Dash

STUDIENARBEIT

des Studienganges Angewandte Informatik

an der Dualen Hochschule Baden-Württemberg Stuttgart

von

Tobias Fürtjes, Lennart Roger Jerker Gustafsson, Kai Fabian Oswald

13. Mai 2017

Bearbeitungszeitraum	x Wochen
Matrikelnummern, Kurs	5335024, TINF-14C
Betreuer	Betreuer
Gutachter der Dualen Hochschule	Titel Vorname Nachname

Ehrenwörtliche Erklärung

Erklärung

gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“
vom 22. September 2011.

Ich habe die vorliegende Arbeit selbstständig verfasst und keine anderen
als die angegebenen Quellen und Hilfsmittel verwendet.

(Datum, Ort)	(Unterschrift)
(Datum, Ort)	(Unterschrift)
(Datum, Ort)	(Unterschrift)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

ARP Address Resolution Protocol

GPIO General Purpose Input/Output

IoT Internet of Things

IDE Integrated Development Environment

IP Internet Protokoll

JSON JavaScript Object Notation

MAC Media-Access-Control

OSI Open Systems Interconnection

PHP PHP: Hypertext Preprocessor

REST Representational State Transfer

SQL Structured Query Language

TCP Transmission Control Protocol

UDP User Datagram Protocol

1 Einleitung

In den letzten Jahren konnten aufgrund technologischer Entwicklungen immer mehr Geräte entwickelt werden, die in den verschiedensten Bereichen des täglichen Lebens eingesetzt werden. Dabei sind die Geräte meist leistungsfähiger oder kleiner geworden, sodass sie in unterschiedlichsten Bereichen eingesetzt werden können. Ein Begriff der in den letzten Jahren für dieses Phänomen immer häufiger genutzt wurde, ist das "Internet of Things (IoT)" zu deutsch "Das Internet der Dinge". Mit dem IoT ist gemeint, dass immer mehr Geräte mit dem Internet verbunden sind und Daten austauschen. Im Rahmen dieser Entwicklung sind verschiedenste Nutzungsszenarien umgesetzt worden, die das tägliche Leben erleichtern oder automatisieren sollen. Diese Studienarbeit wird sich im folgenden mit einem speziellen Szenario befassen, welches unter anderem durch den Onlinehändler Amazon umgesetzt wurde.

Das Szenario umfasst sogenannte "Dash' Buttons". Diese Buttons sind kleine Geräte, welche an unterschiedlichen Stellen in der Wohnung des Kundens angebracht werden können und über einen Druckknopf verfügen. Nach einer Konfiguration durch den Benutzer kann mithilfe eines Drucks auf den Sensor ein zuvor ausgewähltes Produkt bestellt werden. Ein Beispiel wäre das nachbestellen von Waschpulver. So könnte der Button an der Waschmaschine befestigt sein und bei einem geringen Vorrat an Waschpulver wird der Button betätigt und das ausgewählte Waschpulver innerhalb der nächsten Tage geliefert. So entfällt für den Nutzer der händische Bestellvorgang, da dieser automatisch durch den Button und Amazon durchgeführt wird. Dieses Beispiel lässt sich natürlich auf verschiedene andere Möglichkeiten übertragen, so könnten auch Dinge, wie Kaffee. Zahnpasta, Tierfutter, Getränke oder ähnliches nachbestellt werden. Dem Nutzer wird eine Vereinfachung des Kaufprozesses versprochen und der Händler bindet den Kunden stärker an sich, da der Button direkt bei Amazon bestellt. Die Studienarbeit setzt sich zum Ziel eine Untersuchung des Amazonproduktes durchzuführen und eine Alternative zu entwickeln.

2 Aufgabenstellung

Wie bereits in der Einleitung erwähnt, soll sich diese Studienarbeit mit der Untersuchung des Amazon Dash Buttons beschäftigen und eine Alternative entwickeln. Da bereits während der Recherche vor Projektbeginn herausgefunden werden konnte, dass der Button in der Standardkonfiguration nur mit den Services von Amazon zusammenarbeiten kann, soll eine offenere Lösung entwickelt werden. Mit einer offeneren Lösung wird im Rahmen dieser Studienarbeit eine Möglichkeit definiert, die dem Kunden einen ähnlichen Funktionsumfang anbietet. Ein grundlegender Unterschied ist jedoch im Bestellprozess vorhanden. Während die Lösung von Amazon eine direkte Bestellung bei Amazon auslöst, soll die offenere Lösung eine Art Einkaufszettel bereithalten.

Der Nutzer soll dann über ein Webfrontend die Möglichkeit haben den Einkaufszettel zu betrachten und dann die entsprechenden Produkte bei seinem bevorzugten Händler zu bestellen. So kann es auch möglich sein, dass auch Produkte geführt werden, die der Nutzer nicht online bestellen kann. Der Vorteil bei dieser Lösung liegt darin, dass der Nutzer nicht an einen Händler und dessen Preise gebunden ist, sondern die Preise betrachten kann und seine Bestellung dementsprechend aufgeben kann. Im Rahmen dieses Ziel müssen verschiedene Komponenten entwickelt werden, die im Rahmen der Lösung zusammenarbeiten. Diese Komponenten lassen sich unter folgenden Kategorien zusammenfassen:

- Entwicklung von Buttons
- Entwicklung eines Frontends zur Nutzerinteraktion
- Entwicklung eines Backends zur Verarbeitung von Frontendeingaben und der Eingabe von Buttons

Die Entwicklung von Buttons lässt sich dabei in zwei Teile aufteilen. Zum einem soll der Amazon Dash Button betrachtet und analysiert werden. In diesem Rahmen soll sowohl die Einrichtung und Konfiguration als auch die Kommunikation untersucht werden. In Folge dessen soll weiterhin geprüft werden, ob der Button auch für die offene Lösung genutzt werden kann und dabei kein Produkt bei Amazon bestellt. Weiterhin sollen mithilfe von Mikrokontrollern eigene Buttons entwickelt werden, die ebenfalls ein Signal an den entsprechenden Empfänger absenden können. Bei den Buttons steht die Funktionalität im Vordergrund und die Prüfung der Machbarkeit des Projektes.

Weiterhin muss ein Frontend entwickelt werden, welches die Nutzerinteraktion ermöglicht. Das Frontend soll zur Übersicht der Einkaufsliste genutzt werden, aber auch die Verwaltung der Buttons und andere anfallende Verwaltungsfunktionen sollen ermöglicht werden. Um eine Kommunikation zwischen Frontend und den Buttons zu gewährleisten, muss ein Backend entwickelt werden, welches über Schnittstellen die Kommunikation gewährleistet und die Eingaben entsprechend verarbeitet. Als Ergebnis des Projektes soll ein eigener Button entstanden sein, der mithilfe des Backends und Frontends dem Nutzer ermöglicht diesen Button auch zu nutzen. Weiterhin soll zumindest der Amazon Dash Button untersucht worden sein und wenn möglich miteingebunden. Abschließend soll ein Fazit gezogen werden, inwiefern das Projekt umsetzbar ist.

3 Theorie

Im Rahmen dieser Arbeit wurden verschiedene Technologien und Prinzipien eingesetzt. Auf diese soll in den folgenden Unterkapiteln eingegangen werden, damit eine theoretische Grundlage vorhanden ist.

3.1 UDP

UDP steht für "User-Datagram-Protocol" und ist ein verbindungsloses Transportprotokoll. Im Open Systems Interconnection (OSI)-7-Schichten Modell arbeitet es auf der Transportebene und ist für die Zustellung von Netzwerkpaketen von einem Sender zu einem Empfänger zuständig. Im Vergleich zum Transportprotokoll TCP, welches verbindungsorientiert arbeitet, ist es wesentlich einfacher zu verarbeiten, da beispielsweise der Header wesentlich kleiner ist. Allgemein ist es sehr minimal gehalten und dadurch sehr einfach zu implementieren und für sehr einfache Anwendungszwecke geeignet. Allerdings ist auch zu erwähnen, dass es keine Empfangsbestätigung gibt und die Daten nach dem Absenden nicht weiter kontrolliert werden. Somit können die Daten auch im Netzwerk verloren gehen und es wird nicht bemerkt.

Der einfache Header des UDP Protokolls besteht nur aus vier Attributen. Diese jeweils 16 Bit großen Felder enthalten den Quellport, den Zielport, die Checksumme zur Überprüfung des Inhalts und die Länge des gesamten Pakets. Insgesamt ist der Header somit 8 Byte groß. (vgl. [1][2][3])

3.2 TCP

TCP ist ein verbindungsorientiertes Protokoll, dass im OSI sieben Schichten Modell auf der vierten Ebene (Transport) einzuordnen ist. Im sogenannten TCP/Internet Protokoll (IP) Protokollstapel ist es in der dritten von vier Schichten zu finden. Bei einer Übertragung von Daten über TCP übergibt die genutzte Anwendung den Datenstrom an das Protokoll und empfängt ihn auch wieder von dort. Für die Übertragung ist dementsprechend TCP zuständig. Die Hauptaufgaben des Protkolls sind daher die Aufteilung und die Zusammensetzung der Daten von entsprechend vielen Paketen (Segmentierung), das Management der Verbindung und ein entsprechendes Fehlerhandling, welches das korrekte Empfangen von Paketen überwacht. Das Fehlerhandling nutzt eine positive Bestätigung aller Pakete. Dies bedeutet, dass nur nicht vorhandene Pakete erneut angefragt werden, ansonsten davon ausgegangen wird, dass die Daten angekommen sind. Diese Technologie sorgt dafür, dass die Daten auf jeden Fall ankommen, sofern die Verbindung nicht gestört wird. (vgl. [4][5])

Der Header eines TCP Pakets besteht aus 20 Bytes, jedoch kann dieser auch noch erweitert werden, sodass noch einige zusätzliche Bytes in den Header geschrieben werden. Zu den zwingend notwendigen Daten gehört unter anderem der Port auf dem das Paket empfangen wird und der Port über den das Paket gesendet wird. Zudem wird die Nummer im aktuellen Paketstrom benötigt. Zudem wird eine Prüfsumme und Quittierungsnummer mitgegeben, welche zur Kontrolle und Bestätigung genutzt werden. (vgl [4])

3.3 ARP

Das ARP Protrokoll ist ein Protrokoll, dass auf der zweiten Ebene (Sicherungsebene) des OSI Modells wiederzufinden ist. Mithilfe von ARP werden in einem Netzwerk die vergebenen IP Adressen in MAC Adressen beziehungsweise Hardwareadressen ungewandelt. Dieser Prozess ist notwendig, da innerhalb eines lokalen Netzwerkes die Netzwerkpakete über diese MAC Adressen an die entsprechenden Empfänger geleitet werden. Da zuvor die Kommunikation über IP lief, muss diese Umwandlung durchgeführt werden. Diese Unwandlung wird mithilfe von ARP realisiert. Sobald ein Gerät einem Netzwerk beitritt, wird ein ARP Request an die Adresse "FF-FF-FF-FF-FF-FF" geschickt. Diese Adresse ist ein sogenannter Broadcast, der diesen Request an alle Geräte im Netzwerk schickt. Dieser Request wird vom Router so verarbeitet, indem er der, im Request mitgesendeten, MAC Adresse des Geräts eine IP Adresse zuordnet, die für das IP Protrokoll verwendet werden kann. Nach diesem erfolgreichen Mapping einer Media-Access-Control (MAC) Adresse im Netzwerk auf eine IP Adresse, kann die weitere Kommunikation beispielsweise über das Internet Protrokoll oder andere Protrokolle genutzt werden. Aufgrund des hinterlegten Mappings weiß der Router nun, zu welcher MAC Adresse er die entsprechenden Pakete mit einer bestimmten IP Adresse weiterleiten muss. (vgl. [6][7])

3.4 Webserver: Nginx

Nginx bestitzt einen Marktanteil von ca. 20 Prozent aller Webserver und ist somit auf Rang 2 der Webserver für aktive Webseiten hinter Apache und vor Microsoft und Google[8]. In dieser Studienarbeit haben wir uns für nginx entschieden, da ein hoher Wert auf eine flexible Konfiguration und schnelle Serverantworten gelegt wird.

3.4.1 Allgemein

nginx ist ein HTTP und Reverse-Proxy Server, ein Mail-Proxy Server und ein allgemeiner TCP/UDP Proxy-Server.

3.4.2 Features

Die von nginx unterstützten Features werden im Folgenden gegliedert nach den jeweiligen Einsatzgebeiten vorgestellt.[9]

3.4.3 Basic HTTP

– Bereitstellung von statischen Dateien – Modulare Architektur – SSL und TLS SNI support – HTTP/2 Unterstützung – Rewrite Modul: Änderung von URIs anhand von regulären Ausdrücken – Zugriffskontrolle anhand von IP-Adressen oder Passwörtern – IP-basierte Geolocation

3.4.4 Mail Proxy-Server

– Weiterleitung zu IMAP oder POP3 anhand eines externen HTTP Authentifizierungsserver – Authentifizierung durch externen HTTP Authentifzierungsserver und Weiterleitung zu einem internen SMTP-Server – SSL Unterstützung – STARTTLS und STLS Unterstützung

3.4.5 TCP/UDP Proxy-Server

– Allgemeines Proxying von TCP und UDP – SSL und TLS SNI Unterstützung für TCP – Begrenzung von simultanen Verbindungen, die von derselben Adresse kommen – Zugriffkontrolle anhand der Benutzeradresse – Lastausgleich und Fehlertoleranz

3.4.6 Architektur und Skalierbarkeit

– Ein Master und mehrere Worker-Prozesse; Worker brauchen keine Berechtigungen – Flexible Konfiguration

3.5 Arduino

Die Bezeichnung Arduino steht für eine Technologie, die sowohl aus Hardware als auch aus Software besteht. Zudem gibt es zwei Unternehmen, die in ihrem Namen den Begriff Arduino tragen. Zum einem gibt es die Arduino LLC, die die Gruppe der Gründer der Plattform bezeichnet. Zudem gibt es die Arduino S.r.l., was die Firma bezeichnet, die anfangs allein die Arduinoboards produzierte und dann auch verkaufte. Die Arduinoplattform wurde ursprünglich entwickelt, um Beginnern den Einstieg in die Mikrokontrollerprogrammierung zu vereinfachen. Grundsätzlich besteht die sogenannte Arduinoplattform alber sowohl aus der Hardware als auch der Software. Die Hardware umfasst mittlerweile verschiedene Mikrokontroller beziehungsweise Boards, welche für verschiedene Anwendungszwecke genutzt werden können. Die entsprechende Nutzung wird mithilfe der richtigen Programmierung erreicht. Dafür kann der Softwareteil der Lösung genutzt werden, welches aus einer Integrated Development Environment (IDE) besteht und das Schreiben von Programmen vereinfacht. Zudem wird über diese IDE auch die Kommunikation mit dem Mikrokontroller realisiert. Diese Entwicklungsumgebung eignet sich für diverse Mikrokontroller, sofern entsprechende Treiber verfügbar sind, können auch Boards genutzt werden, die nicht direkt mit Arduino zusammenhängen. Zudem kann in verschiedenen Programmiersprachen entwickelt werden, beispielsweise C oder C++. (vgl. [10, 11, 12, 13, 14, 15]) Die komplette Plattform ist open source, auch wenn die Hardware natürlich bezahlt werden muss.

3.6 Python

Python ist eine objektorientierte Programmiersprache und wurde 1991 in der ersten Version von Guido van Rossum veröffentlicht (vgl. [16]). Zu den technischen Merkmalen gehört unter anderem die Unterstützung von Paketen beziehungsweise Bibliotheken, die die entsprechend benötigten Funktionen laden, sofern sie in das Projekt eingebunden werden.

Außerdem ist eines der Merkmale die Einrückung. Python nutzt nicht, im Gegensatz zu anderen Programmiersprachen, bestimmte Klammern oder andere Symbole um den Code zu strukturieren, sondern verwendet stattdessen die Einrückung von den entsprechenden Zeilen Code. Das bedeutet, dass zum Beispiel nach einem Methodenkopf keine geschweifte Klammer oder ähnliches verwendet wird, sondern die nächste Zeile wird um vier Leerzeichen eingerückt. Das Methodenende wird dadurch definiert, dass ein anderer Codeabschnitt wieder auf der gleichen Einrückungsstufe wie der Methodenkopf beginnt (vgl. [17][18]). Zudem ist Python eine dynamische Programmiersprache und es ist möglich Python als Skriptsprache zu verwenden und Skripte

zu schreiben, die mithilfe des Interpreters ausgeführt werden. Durch die Möglichkeit Python auch auf Linux zu verwenden, können diese Skripte auch entsprechend unter Linux verwendet werden.

Python kann für verschiedene Anwendungszwecke genutzt werden, es gibt zum Beispiel verschiedene Möglichkeiten der Stringverarbeitung, die Möglichkeit mit verschiedenen Internetprotokollen zu arbeiten, beispielsweise HTTP, FTP oder SMTP aber auch auf Interfaces des Betriebssystems zuzugreifen, um beispielsweise mit Sockets (z.B. TCP/IP) zu arbeiten. Neben diesen Funktionen, die durch die Standardbibliotheken abgedeckt werden, gibt es auch noch weitere Pakete, die weitere Bibliotheken mit Funktionen hinzufügen. So gibt es zum Beispiel die Möglichkeit das Paket BeautifulSoup oder Requests hinzuzufügen. Letzteres bietet viele Möglichkeiten mit dem HTTP Protkoll zu arbeiten (vgl [19][20][16]).

3.7 WLAN-Chipsatz

3.8 MySQL

MySQL ist ein relationales Datenbankverwaltungssystem, welches in einer Open Source als auch einer kommerziellen Version verfügbar ist. Es basiert auf der Datenbanksprache Structured Query Language (SQL) und wird häufig in Kombination mit PHP: Hypertext Preprocessor (PHP) verwendet. Die kommerzielle Version wird von der Firma MySQL AB entwickelt, die wiederrum im Jahr 2008 von Sun übernommen wurde. Da Sun durch Oracle übernommen wurde, gehört nun auch die Firma MySQL AB zu Oracle. Zudem ist es eines der am häufigsten genutzten Datenbanksysteme. (vgl. [21][22])

3.9 Frontendtechnologien

3.10 REST

3.10.1 REST Allgemein

Im Folgenden soll die Technologie REST vorgestellt werden. Zunächst wird REST etwas genauer anhand von Beispielen erläutert. Anschließend werden die Vor- und Nachteile dargestellt, sowie die Kernpunkte erläutert.

3.10.2 REST Beispiel

Um Daten von einer REST-Schnittstelle zu bekommen, werden die HTTP-Methoden in Verbindung mit einer entsprechenden URI (Uniform Resource Identifier) verwendet. Folgend wird jede für diese Arbeit relevante Methode ein Beispiel gezeigt.

GET /resources/ Liefert eine Liste aller Ressourcen.

GET /resources/4 Liefert die Ressource mit ID = 4.

POST /resources/ Legt eine neue Ressource an.

PUT /resources/4 Aktualisiert die Ressource mit ID = 4.

DELETE /resources/4 Löscht die Ressource mit ID = 4.

Einen tieferen Einblick, wann und wie die Methoden genutzt werden, wird in Abschnitt 3.10.5 gegeben.

3.10.3 Ressourcen

Im Folgenden werden Datensätze als Ressourcen bezeichnet. Dabei kann man die Ressourcen weiter unterteilen:

– Primärressourcen: Unter Primärressourcen fallen komplette, vollständige Ressourcen – Subressourcen: Subressourcen sind Teile von Primärressourcen – Listen: Eine Liste umfasst eine Menge von Ressourcen – Filter: Anwendung von Filterkriterien auf Listen – Paginierung: begrenzte Datenmenge – Projektionen: Teil der Informationen einer Primärressource – Aggregationen: Zusammenfassung mehrerer Ressourcen

3.10.4 Merkmale

Die Kommunikation erfolgt auf Abruf. Der Client ist aktiv und fordert vom passiven Server eine Repräsentation an, bzw. modifiziert eine Ressource. Ressourcen, die Objekte der Anwendung, besitzen eine ihnen zugeordnete URI, mit der sie adressiert werden können. Die Repräsentation einer Ressource kann als Dokument vom Client angefordert werden. Repräsentationen können auf weitere Ressourcen verweisen, die ihrerseits wieder Repräsentationen liefern, die wiederum auf Ressourcen verweisen können. Der Server verfolgt keinen Clientstatus. Jede Anfrage an den Server muss alle Informationen beinhalten, die zum Interpretieren der Anfrage notwendig sind. Caches werden unterstützt. Der Server kann seine Antwort als cache-fähig oder nicht cache-fähig kennzeichnen.

Repräsentationen Um die Ressourcen darzustellen, werden in REST mehrere Repräsentationen der Ressourcen genutzt. Dies entsteht aus der Intention heraus, dass aufgrund der unterschiedlichen Repräsentationen unterschiedliche Anforderungen der Clients bedient werden können. So

Statuscode	Bedeutung
1xx	Informationen
2xx	Erfolgreiche Operationen
3xx	Umleitung
4xx	Client-Fehler
5xx	Server-Fehler

Tabelle 1: Statuscodes

kann zum Beispiel die HTML Repräsentation für einen Webbrowser genutzt werden, während eine JSON Repräsentation für eine Clientapplikation genutzt wird.

3.10.5 HTTP

Um auf eine Ressource zuzugreifen wird das HTTP-Protokoll verwendet. Je nachdem, welche Verwendung die Ressourcen finden, werden geeignete HTTP-Methoden verwendet. Die häufigsten Verwendeten Methoden sind dabei GET, POST, PUT und DELETE. Per GET werden Daten geladen; POST hat mehrere Einsatzgebiete. Häufig wird dadurch eine neuer Datensatz angelegt, oder mehrere Datensätzen aktualisiert. PUT aktualisiert einen bestimmten Datensatz und DELETE löscht einen Datensatz. Dabei müssen sinnvolle Statuscodes zurückgegeben werden. Zusätzlich gibt es noch die Methoden PATCH, HEAD und OPTIONS. Auf diese wird jedoch nicht weiter eingegangen, da diese eine vergleichsweise geringe Bedeutung bei einer REST-Schnittstelle haben. Bei der Implementierung ist unbedingt zu beachten die Methoden idempotent zu gestalten, d.h. Ein mehrmaliges Aufrufen derselben Methode wirkt sich nicht anders aus, als ein einmaliger Aufruf. Einzige Ausnahme dabei ist die POST-Methode. Das Prinzip von REST ist, Ressourcen von einem Zustand in den nächsten Zustand zu überführen.

3.10.6 Statuscodes

Folgende Tabelle zeigt die allgemeine Bedeutung der unterschiedlichen Statuscodes

3.10.7 URI-Design

Eine URL (Uniform Resource Locator) lokalisiert eine Ressource, wobei eine URI (Uniform Resource Identifier) sie identifiziert. Lokalisierung kann dabei auch zur Identifizierung führen. Eine URL ist also immer eine URI. Im Umkehrschluss kann eine URI eine URL sein, muss es aber nicht.

Möchte man REST in einem Satz definieren, dann ergibt sich: „Eine eigene URI für jedes Informationselement.“ [23][24]

An diesem Satz sollte man sich auch beim URI-Design halten. Ressourcen können so durch ihre URI eindeutig identifiziert werden. Dabei sollen diese so konstruiert werden, dass man bereits aus der URI alleine lesen kann, was für eine Ressource dahinter steckt. Eine URI zeigt auf eine Ressource, aber diese kann über mehrere URIs angesprochen werden. Die verschiedenen Ressourcenarten können auf folgende URI-Strukturen abgebildet werden:

Liste: /resources/

Man bekommt mehrere Ressourcen gleicher Art

Primärressource: /resources/1
Man bekommt genau eine unabhängige Ressource
Sekundärressource: /resources/subresources/1
Man bekommt genau eine von einer anderen Ressource abhängige Ressource
Filter: /resources?name=a oder /resources/name=a
Man bekommt mehrere Ressourcen gleicher Art, die mit bestimmten Parametern gefiltert werden.

3.10.8 HATEOAS

HATEOAS ist ein wichtiges Prinzip, welches von vielen selbsternannten REST-Schnittstellen nicht implementiert wird. Dieses Akronym beschreibt, dass die Schnittstelle über Hypertext beziehungsweise Hypermedia angetrieben wird. Genauer gesagt, sollen durch den Aufruf der Haupt-URI alle Funktionalitäten als Hyperlinks unter Angabe der jeweiligen Beziehung zugreifbar sein. Die Schnittstelle dokumentiert sich sozusagen selbst. In seiner im Jahre 2000 verfassten Dissertation "Architectural Styles and the Design of Network-based Software Architectures", definiert Roy Fielding die REST-Architektur. Dabei geht er unter anderem auf Ressourcen und HTTP ein. Als wichtigen Punkt sieht Fielding jedoch auch die Verwendung von Hypermedia und die Verlinkung zwischen verwandten Ressourcen. In 2008 verdeutlicht Fielding anhand eines Blogeintrags seinen Standpunkt zu RESTful API's... "If the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?" – [25]

Damit will er sagen, dass Schnittstellen, welche nicht über Hypertext gesteuert werden, auch keine REST-Schnittstellen sind, sondern ganz normale RPC-Schnittstellen. Um dieses Prinzip zu verdeutlichen wird anhand Abbildung 3 das Richardson Maturity Model veranschaulicht.

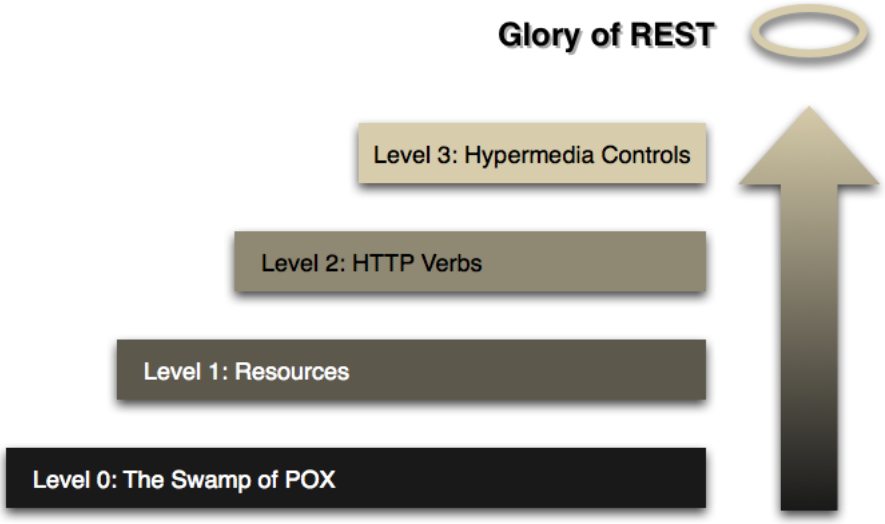

Abbildung 1: Richardson Maturity Model,
Quelle: https://martinfowler.com/articles/images/richardsonMaturityModel/overview.png

Das Richardson Maturity Model unterteilt die REST-Architektur in vier Stufen. Eine echte REST-Schnittstelle implementiert dabei alle vier Stufen. Doch gerade die letzte Stufe wird oft

nicht implementiert, da sie eine gewisse Komplexität birgt.

LEVEL 0 sagt lediglich aus, dass HTTP als Transportprotokoll für RPC-Aufrufe genutzt wird. Die Kommunikation findet über eine einzige URI statt.

LEVEL 1 sagt aus, dass jede Ressource eine eigene URI hat.

In LEVEL 2 werden die bereits Vorgestellten HTTP-Methoden verwendet.

In der letzten Stufe, LEVEL 3, kristallisieren sich die Unterschiede zwischen richtigen REST-Services und solchen, die es gerne sein wollen. Durch den Einsatz von Hypermedia sollen alle Ressourcen miteinander verknüpft werden. Dabei soll beim Aufruf der Haupt-URI alle möglichen Aufrufe zurückgegeben werden. Folgende Tabelle soll die Vor- und Nachteile von HATEOAS verdeutlichen.

Vorteile	Nachteile
Entkopplung von Client und Server, da sich der Client nicht anpassen muss, wenn sich serverseitige Änderungen ergeben	Findet relativ geringe Anwendung
Transparenz von Änderungen in der Ressourcenverteilung: Client kann Links folgen ohne die genaue Serverinfrastruktur zu kennen	Steigert die Komplexität der Implementierung
Serverseitig steuerbarer Anwendungsfluss: Server teilt mit, welche Aktionen möglich sind. Der Client kann sich dynamisch danach richten.	Bei Erweiterung der Schnittstelle müssen natürlich überall Anpassungen gemacht werden.
Selbstbeschreibende API	

Tabelle 2: Vorteile und Nachteile des Richardson Maturity Model

Dass HATEOAS einen gewissen Nutzen bringt ist nicht abzustreiten, jedoch müssen diese mit den Kosten verglichen werden. Der Aufwand kann dem Nutzen deutlich überwiegen, da macht es natürlich keinen Sinn dieses Prinzip anzuwenden. Dies ist vor allem bei kleineren internen Projekten, die nicht zur öffentlichen Verwendung gedacht sind, der Fall. Möchte man aber dem REST-Prinzip hundertprozentig folgen, so muss HATEOAS implementiert werden.

3.10.9 Fazit

[26]

4 Beschreibung der Hardware

In den folgenden Kapiteln wird die ausgewählte Hardware des Projektes vorgestellt. Dabei wird zuerst auf die technischen Merkmale eingegangen und dann auf den genauen Verwendungszweck im Projekt.

4.1 Raspberry PI

4.1.1 Vorstellung des Raspberry PI

Das Raspberry PI ist ein Einplatinencomputer, den es in verschiedenen Ausführungen gibt. Je nach Ausführung variieren die Ausstattungsmerkmale. Zu den Grundsätzlichen Ausstattungsmerkmalen gehören eine CPU, unterschiedlich viel RAM und eine iGPU Einheit. Er wurde von der Raspberry PI Foundation entwickelt und hatte das ursprüngliche Ziel einen günstigen Computer für den Schulunterricht bereitzustellen. Daher ist es auch möglich eine vollständige Linux Distribution als Betriebssystem zu nutzen und es wurde sogar eine speziell angepasste Distribution veröffentlicht, die Raspbian bezeichnet wird. Aufgrund der verschiedenen Möglichkeiten wird er aber mittlerweile auch in vielen anderen Anwendungsgebieten genutzt. Insbesondere die neueren Modelle, die mit mehreren USB Ports, General Purpose Input/Output (GPIO) Pins, einem Ethernet Port und weiteren Anschlüssen ausgestattet sind, werden auch in verschiedensten Projekten privater Personen genutzt. Ein weiteres Merkmal ist die Größe des Raspberry PI's, die mit maximal 93mmx63.5mmx20mm sehr klein ausfällt. Zusätzlich gibt es diverses, bereits für den Raspberry PI ausgelegtes, Zubehör, welches weitere Erweiterungsmöglichkeiten bietet. So gibt es Kameras, Gehäuse, kleine Displays und WLAN Sticks, die den Funktionsumfang erweitern. So wurden bereits diverse Projekte vorgestellt, die zeigen, dass die Einsatzmöglichkeiten des Raspberry Pis wesentlich größer sind. (vgl. [27] [28])

4.1.2 Verwendung im Projekt

Der Raspberry PI wird für das Projekt genutzt, um einen zentralen Server bereitzustellen. Aufgrund der technischen Merkmale kann zeitgleich ein Webserver und ein Datenbankserver betrieben werden. Zudem kann über die USB Anschlüsse ein WLAN Stick angeschlossen werden, sodass die Buttons über das WLAN Netzwerk des Raspberry PIs zum zentralen Server kommunizieren können. Dazu muss mithilfe eines Skriptes der WLAN Stick von einem Empfänger zu einem Sender bzw. Access Point umfunktioniert werden. Um die dann eingehenden Nachrichten der Buttons empfangen zu können, muss zudem noch ein entsprechendes Skript im Hintergrund laufen, dass die Daten empfängt. Aufgrund von mehreren parallel laufenden Prozessen (Webserver, Datenbankserver, WLAN Skript und Skripte zum Empfangen der Daten) wird einiges an Rechenleistung benötigt, die der Raspberry Pi allerdings aufbringen kann.

Der Webserver auf dem Raspberry Pi wird dabei sowohl für das Frontend als auch für das Backend benötigt werden. Das Frontend soll dem Nutzer die Möglichkeit geben, die Liste von Waren zu verwalten und die Buttons zu konfigurieren bzw. Hilfestellung zur Einrichtung zu geben. Das Backend des Servers wird unter anderem aus einer REST Application Programming Interface

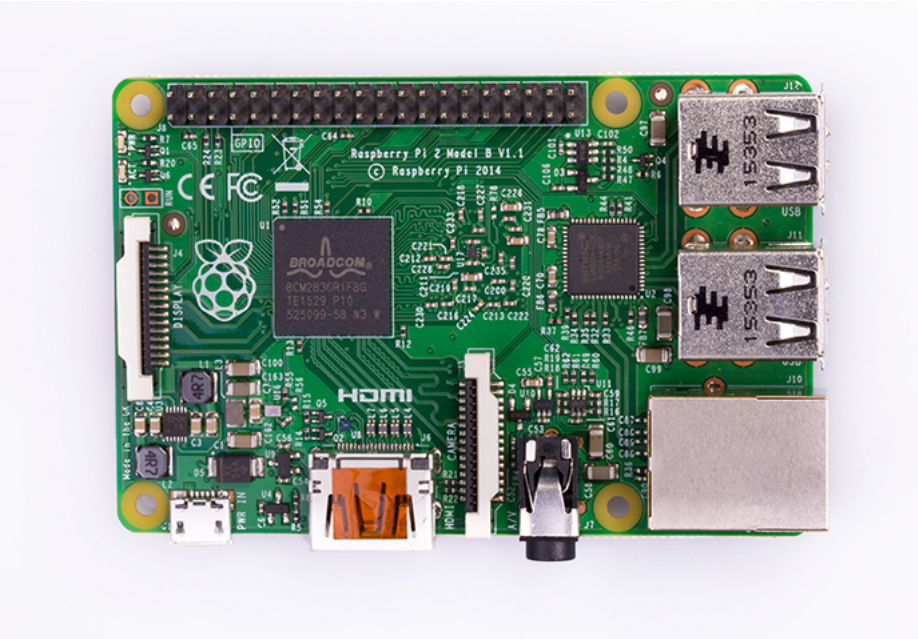

Abbildung 2: RaspberryPi Modell 2 B+,
Quelle: https://www.raspberrypi.org/wp-content/uploads/2016/02/Raspberry-Pi-2-web.jpg

(API) bestehen, die sowohl die Anfragen des Webservers verarbeitet als auch die Anfragen an die Datenbank im allgemeinen. Der Datenbankserver im Hintergrund wird die benötigte Datenbank entsprechend verwalten.

4.2 “Pretzelboard”

4.2.1 Vorstellung des “Pretzelboard”

Das “Pretzelboard” ist ein sogenanntes Elektronikmodul, dass unter verschiedenen Namen bekannt ist. Neben “Pretzelboard” ist der Name “NanoESP-Board” ebenfalls bekannt. Die Besonderheit des Boards ist ein bereits angeschlossenes WLAN Modul, welches sowohl als Sender als auch Empfänger arbeiten kann. Da es zudem den gleichen Microcontroller wie das bekanntere Microcontrollerboard “Arduino Uno” nutzt, kann die Entwicklungsumgebung von Arduino zur Entwicklung von Software genutzt werden. Der Vorteil der bereits erfolgten Kombination und Verdrahtung von WLAN Modul und Microcontroller liegt darin, dass der Nutzer dies nicht mehr machen muss und somit wesentlich weniger Kenntnisse vorausgesetzt werden müssen. (vgl. [29][30][31])

4.2.2 Verwendung im Projekt

Das “Pretzelboard” wird im Projekt als Button genutzt. Sowohl die kleine Größe als auch die bereits vorhandene WLAN Funktionalität sorgen dafür, dass sich das Board dafür besonders gut eignet. Aufgrund der Tatsache, dass es sich bezüglich der Programmierung nicht von einem Arduino Board unterscheidet, besteht die Möglichkeit, dass auf das Wissen und den Support einer bereits größeren Community zurückgegriffen werden kann. Da sich das Board zudem auf ein Elektroniksteckboard setzen lassen kann, ist auch die Verbindung mit einem Button und Statusleuchten möglich. Nach der erfolgreichen Montage kann dann das entsprechende Programm aufgespielt werden und bei vorhandener Stromversorgung kann die entsprechende Nachricht über das WLAN Netzwerk an den Empfänger gesendet werden. Das Pretzelboard kommuniziert diese Nachricht mithilfe des zuvor bereits erwähnten Prokotolls UDP.

4.3 ESP8266 Lua

4.3.1 Vorstellung des ESP8266 Lua

Der ESP8266 Lua ist ein Entwicklungsboard, welches bereits ein integriertes WLAN Modul besitzt und mit einem TCP/IP Stack ausgestattet ist. Einer der vorgesehenen Einsatzzwecke ist unter anderem das Internet of Things. Das Board kann zudem mit Steckbrett arbeiten und bietet daher ähnliche Möglichkeiten, wie das bereits erwähnte "Pretzelboard". Zudem ist es möglich, dass die Software für das ESP8266 Lua Entwicklungsboard ebenfalls über die Arduino IDE entwickelt werden kann. Allerdings sind technische Unterschiede zum Pretzelboard vorhanden, was dazu führt, dass andere Treiber genutzt werden müssen. Diese können nachinstalliert werden und stehen dann zur Nutzung des Boards bereit. (vgl. [32][33])

4.3.2 Verwendung im Projekt

Das Board wird im Rahmen des Projektes als weiterer Button genutzt. Es bietet sich für diese Funktion an, da es ebenfalls recht klein ist und alle benötigten Funktionen bereits vorhanden sind. Neben den vorhandenen Funktionen ist auch die Möglichkeit vorhanden, dass eine bereits durch das Bretzelboard bekannte Entwicklungsumgebung genutzt werden kann. Das bietet die Möglichkeit, dass statt des UDP Protokolls das bereits erwähnte TCP Protokoll ausprobiert werden kann. Der Aufbau des Boards soll ebenfalls durch ein Elektroniksteckboard umgesetzt werden. Dazu wird das Board auf eben dieses Steckboard gesetzt und mit einem Button, Statusleuchten und entsprechenden Kabeln verbunden. Nach der Betätigung des Buttons wird das Board geweckt und eine Funktion schickt ein entsprechendes Datenpaket an einen Empfänger.

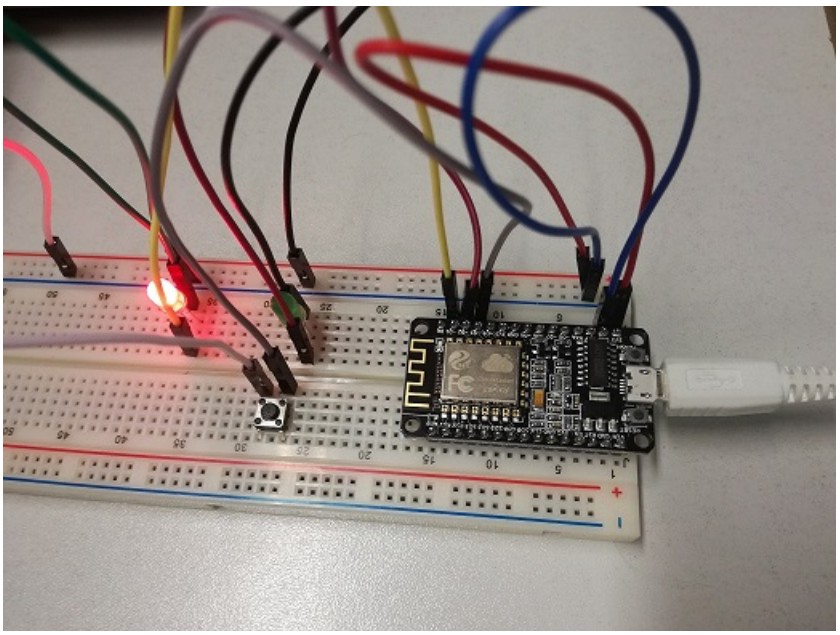

Abbildung 3: ESP8266 im Projekt,Quelle: Eigene Aufnahme

4.4 Amazon Dash Button

4.4.1 Vorstellung des Amazon Dash Buttons

4.4.2 Untersuchung des Amazon Dash Buttons

4.4.3 Verwendung des Amazon Dash Buttons im Projekt

5 Umsetzung des Projektes

5.1 Architektur und Zusammenarbeit der Komponenten

Mithilfe der Technologien und Softwarelösungen, die in Kapitel 3 erläutert worden sind, und der Hardware, die in Kapitel 4 vorgestellt wurde, konnten die verschiedenen Komponenten, die bereits in der Aufgabenstellung in Kapitel 2 genannt worden sind, umgesetzt werden. Im folgenden soll ein Überblick über die Zusammenarbeit der einzelnen Komponenten gegeben werden, bevor in den folgenden Kapiteln auf die genaue Entwicklung der einzelnen Komponenten eingegangen wird.

Für den Nutzer gibt es ein Webfrontend, welches mithilfe eines Nginx Webservers realisiert wird. Dieses Webfrontend greift auf ein REST API-Backend zurück, welches in PHP geschrieben wurde und ebenfalls über den Nginx realisiert wird. Mithilfe dieses Backends werden die verschiedenen API Routen realisiert, die dann den Zugriff auf die Datenbank organisieren. Das Backend besteht ebenfalls aus einigen Skripten, die in Python geschrieben sind. Diese Skripte organisieren die Kommunikationsendpunkte für die Buttons indem sie auf die entsprechende Kommunikationsports für die Datenpakete lauschen und die Daten dann verarbeiten.

Neben der Kommunikation wird ebenfalls ein Teil der REST API in Python realisiert. Dieser Teil ermöglicht sowohl den Status der Ports abzufragen als auch die Skripte neu zu starten, sollte ein Skript nicht gestartet sein oder ein Fehler aufgetreten sein. Diese Daten werden ebenfalls für das Webfrontend verwendet. Die Buttons als weitere Komponente werden durch verschiedene Hardwareelemente realisiert. Mithilfe entsprechender Programme, die auf die Controller geladen werden, stellen sie eine Verbindung zu den definierten Kommunikationspunkten her. Über dieses Kommunikationsprotokoll werden dann, sobald der Button betätigt wurde, die entsprechenden Daten gesendet und dann im Backend verarbeitet.

5.2 Entwicklung der Buttons

Im folgenden soll auf die Entwicklung der Button Komponente im Projekt eingegangen werden. Dabei wird insbesondere auf die Programmierung und Umsetzung eingegangen werden. Bei der Entwicklung der verschiedenen Buttons wurde insbesondere darauf geachtet, dass verschiedene Technologien getestet werden, um möglichst viele technische Möglichkeiten zu untersuchen. Das ist damit zu begründen, dass das Untersuchen von möglichst vielen Möglichkeiten zum Projektziel gehörte.

5.2.1 Entwicklung mit dem "Pretzelboards"

5.2.2 Entwicklung mit dem ESP8266 Lua

5.2.3 Einbindung des Amazon Dash Buttons

5.3 Entwicklung des Frontends

5.3.1 Aufbau und Entwicklung des Frontends

5.4 Entwicklung und Einrichtung des Backends

In den folgenden Kapiteln wird auf die Einrichtung des Backends eingegangen. Dies umfasst sowohl die Einrichtung des Raspberry Pis, den Aufbau und die Entwicklung der REST API und die Entwicklung der Python Skripte. Die Einrichtung des Raspberry PIs wird an dieser Stelle geführt, da er das zentrale Hardwareelemente im Backend ist, da er genutzt wird, um die entsprechende Softwaredienste zur Verfügung zu stellen. Er stellt den zentralen Kommunikationspunkt im Projekt dar, da die Buttons nur ihn kennen, sich aber nicht gegenseitig.

5.4.1 Einrichtung des Raspberrys

Die Einrichtung des Raspberry PIs besteht aus mehreren Schritten an dessen Ende die Verwendung des Raspberrys als zentraler Server steht. Die Installation geht von der bereits installierten und aktuellen Raspbian Distribution aus. Die verschiedenen Schritte werden im folgenden erklärt:

Einrichtung des Nginx

Der Webserver Nginx, welcher bereits in Kapitel 3.4 vorgestellt wurde, dient als Webserver für das Frontend und einen Teil des Backends. Um ihn entsprechend einzurichten, müssen zuerst die entsprechend Pakete mithilfe des Befehles "sudo apt-get install nginx" installiert werden. Nach der erfolgreichen Installation muss die Konfigurationsdatei noch entsprechend angepasst werden.

Bevor der Server gestartet werden kann, sollten die entsprechenden Dateien für die Website heruntergeladen werden und in das Rootverzeichnis des Webservers gelegt werden. Danach kann der Nginx gestartet werden. Die genaue Konfigurationsdatei des Nginx findet sich im Anhang.

Einrichtung der Webseite

Da für das Frontend der Webseite das Slim Framework in der Version 3 genutzt wird, muss noch ein Befehl ausgeführt werden, bevor die Webseite genutzt werden kann. Dieser dient zur Einrichtung. Um das Frontend entsprechend darstellen und entwickeln zu können muss in das Rootverzeichnis des Frontends gewechselt werden. In diesem ist die Datei composer.phar vorhanden. Mithilfe des Befehls "php composer.phar start" werden alle bisherigen Abhängigkeiten für das Projekt installiert und die entsprechenden Konfigurationen vorgenommen.

Einrichtung des SQL Datenbankservers

Für die Datenbank des Projektes wird MySQL genutzt. Dieses muss zuerst über den Befehl "sudo apt-get install mysql" installiert werden. Nach dem Herunterladen wird ein entsprechender Setup gestartet, der den Datenbankserver erfolgreich auf dem Raspberry Pi installiert. Nach der Einrichtung eines Adminstrators (root User) muss ein weiterer Nutzer mit Password angelegt werden, der für dieses Projekt genutzt wird. Außerdem muss mit diesem Nutzer eine Datenbank für dieses Projekt angelegt werden.

Danach muss in der Datenbank ein Skript ausgeführt werden, welches die benötigten Tabellen anlegt (siehe Anhang). Abschließend müssen die Nutzerdaten und der Datenbankname noch in die entsprechende Konfigurationsdatei des Frontends eingetragen werden, welche unter "src/settings.php" zu finden ist.

Einrichtung des WLAN Access Points

Neben dem Frontend und dem Backend stellt der Raspberry Pi auch noch das WLAN zur Verfügung. Dazu wird ein entsprechender WLAN Stick benötigt, der für den Modus als Accesspoint Sender geeignet ist. Außerdem muss eventuell noch der entsprechende Treiber installiert werden. Nach dieser Einrichtung wird der Raspberry Pi so konfiguriert, dass er als WLAN Sender genutzt werden kann. Dazu wird das Programm hostapd benötigt, welches über den Befehl "sudo apt-get install hostapd" installiert wird.

Nach der erfolgreichen Installation müssen zwei Konfigurationsdateien verändert werden. Die erste Konfigurationsdatei ist unter "/etc/hostapd/hostapd.conf" zu finden und wird für die Konfiguration des Programms hostapd benötigt. Dieses Programm wird im Hintergrund dazu genutzt, um die WLAN Verbindung im Allgemein zu gewährleisten und zu kontrollieren (vgl. [34][35]). In dieser Konfigurationsdatei werden die entsprechenden Einstellungen für das WLAN getroffen, beispielsweise Name oder Sicherheitsstufe. Die verwendete Konfigurationsdatei ist im Anhang zu finden.

Die zweite Konfigurationsdatei ist unter "/etc/network/interfaces" zu finden. Diese Datei beinhaltet alle Netzwerkschnittstellen und ihre entsprechende Konfiguration (vgl. [36]). Diese Datei muss ebenfalls verändert werden, damit die wlan Schnittstelle des Raspberry Pi's als Wlan Accesspoint genutzt werden kann. Eine der wichtigsten Einstellungen ist die Unterbindung der Weiterleitung aller Daten. Ohne diese Einstellung könnte ein Amazon Dash Button nämlich weiterhin entsprechende Produkte bestellen.

Diese Möglichkeit soll im Rahmen dieses Projektes unterbunden werden, daher wird die Weiterleitung in der Konfigurationsdatei unterbunden. Die genaue Konfiguration ist im Anhang zu finden und beinhaltet noch weitere, kleinere Veränderungen im Vergleich zu ursprünglichen, die es ermöglichen, dass die Wlanschnittstelle nicht als Empfänger sondern als Sender arbeitet.

Einrichtung von Python:

Im Rahmen der Einrichtung des Raspberry Pi's ist auch die Einrichtung von Python durchzuführen. Für eine erfolgreiche Einrichtung müssen die entsprechenden Softwarepakete installiert werden. Neben der grundlegenden Instalaltion von Python galt es auch die entsprechenden Skripte zu entwickeln. Dies wird allerdings genauer im Kapitel 5.4.2 betrachtet. Allerdings kann bereits an dieser Stelle erwähnt werden, dass sowohl ein Skript für einen UDP und TCP Empfänger entwickelt werden soll.

Diese beiden Skripte sollen für die selbstentwickelten Buttons genutzt werden. Diese sollen entweder per UDP oder TCP ihre Signale an den Empfänger schicken. Durch diese Variation der Protokolle sollen die Unterschiede und die sinnvolle Nutzung der Protokolle betrachtet werden.

Neben diesen Skripten muss allerdings auch noch ein Skript geschrieben werden, welches auf ARP Pakete achtet. Diese werden benötigt, um den Amazon Dash Button zu kontrollieren. Auch dessen Entwicklung ist im Kapitel 5.4.2 genauer beschrieben.

5.4.2 Aufbau und Entwicklung der Python Skripte

Mithilfe der Skriptsprache Python werden die verschiedenen Kommunikationsendpunkte in Form von Sockets realisiert. Um eine bessere Übersichtlichkeit zu gewährleisten und eine zentrale Ver-

waltung zu haben, gibt es ein Verwaltungsskript. Dieses Skript startet die anderen Skripte, die sich um jeweils einen Kommunikationsprotokoll kümmern. So gibt es ein Skript für die Kommunikation über UDP, eins für TCP und eins für ARP, welches für die Amazon Dash Buttons genutzt wird. Aus diesen Gründen gibt es insgesamt vier Python Skripte, die einen wesentlichen Teil des Backends ausmachen.

Entwicklung des Verwaltungsskripts

Das Verwaltungsskript dienst, wie bereits erwähnt, als zentrales Skript, welches als einziges Skript auch gestartet werden muss. Über dieses Skript werden dann alle anderen notwendigen Skripte gestartet, die dann dafür sorgen, dass die Kommunikation ermöglicht wird. Neben dieser Funktionalität wird durch das Verwaltungsskript auch eine REST API realisiert. Diese wird mithilfe des Frameworks Flask (vgl. [7]) umgesetzt. Diese API wird dazu genutzt, um einige Funktionalitäten bereitzustellen, die im Frontend benötigen. Als Beispiel wäre der aktuelle Status der Empfängerskripte (vgl. Kapitel 5.4.2 und Kapitel 5.4.2) zu nennen.

Die genannte REST API kann dann im Verwaltungsskript auf andere Methoden zugegriffen werden, die dann weitere Funktionen umfassen. Die Rückgabewerte dieser Funktionen werden dann im JavaScript Object Notation (JSON) Format an den Webfrontendteil der Anwendung weitergegeben und können dann dort weiter verarbeitet werden.

Entwicklung des UDP Skripts

Da die Übertragung der Datenpakete über das UDP Protokoll ermöglicht werden soll, muss ein entsprechender Empfänger auf dem Raspberry PI vorhanden sein. Dieser Empfänger wird mithilfe eines Skriptes in Python realisiert.

Dieses Skript nutzt die Library "Socket" (vgl. [37]), welches es ermöglicht ein Socket zu erstellen. Dieses Socket wird an eine IP Adresse und einen Port gebunden und wird anschließend für alle eingehenden UDP Pakete genutzt. In einer Endlosschleife, welche manuell unterbrochen werden kann, wird auf eingehende Pakete gewartet. Die Endlosschleife wird benötigt, da ein Button zu jedem Zeitpunkt betätigt werden kann und somit dauerhaft auf ankommende Pakete geachtet werden muss.

Bei Eingang eines entsprechenden Pakets wird ein entsprechendes Request an die REST API geschickt. Da als Übertragungsart allerdings das UDP Protrokoll genutzt wird, kann dem Button keine Rückmeldung gegeben werden, ob der Eintrag in die Datenbank über die REST API erfolgreich war. Zudem kann auch kein allgemeines Feedback zurückgegeben werden. Auch bei anderen Fehlern kann dem Button keine Rückmeldung gegeben werden.

Nach dem erfolgreichen Absenden der Anfrage an die REST API befindet sich das Skript für den UDP Empfänger weiterhin in der Endlosschleife und wartet auf das nächste Datenpaket. Das Skript ist auch im Anhang zu finden.

Entwicklung des TCP Skripts

Für alle Datenpakete, die über das Protokoll TCP empfangen werden, muss ebenfalls ein Skript geschrieben werden, welches diese Datenpakete verarbeitet. Dabei wird genauso vorgegangen, wie beim Skript für UDP. Der einzige Unterschied ist nach dem Absenden der Anfrage an die REST API. Das Skript für TCP Datenpakete wartet nach dem Absenden der Anfrage auf die

Bestätigung und schickt eine entsprechende Antwort zurück an den Button. Dieser verarbeitet ebenfalls die Antwort und kann mithilfe einer Lampe dem Nutzer ein visuelles Feedback geben. Das entsprechende Skript ist im Anhang zu finden.

Entwicklung des ARP Skripts

Da das Mitlesen der IP Datenpakete des Amazon Dash Buttons nicht möglich war, wurde das mitschneiden der ARP Datenpakete notwendig. Im Vergleich zu den Skripten, die in den Kapiteln Entwicklung des UDP Skripts und Entwicklung des TCP Skripts beschrieben sind, ist dieses Skript etwas anders aufgebaut. Es wird zwar ebenfalls die Library "Socket" genutzt, allerdings ist die Konfiguration des Sockets anders, damit ARP Datenpakete gelesen werden können.

Das Lesen dieser Datenpakete beschränkt sich allerdings auf das Erkennen der IP Adresse, die das Paket abschickt bzw. empfängt. Nur diese Informationen werden benötigt, da die Funktionalität des Skript darauf beruht, dass jeder zweite ARP Request eine entsprechende Anfrage an die REST API abschickt. Das nur jeder zweite Request eine entsprechende Anfrage abschickt, ist damit begründet, dass bei einem einzelnen Druck auf den Amazon Dash Button ein ARP Paket von Button zum Router geschickt wird und ein Paket vom Router zum Button. Das bedeutet, dass für jeden Druck zwei Requests abgeschickt werden. Daher löst nur jedes zweite Paket eine Anfrage aus, die dann die Anzahl eines Produktes in der Datenbank erhöht.

6 Ergebnis des Projektes

7 Fazit und Ausblick

Literatur

[1] Elektronik-Kompendium. Tcp - transmission control protocol. http://www.elektronik-kompendium.de/sites/net/0812271.htm.

[2] Elektronik-Kompendium. Udp - user datagram protocol. http://www.elektronik-kompendium.de/sites/net/0812281.htm.

[3] ITwissen.info. Udp :: user datagram protocol :: Udp-protokoll :: Itwissen.info. http://www.itwissen.info/definition/lexikon/user-datagram-protocol-UDP-UDP-Protokoll.html, 23.02.2016.

[4] Tcp - transmission control protocol. https://www.elektronik-kompendium.de/sites/net/0812271.htm.

[5] Tcp :: transmission control protocol :: Tcp-protokoll :: Itwissen.info. http://www.itwissen.info/definition/lexikon/transmission-control-protocol-TCP-TCP-Protokoll.html, 22.11.2016.

[6] Arp - address resolution protocol (ethernet cache). https://www.elektronik-kompendium.de/sites/net/0901061.htm.

[7] Arp (address resolution protocol) :: Arp-protokoll :: Itwissen.info. http://www.itwissen.info/ARP-address-resolution-protocol-ARP-Protokoll.html.

[8] Webserver - marktanteile februar 2017 | statistik. https://de.statista.com/statistik/daten/studie/181588/umfrage/marktanteil-der-meistgenutzten-webserver/.

[9] nginx. https://nginx.org/en/, 14.03.2017.

[10] Arduino - environment. https://www.arduino.cc/en/Guide/Environment.

[11] Arduino - getting started. https://www.arduino.cc/en/Guide/HomePage.

[12] Arduino - getting started. https://www.arduino.cc/en/Guide/HomePage.

[13] Arduino - introduction. https://www.arduino.cc/en/Guide/Introduction.

[14] Getting started. http://www.arduino.org/learning/getting-started.

[15] heise online. Arduino - mikrocontroller für einsteiger. https://www.heise.de/thema/Arduino.

[16] History and license — python 3.6.1rc1 documentation. https://docs.python.org/3/license.html, 05.03.2017.

[17] Kapitel 14: Klassen und methoden. http://python4kids.net/how2think/kap14.htm, 19.08.2013.

[18] 2. lexical analysis — python 3.6.1rc1 documentation. https://docs.python.org/3/reference/lexical_analysis.html, 10.03.2017.

[19] Welcome to python.org. https://www.python.org/doc/essays/blurb/.

[20] Welcome to python.org. https://www.python.org/about/apps/.

[21] Mysql › wiki › ubuntuusers.de. https://wiki.ubuntuusers.de/MySQL/.

[22] Torsten Horn. Sql-grundlagen. http://www.torsten-horn.de/techdocs/sql.htm, 12.03.2017.

[23] Rest: the quick pitch | quoderat on wordpress.com. https://quoderat.megginson.com/2007/02/15/rest-the-quick-pitch/.

[24] Stefan Tilkov, Martin Eigenbrodt, Silvia Schreier, and Oliver Wolf. *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt.verlag, Heidelberg, 3., aktualisierte und erweiterte auflage edition, 2015.

[25] Rest apis must be hypertext-driven » untangled. http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven, 19.03.2017.

[26] Stefan Tilkov, Martin Eigenbrodt, Silvia Schreier, and Oliver Wolf. *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. dpunkt, s.l., 3. aufl. edition, 2015.

[27] Rpi hardware - elinux.org. http://www.elinux.org/RPi_Hardware, 28.12.2016.

[28] Raspberry Pi Foundation. Raspberry pi foundation - about us. https://www.raspberrypi.org/about/, 28.01.2017.

[29] Wifi board: Nanoesp bzw. pretzel-board. http://www.mikrocontroller-elektronik.de/wifi-board-nanoesp-bzw-pretzel-board/.

[30] Technische daten nanoesp & pretzel board. http://iot.fkainka.de/technische-daten.

[31] Franzis Verlag GmbH and München. Pretzel-board. http://www.franzis.de/elektronik/arduino-platinen/pretzel-board, 27.11.2015.

[32] Frank Carius. Msxfaq.de:nodemcu. http://www.msxfaq.de/sonst/bastelbude/nodemcu.htm, 15.01.2017.

[33] Elegiant nodemcu lua esp8266 esp-12e wifi development: Amazon.de: Computer & zubehör. https://www.amazon.de/ELEGIANT-NodeMcu-ESP8266-ESP-12E-Development/dp/B018E741G4/ref=sr_1_1?ie=UTF8&qid=1487860153&sr=8-1&keywords=esp8266+lua.

[34] hostapd(8). https://www.freebsd.org/cgi/man.cgi?query=hostapd&sektion=8&manpath=FreeBSD+6.1-RELEASE.

[35] Wlan router › wiki › ubuntuusers.de. https://wiki.ubuntuusers.de/WLAN_Router/.

[36] interfaces › wiki › ubuntuusers.de. https://wiki.ubuntuusers.de/interfaces/.

[37] 18.1. socket — low-level networking interface — python 3.6.0 documentation. https://docs.python.org/3/library/socket.html, 20.02.2017.

Anhang

Skripte

Python Skripte

Alle Python Skripte, die verwendet wurden:

UDP Skript für den Raspberry PI:

```
#!/usr/bin/python
import socket
import requests
import json
import sys

udp_ip = '192.168.1.1'
udp_port = 8888
buffer_size = 1024

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
status = False
def getStatus():
    return status

def startServer(udp_ip, udp_port, buffer_size):
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    sock.bind((udp_ip, udp_port))
    print("Opened UDP socket!")
    status = True
    while True:
        recvdata, addr = sock.recvfrom(buffer_size)
        print("Received data via UDP: %s" %recvdata)

        url = "http://localhost/api/cart/%s" %recvdata
        #data = {'senderid': recvdata[:1]}
        recvdata = recvdata[:1]
        headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
        try:
            r = requests.post(url, data={}, headers=headers)
            if r.status_code == 200:
                print("Enter data via REST: Sucess: %i" %r.status_code)
            else:
```

```
                print("Enter data via REST, Error: %i" %r.status_code)
        except requests.exceptions.RequestException as e:
            print("Error %s" %e)

def stopServer():
    status = False
    sock.close()

if __name__ == '__main__':
    startServer(udp_ip, udp_port, buffer_size)
```

TCP Skript für den Raspberry PI:

```
#!/usr/bin/python
import socket
import sys
import requests
import json

tcp_ip = '192.168.1.1'
tcp_port = 5005

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
status = False
def getStatus():
    return status

def startServer(tcp_ip, tcp_port, buffer_size):
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((tcp_ip, tcp_port))
    s.listen(1)
    print('Opened TCP socket!')
    while 1:
            conn, addr = s.accept()
            recvdata = conn.recv(buffer_size)
            if not recvdata:
                    print('Error while receiving data')
                    conn.send("400")
            else:
                    print("Received data via TCP: %s" % recvdata)
                    #data = {'senderid':recvdata[:1]} #enter correct body
                    data = {" "}
```

```
                recvdata = recvdata[:1]
                url = "http://localhost/api/cart/%s" % recvdata #enter corre
                headers = {'Content-type': 'application/json', 'Accept': 'tex
                try:
                    r = requests.post(url, data={}, headers=headers, timeout=
                    if r.status_code == 200:
                        conn.send("200")
                        print("Enter data via REST: Success %i" %r.status_cod
                    else:
                        conn.send(str(r.status_code))
                        print("Enter data via REST: Error: %i" %r.status_code
                except requests.exceptions.RequestException as e:
                    print("Error %s" %e)

def stopServer():
    status = False
    s.close()

if __name__ == '__main__':
    startServer(tcp_ip,tcp_port, 1024)
```

ARP Skript für den Raspberry PI:

```
#!/usr/bin/python
import socket
import time
import struct
import binascii
import requests
import json

#create an INET, raw socket

def startServer():
    amazons = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.htons(0x000
    a = "␣"
    b = "␣"
    c = 0
    print "Amazon␣listener␣started"
    while True:
        packet = amazons.recvfrom(2048)
        ethernet_header = packet[0][0:14]
        ethernet_detailed = struct.unpack("!6s6s2s", ethernet_header)
        arp_header = packet[0][14:42]
        arp_detailed = struct.unpack("2s2s1s1s2s6s4s6s4s", arp_header)
        # skip non-ARP packets
        ethertype = ethernet_detailed[2]
        if ethertype != '\x08\x06' or socket.inet_ntoa(arp_detailed[6]) != "192.1
            continue
        else:
            c = c + 1
            if c % 2 == 0 and c != 0:
                print "Enter␣to␣DB"
                url = "http://localhost/api/cart/1" #1 is demo  senderid
                headers = {'Content-type': 'application/json', 'Accept': 'text/pl
                try:
                    r = requests.post(url, data={}, headers=headers, timeout=10)
                    if r.status_code == 200:
                        #conn.send("200")
                        print("Enter␣data␣via␣REST:␣Success␣%i" %r.status_code)
                    else:
                        #conn.send(str(r.status_code))
                        print("Enter␣data␣via␣REST:␣Error:␣%i" %r.status_code)
```

```
                except requests.exceptions.RequestException as e:
                    print("Error %s" %e)

if __name__ == '__main__':
    startServer()
```

Service Skript für den Raspberry PI:

```
#!/usr/bin/python
import udp
import tcp
import amazon
import time
import struct
from threading import Thread
import subprocess
from flask import Flask
app = Flask(__name__)

class udpClass(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
    def run(self):
        udp.startServer('192.168.1.1',8888,1024)

class tcpClass(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
    def run(self):
        tcp.startServer('192.168.1.1',5005,1024)

class amazonClass(Thread):
   def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
   def run(self):
        amazon.startServer()

class myApp(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
```

```
    def run(self):
        app.run(port=8080)

global cudp
global ctcp
global camazon
camazon = amazonClass()
cudp = udpClass()
ctcp = tcpClass()

@app.route('/api/server/udp/start')
def udpstartrest():
    global cudp
    global ctcp
    if cudp.is_alive() == False:
        cudp = udpClass()
        print("test %r" %cudp.is_alive())
    server_status = '{"status":%r}' %(cudp.is_alive())
    res = app.response_class(server_status)
    res.headers["Content-Type"] = "application/json"
    return res

@app.route('/api/server/status')
def statusrest():
    global ctcp
    global cudp
    server_status = '{"udp":%r,"tcp":%r}' % (cudp.is_alive(), ctcp.is_alive())
    res = app.response_class(server_status)
    res.headers["Content-Type"] = "application/json"
    return res

@app.route('/api/server/tcp/start')
def tcpstartrest():
    global cudp
    global ctcp
    if ctcp.is_alive()==False:
        ctcp = tcpClass()
    server_status = '{"status":%r}' %(ctcp.is_alive())
    res = app.response_class(server_status)
    res.headers["Content-Type"] = "application/json"
    return res
```

```
if __name__== '__main__':
    appthread = myApp()
    while True:
        pass
```

MySQL Skript

Das Skript zur Einrichtung der Datenbank

```
--
-- Tabellenstruktur fuer Tabelle `carts`
--

CREATE TABLE IF NOT EXISTS `carts` (
  `id` int(11) NOT NULL,
  `productid` int(11) NOT NULL,
  `quantity` int(11) NOT NULL,
  `updated_at` date NOT NULL,
  `created_at` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Tabellenstruktur fuer Tabelle `productbuttons`
--

CREATE TABLE IF NOT EXISTS `productbuttons` (
  `id` int(11) NOT NULL,
  `productid` int(11) NOT NULL,
  `senderid` int(11) NOT NULL,
  `updated_at` date NOT NULL,
  `created_at` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Tabellenstruktur fuer Tabelle `products`
--

CREATE TABLE IF NOT EXISTS `products` (
  `id` int(11) NOT NULL,
```

```
  `name` varchar(30) NOT NULL,
  `price` float DEFAULT NULL,
  `updated_at` date NOT NULL,
  `created_at` date NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------


--
-- Tabellenstruktur fuer Tabelle `senders`
--

CREATE TABLE IF NOT EXISTS `senders` (
  `id` int(11) NOT NULL,
  `created_at` date DEFAULT NULL,
  `updated_at` date NOT NULL,
  `comment` varchar(600) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Indizes der exportierten Tabellen
--


--
-- Indizes fuer die Tabelle `carts`
--
ALTER TABLE `carts`
  ADD PRIMARY KEY (`id`),
  ADD KEY `id` (`id`);

--
-- Indizes fuer die Tabelle `productbuttons`
--
ALTER TABLE `productbuttons`
  ADD PRIMARY KEY (`id`),
  ADD KEY `id` (`id`);

--
-- Indizes fuer die Tabelle `products`
--
ALTER TABLE `products`
  ADD PRIMARY KEY (`id`),
```

```
  ADD KEY `id` (`id`);

--
-- Indizes fuer die Tabelle `senders`
--
ALTER TABLE `senders`
  ADD PRIMARY KEY (`id`),
  ADD KEY `id` (`id`);

--
-- AUTO_INCREMENT fuer exportierte Tabellen
--


--
-- AUTO_INCREMENT fuer Tabelle `carts`
--
ALTER TABLE `carts`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT fuer Tabelle `productbuttons`
--
ALTER TABLE `productbuttons`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT fuer Tabelle `products`
--
ALTER TABLE `products`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
--
-- AUTO_INCREMENT fuer Tabelle `senders`
--
ALTER TABLE `senders`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT;
```

Konfigurationsdateien

Nginx Konfiguration

Die Konfigurationsdatei des Nginx Webservers.

```
user www-data;
worker_processes 1;
pid /run/nginx.pid;
```

```
events {
        worker_connections 128;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##

        sendfile on;
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

        access_log /var/log/nginx/access.log;
        error_log /var/log/nginx/error.log;

        ##
        # Gzip Settings
        ##
```

```
        gzip on;
        gzip_disable "msie6";

        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript
text/xml application/xml application/xml+rss text/javascript;

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}
```

Hostapd Konfiguration

Die Konfigurationsdatei von hostapd

```
interface=wlan0
#driver=rtl8192cu

ssid=wlantest
hw_mode=g
channel=1
wmm_enabled=1
country_code=DE
ieee80211d=1
auth_algs=1

wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
wpa_passphrase=testwlan
```

Interfaces Konfiguration

Die Konfigurationsdatei “interfaces”

```
# Localhost
auto lo
iface lo inet loopback

# Ethernet
auto eth0
allow-hotplug eth0
iface eth0 inet manual

# WLAN-Interface
auto wlan0
allow-hotplug wlan0
iface wlan0 inet static
address 192.168.1.1
netmask 255.255.255.0

# hostapd und dnsmasq neu starten
up service hostapd restart
up service dnsmasq restart
```