

Smart-Home-Lösungen mittels Sprachsteuerung und Raspberry Pi

STUDIENARBEIT

für die Prüfung zum
Bachelor of Science
des Studienganges Angewandte Informatik
an der
Dualen Hochschule Baden-Württemberg Karlsruhe
Abgabedatum 14.05.2017

Bearbeitungszeitraum	Theoriesemester 5 und 6
Autoren	Maximilian Hirte, Robin Warth
Matrikelnummern	8994521, 6028632
Kurs	TINF15B4
Ausbildungsfirma	Siemens AG Östl. Rheinbrückenstr. 50 76187 Karlsruhe
Betreuer der DHBW	Prof. Dr. Jürgen Röthig

Erklärung

Wir versichern hiermit, dass wir unsere Studienarbeit mit dem Thema: „Smart-Home-Lösungen mittels Sprachsteuerung und Raspberry Pi“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Wir versichern zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum

Maximilian Hirte

Ort Datum

Robin Warth

Kurzfassung

Die Beliebtheit von Smart Home, der intelligenten Verknüpfung von Hauskomponenten, nimmt heute immer weiter zu. Im Zuge dessen versuchen viele Technologie Unternehmen, wie Google und Amazon, sich in diesem Bereich zu etablieren. Sie bieten ihre eigenen Lösungen und Gerätefamilien, welche auch durch Dritthersteller erweitert werden können, der breiten Masse an Nutzern und auch Geräteproduzenten an.

Trotz der mittlerweile weiten Verbreitung und Vielfalt an Smart Home-Lösungen ist die personalisierte Heiamtatisierung noch immer eine sehr kostspielige Angelegenheit. Häufig müssen bereits erworbene Komponenten zu Hause vollständig durch neue teure „smarte“ Komponenten ausgetauscht werden um sie in das Smart Home zu integrieren.

Ziel dieser Studienarbeit ist eine Kostenersparnis durch Anbindung weiterer günstiger oder bereits bestehender Komponenten, an ein bestehendes Smart Home inklusive Sprachsteuerung, zu erreichen. Hierzu werden preiswerte Funksteckdosen und eine Kamera, mittels des Einplatinencomputers Raspberry Pi als Brücke, angebunden.

Hierfür wurden, mit Node.js und Amazon Web Services Lambda, modernen Methoden der Web- und Cloudentwicklung angewendet, sowie Vergleiche von digitalen Assistenten und kabellose Geräteansteuerungen im Hochfrequenzbereich durchgeführt. Das Ergebnis dieser Studienarbeit ist ein eigenständiges Smart Home System mit zusätzlich erfolgreicher Anbindung an die Alexa Produktfamilie von Amazon. Dank Alexa Anbindung können alle, über den Raspberry Pi angebunden, Geräte bequem mittels Sprachsteuerung angesteuert werden.

Abstract

The popularity of Smart Home, the intelligent combination of house components, is growing all the time. In the course of this, many technology companies such as Google and Amazon are trying to establish themselves in this area. They offer their own solutions and device families, which can also be extended by third party manufacturers, to the broad mass of users and also device manufacturers.

Despite the wide spread and variety of smart home solutions, personalized home automation is still a very costly business. Often already purchased components at home have to be completely replaced by new expensive „smart“ components in order to integrate them into the smart home.

The aim of this student research project is to achieve cost savings by connecting further inexpensive or already existing components to an existing smart home including voice control. For this purpose, inexpensive wireless sockets and a camera are connected using the single-board computer Raspberry Pi as a bridge.

For this purpose, modern methods of web and cloud development were applied with Node.js and Amazon Web Services Lambda, as well as comparisons of digital assistants and wireless device controls in the high-frequency range. The result of this student research project is an independent Smart Home System with an additional successful connection to the Alexa product family from Amazon. Because of the Alexa connection, all devices connected via the Raspberry Pi can be conveniently controlled via voice control.

Inhaltsverzeichnis

Abbildungsverzeichnis	VII
Quellcodeverzeichnis	VIII
Tabellenverzeichnis	IX
Abkürzungsverzeichnis	X
1 Einleitung	1
2 Aufgabenstellung	2
3 Grundlagen	3
3.1 Smart Home	3
3.2 Sprachverarbeitung	4
3.3 Analyse und Vergleich von digitalen Assistenten	5
3.3.1 Amazon Alexa	6
3.3.2 Apple Siri	8
3.3.3 Google Assistant	10
3.3.4 Microsoft Cortana	11
3.3.5 Samsung Bixby	12
3.3.6 Vergleichsfazit	13
3.4 Raspberry Pi	15
3.5 Funkstandards	17
3.6 OAuth2	18
3.6.1 Rollen	18
3.6.2 Protokollablauf	19
3.6.3 Berechtigungsvergabe	20
3.6.4 Authorization Code Grant	21
3.6.5 Bearer Token	23
3.7 Development-Tools	23
3.7.1 Amazon Developer Services	24
3.7.2 Amazon Web Services - Lambda	24
3.7.3 Cloud9	25
3.7.4 Express.js	25

3.7.5 FFmpeg	26
3.7.6 Ngrok	26
3.7.7 Node.js	26
3.8 Gebrauchstauglichkeit	27
4 Konzept	29
4.1 Anforderungsanalyse	29
4.1.1 Grund der Umsetzung	29
4.1.2 Funktionale Anforderungen	30
4.1.3 Nicht-Funktionale Anforderungen	31
4.2 Herangehensweise	32
4.3 Architektur	33
4.4 Design	34
4.4.1 Skill-Typen	36
4.4.2 Ansteuerung der Funksteckdosen	38
4.4.3 Ansteuerung der Kamera	40
5 Implementierung	41
5.1 Einrichtung	41
5.1.1 Raspberry Pi	41
5.1.2 Cloud 9	42
5.1.3 Git	42
5.1.4 Node.js	42
5.1.5 FFmpeg	43
5.1.6 Amazon Developer Konto	43
5.1.7 Alexa	44
5.1.8 Amazon Web Services (AWS) Lambda	49
5.1.9 ngrok	50
5.2 Entwicklung	51
5.2.1 AWS Lambda	51
5.2.2 Raspberry Pi	66
6 Zusammenfassung	74
6.1 Ergebnis	74
6.2 Ausblick	74
6.3 Fazit	75
Literatur	XI
Anhang	XVIII

Abbildungsverzeichnis

Abb. 3.1: Raspberry Pi 3 Model B	15
Abb. 3.2: Aufbau des für das Projekt installierten Systems mit Raspberry Pi . .	17
Abb. 3.3: Abstrakter Protokollablauf [RF6749]	19
Abb. 3.4: Authorization Code Grant [RF6749]	22
Abb. 4.1: Raspberry Pi-Alexa-Kommunikation Vereinfacht	34
Abb. 4.2: Raspberry Pi-Alexa-Kommunikation	34
Abb. 4.3: Verwendetes Funkset mit Hausschlüssel	39
Abb. 5.1: Setup Skill-Information	44
Abb. 5.2: Setup Interaction Modell	45
Abb. 5.3: Setup Configuration Endpoints	46
Abb. 5.4: Setup Configuration Account Linking	47
Abb. 5.5: Raspberry Pi-Alexa-Kommunikation Schritt 1	48
Abb. 5.6: Setup Test	49
Abb. 5.7: Raspberry Pi-Alexa-Kommunikation Datenfluss	51
Abb. 5.8: Flussdiagramm Alexa nach Raspberry Pi	65
Abb. 5.9: Webinterface	69

Quellcodeverzeichnis

5.1	GitHub Repository	42
5.2	Node Module Laden	42
5.3	Node.js Installieren	42
5.4	Node.js Installieren	43
5.5	FFmpeg Installieren und Starten	43
5.6	FFmpeg Rekonfigurieren und Starten	43
5.7	FFmpeg Konvertierung	43
5.8	Ngrok Einrichtung	50
5.9	Ngrok Tunnel Aufbau	50
5.10	Export Handler	52
5.11	Request: Report-State	54
5.12	Response: Report-State	56
5.13	Request: Power-Controller	57
5.14	Response: Power-Controller	58
5.15	Request: Discovery	59
5.16	Request: Authorization	61
5.17	Response: Authorization	62
5.18	Benutzerkontrolle	63
5.19	Benutzerinformationen	64
5.20	Node app.js: Webinterface	68
5.21	Node app.js: Report-State	68
5.22	Node.js-Funktion: Erstellen und Senden des Funksignals	70
5.23	Node.js: Starte Aufnahme mit Hilfe von raspivid	72

Tabellenverzeichnis

3.1 Formaler Vergleich der aktuell meist verbreitetsten Sprachassistenten . . . 14

Abkürzungsverzeichnis

API	Application Programming Interface
ARN	Amazon Resource Name
AVS	Alexa Voice Service
AWS	Amazon Web Services
CPU	Central Processing Unit
CSS	Cascading Style Sheets
Dip	Dual In-line Package
GPIO	General Purpose Input/Output
HDMI	High-Definition Multimedia Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LED	lichtemittierende Diode
LWA	Login With Amazon
NPM	Node Package Manager
SSH	Secure Shell
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
USB	Universal Serial Bus
WLAN	Wireless Local Area Network

1. Einleitung

Wir leben in einer Welt, in der jegliche Teilprozesse des täglichen Lebens bereits automatisiert sind bzw. automatisiert werden sollen, so auch das eigene Zuhause. Seit Anfang des Jahrtausends stellen sich Firmen und Institute die Aufgabe, das intelligente Zuhause zu entwickeln [VOE16] und schon mehrere Jahre ist der Begriff Smart Home in aller Munde. Es können jegliche Systeme innerhalb des Eigenheims zentral gesteuert und miteinander verknüpft werden. Über bspw. eine App kann dann der Benutzer seine persönlichen Einstellungen vornehmen und das Lebensgefühl somit auf eine neue Ebene setzen.

In letzter Zeit geht der Trend dahin, diese Hausautomatisierung per Sprachbefehle zu koordinieren. Zu diesem Zweck gibt es verschiedene unabhängige digitale Sprachassistenten, die dem Bewohner das Leben vereinfachen. Besonders Amazon-Alexa und der Google-Assistent ringen im Kampf um die Marktvormachtstellung und überzeugen beide im Bereich der Sprachverständnis und -verarbeitung, aber vor allem auch mit den Fähigkeiten, die sie bieten.

Jedoch bringt eine komplette Wohneinrichtung mit Systemen, die schon standardmäßig die Verbindung zu den digitalen Assistenten gewährleisten, immense Kosten mit sich. Diese ließen sich reduzieren, wenn sich z. B. nicht nur die von Amazon und Google vorgesehenen und zu hohen Preisen angepriesenen Geräte koppeln lassen würden, sondern jegliche Systeme, Geräte und Anlagen, die für den Smart Home-Gebrauch geeignet sind.

Die Motivation dieser Studienarbeit ist somit, eine solche Kostensparnis zu erzielen. Amazon bietet mit seinen Developer Services die Möglichkeit, Alexa mit beliebigen internetfähigen Geräten zu verbinden oder den Alexa Voice Service (AVS) direkt auf nicht-Amazon Geräten einzurichten. In dieser Arbeit wurde das Ziel gesetzt, diese durch Alexa gegebenen Möglichkeiten zu nutzen und somit einen Raspberry Pi mit Alexa zu verbinden. Mithilfe gängiger Funkstandards soll der Raspberry Pi weitere Geräte ansteuern, die ansonsten über keinerlei Funktionen verfügen mit digitalen Sprachassistenten zu interagieren. Neben der Verringerung der Kosten bietet eine solche Lösung eine weitere Stufe der Individualisierbarkeit und vor allem Geräte- bzw. Herstellerunabhängigkeit für die persönliche Smart Home-Zusammenstellung in den eigenen vier Wänden.

2. Aufgabenstellung

Die Aufgabe dieser Studienarbeit besteht darin, eine eigene Smart Home Lösung mittels des von Amazon bereitgestellten Sprachassistenten und eines Raspberry Pi zu entwickeln. Das Hauptaugenmerk liegt dabei darauf, das Repertoire des schon bestehenden Alexa-Systems um weitere Haushaltsgeräte zu erweitern. Diese haben entweder keine Möglichkeit einer Anbindung an Alexa durch einen passenden Skill oder besitzen teilweise gar nicht die benötigte technische Infrastruktur, wie z. B. WLAN oder Rechenkapazität. Diese Anbindung erfolgt somit über den Raspberry Pi, der für diesen Zweck konfiguriert und angeschlossen wird.

Die Mindestanforderung an das Projekt ist die Inbetriebnahme einer Funksteckdose, die über den Raspberry Pi mit einem Alexa Skill per Sprache ein- und ausgeschaltet werden kann. Im Anschluss soll dann auch eine Kamera an das System angeschlossen werden und per Sprachbefehle gesteuert werden.

Nach Möglichkeit soll weiterhin ein Kosten- und Aufwandsvergleich zwischen der eigens konfigurierten und implementierten Variante sowie der teuren Kamera aus dem Amazonhaus aufgestellt werden.

Im Zuge der Ausarbeitung sollen dann neben Alexa auch andere derzeit aktuelle Sprachassistenten unter die Lupe genommen und miteinander verglichen werden. Dabei soll auch abgewägt werden, welche von den Konkurrenzprodukten von Amazon potentiell für die Verwendung in dieser Studienarbeit in Betracht gezogen werden könnten.

3. Grundlagen

In diesem Kapitel werden der Arbeit zu Grunde liegende Begriffe erklärt, Zusammenhänge zwischen diesen aufgezeigt sowie verschiedene Systeme vorgestellt und miteinander verglichen.

3.1. Smart Home

Unter dem Begriff Smart Home wird eine Menge von technischen Systemen, Geräten und Anlagen im Haushalt verstanden, die miteinander interagieren und sowohl zentral als auch dezentral bequem gesteuert werden können. Die damit verbundenen Ziele sind:

- Automatisierung von Alltagsvorgängen,
- Verbesserung von Wohn- und Lebensqualität,
- Erhöhung der häuslichen Sicherheit und
- effiziente Energienutzung. [DEV17]

Die gesamte Haushaltstechnik wird somit vernetzt und kann individuell angepasst und automatisiert werden. So können auch Geräteeinstellungen vorgenommen werden und über eine Schnittstelle, wie Computer oder Smartphone, verwaltet werden. Es ist somit z. B. möglich seine Heizung per Zeitsteuerung zu kontrollieren und auf eigene Bedürfnisse ideal zu zuschneiden. Per App auf dem Smartphone kann die Heizung, das Licht, der Fernseher oder jedes andere angeschlossene Gerät an- und abgeschalten werden. Hierfür muss der Benutzer nicht einmal zu Hause sein. Prinzipiell lassen sich alle Geräte ansteuern, die über WLAN, Bluetooth, ZigBee oder andere Funkstandards verfügen (Siehe 3.5 Funkstandards). [SCH17]

Die nächste Stufe der Entwicklung ist dann die endgerätlose Steuerung. D. h., die Geräte auch zu steuern, ohne jedes mal den PC oder die App zu benutzen. Der Fokus wird immer mehr auf Gesten- und vor allem auf Sprachsteuerung gelegt, weswegen letzteres auch das Thema dieser Arbeit ist.

3.2. Sprachverarbeitung

Seit der zügigen Entwicklung der Computertechnik in der zweiten Hälfte des 19. Jahrhunderts gibt es das Problem und den Wunsch, natürliche Sprache mit Hilfe einer Maschine zu erkennen und zu verarbeiten. In der heutigen Zeit ist die Interaktion mit Maschinen so gut wie unumgänglich. Der herkömmliche Weg über Tastatur, Maus und Touchscreen ist für den Menschen immer noch unzureichend, denn unser Hauptkommunikationsmittel ist die Sprache. [RAO17, S. 1]

Die Sprachverarbeitung unterteilt sich unter anderem in Sprachsynthese, also eine symbolischen Notation in ein Sprachsignal umzusetzen, und in Spracherkennung, die in dieser Arbeit hauptsächlich beleuchtet wird. Der Schwerpunkt hierbei liegt darin ein Sprachsignal oder auch Laut in eine textliche Form umzusetzen, sodass der Computer diese erkennt. Dies ist jedoch nach wie vor ein sehr großes Problem und wird auch in den nächsten beiden Jahrzehnten nicht so gelöst werden können, dass der Computer die menschliche Sprachwahrnehmungsfähigkeit komplett erreicht. Der Grund dafür ist die Vielfältigkeit und Komplexität der natürlichen Sprache. Auf der Ebene des Wortschatzes ist der Computer schon sehr gut aufgestellt und kann die meisten Wörter erkennen und unterscheiden. Doch auf Ebene der Syntaktik und Semantik hat die Maschine bisher keine Chance. Der PC ist somit noch nicht in der Lage die Fähigkeit des Menschen, fortwährend zu untersuchen und zu entscheiden, ob das Gehörte Sinn ergibt. [PFI17, S. 21]

Aus diesem Grund werden für Spracherkennungsassistenten spezielle Anwendungsszenarien konzipiert. Ein solcher digitale Assistent kann somit zwar keine vollständigen Konversationen führen, aber er verfügt z. B. über die Fähigkeit einen Wetterbericht aus dem Internet zu suchen, wenn er nach dem morgigen Wetter gefragt wird. Durch solche Spezialisierung auf Anwendungsfälle werden bisher sehr gute Resultate erreicht, da es weniger Erkennungsfehler gibt. Weiterhin ist diese Lösung kompakter, da sie sowohl weniger Speicher als auch Rechenleistung benötigt. [PFI17, S. 28]

Im Jahr 2017 hat die Mozilla Foundation das Projekt Common Voice durchgeführt, welches aus 400.000 validierten, englischen Sprachaufnahmen von 200.000 Personen besteht. Auf diese Aufnahmen wendeten sie verschiedene Spracherkennungssoftware an, um somit die Fehlerquote bei der automatisierten Übersetzung in Text zu minimieren. Ziel war es, eine Quote unter der Zehn-Prozent-Marke zu erreichen. Das erscheint für uns Menschen relativ viel, aber laut Forschungsergebnissen, auf die sich Mozilla beruft, liegt die Fehlerquote des menschlichen Sprachverständnis bei rund 6%. [MOZ17] Aus diesen Werten lässt sich also schließen, dass vorhandene Software sich im Bereich der

Spracherkennung schon den menschlichen Fähigkeiten annähert.

Die maschinelle Erkennung und Verarbeitung erfolgt über Sprachsignale. Die genauen Algorithmen sowie der gesamte interne technische Ablauf der digitalen Assistenten sind nicht Teil dieser Arbeit und werden nicht weiter erläutert.

3.3. Analyse und Vergleich von digitalen Assistenten

Ein digitaler Sprachassistent (oft auch intelligenter persönlicher Assistent oder auch nur digitaler Assistent) ist ein System zur Sprachverarbeitung. Das System kann gesprochenen Anweisungen Folge leisten und zudem in menschlicher, meist weibliche, Stimme antworten. Die häufigsten Anwendungsfälle stellen dabei einfache Sprachsachen im Internet oder Abarbeiten von einfachen Aufgaben, wie Schreiben und Versenden von Nachrichten oder Erstellen von Kalendereinträgen bzw. Erinnerungen dar.

Im Folgenden werden die zur Zeit fünf bedeutendsten Assistenten genauer untersucht und anhand folgender Kriterien miteinander verglichen:

- Sprachsynthese - Hier wird die Stimme des Assistenten bewertet. Es wird verglichen, welche Stimmen am menschlichsten und sympathischsten klingen. Die Offenheit für Humor zählt hier auch mit dazu.
- Spracherkennung - In diesem Punkt wird die Fähigkeit, die Spracheingabe des Benutzers richtig zu verstehen, verglichen. Muss der Befehl oftmals wiederholt werden schneidet der Assistent hierbei schlechter ab. Die Stimmen- bzw. Benutzererkennung spielt hierbei auch eine Rolle.
- Verbreitung - Vergleichsschwerpunkt ist hierbei, die Anzahl an Plattformen auf die die jeweilige Software zur Verfügung steht.
- Allgemeinwissen - Wichtig für diesen Punkt ist die Fähigkeit, auf Wissensfragen bzw. Suchanfragen die richtige Antwort zu liefern, aber auch diese Antwort passend zu präsentieren (visuell, textlich oder mündlich). Die Bereitstellung von Lokalnachrichten, wie das Kinoprogramm, sowie Navigationsinformationen zählen hier auch mit herein.
- Übersetzungsfähigkeiten - Hier wird verglichen, inwiefern die Assistenten in der Lage sind in andere Sprachen zu übersetzen und dies auch auszugeben. Ein weiterer Punkt ist einzelne Fremdwörter in Sätzen zu verstehen und zu verarbeiten. Ein Beispiel dafür ist englischsprachige Lieder abzuspielen. Dieser Punkt ist für einen allgemeinen Vergleich sehr interessant, aber speziell für diese Studienarbeit eher unwichtig.

- Funktionsumfang - Hier spielt neben den mitgelieferten Funktionen auch die Einbettung in das System eine große Rolle. D. h., inwiefern auf Drittanbieter-Software, wie Musik-Streaming oder Messenger-Dienste, zugegriffen werden kann und wie gut der jeweilige Assistent an die vorliegende Hardware (z. B. smarter Lautsprecher oder auch Mobiltelefon) angepasst ist und mit den dort vorhandenen Anwendungen kooperieren kann.
- Erweiterbarkeit - In diesem Punkt wird bewertet, ob und wie gut Benutzer den Funktionsumfang durch eigene, personalisierte Befehle (Skills, Actions etc.) erweitern können. Außerdem zählt die Anzahl der bereits bestehenden Erweiterungen mit hinein.
- Smart Home Unterstützung - Dies ist ein sehr wichtiger Punkt für diese Studienarbeit. Hier wird verglichen, welche Assistenten Smart Home überhaupt unterstützen und bewertet, wie gut dies funktioniert. Der Punkt Erweiterbarkeit und die damit verbundene Personalisierung des eigenen Systems spielt hierbei auch hinein.
- Gender-Correctness - Dieser Punkt ist für den formalen Vergleich der Sprachassistenten eher unwichtig, aber durch den besonderen Augenmerk auf politische Korrektheit in der heutigen Zeit wurde dieser Punkt mit in den Vergleich dazu genommen. Hier wird angegeben, welche Software auch anbietet, eine Stimme des männlichen Geschlechts auszuwählen.

Anzumerken ist, dass sich der Vergleich größtenteils auf die deutsche Version der Assistenten bezieht (ausgenommen: Bixby), da es möglicherweise Unterschiede zu anderen Sprachen gibt. Zudem ist klar, dass sich die Softwareprodukte während der Ausarbeitung weiterentwickeln. Dieser Vergleich basiert somit auf den Daten, die bis Februar 2018 zur Verfügung standen.

3.3.1. Amazon Alexa

Im Jahr 2015 kam mit den smarten Echo-Lautsprechern auch Alexa auf den Markt und reiht sich somit in der Historie hinter Siri und Google Now ein. Die größte Verbreitung genießt es auf den Echo-Geräten, aber auch auf weiteren Amazon-Geräten, wie Fire-TV oder Fire-Tablets. Der digitale Assistent lässt sich aber auch als App auf dem Smartphone installieren. [BAG17, S. 65]

Die Tester von Heise online, welche die vier bekanntesten Sprachassistenten unter die Lupe genommen haben, beschreiben Alexa mit einer „fast beängstigend guten Spracherkennung und mit einer angenehmen, freundlichen Stimme“. [BLE17, S. 81] Allgemein punktet der Assistent aus dem Hause Amazon in der Art der Kommunikation. Als tägliche Begleiterin im eigenen Heim ist sie sehr sympathisch und kann sogar auf Humor reagieren. Für ein

gewisses Level an Spaß besitzt Alexa Fähigkeiten der Nachahmung von Filmcharakteren, wie Yoda aus Star Wars, oder auch Spaßfakten, wie die Zahl 42 als Antwort auf alles. Auf Basis der Kommunikation hat Alexa somit einen gewissen Vorsprung gegenüber dem Hauptkonkurrenten Google. [BLE17, S. 81]

Auf viele Wissensfragen hat Alexa eine gute und richtige Antwort parat. Jedoch sind Navigationsinformationen, wie der schnellste Weg nach Hamburg, noch sehr ausbaufähig. Sie ist aber auf ihren Echo-Lautsprechern bisher auch größtenteils auf den stationären Betrieb ausgelegt. Somit ist der Assistent im Bereich des Lokalservices wie z. B. dem Kinoprogramm der Stadt besser ausgerüstet, auch wenn manchmal eine zugehörige App von Nöten ist. [BLE17, S. 81] Weiterhin gibt es dazu auch den Echo Show, ein Echo-Gerät mit integriertem Touchscreen, auf welchem die Informationen noch visualisiert werden können. [BAG17, S. 65]

Fehlendes Wissen und Funktionen lassen sich durch Erweiterungen nachrüsten, welche Amazon Skills nennt. Mittlerweile gibt es ca. 3000 deutschsprachige und in den USA bereits über 16.000 Skills (Stand: Oktober 2017 [BLE17]). Diese Skills sollen das Leben, durch z. B. bestimmte Radiosender, Online-Kochbücher oder Fahrpläne, per Sprachzugriff vereinfachen und verbessern. Solche Skills kann auch jeder Benutzer selber implementieren, indem er sich kostenlos bei den Amazon Developer Services registriert und somit zum Entwickler wird. Die Hauptstärke von Alexa liegt also genau in diesen erweiterbaren Skills, wodurch der Assistent immer weiter dazulernt. [BLE17, S. 81]

Das Nachsehen gegenüber der Konkurrenz hat Alexa vor allem beim Verständnis und der Interpretation von Sprachen, welche von der eingestellten abweichen. Versucht der Anwender z. B. bei einer auf Deutsch eingestellten Station einen englischen Titel per Spracheingabe auf dem Musik-Streaming-Dienst zu starten, kann es passieren, dass Alexa nicht das richtige Lied oder den passenden Interpreten abspielt. [BLE17, S. 81]

Weiterhin ist, obwohl der Assistent auf den Online-Shopping-Riesen Amazon optimiert sein soll, der Einkauf mittels Sprache nicht wirklich zufriedenstellend. Oftmals lassen sich Artikel nur in die Merkliste legen und nicht direkt bestellen. Vor allem bei Bekleidung und Schuhen tritt dieses Problem auf. Wenn der Benutzer jedoch einen Artikel bestellen will, den er in der Vergangenheit schon einmal erworben hat, hat Alexa keine Probleme. [BLE17, S. 81]

Ein letztes Manko von Alexa ist das Kontextverständnis. So tut sie sich sehr aufeinanderfolgende Fragen miteinander verknüpfen, was eine der Stärken von Google und dessen Deep Learning ist. Dennoch hat Amazon im Bereich der digitalen Assistenten noch die Nase vorn, da die Echo-Lautsprecher einige Monate vor dem Google Home auf den Markt

kamen. Amazon konnte somit bereits länger Erfahrungen sammeln und von der Anzahl an Skills profitieren, welche von Drittanbietern entwickelt wurden und Alexa kontinuierlich verbessern. [CLA18]

Amazon hat das Marktpotential im Bereich der Sprachassistenten erkannt und möchte somit sein Produkt so weit wie möglich verbreiten und auf so vielen Geräten wie möglich anbieten. Demnächst soll Alexa demzufolge auch direkt in Autos verbaut werden und somit die Hands-Free-Steuerung während der Fahrt erleichtern. Seat und BMW wollen den Assistenten in mehreren Modellen einbauen. Außerdem sprachen die Firmenbosse davon eine Alexa-Cortana-Verbindung zu erstellen und das hieße, dass Alexa dann auch auf Windows-Geräten verwendbar wäre. [BAG17, S. 66]

Amazon hat somit ein Produkt geschaffen, dass dem Benutzer das Leben vereinfachen kann und Zukunftspotential besitzt. Auch im Bereich von Smart Home bietet Alexa eine gute Unterstützung. Es gibt schon verschiedene Geräte, die automatisch mit dem Assistenten gekoppelt werden können. Durch die benutzerdefinierten Skills können auch eigene Projekte realisiert werden, wie bspw. eine Verbindung mit einem Raspberry Pi, welcher als Mittelsmann dient und weitere nicht-Alexa-fähige Geräte ansteuern kann. Aus diesen Gründen ist Alexa auch ein optimaler Kandidat zur Verwendung in dieser Studienarbeit und wurde bereits zu Beginn favorisiert.

3.3.2. Apple Siri

Siri wurde im Jahre 2011 veröffentlicht und ist somit auch die älteste Assistentin. Sie ist nur auf den Apple-eigenen Plattformen iOS, macOS, watchOS sowie tvOS verfügbar. Seit Februar 2018 gibt es auch analog zu den Amazon und Google Lautsprechern einen Apple HomePod, auf dem mit Siri kommuniziert werden kann. [BAG17, S. 68]

Die Vorreiterin punktet vor allem durch die sehr gute Einbettung in das System an sich und der damit verbundenen Interaktion mit den von Apple mitgelieferten Applikationen auf dem Smartphone. Ohne Probleme lassen sich durch einfache Fragen bzw. Aufforderungen der Taschenrechner öffnen und Therme berechnen, eine Wettervorhersage einholen, neue Termine in den Kalender eintragen oder die Verabredungen für den heutigen Tag ausgeben lassen. Weiterhin können E-Mails oder Notizen diktiert, abgespeichert oder versendet werden. Mittlerweile gibt es auch eine ausgereifte Unterstützung für häufig genutzte nicht-Apple Anwendung, z. B. Messenger-Dienste wie WhatsApp. Wenn der Benutzer aber Dienste von externen Anwendungen anfordert, versucht Siri so gut wie immer auf Apple-eigene Apps zuzugreifen. Die angeforderten Ergebnisse sind daher oftmals unzureichend. Der Assistent ist ursprünglich nur für die Ersetzung der Touchbedienung auf den eigenen Geräten entwickelt worden, was auch sehr gut gelungen ist. Unter Apple-Fans erfreut sich Siri daher sehr großer Beliebtheit. [BLE17, S. 85]

Im Bereich der Sprachausgabe überzeugt Siri durch eine angenehme und natürliche Stimme, welche über die Jahre hinweg immer weiter verbessert werden konnte. Die Tester von Heise online beschrieben Siri als „vertrauenerweckend sogar fast freundlich“, [BLE17, S. 85] was auch mit der Personalisierung zu tun hat. Denn der Benutzer kann sich von dem Assistenten mit seinem Namen ansprechen lassen. Zudem stellt sich Siri auf die Stimme des Anwenders ein, wodurch das Benutzererlebnis sowie die Sicherheit zusätzlich verbessert wird. Für Humor ist sie aber eher nicht offen, weswegen sie in normaler Kommunikation milder unterhaltsam abschneidet als ihrer Konkurrenten. Außerdem kann in den Einstellungen, wenn bevorzugt, auch genderkorrekt eine Männerstimme favorisiert werden, was bei weitem nicht alle digitale Assistenten unterstützen. [BLE17, S. 85]

Analog zu den Alexa-Skills und den Actions von Google bietet Apple mit dem SiriKit die Chance, selbst zum Entwickler zu werden. Die Möglichkeiten sind aber im Gegensatz zur Konkurrenz eher beschränkt, da im Prinzip nur Erweiterungen für bestehende Anwendungen implementiert werden können, um diese mit Sprache zu steuern. Die Art von Applikationen, die erweitert werden können, sind mit einer genauen Instruktion in der SiriKit-Dokumentation zu finden. [SKDOC] Apple ist somit für freie Erweiterungen nicht so offen wie die Konkurrenz.

Minuspunkte sammelt Siri im Bereich von Übersetzungen sowie im Fremdsprachenverständnis. In diesen Punkten hinkt sie im Vergleich mit den anderen Assistenten weit hinterher bzw. hat Siri noch Schwierigkeiten einzelne Wörter in andere Sprachen übersetzen. Zudem weist die Apple-Lösung gegenüber Google und Amazon klare Wissenslücken auf. Oftmals bekommt der Anwender gar keine oder falsche Antworten (nicht zur Fragestellung passend) und Wissensfragen werden willkürlich sprachlich oder visuell beantwortet, was für den Benutzer unverständlich und auf Dauer nervig ist. Auch ortsbegrenzte Suchanfragen sind noch nicht wirklich ausgereift. Navigation und z. B. Restaurantvorschläge in der Nähe funktionieren jedoch einwandfrei, aber auf Dinge, wie örtliches Kinoprogramm, kann jedoch nur selten zugegriffen werden. [BLE17, S. 85]

Zusammenfassend lässt sich sagen, dass Siri als älteste Assistentin zwar ausgereift ist, aber dennoch einige Schwächen aufweist und sowohl Amazon als auch Google sie in vielen Bereichen überholt haben. Wie von Apple bekannt, funktioniert alles solange gut, sofern Apple-Produkte und -Software benutzt wird. So auch im Bereich von Smart Home. Auch hier ist der Anwender auf das firmeneigene HomeKit angewiesen, was auch in diesem Bereich eher Minuspunkte sammelt und sich somit nicht für den Gebrauch in dieser Studienarbeit anbietet. [BLE17, S. 85]

3.3.3. Google Assistant

Schon im Jahr 2012 kam Google Now auf den Markt, die Vorversion des heutigen Google Assistant. Somit konnte sich auch dieser Sprachassistent schon einige Jahre auf dem Bereich der Spracherkennung und -verarbeitung entwickeln. Der Assistent ist auf jeglichen Android-Geräten seit Android 6.0 erhältlich und mittlerweile gibt es sogar eine iOS-Version für die Apple-Smartphones. Nachdem der Internetriese immer mehr Marktanteile an Amazon abgeben musste und die Google-Suchanfragen seit der Veröffentlichung des Echo-Lautsprechers zurückgingen, erkannten sie das Potenzial von smarten Lautsprechern. Ende 2016 (2017 mit deutscher Sprachenunterstützung) brachte das Unternehmen daher mit ihrem Google Home einen eigenen smarten Lautsprecher auf den Markt. [BAG17, S. 66 f]

Der Assistent punktet vor allem durch herausragende Smartphone-Integration, sodass die applikationsübergreifende Interaktion sehr gut funktioniert. Per Sprache kann die Kamera geöffnet werden, eine WhatsApp-Nachricht diktiert und versendet sowie sogar der Flugmodus aktiviert werden, was ein selbstständiges Schließen bzw. Beenden des Assistenten zur Folge hat. [BLE17, S. 84]

Weitere Vorteile von Google sind auf dessen riesigen Datenfundus zurückzuführen und somit ist der Assistant auch im Allgemeinwissen marktführend. Er kann z. B. bei ortszugängigen Informationen, wie Kinoprogramm oder auch Navigation, die Konkurrenz ausstechen. Die Entscheidung, welche der Assistent über die Form der Ausgabe des Ergebnisses fällt (mündlich, visuell oder textlich), wirkt intuitiv und gut durchdacht. Weiterhin zählt der Assistent genauso wie Cortana zu den Sprachgenies und kann sogar ganze Sätze übersetzen. [BLE17, S. 84]

Ein Alleinstellungsmerkmal ist die Stimmenerkennung, denn der Google Assistant ist bisher als einziger in der Lage, Stimmen zu unterscheiden und kann somit sogar durch mehrere Benutzer differenziert bedient werden. [CLA18] Hinzu kommt, dass der Assistent kontextbezogene Nachfragen ermöglicht. Google analysiert jede einzelne Frage und merkt sich die Informationen darin, um später einen Zusammenhang zu folgenden Fragen zu haben. Das beherrscht der Assistent eindeutig besser als die Konkurrenz und somit lassen sich annähernd vollständige Gespräche führen. [BLE17, S. 84]

Aufgrund der langen Laufzeit und dem riesigen Entwicklungsaufwand seit der Erstveröffentlichung ist auch die Sprachsynthese so gut, dass man alles versteht. Jedoch hat der Google Assistant auch noch eine recht roboterhafte Stimme. Die Heise online Tester beschrieben die Stimme als „etwas hochnäsig und humorarm“. [BLE17, S. 84] Hierbei hinterlässt Alexa einen wesentlich besseren Eindruck. Bei der Kommunikation spielt das Gefühl eben eine große Rolle. Der Benutzer urteilt hier sehr subjektiv und empathisch

und redet natürlich lieber mit jemandem, der eine freundliche Stimme hat. [BLE17, S. 84]

Analog zu den Alexa Skills gibt es für den Google Assistant die Actions on Google, bei denen benutzerdefinierte Anwendungen implementiert und hinzugefügt werden. Bislang gibt es aber bei weitem weniger vorhandene und vor allem auch kaum deutschsprachige Actions, da Amazon mit ihrem Produkt einfach viel früher dran war und hier klare Marktvorteile genießt. Doch auch mit der geringeren Anzahl an Actions deckt Google mit Hilfe ihrer Suchmaschine ein weites Spektrum an Anwendungsfällen ab und ist somit auch in diesem Bereich auf dem Vormarsch. [CLA18]

Im Bereich von Smart Home nehmen sich beide Assistenten, also von Google und Amazon, nicht viel. Sie unterstützen ähnlich viele Geräte und lassen sich auch noch erweitern. Vorteile hat nur Google bei der Sprachbedienung, da diese oftmals intuitiver erscheint. [CLA18]

Der Google Assistant und Alexa dominieren aktuell größtenteils den Markt. Noch hat Alexa aufgrund der früheren Veröffentlichung gewisse Vorteile an Verbreitungsgrad im Bereich Smart Home, die es für Google gilt aufzuholen. Durch den riesigen Deep Learning Mechanismus von Google dürfte dies aber nicht mehr all zu lange dauern. Für diese Studienarbeit würde ebenfalls der Google Assistant in Frage kommen. Aufgrund der Verbreitung und vermehrt subjektiver Entscheidungen wurde die Entscheidung zugunsten Alexa schon im Vorfeld der Studienarbeit getroffen.

3.3.4. Microsoft Cortana

Im Jahr 2014 kam mit Cortana Microsofts Antwort im Bereich der Sprachassistenten auf den Markt. Cortana ist hauptsächlich auf Geräten mit dem Betriebssystem Windows 10 zu finden. Der Assistent wird hier direkt mitgeliefert.

Cortanas Alleinstellungsmerkmal gegenüber ihren Konkurrenten ist ihre Fähigkeit zu übersetzen, denn sie kann neben Wörtern auch Sätze per Sprachbefehl durch den Microsoft Translator in verschiedenste Sprachen übersetzen und in den meisten Fällen das Ergebnis auch vorlesen. In diesen Tests schneidet der Assistent sogar besser ab, als der Google Assistant. Ähnlich wie Siri ist Cortana auf das Betriebssystem sehr gut zugeschnitten und somit lassen sich Programme ohne Probleme per Sprachsteuerung öffnen. [BLE17, S. 82]

Im Vergleich zu den Konkurrenzprodukten muss Microsoft aber in einigen Punkten zurückstecken. Zwar wirken Cortanas Antworten nicht künstlich, sondern sie zeigt sogar etwas Empathie und Mitgefühl, aber dennoch klingt ihre Stimme hölzern. Für den Benutzer ist dies sehr gewöhnungsbedürftig. [BLE17, S. 82] Weiterhin ist die Anzahl der unterstützten Sprachbefehle gegenüber der Assistenten der Konkurrenz sehr

eingeschränkt. Um diese Lücke zu stopfen gibt es eine von Microsoft bereitgestellte Programmierschnittstelle, aber im Gegensatz zu Amazon und Google ließen sich hier noch nicht viele externe Programmierer finden. Somit gibt es immer noch weniger als 100 Skills, die hinzugekommen sind und diese leider auch nur auf Englisch. [BAG17, S. 67]

Auch im Bereich von Sprachsuchen hat Cortana das Nachsehen gegenüber Google, Amazon, Apple und Samsung. Viele Antworten kann sie nicht selbst geben und öffnet stattdessen den Browser und überlässt dem Benutzer die Arbeit bei der Suche via Suchmaschine. Dabei benutzt sie immer Edge als Webbrowser und Bing als Suchmaschine, was sich auch nicht umstellen lässt. Minuspunkte gibt es nicht nur für diesen Zwang, sondern auch dafür, dass sie die Antworten, welche sie nicht selber beantworten kann, oftmals im Browser auch gar nicht vorliest. Edge muss der Benutzer auch selbst wieder schließen, was nicht sonderlich benutzerfreundlich ist. [BLE17, S. 82]

Die Sprachsuchergebnisse sind zudem auch selten zufriedenstellend. Hierzu ein Beispiel: Auf die Frage „Seit wann gibt es dich?“ öffnet sie die Homepage www.seid-seit.de, was natürlich erstens nichts mit der Fragestellung zu tun hat und zweitens der Benutzer zusätzlich noch das Fenster selbst wieder schließen muss.

Dies alles wird auch der Grund sein, warum Cortana gegenüber den anderen Assistenten wenig Beachtung geschenkt wird. Jedoch soll noch im Laufe des Jahres 2018 eine Kooperation zwischen den beiden Freundinnen Cortana und Alexa ermöglicht werden. Somit könnte Alexa mit einem Sprachbefehl aufgefordert werden Cortana zu öffnen und andersherum. Die Entwickler erhoffen sich dadurch die Schwächen gegenseitig auszumerzen. Somit könnte der Benutzer dann über Cortana bei Amazon einkaufen. Zudem ist einer der wichtigsten Punkte, dass Cortana somit erstmals auf einem intelligenten Echo-Lautsprecher verfügbar sein wird und dadurch auch zum ersten Mal mit Smart Home in Verbindung gebracht wird. Bis dato kommt die Verwendung von Cortana aber nicht für diese Studienarbeit in Frage.

3.3.5. Samsung Bixby

Der jüngste unter den digitalen Assistenten ist die im Jahr 2017 von Samsung veröffentlichte Sprachassistentin Bixby. Diese ist bisher nur auf den Samsung Smartphones Galaxy S9, Galaxy S8 und durch eine Zusatzinstallation auf den Samsung Galaxy S7 Geräten verfügbar und genießt damit eine sehr beschränkte Verbreitung. Die Installation besteht aus Bixby Voice, den eigentlichen Sprachassistenten, Bixby Home, einer digitalen Pinnwand sowie Organisationseinheit und Bixby Vision. Letzteres ist in der Lage ein Objekt mit der Kamera oder in der Galerie App zu scannen und zu diesem Objekt im Internet Informationen zu sammeln. Hierbei können auch Shops gesucht werden, welche das gescannte Objekt verkaufen. [WIG17]

Zunächst wurde der Assistent nur auf Koreanisch und Chinesisch veröffentlicht, aber mittlerweile wurde auch die englische Sprache hinzugefügt. Deutsch fehlt noch immer, wodurch sich der geringe Verbreitungsgrad hierzulande von selbst erklärt.

Ähnlich wie bei Siri kann der Benutzer bei Bixby zwischen verschiedenen Stimmen wählen. Tester beschrieben die Stimme Stephanie mit einem angenehmen Klang, John mit einer sympathischen Stimme und Julia mit einer tieferen Aussprache. Alle drei zeichnen sich aber jeweils durch eine klare und deutliche Kommunikation aus. [REI17]

Beim Sprachverständnis hat der Assistent jedoch noch seine Probleme. In den Einstellungen kann zwischen drei Empfindlichkeitsstufen ausgewählt werden, aber die höchste kommt bei weitem nicht an Google heran. Somit muss das Schlüsselwort „Hi Bixby“ öfters lauter wiederholt werden. Weiterhin gibt es einige Schwierigkeiten mit Dialekt, aber dafür wird die Stimme mit der Zeit gelernt und auch ein manuelles Sprachtraining ist möglich. [REI17]

Im Bereich von übergreifender Interaktion kann Bixby ordentlich punkten. Allgemeine Dinge wie Wecker, Wetter sowie YouTube funktionieren einwandfrei und teilweise sogar besser als bei allen anderen Konkurrenzprodukten. So schießt der Befehl „take a photo“ direkt ein Foto und öffnet nicht nur wie Google die Kamera. In den Feedback-Einstellungen kann der Benutzer zu dem entscheiden, ob die Antworten auf Fragen akustisch, visuell oder ausschließlich textlich sowie kurz oder lang beantwortet werden sollen. [REI17]

Allgemein lässt sich sagen, dass Bixby bisher wenig Zeit hatte sich zu beweisen. Dafür müsste es aber auch für mehr Sprachräume und Geräte zugänglich gemacht werden. Zudem hat es einige Nachteile, da auf denselben Smartphones auf denen Bixby erhältlich ist auch Google mit deren Produkt vertreten ist. Im direkten Vergleich schneidet Bixby durch das schlechtere Sprachverständnis und fehlendes Verknüpfen mit vorhergehenden Fragen hier schlechter ab. Dennoch hat Bixby großes Potential und kann durch mehr Erfahrung auf dem Markt den Vorsprung der Konkurrenz aufholen. [HAR17] Da bisher auch keine Smart Home Unterstützung verfügbar ist, scheidet der Assistent ebenfalls für diese Studienarbeit aus.

3.3.6. Vergleichsfazit

In der folgenden Tabelle wird der Vergleich der Assistenten Amazon Alexa [ALEXAa], Apple Siri [SIRI], Google Assistant [GOASS], Microsoft Cortana [MICOR] und Samsung Bixby [BIXBY] anhand der vorher definierten Kriterien zusammengefasst. Die Einstufung in den jeweiligen Kategorien wurde selbst vorgenommen und soll die Leistungsunterschiede schematisch darstellen. Die Begründung für die jeweilige Einstufung folgt in den nächsten

Unterkapiteln.

	 amazon alexa	 Hey Siri	 Google ASSISTANT	 Hi, I'm Cortana.	 Bixby For Everyone
Sprachsynthese	✓✓✓	✓✓✓	✓✓	✓✓	✓✓
Spracherkennung	✓✓	✓✓	✓✓✓	✓✓	✓✓
Verbreitung	✓✓✓	✓✓	✓✓✓	✓✓	✓
Allgemeinwissen	✓✓	✓	✓✓✓	✓	✓✓
Übersetzungsfähigkeiten	✓	✓	✓✓	✓✓✓	✗
Funktionsumfang	✓✓✓	✓✓✓	✓✓✓	✓✓	✓✓✓
Erweiterbarkeit	✓✓✓	✓	✓✓✓	✓✓	✗
Smart Home Unterstützung	✓✓✓	✓✓	✓✓✓	✗	✗
Gender-Correctness	✗	✓✓	✗	✗	✓✓

Tabelle 3.1.: Formaler Vergleich der aktuell meist verbreitetsten Sprachassistenten

Es bestätigt sich somit die Annahme, dass Alexa und Google Assistant zurecht Marktführer sind, da sie in allen Punkten die Konkurrenzprodukte ausstechen. Im Vergleich miteinander trumpft Google nur mit seinen Übersetzungsfähigkeiten. Dies ist aber speziell für die Studienarbeit ein belangloses Kriterium. Umso wichtiger ist der Bereich Smart Home, in denen beide gleich gut abschneiden und durch die Skills bzw. Actions on Google zudem noch frei erweiterbar sind.

Somit lässt sich sagen, dass sowohl Amazon Alexa als auch der Google Assistant formal beide für die Verwendung in diesem Projekt anbieten. Die Wahl für Alexa beruht daher, wie bereits erwähnt, auf subjektiven Punkten. Bereits vor Beginn waren Geräte wie ein Echo-Dot und jeweils ein Fire-TV und somit auch die Alexa-Sprachassistentin in Besitz und in Anwendung. Die Anschaffung eines smarten Google-Lautsprechers wäre demnach mit zusätzlichen Kosten verbunden. Zudem kann die Installation direkt im eigenen Zuhause verwendet und später mit mehr Geräten erweitert werden. Somit wird in dieser Arbeit Amazons Produkt verwendet und im Folgenden auch ausschließlich behandelt.

3.4. Raspberry Pi

Die Raspberry Pis sind eine Reihe von Einplatinencomputern, die im Vergleich zu herkömmlichen Rechnern heutzutage sehr preiswert sind. Entwickelt werden sie von der Raspberry Pi Foundation, welche das erste Modell im Jahr 2012 auf den Markt gebracht hat. Die ursprüngliche Idee war es, einen kleinen, flexiblen und billigen Rechner, vor allem für Lehr- und Lernzwecke im Bereich von Hardware und Programmierung, bereitzustellen. Weiterhin wird der Raspberry Pi auch sehr gern von Hobbyprogrammierern für Projekte von und für das eigene zu Hause verwendet. [SOP17, S. 1] Die Liste von den Projekten ist mittlerweile über die Jahre riesig geworden und umfasst Dinge, wie einen einfachen Low-Budget Desktop PC, eine retro Videospielkonsole, selbst gebaute Router und eigene personalisierte Smart Home Lösungen, welche in dieser Studienarbeit behandelt werden. Die Vielzahl an Möglichkeiten ist auch das Hauptargument für die hohen Verkaufszahlen: Bis Ende 2017 wurden insgesamt über 17 Millionen Raspberry Pis verkauft. [HEA17]

In dieser Studienarbeit wird ein Raspberry Pi 3 Modell B verwendet, welcher zu Beginn der Bearbeitung die aktuellste Version war. Im Laufe der Bearbeitung wurde dann das Modell B+ veröffentlicht, welches eine nochmals verbesserte Hardware besitzt. Die für das Projekt verwendete Platine ist im folgenden Bild dargestellt. [RASPc]

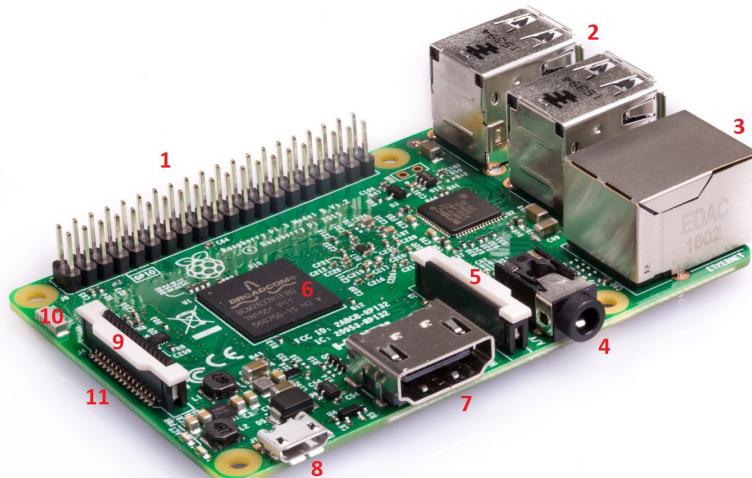


Abb. 3.1.: Raspberry Pi 3 Model B

Besonders auffällig sind die 40 General Purpose Input/Output (GPIO) Pins (1), welche standardmäßig unbelegt sind und an die Ein- und Ausgabekontakte externe Hardware wie LEDs, Sensoren oder Motoren angeschlossen werden können.

An der schmalen Seite befinden sich vier USB-Anschlüsse (2), welche einen Datentransfer

von 480 Mbit/s gewährleisten.

Direkt daneben ist der 10/100 Ethernet-Port (3) platziert, welcher eine Datenrate von 100 Mbit/s schafft und somit den Rechner mit dem Internet oder einem anderen Netzwerk verbinden kann. Für eine drahtlose Verbindung gibt es zusätzlich noch einen Wireless LAN und Bluetooth 4.1 Adapter (10), der sogar eine Datenrate von 150 Mbit/s schafft, Weiterhin gibt es eine kombinierte 3,5 mm Ausgangsbuchse (4) für Audio und Komposit-video sowie Ports für das Camera Interface (5) und das Display Interface (9).

An der Position (7) in der Abbildung 3.1 befindet sich ein Full HDMI-Port, um einen Monitor anzuschließen. Dies wird in diesem Projekt nur bei der erstmaligen Einrichtung benötigt, bevor danach immer per Remote auf den Rechner zugegriffen wird.

Daneben liegt der Micro-USB-Anschluss (8). Dieser wird für die Stromversorgung benutzt und ist für maximal 2,5 Ampere bei 5 Volt ausgelegt.

Auf der Unterseite der Platine unter dem Camera Interface befindet sich der MircoSD-Card-Slot (11). Auf der eingelegten Karte befindet sich dann das Betriebssystem, das auf dem Rechner laufen wird. Eines der bekanntesten und auch speziell für den Gebrauch auch dem Raspberry Pi ausgelegten Systeme ist Raspbian, welches eine Linux-Distribution ist und auf Debian basiert. In dieser Studienarbeit wird die Version Raspbian Stretch verwendet.

In der Mitte auf der Platine befindet sich die 64-bit CPU samt dem Arbeitsspeicher (6). Das gewählte Modell hat eine Prozessorleistung von 1,2 GHz auf 4 Kernen und besitzt 1 GB SDRAM. [SOP17, S. 6-12]

Für das Ansteuern von Funksteckdosen wird noch ein Funkmodul benötigt. In diesem Projekt wurde ein 433MHz Funkmodul (Siehe 3.5 Funkstandards) von Aukru verwendet, welches über einfache Kabel an die entsprechenden GPIO-Pins angeschlossen wurde. Das folgende Bild zeigt den vollständigen Aufbau, der in der Studienarbeit verwendeten Hardware samt Raspberry Pi, Funkmodul, Funksteckdose Netzteil und Kamera.

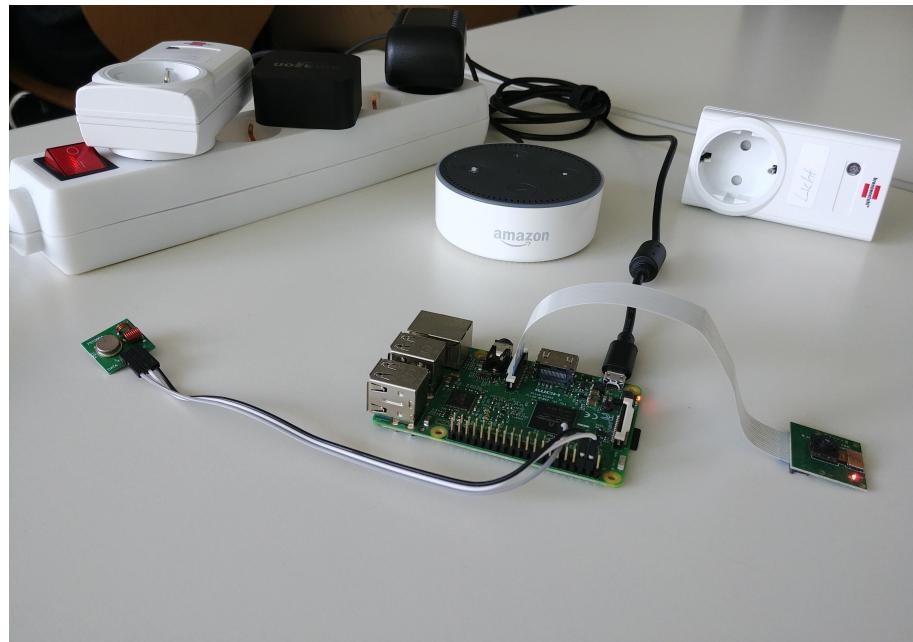


Abb. 3.2.: Aufbau des für das Projekt installierten Systems mit Raspberry Pi

Wie bereits beschrieben wurde ein Monitor nur einmalig zur Einrichtung angeschlossen.

3.5. Funkstandards

Im Bereich der Hausautomation müssen alle Geräte im System miteinander kommunizieren, was zu einem großen Teil über Funk geschieht und wofür es bestimmte Standards gibt. Die wohl am meist bekannten sind WLAN/Wi-Fi und Bluetooth, die ursprünglich nicht für diesen Zweck geschaffen wurden, aber sich trotzdem zur Verwendung anbieten. Weiterhin gibt es speziell für die Hausautomation geschaffene Funkstandards. Beispiele für weitere herstellerunabhängige sind ZigBee, EnOcean oder Z-Wave als Weltmarktführer in der Hausautomation. [SCH15]

Ein für die komplette Hausautomation eher selten eingesetzter Standard ist das ISM-Band (industrial, scientific, medical). Die Nutzung des Frequenzbereich von 433,05 MHz bis 434,79 MHz ist nicht beschränkt, kostenlos, anmeldefrei und wird somit auch gern im Betriebs- und Amateurfunk eingesetzt, wie z. B. in Handfunkgeräten, Funkschaltern oder Funkthermometern. Aus diesem Grund wird das ISM-Band mit 433 MHz (auch als 70-Zentimeter-Band bezeichnet) [RNW07] in dieser Studienarbeit zur Ansteuerung der Funksteckdosen genutzt. (Siehe 4.4.2 Ansteuerung der Funksteckdosen)

Der gewählte Standard hat eine maximale Leistung von 10 mW und eine Reichweite von 0,3 km in der Stadt und 2,5 km auf freier Fläche. Aus diesem Grund liegt die

größte Verwendung bei Geräten wie Garagentoröffnern oder Autoschlüsseln, da der Aktivierungsradius (kleiner 100 m) sehr gering und die Nutzungsdauer (0,1 - 3 Sekunden) kurz ist. Auf Grund der unbeschränkten Nutzung kann es aber häufig auch zu Störungen kommen, weil die Frequenz blockiert ist. Das kann z. B. passieren, wenn ein Funkkopfhörer im Umkreis von 50 Metern auf derselben Frequenz arbeitet. Es kann zudem auch dazu kommen, dass der Anwender mit seiner Funkfernbedienung, die Geräte des Nachbarn steuert. Eine sichere Übertragung kann also nur bei einer leistungsfähigen Kanalcodierung oder bei einer Verwendung von Modulen mit mehreren Kanälen, die dann auf andere Frequenzen ausweichen können, gewährleistet werden. Dies ist dann aber auch mit erhöhten Kosten verbunden. [RNW07]

Bei denen in dieser Studienarbeit verwendeten Funksteckdosen kann der Benutzer selbst einen eigenen Hausschlüssel anlegen, sodass die Steckdose nur auf Signale mit diesem Schlüssel reagiert und nicht auf mögliche Störungen vom Nachbarn mit denselben Steckdosen. Dies wird unter 4.4.2 Ansteuerung der Funksteckdosen noch genauer erläutert.

3.6. OAuth2

Das Autorisierungsframework und offene Protokoll OAuth2 (Open Authorization 2) ist ein Industriestandard [RF6749] für sichere API-Autorisierung in den Bereichen Mobile-, Web-, Heim- und Desktop-Anwendungen. OAuth2 ist der Nachfolger des 2006 erstellten Protokoll OAuth. Das Framework ermöglicht es einer Drittanbieter-Anwendung einen eingeschränkten Zugriff auf einen HTTP(S)-Service zu erhalten. Dies geschieht entweder direkt im Namen eines Ressourceneigentümers, durch Orchestrierung einer Genehmigungsinteraktion zwischen dem Ressourceneigentümer und dem HTTP-Service, oder indem der Drittanbieter-Anwendung der Zugriff auf den Service nach eigenem Ermessen gewährt wird. [OAUTHa]

3.6.1. Rollen

Gemäß der Spezifikation von OAuth2 [RF6749] werden die Beteiligten in vier verschiedene Rollen unterteilt [OAUTHb]:

- Client (Drittanbieter-Anwendung)

Der Client ist die Anwendung, die versucht, Zugriff auf das Benutzerkonto des Benutzers zu erhalten. Es muss die Erlaubnis des Benutzers einholen, bevor es dies tun kann.

- Ressourcenserver (API)

Der Ressourcenserver ist der API-Server, der für den Zugriff auf die Informationen des Benutzers verwendet wird.

- Autorisierungsserver

Dies ist der Server, der die Schnittstelle darstellt, auf der der Benutzer die Anfrage genehmigt oder ablehnt. Die Interaktion zwischen Autorisierungsserver und Ressourcenserver wird über das Protokoll nicht weiter spezifiziert. In kleineren Implementierungen kann dies derselbe Server wie der API-Server sein, aber bei größeren Implementierungen wird dies oft als separate Komponente aufgebaut.

- Ressourcen-Eigentümer (Benutzer)

Der Ressourcen-Eigentümer ist eine Einheit, die Zugang zu einem Teil einer geschützten Ressource gewähren kann. Meistens ist dies eine Person als Endbenutzer, welche Zugriff auf Teile ihres Accounts gewährt.

3.6.2. Protokollablauf

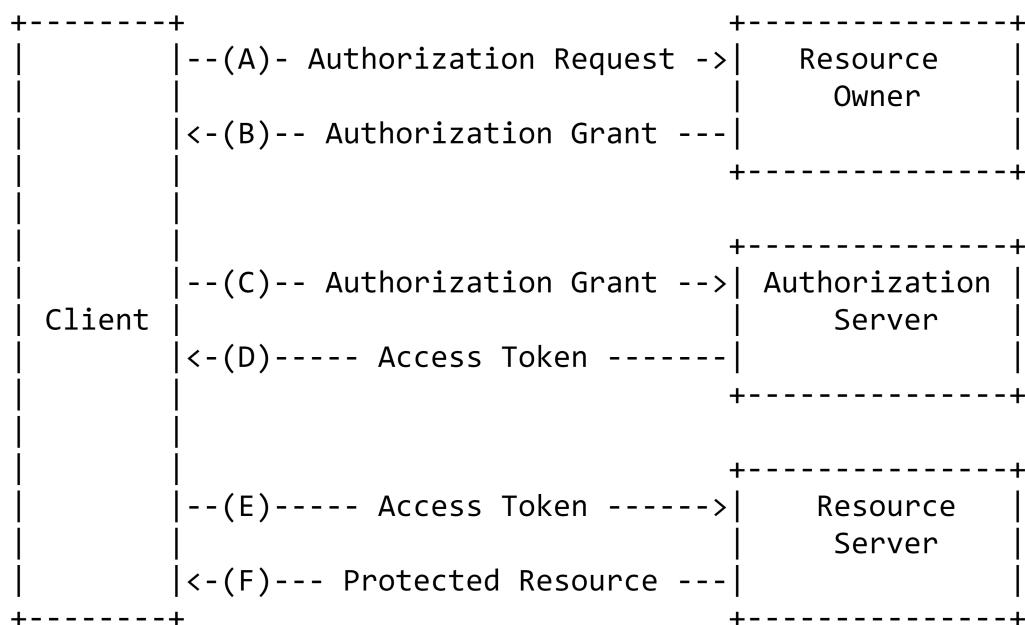


Abb. 3.3.: Abstrakter Protokollablauf [RF6749]

Der in Abbildung 3.3 dargestellte OAuth2-Protokollablauf beschreibt die Interaktion zwischen den vier Rollen und umfasst die folgenden Schritte [RF6749]:

- (A) Autorisierungsanfrage (Authorization Request)

Der Client fordert die Berechtigung des Ressourceninhabers an. Die Berechtigungsanfrage kann direkt an den Ressourceneigentümer gestellt werden, wie abgebildet, oder vorzugsweise indirekt über den Autorisierungsserver als Vermittler.

(B) Berechtigungsvergabe (Authorization Grant)

Der Client erhält eine Berechtigungsvergabe, welche ein Berechtigungsnachweis für die Autorisierung durch den Ressourceneigentümers darstellt. Hierfür gibt es vier, durch OAuth2 spezifizierte, verschiedene Verfahren zur Berechtigungsvergabe. Hierzu mehr in Abschnitt 3.6.3.

(C) Berechtigungsvergabe (Authorization Grant)

Der Client fordert ein Access-Token an, indem er sich beim Autorisierungsserver authentifiziert und die Berechtigungsvergabe vorhält.

(D) Zugriffstoken (Access Token)

Der Autorisierungsserver authentifiziert den Client und validiert die Berechtigungsvergabe und stellt, falls gültig, ein Zugriffstoken aus.

(E) Zugriffstoken (Access Token)

Der Client fordert die geschützte Ressource von dem Ressourcenserver an und authentifiziert sich dabei durch Vorlage des Zugriffstoken.

(F) Geschützte Ressource (Protected Resource)

Der Ressourcenserver validiert das Zugriffstoken und gibt, falls gültig, die geschützten Ressourcen zurück. Diese können z. B. die Identifikation des Benutzers im System, der Benutzername, die E-Mail Adresse oder weitere Daten sein.

3.6.3. Berechtigungsvergabe

Die Art der Berechtigungsvergabe hängt von der Methode ab, die der Client benutzt um eine Autorisierung anzufordern und von den Typen, die vom Autorisierungsserver unterstützt werden. Es werden hierfür fünf verschiedene Verfahren in OAuth2 beschrieben:

- Autorisierungscode Erteilung (Authorization Code Grant)
- Implizite Erteilung (Implicit Grant)
- Kennwortberechtigungen für Ressourceneigentümer gewähren (Resource Owner Password Credentials Grant)

- Kundenanmeldeinformationen gewähren (Client Credentials Grant)
- Erteilung durch Erweiterung (Extension Grants)

Die bevorzugte Methode für den Kunden, eine Berechtigungsvergabe vom Ressourcen-Eigentümer zu erhalten (dargestellt durch die Schritte (A) und (B) in Abbildung 3.3), ist die Verwendung des Autorisierungsserver als Intermediär. Dieses Verfahren wird in der OAuth2 Spezifikation Authorization Code Grant genannt.

Das Alexa Skills Kit unterstützt sowohl Authorization Code Grant (für Custom-Skills und Smart Home-Skills), als auch Implicit Grant (nur für Custom-Skills). Da in dieser Studienarbeit auf die Smart Home-Skill API zurückgegriffen wird, ist die Verwendung des Authorization Code Grant Verfahren unerlässlich.

Im Folgenden wird nun nur auf das Authorization Code Grant Verfahren näher eingegangen, da die anderen Verfahren für die Studienarbeit nicht von Bedeutung sind.

3.6.4. Authorization Code Grant

Das Authorization Code Grant Verfahren wird verwendet, um die beiden Zugriffsarten, Zugriffstoken und Aktualisierungstoken, zu erhalten. Das Verfahren ist für vertrauliche Clients optimiert. Da es sich hierbei um einen redirektionsbasierten Ablauf handelt, muss der Client in der Lage sein, folgende Aufgaben zu erfüllen:

- Interaktion mit dem User-Agenten des Ressourcen-Eigentümers (typischerweise ein Web-Browser)
- Eingehende Anfragen vom Autorisierungsserver empfangen (über Weiterleitung im Browser)

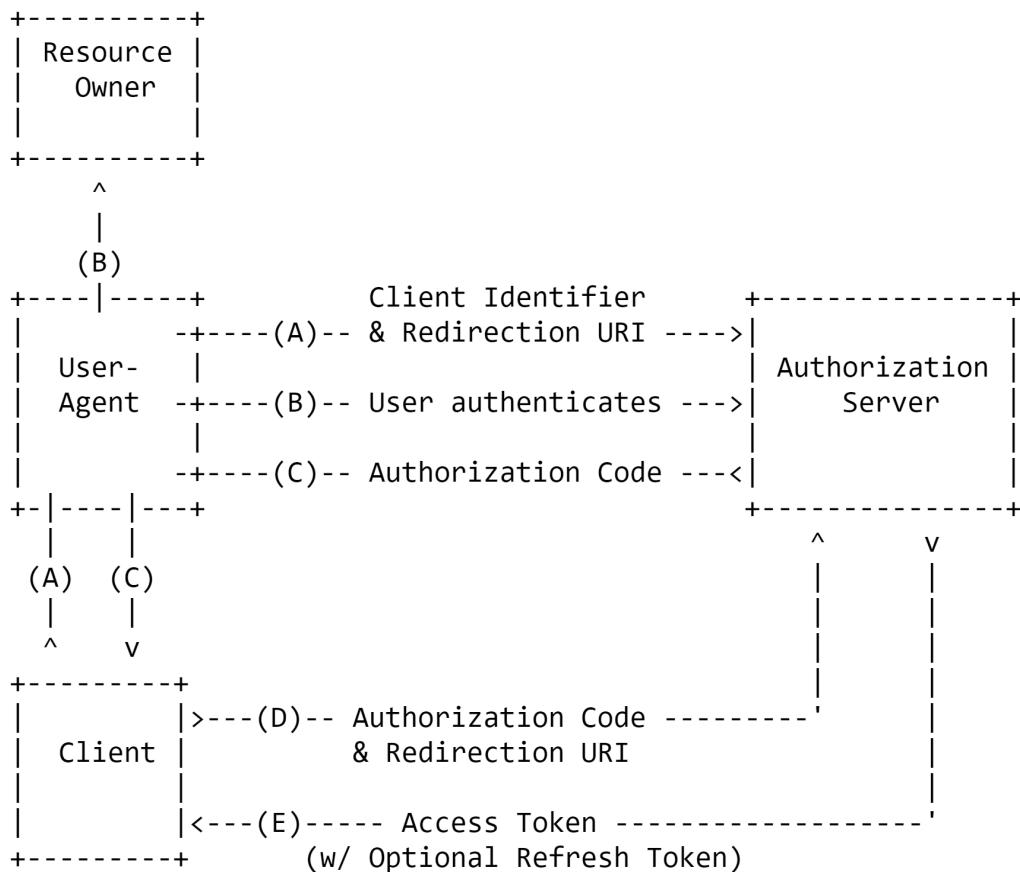


Abb. 3.4.: Authorization Code Grant [RF6749]

Das in Abbildung 3.4 dargestellte Verfahren beinhaltet die folgenden Schritte (hier zu beachten: Schritt A, B und C sind in zwei geteilt, da sie durch den User-Agenten geschleust werden) [RF6749]:

- (A) Client Identifikation & URI Umleitung (Client Identifier & Redirection URI)

Der Client initiiert den Ablauf, indem er den User-Agent des Ressourcen-Eigentümer mit dem Autorisierungsserver verbindet. Der Client verschickt seine Kundenkennung (Client Identifier), den angeforderten Bereich (Requested Scope), den lokalen Zustand (Local State) und eine Umleitungs-URI (Redirection URI), an die der Autorisierungsserver den User-Agent zurück schickt, sobald der Zugriff gewährt (oder verweigert) ist.

- (B) Benutzer authentifizieren (User Authenticates)

Der Autorisierungsserver authentifiziert den Ressourceneigentümer (über den User-Agent) und stellt fest, ob der Ressourceneigentümer die Zugriffsanfrage des Clients gewährt oder verweigert.

- (C) Autorisierungscode Rückgabe (Authorization Code)

Unter der Annahme, dass der Ressourcen-Eigentümer den Zugriff gewährt, leitet der Autorisierungsserver den User-Agenten über die zuvor, in der Anfrage oder bei der Client-Registrierung, bereitgestellte Umleitungs-URI an den Client zurück. Die Umleitungs-URI enthält einen Autorisierungscode und alle lokalen Zustände, die der Kunde zuvor angegeben hat.

- (D) Übergabe von Autorisierungscode & URI Umleitung (Authorization Code & Redirection URI)

Der Client fordert vom Token-Endpunkt des Autorisierungsservers ein Zugriffstoken an, indem er den im vorherigen Schritt erhaltenen Autorisierungscode der Anfrage beifügt. Bei der Anfrage authentifiziert sich der Client beim Autorisierungsserver somit durch den Autorisierungscode. Zudem fügt der Client die Umleitungs-URI, welche er genutzt hat um den Autorisierungscode zu erhalten, seiner Anfrage an, um sich zusätzlich zu verifizieren.

- (E) Rückgabe von Zugriffstoken mit optionalem Aktualisierungstoken (Access Token with Optional Refresh Token)

Der Autorisierungsserver authentifiziert den Client, validiert den Autorisierungscode und stellt sicher, dass die empfangene Umleitungs-URI mit der URI übereinstimmt, welche für die Umleitung des Clients in Schritt (C) verwendet wurde. Falls gültig, antwortet der Autorisierungsserver mit einem Zugriffstoken (Access-Token) und optional mit einem Refresh-Token.

3.6.5. Bearer Token

Das Bearer Token (Inhaberkennzeichen) [RF6750] ist eine genauer spezifizierte Variante des bereits erwähnten Zugriffstoken (Access-Token). Ein Zugriffstoken wird durch den OAuth2 Standard [RF6749] wie folgt beschrieben: „a string representing an access authorization issued to the client“, also eine Zeichenkette, die eine Zugriffsberechtigung repräsentiert, die dem Client erteilt wurde. Mithilfe von Zugriffstoken wird dem Client somit ermöglicht, sich bei einem Autorisierungsserver, ohne Anmeldedaten wie z. B. Benutzername und Passwort, zu authentifizieren.

3.7. Development-Tools

In diesem Abschnitt werden, die zur Entwicklung verwendeten Tools und Frameworks, beschrieben.

3.7.1. Amazon Developer Services

Mit den Amazon Developer Services wird jedem Benutzer, der eine E-Mailadresse besitzt, die mit einem Amazon-Konto verknüpft ist, die Möglichkeit geboten, eigene Applikationen für jegliche Geräte auf Android oder iOS basierend oder für die Amazon eigenen Fire Tablets und Fernseher kostenlos zu entwickeln, zu builden, zu testen und sogar über den Amazon Appstore zu vertreiben. Nach der Registrierung eines Amazon Developer Accounts stehen dem Entwickler etliche Services und APIs zur Verfügung, die direkt eingebunden und benutzt werden können. Das umschließt bspw. vorgefertigte Login-Formulare durch Amazon-Konten, Werbung für die eigene Applikation auf den Amazon-Fire-Geräten zu schalten oder Multi-Screen-Möglichkeiten mit zwei-Wege-Kommunikation zwischen Fire TV und der Smartphone Anwendung. Weiterhin werden verschiedenste Plugins und Extensions wie Unity, Apache Cordova oder Xamarin unterstützt. [ROU17a]

Das für dieses Projekt wichtigste Werkzeug ist das Alexa Skills Kit. Dieses ermöglicht es eigene Skills, also eine Art Fähigkeit bzw. App, selbst zu erstellen und zu programmieren. Mit selbst erstellten oder von Drittanbietern stammenden Skills kann Alexa bestimmte eigens konzipierte Anwendungsfälle, durch Erkennung bestimmter Wörter, erkennen und auf diese reagieren. Diese selbst erstellten Skills können dann auch über den Alexa Skills Store vertrieben werden. [ROU17b]

Über die eigenen Skills können dann über die Verbindung mittels des Raspberry Pi die angeschlossenen Geräte wie Funksteckdosen oder die Kamera, welche sonst nicht von Alexa aus erreichbar wären, angesprochen werden.

3.7.2. Amazon Web Services - Lambda

Amazon Web Services (AWS) ist einer der größten Cloud-Computing-Anbieter weltweit mit einem sehr umfangreichen Repertoire an Diensten. Amazon Web Services (AWS) bietet zahlreiche Cloud-basierte Anwendungen und Services an, wie z. B. diverse Server unterschiedlicher Art, Speicher, Netzwerkdienste, Datenbanken, Entwicklungstools, Verwaltungs-, Analyse- und Überwachungstools und noch mehr [AMA18g].

Einer dieser Cloud-basierten Services ist der AWS Lambda. Es ist „ein serverloser Datenverarbeitungsservice, der Ihren Code beim Eintreten bestimmter Ereignisse ausführt und automatisch für Sie die zugrunde liegenden Datenverarbeitungsressourcen verwaltet“ [AMA18h]. Mit „serverlos“ beschreibt Amazon, dass man sich um die Pflege und Wartung eines Servers nicht kümmern muss. Aus technischer Sicht stellt der Lambda Service dennoch eine Art Server dar, da durch ihn eine webbasierte Schnittstelle geschaffen wird, an welche Clients per HTTP(S) Nachrichten schicken können, um Dienste auszuführen. Amazon beschreibt serverlose Datenverarbeitung wie folgt: „Mit der serverlosen

Datenverarbeitung können Sie Anwendungen und Services erstellen und ausführen, ohne sich über Server Gedanken machen zu müssen. Für serverlose Anwendungen müssen Sie keine Server bereitstellen, skalieren und verwalten. Sie können sie für praktisch jeden Anwendungstyp oder Back-End-Service erstellen, und alles, was zum Ausführen und Skalieren Ihrer Anwendung mit hoher Verfügbarkeit erforderlich ist, wird für Sie durchgeführt “[AMA18i].

AWS Lambda kommt derzeit mit den Programmiersprachen bzw. Skriptsprachen C, Go, Java, JavaScript (mit Node.js) sowie Python zurecht [AMA18h]. Für AWS Lambda zahlt man nur so viel, wie man tatsächlich auch benötigt. Abgerechnet wird über die Anzahl Aufrufe von Lambda. Für eine Millionen Aufrufe zahlt man 0,20 US Dollar, wobei die ersten Millionen kostenlos sind [AMA18j].

AWS Lambda wird in dieser Studienarbeit als Schnittstelle (Siehe Abschnitt 4.4) zwischen der Amazon Alexa Cloud und dem Raspberry Pi verwendet, da eine direkte Kommunikation, zwischen Alexa Skills mit Smart Home-API (Siehe Abschnitt 4.4.1) und nicht-Lambda Servern, von Amazon untersagt wird.

3.7.3. Cloud9

Cloud9 ist „eine Cloud-basierte IDE zum Schreiben, Ausführen und Debuggen von Code“ [C9AWS]. Cloud9 wurde 2016 von Amazon akquiriert, läuft aber dennoch unter einer Open Source Lizenz und kann weiterhin, auch unabhängig von Amazon, kostenlos genutzt werden [C9.io]. In diesem Projekt kommt Cloud9 sowohl im Zusammenhang mit AWS, als auch auf dem Raspberry Pi zum Einsatz. Auf dem AWS Lambda wird direkt in Cloud9 programmiert, ohne zusätzliche Einrichtung. Auf dem Raspberry Pi wurde Cloud9 als Server eingerichtet (Siehe Abschnitt 5.1.2), sodass man von jedem beliebigen Computer, im selben Netzwerk des Raspberry Pi, direkt mit dem Webbrowser, über das Webinterface, auf dem Raspberry Pi entwickeln kann. Da Cloud9 eine integrierte Shell zur Verfügung stellt, wird selbst eine zusätzliche SSH überflüssig gemacht, um mit dem Raspberry Pi zu interagieren.

3.7.4. Express.js

Express.js ist ein „schnelles, offenes, unkompliziertes Web-Framework für Node.js“ [EXPJS]. Durch Express.js wird es den Entwicklern vereinfacht Webanwendungen mit Node.js (Siehe Abschnitt 3.7.7) zu erstellen. Dadurch ist es zu einer Art Standard Server Framework für Node.js geworden. Express.js ist frei verfügbar und wurde unter MIT License veröffentlicht. Damit ist die Wiederverwendung für frei einsehbaren, quelloffenen Code, als auch für

nicht einsehbaren, geschützten Code erlaubt. Express.js kam hier als Erweiterung von Node.js zum Einsatz. Aufgrund der vielen Werkzeugen die Express.js mit sich bringt, stellte es eine Hilfe für die Erstellung des Server Backend des Raspberry Pi dar.

3.7.5. FFmpeg

FFmpeg ist eine Open-Source Software und ein Multimedia-Framework zum Konvertieren, Codieren, Decodieren, Streamen von Multimedia Dateien [FMPEG]. Es besteht aus einer Sammlung von freien Open-Source Computerprogrammen und Programmzbibliotheken. FFmpeg ist auf vielen verschiedenen Plattformen lauffähig (wie z. B. Windows und Linux) und wird zudem auch von bekannter Software (wie z. B. dem VLC Player) verwendet. In dieser Arbeit wird FFmpeg für die Umwandlung und Bereitstellung von Videoinhalten der, an den Raspberry Pi angeschlossenen Kamera, verwendet.

3.7.6. Ngrok

Ngrok ist eine Anwendung mit Webservice zum Tunneln von öffentlichen, aus dem Internet zugreifbaren, ngrok URLs auf den PC eigenen localhost. Ngrok erleichtert den Schritt, vom Debuggen am eigenen Rechner oder Rechner aus dem mit Firewall geschützten Intranet bzw. Heimnetzwerk, hin zu einer aus dem Internet von überall zugänglichen URL. Für das Tunneln von nur einer einzigen Adresse pro Account, auf eine zufällig von ngrok zugewiesene HTTP bzw. HTTPS Adresse, mit nur 40 Verbindungen pro Minute sowie vier gleichzeitig aktiven, ist ngrok völlig kostenlos. Bei Bedarf an mehr fallen Kosten an [NGROK].

Hier in dieser Studienarbeit wird ngrok dazu verwendet, einen Tunnel zwischen AWS Lambda und Raspberry Pi zu schaffen, um eine Portweiterleitung im Heimnetz zu vermeiden und mittels HTTPS für zusätzliche Sicherheit zu sorgen.

3.7.7. Node.js

Node.js ist eine quelloffene, der MIT License unterliegende, serverseitige Plattform für diverse Netzwerkanwendungen und Webserver mit asynchroner, ereignisgesteuerter JavaScript Laufzeitumgebung. Es ist ressourcensparend und ermöglicht dabei dennoch eine große Anzahl gleichzeitig existierender Netzwerkverbindungen. Node.js benutzt die freie JavaScript-Laufzeitumgebung V8 von Google für die Ausführung. V8 wird auch im quelloffenen Webbrowser Google Chrome benutzt, ist in C++ geschrieben und kann in jede C++ Applikation eingebunden oder auch eigenständig ausgeführt werden. Der zu Node.js gehörende und weit verbreitete JavaScript Paketmanager nennt sich Node Package

Manager (NPM). Node.js kam in der Arbeit für die Realisierung des Server-Backend zum Einsatz. [NODEc]

3.8. Gebrauchstauglichkeit

Gemäß „DIN EN ISO 9241“ [ISO06] sind die Kriterien für die Gebrauchstauglichkeit (Usability) über die sieben „Grundsätze der Dialoggestaltung“ beschrieben:

- Aufgabenangemessenheit

„Ein interaktives System ist aufgabenangemessen, wenn es den Benutzer unterstützt, seine Arbeitsaufgabe zu erledigen.“

- Selbstbeschreibungsfähigkeit

„Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können.“

- Erwartungskonformität

„Ein Dialog ist erwartungskonform, wenn er den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen entspricht.“

- Lernförderlichkeit

„Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des interaktiven Systems unterstützt und anleitet.“

- Steuerbarkeit

„Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“

- Fehlertoleranz

„Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“

- Individualisierbarkeit

„Ein Dialog ist individualisierbar, wenn Benutzer die Mensch- System-Interaktion und die Darstellung von Informationen ändern können, um diese an ihre individuellen Fähigkeiten und Bedürfnisse anzupassen.“

4. Konzept

In diesem Abschnitt geht es um den grundsätzlichen Entwurf und das Konzept des praktischen Teils der Studienarbeit. Hier wird dargestellt, welches konkrete Ziel mit der Studienarbeit verfolgt und mit welchen Mitteln es erreicht werden soll.

4.1. Anforderungsanalyse

In der Anforderungsanalyse wird beschrieben, welche funktionalen und welche nicht-funktionalen Anforderungen an das Projekt gestellt sind.

4.1.1. Grund der Umsetzung

Wie einleitend in diese Studienarbeit erwähnt, wird versucht eine mögliche Reduktion der Anschaffungskosten der Hardware zu erzielen. Dafür wurde folgende Kostenaufstellung veranschlagt (mit Stand: Dezember 2017), wobei sich die Berechnung nur auf eine Installation mit einer Kamera bezieht.

Die Amazon-Variante beinhaltet hierbei sowohl einen smarten Echo-Lautsprecher und einer mit Amazon Alexa kompatiblen Kamera. Je nach dem für welche Lautsprecher sich der Kunde entscheidet bezahlt er für den billigsten Echo-Dot 60€ und für den teuersten Lautsprecher bis zu 150€. Die dazugehörige Kamera bewegt sich zwischen 170-190€. Insgesamt liegt der Preis somit bei minimal 230€.

Bei der selbst kreierten Smart Home Variante belaufen sich die Kosten des Raspberry Pis samt Netzteil und SD-Karte auf rund 50€. Viel billiger ist jedoch die einfache Kamera, die nur rund ein Zehntel der Amazon-Kamera kostet. Hier liegt die eigentliche Preisersparnis, da dadurch die Gesamtkosten sich nur auf 60-70€ belaufen.

In diesem Preis noch nicht mit ein berechnet sind die Kosten für das Mikrofon, welches am Raspberry Pi benötigt wird. Eine der Stärken von den Echo-Lautsprechern ist, dass man von überall im Raum seinen Sprachbefehl äußern kann und dieser verarbeitet werden kann, was die wenigsten herkömmlichen Mikrofone können. Dafür werden mehrere zu einem Array zusammengeschaltene Mikrofone benötigt, die auch in den Amazon-Produkten

verbaut sind. Eine speziell für den Raspberry Pi ausgelegte Variante ist von Respeaker erhältlich, welche ein Array von 4 Mikrofonen und damit eine Reichweite von drei Metern besitzt. Die Kosten belaufen sich auf bis zu 30€, aber man benötigt noch zusätzlich einen Lautsprecher. Je nachdem, wie hoch die gewünschte Klangqualität ist, entstehen hier auch noch erhöhte Kosten.

Bei Wahl eines billigen Lautsprecher sind die insgesamten Kosten eines eigenen Smart Home Systems bei rund 100€. Durch die billige Kamera wird die größte Ersparnis erzielt. Möchte der Benutzer das System auch für klangvolle Musik verwenden, benötigt er einen besseren Lautsprecher. Dabei ist es auch eine Überlegung wert, nicht direkt auch einen Echo-Lautsprecher zurück zu greifen, der die Kosten für Lautsprecher und Mikrofon-Array in sich vereint.

Zusammenfassend lässt sich sagen, dass die Raspberry Pi Lösung auf jeden Fall eine günstigere Lösung darstellt, aber sich auch erst richtig lohnt, wenn mehrere Geräte angesteuert werden, denn die eigentliche Ersparnis nur in der externen Hardware liegt.

Neben der möglichen Ersparnis durch billigere Hardware-Alternativen hat der do-it-yourself-Ansatz weitere Vorteile, wie eine erhöhte Individualisierbarkeit sowie eine Unabhängigkeit. Denn mit den eigenen Geräten in der personalisierten Installation hat der Benutzer weit aus mehr Möglichkeiten sein Eigenheim zu automatisieren. Er kann beliebig viele neue Funktionen implementieren und ist dabei unabhängig von teuren Amazon Produkten und deren Richtlinien.

Diese Punkte sind auch die Gründe, weshalb diese Arbeit durchgeführt wurde. Weiterhin sollen die implementierten Skills auch im Nachhinein weiter im Alltag benutzt werden und in Zukunft auch noch durch weitere Geräte und neue Funktionen erweitert werden, wodurch sich eine derartige billigere Installation anbietet.

4.1.2. Funktionale Anforderungen

Nach Beendigung des Projekts soll die Möglichkeit bestehen, Geräte per Sprachbefehle ein und aus zu schalten. Einige der Geräte sollen über Funksteckdosen, als Steuerelement, an das Smart Home angebunden werden. Hierfür stehen beliebig viele Geräte zur Verfügung, wie beispielsweise Lampen, LED-Streifen, Ventilatoren und weitere Stromverbraucher.

Zudem soll eine Kamera, angeschlossen an einen Raspberry Pi, über Sprachbefehle bedient werden können. Bei dem Befehl zum Starten der Kamera, soll diese anfangen aufzunehmen, und bei dem entsprechenden Befehl zum Stoppen, die Aufnahme beenden und auf dem Raspberry Pi abspeichern. Zusätzlich soll auch noch ein Livestream der Kamera ermöglicht werden.

Die Funksteckdosen, die Kamera und eventuell weitere Geräte sollen neben Sprachbefehlen auch über eine webbasierte Hauszentralsteuerung erreichbar sein. Diese Hauszentralsteuerung muss von beliebigen, webbrowserfähigen Geräten, mit Anbindung ans Heimnetz, ansteuerbar sein. Die webbasierte Hauszentralsteuerung soll zudem über ein passendes Untermenü die zur Verfügung stehenden Funktionen aufzeigen und eine Übersicht über mögliche Sprachbefehle bieten. Über die webbasierte Hauszentralsteuerung sollen die selben Funktionen vorliegen, welche auch über Sprachbefehle zur Verfügung stehen.

Zur einfacheren Ansteuerung, erleichterter Bedienbarkeit und Komfort sollen personalisierte Befehle und Befehlsprotokolle erstellt werden können. Idealerweise im Webinterface der Hauszentralsteuerung. Zu diesen Befehlen und Protokollen zählen Befehle zur parallelen Ansteuerung von verschiedenen Geräten, Zeitschaltuhren sowie Protokolle für Energiesparmaßnahmen und Sicherheit.

4.1.3. Nicht-Funktionale Anforderungen

Im Folgenden werden die qualitativen Anforderungen bzw. die Qualitätsattribute des Projektes genauer definiert.

Systemumgebung

Der Zugang zum System erfolgt über ein internes lokales Heimnetzwerk, über Webbrowser und Sprachbefehle. Funktionsrealisierung erfolgt mittels Node.js Server auf Raspberry Pi, Alexa Voice Service unterstützendem Gerät, wie zum Beispiel Amazon Echo Dot, und mit an den Raspberry Pi verbundenen Geräten.

Das Aufrufen der webbasierten Heimzentralsteuerung ist betriebssystemunabhängig und kann von beliebigen Geräten mit JavaScript fähigem Webbrowser erfolgen, wie bspw. Smartphone, Tablet oder Heimcomputer.

Performance

Das System soll direkt auf Sprachbefehle und User-Inputs mittels Webinterface reagieren. Eine maximale Verzögerung von zwei Sekunden, bis der Benutzer ein auditives oder visuelles Feedback bekommt, wird angestrebt.

Bedienbarkeit

Das System soll einfach und intuitiv bedienbar sein und daher möglichst genau den Anforderungen (Siehe 3.8 Gebrauchstauglichkeit, ISO 9241-110:2006) gerecht werden.

Das Modul soll den Benutzer unterstützen, seine Arbeit zu machen und darf dabei keine unnötigen Informationen anzeigen oder dem Benutzer überflüssige Arbeitsschritte auferlegen. Es soll dem Benutzer in jedem Moment klar sein, was er tun kann und was er zu tun hat, um sein Ziel zu erreichen. Informationen wie Rückmeldungen und Beschriftungen von Bedienelementen sowie Funktionen sollen auf den Benutzer natürlich wirken und ihm allgemein bekannt vorkommen. Hinweise sollen dem Benutzer helfen die Bedienoberfläche der webbasierten Heimzentralsteuerung und der Sprachsteuerung kennenzulernen und ihre Funktionen zu verstehen. Bei falscher Bedienung sollen Fehler vom System erkannt werden, dem Benutzer klar gemacht werden, was schief gelaufen ist und ihm somit helfen, es beim nächsten Versuch besser zu machen. Der Benutzer muss immer die Kontrolle über das System haben, um sein Smart Home zu bedienen und ungewünschtes Verhalten abwenden können. Zudem soll der Benutzer sein Smart Home individualisieren können. Daher muss die Möglichkeit zur individuellen Benennung von Befehlen und somit auch Geräten geschaffen werden.

Zuverlässigkeit

Das System, inklusive Sprach- und Webservice, muss allzeit verfügbar sein und nach Unterbrechung, wie zum Beispiel durch einen Stromausfall schnell und möglichst autonom anlaufen. Für Aktionen wie Sprachbefehle oder weitere, durch den Benutzer durchgeführte, Aktionen besteht eine angestrebte Fehlertoleranz von unter zehn Prozent, damit das System allzeit zuverlässig seine Arbeit verrichtet.

4.2. Herangehensweise

Die Bearbeitung der Studienarbeit wurde in verschiedene Phasen unterteilt, welche sich an einem agilen Entwicklungsprozess orientieren. Zunächst erfolgt die Projektdefinition bestehend aus Aufstellung der Aufgabenstellung sowie der Anforderungsanalyse.

Im zweiten Schritt folgt dann die Grundlagenrecherche und Auseinandersetzung mit für das Projekt zur Verfügung stehenden Hard- und Software. Nach Auswahl der Werkzeuge wird dann die Architektur und das Design ausgearbeitet.

Nach den theoretischen Phasen der Planung und Recherche findet dann die Phase der Einrichtung statt. Hierunter fällt die Beschaffung des Raspberry Pi sowie der rest-

lichen Komponenten. Diese werden dann angeschlossen, miteinander verbunden und die ausgewählten Entwicklungsumgebungen installiert. Die genauere Beschreibung der einzelnen Schritte sind unter 5.1 Einrichtung dokumentiert und wurden dann vor der Implementierung durch eine zweite Installation verifiziert.

Im Anschluss der qualitätssichernden Maßnahme der Verifikation der Einrichtung folgt die Phase der eigentlichen Entwicklung. Das umfasst vor allem die Implementierung und das Testen eines eigenen Alexa-Skills, um Funksteckdosen, welche über den Raspberry Pi ein- und ausgeschaltet werden, per Sprache zu steuern. Dafür wird ein Node.js-Skript zur Ansteuerung der Funksteckdosen, ein auf dem Raspberry Pi laufender Webserver und ein AWS Lambda Server implementiert werden. Näheres dazu befindet sich unter 5.1.7 Alexa sowie 5.1.8 Amazon Web Services (AWS) Lambda. Durch die anschließende Dokumentation der Durchführung wird das erste Teilziel der Ansteuerung der Funksteckdose per Sprachsteuerung erreicht.

In der nächsten Phase des Projektes erfolgt dann die Ansteuerung einer der Kamera per Sprachbefehl analog zur Funksteckdose. Auch hier wird wieder ein separater Skill erstellt, der Befehl durch die Verbindung der einzelnen Server an den Raspberry Pi gesendet und dort durch ein weiteres Skript die Kamera gesteuert. Dies alles wird dann umfangreich getestet und dokumentiert.

Im letzten Schritt wird dann das Projekt abgeschlossen und die Funktionalitäten anhand der vorher aufgestellten Anforderungen ausgewertet. Weiterhin wird ein Ausblick erstellt und das Potential für die Zukunft bewertet.

4.3. Architektur

Das System ist unterteilt in einer Art von Client-Server-Architektur. Der Client wird hierbei von dem Alexa Voice Service und der Heimzentralsteuerung im Webbrowser repräsentiert, da diese in erster Linie dazu dienen, Kommandos vom Benutzer entgegen zu nehmen und zudem Informationen über das System bereit zu stellen.

Die Server-Seite wird vom Raspberry Pi vertreten und dem darauf laufenden Node.js Laufzeitumgebung mit Express.js Webserver. Über den Raspberry Pi werden zudem die Smart Home Geräte angesteuert und deren Logik hinterlegt.

Die Verbindung von Server und Client erfolgt über HTTP- und TCP-Protokoll.

4.4. Design

Auf Abbildung 4.1 wird die Trennung und der Zusammenhang zwischen Raspberry Pi (Server) und Amazon Echo Dot (Client) [ECHO] vereinfacht dargestellt. Diese Darstellung entspricht einem frühen Entwurf und somit auch einer vereinfachten Darstellung der Architektur dieses Projekts.



Abb. 4.1.: Raspberry Pi-Alexa-Kommunikation Vereinfacht

Auf dem Raspberry Pi [RASPa] befindet sich die Node.js [NODEa] Laufzeitumgebung mit Express.js als Webserver. Der Express.js Webserver kommuniziert über HTTP mit dem Alexa Voice Service und einem neuen Alexa-Skill [ALEXA_b], welcher speziell für die Anbindung von Raspberry Pi geschaffen wird.

Nach weiterer Einarbeitung in die Thematik und testweiser Implementierung stellt sich heraus, dass eine solch einfache Architektur wie in Abbildung 4.1 sich nicht ohne weitere Komponenten implementieren lässt und eine komplexere Architektur mit erweitertem Konzept erforderlich ist, wie in Abbildung 4.2 zu sehen ist:

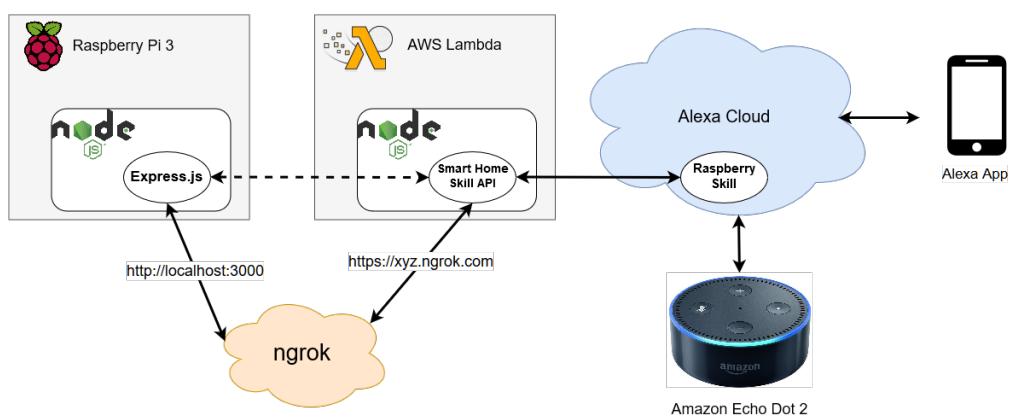


Abb. 4.2.: Raspberry Pi-Alexa-Kommunikation

Um einen Smart Home-Skill (Siehe Abschnitt 4.4.1) mit Amazons Smart-Home-Skill API zu erstellen, wird zwangsläufig ein AWS Lambda Server benötigt. Im Vergleich zu

Abbildung 4.1 wurde daher in Abbildung 4.2 AWS Lambda als Schnittstelle zwischen Raspberry Pi und der Alexa Cloud eingefügt. AWS Lambda wird für diese Studienarbeit auf die Node.js Laufzeitumgebung eingestellt, was die Entwicklung mit JavaScript, sowohl auf AWS Lambda als auch auf dem Raspberry Pi, ermöglicht. Die in Abbildung 4.2 dargestellte, unter AWS Lambda angedeutete, Smart Home-Skill API repräsentiert in diesem Fall die selbst geschriebene Schnittstelle in der Lambda-Funktion. Diese kann Direktiven von der Alexa Cloud entgegennehmen und verarbeiten (Siehe 5.2.1).

Der Webserver des Raspberry Pi ist zunächst einmal nur im selben Netzwerk über seine lokale IP-Adresse im Heimnetz oder Intranet erreichbar. Hier muss also Abhilfe geschaffen werden, um auch über AWS Lambda Zugriff auf den Raspberry Pi zu bekommen. Der Webserver muss dem Rest des Internets und somit auch AWS Lambda zugänglich gemacht werden. Der standardmäßige Ansatz für dieses Einsatzszenario wäre, auf dem Router zu Hause eine Portweiterleitung einzurichten, welche auf den Raspberry Pi verweist. Dies wäre aber recht umständlich und zugleich ein potentielles Sicherheitsrisiko. Mit dieser Portweiterleitung und unverschlüsselter öffentlicher Verbindung zum Internet, wäre das Heimnetz ein leichtes Ziel für Hackerangriffe mit unerlaubtem Eindringen in das Heimnetzwerk. Um die Sicherheit zu erhöhen und eine HTTPS-Verbindung für den Raspberry Pi zu schaffen, müsste man sich zudem extra ein Zertifikat ausstellen lassen. Dies wäre wiederum ziemlich zeit- und kostenintensiv. Die Lösung für dieses Problem bietet hier ngrok, wie man in der Abbildung 4.2 sehen kann. Mittels ngrok wird eine gesicherte Verbindung zwischen dem Raspberry Pi und den Servern von ngrok hergestellt. Ngrok weist dem Raspberry Pi nach Verbindungsaufbau dann eine neue Adresse zu, welche zudem gleich als sichere verschlüsselte Verbindung mit HTTPS daher kommt. Jede Kommunikation mit der ngrok Adresse wird somit direkt zum Raspberry Pi getunnelt.

Der in dieser Arbeit zu erstellende Raspberry Skill (Siehe Abschnitt 5.1.7) wird, nach der Einrichtung, online im Skill-Store von Alexa zugänglich gemacht. Bevor der Skill aber nicht aktiv veröffentlicht wird, erscheint er nur denen Personen, welchen über das Amazon Developer Konto, auch der Zugriff auf diesen Skill gewährt wurde. Der Skill muss über ein Smartphone, mit vorhandener Alexa App, installiert und dann mit einem Amazon Konto verbunden werden. Für diese Verbindung muss der Benutzer bei der Einrichtung eine Accountverknüpfung, gemäß OAuth2-Standard, durchführen. Die Accountverknüpfung ist bei Smart Home-Skills zwingend notwendig, da hierdurch der Skill Rechte auf Daten des Benutzers anfragt, welche der Benutzer aktiv in einer Web-View erteilen muss.

Nach der Verbindung mit dem Amazon Account befindet sich der Skill in dem personalisierten Bereich der Alexa Cloud des Nutzers. Der Skill ist nun über jedes Gerät des

Benutzers zugänglich. Dies können mehrere Echo Geräte, das Smartphone mit Alexa App oder auch weitere Geräte mit Zugriff auf Alexa Services sein. Im Rahmen dieser Arbeit wurde der Skill aber nur, wie in Abbildung 4.2 zu sehen ist, mit einem Amazon Echo Dot der zweiten Generation und der Alexa App, auf einem Smartphone, bedient.

4.4.1. Skill-Typen

Bei der Entwicklung eines eigenen Alexa Skills ist der erste Schritt die Auswahl des gewünschten Skill-Typ. Diese werden im Folgenden vorgestellt. Ein Entwickler, welcher einen eigenen Skill erstellen möchte, wird in diesem Abschnitt als Benutzer angesehen, welcher die Entwicklungswerkzeuge von Amazon nutzen möchte.

Custom Skills Mit einem Custom Skill kann der Benutzer ein beliebigen, auch personalisierten Skill erstellen, um spezielle Anwendungsfelder von Alexa abzudecken und die Benutzerfreundlichkeit zu erhöhen. Dieser Typ ist sowohl der flexibelste, als auch der Skill mit dem größten Konfigurationsaufwand, denn der User muss das Schema der Anfragen- und Antworten-Verarbeitung selbst konfigurieren, schreiben und dafür auch die entsprechenden Daten bereitstellen, damit die Anfrage aufgerufen werden kann.

Zunächst muss der Entwickler ein Intent (dt. Absicht) für den neuen Skill festlegen. Das kann jede erdenkliche Aktion, wie eine einfache Informationssuche, Online-Essensbestellung oder ein Spiel, sein.

Im zweiten Schritt wird dann das Interaction Model definiert, welches ermöglicht, dass der Benutzer über das Voice User Interface mit dem Skill kommunizieren kann. Hier wird festgelegt, welche Wörter gesagt werden müssen, um den personalisierten Skill zu aktivieren. Somit würde zum Beispiel der Befehl „Bestelle eine große Salami Pizza“ einen speziellen Online-Essensbestellungs-Intent ansprechen.

Zuletzt wird noch ein Name für den Skill (Invocation Name) benötigt, damit Alexa den Befehl auch richtig zuordnen kann. Das bereits angesprochene Beispiel könnte demnach folgendermaßen aussehen: „Alexa, bestelle eine große Salami Pizza von Pizza-Restaurant.“ Hier wäre „Pizza-Restaurant“ der Invocation-Name, für den favorisierten Pizza-Lieferdienst.

Weiterhin können visuelle oder Touch-Interaktionen (z. B. für den Echo Show) hinzugefügt werden, um durch diese Kombination ein optimales Benutzererlebnis zu kreieren. [AMA18f]

Smart Home Skills Dieser Skill-Typ ist, wie der Name schon verrät, speziell für die Steuerung von Smart Home Geräten geeignet. Demzufolge hat der Benutzer nicht dieselbe

vielfältige Entscheidungsfreiheit, wie bei den Custom Skills, was aber auch die Entwicklung enorm vereinfacht.

Die Definition eines im letzten Abschnitt erklärten Voice User Interface wird dem Benutzer bei diesem Skill-Typ durch die Smart Home Skill API abgenommen. Es muss somit kein eigenes Interaction Model oder ein Invocation Name erstellt werden. Ein einfacher Befehl, wie „Alexa, schalte das Licht im Wohnzimmer an.“ würde hier ausreichen und kein spezifischer Skill muss angesprochen werden.

Die Smart Home Skill API beinhaltet die Anfragen, die bearbeitet werden können, welche hier als directives (dt. Handlungsanweisungen) bezeichnet werden. Beispiel für derartige Direktiven sind an- und ausschalten, Temperatur erhöhen, Tür schließen oder TV-Kanal ändern. Die Aktivierungswörter sind ebenfalls schon vorherbestimmt und der Benutzer muss selbst nur implementieren, was zum Beispiel geschehen soll, wenn die Direktive „Schalte das Licht an.“ empfangen wird. Der Code, welcher zur Verarbeitung der von Amazon gesendeten Direktiven erforderlich ist, muss in einer AWS Lambda Funktion bereitgestellt werden (Siehe 5.1.8 Amazon Web Services (AWS) Lambda).

Zu beachten ist, dass die API nur auf von ihr unterstützte Direktiven reagieren kann. [AMA18f]

Durch die Smart Home Skill API ist es möglich, im System über die Alexa App nach Geräten zu suchen und die Skills auch ohne Spracheingabe zu testen. Aus diesem Grund werden in dieser Studienarbeit auch Smart Home Skills anstatt Custom Skills verwendet. Zudem spielte bei der Entscheidung auch mit hinein, dass bei diesem Skill-Typ nicht jedes mal der genaue Invocation Name angesprochen werden muss, was auch auf Dauer sehr nervig und benutzerunfreundlich ist. Die Geräte, welche über die Smart Home-Skill API angesteuert werden, lassen sich zudem nahtlos in ein bestehendes Smart Home mit weiteren Geräten, auch von vielen unterschiedlichen Herstellern, integrieren. Nach der Aktivierung des Skills unterscheidet Alexa in der App und bei der Sprachsteuerung nicht zwischen den Geräten unterschiedlicher Hersteller und behandelt diese auf die selbe intelligente Art und Weise. Über die Alexa App kann der Benutzer zudem seine Geräte direkt unterschiedlichen Räumen oder Gerätengruppen zuordnen, was die Benutzerfreundlichkeit durch zusätzliche Logik erhöht. Dies ermöglicht dem Benutzer Aussagen wie „Alexa, schalte alle Lichter im Schlafzimmer an“ oder „Alexa, schalte die Geräte im Wohnzimmer aus“.

Flash Briefing Skills Mit diesem Skill-Typ kann der Benutzer Alexa bitten, die aktuellen Nachrichten vorzutragen. Hierbei gibt es auch wieder eine Flash Briefing Skill API, welche über Befehle wie „Alexa, gib mir eine Tageszusammenfassung.“ angesprochen werden

kann.

Der Benutzer muss lediglich einen Namen, eine Beschreibung sowie ein Bild für den Skill bestimmen, damit dieser im Skill Store wieder gefunden werden kann. Zuletzt müssen dann nur noch die Quellen für die Nachrichteninhalte angegeben werden, aus denen Alexa dann ihre Zusammenfassungen vorliest. [AMA18f]

Video Skills Der Video Skill ermöglicht es Videoinhalte aus Fernsehsendungen und Filmen über die Video Skill API bereitzustellen. Auch hier sind die Aktivierungsphrasen schon vorherbestimmt. Die Arbeit des Benutzers beschränkt sich auch wieder auf die Erstellung des Erscheinungsbildes im Skill Store und der Definition der Sucheinstellungen sowie auf welchen Geräten das Video (Smart TV, Echo Show) wiedergegeben werden kann. [AMA18f]

List Skills Ein List Skill ermöglicht das Benutzen von sogenannten Event-Listen. Der Anwender kann somit seine eigene Listen, wie Todo-Listen, managen, kann aber auch dritten Anwendungen oder Skills erlauben Listeneinträge vorzunehmen. Der Skill realisiert Änderungen in solch einer Liste und kann dann darauf reagieren. Die Erstellung dieses Typs erfolgt durch das Alexa Skills Kit Command Line Interface, welches dem Benutzer die größte Arbeit abnimmt.

4.4.2. Ansteuerung der Funksteckdosen

In dieser Studienarbeit wurden Funksteckdosen der Marke Brennenstuhl verwendet. In der folgenden Abbildung 4.3 ist sowohl die Fernbedienung (links) als auch eine Steckdose (rechts) mit Einstellung auf den, an der Fernbedienung eingestellten, Hausschlüssel abgebildet, welcher bereits in den Grundlagen unter 3.5 Funkstandards angesprochen wurde.



Abb. 4.3.: Verwendetes Funkset mit Hausschlüssel

An der Fernbedienung muss zunächst der gewünschte Hausschlüssel eingestellt werden, sodass nur die Steckdosen mit der dazugehörigen Einstellung auf die Signale reagieren. Der hier verwendete Hausschlüssel, welcher durch die 5-Dip-Schalter links im Bild 4.3 realisiert wird, lautet somit 01110.

Dieselbe Bitbelegung befindet sich auch am Anfang der Steckdoseneinstellung. Die restlichen fünf Bit beschreiben, auf welche Fernbedienungsbelegung reagiert werden soll. Wenn das erste Bit der restlichen fünf auf eins (d. h. oben) steht, kann die Steckdose durch Bedienung der Tasten unter A an- und ausgeschaltet werden. Bei dem hier dargestellten Beispiel reagiert die Steckdose auf die Belegung D der Fernbedienung.

Die Ansteuerung der Funksteckdosen soll in diesem Projekt programmatisch über das an den Raspberry Pi angeschlossene Funkmodul erfolgen. Die an die Programmiersprache gestellte Anforderung ist somit, dass sie die Möglichkeiten besitzt, die GPIO-Pins des Raspberry Pis auszulesen bzw. auf diese zu schreiben. Da bereits ein Webserver per Node.js zur Bearbeitung der Befehle aufgesetzt ist, sollen weitere Skriptaufrufe zu anderssprachigen Anwendungen soweit wie möglich vermieden werden. Da der Node Package Manager

durch das „onoff“-Paket das Schreiben und Lesen der GPIO-Pins ermöglicht [NODEd], wird in der Studienarbeit die Ansteuerung der Funksteckdosen ein Node.js-Skript verwendet, welches im Kapitel der Implementierung unter 5.2.2 Funksteckdosen-Ansteuerung genauer erläutert wird.

Das Skript muss dann in dem zu sendenden Signal sowohl den Hausschlüssel als auch das gewählte Gerät (A-E) sowie die Information, ob ein oder aus geschaltet wird, codieren. Der Funkempfänger in der Steckdose erwartet dabei ein Signal, das insgesamt 16 Byte lang ist. In den ersten fünf Byte wird entsprechend der fünfstelligen Dip-Schalterbelegung des Hausschlüssels der jeweilige Systemcode übergeben. In den fünf darauf folgenden Bytes wird dann die Geräte-ID, also A bis E, codiert und danach kommen zwei Bytes zur Unterscheidung, ob das Gerät an oder abgeschaltet wird. Die letzten 4 Bytes dienen zur Synchronisation des Signals und sind somit auch immer gleich. Dieser Code muss vom Node.js-Skript anhand der jeweiligen Parameter des Systems und der Geräte dann aufgestellt werden und in ein 128-bit-Signal umgesetzt werden, welches dann über das Funkmodul gesendet wird. Der genau Aufbau und Erklärung dieses Skripts befindet sich unter 5.2.2 Funksteckdosen-Ansteuerung.

4.4.3. Ansteuerung der Kamera

Im Bild 3.2 im Grundlagenkapitel ist die für das Projekt verwendete Kamera für den Raspberry Pi zu sehen. Das gewählte Modell besitzt 5 Megapixel und kann durch das angebrachte Flachbandkabel mit dem Raspberry Pi 3 und allen anderen Versionen des Einplatinencomputer einfach verbunden werden. Dies erfolgt durch das in den Grundlagen unter 3.4 Raspberry Pi erwähnte Camera Interface.

Um, ebenso wie bei den Funksteckdosen, weitere Skriptaufrufe zu anderen Sprachen zu vermeiden, wird auch bei der Ansteuerung der Kamera ein Node.js-Skript verwendet. Das verwendete NPM-Paket, sowie die programmatischen Details befinden sich unter 5.2.2 Kamera-Ansteuerung.

Die Aufnahmen der Kamera werden zunächst im .h264-Format gespeichert und müssen dann dementsprechend noch in das Containerformat .mp4 konvertiert werden. Die Umwandlung von reinem Video-Codec in einen Container erfolgt in dieser Studienarbeit mit FFmpeg, was ebenfalls im Abschnitt 5.2.2 Kamera-Ansteuerung erläutert wird.

5. Implementierung

In diesem Kapitel werden die einzelnen Entwicklungsschritte der in der Architektur und Design vorgestellten Elemente genauer, aufgeschlüsselt dargestellt.

5.1. Einrichtung

Im Folgendem wird die Einrichtung der gewählten Systeme, Plattformen und Server erläutert. Die Reihenfolge der Unterkapitel spiegelt dabei auch die Reihenfolge der einzelnen Einrichtungsschritte wieder.

5.1.1. Raspberry Pi

Zunächst muss der Raspberry Pi eingerichtet werden. [W3SCHa] Die Entscheidung für die Geräteauswahl fiel auf den Raspberry Pi 3 Model B. Das Projekt ist aber auch auf alle anderen Raspberry Pi Modellen portierbar.

Für eine einfache Bedienung des Raspberry Pi kommt in diesem Projekt das „Raspbian Stretch with Desktop“ [RASPb] zum Einsatz. Solange die Ansteuerung der GPIO Ports gewährleistet ist, kann auch ein beliebiges anderes Betriebssystem auf dem Raspberry Pi eingerichtet werden.

Um ein Betriebssystem für den Raspberry Pi zu installieren, muss das Betriebssystem auf der Micro-SD-Karte des Raspberry Pi installiert („geflasht“) werden. Am einfachsten geht dies mit dem kostenlosen Programm Etcher. [ETC17] Etcher ist für Windows, Linux und MacOS erhältlich. In Etcher muss lediglich das Medium (die Micro-SD-Karte) sowie ein Image zur Installation ausgewählt und die Installation gestartet werden. Sobald Etcher das Betriebssystem auf dem Raspberry Pi installiert hat, kann der Datenträger in den Raspberry Pi gesteckt werden und von dort direkt gebootet werden.

Der Raspberry Pi kann entweder über direkt angeschlossene Peripheriegeräte oder per Remote bedient werden. Für die Bedienung über eine SSH empfiehlt sich PuTTY. [PuTTY] In unserem Projekt wurde außerdem RealVNC [RVNC] verwendet, welches auf dem

Raspbian Stretch bereits vorinstalliert ist. RealVNC hat gegenüber SSH den Vorteil, dass der gesamte Desktop über Remote bedient werden kann.

5.1.2. Cloud 9

Für die Entwicklung auf dem Raspberry Pi wurde die Web-IDE Cloud 9 [C9.io] verwendet, welche auf dem Raspberry Pi als Server installiert ist. Zur Einrichtung von Cloud 9 wurde die online verfügbare Anleitung von Joshua Lunsford verwendet. [LUN17] Die Web-IDE ermöglicht besonders einfaches Entwickeln und Testen über jeden Webbrowser auf Computern im selben Netzwerk wie der Raspberry Pi.

5.1.3. Git

Zur Sicherung und Versionierung wurde ein GitHub Repository eingerichtet.

```
1 git clone https://github.com/RobinWarth/Smart-Home-Solutions.git
2 git remote add origin https://github.com/RobinWarth/Smart-Home-
   ↘ Solutions.git
```

Quellcode 5.1: GitHub Repository

5.1.4. Node.js

Für das Projekt wird Node.js in Version 9.x verwendet. Um das aktuellste Node.js v9.x zu installieren müssen folgende Befehle ausgeführt werden: [NODEb]

```
1 curl -sL https://deb.nodesource.com/setup_9.x | sudo -E bash -
2 sudo apt-get install -y nodejs
```

Quellcode 5.2: Node Module Laden

Nach dem Download des Git-Repository, müssen zunächst einmal noch die Node Module für das Projekt geladen werden. Diese sind über die package.json beschrieben. Jedes Modul oder auch Paket besitzt seine eigene package.json. Ein Paket, wie dieses Projekt, kann daher auch Module enthalten, welche wiederum selbst auf weitere Module angewiesen sind. Über den Node Package Manager (NPM) werden diese Module und Pakete aus den package.json Dateien rekursiv ausgelesen und geladen. Diese Prozedur muss über folgenden Befehl im Ordner /Code/Node/ gestartet werden:

```
1 npm install
```

Quellcode 5.3: Node.js Installieren

Um den Server zu starten, muss der folgende Befehl mit Super-User Rechten ausgeführt werden:

```
1 [sudo] node app
```

Quellcode 5.4: Node.js Installieren

Die Super-User Rechte werden benötigt um einen Ordner, außerhalb des Projekts, automatisch zu erstellen, in welchem Kameraaufnahmen gespeichert werden.

5.1.5. FFmpeg

Da für die Einrichtung von FFmpeg einige Schritte notwendig sind, wurde die Einrichtung speziell für diese Studienarbeit mithilfe eines zusammenfassenden Bash-Skripts (lauffähig unter Raspberry Pi 2/3 B+) vereinfacht. Das Skript befindet sich im Projekt unter dem Pfad /Installation-Scripts/. Zur Ausführung des Skripts müssen dem Skript zuvor lediglich Schreib-Rechte zugewiesen werden:

```
1 chmod 755 Install-Scripts/ffmpeg_i # Schreibzugriff fr Skript gewhren
2 ./ffmpeg_i # fhrt Skript aus
```

Quellcode 5.5: FFmpeg Installieren und Starten

Da bei der Installation Dateien aus einem Git Repository heruntergeladen und kompiliert werden müssen, nimmt die Installation etwas Zeit in Anspruch. Auf einem Raspberry Pi 3 B+ dauert es etwa 30 Minuten bis FFmpeg mit dem Skript eingerichtet und konfiguriert ist.

Notizen... Ändern und starten von FFmpeg:

```
1 chmod 755 Install-Scripts/reconfigure_ffmpeg # Skript Schreibzugriff
    ↪ gewaehren
2 ./reconfigure_ffmpeg # Skript ausfuehren
```

Quellcode 5.6: FFmpeg Rekonfigurieren und Starten

Beispiel konvertieren von .h264 Video-Codec in .mp4 Container:

```
1 sudo ffmpeg -framerate 24 -i 14Apr2018_10-23-02.h264 -c copy 14
    ↪ Apr2018_10-23-02.mp4
```

Quellcode 5.7: FFmpeg Konvertierung

5.1.6. Amazon Developer Konto

Für das Erstellen von Alexa Skills muss ein Amazon Developer Konto erstellt werden. [AMA18a]

5.1.7. Alexa

Die Nachfolgende Einrichtung eines Alexa Skills bezieht sich auf die bis Anfang 2018 verfügbare Alexa Skills Kit Developer Console, welche nun ein neues Design angenommen hat. Einzelne Bearbeitungsschritte können daher geringfügig abweichen oder eine veränderte Gestalt aufweisen. Zunächst einmal müssen im ersten Schritt unter Skill Information grundlegende Informationen über den Skill festgelegt werden, welche die weiteren Schritte und deren Komplexität beeinflussen.

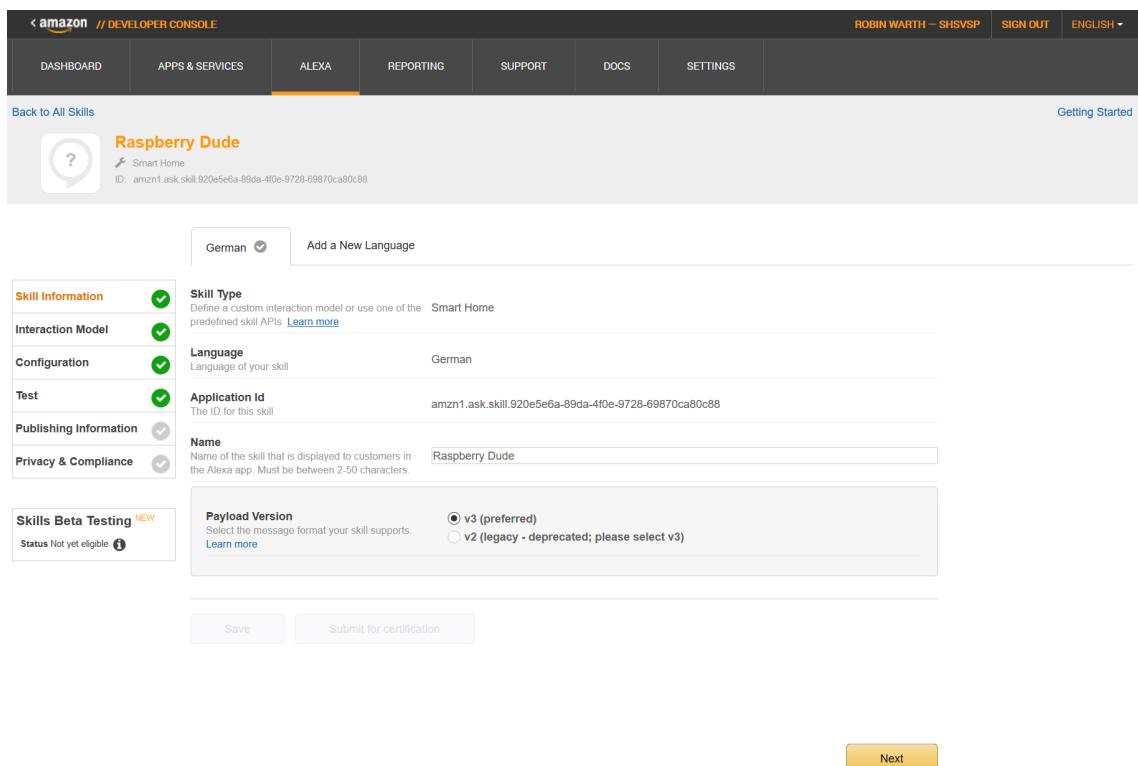


Abb. 5.1.: Setup Skill-Information

Wie schon unter 4.4.1 Skill-Typen beschrieben, fiel die Entscheidung auf einen Skill, welcher die Smart Home Skill API unterstützt. Außerdem müssen in diesem Schritt noch die unterstützten Sprachen des Skills sowie ein Skill Name festgelegt werden.

Die Payload Version bestimmt, in welcher Form und Stil Nachrichten im JSON-Format zwischen Skill, Alexa Smart Home API und Server ausgetauscht werden sollen. Dies wird im weiteren Verlauf noch einmal erneut aufgegriffen, wenn der Aufbau und Stil der JSON-Nachrichten weiter erläutert wird (Siehe 5.1.8 Amazon Web Services (AWS) Lambda). Auch wenn hier noch zusätzlich die Auswahl einer Payload Version aufgeführt ist, so lässt sich doch nur die Payload Version „v3“ auswählen. Payload Version „v2“ ist

zum Zeitpunkt dieser Studienarbeit schon so veraltet, dass sie nur noch für ältere schon erstellte Skills noch aufgeführt wird.

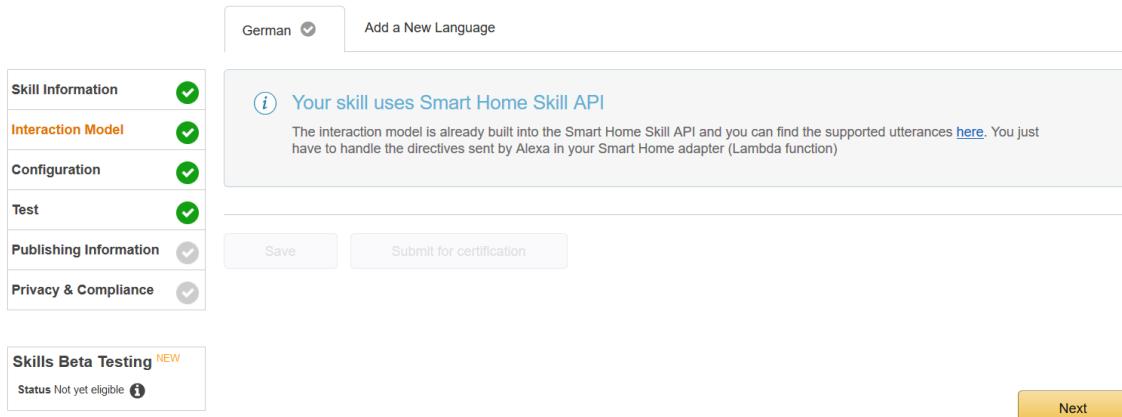


Abb. 5.2.: Setup Interaction Modell

Da in diesem Skill die Smart Home Skill API zum Einsatz kommt, muss in dem Schritt Interaction Model nichts weiter getan werden. Das Interaktionsmodell ist bereits in die API eingebaut.

Im Schritt Configuration wird zunächst der Service Endpunkt festgelegt, welcher die von Alexa gesendeten Direktiven verarbeiten kann. Da hier die Smart Home Skill API zum Einsatz kommt, verpflichtet Amazon die Nutzung eines Amazon Web Services (AWS) Lambda Server. Zu dem Aufbau und der Einrichtung des AWS Lambda Server wird in Abschnitt 5.1.8 Amazon Web Services (AWS) Lambda noch gezielt eingegangen.

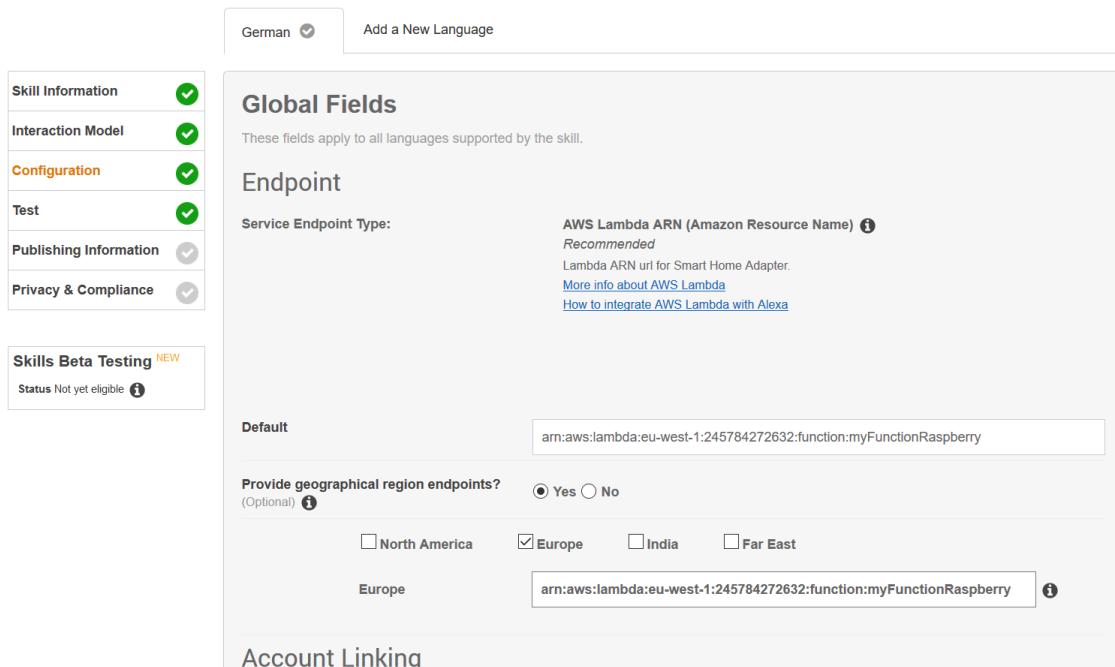


Abb. 5.3.: Setup Configuration Endpoints

Wurde statt der Smart Home Skill API das Custom Interaction Model als Skill Typ gewählt, ist es möglich anstatt dem AWS Lambda Server auch jeden beliebigen anderen zertifizierten HTTPS Webserver anzugeben.

Bei Amazon Web Services wie dem Lambda Server (Siehe 5.1.8 Amazon Web Services (AWS) Lambda) wird jedem Server eine eindeutige Adresse zur Ressource und der darin enthaltenen Funktionen zugeteilt. Diese Adresse nennt sich Amazon Resource Name (ARN). Die ARN des Webservices dieser Studienarbeit lautet:

„arn:aws:lambda:eu-west-1:245784272632:function:myFunctionRaspberry“.

In der ARN sind Informationen über den Server-Typ (hier: Lambda), über den Server Standort (hier: eu-west, Irland) und die Funktion (hier: myFunctionRaspberry) enthalten, welche den Endpunkt repräsentiert und die Direktiven von Alexa entgegennimmt.

Das Account Linking ist für Skills der Smart Home Skill API zwingend notwendig, aber für Skills mit Custom Interaction Model ist dies optional. Das Account Linking erfolgt mit Hilfe des OAuth2 Authentifizierungsstandard. [OAUTHa]

Account Linking

Authorization URL
The url where customers will be redirected in the companion app to enter login credentials.

Client Id
Unique public string used to identify the client requesting for authentication.

Domain List (Optional)
The list of domains that the authorization URL will fetch content from. You can provide up to 30 domains.
[Add domain +](#)

Scope
List of permissions to request from the skill user. You can provide up to 15 scopes.
[Add scope +](#)
1 X

Redirect URLs (Optional)
The list of valid HTTPS redirection endpoints that could be requested during authorization to redirect the user back to after the authorization process.
[Learn more.](#)

Authorization Grant Type (Optional)
Specifies the OAuth authorization grant that Alexa uses to obtain an access token from your provider.
[Learn more.](#)
 Implicit Grant Auth Code Grant

Access Token URI
This URI will be used for both access token and token refresh requests.

Client Secret

Client Authentication Scheme (Optional)

Permissions

Request users to access resources and capabilities
 Send Alexa Events i
Please request permissions to resources and capabilities that are absolutely core to the customer experience delivered by the skill.
[Learn More](#)

Alexa Skill Messaging
Use clientId and clientSecret for skill messaging, enabling out-of-session (non-customer invoked) events to be sent to skill code.
Client ID amzn1.application-oa2-client.bf5d5e600cb941528035489d8d0eabf2
Client Secret ***** [Show](#)

Privacy Policy URL
Link to the Privacy Policy for this skill. This is mandatory for account linking.

Abb. 5.4.: Setup Configuration Account Linking

Das Account Linking ist notwendig, um jeden Benutzer des Skills eindeutig identifizieren zu können. Im Allgemeinen wird hier eine Verknüpfung zwischen Amazon-Alexa-Account und dem Account beim Hersteller der Smart Home Geräte, wie z. B. Lampen, geschaffen.

In diesem Projekt erfolgt das Account Linking mit einem Amazon Account. Die Einstel-

lungen sind hier alle auf Login With Amazon (LWA) getrimmt. [AMA18b] Der Aufwand, um einen guten, sicheren und verlässlichen OAuth2-Server selbst zu erstellen, wäre für diese Studienarbeit zu groß gewesen. Zudem muss der OAuth2-Server auch von Amazon akzeptiert sein und daher wären zusätzliche gültige Zertifikate notwendig. Die Möglichkeiten, welche durch Account Linking geschaffen werden, würden ohnehin nicht ausgereizt werden in dieser Studienarbeit und dem Einsatzgebiet, von dem hier erstellten Alexa Skill. [AMA18c]

Das Account Linking erfolgt gemäß OAuth2-Standard, wie in Abschnitt 3.6 erläutert wurde. Somit stellt das Account Linking und die damit verbundenen Mechanismen zur Authentifizierung den ersten Schritt (Siehe Abbildung 5.5) im Datenfluss zwischen den einzelnen Servern und Komponenten dar.

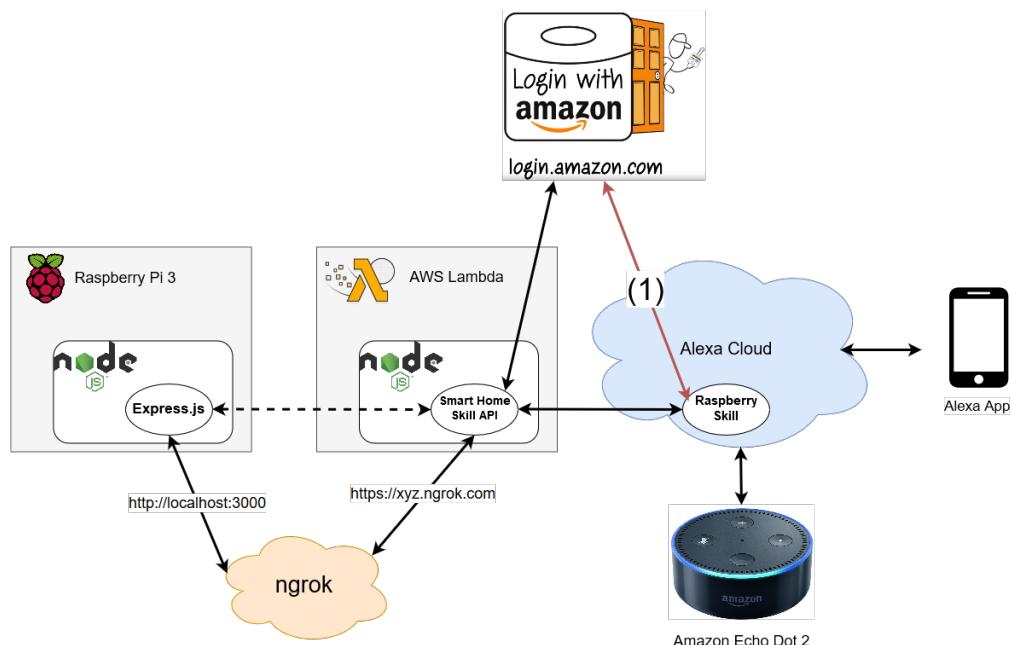


Abb. 5.5.: Raspberry Pi-Alexa-Kommunikation Schritt 1

Im weiteren Verlauf des Abschnitts 5 wird die Grafik in Abbildung 5.5 nun mehrfach aufgegriffen und um weitere Schritte des Datenflusses ergänzt.

Um den Skill mit der Alexa App auf dem Smartphone und einem Amazon Echo Gerät zu testen, muss nur der Schalter in Schritt Test umgelegt werden. Der Skill wird nun für die mit dem Konto verbundenen Geräte automatisch zum Testen eingerichtet. Alternativ kann der Skill auch mittels, dem zur Zeit der Studienarbeit in der Beta-Phase befindlichen, Test-Simulator ausgeführt werden. Der Test-Simulator bietet dieselbe Möglichkeit zum Testen wie ein Echo-Gerät und sogar zusätzliche Funktionen. Im Test-Simulator kann

direkt mit Alexa über Sprache oder geschriebenem Text interagiert werden. Wurde zu Beginn das Custom Interaction Model statt der Smart Home Skill API gewählt, lassen sich im Test Simulator auch die JSON-Direktiven anzeigen, welche von Alexa oder dem Skill-Endpoint gesendet werden.

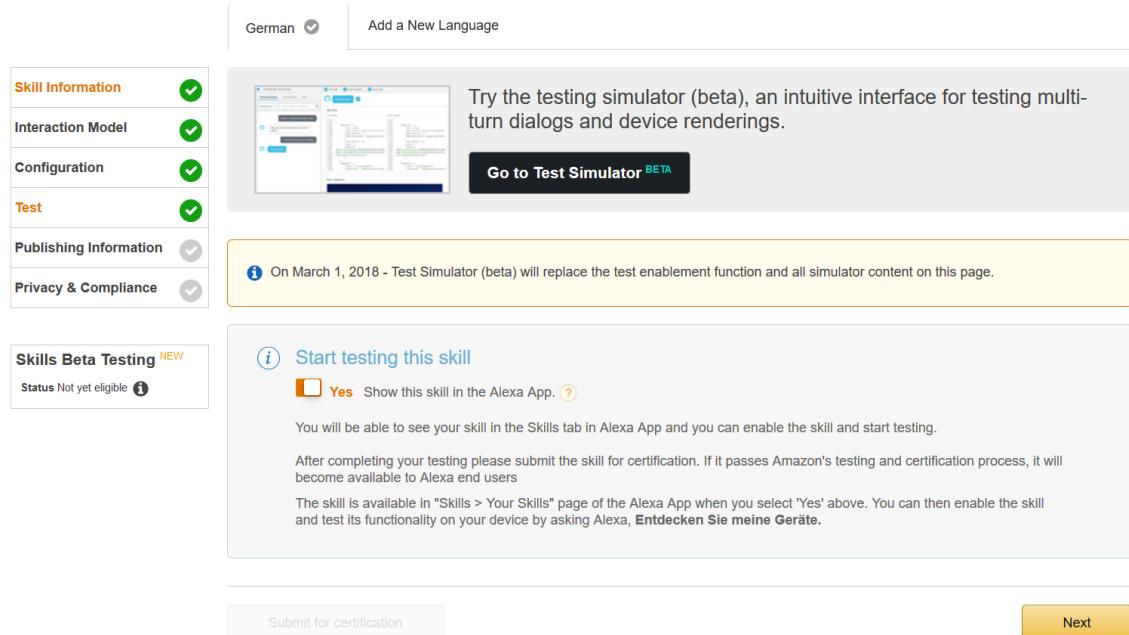


Abb. 5.6.: Setup Test

5.1.8. Amazon Web Services (AWS) Lambda

Um einen Alexa Skill mit der Smart Home Skill API zu erstellen, wird ein AWS Lambda Server vorausgesetzt. Hierfür muss ein Konto erstellt und ein Server gemietet werden.

Bevor der Lambda Server eingerichtet wird, muss ein Alexa Skill bereits erstellt sein. Bei der Einrichtung des AWS Lambda Server für den Skill wird dessen ID benötigt. Ebenfalls ist ein AWS Account zwingend notwendig.

Nach der Anmeldung bei AWS wird zu Lambda navigiert. Für den Lambda Server ist es wichtig, die richtige Region auszuwählen. Mögliche Regionen für Alexa Skills sind Asia Pacific (Tokyo), EU (Ireland), US East (N. Virginia) und US West (Oregon). [AMA18d]

Als nächstes muss eine Lambda Function werden, um in dieser die von Alexa gesendeten Direktiven zu entgegen zu nehmen und zu verarbeiten.

Bei der Erstellung einer Lambda Function kann man aus verschiedenen Blaupausen oder eine leere Funktion, ohne Codebeispiel, wählen. Für den Einsatz der Lambda Function in dieser Studienarbeit hat zur Zeit der Bearbeitung leider keine passende Blueprint existiert. Diese waren veraltet und haben eine alte Payload-Version der Alexa-Direktiven unterstützt. Dank Codebeispielen in der Online Dokumentation von Amazon Alexa kann die Lambda Funktion dennoch zügig einsatzbereit gemacht werden. Der Name der hier erstellten Lambda Funktion lautet „myFunctionRaspberry“.

Danach muss eine Rolle gewählt werden, da zu diesem Zeitpunkt noch keine eigene definierte Rolle existiert. Für dieses Projekt wurde eine Rolle Namens „myRoleRaspberry“ mithilfe des Templates „Basic Edge Lambda permissions“ erstellt.

Durch das Betätigen des „Create Function“ Buttons wird die Funktion von AWS eingerichtet. Um auf Ereignisse zu reagieren, werden Trigger zur Funktion hinzugefügt. Standardmäßig wurde ein Trigger „Amazon CloudWatch Logs“ bereits der Funktion hinzugefügt. Dieser ist für das Logging zuständig. Jede Log Ausgabe wie z. B. „console.log('test');“ landet somit automatisch in den Amazon CloudWatch Logs mit Datum und eindeutigen Identifikatoren. Um die Lambda Funktion für die Alexa Smart Home Skill API bereit zu machen, wird der Alexa Smart Home Trigger ausgewählt. Dieser Trigger muss nun nur noch mithilfe der Skill-ID bereit gemacht werden.

5.1.9. ngrok

Um einen Tunnel mittels ngrok aufbauen zu können muss ngrok zunächst einmal, über die Homepage [NGROK], heruntergeladen und entpackt werden. Für einen ngrok Tunnel wird außerdem ein kostenloser ngrok Account benötigt. Ist der Account erstellt, wird von ngrok ein Authentifizierungstoken ausgestellt, welches zur Einrichtung auf dem Gerät benötigt wird, um sich zu authentifizieren:

- 1 Download ngrok for Linux(ARM) . . .
- 2 unzip ngrok-stable-linux-arm.zip -d [destination]
- 3 ./ngrok authtoken [token received after ngrok-Account creation]

Quellcode 5.8: Ngrok Einrichtung

Um einen Tunnel mit Basic-Authentifizierung, auf einen bestimmten Port (hier: 3000) zu starten, wird der folgende Befehl ausgeführt:

- 1 ./ngrok http -auth="Benutzer:Passwort" 3000

Quellcode 5.9: Ngrok Tunnel Aufbau

5.2. Entwicklung

In diesem Abschnitt wird auf die Entwicklung der einzelnen Komponenten eingegangen und deren Kommunikation näher beschrieben. Auf die in folgender Grafik (Siehe Abbildung: 5.7) gezeigten Verarbeitungsschritte wird in den nachfolgenden Themenbereichen zu den Komponenten näher eingegangen.

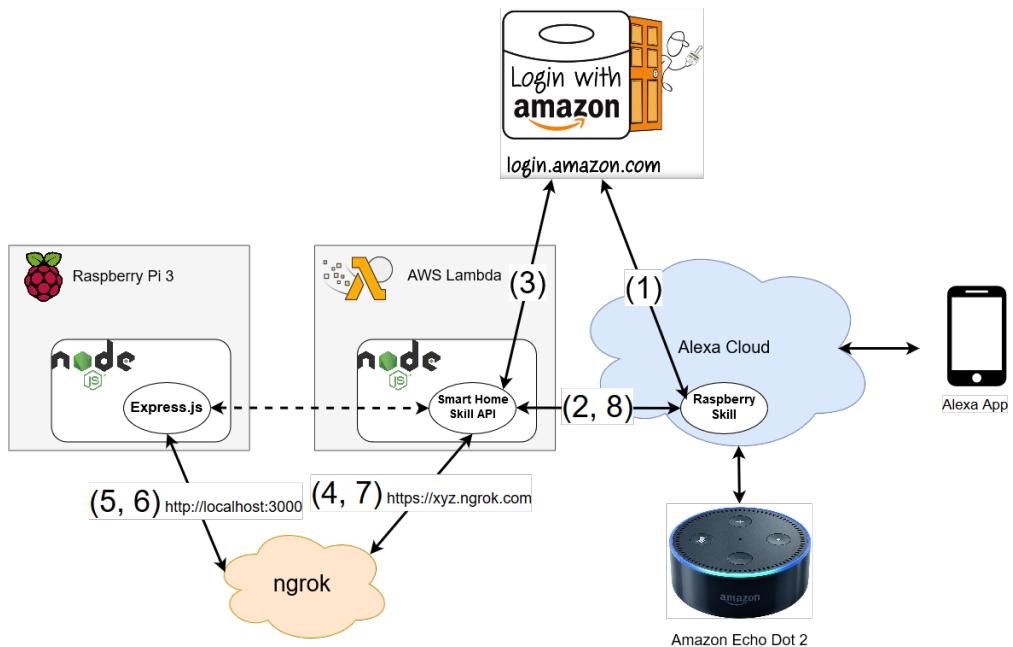


Abb. 5.7.: Raspberry Pi-Alexa-Kommunikation Datenfluss

5.2.1. AWS Lambda

In diesem Abschnitt wird die Entwicklung rund um AWS Lambda beschrieben. Hierzu gehört die Kommunikation zwischen AWS Lambda und der Alexa Cloud, das Erhalten und Verarbeiten von Benutzerinformationen mit LWA und die Kommunikation mit dem Raspberry Pi.

Export Handler

Der Export Handler wird in Schritt zwei und acht der Abbildung 5.7 benötigt. Mit ihm wird die Kommunikation zwischen der Alexa Cloud und AWS Lambda geregelt.

Mit `exports.handler` wird die Funktion definiert, welche ausgeführt wird, sobald eine Direktive an die Lambda Funktion gesendet wird. In dem Parameter „`request`“ stecken die Informationen, welche mit der Direktive gesendet werden. Der Parameter „`context`“ wird von AWS Lambda dazu verwendet Laufzeitinformationen zur ausgeführten Lambda-

Funktion bereitzustellen.

Hier im exports.handler werden die verschiedenen Typen an Direktiven zunächst einmal anhand vom „header“ unterschieden und entsprechend weitere Aktionen durchgeführt.

```
1 exports.handler = (request, context) => {
2     if (request.directive.header.namespace === 'Alexa' && request.
3         ↪ directive.header.name === 'ReportState') {
4         console.log("DEBUG:", "ReportState Request", JSON.stringify(
5             ↪ request));
6         handleReportState(request, context);
7     }
8     else if (request.directive.header.namespace === 'Alexa.Discovery'
9             && request.directive.header.name === 'Discover') {
10        console.log("DEBUG:", "Discover Request", JSON.stringify(
11            ↪ request));
12        handleDiscovery(request, context);
13    }
14    else if (request.directive.header.namespace === 'Alexa.
15        ↪ PowerController') {
16        if (request.directive.header.name === 'TurnOn' || request.
17            ↪ directive.header.name === 'TurnOff') {
18            console.log("DEBUG:", "TurnOn or TurnOff Request", JSON.
19                ↪ stringify(request));
20            handlePowerControl(request, context);
21        }
22    }
23    else if (request.directive.header.namespace === 'Alexa.
24        ↪ Authorization') {
25        if (request.directive.header.name === 'AcceptGrant') {
26            console.log("DEBUG:", "Authorization Request", JSON.
27                ↪ stringify(request));
28            handleAuthorizationAcceptGrant(request, context);
29        }
30    }
31    else {
32        console.log("DEBUG:", "Unknown Directive", JSON.stringify(
33            ↪ request));
34    }
35}
```

```
24 }  
25 };
```

Quellcode 5.10: Export Handler

Direktiven und Events

Die Direktiven, welche von Alexa gesendet werden, sind Nachrichten im JSON-Format. Sie beschreiben Aktionen, welche ausgeführt werden sollen, sowie Informationen zu dem Client, welcher die Aktion ausführen möchte. Die Antworten, die auf die Direktiven folgen, werden im Alexa Skill Umfeld als Events bezeichnet.

Die Direktiven sind wie folgt aufgebaut [AMA18e]:

- header

Im header stehen beschreibende Informationen über die Direktive:

- namespace: Ein String, der die Kategorie der Nachricht und der Nutzdaten angibt.
- name: Der Name der Direktive, wie z. B. TurnOn (Einschalten).
- payloadVersion: Die Version der Schnittstelle, an welche diese Nachricht gesendet wird.
- messageId: Ein eindeutiger Identifikator für eine einzelne Anfrage oder Antwort. Dieses Attribut ist wichtig für Zwecke wie Tracking, um eine Nachricht eindeutig zu identifizieren und in den Logs abzulegen.
- correlationToken: Ein Token, das eine Direktive und ein oder mehrere damit verbundene Events identifiziert. Eine Antwort auf eine Direktive muss also auch zwingend ein correlationToken enthalten, damit der Kontext und Grund der Antwort ersichtlich ist. Wird auf eine Direktive direkt (synchron) geantwortet, so ist ein correlationToken nicht immer notwendig. Eine Antwort auf eine Direktive muss jedoch nicht zwingend vom selben Server kommen, an welchen die Direktive gesendet wurde. In diesem Fall muss ein correlationToken zwingend mitgesendet werden, damit das Event einer Direktive zugeordnet werden kann.

- endpoint

Ein Endpunktobjekt identifiziert das Ziel einer Direktive und den Ursprung eines Ereignisses.

- scope: Ein polymorphes Objekt, welches die Authentifizierung des Nachrichtenaustauschs beschreibt. Der Skill, welcher in dieser Studienarbeit erstellt wird, nutzt OAuth2 mit Bearer Token, weshalb sich hier im scope auch lediglich ein Bearer Token mit Beschreibung, zur Erkennung als solches, befindet.
- endpointId: Identifiziert einen Endpunkt, wie z. B. eine einzelne Funksteckdose oder Kamera, in Verbindung mit dem Benutzer eindeutig.
- cookie: Eine Liste von Schlüssel-Wertpaaren, die dem Endpunkt zugeordnet sind. Diese werden während der Geräteerkennung, im Discovery-Prozess, bereitgestellt und in jeder Nachricht für einen Endpunkt gesendet. Die Cookies sind optional. Sie werden in diesem Projekt nicht benötigt und daher nur exemplarisch mit einem beispielhaften Wertepaar dargestellt.
- payload

Die Nutzlast (payload), die mit einer Nachricht gesendet wird, unterscheidet sich in der Art und dem Typ der Direktiven oder der darauf folgenden Event Antwort. Bei einer Discovery-Direktive befindet sich in der payload z. B. die Authentifizierung mit Bearer Token, während bei einer ReportState-Direktive die payload leer bleibt.

Im folgenden Abschnitt werden einige Beispiele für Direktiven (Request) und ihrer Antworten (Response/Event) vorgestellt, welche in dieser Studienarbeit auch tatsächlich verarbeitet werden. Für eine bessere Lesbarkeit wurden das Bearer Token, die messageId und das correlationToken durchgängig mit „SomeRandomString“ ersetzt:

Report-State Über die Report-State-Direktive wird der Status der Smart Home Geräte abgefragt. Dies geschieht insbesondere dann, wenn der Benutzer über sein Smartphone in der Alexa App seine Geräte ansieht. In diesem Fall werden in kleinen zeitlichen Abständen kontinuierlich ReportState-Direktiven gesendet. Der folgende Code zeigt ein Beispiel für einen derartigen Report-State, welche nach dem derzeitigen Status des Geräts (Endpunkt) mit der ID „wirelessSwitch1“ fragt:

```

1 {
2   "directive": {
3     "header": {
4       "namespace": "Alexa",
5       "name": "ReportState",
6       "payloadVersion": "3",
7       "messageId": "SomeRandomString",
8       "correlationToken": "SomeRandomString"
9     },
10    "endpoint": {

```

```

11      "scope": {
12          "type": "BearerToken",
13          "token": "SomeRandomString"
14      },
15      "endpointId": "wirelessSwitch1",
16      "cookie": {
17          "key1": "This is just an example!"
18      }
19  },
20  "payload": {}
21 }
22 }
```

Quellcode 5.11: Request: Report-State

Die Direktiven von ReportState (Siehe Quellcode 5.11) und PowerController (Siehe Quellcode 5.13) ähneln sich sehr in ihrer Struktur und Beschreibung. Der einzige Unterschied liegt in den Abweichungen im header. Während bei einer ReportState-Direktive der namespace „Alexa“ lautet, ist der dieser bei einer PowerController-Direktive „Alexa.PowerController“. Das name-Attribut ist ebenfalls unterschiedlich. Er lautet entweder „ReportState“ oder bei der PowerController-Direktive „TurnOn“ oder „TurnOff“. Der Name in PowerController bestimmt die Intention der Direktive, also ob das Gerät (oder Licht) an oder aus geschaltet werden soll.

Analog zu den Direktiven sind auch die Antworten, in der Alexa Dokumentation „Events“ genannt werden, von ReportState sehr ähnlich aufgebaut. Sie unterscheiden sich wiederum nur im header, welcher sich aber im Event an einer anderen Stelle befindet, als bei den Direktiven. In den Events liegt der Unterschied zwischen ReportState und PowerController sogar ausschließlich im header-Attribut name. Dieses hat im ReportState den Wert „ReportState“ und im PowerController-Event den Wert „Response“.

Wie bereits erwähnt, haben die Event-JSONs einen anderen Aufbau als die der Direktiven. Es folgt nun eine Beschreibung des Aufbaus in ähnlicher Form wie die Beschreibung der Direktiven, geltend für ReportState und PowerController. [AMA18e]:

- context

Das Context-Objekt wird dazu verwendet, um den Zustand eines Endpunkts in einem beliebigen Event zu bestimmen:

- properties

Ein Kontext enthält ein Array von meldepflichtigen properties (Eigenschafts-

objekten) und deren Zustand:

- namespace: Ein String, der die Kategorie der Nachricht und der Nutzdaten angibt.
- name: Der Name der Eigenschaft.
- value: Ein Einzelter Wert einer Eigenschaft, wie z. B. „On“, oder aber ein weiteres Unterobjekt mit Werten, wie z. B. „{ „value“: 25.0, „scale“: „Celsius“ }“ für eine Temperaturangabe.
- timeOfSample: Der Zeitpunkt, zu dem der Eigenschaftswert im ISO 8601-Format abgetastet wurde. Angezeigt wird der Wert im UTC Format.
- uncertaintyInMilliseconds: Gibt die Unsicherheit der gemeldeten Eigenschaft in Millisekunden der verstrichenen Zeit an, seit der Wert der Eigenschaft möglicherweise abgerufen wurde. Wenn z. B. der Wert erhalten wird, indem alle 60 Sekunden ein Hardwaregerät abgefragt wird, dann beträgt die Unsicherheit in der Zeit des Abtastwertes 60.000 (in Millisekunden).

▪ event

Beschreibungen über das Event:

- header

Der header der Events ist analog zu dem header der Direktiven aufgebaut (Siehe 5.2.1 Direktiven und Events).

- endpoint

Auch die Endpunktobjekte der Events werden analog zu denen der Direktiven definiert (Siehe 5.2.1 Direktiven und Events).

- payload

Die Payload (Nutzlast) einer Nachricht kann je nach Art der Nachricht variieren. In den Folgenden Beispielen wird die Payload nicht benötigt und bleibt dementsprechend leer.

- payload

Die Payload (Nutzlast) einer Nachricht kann je nach Art der Nachricht variieren. In den Folgenden Beispielen wird die Payload nicht benötigt und bleibt dementsprechend leer.

```
1 {  
2   "context": {  
3     "properties": [  
4   ]  
5 }
```

```

4      {
5          "namespace": "Alexa.PowerController",
6          "name": "powerState",
7          "value": "OFF",
8          "timeOfSample": "2018-03-21T08:19:13.691Z",
9          "uncertaintyInMilliseconds": 500
10     }
11   ]
12 },
13 "event": {
14     "header": {
15         "messageId": "SomeRandomString",
16         "correlationToken": "SomeRandomString",
17         "namespace": "Alexa",
18         "name": "StateReport",
19         "payloadVersion": "3"
20     },
21     "endpoint": {
22         "scope": {
23             "type": "BearerToken",
24             "token": "SomeRandomString"
25         },
26         "endpointId": "wirelessSwitch1",
27         "cookie": {}
28     },
29     "payload": {}
30 },
31 "payload": {}
32 }
```

Quellcode 5.12: Response: Report-State

Power Controller Über den PowerController wird versucht, aktiv einen Gerätestatus zu verändern, wenn z. B. der Benutzer sein Licht einschaltet oder die Heizung aufdrehen möchte.

```

1 {
2     "directive": {
3         "header": {
```

```

4      "namespace": "Alexa.PowerController",
5      "name": "TurnOn",
6      "payloadVersion": "3",
7      "messageId": "SomeRandomString",
8      "correlationToken": "SomeRandomString"
9    },
10     "endpoint": {
11       "scope": {
12         "type": "BearerToken",
13         "token": "SomeRandomString"
14       },
15       "endpointId": "wirelessSwitch1",
16       "cookie": {
17         "key1": "This is just an example!"
18       }
19     },
20     "payload": {}
21   }
22 }
```

Quellcode 5.13: Request: Power-Controller

Die in der Antwort (Response/Event, Siehe Quellcode: 5.12 und 5.14) stehenden properties-Daten, stammen direkt vom Server des Raspberry Pi. Die properties und damit verbundenen Gerätzustände für jedes Gerät, welches mit dem Raspberry Pi verbunden ist, werden zur Laufzeit des Servers im Arbeitsspeicher gehalten und behandelt wie eine Art In-Memory-Datenbank. Sie sind in der selben Objektstruktur gespeichert, wie in den Events. Dies ermöglicht eine performante Verarbeitung und vereinfacht die Entwicklung durch bessere Lesbarkeit und Verständnis.

```

1 {
2   "context": {
3     "properties": [
4       {
5         "namespace": "Alexa.PowerController",
6         "name": "powerState",
7         "value": "ON",
8         "timeOfSample": "2018-03-21T08:19:11.976Z",
9         "uncertaintyInMilliseconds": 500
10      }
11    ]
12  }
13}
```

```

11     ]
12   },
13   "event": {
14     "header": {
15       "messageId": "SomeRandomString",
16       "correlationToken": "SomeRandomString",
17       "namespace": "Alexa",
18       "name": "Response",
19       "payloadVersion": "3"
20     },
21     "endpoint": {
22       "scope": {
23         "type": "BearerToken",
24         "token": "SomeRandomString"
25       },
26       "endpointId": "wirelessSwitch1",
27       "cookie": {}
28     },
29     "payload": {}
30   },
31   "payload": {}
32 }
```

Quellcode 5.14: Response: Power-Controller

Discovery Über die Discovery-Direktive fragt Alexa nach verfügbaren Smart Home Geräten. Diese Direktive wird gesendet, wenn der Benutzer aktiv in der Alexa App nach Geräten suchen lässt oder „Alexa, finde meine smarten Geräte“ sagt. Ein Discovery-Request wird wieder, analog zu den vorherigen Direktiven, anhand des namespace „Alexa.Discovery“ und am Name „Discovery“ erkannt.

```

1 {
2   "directive": {
3     "header": {
4       "namespace": "Alexa.Discovery",
5       "name": "Discover",
6       "payloadVersion": "3",
7       "messageId": "SomeRandomString" },
8     "payload": {
```

```

9      "scope": {
10         "type": "BearerToken",
11         "token": "SomeRandomString"
12     }
13 }
14 }
```

Quellcode 5.15: Request: Discovery

Wird in der Lambda Funktion eine Discovery-Anfrage registriert, so wird eine Discovery-Anfrage an den Server des Raspberry Pi gesendet, von wo aus dann eine komplette Geräteübersicht in korrekter JSON-Form (Siehe Anhang A.1 Response: Discovery) des von Alexa angefragten Events (Siehe Quellcode: 5.15) zurückgesendet wird.

Die Antwort auf eine Discovery-Direktive hat die folgende Struktur. Wie auch bei den anderen Direktiven und Events können die Nachrichten, bei unterschiedlichen Nutzungs-szenarien, auch eine abweichende Struktur besitzen. Die hier erläuterte Struktur orientiert sich direkt am Einsatzszenario dieser Studienarbeit und besitzt demzufolge nur Endpunkte in Form von Lichtern und Schaltern [AMA18k]:

- header

Das header-Objekt enthält analog zu den zuvor Beschriebenen headern beschreibende Informationen über das Event.

- payload

Die Nutzdaten des Events:

- endpoints

Eine Auflistung der Endpunkte, welche mit dem Raspberry Pi verknüpft sind, inklusive Beschreibung der Fähigkeiten und Typbezeichnung:

- endpointId: Eindeutige Gerätbezeichnung
- manufacturerName: Herstellername
- friendlyName: Anzeigename, welcher später in der App gezeigt wird und in Befehlen an Alexa genannt wird.
- description: Beschreibung des Geräts
- displayCategories: Kategorien der Geräts. Gibt der Alexa App Hinweise, wie und wo das Gerät angezeigt werden soll.
- cookie: (optional) Eine Liste von Schlüssel-Wertpaaren, die dem Endpunkt zugeordnet sind. In den Codebeispielen nur exemplarisch dargestellt.

- capabilities:

Eine Auflistung der verschiedenen Fähigkeiten des Geräts. Hier unterscheiden sich die Beschreibungen zwischen verschiedenen Geräten natürlich stark, da natürlich eine Kamera verglichen zu einer Lampe bspw. kaum ähnliche Fähigkeiten besitzt.

- type: Gibt die Art der Fähigkeit an, die bestimmt, welche Felder die Fähigkeit hat.
- interface: Der Name der Schnittstelle, die die Aktionen für das Gerät beschreibt.
- version: Gibt die interface-Version an, welche der Endpunkt unterstützt.
- properties:

 - Eigenschaften einer Fähigkeit:
 - properties.supported: Gibt die vom Endpunkt unterstützten Eigenschaften an
 - properties.retrieveable: Gibt an, ob die für diesen Endpunkt aufgelisteten Eigenschaften für das Reporting (mit Report-State-Direktiven) abgerufen werden können.

Authorization Eine Authorization-Direktive wird gesendet wenn der Benutzer das Account-Linking durchführt. Diese Direktive ist dafür vorgesehen einen neuen Benutzer in seinem System zu registrieren.

Der header dieser Direktive hat einen ähnlichen Aufbau wie die zuvor beschriebenen Direktiven. Die payload besteht hier aus zwei polymorphen Objekten [AMA18m]:

- grant

Das grant-Objekt enthält einen OAuth2-Authorization-Code, welcher bei Bedarf gegen ein Access- (Bearer-) oder Refresh-Token eingetauscht werden kann.

- grantee

Das grantee-Objekt beinhaltet ein Bearer Token zur Authentifizierung des Benutzers.

```

1 {
2   "directive": {
3     "header": {
4       "namespace": "Alexa.Authorization",
5       "name": "AcceptGrant",

```

```

6      "payloadVersion": "3",
7      "messageId": "SomeRandomString"
8  },
9  "payload": {
10     "grant": {
11         "type": "OAuth2.AuthorizationCode",
12         "code": "SomeRandomString"
13     },
14     "grantee": {
15         "type": "BearerToken",
16         "token": "SomeRandomString"
17     }
18 }
19 }
20 }
```

Quellcode 5.16: Request: Authorization

In diesem Projekt wird die Authorization-Direktive nur entgegengenommen und daraufhin umgehend ein Event zurück geschickt. Das Sichern der Benutzerdaten, z. B. in einer Datenbank, wird in dieser Studienarbeit nicht benötigt (Siehe Abschnitt 5.2.1).

Wurde die Direktive erfolgreich bearbeitet, wird mit einer sehr schlichten Antwort direkt ein Event zurückgeschickt:

```

1 {
2   "event": {
3     "header": {
4       "messageId": "SomeRandomString",
5       "namespace": "Alexa.Authorization",
6       "name": "AcceptGrant.Response",
7       "payloadVersion": "3"
8     },
9     "payload": {}
10   }
11 }
```

Quellcode 5.17: Response: Authorization

Benutzerverwaltung

In diesem Abschnitt wird nun näher auf das Management der Benutzer und den Datenfluss zur Authentifizierung eines Benutzers im System eingegangen. Hinsichtlich der Architekturübersicht entspricht dies Schritt drei in Abbildung 5.7.

Um das Ergebnis der Studienarbeit zu veröffentlichen und damit marktreif zu machen, wäre eine Datenbank für Benutzer als zusätzliche Komponente notwendig. Die Anzahl derzeitiger Benutzer beschränkt sich aber auf nur zwei Personen, was eine Datenbank zu diesem Zeitpunkt überflüssig macht und die dafür gesparte Zeit sinnvoller in neue Features, Stabilität und Performance-Optimierung investiert wurde.

Die Benutzerverwaltung wird hier in der Arbeit direkt mithilfe einer einfachen Fallunterscheidung im Quellcode 5.18 durchgeführt. Unterschieden wird anhand der eindeutigen Amazon-Account-ID:

```

1 exports.checkUser = (bearerToken, callbackUserHost) => {
2     obtainCustomerProfileInformation(bearerToken, (user) => {
3         if (user.user_id === 'amzn1.account.SomeRandomString') { // 
4             ↪ user_id Robin
5             callbackUserHost('robin.eu.ngrok.io', 'benutzer:passwort')
6             ↪ ;
7         }
8         else if (user.user_id === 'amzn1.account.SomeRandomString') {
9             ↪ //user_id Max
10            callbackUserHost('max.eu.ngrok.io', 'test:test');
11        }
12        else {
13            callbackUserHost('xyz.eu.ngrok.io', 'test:test');
14        }
15    });
16};

```

Quellcode 5.18: Benutzerkontrolle

Für jeden Benutzer wird die ngrok Adresse, mit zugehörigem Passwort für die Basic Authentifizierung, an die aufrufende Funktion über eine Callback-Funktion übergeben.

Um überhaupt an die Amazon-Account-ID zu kommen, muss im-OAuth2-Stil ein Bearer Token an den Ressourcenserver (Siehe Abschnitt 3.6.1) übergeben werden, welcher im Austausch dazu die angeforderten Benutzerinformationen als Antwort zurück liefert. Das

Bearer Token wird bei jeder von der Alexa Cloud gesendeten Direktive, an die Lambda Funktion übergeben (Siehe Abschnitt 5.2.1). Dies geschieht, indem im Quellcode 5.18 die Funktion „obtainCustomerProfileInformation“ (Siehe Quellcode 5.19) aufgerufen wird:

```

1 let obtainCustomerProfileInformation = (bearerToken,
2   ↪ callbackProfileInformation) => {
3   const options = {
4     hostname: 'api.amazon.com',
5     path: '/user/profile',
6     method: 'GET',
7     headers: {
8       Authorization: 'bearer ' + bearerToken
9     }
10    };
11    const req = https.request(options, (res) => {
12      res.setEncoding('utf8');
13      res.on('data', (data) => {
14        console.log(`RESPONSE Obtain Customer Profile Information:
15          ↪ \${data}`);
16        callbackProfileInformation(JSON.parse(data));
17      });
18      res.on('end', () => {
19        console.log(`RESPONSE Obtain Customer Profile Information:
20          ↪ No more data in response`);
21      });
22    });
23    req.on('error', (e) => {
24      console.error(`RESPONSE-Error Obtain Customer Profile
25          ↪ Information: problem with request: \${e.message}`);
26    });
27    req.end();
28  };

```

Quellcode 5.19: Benutzerinformationen

Quellcode 5.19 zeigt anschaulich wie das Bearer Token mit HTTPS an den Ressourcenserver übergeben wird. Hierfür wird ein HTTP(S) GET-Request, mit zusätzlichem „Authorization“-Attribut und darin enthaltenem Bearer Token, an den Ressourcenserver (Amazon API) gesendet. Mittels Callback-Funktionen wird daraufhin auf die Antwort des Ressourcenserver gewartet. In der Callback-Funktion von „res.on()“ werden die Daten

verarbeitet und wiederum mittels Callback-Funktion „callbackProfileInformation()“ an die aufrufende Funktion „checkUser()“ (Siehe Quellcode 5.18) übergeben.

Datenfluss

In diesem Absatz geht es nun um den Datenfluss von einem Kommando, gerichtet an Alexa, hin zu einem Befehl, welcher auf dem Raspberry Pi ausgeführt und ordnungsgemäß verarbeitet werden soll (Siehe Abschnitt 5.2.2). Dieser Abschnitt stellt die Schritte eins bis fünf der Grafik 5.7 detaillierter dar.

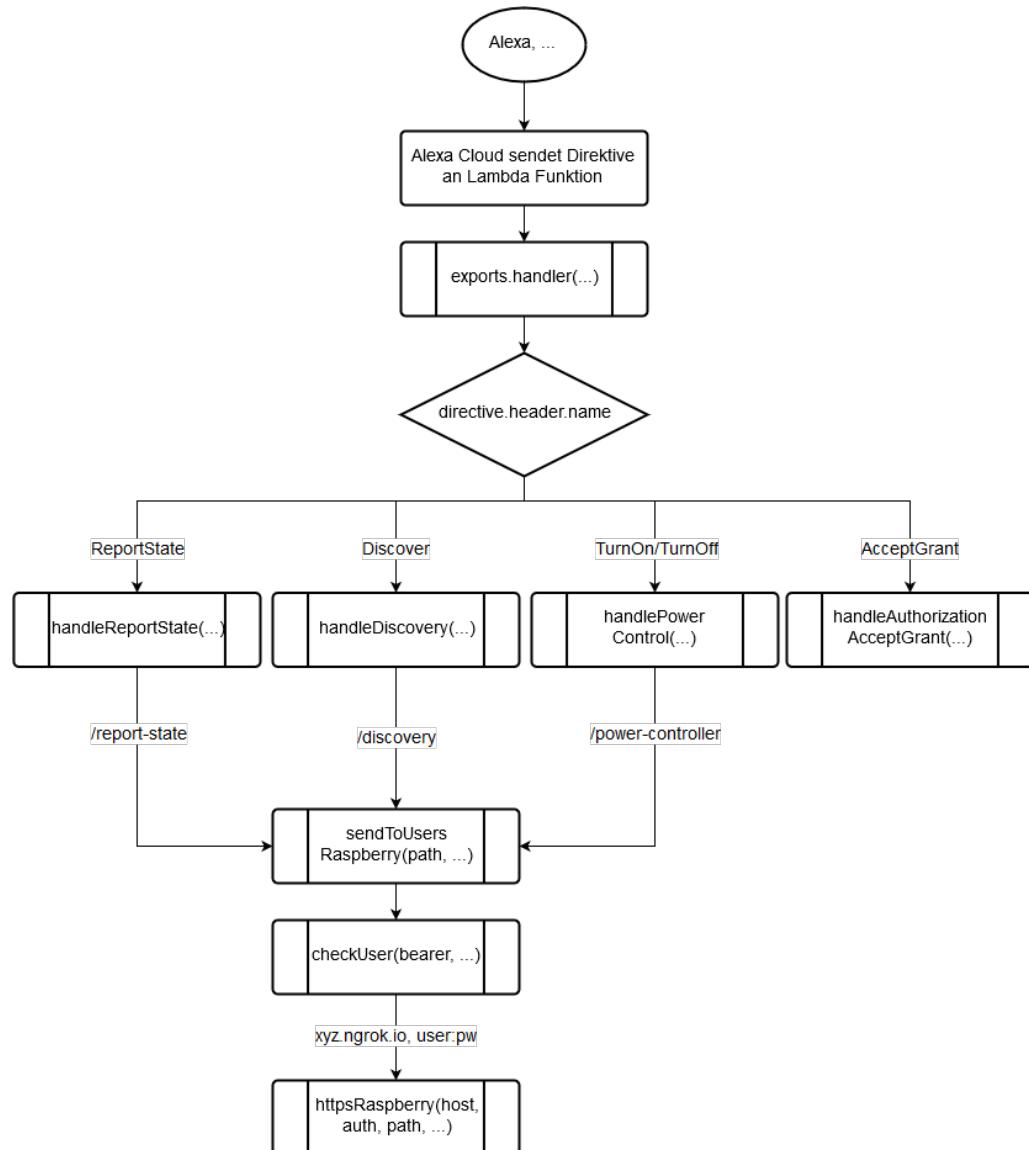


Abb. 5.8.: Flussdiagramm Alexa nach Raspberry Pi

Im Grunde beginnt es immer mit einer an Alexa gerichteten Anfrage. Dies kann ein gesprochenes Kommando sein, wie z. B. „Alexa, schalte das Licht an“, oder aber eine Betätigung eines virtuellen Schalter in der Alexa App auf dem Smartphone. Beide

Anfragen ziehen dieselben Konsequenzen nach sich. In der Alexa Cloud des Benutzers wird registriert, um welches Gerät es sich bei der Anfrage handelt und entsprechend mit dem richtigen Skill interagiert, wie z. B. dem hier in der Studienarbeit erstellten Skill. Der Skill führt Informationen über die weitere Verarbeitung der Anfrage und kann somit bestimmen, an welche Adresse eine Direktive mit den Befehlen gesendet werden soll. Bei Smart Home-Skills handelt es sich hierbei immer um eine AWS Lambda Funktion, an welche die Direktive gesendet werden soll.

Sobald in der Lambda Funktion eine Direktive ankommt, wird diese zunächst in der handler-Funktion kategorisiert, wie es in Abschnitt 5.2.1 näher beschrieben ist, um für das weitere Vorgehen eine entsprechende Funktion (Siehe Abbildung 5.8) zu wählen.

Handelt es sich bei einer Anfrage um eine Authorization-Direktive (Siehe 5.2.1), so wird umgehend ein Event an die Alexa Cloud zurück geschickt. Der Raspberry Pi ist bei dieser Direktive nicht von Bedeutung.

In den Funktionen „handleReportState()“, „handleDiscovery()“ und „handlePowerControl()“ werden die Direktiven weiter verarbeitet, sowie ihnen Daten und Informationen extrahiert, welche für die Weiterverarbeitung auf dem Raspberry Pi von Bedeutung sind. Gefolgt von einigen Vorverarbeitungen ist der Aufruf der Funktion „sendToUsersRaspberry()“. Bei diesem Funktionsaufruf wird unter anderem der Pfad für den Server des Raspberry Pi übergeben, an welchen die vorverarbeiteten Anfragen weitergeschickt werden sollen. Dieser Pfad unterscheidet sich je nach Funktion (Siehe Abbildung 5.8).

Nach dem Aufruf der Funktion „sendToUsersRaspberry()“ wird zunächst einmal der Benutzer überprüft, von welchem die Anfrage gesendet wurde, um die Adresse des dem Benutzer zugeordneten Raspberry Pi ausfindig zu machen. Diese Überprüfung erfolgt durch die Funktion „checkUser()“, wie es in Abschnitt 5.2.1 näher beschrieben ist.

Nachdem der Benutzer überprüft wurde und die Adresse und Anmeldedaten für die Basic Authentifizierung des ngrok Tunnels des Raspberry Pi feststehen, wird eine POST-Request über HTTPS an den Raspberry Pi geschickt. Dies geschieht in ähnlicher Form wie in dem Codebeispiel aus Abschnitt 5.2.1. Die relevanten Nutzdaten welche aus den Direktiven extrahiert wurden, werden im body des POST-Request mitgesendet.

5.2.2. Raspberry Pi

Der implementierte Code auf dem Raspberry Pi ist vorrangig in JavaScript geschrieben. Dieser unterteilt sich in Code für die Node.js-Laufzeitumgebung für die Imple-

mentierung des Webservers, Quelltext für die Erstellung des Webinterfaces sowie zur Geräteansteuerung über Node.js. Zusätzlich wird an manchen Stellen noch HTML und CSS benötigt. Dieser Abschnitt behandelt die Schritte fünf und sechs aus Abbildung 5.7 sowie die, auf dem Raspberry Pi laufende, Prozesse zwischen den Schritten.

Webserver

Mittels Node.js-Laufzeitumgebung wird, unter Zuhilfenahme des Express.js-Frameworks, ein Webserver implementiert. Der Webserver ist zunächst nur lokal im Heimnetzwerk bzw. im Intranet erreichbar über die lokale IP-Adresse auf Port 3000, wenn nicht im Code selbst angepasst. Mittels ngrok-Tunnel wird der Server aber auch aus dem Internet erreichbar.

Der Server ist erreichbar mittels HTTP GET- bzw. POST-Request über die folgenden Pfade:

- **GET**

Über die GET-Methode werden Informationen angefordert, jedoch können keine Daten im body des HTTP-Request gesendet werden. Im Quellcode-Beispiel 5.20 wird die GET Methode mittels „app.get(...“ definiert.

- „/“: Anfragen an das Home Verzeichnis liefern ein Webinterface zur Smart Home-Steuerung zurück (Siehe Abschnitt 5.2.2).
- „/discover“: Bei einer Discovery Anfrage werden die über den Raspberry Pi verfügbaren Endpunkte, in einer von Alexa verständlichen JSON-Form, zurück gesendet (Siehe Abschnitt 5.2.1)

- **POST**

Über die POST-Methode können Informationen angefordert und gleichzeitig Daten über den body des HTTP-Request gesendet werden. Im Quellcode-Beispiel 5.21 wird die POST Methode mittels „app.post(...“ definiert.

- „/report-state“: Für eine Report-State Anfrage muss im Body der Anfrage eine endpointId mitgeliefert werden. Die endpointId dient dazu, den Endpunkt zu identifizieren, von welchem gerade Daten angefordert werden. Die Antwort auf die Anfrage besteht aus einem JSON-Objekt, welches alle Eigenschaften des Endpunkt enthält, wie namespace, name, value, timeOfSample und uncertaintyInMilliseconds. (Siehe Abschnitt 5.2.1)
- „/power-controller“: Die Power-Controller-Anfrage ist der Report-State-Anfrage sehr ähnlich. Der einzige Unterschied liegt darin, dass bei einer Power-State-Anfrage ein gewünschter Zustand mitgesendet wird, welchen ein Endpunkt

annehmen soll. Z. B. wird ein Zustand „ON“ mitgesendet, welcher den mit der endpointId definierten Endpunkt einschalten soll. (Siehe Abschnitt 5.2.1)

Im Code besteht eine Trennung zwischen den Modulen „app“, „model“ und „public“. Unter dem Verzeichnis „public“ befinden sich sämtliche HTML, CSS und JavaScript Dateien, welche für das Webinterface benötigt werden. Das Webinterface in public wird über das app-Modul eingebunden und überliefert (Siehe Quellcode 5.20):

```

1 const express = require('express');
2 const app = express();
3
4 app.use(express.static('public'));
5 app.use("/styles", express.static(__dirname + '/public/css'));
6 app.use("/scripts", express.static(__dirname + '/public/js'));
7 app.use("/images", express.static(__dirname + '/public/images'));
8
9 app.get('/', function(req, res) {
10   res.sendfile(__dirname + '/public/index.html');
11 });

```

Quellcode 5.20: Node app.js: Webinterface

In dem Node.js-Modul app werden die Anfragen definiert, bearbeitet und gesteuert. Das Modul model enthält die Daten über die Endpunkte und liefert bzw. modifiziert diese über export-Schnittstellen. Wie in Quellcode 5.21 zu sehen ist, wird, im Stil des modularen Aufbaus von Node.js, das Modul model in app eingebunden und über Schnittstellen wie „getDevice(...)“ verwendet.

```

1 const model = require("./model.js")
2
3 app.post('/report-state', function(req, res) {
4   let deviceAttempt = req.body;
5   let device = model.getDevice(deviceAttempt);
6
7   res.send(JSON.stringify(device));
8 });

```

Quellcode 5.21: Node app.js: Report-State

Webinterface

Das einfache Webinterface (Siehe Abbildung 5.9), welches mithilfe des Front-End-Web-Framework Bootstrap [BSTRAP] gebaut wurde, stellt eine alternative Bedienoberfläche

für die Ansteuerung der Geräte, unabhängig von Alexa, dar. Es dient vorwiegend zum testen des Node.js-Scripts und der einfachen Ansteuerung der Funksteckdosen sowie der Kamera ohne Alexa. Z. B. können hiermit Tests ausgeführt werden, welche Alexa als Fehlerquelle ausschließen können. Mit dem Webinterface können Geräte an und aus geschaltet, Videoaufnahmen durchgeführt und ein Kamera Live-Stream betrachtet werden. Das Webinterface kann aber auch als eine Art Alternative zur Fernbedienung der Geräte dienen, welche keine Einschränkungen, hinsichtlich der Reichweite der 433MHz Sendeleistung (Siehe 3.5), besitzt und daher im gesamten Internet und Heimnetz erreichbar ist, auch ganz ohne Alexa.

Das Webinterface ist erreichbar über den Pfad „/“ des Node.js Server, auf Port 3000 im Heimnetzwerk bzw. HTTPS Standard-Port 443 der ngrok HTTPS-Adresse, auf dem Raspberry Pi.

Home Central v0.1



Abb. 5.9.: Webinterface

Funksteckdosen-Ansteuerung

Das Node.js-Skript 5.22 erstellt anhand der überlieferten Daten des Hausschlüssels und der Geräte-ID das Funksignal, welches dann über die GPIO-Ausgänge vom Funkmodul gesendet wird. Zur Ansteuerung der GPIO-Pins muss vorher das NPM-Paket „onoff“ [NODEd] eingebunden werden. Der folgende Code ist an den Python-Code zur Ansteuerung einer herkömmlichen Funksteckdose von Heiko H. [HEI12] angelehnt, in JavaScript übersetzt und für das Projekt angepasst.

```
1 _switch(sw){  
2     this.bit = [142, 142, 142, 142, 142, 142, 142, 142, 142, 142,  
3                 ↪ 142, 136, 128, 0, 0, 0];  
4     for (let i = 0; i < 5; i++) {  
5         if(this.key.charAt(i) == 1){  
6             this.bit[i] = 136;  
7         }  
8     }  
9     let x = 1;  
10    for (let i = 1; i < 6; i++) {  
11        if((this.device & x) > 0){  
12            this.bit[4 + i] = 136;  
13        }  
14        x = x << 1;  
15    }  
16    if(sw === "HIGH"){  
17        this.bit[10] = 136;  
18        this.bit[11] = 142;  
19    }  
20    let bangs = [];  
21    for (let y = 0; y < 16; y++) {  
22        x = 128;  
23        for (let i = 1; i < 9; i++) {  
24            let b = (this.bit[y] & x) > 0 ? 1 : 0;  
25            bangs.push(b);  
26            x = x >> 1;  
27        }  
28        this.output.writeSync(0);  
29        for (let i = 0; i < this.repeat; i++) {
```

```

30     for (let b of bangs) {
31         this.output.writeSync(b);
32         sleep.usleep(this.pulseLength);
33     }
34 }
35 }
```

Quellcode 5.22: Node.js-Funktion: Erstellen und Senden des Funksignals

Zunächst wird, wie bereits unter 4.4.2 Ansteuerung der Funksteckdosen beschrieben, eine Liste für die 16 zu sendenden Bytes erstellt und mit Platzhaltern befüllt. Die ersten zehn Bytes stehen hier für den Bit-Schlüssel, mit welchem die Funksteckdosen identifiziert werden. Die 142 bedeutet hierbei eine Schalterstellung mit Wert 0 und 136 eine Schalterstellung mit Wert 1, welche später in den entsprechenden Binärkode des Funksignals übersetzt werden. Diese Schalterstellungen entsprechen den Einstellungen des Hausschlüssels (Bit 0-4 Siehe „this.bit“ in Quellcode 5.22) und Steckdosen-Schlüssels (Bit 5-9) der Fernbedienung und Funksteckdosen, sowie der Codierung für „Einschalten“ und „Ausschalten“ (Bit 10-11). Die letzten 4 Bytes für die Synchronisation sind dabei schon feststehend, da diese bei jeder Übertragung gleich bleiben.

In den nächsten beiden Schritten werden die Bytes für den jeweiligen Hausschlüssel und die Geräte-ID gesetzt. Die Geräte-ID wurde dabei als eine Zweierpotenz übergeben. Das Steckdose auf der Position A ist somit 1, B ist 2 und C 4 etc. Für eine 1 an der jeweiligen Binärstelle des übergebenen Werts wird das Byte jeweils auf 136 gesetzt. Danach folgt die Codierung der Schaltrichtung (ein oder aus). Die Byte-Liste ist auf „Ausschalten“ voreingestellt und wenn „HIGH“ (für Anschalten) übertragen wurde, werden die Bytes getauscht.

Im letzten Schritt vor dem Absetzen des Signals wird die Liste Byte für Byte in die entsprechenden Binärcodes übersetzt und zu einer 128-bit langen Bit-Liste zusammengeführt. Dabei wird durch bitweise Verschiebung der 128 (Siehe Variable „x“ in Quellcode 5.22 nach rechts, überprüft, ob an der jeweiligen Position des aktuellen Bytes im Binärkode eine 1 oder eine 0 steht. Dementsprechend wird dann der jeweilige Ausschlag (engl. bang) an das Funksignal angehängt.

Zuletzt muss die hieraus resultierte Signalfolge über die GPIO-Ausgänge an das Funkmodul übergeben werden, wo es dann abgeschickt wird. Über das angesprochene GPIO-Paket wurde ein output-Pin definiert, an den die „Data“ -Leitung des Funkmoduls angeschlossen ist. Über diesen können dann die Einsen und Nullen synchron übertragen werden.

Zwischen jeder Stelle des Codes erfolgt eine kurze Pause bis zur nächsten potentiellen

Änderung des Ausschlags, welche auch als Puls bezeichnet wird. Diese beträgt hierbei 300 Mikrosekunden. Die EventLoop von JavaScript ist für solche geringen Zeitangaben aber nicht präzise genug und würde somit das Signal nicht richtig absenden. Aus diesem Grund wird ein weiteres NPM-Paket benötigt. Das „sleep“-Paket [NODEe] ist in der Lage durch C++ Bindings die Ausführung für kurze und exakte Zeiträume zu unterbrechen.

Kamera-Ansteuerung

Die ersten Versuche, die Kamera über ein Node.js-Skript anzusteuern, wurden mit dem NPM-Paket „raspicam“ durchgeführt. Dieses verwendet aber auch nur das Kommandozeilenwerkzeug „raspivid“ auf umständliche Art und Weise. [RASPd] Aus diesem Grund fiel schlussendlich die Entscheidung darauf, raspivid direkt und ohne Umwege über zusätzliche Pakete zu benutzen. Es wird nur das Node.js eigene Paket „child_process“ benötigt, um aus dem Node.js-Skript heraus, Kommandozeilenbefehle auszuführen und somit neue Prozesse starten zu können.

Beim Befehl zum Starten der Kamera wird durch folgenden Befehl der raspivid-Prozess gestartet, der der Kamera über das Camera Interface übermittelt, dass die Aufnahme beginnt.

```
1 // raspivid -o video.h264 -t 3600000 -w 640 -h 360 -fps 10
2 this.raspividProcess = spawn('raspivid', ['-o', this.output, '-t',
  ↪ this.RaspividTime, '-w', this.RaspividWidth, '-h', this.
  ↪ RaspividHeight, '-fps', this.RaspividFps]);
```

Quellcode 5.23: Node.js: Starte Aufnahme mit Hilfe von raspivid

Auskommentiert steht ein beispielhafter Kommandozeilenbefehl, um die Aufnahme zu starten, und danach folgt der verwendete Aufruf aus dem Quelltext samt der zu übergebenden Parameter. Der Parameter „output“ gibt somit den Zielpfad samt den gewünschten Dateinamen mit entsprechendem Dateityp an, an dem die Aufnahme abgespeichert werden soll und hier relativ zum Projektpfad gewählt wurde. Danach folgt der Timeout der Aufnahme, welcher in Millisekunden angegeben wird. Das bedeutet, dass die Aufnahme nach dieser Zeit automatisch angehalten und abgespeichert wird. Für dieses Projekt wurde der Timeout auf eine Stunde gesetzt, um längere Aufnahmen zu ermöglichen, aber durch zu lange Aufnahmen, den eher kleinen Speicher des Raspberry Pi nicht unnötig auszulasten.

Die letzten Parameter stehen für die Videoeinstellungen. Es wird die Auflösung sowie die Framerate in Frames pro Sekunde angegeben, welche hier jeweils nicht wirklich groß ist, um die Größe der Aufnahmen weiter zu begrenzen.

Wird nun der Befehl zum Beenden der Aufnahme abgesetzt, muss der raspivid-Prozess nur noch durch „this.raspividProcess.kill();“ wieder beendet werden. Die Bild-Daten der Aufnahme werden durchgehend, während dem filmen, in die zuvor definierte „output“ Datei geschrieben. Beim Beenden des Prozesses ist die .h264 Datei somit automatisch am vorher definierten Ort abgespeichert. Um jeweils eindeutige Dateinamen zu haben und somit Namensproblemen vorzubeugen, wurde für den Dateinamen das aktuelle Datum und Uhrzeit beim Start der Aufnahme verwendet. Ein Beispiel für einen derartigen Namen ist z. B. „19Apr2018_17-09-32.h264“.

Da die Kameraaufnahmen nur als .h264 Dateien gespeichert werden, müssen diese zunächst in einen Container wie MP4 oder WebM gepackt werden. Dies Umwandlung von reinem Video-Codec in einen Container erfolgt mit FFmpeg (Siehe Abschnitt 5.1.5 zur Einrichtung). Die Entscheidung fiel hier zugunsten von MP4, da dieses Containerformat derzeit von den meisten Webbrowsersn unterstützt wird.

Der Live-Stream wird mit Hilfe der Programmbibliothek FFserver realisiert, welche zu FFmpeg gehört. Mit FFserver kann ein selbst konfigurierter, individueller Webserver mit Video Live-Stream erstellt werden. Mit der richtigen Konfiguration nimmt der Server einzelne Bilder der Raspberry Pi Kamera entgegen, konvertiert diese mit FFmpeg und stellt sie in einem gewünschten Format als Webserver zur Verfügung. Für die Konfiguration und Einrichtung des FFserver sind einige Schritte und Einstellungen notwendig, welche aber in dieser Studienarbeit, aufgrund mangelnder Relevanz, nicht weiter detailliert erläutert werden und aber dafür Shell Skripte, zur schnellen, unkomplizierten Einrichtung des Projekts, zur Verfügung stehen (Siehe Abschnitt 5.1.5). Der Live-Stream hat das Dateiformat MJPG, ein Videocodec, bei welchem jedes einzelne Bild als JPEG-Bild komprimiert und dann sofort übertragen wird. MJPG hat gegenüber anderer Live-Stream Formate, wie einem gepufferten und fragmentierten MP4 Stream, den Vorteil, eine geringere Latenz und sehr gute Browserunterstützung zu gewährleisten.

Da FFserver auf einen anderen Port reagiert, als der in dieser Arbeit erstellte Node.js Server, wurde im Node.js Server ein Proxy, mithilfe des Node.js Moduls „http-proxy“ [<https://www.npmjs.com/package/http-proxy>], eingerichtet, welcher den Datenverkehr von und zu der FFserver Instanz durchleitet. Dies erspart einen weiteren kostenpflichtigen Ngrok-Tunnel. Der Live-Stream ist über ein iFrame in das Webinterface integriert.

6. Zusammenfassung

Abschließend wird in diesem Kapitel das Projekt in der Retrospektive betrachtet. Die implementierten Funktionen werden anhand der anfangs aufgestellten Anforderungen bewertet, ein Ausblick auf mögliche Erweiterungen aufgezeigt sowie Probleme während der Bearbeitung dargestellt.

6.1. Ergebnis

Auf funktionaler Ebene wurden alle vorher gesetzten Anforderungen im vorgeschriebenen Zeitrahmen erfolgreich umgesetzt. Über Sprachbefehle können jegliche an die Funksteckdose angeschlossene Geräte problemlos an- und ausgeschaltet werden. Mit Alexa lassen sich auch einwandfrei Videoaufnahmen mit Hilfe der angeschlossenen Kamera aufzeichnen.

Weiterhin wurde das Webinterface, welches als Hauszentralsteuerung fungiert, ebenfalls mit vollständigem Funktionsumfang implementiert. Von hier aus können sowohl die Steckdosen geschalten als auch Videoaufnahmen gestartet und beendet werden. Weiterhin befindet sich auf der Plattform ein Livestream, welcher in der Lage ist dauerhaft das aktuelle Bild zu übertragen.

Auch im Bereich der nicht-funktionalen Anforderungen wurden guten Ergebnisse erzielt. Die geschaffene Systemumgebung ist stabil und zuverlässig. Es kam bisher zu keinen Ausfällen, was diese Behauptung unterstützt. Auch die Bedienung des Systems über Heimzentralsteuerung im Browser oder Alexa-App wird auf einfache und intuitive Art und Weise gewährleistet. Die Sprachbefehle „Licht/Kamera an“ usw. sind einleuchtend und auch das Webinterface ist übersichtlich und selbsterklärend. Das Projekt gilt somit als erfolgreich abgeschlossen.

6.2. Ausblick

Obwohl die gestellten Herausforderungen alle erfüllt wurden sind nach wie vor Verbesserungspotentiale bzw. mögliche Erweiterungen immer vorhanden. Das Webinterface als Zentralsteuerung bietet hier beispielsweise mehrere Möglichkeiten. So wäre es zum

Beispiel nicht schlecht, die Farbdarstellung und Videoqualität der Kamera, bei gleichbleibender Bandbreite, sowohl bei Aufnahmen als auch beim Livestream zu optimieren.

Weiterhin könnten auch Einstellungsmöglichkeiten für die Videos angeboten werden. Eine Idee ist dabei, zeitgesteuerte Videos einzuführen, um somit Aufnahmen mit einer voreingestellten Zeit zu starten oder zu beenden. Zudem wäre es gut, wenn der Benutzer direkt auf der Weboberfläche einstellen könnte, wie hoch die Auflösung und die Framerate der Aufnahme betragen soll. Wenn eine geringe Auflösung gewünscht ist, kann somit Speicherplatz gespart werden.

Ein weiterer Punkt ist, dass der Speicherort, der vom Nutzer für die Videos vorgesehen ist, direkt im Webinterface einstellbar ist. Genauso gut wäre auch, ihm die Möglichkeit zu geben, neben dem Anschauen der Aufnahmen, diese auch direkt zu löschen, falls sie nicht mehr benötigt werden.

Im Bezug auf die Ansteuerung der Funksteckdosen ist es für den Anwender ebenso von Vorteil, wenn dieser seinen verwendeten Hausschlüssel direkt auf der Weboberfläche eingeben kann. Das spart Zeit bei der Konfiguration und das System könnte somit einfacher auf einen anderen Haushalt übertragen oder migriert werden.

Praktisch wäre auch, wenn der jeweilige Gerätetestatus, analog zur Alexa-App, auf dem Interface angezeigt werden würde. Somit könnte der Benutzer ohne Alexa-App, auch wenn er nicht daheim ist, einsehen, welche Geräte derzeit aktiv sind. Zusätzlich könnte auch noch der 433 MHz Frequenzbereich dauerhaft abgehört werden, um Änderungen des Gerätetestatus, ausgelöst von der Funkfernbedienung, abzuhören und diese ebenso als User-Feedback auf der Weboberfläche anzuzeigen.

Neben den funktionalen Erweiterungen wäre es auch spannend zu sehen, inwiefern eine Installation mit Googles Konkurrenzprodukt zu Alexa einfach umzusetzen ist. Eine Zweitinstallation mit dem Google Assistant würde weitere Vergleichsmöglichkeiten der digitalen Assistenten bilden. Neben dem Preisvergleich könnte somit auch der Implementierungsaufwand mit den jeweiligen Actions bzw. Skills verglichen werden und damit bessere Aussagen für Hobbyentwickler, die ihr eigenes Heim selbstständig automatisieren wollen, getroffen werden.

6.3. Fazit

Zusammenfassend lässt sich sagen, dass auch ohne die erwähnten Erweiterungen das Ziel der Umsetzung dieser Studienarbeit erfolgreich und sehr zufriedenstellend erreicht wurde. Mit der geschaffenen Installation wird eine Preisersparnis im Vergleich zu einer reinen Alexa-Installation mit den angepriesenen Amazonprodukten ermöglicht, welche

auch immer größer wird, umso mehr Geräte angeschlossen werden.

Jedoch sind während der Bearbeitung auch einige Probleme aufgetreten, welche aber konstruktiv behoben werden konnten. So wurde z. B. die anfangs im Konzept vorgestellte lückenhafte Architektur mit Hilfe von ngrok verbessert. Dadurch konnte die Verbindung des Lambda Server mit dem auf Localhost laufenden Express-Server getunnelt werden, wodurch keine Portweiterleitung im eigenen Heimnetzwerk eingerichtet werden musste.

Weiterhin beanspruchte die Ansteuerung der Funksteckdosen und der Kamera viel mehr Zeit als ursprünglich veranschlagt. Das Problem war hierbei zum einen, dass die kurze Pause zwischen dem Senden einzelner Ausschläge für die Steckdosen so kurz waren, sodass diese von Node.js selbst nicht messbar bzw. ausführbar waren. Das eigentlich richtig zusammengebaute Funksignal wurde dadurch mit falschen Pulsfrequenzen übertragen, was dann aber durch das im Kapitel 5.2.2 Funksteckdosen-Ansteuerung angesprochene Paket gelöst werden konnte.

Ein weiteres Paket, welches Probleme bereitete war „raspicam“ zur Ansteuerung der Kamera. Dabei ließen sich die Videoaufnahmen nicht richtig beenden und danach wieder ordentlich starten. Aus diesem Grund wurde später dann das Paket verworfen und mit den Prozessen von „raspivid“ (Siehe 5.2.2 Kamera-Ansteuerung) direkt gearbeitet. Dies funktioniert auch wie gewünscht. Es kam nur zu Komplikationen mit dem Livestream, denn der Livestream kann nicht gleichzeitig laufen, wenn gerade eine Kameraaufnahme läuft. Somit wird zur Lösung des Problems, der Prozess für den Livestream beendet und direkt nach Beenden der Aufnahme wieder gestartet.

Trotz der aufgetretenen Probleme ist das Ergebnis dieser Studienarbeit bemerkenswert, da jegliche Anforderungen ordentlich umgesetzt und dokumentiert wurden. Die selbst gebastelte und personalisierte Hausautomation mittels Raspberry Pi kann auch weiterhin in Zukunft für den Eigenbedarf genutzt und einfach erweitert werden. Zudem konnten wertvolle Erfahrungen im Bereich der Softwareentwicklung sowie Hardwareansteuerung gesammelt werden.

Literatur

- [ALEXAa] AMAZON.COM. *Amazon Alexa Logo*. abgerufen am 16.03.2018 09:24 Uhr. 2018. URL: https://m.media-amazon.com/images/G/01/mobile-apps/dex/avs/docs/ux/branding/mark2._TTH_.png [siehe S. 13].
- [ALEXAb] AMAZON.COM. *Amazon Alexa Logo*. abgerufen am 01.12.2017 15:23 Uhr. 2017. URL: https://www.wink.com/img/vendor/logo_amazon.svg [siehe S. 34].
- [AMA18a] AMAZON.COM. *Amazon Developer*. abgerufen am 27.02.2018 12:23 Uhr. 2018. URL: <https://developer.amazon.com/de/> [siehe S. 43].
- [AMA18b] AMAZON.COM. *Login with Amazon for Websites*. abgerufen am 27.02.2018 12:33 Uhr. 2018. URL: <https://developer.amazon.com/de/docs/login-with-amazon/web-docs.html> [siehe S. 48].
- [AMA18c] AMAZON.COM. *Authenticate an Alexa User to a User in Your System with Account Linking*. abgerufen am 27.02.2018 12:33 Uhr. 2018. URL: <https://developer.amazon.com/de/docs/smarthome/authenticate-an-alexa-user-account-linking.html> [siehe S. 48].
- [AMA18d] AMAZON.COM. *Host a Custom Skill as an AWS Lambda Function*. abgerufen am 20.03.2018 15:05 Uhr. 2018. URL: <https://developer.amazon.com/de/docs/custom-skills/host-a-custom-skill-as-an-aws-lambda-function.html> [siehe S. 49].
- [AMA18e] AMAZON.COM. *Smart Home Skill API Message Reference*. abgerufen am 27.02.2018 12:40 Uhr. 2018. URL: <https://developer.amazon.com/docs/smarthome/smart-home-skill-api-message-reference.html> [siehe S. 53, 55].
- [AMA18f] AMAZON.COM. *Understanding the Different Types of Skills — ASK*. abgerufen am 21.03.2018 10:36 Uhr. 2018. URL: [#Custom_Skills](https://developer.amazon.com/de/docs/ask-overviews/understanding-the-different-types-of-skills.html) [siehe S. 36–38].

- [AMA18g] AMAZON.COM. *Amazon Web Services*. abgerufen am 21.03.2018 13:45 Uhr. 2018. URL: <https://aws.amazon.com/de/> [siehe S. 24].
- [AMA18h] AMAZON.COM. *AWS Lambda*. abgerufen am 21.03.2018 13:47 Uhr. 2018. URL: <https://aws.amazon.com/de/lambda/features/> [siehe S. 24, 25].
- [AMA18i] AMAZON.COM. *Serverlose Datenverarbeitung*. abgerufen am 21.03.2018 14:09 Uhr. 2018. URL: <https://aws.amazon.com/de/serverless/> [siehe S. 25].
- [AMA18j] AMAZON.COM. *AWS Lambda Preise*. abgerufen am 21.03.2018 14:36 Uhr. 2018. URL: <https://aws.amazon.com/de/lambda/pricing/> [siehe S. 25].
- [AMA18k] AMAZON.COM. *Alexa Discovery Direktive*. abgerufen am 24.03.2018 18:23 Uhr. 2018. URL: <https://developer.amazon.com/de/docs/device-apis/alexa-discovery.html> [siehe S. 60].
- [AMA18l] AMAZON.COM. *Alexa Authorization Direktive*. abgerufen am 25.03.2018 19:40 Uhr. 2018. URL: <https://developer.amazon.com/de/docs/device-apis/alexa-authorization.html#oauth2.authorizationcode> [siehe S. 61].
- [BAG17] J. BAGER. „Assistent allgegenwärtig - Digitale Assistenten: vom Spielzeug für Nerds zur Bedienoberfläche für alles“. In: *c't 22/2017* [2017]. Online unter <https://www.heise.de/ct/ausgabe/2017-22-Digitale-Assistenten-vom-Spielzeug-fuer-Nerds-zur-Bedienoberflaeche-fuer-alles-3854034.html> - abgerufen am 06.02.2018 12:14 Uhr, S. 64–69 [siehe S. 6–8, 10, 12].
- [BIXBY] Samsung GROUP. *Samsung Bixby Logo*. abgerufen am 16.03.2018 09:41 Uhr. 2018. URL: <https://www.bixbysamsung.com/img/bixby-for-everyone.png> [siehe S. 13].
- [BLE17] Reche M. BLEICH H. „Freundlich, hölzern, clever und arrogant - Vier Sprachassistenten in ihrem natürlichen Umfeld“. In: *c't 22/2017* [2017]. Online unter <https://www.heise.de/ct/ausgabe/2017-22-Vier-Sprachassistenten-in-ihrem-natuerlichen-Umfeld-3853597.html> - abgerufen am 13.11.2017 10:58 Uhr, S. 80–85 [siehe S. 6–12].

- [BSTRAP] GETBOOTSTRAP.COM. *Bootstrap*. abgerufen am 26.03.2018 14:22 Uhr. 2018. URL: <https://getbootstrap.com/> [siehe S. 68].
- [C9.io] C9.IO. *C9.io Github Repository*. abgerufen am 27.02.2018 11:57 Uhr. 2018. URL: <https://github.com/c9> [siehe S. 25, 42].
- [C9AWS] AMAZON.COM. *AWS Cloud9*. abgerufen am 21.03.2018 10:55 Uhr. 2018. URL: <https://aws.amazon.com/de/cloud9/> [siehe S. 25].
- [CLA18] G. CLAUSER. *Amazon Echo vs. Google Home: Which Voice Controlled Speaker Is Best for You?* abgerufen am 19.02.2018 11:12 Uhr. 2018. URL: <https://thewirecutter.com/reviews/amazon-echo-vs-google-home/> [siehe S. 8, 10, 11].
- [DEV17] DEVOLO AG. *Definition Smart Home*. abgerufen am 17.11.2017 14:22 Uhr. 2017. URL: <https://www.devolo.de/smart-home/> [siehe S. 3].
- [ECHO] Amazon ECHO. *Amazon Echo Dot 2*. abgerufen am 01.12.2017 15:24 Uhr. 2017. URL: https://images-eu.ssl-images-amazon.com/images/I/41iz5Tw82IL._SY300_QL70_.jpg [siehe S. 34].
- [ETC17] RESIN.IO. *Etcher Download*. abgerufen am 27.10.2018 11:35 Uhr. 2018. URL: <https://etcher.io/> [siehe S. 41].
- [EXPJS] Inc STRONGLOOP. *Express - Schnelles, offenes, unkompliziertes Web-Framework für Node.js*. abgerufen am 21.03.2018 12:01 Uhr. o.J. URL: <http://expressjs.com/de/> [siehe S. 25].
- [FMPEG] Fabrice Bellard + COMMUNITY. *FFMPEG*. abgerufen am 16.04.2018 15:12 Uhr. 2018. URL: <https://www.ffmpeg.org/> [siehe S. 26].
- [GOASS] Google LLC. *Google Assistant Logo*. abgerufen am 16.03.2018 09:32 Uhr. 2018. URL: <https://i1.wp.com/www.techdotmatrix.com/wp-content/uploads/2017/11/Google-Assistant-troubleshoot.png?fit=800%5C2C424&ssl=1> [siehe S. 13].
- [HAR17] D. HARDAWAR. *One week with Samsung Bixby - It has potential, but can't yet compete with Apple, Google or Amazon*. abgerufen am 09.02.2018 13:17 Uhr. 2017. URL: <https://www.engadget.com/2017/08/23/samsung-bixby/> [siehe S. 13].
- [HEA17] N. HEATH. *What is the Raspberry Pi 3? Everything you need to know about the tiny, low-cost computer*. abgerufen am 28.03.2018 10:04 Uhr. 2017.

URL: <http://www.zdnet.com/article/what-is-the-raspberry-pi-3-everything-you-need-to-know-about-the-tiny-low-cost-computer/> [siehe S. 15].

- [HEI12] H. HEIKO. *elropi.py - Remote switch Elro using Python on Raspberry PI*. abgerufen am 10.11.2017 14:01 Uhr. 2012. URL: <https://pastebin.com/aRipYrZ6> [siehe S. 70].
- [ISO06] ISO 9241: *Ergonomie der Mensch-System-Interaktion - Teil 110: Grundsätze der Dialoggestaltung*. Standard. 2006 [siehe S. 27].
- [LUN17] J. LUNSFORD. *Setup JavaScript IDE on Pi 3*. abgerufen am 27.02.2018 12:03 Uhr. 2017. URL: <https://dev.to/jtlunsford/setup-javascript-ide-on-pi-3> [siehe S. 42].
- [MICOR] Microsoft CORPORATION. *Microsoft Cortana Logo*. abgerufen am 16.03.2018 09:38 Uhr. 2018. URL: <https://www.nimblechapps.com/wp-content/uploads/2015/01/I-am-Cortana1.jpg> [siehe S. 13].
- [MOZ17] H. BRAUN. *Mozilla Common Voice: Sprachsteuerung für alle und ohne Rückgriff auf die Cloud*. abgerufen am 06.12.2017 14:01 Uhr. 2017. URL: <https://www.heise.de/newsticker/meldung/Mozilla-Common-Voice-Sprachsteuerung-fuer-alle-und-ohne-Rueckgriff-auf-die-Cloud-3904454.html> [siehe S. 4].
- [NGROK] Alan SHREVE. *Ngrok*. abgerufen am 21.03.2018 10:02 Uhr. 2018. URL: <https://ngrok.com/> [siehe S. 26, 50].
- [NODEa] NODE.JS FOUNDATION. *Node.js Logo*. abgerufen am 01.12.2017 15:25 Uhr. 2017. URL: <https://nodejs.org/static/images/logos/nodejs-new-pantone-black.png> [siehe S. 34].
- [NODEb] NODE.JS FOUNDATION. *Installing Node.js via package manager*. abgerufen am 27.02.2018 11:50 Uhr. 2018. URL: <https://nodejs.org/en/download/package-manager/> [siehe S. 42].
- [NODEc] NODE.JS FOUNDATION. *Node.js*. abgerufen am 21.03.2018 11:36 Uhr. 2018. URL: <https://nodejs.org/en/> [siehe S. 27].
- [NODEd] NPM. *onoff-npm*. abgerufen am 02.04.2018 15:15 Uhr. 2018. URL: <https://www.npmjs.com/package/onoff> [siehe S. 40, 70].

- [NODEe] NPM. *sleep -npm*. abgerufen am 02.04.2018 19:42 Uhr. 2018. URL: <https://www.npmjs.com/package/sleep> [siehe S. 72].
- [OAUTHa] IETF OAuth Working GROUP. *OAuth 2.0*. abgerufen am 27.02.2018 12:27 Uhr. 2018. URL: <https://oauth.net/2/> [siehe S. 18, 46].
- [OAUTHb] Aaron PARECKI. *OAuth 2.0 Simplified*. abgerufen am 16.03.2018 15:21 Uhr. 2018. URL: <https://aaronparecki.com/oauth-2-simplified/> [siehe S. 18].
- [PFI17] Kaufmann T. PFISTER B. *Sprachverarbeitung. Grundlagen und Methoden der Sprachsynthese und Spracherkennung*. 2. Aufl. Zürich, Schweiz: Springer Vieweg, 2017. ISBN: 978-3-662-52838-9 [siehe S. 4].
- [PuTTY] S. TATHAM. *Download PuTTY: latest release (0.70)*. abgerufen am 27.02.2018 11:40 Uhr. 2017. URL: <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html> [siehe S. 41].
- [RAO17] Manjunath K.E. RAO S. K. *Speech Recognition Using Articulatory and Excitation Source Feature. Speech Technology*. Indien: Springer, 2017. ISBN: 978-1-4842-2922-4 [siehe S. 4].
- [RASPa] Raspberry Pi FOUNDATION. *Raspberry Pi Logo*. abgerufen am 01.12.2017 15:26 Uhr. 2017. URL: <https://www.raspberrypi.org/app/uploads/2011/10/Raspi-PGB001.png> [siehe S. 34].
- [RASPb] Raspberry Pi FOUNDATION. *Download Raspbian for Raspberry Pi*. abgerufen am 26.02.2018 11:29 Uhr. 2018. URL: <https://www.raspberrypi.org/downloads/raspbian/> [siehe S. 41].
- [RASPc] Raspberry Pi FOUNDATION. *Raspberry Pi 3 Model B*. abgerufen am 28.03.2018 09:59 Uhr. 2018. URL: <https://www.raspberrypi.org/app/uploads/2017/05/Raspberry-Pi-3-1-1619x1080.jpg> [siehe S. 15].
- [RASPd] Raspberry Pi FOUNDATION. *RASPIVID*. abgerufen am 19.04.2018 17:12 Uhr. 2018. URL: <https://www.raspberrypi.org/documentation/usage/camera/raspicam/raspiivid.md> [siehe S. 72].
- [REI17] A. REINHARDT. *Samsung Bixby Voice im Alltags-Check*. abgerufen am 09.02.2018 13:18 Uhr. 2017. URL: <https://www.teltarif.de/samsung->

- bixby-sprachassistent-erfahrungsbericht/news/70083.html [siehe S. 13].
- [RF6749] Internet Engineering Task Force (IETF). *RFC 6749*. abgerufen am 15.03.2018 12:00 Uhr. 2012. URL: <https://tools.ietf.org/html/rfc6749> [siehe S. 18, 19, 22, 23].
- [RF6750] Internet Engineering Task Force (IETF). *RFC 6750*. abgerufen am 19.03.2018 13:32 Uhr. 2012. URL: <https://tools.ietf.org/html/rfc6750> [siehe S. 23].
- [RNW07] RN-WISSEN. *Funkmodule*. abgerufen am 01.04.2018 18:46 Uhr. 2007. URL: <http://rn-wissen.de/wiki/index.php?title=Funkmodule> [siehe S. 17, 18].
- [ROU17a] M. ROUSE. *Amazon Developer Services*. abgerufen am 09.01.2018 09:57 Uhr. 2017. URL: <http://searchaws.techtarget.com/definition/Amazon-Developer-Services> [siehe S. 24].
- [ROU17b] M. ROUSE. *Alexa Skills Kit*. abgerufen am 09.01.2018 10:31 Uhr. 2017. URL: <http://searchaws.techtarget.com/definition/Alexa-Skills-Kit> [siehe S. 24].
- [RVNC] RealVNC LIMITED. *Remotezugriffssoftware für Desktop-Computer und Mobilgeräte — RealVNC*. abgerufen am 27.02.2018 11:46 Uhr. 2018. URL: <https://www.realvnc.com/de/> [siehe S. 41].
- [SCH15] K. SCHILLER. *Smart Home Funkstandards für die Hausautomation*. abgerufen am 01.04.2018 18:16 Uhr. 2015. URL: <https://www.homeandsmart.de/smart-home-funk-standards-uebersicht-hausautomation> [siehe S. 17].
- [SCH17] K. SCHILLER. *Was ist ein Smart Home? Geräte und Systeme*. abgerufen am 10.11.2017 12:20 Uhr. 2017. URL: <https://www.homeandsmart.de/was-ist-ein-smart-home> [siehe S. 3].
- [SIRI] IOSWELT. *Apple Siri Logo*. abgerufen am 16.03.2018 09:29 Uhr. 2018. URL: <http://ioswelt.de/wp-content/uploads/2017/10/59e781abe09a00.21036803.jpeg> [siehe S. 13].

- [SKDOC] Apple Inc. *SiriKit — Apple Developer Documentation*. abgerufen am 19.03.2018 13:31 Uhr. 2018. URL: <https://developer.apple.com/documentation/sirikit> [siehe S. 9].
- [SOP17] M. E. SOPER. *Expanding Your Raspberry Pi. Storage, printing, peripherals, and network connections for your Raspberry Pi*. Indianapolis, USA: apress, 2017. ISBN: 978-1-4842-2922-4 [siehe S. 15, 16].
- [VOE16] F. VÖLKEL. *Die Historie des Smart Home von 1963 - Heute*. abgerufen am 02.02.2018 10:53 Uhr. 2016. URL: https://www.smallest-home.com/blog/smart_home_historie_1939_bis_2017_frank_voelkel/ [siehe S. 1].
- [W3SCHa] o.V. *Node.js and Raspberry Pi*. abgerufen am 14.11.2017 10:00 Uhr. o.J. a. URL: https://www.w3schools.com/nodejs/nodejs_raspberrypi.asp [siehe S. 41].
- [WIG17] K. WIGGERS. *How to use the brand-new Samsung Bixby voice assistant, and everything it can do - Bixby 2.0 brings improved speech recognition and third-party app support*. abgerufen am 15.02.2017 15:23 Uhr. 2017. URL: <https://www.digitaltrends.com/mobile/samsung-bixby-how-to-use/> [siehe S. 12].

Anhangsverzeichnis

A Quellcode	XIX
A.1 Response: Discovery	XIX

A. Quellcode

A.1. Response: Discovery

```
1  {
2      "header": {
3          "namespace": "Alexa.Discovery",
4          "name": "Discover.Response",
5          "payloadVersion": "3",
6          "messageId": "SomeRandomStraing"
7      },
8      "payload": {
9          "endpoints": [
10              {
11                  "endpointId": "wirelessSwitch1",
12                  "manufacturerName": "Raspberry Dude",
13                  "friendlyName": "Licht",
14                  "description": "Smart Device Switch",
15                  "displayCategories": [
16                      "LIGHT"
17                  ],
18                  "cookie": {
19                      "key1": "This is just an example!"
20                  },
21                  "capabilities": [
22                      {
23                          "type": "AlexaInterface",
24                          "interface": "Alexa",
25                          "version": "3"
26                      },
27                      {
28                          "interface": "Alexa.PowerController",
29                          "version": "3",

```

```
30         "type": "AlexaInterface",
31         "properties": {
32             "supported": [
33                 {
34                     "name": "powerState"
35                 }
36             ],
37             "retrievable": true
38         }
39     }
40 ]
41 },
42 {
43     "endpointId": "wirelessSwitch2",
44     "manufacturerName": "Raspberry Dude",
45     "friendlyName": "PC",
46     "description": "Smart Device Switch",
47     "displayCategories": [
48         "SWITCH"
49     ],
50     "cookie": {
51         "key1": "This is just an example!"
52     },
53     "capabilities": [
54         {
55             "type": "AlexaInterface",
56             "interface": "Alexa",
57             "version": "3"
58         },
59         {
60             "interface": "Alexa.PowerController",
61             "version": "3",
62             "type": "AlexaInterface",
63             "properties": {
64                 "supported": [
65                     {
66                         "name": "powerState"
67                     }
68                 ]
69             }
70         }
71     ]
72 }
```

```
68      ],
69      "retrievable": true
70    }
71  ]
72 },
73 {
74   "endpointId": "wirelessSwitch3",
75   "manufacturerName": "Raspberry Dude",
76   "friendlyName": "TV",
77   "description": "Smart Device Switch",
78   "displayCategories": [
79     "SWITCH"
80   ],
81   "cookie": {
82     "key1": "This is just an example!"
83   },
84   "capabilities": [
85     {
86       "type": "AlexaInterface",
87       "interface": "Alexa",
88       "version": "3"
89     },
90     {
91       "interface": "Alexa.PowerController",
92       "version": "3",
93       "type": "AlexaInterface",
94       "properties": {
95         "supported": [
96           {
97             "name": "powerState"
98           }
99         ],
100        "retrievable": true
101      }
102    }
103  ]
104}
105}
```

```
106      ]  
107    }  
108 }
```