Goethe Universität Frankfurt Main

- Fachbereich Informatik -

Entwicklung eines Node.js-basierten IoT-Developer-Kit für den Raspberry Pi mit Demonstrationsbeispiel

Abschlussarbeit zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.)

vorgelegt von

Stefan Chalupka – 6037666

am

15. Oktober 2019

1. Betreuer:	Prof. Dr. Detlef Krömker
2. Betreuer:	Dr. Ing. The Anh Vuong

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe.

Ebenso bestätige ich, dass ich diese Arbeit nicht, auch nicht auszugsweise, für eine andere Prüfung oder Studienleistung verwendet habe.

_____

Frankfurt am Main, den 14. Oktober 2019

Abstract

Die vorliegende Bachelorarbeit beschreibt die Entwicklung eines Software-Kits auf Basis von Node.js, mit der es erleichter werden soll, Smart-Home-Anwendungen für den Einplatinencomputer „Raspberry Pi“ zu erstellen. Diese Software wird mit dessen Pins für allgemeine Ein- und Ausgabezwecke kommunizieren und schaltet diese gegebenenfalls. Auf diesem Wege soll es vereinfacht werden, eigene Projekte in Richtung Smart-Home umzusetzen, um sich so aus einem Raspberry Pi ein „Internet of Things“-Gerät zu bauen. Auf diesem Wege kann das eigene Heim automatisiert werden. Smart-Home bezeichnet ein praktisches Heim-Setup, bei dem Geräte über ein Mobilgerät oder ein anderes vernetztes Gerät von jedem mit dem Internet verbundenen Ort der Welt aus automatisch oder manuell ferngesteuert werden können. Es wird auf den konzeptionellen Aufbau der Software eingegangen und dieser in einzelnen Schritten beschrieben. Zum Abschluss wird mit einem Anwendungsbeispiel die Funktionsweise des Software-Kits dargestellt und auf die Vor- und Nachteile der Software eingegangen.

Danksagung

An dieser Stelle danke ich all denjenigen, die mich während meines Studiums und der Anfertigung dieser Arbeit unterstützt haben. Einen besonderen Dank gebührt meiner Familie, die mich immer unterstützt und mir mit Rat und Tat zur Seite steht. Auch bei meinen Freunden möchte ich mich bedanken, da sie mich während des Studium immer wieder motivierten weiterzumachen, mich in schweren Zeiten unterstützten und mich auch gestresst ertragen haben.

Einem weiteren Dank gilt meinem Betreuer Dr. Ing. The Anh Vuong der mich während der Bearbeitungszeit dieser Arbeit unterstützte und mir mit hilfreichen Tipps weiterhalf.

Inhaltsverzeichnis

1. Einleitung....7
1.1 Motivation und Ziel der Arbeit....7
1.2 Aufgabenstellung....8
2. Grundlagen....9
2.1 Frontend-Entwicklung....9
2.1.1 Hypertext Markup Language (HTML)....9
2.1.2 Cascading Style Sheets (CSS)....10
2.1.3 Javascript....11
2.2 Backend-Entwicklung....13
2.2.1 Node.js....13
2.2.1.1 Express....14
2.2.1.2 Path....15
2.2.1.3 File System (fs)....15
2.3 application programming interface (‚API‘)....16
2.3.1 Hypertext Transfer Protocol (HTTP)....16
2.3.2 Das JSON – Format....18
2.4 Der Raspberry Pi....19
2.4.1 general purpose input/output (GPIO)....20
3. Konzept / Anforderungen....21
3.1 Aufbau des Development-Kit....21
3.2 Erste Ebene – Die GPIO Klasse....22
3.3 Zweite Ebene – Der Server und die JSON-API....23
3.4 Dritte Ebene – Die Anwendung und die JSON Kommunikation....25
3.5 Anforderungen....26
4. Realisierung....27
4.1 Arbeitsumgebung / Werkzeuge....27
4.1.1. Rasbian / Rasperry Pi....27
4.1.2. PuTTY....28
4.1.3. GitHub....28
4.2 Aufbau der GPIO-Klasse für Node.js in Javascript....29
4.3 Aufbau eines Express-Servers mit einer API-Schnittstelle zu den GPIO-Pins....31
4.4 Aufbau einer möglichen Anwendung mit Kommunikation über die API....35
5. Ergebnis....37
5.1 Wie gut funktioniert das Kit in der Anwendung?....37
5.2 Wurden die Anforderungen erreicht?....38
5.3 Was kann verbessert werden?....39
6. Fazit....41
6.1 Ausblick....41
6.2 Zusammenfassung....42
Verzeichnisse....44
1 Literaturverzeichnis....44
2 Abbildungsverzeichnis....44
3 Quellcode....44

1. Einleitung

1.1 Motivation und Ziel der Arbeit

Man kommt nach einem anstrengenden Arbeitstag im gut klimatisiertem Zuhause an, das Essen ist gekocht, die Wohnung gesaugt und geputzt, es läuft die entspannende Musik vom Lieblingsinterpreten und selbst die Wäsche ist gewaschen.

Das klingt für viele Menschen aktuell noch nach Zukunftsmusik, die Entwicklung dessen hat aber bereits begonnen. Mit Smart-Home-Anwendungen wie Googles Assistent oder Amazons Alexa automatisieren sich immer mehr Menschen Stück für Stück ihr Eigenheim und erleichtern sich so ihren Alltag. Laut einer Studie haben 28% der Erwachsenen in Amerika ein oder mehr Smart-Home Geräte zuhause, wobei zwischen den 18-34 jährigen 47% ein solches Gerät bestizen[1]. Allein im Jahr 2017 hat Amazon nach Hochrechnungen mehrere Millionen Echo-Geräte verkauft[2]. Auch die Verkaufszahlen der internetfähigen Haushaltsgeräte (Internet of Things) wie Fernseher, Waschmaschinen, Kühlschränke oder Staubsauger, um nur ein paar zu nennen, steigen seit 2015 stetig[3]. Nichtsdestotrotz geben nach einer repräsentativen Umfrage von „G DATA“ 62% der deutschen an, dass sie glauben, dass Unternehmen trotz DSGVO nicht verantwortungsvoll mit ihren persönlichen Daten umgehen[4a]. Des Weiteren sind zum Beispiel in Deutschland, nach einer Studie von „Kantar TNS“, 58% der Verbraucher dagegen, dass vernetzte Geräte ihre Aktivitäten überwachen, selbst wenn dies ihr Leben einfacher machen würde[4b].
Daraus lässt sich erkennen, dass in der Gesellschaft das Interesse an der Automatisierung von Aufgaben und einer daraus resultierenden Arbeitserleichterung, sowohl Zuhause als auch in der Industrie, wächst. Dennoch misstrauen die Verbrauchen wegen der vielen aufkommenden Datenskandalen diesen Smart-Home-Anwendungen. Um eigene Projekte umzusetzen, kaufen sich daher viele Bastler und Entwickler einen Raspberry Pi der sehr beliebt ist wegen seiner winzigen Steckerleiste für allgemeine Ein- und Ausgabezwecke (GPIO). Deshalb soll Ziel dieser Arbeit ein Software-Paket sein, mit dem es vereinfacht werden soll (Web-)Anwendungen zu entwickeln die über einen Node.js-Server mit den GPIO-Pins kommunizieren. Es soll also lesender und schreibender Zugriff auf die Pins über ein Netzwerk erlangt werden.

Die Kommunikation soll dabei sehr einfach gehalten sein, um auch Programmiereinsteiger und Bastler dafür begeistern zu können, mit diesem Kit schnell und einfach Anwendungen zu entwerfen, mit denen sie ihr Zuhause kostengünstig automatisieren und sich gegebenenfalls ihre eigenen IoT-Geräte bauen können.

1.2 Aufgabenstellung

Die Aufgabe in dieser Abschlussarbeit ist es ein Software-Kit zu entwerfen mit dem es einfacher möglich sein soll, ein ‚Internet of Things'-Gerät zu entwickeln, welches über eine (Web-)Anwendung gesteuert werden können soll. Dabei soll die Webanwendung mit einem Webserver kommunizieren der auf der Plattform Node.js läuft. Dies soll ermöglicht werden, in dem die Steuerung der GPIO-Pins über ein Netzwerk verfügbar gemacht wird. Anwender können so über ein Netzwerk die Pins ein und ausschalten, aber auch abfragen welcher Wert an einem Pin anliegt. Auf diesem Wege braucht sich der Entwickler nicht um die Implementierung der Pin-Steuerung kümmern, sondern kann diese nach Start des Node.js-Servers leicht mit einfachen HTTP-Anfragen kontrollieren.

Zur besseren Entwicklung und leichteren Übersicht der Status der Pins sollte eine Übersichtwebseite entworfen werden, in der die Status-Informationen der Pins übersichtlich und bildlich dargestellt werden und per Schnellzugriff gesteuert werden können. Das hilft enorm bei der Konzeption einer neuen Schaltung, da man nach Belieben einzelne Pins anschließen und diese leicht bedienen kann und sieht welche der Pins eine Spannung anliegen haben. Als Zweites sollte eine API programmiert werden, über die der Entwickler später seine Programme mit dem Software-Kit kommunizieren lassen kann.

Um die Aufgabe umzusetzen sollte das Software-Kit in verschiedene, aufeinander aufbauende Stücke strukturiert werden, um so eine Software zu entwickeln bei der die Bausteine leicht austausch- oder erweiterbar sind. Um die Entwicklung zu vereinfachen und die Softwareschichten besser aufeinander abzustimmen wird die Software nach dem bottom-up Prinzip programmiert, also angefangen bei den einzelnen Klassen, die dann Stück für Stück zusammen geführt werden, bis hin zu einem kompletten Softwarepaket. Aus diesem Grund wird als erstes die GPIO-Klasse geschrieben, welche das Fundament der Software ist. Mit dieser Klasse kann der Node.js-Server mit den Pins arbeiten. Danach sollte die ‚Middleware' programmiert werden, also der Server inklusive der API. Die Middleware ist das Bindeglied zu der späteren Anwendung eines Entwicklers und beantwortet die HTTP-Anfragen. Zum Schluss wird ein Anwendungsbeispiel erstellt, um mit dessen Hilfe, Stärken und Schwächen des Software-Kits zu ermitteln.

Für ein besseres Verständnis der Arbeit werden zunächst die Grundlagen aufgezeigt, die zum Entwickeln des Software-Kits nötig sind und erklärt warum diese benötigt werden. Danach wird im 3. Kapitel auf den konzeptuellen Aufbau des Software-Kits eingegangen und die einzelnen Schritte, Funktionen und Zusammenhänge erklärt. Auch auf Anforderungen an das Softwarepaket wird in dem Kapitel eingegangen, sodass im 4. Kapitel die Realisierung der Software beschrieben werden kann. Darunter fallen zum Beisiel die Methoden die benutzt werden, um das Software-Kit zu erstellen. Am Ende der Arbeit werden die Eigenschaften des Kits mit den Anforderungen verglichen und es wird ein Ausblick in die Zukunft gegeben. Schließlich wird die Arbeit und deren Funktionsweise kritisch zusammengefasst.

2. Grundlagen

In diesem Kapitel werden die Grundlagen erläutert, die zum Verständnis für das Erstellen des Softwarekit benötigt werden. Da für das Kit ein Server, eine API und eine Webseite geschrieben werden müssen, ist es erforderlich Grundkenntnisse in der Webentwicklung zu haben. Aus diesem Grund wird als nächstes auf die Elementaren Konzepte der Frontend- und Backend-Entwicklung (auch Fullstack-Development genannt) eingegangen. Damit der Client bzw. eine spätere Anwendung eines Entwicklers mit dem Server kommunizieren kann, braucht dieser eine Schnittstelle, auch API genannt. Auf diese Thematik wird nach der Backend-Entwicklung eingegangen. Da die Arbeit auf einem Raspberry Pi programmiert und ausgeführt wird, bekommt auch er ein Platz in diesem Kapitel.

2.1 Frontend-Entwicklung

Bei der Frontend-Entwicklung geht es wie der Name es schon deuten lässt, um den Teil der Software, die der Nutzer sieht und über die er Dinge eingibt. Frontends gibt es bei fast jeder Software, nicht nur bei Homepages. Das Frontend beinhaltet alle sichtbaren Inhalte. Dazu gehört auch das Design oder die Aufteilung einer Webseite, Bilder, Texte - eben alles, mit dem der Nutzer interagiert. Für die Programmierung des Frontends werden in der Regel die Beschreibungssprache HTML (Hypertext Markup Language), die Stylesheet-Sprache CSS (Cascading Style Sheets) und die Skriptsprache JavaScript verwendet.

2.1.1 Hypertext Markup Language (HTML)

HTML ist, wie der Name schon sagt, eine Auszeichnungssprache (engl. markup = Auszeichnung) und beschreibt die Struktur einer Webseite. HTML besteht aus einer Serie von ineinander verschachtelten Elementen bei denen der Browser weiß, wie der Inhalt anzuzeigen ist. Diese Elemente werden durch sogenannte <tags> repräsentiert, welche von den Browsern benutzt werden um die Seite zu rendern, jedoch werden die tags dabei nicht angezeigt[5]. HTML gibt also lediglich die Struktur eines Dokuments an, jedoch nicht sein Erscheinungsbild. Die Sprache wurde um das Jahr 1990 am CERN entwickelt und erschien in ihrer ersten Version am 3. Nov. 1993. Seit dem wurde HTML mehrfach weiterentwickelt und bekam im Laufe der Jahre immer wieder Korrekturen und Neuauflagen, sodass man heute (seit 28. Oktober 2014) bei HTML5 angekommen ist. Die Sprache selbst und ihre Syntax ist umfangreich, jedoch einfach aufgebaut und leicht zu erlernen. Ein HTML Dokument besteht fast immer aus[6]:

- Dokumenttypdeklaration: Besteht aus Dokumenttypdeklaration (DTD)
- Basiselement (HTML-Element): Umgibt alle anderen Elemente
- Dokumentkopf (HEAD-Element): Enthält technische Informationen und Verweise auf externe Dateien oder Bibliotheken, die genutzt werden sollen.
- Dokumentkörper (BODY-Element): Enthält die Inhalte des Dokuments für die Ausgabe im Browser.

Elemente bestimmen Bereiche eines Dokuments und geben im besten Falle an, welche Ausgabe in diesem Bereich zu erwarten ist. Sie können auch wie schon erwähnt ineinander verschachtelt werden, wodurch eine Hierarchie mit verschiedenen Ebenen entsteht. Bei der Verschachtlung muss jedoch die Semantik zulässig bleiben, da ein Element nicht jedes beliebige Element enthalten darf. Elemente bestehen immer aus einer eckigen öffnenden Klammer ‚<', einer Elementbezeichnung z.B. ‚div' und einer eckigen schließenden Klammer ‚>', wobei es pro Bereich immer ein öffnendes und ein schließendes Element gibt. Anders als das öffnende Element hat das schließende Element noch einen Schrägstrich ‚/' vor der Elementbezeichnung. Zum besseren Verständnis folgt hier ein kleines Beispiel.

```
<!DOCTYPE html>
	<html>
		<head>
			<title> </title>
			<meta charset="utf-8"/>
			<link rel="stylesheet" href="style.css"/>
			<script src=“script.js“/>
		</head>
		<body>
			<header> </header>
			<div id=‘Beispiel_ID‘ class=‘Beispiel_Class‘> <div>
			<button onclick=Beispiel_Funktion() >
			<script> </script>
		</body>
	</html>
```

Wie man im vorliegendem Beispiel sehen kann gibt es die Möglichkeit, Elemente mit Attributen zu versehen, um so genauere Eigenschaften über diese zu definieren. Diese Angaben können zum Beispiel CSS-Klassen oder Javascript-Funktionen sein. Javascript kann über eine „Source“-Angabe nachgeladen oder direkt im Dokument zwischen den „<script>“-tags beschrieben werden (`<script> Javascriptcode </script>`).

2.1.2 Cascading Style Sheets (CSS)

CSS ist eine Stylesheet-Sprache, die das Aussehen eines HTML-Dokuments beschreibt, also zum Beispiel welche Farben, Formen, Größen oder Positionen Elemente haben. Mit ihr kann das Aussehen einzelner Elemente oder ganzer Gruppen definiert werden. Das arbeiten mit Gruppen innerhalb CSS kann viel Zeit sparen, da man die Elemente einer Gruppe über eine Klassenangabe gleichzeitig bearbeiten kann. Auch kann man ein Stylesheet für mehrere Webseiten gleichzeitig benutzen und so das Aussehen mehrerer Seiten gleichzeitig bearbeiten. Die Stylesheets können in externen Dateien mit der Endung ‚.css' abgespeichert sein, in einem `<style>`-Element angeben sein oder an dem Element selbst über deren Style-Attribut eingebunden werden. Hier sollte darauf geachtet werden, wie man Style-Anweisungen einbindet, da sich Angaben bei verschiedenen Style-Anweisungen durch stärker-bindende Anweisungen überschreiben können. So bindet eine Angabe über das Style-Attribut stärker als

die Angaben im Style-tag. Diese Angaben hingegen binden stärker als die Angaben, die über eine CSS-Datei eingeladen werden. Diese Anweisungen werden mit Style-Eigenschaften bestimmt, bei der jede Eigenschaft einen Wert (teilweise auch mehrere Werte) zugewiesen bekommt. Dabei werden Eigenschaft und Wert mit einem Doppelpunkt ‚:' getrennt und bei mehreren Angaben mit einem Semikolon ‚;' verknüpft. Der Block wird mit schweifenden Klammern umschlossen und diesem ein Selektor vorangestellt. Dieser Selektor gibt an auf welches HTML-Element eine Style-Anweisung angewendet werden soll. Es sollte dabei auf eine richtige Syntax geachtet werden da bei falschen Eingaben der ganze Anweisungsblock nicht funktioniert. Auch hier gibt es zum besseren Verständnis ein Beispiel[7].

```
button {
    background-color: #4CAF50; /* Kommentar */
    color: white;
    font-size: 16px;
    padding: 15px 32px;    }
```

Eine genaue Erläuterung der möglichen Attribute und deren Eigenschaften und Werte findet man schnell im Internet und würde hier zu weit führen. Attribute können jedoch nicht nur statisch gesetzt werden sonder auch dynamisch erzeugt und verändert werden. Für so eine dynamische Anpassung einer Webseite benötigt man Javascript.

2.1.3 Javascript

Javascript ist eine Skript-Sprache die 1995 von Netscape entwickelt wurde, um Formulareingaben der Webseitenbesucher zu überprüfen. Mittlerweile findet JavaScript in allen Browsern Verwendung, wo es dafür sorgt, dass Webseiten sich dynamisch aufbauen oder sich dem Nutzer anpassen können. So können komplexe Kontrollabfragen programmiert oder Datenbanken verknüpft werden, um zum Beispiel Benutzereingaben zu prüfen oder um auf ein Ereignis zu reagieren. Auch umfangreiche Browsergames können mit Javascript geschrieben werden[8]. Da Javascript sowohl auf die Texte eines HTML-Dokuments zugreifen und diese verändern kann, als auch auf die Style-Anweisungen von Elementen zugreifen kann, ist es ein gutes Werkzeug zum Bauen einer Webseite. Dies kann gut umgesetzt werden, indem sämtliche HTML-Elemente ein ID-Attribut bekommen, die möglichst ein eindeutigen Wert hat. Über die ID kann mit Javascript auf ein Element zugegriffen werden und anschließend einer Variablen als Wert zugewiesen werden.

```
var Element = document.getElementById("id")
```

Soll einem Element zum Beispiel ein Text zugeordnet werden, dessen Style verändert werden oder soll es ausgeblendet werden, kann auf dessen Attribute zugegriffen werden. Diese können dann neu konfiguruert werden.

```
Element.innerHTML = "Hello JavaScript";
Element.style.fontSize = "35px";
Element.style.display = "none";
```

Informationen, die von einem Server geladen oder lokal berechnet wurden, können an ein HTML-Element angehängt und dem Nutzer angezeigt werden. Dies geht auch anders herum, sodass die Eingaben der User per klick an einen Server gesendet werden. Im folgenden Beispiel ist ein HTML-Eingabefeld beschrieben, in das Daten eingegeben werden, die nach bestätigen des Buttons durch die Funktion `Beispiel_Funktion()` an einen Server mit der Adresse `/ServeradresseUndURL.php` geschickt werden.

```
<html>
	<body>
		<p>Gebe hier deine Daten ein die du zum Server schicken willst</p>
		<form id="id" action="/ServeradresseUndURL.php">
			Data: <input type="text" name="data"><br>
			<input type="button" onclick="Beispiel_Funktion()" value="Klick">
		</form>
		<script>
			function Beispiel_Funktion() {
				document.getElementById("id").submit();
			}
		</script>
	</body>
</html>
```

Javascript ist mit seinem standardisiertem Sprachkern ECMAScript (ECMA 262) eine dynamisch typisierte, objektorientierte Skriptsprache, mit der je nach Bedarf objektorientiert, prozedural oder funktional programmieren werden kann[9]. Bei der Programmierung mit Javascript sollte darauf geachtet werden, dass JavaScript eine ereignisgesteuerte Sprache ist. Dies bedeutet, dass JavaScript nicht auf eine Antwort wartet, sondern weitere Aufgaben weiterhin ausführt, während es auf andere Ereignisse wartet. Es ist nicht so, dass in JavaScript die Funktionen nicht in der gewünschten Reihenfolge ausgeführt werden, vielmehr wird nicht auf eine Antwort von first() gewartet, bevor second() ausgeführt wird. Es kann nicht einfach eine Funktion nach der anderen aufgerufen und gehofft werden, dass sie in der richtigen Reihenfolge ausgeführt werden. Dafür gibt es Callback-Funktionen. Die sorgen dafür, dass bestimmter Code second() erst ausgeführt wird, wenn der andere Code first() bereits ausgeführt wurde[10].

```
function first(number, callback) {
	alert(„Started task Number ${number}.“);
	callback();
}
function second(){
	alert(„Finished task“);
}
first(„42“, second);
```

JavaScript findet nicht nur Verwendung auf Webseiten. Auch viele Desktop- und Serverprogramme verwenden JavaScript, das bekannteste Serverprogramm ist Node.js. Auch einige Datenbanken, wie z. B. MongoDB verwenden JavaScript als Programmiersprache[11].

2.2 Backend-Entwicklung

Anders als bei der Frontend-Entwicklung geht es hier nicht darum, was der Nutzer sehen kann sonder um die Entwicklung dessen, was sich im Hintergrund abspielt. Das Frontend ist zwar die Oberfläche, aber ohne Backend gäbe es dieses nicht, auch wenn das Backend für den Nutzer unsichtbar ist. Im Backend werden die programmierten Funktionen einer Software festgelegt. Das Backend besteht aus der Programmierung der Logik einer Webseite. Die Designs und alle Inhalten wie Texte oder Bilder, die später im Frontend zu sehen sind, werden im Backend berechnet und eingespielt. Auch Datenbanken sind Teil des Backend. Dies kann man sich vorstellen wie ein Konzert. Das Frontend ist die Bühne, auf der das Stück gespielt wird und das was dahinter passiert, das Anlegen der Kostüme, die Technik und die Maske, sind das Backend[12]. In dieser Arbeit ist das Backend der Webserver, dessen Funktionen mit den Pin des Raspberry Pi kommunizieren. Er wird ebenfalls in Javascript geschrieben und auf der Plattform Node.js ausgeführt.

2.2.1 Node.js

Node.js ist eine asynchrone ereignisgesteuerte JavaScript-Laufzeitumgebung, die auf der V8-JavaScript-Engine von Chrome basiert. Wegen diesen Eigenschaften eignet sich Node.js gut zum Erstellen skalierbarer Netzwerkanwendungen. So können viele Verbindungen gleichzeitig verarbeitet werden, wobei bei jeder neuen Verbindung eine Callback-Funktion ausgelöst wird. Bei ausbleibenden Verbindungen ist keine Arbeit zu erledigen und Node.js ist im Ruhezustand. Dies steht im Gegensatz zum heute gebräuchlicheren Parallelitätsmodell, bei dem Betriebssystem-threads zum Parallelisieren verwendet werden. Thread-basiertes Networking ist jedoch nicht effizient und schwierig zu verwenden. Da es kaum Prozessblockierende Funktionen gibt, ist es sinnvoll, skalierbare Systeme, in Node.js zu entwickeln. Ein einfacher Server der auf Anfragen mit „Hallo Welt“ antwortet, kann mit dem ‚http‘-Modul wie folgt entwickelt werden.

```
const http = require('http');
const server = http.createServer ((req, res) => {
	res.statusCode = 200;
	res.setHeader ('Content-Type', 'text/plain');
	res.end ('Hallo Welt');
});

server.listen(3000, '192.168,178.3', () => {
	console.log (`Server läuft unter http://192.168.178.3':3000/`);
});
```

Auf dem Raspberry Pi unter Verwendung von Rasbian lässt sich Node.js im Terminal mit

```
pip install node
```

installieren und danach (optional mit einer Datei) starten mit

```
node
node Datei.js
```

Mit der Anweisung require() können in Node.js externe Module, Bibliotheken, Frameworks oder selbst geschriebene Funktionen geladen und einer Variable zugeordnet werden, sodass man diese dann benutzen kann. So kann dann auch die selbstgeschriebene Klasse GPIO im Quellcode des Servers geladen und benutzt werden. Weitere sinnvolle Module sind Path, Filesystem und das Webframework ‚Express' zum einfacheren Erstellen eines Webservers oder einer API.

2.2.1.1 Express

Express ist ein, als kostenlose Open-Source-Software veröffentlichtes, Webanwendungsframework für Node.js, das zum Erstellen von Webanwendungen und APIs entwickelt wurde. Express wurde als Server-Framework-Standard für Node.js bezeichnet und wird mitunter von Fox Sports, PayPal, Uber und IBM verwendet[13]. Die Philosophie des Frameworks besteht darin, robuste, kleine Tools für HTTP-Server bereitzustellen, die sich für Anwendungen mit einer App, Websites oder öffentliche HTTP-APIs eignen. Ein einfacher Server der Anfragen mit einer beliebigen URL mit einer Index-Seite beantwortet und Anfragen mit ‚/api' als Pfad in der URL mit einem Dictionary im JSON-Format beantwortet, sieht wie folgt aus.

```
var express  = require('express');
var path     = require(‚path');
var app      = express();

app.set('view engine', 'ejs');
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());

app.get('/\*', function(req, res){
      res.render('index', LEDs_dict);
});
app.get('/api*', function(req, res){
      res.json(LEDs_dict)
});
app.listen(3000, function () {
    console.log('Simple LED Control Server Started on Port: 3000!')
});
```

Mit app.set und app.use können und müssen einzelne Einstellungen für den Server konfiguriert werden. So wird hier mit app.set mitgeteilt welche ‚view engine' genutzt werden soll. In dem Fall steht ‚ejs' für ‚embedded JavaScript'. Mit app.use wird der App bzw. dem Programm mitgeteilt, dass der Pfad ‚/public' für den statischen Inhalt, also Bilder und Scripte, benutzt werden soll. Der Ausdruck express.json() steht für eine integrierte Middleware-Funktion die eingehende Anfragen mit JSON-Payload analysiert. Dies ist wichtig für die API. Die Funktion basiert auf dem Modul ‚body-parser'[13].

Die Variablen req und res stehen für Request und Response und sind Objekte mit deren Daten man hier gut Arbeiten kann. So kann zum Beispiel bei dem Request-Object die IP, Bildschirmmaße oder mitgesendete Daten abgefragt werden. Bei dem Response-Objekt kann der Header modifiziert und der Status-Code angegeben werden z. B. 404 Not Found, 400 Bad Request oder 200 OK.

2.2.1.2 Path

Das Node.js-Modul ‚path' bietet Funktionen zum Arbeiten mit Datei- und Verzeichnispfaden. Da die Standardoperation des Pfadmoduls von dem verwendeten Betriebssystem abhängt, auf dem eine Node.js-Anwendung ausgeführt wird, geht das Pfadmodul speziell bei Windows-Betriebssystem davon aus, dass Pfade in dessen Stil verwendet werden. Daher kann die Verwendung von path.basename() unter Linux und Windows zu unterschiedlichen Ergebnissen führen. Die path.join()-Methode fügt angegebenen Pfadsegmente zu einem Pfad zusammen. So können beliebig viele Pfadsegmente angegeben werden. Die angegebenen Pfadsegmente müssen, durch Komma getrennte, Zeichenfolgen sein.

```
path   =require('path');

//On POSIX
path.basename('C:\\audio\\track.mp3'); //Returns: 'C:\\audio\\track.mp3'

//On Windows
path.basename('C:\\audio\\track.mp3'); //Returns: 'track.mp3'

path.join('/home/pi/Bachelorarbeit-GPIO', 'public');
//Returns:  '/home/pi/Bachelorarbeit-GPIO/public'
```

2.2.1.3 File System (fs)

Das Node.js-Modul ‚fs' bietet eine Schnittstelle für die Interaktion mit dem Dateisystem. Dies geschieht in enger Anlehnung an POSIX-Standardfunktionen. Funktionen für Interaktionen mit dem Dateisystem stehen bei diesem Modul in synchronen und asynchronen Formen zur Verfügung. Asynchrone Funktionen nehmen als letztes Argument eine Callback-Funktion entgegen, die nach Vollendung ausgeführt wird. Den Callback-Funktionen werden Argumente übergeben die von der fs-Funktion ermittelt wurden. Diese können je nach Methode verschieden aussehen. Das erste Argument ist dabei immer für eine Ausnahme reserviert und kann bei Erfolg ‚null' oder ‚undefined' sein. Mit Hilfe dieses Moduls kann überprüft werden, ob Ordner, Pfade oder Dateien existieren. Deren Inhalt kann gegebenenfalls ausgeben werden[15].

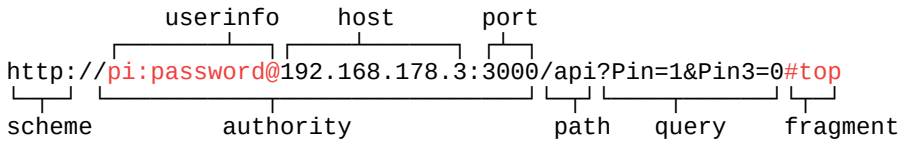
```
var fs       = require('fs');

fs.readFile('/etc/beispiel.txt', (err, data) => {
       if (err) throw err;
       console.log(data);});

fs.writeFile('/etc/beispiel.txt', data, (err) => {
       if (err) throw err;
       console.log('Data has been written!');});

fs.exists('/etc/beispiel.txt', (bool) => {
       console.log(bool);});
```

2.3 application programming interface (‚API‘)

Eine API ist eine Programmschnittstelle, die von einem Softwaresystem bereitgestellt wird, damit andere fremde Programme, bestimmte Teile dieses Softwaresystems für sich benutzen können. Eine Schnittstelle ist ein Teil eines Systems, dass die Kommunikation zweier von einander unbekannten Systemen ermöglicht. So stellt ein System eine Schnittstelle bereit, damit ein fremdes System über vorher festgelegte Kriterien Anfragen an dieses System stellen kann, um dessen Dienste und Funktionen zu nutzen. Dabei weiß das fremde Programm nicht, wie die Daten vom System im Hintergrund berechnet oder ermittelt wurden. Eine API ist eine Art „Abkommen“ zwischen Client und Server, bei dem der Client, wenn er eine Anfrage in einem bestimmten Format stellt, immer eine Antwort in einem bestimmten Format erhält oder eine definierte Aktion einleitet. Eine API wird an vielen Stellen genutzt, sodass man sie unter anderem in webbasierten Systemen (Web-API), Betriebssystemen, Datenbanksystemen, oder in Computerhardware und Softwarebibliotheken findet. In dieser Arbeit wird eine Web-API entwickelt, damit Web-Entwickler, die später mit diesem Tool arbeiten, leicht und schnell mit der Software interagieren können. Diese Interaktionen finden über HTTP-Anfragen an den Pfad ‚/api‘ statt. Für die Verwendung einer API wird normalerweise eine Dokumentation bereitgestellt. Die Verwendung des Programms ist in der Datei „README.md“ dokumentiert.

2.3.1 Hypertext Transfer Protocol (HTTP)

Das Hypertext Transfer Protocol (HTTP) ist ein Anwendungsprotokoll für verteilte, kollaborative Informationssysteme und ist die Grundlage der Datenkommunikation für das World Wide Web. Die Entwicklung von HTTP wurde 1989 am CERN initiiert und wurde seit dem mehrmals überarbeitet und mit mehreren Sicherheitsmechanismen wie Transport Layer Security (TLS) ausgestattet. HTTP arbeitet als Anforderungs-Antwort-Protokoll (request-response protocol) und ist im Grunde die Sprache mit der Server und Clients über das Netzwerk miteinander kommunizieren. Dies funktioniert so, dass der Browser (Client) eine Anfrage an den Server schickt und diesem mitteilt, was er möchte. Ist die Anfrage zulässig, schickt der Server dem Client die Ressourcen als Antwort zurück. Die Art der Antwort variiert dann je nach Anfrage(-Methode). Die Adresse des zu antwortenden Servers wird mithilfe der URL im Netzwerk ermittelt. Diese wird unter Einschluss aller optionalen Komponenten beispielsweise so aufgebaut[16].



Bei HTTP gibt es je nach Version verschiedene Request-Methoden. Bei der Version HTTP/1.0 gab es die Methoden GET, HEAD, POST. Diese wurden mittlerweile durch HTTP/1.1 mit OPTIONS, PUT, DELETE, TRACE and CONNECT ergänzt. Diese sind aber für diese Arbeit nicht notwendig und damit im Weiteren irrelevant. Die GET-Anfrage ist eine der am häufigsten verwendeten HTTP-Methoden und wird genutzt um Daten von einer angegeben

Ressource anzufordern. Hier ist zu beachten, dass eine Abfrage (query) in der URL einer GET-Anfrage gesendet wird. Des Weiteren können die Anfragen zwischengespeichert werden, verbleiben im Browserverlauf und können mit einem Lesezeichen versehen werden. Es sollten niemals sensible Daten über die URL versendet werden, da sie auf dem Weg zum Server von Dritten eventuell eingesehen werden können. Die Antwortnachricht einer GET-Anfrage enthält unter anderem folgende Elemente

- Statuszeile — z. B. HTTP / 1.1 200 OK
- Antwortkopf (header) — z. B. Inhaltstyp: Text / HTML
- Nutzlast (body) — z. B. HTML-Seite

Die HEAD-Anfrage ist fast identisch zur GET-Anfrage, jedoch bekommt man hier als Antwort nur die Statuszeile und den Antwortkopf zugesendet, also ohne Nutzlast. Dies ist nützlich um zu überprüfen welche Anforderungen zurückgegeben werden, bevor tatsächlich eine GET-Anfrage gesendet wird, beispielsweise vor dem Herunterladen einer großen Datei. Da jedoch in diesem Softwareprojekt keine großen Mengen, an Daten, verschickt werden, wird diese Methode im weiteren auch nicht benötigt. Die POST-Anfrage wird genutzt, um Daten an einen Server zu senden, um dort eine Ressource zu erstellen oder zu aktualisieren. Die Daten werden als Nutzlast also im ‚body‘ platziert. Die POST-Anfragen gehören neben den GET-Anfragen mit zu den am häufigsten verwendeten HTTP-Anfrage-Methoden, jedoch werden sie niemals zwischengespeichert, verbleiben nicht im Browserverlauf und können nicht mit einem Lesezeichen versehen werden[17]. Mit diesen beiden zuvor dargelegten HTTP-Methoden können die erstellten Webseiten mit der Software, auf dem Node.js-Server, kommunizieren und Daten anfragen und versenden. Diese Daten werden im JSON-Format versendet.

2.3.2 Das JSON – Format

JSON steht für JavaScript Object Notation und ist eine sprachunabhängige Syntax zum Speichern und leichtgewichtigem Austauschen von Daten. Das ist nötig, weil beim Datenaustausch zwischen einem Browser und einem Server nur Daten als Text versendet werden können. JavaScript-Objekte werden in ein JSON-Text konvertiert und an den Server gesendet. Entgegengesetzt kann ein JSON-Text wieder in ein JavaScript-Objekt konvertiert werden. Die Daten können dabei beliebig verschachtelt werden und müssen jedoch einen der vorgeschriebenen Datentypen haben. Die Daten müssen immer aus einem Paar aus Schlüssel und einem Wert, getrennt durch einen Doppelpunkt bestehen[18].

- Nullwert null
- Boolescher Wert true, false
- Zahl zb, 42
- Zeichenkette beginnt & endet mit Anführungszeichen (**"**)
- Array beginnt mit **[** & endet mit **]**
- Objekt beginnt mit **{** & endet mit **}**

```
var xObj = {title: "John Wick", id: 911, gender: "male"};
var xJSON = JSON.stringify(xObj);

//Returns:  '{"title":"John Wick", "id":911, "gender":"male"}'


var xJSON = '{"title":"John Wick", "id":911, "gender":"male"}';
var xObj = JSON.parse(xJSON);

//Returns: [object Object]
```

2.4 Der Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer, welcher von der Raspberry Pi Foundation aus United Kingdom hergestellt wird und mittlerweile in der 4ten Generation erhältlich ist. Seit seinem Erscheinen im Jahr 2012 wurde er bereits mehr als 22 Millionen mal verkauft[19].

Der Raspberry Pi hat etwa die Größe eines Smartphones, einen geringen Stromverbrauch von 5 Watt ohne Peripheriegeräte (Stromkosten/Jahr bei 30Cent/kWh - 15€/Jahr) und kostet in der Anschaffung nur knapp 40€. Deshalb ermöglicht er gerade für junge Menschen einen kostengünstigen Einstieg ins Entwicklerleben. Die aktuelle Version (Raspberry Pi 4 B) hat unter anderem ein LAN-Port, vier USB-Ports und einen Bluetooth- und WLAN-Chip sowie ein DSI-Anschluss für Displays. Der Prozessor ist aus der ARM Familie. Seinen Erfolg verdankt dieser Einplatinencomputer jedoch vielmehr der Leiste mit Steckern für allgemeine Ein- und Ausgabezwecke; in der Fachsprache auch General Purpose Input/Output (GPIO) genannt. Die Zielgruppe des Raspberry Pi sind vorwiegend Informatiker, Ingenieure, Studenten oder Bastler um deren Erwerb von Programmier- und Hardware-Kenntnissen zu erleichtern[19a]. Durch seiner großen und wachsenden Beliebtheit gibt es zahlreiche Bauanleitungen und Ideen für die Verwendung des Mini-Computers und eine große Gemeinschaft an Entwicklern, die bei Problemen und Fragen guten Support bieten. Die Projekte gehen von Medien-Center, Spielekonsole über Proxy-Server bis hin zu intelligenten Spielzeugrobotern. Deshalb eignet sich der Mini-Computer mit seiner GPIO Steckerleiste sehr gut für die Entwicklung einer eigenen Heimautomatisierung. Da seit dem Erscheinen des ersten Raspberry Pi im Jahr 2012 immer wieder neue Modelle herauskamen, um den Anforderungen und Bedürfnissen der Nutzer gerecht zu werden, muss wegen der je nach Generation und Model variierenden Systemarchitekturen darauf geachtet werden welches Model benutzt wird. Eine genauere Erläuterung der Unterschiede von z. B. CPU oder Arbeitsspeicher würde den Rahmen dieser Arbeit überschreiten, weshalb hier nur auf die, für die Arbeit relevanten, Unterschiede der GPIO Anschlüsse eingegangen werden kann. In den ersten Modellen der Raspberry Pi Serie hatten die Platinen der Modelle 1A & 1B nur 26 Stecker an der GPIO-Steckerleiste. Erst 2 Jahre später bekam der Raspberry Pi seine Steckerleiste mit 40 Pins. Bei dem Lite-Model „Zero W“ ist darauf zu achten, dass dieser wegen seiner sehr kompakten Bauweise keine Pins hat, sondern Lötstellen besitzt. Der Raspberry Pi kann wegen seines geringen Stromverbrauchs auch mit einer Powerbank betrieben werden und ist deshalb auch für Projekte im Garten oder der Natur geeignet. Hier sollte jedoch zum Schutz auf eine Hülle um die Platine geachtet werden.

2.4.1 general purpose input/output (GPIO)

Ein GPIO-Pin, zu Deutsch auch Allzweckeingabe/-ausgabe-Pin, ist ein allgemeiner Kontaktstift an einem integrierten Schaltkreis (IC). Sein Verhalten ist durch Programmierung frei bestimmbar und kann per Software gesteuert werden. Da den GPIO-Kontakten im allgemeinen kein Zweck vorgegeben ist, sind sie standardmäßig unbelegt. Ein beliebtes Gerät, welches GPIO-Pins verwendet, ist der Raspberry Pi. Am Raspberry Pi, im speziellen am 3B+ sind an der Seite der Platine eine Reihe von Steckern, von denen einige als GPIO frei programmierbar sind. Mit derer Hilfe können viele Projekte realisiert werden in denen der Raspberry Pi als zentrale Steuereinheit mit externen Sensoren oder Bauteilen über die GPIO-Schnittstelle kommuniziert und Daten austauscht. Die Pins fungieren als Schalter, die 3,3 Volt ausgeben, wenn sie auf HIGH also 1 stehen und keine Spannung ausgeben, wenn sie auf LOW also 0 eingestellt sind. Es kann ein Gerät, an einem bestimmten GPIO-Pin angeschlossen und per Software gesteuert werden, aber auch geprüft werden ob ein Signal (Spannung) einer externen Quelle anliegt[19b].

Um einen Pin zu benutzen, muss er zunächst Aktiviert werden, also dem System muss mitgeteilt werden das der Pin nun verwendet wird. → export
Wenn ein Pin aktiviert ist muss seine Funktion (Richtung) eingestellt werden, also ob mit dem Pin gelesen oder geschrieben wird. → direction → in / out
Nach Exportieren eines Pins steht dieser standardmäßig auf in.
Wenn der Pin auf schreibend (out) gesetzt ist, kann dessen Wert gesetzt werden; also der pin kann ein- und ausgeschaltet werden. Wenn der Pin lesend (in) eingestellt ist, kann sein Wert abgefragt werden. → value → 1 / 0

Diese Operationen müssen im Terminal mit Administratorrechten ausgeführt werden. Hier wird beispielsweise der Pin 4 eingeschaltet und der Pin 5 gelesen danach werden beide wieder ausgeschaltet.

```
sudo -i
echo 4 > /sys/class/gpio/export
echo out > /sys/class/gpio/gpio4/direction
echo 1 > /sys/class/gpio/gpio4/value

echo 5 > /sys/class/gpio/export
echo in > /sys/class/gpio/gpio5/direction
cat /sys/class/gpio/gpio5/value

echo 4 > /sys/class/gpio/unexport
echo 5 > /sys/class/gpio/unexport
```

3. Konzept / Anforderungen

Nachdem die Grundlagen erläutert und ein Überblick über benötige Werkzeuge geschaffen wurden, wird in diesem Kapitel der Aufbau des Software-Kits konzeptionell beschrieben und auf Anforderungen eingegangen die das Kit mitbringen sollte. Bei dem Konzept des Softwarekits wird beschrieben wie mit den vorgestellten Werkzeugen das Projekt umgesetzt werden kann, wie diese Werkzeuge miteinander interagieren und daraus ein Gesamtwerk entsteht. Im Anschluss werden Anforderungen an das Software-Kit erörtert.

3.1 Aufbau des Developer-Kit

Das Software-Kit und dessen Entwicklung wird in 3 Ebenen aufgeteilt. Dies soll der besseren Übersicht dienen und für einen strukturierten sowie leicht austauschbaren Programmcode sorgen. Dabei wird die Entwicklung nach dem bottom-up Prinzip hardwarenah angefangen, weshalb als erstes eine Javascript-Klasse implementiert wird, die Funktionen zum Bedienen der GPIO-Pins über das Dateisystem zur Verfügung stellt. Mithilfe dieser Funktionen sollten die Pins ein und ausgeschaltet werden können oder ausgegeben werden, ob an einem Pin eine externe Spannung anliegt oder nicht. Auf diese Ebene aufbauend, wird der Node.js Server entwickelt, auf dem später die API läuft und der bei Anfragen eine Übersichts-Webseite übermittelt. Dieser soll auf API-Anfragen und Anfragen der Webseite antworten und gegebenenfalls mithilfe der GPIO-Klasse mit den Pins interagieren. Als nächstes wird die Beispiel-Anwendung geschrieben. Sie bildet die letzte Schicht. Dies wird eine Seite sein, die von dem Node.js-Server nach Aufrufen der Webseite übermittelt und auf dem Browser ausgeführt wird. Dieses Anwendungsbeispiel soll dann nur über die API mit dem Server kommunizieren. Dabei findet die Kommunikation über HTTP-Anfragen statt, die Daten im standardisiertem JSON-Format austauschen.

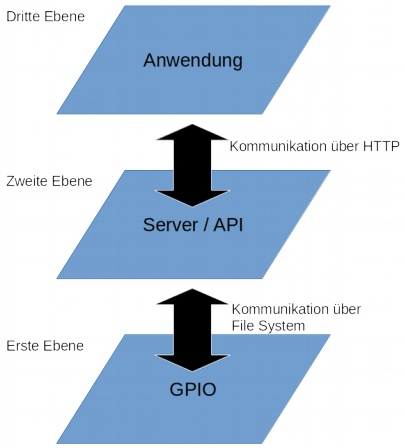


Abbildung 1: Layout des Konzepts

Das Projekt wird in einem Repository von GitHub organisiert und dokumentiert, sodass es dort nach Veröffentlichung von anderen Entwicklern heruntergeladen und benutzt werden kann. Es sollte leicht ausführbar sein, indem Node.js gestartet und das Script des Servers als Argument mit übergeben wird. Sonst sollten keine weiteren Eingaben getätigt werden müssen.

3.2 Erste Ebene – Die GPIO Klasse

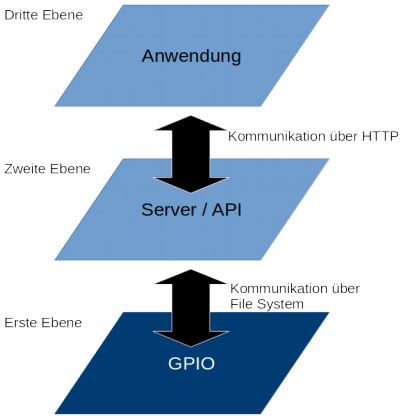


Abbildung 2: Layout des Konzepts – 1. Ebene markiert

Die Javascript-Klasse GPIO ist wie bereits bekannt die unterste Ebene des Software-Kits und wird somit als erstes programmiert. Sie soll dem späteren Skript, in diesem Fall dem Node.js-Server, die Schnittstelle zu den Pins bieten. Dabei wird sie so entworfen, dass deren Funktionen unter anderem im Hauptskript importiert und verwendet werden können. Beim Aufruf dieser Klasse müssen die GPIO Port-Nummer und bei Set-Funktionen ein Wert angegeben werden. Bei Get-Funktionen wird der Wert zunächst ausgelesen und dann als Zeichenkette (String) zurückgegeben.

Die Klasse sollte folgende Funktionen bieten:

- Pin x wird geöffnet → export
- Pin x wird eine Richtung in/out zugewiesen → set direction
- Richtung von Pin x wird geprüft und zurückgegeben → get direction
- Pin x wird ein Wert 1/0 zugewiesen und damit Ein/Ausgeschaltet → set value
- Wert von Pin x wird geprüft und zurückgegeben → get value
- Pin x wird geschlossen → close

Die Funktionen werden dabei so gebaut, dass sie nach ausführen einer Aktion eine Meldung ausgeben, ob diese Erfolgreich war oder Fehler aufgetreten sind. Es kann passieren, dass ein Pin eingeschaltet werden soll (value → 1), aber dessen Richtung noch auf ‚in' steht oder dass ein anliegender Wert eines Pins ausgegeben werden soll, aber dieser noch auf ‚out' steht. Wenn möglich, sollten die Funktion solche fehlerhaften Vorbedingungen von selbst beheben. Dies sorgt für eine bessere Stabilität da eine Funktion dann trotz dieser fehlerhaften Vorbedingungen ausgeführt werden kann. Jedoch sollte nun darauf geachtet werden, dass man nicht aus Versehen einen Pin einschaltet, an dem elektrisch empfindliche Bauteile angeschlossen sind.

Bei den meisten Raspberry Pi Systemen wird der Systempfad zu den GPIO Daten und Funktionen ‚/sys/class/gpio/' sein. Es kann jedoch bei Versionen mit einem älterem Kernel sein, dass der Pfad ‚/sys/devices/virtual/gpio' lautet. Das sollte beachtet werden und gegebenenfalls anpassbar sein. Bei der Programmierung sollte zudem darauf geachtet werden, dass die Funktionen asynchron gestaltet und mit einer Callback-Funktion ausgestattet werden, damit ein späteres Programm nicht blockiert wird und sich dessen Laufzeit damit verschlechtert. Um einen Wert wie 0 oder 1 in eine Datei zu schreiben oder einen Wert einer Datei auszugeben wird das Node.js-Modul ‚File System', insbesondere die Funktionen fs.exists, fs.readFile und fs.writeFile, verwendet. Mit ihnen sollten sich die Funktionen der GPIO-Klasse verwirklichen lassen.

3.3 Zweite Ebene – Der Server und die JSON-API

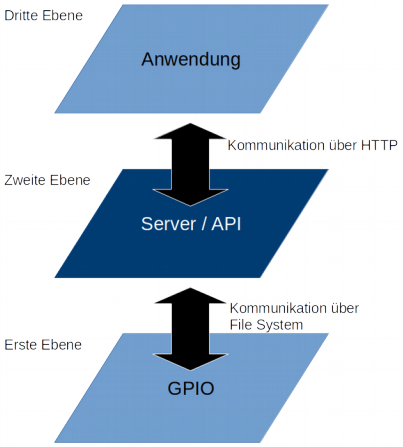


Abbildung 3: Layout des Konzepts – 2. Ebene markiert

Der Server ist das Kernstück des Software-Kits, dessen Logik auf den Funktionen der GPIO-Klasse aufbaut. Er soll später die Anfragen der Anwendungen über eine API entgegennehmen und auf Befehle zur Pin-Steuerung reagieren. Zusätzlich werden hier weitere Funktionen für eine Übersichtsseite implementiert, damit die Entwickler die Möglichkeit haben, die Pins per Klick zu steuern und so ihre Schaltungen ausprobieren können. Die Umsetzung des Servers erfolgt über eine Verknüpfung der Methoden des Node.js-Frameworks „Express“ mit den Funktionen der GPIO-Klasse zu einem Gesamtsystem.

Der Express-Server antwortet auf HTTP-Anfragen und ruft je nach Anfrage-URL bzw. Pfad eine andere Funktion auf, um eine Antwort zurückzugeben. So sollte der Server bzw. die darauf laufende Software auf verschiedene Anfragen verschieden reagieren und mit passenden Daten antworten. Anfragen der Webseite werden beispielsweise so aussehen:

```
http://192.168.178.3:3000/index/
http://192.168.178.3:3000/Anwendungsbeispiel/\*
http://192.168.178.3:3000/led/out/
http://192.168.178.3:3000/led/export/
http://192.168.178.3:3000/led/unexport
```

Das Framework Express beantwortet solche HTTP-Anfragen mithilfe des (req, res)-Objektes, das für die Anfrage selbst (Req = Request) und die Antwort auf die Anfrage (Res = Response) steht. In den Attributen dieses Objekts werden die Daten gespeichert, die versendet wurden oder noch gesendet werden, sodass dort nach Schalten der Pins die Daten abgelegt und dann versendet werden können. Da bei API-Anfragen nur JSON-Daten und keine Seite mit zurückgeschickt werden muss, beim normalen Aufrufen der Webseite aber schon, variieren die Antworten des Servers je nach Anfrage. Die zu versendenden HTML Seiten (also Übersichtsseite und Anwendungsbeispiele), die der Server verwendet, sind im Ordner „views“ abgelegt. Die genutzten Bilder, JavaScript und CSS Dateien werden im Ordner „Public“ abgelegt, damit diese nachgeladen werden können. Ein weiterer Unterschied ist, dass die API bei einer einzelnen Anfrage mehrere Pins „gleichzeitig“ schaltet, die Webseite dies jedoch meist einzeln macht (Anwendungsbezogen durch Klick-Steuerung). Es sollte darauf geachtet werden, dass mit GET-Anfragen nur Informationen abgefragt, jedoch keine verändert werden können. Damit sollen Pins nicht, durch falsch eingegebene URL-Eingaben, verändert werden können. Veränderungen der Richtungen oder Werte der Pins müssen mit POST-Anfragen ausgeführt werden. Es sollte zudem möglich sein, zu bestimmen, ob man nur die

Informationen eines, zweier oder aller Pins haben möchte. Es wird eine Datenstruktur gebraucht, um sowohl mit den Pins arbeiten zu können, als auch die Status-Informationen der Pins an die Anwendungen weitergeben zu können oder auf deren POST-Befehle reagieren zu können. Der Server sollte darauf achten, dass die Eingaben über die API richtig sind und gegebenenfalls eine Fehlermeldung zurückgeben. Damit weiß der Entwickler, warum etwas nicht funktioniert hat. Die API wird nach dem Beispiel der RESTfull-API gestaltet, bei denen es die Funktionen GET, POST, PUT und DELETE gibt. Da hier keine Daten verändert oder gelöscht werden, werden in diesem Kontext nur die Methoden GET und POST gebraucht,.

Der Server bzw. das Software-Kit sollte von der Struktur her weniger statisch als dynamisch gestaltet sein, weshalb Dateipfade immer relativ und nicht absolut angegeben werden, sodass das Programm von allen Positionen im Dateisystem unkompliziert gestartet werden kann. Der Quellcode sollte leicht modifizierbar gestaltet werden, damit dieser wenn nötig leichter verändert werden kann. So können Veränderungen an den GPIO-Pfaden durch eventuelle System-Updates, repariert bzw. angepasst werden. Auch der Port des Servers und das Ausgabeverhalten in der Konsole (Logging) sollte bestimmbar sein.

3.4 Dritte Ebene – Die Anwendung und die JSON Kommunikation

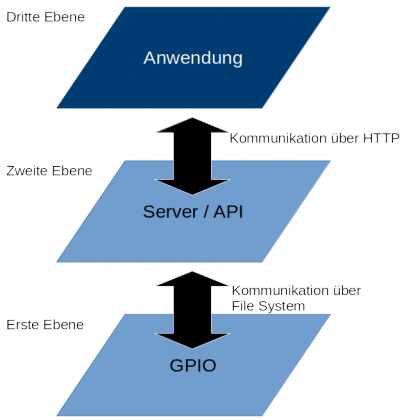


Abbildung 4: Layout des Konzepts – 3. Ebene markiert

Wenn der Server funktionsfähig ist und die Pins über die API ansprechbar sind, können nun beliebige Funktionen für eine Hausautomatisierung ausgedacht und entwickelt werden. Die Webseiten oder Programme, die ein Entwickler dann schreiben kann, können dann mit den Pins interagieren. Für die Interaktion kann zum Beispiel der Wert eines Pins x ausgelesen werden und anhand des Ergebnisses wird der Pin y dann geschaltet. Um die Informationen eines Pins zu holen sollte eine einfache GET-Anfrage benutzt werden. Dabei kann dann entweder die Information eines bestimmten Pins abgefragt werden oder standardmäßig die Information aller Pins. Bei der Anfrage sollte der Anfragende, die Informationen in einem JSON-Format bekommen, damit er leicht mit den Daten weiterarbeiten kann. Diese Informationen beinhalten dann Richtung und Wert eines, mehrerer oder aller PINs. Mit einer POST-Anfrage an die API können die Pins dann konfiguriert und geschaltet werden. Dabei sollte es reichen, nur die Pins anzugeben, die benutzt werden. Mit diesen beiden Werkzeugen können nun z. B. mithilfe von jQuery leicht Anfragen in eine Webseite eingebaut werden, um sich so eine beliebige Logik mit elektrischen Bauteilen, Platinen und Sensoren zu entwerfen. So kann in größeren Heim-Projekten ein eigenes „Internet of Things“-Gerät entwickelt werden, welches sich über den Webbrowser im eigenen Heimnetz bedienen lässt.

Als Anwendung sind die Übersichtsseite und zwei kleine Beispielanwendungen geplant. Als Anwendungsbeispiel wird es einmal eine einfache Seite geben, bei der die API-Anfragen GET und POST beispielhaft vorgeführt sind und bei der anderen Seite kann ein Wecker gestellt werden, der als Alarm eine POST-Anfrage ausführt. Auf der Übersichtsseite die es geben wird, kann jeder Pin mit einfachen Mausklicks konfiguriert und geschaltet werden. Die Seite soll einen Überblick verschaffen, welche Pins wie konfiguriert sind und die Bedienung derer erleichtern. Es sollte sehr intuitiv gestaltet sein, damit der Entwickler keine große Einarbeitung braucht, sondern gleich mit seiner Arbeit beginnen kann. Nach der Konfiguration eines Pins soll dieser schaltbar (anklickbar) sein. Beim Anklicken eines Pins wird eine Anfrage an den Server geschickt um den gewählten Pin ein/aus zu schalten oder um dessen Wert abzufragen.

Die Seiten sollten alle mithilfe von CSS Klassen im gleichen Design programmiert werden, damit sehen sie nicht nur schöner aus, sie sind auch leichter anpassbar. So können Hintergründe und Farben für die ganze Seite oder gleich mehrerer Seiten schnell anpassbar gemacht werden. Die JavaScript Dateien, für die dynamische Veränderung der Seiten, sollten ebenfalls übersichtlich sein und in externe Dateien ausgelagert werden.

3.5 Anforderungen

Das Entwickler-Kit (Developer-Kit) sollte in erster Linie für den Entwickler eine Unterstützung in der Steuerung der GPIO-Pins sein, damit dieser sich mehr darauf konzentrieren kann, was er bauen möchte, anstatt wie er es umsetzt. Eine wichtige Anforderung an das Kit ist deshalb, dass es verlässlich ist. Die Angaben der Software sollten immer die richtigen Informationen über die Pins zurückgeben. Der Entwickler muss sich darauf verlassen können, dass die ihm vorliegenden Daten stimmen, damit er auf dieser Sicherheit eigene Projekte entwickeln kann die dann auch immer so funktionieren, wie er sich das vorstellt. Zudem soll das Entwickler-Kit intuitiv, leicht verständlich und leicht benutzbar sein, damit es möglichst oft verwendet wird oder auch von der Gesellschaft weiterentwickelt wird. Der Entwickler sollte das Kit, nach einer kurzen Installation von Node.js und dessen Bibliotheken, einfach und schnell verwenden können. Zudem sollte er sich auf der Webseite leicht zurecht finden. Damit das Programm bei falschen Eingaben über die API nicht abstürzt, sollte es diesen gegenüber abgesichert sein. Bei solchen Falscheingaben ist es immer gut, wenn der Entwickler eine Rückmeldung über die Art des Fehlers erhält, damit er diese einfacher beheben kann. Zusätzlich sollte es bei der API möglich sein, anzugeben von welchen Pins man die Informationen haben möchte, damit nicht die Statusinformation aller Pins versendet werden müssen. So wird der Datenverkehr im Netzwerk minimiert. Die GPIO-Klasse sollte so implementiert werden, dass sie auch von anderen Programmen ohne Server und API verwendet werden kann und somit ein unabhängiges Programm-Modul ist. Da dieses Software-Kit für die Zwecke der Heimautomatisierung entwickelt wird ist es nicht notwendig, dass die GPIO-Klasse schnell in ihrer Laufzeit oder reich im Umfang ihrer Funktionen programmiert wird. Es wäre jedoch Wünschenswert. Soll die GPIO-Klasse nun doch erweitern werden, kann entweder der Code erweitert oder die GPIO-Klasse ausgetauscht werden. Bei der Recherche für diese Arbeit wurden 3 verschiedene GPIO-Klassen auf GitHub.com gefunden, die jedoch alle ihre Vor- und Nachteile hatten.

4. Realisierung

Damit man die Entwicklung der Software besser nachvollziehen oder nachahmen kann, werden in diesem Kapitel zunächst die verwendeten Programmier-Werkzeuge, sowie die Arbeitsumgebungen vorgestellt. Diese waren bei der Entwicklung des Software-Kits sehr hilfreich und teilweise essentiell. Anschließend werden die einzelnen Software-Module in ihrer Entwicklung skizziert und ihr Aufbau erläutert.

4.1 Arbeitsumgebung / Werkzeuge

Die Gesamte Software wurde in einem Heimnetzwerk mit einem Laptop über Secure Shell (SSH) mithilfe von Putty auf dem Raspberry Pi programmiert. Auf dem verwendeten Raspberry Pi 3B+ läuft dessen Standard-Betriebssystem Rasbian in Version 9.4. Da bei der Entwicklung nur so ein Raspberry Pi zur Verfügung stand, kann im weiteren Verlauf der Arbeit nur noch auf dieses Model eingegangen werden. Jedoch sollten andere Modelle mit 40 Pins ebenfalls funktionieren. Zur besseren Verwaltung des Quellcodes wurde ein Repository auf GitHub eingerichtet, mit dem es erleichtert wird auf mehreren Geräten zu entwickeln.

4.1.1. Raspbian / Raspberry Pi

Raspbian ist mit seiner Desktopumgebung LXDE, die für geringe Rechnerressourcen ausgelegt ist, ein Linux-Betriebssystem für den Mikro-Computer Raspberry Pi. Das Betriebssystem eignet sich wegen der vorinstallierten Programme für Bastler, Büroarbeiten oder das Surfen im Internet. Das Betriebssystem basiert auf Debian und wurde für die ARM-Prozessoren ARMv6 des Raspberry Pi, ARMv7 des Raspberry Pi 2 und ARMv8 des Raspberry Pi 3 konzipiert[20]. Nach dem Download des Betriebssystems unter

```
https://www.raspberrypi.org/downloads/raspbian/
```

kann das Betriebssystem-Abbild nach folgender Anleitung mit einem Schreibprogramm auf eine SD-Karte gebrannt werden:

```
https://www.raspberrypi.org/documentation/installation/installing-images/
```

Anschließend kann die beschriebene SD-Karte in den Raspberry Pi gesteckt und dieser hochgefahren werden. Nach der Installation des Betriebssystems und Konfiguration des Zugangs zum Heimnetzwerk kann der Raspberry Pi nun verwendet werden. Zur Sicherheit sollte man zu Beginn der Arbeit das Standard-Passwort ändern und alles updaten, in dem man im Terminal eingibt:

```
passwd pi
sudo apt update && sudo apt upgrade && sudo reboot
```

4.1.2. PuTTY

Nach dem der Raspberry Pi vollständig eingerichtet ist, kann man sich mit diesem, über ein bestehendes Netzwerk, mit SSH verbinden. SSH ist ein Netzwerk-Protokoll mit dessen Hilfe man sichere verschlüsselte Netzwerkverbindung mit einem entfernten Gerät herstellen kann. An einem Linux-Betriebssystem ist solch eine SSH-Verbindung schon nativ über ein Terminaltfenster möglich; bei Windows-Betriebssystemen kann das frei zugängliche Programm „PuTTY“ hier heruntergeladen werden:

```
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
```

Windows hat keine Software vorinstalliert, um verschlüsselte SSH-Verbindungen aufzubauen. PuTTY ist ein schneller und kleiner SSH- & Telnet-Client für Windows, der eine Reihe an Protokollen unterstützt. Mit Putty kann mit entfernten Rechnern und Systemen im Netzwerk via SSH kommuniziert und sicher Daten ausgetauscht werden. Ein übliches Einsatzgebiet ist etwa die Administration entfernter Linux-Rechner von Windows-PCs aus[21]. Aus diesem Grund eignet sich PuTTY hervorragend, um eine Verbindung zu dem Raspberry Pi aufzubauen, um dann dort zu programmieren und um den Server starten zu können. Man braucht für die Verbindung lediglich die IP-Adresse des Gerätes mit dem man sich verbinden möchte und den Nutzernamen („pi“) inklusive Passwort („raspberry“) und den Port (22).

4.1.3. GitHub

Um nun die erste geschrieben Zeilen Code auch sicher speichern zu können, kann ein Repository auf GitHub verwendet werden. GitHub ist ein Dienst im Internet, der Entwicklungsprojekte auf seinen Servern bereitstellt (Filehosting). Das hochladen auf GitHub hat den Vorteil, dass der Quellcode in der Cloud gespeichert und von überall auf der Welt abrufbar ist. Praktisch ist das Versionsverwaltungssystem von Git. Damit kann man leichter zu alten Ständen springen oder mit mehreren Leuten programmieren, es lassen sich Fehler leichter finden und rückgängig machen, aber auch Veränderungen am Code, seit dem letzten Stand, können angezeigt werden. Es ist einfacher Projekte zu entwickeln und deren Daten wie Bilder und Konfigurationsdateien immer mit GitHub zu synchronisieren, da diese dann von dort ohne Schwierigkeiten auf anderen Geräten heruntergeladen werden können. Falls mal etwas widerwillig gelöscht wurde, kann es so wiederhergestellt werden. Bei der Arbeit mit GitHub sollte man folgende Befehle im Terminal können/kennen[22]:

- `git clone [URL]:` Herunterladen eines Git-Repository von der URL
- `git init:` Initialisieren eines lokalen Git-Projekts. Kann später auf Github oder einen anderen Host geladen werden.
- `git add:` Hinzufügen der Dateien zum lokalen Projekt
- `git commit:` Hält aktuelle Änderungen fest und legt Stand an, zu dem zurückgesprungen werden kann. Dateien müssen commited werden, bevor sie an ein Repository gesendet werden können.
- `git push:` Senden aller Commits an das Repository auf den Servern von Github
- `git pull:` Holen aller aktuellen Commits aus dem Repository von GitHub

4.2 Aufbau der GPIO-Klasse für Node.js in Javascript

Wie schon erwähnt wird bei dem Projekt mit der Entwicklung der GPIO-Klasse angefangen. Damit diese dann funktionsfähig im Quellcode des Servers eingebunden werden kann, müssen die sechs Funktionen der Klasse exportiert werden. So können diese von einem anderen Programm aufgerufen beziehungsweise importiert werden. Das wurde mit dem Schlagwort „exports.“ vor jedem Funktionsnamen in der Klasse implementiert und sieht beispielsweise so aus:

```
exports.open = function (Pin) {
      //DO SOMETHING }
```

Die GPIO-Klasse soll später Informationen über die GPIO-Pins auslesen oder deren Steuerung übernehmen können. Für einen exportierten GPIO-Pin, beispielsweise den vierten, lautet der System-Pfad für dessen Informationen

```
/sys/class/gpio/gpio4/
```

Wenn man nun auf ein Pfad oder eine Datei im Dateisystem zugreifen möchte braucht man für die Laufzeitumgebung Node.js das vorgestellte Modul „FileSystem“ oder auch nur „fs“. Das Modul wird am Anfang des Codes mit require() eingeladen und der gleichnamigen Variablen „fs“ zugeordnet, sodass dessen Funktionen im Code Verwendbar sind. Um zu prüfen ob ein Pin exportiert ist, reicht es zu schauen ob dessen Ordner unter `/sys/class/gpio/` existiert.

```
fs.existsSync(/sys/class/gpio/gpio4)
```

Falls der Pin nicht exportiert ist, existiert auch kein passender Ordner. Das Prüfen auf Existenz und das eventuelle folgende Einschalten (Importieren) bei Nichtexistenz, wird mit der ersten Funktion der Klasse namens „open“ implementiert. Diese Funktion meldet zurück, ob ein Pin offen war oder eingeschaltet wurde und führt nach Erfolg eine beliebige Callback-Funktion aus. Im Gegensatz zur „open“-Funktion sorgt die „close“-Funktion nach erfolgreicher Existenzprüfung für einen „unexport“ des angegebenen Pins. Der (un)export wird umgesetzt, indem mit dem fs-Modul wie im folgenden Beispiel die Nummer des jeweiligen Pins in die Datei „export“ bzw. „unexport“ geschrieben wird.

```
fs.writeFile(/sys/class/gpio/export', 4, function (err) {})
fs.writeFile(/sys/class/gpio/unexport', Pin, function (err) {})
```

Nachdem Pin 4 exportiert wurde, erscheint unter dem Pfad `/sys/class/gpio/` ein Ordner `gpio4` der die Dateien `direction` und `value` enthält. Um dem System mitzuteilen, ob an dem Pin 4 geschrieben oder gelesen werden soll, muss in die Datei `direction` der Wert „out“ für schreiben oder der Wert „in“ für lesen geschrieben werden. Wenn die Datei `direction` existiert, kann davon ausgegangen werden das der jeweilige Pin exportiert ist. Aus diesem Grund reicht eine Existenzprüfung der Datei unter dem jeweiligen GPIO-Ordner. Wenn die

Datei existiert kann der gewünschte Wert eingetragen werden. Existiert die Datei jedoch nicht kann die open-Funktion aufgerufen werden, damit der Pin zunächst exportiert wird. Als Callback-Funktion wird dann das Schreiben des Wertes in die Datei übergeben. Dies kann dann Beispielweise so aussehen

```
if (fs.existsSync('/sys/class/gpio/gpio4/direction')) {
      fs.writeFile('/sys/class/gpio/gpio4/direction', out, function(err) {})
} else {
      exports.open(4, function() {
      fs.writeFile('/sys/class/gpio/gpio4/direction',out, function(err) {})}}
```

Manchmal soll kein Pin konfiguriert werden, sondern es soll herausgefunden werden, wie er konfiguriert ist. Für diese Möglichkeit gibt es die getDirection-Funktion, die das Auslesen der Datei `direction` ermöglichen soll. In dieser Funktion wird erst die Existenz der Datei `direction` geprüft und dann entweder der Wert der Datei oder eine Meldung, dass der Pin nicht exportiert ist, ausgegeben. Wenn ein Pin exportiert und auf `direction` out konfiguriert ist, kann der Pin eingeschalten werden, indem eine 1 in die Datei `value` geschrieben wird. Das Schreiben eines Wertes in die Datei `value`, also das Ein- und Ausschalten eines Pins wird mit der writeValue-Funktion ausgeführt. Diese Funktion prüft zunächst ob der Pin exportiert und der Wert der Datei `direction` auf out ist, und schreibt dann bei Erfolg den gewünschten Wert in die Datei `value,` um den physischen Pin ein- oder auszuschalten. Das Auslesen des Wertes der Datei `value` bzw. das Prüfen, ob ein Pin ein oder ausgeschaltet ist wird in der Funktion readValue-Funktion fast gleich zur writeValue-Funktion umgesetzt. Erst wird geprüft, ob der Pin exportiert wurde, dann aber wird geprüft ob der Wert der Datei `direction` auf in steht. Bei Nichterfolg wird der Pin zunächst mit der setDirection-Funktion konfiguriert und dann die Datei ausgelesen.

Damit der Dateipfad in den Funktionen leicht austauschbar ist, wurde er nicht statisch einprogrammiert, sonder mithilfe einer Variablen „path“ umgesetzt. Bei älteren Raspberry Pi kann es zum Beispiel vorkommen das der Systempfad zu den GPIO anders ist.

```
var path = '/sys/class/gpio/';
var path = '/sys/devices/virtual/gpio';
```

Da alle Terminal-Ausgaben hinter if-Abfragen stehen, welche den Wert der Variablen LOGGING prüfen, können bei den sechs Funktionen alle Ausgaben generell ausgeschaltet werden. Dazu muss die Variable LOGGING auf false gesetzt werden.

```
var LOGGING = true;
```

Die Javascript-Datei wird als GPIO.js im Projektordner abgespeichert und kann über require('./GPIO.js') in ein Skript geladen werden.

4.3 Aufbau eines Express-Servers mit einer API-Schnittstelle zu den GPIO-Pins

Nachdem die GPIO-Klasse fertig programmiert ist, kann die Entwicklung des Servers begonnen werden. Der Server wird mit dem Node.js-Modul Express aufgebaut. Dazu wird das Modul mit require('express') der Variablen „express“ zugeteilt und über die Variable „app“ ausgeführt.

```
var express  = require('express');
var app      = express();
```

Zunächst muss der Server mit ein paar Einstellungen konfiguriert werden. So ist der Pfad des Ordners anzugeben, aus dem die JavaScript und CSS-Klassen aber auch die Bilder der Webseite geladen werden sollen. Zudem muss die „view engine“ gesetzt werden, die in diesem Fall „Embedded JavaScript“ ist. EJS ist eine Template-Sprache, mit der HTML-Markups mit einfachem JavaScript generiert werden können. Aus diesem Grund haben die HTML-Dokumente in diesem Projekt die Endung ejs. Da der Server Daten im JSON-Format empfangen wird, muss auch das vorkonfiguriert werden. Mit einer Angabe des Ports kann der Server nun mithilfe der listen-Methode gestartet werden. Es ist Ratsam den Start des Servers über die Konsole, unter Angabe des Ports, mitzuteilen. Diese drei Konfigurationen und die Portangabe werden folgendermaßen aufgebaut:

```
app.set('view engine', 'ejs');
app.use(express.static(path.join(__dirname, 'public')));
app.use(express.json());

app.listen(3000, function () {
      console.log('LED Control Server started on port: 3000!')});
```

Der nun erstellte Server hat bisher noch keine Funktion und reagiert nicht auf Anfragen. Damit bei einer eingehenden GET-Anfrage eine Seite versendet wird, muss mitgeteilt werden, auf welche Anfrage, welches HTML-Dokument versendet werden soll. So kann die Übersichtsseite der Pins „index.ejs“, welche als Startseite dient, bei einer beliebigen GET-Anfrage zurückgegeben werden, indem man als Pfad ein Sternchen „*“ angibt, sodass alle möglichen Anfragen passen. Möchte man ein zweites Dokument zurückgeben wenn eine Anfrage kommt muss dies mit einem Pfad spezifiziert werden damit das entsprechende Dokument zurückgegeben werden kann.

```
app.get('/Anwendungsbeispiel/\*', function(req, res){
      res.render('Anwendung', LEDs_dict);});

app.get('/\*', function(req, res){
      res.render('index', LEDs_dict);});
```

Wie im vorhergehenden Beispiel zu sehen ist, wird nicht nur das HTML-Dokument versendet, sondern auch Daten. Diese Daten sind ein Dictionary, welches verwendet wird um die Informationen über die Pins während der Programmlaufzeit zu speichern. Da die Pins nach einem Neustart des Raspberry Pi zurückgesetzt werden, müssen die Informationen auch nicht physisch auf einem Datenträger gespeichert werden. Ein Dictionary in Javascript hat die gleiche Syntax wie das JSON-Format und eignet sich deshalb hier besonders gut als grundlegende Datenstruktur für das Halten der Informationen der Pins im Arbeitsspeicher. Da es hier zwei elementare Informationen pro Pin gibt, müssen die Informationen in dem Dictionary auf zwei Ebenen verschachtelt werden. Hier gibt es zwei Möglichkeiten dies umzusetzen. Die erste Möglichkeit ist es, die Pins als Dictionary-Schlüssel (Key) in die erste Ebene zu setzen und ein weiteres Dictionary als Dictionary-Wert (Value) für jeden Pin zu setzen. In dem Dictionary stehen Informationen über Richtung und Wert des Pins. Die andere Möglichkeit ist die Richtung und den Wert als Schlüssel in die Erste Ebene zu setzen und ein weiteres Dictionary mit je allen Pins und deren Informationen in die zweite Ebene zu setzen.

```
var LEDs_dict = {
      "pin1":{"direction":"not set","value":"not set"},
      "pin2":{"direction":"not set","value":"not set"},
      "pin3":{"direction":"not set","value":"not set"},
      ...
      "pin26":{"direction":"not set","value":"not set"}},

var LEDs_dict = {
      "direction":
            {"pin1":"not set", "pin2":"not set",...,"pin26":"not set"},
      "value":
            {"pin1":"not set", "pin2":"not set",...,"pin26":"not set"},}
```

Es macht kein großen Unterschied, welche dieser Versionen genommen wird, da bei der Verwendung der je anderen einfach die for-Scheifen gedreht werden müssen. Aus diesem Grund kann die Variante nach Belieben ausgesucht werden. In diesem Projekt wurde die zweite Variante implementiert. Der Vorteil des Dictionary ist seine Verfügbarkeit. Es kann leicht auf die Informationen der Pins zugegriffen und bei Bedarf aktualisiert werden. Bei dem Start des Programms wird das Dictionary zunächst mit ‚not set'-Werten initialisiert, da davon auszugehen ist, dass die Pins nicht manuell über das Dateisystem eingeschalten wurden. Kommt nun eine GET-Anfrage kann einfach das aktuelle Dictionary mit der Webseite zurückgegeben werden ohne Weitere Berechnungen tätigen zu müssen. Bei Anfragen von der Webseite wird jedoch nicht nur das Dictionary zurückgegeben, sondern auch die Webseite selbst, also das HTML-Dokument „index.ejs". Express rendert die Daten des Dictionary in das HTML-Dokument und gibt dieses als Antwort zurück. Bei der API wird auf das Rendern verzichtet, da hier nur Daten übergeben werden sollen. Aus diesem Grund haben API und Webseite ihre eigenen voneinander unabhängigen Funktionen. Da die Webseite und ihre Funktionen vor der API entwickelt wurden, werden sie hier als erstes beschrieben.

Auf der Webseite können die Pins in ihrer Konfiguration (Richtung) eingestellt werden und dann, der Wert je nach Konfiguration per Klick gelesen oder geschrieben werden. Veränderungen an den Pins werden immer mit POST-Anfragen umgesetzt, damit nicht aus Versehen, durch falsche URL-Eingaben, ungewollte Eingaben getätigt werden. Wenn nun die Pins auf der Webseite gesteuert werden sollen, sollte eine POST-Anfrage gesendet werden, die über den Pfad je nach Aufgabe eine Funktion auslöst. So kann zum Beispiel zum Konfigurieren der Richtungen der Pins eine Anfrage an den Server geschickt werden, in dessen Pfad angegeben ist, ob die Pins lesend oder schreibend benutzt werden sollen.

- '/led/export/out'
- '/led/export/in'

Welche Pins dabei modifiziert werden sollen, kann aus der mitgelieferten Liste im Request-Body entnommen werden. Hier kann leicht über die Liste iteriert werden und für jedes Element in der Liste wird die Funktion „setDirection“ der GPIO-Klasse aufgerufen. Nachdem die Funktion für jedes Element ausgeführt und die Informationen in das Dictionary eingetragen wurden, kann die Webseite mit dem aktualisiertem Dictionary als Antwort zurückgeschickt werden. Nachdem ein paar Pins konfiguriert wurden, kann auf der Übersichtsseite per Mausklick eine POST-Anfrage getätigt werden, um je nach Konfiguration einen Pin entweder ein bzw. auszuschalten oder um abzufragen, welchen Wert dieser hat. Dafür muss bei der Anfrage im Pfad die Richtung des Pins und die Nummer des Pins angegeben werden.

- '/led/in/4'
- '/led/out/24'

Je nach Richtung wird hier bei eingehenden POST-Anfragen eine eigene Funktion aufgerufen und die Nummer des Pins, mithilfe einer „substring“-Funktion, aus dem Pfad ausgelesen. Soll der Pin gelesen werden, wird die Funktion „readValue“ aus der GPIO-Klasse mit der Pin-Nummer aufgerufen und anschließend der zurückgegebene Wert in das Dictionary geschrieben. Das aktualisierte Dictionary kann mit der Webseite an den Anfragenden zurückgegeben werden. Wenn jedoch ein Pin beschrieben werden soll, also der Pin ein- oder ausgeschaltet werden soll, wird der Wert zunächst aus dem Dictionary gelesen und der jeweils andere Wert ($0 \rightarrow 1$, $1 \rightarrow 0$) wird dann mithilfe der Funktion „writeValue“ aus der GPIO-Klasse ins Dateisystem eingetragen. Falls der Wert „not set“ ist, wird er vorher mit 0 überschrieben. Auch hier wird der neue Wert wieder in das Dictionary geschrieben und mit der Webseite zurückgegeben. Die letzte Funktion auf der Übersichtsseite soll das Ausschalten aller Pins ermöglichen. Obwohl hier keine Daten gesendet werden, wird auch diese Möglichkeit, unter Angabe des entsprechenden Pfads, über eine POST-Anfrage realisiert.

- '/unexport/'

Die aufgerufene Funktion iteriert über jedes Element des Dictionary, also über alle Pins, und übergibt die Nummer der Pins an die Funktion „close“ aus der GPIO-Klasse und sorgt somit für deren „unexport“. Das Dictionary wird anschließend reinitialisiert und mit der Webseite zurückgegeben.

Die Funktionen der API sind ähnlich aufgebaut wie die der Webseite. Da hier jedoch keine Benutzeroberfläche benutzt wird, reichen hier zwei Funktionen aus. Eine Funktion ist für das Lesen und Zurückgeben von Informationen und eine Funktion ist für das Konfigurieren und Beschreiben eines Pins. Für das Auslesen der Informationen der Pins muss eine GET-Anfrage an die API gesendet werden. Über den Pfad kann hier in einer festgelegten Syntax die Auswahl getroffen werden, von welchen Pins Inforationen zurückgegeben werden sollen.

- '/api?Pin=1&Pin=5&Pin=17'

Wird eine Auswahl angegeben, wird zunächst geprüft, ob die Eingabe korrekt. Dies bedeutet das geprüft werden muss, ob die angegebenen Pins existieren, also deren Nummer zwischen 0 und 27 liegt und ob der dazugehörige Wert korrekt ist, also entweder 0/1 oder in/out . Zudem darf auf einen lesenden Pin nicht geschrieben werden. Für die korrekten Eingaben wird die Funktion den entsprechend aktuellen Wert, in einem neu erstelltem, temporären Dictionary eintragen. Für Pins die auf lesend konfiguriert sind, wird die GPIO-Funktion „readValue“ aufgerufen. Nachdem die Anfrage abgearbeitet ist wird das neu erstelle Dictionary im JSON-Format zurückgegeben. Falls keine Angaben im Pfad gemacht wurden, wird nur über die Pins im Dictionary iteriert und für die, die lesend konfiguriert sind ebenfalls die GPIO-Funktion „readValue“ aufgerufen. Das Dictionary wird dabei aktualisiert und im JSON-Format zurückgegeben. Möchte man Zustände der Pins modifizieren, muss dies mit einer POST-Anfrage getan werden. In der POST-Anfrage an die API muss im JSON-Format angegeben werden, was gemacht werden soll. Wenn die Richtung eines Pin gesetzt werden soll, muss der Pin mit der Richtung im Dictionary angegeben sein.

```
{"direction":{"pin1":"in","pin2":"out","pin3":"not set"}}
```

Nun wird geprüft ob die angegebene Richtung mit der Richtung im Dictionary übereinstimmt und gegebenenfalls die Richtung mit der GPIO-Funktion „setDirection“ aktualisiert. Ist ein Pin auf schreibend gesetzt, kann sein Wert unter Angabe des neuen Wertes geändert werden.

```
{"value":{"pin1":1,"pin2":0,"pin3":"not set"}}
```

Auch hier wird zunächst geprüft, ob der gegebene Wert aktuell ist und gegebenenfalls mit der GPIO-Funktion „writeValue“ aktualisiert. Falsche Eingaben werden, durch passende If-Abfragen, erkannt und dem Entwickler in der Antwort mitgeteilt. So muss ein Wert genommen werden der entweder in/out oder 1/0 oder „not set“ ist. Die Auswahl begrenzt sich auf Pins zwischen 0 und 27. Nach der Bearbeitung der Anfrage wird das aktualisierte Dictionary im JSON-Format zurückgegeben

Der Programmcode des Servers wurde so geschrieben, dass er leicht veränderbar ist. So können GPIO-Nummern, durch ein angelegtes Dictionary, leicht verändert werden. Ebenfalls lässt sich hier, wie in der GPIO-Klasse, die gesamte Ausgabe im Terminal ausschalten, in dem die Variable LOGGING auf false gesetzt wird. Durch ändern der Variable PORT im Quellcode, lässt sich der Port des Servers einstellen.

4.4 Aufbau einer möglichen Anwendung mit Kommunikation über die API

Die Übersichtsseite und die beiden Seiten für die Anwendungsbeispiele werden zum Schluss Programmiert und sind somit der letzte Teil des Developer-Kits. Da die Übersichtsseite vor den Anwendungsbeispielen programmiert wurde und wesentlich umfangreicher als diese ist, wird sie als erstes beschrieben.

Die Übersichtsseite ist so aufgebaut, dass es eine Auswahl der 26 Pins gibt, in der die Aufgabe der Pins konfiguriert werden kann. Wenn die Pins auf lesend oder schreiben (in/out) konfiguriert sind, blendet für jeden konfigurierten Pin ein eigenes Eingabefeld im Browser auf. In diesen Eingabefeldern gibt es ein Button, welcher eine POST-Anfrage unter Angabe der GPIO-Nummer und der konfigurierten Richtung an den Server schickt. Die Auswahl ist so realisiert, dass es für jeden Pin ein Feld gibt, auf das geklickt werden kann, um dieses zu markieren. Beim Markieren der Felder werden die HTML-IDs der markierten Felder in eine Liste eingefügt. Die markierten Felder können anschließend über zwei weitere Buttons einer Richtung zugewiesen werden. Beim Klicken auf eines der beiden Buttons wird die Liste mit den IDs der LEDs unter Angabe der Richtung an den Server geschickt, um diese zu exportieren. Dabei wird der Pfad '/led/export/out' oder '/led/export/in' aufgerufen. Die Liste wird anschließend geleert und die Eingabefenster der Pins entsprechend ihrer Richtung eingeblendet.

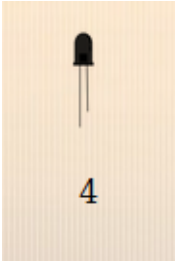


Abbildung 5: Bild eines Auswahl-Elements

```
<button onclick="make_direction('out')">'out'</button>
```

Die eingeblendeten Eingabefelder bestehen aus einem Textfeld in dem der Status des Pins steht, einem Bild einer LED in der Farbe des entsprechenden Zustandes (Grün für 1, Rot für 0, Grau für „not set“) und einem Button. Bei Klick auf einen der Button wird eine POST-Anfrage ausgelöst, die den gewählten Pin ein/aus schaltet oder den Wert des Pins aktualisiert. Dabei werden die Daten aktualisiert und die Seite neu geladen.

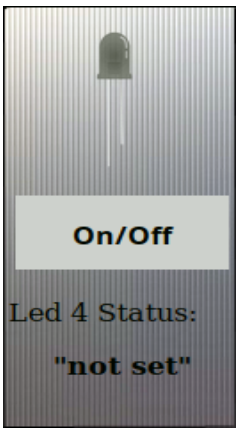


Abbildung 6: Bild eines Schalt-Elements

```
<form action="/led/out/4" method="post">
      <button type="submit">On/Off</button>
</form>

<form action="/led/in/5" method="post"/>
      <button type="submit">Check</button>
</form>
```

Bei jedem Aufruf der Seite sind alle Eingabefenster erst einmal ausgeblendet. Mithilfe des mitgesendeten Dictionary kann eine Funktion die Eingabefenster nun einblenden, falls diese

mit einer Richtung konfiguriert sind. Dabei wird das Bild der grauen LED je nach Status in eine grüne oder rote LED gewechselt und der Status-Text aktualisiert. Da alle Buttons, Bilder, Texte und Rahmen einer HTML-Klasse zugeordnet sind und eine ID haben, kann das Aussehen der Webseite leicht verändert und angepasst werden, dabei bleibt die HTML-Struktur aber erhalten. Die Konfigurationen für das Aussehen der Seite (CSS) und die Skripte (Javascript) wurde in externe Dateien ausgelagert, damit eine bessere Übersicht herrscht und der Code sinnvoller strukturiert werden kann. Für einen leichteren Übergang zu den Anwendungsbeispielen wurden am unteren Ende der Seite ein Menü mit Buttons eingefügt, über die man, ebenfalls durch klicken, zu den Seiten der Anwendungsbeispiele gelangt. Hier werden GET-Anfragen an den Server geschickt, der mit den HTML-Dokumenten der Beispiele antwortet.

Die Webseiten sind stilistisch und funktional minimal gehalten, damit die beispielhaften Anwendungen leicht von Entwicklern nachvollzogen werden können. Die Funktionen der Anwendungsbeispiele arbeiten über HTTP-Anfragen nur mit der API des Servers. Im ersten Beispiel ist das Bild einer Laterne und eines Schalters zu sehen. In der rechten oberen Ecke können die beiden Objekte über zwei Menüs den GPIO-Pins zugewiesen werden. Die Laterne stellt dabei den lesenden Pin dar. Sie ist mithilfe der GET-Anfrage implementiert und fragt bei einem Mausklick auf das Bild, alle halbe Sekunde, nach dem aktuellen Status des vorher definierten Pins. Die Antwort des Servers ist das Dictionary im JSON-Format. Der Inhalt des Dictionary kann anschließend mit einer If-Abfrage geprüft werden. Ist der Status 0 wird die Laterne rot. Ist der Status 1 wird die Laterne grün. Bei dem Status „not set“ bleibt die Laterne weiß und das Abfragen wird beendet. Der Schalter der auf der Seite zu sehen ist, stellt einen schreibenden Pin dar. Wenn auf ihn geklickt wird, löst das eine Funktion aus, die zunächst den aktuellen Status der Pins über die API abfragt und dann den entgegengesetzten Wert über eine POST-Anfrage im JSON-Format auf den Pin schreibt. Die HTTP-Anfragen können in Javascript auf verschiedene Weise umgesetzt werden. So kann das Modul jQuery bei den Anfragen an den Server sehr hilfreich sein. Die beiden vorgestellten Methoden stellen die Funktionen der API beispielhaft dar und geben den Entwickler eine Hilfestellung, wie eigene Projekte umgesetzt werden können. Das zweite Beispiel-Dokument ist ein mehr praxisbezogenes Beispiel. Auf der Seite ist ein Wecker zu sehen, der nach dem Erreichen der angegebenen Uhrzeit eine beliebige Funktion auslöst. Da die Umsetzung des Weckers keine Relevanz für die Entwicklung des Developer-Kits hatte, wurde der Quellcode des Weckers aus Zeitgründen leicht modifiziert, aus einem Beispiel, aus dem Internet, übernommen und nicht selbst entwickelt[23]. Der Wecker schickt, nach Erreichen der angegebenen Uhrzeit, eine POST-Anfrage an den Server. Dabei wandelt er den angegeben String im Eingabefeld in ein JSON-formatierten String und sendet diesen an die API des Servers. Auf diesem Wege ist es möglich einen bestimmten Pin um eine bestimmte Uhrzeit ein und auszuschalten. Beide Dokumente importieren ihre CSS- und JavaScript-Dateien aus dem Ordner „/public“. Über die CSS-Dateien können die Hintergründe und Designs leicht nach Belieben geändert werden. Die Bilder auf den Webseiten sind open source und frei verwendbar. Die Farben der LEDs konnten leicht mit Gimp bearbeitet werden.

5. Ergebnis

Nachdem das Developer-Kit soweit fertig entwickelt und auf GitHub der neuste Stand der Software hochgeladen ist, kann das Kit dort von Entwicklern zur Benutzung heruntergeladen werden. Nach einer kurzen Installation der nötigen Bibliotheken nach der Anleitung in der „README.md“-Datei, kann das Kit schon verwendet werden. In diesem Kapitel wird beschrieben was an dem Developer-Kit gut funktioniert und was nicht gut funktioniert. Es wird darauf eingegangen ob die Anforderungen erfüllt wurden und beschrieben was noch verbessert werden kann.

5.1 Wie gut funktioniert das Kit in der Anwendung?

Der Code des Developer-Kit ist gut strukturiert und zum Verständnis, sauber in einzelne Abschnitte aufgeteilt. So kann der Entwickler mithilfe der Kommentare den Code nachvollziehen und falls er will, Änderungen vornehmen. Nach dem Starten des Servers wird dem Entwickler im Terminal das Starten des Programms und der Port mitgeteilt. Nun kann er entweder im Browser unter der IP-Adresse des Raspberry Pi und dem Port die Übersichtsseite aufrufen. Bei der Benutzung der Webseiten treten soweit keine Probleme auf. So können auf der Übersichtsseite die Pins problemlos konfiguriert und geschaltet werden. Die Beispielanwendungen geben einen guten Einblick in die Funktionen der API. So kann der Entwickler, im Quellcode der Webseite, sehen, auf welche Weise die API angesprochen werden muss. Die Syntax wird dem Anwender dabei auch im zweiten Anwendungsbeispiel dargestellt. Auf diese Weise ist es möglich, dass der Entwickler sich auf spielende Weise mit den GPIO-Pins auseinandersetzen kann. Durch die Erklärungen auf der Übersichtsseite wird dem Entwickler bei seinem ersten Kontakt mit dem Kit weitergeholfen. Der Entwickler kann sich nun nach den ersten Erfahrungen mit dem Kit, selbst ein eigenes Programm entwickeln das über die API die GPIO-Pins schaltet. Bei der Entwicklung der Kommunikation mit der API im JSON-Format, können immer mal wieder Programmierfehler aufkommen. Diese sind nervig und das Suchen der Fehler beansprucht Zeit. Die Rückmeldungen der API bei falschen Eingaben kann dafür Abhilfe schaffen. Der Server meldet, wenn die Konfigurationsparameter falsch sind (in/out/1/0) oder die GPIO-Nummer (1,2,3,…,26) ungültig ist. Der Server lief während der Entwicklung, teilweise stundenlang, ohne das Probleme aufgetreten sind und verbrauchte dabei wenig Systemleistung. Die einschaltbaren Ausgaben im Terminal bei HTTP-Anfragen sind zusätzlich sehr praktisch, da sie Angeben, von welchem Gerät (IP-Adresse) welche Anfrage kam und was das Programm macht. Das Anwendungsbeispiel mit der Laterne und dem Schalter verdeutlicht, dass durch eine entsprechende Implementierung die aktuellen Informationen eines Pins abgefragt und verwendet werden können, sodass es möglich ist ein live-Status des Pins anzuzeigen. Die Wiederholungsrate des Pin-Status von 0,5 Sekunden/Anfrage kann dabei noch weiter nach unten gesetzt werden. So sind auch komplexere Verwendungen der Pins möglich.

5.2 Wurden die Anforderungen erreicht?

Nach den Anforderungen, die im Kapitel 3 aufgestellt wurden, sollte das Kit übersichtlich und einladend gestaltet werden. Entwickler sollten durch lesen der Quellcodes und anschauen der Webseite die Bedienung der API schnell verstehen, um möglichst bald eigene Projekte umsetzen zu können, die auf diese Software aufbauen. Der Quellcode des Softwarepakets ist in einzelne Dateien aufgeteilt und die CSS- und JS-Scripte wurden der Übersicht halber ebenfalls in externe Dateien ausgelagert. Die Aktionen der Funktionen sind im Quellcode in den Kommentaren beschrieben. Die Webseite ist durch einfache Gestaltung und den Inhalt beschreibende Texte einladend gestaltet und intuitiv bedienbar. Nach kurzer Zeit auf der Übersichtswebseite kann sich der Entwickler die Anwendungsbeispiele ansehen. Über eine der Webseiten kann der Entwickler erste API-Anfragen stellen und sich die Quellcodes zum Verständnis anschauen. Aus diesen Gründen ist die Software durchaus übersichtlich strukturiert und einladend gestaltet.

Weitere Anforderungen waren, dass das Programm verlässlich ist, immer die richtigen Angaben zurück gibt und bei Falscheingaben nicht abstürzt. Die Webseite ist nur durch Mausklicks bedienbar und kann so nur Anfragen an den Server senden, auf die er eingestellt ist, sodass dort keine Fehler zu erwarten sind. Auch die Anwendungsseiten sind so programmiert, dass sie fehlerfrei mit der API kommunizieren. Aus diesem Grund können sie als Vorlagen für eigene Projekte angesehen werden. Die API ist so programmiert, dass sie bei falschen Pin eingaben, falschen Richtungswerten, oder ungültigen Nummerierungen eine entsprechende Meldung in der JSON-Antwort zurückgibt. Die Werte während der Test, die die API zurückgegeben hat, waren immer korrekt und auch das Schalten der GPIO-Pins funktionierte, außer in Ausnahmefällen, immer. Diese Ausnahmefälle entstehen durch einen Fehler im Programm, der entsteht, wenn innerhalb einer Anfrage die Richtung eines Pins auf out gesetzt wird und gleichzeitig den Wert dieses Pins beschrieben wird. Dieser Fehler konnte während der Programmierung des Kits in der Zeit nicht mehr behoben werden. Beim gleichzeitigen konfigurieren und beschreiben kommt es durch Race-Kondition in der Bearbeitungen der for-Schleifen zu Fehlern beim Schalten der GPIO-Pins. Dieser Fehler kann umgangen werden, indem direction und value nicht gleichzeitig in einer HTTP-Anfrage konfiguriert werden. Für ein unexport der Pins kann gesorgt werden, durch setzen der beiden Werte ‚direction' und ‚value', auf ‚not set'. Solange auf dieses Vorgehen beim Konfigurieren und Schalten der Pins geachtet wird, sind keine Fehler in der Verwendung der API zu erwarten.

Eine letzte Anforderung war, dass die Klasse der GPIO-Pins auch von anderen Programmen leicht verwendet werden kann und austauschbar ist. Da die Funktionen der Klasse in eine externe Javascript-Datei ausgelagert wurde, die für andere Programme importierbar ist, kann sie leicht von diesen verwendet werden. Somit sind alle zuvor aufgestellten Anforderungen an das Developer-Kit erreicht. Die GPIO-Klasse und der Server können noch weiter optimiert werden und weitere Funktionen oder effizientere Zugriffsmethoden können der GPIO-Klasse zugefügt werden. Nach beliebig kann diese auch ausgetauscht werden.

5.3 Was kann verbessert werden?

In einem Projekt in dem gearbeitet oder programmiert wird, passieren auch Fehler. So lief auch in diesem Projekt nicht alles gut und ein paar Dinge sollten noch verbessert werden.

Manchmal kommt es in der GPIO-Klasse zu den Problemen eines Laufzeitfehlers, die beim Exportieren und anschließendem Konfigurieren der Richtung des Pins oder dem Konfigurieren der Richtung und dem anschließendem ein/ausschalten des Pins entstehen. Dadurch kommt es auch beim Ausführen der Aktionen im Server zu Zugriffsfehlern, sodass die Aktionen in diesen Fällen nicht erfolgreich sind. Aus diesen Gründen wurden an benötigten Stellen kurze Pausen mit setTimeout() eingelegt, sodass die Funktion kurz wartet und somit genug Zeit bleibt, um die andere Funktion währenddessen zu beenden. Diese Fehlerüberbrückung sollte an verschiedenen Stellen überarbeitet werden.

Im Quellcode der GPIO-Klasse ist der Pfad, der zu den Pins zeigt, in zweifacher Ausgabe angegeben, da es je nach Version des Kernels zu unterschiedlichen Systempfaden im Dateisystem kommen kann. Der entsprechend richtige Pfad kann aktuell nach Bedarf manuell ausgewählt werden, indem der je andere Wert auskommentiert wird. Diese Auswahl kann mit einer Abfrage überarbeitet werden, die automatisiert die Kernelversion abfragt und den Wert gegebenenfalls ändert, sodass der Entwickler dies nicht selbst anpassen muss. Auch werden die Eingaben für die GPIO-Funktionen aktuell noch von den Funktionen des Server überprüft und nur richtige Werte weitergegeben. Diese Eingabeüberprüfung der Werte könnte noch in der GPIO-Klasse implementiert werden.

Der Code ist wegen der ganzen if-Abfragen zur Kontrolle der JSON-Eingaben über die API an manchen Stellen durchaus redundant. Diese redundanten If-Abfragen könnte man durch optimieren des Codes beseitigen und den Code durch besser gewählte If-Abfragen noch sinnvoller strukturieren. Auch der Quellcode des Servers kann durch Beseitigung redundanter Kontroll-Abfragen der API-Eingaben noch besser verknüpft werden, sodass der dieser Code kleiner und dadurch übersichtlicher wird. Auch der Code für die Übersichts-Webseite ist zwar trotz seiner Länge übersichtlich, hat jedoch einige Redundanzen. Hier wurden für alle 26 Pins jeweils die gleichen Zeilen Code geschrieben, die sich nur in den Zahlen unterscheiden. Diese Codezeilen können mit einer passenden Schleife, in Javascript, erstellt werden, um somit einen Großteil der Codezeilen zu sparen.

Bei Bedarf könnte die zugrundeliegende Datenstruktur, das Dictionary, weiter verkleinert werden indem die Strings „value“ und „direction“ mit einem entsprechendem Kürzel ausgetauscht werden und die redundanten Strings der Pins nur auf ihre ID gekürzt werden (z.B. „pin25“ → „25“, „direction“ → „dir“, „value“ → „val“). Damit würde es zu einer Einsparung des Datenverkehrs kommen, jedoch wäre der Quellcode dann auch weniger intuitiv verständlich. Aktuell wird auf der Übersichtsseite die Informationen der Pins nach

einem Mausklick einzeln abgefragt bzw. aktualisiert. Diese Aktualisierung könnte noch automatisiert werden, sodass der Server zum Beispiel alle 0,5 Sekunden alle Pins aktualisiert und die aktualisierten Informationen der Pins bei der nächsten Anfrage ohne weitere Rechnungen zurückgibt.

Auf der Webseite ist für den Entwickler ein Bild mit einem Layout der GPIO-Pinbelegung hinterlegt. Durch dieses Bild kann der Entwickler bereits bei der Verwendung der Übersichtsseite das Layout betrachten und schauen wo der entsprechende Pin ist. Dort ließe sich noch eine Vergrößerung dieses Bildes, nach einem Mausklick darauf, implementieren, damit der Entwickler das Bild in größerer Version betrachten kann.

6. Fazit

Abschließend wird in diesem Kapitel ein Ausblick gegeben, was als nächstes in der Entwicklung des Kits gemacht werden kann, wie in Zukunft damit gearbeitet werden kann und welche Vorteile zu erwarten sind. Die Arbeit wird kurz zusammengefasst, sodass ein abschließendes Fazit über den Wert der Arbeit gegeben werden kann.

6.1 Ausblick

Die Intention dieses Softwareprojektes war wie anfangs beschrieben, das wachsende Interesse der Verbraucher, an Geräten für eine Heimautomatisierung. Da jedoch nicht nur das Interesse, sondern auch das Misstrauen gegenüber der Firmen, in Bezug auf den Umgang mit personenbezogenen Daten wächst, sollte ein Softwarepaket entworfen werden, dass die Entwicklung eines eigenen „Internet of Things“-Gerät, vereinfacht. Das entworfene Kit ist nach wenigen Schritten der Installation betriebsbereit und der Entwickler kann sich mithilfe des Quellcodes der Webseiten ein eigenes Programm nach seinen Vorstellungen bauen. Durch die erstellte Übersichtsseite ist es dem Entwickler möglich, selbstentworfene Schaltungen bereits ohne eigene Software über Mausklicks zu steuern und zu testen, weshalb das Kit dort schon mal eine gute Unterstützung für einen schnellen Start in ein eigenes Projekt bietet.

Durch die Entwicklung des zweiten Anwendungsbeispiels kann der Entwickler nach dem Start der Software die Pins seines Projektes mit der vorgefertigten POST-Funktion steuern. Auf diesem Weg kann er erste automatisierte Befehle an seine Schaltung senden und diese Befehle bald darauf selbst in seine eigenen Webseiten einbinden. Bei der Entwicklung mit dem Kit bekommt der Entwickler sowohl auf der vorgefertigten Übersichtsseite eine gute Kontrollinstanz für die Pins und deren Steuerungen, als auch die Information in kurzer und langer Version über die API.

Nachdem die im vorigen Kapitel beschrieben Korrekturmöglichkeiten in kommenden Updates der Software umgesetzt werden, wird das Programm eine gute Softwaregrundlage sein, die sowohl stabil läuft, verlässlich, als auch effizient ist. Auch die Beispielanwendungen können sowohl in der Stückzahl als auch in ihrer Komplexität ausgeweitet werden, damit Anwender des Kits Beispiele von größerem Umfang anschauen können.

Die Möglichkeiten der Anwendungen, welche über die GPIO-Pins laufen sind dabei enorm vielfältig. Die Pins können dabei beliebig mit verschiedenen Sensoren und Schaltkreisen verbunden werden, wobei hier lediglich auf die physikalischen Gegebenheiten des elektrischen Stroms zu achten ist. Es bleibt abzuwarten, wie gut das Kit wirklich bei der Entwickler-Gemeinschaft ankommt und wie sehr die Entwickler, in ihren Projekten, von dem Werkzeug im täglichen Gebrauch profitieren werden können. Jedoch hat es, durch seine hohe Anpassbarkeit und der einfachen Struktur den Vorteil, von Entwicklern, nach Bedarf angepasst werden zu können. Es bleibt zu hoffen, dass das Developer-Kit einigen Menschen mit Misstrauen doch mit dem Thema Smart-Home in Verbindung bringen kann und diese sich ihr Leben mit eigenen Projekten verschönern und vereinfachen werden.

6.2 Zusammenfassung

Es hat sich gezeigt, dass sich durch grundlegende Konzepte der Webentwicklung und der Verknüpfung dieser Konzepte zu einem Gesamtsystem Schritte in Richtung eigener Heimautomatisierung kostengünstig umsetzen lassen. Es ist nicht nötig sich für teures Geld, Geräte zu kaufen, die das Eigenheim automatisieren, aber denen man nicht vertraut. Wenn Entwickler sich auf Grundlage der Software eigene Projekte umsetzen, oder die Software erweitern, haben sie in dem Kit eine Alternative die durchaus transparent mit ihren Daten umgeht, sofern hier überhaupt persönliche Daten genutzt werden. Das Kit ist für Entwickler auf deren Bedürfnisse anpassbar und extrem transparent, sodass der Entwickler immer die Möglichkeit hat, zu erfahren was die Geräte machen. Zusätzlich hat er hier die Kontrolle über seine Daten. Es wurde mit vergleichbar wenig Aufwand gezeigt. dass es sich lohnen kann, Dinge nicht immer nur zu kaufen, sondern auch mal etwas selbst zu versuchen zu bauen. Mithilfe der in den Grundlagen vorgestellten Werkzeuge kann das Developer-Kit leicht nachvollzogen werden, sodass die Funktionen in anderen Programmen übernommen oder überarbeitet werden können.

Die Entwicklung des Developer-Kits wurde nach dem „Bottom-Up“-Konzept erarbeitet, was sich nicht als Fehler erwiesen hat. Jedoch war es weniger ein sukzessiver Entwicklungs-Prozess, sondern eher eine agile Programmierung, wie man sie von „scrum“-Projekten kennt. Dies liegt an den aufkommenden Fehlern, die bei der Entwicklung der nächst höheren Ebene auffallen. Da die Anfragen der Webseiten und der Server aufeinander abgestimmt werden müssen, ist die Programmierung der Software teilweise eher wie ein Ping-Pong-Spiel. Dennoch hat die Aufteilung der Software, in drei eigene Codefragmente, für einen saubereren Code gesorgt, da dieser übersichtlicher und ausgedünnter ist. Weitere Verbesserungen der Übersichtlichkeit konnten durch Anwenden der CSS-Klassen umgesetzt werden, da dadurch viele Angaben in den Style-Attributen der HTML-Elemente weggefallen sind. Die Anforderungen welche im 3.Kapitel an das Kit gestellt wurden, sind überwiegend umgesetzt, wobei es bei Anfragen an die API noch zu Fehlern kommen kann. Dieser Fehler tritt auf, wenn über eine API-Anfrage ein beliebiger Pin in seiner Richtung konfiguriert wird und direkt ein oder ausgeschaltet wird. Durch Aufteilen dieser Schritte in 2 Anfragen, kann dieses Problem jedoch leicht behoben werden. Da es bei der Heimautomatisierung meinst nur nötig ist etwas ein oder auszuschalten oder einen Motor in Gang zu setzen, reichen die Funktionen des Developer-Kits für die meistens Anwendungen aus. Bei komplexeren Anforderungen kann die GPIO-Klasse und die API mit weiteren Funktionen erweitert werden, sodass zum Beispiel die Schnittstellen I2C und SPI verwendet werden können.

Sowohl die Übersichtsseite, als auch die Anwendungsseiten, funktionieren wie erwartet und lassen wenig Platz für falsche Eingaben. Der Anwender bekommt durch entsprechende Beschriftungen klare Anweisungen, wie die Seiten zu verwenden sind und kann sich so mit der Funktionsweise der Software vertraut machen. Geht er auf das Wecker-Beispiel, kann er dort die Syntax der JSON-Anfragen studieren. Durch den Quellcode bzw. dem JavaScript-Code

dieser Seite kann er dann schnell sehen, wie er eine GET- oder POST-Anfrage an die API aufbaut. Auf diesem Wegen ist es möglich, über ein Netzwerk, die programmierbaren Ein- und Ausgabe-Pins für allgemeine Zwecke zu nutzen.

Da das Software-Kit ohne Vorkenntnisse in Webprogrammierung, also HTML, CSS und JavaScript mit vergleichsweise geringem Aufwand programmiert wurde, kann davon ausgegangen werden, dass auch Programmieranfänger den Quellcode, mit etwas Fleiß schnell nachvollziehen können.

Verzeichnisse

1 Literaturverzeichnis

Literaturverzeichnis

1: Americans Ready for the Smart Home; 2015, August; https://blog.coldwellbanker.com/americans-ready-for-the-smart-home/
2: 10 Mio. Echo-Geräte; 2018, Januar; https://www.housecontrollers.de/allgemein/verkaufsschlager-alexa-amazon-verkauft-2017-mehrere-zehn-millionen-echo-geraete/
3: wachsende Verkaufszahlen; 2016, November; https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#53baddf2292d
4a: Misstrauen-Studie a; 2019, Sep; https://www.gdata.de/news/2018/09/31109-g-data-security-umfrage-deutsche-misstrauen-smart-home-geraten/
4b: Misstrauen-Studie; 2019, Sep; https://www.telecom-handel.de/consumer-communications/smart-home/steigendes-misstrauen-datennutzung-vernetzte-geraete-1421085.htm
5: HTML INTRO; 2019, Sep.; https://www.w3schools.com/html/html_intro.asp
6: Aufbau eines HTML Dokuments; 2019, ; https://speakerdeck.com/eliashaeussler/einstieg-in-die-frontend-entwicklung-1-html-grundlagen?slide=8
7: CSS Grundlagen; 2018, Dez; https://speakerdeck.com/eliashaeussler/einstieg-in-die-frontend-webentwicklung-2-css-grundlagen?slide=7
8: Was ist JavaScript? Eine verständliche Erklärung ; 2016, Dez; https://www.giga.de/extra/javascript/tipps/was-ist-javascript-eine-verstaendliche-erklaerung/
9: Javascript Wikipedia DE; 2019, Sep; https://de.wikipedia.org/wiki/JavaScript
10: Callback-Grundlagen; 2017, Jun; https://codeburst.io/javascript-what-the-heck-is-a-callback-aba4da2deced
11: JavaScript; 2019, Sep; https://www.w3schools.com/Js/
12: Frontend und Backend; 2019, Feb; https://praxistipps.chip.de/backend-und-frontend-was-ist-das-einfach-erklaert_41384
13: Express-Grundlagen; 2019, Sep; https://en.wikipedia.org/wiki/Express.js
15: FS-Grunlagen; 2019, Sep; https://nodejs.org/api/fs.html
16: HTTP_Grundlagen; 2019, Sep; https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#Message_format
17: GET-POST-Grundlagen; 2019, Sep; https://www.w3schools.com/tags/ref_httpmethods.asp
18: JSON-Grundlagen; 2019, Sep; https://www.w3schools.com/js/js_json_intro.asp
19: Annual Review 2018; , ; https://static.raspberrypi.org/files/about/RaspberryPiFoundationReview2018.pdf
19a: RASPBERRYPI-INDOS; 2019, Okt; https://en.wikipedia.org/wiki/Raspberry_Pi
19b: GPIO-INFOS; 2019, Okt; https://www.raspberrypi.org/documentation/usage/gpio/
20: RASBIAN-Infos; 2019, Sep; https://www.heise.de/download/product/raspbian-91329
21: Putty-Infos; 2019, Sep; https://www.heise.de/download/product/putty-7016
22: Github-Tutorial: Diese Basics müssen Sie können; 2016, Nov; https://praxistipps.chip.de/github-tutorial-diese-basics-muessen-sie-koennen_50784
23: JAVASCRIPT-WECKER ; 2019 , Okt; http://javascriptkit.com/script/script2/alarm.shtml

2 Abbildungsverzeichnis

Abbildungsverzeichnis

Abbildung 1: Layout des Konzepts....22
Abbildung 2: Layout des Konzepts – 1. Ebene markiert....23
Abbildung 3: Layout des Konzepts – 2. Ebene markiert....24
Abbildung 4: Layout des Konzepts – 3. Ebene markiert....26
Abbildung 5: Bild eines Auswahlelements....36
Abbildung 6: Bild eines Schaltelements....36

3 Quellcode und Werkzeuge

https://github.com/chalupka95/Bachelorarbeit-GPIO
Github, Raspberry Pi, PuTTY, Gimp, LEDs