Fachbereich 2
Informatik und Ingenieurwissenschaften
Studiengang Informatik

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science in Informatik

Ansteuerung der GPIO-Anschlüsse bei Einplatinencomputern via Web-Services

Vorgelegt von: Tobias Orth
Matrikelnummer: 951311

Referent: Prof. Dr. Christian Baun
Korreferent: Prof. Dr. Thomas Gabel

Eingereicht am: 10.03.2016

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt habe. Alle Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter der Angabe der Quellen kenntlich gemacht.

Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Frankfurt am Main, den 10.03.2016

Tobias Orth

Danksagung

Hiermit möchte ich mich bei allen Menschen bedanken, die mich während meines kompletten Studiums unterstützt haben.

Zunächst möchte ich mich bei Herrn Prof. Dr. Christian Baun für die Bereitstellung der technischen Mittel und der Betreuung bedanken. Herrn Prof. Dr. Thomas Gabel danke ich für die Bereitschaft und der Übernahme der Zweitkorrektur.

Besonders bedanken möchte ich mich bei meiner Familie, meiner Freundin und meinen Freunden für den Rückhalt und der Motivation während der Studienzeit und besonders während der Bachelorarbeit.

Herrn Wolfgang Dietrich und Herrn Jonas Leinweber danke ich für die korrekturlesung

Kurzfassung

Die vorliegende Bachelorarbeit befasst sich mit der Ansteuerung der GPIO-Anschlüsse des Einplatinencomputers Raspberry Pi über einen RESTful-Webservice der mithilfe einer Webseite konsumiert wird. Ziel der Arbeit ist es, über den Webservice das Steuern von einem oder mehreren GPIO-Anschlüssen zu realisieren. Der Webservice wird in Java und Python dargestellt und auf seine Flexibilität und Eigenschaften untersucht. Das Ansteuern der GPIO-Anschlüsse wird mithilfe von Ansteuerungsbibliotheken ermöglicht. Mithilfe von LEDs, die über ein Steckbrett direkt mit den GPIO-Anschlüssen des Raspberry Pi verbunden sind, werden die Zustände der Anschlüsse dargestellt. Das Ergebnis dieser Untersuchung zeigt, dass eine Steuerung der GPIO-Anschlüsse über einen RESTful –Service realisiert werden kann

Abstract

This bachelor thesis is about controling the GPIO-Pins of a Raspberry Pi over a RESTful-Webservice. The webservice is consumed by a html webpage which represeants the data and allows the user to interact with the webservice. The state of the GPIO-Pins is represented by leds wich are conected to a GPIO-Pin. Objective is, to realize a complete control over the GPIO-Ports over the webservice. The webservice is implemented in java and phyton and will be analyzed about its flexibility and usage. At the end you will see that there are much more possibilities beyond flashing single leds, such as controling motors or a intilligent home.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

GPIO:	General Purpose Input/Output
REST:	Representational State Transfer
SOAP:	Simple Object Access Protocol
LAN:	Local Area Network
WAN :	Wide Area Network
LED:	Light-Emitting Diode
API:	Application Programming Interface
URI:	Uniform Resource Identifier
URL:	Uniform Resource Locator
XML:	Extensible Markup Language
JSON:	JavaScript Object Notation
HTTP:	Hypertext Transfer Protocol
SSH:	Secure Shell
HTML:	Hypertext Markup Language
HATEOS:	Hypermedia as the Engine of Application State

1. Einleitung

Einplatinencomputer wurden das erste Mal 1970, mit der zunehmenden Verbreitung von Mikroprozessoren, auf den Markt gebracht. Damals wie heute decken sie den Bedarf an preiswerten Entwicklungssystemen ab. In der Industrie werden diese vorwiegend für Mess-, Steuer- oder Regelungstechniken eingesetzt. Einplatinencomputer oder engl. „Single Board Computer“ sind Scheckkarten große Computersysteme die alle nötigen Komponenten auf einer Leiterplatine zusammenfassen. [1][3] Heutige Einplatinencomputer zeichnen sich vor allem durch ihren niedrigen Preis, ihrer hohen Flexibilität und die vielen Einsatzmöglichkeiten aus. Im Privatgebrauch werden sie hauptsächlich als Spielkonsole, Steuereinheit oder als Minirechner eingesetzt. [2]

Der Bekannteste und meistverkaufte Einplatinencomputer ist der Raspberry PI (über 700 Millionen verkaufte Geräte, Stand Oktober 2015). [4] Der Raspberry Pi wird von der britischen Raspberry Pi Foundation entwickelt. Das Ziel ist die Weiterbildung von Kindern und Erwachsenen in Computerwissenschaften zu fördern. [3] Das in dieser Arbeit eingesetzte Modell Raspberry Pi B+ ist bereits für einen Preis von 25,99€[1] erhältlich. Durch seine vielseitigen Einsatzmöglichkeiten und den niedrigen Preis ist er besonders für Entwickler und Einsteiger interessant. So kann der Raspberry Pi als Mediacenter, Webserver, Spielkonsole oder als Steuereinheit eingesetzt werden.

Als Steuereinheit verwaltet der Raspberry Pi Haushaltsgeräte oder Sensoren an einer zentralen Stelle. So kann z.B. das Licht ein- und ausgeschaltet oder das Öffnen von Türen realisiert werden. Diese Schaltungen werden über die GPIO-Anschlüsse des Raspberry Pi ermöglicht. [5][2] Die GPIO-Pins bieten eine Schnittstelle für die Ansteuerung diverser Hardware und ermöglichen die Realisierung komplexer Schaltungen. Nebst den oben genannten Beispielen lassen sich über die GPIO-Pins beispielsweise Textausgaben auf LCD-Modulen, Temperaturfühler oder komplexe Mechaniken programmieren. [5] Diese werden meist über Apps auf Tablets oder Smartphones gesteuert und verwaltet.

[1] Quelle: https://www.alternate.de/Raspberry-Pi-Foundation/Raspberry-Pi-B+-Mainboard/html/product/1149754?event=search

Thema dieser Arbeit ist das Ansteuern und Schalten der GPIO-Anschlüsse mittels eines Webservices. Webservices sind Softwareanwendungen die über ein Netzwerk (LAN/WAN) Dienste bereitstellen. Es gibt verschiedene Möglichkeiten Webservices zu implementieren, die bekanntesten sind SOAP (Simple Object Access Protocol) und XML-RPC (Extensible Markup Language Remote Procedure Call). Nebst diesen bekannten Lösungen gibt es die Representational-State-Transfer-Architektur, auch kurz REST genannt. [8] REST ist besonders interessant, da versucht wird das Interface auf eine Menge definierter Standard-Operationen (GET, POST, PUT, DELETE) zu beschränken. Die REST-Architektur entwickelte sich aus dem 1994 von Roy Fielding entworfene „HTTP Object Model“. Fielding entwickelte seine Idee weiter bis er im Rahmen seiner Dissertation im Jahre 2000 den REST-Architekturstil veröffentlichte. [6][7]

Der Webservice wird dafür in Java und in Python bereitgestellt und auf ihre Eigenschaften verglichen. Java ist eine plattformunabhängige Programmiersprache und kommt heutzutage in einer Vielzahl von Geräten zum Einsatz. Für den Raspberry Pi ist die Sprache Python besonders beliebt. Viele Projekte die in der Community präsentiert werden, sind in Python geschrieben.

Grundlage für das Ansteuern der GPIO-Anschlüsse sowie das Bilden eines Webservice ist die Kenntnis über die Funktionalität des Raspberry Pi sowie die Funktionalität von RESTful Webservices. Aus diesem Grund beschäftigt sich diese wissenschaftliche Arbeit mit diesen Techniken.

1.1 Aufbau der Arbeit

Diese Bachelorarbeit ist in vier Kapitel unterteilt. Nachdem das erste Kapitel die Einleitung beinhaltet und eine Einführung in die Thematik vermittelt hat, werden im zweiten Kapitel die Grundlagen erläutert. Es werden der Raspberry PI, Webservices sowie die eingesetzten Techniken und die benötigten Bibliotheken und Module vorgestellt.

Der Fokus der Arbeit richtet sich auf das dritte Kapitel. In diesem Kapitel werden die genaue Realisierung des Webservices sowie die Ansteuerung der GPIO-Anschlüsse betrachtet. Hierfür wird zuerst die verwendete Hardware aufgezeigt und auf die Implementierung eingegangen. Anschließend wird die Funktionalität der GPIO-Pins und des Webservices näher beschrieben.

Das vierte Kapitel bildet den inhaltlichen Abschluss dieser Arbeit und enthält eine Zusammenfassung sowie eine Bewertung der entwickelten Lösungen. Abschließend wird eine Aussicht mit weiterführenden Ideen für dieses Projekt gestellt.

2. Grundlagen

In diesem Kapitel werden die Grundlagen zum Thema dieser Arbeit erörtert. Es werden der Einplatinencomputer Raspberry Pi sowie Webservices vorgestellt. Lösungen für den Webservice und das ansteuern der Pins werden in Java und Python untersucht. Es werden Begrifflichkeiten definiert die für das Verständnis der Thematik dieser Arbeit unabdingbar sind.

2.1 Der Raspberry Pi Einplatinencomputer

Der Raspberry Pi, der von der Raspberry Pi Foundation entwickelt wurde, ist seit Anfang 2012 auf dem Markt erhältlich. Die Raspberry Pi Foundation verfolgt das Ziel, kostengünstige Computer bereitzustellen um Kinder und Erwachsene im Bereich Computerwissenschaften zu fördern und das Aneignen von Programmiersprachen und Hardwarekenntnissen zu erleichtern. [3][19] Trotz der geringen Größe ist der Raspberry Pi ein vollwertiger Computer der durch seine zahlreichen Anschlüsse viele Erweiterungsmöglichkeiten bietet. Die Beliebtheit des Raspberry Pi spiegelt sich auch durch die hohen Verkaufszahlen wieder, so wurden 700 Millionen Stück (Stand Oktober 2015) verkauft. [4] Besonders aufwarten kann der Raspberry Pi mit seinem vollwertigen Betriebssystem. Das sogenannte Raspbian basiert auf Debian und wurde speziell für den Raspberry Pi entwickelt. [20]

In dieser Arbeit wird der Raspberry Pi Model B+ verwendet. Dieser kam Mitte 2014 auf den Markt und löst damit das Model B ab. Das Model B+ bietet mehr Leistung sowie eine größere Anzahl an GPIO- und USB-Anschlüssen. Des Weiteren wurde die Stromaufnahme reduziert, die Audioausgabe verbessert und der SD-Kartenslot wurde durch einen Micro-SD-Kartenslot ersetzt. [21]

Eine genaue Übersicht über die Hardwaredaten [22] des Raspberry Pi Model B+ sieht wie folgt aus:

- Chipsatz: Broadcom BCM2835
- Chip Architektur: ARM11
- CPU: 700 MHz ARM1176JZFS
- GPU Dual Core VideoCore IV8
- Arbeitsspeicher: 512 MB SDRAM
- Betriebssystem: Linux (Raspian) von SD-Karte
- Abmessungen: 85 x 56 x 17 mm
- Stromzufuhr: 5V über Micro-USB
- Netzwerk: 10/100 MB/s
- Video Output: HDMI
- Audio Output: HDMI, 3,5 mm Klinkenstecker
- USB: 4x USB 2.0
- GPIO-Anschlüsse: 40x 2,54 mm

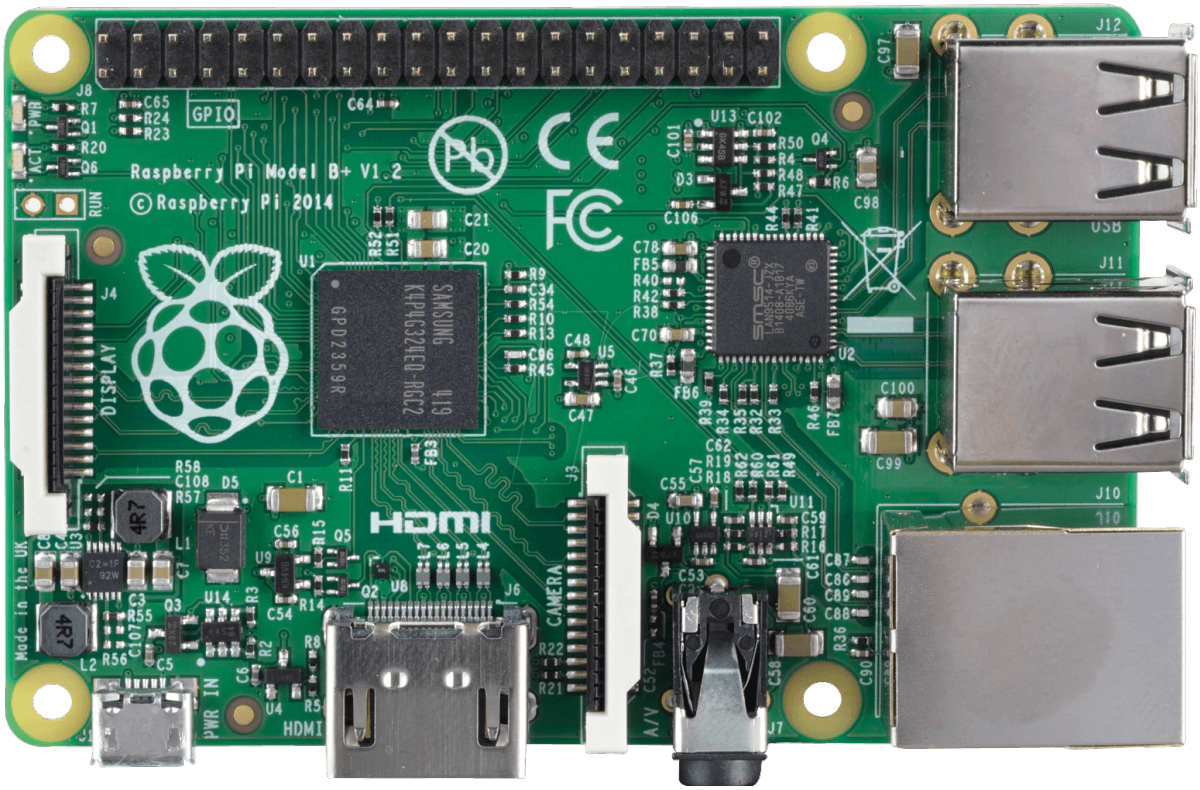

Abbildung 2.1: Raspberry Pi Modell B+[2]

[2] Bildquelle: https://cdn-reichelt.de/bilder/web/xxl_ws/A300/RASPBERRY_PI_B_PLUS_06.png

2.1.1 GPIO – General Purpose Input/Output

Die 40-Polige Steckerleiste auf dem Raspberry Pi ermöglicht das Schalten verschiedener Signale, diese Eingabe und Ausgabe-Anschlüsse werden als General Input/Output zusammengefasst. Die GPIO-Anschlüsse ermöglichen das Ansteuern von diversen Hardwarekomponenten wie Leuchtioden (LED) oder Schrittmotoren. Die GPIO-Signale werden direkt vom BCM-2835 Chip auf dem Raspberry Pi verarbeitet, deshalb ist beim Anschluss von höherwertiger Spannung an die GPIO-Anschlüsse höchste Vorsicht geboten, da es hier zu einer Beschädigung des Raspberry Pi kommen kann. [25] Die Anschlüsse sind physikalisch von links nach rechts auf dem Board nummeriert (siehe Abbildung 2.2).

Die GPIO-Anschlüsse des Pi übernehmen dabei verschiedene Aufgaben und können nicht alle genutzt werden. Die 40 Anschlüsse teilen sich auf in zwei 3,3 Volt, zwei 5-Volt Strompins, acht 0V Massepins, zwei DNC (do not connect) und 26 GPIO – Anschlüsse auf. Neun der GPIO-Anschlüsse verfügen dabei über erweiterte Funktionen, diese umfassen I2C, SPI und UART. [26] Die zwei DNC Anschlüsse sind nur für den erweiterten Gebrauch und lassen sich durch die P4J bzw RPi.GPIO Bibliothek (siehe 2.1.1.2 und 2.1.1.3) als GPIO-Anschlüsse ansprechen.

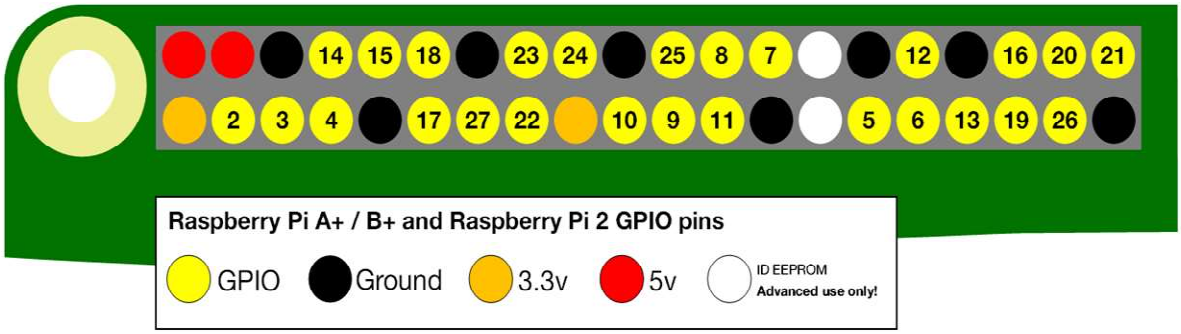

Abbildung 2.2: GPIO-Pin-Belegung für den Raspberry Pi B+[3]

Es gibt eine Besonderheit bei den GPIO-Anschlüssen 2(SDA), 3(SCL) und 14(TXD). Wenn an diese LEDs angeschlossen sind, flimmern diese leicht beim Hochfahren des Raspberry Pi. Dies liegt an den speziellen Funktionen der Anschlüsse. Wenn der Zustand über die Konsole abgefragt wird, erscheinen die Anschlüsse als ausgeschaltet. Nach der Abfragen erscheinen auch die LEDs als ausgeschaltet und es liegt kein Strom an den GPIO-Anschlüssen an [40]

[3] Bildquelle: https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/images/gpio-numbers-pi2.png

2.1.2 Ansteuerung der GPIO-Anschlüsse

Das Ansteuern der GPIO-Anschlüsse erfolgt über Bibliotheken, die für die jeweilige Programmiersprache auf dem Raspberry Pi installiert werden müssen. Getestet werden WiringPi, Pi4J und RPi.GPIO. Die Nutzung und Eigenschaften werden im folgenden Abschnitt verdeutlicht.

2.1.2.1 WiringPi

WiringPi ist eine Bibliothek die den Zugriff auf die GPIO-Anschlüsse des Raspberry Pi ermöglicht. Entwickelt wurde WiringPi in C für den BCM2835 Prozessor und kann in C und C++ genutzt werden. [9] Der Entwickler von WiringPi hat ein eigenes Schema zur Pin-Nummerierung der GPIO-Anschlüsse entwickelt, da es immer wieder zu Problemen und Änderungen der Pin-Bezeichnung bei neuen Raspberry Pi Revisionen kam. Dieses Schema zur Pin Nummerierung ermöglicht es Programmen auch nach der Änderung des Modells noch ordnungsgemäß zu funktionieren. [10] Dies ist besonders im weiteren Verlauf der Arbeit wichtig, da Pi4J auf diesem Schema aufbaut. Eine Abbildung des Schemas für den Raspberry PI sieht wie folgt aus:

P1: The Main GPIO connector							
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin
		3.3v	1	2	5v		
8	Rv1:0 - Rv2:2	SDA	3	4	5v		
9	Rv1:1 - Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1:21 - Rv2:27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3.3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
WiringPi Pin	BCM GPIO	Name	Header		Name	BCM GPIO	WiringPi Pin

Abbildung 2.3: WiringPi Pin Schema[4]

[4] Bildquelle: http://wiringpi.com/wp-content/uploads/2013/03/gpio1.png

Dieses Schema sieht für den Raspberry Pi B+ vervollständigt wie folgt aus [24]:

Wiring PI Pin	BCM GPIO	Name	Header		Name	BCM GPIO	Wiring PI Pin
		3,3v	1	2	5v		
8	Rv1: 0 – Rv2:2	SDA	3	4	5v		
9	Rv1: 1 – Rv2:3	SCL	5	6	0v		
7	4	GPIO7	7	8	TxD	14	15
		0v	9	10	RxD	15	16
0	17	GPIO0	11	12	GPIO1	18	1
2	Rv1: 21 – Rv2: 27	GPIO2	13	14	0v		
3	22	GPIO3	15	16	GPIO4	23	4
		3,3v	17	18	GPIO5	24	5
12	10	MOSI	19	20	0v		
13	9	MISO	21	22	GPIO6	25	6
14	11	SCLK	23	24	CE0	8	10
		0v	25	26	CE1	7	11
30	DNC	SDA0	27	28	SCL0	DNC	31
21	5	GPIO21	29	30	0v		
22	6	GPIO22	31	32	GPIO26	12	26
23	13	GPIO23	33	34	0v		
24	19	GPIO24	35	36	GPIO27	16	27
25	26	GPIO25	37	38	GPIO28	20	28
		0v	39	40	GPIO29	21	29

Tabelle 2.1: WiringPi Schema für den Raspberry Pi B+

WiringPi wird in dieser Arbeit nur in Abhängigkeit von Pi4j verwendet. Eine direkte Einbindung in den Webservice findet nicht statt, da WiringPi zum Zeitpunkt dieser Arbeit nicht in Java oder Python zur Verfügung steht. Ein Verständnis über das Schema für die Pin-Nummerierung von WiringPi ist für die Arbeit dennoch nötig.

2.1.2.2 Pi4J

Das Pi4J Projekt stellt mit seinem gleichnamigen API (Application Programming Interface) eine benutzerfreundliche objekt-orientierte Programmierschnittstelle für Java Entwickler zur Verfügung um die GPIO-Anschlüsse auf dem Raspberry Pi anzusprechen. Dieses Projekt befindet sich zum Zeitpunkt dieser Arbeit noch in der Entwicklung und liegt in der Version 0.9 vor. [11] Pi4j ermöglicht das Ansprechen der GPIO-Anschlüsse nach dem WiringPi Nummerierungsschema. Hierfür implementiert Pi4j einen JNI (Java Native Interface) Wrapper der WiringPi Bibliothek. WiringPi muss hierfür auf dem RaspberryPi installiert sein um die Funktionen zur Laufzeit zu garantieren. [12] (siehe Abschnitt 3.3.1)

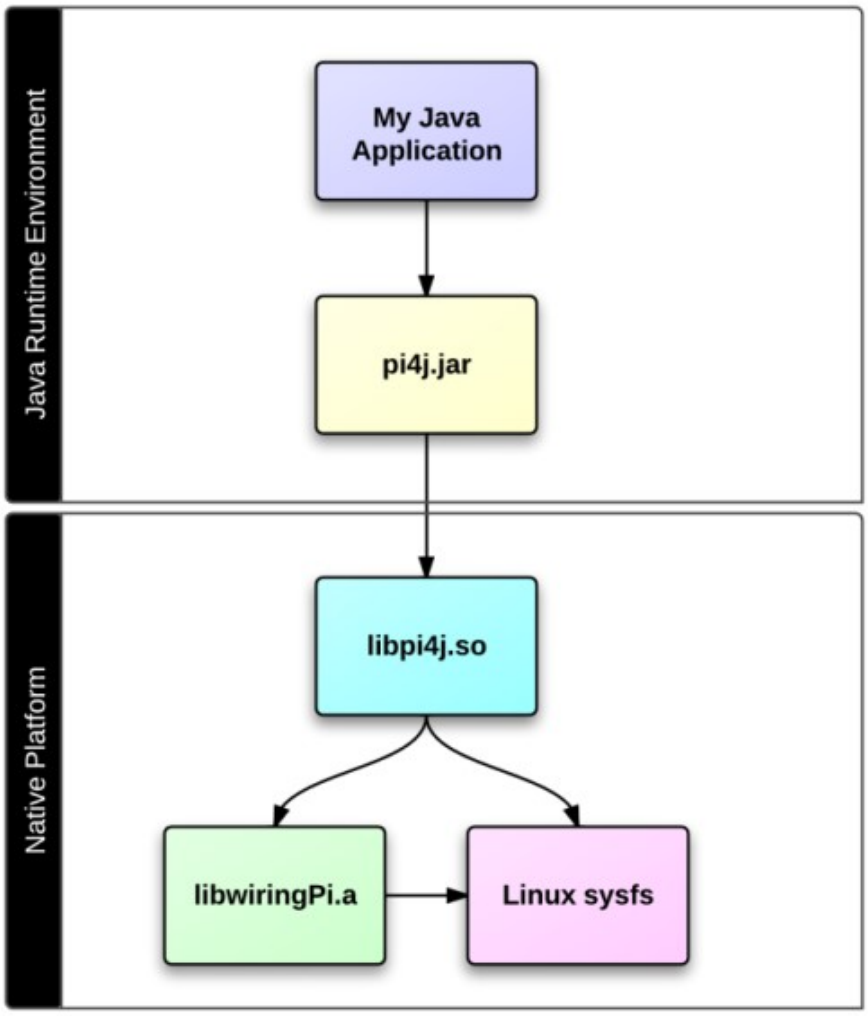

Abbildung 2.4: Pi4j Laufzeit verhalten[5]

Pi4j bietet viele Interessante Funktionen um die GPIO-Anschlüsse anzusprechen. Die gegebenen Funktionen gehen über die Funktionalität von WiringPi hinaus und bieten so mehr Möglichkeiten und Flexibilität.
Die folgende Tabelle zeigt die Initialisierung der GPIO-Anschlüsse und verwendbare Funktionen von Pi4J:

```
01 import com.pi4j.io.gpio.GpioController;
02 import com.pi4j.io.gpio.GpioFactory;
03 import com.pi4j.io.gpio.GpioPinDigitalOutput;
04 import com.pi4j.io.gpio.PinState;
05 import com.pi4j.io.gpio.RCMPin;
06
07 public class Pi4Example {
08     public static void main(String[] args) {
09
10         /*Erstellen des GpioControllers um zugriff auf die
11          * GPIO-Anschlüsse zu erhalten */
```

[5] Bildquelle: http://pi4j.com/images/dependencies.png

```
12          final GpioController gpio = GpioFactory.getInstance();
13
14          //GPIO-Pin 2 als DigitalOutput Pin definieren und zuweisen
15          GpioPinDigitalOutput pin =
                        gpio.provisionDigitalOutputPin(RCMPin.GPIO_02);
16
17          /*Beispiel Funktionen von Pi4J: */
18
19          //Verhalten des Pins beim Beenden der Applikation festlegen
20          pin.setShutdownOptions(true, PinState.LOW);
21
22          //Pin 2 einschalten
23          pin.high();
24
25          //Den aktuellen Zustand des Pins umschalten
26          pin.toggle();
27
28          //Den GPIO-Pin für eine Sekunde einschalten
29          pin.pulse(1000, true);
30
31          //Alle GPIO-Aktivitäten beenden
32          gpio.shutdown();
33      }
34
35  }
```

Tabelle 2.2: Beispielfunktionen von Pi4j

Die Ansteuerung der GPIO-Anschlüsse mittels Pi4j war erfolgreich. Es ist möglich, die GPIO-Anschlüsse mit Pi4J anzusprechen und den Zustand zu ändern.

2.1.2.3 RPi.GPIO

Um ein Ansteuern der GPIO-Anschlüsse unter Python zu realisieren, wird das RPi.GPIO-Modul der Python Software Foundation genutzt. [13] Als einzige Bibliothek ermöglicht es RPI.GPIO zwischen zwei Schemen für die Pin-Nummerierung zu wählen. Das BCM Schema ist analog der Belegung des P1-Headers des Raspberry Pi. Wie WiringPi und Pi4j bietet dies den Vorteil, dass der Code nicht von der Board Revision des Raspberry Pi abhängig ist. Alternativ lassen sich die GPIO-Anschlüsse auch nach dem BOARD Nummerierungsschema ansprechen. Dieses Schema richtet sich nach den Channels des Broadcom Chips an die die jeweiligen GPIO-Pins angeschlossen sind. BOARD hat jedoch den entscheidenden Nachteil, dass der Code abhängig von der Board Revision ist und somit nicht auf jedem Raspberry Pi gleich ausgeführt werden kann. [14]

Tabelle 2.3 zeigt die Initialisierung von RPi.GPIO und verwendbare Funktionen [14] des Moduls:

```
#Impotieren des RPi.GPIO-Moduls um den zugriff
#auf die GPIO-Anschlüsse zu ermöglichen
import RPi.GPIO as GPIO

#Pin-Schema festlegen
GPIO.setmode(GPIO.BCM)

#GPIO-Pin 0 als Outputpin zuweisen
GPIO.setup(0, GPIO.OUT, initial=GPIO.LOW)

#Pin einschalten
GPIO.output(0, 1)

#Pin ausschalten
GPIO.output(0, 0)

#Beenden der GPIO-Aktivitäten
GPIO.cleanup()
```

Tabelle 2.3: Beispielfunktionen von RPI.GPIO

Die Untersuchung des RPi.GPIO Modules ergab das es für den Einsatz in dieser Arbeit geeignet ist.

Der folgende Abschnitt beschreibt die Funktionsweisen von Webservices. Es werden Webservices im Allgemeinen erklärt sowie ein Vergleich zwischen der SOAP- und REST-Architektur vollzogen.

2.2 Webservices

Webservices haben in den letzten Jahren eine enorme Bedeutung erlangt. Ob in Onlineshops, im Cloudcomputing oder beim Steuern intelligenter Heimsysteme kommen moderne Webservices zum Einsatz. Webservices stellen wohldefinierte Softwareanwendungen dar, die Daten über ein Netzwerk bereitstellen. Jeder Webservice besitzt mindestens eine eindeutige URI (Uniform Resource Identifier) oder auch URL (Uniform Resource Locator) über die dieser zu erreichen ist. [39] Die bereitgestellten Daten des Webservices werden im maschinenlesbaren Format XML (Extensible Markup Language) übertragen. [8][18] Eine Client-Anwendung stellt eine Anfrage an den Webservice und die daraus resultierende Information wird dem Client in XML zurückgeliefert. Der Client kann dabei eine Webseite oder eine Applikation auf

Smartphones oder dem Computer sein. [6] Dies nennt sich auch „Konsumieren eines Webservices“.

Es gibt verschiedene Möglichkeiten einen Webservice zu implementieren, am häufigsten kommt hierbei die XML-RPC Weiterentwicklung SOAP zum Einsatz. SOAP ist ein komplexer Webservice, der neben dem eigentlichen Service auch eine Schnittstellendefinition sowie Middleware benötigt. Eine Alternative stellt die REST-Architektur dar. Die Eigenschaften der SOAP- und REST-Architektur werden im folgenden Abschnitt genauer betrachtet.

SOAP: Als Weiterentwicklung von XML-RPC wurde SOAP 1999 entwickelt. Das SOAP-Protokoll nutzt XML-basierende Nachrichten um entfernte Prozessaufrufe darzustellen und den Austausch von Daten zu ermöglichen. SOAP kann dabei zum Datenaustausch, als Nachrichtensystem oder für entfernte Prozessaufrufe zum Einsatz kommen. [23] Ein Client stellt seine Anfrage über das Hypertext Transfer Protocol (HTTP) an den SOAP-Service. Die Anfrage wird vom Client in einer XML-Nachricht verpackt und als SOAP-Nachricht verschickt. Eine SOAP-Nachricht setzt sich aus dem SOAP-Envelope, dem SOAP-Header und dem SOAP-Body zusammen:

- **SOAP-Envelope:** Bildet die oberste Ebene der Nachricht und enthält den Header und den Body

- **SOAP-Header:** SOAP-Header sind optional und werden für Erweiterungen genutzt wie z.B. Sicherheitsinformationen oder Transaktionskontexte

- **SOAP-Body:** Enthält die Nutzdaten wie den Funktionsaufruf, Fehlermeldungen und Rückgabewerte.

Der Server verarbeitet die Anfrage und prüft, ob die im SOAP-Body angegebene Methode in der Schnittstellenbeschreibung vorhanden ist. Die Antwort wird wieder in XML verpackt und als SOAP-Nachricht zurück an den Client gesendet. Um einen SOAP-Webservice zu erstellen, muss der Server, die Schnittstellen sowie die SOAP-Nachricht entwickelt werden. Zusätzlich muss darauf geachtet werden welches Protokoll zum Einsatz kommt, da SOAP neben HTTP auch noch diverse andere Protokolle unterstützt. [38]

SOAP ist somit ein Schwergewicht unter den Webservices, welches für größere Webservices mit diversen Sicherheitsaspekten sehr geeignet ist. Für kleine Projekte oder Webservices, die eine einfache Interaktion mit einem System bereitstellen, ist SOAP zu komplex.

REST: Roy Thomas Fielding beschreibt in seiner Desertation ein Architekturmodell, das erörtert wie das Word-Wide-Web funktionieren sollte. Fielding verweist auf sechs

Eigenschaften, die ein Dienst haben muss um als eine „RESTful Application“ zu gelten: [27]

- **Client-Server:** Als Anforderung gilt die Client-Server-Architektur. Durch die Trennung von Client und Server wird eine größere Skalierbarkeit und Flexibilität erreicht, da die einzelnen Komponenten sich unabhängig voneinander entwickeln können.

- **Stateless (Zustandslosigkeit):** Die Anfragen eines Clients an den Server enthalten alle Informationen die benötigt werden um vom Server bearbeitet zu werden. Hierbei sollten zwischen zwei Anfragen keine Zustandsinformationen gespeichert werden. Zustandslosigkeit fördert die Zuverlässigkeit und die Skalierbarkeit von Webservices. So ermöglicht Zustandslosigkeit eine vereinfachte Lastenverteilung über diverse Maschinen.

- **Cache:** Wie im World Wide Web können Clients Antworten speichern. Cache setzt voraus, dass Daten in einer Antwort Explizit beschreiben, ob diese gespeichert werden können oder nicht. Durch das Speichern von Antworten wird die Netzwerkeffizienz gesteigert, da ein Client die Daten für eine spätere Verwendung speichern kann und nicht dieselbe Funktion mehrmals aufrufen muss.

- **Uniform Interface (Einheitliche Schnittstellen):** Die zentrale Eigenschaft die REST von anderen Architekturstilen unterscheidet. Die Einheitlichen Schnittstellen ermöglichen eine einfache Nutzung eines Webservice auf Basis der REST-Architektur.

- **Layered System (Mehrschichtiges System):** Das Mehrschichtige System vereinfacht die Architektur, da einem Benutzer nur eine Schnittstelle geboten werden muss. Ebenen, die hinter dieser Schnittstelle liegen, können verborgen bleiben, was die Architektur eines Webservices insgesamt vereinfacht.

- **Code-On-Demand:** Code-on-Demand ist optional in Fieldings Beschreibung. Ein Server kann die Funktionalität eines Clients erweitern, indem er dem Client Scripts oder Applets zum Download bereitstellt. Da dies aber von Nachteil für die Transparenz ist, wird Code-On-Demand als optional in der REST Architektur angesehen.

Durch die einheitlichen Schnittstellen (Uniform Interfaces) stellt REST eine begrenzte Anzahl von Funktionen bereit, mit denen Ressourcen angefordert oder geändert

werden können. Für die Kommunikation wird das HTTP- oder HTTPS-Protokroll eingesetzt. Bei der Verwendung des HTTP-Protokolls stellt REST folgende Befehle zur Kommunikation bereit[6]:

- **GET:** Fordert eine Ressource vom Server an. GET weist keine Nebeneffekte auf, da der Zustand am Server nicht verändert wird.

- **POST:** Mit POST ist es möglich, neue Ressourcen zu dem Webservice hinzuzufügen.

- **PUT:** Aktualisiert oder ersetzt die aktuelle Ressource.

- **DELETE**: Löscht die angegebene Ressource.

Es gibt zwei unterschiedliche Arten von HTTP-Nachrichten, die zwischen Client und Server ausgetauscht werden. Ein HTTP-Request stellt eine Anfrage vom Client an den Server dar, während eine HTTP-Response Nachricht die Antwort vom Server darstellt. [40] Der Aufbau einer HTTP-Nachricht sieht dabei wie folgt aus:

- **Methode**: Zeigt an welche Funktion (GET/POST/PUT/DELETE) zur Kommunikation genutzt wird

- **URL**: Adresse unter welcher der Webservice erreicht werden kann.

- **Header**: Im Header werden Argumente und Parameter, die zur Datenübertragung benötigt werden, festgelegt wie z.B.: welche Datentypen als Antwort erwartet werden, die gewünschte Sprache und der gewünschte Zeichensatz sowie Informationen über den Client, der die Anfrage stellt.

- **Body**: Wenn Daten in einer Anfrage übertragen werden, sind diese im Body der HTTP-Nachricht gespeichert. Bei einer HTTP-Request-Anfrage kann der Body leer sein, während bei einer HTTP-Response-Antwort immer Daten im Body enthalten sind. Der Datentyp, welcher übertragen wird, ist im Header festgelegt.

Leonard Richardson entwickelte dazu einen Maßstab, der angibt, wie strikt ein Service REST implementiert. Das REST Maturity Model (RMM) unterscheidet in 4 Level, die in Tabelle 2.4 verdeutlicht werden [28]:

Level	Name	Eigenschaft
0	The Swamp of POX	XML-RPC oder SOAP, nur eine URI, Verwendung einer HTTP Methode (PUT)
1	Resources	Verwendet Ressourcen. Verschiedene URIs, Verwendet eine http Methode (PUT)
2	HTTP Verbs	Verwendet verschiedene URIs und Ressourcen, verwendet mehrere HTTP-Methoden (GET/PUT)
3	Hypermedia Control (HATEOS)	Verwendet verschiedene URIs und Ressourcen, verwendet mehrere HTTP-Methoden (GET/PUT), verwendet Hypermedia zur Navigation

Tabelle 2.4: REST Maturity Model

In dieser Arbeit wird ein Webservice auf Basis der REST-Architektur genutzt. REST ist durch vorgegebene Struktur ein Leichtgewicht unter den Webservices und bietet interessante Möglichkeiten, einen Webservice zu realisieren. Durch die Verwendung des HTTP-Protokolls und die somit bereits vorgegebenen Schnittstellen ermöglicht ein RESTful-Webservice eine flexible Kommunikation und erleichterte Programmierung mit denselben Funktionen, die ein SOAP-Webservice zur Verfügung stellen kann.

2.2.1 Bibliotheken und Werkzeuge

Um einen RESTful-Webservices zu realisieren werden weitere spezielle Bibliotheken und Module benötigt. Die hierfür verwendeten Bibliotheken und Module stehen dabei jeweils nur für eine Sprache bereit.

2.2.1.1 Jersey

Die Java Programmierschnittstelle für REST wird JAX-RS genannt. JAX-RS ist seit der Version 1.1 ein offizieller Bestandteil der Java Plattform. Im Mai 2013 ist JAX-RS in der Version 2.0 mit der Java 7-Plattform erschienen und stellt seit dem die Basis für RESTful-Webservices unter Java dar. [8][15]

Eine Referenzimplementierung der JAX-RS API stellt das Open-Source-Projekt Jersey dar. Das Jersey-Framework erweitert die JAX-RS-Spezifikationen um einige Funktionen, die es erleichtern einen RESTful-Webservice zu realisieren. [16]

Tabelle 2.4 zeigt die Initialisierung und Beispielfunktionen für HTTP GET- und POST-Methoden mithilfe von Jersey:

```
01 import javax.ws.rs.Consumes;
02 import javax.ws.rs.GET;
03 import javax.ws.rs.POST;
04 import javax.ws.rs.Path;
05 import javax.ws.rs.Produces;
06 import javax.ws.rs.core.MediaType;
07
08 /**
09  * Root URI für die Ressource zuweisen
10  */
11 @Path("Jersybeispiel")
12 public class Jersybeispiel {
13
14     /**
15      * Methode zum Behandeln der HTTP-GET-Anfragen.
16      * Ein Objekt des Medientyps Text wird
17      * zurückgeliefert.
18      */
19     @GET
20     @Produces(MediaType.TEXT_PLAIN)
21     public String getString() {
22         return "Erfolg!";
23     }
24
25     /**
26      * Methode zum Behandeln von HTTP-Post-Anfragen.
27      * Die Methode konsumiert den Namen eines Benutzers
28      * als Text und gibt ihn dann samt einer Begrüßung
29      * wieder aus.
30      */
31     @POST
32     @Consumes(MediaType.TEXT_PLAIN)
33     @Produces(MediaType.TEXT_PLAIN)
34     public String greetings (String Name) {
35         String text = "Willkommen ";
36         text.concat(Name);
37         return text;
38     }
39 }
```

Tabelle 2.5: Beispielfunktionen von Jersey

2.2.1.2 web.py

web.py ist ein Web-Framework welches von Aaron Schwarz entwickelt wurde und die Entwicklung von RESTful-Webservices in Python erlaubt. Das Framework liegt zum Zeitpunkt der Arbeit in Version 0.37 vor und bietet eine vollwertige Programmierschnittstelle um einen Webservice zu erstellen. [17]

Tabelle 2.6 zeigt ein Funktionsbeispiel für POST und GET Funktionen mit web.py:

```
01 #web.py Modul importieren
02 import web
03
04 #URIS der Webanwendung festlegen
05 uris = (
06     '/', 'index'
07 )
08
09 #Die Klasse index behandelt HTTP-GET-Anfragen und gibt
10 #"Hello, world" zurück
11 class index:
12     def GET(self):
13         return "Hello, world!"
14
15     def POST(self,name):
16         return "Hello " + name
17
18
19 #Starten des Webservice
20 if __name__ == "__main__":
21     app = web.application(uris, globals())
22     app.run()
```

Tabelle 2.6: Beispielfunktionen von web.py

3. Ansteuerung und Realisierung

Diese Kapitel beschreiben den Aufbau sowie die Steuerung der GPIO-Anschlüsse eines Raspberry Pi über einen „RESTful“ Webservice. Es werden zunächst auf die Anforderungen eingegangen sowie auf die Installation der abhängigen Bibliotheken und Module. Im Anschluss folgt eine Beschreibung der Funktionalität der einzelnen Komponenten.

3.1 Anforderungen

Um diese Arbeit zu realisieren, sind außer dem Raspberry Pi weitere Hard- und Softwarekomponenten nötig. Die GPIO-Anschlüsse sind über Steckverbindungen (Jumperkabel) mit einem Steckbrett (Breadboard) verbunden. Auf diesem Steckbrett befinden sich LEDs sowie Widerstände um die Stromausgabe der GPIO-Anschlüsse auf eine angemessene Stärke für die LEDs zu regeln. Beim Anschließen der einzelnen Komponenten ist Vorsicht geboten, da es bei einer falschen Verkabelung im schlimmsten Fall zu einer Beschädigung einzelner Komponenten kommen kann [5]

Um den Raspberry Pi nutzen zu können, werden Peripheriegeräte wie Maus, Tastatur und ein HDMI-Kabel benötigt. Um störende Kabel zu reduzieren, lässt sich der Raspberry Pi auch alternativ über das Netzwerk mittels einer Secure Shell- (SSH) oder Remotedesktopverbindung administrieren. Maus, Tastatur und Video-Anschluss sind somit überflüssig.

Um einen Webservice in Java zu realisieren wird eine Entwicklungsumgebung benötigt. Die Spring Tool Suite (STS) ist eine modifizierte Eclipse-Version, die speziell für die Entwicklung von Webanwendungen in Java ausgelegt ist. STS hat bereits alle nötigen Erweiterungen und Frameworks, die zur Entwicklung von Webservices in Java benötigt werden. [29]

Um den Java Webservice sowie die Webseite, die den Service konsumiert, bereitzustellen, wird ein Webserver benötigt. Hierfür kommt Tomcat zum Einsatz. Tomcat ist ein Open-Source-Webserver der von der Apache Software Foundation entwickelt wird. [30]

Der Raspberry Pi wird über eine microSD-Speicherkarte gebootet, auf welcher vorher das Linux basierende Raspian „Wheezy“ installiert wurde (siehe Abschnitt 3.3). Die Stromversorgung des Raspberry Pi sowie der LEDs auf dem Steckbrett erfolgt über ein Mikro-USB Kabel, welches direkt am Raspberry Pi angeschlossen wird.

3.2 Schaltungsaufbau

Es werden alle nutzbaren GPIO-Anschlüsse sowie ein 0v (Ground) Anschluss benutzt. Die gesamt Anzahl von nutzbaren GPIO-Anschlüssen liegt bei dem Modell B+ bei 28. Es werden also 28 LEDs benötigt um die Zustände der GPIO-Anschlüsse darzustellen. Damit eine LED korrekt funktioniert, wird für jede LED ein Widerstand benötigt. Diese werden vor die LEDs geschaltet, um den Strom auf eine passende Stärke anzupassen. Um den Stromkreislauf zu schließen muss jede LED mit einem 0v Anschluss verbunden werden. Um nur einen 0v Anschluss des Raspberry Pi zu belegen, werden die 0v Ausgänge der LED mit einer Überbrückung auf eine Schiene im Steckbrett geschaltet. So ist es möglich eine Schaltung zu realisieren die mehrere Komponenten bedient, aber nur einen 0v Anschluss benötigt

Daraus folgt, dass 28 LEDs, 28 Wiederstände, 29 Jumperkabel sowie 28 Überbrückungskabel benötigt werden, um die Zustände der GPIO-Anschlüsse darzustellen. Jeder GPIO-Anschluss wird dabei einzeln mit einem Widerstand und einer LED verbunden. Die LEDs sind dabei physikalisch von links nach rechts auf dem Steckbrett angeordnet und nummeriert. Abbildung 3.1 zeigt den Anschluss einer LED an den Raspberry Pi.

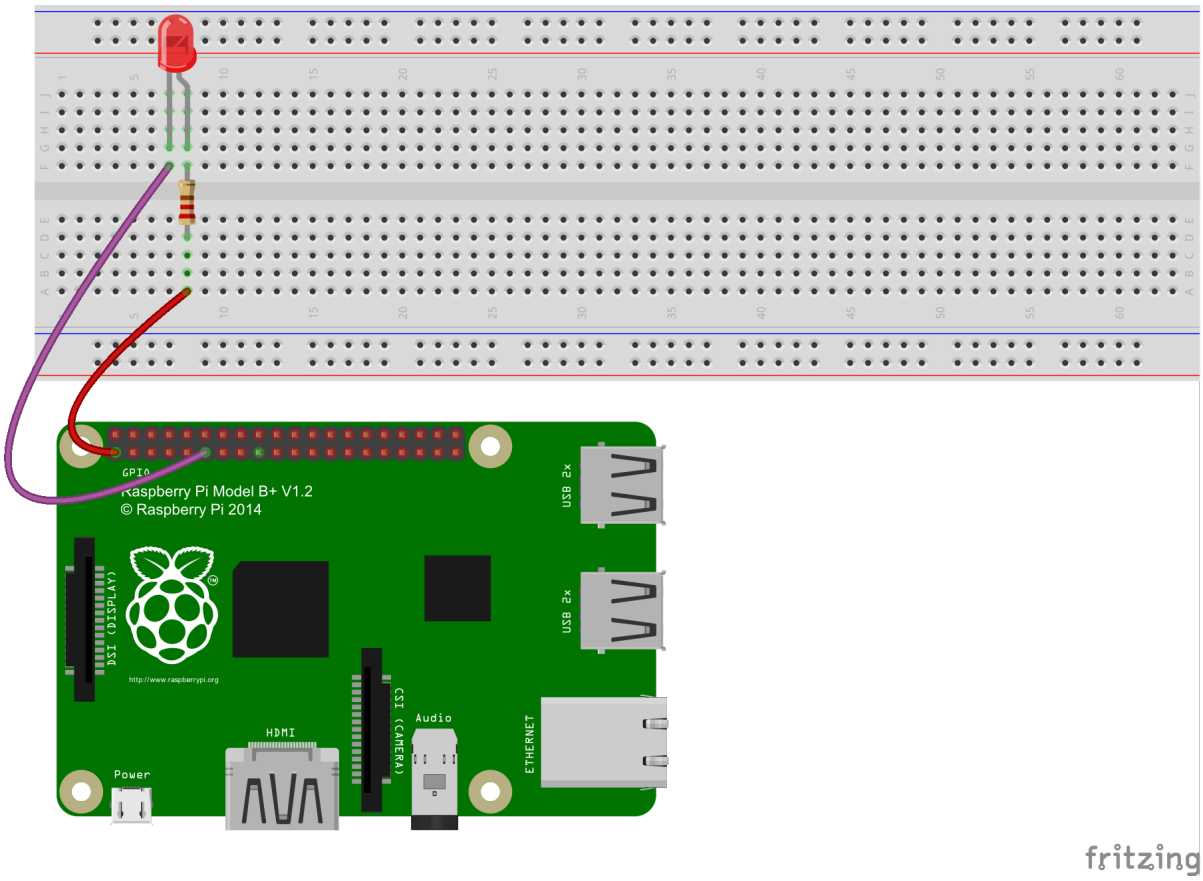

Abbildung 3.1: Anschluss einer LED [6]

[6] Bildquelle: http://www.sunfounder.com/js/edit/attached/image/20150506/20150506163025_85954.png

Tabelle 3.1 zeigt die Verbindungsübersicht der LEDs mit den GPIO-Anschlüssen:

GPIO-Pin	LED		GPIO-Pin
02	0	1	25
06	2	3	05
04	4	5	03
29	6	7	28
24	8	9	01
00	10	11	27
16	12	13	15
26	14	15	23
14	16	17	12
13	18	19	10
11	20	21	22
21	22	23	07
09	24	25	08
31	26	27	30

Tabelle 3.1: LED/GPIO Anschluss-Übersicht

Die folgenden Abbildungen 3.2 und 3.3 stellen den Schaltungsaufbau sowie die angeschlossenen Verbindungen grafisch dar:

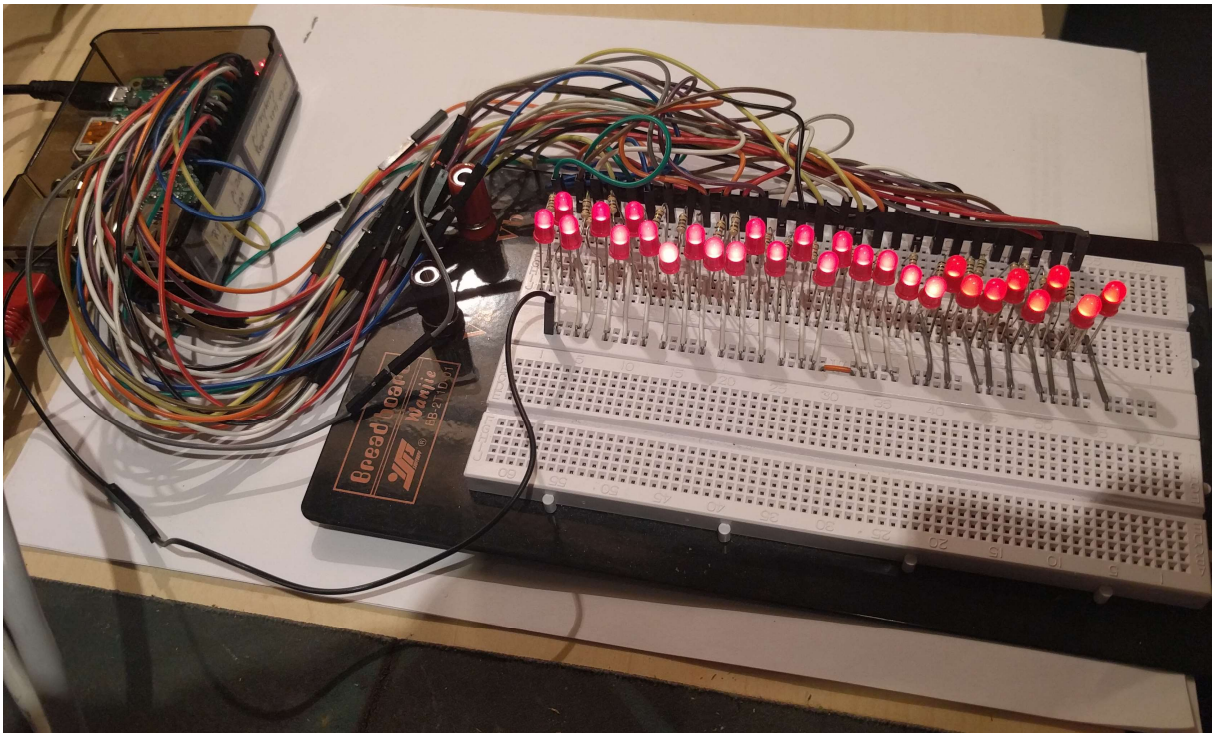

Abbildung 3.2: Schaltungsaufbau

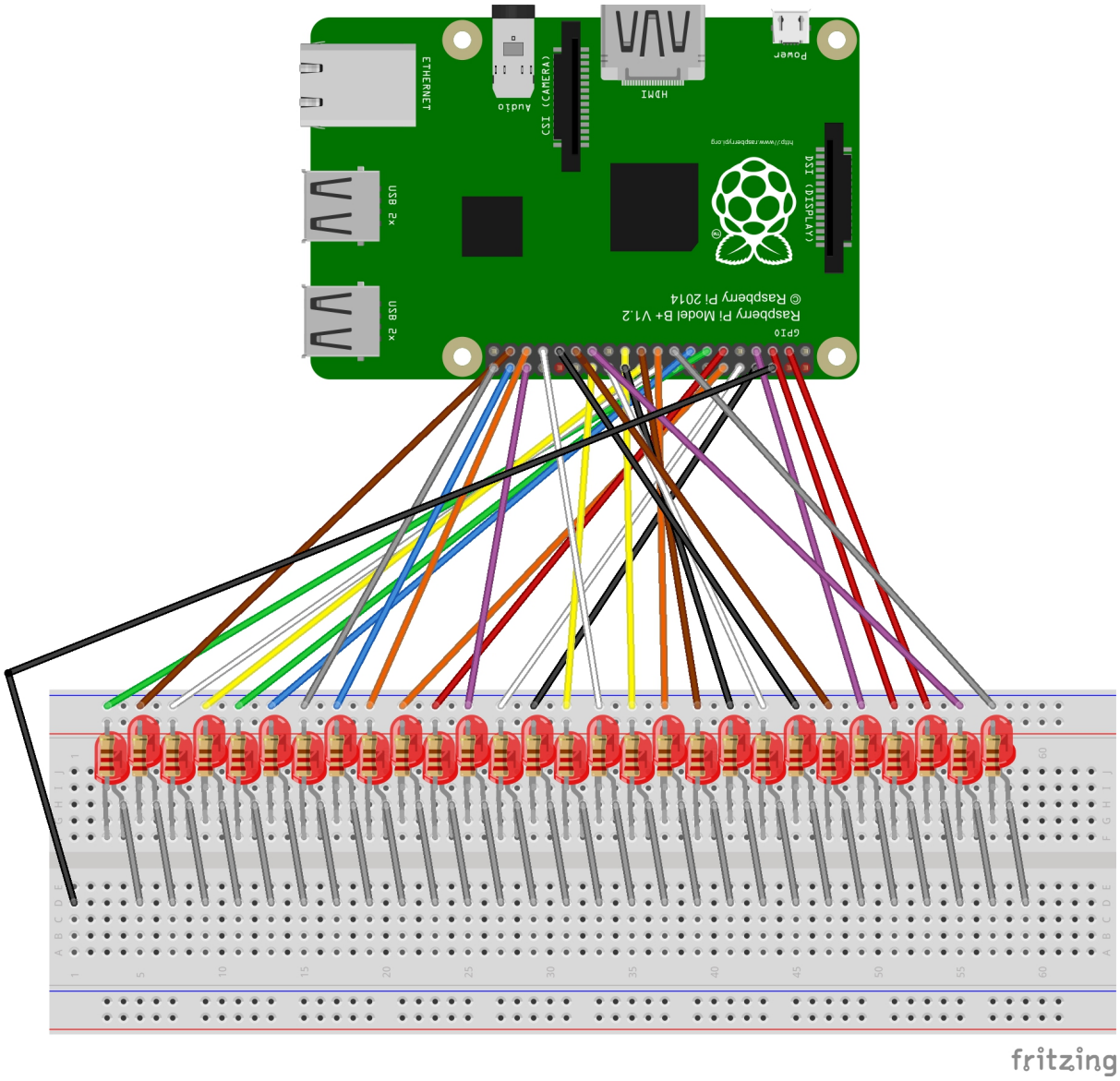

Abbildung 3.3: Übersicht der Schaltung als Fritzing-Diagramm [7]

[7] Quelle: http://fritzing.org/download/

3.3 Konfiguration des Raspberry Pi

Um den Raspberry Pi in Betrieb nehmen zu können, muss zuerst ein Betriebssystem heruntergeladen und installiert werden. Die Raspberry Pi Foundation empfiehlt Raspbian oder NOOBS als Betriebssystem einzusetzen. In dieser Arbeit wird das Betriebssystem „Raspbian - Wheezy “ eingesetzt welches direkt von der Webseite der Raspberry Pi Foundation bezogen werden kann. Raspbian zählt zu den beliebtesten Linux-Distributionen für den Raspberry Pi. Das auf Debian basierende Betriebssystem wird kontinuierlich mit Updates versorgt und ist besonders für die bereits vorinstallierten Programme sowie für die gute Performance beliebt. [2] Wheezy ist die aktuelle Distribution von Raspian und hat ein Größe von 990 Megabyte.

Um Raspbian auf einer Micro-SD-Karte unter Windows zu installieren, muss es zuerst entpackt und anschließend mit dem Zusatzprogramm Win32 Disk Imager auf der SD-Karte installiert werden. Wind32 Disk Imager kann über die Sourceforge Webseite des Herstellers bezogen werden. Nach der Installation und dem Starten von Win32 Disk Imager, muss der Pfad zur Raspbian-Abbilddatei sowie der Pfad zur Micro-SD-Karte ausgewählt werden auf welcher Raspbian installiert werden soll (z.B. „E:\“). Mit dem Betätigen des „Write“-Knopfes beginnt die Installation.

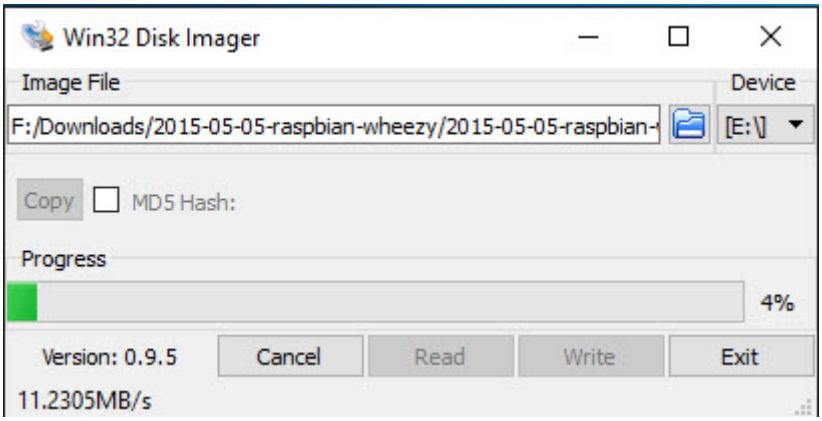

Abbildung 3.4: Installation von Raspian-Wheezy

Sobald die Installation abgeschlossen ist, kann die Micro-SD-Karte in den Raspberry Pi eingeschoben werden. Um den Raspberry Pi über SSH zu administrieren, wird noch ein LAN-Kabel benötigt. Sobald alles angeschlossen ist, kann die Powerversorgung über den Micro-USB Stecker beginnen.

Der Raspberry Pi ist so konfiguriert, dass er eine IP-Adresse automatisch per DHCP-Server des Routers bezieht. Um den Raspberry Pi administrieren zu können ist es nötig, die IP-Adresse herauszufinden. Sollte der Raspberry Pi an einen Monitor angeschlossen sein, lässt sich die IP-Adresse über den Konsolenbefehl „*ifconfig*“

anzeigen. Sollte der Raspberry Pi nur über das Netzwerk angeschlossen sein, kann die IP-Adresse aus dem Router ausgelesen werden. Sobald die IP-Adresse gefunden wurde, kann mithilfe des Programms „Putty“ mit der Konfiguration des Pis begonnen werden.

Um eine Verbindung herzustellen, ist die ermittelte IP-Adresse einzugeben und als Protokoll SSH auszuwählen. Über den Knopf „Open“ wird die Verbindung hergestellt.

Der Login in Raspbian erfolgt über den Standardbenutzer „*pi*“ mit dem Passwort „raspberry“. Nach dem ersten Login müssen die Raspbian-Systemeigenschaften angepasst werden. Um diese zu ändern muss das entsprechende Menü mit dem Befehl „*sudo raspi-config*“ aufgerufen werden. Die folgende Tabelle und Abbildung zeigt die anzupassenden Einstellungen:

Einstellung	Ebene	Kontext
Expand Filesystem	Hautpmenü	Ermöglicht Raspbian die komplette Größe der SD-Karte zu nutzen (Standard 2GB)
Change User Password	Hauptmenü	Ändern des Benutzerkennworts.
Internationalisation Options	Hauptmenü	Ermöglicht das Anpassen von Sprache, Tastaturlayout sowie der Zeitzone
Change Locale	Internationalisation Options	Ändern des verwendeten Zeichensatzes. (Für Deutsch ist „de_DE.UTF-8 UTF-8“ auzuwählen
Change Timezone	Internationalisation Options	Ändern der Zeitzone. Hierfür ist zuerst Europa und danach Berlin auszuwählen.
Change Keyboard Layout	Internationalisation Options	Ändern des Tastatur-Layouts. Im Untermenü ist zuerst „Other“ und anschließend „German“ auszuwählen.

Tabelle 3.2: Einstellungen unter Raspbian

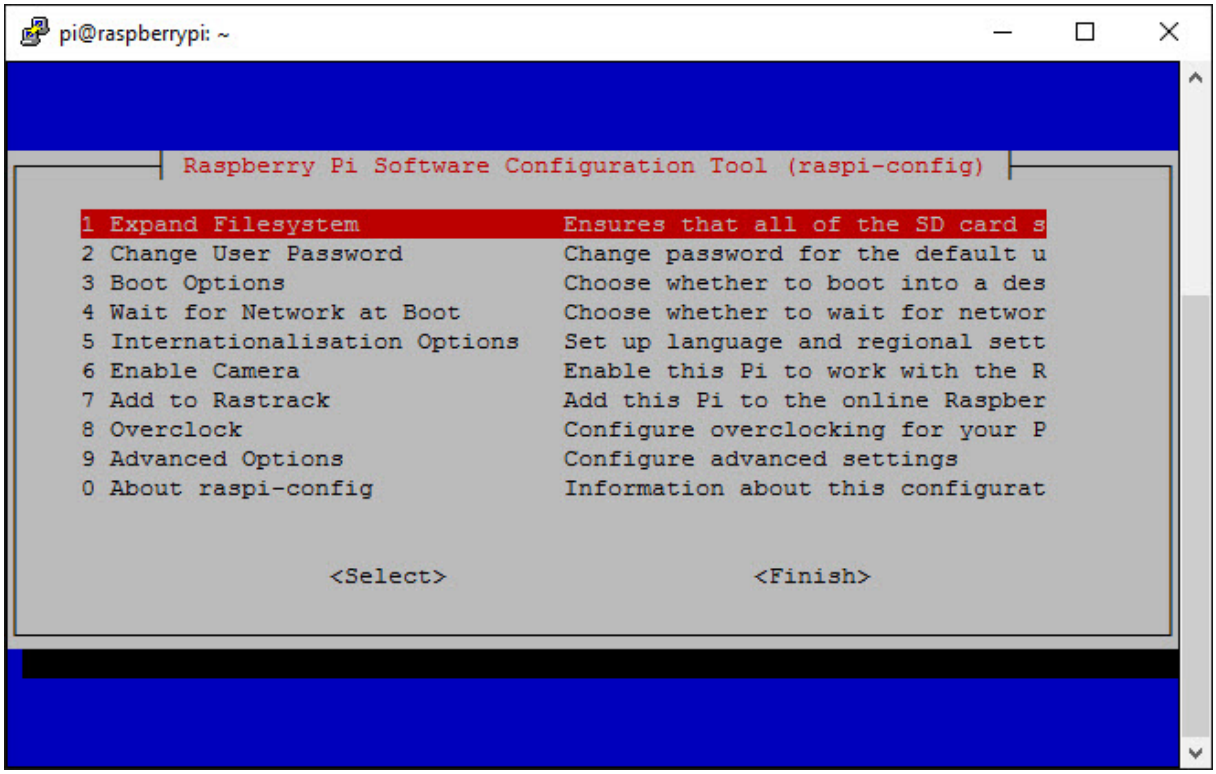

Abbildung 3.5: raspi-config

Sobald alle Einstellungen geändert wurden, sollte das System aktualisiert werden. Hierfür wird zuerst der Befehl *„sodo apt-get update*“ ausgeführt und anschließend „*sudo apt-get upgrade*“, um alle installierten Anwendungen auf den neuesten Stand zu bringen.

Um die geänderten Einstellungen zu übernehmen, muss der Raspberry Pi anschließend neugestartet werden. Dies erfolgt über den Befehl „*sudo shutdown –r now*“. Der Zusatz –r zeigt dabei einen Neustart des Systems an.

Sobald der Raspberry Pi wieder hochgefahren ist, sollte als letzte Konfiguration noch eine feste IP-Adresse vergeben werden. Dies stellt sicher, dass der Raspberry Pi jedes Mal mit der gleichen Adresse im Netzwerk erreichbar ist und der Webservice somit immer die gleiche Adresse besitzt. Um dem Raspberry Pi eine statische IP-Adresse zuzuweisen, müssen die Interfaces unter „*/etc/network/interfaces*“ mit einem beliebigen Editor bearbeitet werden. Ein kompletter Aufruf sieht mit Texteditor Nano wie folgt aus: „*sudo nano /etc/network/interfaces*“. Hier muss die Schnittstelle „eth0“ angepasst werden. Eine vollständige IPV4-Konfiguration mit einer statischen Adresse sieht wie folgt aus:

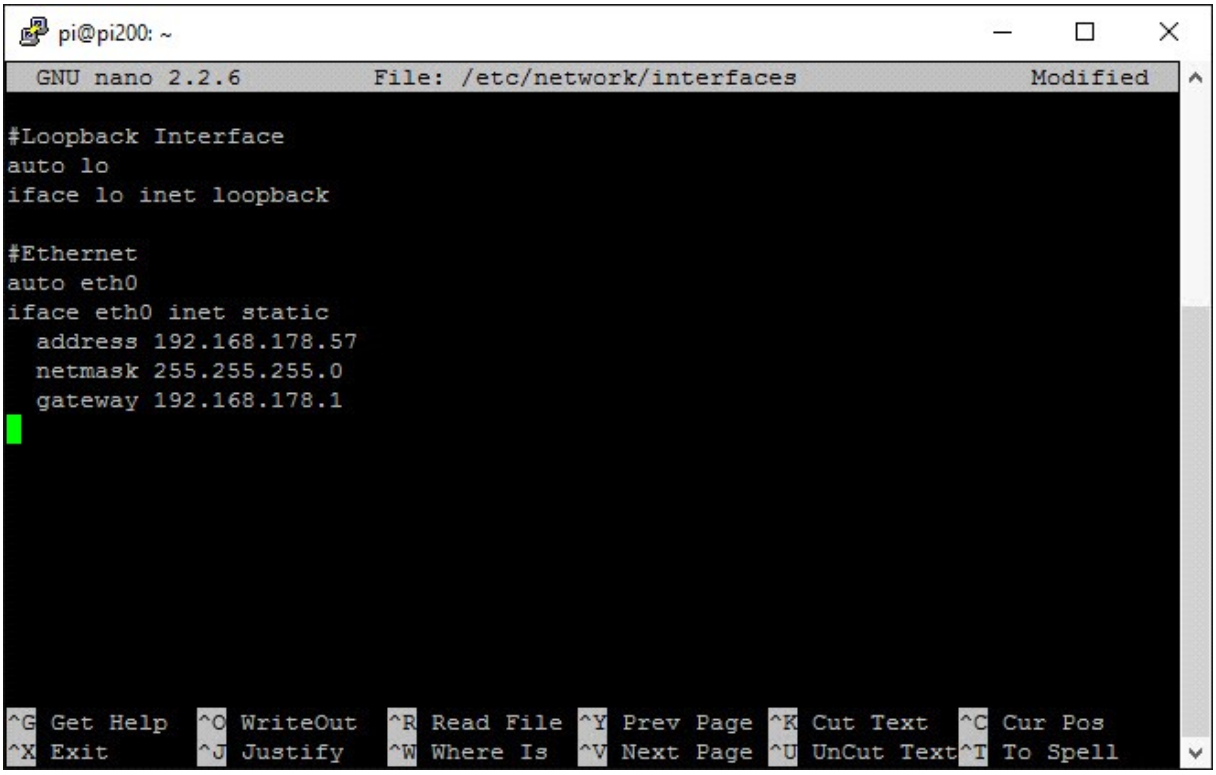

Abbildung 3.6: Statische IPV4-Konfiguration

Der Raspberry Pi ist nun für die Verwendung konfiguriert und es kann mit der Installation der abhängigen Bibliotheken und Module fortgefahren werden.

3.3.1 Installation der abhängigen Bibliotheken und Module

Die Installation der abhängigen Pakete und Module erfolgt über die Kommandozeile von Raspbian. Es wird die Installation von Java, WiringPi, RPi.GPIO sowie Tomcat erläutert. Bevor mit der Installation der Pakete begonnen werden kann, sollte die Quellenliste von Raspian mittels „*sudo apt-get update*“ aktualisiert werden. Somit wird gewährleistet, dass die bezogenen Bibliotheken und Anwendungen auf dem aktuellen Stand sind.

Java: Java wird als Development Kit (JDK) für Tomcat sowie den Webservice benötigt. JDK liegt zum Zeitpunkt dieser Arbeit in Version 8 Update 69 vor. Mithilfe des Befehls „*sudo apt-get install oracle-java8-jdk*“ wird Java auf dem Raspberry Pi installiert. Die Verwendung von Java ist somit möglich. [35]

WiringPi: WiringPi kann über den Online-Dienst GitHub von GIT bezogen werden. GIT ist ein webbasierter Onlinedienst für Entwickler, um gemeinsam an Open-Source-

Projekten arbeiten zu können. Das Archiv GitHub stellt die Bibliotheken dann zum Download bereit. Die Installation erfolgt über den Befehl „sudo apt-get install git-core“. Nach der Installation kann WiringPi mit dem Befehl „git clone git://git.drogon.net/wiringPi“ bezogen werden. Nun wird mit dem Befehl „*cd wiringPi*“ in das Verzeichnis von WiringPi gewechselt und mittels „*git pull origin*“ die Bibliothek aktualisiert. Anschließend kann WiringPi mit dem Befehl „*./build*“ installiert werden. [31]

RPI.GPIO: Um das RPI.GPIO-Modul beziehen zu können, muss zuerst die Python Entwicklungsumgebung nachinstalliert werden. Nach der Installation mittels „*sudo apt-get install python-dev*“ kann das Modul mit „*sudo apt-get install python-rpi.gpio*“ bezogen werden. [32]

Tomcat: Die eingesetzte Tomcat Version 8 steht in zwei verschiedenen Varianten zum Herunterladen bereit. So kann Tomcat als Service oder alleinstehend bezogen werden. [34] Der Service bietet den Vorteil, dass Tomcat mit dem Betriebssystem startet und dauerhaft verfügbar ist. Da dies aber für die Entwicklung hinderlich ist, wird in dieser Arbeit die alleinstehende Version genutzt. Diese bietet den Vorteil, dass Tomcat nur aktiv ist, wenn es gewünscht ist. Das Tomcat-Paket kann von der Apache Tomcat-Seite[8] heruntergeladen werden. Anschließend muss das Paket mittels „*tar –xvzf apache-tomcat-8.0.28.tar.gz*“ entpackt werden.

Nachdem Tomcat entpackt ist, muss der Webserver noch konfiguriert werden. Dafür muss die Datei tomcat-users.xml im Unterverzeichnis „conf/“ bearbeitet werden. Der Befehl „*sudo nano tomcat-users.xml*“ öffnet die Datei im Schreibprogramm Nano. In der Zeile „<tomcat-users> < /tomcat-users >“ kann ein Benutzer wie folgt angelegt werden [33]:

```
<tomcat-users>
<user username="admin" password="W3bserv!c3" roles="manager-gui,admin-gui"/>
</tomcat-users>
```

Tabelle 3.3: Anlegen eines Benutzers für Tomcat

Tomcat ist somit konfiguriert und kann eingesetzt werden. Tomcat lässt sich über die Scripts „startup.sh“ und shutdown.sh“ im Unterverzeichnis „bin“ der Tomcat-Installation steuern.

[8] http://tomcat.apache.org/download-80.cgi

```
pi@pi200 ~/apache-tomcat-8.0.28/bin $ sudo ./startup.sh
Using CATALINA_BASE:   /home/pi/apache-tomcat-8.0.28
Using CATALINA_HOME:   /home/pi/apache-tomcat-8.0.28
Using CATALINA_TMPDIR: /home/pi/apache-tomcat-8.0.28/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /home/pi/apache-tomcat-8.0.28/bin/bootstrap.jar:/home/pi/
apache-tomcat-8.0.28/bin/tomcat-juli.jar
Tomcat started.
pi@pi200 ~/apache-tomcat-8.0.28/bin $
```

Abbildung 3.7: Starten von Tomcat

Um zu testen, ob Tomcat ordnungsgemäß funktioniert, kann die Adresse des Raspberry PI mit dem Zusatz: „:8080“ in die Adresszeile des Browsers eingegeben werden. Abbildung 3.8 zeigt die Tomcat Willkommensseite nach erfolgreicher Konfiguration:

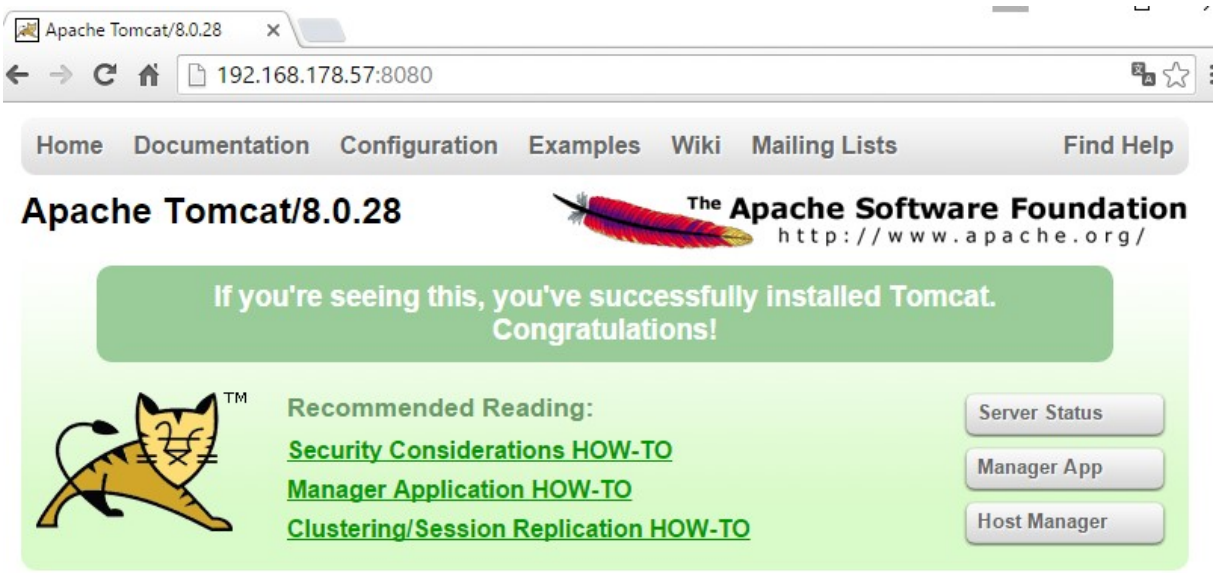

Abbildung 3.8: Tomcat Willkommensseite

3.4 RESTful-Webservice

Der RESTful-Webservice zur Ansteuerung der GPIO-Anschlüsse wird in Java wie Python gleich entwickelt. Es wird eine abstrakte Klasse Pin erstellt, die alle benötigten Informationen wie die ID, den GPIO-Anschluss (Port) sowie den Zustand des jeweiligen Anschlusses abbildet. Um die Anschlüsse steuern zu können stellt der Webservice zwei URIs bereit. Jeder URI besitzt eine „GET“ und eine „PUT“ Methode um das kommunizieren mit dem Webservice zu erlauben. Um die Daten zu übertragen wird die XML-Alternative JSON (JavaScript Object Notation) eingesetzt. JSON ist ähnlich wie XML ein schlankes Datenaustauschformat, welches für Benutzer einfach zu lesen und schreiben ist sowie durch Maschinen einfach zu verarbeiten ist. [36]

Die folgende Tabelle gibt eine Übersicht über die URIs sowie Ihre Methoden:

URI	GET	PUT
/gpio-service/webapi/board	Liefert alle Pins im Format JSON zurück.	Erwartet eine Liste mit Pins im Format JSON. Die Liste kann hierbei 1-28 Pins enthalten.
/gpio-service/webapi/board/(.*)	Liefert einen spezifischen Pin im Format JSON zurück.	Nimmt einen Pin im Format JSON entgegen und ändert den Zustand des entsprechenden Pins

Tabelle 3.4: URIs des RESTful-Webservice

Der Webservice ist somit im REST Maturity Model auf Level 2 und implementiert alle Voraussetzungen der REST-Architektur. Der letzte Level des RMM kann nicht erreicht werden, da der Webservice keine Navigation bereitstellt und somit HATEOS nicht implementieren kann.

Die folgenden Abschnitte zeigen die Realisierung des Webservice in Java und Python. Hierbei werden nur die grundlegenden Funktionen des Webservice gezeigt. Der komplette Quellcode kann dem Anhang entnommen werden.

3.4.1 Java

Der RESTful-Webservice in Java wird in der Entwicklungsumgebung Spring Tool Suite unter Windows entwickelt. Nach dem Download und der Installation[9], muss ein neues Maven Projekt angelegt werden. Dies kann über den eingebauten Konfigurator oder über die Kommandozeile erstellt werden (siehe Abbildung 3.9). Beim Erstellen des Projekts sind die Abhängigkeiten „*glassfish.jersey.archetype*“ sowie „*jersey-quickstart-webapp*“ auszuwählen. Beim Erstellen des Projekts über die Kommandozeile ist folgender Befehl einzugeben: „*mvn generate –DarchetypeGroupeId=org.glassfish.jersey.archetypes –DarchetypeArtifactId=jersey-quickstart-webapp –DarchetypeVersion=2.2*“ Die folgende Abbildung zeigt den Erstellprozesses von Maven über die Kommandozeile:

[9] Quelle: https://spring.io/tools/sts/all

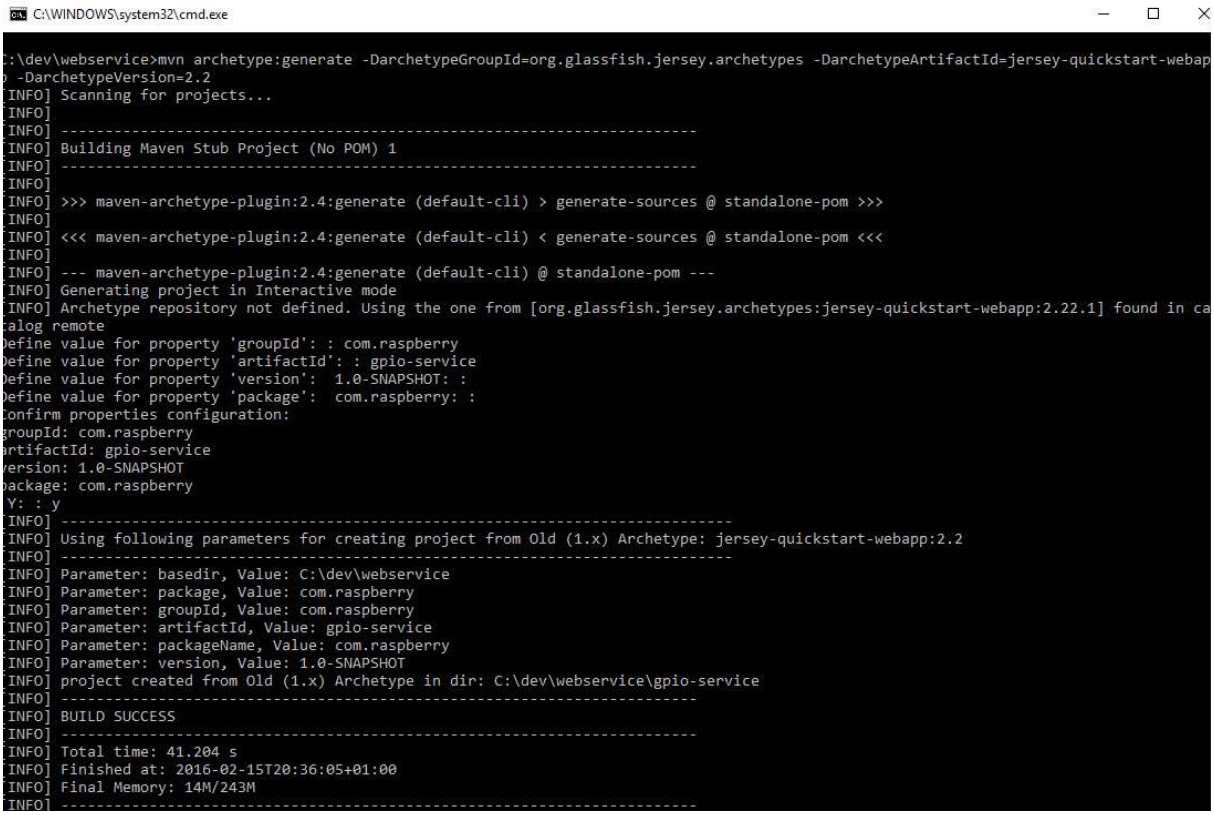

Abbildung 3.9: Erstellen des Maven Projekts.

Nach Abschluss der Konfiguration erstellt STS ein Projektverzeichnis und lädt alle benötigten Abhängigkeiten für das Projekt herunter. Nach dem Anlegen des Projekts ist die Adresse, unter welcher der Webservice erreichbar ist, vorgegeben. Dieser setzt sich aus *„%Name des Projekts%/webapi/%Pfad der Ressource%*“ zusammen. Die URL lässt sich nach dem Namen des Projekts, in der web.xml, beliebig verändern (siehe Abbildung 3.10).

Zusätzlich müssen in den Einstellungen des Projekts noch die Pi4J-Bibliotheken eingebunden werden, um eine Ansteuerung der GPIO-Anschlüsse zu erlauben. Dies geschieht über die „*Projekt Eigenschaften*“ -> „*Java Build Path*“. Dort muss im Reiter „*Libraries*“ der Pfad zu der PI4J-Bibliothek eingetragen werden.

Das so entstandene Projektverzeichnis sieht wie folgt aus:

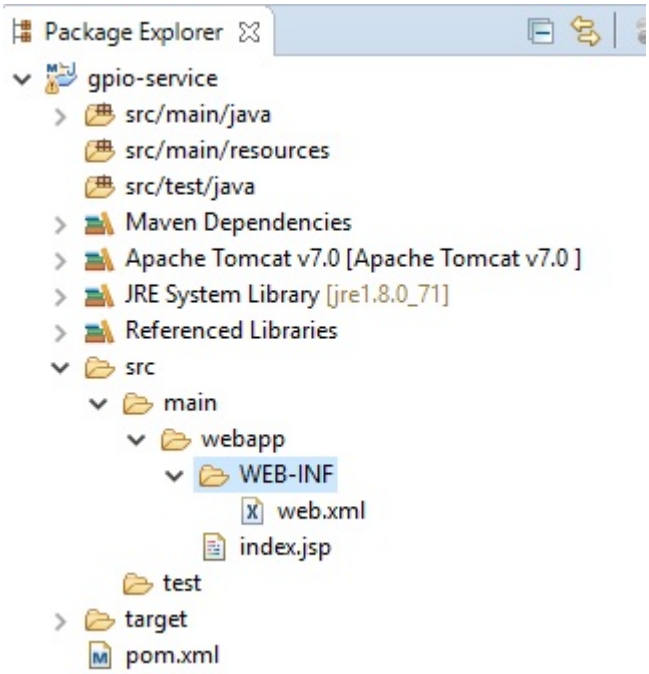

Abbildung 3.10: Java Projekt Struktur

- **src/main/java**: In diesem Verzeichnis werden alle Java-Klassen und Ressourcen gespeichert.

- **Referenced Libraries**: Beinhaltet alle nötigen Abhängigkeiten wie z.B. Pi4J

- **pom.xml**: Beinhaltet die Webservice-Informationen wie den Namen, Abhängigkeiten und Jersey Eigenschaften.

- **web.xml**: In der web.xml Datei kann die URI-Vorgabe geändert werden.

Der Webservice kann nun erstellt werden. Hierfür werden folgende Pakete in der Projektstruktur festgelegt:

- **com.raspberry**: Enthält die Ressource, die vom Webservice bereitgestellt wird.
- **com.raspberry.model**: Beinhaltet die abstrakte Klasse Pin sowie eine Klasse zum verwalten aller GPIO-Anschlüsse. Diese wird über das Interface angesprochen um einen mehrfachen Aufruf zu ermöglichen. Um eigene Klassen über den Webservice ausgeben zu können, müssen diese mit dem Zusatz „@XmlRootElement" versehen werden.
- **com.raspberry.repository**: Enthält das Interface und die Implementierung des Interfaces.

Die Ressource „BoardPinRessource" im Paket „com.raspberry" stellt dabei den eigentlichen Webservice zur Verfügung. Eine Übersicht der „BoardPinRessource" wird in folgender Tabelle verdeutlicht:

```
01 import java.util.List;
02
03 import javax.ws.rs.Consumes;
04 import javax.ws.rs.GET;
05 import javax.ws.rs.PUT;
06 import javax.ws.rs.Path;
07 import javax.ws.rs.PathParam;
08 import javax.ws.rs.Produces;
09 import javax.ws.rs.core.MediaType;
10
11 import com.raspberry.model.Pin;
12 import com.raspberry.repository.PinHolderInterface;
13 import com.raspberry.repository.PinHolderStub;
14
15 @Path("/board/")
16 public class BoardPinRessource
17     {
18         PinHolderInterface gpiofactory = new PinHolderStub();
19
20         @GET
21         @Produces(MediaType.APPLICATION_JSON)
22         public List<Pin> showAllPins()
23             {
24             return gpiofactory.getAllPins();
25             }
26
27         @PUT
28         @Consumes(MediaType.APPLICATION_JSON)
29         @Produces(MediaType.APPLICATION_JSON)
30         public List<Pin> setMultiplePins(List<Pin> pinstoSet)
31             {
32             List<Pin> allPins = gpiofactory.setMultiplePins(
                                    pinstoSet);
33             return allPins;
34             }
35
36         @GET
37         @Produces(MediaType.APPLICATION_JSON)
38         @Path("{pinId}")
39         public Pin showSpecificPin(@PathParam("pinId")String
                                       pinId)
40             {
41             return gpiofactory.getAllPins().get(Integer.
                                            parseInt(pinId));
42             }
43
44         @PUT
45         @Path("{pinId}")
46         @Consumes(MediaType.APPLICATION_JSON)
47         @Produces(MediaType.APPLICATION_JSON)
48         public Pin setPinState(Pin apin)
49             {
50             Pin anewPin=gpiofactory.setPin(apin,apin.
                          getIversedState());
51             return anewPin;
52             }
53     }
```

Tabelle 3.5: RESTful-Webservice in Java

- Zeile 1 – 13: Importieren der benötigten Bibliotheken
- Zeile 15-17: Festlegen des URI-Pfades „/board“ und die Definition der Klasse „BoardPinRessource“.
- Zeile 18: Initialisierung des GPIO-Interfaces, welches das Schalten und Auslesen der GPIO-Anschlüsse ermöglicht.
- Zeile 20 -25: Definition der „GET“-Methode welche alle GPIO-Anschlüsse ausgibt. @Produces gibt dabei an, in welchem Format die Daten ausgegeben werden.
- Zeile 27 – 34: Definition der „PUT“-Methode welche das Schalten mehrerer Anschlüsse ermöglicht. @Consumes gibt dabei an, welches Datenformat erwartet wird.
- Zeile 36 – 52: Definition der „GET“- und „PUT“-Methode um einen spezifischen GPIO-Anschluss anzuzeigen und gegebenenfalls zu ändern. Der zusätzliche Pfadparameter wird hierfür als ID des Anschlusses genutzt.

Der entwickelte Webservice wird nun als Web Application Archive (.WAR) exportiert und in das Unterverzeichnis Webapps der Tomcat-Installation kopiert. Um den Webservice anschließend zu aktivieren, muss Tomcat noch mithilfe von „*sudo startup.sh*“ im Unterverzeichnis „*bin*“ der Tomcat Installation-gestartet werden.

3.4.2 Python

Für Python wird keine Entwicklungsumgebung benötigt. Anwendungen können so einfach mithilfe von Schreibprogrammen wie z.B. Nano oder einem anderen beliebigen Texteditor geschrieben werden. Bei der Entwicklung von Python-Projekten ist besonders auf die Syntax zu achten. Python bietet keine Klammern um Funktionen zu definieren, dies geschieht über den Zeileneinschub. [37]

Der folgende Quellcode in Python zeigt, wie der Webservice zur Ansteuerung der GPIO-Anschlüsse umgesetzt werden kann.

```
01 import web
02 import json
03 import RPi.GPIO as GPIO
04
05 GPIO.setmode(GPIO.BCM)
06
07 urls = (
08     '/gpio-service/webapi/board', 'list_Pins',
09     '/gpio-service/webapi/board/(.*)', 'get_pin'
10 )
11
12 class list_Pins:
13
14     def GET(self):
15      output = json.dumps([pin.__dict__ for pin in pins],
                sort_keys=True)
```

```
16      return output
17
18     def PUT(self):
19      data = web.data()
20      jsonobj = json.loads(data)
21      switchpinlist = []
22      iter = len(jsonobj)
23      for i in range (0,iter):
24       switchpinlist.append(Pin(jsonobj[i]['id'], jsonobj[i][
                            'gpioPort'], jsonobj[i]['state']))
25
26      listiter = len(switchpinlist)
27      for x in range(0,listiter):
28       index = switchpinlist[x].getID()
29
30       currstate = pins[len(pins)-index -1].getState()
31
32       if currstate is 'HIGH':
33          GPIO.output(len(pins)-index -1, 0)
34       else:
35          GPIO.output(len(pins)-index -1, 1)
36
37      output=doList()
38      return output
39
40 class get_pin:
41    def GET(self, pinid):
42     index=int(pinid)
43     pin = pins[len(pins)-index -1]
44     output = json.dumps(pin.__dict__, sort_keys=True)
45     return output
46
47    def PUT(self,pin):
48     index=int(pin)
49     currstate = pins[len(pins)-index -1].getState()
50     if currstate is 'HIGH':
51          GPIO.output(len(pins)-index -1, 0)
52    else:
53          GPIO.output(len(pins)-index -1, 1)
54    output=doList()
55    return output
56
57 if __name__ == "__main__":
58     app = web.application(urls, globals())
59     app.run()
```

Tabelle 3.6: RESTful-Webservice in Python

- Zeile 1-3: Importieren der benötigten Module.
- Zeile 5: Einstellen des GPIO-Ansteuerungsschemas
- Zeile 7-10: Festlegen der benötigten URIs sowie der Funktionen die bei einem Aufruf des jeweiligen URI ausgelöst werden.
- Zeile 12: Definieren der Klasse „list_pins“ für die Darstellung und Interaktion mit einem oder mehreren GPIO-Anschlüssen.
- Zeile14 - 16: Definition der „GET“-Methode: Es werden alle GPIO-Anschlüsse mit Ihrer ID, dem Anschluss sowie dem aktuellen Zustand ausgegeben.

- Zeile 18-20: Definition der „PUT“-Methode. Die übertragenen Daten werden in das Array data geschrieben und im Anschluss als JSON-Objekt geladen.
- Zeile 21 – 25: Durch die erhaltene Liste iterieren und eine neue Liste mit Pins, deren Zustand gewechselt werden soll, erstellen.
- Zeile 26 – 36: Jeder Eintrag in der erstellten Liste wird nun aufgerufen und der Zustand geändert.
- Zeile 37: Erstellen einer neuen aktuellen Übersicht der GPIO-Anschlüsse.
- Zeile 38: Ausgabe der aktualisierten Zustände. Es werden alle GPIO-Pins zurückgeliefert.
- Zeile 40: Definition der Klasse „get-pin“ für die Interaktion mit einem expliziten GPIO-Anschluss
- Zeile 41-45: Definition der „GET“-Methode. Hierfür wird zuerst ausgelesen welcher Anschluss angezeigt werden soll. Anschließend wird dieser geladen und ausgegeben.
- Zeile :47-55: Definition der „PUT“-Methode. Analog zur „GET“-Methode wird der entsprechende Anschluss ausgelesen und geändert. Anschließend wird eine aktuelle Übersicht des Anschlusses zurückgeliefert.

Der Webservice wird mithilfe des Befehls „*sudo python gpiorest.py*“ gestartet und ist danach über die Adresse des Raspberry Pi erreichbar.

3.5 Konsumieren des Webservice

Der so erstellte Webservice lässt sich auf diverse Arten konsumieren. So lässt sich Beispielsweise eine native Anwendung oder eine Applikation auf einem Smartphone entwickeln, die mit dem Webservice kommuniziert und so ein Schalten der GPIO-Anschlüsse ermöglicht. In dieser Arbeit dient eine Webseite zur Veranschaulichung und Kommunikation mit dem Webservice. Hierfür wird sowohl Hypertext Markup Language (HTML) sowie JavaScript eingesetzt.

Die erstellte Webseite unterteilt sich dabei in eine HTML- sowie eine JavaScript-Datei. Die HTML-Datei stellt eine Webseite mit einer Tabelle zur Verfügung. Beim Aufruf der Webseite wird das JavaScript ausgeführt. Dieses ruft die URL „*192.168.178.57:8080/gpio-service/webapi/board*“ mittels GET-Funktion auf. Der erhaltene JSON-Datensatz wird daraufhin in der Tabelle der HTML-Webseite angezeigt.

```
01 $(document).ready(function() {
02
03    var JSONService = "http://192.168.178.57:8080/gpio-
                        service/webapi/board/";
04    $.ajax({
05          type: "GET",
06          contentType: "application/json",
07          dataType: "json",
08          url: JSONService
09     })
10    .done(function (response)
11    {
12          for(var i =0;i < response.length;i++)
13          {
14                var item = response[i];
15                var tablerow ='';
16                tablerow += '<tbody><tr><td align="center">' + item.
                  gpioPort + '</td><td align="center">' +item.
                  id +'</td><td align="center">' + item.state +
                  '</td><td align="center"><input
                  type="checkbox" id= ' + item.id +' value = '+
                  i+' name="check_list"/></td></tr></tbody>';
17                $('#gpio_table').append(tablerow);
18          }
19
20
21       })
22
23 });
```

Tabelle 3.7: JavaScript mit GET-Funktion

- Zeile 1: Deklaration der document.ready-Funktion. Diese wird aufgerufen sobald die HTML-Datei angewählt wird.
- Zeile 3: Festlegen der Adresse des Webservices

- Zeile 4- 9: Definition der GET-Funktion. Es werden Parameter wie die zu nutzende Funktion, der zu erwartende Medientyp sowie die Adresse des Webservices angegeben
- Zeile 10: Die .done Funktion wird automatisch aufgerufen sobald die eigentliche GET-Anfrage fertig ist.
- Zeile 12-16: Für jeden enthaltenen Eintrag in der Antwort des Webservices wird eine Tabellenzeile im Format HTML erstellt.
- Zeile 17: Die erstellte Tabellenzeile wird an die Tabelle der Webseite angehängt.

Abbildung 3.11 zeigt die grafische Darstellung der Webseite nach dem Aufruf:

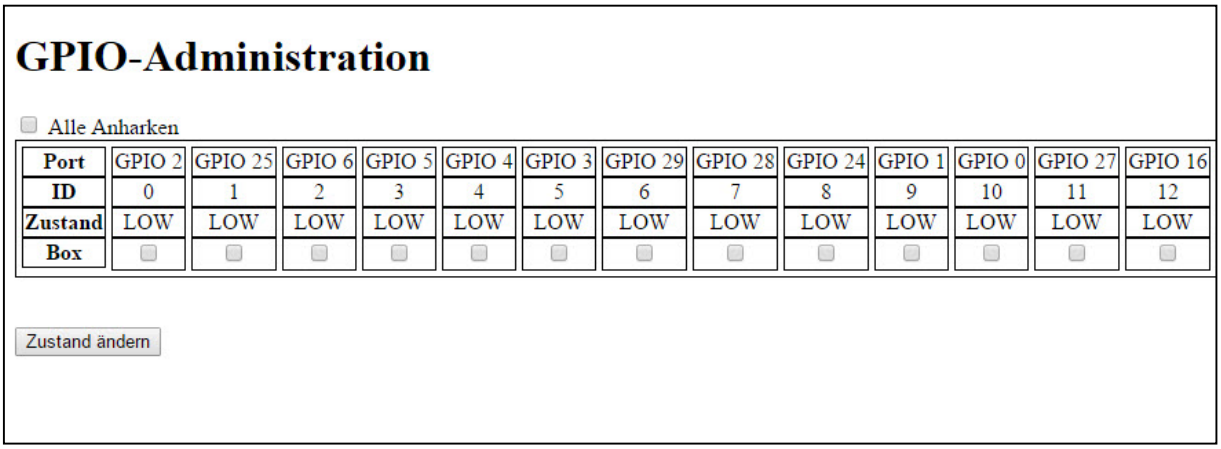

Abbildung 3.11: Webseite zum konsumieren des Webservice

Die so erhaltene Darstellung bietet eine Übersicht über die einzelnen GPIO-Anschlüsse sowie deren Zustand. Über die Checkboxen lassen sich die Anschlüsse auswählen und über den Knopf „Zustand ändern“ schalten. Es ist eine Auswahl von einem oder mehreren Anschlüssen möglich. Sollte kein Anschluss ausgewählt worden sein, wird eine Fehlermeldung eingeblendet.

```
01 function Send()
02 {
03    var checkboxes = new Array();
04    var checked = new Array();
05    checkboxes = document.getElementsByName('check_list');
06    for (i = 0; i < checkboxes.length; i++)
07    {
08          if(checkboxes[i].checked == true)
09          {
10           var PinObject = new Object();
11           PinObject.id = checkboxes[i].id;
12           PinObject.gpioPort = "somePort";
13           PinObject.state = "someState";
14             checked.push(PinObject);
15          }
16    }
17    if(checked.length > 0)
18    {
```

```
19          var jsonData = JSON.stringify(checked);
20
21          $.ajax
22          ({
23                headers:
24                {
25                 'Accept': 'application/json',
26                 'Content-Type': 'application/json'
27                },
28                type: "PUT",
29                url: 'http://192.168.178.57:8080/gpio-
                  service/webapi/board/',
30                dataType: 'json',
31                data: jsonData,
32                success: function (data)
33                {
34                 tbl=document.getElementById('gpio_table');
35                       while(tbl.rows.length>1)
36                 {
37                   tbl.deleteRow(1);
38                 }
39
40                 for (var i = 0, len = data.length; i < len; i++)
41                 {
42                  var item = data[i];
43                  var tablerow ='';
44                  tablerow += '<tbody><tr><td align="center">' + item.
                    gpioPort + '</td><td align="center">' +item.
                    id +'</td><td align="center">' + item.state
                    +'</td><td align="center"><input
                    type="checkbox" id= ' + item.id +' value =
                    '+i+' name="check_list"/></td></tr></tbody>
                    ';
45                  $('#gpio_table').append(tablerow);
46                 }
47
48                 mainbox = document.getElementById("mainbox");
49                 mainbox.checked = false;
50                  }
51             })
52
53          }
54     else
55     {
56          alert("Es wurden keine Anschlüsse ausgewählt!!");
57     }
58 }
```

Tabelle 3.8: PUT-Methode zum Interagieren mit dem Webservice

- Zeile 1: Definition der Funktion
- Zeile 3-4: Definition von Variablen, „checkboxen“ enthält alle Checkboxen während „checked“ nur die ausgewählten enthält.
- Zeile 5: Laden aller Checkboxen
- Zeile 6 - 16 : Jede Checkbox überprüfen, ob sie ausgewählt wurde, falls ja, erstelle ein Pin-Objekt und speichere dies in „checked“

- Zeile 17: if-Statement zum Abfangen von Fehlern, falls keine Checkbox ausgewählt wurde (siehe Zeile 54).
- Zeile 19: Die ausgewählten Anschlüsse werden in das JSON-Format umgewandelt
- Zeile 21 – 31: Funktion für die PUT-Methode. Analog zur GET-Funktion werden hier Variablen wie der Typ, das Format der Daten sowie die URL zum Webservice angegeben. Da die PUT-Methode Daten zurückliefert, müssen hierfür extra Header gesetzt werden, die eine Antwort im Format JSON erlauben.
- Zeile 32 - 38: Die Funktion .success wird nur im Erfolgsfall aufgerufen. Hier wird zuerst die Tabelle der HTML-Webseite ausgewählt und anschließend die alten Daten gelöscht.
- Zeile 40 - 46: Erstellen der neuen Tabellenzeilen mit aktualisierten Daten. Diese werden im Anschluss an die Tabelle angehängt.
- Zeile 48 - 49: Sollten alle Anschlüsse über die Checkbox „Alle auswählen“ ausgewählt worden sein, muss aus dieser Checkbox der Haken entfernt werden.
- Zeile 54-58: Ausgabe einer Fehlermeldung, falls keine GPIO-Anschlüsse ausgewählt wurden.

Die Webseite ist somit in der Lage den kompletten Webservice zu konsumieren und die GPIO-Anschlüsse sowohl anzuzeigen als auch einzeln oder alle auf einmal zu schalten.

4.Zusammenfassung

Die vorliegende Arbeit befasste sich mit der Ansteuerung der GPIO-Anschlüsse des Einplatinencomputers Raspberry Pi mithilfe eines RESTful-Webservices. Für die Realisierung des Webservices wurden zwei verschiedene Programmiersprachen getestet. Hierfür wurden die einzelnen GPIO-Anschlüsse mit einem Steckbrett verbunden, auf dem 28 LEDs angebracht sind. Das Steckbrett bietet den Vorteil, die Anschlüsse übersichtlicher zu gestalten und Schaltungen flexibel zu verändern.

Nachfolgend werden der Java- und Python-Webservice einer Bewertung unterzogen, in welcher die Vor- und Nachteile der jeweiligen Lösung hervorgehoben werden. Abschließend wird noch ein Ausblick über die weiteren Möglichkeiten des Webservice vermittelt.

4.1 Bewertung

Um den erstellten Webservice abschließend bewerten zu können müssen folgende Kriterien beachtet werden:

Kriterium	Bewertung
Aufwand	Welcher Aufwand ist nötig um das Projekt zu realisieren? Hierzu zählen benötigte Bibliotheken und Module sowie Software, die für die Entwicklung benötigt wurde.
Flexibilität	Es wird getestet, welche Funktionen der Webservice mit den integrierten Bibliotheken und Modulen ermöglicht.
Laufzeit	Für die Laufzeit werden jeweils drei GET- sowie drei PUT-Anfragen an den Webservice gestellt. Es wird die Zeit gemessen, die der Webservice bis zur Antwort benötigt
Erweiterungsmöglichkeit	Abschließend wird ein Ausblick auf die Erweiterungsmöglichkeiten des jeweiligen Webservices gegeben.

Tabelle 4.1: Bewertungskriterien

- **Aufwand**: Der Aufwand, um einen Webservice in Java zu realisieren, ist wesentlich komplexer als für einen gleichwertigen Webservice in Python. Neben

der P4J-Bibliothek zum Ansteuern der GPIO-Anschlüsse, wird eine Entwicklungsumgebung (STS) sowie ein Webserver (Tomcat) benötigt. Das Java-Projekt umfasst eine komplette Projektstruktur, während für Python nur das Modul RPi.GPIO sowie eine Datei mit dem Quellcode benötigt werden. Der programmiertechnische Aufwand stellt sich wie folgt dar:

  - **Java**: Fünf Dateien mit insgesamt 356 Zeilen Quellcode
  - **Python**: Eine Datei mit insgesamt 171 Zeilen Quellcode

- **Flexibilität**: Sowohl der in Java als auch der in Python geschriebene Webservice bieten rudimentär die gleichen Funktionen. Der Mehraufwand für den Java-Webservice wird durch die größere Flexibilität ausgeglichen. So ist es möglich, die URL des Webservice anzupassen oder in der Programmierumgebung direkt zu testen. Die P4J-Bibliothek bietet mehr Funktionen als das Python-äquivalent RPi.GPIO. Da P4J nicht nur die WiringPi Bibliothek implementiert, sondern auch eigene Funktionen (z.B.: Pulse()) [41] anbietet, lassen sich mit der P4J-Bibliothek die GPIO-Anschlüsse flexibler ansteuern.

  Durch den Einsatz von Tomcat bietet der Java Webservice einen weiteren entscheidenden Vorteil gegenüber Python: so lässt sich in Tomcat einstellen, was alles aufgezeichnet werden soll. Standardmäßig werden hier nur Fehler behandelt, aber es können zu Entwicklungszwecken auch Aufrufe und die übertragenden Daten aufgezeichnet werden. Der Python-Webservice bietet nur eine Ausgabe auf der Kommandozeile zur Laufzeit, die Aufrufe und Fehler ausgibt.

- **Laufzeit**: Für die Laufzeitmessung wurden jeweils drei GET- sowie drei PUT-Anfragen an den Webservice gesendet. Es wurde ein Durchschnittswert für die GET- sowie die PUT-Anfragen gebildet. Der daraus resultierende Durchschnittswert stellt eine komplette Interaktion mit dem Webservice dar (Aufruf sowie anschließende Übertragung von Daten) Die Laufzeitmessung ergab folgende Werte:

Sprache	Typ	Zeit in ms
Python	GET	34
		39
		32
	PUT	40
		44
		48
Java	GET	59
		72
		67
	PUT	116
		132
		133

Sprache	Methode	Zeit in ms
Python	GET	35
	POST	44
	Durchschnitt	**57**
Java	GET	66
	POST	127
	Durchschnitt	**96,5**

Tabelle 4.2: Laufzeit der Webservices

Es zeigt sich, dass der Python Webservice wesentlich schneller ist als der Java-Webservice. Dies ist möglich, da Python auf eine zusätzliche Softwareschicht wie Tomcat verzichtet.

- **Erweiterungsmöglichkeiten**: Es ist sowohl in Java als auch in Python möglich, durch das Einfügen zusätzlicher Bibliotheken und Module weitere Funktionalitäten bereitzustellen. Da Python in jedem beliebigen Texteditor entwickelt werden kann, fehlt es an Hilfsmitteln wie Syntax-Hervorhebungen oder Live-Debugging. Dies kann in größeren Projekten das Entwickeln erschweren, da leicht die Übersicht im Quelltext verloren geht.

Die Bewertung zeigt abschließend folgende Eigenschaften auf:
In der Realisierung mit Java wurde verdeutlicht, dass diese zwar komplexer im Aufbau, dafür aber sehr flexibel in der Programmierung ist. Die Ansteuerung der GPIO-Anschlüsse wird hierbei durch die Pi4J-Bibliothek ermöglicht. Durch das Verwalten des Webservice durch Tomcat lassen sich Aufrufe sowie übertagende Daten nachverfolgen und somit Fehlerquellen beseitigen. Hierfür ist das Laufzeitverhalten des Java-Webservices durch die Verbindung mit Tomcat langsamer.

Für kleinere Projekte ist eine Realisierung mit Python denkbar. Python lässt sich in jedem beliebigen Texteditor entwickeln und verfügt über viele vorgefertigte Funktionen. Um den entwickelten Webservice bereitzustellen wird keine Zusatzsoftware wie Tomcat benötigt. Der realisierte Webservice benötigt zusätzlich das RPi.GPIO Modul um die GPIO-Anschlüsse steuern zu können. Der Python-Webservice verdeutlicht die simplen Möglichkeiten, einen Webservice zu erstellen, doch gestaltet sich Administration und Fehlersuche wesentlich aufwändiger. Das Konsumieren eines Webservice in Python gestaltet sich jedoch schwieriger, da über

Python keine Webseite gehostet werden kann. Ein Parallelbetrieb mit Tomcat ist nicht ohne weiteres möglich, da der Webservice in Python bereits Port 8080 belegt.

Abschließend lässt sich sagen, dass eine Ansteuerung der GPIO-Anschlüsse des Einplatinencomputers Raspberry Pi über einen RESTful-Webservice erfolgreich war. Es wurde sowohl in Java als auch in Python eine Anleitung über den Aufbau sowie die Funktionsweise des Webservices vermittelt.

4.2 Ausblick

In dieser Arbeit wurden keine HTTP-Sicherheitsmechanismen implementiert. Es ist möglich den Webservice gegen fremden Zugriff zu schützen. Hierfür kann ein HTTP-Authentication Header implementiert werden. [42] Es stehen mehrere Methoden zur Auswahl um eine sichere Verbindung mit den Webservice herzustellen. Die bekanntesten sind:

- Basic Authentifizierung über HTTPS
- Session mithilfe von Cookies
- Token (OAuth2)
- Query Authentification

Auch eine Implementierung von HATEOS zum Navigieren zwischen allen und einem GPIO-Anschluss wäre denkbar. Somit würde der Webservice alle Vorgaben des REST Maturity-Models implementieren. Es würde eine Webseite geben, welche alle Anschlüsse anzeigt und eine Webseite, die explizit nur einen Anschluss anzeigt. Hier können zusätzliche Datenfelder für zusätzliche Informationen sorgen. Es wäre möglich, die Zeit anzugeben wie lange ein GPIO-Anschluss den Zustand besitzt oder detaillierte Angaben, was an diesen Anschluss angeschlossen ist.

5. Literaturverzeichnis

[1] Wikipedia – Einplatinencomputer
https://de.wikipedia.org/wiki/Einplatinencomputer
letzter Abruf am 08.03.2016

[2] Daniel Kampert-Raspberry PI – Der praktische Einstig
2., Aktualisierte und erweiterte Auflage 2015

[3] Raspberry PI Foundation – what is a raspberry pi
https://www.raspberrypi.org/help/what-is-a-raspberry-pi/
letzter Abruf am 08.03.2016

[4] Wikipedia – Raspberry Pi
https://de.wikipedia.org/wiki/Raspberry_Pi
letzter Abruf 08.03.2016

[5] Maik Schmidt Raspberry Pi - Einstieg, Optimierung Projekte
1. Auflage 2013, 1. Aktualisierter Nachdruck

[6] Leonard Richardson & Sam Ruby - RESTful Web Services
First Edition, 2007

[7] Wikipedia – Representational State Transfer
https://de.wikipedia.org/wiki/Representational_State_Transfer
letzter Abruf 08.03.2016

[8] Developing RESTful Services with JAX-RS 2.0, WebSockets, and JASON
First Edition 2013

[9] Wiring Pi – About
http://wiringpi.com/
letzter Abruf: 08.03.2016

[10] Wiring Pi – Pins
http://wiringpi.com/pins/
letzter Abruf: 08.03.2016

[11] The Pi4J Project – Index
http://pi4j.com/index.html
letzter Abruf: 08.03.2016

[12] The Pi4J Project – Dependencies
http://pi4j.com/dependency.html
letzter Abruf: 08.03.2016

[13] RPi.GPIO 0.6.1 – Python Package
https://pypi.python.org/pypi/RPi.GPIO
letzter Abruf: 08.03.2016

[14] RPi.GPIO Documentation - module basics – sourceforge
http://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/
letzter Abruf: 08.03.2016

[15] Wikipedia – Java API for RESTful Web Services
https://de.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services
letzter Abruf: 08.03.2016

[16] Jersey – RESTful Web Services in Java
https://jersey.java.net/index.html
letzter Abruf: 08.03.2016

[17] web.py – Welcome to web.py
https://jersey.java.net/index.html
letzter Abruf: 08.03.2016

[18] Wikipedia – Webservices
https://de.wikipedia.org/wiki/Webservice
letzter Abruf: 08.03.2016

[19] Raspberry Pi Foundation – About
https://www.raspberrypi.org/about/
letzter Abruf: 08.03.2016

[20] Raspbiann – Home
https://www.raspbian.org/
letzter Abruf: 08.03.2016

[21] Raspberry Pi Foundation – model-b-plus
https://www.raspberrypi.org/products/model-b-plus/
letzter Abruf: 08.03.2016

[22] Raspberry Pi Model B+ - Datasheet
https://www.adafruit.com/datasheets/pi-specs.pdf
letzter Abruf: 08.03.2016

[23] Wikipeidia – SOAP
https://de.wikipedia.org/wiki/SOAP
letzter Abruf: 08.03.2016

[24] P4J- 8Header-b-plus
http://pi4j.com/images/j8header-b-plus-large.png
letzter Abruf: 08.03.2016

[25] Klaus Dembowski-Raspberry Pi – Das technische Handbuch
2. erweiterte und überarbeitete Auflage 2015

[26] Raspberry Pi Foundation - gpio-plus-and-raspi2
https://www.raspberrypi.org/documentation/usage/gpio-plus-and-raspi2/
letzter Abruf: 08.03.2016

[27] Roy Thomas Fielding – Architectural Styles and the Design of Network-based Software Architectures – Dissertation
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
letzter Abruf: 08.03.2016

[28] Martin Fowler – Richardson Maturity Model
http://martinfowler.com/articles/richardsonMaturityModel.html
letzter Abruf: 08.03.2016

[29] Spring Tool Suite –
https://spring.io/tools
letzter Abruf: 08.03.2016

[30] Apache Software Foundation – Apache Tomcat
http://tomcat.apache.org/index.html
letzter Abruf: 08.03.2016

[31] WiringPi – Download and Install
http://wiringpi.com/download-and-install/
letzter Abruf: 08.03.2016

[32] Raspberry Pi Guide – Raspberry Pi GPIO How-To
http://raspberrypiguide.de/howtos/raspberry-pi-gpio-how-to/
letzter Abruf: 08.03.2016

[33] DigitalOcean – How To Install and Configure Apache Tomcat on a Debian Server
https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-apache-tomcat-on-a-debian-server
letzter Abruf: 08.03.2016

[34] Apache Tomcat 7.0.67 – README
http://mirror.netcologne.de/apache.org/tomcat/tomcat-7/v7.0.67/README.html
letzter Abruf: 08.03.2016

[35] RaspberryPi.org – Oracle Java on Raspberry Pi
https://www.raspberrypi.org/blog/oracle-java-on-raspberry-pi/
letzter Abruf: 08.03.2016

[36] JSON – Einführung in Json
http://www.json.org/json-de.html
letzter Abruf: 08.03.2016

[37] Dusty Phillips – Python 3 Object-oriented Programming
Second Edition August 2015

[38] TechRepublic – An Introduction into SOAP
http://www.techrepublic.com/article/an-introduction-to-the-simple-object-access-protocol-soap/
letzter Abruf: 08.03.2016

[39] W3C – Naming and Adressing
https://www.w3.org/Addressing/
letzter Abruf: 08.03.2016

[39] W3C – HTTP Message
https://www.w3.org/Protocols/rfc2616/rfc2616-sec4.html
letzter Abruf: 08.03.2016

[40] Raspberry Pi – GPIO state after boot
https://www.raspberrypi.org/forums/viewtopic.php?f=44&t=24491
letzter Abruf: 08.03.2016

[41] The PI4J Project – control
http://pi4j.com/example/control.html
letzter Abruf: 08.03.2016

[42] restcookbook – loggingin
http://restcookbook.com/Basics/loggingin/
letzter Abruf: 08.03.2016

6. Anhang

6.1 Webservice

6.1.1 Java Quellcode

Pin:

```
01 package com.raspberry.model;
02
03 import javax.xml.bind.annotation.XmlRootElement;
04
05 @XmlRootElement
06 public class Pin
07     {
08         private int id;
09         private String state;
10         private String gpioPort;
11
12
13         public int getId()
14             {
15             return id;
16             }
17         public void setId(int id)
18             {
19             this.id = id;
20             }
21         public String getState()
22             {
23             return state;
24             }
25         public void setState(String state)
26             {
27             this.state = state;
28             }
29         public String getGpioPort()
30             {
31             return gpioPort;
32             }
33         public void setGpioPort(String gpioPort)
34             {
35             this.gpioPort = gpioPort;
36             }
37
38         public String getIversedState()
39             {
40             if (state.equals("HIGH"))
41                 {
42                 return "LOW";
43                 }
44             else
45                 {
46                 return "HIGH";
47                 }
48             }
49     }
50
```

GpioHolder:

```
01 import java.util.ArrayList;
02 import java.util.List;
03
04 import com.pi4j.io.gpio.GpioController;
05 import com.pi4j.io.gpio.GpioFactory;
06 import com.pi4j.io.gpio.GpioPinDigitalOutput;
07 import com.pi4j.io.gpio.RCMPin;
08
09 /**
10  * Die Klasse GpioHolder enthält eine
11  * Liste mit allen GPIO-Anschlüssen,
12  * sowie Methoden um mit den Anschlüssen
13  * zu Interagieren.
14  *
15  * @Autor Tobias Orth
16  */
17 public class GpioHolder
18     {
19         /*Variablen Dekleration*/
20         private List<Pin> list = new ArrayList<Pin>();
21         private GpioPinDigitalOutput gpiopins[]=new
                                                  GpioPinDigita
                                                  lOutput[28];
22         private static GpioHolder instance;
23
24         /*Konstruktor welche die Anschlüsse initialisiert*/
25         protected GpioHolder()
26             {
27             final GpioController gpio = GpioFactory.
                                           getInstance();
28
29             GpioPinDigitalOutput boardLed0 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_02);
30             GpioPinDigitalOutput boardLed1 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_25);
31             GpioPinDigitalOutput boardLed2 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_06);
32             GpioPinDigitalOutput boardLed3 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_05);
33             GpioPinDigitalOutput boardLed4 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_04);
34             GpioPinDigitalOutput boardLed5 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_03);
35             GpioPinDigitalOutput boardLed6 = gpio.
                                                provisionDigital
                                                OutputPin(
                                                RCMPin.GPIO_29);
36             GpioPinDigitalOutput boardLed7 = gpio.
                                                provisionDigital
```

```
                                          OutputPin(
                                          RCMPin.GPIO_28);
37          GpioPinDigitalOutput boardLed8 = gpio.
                                          provisionDigital
                                          OutputPin(
                                          RCMPin.GPIO_24);
38          GpioPinDigitalOutput boardLed9 = gpio.
                                          provisionDigital
                                          OutputPin(
                                          RCMPin.GPIO_01);
39          GpioPinDigitalOutput boardLed10 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_00)
                                           ;
40          GpioPinDigitalOutput boardLed11 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_27)
                                           ;
41          GpioPinDigitalOutput boardLed12 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_16)
                                           ;
42          GpioPinDigitalOutput boardLed13 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_15)
                                           ;
43          GpioPinDigitalOutput boardLed14 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_26)
                                           ;
44          GpioPinDigitalOutput boardLed15 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_23)
                                           ;
45          GpioPinDigitalOutput boardLed16 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_14)
                                           ;
46          GpioPinDigitalOutput boardLed17 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_12)
                                           ;
47          GpioPinDigitalOutput boardLed18 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_13)
                                           ;
48          GpioPinDigitalOutput boardLed19 = gpio.
                                           provisionDigita
                                           lOutputPin(
                                           RCMPin.GPIO_10)
                                           ;
49          GpioPinDigitalOutput boardLed20 = gpio.
                                           provisionDigita
```

```
                                                lOutputPin(
                                                RCMPin.GPIO_11)
                                                ;
50              GpioPinDigitalOutput boardLed21 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_22)
                                                ;
51              GpioPinDigitalOutput boardLed22 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_21)
                                                ;
52              GpioPinDigitalOutput boardLed23 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_07)
                                                ;
53              GpioPinDigitalOutput boardLed24 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_09)
                                                ;
54              GpioPinDigitalOutput boardLed25 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_08)
                                                ;
55              GpioPinDigitalOutput boardLed26 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_31)
                                                ;
56              GpioPinDigitalOutput boardLed27 = gpio.
                                                provisionDigita
                                                lOutputPin(
                                                RCMPin.GPIO_30)
                                                ;
57
58
59
60              gpiopins[0]  = boardLed0;
61              gpiopins[1]  = boardLed1;
62              gpiopins[2]  = boardLed2;
63              gpiopins[3]  = boardLed3;
64              gpiopins[4]  = boardLed4;
65              gpiopins[5]  = boardLed5;
66              gpiopins[6]  = boardLed6;
67              gpiopins[7]  = boardLed7;
68              gpiopins[8]  = boardLed8;
69              gpiopins[9]  = boardLed9;
70              gpiopins[10] = boardLed10;
71              gpiopins[11] = boardLed11;
72              gpiopins[12] = boardLed12;
73              gpiopins[13] = boardLed13;
74              gpiopins[14] = boardLed14;
75              gpiopins[15] = boardLed15;
76              gpiopins[16] = boardLed16;
77              gpiopins[17] = boardLed17;
78              gpiopins[18] = boardLed18;
79              gpiopins[19] = boardLed19;
80              gpiopins[20] = boardLed20;
```

```
81              gpiopins[21] = boardLed21;
82              gpiopins[22] = boardLed22;
83              gpiopins[23] = boardLed23;
84              gpiopins[24] = boardLed24;
85              gpiopins[25] = boardLed25;
86              gpiopins[26] = boardLed26;
87              gpiopins[27] = boardLed27;
88
89              for (int i = 0; i < gpiopins.length; i++)
90                  {
91
92                  Pin tmpPin = new Pin();
93                  tmpPin.setId(i);
94                  tmpPin.setState(gpiopins[i].getState().
                                    toString());
95                  tmpPin.setGpioPort(gpiopins[i].getPin().
                                       toString());
96                  list.add(tmpPin);
97                  }
98              }
99
100         /*Methode zum Zurückgeben aller Pins*/
101         public List<Pin> getallPins()
102             {
103             return list;
104             }
105
106         public Pin setaPinHIGH(int position)
107             {
108             gpiopins[position].high();
109             Pin tmp= new Pin();
110             tmp.setId(position);
111             tmp.setGpioPort(gpiopins[position].getPin().
                                toString());
112             tmp.setState(gpiopins[position].getState().
                             toString());
113             return tmp;
114             }
115
116         /*Methode welches den Zustand eines Pins ändert (Tog
                                                           gle
                                                           )*/
117         public Pin toggleAPin(Pin aPin)
118             {
119             int position = aPin.getId();
120             gpiopins[position].toggle();
121
122
123             Pin tmp= new Pin();
124             tmp.setId(position);
125             tmp.setGpioPort(gpiopins[position].getPin().
                                toString());
126             tmp.setState(gpiopins[position].getState().
                             toString());
127             return tmp;
128             }
129
130         /*Klasen interne Methode zum Aktualisieren der Pin-
           Liste*/
131         private void refreshList()
132             {
133             list.clear();
```

```
134
135             for (int i = 0; i < gpiopins.length; i++)
136                 {
137                 Pin tmp = new Pin();
138                 tmp.setId(i);
139                 tmp.setState(gpiopins[i].getState().toString(
                                    ));
140                 tmp.setGpioPort(gpiopins[i].getPin().
                                        toString());
141                 list.add(tmp);
142                 }
143
144             }
145
146          /*Methode welche die Liste von Anschlüssen
           zurückliefert*/
147          public GpioPinDigitalOutput[] getGpiopins()
148              {
149              return gpiopins;
150              }
151
152          /*Methode um einen Anschluss expliziet
           auszuschalten*/
153          public Pin setaPinLOW(int boardPosition)
154              {
155              gpiopins[boardPosition].low();
156              Pin tmp= new Pin();
157              tmp.setId(boardPosition);
158              tmp.setGpioPort(gpiopins[boardPosition].getPin().
                                    toString());
159              tmp.setState(gpiopins[boardPosition].getState().
                                toString());
160              return tmp;
161              }
162
163          /*Methode um mehrere Anschlüsse zu schalten (Toggle)
                                                          */
164          public List<Pin> setMultiplePins(List<Pin> pinstoSet)
165              {
166              for (int i = 0; i < pinstoSet.size(); i++)
167                  {
168                  gpiopins[pinstoSet.get(i).getId()].toggle();
169                  }
170              refreshList();
171              return list;
172              }
173
174          /*Die Klasse wird als Singleton erstellt da es nur
           eine
175            * Instanz der Klasse geben kann*/
176          public static GpioHolder getInstance()
177              {
178              if (instance == null)
179                  {
180                  instance = new GpioHolder();
181                  }
182              return instance;
183              }
184      }
```

PinHolderInterface:

```
01 package com.raspberry.repository;
02
03 import java.util.List;
04
05 import com.raspberry.model.Pin;
06
07 public interface PinHolderInterface
08     {
09
10     List<Pin> getAllPins();
11
12     Pin setPin(Pin apin, String order);
13
14     List<Pin> setMultiplePins(List<Pin> pinstoSet);
15
16     }
```

PinHolderStub:

```
01 package com.raspberry.repository;
02
03 import java.util.List;
04
05 import com.raspberry.model.GpioHolder;
06 import com.raspberry.model.Pin;
07
08 public class PinHolderStub implements PinHolderInterface
09     {
10         @Override
11         public List<Pin> getAllPins()
12             {
13             return GpioHolder.getInstance().getallPins();
14             }
15
16         @Override
17         public Pin setPin(Pin apin, String order)
18             {
19             System.out.println("Order = " + order);
20             Pin anewPin;
21             if (order.equals("LOW"))
22                 {
23                 System.out.println("Set Pin " + apin.getId()
24                                    + "LOW");
25                 anewPin= GpioHolder.getInstance().setaPinLOW(
26                          apin.getId());
27                 }
28             else
29                 {
30                 System.out.println("Set Pin " + apin.getId()
31                                    + "HIGH");
32                 anewPin=GpioHolder.getInstance().setaPinHIGH(
33                         apin.getId());
34                 }
35
36             return anewPin;
37             }
```

```
38
39          @Override
40          public List<Pin> setMultiplePins(List<Pin> pinstoSet)
41              {
42              return GpioHolder.getInstance().setMultiplePins(
43                                                  pinstoSet);
44              }
45
46      }
```

6.1.2 Python Quellcode

```
01 #!/usr/bin/env python
02 #Importieren der benötigten Module
03 import web
04 import json
05 import RPi.GPIO as GPIO
06
07 #Festlegen des GPIO-Moduses
08 GPIO.setmode(GPIO.BCM)
09 GPIO.setwarnings(False) # fuer channel 1 & 3
10
11 #Array der GPIO-Anschlüsse die genutzt werden.
12 #Da im Modus BCM die Anschlüsse anderweitig
13 #Nummeriert sind, muss die Liste umgedreht werden
14 pinstorev = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,
               19,20,21,22,23,24,25,26,27]
15 pinstoUse = pinstorev[:]
16 pinstoUse.reverse()
17
18 #Festlegen welche URIs auf welche Funktion reagieren
19 urls = (
20     '/gpio-service/webapi/board', 'list_Pins',
21     '/gpio-service/webapi/board/(.*)', 'get_pin'
22 )
23
24 #Funktion doList(): erstellt eine Aktuelle Übersicht
25 #über den Zustand der GPIO-Anschlüsse. Diese wird
26 #nach jeder Interaktion mit dem Webservice aufgerufen
27 #um eine aktuelle Liste zu erstellen
28 def doList():
29 #global zeigt an das diese Variable global genutzt
30 #werden kann.
31    global pins
32    pins = []
33    currstate = ''
34 #Die Pins werden auf Ihren aktuellen Zustand überprüft
35 #und als Abstrakte Klasse Pin zu dem Array Pins hinzugefügt
36    for i in pinstoUse:
37          #Unterscheide ob der Anschluss High oder Low ist
38          if GPIO.input(pinstoUse[i]) == GPIO.HIGH:
39                currstate = 'HIGH'
```

```
40          else:
41                currstate = 'LOW'
42          port = "GPIO " + str(pinstoUse[i])
43          pins.append(Pin(i,port,currstate))
44    #Nach erfolgreichen erstellen der Liste wird nun
45    #ein json Objekt erstellt
46    output = createMessageAll()
47    return output
48
49 #Erstellt ein json Objekt und liefert dieses zurück
50 def createMessageAll():
51    output = json.dumps([pin.__dict__ for pin in pins],
             sort_keys=True)
52    return output
53
54 #Die Klasse Pin stellt ein abstraktes Modell der
55 #GPIO-Anschlüsse da. Diese enthalten eine ID,
56 #den Zustand sowie der tatsächliche GPIO-Pin
57 #an dem dieser Angeschlossen ist.
58 class Pin(object):
59
60    #Konstruktor
61     def __init__(self, id, gpioPort, state):
62         self.id=id
63         self.gpioPort=gpioPort
64         self.state=state
65     #Getter Methoden für die einzelnen Attribute
66     def getID(self):
67      return self.id
68
69     def getState(self):
70      return self.state
71
72     def getgpioport(self):
73      return self.gpioPort
74
75 #Die Klasse list_Pins wird aufgerufen wenn die URI
76 #/gpio-service/webapi/board genutzt wird. Diese stellt
77 #sowohl eine GET-, sowie eine POST-Methode zur Verfügung.
78 #Es wird eine Interaktion mit n Pins erwartet.
79 class list_Pins:
80     #Die GET-Funktion liefert die Aktuelle Übersicht
81     #der GPIO-Anschlüsse zurück
82     def GET(self):
83      output = json.dumps([pin.__dict__ for pin in pins],
                sort_keys=True)
84      return output
85
86     #POST-Funktion nimmt eine Liste der Anschlüsse
87     #im Format json entgegen und liest diese aus.
88     #Der Zustand der übertragenen Pins wird gewechselt.
89     def POST(self):
90     #Daten entgegennehmen
91      data = web.data()
92
93      #json Daten de-crypten
94      jsonobj = json.loads(data)
95
96      #Es wird nun ein leeres Array erstellt welches mit
97      #Objekten vom Typ Pin beladen werden.
98      switchpinlist = []
99      iter = len(jsonobj)
```

```
100
101     #Durch die json Daten Iterieren
102     for i in range (0,iter):
103
104      #Erstellen eines neuen Pin Onjekts anhand der Daten.
105      switchpinlist.append(Pin(jsonobj[i]['id'], jsonobj[i][
                            'gpioPort'], jsonobj[i]['state']))
106
107     #ID der übertragenen Pins auslesen und den
108     #Zustand des Entsprechenden GPIO-Anschlusses
109     #wechseln. Um den korrekten Pin zu adressieren
110     # wird die Länge des Arrays minus der gewünschten
111     #id und minus eins gerechnet. Das ergibt für die ID 5
112     # folgende Rechnung: 28 - 5 - 1 = 22. Somit wird Port
113     # 22 geschaltet.
114     listiter = len(switchpinlist)
115     for x in range(0,listiter):
116      index = switchpinlist[x].getID()
117      currstate = pins[len(pins)-index -1].getState()
118      if currstate is 'HIGH':
119               GPIO.output(len(pins)-index -1, 0)
120      else:
121               GPIO.output(len(pins)-index -1, 1)
122
123     #Erstellen und zurückliefen einer Aktuellen Übersicht
124     output=doList()
125     return output
126
127 #Die Klasse get_pin wird verwendet wenn eine einzelner GPIO-
    Anschluss
128 #betrachtet wird. Die genutzte URI ist board/(*)
129 class get_pin:
130 #Die GET-Funktion liefert eine Übersicht über einen
131 #bestimmten GPIO-Anschluss
132  def GET(self, pinid):
133   index=int(pinid)
134   pin = pins[len(pins)-index -1]
135   output = json.dumps(pin.__dict__, sort_keys=True)
136   return output
137
138   #Die entsprechende POST-Funktion nimmt keine json Daten
139   # entgegen da hier nur ein Anschluss angesprochen werden kann.
      #Die ID fuer den Entsprechenden Anschluss wird der
140   #URL entnommen.
141  def POST(self,pin):
142         index=int(pin)
143         currstate = pins[len(pins)-index -1].getState()
144         if currstate is 'HIGH':
145               GPIO.output(len(pins)-index -1, 0)
146         else:
147               GPIO.output(len(pins)-index -1, 1)
148         output=doList()
149         return output
150
151 #OnStartUp uebernimmt Aufgaben die bei einem ersten Aufruf
152 #der Webservice erledigt werden sollen. Hier werden die GPIO-
153 #Anschluesse initialisiert und zugewiessen.
154 def onStartup():
155   global pins
156   pins = []
157   for i in pinstoUse:
158         GPIO.setup(pinstoUse[i], GPIO.OUT, initial=GPIO.LOW)
```

```
159         if GPIO.input(pinstoUse[i]) == GPIO.HIGH:
160               currstate = 'HIGH'
161         else:
162               currstate = 'LOW'
163         port = "GPIO " + str(pinstoUse[i])
164         pins.append(Pin(i,port,currstate))
165
166 onStartup()
167
168 #Starten der Applikation
169 if __name__ == "__main__":
170     app = web.application(urls, globals())
171     app.run()
```

6.2 Client Webseite

6.2.1 HTML

Index.html

```
01 <!DOCTYPE html>
02 <html lang="de">
03  <head>
04     <style>
05                     table, td, th
06                     {
07                             border: 1px solid;
08                     }
09
10                     thead
11                     {
12                             float: left;
13                     }
14
15
16                     thead th
17                     {
18                             display: block;
19                     }
20
21                     tbody
22                     {
23                             float: left;
24                     }
25
26                     tbody td
27                     {
28                     display: block;
29                     }
30    </style>
31    <meta charset="utf-8" />
32    <title>REST-GPIO-Client</title>
33    <script src="http://code.jquery.com/jquery-latest.js"><
                  /script>
34    <script src="jquerystart.js"></script>
35    <script type="text/javascript">
36
```

```
37              function handleAll()
38              {
39                      mainbox = document.getElementById("mainbox");
40                      checkboxes =
                             document.getElementsByName('check_list');
41
42                      if(mainbox.checked)
43                      {
44                              for (i = 0; i < checkboxes.length; i++)
45                              {
46                                      checkboxes[i].checked = true;
47                              }
48                      }
49                      else
50                      {
51                              for (i = 0; i < checkboxes.length; i++)
52                              {
53                                      checkboxes[i].checked = false;
54                              }
55                      }
56              }
57
58              function Send(chk)
59              {
60                      var checkboxes = new Array();
61                      var checked = new Array();
62                      checkboxes =
                          document.getElementsByName('check_list');
63                      for (i = 0; i < checkboxes.length; i++)
64                      {
65                              if(checkboxes[i].checked == true)
66                              {
67                                      var PinObject = new Object();
68                                      PinObject.id = checkboxes[i].id;
69                                      PinObject.gpioPort = "somePort";
70                                      PinObject.state = "someState";
71
72                                      checked.push(PinObject);
73                              }
74                      }
75                      console.log("Es wurden "+ checked.length + "
                                    ausgewählt")
76                      if(checked.length > 0)
77                      {
78                              var jsonData = JSON.stringify(checked);
79
80                              $.ajax
81                              ({
82                                      headers:
83                                      {
84                                              'Accept':
                                                    'application/json',
85                                              'Content-Type':
                                                     'application/json'
86                                      },
87                                      type: "POST",
88                                      url:
                                        'http://192.168.178.57:8080/gpio-
                                        service/webapi/board/',
89                                      dataType: 'json',
90                                      data: jsonData,
91                                      success: function (data)
```

```
92                                          {
93                                                  console.log("Antwort von
                                                  POST länge; " + data.length);
94                                                  console.log(data);
95
96
        tbl=document.getElementById('gpio_table');
97
98                                                  while(tbl.rows.length>1)
99                                                  {
100                                                         tbl.deleteRow(1);
101                                                 }
102
103                                                 for (var i = 0, len =
                                                     data.length; i < len; i++)
104                                                 {
105                                                         var item = data[i];
106                                                         var tablerow ='';
107                                                         tablerow +=
                        '<tbody><tr><td align="center">' + item.
                        gpioPort + '</td><td align="center">' +
                        item.id +'</td><td align="center">' +
                        item.state +'</td><td align="center"><
                        input type="checkbox" id= ' + item.id +'
                        value = '+i+' name="check_list"/></td><
                        /tr></tbody>';
108
        $('#gpio_table').append(tablerow);
109
        console.log(data[i]);
110                                                 }
111                                                 mainbox =
                                            document.getElementById("mainbox");
112                                                 mainbox.checked = false;
113                                         }
114                                 })
115                         }
116                         else
117                         {
118                                 alert("Es wurden keine Anschlüsse
                                        ausgewählt!!");
119                         }
120             }
121
122     </script>
123   </head>
124   <body>
125     <h1>GPIO-Administration</h1>
126     <label> <input id ="mainbox" type="checkbox"
                        name="checkall" value="Alle Anharken"
                        onchange="handleAll()"> Alle Anharken <
                        /label>
127     <table id="gpio_table" name="gpiotable">
128             <thead>
129             <tr>
130                     <th>Port</th>
131                     <th>ID</th>
132                     <th>Zustand</th>
133                     <th>Box</th>
134             </tr>
135             </thead>
136     </table>
```

```
137     <br></br>
138     <input id="Klickmich" type="button" value="Zustand ändern"
              onclick="Send('gpio_table')" />
139  </body>
140 </html>
```

6.2.2 JavaScript

Jquersstart.js

```
01 $(document).ready(function() {
02
03    var JSONService = "http://192.168.178.57:8080/gpio-
                        service/webapi/board/";
04    $.ajax({
05          type: "GET",
06          contentType: "application/json",
07          dataType: "json",
08          url: JSONService
09     })
10    .done(function (response)
11    {
12          for(var i =0;i < response.length;i++)
13          {
14                var item = response[i];
15                var tablerow ='';
16                tablerow += '<tbody><tr><td align="center">' + item.
                  gpioPort + '</td><td align="center">' +item.
                  id +'</td><td align="center">' + item.state +
                  '</td><td align="center"><input
                  type="checkbox" id= ' + item.id +' value = '+
                  i+' name="check list"/></td></tr></tbody>';
17                $('#gpio_table').append(tablerow);
18          }
19
20
21       })
22
23 });
```