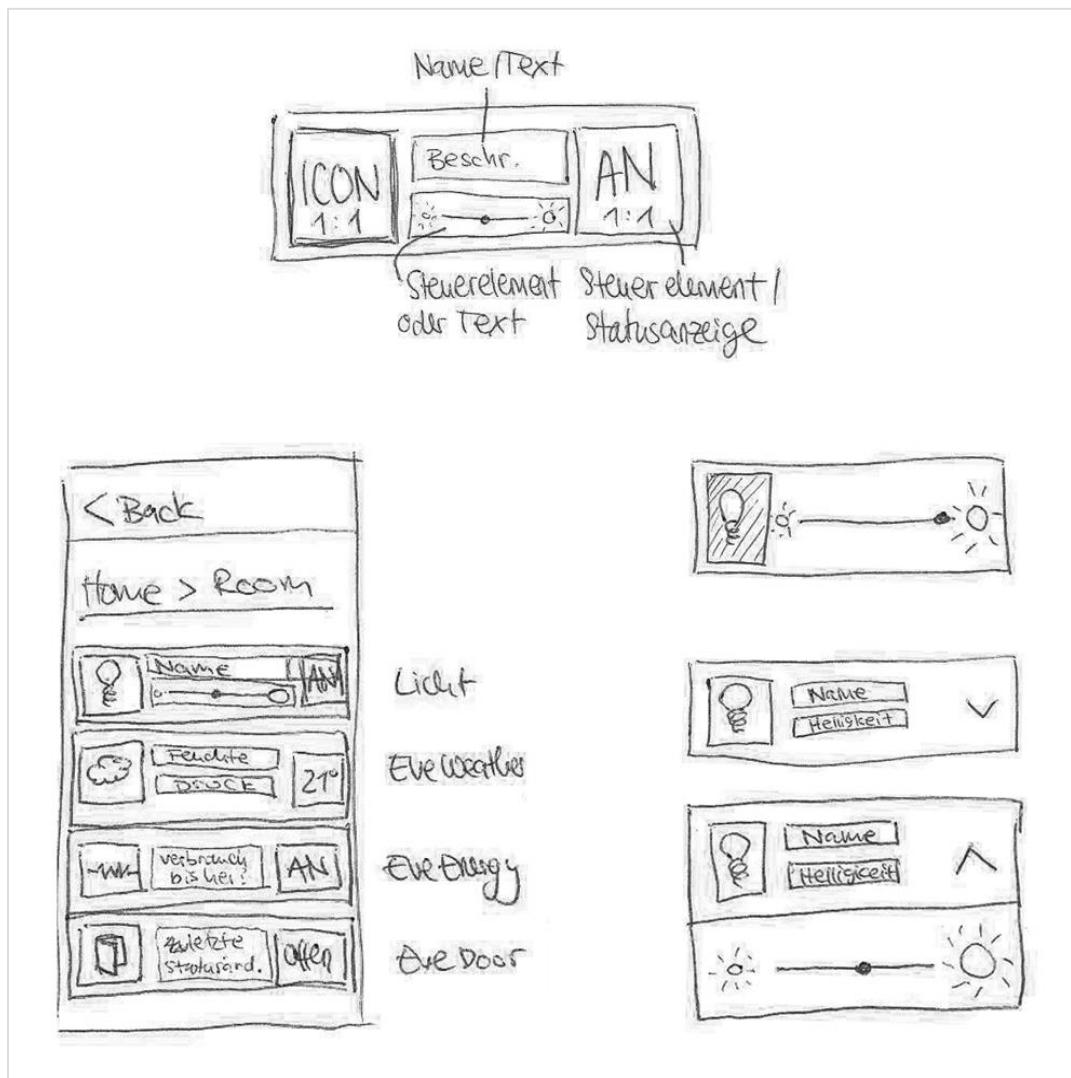


Abbildung 4-10: Skizzen zum Thema Complications
(Eigendarstellung)

5 Umsetzung

Die prototypische Umsetzung wurde damit begonnen, die Startseite zu entwickeln, die die Geräte eines Rooms anzeigt. Neben der Steuerung der Geräte lag hierbei der Fokus auf der Erstellung von Layouts für die verschiedenen Kategorien der Geräte. Erst im Laufe der Entwicklung wurde die Funktion ergänzt, dass man den Raum wechseln kann.

Die Implementierung erfolgte mit Apples Programmiersprache Swift und der zu dem Zeitpunkt der Entwicklung aktuellen Xcode-Version 7.2.1.

In den folgenden Kapiteln geht es um die prototypische Entwicklung von Smart Living.

5.1 Aufbau der App

In diesem Kapitel wird der Aufbau der App beschrieben, wie er im Endzustand der Entwicklung vorzufinden ist. Dies ist wichtig, um einen Überblick über den Aufbau der App zu bekommen. Dafür ist auch das Xcode-Storyboard in Anhang III zu finden.

Die Startseite ist das Zentrum der App, siehe Abbildung 5-1. Diese wird unmittelbar nach dem Starten angezeigt. Direkt am oberen Rand wird der Name des Hauses und des aktuellen Zimmers angezeigt. Rechts daneben befindet sich ein nach unten

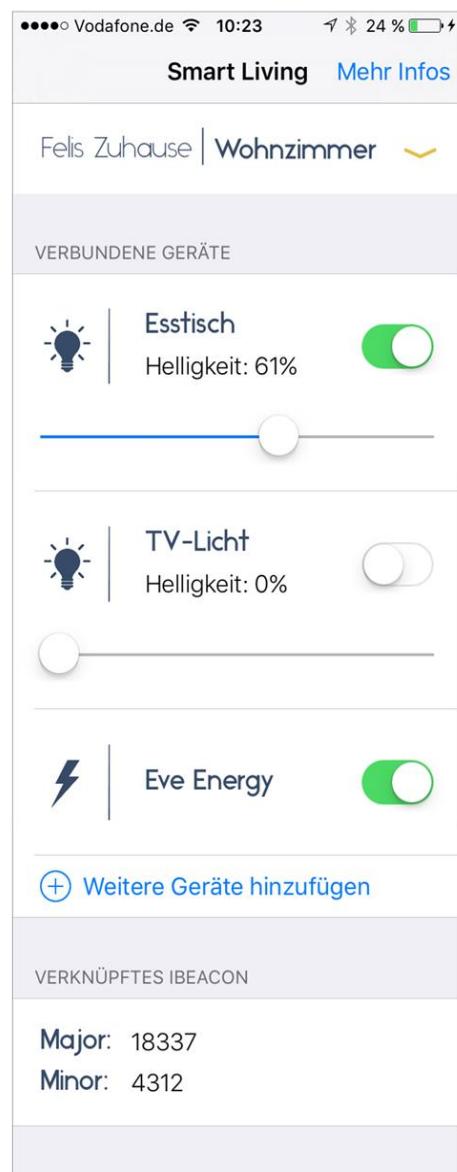


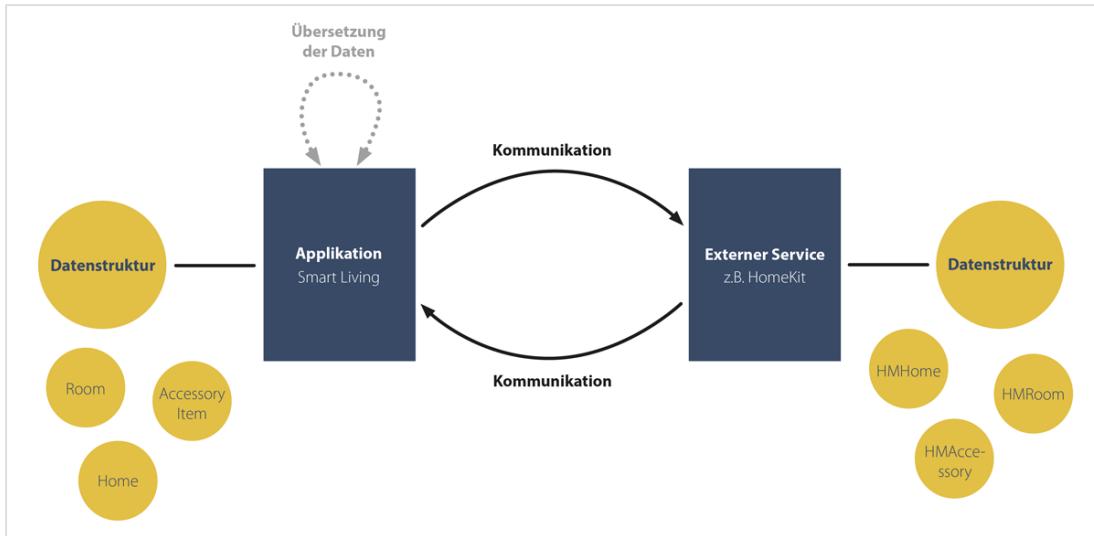
Abbildung 5-1: Startseite im Endzustand der Entwicklung
(Eigendarstellung)

gerichteter, gelber Pfeil, mit dem man zu einem anderen Raum navigieren kann.

Unter der Kopfzeile befinden sich zwei Blöcke. Im oberen Teil befindet sich die Sektion *Verbundene Geräte*. Hier werden alle verbundenen Geräte aufgelistet, die sich in dem ausgewählten Raum befinden. Jede Gerätekategorie hat ein eigenes Design, da sie unterschiedliche Elemente besitzen und sich somit im Aufbau voneinander unterscheiden. Die zweite Sektion nennt sich *Verknüpftes iBeacon*. Hier gibt es Informationen darüber, ob ein iBeacon mit diesem Raum verknüpft wurde oder ob man eins hinzufügen kann. Wenn ein iBeacon verknüpft ist, dann wird die entsprechende *Major* und *Minor* angezeigt, die zur Identifikation eines iBeacons dient. Wenn das Gegenteil der Fall ist, wird die Möglichkeit geboten mit einem Hinzufügen-Button ein neues iBeacon zu kombinieren.

Mit dem in der Navigation befindlichen Button *Mehr Infos* gelangt der Nutzer an weiteren Informationen. Man wird auf eine Übersichtsseite geleitet, die die Menüpunkte *Über die App*, *Info-Galerie*, *FAQs*, *Einstellungen* und *Impressum* aufweist.

5.2 Trennung der Datenstrukturen



*Abbildung 5-2: Trennung der Datenstrukturen
(Eigendarstellung)*

Beim Arbeiten mit externen Services ist es sinnvoll, die Abhangigkeiten von den Services zu minimieren. Das ist notwendig, damit bei anderungen im Service nicht die vollstandige Applikation, sondern nur die Schnittstelle angepasst werden muss. Deswegen wird die Datenstruktur der externen Services in der Applikation in eine eigene interne Struktur ubersetzt, was zur Verdeutlichung in Abbildung 5-2 dargestellt wurde.

Für die Übersetzung zwischen den Datenstrukturen gibt es mehrere Funktionen, die aus den Daten des eingehenden Objekts ein neues und für die jeweilige Datenstruktur passendes Objekt erstellen oder es aus dem Speicher auslesen und zurückgeben.

Die Trennung der Datenstrukturen war ein großer Bestandteil der Arbeit und wird im Laufe der kommenden Kapitel noch weiter erläutert.

5.3 Aufbau der Klassen-Struktur

Um die Trennung der Strukturen umsetzen zu können, gibt es in der Applikation eine Klasse, namens `ContextHandler`, die die externen Services verwaltet und der App die entsprechenden Daten zur Verfügung stellt. Der `ContextHandler` ist die Schnittstelle zwischen den Services und der internen App-Struktur.

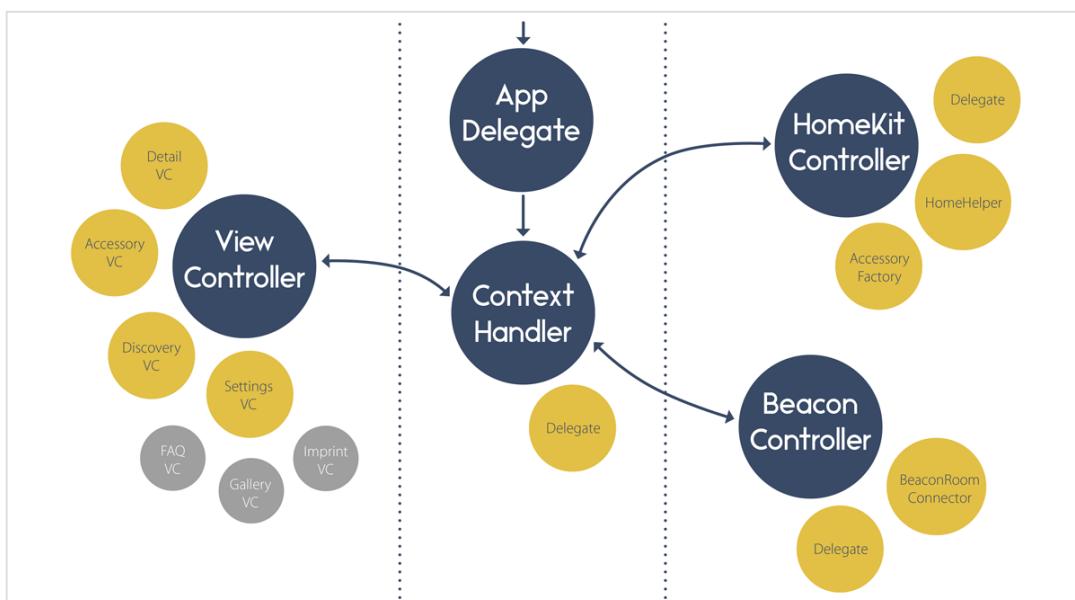


Abbildung 5-3: Klassen-Übersicht
(Eigendarstellung)

In Abbildung 5-3 ist eine grobe Übersicht der Klassen abgebildet. Im linken Abschnitt sind die `ViewController` dargestellt. Diese kontrollieren die Views, die es in der App gibt. Der wichtigste Controller ist der `DetailViewController`, der die Startseite repräsentiert.

Auf der rechten Seite befinden sich die externen Services für HomeKit und die iBeacons, die jeweils in einer eigenen Klasse verwaltet werden. HomeKit wird im `HomeKitController` bearbeitet und die iBeacon-Technologie im `BeaconController`.

5.4 HomeKit Service

Um HomeKit in das Projekt einzubinden, muss man zuerst einige Einstellungen in Xcode vornehmen. Wie in Abbildung 5-4 in fünf Schritten beschrieben, muss in den Projekteinstellungen unter *Capabilities* HomeKit aktiviert werden.

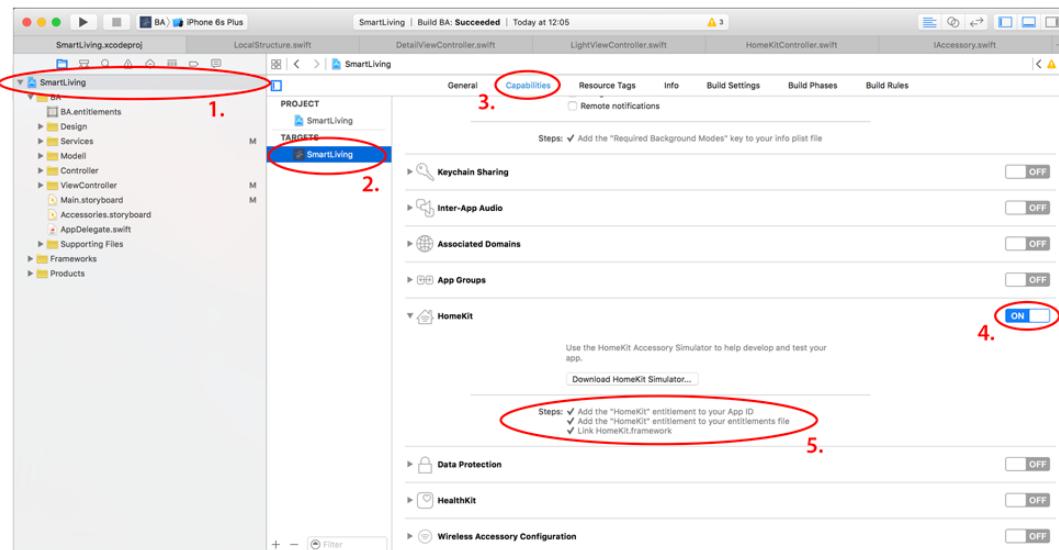


Abbildung 5-4: Aktivieren von HomeKit
(Screenshot von Xcode)

Dabei müssen die folgenden drei Punkte in Schritt 5 zwingend erfüllt sein.

1. Einbinden des Frameworks in das Projekt
2. Hinzufügen des *HomeKit-Entitlements* zur App-ID
3. Hinzufügen des *HomeKit-Entitlements* zu den *Entitlement-Files*

Um nun auf das HomeKit-Framework zugreifen zu können, muss man in den Klassen, die auf den HomeKit-Service zugreifen sollen, `import HomeKit` angeben.

Im Folgenden werden nun die Schritte beschrieben, die beim Entwickeln mit dem HomeKit-Service durchlaufen werden.

5.4.1 Laden der HomeKit-Daten

Wie bereits in Kapitel 5.1: *Aufbau der App* beschrieben, soll die Startseite den Namen des Homes und des Rooms mit den zugehörigen Geräten anzeigen. Dafür müssen die HomeKit-Daten für den Home und den Room aus der `common database` ausgelesen werden, was in der Klasse `HomeKitController` durchgeführt wird.

Der `HMHomeManager` verwaltet die `common database` und kann die Home-Einstellungen auslesen und ändern. Dafür muss jedoch das `Delegate` gesetzt werden.

```
1 homeManager.delegate = self
```

Code-Auszug 1: `homeManager.delegate`

Nun muss der Callback `homeManagerDidUpdateHomes` vom `HomeManager` abgewartet werden, der die gespeicherten HomeKit-Daten zur Verfügung stellt.

```
1 func homeManagerDidUpdateHomes(manager: HMHomeManager) {
2
3     //1 Set homes
4     homes = manager.homes
5
6     //2 Set rooms
7     for home in homes {
8         rooms += home.rooms
9     }
10
11    let homeWithoutRoom = homes.filter{ $0.rooms.isEmpty }
12
13    if homes.isEmpty || rooms.isEmpty {
14
15        //3 if home or room is empty, create one default home and room
16        initialHomeSetup("Default Home", roomName: "Default Room")
17
18    } else if !homeWithoutRoom.isEmpty {
19
20        //4 find home without room, create one default room
21        for home in homeWithoutRoom {
22            initialHomeSetup(home.name, roomName: "Default Room")
23        }
24
25    } else {
26        homesAreSet()
27        roomsAreSet()
28    }
29
30    [...]
31}
```

Code-Auszug 2: `func homeManagerDidUpdateHomes(manager: HMHomeManager)`

Wenn der Callback aufgerufen wird, ist der `HomeManager` mit dem Bereitstellen der Home-Daten fertig. Dieser Zeitpunkt ist ideal, um die Homes vom `homeManager` in eine interne Variable `homes` zu speichern, siehe Code-Auszug 2 in Zeile 4. Wenn keine HomeKit-Daten zur Verfügung stehen, werden mithilfe der Funktion `initialHomeSetup` ein *Default Home* und ein *Default Room* angelegt, siehe Code-Auszug 2 in Zeile 13. Darüber wird der Nutzer mit einer Mitteilung – auch *Alert* genannt – informiert.

5.4.2 Umwandlung in interne Datenstruktur: Homes und Rooms

In der folgenden Abbildung 5-5 werden die Unterschiede der Datenstrukturen in HomeKit und der internen Struktur aufgezeigt. Deutlich wird, dass die interne Struktur Home und Room reduziert wurde.

HMHome	Home	HMRom	Room
uniqueIdentifier: NSUUID	id: NSUUID	uniqueIdentifier: NSUUID	id: NSUUID
name: String	name: String	name: String	name: String
primary: Bool	primary: Bool	accessories: [HMAccessory]	homelD: NSUUID
accessories: [HMAccessory]			
rooms: [HMRom]			
zones, serviceGroups, actionSets, triggers			

Abbildung 5-5: Home und Room – Datenstrukturen
(Eigendarstellung)

Für die Umwandlung der HomeKit-Struktur in die interne Struktur gibt es eine Klasse `HomeHelper`. Diese stellt mehrere Funktionen zur Verfügung, die `HMHome` und `HMRom` in `Home` und `Room` umwandeln können und umgekehrt.

In Code-Auszug 3 wird die Funktion `serviceToLocalHomes`, die die HomeKit-Homes in eine interne Struktur Homes umwandelt, beschrieben. Als Parameter besitzt sie ein Array von `HMHome`s und gibt nach der Übersetzung ein Array aus internen `Home`s zurück.

Mit `Home(id: home.uniqueIdentifier, name: home.name, primary: home.primary)`, siehe Zeile 4, wird für jedes HomeKit-Home ein neues internes Home erstellt. Die Attribute der ID, des Namens und der Angabe über das `primaryHome` werden dabei vom `HMHome` übernommen.

```

1 func serviceToLocalHomes(homeKitHomes: [HMHome]) -> [Home]? {
2     localHomes = []
3     for home in homeKitHomes {
4         localHomes?.append(Home(id: home.uniqueIdentifier, name: home.name,
5 primary: home.primary))
6     }
7     return localHomes
8 }
```

Code-Auszug 3: `func serviceToLocalHomes(homeKitHomes: [HMHome]) -> [Home]?`

5.4.3 Laden der bereits verbundenen Accessories

Im Folgenden wird anhand des Sequenzdiagramms in Anhang IV das Vorgehen zum Laden der bereits verbundenen Accessories erklärt.

Nachdem der DetailViewController geladen wurde, stellt er eine Anfrage an den ContextHandler (Schritt 24, 27, 30). Dieser soll ihm den Home- und Room-Namen zurückgeben und das Laden der dazugehörigen Accessories starten. Da der ContextHandler das aktuelle Home und Room gespeichert hat, kann er den Namen für die beiden direkt zurückgeben (Schritt 26 und 29). Um die Accessories zu laden, muss der ContextHandler mit dem HomeKitController kommunizieren. Mit der Funktion `retrieveAccessoryForRoom` (Schritt 32) und der `homeID` und `roomID` werden vom homeManager die verbundenen Accessories für den Raum ausgelesen (Schritt 35). Das Array aus HMAccessories wird nun gemappt.

```
1 let localPairedAccessories: [AccessoryItem] = homeKitAccessories!.map{
2     createAccessoryItem($0)
3 }
```

Code-Auszug 4: let localPairedAccessories: [AccessoryItem]

Das bedeutet, für jedes Element aus dem Array wird die Funktion `createAccessoryItem` ausgeführt. Diese wandelt die HMAccessories in AccessoryItems um und speichert sie in einem Array namens `pairedAccessories`, welches direkt an den ContextHandler weitergegeben wird (Schritt 36).

Im ContextHandler wird das `pairedAccessories` beobachtet und bei Änderungen des Arrays die Funktion `retrieveViewControllerList` aufgerufen, die ein Array mit ViewControllern passend zu dem Accessory-Typ zusammenstellt.

```
1 var pairedAccessories : [AccessoryItem] = [] {
2     didSet {
3         if !pairedAccessories.isEmpty {
4             viewControllerArray = retrieveViewControllerList()
5         }
6     }
7 }
```

Code-Auszug 5: var pairedAccessories : [AccessoryItem]

Dabei werden alle Elemente von `pairedAccessories` in der Funktion `assignAccessoryToViewController` nach ihrem Typ gefiltert, siehe Code-Auszug 6 in Zeile 2, und ein passender ViewController wird initialisiert. Diesem wird das Accessory für die Steuerung direkt mitgegeben.

```
1 func assignAccessoryToViewController (accessory: AccessoryItem) ->
2     UIViewController? {
3         switch accessory {
4             case is Lamp:
```

```
5     let controller =
6 accessoryStoryboard?.instantiateViewControllerWithIdentifier("LightViewController")
7 as! LightViewController
8     controller.accessory = accessory
9     controller.contextHandler = self
10    return controller
11 case is WeatherStation:
12     let controller =
13 accessoryStoryboard?.instantiateViewControllerWithIdentifier("WeatherViewController")
14 ) as! WeatherViewController
15     controller.accessory = accessory
16     controller.contextHandler = self
17     return controller
18 case is EnergyController:
19     let controller =
20 accessoryStoryboard?.instantiateViewControllerWithIdentifier("EnergyViewController")
21 ) as! EnergyViewController
22     controller.accessory = accessory
23     controller.contextHandler = self
24     return controller
25 case is DoorWindowSensor:
26     let controller =
27 accessoryStoryboard?.instantiateViewControllerWithIdentifier("DoorWindowViewController")
28 ) as! DoorWindowViewController
29     controller.accessory = accessory
30     controller.contextHandler = self
31     return controller
32 case is Diverse, is Information:
33     let controller =
34 accessoryStoryboard?.instantiateViewControllerWithIdentifier("DiverseViewController")
35 ) as! DiverseViewController
36     controller.accessory = accessory
37     controller.contextHandler = self
38     return controller
39 default:
40     return nil
41 }
42 }
```

Code-Auszug 6: `func assignAccessoryToViewController (accessory: AccessoryItem) -> UIViewController?`

Wenn das `viewControllerArray` fertig zusammengestellt wurde, wird eine Benachrichtigung an den `DetailViewController` gesendet, der dann aus dem Array die Tabelle für die Anzeige der verbundenen Accessories füllen kann. Bei jeder Änderung des `viewControllerArrays`, wird die Tabelle aktualisiert.

Die Zellen der Tabelle zeigen jeweils ein Accessory an. Der View, der in der Zelle angezeigt werden soll, wird aus dem jeweiligen passenden `ViewController` geladen. Dies ist notwendig, da die Accessories unterschiedliche Layouts und Elemente besitzen. Ein Licht hat beispielsweise einen Slider zum Steuern der Helligkeit und einen Switch zum Ändern des Status. Dessen View wird aus dem `LightViewController` geladen. Die Wetterstationen besitzen währenddessen gar keine steuerbaren Elemente, sondern nur eine Anzeige über die Temperatur, Luftfeuchtigkeit und Luftdruck. Das Layout wird aus dem `WeatherViewController` geladen. In Code-Auszug 7 wird in `cellForRowAtIndexPath` dieses Vorgehen umgesetzt.

```
1 override func tableView(tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
2     [...]
3     let vcInRow = viewControllerArray[row]
4     let cell = tableView.dequeueReusableCell(withIdentifier: "accessoryCell")!
5     var view: UIView?
6
7     switch vcInRow {
8     case is LightViewController:
9         view = vcInRow.view as! LightView
10        break
11    case is WeatherViewController:
12        view = vcInRow.view as! WeatherView
13        break
14    case is EnergyViewController:
15        view = vcInRow.view as! EnergyView
16        break
17    case is DoorWindowViewController:
18        view = vcInRow.view as! DoorWindowView
19        break
20    case is DiverseViewController:
21        view = vcInRow.view as! DiverseView
22        break
23    default: break
24    }
25
26    view!.frame = cell.contentView.frame
27
28    if !cell.contentView.subviews.isEmpty {
29        for subview in cell.contentView.subviews {
30            subview.removeFromSuperview()
31        }
32    }
33
34    cell.contentView.addSubview(view!)
35
36    return cell
37    [...]
38 }
39 }
```

Code-Auszug 7: `override func tableView(tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell`

5.4.4 Umwandlung in interne Datenstruktur: Accessories

Ebenso wie die interne Struktur der `Homes` und `Rooms`, ist auch die Struktur der `AccessoryItems` reduziert, gegenüber der Struktur von HomeKit.

Für ein `AccessoryItem` ist eine eindeutige ID, ein beschreibender Name, die Erreichbarkeit, dessen Characteristics und ein Block notwendig.

HMAccessory	AccessoryItem
uniqueIdentifier: NSUUID	uniqueID: NSUUID
name: String	name: String
reachable: Bool	reachable: Bool
room: HMRoom	characteristics: [CharacteristicKey: AnyObject]
services: [HMServices]	characteristicBlock: () -> ()
room, bridged, blocked, category	

Abbildung 5-6: Accessory – Datenstruktur
(Eigendarstellung)

Die ID dient als eindeutiger Schlüssel und der Name ist für Siri relevant. Die Erreichbarkeit ist wichtig, damit der Nutzer informiert werden kann, wenn die Kommunikation mit Geräten nicht funktioniert, da beispielsweise das Bluetooth ausgeschalten ist oder man außer Reichweite ist. Für die Speicherung der Characteristics wurden eigene Keys angelegt, siehe Code-Auszug 8. Die Keys wurden angelegt, um Schreibfehler in den Strings zu vermeiden.

```

1 enum CharacteristicKey: String {
2     case serviceName = "serviceName"
3     case brightness = "brightness"
4     case powerState = "powerState"
5     case temperature = "temperature"
6     case humidity = "humidity"
7     case pressure = "pressure"
8     case doorState = "doorState"
9 }
```

Code-Auszug 8: enum CharacteristicKey: String

Der CharacteristicBlock wird gebraucht, da die Characteristics asynchron ausgelesen werden. Mit dem Block wird überprüft, ob Characteristics schon ausgelesen wurden. Wenn nicht, dann wird der Block gesetzt und ausgeführt, wenn die fehlenden Characteristics anschließend ausgelesen werden.

Für die AccessoryItems wurde das *Factory-Pattern* angewendet. Das bedeutet, es gibt ein Interface, das festlegt, welche Funktionen und Attribute die Unterklassen implementieren müssen. Die Unterklassen können dann individuell entscheiden, wie sie damit umgehen. Das ist von Vorteil, da Lichter ihre Characteristics anders verarbeiten als Wetterstationen. Es gibt die Unterklassen Lamp, WeatherStation, EnergyController, DoorWindowSensor, Information und Diverse.

In Code-Auszug 9 sieht man die `canHandle`-Funktion aus der `Lamp`-Klasse. Diese gibt nur dann `true` zurück, wenn der `ServiceType` ein `HMSERVICETypeLightbulb` ist.

```

1 func canHandle(service: HMService, name: String?) -> Bool {
2     if service.serviceType == HMSERVICETypeLightbulb {
3         return true
4     } else {
5         return false
6     }
7 }
```

Code-Auszug 9: func canHandle(service: HMService, name: String?) -> Bool

Mit der `canHandle`-Funktion kann also über den Service des HomeKit-Accessories abgefragt werden, zu welcher Unterklasse das `HMAccessory` gehören soll.

Mit der Funktion `createAccessoryItem` wird ein `HMAccessory` in ein `AccessoryItem` umgewandelt, siehe Code-Auszug 10. In Zeile 9 wird eine Variable `newAcc` für ein `AccessoryItem` angelegt. Dieser wird noch der Name, die ID und das Attribut, der Erreichbarkeit gegeben, welche vom HomeKit-Accessory übernommen werden.

```

1 func createAccessoryItem(homeKitAccessory: HMAccessory) -> AccessoryItem {
2     var hmService : HMService?
3     let hmName = homeKitAccessory.name
4
5     //set delegate to detect changes to HMAccessory
6     homeKitAccessory.delegate = self
7
8     hmService = retrieveHMServices(homeKitAccessory)
9     var newAcc = accessoryFactory.accessoryForServices(hmService!, name: hmName)!
10    newAcc.name = hmName
11    newAcc.uniqueID = homeKitAccessory.uniqueIdentifier
12    newAcc.reachable = homeKitAccessory.reachable
13    return newAcc
14 }
```

Code-Auszug 10: func createAccessoryItem(homeKitAccessory: HMAccessory) -> AccessoryItem

Das Erstellen eines `AccessoryItems` funktioniert mit der Funktion `accessoryForServices` aus der `AccessoryFactory`-Klasse, die ein `AccessoryItem` des entsprechenden Typs zurückgibt. Hier wird über den `HMServices` entschieden, zu welchem Typ das Accessory gehören soll. Anschließend wird ein neues Objekt dieses Typs instanziert und zurückgegeben.

```

1 func accessoryForServices(service: HMService, name: String?) -> AccessoryItem? {
2     arrayOfType = [lampService, weatherStationService,
3     energyControllerService, doorWindowSensorService, informationService,
4     diverseService]
5
6     let canHandleServiceAccessory = arrayOfType!.filter{
7         $0.canHandle(service, name: name)
8     }.first
9
10    //create a new instance of this type
11    switch canHandleServiceAccessory {
12        case is Lamp:
```

```

13     return Lamp()
14     case is WeatherStation:
15         return WeatherStation()
16     case is EnergyController:
17         return EnergyController()
18     case is DoorWindowSensor:
19         return DoorWindowSensor()
20     case is Information:
21         return Information()
22     case is Diverse:
23         return Diverse()
24     default: return nil
25 }
26 }
```

Code-Auszug 11: func accessoryForServices (service: HMService, name: String?) -> AccessoryItem?

5.4.5 Laden der Characteristics von Accessories

Die Characteristics werden in HomeKit asynchron geladen, da nicht immer garantiert werden kann, dass das Laden schnell genug abläuft. Wenn Geräte nicht erreichbar sind, könnte das beispielsweise den Prozess blockieren.

Das Laden der Characteristics wird am Beispiel des manuellen Aktualisierens beschrieben.

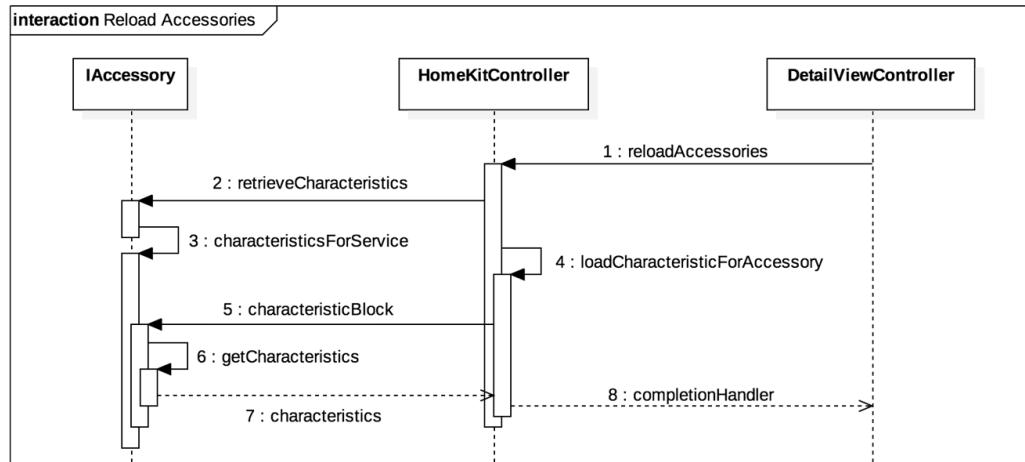


Abbildung 5-7: Sequenzdiagramm – Reload Accessories
(Eigendarstellung)

Im Folgenden sieht man die Funktion `reloadAccessories` aus dem `HomeKitController`.

```

1 func reloadAccessories(completionHandler: () -> ()) {
2     var hmService: HMService?
3     var localPairedAccessories: [AccessoryItem] = []
4     var accessoryViews: [Bool] = []
5
6     for var acc in pairedAccessories! {
7         //1 find HMService for AccessoryItem
8         let hmAcc = getHMAccessory(acc)
9         hmService = retrieveHMServices(hmAcc)
10    }
```

```

11     //2 loading characteristic for AccessoryItem started
12     acc.retrieveCharacteristics(hmService!)
13
14     loadCharacteristicForAccessory(acc, completionHandler: { accessory in
15
16         if let acc = accessory {
17             localPairedAccessories.append(acc)
18             accessoryViews.append(self.completedAccessoryView(acc))
19         }
20
21         if self.homeKitAccessories?.count == localPairedAccessories.count {
22             self.pairedAccessories = localPairedAccessories
23
24             if !accessoryViews.contains(false) {
25                 completionHandler()
26             }
27         }
28     })
29
30 }
31

```

Code-Auszug 12: func reloadAccessories(completionHandler: () -> ())

In Zeile 12 wird das Laden der Characteristics mit `retrieveCharacteristics` angestoßen. In dieser Funktion werden dann die `characteristicsForService` für das Accessory ausgelesen, was am Beispiel der Klasse `Lamp` gezeigt wird. Das Laden wird nun asynchron im Hintergrund ausgeführt.

```

1 func characteristicsForService(service: HMService, completionHandler:
2 [CharacteristicKey : AnyObject] -> () ) {
3     characteristics.removeAll()
4
5     for characteristic in service.characteristics {
6
7         if characteristic.characteristicType == (HMCharacteristicTypePowerState
8 as String) {
8         getCharacteristicValue(characteristic, completion: { value, error in
9             if let value = value {
10                 self.characteristics[CharacteristicKey.powerState] = value
11             as! NSNumber
12                 if self.characteristics.count ==
13 service.characteristics.count {
14                     completionHandler(self.characteristics)
15                 }
16             }
17         })
18     }
19
20     [...]
21 }
22
23

```

Code-Auszug 13: func characteristicsForService(service: HMService, completionHandler: [CharacteristicKey : AnyObject] -> ())

In Zeile 9 wird die HomeKit-Funktion `getCharacteristicValue` aufgerufen, um die Werte auszulesen. Diese werden, wenn sie geladen wurden, dem passenden `CharacteristicKey` zugewiesen.

Nachdem das Laden angestoßen wurde, siehe Code-Auszug 12, wird die Funktion `loadCharacteristicForAccessory` aufgerufen, die in Code-Auszug 14 beschrieben wird. Hier wird zuerst abgefragt, ob das Accessory erreichbar ist. Wenn es nicht erreichbar ist, wird gar nicht versucht die Characteristics zu laden. Ansonsten wird überprüft, ob es schon Characteristics für das Accessory gibt. Wenn nicht, wird ein `characteristicBlock` gesetzt, siehe Zeile 8. Dieser soll bei Ausführung die Characteristics erneut versuchen auszulesen. Wenn die Characteristics bereits geladen werden konnte, sollen diese dem Accessory gesetzt werden.

```

1 func loadCharacteristicForAccessory(var accessory: AccessoryItem,
2 completionHandler: (AccessoryItem?) -> ()) {
3     if accessory.reachable == false {
4         completionHandler(accessory)
5     } else {
6         // for every AccessoryItem check if its characteristics is empty
7         if accessory.characteristics.isEmpty {
8             accessory.characteristicBlock = { () in
9                 // get and save loaded characteristics
10                accessory.characteristics = accessory.getCharacteristics()!
11                // save accessories with characteristics in pairedAccessories
12                dispatch_async(dispatch_get_main_queue()) {
13                    completionHandler(accessory)
14                }
15            }
16        } else {
17            // get and save loaded characteristics
18            accessory.characteristics = accessory.getCharacteristics()!
19            // save accessories with characteristics in pairedAccessories
20            dispatch_async(dispatch_get_main_queue()) {
21                completionHandler(accessory)
22            }
23        }
24    }
}

```

Code-Auszug 14: func loadCharacteristicForAccessory (var accessory: AccessoryItem, completionHandler: (AccessoryItem?) -> ())

Wenn im Hintergrund das Laden der Accessories gelungen ist, wird in `retrieveCharacteristics` geprüft ob der `CharacteristicBlock` gesetzt wurde. Wenn der Block gesetzt wurde, wird dieser ausgeführt und wieder auf `nil` gesetzt.

```

1 mutating func retrieveCharacteristics(service: HMService) {
2     characteristicsForService(service, completionHandler: { characteristics in
3         [...]
4         self.characteristics = characteristics
5
6         if let block = self.characteristicBlock {
7             block()
8             self.characteristicBlock = nil
9         }
10    })
11 }

```

Code-Auszug 15: mutating func retrieveCharacteristics(service: HMService)

Da die Characteristics nun erfolgreich geladen wurden, können diese im Block mit `getCharacteristics`, siehe Code-Auszug 16, abgerufen werden.

```

1 func getCharacteristics() -> [CharacteristicKey:AnyObject]? {
2     if !characteristics.isEmpty {
3         return characteristics
4     } else {
5         return nil
6     }
7 }
```

Code-Auszug 16: func getCharacteristics()

5.4.6 Steuerung der Accessories

Um das Benutzen von Steuerelementen in den Views registrieren zu können, benötigen die `AccessoryViews` mit steuerbaren Elementen ein Delegate. Dieses wird aktiviert, wenn sich beispielsweise der Slider ändert. Das Delegate übergibt den neuen Slider-Wert an den passenden `AccessoryViewController`. Dort wird auch der Text in der App angepasst, „Helligkeit: 77%“, siehe Code-Auszug 17 in Zeile 3 und die aktualisierte Characteristic an den `HomeKitController` weitergegeben, siehe Zeile 4 und 5.

```

1 func accViewSliderChanged(value: Float) {
2     [...]
3     lightView!.brightness!.text = "Helligkeit: \(Int(value))%"
4     contextHandler!.homeKitController!.setNewValues(accessory!, characteristic:
5             [.brightness:value])
6 }
```

Code-Auszug 17: func accViewSliderChanged(value: Float)

In der Funktion `setNewValues` im `HomeKitController` muss die `HMCharacteristic` für das `HMAccessory` gefunden werden, damit die neue Characteristic an das Gerät gesendet werden kann.

Dafür wird zuerst das `HMAccessory` zum veränderten `AccessoryItem` gesucht, siehe Code-Auszug 18 in Zeile 7. Anschließend wird der `HMCharacteristicType` für den `CharacteristicKey` gesucht, siehe Zeile 10 und folgende. Dieser wird gebraucht um den korrekten Typ des übergebenen Wertes zu bestimmen, also die Umwandlung von `AnyObject` zu `Int`, `Bool` oder ähnlichem. Das ist notwendig, um den geänderten Wert in die entsprechende `HMCharacteristic` schreiben zu können, siehe Zeile 28.

```

1 func setNewValues(accessory: AccessoryItem, characteristic: [CharacteristicKey :
2 AnyObject]) {
3     let characteristicKey = characteristic.map{ $0.0 }.first!
4     var characteristicValue = characteristic.map{ $0.1 }.first!
5
6     //1 find HMAccessory for AccessoryItem
7     let hmAccessory = getHMAccessory(accessory)
8 }
```

```

9         //2 find HMCharacteristicType for CharacteristicKey
10        let homeKitType = dictKeyToHomeKitType!.filter{ $0.0 == characteristicKey
11        }.first.map{ $0.1 }
12
13        //3 change CharacteristicValue to correct type
14        if homeKitType == (HMCharacteristicTypeBrightness as String){
15            characteristicValue = characteristicValue as! Int
16        } else if homeKitType == (HMCharacteristicTypePowerState as String){
17            characteristicValue = characteristicValue as! Bool
18        }
19
20        //4 find HMCharacteristic to write new value
21        let hmService = hmAccessory.services.filter{ ($0.serviceType ==
22        HMServiceTypeOutlet) || ($0.serviceType == HMServiceTypeLightbulb) }.first
23        let characteristic = hmService!.characteristics.filter{
24        $0.characteristicType == homeKitType }.first
25
26        //5 write new value on HMCharacteristic
27        characteristic!.writeValue(characteristicValue, completionHandler: {
28            error in
29            if let error = error {
30                NSLog("Failed to update value \(error)")
31            }
32        })
33    }

```

Code-Auszug 18: func setNewValues(accessory: AccessoryItem, characteristic: [CharacteristicKey : AnyObject])

So wird ein durch den Nutzer geänderter Wert an HomeKit übergeben und dort in die richtige `HMCharacteristic` geschrieben.

5.4.7 Arbeiten mit dem HomeKit Accessory Simulator

Neben den realen HomeKit kompatiblen Geräten, können Entwickler auch mit dem HomeKit Accessory Simulator entwickeln und testen. Dort können simulierte Geräte erstellt werden. Die Entwicklung für die Steuerung der Simulator-Geräte ist identisch zu der Steuerung von echten Geräten. So müssen keine Änderungen gemacht werden, wenn man vom Entwickeln mit dem Simulator zum Arbeiten mit echten Geräten wechselt. Die im Simulator angelegten Accessories verhalten sich genauso wie echte Geräte.

Neue Accessories können über den Plus-Button links unten im Simulator angelegt werden. Dafür muss man einen Namen und einen Service angeben. Alle Accessories haben standardmäßig schon den `InformationService` eingerichtet. Es können mehrere Services aus einer Liste hinzugefügt werden und auch mehrere Characteristics.

Um ein Simulator-Gerät mit der App zu verknüpfen, muss der Setup Code, der unter dem Accessory Namen sichtbar ist, in die App eingegeben werden. Bei echten Geräten ist der Setup Code meist am Gerät befestigt.

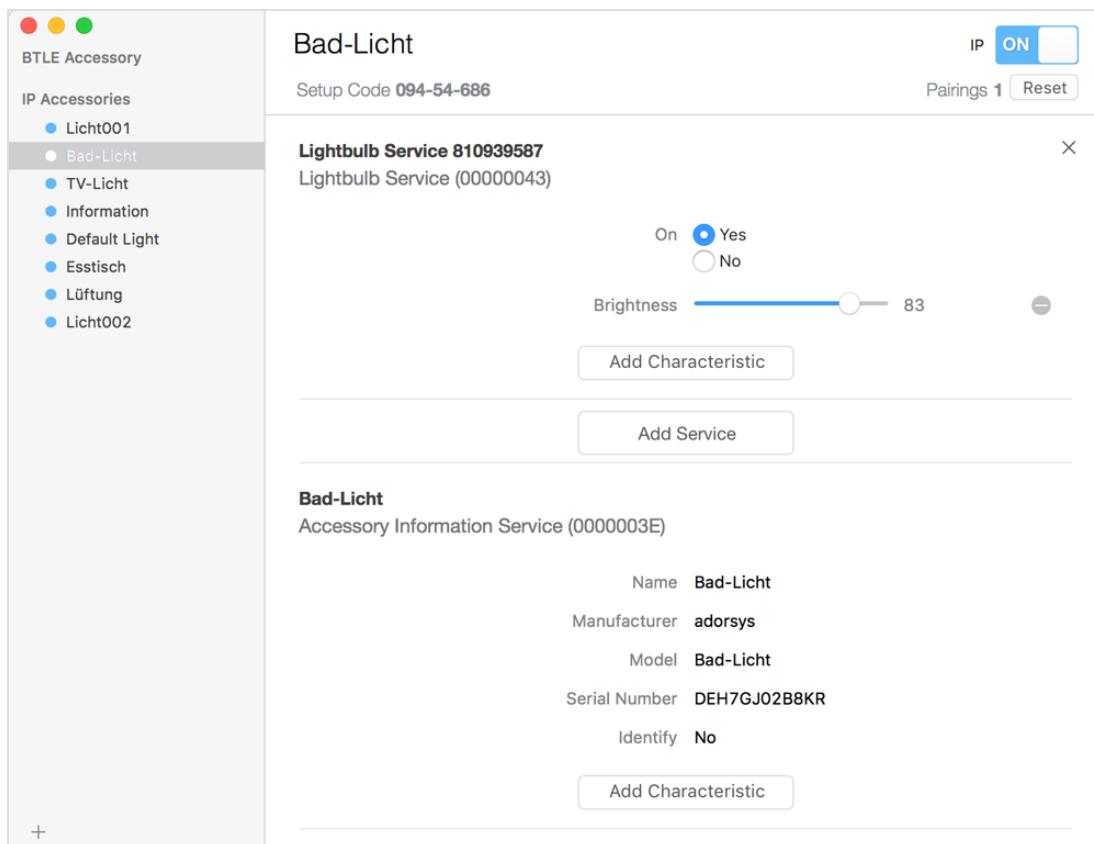


Abbildung 5-8: HomeKit Accessory Simulator
(Screenshot von Simulator)

5.5 iBeacon Service

iBeacons werden verwendet, um zu erkennen, wann man einen Raum betritt oder verlässt und dann dem Nutzer Arbeit abzunehmen, indem bestimmte Aktionen durchgeführt werden, wie beispielsweise das Ausschalten der Lichter beim Verlassen des Raumes.

Die Funktionen der iBeacons werden über das CoreLocation SDK implementiert. Nachdem ein CLLocationManager initialisiert wurde, wird das Delegate gesetzt.

```
1 locationManager.delegate = self
```

Code-Auszug 19: `locationManager.delegate`

5.5.1 Autorisierung

Der erste Schritt ist die Autorisierung. Der Nutzer wird beim Installieren der App gefragt, ob die App auf den Standort zugreifen darf und ob sie das auch im Hintergrund darf. Das ist bei iBeacons notwendig, da es dann möglich ist gewisse

Aktionen auszuführen, selbst wenn die App geschlossen ist. Beispielsweise kann man Mitteilungen an den Nutzer schicken, wenn man in die Nähe eines iBeacons kommt.

5.5.2 Start Monitoring

Nachdem die Autorisierung über das Lesen des Standorts erfolgreich abgeschlossen wurde, muss noch geprüft werden, ob das Gerät fähig ist iBeacons zu erkennen.

Wenn das zutrifft, wird das Monitoring gestartet, siehe Zeile 4, ebenso wie das Suchen nach iBeacons in der Region, siehe Zeile 7.

```
1 func startMonitoringInRegion (region: CLBeaconRegion) {  
2     // 3 Start monitoring for region  
3     locationManager.startMonitoringForRegion(region)  
4  
5     // 4 Start ranging beacons in region  
6     locationManager.startRangingBeaconsInRegion(region)  
7  
8     monitoringStarted = true  
9 }  
10 }
```

Code-Auszug 20: func startMonitoringInRegion (region: CLBeaconRegion)

5.5.3 Location Manager Delegates

Anschließend werden noch einige Delegate-Methoden vom LocationManager implementiert.

Die `didStartMonitoringForRegion` und `didDetermineState` sind für Änderungen im Suchprozess zuständig. Die Erste wird verwendet, wenn eine neue Region gesucht wird. Die Zweite wird aufgerufen, wenn sich der Abstand zum iBeacon ändert. Hier kann festgelegt werden, dass man erneut die Suche starten soll, wenn man sich innerhalb der Region befindet und dass die Suche beendet werden soll, wenn man sich außerhalb der Region befindet. Mithilfe der Funktion `didChangeAuthorizationStatus` wird überprüft, ob sich der Autorisierungsstatus geändert hat. Das kann vorkommen, wenn der Nutzer das Lesen seines Standorts deaktiviert hat. Die wichtigste Delegate Methode ist die `didRangeBeacons`-Methode, siehe Code-Auszug 21. Diese wird aufgerufen, wenn ein iBeacon entdeckt wurde. Hier wird entschieden, was ausgeführt werden soll, wenn ein iBeacon gefunden wurde. Zusätzlich gibt es noch einige Error-Callbacks. Davon sind in Smart Living zwei implementiert, einerseits die `monitoringDidFailForRegion`-Funktion und andererseits die `rangingBeaconsDidFailForRegion`-Funktion. Hier werden programminterne Meldungen ausgegeben, was den Fehler hervorgerufen hat.

Näher wird nun auf die didRangeBeacons-Methode eingegangen.

```

1 func locationManager(manager: CLLocationManager, didRangeBeacons beacons:
2 [CLBeacon], inRegion region: CLBeaconRegion) {
3
4     //1 filter ranged beacons
5     let knownBeacons = beacons.filter{ $0.proximity != CLProximity.Unknown }
6
7     if !knownBeacons.isEmpty {
8         //2 set closest beacon
9         let closestBeacon = knownBeacons[0]
10
11         //3 check if last beacon is closest beacon
12         if closestBeacon.major != lastBeacon?.major && closestBeacon.minor != lastBeacon?.minor {
13
14             //4 delegate with information which beacon was found
15             delegate?.beaconFound(self, major:
16             closestBeacon.major.integerValue, minor: closestBeacon.minor.integerValue)
17
18             //5 set closestBeacon as last found beacon
19             lastBeacon = closestBeacon
20
21         }
22     }
23 }
```

Code-Auszug 21: func locationManager(manager: CLLocationManager, didRangeBeacons beacons: [CLBeacon], inRegion region: CLBeaconRegion)

Als erstes wird das Array von iBeacons gefiltert, indem alle Beacons mit der Entfernung Unknown aussortiert werden, siehe Zeile 5, da diese uninteressant sind. Wenn nach der Filterung noch iBeacons vorhanden sind, wird das erste iBeacon im Array als das closestBeacon gesetzt. Nun wird überprüft, ob es dasselbe iBeacon ist, das als lastBeacon gesetzt wurde. Wenn es kein lastBeacon gibt, springt man automatisch in die Abfrage. Hier wird ein Delegate aktiviert, das den ContextHandler benachrichtigt, welches iBeacon gefunden wurde.

5.5.4 Ablauf nach Finden eines iBeacon

In Anhang V ist ein Sequenzdiagramm abgebildet, das den Vorgang beim Finden eines iBeacons definiert. Dieses wird im Folgenden erklärt.

In Schritt 1-7 wird der BeaconController initialisiert und die Suche nach iBeacons gestartet. In Schritt 19 wird ein iBeacon gefunden. Das Delegate wird im ContextHandler verarbeitet. Hier wird die *Major* und *Minor* gespeichert und ein Delegate für den ViewController ausgelöst. Dieser ist dafür zuständig, dass dem Nutzer ein Alert mit der Information, dass ein iBeacon gefunden wurde, angezeigt werden kann. In dem Alert soll auch angezeigt werden, zu welchem Home und Room das iBeacon gehört, wenn es schon verknüpft wurde. Dafür muss in Schritt 21, das connectorArray nach den mitgegebenen *Major* und *Minor* gefiltert werden. Mit dem passenden Home und Room als String wird nun ein weiteres delegate

ausgelöst und im `HomeKitController` wird nach den IDs für den Home und Room gesucht (Schritt 22). Diese werden als `completionHandler` zurückgegeben. Im `HomeKitController` wird dafür das gespeicherte Array nach den mitgegebenen Namen gefiltert. Es wird auch der Erfolgsstatus zurückgegeben. Wenn es zu beiden Namen ein Ergebnis gibt, wird `success = true` und die IDs mitgegeben. Wenn keine `homeID` oder keine `roomID` für die mitgegebenen Namen gefunden wurde, liefert der Status `success = false` zurück.

Nur wenn der Status erfolgreich ist, siehe Code-Auszug 22 in Zeile 3, und der Raum des iBeacons nicht der aktuelle Raum ist, wird im `DetailViewController` ein Alert mit der Nachricht „*Bist du im >Raum< in >Home<? Willst du die dafür relevanten Geräte sehen?*“ angezeigt.

```

1 controller!.findHMRoomForBeacon(plistHome, room: plistRoom) { success, homeID,
2   roomID in
3     if success {
4       if self.room != plistRoom {
5         self.alertShowBeaconRoom(homeID!, roomID: roomID!, message: "Bist du
6 im >\(plistRoom)< in >\(plistHome)<? Willst du die dafür relevanten Geräte sehen?")
7       }
8     }
9 }
```

Code-Auszug 22: `findHMRoomForBeacon(plistHome, room: plistRoom)`

Wenn dem Alert zugestimmt wird, dass der Raum gewechselt werden soll, wird der View gesperrt, ein Ladesymbol angezeigt, siehe Zeile 24 in Anhang V und die `homeID` und `roomID` an den `ContextHandler` gegeben, siehe Zeile 25. Der `DetailViewController` bekommt eine Nachricht, dass sich der Raum geändert hat, siehe Zeile 26 und lädt die entsprechenden Daten, siehe Zeile 27. Damit wurde der Raum gewechselt und der View wird der Benutzung wieder freigegeben, siehe Zeile 28.

5.5.5 Verbindung von HomeKit und iBeacons

In diesem Abschnitt wird genauer erklärt, wie die Verbindung zwischen HomeKit und iBeacon hergestellt und gespeichert wird. Ein iBeacon repräsentiert hierbei einen `HMRoom`. Um die Verbindung zwischen einem iBeacon und einem Raum zu speichern, gibt es die `BeaconRoomConnector`-Klasse.

```

1 class BeaconRoomConnector {
2
3   let major: Int
4   let minor: Int
5   let home: String
6   let room: String
7
8   init (major: Int, minor: Int, home: String, room: String) {
9     self.major = major
10    self.minor = minor
11 }
```

```

11     self.home = home
12     self.room = room
13 }
14
15 init (dict: NSDictionary) {
16     self.major = dict["major"] as! Int
17     self.minor = dict["minor"] as! Int
18     self.home = dict["home"] as! String
19     self.room = dict["room"] as! String
20 }
21
22 var description: String {
23     return "Major: \(major), Minor: \(minor), Home: \(home), Room: \(room)"
24 }
25
26 func dictionaryForEntity() -> NSDictionary {
27     return [ "major" : major, "minor" : minor, "home" : home, "room" : room ]
28 }
29 }
```

Code-Auszug 23: class BeaconRoomConnector

Diese speichert die Major und Minor des iBeacons mit dem `home` und `room` von HomeKit, beides als Strings, siehe Code-Auszug 23 in Zeile 3-6. Die Verbindung wird dann als Array von `BeaconRoomConnector`-Objekten in einer `info.plist` gespeichert. Eine `Plist` ist eine Liste mit Informationen, welche auf dem Smartphone gespeichert wird. Man kann neue Informationen in sie hineinschreiben und sie auslesen. Bei einer Neuinstallation der App wird diese Liste jedoch zurückgesetzt. Der `ContextHandler` lädt die Daten aus der `Plist` und schreibt dort alle neuen Verbindungen hinein.

Mit den beschriebenen Funktionen werden das Suchen nach iBeacons und das Reagieren auf gefundene iBeacons implementiert.

5.6 Finaler Funktionsumfang vom Prototypen

In diesem Kapitel gibt es eine Zusammenfassung über die Funktionen, die während der Entwicklung implementiert wurden. Einige davon wurden schon in vorherigen Kapiteln genauer beschrieben, hier aber für die Vollständigkeit trotzdem aufgenommen.

Auslesen der HomeKit-Daten

Für Smart Living ist es irrelevant, ob HomeKit-Daten verfügbar sind oder nicht. Man kann die App also über einen iPhone-Simulator oder über ein iPhone mit mindestens iOS 8.1 laufen lassen, dass im Apple Developer Programm teilnimmt. Wenn keine HomeKit-Daten gefunden wurden, werden automatisch ein Default Home und ein Default Room erstellt, siehe Kapitel 5.4.1. In diesem Fall wird über ein Alert Bescheid gegeben, dass keine Daten gefunden wurden. Man kann jedoch manuell keine neuen Homes und Rooms hinzufügen, bearbeiten oder löschen.

Trennung der Daten

Die Daten der Services werden getrennt von den Daten aus der App. Das bedeutet, dass die HomeKit Accessories in AccessoryItems umgewandelt werden, siehe Kapitel 5.4.4. Ebenso gibt es bei den HomeKit Homes und Rooms eine Umwandlung in interne Homes und Rooms, siehe Kapitel 5.4.2.

Anzeige von Raum und zugehörigen Accessories

Wenn man die App öffnet, erhält man eine Übersicht über die Accessories aus einem Raum. In der obersten Zeile sieht man den Namen des Homes und des Rooms in dem man sich gerade in der App befindet.

Darunter ist eine Liste mit der in diesem Raum verbundenen Accessories. Jede Zeile hat eine angepasste Höhe, je nach Layout. Die letzte Zeile enthält einen Plus-Button, mit dem man neue Accessories hinzufügen kann.

Unter der Sektion der Accessories befindet sich die Sektion der iBeacons. Wenn kein iBeacon verbunden ist, befindet sich dort ein Button zum Hinzufügen eines Beacons. Im anderen Fall wird dort die *Major* und *Minor* angezeigt.

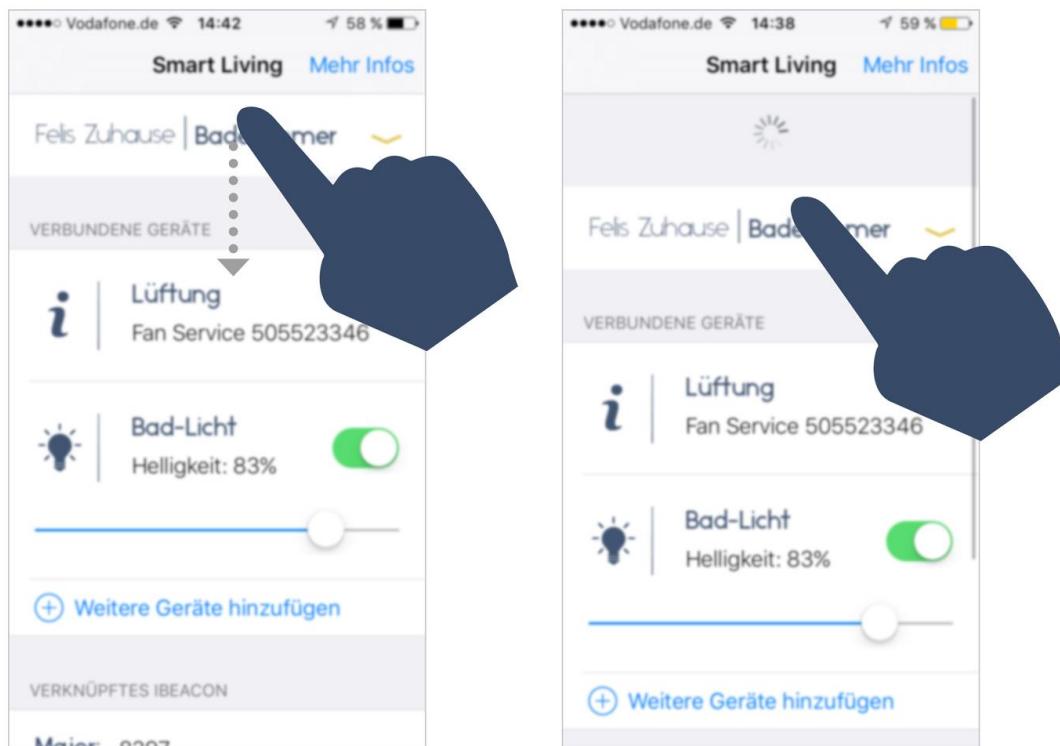


Abbildung 5-9: Gestenerklärung Pull-To-Refresh
(Eigendarstellung)

Um eine neue Version dieser Anzeige zu erhalten, ist ein Pull-To-Refresh eingebaut, siehe Abbildung 5-9. Das ist eine Geste, bei der man den oberen Rand nach unten

zieht. Danach sieht man einen Lade-Indikator. Erst wenn alle Characteristics geladen wurden und anzeigenbereit sind, hört der Indikator auf mit Laden und die Anzeige verschiebt sich wieder nach oben.

Anangepasste Anzeige für unterschiedliche Accessory-Arten

Für verschiedene Typen an Accessories gibt es ein eigenes Design: Licht, Weather, Energy, Door & Window, Information und Diverse.

Für die im Simulator erstellten Lichter werden die Characteristics: Helligkeit, An/Aus ausgelesen und angezeigt mit einem Slider und einem Switch. Hier gibt es noch eine zusätzliche interne Logik. Wenn der Switch auf Aus gesetzt wird, dann zeigt der Slider auch eine Helligkeit von 0% an. Der Helligkeitswert wird aber gespeichert und wenn man das Licht wieder anschaltet, wird der vorherige Wert gesetzt. Wenn es generell keine Helligkeitssteuerung gibt, dann wird der Slider ausgeblendet.

Für den Eve Energy gibt es einen Switch um die An/Aus Characteristic anzuzeigen und sie darüber zu steuern.

Beim Eve Weather kann man momentan die Temperatur und die Luftfeuchtigkeit auslesen. Diese werden angezeigt und sind aber nicht steuerbar. Wie man den Luftdruck ausliest, ist mir noch nicht klar, da es sich um Custom Characteristics handelt. In der App wird für den Wert des Luftdrucks momentan noch eine feste Zahl angezeigt.

Bei dem Eve Door & Window kann man auslesen, ob der Zustand offen oder geschlossen ist. Hier gibt es keine Möglichkeit eine Mitteilung zu bekommen, wenn sich der Status ändert. Das wird von Eve Door & Window noch nicht unterstützt. Es kann momentan also keine Mitteilung auf Smartphones verschickt werden, wenn ein Fenster geschlossen wurde. Ebenso gibt es einen Zähler, der mitzählt, wie oft sich der Status schon geändert hat. Diese Zahl wird intern auf dem Smartphone gespeichert, wird aber zurückgesetzt, wenn man die App neu installieren sollte.

Für alle diese Typen gibt es momentan ein Design. Wenn ein Gerät hinzugefügt wird, das in keine dieser Kategorien passt, wird der Typ Diverse ausgewählt und es wird nur der Name des Geräts und des Services angezeigt. Alle Accessories können anzeigen, dass sie außer Reichweite sind.

Hinzufügen von Accessories

Der Nutzer kann jedem Raum neue Accessories hinzufügen. Nach einem Klick auf „+ Weitere Geräte hinzufügen“, wird man auf einen weiteren Screen geleitet, der neue Accessories in einer Liste anzeigt, siehe Abbildung 5-10 auf Seite 60. Die Liste wird automatisch ergänzt. Aus der Liste kann man sich ein Accessory auswählen und es mit einem Klick hinzufügen. Nach dem HomeKit spezifischen Ablauf des Hinzufügens,

werden die Characteristics für das Accessory automatisch geladen. Das hinzugefügte Accessory wird bei erneutem Klick auf Hinzufügen nicht mehr in der Tabelle angezeigt. Das Anzeigen von realen Geräten in der Liste funktioniert ebenso wie das Hinzufügen.

Auslesen der Accessories

Es können Characteristics von verbundenen Accessories ausgelesen werden. Bei externen Änderungen über den HomeKit Accessory Simulator werden die Characteristics und auch die Anzeige dementsprechend aktualisiert. Bei den Elgato-Produkten funktioniert die Benachrichtigung über veränderte Werte nicht, von daher muss eine automatische Aktualisierung nach X Minuten eingerichtet werden. Momentan kann man über die Geste Pull-To-Refresh die Werte neu laden.

Eve Geräte besitzen Custom Services

Die Eve-Geräte besitzen neben dem Informationsservice und einigen HomeKit-Services auch Custom Services. Diese können momentan noch nicht ausgelesen werden. Deswegen muss der Service gefunden werden, der auslesbare Characteristics besitzt, um sie in der App anzuzeigen.

Steuern der Accessories

Man kann hinzugefügte Accessories steuern, wenn sie steuerbare Elemente besitzen. Das funktioniert mit Simulator-Geräten und auch mit dem Eve Energy, der das bei bisher einzige steuerbare reale HomeKit-Gerät ist. Auch die Steuerung über Siri funktioniert.

iBeacon-Erkennung und Reaktion

Die App kann iBeacons finden, wenn sie in Reichweite sind und Bluetooth eingeschalten ist. Bei Erkennung eines iBeacons gibt es einen Alert, ob man zu dem gefundenen Raum wechseln möchte. Wenn man den Raum wechselt, wird die Ansicht des Raumes aktualisiert. Die Verknüpfung von iBeacon und Raum wird in einer `Info.plist` gespeichert. Wenn einem Raum noch kein iBeacon zugeordnet wurde, kann man über ein „+ iBeacon hinzufügen“ das nächste iBeacon hinzufügen. Momentan wird noch nicht geprüft, ob das iBeacon noch nicht verknüpft wurde. Wenn einem Raum ein iBeacon schon zugeordnet wurde, wird es mit Angabe der *Major* und *Minor* angezeigt. Die Möglichkeit ein iBeacon hinzuzufügen gibt es dann nicht mehr.

Wechseln des Home und des Rooms

In der obersten Zeile gibt es neben der Anzeige der Home und Room-Namen einen kleinen gelben Pfeil, der bei Klick ein Alert Sheet aufruft, mit dem man den Raum wechseln kann. Anschließend werden die neuen Accessories geladen.

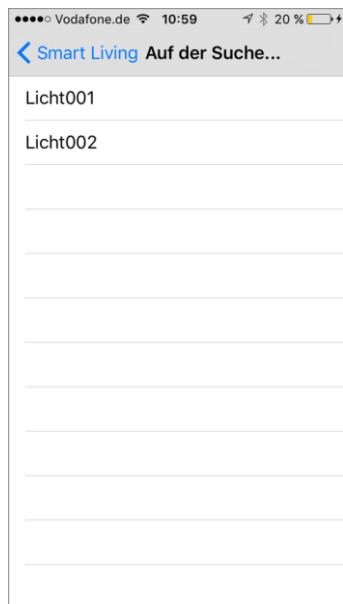


Abbildung 5-10: Seite mit noch nicht verbundenen Accessories (Screenshot von Smart Living)



Abbildung 5-11: Seite mit ergänzenden Informationen (Eigendarstellung)



Abbildung 5-12: Seite mit Info-Galerie (Screenshot von Smart Living)

Ergänzende Inhalte

In der Navigation gibt es auf der Startseite einen Button *Mehr*. Bei Klick darauf, landet man auf einer Übersichtsseite mit ergänzenden Inhalten.

Eine der Seiten ist eine Informationsseite über das Projekt Smart Living. Dort wird aufgezeigt, in welchem Rahmen die App entstanden ist, siehe Abbildung 5-11.

Eine Galerie mit Info-Screens ist ein weiterer Punkt der ergänzenden Inhalte. Diese Screens wurden für den Klick-Dummy entworfen und sind nun auch in der echten Applikation sichtbar. Auf den Screens findet man eine Übersicht über verwendete Technologien und kurze Erklärungen, siehe Abbildung 5-12.

Eine FAQ-Seite, auf der eine Auswahl an Fragen und Antworten angezeigt wird, wurde auch implementiert. Für eine technisch basierte Applikation ist das wichtig, damit das Verständnis für die Themen gelegt wird. Hier gibt es auch eine Möglichkeit nach Stichwörtern zu suchen und die Meldungen nach diesem Begriff zu filtern. Das dient der Übersichtlichkeit und dem schnellen Finden von Begriffen.

Um zu gewährleisten, dass der Nutzer schnell zu den App-Einstellungen gelangt, gibt es einen Link zu den Einstellungen. Dort kann man den Zugriff auf die Home-Einstellungen verwalten, ebenso wie die Einstellungen für die iBeacons, ob auf den Standort zugegriffen werden darf.

Der letzte Punkt ist das Impressum. Hier wurden Telefonnummern, Adressen und E-Mail-Adressen verlinkt, sodass man bei Klick direkt zu den dazugehörigen Aktionen geführt wird. Bei Klick auf eine Straße wird man beispielsweise zu der Karten App weitergeleitet, bei der man sehen kann, wo sich die angegebene Adresse befindet.

Umsetzung des Designs

Das App Icon, siehe Abbildung 3-7, wurde für die verschiedenen Device-Größen eingepflegt. Außerdem wurde das Logo auf einem schlichten Hintergrund in den Launch Screen eingesetzt. Außerdem wurden die Icons für die verschiedenen Accessory Typen angepasst und minimalisiert.

Autolayout

In der App-Entwicklung wurde darauf geachtet, die Screens mit Autolayout zu bauen. Das bedeutet, dass sich der View automatisch an die Inhalte anpasst und dabei definierte Regeln, sogenannte *Constraints*, einhält. Das Design kann somit auch für den Landscape-Modus festgelegt werden. Somit ist es möglich, sich die App im Portrait und Landscape-Modus ansehen und die Inhalte werden richtig angezeigt.

5.7 Ausblick

Es gibt einige Punkte, die das Projekt noch verbessern und erweitern würden. Vor Allem die in dem Kapitel Usability angesprochenen Punkte, würden die Nutzerfreundlichkeit deutlich erhöhen.

Generell gilt, je mehr Geräte es in der nächsten Zeit gibt, desto mehr Anwendungsfälle sind für die App denkbar.

Im Folgenden sind mehrere Aspekte als Ausblick beschrieben worden.

HomeKit

Es gibt noch Funktionen, die von HomeKit bereitgestellt werden, aber aufgrund der Zeitbegrenzung noch nicht eingebaut wurden, wie die Action Sets (mit iBeacons, Geofencing und Uhrzeiten als Auslöser) oder das Erstellen von Zonen. Dies ist einer der wichtigsten Punkte für die App und sollte als Erstes bearbeitet werden.

Der Nutzer sollte auch die Möglichkeit bekommen, Accessories aus einem Raum zu löschen. Genauso, wie das Hinzufügen und Löschen von Homes und Rooms noch nicht möglich ist.

Da die Eve Geräte beispielsweise keine Mitteilung senden, wenn sich ein Wert geändert hat, sollte man ungefähr alle 10 Minuten die Characteristics automatisch neu laden.

Ein weiterer wichtiger Punkt ist das Auslesen von Custom Services und Characteristics. Eve hat einige davon, wie den Luftdruck oder den Stromverbrauch. Bisher ist es mir nicht gelungen.

Bisher ist das Hinzufügen von Gerätetypen begrenzt, diese Typen sollen erweitert werden, genauso wie die dazugehörigen Characteristics. Beispielsweise soll der Farbwert einer Lampe ausgelesen werden.

Zusätzlich sollte das Projekt erweitert werden, indem neue Gerätetypen möglich sind und auch weitere Characteristics.

Individualität

Um das Thema der Individualität in der App umsetzen zu können, muss eine Möglichkeit geschaffen werden, dass die vom Nutzer angelegten Layouts, gespeichert werden und nicht nach jedem Neustart der App verloren sind. Dafür ist Core Data zur Datenspeicherung notwendig. Hier kann beispielsweise die Reihenfolge der angezeigten Accessories gespeichert werden, ebenso wie die anzuzeigenden Informationen für jedes Accessories.

iBeacon

Das Thema iBeacon ist noch nicht voll ausgeschöpft. Hier könnte man ändern, dass nicht über einen aufdringlichen Alert gefragt wird, ob die Anzeige gewechselt werden soll, sondern über einen iBeacon-Button, der farbig aufleuchtet, wenn man in der Nähe eines Beacons ist. Eine andere Möglichkeit wäre, dass eine Benachrichtigung in den Screen fliegt, die nach einer Zeit wieder verschwindet.

Der Nutzer soll die Möglichkeit bekommen, ein verknüpftes iBeacon einem anderen Raum zuzuweisen, oder einfach die Verbindung zu löschen.

Zusätzlich sollen die iBeacon-Informationen ausgegraut werden, wenn kein iBeacon gefunden wurde oder Bluetooth ausgeschalten ist, somit bekommt der Nutzer Feedback darüber, dass die Funktion momentan nicht genutzt werden kann.

Navigation

Die Navigation für die App könnte auch angepasst werden. Es wäre beispielsweise möglich den Raum zu wechseln, indem man nach links und rechts wischt.

Eine Navigation wie im Klick-Dummy würde einen schnellen Zugriff auf die unterschiedlichen Kategorien bieten. Des Weiteren sollte es für ein Accessory dann eine Detail-Seite geben, in der zusätzlich Informationen angezeigt werden und das Accessory auch bearbeitet werden kann. Die Steuerung über die Startseite darf aber nicht aufgegeben werden.

Schließlich ist zu überlegen, ob es, wie im Klick-Dummy, noch die Option auf eine Hilfestellung zum Einrichten der App geben soll.

Schnellzugriff

Der in dem Kapitel der Usability besprochene Schnellzugriff soll mithilfe der Mitteilungszentrale umgesetzt werden. Dort soll es die Möglichkeit geben, Szenen zu aktivieren oder alle Geräte an- oder auszustellen. Ebenso soll der Nutzer die Möglichkeit bekommen, zu entscheiden, was er dort angezeigt bekommt.

Weiteres

Die Fragen auf der FAQ-Seite sollen zum Auf- und Zuklappen sein. So würde man leichter einen Überblick bekommen, wie viele Fragen es insgesamt gibt.

6 Zusammenfassung und Fazit

Das Ziel der Arbeit war die Konzeption und die prototypische Umsetzung einer Home Automation-Applikation für iOS-Geräte. Die im Kapitel der Produktvision genannten Punkte wurden erarbeitet und es wurde definiert, welchen Fokus man bei der Entwicklung in Hinsicht der Usability einer Home Automation-Applikation legen sollte. Das Ziel in der anschließenden prototypischen Umsetzung war die Steuerung von HomeKit-kompatiblen Geräten unter Einbindung der erarbeiteten Usability-Punkte.

Der gesamte Aufbau der Arbeit beschreibt den systematischen Weg zu einem Produkt. Das Einlesen in die Grundlagen der Home Automation diente der Vorbereitung und dem tieferen Einstieg in das Thema. Danach begann die Produktvision, hier wurde ein Konzept erstellt mit allen wichtigen Funktionen, die sich aus der Recherche ergaben. Die Umsetzung in einem Klick-Dummy vertiefte die konzeptionellen Ideen und es wurden neue Probleme und Verbesserungsvorschläge wahrgenommen. Dies diente der Vorbereitung für die letztendliche Umsetzung.

Die in der Produktvision unter Usability beschriebenen Punkte der Individualität und des Schnellzugriffs wurden noch nicht umgesetzt. Die damit erhoffte Nutzerfreundlichkeit konnte somit nicht erreicht werden. Jedoch sind die Punkte definiert und somit bereit für die Implementierung. In Zahlen ausgedrückt, wurden erst drei von zehn Stufen einer guten Usability erreicht, indem die Geräte direkt über die Startseite steuerbar sind, man den Raum mithilfe von iBeacons wechseln kann und das Hinzufügen von Geräten einfach aufgebaut ist.

Insgesamt ist festzuhalten, dass bereits mehrere wichtige Funktionalitäten umgesetzt werden konnten, womit ich sehr zufrieden bin.

Schließlich wird noch ein Fazit zu einigen Aspekten gezogen.

Erstes Projekt

Aus meinem ersten eigenen Projekt konnte ich viele Erfahrungen sammeln, unter anderem wie aufwändig die Erstellung eines Konzepts ist und auf welche Probleme man erst bei der Entwicklung stoßen kann, an die man vorher nicht gedacht hat.

Vertiefen und Erweitern von Swift-Kenntnissen

Die Entwicklung war eine sehr intensive Übung für meine Swift-Kenntnisse. Neben dem Arbeiten mit Closures, Completion Handlern wurden auch Grundkenntnisse, wie das Filtern und Mappen eines Arrays vertieft. Das Implementieren von Collection Views und gruppierten Table Views mit unterschiedlichen Sektionen konnte wiederholt werden, ebenso das Arbeiten mit Table View Headern. Auch im Erstellen einer Bildergalerie und einer Suche wurde mein Wissensstand erweitert.

Trennung der Datenstrukturen

Die Trennung der Datenstrukturen hat einen Großteil der Entwicklung beansprucht. Ohne viel Aufwand hätte man dieselbe Funktionalität für den Nutzer bereitstellen können, aber mit dem Unterschied, dass man von den Services abhängig ist. Im Nachhinein betrachtet war die Trennung aber trotzdem gut investierte Zeit. Das Projekt bleibt so leichter erweiterbar und auch bei Änderungen eines Services, reduziert sich die anstehende Arbeit.

Erstellung von Sequenzdiagrammen

Die Übersichten der Klassenkommunikation wurden erst am Schluss der Entwicklung erstellt. Diese wären im Laufe der Entwicklung sehr hilfreich gewesen, da ich aufgrund der steigenden Komplexität gelegentlich den Überblick verloren hatte.

Aufgrund der Erstellung der Übersichten überarbeitete ich einige Code-Fragmente, da die Klassen-Kommunikation umständliche Schritte beinhaltete.

Erfolgreiche Unterstützung

Abschließend bedanke ich mich bei der Firma *adorsys* für die kompetente Betreuung während der Bachelor-Arbeit, vor allem von Steffen Blümm. Besonders hervorheben möchte ich die Entscheidungsfreiheit, bezüglich des Themas und der Realisierung.

a. Literaturverzeichnis

Angetestet: Elgato Eve (HomeKit) | Macerkopf.de (2015). Online verfügbar unter <http://www.macerkopf.de/2015/08/30/angetestet-elgato-eve-homekit/>, zuletzt aktualisiert am 30.08.2015, zuletzt geprüft am 08.02.2016.

Anwesenheitserkennung bei mehreren Bewohnern | HomeKit.tips (2016). Online verfügbar unter <http://homekit.tips/anwesenheitserkennung-bei-mehreren-bewohnern/>, zuletzt aktualisiert am 17.02.2016, zuletzt geprüft am 18.02.2016.

Apple HomeKit und das Smart Home der Zukunft | meintechblog.de (2015). Online verfügbar unter <http://www.meintechblog.de/2015/12/apple-homekit-und-das-smart-home-der-zukunft/>, zuletzt aktualisiert am 22.12.2015, zuletzt geprüft am 17.02.2016.

Eve Landing | elgato.com (2015). Online verfügbar unter <https://www.elgato.com/en/eve>, zuletzt aktualisiert am 06.11.2015, zuletzt geprüft am 06.11.2015.

Free mobile & web prototyping | Marvel. Online verfügbar unter <https://marvelapp.com/>, zuletzt geprüft am 06.11.2015.

Geofencing | Wikipedia (2016). Online verfügbar unter <https://de.wikipedia.org/w/index.php?oldid=140335627>, zuletzt aktualisiert am 12.01.2016, zuletzt geprüft am 13.02.2016.

Getting the Home Layout | Apple Developer (2015). Online verfügbar unter https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/HomeKitDeveloperGuide/FindingandAddingAccessories/FindingandAddingAccessories.html#/apple_ref/doc/uid/TP40015050-CH3-SW1, zuletzt aktualisiert am 08.04.2015, zuletzt geprüft am 07.02.2016.

HomeKit Basics: Auslöser verständlich erklärt | HomeKit.tips (2016). Online verfügbar unter <http://homekit.tips/homekit-basics-ausloeser-verstaendlich-erklaert/>, zuletzt aktualisiert am 13.02.2016, zuletzt geprüft am 14.02.2016.

HomeKit Produktübersicht | HomeKit.tips (2016). Online verfügbar unter <http://homekit.tips/offizielle-homekit-produkte/uebersicht-homekit-produkte/>, zuletzt aktualisiert am 18.02.2016, zuletzt geprüft am 18.02.2016.

HomeKit-kompatible Heimelektronik nutzen | Apple Support (2015). Online verfügbar unter <https://support.apple.com/de-de/HT204893>, zuletzt aktualisiert am 11.12.2015, zuletzt geprüft am 13.02.2016.

Ikaros - Free Typeface | Behance (2015). Online verfügbar unter <https://www.behance.net/gallery/25963023/Ikaros-Free-Font>, zuletzt geprüft am 15.02.2016.

Introducing HomeKit - WWDC 2014 - Videos | Apple Developer (2014). Online verfügbar unter <https://developer.apple.com/videos/play/wwdc2014-213/>, zuletzt geprüft am 07.02.2016.

Lars Hinrichs (2014): Mein intelligentes Haus: Das Smart Home Betriebssystem. Hg. v. The Huffington Post. Online verfügbar unter http://www.huffingtonpost.de/lars-hinrichs/intelligentes-haus-smart-home-betriebssystem_b_6131420.html, zuletzt aktualisiert am 11.10.2014, zuletzt geprüft am 29.10.2015.

Lost Phone | august (2016). Online verfügbar unter http://support.august.com/customer/en/portal/articles/2169319-lost-phone?b_id=10917, zuletzt aktualisiert am 18.02.2016, zuletzt geprüft am 22.02.2016.

MFi Program | Apple Developer (2016). Online verfügbar unter <https://developer.apple.com/programs/mfi/>, zuletzt geprüft am 18.02.2016.

Praxistest Philips hue Lampensystem | MacTechNews (2015). Online verfügbar unter <http://www.mactechnews.de/news/article/Praxistest-Philips-hue-Lampensystem-Volle-Lichtkontrolle-mit-iOS-und-Apple-Watch-161755.html?page=4>, zuletzt aktualisiert am 11.07.2015, zuletzt geprüft am 29.10.2015.

Real-world context for your apps | Estimote (2016). Online verfügbar unter <http://estimote.com/>, zuletzt aktualisiert am 13.01.2016, zuletzt geprüft am 07.02.2016.

Robert di Marcoberardino (2015): Z-Wave, Zigbee, Homekit — der Dschungel im Wohnzimmer. Hg. v. Elektronik Praxis. Online verfügbar unter <http://mobil.elektronikpraxis.de/artikel/507720/>, zuletzt aktualisiert am 13.10.2015, zuletzt geprüft am 29.10.2015.

Siri (Software) | Wikipedia (2016). Online verfügbar unter <https://de.wikipedia.org/w/index.php?oldid=151321926>, zuletzt aktualisiert am 13.02.2016, zuletzt geprüft am 14.02.2016.

Smart Home für die Mehrheit der deutschen Befragten noch zu teuer | GfK (2015). Online verfügbar unter <http://www.gfk.com/de/insights/press-release/smart->

home-fuer-die-mehrheit-der-deutschen-befragten-noch-zu-teuer/, zuletzt aktualisiert am 09.12.2015, zuletzt geprüft am 17.02.2016.

The super fast color schemes generator | Colors von Fabrizio Bianchi (2015).
Online verfügbar unter <https://colors.co/app/0e1116-f4e285-e2c044-587b7f-374a67>, zuletzt geprüft am 15.02.2016.

b. Abbildungsverzeichnis

Abbildung 1-1: Umfrage über Interesse an neuen Technologien	7
Abbildung 2-1: Entwicklung von HomeKit in zeitlicher Übersicht.....	11
Abbildung 2-2: HomeKit Struktur	14
Abbildung 2-3: Ortsabhängige Erinnerung mit Geofencing	17
Abbildung 2-4: Umfrage über Bedenken bei Home Automation.....	18
Abbildung 3-1: Brainstorming Namensfindung	21
Abbildung 3-2: HomeKit-Zertifikat.....	22
Abbildung 3-3: Verfügbare Geräte in Deutschland	23
Abbildung 3-4: Ausgewählte Farbpalette	24
Abbildung 3-5: Ikaros Font.....	25
Abbildung 3-6: Verwendung der Schriftart Ikaros.....	25
Abbildung 3-7: App-Icon	26
Abbildung 3-8: Icons zur Darstellung der Accessory-Typen	26
Abbildung 4-1: Skizze vom Aufbau der App.....	28
Abbildung 4-2: QR-Code zum Laden des Klick-Dummys	29
Abbildung 4-3: Hilfestellung zum Installieren des Klick-Dummys	29
Abbildung 4-4: Hilfe-Themen im Klick-Dummy	30
Abbildung 4-5: Startseite des Klick-Dummys.....	30
Abbildung 4-6: Layout-Skizzen auf Karteikarten	31
Abbildung 4-7: Arbeiten mit Adobe Photoshop	32
Abbildung 4-8: Arbeiten mit Marvel	33
Abbildung 4-9: Aufbau der Navigation im Klick-Dummy.....	34
Abbildung 4-10: Skizzen zum Thema Complications	35
Abbildung 5-1: Startseite im Endzustand der Entwicklung	36
Abbildung 5-2: Trennung der Datenstrukturen.....	37
Abbildung 5-3: Klassen-Übersicht.....	38
Abbildung 5-4: Aktivieren von HomeKit	39
Abbildung 5-5: Home und Room – Datenstrukturen.....	41
Abbildung 5-6: Accessory – Datenstruktur.....	45
Abbildung 5-7: Sequenzdiagramm – Reload Accessories.....	47
Abbildung 5-8: HomeKit Accessory Simulator	52
Abbildung 5-9: Gestenerklärung Pull-To-Refresh	57

Abbildung 5-10: Seite mit noch nicht verbundenen Accessories	60
Abbildung 5-11: Seite mit ergänzenden Informationen.....	60
Abbildung 5-12: Seite mit Info-Galerie	60

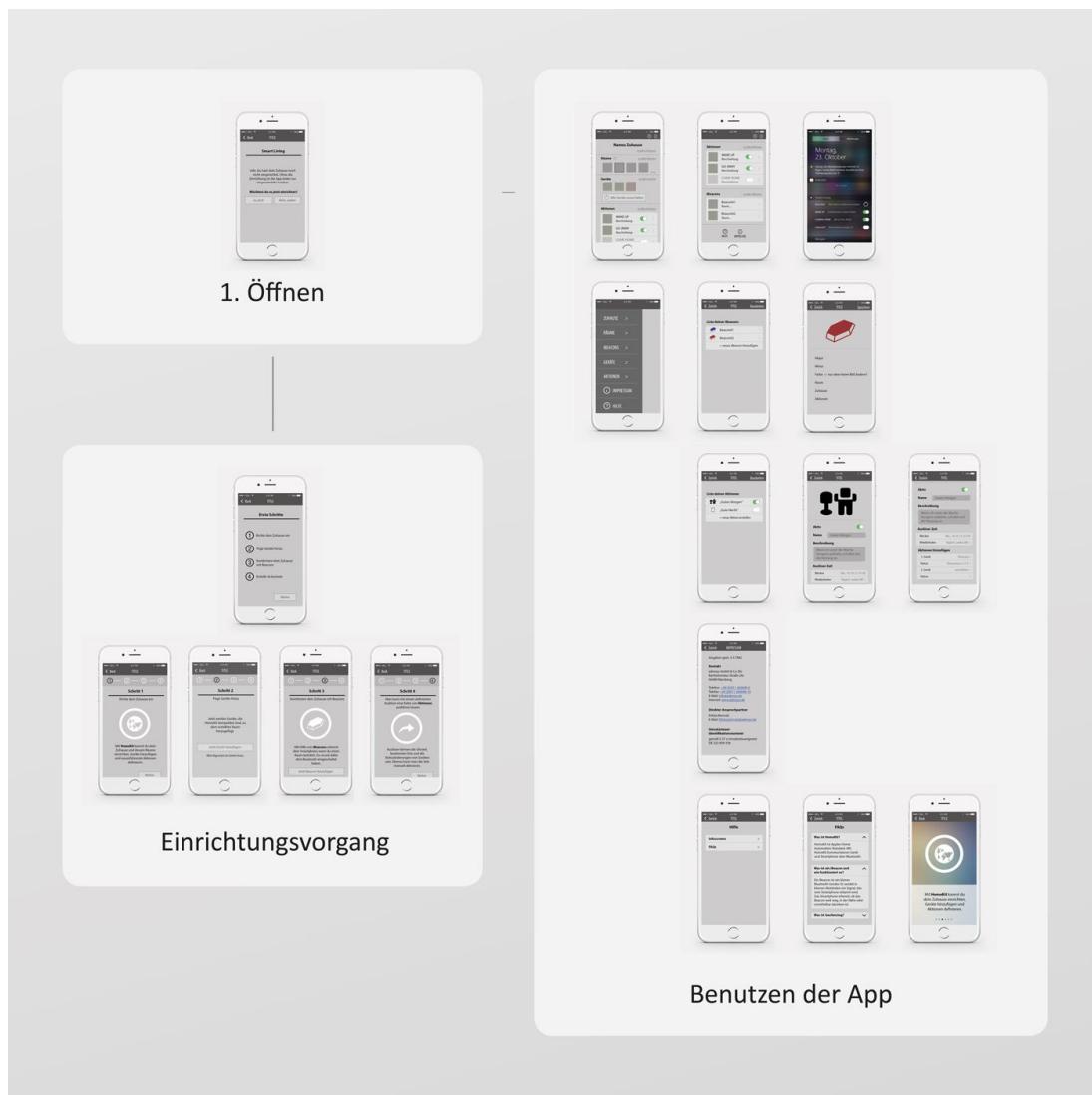
c. Anhang

I.	Schnellzugriff über Mitteilungszentrale.....	72
II.	Klick-Dummy Workflow	73
III.	Xcode-Storyboard	74
IV.	Sequenzdiagramm: Laden der Accessories.....	75
V.	Sequenzdiagramm: Detect Beacon.....	76

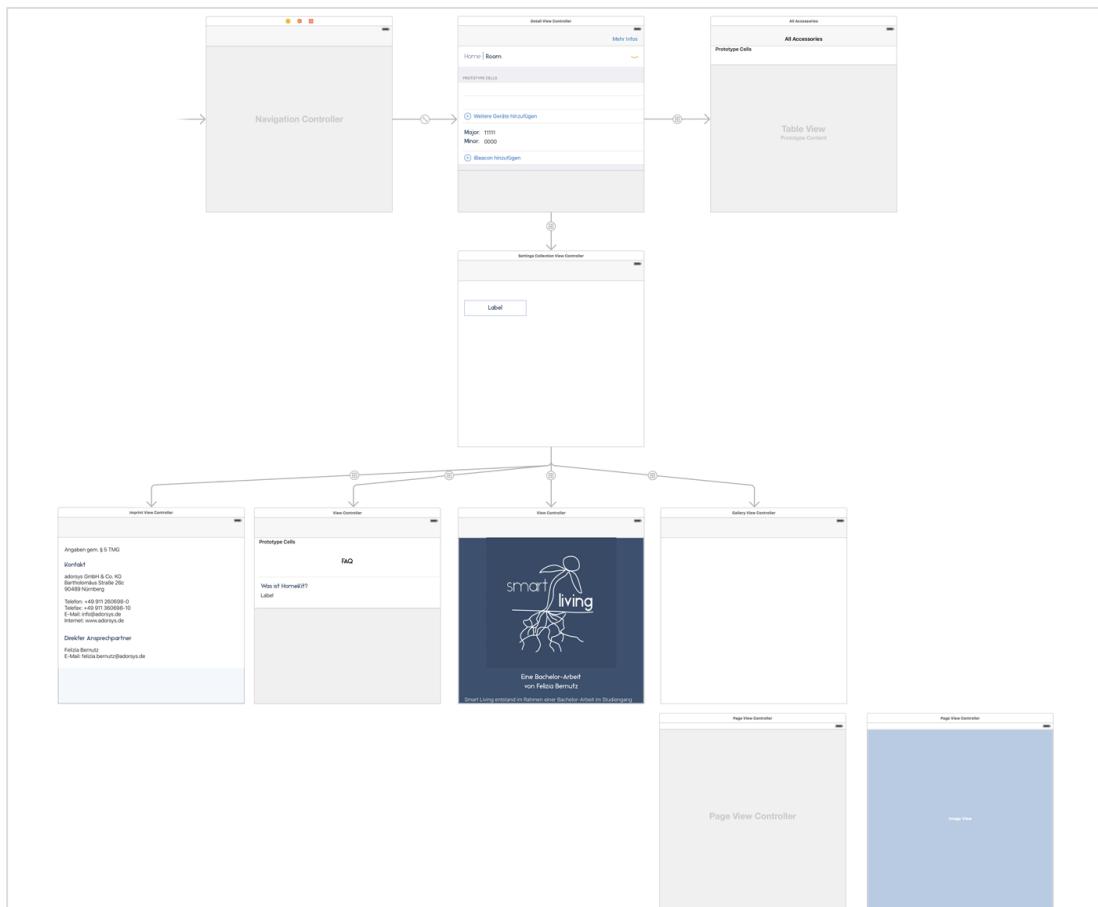
I. Schnellzugriff über Mitteilungszentrale



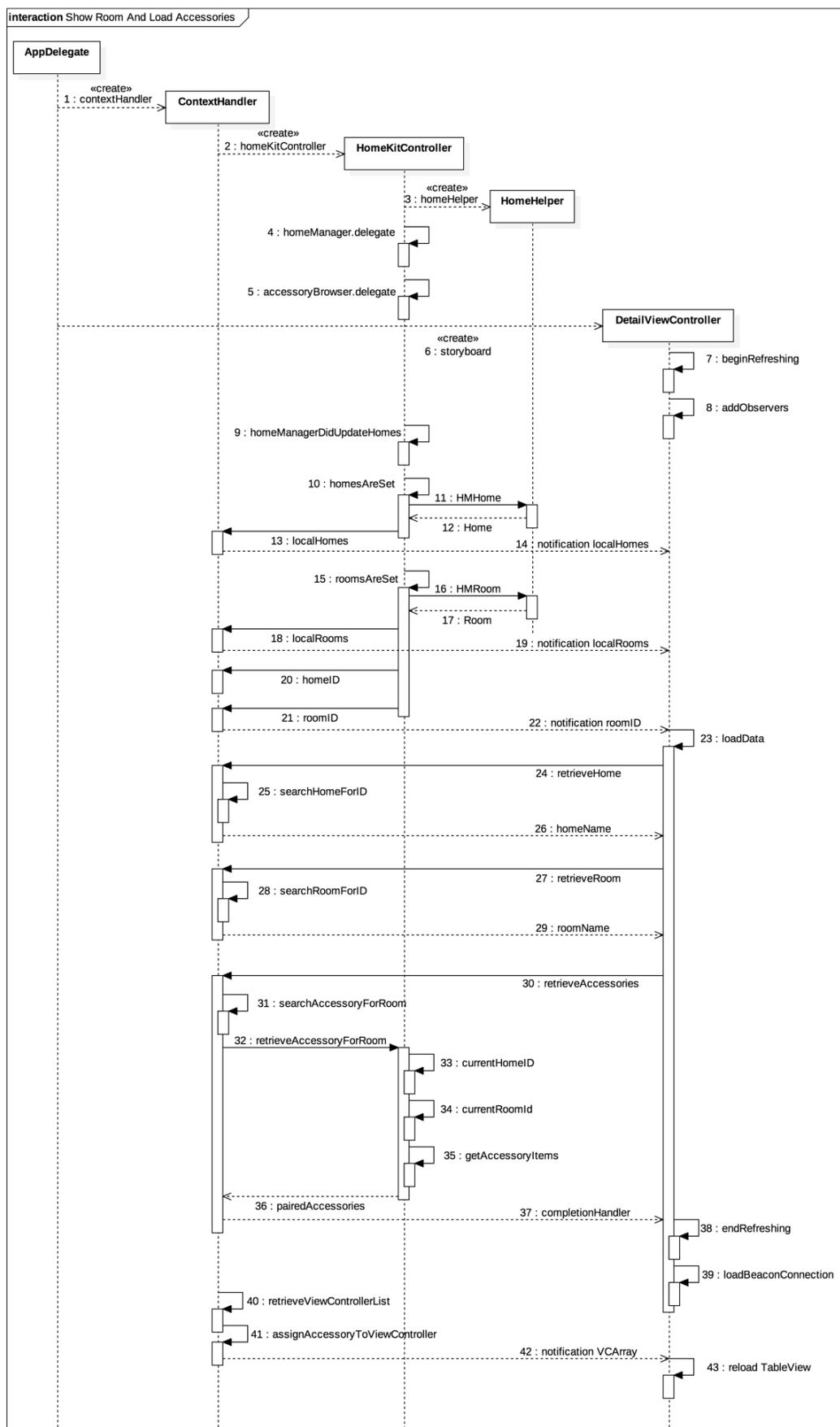
II. Klick-Dummy Workflow



III. Xcode-Storyboard



IV. Sequenzdiagramm: Laden der Accessories



V. Sequenzdiagramm: Detect Beacon

