

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

Inhaltsverzeichnis

Abbildungsverzeichnis	V
Quellcodeverzeichnis	VI
Tabellenverzeichnis	VII
Abkürzungsverzeichnis	VIII
1 Einleitung - erster Entwurf	1
2 Aufgabenstellung	2
3 Grundlagen	3
3.1 Smart Home	3
3.2 Sprachverarbeitung	3
3.3 Analyse und Vergleich von digitalen Assistenten	5
3.3.1 Amazon Alexa	7
3.3.2 Apple Siri	9
3.3.3 Google Assistant	10
3.3.4 Microsoft Cortana	12
3.3.5 Samsung Bixby	13
3.4 Raspberry Pi	14
3.5 Funkstandards	16
3.6 OAuth2	17
3.6.1 Rollen	17
3.6.2 Protokollablauf	18
3.6.3 Berechtigungsvergabe	19
3.6.4 Authorization Code Grant	20
3.6.5 Bearer Token	22
3.7 Development-Tools	22
3.7.1 Amazon Developer Services	22
3.7.2 Amazon Web Services - Lambda	23
3.7.3 Cloud9	23
3.7.4 Express.js	24
3.7.5 Ngrok	24

3.7.6 Node.js	24
3.8 Gebrauchstauglichkeit	25
4 Konzept	27
4.1 Anforderungsanalyse	27
4.1.1 Grund der Umsetzung	27
4.1.2 Funktionale Anforderungen	28
4.1.3 Nicht-Funktionale Anforderungen	29
4.2 Herangehensweise	30
4.3 Architektur	31
4.4 Design	32
4.4.1 Skill-Typen	34
4.4.2 Ansteuerung der Funksteckdosen	36
5 Implementierung	39
5.1 Einrichtung	39
5.1.1 Raspberry Pi	39
5.1.2 Python	40
5.1.3 Node.js	40
5.1.4 Cloud 9	40
5.1.5 Git	40
5.1.6 Amazon Developer Konto	40
5.1.7 Alexa	41
5.1.8 Amazon Web Services (AWS) Lambda	46
5.1.9 ngrok	47
5.2 Entwicklung	47
5.2.1 AWS Lambda	48
5.2.2 Raspberry Pi	64
6 Ergebnis, Fazit und Ausblick	70
Literatur	IX
Anhang	XVI

Abbildungsverzeichnis

Abb. 3.1: Raspberry Pi 3 Model B	15
Abb. 3.2: Abstrakter Protokollablauf [RF6749]	18
Abb. 3.3: Authorization Code Grant [RF6749]	20
Abb. 4.1: Raspberry Pi-Alexa-Kommunikation Vereinfacht	32
Abb. 4.2: Raspberry Pi-Alexa-Kommunikation	32
Abb. 4.3: Verwendetes Funkset mit Hausschlüssel	37
Abb. 5.1: Setup Skill-Information	41
Abb. 5.2: Setup Interaction Modell	42
Abb. 5.3: Setup Configuration Endpoints	43
Abb. 5.4: Setup Configuration Account Linking	44
Abb. 5.5: Raspberry Pi-Alexa-Kommunikation Schritt 1	45
Abb. 5.6: Setup Test	46
Abb. 5.7: Raspberry Pi-Alexa-Kommunikation Datenfluss	48
Abb. 5.8: Flussdiagramm Alexa nach Raspberry Pi	63
Abb. 5.9: Webinterface	67

Quellcodeverzeichnis

5.1	Node.js Installieren	40
5.2	GitHub Repository	40
5.3	Export Handler	48
5.4	Request: Report-State	51
5.5	Response: Report-State	54
5.6	Request: Power-Controller	55
5.7	Response: Power-Controller	56
5.8	Request: Discovery	57
5.9	Request: Authorization	59
5.10	Response: Authorization	59
5.11	Benutzerkontrolle	60
5.12	Benutzerinformationen	61
5.13	Node app.js: Webinterface	66
5.14	Node app.js: Report-State	66
5.15	Node.js-Funktion: Erstellen und Senden des Funksignals	67

Tabellenverzeichnis

3.1 Formaler Vergleich der aktuell meist verbreitetsten Sprachassistenten . . . 6

Abkürzungsverzeichnis

API Application Programming Interface

ARN Amazon Resource Name

AVS Alexa Voice Service

AWS Amazon Web Services

CPU Central Processing Unit

Dip Dual In-line Package

GPIO General Purpose Input/Output

HDMI High-Definition Multimedia Interface

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

JSON JavaScript Object Notation

LED lichtemittierende Diode

LWA Login With Amazon

NPM Node Package Manager

SSH Secure Shell

URI Uniform Resource Identifier

USB Universal Serial Bus

WLAN Wireless Local Area Network

1. Einleitung - erster Entwurf

Wir leben mittlerweile 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 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 ab Werk 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 haben wir es uns als 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 Sprachassistentens und eines Raspberry Pis zu entwickeln. Das Hauptaugenmerk liegt dabei das Repertoire des schon bestehenden Alexa-Systems um weitere Haushaltsgeräte zu erweitern. Diese besitzen entweder keine Möglichkeit einer Anbindung an Alexa (passender Alexa Skill) oder haben teilweise gar nicht die benötigte technische Infrastruktur, wie z. B. WLAN oder Rechenkapazität. Diese Anbindung erfolgt dann über den Raspberry Pi, der für diesen Zweck konfiguriert und angeschlossen wird. Die Mindestanforderung an das Projekt ist die Inbetriebnahme einer Funksteckdose, die dann 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 dann ein Kosten- und Aufwandsvergleich zwischen der eigens konfigurierten und implementierten Variante sowie der teuren, dazu zu erwerbenden 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 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
- effiziente Energienutzung [DEV17]

Die gesamte Haushaltstechnik wird somit vernetzt und kann individuell angepasst und automatisiert werden. So können z. B. 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. [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 Interaktion mit Maschinen so gut wie unumgänglich. Der herkömmliche Weg über Tastatur, Maus und Touchscreen ist für den Menschen immer noch unzureichend, denn das 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 nicht in den nächsten beiden Jahrzehnten 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, zu simulieren. [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 zum Beispiel 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 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. Diese Aufnahmen ließen sie durch verschiedene Spracherkennungssoftware laufen, um somit die Fehlerquote bei der automatisierten Übersetzung der Sprachaufnahmen 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 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 Sprachsuchen 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, auf welcher Anzahl an Plattformen die jeweilige Software zur Verfügung steht.
- Allgemeinwissen - Wichtig für diesen Punkt ist die Fähigkeit, auf Wissenfragen 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 rein.
- Übersetzungsfähigkeit - 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 allgemein sehr interessant, ist 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, denn hier wird verglichen, welche Assistenten Smart Home überhaupt unterstützen und wie gut dies geschieht. 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 die politische Korrektheit in der heutigen Zeit wurde dieser Punkt mit in dazugenommen. Hier wird angegeben, welche Software auch anbietet eine Stimme des männlichen Geschlechts auszuwählen.

In der folgenden Tabelle werden anhand der beschriebenen Merkmale die Assistenten Amazon Alexa [ALEXAa], Apple Siri [SIRI], Google Assistant [GOASS], Microsoft Cortana [MICOR] und Samsung Bixby [BIXBY] miteinander verglichen. 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	✓✓	✓	✓✓✓✓	✓	✓✓
Übersetzungs-fähigkeit	✓	✓	✓✓	✓✓✓✓	✗
Funktionsumfang	✓✓✓	✓✓✓	✓✓✓✓	✓✓	✓✓✓✓
Erweiterbarkeit	✓✓✓	✓	✓✓✓✓	✓✓	✗
Smart Home Unterstützung	✓✓✓	✓✓	✓✓✓✓	✗	✗
Gender-Correctness	✗	✓✓	✗	✗	✓✓

Tabelle 3.1.: Formaler Vergleich der aktuell meist verbreitetsten Sprachassistenten

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, 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 die zugehörige App von Nöten ist. [BLE17, S. 81] Seit neuestem 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 16000 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 Erfahrung 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 erreichbar wäre. [BAG17, S. 66]

Amazon hat somit ein Produkt geschaffen, dass dem Benutzer das Leben vereinfachen kann und auch 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. Beispielsweise 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 Apps 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 anzeigen lassen. Weiterhin können Emails oder Notizen diktiert, abgespeichert oder versendet werden. Mittlerweile gibt es auch eine ausgereifte Unterstützung für häufig genutzte nicht-Apple Apps, 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 wurden und dies ist auch sehr gut gelungen. 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 minder 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]

Ähnlich wie mit den Alexa-Skills und die 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 Assistant 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 Restaurantvorschläge in der Nähe funktionieren einwandfrei. Auf Anfragen, wie örtliches Kinoprogramm, können 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 manchen 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 der Sprachassistent von Google 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 Google 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 Google Assistant auch im Allgemeinwissen marktführend. Er kann z. B. bei

ortsbezogenen Informationen, wie Kinoprogramm oder auch Navigation, die Konkurrenz ausstechen. Die Entscheidung, welche der Assistent über die Form der Ausgabe fällt, also die mündliche oder visuelle Ausgabe des Ergebnisses, 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 mehrere Benutzer differenziert bedienen. [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 Assitant auch in der aktuellen Version 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 genoss. 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 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 Windows 10 Betriebssystem 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 Googles 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 das auch nur auf Englisch und somit bisher nicht für den deutschsprachigen Raum bestimmt. [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 und 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.

3.4. 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, über die Jahre ist mittlerweile riesig geworden und umfasst Dinge, wie ein einfache low Budget Desktop PC, eine retro Video Spielkonsole, selbstgebaute Router und eigene personalisiertes Smart Home, was auch in dieser Studienarbeit behandelt wird. 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]

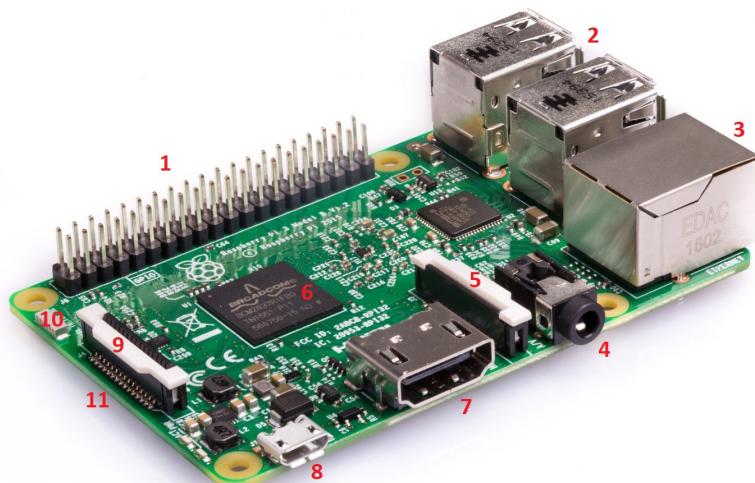


Abb. 3.1.: Raspberry Pi 3 Model B

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 kann.

An der schmalen Seite befinden sich 4 USB-Anschlüsse (2), welche einen Datentransfer von 480 Mb/s gewährleisten.

Direkt daneben ist der 10/100 Ethernet-Port (3) platziert, welcher eine Datenrate von 100Mb/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 150Mb/s schafft, Weiterhin gibt es eine kombinierte 3,5mm 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 2,5 Ampere ausgelegt.

Auf der Unterseite der Platine unter dem Camera Interface befindet sich der MicroSD-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 System 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 samte Raspberry Pi, Funkmodul, Netzteil und LAN-Kabel.

— fancy Bild —

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 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 Dingen wie Garagentoröffnern oder Autoschlüsseln, da der Aktivierungsradius (kleiner 100m) sehr gering und die Nutzungsdauer kurz ist (0,1 - 3 Sekunden). 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 Funkfernbedienungen, 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. Das ist dann aber auch mit erhöhten Kosten verbunden. [RNW07]

Bei dem Beispiel der Funksteckdose kann der Benutzer somit 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 gewährt kann. Meistens ist dies eine Person als Endbenutzer, welche Zugriff auf Teile ihres Accounts gewährt.

3.6.2. Protokollablauf

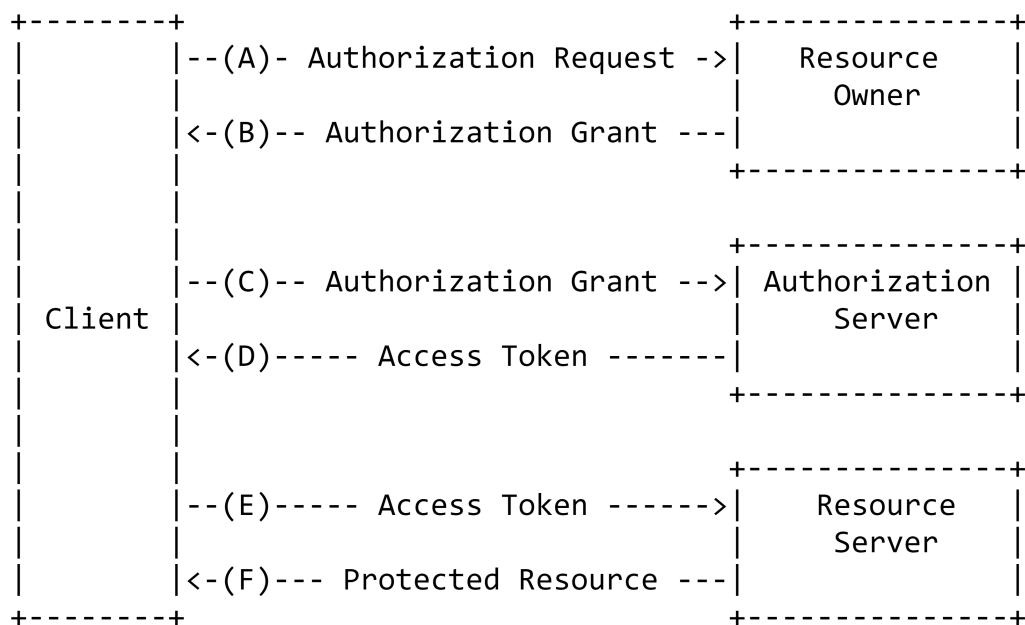


Abb. 3.2.: Abstrakter Protokollablauf [RF6749]

Der in Abbildung 3.2 dargestellte OAuth2 Protokollablauf beschreibt die Interaktion zwischen den vier Rollen und umfasst die folgende 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 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 Ressourceneigentümer zu erhalten (dargestellt durch die Schritte (A) und (B) in Abbildung 3.2), 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)

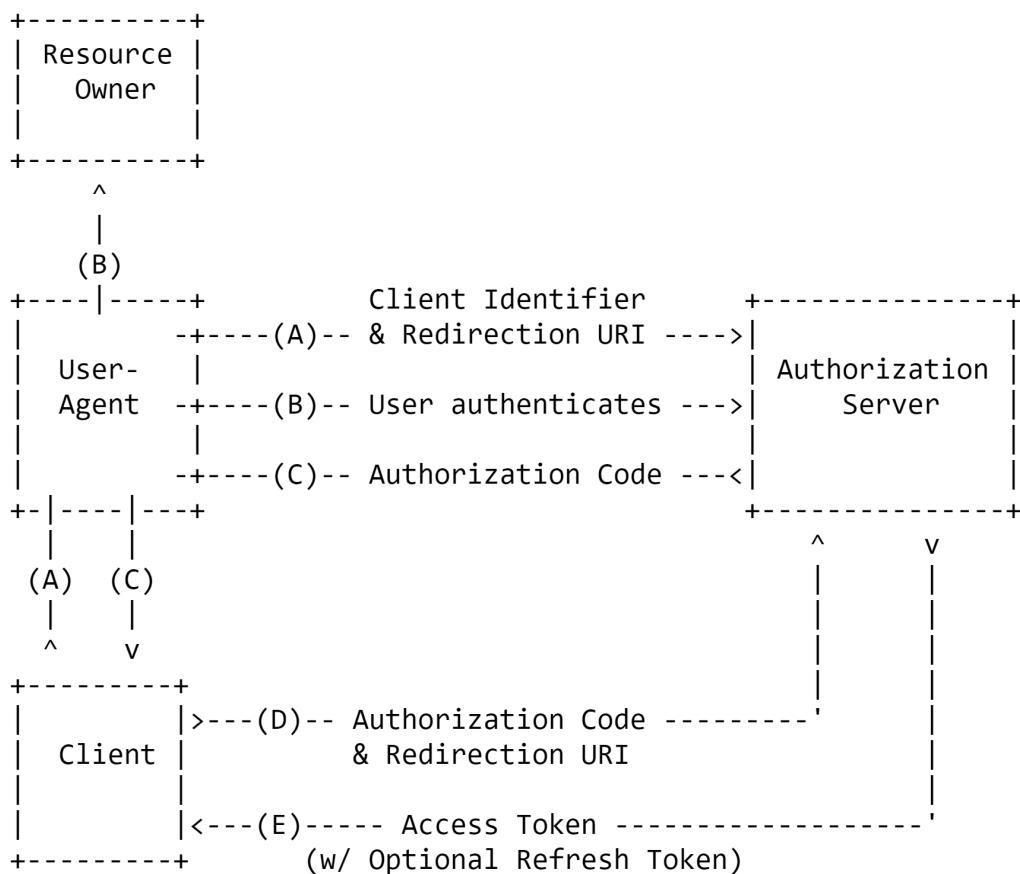


Abb. 3.3.: Authorization Code Grant [RF6749]

Das in Abbildung 3.3 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 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 beispielsweise 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 Drittanbieter stammenden Skills kann Alexa auf bestimmte eigens konzipierte Anwendungsfälle, durch Erkennung bestimmter Wörter, erkennen und auf diese reagieren. Diese selbsterstellten 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, Entwickertools, Verwaltungs-, Analyse- und Überwachungstools und noch mehr [AMA18g].

Einer dieser Cloud-basierten Services ist der AWS Lambda, er 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 Lamda 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 5.1.4), 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 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 3.7.6) 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. Ngrok

Ngrok ist eine Anwendung mit Webservice zum tunneln von öffentlichen, aus dem Internet zugreifbaren, ngrok URLs auf den PC eigenen localhost. Ngrok erleichtert einem 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. Wer mehr möchte muss natürlich zahlen. [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.

3.7.6. Node.js

Node.js ist eine quelloffene, der MIT License unterliegende, serverseitige Plattform für diverse Netzwerkanwendungen und Webserver mit asynchroner, ereignisgesteuerter Ja-

vaScript Laufzeitumgebung. Node.js 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 weitverbreitete 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, wobei sich die Berechnung nur auf 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 selbstkreirten 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 somit die eigentliche Preisersparnis, da dadurch die Gesamtkosten sich nur auf 60-70€ belaufen.

In diesem Preis noch nicht mit einberechnet sind die Kosten für das Mikrofon, welches am Raspberry Pi benötigt wird. Ein 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, wo die gewünschte Klangqualität liegt, 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 ein Überlegung wert, nicht direkt auch einen Echo-Lautsprecher zurückgreift, 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. Es sollen verschiedene Einsatzmöglichkeiten für die Kamera geschaffen werden, wie Zeitraffer-Aufnahmen, Überwachungskamera oder stationärer Fotoapparat mit Fernsteuerung über Sprachbefehl und Webinterface.

Die Funksteckdosen, die Kamera und eventuell weitere Geräte sollen neben Sprachbefehlen

auch über eine webbasierte Hauszentralsteuerung erreichbar sein. Diese Hauszentralsteuerung muss von beliebigen und 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.

(- Ansteuern von Geräten mittels Infrarotsender)

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 Raspberry Pi verbundenen Geräten.

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

Performance

Das System soll direkt auf Sprachkommandos 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 erfolgt dann die Grundlagenrecherche und Auseinandersetzung mit für das Projekt zur Verfügung stehende Hard- und Software. Nach Auswahl der Werkzeuge erfolgt dann die Ausarbeitung der Architektur und des Design.

Nach den theoretischen Phasen der Planung und Recherche folgt dann die Phase der Einrichtung. Hierunter fällt die Beschaffung des Raspberry Pi sowie der restlichen Kompo-

nenten. 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 Funk vom Raspberry Pi ein- und ausgeschaltet werden, per Sprache zu steuern. Dafür wird ein Python-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 repräsentiert von dem Alexa Voice Service und der Heimzentralsteuerung im Webbrower, 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 vertreten vom Raspberry Pi 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 über ein neues Alexa Skill [ALEXAb], welches 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:

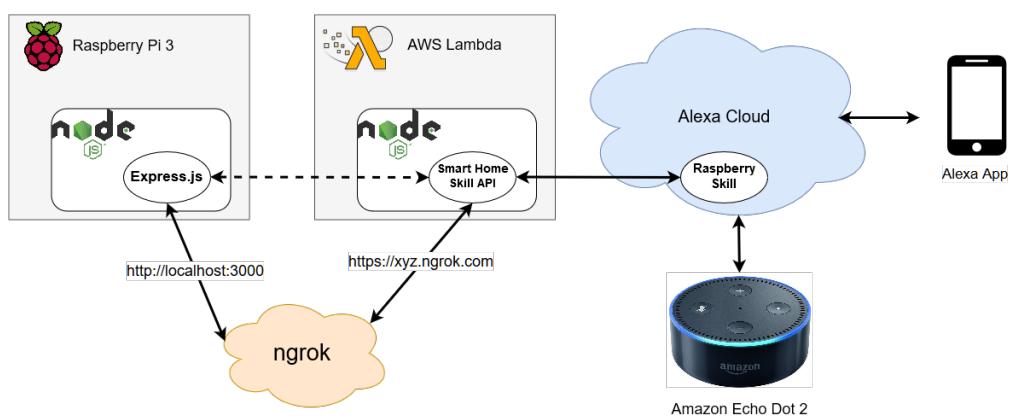


Abb. 4.2.: Raspberry Pi-Alexa-Kommunikation

Um einen Smart Home-Skill (siehe: Abschnitt 4.4.1) mit Amazon's 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 Rasberry 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, welche Direktiven von der Alexa Cloud entgegennehmen und verarbeiten kann (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 unser 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 mit 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 weißt dem Raspberry Pi nach Verbindungsauftbau 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 installierter Alexa App, installiert und dann mit einem Amazon Konto verbunden werden. Für die Verknüpfung mit dem Amazon Konto 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 per-

sonalisierten 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.

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, wird in einer AWS Lambda Funktion bereitgestellt (Siehe 5.1.8 Amazon Web Services (AWS) Lambda).

Zu beachten ist, dass die API nur auf von ihr unterstützten 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 Gerät das Video (Smart TV, Echo Show) wiedergegeben wird. [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 darauf dann reagieren. Die Erstellung dieses Types erfolgt durch das Alexa Skills Kit Command Line Interface, welches die größte Arbeit abnimmt.

4.4.2. Ansteuerung der Funksteckdosen

In dieser Studienarbeit wurden Funksteckdosen der Marke Brennenstuhl verwendet. In der folgenden Abbildung ist sowohl die Fernbedienung (links) als auch eine Steckdose mit dem verwendeten Hausschlüssel (rechts) abgebildet, welcher bereits in den Grundlagen unter 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 Skript-Aufrufe 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 Geräteansteuerung 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 Dipschalterbelegung des Hausschlüssels der jeweilige Systemcode übergeben. In den fünf darauffolgenden 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 Geräteansteuerung.

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.

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 MicroSD-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 MicroSD-Karte) sowie ein Image zur Installation ausgewählt und die Installation gestartet werden. Sobald Etcher das Betriebssystem auf dem Raspberry Pi installiert hat, kann die MicroSD-Card in den Raspberry Pi gesteckt werden und von dort direkt gebootet werden.

Der Raspberry Pi kann entweder über direkt angeschlossene Peripheriegeräte bedient werden oder per Remote. 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.

Hier folgt demnächst ein Bild vom Raspberry Pi, welcher für dieses Projekt benutzt wurde.

5.1.2. Python

[folgt]

5.1.3. Node.js

Für das Projekt wird Node.js in Version 9.x verwendet. Um das aktuellste Node.js v9.x zu installieren führt man folgende Befehle aus: [NODEb]

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

Quellcode 5.1: Node.js Installieren

5.1.4. 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.

5.1.5. Git

Zur Sicherung und Versionierung wurde ein GitHub Repository eingerichtet.

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

Quellcode 5.2: GitHub Repository

5.1.6. Amazon Developer Konto

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

5.1.7. Alexa

Zunächst einmal müssen im ersten Schritt unter Skill Information grundlegende Informationen über den Skill festgelegt werden, welche die weiteren Schritte und deren Komplexität beeinflussen.

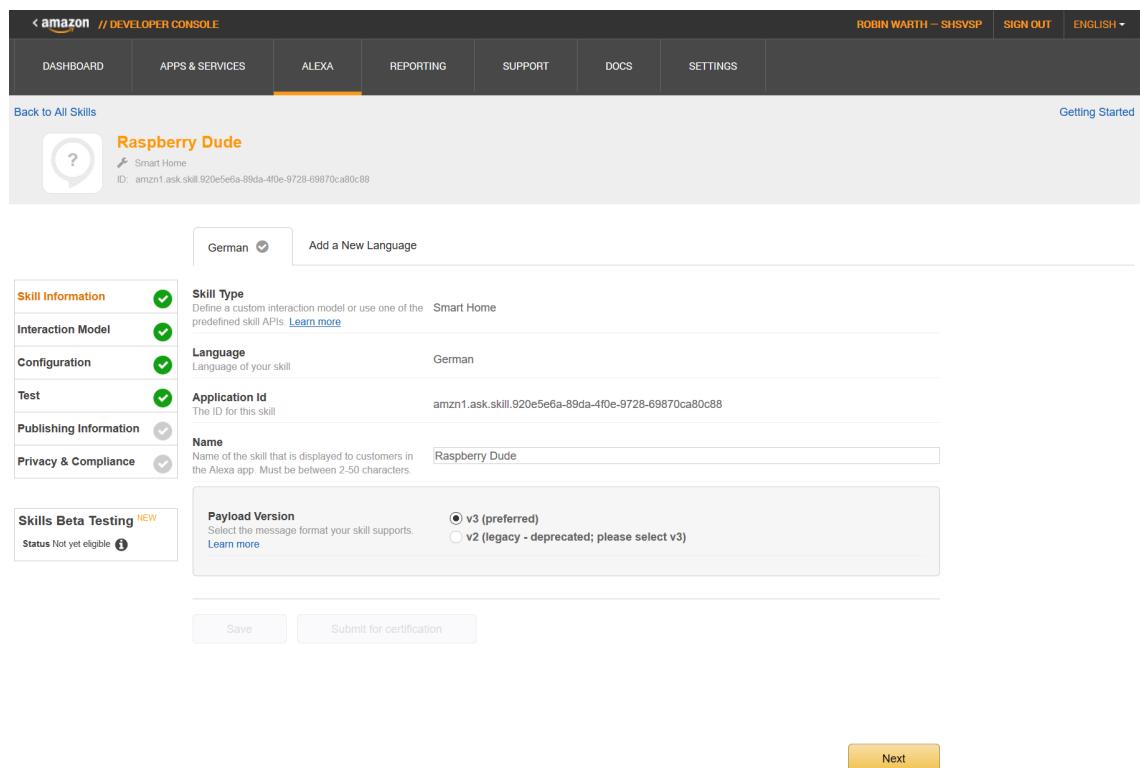


Abb. 5.1.: Setup Skill-Information

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. [Verweis auf das Kapitel] 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 man sie nicht einmal auswählen kann. Sie ist nur für ältere schon erstellte Skills noch aufgeführt

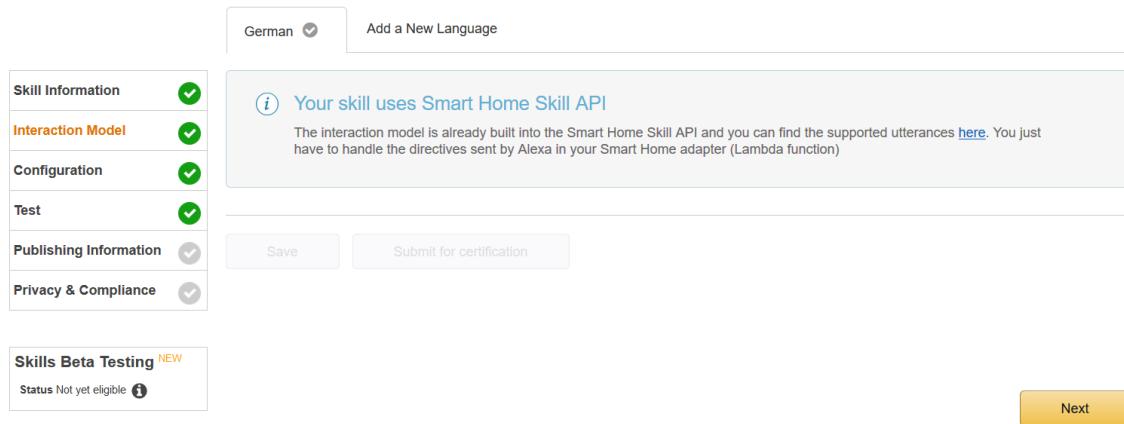


Abb. 5.2.: Setup Interaction Modell

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 der von Alexa gesendeten Direktiven geeignet verarbeiten kann. Da hier die Smart Home Skill API zum Einsatz kommt, wird man von Amazon dazu verpflichtet einen Amazon Web Services (AWS) Lambda Server zu nutzen. Zu dem Aufbau und der Einrichtung des AWS Lambda Server wird in Abschnitt [bla bla keine Ahnung, AWS Lambda] noch gezielt eingegangen.

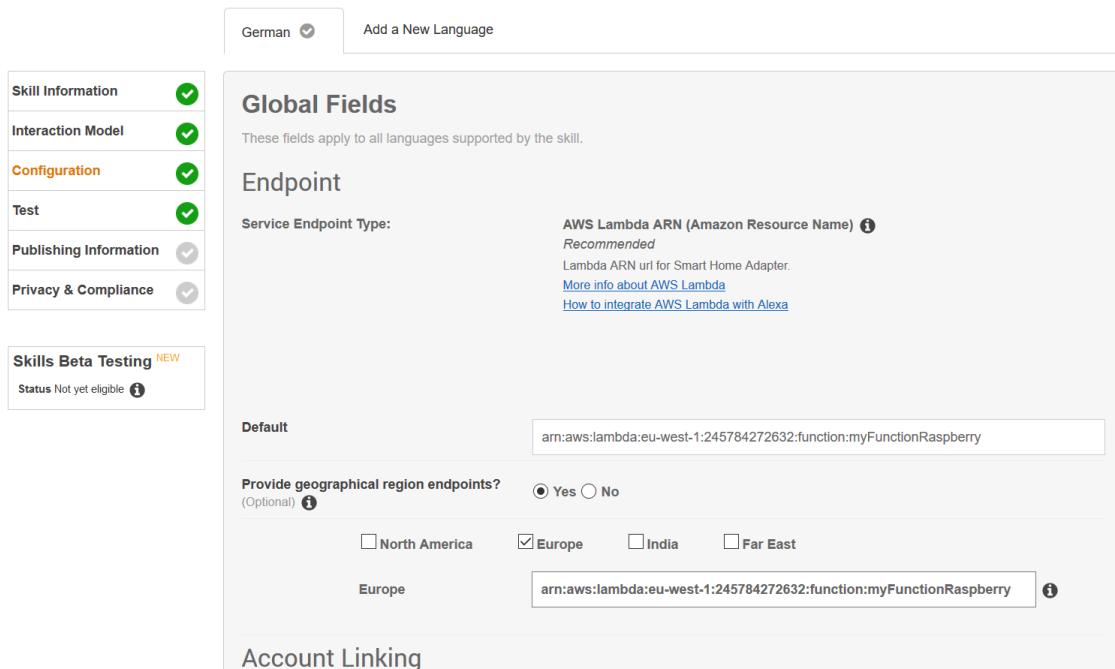


Abb. 5.3.: Setup Configuration Endpoints

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 Alexa-Amazon-Account und dem Account beim Hersteller der Smart Home Geräte, wie z. B. Lampen, geschaffen.

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

lungen hier sind 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.

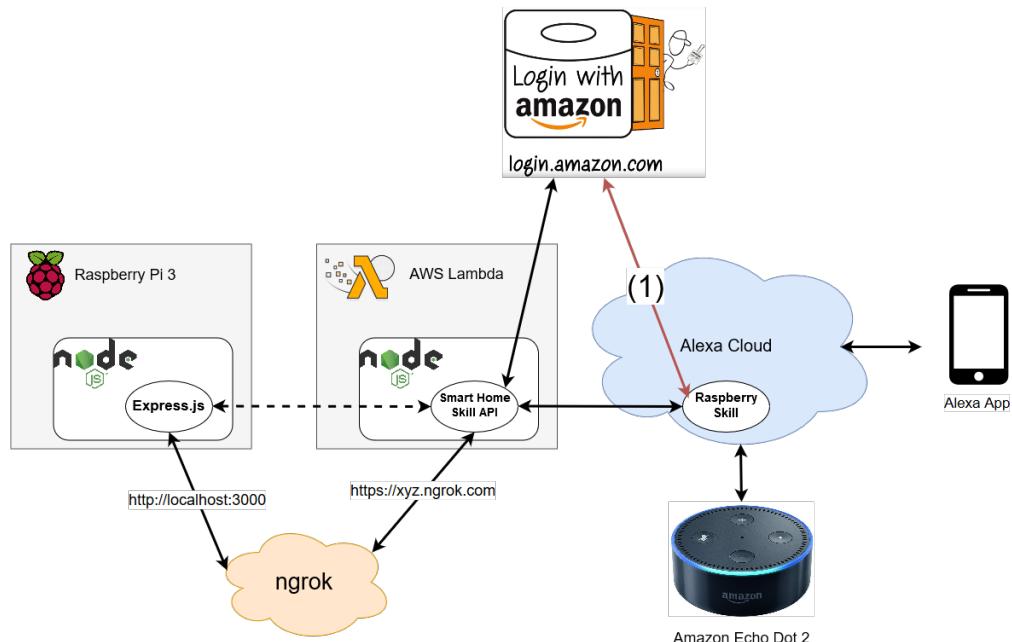


Abb. 5.5.: Raspberry Pi-Alexa-Kommunikation Schritt 1

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.

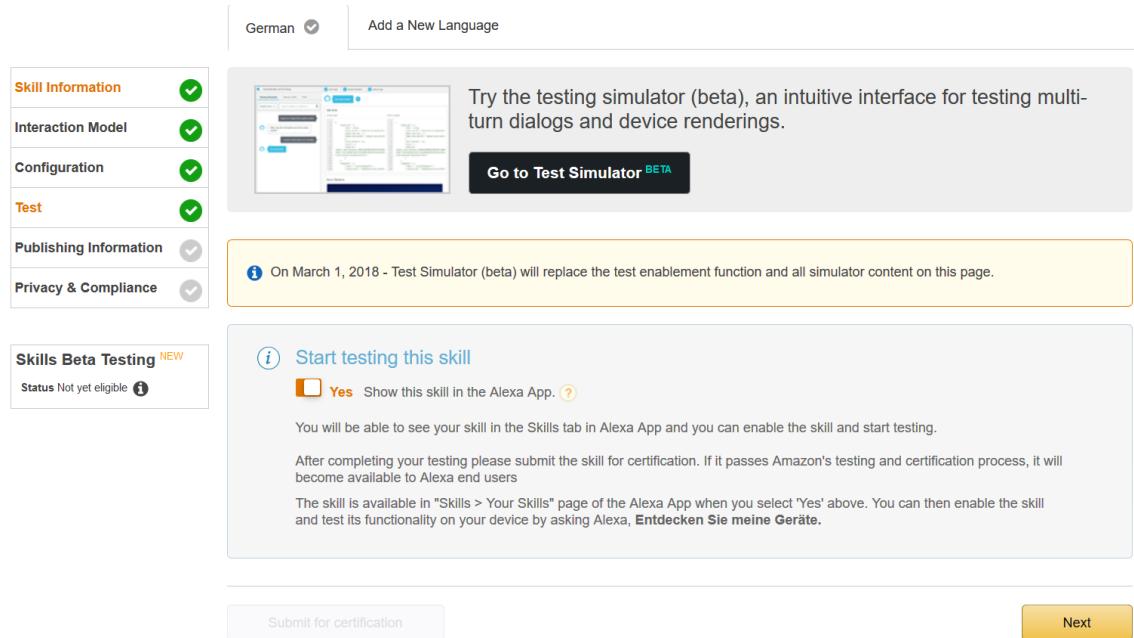


Abb. 5.6.: Setup Test

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 für AWS erstellt und ein AWS Lambda 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 Alexa Skill wird die Skill-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 Project 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 Lamda 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**

[folgt...]

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.

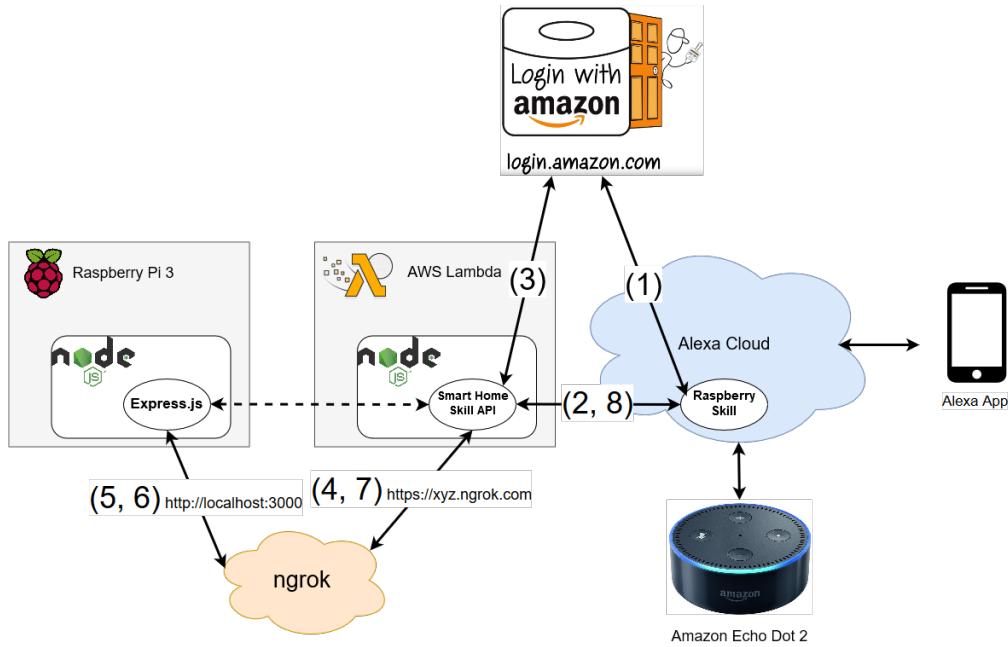


Abb. 5.7.: Raspberry Pi-Alexa-Kommunikation Datenfluss

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 Erlangen 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 export-handler werden die verschiedenen Typen an Direktiven zunächst einmal anhand von dem „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  else {
23    console.log("DEBUG:", "Unknown Directive", JSON.stringify(
24      ↪ request));
25  }
26};
```

Quellcode 5.3: Export Handler

Direktiven und Events

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

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 Trackingzwecke, 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 wird.

- **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 ertellt 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) welche 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, anstatt im header, 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 ReportState-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. Hier ein Beispiel für eine ReportState-Direktive, welche nach dem derzeitigen Status des Geräts (Endpunkt) mit der ID „wirelessSwitch1“ frägt:

```
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    }
```

Quellcode 5.4: Request: Report-State

Die Direktiven von ReportState (siehe Quellcode xyz) und PowerController (siehe Quellcode xyz) 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 namespace bei einer PowerController Direktive „Alexa.PowerController“. Das name Attribut ist ebenfalls unterschiedlich. Bei einer ReportState Direktive lautet der Name immer „ReportState“, während bei einer PowerController Direktive zwischen „TurnOn“ und „TurnOff“ unterschieden wird. 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, welche 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 soeben erwähnt, haben die Event-JSONs einen anderen Aufbau als die der Direktiven. Es folgt nun eine Beschreibung des Aufbaus, geltend für ReportState und PowerController, in ähnlicher Form wie die Beschreibung der Direktiven [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 (Eigenschaftsobjekten) und deren Zustand:

- namespace: Ein String, der die Kategorie der Nachricht und der Nutzdaten angibt.
- name: Der Name der Eigenschaft.
- value: Ein Einzelner 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 60000 (in Millisekunden).
- event

Beschreibungen über das Event:

- header

Analog zu header von Direktiven:

- messageId: Ein eindeutiger Identifikator für eine einzelne Anfrage oder Antwort. Dieses Attribut ist wichtig für Trackingzwecke, 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 enthält ein correlationToken, damit der Kontext und Grund der Antwort ersichtlich wird. 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.
- namespace: Ein String, der die Kategorie der Nachricht und der Nutzdaten angibt.
- name: Der Name des Events wie z. B. StateReport (Statusbericht).
- payloadVersion: Die Version der Schnittstelle, an welche diese Nachricht gesendet wird.
- endpoint

Analog zu endpoint von Direktiven:

- scope: ...
- endpointId: ...
- cookie: ...
- 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                 "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 5.5: Response: Report-State

Power Controller Über den PowerController wird versucht, aktiv einen Gerätestatus zu verändern, wenn z. B. wenn 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 5.6: Request: Power-Controller

Die in der Antwort (Response/Event, siehe Quellcode: xy z) stehenden properties Daten, stammen direkt vom Server des Raspberry Pi. Die properties und damit verbundene Gerätezustände 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     "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 5.7: Response: Power-Controller

Discovery Über die Discovery-Direktive frägt 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 erkennt man wieder, analog zu den vorherigen Direktiven, anhand des namespace „Alexa.Discovery“ und am Name „Discovery“.

```

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 5.8: 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 des von Alexa angefragten Events (siehe Quellcode: xyz) 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 Nutzungszenarien, auch eine abweichende Struktur besitzen. Die hier erläuterte Struktur orientiert sich direkt am Einsatzszenario dieser Studienarbeit und besitzt demzufolge nur endpoints 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 Gerätebezeichnung

- 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 beispielsweise kaum ähnliche Fähigkeiten verglichen zu einer Lampe 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 ReportState Direktiven) abgerufen werden können.

Hier ist deine Referenz auf den Anhang: A.1 Response: Discovery

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, während dieser den Account-Linking Prozess durchführt.

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, welchen man bei Bedarf gegen ein Access- (Bearer-) oder Refresh-Token eintauschen 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 5.9: 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 5.10: 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 etwas 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 im Quellcode 5.11 durchgeführt, mithilfe einer einfachen Fallunterscheidung. 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  }
13}
```

```

9      else {
10         callbackUserHost('xyz.eu.ngrok.io', 'test:test');
11     }
12   );
13 };

```

Quellcode 5.11: 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 5.11 die Funktion „obtainCustomerProfileInformation“ (siehe Quellcode 5.12) 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 };

```

```
20 |     req.on('error', (e) => {
21 |         console.error(`RESPONSE-Error Obtain Customer Profile
22 |             ↪ Information: problem with request: \${e.message}`);
23 |     });
24 |     req.end();
25 | };
```

Quellcode 5.12: Benutzerinformationen

Quellcode 5.12 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 5.11) 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 (siehe Abschnitt 5.2.2) werden soll. Dieser Abschnitt stellt die Schritte eins bis fünf der Grafik 5.7 detaillierter dar.

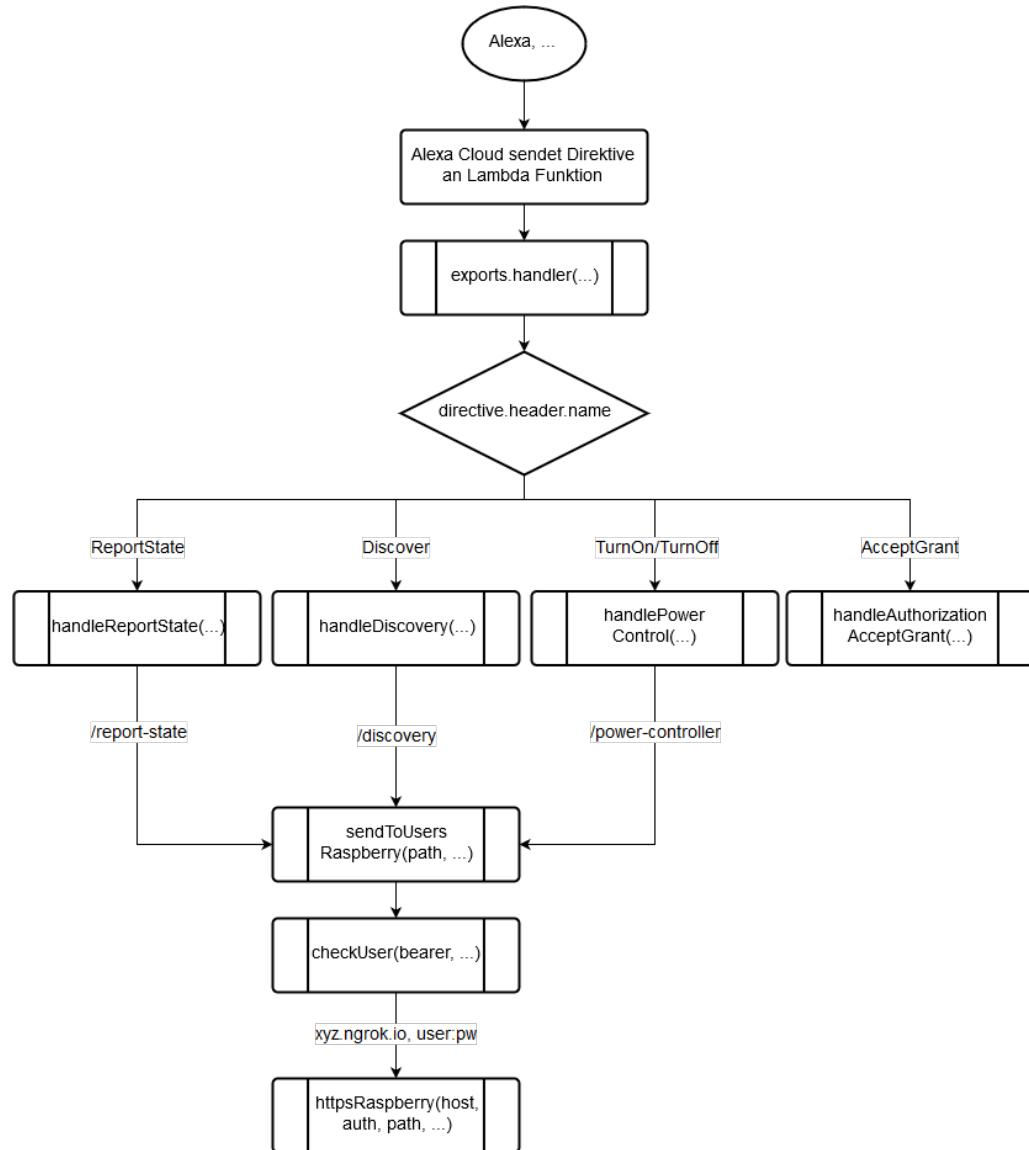


Abb. 5.8.: Flussdiagramm Alexa nach Raspberry Pi

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 die selben 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()“. Beim Aufruf dieser Funktion 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 aufgeteilt in JavaScript Code, in der Node.js Laufzeitumgebung, für die Implementierung des Webservers, zusätzlicher HTML, CSS und etwas JavaScript Code für die Erstellung des Webinterfaces, sowie wieder JavaScript Code zur Geräteansteuerung über Node.js. 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 5.13 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).
- „/discovery“: 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 5.14 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 5.13):

```

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 5.13: 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 man in Quellcode 5.14 sehen kann, 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 5.14: 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 Python Scripts und der einfachen Ansteuerung der Funksteckdosen ohne Alexa. Z. B. können hiermit Tests ausgeführt werden, welche Alexa als Fehlerquelle ausschließen können. 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

Geräteansteuerung

Das Node.js-Skript 5.15 erstellt anhand der überlieferten Daten des Hausschlüssels und des Gerätecodes 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, welches in diesem Projekt eingebunden ist. 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    }
29    this.output.writeSync(0);
30    for (let i = 0; i < this.repeat; i++) {
31        for (let b of bangs) {
32            this.output.writeSync(b);
33            sleep.usleep(this.pulseLength);
34        }
35    }
}

```

Quellcode 5.15: 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 5.15) 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 5.15 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.

6. Ergebnis, Fazit und Ausblick

[folgt]

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. 6].
- [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. 32].
- [AMA18a] AMAZON.COM. *Amazon Developer*. abgerufen am 27.02.2018 12:23 Uhr. 2018. URL: <https://developer.amazon.com/de/> [siehe S. 40].
- [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. 45].
- [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. 45].
- [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. 46].
- [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. 50, 52].
- [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. 34–36].

- [AMA18g] AMAZON.COM. *Amazon Web Services*. abgerufen am 21.03.2018 13:45 Uhr. 2018. URL: <https://aws.amazon.com/de/> [siehe S. 23].
- [AMA18h] AMAZON.COM. *AWS Lambda*. abgerufen am 21.03.2018 13:47 Uhr. 2018. URL: <https://aws.amazon.com/de/lambda/features/> [siehe S. 23].
- [AMA18i] AMAZON.COM. *Serverlose Datenverarbeitung*. abgerufen am 21.03.2018 14:09 Uhr. 2018. URL: <https://aws.amazon.com/de/serverless/> [siehe S. 23].
- [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. 23].
- [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. 57].
- [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. 59].
- [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. 7–10, 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. 6].
- [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. 7–12].
- [BSTRAP] GETBOOTSTRAP.COM. *Bootstrap*. abgerufen am 26.03.2018 14:22 Uhr. 2018. URL: <https://getbootstrap.com/> [siehe S. 66].

- [C9.io] C9.io. *C9.io Github Repository*. abgerufen am 27.02.2018 11:57 Uhr. 2018. URL: <https://github.com/c9> [siehe S. 24, 40].
- [C9AWS] AMAZON.COM. *AWS Cloud9*. abgerufen am 21.03.2018 10:55 Uhr. 2018. URL: <https://aws.amazon.com/de/cloud9/> [siehe S. 23].
- [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, 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. 32].
- [ETC17] RESIN.IO. *Etcher Download*. abgerufen am 27.10.2018 11:35 Uhr. 2018. URL: <https://etcher.io/> [siehe S. 39].
- [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. 24].
- [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. 6].
- [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. 14].
- [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. 14].

- [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. 67].
- [ISO06] ISO 9241: *Ergonomie der Mensch-System-Interaktion - Teil 110: Grundsätze der Dialoggestaltung*. Standard. 2006 [siehe S. 25].
- [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. 40].
- [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. 6].
- [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. 24].
- [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. 32].
- [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. 40].
- [NODEc] NODE.JS FOUNDATION. *Node.js*. abgerufen am 21.03.2018 11:36 Uhr. 2018. URL: <https://nodejs.org/en/> [siehe S. 25].
- [NODEd] NPM. *onoff -npm*. abgerufen am 02.04.2018 15:15 Uhr. 2018. URL: <https://www.npmjs.com/package/onoff> [siehe S. 38, 67].
- [NODEe] NPM. *sleep -npm*. abgerufen am 02.04.2018 19:42 Uhr. 2018. URL: <https://www.npmjs.com/package/sleep> [siehe S. 69].
- [OAUTHa] IETF OAuth Working GROUP. *OAuth 2.0*. abgerufen am 27.02.2018 12:27 Uhr. 2018. URL: <https://oauth.net/2/> [siehe S. 17, 43].

- [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. 17].
- [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. 39].
- [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. 32].
- [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. 39].
- [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. 14].
- [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, 14].
- [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. 17, 18, 20–22].
- [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. 22].

- [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. 16, 17].
- [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. 22].
- [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. 22].
- [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. 39].
- [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. 16].
- [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/2017/10/59e781abe09a0021036803.jpeg> [siehe S. 6].
- [SKDOC] Applce 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. 14, 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. 39].
- [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. 13].

Anhangsverzeichnis

A Quellcode	XVII
A.1 Response: Discovery XVII

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 }
```