Autarke Sprachsteuerung eines Smarthome Servers auf Linux Basis

Schule: Werner von Siemens Schule Hildesheim

Klasse / Fach: WBGT19IT
Fachpraxis Informationstechnik

Betreuende Lehrkräfte: Lindner, Manuel *(FPI)*
Tischmann, Oliver *(FPI)*
Schleisieck, Wiebke *(Deutsch)*

Verfasser: Hunold, Julius *(WBGT19A)*
Porsch^1, Daniel *(WBGT19A)*
Prelle, Marcel *(WBGT19B)*

Ort, Datum: Hildesheim
9. Juni 2021

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Es kommt immer häufiger vor, dass bei bekannten Smarthome und Sprachsteuerungsdiensten Daten geleakt werden und mit diese nicht mit dem vollen Datenschutz behandelt werden, deswegen befassen wir uns im Folgenden mit der Fragestellung, ob es möglich ist, ein Smarthome System selbstständig aufzubauen, welches eine Sprachsteuerung besitzt und autark ohne Internetverbindung funktionieren kann.

Diese Arbeit verfolgt das Ziel, ein autarkes Smarthome System mit einer offline Spracherkennung zu entwickeln, welche sowohl einsatzfähig, praxistauglich als auch mit Linux kompatibel ist. Um das Ziel zu erreichen, entwickeln wir mittels Python und Mikrocontrollern ein eigenes System.

In Kapitel 2 wird die Planung und die Umsetzung der einzelnen Module mittels ESP2866 näher beschrieben. Zudem wird die Darstellung der ausgelesen Daten und die Einsatzmöglichkeiten der Geräte näher erläutert.

Im nächsten Kapitel steht die Schnittstelle zwischen den Smarthome-Geräten und dem Smarthomeserver/der Sprachsteuerung im Mittelpunkt. Dabei wird der generelle Nutzen einer API, die Gründe für eine solche Nutzung und die Umsetzung näher beschrieben.

Das vierte Kapitel handelt über den Webserver und die dazugehörigen Features. Nach einer kurzen Einleitung folgt die Beschreibung unseres alten Lösungsansatzes, der auf Pythons "socket" basiert, zusammen mit der Begründung, weshalb wir uns für eine andere Lösung entschieden haben. Im weiteren Verlauf wird der Webserver "Flask" erklärt, zusammen mit den Plugins, die wir für die Webserver-Lösung nutzen. Zum Ende des Kapitels wird auf den Aufbau des Server-Dateisystems und den der Website eingegangen.

Im Kapitel 5 wird sich mit der Sprachsteuerung befasst. Dabei steht die technische Funktionsweise einer „automatic speech recognition" und einem „text-to-speech"-System genauerer beschrieben, sowie der Vergleich von Online- und Offlinesystemen ein wesentlicher Bestandteil des Kapitels ist. Abschließend wird unsere Umsetzung erklärt und auf das Programm genauerer eingegangen.

Das vorletzte Kapitell handelt von der Arbeitsdokumentation und umfasst unsere Arbeitsteilung und die zeitliche Planung der einzelnen Schwerpunkte.

Die vorliegende Facharbeit endet mit einem abschließenden Fazit, das die Ergebnisse zusammenfassend reflektiert. Des Weiteren wird zum Ende unsere Arbeitserfahrungen, Eindrücke und Verbesserungsvorschläge beschrieben.

2 Arduino Geräte/Arduino Server

Wir benutzten sogenannte Module, welche eine drahtlose Verbindung zu einem Webserver erstellen können, um diese in verschiedenen Orten/Räumen platzieren zu können. Ein Modul besteht aus einem Mikrocontroller und einer Lochrasterplatine, welche mithilfe zweier Buchsenleisten mit dem Mikrocontroller verbunden beziehungsweise zusammengesteckt werden. Wie bereits erwähnt benutzen wir für unsere Module, da wir mehrere Module haben, mehrere NodeMCU ESP8266 als Mikrocontroller. Für jedes Modul gibt es unterschiedliche elektronische Komponenten zum Beispiel ein Temperatur Sensor oder eine RGB-LED welche an die verschiedenen Lochrasterplatinen gelötet wurden, jede Platine hat neben der zum Beispiel verwendeten RGB-LED zwei weitere LEDs, welche als Status LEDs verwendet werden, jeweils eine Rote und Grüne 3mm LED. Die Rote Status LED gibt an, ob das Modul eingeschaltet ist und die Grüne LED zeigt, ob es mit dem Netzwerk verbunden ist, wenn sie blinkt, ist das Modul noch nicht mit dem Netzwerk verbunden, leuchtet sie ohne Unterbrechung ist das Modul vollständig mit dem Netzwerk verbunden. Um diese mit dem Mikrocontroller zu steuern wurden Steck und Schaltpläne entworfen, welche im Anhang zu finden sind. Ebenfalls wurde mithilfe des ohmschen Gesetzes die notwendigen Widerstände für die Rote und Grüne Status LED berechnet, für die Rote LED wird ein Widerstand von mindestens 680Ω benötigt und für die Grüne ein Widerstand von mindestens 560Ω. Auch hat jedes einzelne Modul sein eigenen Quellcode, wodurch uns ermöglicht wird die einzelnen elektronischen Komponenten auszulesen und/oder zu steuern.

2.1 Erklärung des ähnlichen Quellcodes

Da es innerhalb der verschiedenen Module im Quellcode Ähnlichkeiten gibt und wir diese nicht in jedem Modul nochmal separat erklären wollen, fassen wir diese in den folgenden Absätzen zusammen. Da wir verschiedene Module mit unterschiedlicher Quellcode länge haben können die Zeilen Angaben variieren. Der hier abgebildete Beispiel Quellcode stammt vom Switch-Modul.

[FEHLT]

In dem oben abgebildeten Quellcode wird zunächst in der 2 Zeile die benötigte Programmierbibliothek ESP8266WIFI etabliert, wobei beim Temperatur-Modul noch weitere Bibliotheken vorkommen, welche in der Temperatur-Modul eigenen Erklärung erklärt werden. Diese Programmierbibliothek sorgt dafür das man die WiFi-Funktionen des Integrierten WLAN-Moduls des Mikrocontrollers verwenden kann. Danach werden in den Zeilen 9 und 10 die Anmelde Daten für das WiFi permanent festgelegt, welches mithilfe der const Syntax erfolgt, welche besagt das die definierten Werte nicht mehr verändert werden können. Auch wird der Webserver Port und die variable „header" festgelegt, um die Kommunikation zwischen den Geräten und auf der Webseite zu gewährleisten. Zum Schluss wird ein Time-out definiert, um ein mögliches Time-out zu erfassen. Ein Time-out ist eine

Fehlermeldung, welche nach einer vorher festgelegten Zeit auftritt, wenn das Gerät, in unserem Beispiel das Modul, keine Antwort vom Server, anderen Programmen oder Geräten erhält.[1] Dafür benötigt man die Aktuelle Zeit sowohl als auch die maximale Zeit, um so die Zeit der Antwort zu berechnen.

[FEHLT]

In dieser Abbildung des Quellcodes wird der Inhalt der setup() Funktion aller Module dargestellt. Zuvor muss erstmal geklärt werden was überhaupt die setup() Funktion bewirkt. Die setup() Funktion wird benutzt um die Variablen, Pin Belegungen, Programmierbibliotheken und vieles ähnliches innerhalb der geschweiften Klammern zu initialisieren und somit den zu programmierenden Mikrocontroller funktionsfähig zu machen. Ebenfalls wird diese Funktion nur einmal ausgeführt, und zwar wenn das Gerät ein- oder neugestartet wird. In der ersten Zeile wird der Serielle Monitor gestartet damit man zeitgleich eine visuelle Ausgabe ausgewählter Daten hat. Danach wird mit einer Zeitverzögerung von 500 Millisekunden die Verbindung zum WiFi hergestellt sowohl als auch die Benachrichtigung der Verbindung mit dem WiFi im seriellen Monitor. Auch wird Zeitgleich der angeschlossene Pin der Grünen Status LED als „OUTPUT" deklariert. Daraufhin wird mithilfe einer Schleife so lange versucht das Modul mit dem Netzwerk zu verbinden, bis es verbunden ist. Innerhalb dieser Schleife entsteht auch das „Blinken" der LED, indem diese LED mit einer Zeitverzögerung von 500 Millisekunden ein- und ausgeschaltet wird. Wurde eine Verbindung mit dem Netzwerk hergestellt wird die Schleife verlassen und die Grüne Status LED permanent eingeschaltet. Auch wird die erfolgreiche Verbindung und die zugeteilte IP-Adresse im seriellen Monitor präsentiert. Daraufhin wird der Webserver in Zeile 54 gestartet, um eine eigene Webseite zu erstellen in welcher dann notwendige Informationen für andere Module und/oder APIs abgebildet und gespeichert werden.

[FEHLT]

In dem oberhalb abgebildeten Quellcode wird die loop() Funktion gestartet, innerhalb dieser Funktion findet man das eigentliche Programm der Module. In Zeile 65 des Quellcodes wird auf eine gültige Serververbindung gewartet, wenn diese zutritt, wird das eigentliche Programm der Module ausgeführt. Zeitgleich wird im seriellen Monitor eine Benachrichtigung ausgegeben das eine „Neue Verbindung zum Server" besteht.

[FEHLT]

In diesen Zeilen des Quellcode wird die Webseite des eigenen Moduls erstellt, in welcher die Daten des jeweiligen Modules dargestellt werden.

[1] Quelle: Time-out

[FEHLT]

Diese oben abgebildeten Zeilen sorgen dafür das sämtliche Schleifen und Funktionen geschlossen oder gestoppt werden.

2.2 Switch-Modul

Mithilfe des Switch-Moduls kann der Zustand unterschiedlicher elektronischer Komponenten, durch ein digitales Signal, der Status eines GPIO-Pins (I/O-Ports des Mikrocontroller) verändert werden, welches zwischen zwei Zuständen wechselt. In unserem Beispiel haben wir ein Switch-Modul mit einer nicht dimmbaren LED gebaut, welche nur ein und aus geschalten werden kann. Um diese LED mit dem Mikrocontroller ohne Probleme benutzen zu können haben wir jeweils ein Steck- und Schaltplan entworfen, welche im Anhang zu finden sind. Auch ein notwendiger Widerstand wurde mithilfe des Ohmeschen Gesetzes berechnet, der berechnete Widerstand bei der Grünen 5mA LED ergab mindestens 43Ω.

2.2.1 Erklärung des Quellcodes

Der hier erklärte Quellcode ist auch im Anhang in voller Länge zu finden.

[FEHLT]

Der oben abgebildete Quellcode definiert die benötigten Ports für das Switch-Modul, in dem Fall wird aus Port 5 „switch" und aus Port 15 „led_wifi". Auch wird in Zeile 24 die Variable „switch_state" definiert.

[FEHLT]

In dem oberhalb abgebildeten Quellcode handelt es sich um ein Teil der loop() Funktion des Switch-Modul. Innerhalb dieser Zeilen wird der Status der Variable „switch_state" abhängig vom Befehl des Benutzers auf „true" oder „false" gesetzt.

[FEHLT]

In dem oben abgebildeten Quellcode handelt es sich ebenfalls um ein Teil der loop() Funktion. Hier werden zu Beginn die vorher veränderte Variable „switch_state" auf der Modul-eigenen Webseite dargestellt, damit unsere eigene API diese verarbeiten kann. Daraufhin wird der Status des Switch Ports verändert, in dem Fall von „HIGH" auf „LOW" oder umgekehrt. Auch wird diese Veränderung visuell im Seriellen Monitor dargestellt.

2.3 LED-Modul

Durch das LED-Modul kann eine dimmbare LED angesteuert werden. Der beigeführte Quellcode kann auf Wunsch die dimmbare LED ein- und ausschalten sowohl als auch die Leuchtstärke verstellen. Dieses Modul ist nicht mit dem Switch-Modul vergleichbar, da es auch mit PWM-Signalen arbeitet. Die PWM („Pulse Width Modulation", deutsch Pulsweitenmodulation oder Pulsbreitenmodulation) ist eine digitale Modulationsart, bei der eine Technische Größe konstant zwischen zwei Werten gewechselt wird, wodurch ein Impuls generiert werden kann.[2] Um diese LED mit dem Mikrocontroller ohne Probleme benutzen zu können wurden Steck- und Schaltplan entworfen, welche im Anhang zu finden sind. Auch ein notwendiger Widerstand wurde mithilfe des Ohmeschen Gesetzes berechnet, der berechnete Widerstand bei der 5mA LED ergab mindestens 43Ω.

2.3.1 Erklärung des Quellcodes

Der hier erklärte Quellcode ist auch im Anhang in voller Länge zu finden.

[FEHLT]

In dem oben abgebildeten Quellcode werden zuerst die benötigten Ports definiert, in dem Fall wurde aus Port 5 „led" und aus Port 15 „led_wifi". In den darauffolgenden Zeilen werden auch die notwendigen Variablen definiert.

[FEHLT]

In dem oberhalb abgebildeten Quellcode wird innerhalb der setup() Funktion der „pinMode" des Pins „led", welcher zuvor definiert wurde, als Output definiert, um so die LED zu steuern.

[FEHLT]

Innerhalb dieser Zeilen wird der Status der Variable „led_state" abhängig vom Befehl des Benutzers auf „true" oder „false" gesetzt.

[FEHLT]

Innerhalb der oben abgebildeten Zeilen wird die Leuchtstärke der LED auf 100% definiert, wobei der Wert neben dem „GET" innerhalb der if-Abfrage die gewünschte Leuchtstärke beziehungsweise den Dimmfaktor angibt. Wir haben uns entschieden die Leuchstärke in zehn Prozent Segmenten aufzuteilen umso eine etwas genauere Dimmbarkeit herzustellen, diese lässt sich natürlich auch ausbauen. Da wir elf Leuchtstärkesegmente haben wiederholt sich unser Quellcodeabschnitt elf-mal.

[2] Quelle PWM

2.4 RGB-Modul

Mithilfe des RGB-Moduls kann die Helligkeit als auch die Farbe einer RGB-LED verändert werden. Wie beim LED-Modul arbeitet das RGB-Modul mit den PWM werten, um Helligkeit und Farbe zu verändern. Damit die RGB-LED ohne Probleme mit dem Mikrocontroller gesteuert werden kann haben wir zuvor Steck- und Schaltpläne entworfen, welche im Anhang zu finden sind. Da jeder Farb-Pin der LED einen anderen Widerstand benötigt mussten drei Widerstände mithilfe des Ohmeschen Gesetz berechnet werden, die berechnete Widerstand 75Ω für den Roten Pin und jeweils 12Ω für den Grünen und Blauen Pin.

2.4.1 Erklärung des Quellcodes

Der hier erklärte Quellcode ist auch im Anhang in voller Länge zu finden.

[FEHLT]

In dem oben abgebildeten Quellcode werden zuerst die benötigten Ports für das RGB-modul definiert, in dem Fall wird Port 0 zu „red", „Port 4" zu „green", „Port 5" zu „blue" und „Port 15" zu „led_wifi". In den darauffolgenden Zeilen werden auch die notwendigen Variablen definiert und mit sinnvollen/benötigten Werten versehen.

[FEHLT]

Hier wird innerhalb der setup() Funktion der „pinMode" der Pins „red","green" und „blue", welche zuvor definiert wurden, als Output definiert, um so die RGB-LED Farben anzusteuern.

[FEHLT]

Innerhalb dieser Zeilen wird der Status der Variable „led_state" abhängig vom Befehl des Benutzers auf „true" oder „false" gesetzt.

[FEHLT]

In diesen Zeilen werden wie beim LED-Modul die Dimmstufen definiert und müssen dementsprechend nicht ein zweites-mal erklärt werden.

[FEHLT]

Innerhalb des oben abgebildeten Quellcode handelt es sich die darstellung der Farbe „coldwhite" mit der RGB-LED. Auch kann mithilfe des wiederholen und das Verändern der Variablen die Farben verändert werden, in unserem Quellcode können wir aktuell nur zehn Farben darstellen, und zwar die Farben „coldwhite ", „warmwhite", „red", „green", „blue", „yellow", „orange", „turquoise", „pink" und „purple".

2.5 Temperatur-Modul

Mithilfe des Temperatur-Modul kann die Temperatur gemessen werden. Bei dem Temperatur Sensor handelt es sich um ein DS18S20 vom Hersteller „DALLAS SEMICONDUCTOR", welcher digitale Temperatur Messungen in Celsius durchführt. Um den Sensor ohne Probleme mit dem Mikrocontroller zu benutzen wurden Steck- und Schaltplan entworfen, welche im Anhang zu finden sind. Auch wurde ein notwendiger Widerstand mithilfe des Ohmeschen Gesetzes berechnet, der berechnete Widerstand für die sichere Inbetriebnahme des Sensors ergab mindestens 2,7KΩ, wir benutzten aufgrund des Fehlens von 2,7KΩ Widerständen ein Widerstand von 3,3KΩ.

2.5.1 Erklärung des Quellcodes

Der hier erklärte Quellcode ist auch im Anhang in voller Länge zu finden.

[FEHLT]

In der Abbildung des Quellcodes werden benötigte Programmbibliotheken initialisiert, in dem Fall die OneWire, die DallasTemperature und die ESP8266WIFI Bibliothek. Mithilfe der OneWire und DallasTemperature Bibliotheken kann der Temperatur Sensor benutzt werden und gelesen werden.[3, 4] Die ESP8266 Bibliothek wurde im ähnlichen Quellcode bereits erklärt. Folgend wurden die benötigten Ports definiert, und zwar das der „Port 16" zu „data" wird und, wie bei den anderen Modulen, „Port 15" zu „led_wifi". Ebenfalls werden in den Zeilen 26 bis 29 benötigte variablen definiert.

[FEHLT]

Hier wird, wie in den anderen Modulen, der „data" Pin als „INPUT" definiert.

[FEHLT]

In diesen Zeilen wird der Sensor nach der Temperatur abgefragt, diese Temperatur wird zeitgleich im seriellen Monitor dargestellt und in einer separaten Variablen abgespeichert, damit diese zukünftig leichter abrufbar ist.

3 API

Die verkürzte Schreibweise API steht für „Application Programming Interface" und beschreibt eine Schnittstelle zwischen verschiedenen Programmen. Dabei wird es zur Kommunikation zwischen verschiedenen Programmen verwendet[5].

[3] Quelle OneWire
[4] Quelle DallasTemperature
[5] (2)

Abbildung 1 Funktionsweise einer API

Ein Beispiel wäre das Bauen mit Klemmbausteinen. Sie müssten nicht jedes Baustein selbst herstellen sonst greifen auf ein Sortiment ans Bauteilen zurück und müssten die Steine nur zusammenstecken. Das Wesentliche einer API hat diese Funktion und erleichtert Ihnen die Arbeit um ein vielfachen und Sie können sich auf das Wesentliche konzentrieren[6].

Dabei ist die API keine Datenbank oder ein einzelner Server, auf dem die Daten abgerufen werden können, sondern lediglich ein Programm (Code) der die Beschaffung der Daten vereinfacht. Er „erklärt" dem eigentlichen Programm nur, wie es an die Daten kommt. Die Rechenleistung/Verarbeitung der Daten ist je nach API anders geregelt. Einige benötigen einen Input und verarbeiten die Daten auf eigenen Servern und stellen am Ende das Ergebnis zur Verfügung. Bei anderen APIs werden die Daten lokal auf dem eigenen System verarbeitet. Zusammenfassend kann man sagen, dass sowohl offline als online Schnittstellen zur für ein vielfachen vereinfacht werden[7].

3.1 Gründe für eine API

Bei der Benutzung von APIs werden Aufgaben verteilt und der Programmcode modularisiert. Daraus ergeben sich folgende Vorteile:

Bessere Stabilität von Software

Die Nutzung von Schnittstellen ermöglicht die Aufgabenteilung innerhalb eines Projektes. Diese Unterprogramme laufen unabhängig voneinander, sodass bei einem Fehler im Code die restlichen Module oftmals nicht betroffen sind, womöglich sie dennoch in Ihrer Funktionsweise eingeschränkt sein mögen. Diese Stabilität ist oft nützlich, wenn man an größeren Projekten, oder Projekten mit hoher Bedeutung für seine Nutzer arbeitet.

Fehler lassen sich leichter eingrenzen

Modularer Programmcode ermöglicht eine vereinfachte Fehlererkennung, da sich auftretende Probleme leichter eingrenzen und somit beheben lassen. Dies vereinfacht den Workflow und verhindert

[6] (5) -› Bedeutung von API - Beispiel
[7] (5)

eine Reihe von potenziellen weiteren Fehlern, die auftreten könnten, wenn man versucht Bugs in einem einzigen Programm zu beheben.

Einfachere Weiterentwicklung / Verbesserung

Durch die Aufteilung des Codes wird ein Projekt übersichtlicher und die Bereiche der Darstellung sind klar von denen der Funktionalität abgegrenzt. Hierdurch wird auch die Weiterentwicklung einzelner Bestandteile des Projektes vereinfacht. Sollten zum Beispiel Änderungen am UI notwendig sein, so ist es deutlich unwahrscheinlicher, dass man bei der Arbeit an diesen Änderungen neue Fehler in das Programm bringt. Ebenso kann man auch das Programm um seine Funktionalität erweitern, ohne das UI zu beeinträchtigen. Diese voneinander getrennten Workflows sind gegenüber einem einzigen Programm zu präferieren.

Verschiedene Programmiersprachen

Modulare Projekte ermöglichen auch die Nutzung von verschiedenen Programmiersprachen, sodass man für den jeweiligen Anwendungszweck die ideale Sprache wählen kann. Dies ermöglicht eine effizientere Lösung, kann die Performance verbessern oder schlichtweg eine Menge Code sparen.

3.2 Erklärung des Codes

[FEHLT]

4 Webserver

4.1 Die Auswahl eines geeigneten Servers für das Smarthome-Projekt

Ein Grundbestandteil unseres Smarthome-Systems ist ein Server, der die Möglichkeit bieten muss, über das lokale Netzwerk mit Steuerungscontrollern für Smarthome-Geräte zu kommunizieren, in diesem Fall mehrere Arduino-Mikrocontroller, und der eine Steuerung über eine „Speech-To-Text"-Spracherkennung ermöglicht. Dazu muss der Server unter dem Betriebssystem Linux lauffähig sein. Im Laufe der Durchführung des Projekts wurde hauptsächlich mit zwei Netzwerkinterfaceobjekten gearbeitet. Zum einen „socket", das Netzwerkinterface, welches Bestandteil der Programmiersprache Python ist und worauf die Anfänge unseres Projekts basierten. Zum anderen das Web-Framework „Flask", welches in Python geschrieben wurde und darauf aufbaut. Flask bindet zudem die „Jinja - Engine", zum Generieren und Erstellen von Webdokumenten in Python, und die Kommunikationsbibliothek „Werkzeug" ein, das zum Erleichtern der Kommunikation zwischen Webserver und Webanwendung genutzt werden kann.

4.2 Die ursprüngliche Realisierung mit Pythons Netzwerkinterfaceobjekt „socket"

Zum Start des Projektes lag der Fokus hauptsächlich auf dem Netzwerkinterfaceobjekt „socket". Mit socket ist es möglich Netzwerkkommunikation selbst zu planen und mit Hilfe der Programmiersprache Python zu realisieren. socket ist ein dynamisches Netzwerkobjekt, welches sowohl als Server, als auch als Client implementiert werden kann. Die ursprüngliche Realisierung hat es vorgesehen, verschiedene Dateien, in diesem Fall Textdokumente, zu nutzen, um Daten, die über das lokale Netzwerk gesendet werden, zu verwalten.

Von der Client-Server-Software unabhängig sind die Sprachsteuerung und die Steuerungssoftware der Smarthome-Geräte. Smarthome-Geräte sind über komplette Räume verwaltet und werden gemeinsam durch eine Steuerungssoftware gesteuert. Der Client, der mit Hilfe von socket realisiert ist, befindet sich auf demselben Gerät wie die Steuerungssoftware. Ein Raum des Smarthome-Systems bildet und nutzt einen gemeinsamen Client.

Der Server dient als zentraler Punkt des Smarthome-Systems und die Spracherkennung befindet sich auf demselben Gerät. Der Server verwaltet den Client jedes Raumes in einem eigenen Thread, damit das Verwalten mehrerer Räume zur selben Zeit funktionieren kann. Durch die Trennung von Server-Software und Sprachsteuerung können Server und Sprachsteuerung über das Dateisystem des Servers interagieren.

Der Start der Netzwerkkommunikation geht vom Client aus, der sich nach dem Start der Steuerung direkt mit dem Server verbindet. Ein Problem bei der Realisierung ist hier, dass der Server bereits erreichbar sein muss, ansonsten beendet sich die Client-Software durch ein Timeout nach kurzer Zeit selbst. Nach erfolgreichem Verbinden sendet der Client seine Identifikation, welches der Raumname ist, damit der Server den Client zuordnen kann. Nach seiner Identifikation sendet der Client im Anschluss seine Daten, die er aus seinem Dateisystem liest. Der Server legt nun einen eigenen Ordner für den Client im eigenen Dateisystem an, erstellt die Dateien und speichert darin die Daten der Smarthome-Geräte des Clients. Dieses Senden der Daten wiederholt der Client nun in einer Endlosschleife. um die Kommunikation mit dem Server aufrecht zu erhalten. Dies sorgt ebenfalls dafür, dass die Daten der Smarthome-Geräte des Clients auf dem Server immer aktuell sind. Nachdem der Server die Daten gespeichert hat, vergleicht er die Einstellungen der Geräte zum Vergleich davor. Liegen hier Änderungen vor, antwortet der Server mit diesen und überträgt sie zum Client, dessen Steuerungssoftware die Einstellungen anwendet.

Erfolgt nun eine Anfrage durch die Sprachsteuerung, z.B. für die Raumtemperatur, liest die Sprachsteuerung auf dem Server den Temperaturwert aus der jeweiligen Datei des jeweiligen

Raumes aus, der immer aktuell ist und gibt diesen wieder. Bei einer Änderung an einem Gerät, um z.B. die Rollladen eines Smarthome-Raumes herunterzufahren, schreibt die Sprachsteuerung die Änderung in die jeweilige Datei. Der Server erkennt die Änderung bei der nächsten Prüfung der Einstellungen und gibt diese an den Client weiter, der sie bei sich speichert. Die Steuerungssoftware bemerkt dann ebenfalls die Änderung in der jeweiligen Datei, wendet die neue Einstellung an und fährt die Rollladen herunter. Um Bloating des Dateisystems des Servers zu vermeiden, löscht der Server sämtliche Daten des Clients aus seinem Dateisystem, nachdem dieser die Verbindung getrennt hat, oder die Verbindung abgebrochen ist. Aus diesem Grund muss jeder Client des Smarthome-Systems vorab konfiguriert und angepasst werden, denn die Start-Einstellungen des Clients sind ebenfalls die Start-Einstellungen, die der Server erhält.

4.3 Das Microframework „Flask" als Webserver für das Projekt

Die Planung innerhalb des Projekts hatte bereits sehr früh einen Webserver, welcher eine Website die über alle gängigen Browser, wie Google Chrome, Mozilla Firefox und für Mobilgeräte Safari auf dem iPhone, aufrufbar macht. Die Website dient als Hauptsteuerungselement des Smarthomes, die zusätzlich mit einer Spracherkennung ausgestattet wird, um eine Steuerung zusätzlich durch gesprochene Befehle zu ermöglichen.

Zusätzlich soll die Website die Möglichkeit bieten, mehrere Nutzer anlegen zu können, für die sich bestimmte Geräte des Smarthomes zuweisen und für die sich eigene Favoriten, die auf der Startseite der Steuerung, nach einem Login, angezeigt werden und sich schnell und leicht konfigurieren lassen. Die gesamte Verwaltung der einzelnen Räume, dem Erstellen neuer Einträge für Geräte und Räume und der Zugriffsverwaltung bleibt einem Admin-Profil vorbehalten, um die Sicherheit zu erhöhen und eine übersichtliche Verwaltung zu gewährleisten.

Das Microframework „Flask", welches auf Python aufbaut, bietet eine passende Basis für diese Website-Steuerung. Zudem können mit Hilfe der session-Methode von Flask einzelne Nutzer der Website zugeordnet werden und so eine Profil-Verwaltung möglich gemacht werden. Dadurch, dass es sich bei Flask nur um ein Framework handelt, baut es auf benutzerdefinierte Websitekomponenten im Front-End und im Back-End, die selbst zu implementieren sind, auf. Flask beinhaltet nur absolut notwendige Komponenten für eine fehlerfreie Ausführung des Frameworks. Flask ist, durch den Aufbau mit Python ebenfalls auf allen gängigen Betriebssystemen, wie Windows, Linux und Mac lauffähig, solange das Betriebssystem die aktuelle Version von Python unterstützt.

Zusätzlich zu Flasks eigenen Funktionen stehen außerdem die Funktionen der „Jinja-Engine" zur Verfügung. Die Jinja-Engine ist direkt in Flask implementiert und bietet die Möglichkeit, HTML-Dokumente für Webbrowser zu generieren. Übliche HTML-Formatierung wird hierbei mit einer Programmiersprache, die Python ähnelt, ersetzt und ermöglicht so die Erstellung von dynamischen Webapplikationen

über Python. Eine weitere Implementierung in Flask ist das Toolset „Werkzeug", welches eine Funktions- und Befehlsbibliothek bietet, die es möglich macht, Webapplikationen dynamisch miteinander kommunizieren zu lassen und Probleme, die durch fehlende Kompatibilität auftreten zu minimieren. Während der Entwicklung bietet Flask einen umfangreichen Debug-Modus, der Fehler bei der Programmierung überschaubar und präzise anzeigt und die Fehlerbehebung erheblich erleichtert.

4.4 Die Jinja-Engine für den Aufbau der Smarthome-Websites

Mit Hilfe der Jinja-Engine, die in das Flask-Mikroframework integriert ist, lassen sich voll funktionsfähige Websites, die dem Benutzer bei Seitenaufruf zugesendet werden, erstellen. Die Besonderheit ist, dass dies mit einem Template oder einer Blaupause einer Website geschieht. Das „Template"-Modul, welches Teil der Jinja-Engine ist, ermöglicht es Website-Elemente in einer Syntax, der der Programmiersprache Python sehr ähnelt, auszulegen und bei Anfrage generieren zu lassen. Funktionsweisen zum Erstellen von Webdokumenten setzen hier kein Webdokument voraus. Aus jedem beliebigen Dokument, welches ein kompatibles Anzeigeformat, wie ASCII, benutzt, kann von der Jinja-Engine in ein Webdokument umgewandelt und über einen Browser ausgegeben werden. Dadurch, dass Websites auf Anfrage durch eingebettete Befehle generiert werden, ist es möglich, Schleifen, wie die for-Schleife, oder Abfragen, die ein Wahr oder Falsch wiedergeben, in das Website-Template zu implementieren. Diese Funktionsweisen sorgen dafür, dass Website-Code eingespart und dynamisch angelegt werden kann.

Alle Jinja-Formatierungseinbettungen werden sowohl mit einer geschwungenen Klammer gestartet als auch beendet und sie geben somit die Grenzen der Einbettung an. Die Einbettungen sind im Normalfall nicht Teil des finalen Webdokuments, welches zum Nutzer gesendet wird. Innerhalb der begrenzenden Klammern gibt es Zuweisungszeichen, wie ein Prozentsymbol oder ein weiteres Klammerpaar, die festlegen, wie die Einbettung genutzt wird.

Die Prozent-Zeichen werden verwendet, wenn in der Einbettung ein Statement benutzt wird. Dies können Schleifen, wie eine for-Schleife, oder Abfragen, wie eine if-Abfrage, sein. Zum anderen können hier auch Code-Blöcke, die aus anderen Templates stammen, erneut benutzt, eingefügt und, je nach Anwendung, manipuliert werden. Statements dienen zum Aufbauen der Website und ermöglichen Funktionen, die in herkömmlicher HTML-Sprache nicht möglich sind. Befindet sich in der Einbettung ein weiteres Paar geschwungene Klammern, bedeutet dies, dass sich in der Einbettung generierter Text in Form eines Strings befindet. Der Code innerhalb dieser Einbettungen muss Text, Zahlen, oder Zeichen wiedergeben, die dann in die Website integriert werden. Zum Beispiel kann hier auf eine vorher angelegte Variable referenziert werden, oder aus einem Daten-Array können angegebene Daten ausgelesen und über das Webdokument ausgeben werden.

Ein optionaler Operator in der Syntax ist der vertikale Balken ("|"). Dieser Operator kann genutzt werden, um Zusatzformatierungen, wie Schriftgröße, während der Generierung zu übergeben.

4.5 Die Implementierung von JQuery für mehr Website-Funktionalität

Wir haben uns entschieden, für das Hinzufügen von Funktionalität und zum vereinfachten Implementieren von dynamischen Design-Veränderungen der Website, die JavaScript-Library „JQuery" in unseren Website-Code zu integrieren.

JQuery fasst oft genutzte Funktionen, die in reinem JavaScript sehr große und umfangreiche Codeblöcke wären, in einfach zu bedienende neue JavaScript Objekt-Klassen und zugehörige Funktionen zusammen. So bietet JQuery z.B. die Methode „hide", mit der sich, je nach Konfiguration, einzelne oder ganze Gruppen von gleichen Website-Elementen, wie Text, mit beispielsweise einem Button auf der Website, verstecken lassen. Solche Funktionen können nicht nur Buttons hinzugefügt werden, sondern auch dem Mauszeiger, wenn sie sich beispielsweise über einem bestimmten Text befindet, oder wenn in ein Textfeld etwas bestimmtes eingetragen wird.

Diese Funktionalität ermöglicht es, dem Smarthome-Webinterface dynamische Funktionen hinzuzufügen, wie das Zeigen der Favoriten, oder das Zeigen aller Räume, was mit zwei „toggle"-Methoden, die mit dem „Alle Räume"- Button auf der Startseite des Interfaces per Klick getauscht werden kann. Diese Funktionalität ohne JQuery hinzuzufügen, würde einen umfangreichen JavaScript-Code erfordern, der sehr umständlich in das Interface zu implementieren wäre.

4.6 Der Aufbau des Dateisystems des Webservers

Die meisten erforderlichen Dateien und Zusätze für die Ausführung des Webservers befinden sich lokal auf dem Computer, der als Server dient. Extern wird der Server mit JQuery und Schriftartinformationen versorgt, dessen Quell-Adressen in den jeweiligen Basisinformationen der Websites hinterlegt sind. Diese werden im Front-End, der Teil der Website, den der Nutzer sieht, angefragt und zum Anzeigen der Website über andere Service-Provider wie Google bereitgestellt. Je nach Anfrage, greift Flask während der Ausführung auf verschiedene Dateien zu, um angefragte Websites zu generieren, oder um mit Smarthome-Komponenten kommunizieren zu können, die über das Server-Dateisystem konfiguriert worden sind. Alle erforderlichen Dateien befinden sich in drei Ordnern, innerhalb des root-Verzeichnisses des Servers. Diese heißen „hub", „smarthome" und „templates".

4.6.1 Der Ordner „hub"

Innerhalb des „hub"-Orders befinden sich alle Dateien, die zum Anzeigen der sogenannten Roadmap erforderlich sind. Die Roadmap, die auf diese Weise über einen Webbrowser erreichbar ist, gibt Aufschluss über unvollendete Features der Steuerung und Features die zukünftig integriert werden. Die

Idee der Roadmap kam während des Projektes erst später auf, weshalb dieses Feature selbst nicht fertig gestellt ist.

Innerhalb des „hub"-Verzeichnisses befindet sich ein „static", und ein „templates"-Ordner, zusammen mit dem Python-Script „hub.py". Das Python-Script verbindet Informationen für die Website-Generierung der Roadmap mit Flask, um die Website über einen Webbrowser aufrufen zu können. Innerhalb des Python-Scripts wird ein Objekt der Klasse „Blueprint" erstellt und mit dem URL-Suffix „/roadmap" verbunden. Ruft ein Nutzer die Roadmap mit der Server-IP und dem Suffix auf, wird die Website generiert und im Browser angezeigt.

Die Website-Templates, aus denen die Website generiert wird, befinden sich im gleichnamigen Ordner „templates", in dem sich „base_hub.html", „hub_index.html" und „roadmap.html" befinden. „base_hub.html" beinhaltet Grundinformationen zum Aufbauen der Website im Browser und Verlinkungen zu Informationen zu bestimmten Schriftarten, die die Website benutzt, um diese anzuzeigen. Diese Grundinformationen werden mit Hilfe der Jinja-Engine in einen Informationsblock zusammengefasst, der in andere Website-Dokumente integriert werden kann. Die Datei „roadmap.html" fügt der „hub_base.html" den Inhalt der Roadmap, geschrieben in HTML und CSS, hinzu. Die Website ist nicht interaktiv und der gesamte Inhalt wird statisch dargestellt. Interaktivität, beispielsweise durch klickbare Kacheln, lässt sich aber im Nachhinein implementieren.

Der Ordner „static" beinhaltet JavaScript- und CSS-Bibliotheken, die JavaScript und CSS websitebasiert erweitern. CSS dient dazu, Websites zu designen und mit einem Style zu verstehen, JavaScript wird benutzt, um interaktive Elemente der Website hinzuzufügen.

4.6.2 Der Ordner „smarthome"

Ähnlich wie im „hub"-Ordner gibt es im „smarthome"-Ordner ebenfalls die Unterordner „templates" und „static", beide haben denselben Nutzen wie im „hub"-Ordner. Der „static"-Ordner beinhaltet CSS und JavaScript-Bibliotheken und der „templates"-Ordner beinhaltet die Website-Templates, aus der die Jinja-Engine Websites generiert. Zusätzlich befindet sich hier auch der Ordner „settings" und diverse Python-Scripts und Dateien, die Konfigurationsinformationen beinhalten, wie Text- und CSV-Dateien.

Im „settings"-Ordner befinden sich Textdateien, die zum Speichern von Einstellungen für jeden Account benötigt werden. Wird auf der Website ein neues Profil erstellt, werden hier zwei Textdateien mit dem Namen „‹profilname›.txt" und „‹profilname›_favs.txt" erstellt. Die Datei „‹profilname.txt›" speichert den Wert des gewählten Website-Designs, wie z.B. Darkmode, und die festgelegte Sprache, da wir uns entschlossen haben, sowohl Deutsch als auch Englisch als einstellbare Sprache zur Verfügung zu stellen. Die Datei „‹nutzername›_favs.txt" speichert die festgelegten Favoriten, die sich ebenfalls in den Account-Einstellungen festlegen lassen. Diese werden auf der Startseite der

Steuerungswebsite, nach erfolgreichem Login für einfache Zugänglichkeit, direkt angezeigt. Die maximale Anzahl beträgt, aus Platz- und Designgründen der Website, fünf Favoriten.

Zu den anderen Dokumenten im „smarthome"-Ordner gehören die CSV-Dateien „devices.csv", „rooms.csv" und „users.csv". Wir haben uns hier für CSV-Dateien entschieden, da diese, wie andere Konfigurationsformate z.B. XML, mit Hilfe einer bestimmten Syntax ausgelesen werden können. Die Informationen innerhalb einer CSV-Datei werden in unserem Fall durch ein Semikolon getrennt und können auf diese Weise einfacher in ein Programm eingelesen werden. In der „devices.csv" sind Informationen zu den einzelnen Geräten, die an das Smarthome-System angeschlossen sind, hinterlegt. Es werden die ID, die das Gerät innerhalb der Steuerung identifiziert und einzigartig ist, gespeichert und die lokale IP-Adresse, die für die Kommunikation mit dem Gerät über das lokale Netzwerk mithilfe der API benötigt wird, um Daten auszutauschen. Zudem ist jedem Gerät eine Typ-ID zugewiesen, mit der dem Gerät eine bestimmte Auswahl an Einstellungsmöglichkeiten innerhalb der Steuerung zugewiesen wird. Außerdem werden hier die, für Menschen auf der Website lesbaren, Bezeichnungen der einzelnen Geräte und der mit ihnen verbundener Raum gespeichert. Ähnlich aufgebaut ist auch die Datei "rooms.csv", in der die Konfiguration aller Räume innerhalb des Smarthome-Systems hinterlegt sind. Jedem Raum wird eine einzigartige ID zugewiesen und sie erhalten eine Icon-Klasse, um dem Gerät auf der Website ein passendes grafisches Icon, zur verbesserten Benutzerfreundlichkeit, zuzuweisen. Auch hier wird dem Raum ein Name sowie eine Kategorie zugewiesen, um Geräte mit gleicher Kategorie diesem Raum zuzuweisen. Die "users.csv"-Datei enthält alle registrierten Accounts auf der Website. Es werden Benutzername, den sich der Nutzer bei Erstellung des Accounts gibt, sowie das Passwort, das ebenfalls bei Accounterstellung festzulegen ist, gespeichert. Hier werden, wie in den anderen CSV-Dateien, einzelne Datenabschnitte, wie der Benutzername und das Passwort durch ein Semikolon getrennt, um dem Server-Programm mitzuteilen, wann ein bestimmter Datensatz aufhört und wann ein anderer anfängt.

Informationen zu den unterstützten Sprachen des Webinterfaces, das auf Deutsch und auf Englisch verfügbar ist, beinhaltet das Python-Script "lang.py". Hier sind die Texte des Webinterfaces in beiden Sprachen hinterlegt und werden je nach Konfiguration eingelesen. Texte sind vor der Generierung der Website nicht enthalten und müssen während des Generierungsprozesses hinzugefügt werden. Je nach angefragter Sprache wird entweder der deutsche Text bei der Generierung verwendet oder der englische.

Ein weiterer Python-Script, der für das Generieren von angefragten Websites erforderlich ist, ist "themes.py". In diesem Script sind Informationen hinterlegt, die bestimmen wie sich das Design der Website bei einem gewählten Design, z.B. Darkmode, verändert, welche Farben und Akzente in CSS zum Stylen der Website genutzt werden.

"api.py" ist ein Python-Script, den das Webinterface zum Kommunizieren mit hinterlegten Geräten verwendet. Für jedes hinterlegte Gerät werden, im Backend des Servers, Anfragen generiert und diese mit Hilfe der "devices.csv"-Datei zur festgelegten IP gesendet. Das angefragte Gerät antwortet und teilt der API angefragte Daten mit, die dann im Webinterface sichtbar dargestellt, wie Temperaturwerte, oder zu Konfigurationszwecken dienen, wie das Prüfen auf Verfügbarkeit eines hinterlegten Gerätes. Mehr zu der Funktionsweise der API, die wir für das Smarthome-System entworfen haben, befinden sich in der Sektion der API.

4.6.3 Das Python-Script "smarthome.py"

Das umfangreichste Python-Script in diesem Ordner ist "smarthome.py". Dieses Script ist die Basis des Webinterfaces. Die Main-Page und Jede Sub-Page der Steuerung wird hier definiert und sind von hier aus ineinander gekoppelt.

Das Script beginnt damit, erforderliche Komponenten, wie externe Bibliotheken und Informationen aus anderen Python-Scripts, wie „lang.py" und „themes.py" zu importieren, um sie für sich benutzbar zu machen. Danach wird, mit Hilfe der Klasse „Blueprint", ein Netzwerk aus Template-Dateien verknüpft, das zum logischen Ablauf des Webinterfaces erforderlich ist. Anschließend wird dieser Blueprint, über die Variable „sh", dem Start-Scripts des Servers zur Verfügung gestellt. Nach Erstellung des Blueprints wird dieser mit der Methode „secret_key" für eine geschützte und verschlüsselte Datenübertragung vorbereitet, um die Sicherheit in der Kommunikation zwischen Benutzer und Server zu erhöhen.

Als nächstes wird die Logik und Programmierung der verschiedenen Websites, aus der das Smarthome-Webinterface besteht, initialisiert. Jeder dieser Code-Blöcke beginnt mit „@sh.route(‹URL-Zusatz›)". Wird über den Webserver eine bestimmte Seite, die hier gelistet ist, angefragt, wie beispielsweise die Startseite unter „/", die über den Webserver an die Adresse „‹eigene lokale IP›:5010/smarthome" gebunden ist, so wird dieser Code-Block vom Server in der ersten Zeile innerhalb des „@sh.route("/")"-Blocks, der mit „def sh_index():" beginnt, von oben nach unten ausgeführt.

Die Ausführung beginnt mit dem Öffnen und Lesen der Inhalte der „devices.csv", dessen Daten in eine Python-List, ein Array, das mit Daten befüllt werden kann, eingelesen werden. Zu Kontrollzwecken und zum Debugging werden diese Inhalte nun einmal in der Server-Konsole ausgegeben, um gegebenenfalls Fehler zu erkennen. Im nächsten Schritt werden diese Daten in eine weitere Python-List eingelesen und sortiert. Von jedem eingetragenen Gerät wird die jeweilige ID und die zugehörige lokale IP, geordnet in der neuen Liste gespeichert. Im weiteren Verlauf wird nun die API, zum Kommunizieren mit den eingetragenen Geräten, mit den vorher gesammelten Daten versorgt.

Das Programm spricht über die API, mit Hilfe einer for-Schleife, jedes Gerät an, für das es Daten erhält. Für den Status, den die Geräte zurückgeben wird, erneut eine Python-List erstellt, in welches die erhaltenen Gerätedaten eingelesen und abgespeichert werden. Auch hier werden zu Debugging-Zwecken alle gesammelten Daten in der Server-Konsole ausgeben. Im nächsten Schritt werden, mit Hilfe der „rooms.txt", Raum-Daten generiert, die ebenfalls in einer Python-List gespeichert werden. Diese Daten dienen dazu, die Räume auf Code-Basis zu unterscheiden. Als letzter Vorbereitungsschritt zur Generierung wird mit der „request.user_agent.platform"-Methode die Plattform des Benutzers angefragt, um zu entscheiden, ob der Benutzer die Desktop-Variante der Smarthome-Steuerung oder die Variante für mobile Browser, wie Safari auf dem iPhone, erhält. Nachdem alle erforderlichen Daten aus den Räumen gesammelt wurden, beginnt das Programm nun, die Konfiguration der Website einzuleiten.

Der erste Schritt besteht darin, zu erfassen, ob der Benutzer einen Account besitzt, eingeloggt ist, und so bestimmte Einstellungen, welche die Website beeinflussen wie die Sprache und das Theme, getroffen hat. Ist die Person eingeloggt, werden der Name, die eingestellte Sprache und das eingestellte Theme aus den Account-Dateien des Servers ausgelesen und in Variablen gestellt. Als nächstes wird die Sprache, in der die Website generiert wird, festgelegt aus der Einstellung des Accounts, zudem wird das ausgewählte Theme auf die gleiche Weise festgelegt, welches entweder Darkmode oder Lightmode ist. Als Letztes werden die eingestellten Favoriten des Accounts aus der jeweiligen „‹Name des Nutzers›_favs.txt" ausgelesen, in einer Python-List gespeichert und die jeweiligen Informationen aus den hinterlegten Geräten, die der Account ausgewählt hat, gesammelt.

Ist der Benutzer des Webinterfaces nicht eingeloggt, werden die erforderlichen Variablen mit Standard-Einstellungen geladen. Einen Namen gibt es dann nicht, die Standardeinstellung des Themes is der Lightmode und die standardmäßig eingestellte Sprache der Website ist Deutsch.

Der letzte Schritt gibt ein Template der angeforderten Website, mit allen gesammelten Daten, die in den Variablen gespeichert sind, an den Webserver weiter. Hier wird geprüft, ob es sich bei der Plattform des Nutzers entweder um Windows, Mac, oder Linux handelt, oder ob der Nutzer die Website mit einem mobilen Browser aufgerufen hat. Nutzt der Benutzer Windows, Mac oder Linux wird das Template zum Generieren einer Website für einen Desktopbrowser an den Webserver weitergegeben. Wird ein mobiler Browser genutzt, erhält der Server das Template für eine Website, die optimiert ist, für Smartphones.

Mit dem erhaltenen Template generiert der Webserver mit der Jinja-Engine die Website. Die erforderlichen Template-Webdokumente befinden sich in dem Ordner „templates" im smarthome-Verzeichnis, die der Server beim Generieren verarbeitet. Nachdem die Jinja-Engine die Website generiert

hat, wird das fertige Webdokument über den Webserver an den Benutzer gesendet. Zu Debugging-Zwecken wird der Anfrageverkehr des Servers in der Server-Konsole ausgegeben.

4.6.4 __init__.py zum Starten des Webservers

Um den Webserver, der das Steuerungsinterface im lokalen Netzwerk, bereitstellt, zu starten, wird die Datei „__init__.py" auf dem Computer, der als Server dienen soll, ausgeführt. Innerhalb der Start-Datei befinden sich Import-Befehle für die Flask-Bibliothek an sich, um Flask innerhalb des Programms nutzen zu können und der Import-Befehl „render_template", der nötig ist, da Flask auf unserer Webserver-Basis mit den Templates, zum dynamischen Erstellen von Websites, arbeiten muss. Als erste Schritte initialisiert der Server sowohl den Blueprint für das Smarthome-Webinterface als auch den Blueprint für die Roadmap und koppelt diese zu den URL-Suffixes „/smarthome" und „/". Auch der Webserver wird verschlüsselt, um die Sicherheit in der Kommunikation mit Nutzern zu verbessern.

Zudem haben wir für den Webserver eigene Webdokumente entwickelt, die bei Fehlern innerhalb des Servers für den Nutzer angezeigt werden. Wenn z.B. eine Website durch einen Fehler nicht erreichbar ist. Website-Fehlercodes, die durch eigene Fehler-Webpages ersetzt werden, sind Fehler mit den Codes 403, 404, 405 und 500, da diese meistens Auftreten, wenn sich der Fehler innerhalb des Servers befindet. Als letztes initialisiert sich Flask selbst bei Ausführung auf den eigenen Computer und ist über den Port „5010" erreichbar. Zusätzlich wird Flask wärend der Entwicklung im Debug-Modus gestartet, um mehr Informationen über den Status des Webservers zu erhalten.

4.7 Der Aufbau der Steuerungswebsite

Besucht ein Nutzer das Webinterface der Smarthome-Steuerung, mit dem Aufrufen der URL „‹IP des Server-Computers›:5010/smarthome", kommt der Nutzer auf die Hauptseite des Interfaces. Standardmäßig wird hier die Liste der eingestellten Favoriten angezeigt, um schnelle und einfache Bedienung für bereits registrierte und eingeloggte Benutzer zu gewährleisten. Ist man nicht eingeloggt, wird man, anstatt ausgewählte Favoriten zu sehen, darauf hingewiesen sich einzuloggen.

An der oberen Grenze des Webinterfaces wird die sogenannte „Nav-Bar" angezeigt, in der eine die Startseite, eine Übersicht aller Räume und eine Detailansticht jedes einzelnen Raumes per Klick aufgerufen werden kann. Die Buttons der einzelnen Räume werden über die Jinja-Engine des Webservers dynamisch angelegt, um ständiges, umständliches Umprogrammieren des Serverprogramms zu vermeiden, wenn Änderungen in der Raumauswahl auftreten sollten. Die Detailansicht der einzelnen Räume macht detailliertes Einstellen der verschiedenen Geräte eines Raumes möglich, wie das Verändern der Helligkeit einer Lampe oder das Ändern der Farbe, wenn das Gerät kompatibel ist.

Auf der rechten Seite der Nav-Bar befindet sich der Button für die Account-Verwaltung und die aktuelle Uhrzeit.

Die Accountverwaltungs-Schaltfläche ermöglicht es, sich in das Webinterface einzuloggen und, nachdem man eingeloggt ist, seinen Account zu konfigurieren, wie das Ändern der Sprache oder des Themes und das Einstellen der anzuzeigenden Favoriten auf der Startseite. Dem Admin-Account sind zudem einige Einstellungen zur Grundkonfiguration des Webinterfaces zugänglich, wie das Erstellen neuer Räume, das Hinzufügen neuer Geräte, das Erstellen neuer Accounts und das Konfigurieren der Sprachsteuerung. Diese Features befinden sich jedoch noch in der Entwicklung und sind noch nicht fertiggestellt, da noch einige Designänderungen vorzunehmen sind. Navigiert man nun als eingeloggter Benutzer zur Startseite zurück, werden dort die ausgewählten Favoriten für schnelle Konfiguration angezeigt. Insgesamt bietet das Webinterface noch Möglichkeiten zur Verbesserung, doch Funktionalität, Nutzerfreundlichkeit sind gegeben.

5 Sprachsteuerung

Bei der Sprachsteuerung wird das Gesagte mittels einer ASR in Textform umgewandelt/übersetzt. Dabei funktionieren „Automatic Speech Recognition" (kurz ASR) ähnlich wie „Automatic Voice Recognition (kurz AVR) so, dass diese eine Sprachanalyse in Form einer automatischen Interpretation der menschlichen Sprache durchführen[8] und das Gesagte mit vielen Möglichkeiten abgleichen. Am Ende wird das Wort/der Umlaut mit der höchsten Übereinstimmung als eine Zeichenfolge (string) zurückgegeben.

In unserer Spracherkennung arbeiten wir mit einer online API, um das Gesagte umzuwandeln, da es ein sehr langwieriger und aufwendiger Weg ist, ein Programm zu schreiben, was eine Interpretation des Gesagten mit einer hohen Treffsicherheit anlegen kann. Die Entscheidung und Begründung der ausgewählten Programmierschnittstelle und dessen Verwaltung wird im Folgen näher erörtert.

Anschließend wird die Verarbeitung des Gesagten, die Eingliederung eines Hotwordsystems und die Erkennung von Befehlen aufgrund von Keywords erläutert.

[8] (1)

5.1 Funktionsweise von einer ASR im Detail

Eine ASR (automatic speech recognition) funktioniert in mehreren Schritten, bei denen versucht wird, möglichst treffsicher analoge Signale in ein digitales Signal umzuwandeln[9]. Im Folgenden wird die Funktionsweise/der Funktionsablauf vereinfacht dargestellt und erklärt.

Das Ablauf lässt sich auf wenige Schritte begrenzen und beginnt in der Regel mit:

1. Der Aufnahme des Gesprochenen oder der Bereitstellung einer Audioaufnahme mit dem zu analysierenden Inhalt.
2. Als nächstes wird die Aufnahme mithilfe einer Merkmalssektion in Sektionen zerteilt, bei der die Tonspur auf Sprachmerkmale auf Vokale und Konsonanten überprüft wird, um im nächsten Schritt besser abgleichen werden zu können. Der Vorteil der Zerlegung des Gesagten ist, dass dadurch einfacher Gesprochenen abgeglichen werden kann und somit präzisere Analysen angefertigt werden können.
3. Im dritten Schritt werden die verschiedenen Sektionen mit verschiedenen Hilfsmitteln analysiert.
    a. Ein Hilfsmittel wären beispielsweise. trainierte Modelle, welche Ähnlichkeiten mit gesprochenen Wörtern identifizieren. Ein Vorteil von Sprachmodellen ist die bessere Erkennung von nicht standardisierten Wortschätzen und unklarer Ausdrucksweise.
    b. Eine andere Möglichkeit ist die Verwendung von Akustikmodellen, welche die selektierten Bereiche mit Wortlaute vergleicht, um anschließend ein Anreihung von Buchstaben zu ergeben. Diese Zeichenketten werden abschließend mithilfe von Wörterbüchern zu „Worten" zusammengesetzt.
4. Der letzte Schritt ist die Rückbildung der Sektionen in einen „gesprochenen Satz", welcher mittels einem digitalen Format (meist STRING) ausgegeben werden kann.

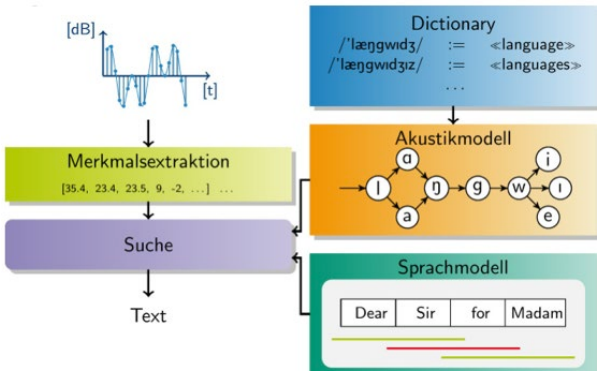

Abbildung 2 Funktionsweise einer ASR

[9] (12)

5.2 Vergleich von offline und online Spracherkennungen

Es gibt unterschiedliche Programme, um die menschliche Sprache in Text umzuwandeln. Hier muss man grundsätzlich zwischen Software im Internet und lokaler Software unterscheiden, welche jeweils Vor- und Nachteile aufweisen.

1. Geschwindigkeit

Die Geschwindigkeit wird allgemein als Latenz beschrieben und gibt die Zeitverzögerung zwischen der Ursache und der Auswirkung an.[10] Da der Weg zu einem Clouddienst, welcher die Daten verarbeitet, grundsätzlich länger ist als bei einem lokalen Dienst, ist die Latenz offline deutlich besser. Der Cloudservice benötigt 6,5-7x länger als das eigene System, jedoch ist dieser Unterschied bei gutem Internet nicht zu bemerken[11].

2. Genauigkeit

Die Genauigkeit (Wortfehlerquote) wird durch drei wesentlichen Faktoren beeinflusst.

1. Der erste Faktor ist die Rechenleistung, die die mögliche Leistung, welches das System erbringen kann, beschreibt.
2. Der zweite Faktor ist die Datengrundlage des System. Diese wird durch Dateien und Trainingsprogramme beschrieben, welche bei der Abfrage als Referenz annimmt. Das Gesprochene wird daraufhin mit diesen Dateien verglichen und die höchste Übereinstimmung „gewinnt".
3. Der dritte und wichtigste Faktor ist der Algorithmus des Systems, welcher das Eingesprochene verarbeitet. Ein System kann leistungsstark sein und eine gute Datengrundlage bieten, jedoch benötigt es ein gutes Programm/Algorithmus, welcher diesen Vorteil auch optimal ausnutzt und ausreizen kann. Im Cloudbereich werden Abfragen häufig zur Weiterbildung und Verbesserung der Systeme genutzt.

Abschließend lässt sich im Punkt der Genauigkeit sagen, dass Clouddienste erfahrungsgemäß hier deutlich besser abschneiden, da sie in der Regel eine bessere Rechenleistung, eine bessere Datengrundlage und bessere Algorithmen verwenden.

[10] (7)
[11] (3)

3. Abhängigkeit

Im Bereich der Abhängigkeit der Abhängigkeit ist eine lokal betriebene Erkennung wesentlich besser, da man nicht auf die Internetverbindung angewiesen ist. Hier bleibt lediglich das eigene System, welches die Grundlage für die Erkennung bietet.

4. Datenschutz

Der Datenschutz ist bei lokalen Systemen 0ebenfalls wesentlich besser. Die eigenen Daten bleiben bei dem Erstellen und dieser hat die alleinige Kontrolle über diese. Bei einer online Erkennung werden die Dateien auf Server übertragen, beiden diese oftmals neben der eigenen Erkennung auch zum Training von weiteren Programmen verwendet werden.

5. Fazit

Abschließend kann man sagen, dass online Spracherkennungen eine deutlich bessere Genauigkeit haben und man den Geschwindigkeitsunterschied nicht bemerkt, jedoch die Abhängigkeit und der Datenschutz nicht optimal sind. Im lokalen Bereich ist dieser deutlich besser, da die Daten auf dem eigenen System liegen und jeder selbst entscheiden kann, wie mit seinen Daten umgegangen wird.

5.3 Spracherkennung im Vergleich

Im Folgenden werden zwei Sprachsteuerungen näher erläutert und die Gründe für deren Nutzung aufgefasst.

5.3.1 Google (speech_recognition)

In der Facharbeit haben wir uns aufgrund der folgenden Gründen für die Benutzung der Google Spracherkennung (speech_recognition) als Demonstrations-ASR entschieden.

1. Einfache Einbindung

Die Benutzung der Google ASR bietet eine einfache und unkomplizierte Einbindung in das eigene System. Es wird ein Audio-Stream des ausgewählten Mikrofons an Google Server gesendet und am Ende wird einem der fertige String wiedergegeben. Zudem ist es bedingt möglich, nicht eingestellte Sprachen zu erkennen, wie bspw. Englisch.

2. Mehr Leistung für die Verarbeitung

Durch das Outsourcen der Spracherkennung auf andere Server ist es wesentlich einfacherer, sich auf die Verarbeitung des Strings zu fokussieren. Des Weiteren müssen Nutzer keine Rechenleistung und große Speicher für die Onlineverarbeitung zur Verfügung stellen.

3. Python 3.9 kompatibel

Durch das Benutzen eines Servers und dem „Flasksystem" ist es wesentlich kompatibler, bei allen Programmen die gleiche Python Version zu benutzen.

5.3.2 CMUSphinx (pocketsphinx)

Nach der Fertigstellung der Facharbeit soll das ASR-System von CMUSphinx zum Einsatz kommen. Der Grund für das spätere Wechseln liegt im Mehraufwand für dieses System. Des Weiteren wird pocketsphinx nicht mehr weiterentwickelt und wird nur bis Python 3.6 unterstützt.

Es gibt jedoch auch viele positive Punkte, die für die Benutzung des CMUSphinx-Systems sprechen:

1. Offlinebenutzung

Das System ist für den Betrieb auf dem eigenen Server ausgelegt und benötigt somit keine Internetverbindung, wodurch man ein vollständiges autarkes System erschaffen kann. Bei einem Internetausfall ist das eigene System davon wenig betroffen und die Sprachsteuerung kann weiterarbeiten. Durch die Abtrennung vom Internet ist es ebenfalls sicherer gegenüber Sicherheitslücken und Updates, welche von Systemfehlern zu Systemabstürzen führen können.

2. Open Source

Der komplette Quellcode des Programms gibt es online bei GitHub/Open Source Plattformen und stehen somit komplett kostenlos zur Verfügung. Zudem ist die Dokumentation und Fehlersuche bei solchen Projekten sehr gut, da die eigenen Probleme häufig bereits aufgetreten sind oder es eine große Community dahinter gibt, die einem helfen kann.

3. Sprachdateien lassen sich anpassen

Die Sprachmodelle/Abgleichdateien der Sprachsteuerung sind öffentlich zugänglich und das System kann auf eigene Hotwords trainiert werden. Dadurch ist auch das Nutzen von eigenen speziellen Namen schnell umsetzbar und man ist nicht auf das evtl. gegebene feste Sprachmodell angewiesen.

4. Besserer Datenschutz

Durch den Offlinebetrieb sind die eigenen Daten in eigener Verwahrung und somit auch besser geschützt bzw. hat man die alleinige Kontrolle, was mit diesen passiert. Im Vergleich zu Onlinediensten liegen die Dateien lokal und werden nicht zur Weiterentwicklung und Anpassung des eigenes Werbebildes genutzt.

5.4 Sprachausgabe im Detail

Bei der Sprachausgabe wird mittels einer Sprachsynthese digitaler Text in ein analoges Audiosignal/Audiodatei umgewandelt. Die Sprachsynthese ist eine künstliche Nachbildung einer natürlichen Sprache und wird nicht aus aufgenommen Aufnahmen abgespielt, sondern neu erzeugt[12].

5.4.1 Wie wird die Stimme erstellt?

Wie der Name „Sprachsynthese" bereits sagt, funktioniert das System synthetisch. Das bedeutet, dass die Ausgabe aus kleinen Toneinheiten, die als Units bezeichnet werden, erschaffen wird. Diese Toneinheiten bestehen anschließend aus einzelnen Phoneme (kleinste bedeutungsunterscheidenden sprachlichen Einheiten[13]) wie A oder O, Diphthonge (ein gebildeter Laut aus zwei Vokalen[14]) und ganzen Silben. Der Grund für unterschiedliche Speichermethoden ist der Grund, dass gleiche Buchstaben in verschiedenen Wortumgebungen anders ausgesprochen werden.

Anschließend werden die Units mithilfe von Algorithmen konkatenativ (eine Verkettung von Zeichen und Zeichenketten[15]) zu neuen und liquiden Satz zusammengeführt. Dieser Teil wird als Synthese bezeichnet. Um die Stimmmelodie natürlicher klingen zu lassen, sind Textzeichen von Bedeutung, damit das Programm erkennt, wo Sprachpausen gemacht werden müssen, die Stimme gehoben werden muss oder ein Wort konkret betont werden muss[16].

5.4.2 Aufbau eins Text-to-Speech-Systems

Die Umwandlung von digitalem Text in eine analoge Audioform erfolgt in mehreren Ereignissen. Im Folgenden werden die einzelnen Schritte näher erläutert.

1. Textanalyse

Die Textanalyse erfolgt in zwei verschiedenen Schritten:

a. Der erste Schritt in der Textanalyse ist die Segmentierung des Textes in einzelne Token. Bei der Konvertierung wird die orthographische (rechtschreibliche Form des Token gebildet und durch Expansionen von Abkürzung die Sprechform gebildet. Beispielsweise wird die verkürzte Schreibweise „ggf." in die gesprochene Form „gegeben falls" oder aus den Zahl „12" und der Jahreszahl „2002" die ausgeschrieben Form „zwölf" und „zweitausendzwei". Jedoch entstehen bei der Expansion/Konvertierung von Zahlen/Abkürzungen unter

[12] (10)
[13] (6)
[14] (8)
[15] (9)
[16] (10)

Umständen auch Komplikationen. Ein gutes Beispiel für die Komplexität des Vorgangs ist die Nummer „1", denn die Form kann durch das was sie bezeichnet stark abweichen, bspw. bei „1 Kilo" = „ein Kilo", „Hausnummer 1" = „Hausnummer eins" oder „1 Löwin verfolgt 1 Hund" = „eine Löwin verfolgt einen Hund".

b. Während der Analyse des Textes wird zeitgleich auch eine Kontextanalyse durchgeführt, welche die Umgebung des Tokens bewertet. Durch diesen Vorgang werden bei Abkürzungen die richtige Formulierung (Zeitform, Kasus, Konjunktion usw.) ermittelt. Ohne solche Analysen könnte nicht bestimmt werden, wie das Wort korrekt expandiert werden soll. Ein Beispiel dafür wäre „fem." (feminin), welches ohne Kontextanalyse nicht ersichtlich wäre, ob es „feminin", „femininer" usw. heißen soll. Ein weitere Aspekt in der Kontextanalyse ist die Betonung von Wörtern, um die Bedeutung des Ausdrucks eindeutig zu machen. Ein Beispiel wäre „modern" und „modern"

2. Phoneme

Phoneme werden im Allgemeinen als kleinste bedeutungsunterscheidende sprachliche Einheit[17] definiert. Nach der Textanalyse geht das Programm in den nächsten Schritt über und prüft, welche Ausspracheregeln angewandt werden können. Hierfür ist die Festlegung der Phoneme essenziell. Die Hauptschwierigkeit liegt in der Natur der Sache, da ein einzelner Buchstabe mehreren Phonemen, aber auch keinem entsprechen kann. Ein Phonem kann auch auf unterschiedliche Weisen ausgesprochen werden. Das Programm kann hier mithilfe zweier Methoden die Aussprache festlegen.
Zum einen gibt es die Wörterbuch basierten Lösungen, bei denen das Programm auf eine Datenbank von Wörtern inklusive der Aussprachedefinition in Form von Morphemen zugreift. Für den Fall, dass ein Wort nicht im Wörterbuch vorhanden ist, wird es anhand von Ausspracheregeln bestimmt und dem Wörterbuch hinzugefügt.
Die zweite Lösung greift auf ein Regelwerk an Ausspracheregeln zu. Hier werden auftretende Ausnahmen in einer Datenbank als einzige Wörter direkt und definitiv festgelegt, um Fehler zu minimieren.

3. Prosodie Generierung

Nach der Aussprachedefinition geht das Programm in den nächsten Schritt über. Dieser entscheidet über die Natürlichkeit des Ausgegebenen. Die Prosodiegenerierung entscheidet über die zeitliche

[17] (6)

Länge der einzelnen Silben und die Amplitudenmodellierung. So nimmt sie Einfluss auf die Akzentuierung und das Sprechtempo. Dies ist besonders wichtig, um dafür zu sorgen, dass der Rezipient das Gesprochene versteht. Die Sinngebung eines Satzes erfolgt oftmals über kleine Details. Auch wird hier der Modus und die Beziehung einzelner Segmente verdeutlicht.
Die Prosodie wird oft anhand der Syntax eines Satzes abgeleitet. Dies ist aber nicht immer möglich, vor allem wenn die Syntax kein eindeutiges Ergebnis liefert.
Ist dies der Fall, so greift das Programm auf semantisches und pragmatisches Wissen zu. Besondere Relevanz findet dieses Wissen in der Betonung von Negierungen, wenn es darum geht, das Verneinte zu betonen, um die Sinnhaftigkeit richtig wiederzugeben. Semantisches und pragmatisches Wissen stehen zurzeit den wenigsten Programmen zur Verfügung.

4. Signalverarbeitung

Die Daten der vorherigen Verarbeitung werden nun mit der Synthese verarbeitet, sodass ein Audiosignal erstellt wird. Dabei werden die gebrauchten Lauten aus einer Datenbank gesucht und der passendste Kandidat gewählt. Das bezeichnet mal als die (konkatenative) Synthese, bei der aus einzelnen Tonstücken durch eine Verkettung von Zeichen oder Zeichenketten neue Sätze erschaffen werden.

Zusammenfassend bekommt das System eine Texteingabe, die es, bevor es zur einem Sprachsignal transformiert werden kann, mithilfe einer Textanalyse analysiert und anschließend in eine phonetische Beschreibung umgewandelt. Der dritte Schritt ist die Generation der Prosodie, welche dann dazu führt, dass nunt aus den vorliegenden Informationen eine Sprachdatei entsteht.

Quelle: (1)

5.5 TTS-System pyttsx3

Wir benutzen in unserem Voice-Assistent „pyttsx3" als Schnittstelle zwischen dem Programm und der „Sprach"-Ausgabe. Diese ist das Bindeglied zwischen dem Programm und der systemintegrierten Sprachsynthese. Die Vorteile in dieser API ist die Auslagerung die lokalen Systems. Auf Windows (XP und höher) wäre dies „SAPI5" und auf Linux (Ubuntu) wäre es „espeak".

5.6 Erklärung des Codes

Im Folgenden werden verschiedene Dateien erklärt:

Den vollständigen Code finden Sie im Anhang.

5.6.1 main.py

In der main.py werden zuerst die anderen Skripte importiert,

```
1 from sr import f_sr
2 from processing import f_processing
3 from so import f_so
```

Abbildung 3 Sprachsteuerung - main.py Part 1

Im Anschluss wird das Schlagwort und evtl. andere Formen sowie die Variable für den Loop definiert.

```
5 start = 1
6 name = ["hermann","herman"]
```

Abbildung 4 Sprachsteuerung - main.py Part 2

Das wirkliche Programm startet mit einer Statusmeldung in der Konsole und einem agiert in einem Loop, damit es immer wieder eine neue Abfrage gibt.

```
8 print("SR: programm starts")
9 while start == 1:
```

Abbildung 5 Sprachsteuerung - main.py Part 3

Im Anschluss wird die Spracherkennung gestartet, der Inhalt in der Variable „hotword" gespeichert und in der Konsole ausgegeben. Der Inhalt wird komplett klein konvertiert, um Fehler bei der Groß-Kleinschreiben Erkennung zu vermeiden.

```
10 hotword = f_sr()
11 hotword = hotword.lower()
12 print(hotword)
```

Abbildung 6 Sprachsteuerung - main.py Part 4

Falls der Inhalt „stop" oder „beende" enthalten sollte, wird das Programm beendet

.

```
13 if "stop" in hotword or "beende" in hotword:
14     print("SR: stop")
15     f_so(output="Ok, ich beende mich. Ihnen noch einen schönen Tag")
16     exit()
```

Abbildung 7 Sprachsteuerung - main.py Part 5

Anschließend wird überprüft, ob der Inhalt der Hotwordabfrage mit der Liste „name" übereinstimmt.

```
17 if hotword in name:
18     print("SR: hotword detection match")
```

Abbildung 8 Sprachsteuerung - main.py Part 6

Wenn es eine Übereinstimmung gab, wird es ein auditives Signal als Bestätigung, dass das System das Schlagwort erkannt hat, gesendet.

```
19     f_so(output="Ich höre")
```

Abbildung 9 Sprachsteuerung - main.py Part 7

und es wird erneut die Spracherkennung gestartet, um den Befehl zu hören.

```
20     inp_sr = f_sr()
```

Abbildung 10 Sprachsteuerung - main.py Part 8

Anschließend wird das Aufgenommene in Textform dargestellt und wieder in kleine Buchstaben konvertiert.

```
22     #Visuelle Ausgabe des Inputs
23     print("SR: inp_sr -> " + inp_sr)
24     inp_sr = inp_sr.lower()
```

Abbildung 11 Sprachsteuerung - main.py Part 9

Der Input wird nun zur Verarbeitung an das „processing"-Script weitergegeben und der verabeitete Text in die Variable „output" geschrieben.

```
25     output = f_processing(inp_sr)
```

Abbildung 12 Sprachsteuerung - main.py Part 10

welche anschließend als visuelles Audiosignal und in der Konsole ausgegeben wird.

```
26     print("SR: " + str(output))
27     f_so(output)
```

Abbildung 13 S
```
1 //Benötigte Bibliotheken
2 #include <OneWire.h>
3 #include <DallasTemperature.h>
4 #include <ESP8266WiFi.h>
5
6 //Port Definition
7 #define data 16
```

```
8  #define led_wifi 15
9
10 //WiFi Login (Name, Passwort)
11 const char* ssid = "SSID";
12 const char* password = "PASSWORD";
13
14 //Webserverport (HTTP Port)
15 WiFiServer server(80);
16
17 //Variablen zum Speichern der HTTP requests
18 String header;
19
20 //Definierung des Timouts (Wann ist eine Webserver Anfrage ein Time-
out)
21 unsigned long currentTime = millis(); //Aktuelle Zeit
22 unsigned long previousTime = 0;
23 const long timeoutTime = 2000; //Maximale Zeit (Danach ist es ein
Timeout)
24
25 //Definierung der Variablen
26 int led_led_dim;
27 int led_led = 255;
28 float led_dim = 1;
29 bool led_state = 0;
30
31 void setup() {
32 Serial.begin(115200); //Starten des seriellen Monitors
33 delay(500);
34
35 //WiFi-Verbingung herstellen
36 //-> Visuelle Ankündigung via serieller Schnittstelle
37 Serial.print("Verbinden zum Netzwerk: ");
38 Serial.println(ssid);
39
40 //-> Verbindung wird aufgebaut (versucht)
41 WiFi.begin(ssid, password);
42
43 //-> Wiederholt des Vorgang, bis WiFi Verbindung aktiviert ist.
44 pinMode(led_wifi, OUTPUT);
45 while (WiFi.status() != WL_CONNECTED) {
46 digitalWrite(led_wifi, HIGH);
47 delay(500);
48 Serial.print(".");
49 digitalWrite(led_wifi, LOW);
50 }
51 digitalWrite(led_wifi, HIGH);
52
53 //-> Visuelle Ausgabe der IP
54 Serial.println("\nVerbindung zum Netzwerk hergestellt");
55 Serial.println("IP-Adresse: ");
56 Serial.println(WiFi.localIP());
57
58 //Webserver wird gestartet
59 server.begin();
60
61 //Definierung der Digitaleneingänge
62 pinMode(data, INPUT);
63 }
64
65 void loop() {
```

```
66 //Wartet auf eine neue Serververbindung
67 WiFiClient client = server.available();
68
69 //Wenn sich jemand verbindet:
70 if (client) {
71 currentTime = millis();
72 previousTime = currentTime;
73 String currentLine = "";
74
75 //Visuelle Benachrichtigung bei einer neuen Verbindung
76 Serial.println("Neue Verbindung zum Server.");
77
78 //Wenn der Client kleinen Timeout gibt
79 while (client.connected() && currentTime - previousTime <= timeout-
Time) {
80 currentTime = millis();
81 // Wenn die Verbindung erfolgreich ist: (Liest die Antwort des Cli-
ents)
82 if (client.available()) {
83 char c = client.read();
84 Serial.write(c);
85 header += c;
86
87 //Er wartet darauf , dass die HHTP request vom Client fertig gelesen
ist
(wird mit zwei "new lines beendent")
88 if (c == '\n') {
89
90 //Die Antwort auf die HTTP request
91 if (currentLine.length() == 0) {
92
93 //Gibt dem Client bescheid, dass der Webserver existiert (Webproto-
koll,
Version, Status)
94 client.println("HTTP/1.1 200 OK");
95
96 //Gibt das Format der Antwort (-> HTML)
97 client.println("Content-type:text/html");
98
99 //Was der Client mit der Verbindung machen soll
100 client.println("Connection: close");
101 client.println();
102
103 //Get-request Verarbeitungen (-> /Befehl)
104 //-> An/Aus
105 if (header.indexOf("GET /") >= 0) {
106
107 }
108
109 //Die Webseite des Webservers
110 client.println("<!DOCTYPE html><html>");
111 client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
112 client.println("<link rel=\"icon\" href=\"data:,\">");
113
114 //->Der Sensor wird abgefragt
115 sensors.requestTemperatures();
116 Serial.print(sensors.getTempCByIndex(0));
117 double temp = sensors.getTempCByIndex(0);
118
```

```
119 //->Die Darstellung der Werte für die API
120 Serial.print(temp);
121 client.println(String("<span id='temp'>") + temp +
String("</span>"));
122 delay(100);
123
124 client.println("</body></html>");
125
126 //Die HHTP response endet mit einer leeren Zeile
127 client.println();
128 // Geht aus dem Webloop
129 break;
130 }
131 //Die letzte Zeile der HHTP request wird gelöscht
132 else {
133 currentLine = "";
134 }
135 }
136
137 //Wenn mehr vom Client kommt (HTTPS request) als erwartet, wird es an
currentLine angehangen
138 else if (c != '\r') {
139 currentLine += c;
140 }
141 }
142 }
143 //Leert den Header
144 header = "";
145 //Schließt die Verbindung
146 client.stop();
147 Serial.println("Client disconnected.");
148 Serial.println("");
149 }
150 }prachsteuerung - main.py Part 11
```

Nach dem Ablauf, startet die Schleife erneut.

5.6.2 sr.py

In der sr.py wird am Anfang die API von Google importiert

```
1 import speech_recognition as sr
```

Abbildung 14 Sprachsteuerung - sr.py Part 1

und die Funktion definiert.

```
3 def f_sr():
```

Abbildung 15 Sprachsteuerung - sr.py Part 2

Anschließend wir „r" mit dem der Spracherkennung definiert.

```
5     r = sr.Recognizer()
```

Abbildung 16 Sprachsteuerung - sr.py Part 3

Als nächstes wird das Mikrofon als Audioquelle gesetzt und dem Programm mitgeteilt, dass es zuhören soll.

```
6     with sr.Microphone() as source:
7           audio = r.listen(source)
```

Abbildung 17 Sprachsteuerung - sr.py Part 4

Im Anschluss wird versucht, das Aufgenommene zu erkennen und der Inhalt wird in den String „inp" geladen.

```
8           try:
9                 inp  =  r.recognize_google(audio,  language="de-DE")
```

Abbildung 18 Sprachsteuerung - sr.py Part 5

Falls sie Sprachsteuerung nichts erkennen sollte, setzt sie den inp auf fail und gibt über die Konsole den Fehler bekannt.

```
13          except sr.RequestError as e:
14                print( "SR: speech_recognition couldn't receive any-
                  thing from the google server. Error:; {0} !".for-
                  mat(e))
16                inp = "noconnection"
```

Abbildung 19 Sprachsteuerung - sr.py Part 6

Wenn die SR nichts empfangen kann/keine Verbindung zum Server besteht, wird ebenfalls einen den Fehler in der Konsole ausgegeben und der „inp" auf „noconnention" gesetzt.

```
8           try:
9                 inp  =  r.recognize_google(audio,  language="de-DE")
```

Abbildung 20 Sprachsteuerung - sr.py Part 7

Zum Ende wird der endgültige „inp" in die Konsole geschrieben und der String in das andere Script zurückgegeben.

```
17    print("SR: speech recognition recognized content: " + inp)
18    return inp
```

Abbildung 21 Sprachsteuerung - sr.py Part 8

5.6.3 so.py

Zu Beginn wird das Programm „pyttsx3" importiert, damit es auf die Funktionen zugreifen kann.

```
1 import pyttsx3
```

Abbildung 22 Sprachsteuerung - so.py Part 1

Im Anschluss wird die systeminterene Sprachsynthese definiert und

```
2 engine = pyttsx3.init()
```

Abbildung 23 Sprachsteuerung - so.py Part 2

in der Funktion „f_so(output)" startet mit dem String „output".

```
4 def f_so(output):
```

Abbildung 24 Sprachsteuerung - so.py Part 3

Die Geschwindigkeit der Sprachsynthese wird auf 150 gesetzt, damit diese ein besseres Verständnis hat.

```
5     engine.setProperty('rate', 150)
```

Abbildung 25 Sprachsteuerung - so.py Part 4

Abschließend wird der Sting ausgegeben.

```
6     engine.say(str(output))
```

Abbildung 26 Sprachsteuerung - so.py Part 5

5.6.4 processing.py

Als Erstes werden die die Erweiterungsmodule und die Zeitangaben geladen.

```
1 import csv
4 from datetime import datetime
5 import locale
6 locale.setlocale(locale.LC_TIME, "de_DE")
8 from api import f_api
```

Abbildung 27 Sprachsteuerung - processing.py Part 1

Im Anschluss wird die Funktion definiert und

```
10 def f_processing(inp_sr):
```

Abbildung 28 Sprachsteuerung - processing.py Part 2

die Datei „devices.csv" im Format „utf-8" geöffnet, damit der Zeichensatz vollständig gelesen werden kann (bswp. Ä,Ö oder Ü).

```
21      with open("devices.csv", newline="", encoding="utf-8") as f:
22            reader = csv.reader(f)
23            data = list(reader)
```

Abbildung 29 Sprachsteuerung - processing.py Part 3

Danach werden die Positionen der jeweiligen Punkte in jeder Zeile definiert, indem diese mit der ersten Zeile abgeglichen werden.

```
26      pos = str(data[0][0]).split(";")
27      for typ in range(0, len(pos)):
28            if pos[typ] == "ID":
29                  ID_pos = typ
30            if pos[typ] == "Zimmer":
31                  room_pos = typ
32            if pos[typ] == "Name":
33                  name_pos = typ
```

Abbildung 30 Sprachsteuerung - processing.py Part 4

Wenn nicht alle wichtigen Positionen geladen werden konnten, werden keine Befehle über das Smarthome angenommen und es funktioneren nur die Basisfunktionen.

```
35      if ID_pos == "" or room_pos == "" or name_pos == "":
36            print("PR: ERROR > devices paramenter not working")
37            print("PR:  devices  are  not  available,  only  other  funktions
              available")
```

Abbildung 31 Sprachsteuerung - processing.py Part 5

Falls jedoch die Daten alle einen Inhalt enthalten, wird der String „inp_sr" in eine Liste umgewandelt, damit man besser mit diesem Arbeiten kann.

```
39      else:
40            #Der Input wird in eine Liste umgewandelt (Array)
41            inp_sr = inp_sr.split(" ")
```

Abbildung 32 Sprachsteuerung - processing.py Part 6

Im Anschluss wird aus „inp_sr" der Raum gesucht, indem jede Position mit jedem möglichen Raum abgeglichen wird. Nachdem der Raum bestimmt wurde, wird dieser in die Liste „check" angehangen.

```
44          for room in inp_sr:
45                for device in range(1, len(data)):
46                      argument = data[device][0]
47                      argument = str(argument).split(";")
48                      if argument[room_pos] == room:
49                            check.clear()
50                            check.append(room)
51                            print(
```

Abbildung 33 Sprachsteuerung - processing.py Part 7

Falls ein Raum festgestellt wurde,

```
55          if len(check) == 1:
```

Abbildung 34 Sprachsteuerung - processing.py Part 8

wird geschaut, ob auch ein gespeicherter Name von einem Gerät gesagt wurde. Nachdem die Abfrage erfolgreich war, wird mit dem Namen und dem Raum die ID gesucht.

```
56                for name in inp_sr:
57                      for device in range(1, len(data)):
58                            argument = data[device][0]
59                            argument = str(argument).split(";")
60                            #Wenn es den Gerätenamen gibt, wird
                              geschaut ob es in dem ermittelten Raum
                              das Gerät gibt
61                            if argument[name_pos] == name:
62                                  print("PR: devices name = " +
                                    str(name))
63                                  if argument[room_pos] == check[0]:
64                                        print("PR: devices id found")
65                                        check.append(name)
66                                        id = argument[ID_pos]
67                                        print("PR: devices id = " +
                                          str(id))
68                                        break
```

Abbildung 35 Sprachsteuerung - processing.py Part 9

Andernfalls wird es über die Konsole ausgegeben.

```
88          else:
89                print("SR: Es wurde kein passender Raum gefunden")
```

Abbildung 36 Sprachsteuerung - processing.py Part 10

Wenn die ID gefunden wurde,

```
70          if id != "":
```

Abbildung 37 Sprachsteuerung - processing.py Part 11

wird nach weiteren Befehlen für das Gerät gesucht (bswp. dem Dimmfaktor) und diese dann über den Sting „inp" an die API weitergegeben.

```
71              for command in inp_sr:
72                  inp = ""
73                  if "%" in command:
74                      command = command[:-1]
75                      inp = 0.01 * int(command)
76                      print("PR: inp_api -> " + str(inp))
77                      break
78                  elif command == "an" or command == "ein":
79                      print("PR: inp_api -> on")
80                      inp = "on"
81                  elif command == "aus" or command == "ab":
82                      print("PR: inp_api -> off")
83                      inp = "off"
84                      break
85              print("PR: ID - " + id)
86              output = f_api(id, inp)
```

Abbildung 38 Sprachsteuerung - processing.py Part 12

Falls das Gerät nicht dem ausgewählten Raum zugeordnet werden kann, wird es über die Konsole ausgegeben.

```
91          #Wenn das Gerät nicht dem Raum zugeordnet werden konnte
92          if len(check) == 1:
93              print("PR: Das Gerät konnte dem Raum nicht
                zugeordnet werden")
```

Abbildung 39 Sprachsteuerung - processing.py Part 13

Nachdem der „inp_sr" auf die Smarthomebefehle überprüft wurde, wird diese im Anschluss auf die Basisfunktionen

```
95      for command in inp_sr:
```

Abbildung 40 Sprachsteuerung - processing.py Part 14

wie die Uhrzeit oder das Datum überprüft.

```
96          if "wetter" in command:
97                output = "weather is coming soon"
98          if "uhr" in command or "uhrzeit" in command or "zeit" in
            command or "spät" in command:
99                time = datetime.now()
100               output = time.strftime("%H:%M")
101         if "tag" in command or "datum" in command:
102               time = datetime.now()
103               output = time.strftime("%A, der %d.%m.%Y")
```

Abbildung 41 Sprachsteuerung - processing.py Part 15

Zum Schluss wird der Output an das andere Script zurückgeschickt.

```
106         print("PR: Output -> " + str(output))
107         return output
```

Abbildung 42 Sprachsteuerung - processing.py Part 16

6 Arbeitsdokumentation

6.1 Arbeitsteilung

Name	Aufgaben	Facharbeit
Hunold, Julius	- Entwicklung der API - Entwicklung der Sprachsteuerung - Entwicklung des Arduino-Codes	- 3 API - 5 Spracherkennung
Porsch, Daniel	- Entwicklung des Webinterface - Unterschiede Online/Offline Sprach-erkennungen - Entwicklung der Arduinoboards	- 5.2 Vergleich online/offline ... - 2 Arduino Geräte/Server
Prelle, Marcel	- Entwicklung des Webinterface - Aufstellung der Zeitpläne und der Organisation	- 4 Webserver

Tabelle 1 Arbeitsteilung

6.2 Zeitplan

Termin → / Aufgabe ↓	20. Jan	03. Feb	17. Feb	03. März	17. März	14. April	28. April	12. Mai	26. Mai	09. Juni	23. Juni
Generelle Planung											
Prototyp											
Webserver											
Smarthome Geräte											
Sprachsteuerung											
Facharbeit											
Präsentation											

Tabelle 2 Zeitplan

7 Fazit

[FEHLT]

Quellenverzeichnis

1. **Tim, Tech With.** YouTube. [Online] 05. 04 2020. [Zitat vom: 06. 06 2021.] https://www.youtube.com/watch?v=3QiPPX-KeSc.

2. **linguatec.** linguatec. [Online] [Zitat vom: 05. 06 2021.] https://www.linguatec.de/text-to-speech/tts-technologie/.

3. **ITWissen.** ITWissen. [Online] 09. 03 2019. [Zitat vom: 25. 05 2021.] https://www.itwissen.info/ASR-automatic-speech-recognition-Automatische-Spracherkennung.html#:~:text=Die%20automatische%20Spracherkennung%2C%20Automatic%20Speech,automatischen%20Interpretation%20von%20menschlicher%20Sprache.

4. **Luber, Dipl.-Ing. (FH) Stefan.** DEV-INSIDER. [Online] 08. 03 2017. [Zitat vom: 25. 05 2021.] https://www.dev-insider.de/was-ist-eine-api-a-583923/.

5. **Ray, Debosmit.** stackoverflow. [Online] 04 2016. [Zitat vom: 25. 05 2021.] https://stackoverflow.com/questions/36277050/difference-between-online-and-offline-speech-to-text-conversion.

6. **Redaktion ComputerWeekly.de, TechTarget.** Computer-Weekly. [Online] 04 2020. [Zitat vom: 25. 05 2021.] https://www.computerweekly.com/de/definition/Programmierschnittstelle-API.

7. **Talend.** Talend. [Online] [Zitat vom: 25. 05 2021.] https://www.talend.com/de/resources/was-ist-eine-api/.

8. **Duden/Phonem.** Duden. [Online] [Zitat vom: 05. 06 2021.] https://www.duden.de/rechtschreibung/Phonem.

9. **Duden/Latenz.** Duden. [Online] [Zitat vom: 25. 05 2021.] https://www.duden.de/rechtschreibung/Latenz.

10. **Duden/Diphthong.** Duden. [Online] [Zitat vom: 05. 06 2021.] https://www.duden.de/rechtschreibung/Diphthong.

11. **Duden/Konkatenation.** Duden. [Online] [Zitat vom: 05. 06 2021.] https://www.duden.de/rechtschreibung/Konkatenation.

12. **News-Stream.** newsstreamproject. [Online] 17. 06 2017. [Zitat vom: 05. 06 2021.] https://newsstreamproject.org/explainer-wie-funktioniert-eigentlich-spracherkennung/.

13. **retresco.** retresco. [Online] [Zitat vom: 05. 06 2021.] https://www.retresco.de/lexikon/spracherkennung/.

14. **botschaft.** botschaft. [Online] [Zitat vom: 05. 06 2021.] https://botschaft.digital/glossar/text-to-speech/#:~:text=Unter%20Text%2Dto%2DSpeech%20bzw,Flie%C3%9Ftext%20in%20eine%20akustische%20Sprachausgabe.

15. **Krisnawati, Lucia D.** UNI München. [Online] [Zitat vom: 05. 06 2021.] https://www.cis.uni-muenchen.de/~hs/teach/13w/intro/pdf/15asr.pdf.

16. **Laser & Co. Solutions GmbH.** myavr. [Online] [Zitat vom: 05. 06 2021.] http://einsteiger.myavr.de/index.php?id=5.

17. **rs-online.** rs-online. [Online] [Zitat vom: 05. 06 2021.] https://de.rs-online.com/web/generalDisplay.html?id=ideen-und-tipps/mikrocontroller-leitfaden.

18. **community, mikrocontroller.** mikrocontroller. [Online] [Zitat vom: 05. 06 2021.] https://www.mikrocontroller.net/articles/Mikrocontroller.

19. **mikrocontroller-elektronik.** mikrocontroller-elektronik. [Online] 12. 01 2017. [Zitat vom: 05. 06 2021.] https://www.mikrocontroller-elektronik.de/nodemcu-esp8266-tutorial-wlan-board-arduino-ide/.

20. **elektronik-kompendium.** elektronik-kompendium. [Online] [Zitat vom: 05. 06 2021.] http://www.elektronik-kompendium.de/sites/kom/index.htm.

21. **RJTC.** RJTC. [Online] [Zitat vom: 05. 06 2021.] https://www.pjrc.com/teensy/td_libs_OneWire.html.

22. **hope, computer.** computer hope. [Online] 10. 05 2017. [Zitat vom: 05. 06 2021.] https://www.computerhope.com/jargon/t/timeout.htm.

23. **antratek.** antratek. [Online] [Zitat vom: 05. 06 2021.] https://cdn.antratek.nl/media/product/5df/nodemcu-v2-lua-based-esp8266-development-kit-16608-42d.jpg.

24. **Grokhotkov, Ivan.** readthedocs. [Online] 2017. [Zitat vom: 05. 06 2021.] https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html.

25. **pollin.** pollin. [Online] [Zitat vom: 05. 06 2021.] https://www.pollin.de/productdownloads/D180014D.PDF.

26. **sentdex.** YouTube. [Online] 11. 03 2019. [Zitat vom: 06. 06 2021.] https://www.youtube.com/watch?v=Lbfe3-v7yE0.

27. **Pallets.** palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/.

28. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/foreword/.

29. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/installation/.

30. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/quickstart/.

31. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/tutorial/templates/.

32. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/tutorial/views/.

33. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/templating/.

34. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/testing/#logging-in-and-out.

35. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/config/.

36. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://flask.palletsprojects.com/en/2.0.x/debugging/.

37. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://jinja.palletsprojects.com/en/3.0.x/api/.

38. –. palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://jinja.palletsprojects.com/en/3.0.x/templates/.

39. **Bodnar, Jan.** ZetCode. [Online] 06. 07 2002. [Zitat vom: 06. 06 2021.] https://zetcode.com/python/jinja/.

40. **Pallets.** palletsprojects. [Online] [Zitat vom: 06. 06 2021.] https://werkzeug.palletsprojects.com/en/2.0.x/tutorial/.

41. **LogicalBranch.** stackoverflow. [Online] 05 2016. [Zitat vom: 06. 06 2021.] https://stackoverflow.com/questions/37004983/what-exactly-is-werkzeug.

42. **jquery.** jquery. [Online] [Zitat vom: 06. 06 2021.] https://jquery.com/.

43. –. jquery. [Online] [Zitat vom: 06. 06 2021.] https://api.jquery.com/.

44. **w3schools.** w3schools. [Online] [Zitat vom: 06. 06 2021.] https://www.w3schools.com/jquery/jquery_intro.asp.

45. –. w3schools. [Online] [Zitat vom: 06. 06 2021.] https://www.w3schools.com/jquery/jquery_get_started.asp.

46. –. w3schools. [Online] [Zitat vom: 06. 06 2021.] https://www.w3schools.com/jquery/jquery_syntax.asp.

47. –. w3schools. [Online] [Zitat vom: 06. 06 2021.] https://www.w3schools.com/jquery/jquery_selectors.asp.

48. –. w3schools. [Online] [Zitat vom: 06. 06 2021.] https://www.w3schools.com/jquery/jquery_events.asp.

49. –. w3schools. [Online] [Zitat vom: 06. 06 2021.] https://www.w3schools.com/jquery/jquery_hide_show.asp.

Erklärung der Verfasser

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und alle Formulierungen, die wörtlich oder dem Sinn nach aus anderen Quellen entnommen wurden, kenntlich gemacht habe. Verwendete Informationen aus dem Internet können der Schule auf Verlangen vollständig im Ausdruck zur Verfügung gestellt werden. Sofern sich - auch zu einem späterem Zeitpunkt - herausstellt, dass die Arbeit oder Teile davon nicht selbstständig verfasst wurden, die Zitathinweise fehlen oder Teile ohne Quellennachweis aus dem Internet entnommen wurden, so wird die Arbeit auch nachträglich mit Null Punkten gewertet.

Hildesheim, 9. Juni 2021	
Ort, Datum	Hunold, Julius
Hildesheim, 9. Juni 2021	
Ort, Datum	Prelle, Marcel
Hildesheim, 9. Juni 2021	
Ort, Datum	Porsch, Daniel

Anhang

Aus Gründen der Komplexität des Projektes ist es leider nicht möglich alles und den kompletten Quellcode des Systems analog zur Verfügung zu stellen und aus diesem Grund kann alles online abgerufen und heruntergeladen werden. Wir bitten um Ihr Verständnis.

Online

Link: https://julius.hunold.de/facharbeit

Analoge

Switch Module

Schaltplan

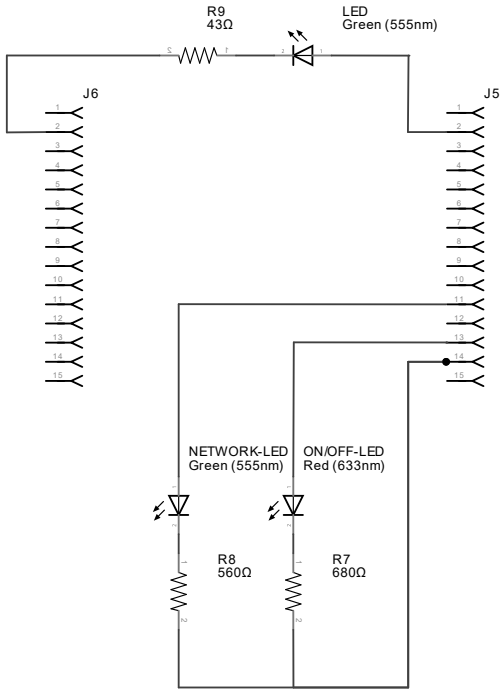

Abbildung 43 Switch Module - Schaltplan

Steckplatine

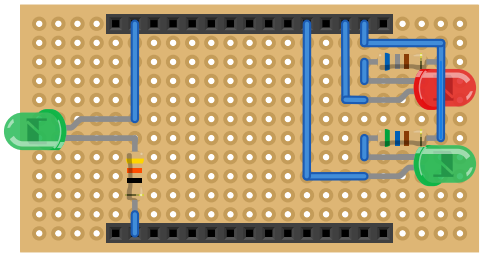

Abbildung 44 Switch Module - Steckplatine

RGB Module

Schaltplan

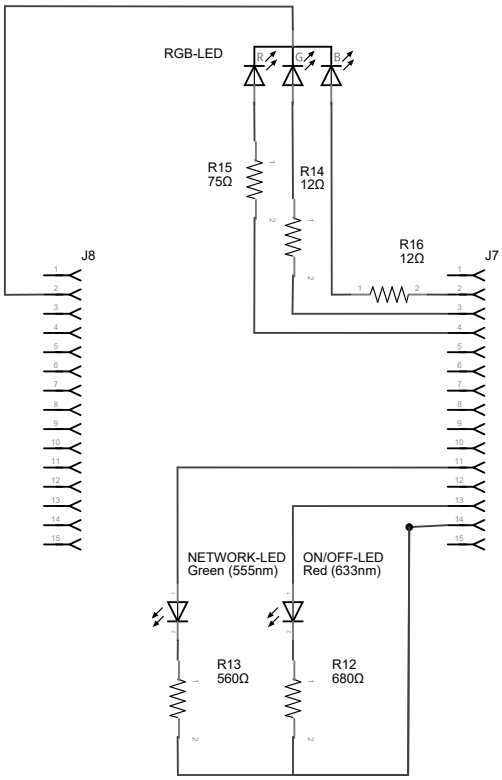

Abbildung 45 RGB Module - Schaltplan

Steckplatine

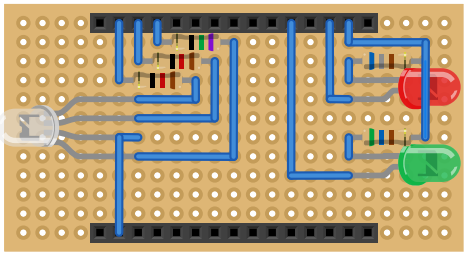

Abbildung 46 RGB Module - Steckplatine

Temp Module

Schaltplan

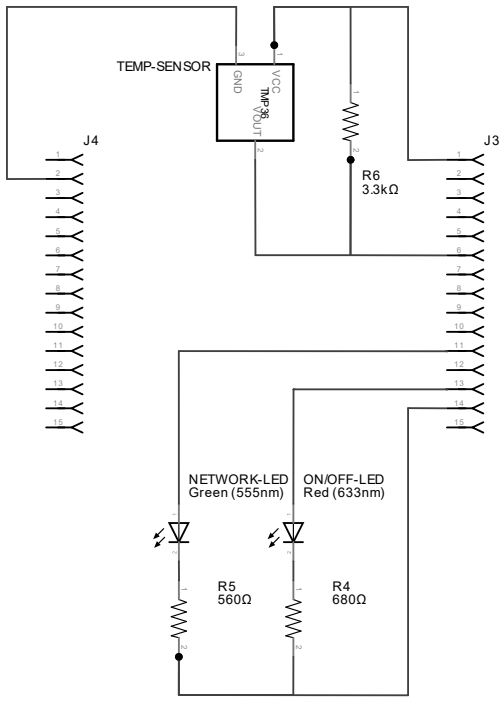

Abbildung 47 Temp Module - Schaltplan

Steckplatine

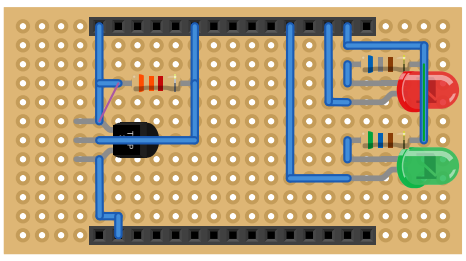

Abbildung 48 Temp Module - Steckplatine