

Studienarbeit

Mobiles Frontend & Backend für Smart Lights

**Hanne Nobis**

Matrikel-Nr. 7543235

Kurs TINF11B

DHBW Stuttgart

**Michael Strobel**

Matrikel-Nr. 4135863

Kurs TINF11B

DHBW Stuttgart

Betreuer: B.Sc. Sebastian Bejga

Zeitraum: 23. Oktober 2013 – 9. Juni 2014

# Ehrenwörtliche Erklärung

Wir erklären hiermit ehrenwörtlich, dass wir die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt haben. Aus den benutzten Quellen, direkt oder indirekt, übernommene Gedanken haben wir als solche kenntlich gemacht.

Diese Arbeit wurde bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Stuttgart, den 6. Juni 2014

Michael Strobel

Stuttgart, den 6. Juni 2014

Hanne Nobis

# Kurzbeschreibung

Diese Studienarbeit dokumentiert die Konzeption und Entwicklung eines Systems zur Steuerung und Automatisierung der Philips Hue-Leuchten. Nach einer detaillierten Beschreibung der Aufgabenstellung werden die drei wesentlichen Bereiche der Umsetzung beschrieben: Die Auswahl und Inbetriebnahme der benötigten Hardware, der Entwicklung eines Backends zur Kontrolle der Automatisierung sowie das Design und die Implementierung eines Frontends für mobile Geräte, das die Funktionalität des Backends für den Benutzer zugänglich macht. Die Arbeit schließt mit einem Ausblick auf eventuelle Weiterentwicklungen sowie einem Fazit über die getroffenen Entscheidungen sowie Erfahrungen, die aus dem Projekt gewonnen werden konnten.

Inhaltsverzeichnis

[Ehrenwörtliche Erklärung 2](#_Toc389857404)

[Kurzbeschreibung 3](#_Toc389857405)

[Abbildungsverzeichnis 7](#_Toc389857406)

[Tabellenverzeichnis 9](#_Toc389857407)

[Abkürzungsverzeichnis 10](#_Toc389857408)

[1 Aufgabenstellung 12](#_Toc389857409)

[2 Hardware 13](#_Toc389857410)

[2.1 Auswahl der Komponenten 13](#_Toc389857411)

[2.1.1 Raspberry Pi und Peripherie 13](#_Toc389857412)

[2.1.2 Arduino und Peripherie 14](#_Toc389857413)

[2.2 Zusammenbau und Installation 15](#_Toc389857414)

[2.2.1 Philips Hue-Leuchten 15](#_Toc389857415)

[2.2.2 Aufsetzen des Raspberry Pi 16](#_Toc389857416)

[2.2.3 Zusammenbau des Arduino 18](#_Toc389857417)

[3 Entwicklung des Backends 19](#_Toc389857418)

[3.1 Funktionsumfang 19](#_Toc389857419)

[3.1.1 Funktionen der Hue-Bridge 19](#_Toc389857420)

[3.1.2 Favoriten und Szenen 20](#_Toc389857421)

[3.1.3 Automatisierung 20](#_Toc389857422)

[3.1.4 Party-Modus 21](#_Toc389857423)

[3.1.5 Weitere Funktionen des Backends 22](#_Toc389857424)

[3.1.6 Funktionen des Arduino 22](#_Toc389857425)

[3.2 Auswahl der Software 22](#_Toc389857426)

[3.2.1 Grundbausteine 23](#_Toc389857427)

[3.2.2 Module und Erweiterungen 24](#_Toc389857428)

[3.2.3 Zusätzliche Software 25](#_Toc389857429)

[3.3 Architektur 25](#_Toc389857430)

[3.4 Implementierung 26](#_Toc389857431)

[3.4.1 Grundstruktur 26](#_Toc389857432)

[3.4.2 Socket.IO-Verbindung 27](#_Toc389857433)

[3.4.3 Netzwerk-Geräte-Erkennung 29](#_Toc389857434)

[3.4.4 Sprachsteuerung 30](#_Toc389857435)

[3.4.5 Implementierung des Arduino 31](#_Toc389857436)

[3.5 Installation 33](#_Toc389857437)

[3.5.1 Einrichten der Entwicklungsumgebung 33](#_Toc389857438)

[3.5.2 Deployment auf dem Raspberry Pi 34](#_Toc389857439)

[4 Entwicklung des Frontends 36](#_Toc389857440)

[4.1 Softwareentscheidungen 36](#_Toc389857441)

[4.1.1 WebApp 36](#_Toc389857442)

[4.1.2 jQuery Mobile 37](#_Toc389857443)

[4.1.3 AngularJs 38](#_Toc389857444)

[4.2 Layout und Design 39](#_Toc389857445)

[4.2.1 Responsive Webdesign 39](#_Toc389857446)

[4.2.2 Mobile First 40](#_Toc389857447)

[4.2.3 Layoutkonzept 41](#_Toc389857448)

[4.2.4 Designkonzept 42](#_Toc389857449)

[4.3 Implementierung 43](#_Toc389857450)

[4.3.1 Javascript-Frameworks 43](#_Toc389857451)

[4.3.2 Grundstruktur 44](#_Toc389857452)

[4.3.3 Implementierungskonzept 49](#_Toc389857453)

[5 Ausblick 57](#_Toc389857454)

[6 Fazit 58](#_Toc389857455)

[Quellenverzeichnis 60](#_Toc389857456)

[Anhang A: Architektur-Übersicht 63](#_Toc389857457)

[Anhang B: Hardware 64](#_Toc389857458)

[Anhang C: Raspberry Pi Setup-Script 68](#_Toc389857459)

[Anhang D: Frontend 72](#_Toc389857460)

# Abbildungsverzeichnis

[Abbildung 1: Raspberry Pi Soundkarten-Konfiguration 17](#_Toc389857461)

[Abbildung 2: Raspberry Pi WLAN-Konfiguration 17](#_Toc389857462)

[Abbildung 3: Backend-Code zum Einbinden der Controller 27](#_Toc389857463)

[Abbildung 4: Grundaufbau für Backend-Controller am Beispiel der Szenenverwaltung 27](#_Toc389857464)

[Abbildung 5: Event-Listener im Backend 27](#_Toc389857465)

[Abbildung 6: Dummy-Socket-Implementierung im REST-Controller 29](#_Toc389857466)

[Abbildung 7: Samples und Frames bei der Arduino-Takterkennung 32](#_Toc389857467)

[Abbildung 8: Installation der Entwicklungsumgebung unter Linux 33](#_Toc389857468)

[Abbildung 9: Aufrufen des Installations-Scripts für den Raspberry Pi 34](#_Toc389857469)

[Abbildung 10: Erstentwürfe für Smartphone und Tablet 41](#_Toc389857470)

[Abbildung 11: Erstentwurf >1100px Breite 42](#_Toc389857471)

[Abbildung 12: Veranschaulichung des Designkonzepts anhand der light.html 43](#_Toc389857472)

[Abbildung 13: Einbinden des Logout Icon 44](#_Toc389857473)

[Abbildung 14:Navigationsmenü für Tablet und Smartphone 45](#_Toc389857474)

[Abbildung 15: CSS-Klasse in Abhängigkeit von der Bildschirmgröße 45](#_Toc389857475)

[Abbildung 16: CSS-Klasse des inhaltumfassenden Containers 46](#_Toc389857476)

[Abbildung 17: CSS-Klasse des Navigationscontainers 46](#_Toc389857477)

[Abbildung 18:CSS-Klasse des internen Inhaltscontainers 46](#_Toc389857478)

[Abbildung 19: Containereinteilung im Desktopmodus 47](#_Toc389857479)

[Abbildung 20: CSS-Klassen der Untercontainer A und D 47](#_Toc389857480)

[Abbildung 21: CSS-Klassen Untercontainer B und C im Vergleich Smartphone und normale Ansicht 48](#_Toc389857481)

[Abbildung 22: Containereinteilung im für Tablet und Smartphone 49](#_Toc389857482)

[Abbildung 23: Startseite für Tablet und Smartphone 50](#_Toc389857483)

[Abbildung 24: Implementierung des IndexCtrl-Controllers 50](#_Toc389857484)

[Abbildung 25: Implementierung eines ng-repeat 51](#_Toc389857485)

[Abbildung 26: remove-Funktion aus dem SceneCtrl 52](#_Toc389857486)

[Abbildung 27: Implementierung der Submenu Funktionen 52](#_Toc389857487)

[Abbildung 28: Einsatz eines Submenu 52](#_Toc389857488)

[Abbildung 29: Vergleich Tablet und Smartphone Submenu 53](#_Toc389857489)

[Abbildung 30: verschiedene Submenu Implementierungen 53](#_Toc389857490)

[Abbildung 31: Farbauswahl Direktive 54](#_Toc389857491)

[Abbildung 32: Aufruf der Farbauswahl-Direktiven 54](#_Toc389857492)

[Abbildung 33: Auszug aus der hueperCheckbox-Direktive 55](#_Toc389857493)

# Tabellenverzeichnis

[Tabelle 1: Mögliche Optionen der Automatisierung 21](#_Toc389857494)

[Tabelle 2: Zentrale npm-Module des Backends 24](#_Toc389857495)

# Abkürzungsverzeichnis

**API** Application Programming Interface

**ARP** Address Resolution Protocol

**CRUD** Create, Read, Update, Delete

**CSS** Cascading Style Sheets

**DHCP** Dynamic Host Configuration Protocol

**DMX** Digital Multiplex

**GPIO** General Purpose Input/Output

**GPU** Graphics Processing Unit

**HDMI** High-Definition Multimedia Interface

**HTML** Hypertext Markup Language

**I²C** Inter-Integrated Circuit

**IDE** Integrated Development Environment

**IP** Internet Protocol

**JSON** JavaScript Object Notation

**LAN** Local Area Network

**MAC** Media Access Control

**NFC** Near Field Communication

**NPM** Node Package Manager

**RAM** Random Access Memory

**REST** Representational State Transfer

**RFID** Radio Frequency Identification

**SD** Secure Digital

**SPI** Serial Peripheral Interface

**SQL** Structured Query Language

**TCP** Transmission Control Protocol

**USB** Universal Serial Bus

**UI** User Interface

**WLAN** Wireless Local Area Network

# Aufgabenstellung

Die Philips Hue-Leuchten[[1]](#footnote-1) bieten die Möglichkeit, ihre Farbe sowie die Helligkeit ferngesteuert zu ändern. Standardmäßig ist dafür jeweils eine App für Android- und iOS-Geräte verfügbar. Für Entwickler wird eine REST-API[[2]](#footnote-2) zur Verfügung gestellt, um die Leuchten mit eigenen Anwendungen über das Netzwerk steuern zu können und sie so auch um weitere Funktionalitäten zu erweitern.

Aufbauend auf dieser REST-API soll nun ein System entwickelt werden, das neben einer plattformunabhängigen Oberfläche für mobile Geräte die Leuchten um weitere intelligente Funktionen und Automatisierungen erweitert.

Zum effizienten dauerhaften Betrieb eines solchen Systems ist eine Hardware-Plattform auf Basis eines Mikrocomputers nötig. Dieser erlaubt durch das Einbinden in das lokale Netzwerk sowie den Anschluss von Peripheriegeräten etwa das Messen von Umwelt-Sensoren, Lesen von Chipkarten oder Erkennen von im Netzwerk angemeldeten Geräten.

Um die Hardware nutzen zu können, automatisierte Abläufe zu ermöglichen und eine Verbindung aller beteiligten Peripheriegeräte sowohl mit den Leuchten als auch mit der Benutzer-Oberfläche herzustellen, muss ein Backend-System entwickelt werden, welches als Server auf dem Mikrocomputer läuft. Einstellungen und Regeln für automatisierte Abläufe werden in einer Datenbank gespeichert. Die Funktionalität der Hue-REST-API sowie die erweiterten Funktionen der Hardware-Plattform werden als Interfaces für Benutzer-Oberflächen oder andere Programme zur Verfügung gestellt.

Die Benutzer-Steuerung des Backends soll über eine web-basierte Oberfläche erfolgen, die für mobile Geräte optimiert ist. Durch die Implementierung als Web-Applikation wird eine Plattformunabhängigkeit erreicht, wodurch die Benutzung der Oberfläche im Gegensatz zur mitgelieferten App auch auf Geräten mit anderen Betriebssystemen oder vom Computer aus möglich gemacht wird.

# Hardware

Das folgende Kapitel behandelt die Auswahl der benötigten Hardware sowie deren Zusammenbau und die Grundinstallation des Raspberry Pi Mikrocomputers.

## Auswahl der Komponenten

Die Grundlage für das System bilden die Philips Hue-Leuchten. Diese sind als Starter-Pack für etwa 200€ verfügbar. Dieses enthält neben drei Leuchten die sogenannte „Bridge“, die per LAN mit dem lokalen Netzwerk verbunden wird. Die Leuchten werden über die Bridge gesteuert, welche empfangene Befehle per Funk an die betroffenen Leuchten überträgt.

Nachfolgend sind die Hardware-Komponenten in zwei Bereiche aufgeteilt: Den Raspberry Pi-Mikrocomputer für den Betrieb des Backends sowie die Arduino-Mikroprozessor-Plattform für den Anschluss der Umwelt-Sensoren und des RFID/NFC-Lesegeräts.

Die komplette Hardware-Liste inklusive Links zu Online-Shop-Angeboten befindet sich in Anhang B: Hardware (S. 64). Eine Gesamt-Architekturübersicht zum Zusammenspiel von Hardware- und Software-Komponenten ist in Anhang A: Architektur-Übersicht (S. 63) enthalten.

### Raspberry Pi und Peripherie

Der Raspberry Pi[[3]](#footnote-3) ist ein Mikrocomputer auf ARM-Basis. Er ist mit 700 MHz getaktet und verfügt über 512MB RAM. Als Betriebssystem sind verschiedene speziell angepasste Linux-Distributionen verfügbar.

Für diese Studienarbeit wurde er aus folgenden Gründen ausgewählt:

* Er kostet nur etwa 40€ und gehört damit zu den günstigsten Mikrocomputern
* Durch seine Verbreitung und die große Community gibt es eine umfangreiche Auswahl an nativ unterstützter Software oder Portierungen

Der Raspberry Pi soll folgende Aufgaben übernehmen:

* Betrieb des Backends und der Datenbank
* Hosting des Frontends
* Verbindung mit den anderen Hardware-Komponenten sowie den Hue-Leuchten

Folgende zusätzliche Peripherie wurde für den Raspberry Pi ausgewählt:

* Eine USB-Soundkarte sowie ein Mikrofon für die Sprachsteuerung
* Ein USB-WLAN-Stick, um den Raspberry Pi auch in drahtlose Netzwerke einbinden zu können
* Ein aktiver USB-Hub, um alle Peripheriegeräte und eine Tastatur anschließen zu können sowie die ausreichende Stromversorgung aller Komponenten zu gewährleisten
* Eine 8GB SD-Karte, die als Festplatte fungiert
* Ein Netzteil
* Ein Gehäuse

Da der Raspberry Pi nur über digitale GPIO-Ports zum Anschluss von Sensoren verfügt, ist es ohne zusätzliche Elektronik oder weitere Peripherie nicht möglich, analoge Sensoren wie etwa einen Helligkeits-Sensor direkt anzuschließen. Daher wurde zusätzlich eine Mikroprozessor-Plattform ausgesucht, die für diesen Zweck besser geeignet ist.

### Arduino und Peripherie

Eine Plattform, die das Anschließen verschiedener Arten von Sensoren und Erweiterungen ermöglicht, ist der Arduino[[4]](#footnote-4). Er besitzt sowohl analoge als auch digitale Eingänge und unterstützt die Bus-Systeme I²C und SPI zur Kommunikation mit weiteren Peripheriegeräten. Durch seine Bauform erlaubt er das Aufstecken von Erweiterungskarten (sogenannten „Shields“), die seine Funktionalität erweitern oder noch andere Anschlüsse bereitstellen. In den meisten Fällen sind diese Shields so aufgebaut, dass mehrere von ihnen aufeinander gesteckt werden können und die unbenutzten Pins an die darüber liegenden durchschleifen. Für den Arduino ist ein umfangreiches Angebot an Erweiterungen und Sensoren verfügbar, was der Hauptgrund der Entscheidung für diese Plattform war. Programmiert werden kann er über eine eigene IDE in der Sprache C.

Für das Projekt wurde der Arduino Uno ausgewählt. Dieser kann über USB mit einem Computer bzw. dem Raspberry Pi verbunden werden und über einen simulierten seriellen Port Nachrichten mit ihm austauschen.

Folgende Shields wurden für das Projekt verwendet:

* Das Adafruit RFID/NFC Shield zum Erkennen von Chipkarten[[5]](#footnote-5)
* Das Seeedstudio Grove Base Shield zum Anschließen von Sensoren[[6]](#footnote-6)

Für das Seeedstudio Grove Base Shield wurden die folgenden Sensoren ausgewählt:

* Ein Licht-Sensor zum Automatisieren der Leuchten nach der Umgebungshelligkeit
* Ein Bewegungsmelder
* Ein Sound-Sensor, der zur Musik-Takterkennung genutzt werden soll

Zusätzlich wurden folgende Komponenten benötigt:

* Ein USB-A auf USB-B-Adapterkabel zum Verbinden des Arduino mit dem Raspberry Pi
* 4-Pin-Verbinderkabel zum Anschließen der Sensoren an das Base Shield
* Ein Stackable Header Kit, um das Base Shield auf das RFID/NFC Shield montieren zu können
* RFID-Tags zum Testen des RFID/NFC Shields

## Zusammenbau und Installation

Nachfolgend wird der Zusammenbau der Hardware sowie die Grund-Installation und Einrichtung des Raspberry Pi beschrieben.

### Philips Hue-Leuchten

Das Herzstück des Philips Hue Starter-Sets ist die Bridge zur Steuerung der Leuchten. Diese muss neben dem Netzanschluss per LAN mit dem lokalen Netzwerk verbunden werden. Eine Bridge unterstützt bis zu 50 Leuchten.

Die Philips Hue-Leuchten besitzen ein normales E27-Gewinde und können dadurch einfach in handelsübliche Fassungen geschraubt werden. Beachtet werden muss lediglich die Funk-Reichweite der Bridge. Diese ist vor allem abhängig von Wänden und anderen Hindernissen zwischen ihr und den Leuchten. Die Reichweite kann durch die Leuchten selbst erhöht werden, da sie als Repeater fungieren und das Signal an Leuchten außer Reichweite der Bridge weiterleiten.

Die Funktionsfähigkeit der Installation kann mit der offiziellen Hue-App auf einem iPhone oder Android-Gerät getestet werden. Dafür muss sich das Gerät im selben Netzwerk wie die Bridge befinden, was etwa durch die Verwendung eines WLAN-Routers erreicht werden kann. Zur Benutzer-Authentifizierung muss der sogenannte „Link-Button“ auf der Bridge gedrückt werden. Dieser erlaubt für 30 Sekunden die Registrierung neuer Anwendungen.

### Aufsetzen des Raspberry Pi

Das Aufsetzen des Raspberry Pi-Mikrocomputers umfasst den Zusammenbau, die Installation des Betriebssystems, die Grundeinstellungen sowie das Einrichten der angeschlossenen Peripheriegeräte.

#### Zusammenbau

Folgende Schritte sind zum Zusammenbau des Raspberry Pi nötig:

* Einbau des Raspberry Pi in das Gehäuse
* Anschluss eines Bildschirms per HDMI oder Composite
* Verbinden mit dem USB Hub
* Anschluss der Soundkarte mit Mikrofon, sowie einer Tastatur an den USB-Hub. Nach seinem Zusammenbau wird der Arduino ebenfalls an den USB Hub angeschlossen.
* Bei Verwendung eines drahtlosen Netzwerks wird der USB WLAN-Stick ebenfalls an den USB Hub angeschlossen
* Anschluss der Netzteile an den Raspberry Pi und den USB Hub

#### Installation

Als Festplatten-Medium für den Raspberry Pi dient die SD-Karte. Diese muss vor dem Einbau formatiert und mit einem Betriebssystem bespielt werden. Auf der offiziellen Website stehen mehrere Images für verschiedene speziell angepasste Betriebssysteme bereit. Eine spezielle Variante ist das NOOBS-System (New Out Of Box Software). Es enthält Images verschiedener Betriebssysteme, die dann auf dem Raspberry Pi selbst ausgewählt und installiert werden können. Außerdem erlaubt es eine schnelle Neuinstallation des Betriebssystems, ohne die SD-Karte ausbauen und neu bespielen zu müssen, was vor allem beim Testen von Installations-Prozeduren von großem Vorteil ist.

Das Image wird als ZIP-Datei heruntergeladen und muss einfach auf die SD-Karte entpackt werden. Danach kann diese in den Raspberry Pi eingesetzt werden.

Um ihn zu starten, müssen er sowie der USB Hub einfach mit dem Strom verbunden werden. Je nach Bildschirm-Anschluss muss auf der Tastatur eine der Tasten 1-4 gedrückt werden, um ein Signal zu erhalten.

Nun erscheint eine Betriebssystem-Auswahl. Nach dem Umstellen der Sprache und des Tastatur-Layouts auf Deutsch wurde für diese Studienarbeit Raspbian als Betriebssystem ausgewählt, welches auf der Debian-Linux-Distribution basiert.

Nach dem Neustart nach der erfolgreichen Installation öffnet sich das Programm raspi-config, in dem Grundeinstellungen vorgenommen werden können. Unter „Advanced Options“ wurden folgende Optionen verändert:

* Der Hostname, über den der Raspberry im Netzwerk erreichbar ist; standardmäßig lautet er „raspberrypi“
* Der SSH-Server muss aktiviert werden, um einen Remote-Zugriff über das Netzwerk zu ermöglichen
* Bei der „Split Memory“-Einstellung kann der Anteil des Speichers für die GPU auf 16MB reduziert werden, da keine grafische Desktop-Umgebung benötigt wird

Der Standard-Benutzer ist „pi“ mit dem Passwort „raspberry“, was ebenfalls im raspi-config-Programm geändert werden kann.

#### Einrichten der Peripherie

Damit der Raspberry Pi die externe Soundkarte verwendet, muss die Datei */home/pi/.asoundrc* mit folgendem Inhalt angelegt werden:

pcm.!default {

type hw

card 1

}

ctl.!default {

type hw

card 1

}

Abbildung 1: Raspberry Pi Soundkarten-Konfiguration

Bei der Verwendung eines USB-WLAN-Sticks muss das Drahtlos-Netzwerk in die Datei */etc/wpa\_supplicant/wpa\_supplicant.conf* eingetragen werden:

network={

ssid="<SSID>"

psk="<PASSWORT>"

}

Abbildung 2: Raspberry Pi WLAN-Konfiguration

Um die Einstellung zu übernehmen, muss der Raspberry Pi neugestartet werden.

Der Raspberry Pi kann auch so konfiguriert werden, dass die Philips Hue-Bridge an dessen LAN-Port angeschlossen werden kann und durch eine Verbindungs-Überbrückung dessen USB-WLAN-Stick benutzen kann, um sich mit dem Internet zu verbinden[[7]](#footnote-7). Eine ausführliche Beschreibung ist in der Setup-Anleitung im Git-Repository enthalten (*docs/setup/raspberry.md)*.

### Zusammenbau des Arduino

Die meisten Komponenten für den Arduino werden schon fertig zusammengebaut geliefert. Lediglich das Adafruit RFIF/NFC-Shield verfügt noch nicht über Anschluss-Pins. Die mitgelieferten Pins erlauben kein Aufstecken eines weiteren Shields, deshalb muss hier stattdessen das Arduino Stackable Header Kit verwendet werden. Dieses wird in die dafür vorgesehenen Bohrungen gesteckt und unten angelötet.

Nun kann das RFID/NFC-Shield auf den Arduino aufgesteckt werden. Auf das RFID/NFC-Shield wiederum wird das Seeedstudio Grove Base Shield aufgesteckt.

Die Sensoren wurden per 4-Pin-Verbinderkabel mit den folgenden Anschlüssen des Base Shields verbunden:

* Der Licht-Sensor an A0
* Der Sound-Sensor an A2
* Der Bewegungsmelder an D8

# Entwicklung des Backends

Das Backend des Projekts stellt das logische Bindeglied zwischen den Philips Hue Leuchten, dem Frontend sowie allen Peripheriegeräten und Sensoren dar. Es kümmert sich um die Steuerung der Leuchten, das Sammeln, Bereitstellen und Speichern von Daten sowie die Durchführung automatisierter Abläufe.

Im folgenden Kapitel wird zuerst auf den Funktionsumfang des Backends, die Auswahl der verwendeten Software-Komponenten sowie auf die gewählte Architektur eingegangen. Nach einer detaillierten Beschreibung verschiedener Aspekte der Implementierung folgt noch die Einrichtung der Entwicklungsumgebung, um das Projekt auf einem normalen Computer installieren und Änderungen vornehmen zu können.

Der Code und die vollständige Entwickler-Dokumentation der Studienarbeit sind auf GitHub verfügbar.[[8]](#footnote-8)

## Funktionsumfang

Nachfolgend wird ein Überblick über den Funktionsumfang des Backends gegeben. Beginnend mit der bereits bestehenden Funktionalität der Philips Hue-Bridge wird auf weitere Funktionsbereiche sowie die Funktionalität des Arduino eingegangen.

### Funktionen der Hue-Bridge

Die REST-API der Philips Hue-Bridge umfasst folgende Funktionen[[9]](#footnote-9):

* Auslesen der Konfiguration und des aktuellen Zustands der Leuchten
* Steuern und Verwalten einzelner Leuchten
* Steuern und Verwalten von Gruppen aus mehreren Leuchten
* Zeitgesteuerte Kommandos, die einmalig ausgeführt werden
* Benutzer-Registrierung und –Verwaltung
* Ändern der Konfiguration
* Finden der Bridge im lokalen Netzwerk über einen Internet-Portal-Service

Im Wesentlichen soll der Zugriff auf diese Funktionen auch vom Backend bereitgestellt werden, sodass sie über das Frontend genutzt werden können. Auf die Übernahme der zeitgesteuerten Kommandos wurde verzichtet, da diese Funktionalität durch die Automatisierungs-Funktionen abgedeckt wird. Auch das Ändern der Bridge-Konfiguration wurde nicht vollständig übernommen, sondern nur einzelne Funktionen wie das software-seitige Drücken des Link-Buttons und das Aktualisieren der Firmware wurden implementiert.

### Favoriten und Szenen

Die Philips Hue-Leuchten verfügen über eine große Anzahl einstellbarer Farben, Helligkeiten und einen Farbwechsel-Effekt. Möchte man eine bestimmte Farbe oder Einstellung mehrfach verwenden, kann man diese als Favorit abspeichern.

Um mehrere Leuchten auf einmal einstellen zu können, kann man die gewünschten Farben und Einstellungen jeder Leuchte in einer Szene speichern. Wendet man sie an, nehmen alle betroffenen Leuchten den jeweils abgespeicherten Zustand an.

### Automatisierung

Das Automatisierungs-System ist das Kernstück des Backends. Durch beliebige Regeln kann eingestellt werden, wann welche Aktionen ausgeführt werden und wie auf Umwelt-Ereignisse reagiert werden soll.

Ein Eintrag in der Automatisierung besteht im Wesentlichen aus drei Komponenten:

* Trigger geben an, bei welchen Ereignissen ein Eintrag überprüft werden soll
* Bedingungen können zusätzliche Überprüfungen durchführen, bevor ein Eintrag angewendet wird
* Aktionen legen die Kommandos fest, die ausgeführt werden sollen, wenn die Bedingungen erfolgreich ausgewertet wurden und können verzögert ausgeführt werden

Folgende Optionen sind jeweils möglich:

|  |  |
| --- | --- |
| Typ | Mögliche Optionen |
| Trigger | Licht-Intensität, Bewegung, RFID/NFC-Tag, Netzwerk-Gerät, Sprachbefehl, Uhrzeit, Intervall, benutzerdefiniertes Ereignis |
| Bedingung | Licht-Intensität, letzte registrierte Bewegung, letzte Benutzung eines RFID/NFC-Tags, letzte Aktivität eines Netzwerk-Geräts, Uhrzeit, Wochentag, aktive Benutzer, Zustand der Leuchten, Party-Modus-Einstellung |
| Aktion | Eine Leuchte, eine Gruppe oder alle Leuchten steuern, Szene anwenden, Party-Modus starten oder stoppen, andere Automatisierungs-Einträge (de)aktivieren, benutzerdefiniertes Ereignis auslösen |

Tabelle 1: Mögliche Optionen der Automatisierung

Eine vollständige Dokumentation der Automatisierungs-Optionen ist im Git-Repository unter *docs/implementierung/automatisierung.md* vorhanden.

Darüber hinaus kann noch eingestellt werden, dass ein Eintrag nur einmalig ausgeführt wird und sich danach automatisch löscht.

Durch diese Einteilung können auch komplexere Einträge und Abläufe modelliert werden. Es ist beispielsweise möglich, durch ein RFID-Tag den Bewegungsmelder für einen festgelegten Zeitraum scharfzuschalten, der bei einer Bewegung die Leuchten für eine Minute einschaltet, sofern die Umgebungshelligkeit unter einem gewissen Schwellwert liegt.

In den Bereich der Automatisierung fällt außerdem noch das Verwalten und Benennen von bekannten RFID/NFC-Tags sowie Netzwerk-Geräten.

### Party-Modus

Der Party-Modus kann die Einstellung der Leuchten zeit- oder musikgesteuert verändern. Es können verschiedene Konfigurationen für den Party-Modus angelegt werden, von denen aber höchstens eine einzige aktiv sein kann.

Der Party-Modus basiert auf der zufälligen Auswahl von Einstellungen aus festgelegten Bereichen. Eine Party-Modus-Konfiguration kann mehrere verschiedene Bereiche für Zustands-Einstellungen enthalten, etwa einen rot-orangen und einen gelb-grünen Bereich. Für die Zeit bis zur nächsten Änderung des Zustands, die Überblendzeit sowie die Anzahl der in einem Schritt zu ändernden Leuchten können ebenfalls Bereiche angegeben werden, aus denen ein zufälliger Wert generiert wird.

Dadurch können unter anderem auch Effekte wie etwa ein Lagerfeuer-Modus erzeugt werden, indem in schnellen Intervallen mehrere Leuchten eine zufällige Farbe aus dem rot-orange-Spektrum mit zufälliger Helligkeit zugewiesen bekommen.

Der Takt der Musik, der vom Arduino ermittelt wird, kann ebenfalls dazu genutzt werden, einen Schritt im Party-Modus zu aktivieren. Um die Geschwindigkeit zu begrenzen, kann eine maximale Taktfrequenz eingestellt werden.

### Weitere Funktionen des Backends

Um die Anwendung vor unberechtigtem Zugriff zu schützen, kann ein Passwort vergeben werden, mit dem sich der Benutzer zuerst anmelden muss, um die bereitgestellten Funktionen nutzen zu können.

Ebenso ist eine Möglichkeit zum Speichern der Konfiguration für die Anwendung vorhanden. Diese ist unterteilt in zwei Bereiche:

* Einstellungen, die der Endbenutzer sehen und einfach ändern kann, wie etwa die Standard-Überblendzeit der Leuchten und Einstellungen zur Spracherkennung
* Interne Einstellungen wie der Benutzername für die Hue-Bridge und das Applikations-Passwort, die nur indirekt oder durch zusätzliche Prüfungen geändert werden können

### Funktionen des Arduino

Der Arduino soll eigenständig laufen und periodisch oder bei auftretenden Ereignissen Nachrichten an das Backend schicken:

* Periodischer Durchschnittswert der Umgebungs-Helligkeit
* Erkannte Bewegung
* Erkannter RFID/NFC-Tag
* Erkannter Takt-Schlag

## Auswahl der Software

In der Beschreibung der Studienarbeit wurde ein Backend auf Basis eines Node.JS-Servers mit einer MongoDB und Websockets vorgeschlagen, was im Wesentlichen so umgesetzt wurde. Nachfolgend werden alle verwendeten Software-Komponenten für das Backend kurz beschrieben, unterteilt in Grundbausteine, Module und Erweiterungen sowie zusätzlich genutzte Software.

### Grundbausteine

Node.JS[[10]](#footnote-10) ist eine Laufzeitumgebung für JavaScript-Anwendungen. Er zeichnet sich vor allem durch seine effiziente ereignisgesteuerte Architektur sowie seinen geringen Speicherbedarf aus. Durch Bindings von nativ kompilierten Erweiterungen auf JavaScript-Funktionen können auch systemnahe und komplexe Vorgänge effizient implementiert werden. Die standardmäßig enthaltenen Module erlauben beispielsweise das Bereitstellen eines Web-Servers oder den Zugriff auf das Dateisystem sowie das Aufrufen anderer Anwendungen. Neben Versionen für Windows, MacOS und Linux ist auch eine spezielle Version für den Raspberry Pi verfügbar.

Als Datenbank zum Speichern der Anwendungs-Konfiguration wurde die MongoDB[[11]](#footnote-11) ausgewählt. Dabei handelt es sich um eine dokumentenorientierte NoSQL-Datenbank. Dokumente werden in sogenannten Collections gespeichert. Im Gegensatz zu traditionellen relationalen Datenbanken wird keine feste Struktur für Dokumente einer Collection erzwungen. Da die MongoDB ebenfalls auf einem JavaScript-Interface basiert, können Dokumente direkt als JavaScript-Objekte ausgelesen und abgespeichert werden, ohne sie zuerst in eine Query-Sprache umwandeln zu müssen.

Für den Raspberry Pi ist die MongoDB allerdings nur bedingt geeignet. Einerseits wird sie nicht nativ für ARM-Plattformen zur Verfügung gestellt, sodass auf Portierungen von Dritten zurückgegriffen werden muss. Für diese Studienarbeit wurde die Version des ArduPi-Projekts von Brice Morin verwendet[[12]](#footnote-12). Diese beinhalten oft eine ältere Version der MongoDB und werden nicht aktualisiert. Andererseits ist die MongoDB auf 32bit-Systemen starken Beschränkungen unterworfen. Für diese Studienarbeit relevant ist vor allem das standardmäßig deaktivierte Journaling. Im Journal speichert die MongoDB alle 100ms die letzten Änderungen, um bei einem System-Ausfall einen konsistenten Zustand wiederherstellen zu können. Da der Raspberry Pi keinen physikalischen Schalter zum Herunterfahren besitzt, ist die Wahrscheinlichkeit hoch, dass er einfach durch das Entfernen der Stromquelle ausgeschaltet wird. In diesem Fall ist die Datenbank ohne Journaling inkonsistent und muss vor dem nächsten Starten repariert werden.

Da die erwartete Menge an Daten für dieses Projekt eher gering ist und durch das Caching aller Einträge im Node.JS-Server allgemein nur recht wenige Operationen in der MongoDB ausgeführt werden, ist sie trotz der Einschränkungen für dieses Projekt geeignet. Mit der im nächsten Punkt erwähnten mongoose-Erweiterung können die Daten sehr einfach vom Node.JS-Server verwaltet werden, was der Hauptgrund für die Auswahl der MongoDB war.

### Module und Erweiterungen

Node.JS-Applikationen können über das Programm npm um zusätzliche Module erweitert werden. Im zentralen Repository sind derzeit etwa 68000 Module enthalten (Stand April 2014) [[13]](#footnote-13). Die Implementierung des Projekts basiert im Wesentlichen auf den folgenden Modulen:

|  |  |
| --- | --- |
| Name | Beschreibung |
| Express | Stellt eine Webserver-Plattform zur Verfügung  Wird für die Auslieferung des Frontends sowie die Bereitstellung der REST-API verwendet |
| Socket.IO | Ermöglicht bidirektionale Echtzeit-Kommunikation zwischen Backend und Frontend durch Websockets oder andere Ansätze in älteren Browsern  Wird zur Zustands-Synchronisation und für die Kontrolle des Backends verwendet |
| Mongoose | MongoDB Objekt-Modellierung und Validierung  Wird zum Auslesen und Verwalten von Daten in der MongoDB verwendet |
| node-hue-api | Verbindungsaufbau und Steuerung der Philips Hue-Bridge |
| Serialport | Verbindung und Kommunikation mit dem Arduino |
| Forever | Startet einen Node.JS-Server neu, wenn sich dieser durch eine Fehler beendet  Wird für das Deployment auf dem Raspberry Pi verwendet |

Tabelle 2: Zentrale npm-Module des Backends

Im Rahmen der Netzwerk-Geräteerkennung werden die folgenden zusätzlichen Module verwendet:

* ipv4-range zum Ermitteln aller IP-Adressen im Netzwerk
* ping zum Kontaktieren einer IP-Adresse
* network-address zum Herausfinden der eigenen IP-Adresse
* arp-a (modifizierte Version) zum Zuordnen der IP-Adressen zu physikalischen MAC-Adressen

Die Spracherkennung verwendet außerdem eine modifizierte Version des speakable-Moduls.

### Zusätzliche Software

Für die Spracherkennung wird folgende zusätzliche Software verwendet:

* Julius[[14]](#footnote-14) ist ein Programm zur kontinuierlichen Spracherkennung basierend auf einem Akustik-Modell, einem Vokabular sowie einer Grammatik
* SoX[[15]](#footnote-15) kann dazu genutzt werden, eine Aufnahme zu starten, wenn der Mikrofon-Pegel einen gewissen Wert überschreitet, und sie zu stoppen, wenn wieder Stille herrscht

Die Netzwerk-Geräteerkennung verwendet unter Linux das Programm Nmap[[16]](#footnote-16) anstelle des normalen Ping-Programms. Dies liegt daran, dass das gleichzeitige Starten vieler Ping-Aufrufe beim Testen auf dem Raspberry Pi sehr langsam war und zu viele System-Ressourcen verbrauchte.

Als Versionskontrollsystem für die Implementierung der Studienarbeit wird Git[[17]](#footnote-17) eingesetzt. Es erlaubt das gleichzeitige Bearbeiten von Dateien durch mehrere Benutzer sowie das Nachverfolgen und Zurücksetzen von Änderungen.

## Architektur

Das Backend ist in einer Controller-Model-State-Architektur aufgebaut:

Die gesamte Funktionalität wird durch Controller abgedeckt. Controller können Funktionen zur Verfügung stellen und auf Funktionen anderer Controller zugreifen. Auf der Implementierungs-Ebene sind alle Controller gleichwertig, logisch können sie aber in zwei Bereiche unterteilt werden: Generische Controller bauen die Verbindung zu Peripheriegeräten wie der Hue-Bridge und dem Arduino oder zu anderer Software wie der MongoDB auf oder kümmern sich etwa um die Socket.IO-Verbindungen. Funktionale Controller bauen darauf auf, indem sie deren Funktionalität erweitern und auch dem Frontend Funktionen zur Verfügung stellen, wie etwa die Controller für Gruppen oder Szenen.

Alle in der MongoDB gespeicherten Daten werden als mongoose-Models repräsentiert. Diese werden dazu genutzt, die Einträge auszulesen und zu verwalten.

Der State ist ein globales Zustandsobjekt, das sowohl den aktuellen Zustand der Hue-Bridge als auch alle aus der MongoDB ausgelesenen Dateien enthält. Wird eine beliebige Änderung vorgenommen, wird sie gleichzeitig mit dem tatsächlichen Anwenden in das State-Objekt geschrieben, etwa beim Ändern der Farbe einer Leuchte. Verbindet sich ein neuer Client per Socket.IO, wird ihm nach erfolgreicher Anmeldung das gesamte State-Objekt übertragen. Bei Änderungen werden die betroffenen Bereiche des State-Objekts mit allen verbundenen Clients synchronisiert. Bei den meisten von einem Client ausgehenden Änderungen wird der veränderte State nur an die anderen verbundenen Clients übertragen, da der Auslöser seinen State selbst entsprechend anpassen kann. Tritt dabei im Backend ein Fehler auf, wird der ursprüngliche State wiederhergestellt.

Manche Controller können ihren Dienst erst aufnehmen, wenn bestimmte Models geladen sind oder andere Controller Verbindungen aufgebaut haben. Außerdem kann erst nach dem Laden aller Controller sichergestellt werden, dass ein Controller auf die Funktionen aller anderen Controller zugreifen kann. Aus diesem Grund wird das Backend durch ein Event- und Dependency-System getragen, das es erlaubt, eine Funktion erst auszuführen, sobald die nötigen Voraussetzungen dafür geschaffen wurden.

## Implementierung

Nachfolgend werden verschiedene Aspekte der Implementierung des Backends näher beschrieben. Beginnend mit der Grundstruktur des Backends wird auf die Socket.IO-Verbindung, die Netzwerk-Geräte-Erkennung, die Sprachsteuerung sowie die Programmierung des Arduino eingegangen.

Die vollständige Entwickler-Dokumentation für die Implementierung ist im Git-Repository im Ordner *docs/implementierung* enthalten.

### Grundstruktur

Der Zugriff der einzelnen Controller auf das State-Objekt sowie auf andere Controller, das Event-System und weitere Module wird über ein globales App-Objekt realisiert. Dieses wird beim Laden an jeden Controller übergeben, während der Controller selbst ebenfalls dem Objekt hinzugefügt wird.

Folgender Code bindet alle Dateien im Controller-Verzeichnis ein:

*require*(**'fs'**).readdirSync(***\_\_dirname*** + **'/server/controllers'**).forEach(**function**(file) {

**var** controller;

**if**(file.match(/\.js$/) !== **null**) {

controller = file.replace(/\.js$/, **''**);

***app***.**controllers**[controller] = *require*(**'./server/controllers/'** + controller)(***app***);

}

});

Abbildung 3: Backend-Code zum Einbinden der Controller

Jeder Controller exportiert eine Funktion, die das App-Objekt als Parameter übergeben bekommt. Über das Event-Framework wird die Funktionalität des Controllers aktiviert, wenn alle Voraussetzungen erfüllt sind. Die Funktionen, die für andere Controller zur Verfügung gestellt werden sollen, gibt er in einem Objekt zurück:

***module***.**exports** = **function**(globalApp) {

***app*** = globalApp;

***app***.**events**.on(**'ready'**, **function**() {

*init*();

});

**return** {

**applyScene**: *applyScene*

};

};

Abbildung 4: Grundaufbau für Backend-Controller am Beispiel der Szenenverwaltung

Die generischen Controller stellen den funktionalen Controllern oft Listener-basierte Funktionen bereit. So können zum Beispiel Listener auf bestimmte Socket.IO-Befehle, Arduino-Nachrichten oder Änderungen von bestimmten Konfigurations-Einträgen angelegt werden.

***app***.**controllers**.**socket**.addSocketListener(*socketListener*);

***app***.**controllers**.**arduino**.addListener(*arduinoListener*);

Abbildung 5: Event-Listener im Backend

### Socket.IO-Verbindung

Alle Befehle des Frontends werden über Socket.IO an das Backend gesendet. Nach ihrer Ausführung wird der betroffene Bereich des zentralen State-Objekts mit allen anderen verbundenen Clients synchronisiert.

Ist die Anwendung mit einem Passwort geschützt, muss sich der Client erst damit anmelden, bevor das Backend Befehle über das Socket akzeptiert. Der Login-Vorgang läuft folgendermaßen ab:

* Der Client baut eine Socket-Verbindung auf
* Der Server sendet eine „login.required“-Nachricht mit dem Inhalt *true*, um dem Client zu signalisieren, dass ein Login erforderlich ist
* Der Client sendet das Passwort in einer „login“-Nachricht
* Der Server sendet eine „login“-Nachricht mit dem Inhalt *true*, um zu bestätigen, dass der Login erfolgreich war
* Das Socket wird im Backend der „login“-Gruppe hinzugefügt, an die State-Änderungen gesendet werden
* Die Listener der anderen Controller werden dem Socket hinzugefügt
* Der Server schickt dem Client das komplette State-Objekt

Da viele Controller dem Client eine ähnliche CRUD-Funktionalität bereitstellen, um Datensätze verschiedener Models auslesen, anlegen, bearbeiten und löschen zu können, wurde dies in einer Helper-Funktion gebündelt.

Am Beispiel der Szenenverwaltung führt die CRUD-Helper-Funktion folgende Aktionen aus:

* Die Szenen werden aus der MongoDB ausgelesen und im State-Objekt gespeichert
* Die Socket-Listener „scene.create“, „scene.update“ und „scene.delete“ werden registriert
* Bei Fehlern werden die Notifications „scene.create“ und „scene.update“ erzeugt
* Nach dem Laden wird das „scene.ready“-Event ausgelöst
* Die Funktion gibt ein Objekt zurück, mit dem wiederum Listener auf durchgeführte Aktionen hinzugefügt werden können. Dies wird beispielsweise dafür genutzt, Automatisierungs-Einträge zu bereinigen, die ansonsten eine gelöschte Szene anwenden würden.

Die vollständige Dokumentation und Auflistung aller möglichen Befehle ist im Git-Repository in *docs/implementierung/socket\_io.md* enthalten.

Um auch von anderen Anwendungen und Geräten gesteuert werden zu können, die über keine Socket.IO-Implementierung verfügen, werden alle Socket.IO-Befehle ebenfalls über eine REST-API zur Verfügung gestellt. Der REST-Controller stellt dafür ein Dummy-Socket zur Verfügung, das dieselben Funktionen wie ein Socket.IO-Socket zur Verfügung stellt. Der Socket-Controller sorgt nun dafür, dass alle Listener, die den normalen Sockets von den anderen Controllern hinzugefügt werden, auch am REST-Dummy-Socket registriert werden. Dadurch werden intern für Socket- und REST-Aufrufe dieselben Funktionen aufgerufen.

**var *socketDummy*** = {

on: **function**(event, listener) {

***socketDummyListeners***.push([

event,

listener

]);

},

**isDummySocket**: **true**,

emit: **function**(event, body) {

**console**.log(**'[rest] Tried to emit data to the REST dummy socket: '**, event, body);

}

};

Abbildung 6: Dummy-Socket-Implementierung im REST-Controller

### Netzwerk-Geräte-Erkennung

Innerhalb von lokalen Netzwerken werden Geräte anhand ihrer IP-Adresse angesprochen. Diese wird jedoch häufig von einem Router durch DHCP automatisch zugewiesen und kann sich bei jeder erneuten Anmeldung am Netzwerk ändern.

Daher wurde entschieden, für die Studienarbeit die MAC-Adresse eines Geräts zur Identifizierung zu verwenden. Diese ist für jedes Gerät eindeutig und ändert sich in der Regel nie, auch wenn sie prinzipiell beliebig verändert werden kann. Der Nachteil daran ist, dass von einer Applikation aus im Gegensatz zur IP-Adresse nicht einfach ermittelt werden kann, ob ein Gerät mit einer bestimmten MAC-Adresse gerade im Netzwerk angemeldet ist.

Obwohl alle Geräte im Netzwerk auf TCP/IP-Basis kommunizieren, werden die einzelnen Pakete innerhalb des Netzwerks auf MAC-Basis adressiert. Daher muss jedes Gerät über eine ARP-Tabelle verfügen, in der es die Zuordnungen von IP-Adressen zu MAC-Adressen speichert.

Diese ARP-Tabelle wird nun ausgelesen, um die aktuellen IP-Adressen der registrierten Geräte zu erhalten. Durch einen Ping-Befehl wird dann periodisch deren Anwesenheit im Netzwerk überprüft. Da die ARP-Tabelle aber nur mit Geräten befüllt wird, mit denen Bereits eine Kommunikation stattgefunden hat, müssen die Ping-Befehle den gesamten Netzwerk-Adressbereich abdecken, um mit allen möglichen IP-Adressen kommuniziert zu haben. In einem typischen Heim-Netzwerk umfasst der IP-Bereich 255 Adressen.

Auf dem Raspberry Pi stellte sich bei einem ersten Test heraus, dass das Starten von 255 Ping-Befehlen eine enorme Menge an Ressourcen beansprucht, was den stabilen Betrieb des gesamten Backends gefährdet. Daher wurde das Programm Nmap als Alternative verwendet, da es innerhalb eines einzigen Prozesses jeder IP-Adresse im Netzwerk-Segment einen Ping-Befehl schicken kann und daher wesentlich weniger Ressourcen in Anspruch nimmt.

### Sprachsteuerung

Bei der Entwicklung der Sprachsteuerung standen mehrere Ansätze zur Auswahl:

* Lokale Erkennung oder Erkennung durch einen Internet-Dienst
* Festgelegtes Vokabular oder gesamtes Wörterbuch einer Sprache

Dabei sind vor allem die Faktoren Leistungsbedarf, Dauer der Auswertung und Erkennungsrate von Bedeutung. Auch der Aspekt, dass viele Menschen nicht die gesamte in einem Raum aufgezeichnete Kommunikation an einen Internet-Dienst übertragen möchten, darf nicht vernachlässigt werden.

Dem gegenüber steht die relativ geringe Rechenleistung des Raspberry Pi, den eine lokale Spracherkennung bei großem Vokabular stark beanspruchen kann.

Daher wurden für diese Studienarbeit zwei Ansätze weiter verfolgt:

* Aufzeichnung der Befehle mit dem Programm SoX und Online-Auswertung durch die Google Speech API mit einer modifizierten Version des npm-speakable-Moduls; damit können beliebige Sätze in englischer Sprache erkannt werden
* Lokale Auswertung auf dem Raspberry Pi mit Julius, einem Akustik-Modell von VoxForge[[18]](#footnote-18) sowie einem begrenzten Vokabular

In der Konfiguration des Backends ist die Sprachsteuerung als Ganzes optional aktivierbar. Dazu kann noch ausgewählt werden, ob die Google Speech API oder Julius für die Spracherkennung verwendet werden soll.

Als Alternative für Julius wurde noch PocketSphinx[[19]](#footnote-19) getestet, aufgrund des vergleichsweise hohen Leistungsbedarfs auf dem Raspberry Pi schied es aber als Kandidat für die Spracherkennung aus.

Die folgende Grammatik wurde für die Julius-Spracherkennung definiert:

* Start-Befehle: Computer, Hue
* Kommandos: Scene, initiate, stop
* Bereiche: All, Zahlen von zero bis nine
* Werte: Off, on, dark, bright, blue, green, red, yellow, white, purple, orange

Diese Wort-Kategorien können folgendermaßen zusammengesetzt werden (Kategorien in eckigen Klammern optional):

* Start Wert [Wert]
* Start Bereich [Wert]
* Start Kommando [Bereich / Wert]

Dadurch können beispielsweise Befehle wie „Computer one blue“ erkannt werden.

Die Möglichkeit zum Modifizieren der Grammatik ist im Git-Repository in *docs/implementiernug/sprachsteuerung.md* dokumentiert.

### Implementierung des Arduino

Der Arduino wird in der Sprache C programmiert, die um Arduino-eigene Funktionen erweitert wurde[[20]](#footnote-20). Ein Programm besteht aus einer Setup-Funktion, die einmalig zu Beginn ausgeführt wird, sowie einer Loop-Funktion, die das Hauptprogramm enthält und immer wieder aufgerufen wird.

Einen guten Einstieg in die Arduino-Plattform bietet etwa das Buch „Getting Started with Arduino“ von Massimo Banzi[[21]](#footnote-21).

Da der Arduino über keine Timer-Interrupts verfügt, können zeit- und intervall-gesteuerte Funktionen nach folgendem Prinzip angelegt werden:

* Die Funktion millis() gibt die Anzahl der vergangenen Millisekunden seit dem Start des Programms zurück
* Diese Zeit wird mit der letzten Aufruf-Zeit der Funktion verglichen
* Überschreitet die Differenz einen festgelegten Wert, wird die Funktion erneut aufgerufen und deren Aufruf-Zeit auf die aktuelle Zeit gesetzt

Die Musik-Takt-Erkennung funktioniert basiert auf einem recht primitiven Schall-Energie-Algorithmus, wie ihn etwa Frédéric Patin beschreibt[[22]](#footnote-22). Dies ist einerseits in der geringen Rechenleistung und Hauptspeichergröße des Arduino begründet, andererseits liefert der eingesetzte Sound-Sensor nur die Schall-Intensität als Messwert.

Bei jedem Zyklus der Loop-Funktion wird der aktuelle Wert des Sensors gemessen. Da er manchmal trotz Umgebungsgeräuschen 0 zurückgibt, wird so oft gemessen, bis ein Wert ungleich 0 herauskommt. Der gemessene Wert wird zum Verstärken der Unterschiede quadriert und dann auf eine Summe aufaddiert (Sample-Summe). Die Summe besteht aus einer festgelegten Anzahl von Messungen. Wurde n mal gemessen, wird die Summe in ein Array zusammen mit früheren Summen gespeichert (Frames).



Abbildung : Samples und Frames bei der Arduino-Takterkennung

Um zu erkennen, ob die aktuelle Sample-Summe einen Taktschlag darstellt, wird ihr Wert mit dem Durchschnittswert der gespeicherten Frames verglichen. Überschreitet der Unterschieds-Faktor einen festgelegten Schwellwert, wird ein Schlag erkannt. Der am besten geeignete Schwellwert wurde durch experimentelle Versuche bestimmt. Da bei höherer Musik-Lautstärke der Unterschied zwischen Taktschlägen und restlicher Musik nicht so groß ist, wird die Amplitudensumme des aktuellen Samples mit in den Schwellwert eingerechnet.

Um eine zu schnelle Takterkennung zu verhindern, darf ein Schlag erst eine festgelegte Zeitspanne nach dem letzten erkannt werden.

Nachrichten werden als JSON-Zeichenketten an den Raspberry Pi übertragen. Das gesamte Kommunikations-Protokoll des Arduino ist im Git-Repository unter *docs/implementierung/arduino.md* enthalten.

## Installation

Nachfolgend wird die Einrichtung der Entwicklungsumgebung für das Projekt sowie das Deployment der Anwendung auf dem Raspberry Pi beschrieben.

### Einrichten der Entwicklungsumgebung

Die Entwicklungsumgebung unter Windows benötigt folgende Software:

* Node.JS
* MongoDB
* Git
* Arduino IDE
* Python (Version 2.x)
* Eine beliebige Version des Microsoft Visual Studios
* SoX
* Julius
* Ein beliebiger Text-Editor oder IDE

Eine vollständige Liste mit allen Download-Links ist im Git-Repository unter *docs/setup/entwicklungsumgebung.md* enthalten.

Unter Linux lässt sich die benötigte Software bequem über apt-get installieren. Um mit dem Arduino kommunizieren zu können, muss der aktuelle Benutzer der Gruppe „dialout“ hinzugefügt werden[[23]](#footnote-23):

sudo apt-get install git python build-essential nodejs mongodb arduino sox nmap julius

sudo usermod -aG dialout username

Abbildung 8: Installation der Entwicklungsumgebung unter Linux

Die Einrichtung des Projekts umfasst folgende Schritte, die ebenfalls in der Entwickler-Dokumentation ausführlicher erläutert werden:

* Klonen des Git-Repositories
* Installieren der npm-Module
* Herunterladen des VoxForge Akustik-Modells für Julius
* Anlegen eines Datenverzeichnisses und Starten der MongoDB
* Starten des Node.JS mit der Datei *nodejs/server.js*
* Das Frontend ist nun unter <http://localhost:8080> erreichbar

### Deployment auf dem Raspberry Pi

Nach der Grund-Installation des Betriebssystems und Einrichten der Peripherie kann die Applikation auf dem Raspberry Pi installiert werden.

Um diesen Prozess drastisch zu vereinfachen, wurde ein Installations-Script erstellt, das alle benötigten Schritte automatisiert durchführt. Nachdem es auf den Raspberry Pi geladen wurde, muss es ausführbar gemacht werden und kann dann aufgerufen werden:

sudo chmod +x setup.sh

sudo ./setup.sh

Abbildung 9: Aufrufen des Installations-Scripts für den Raspberry Pi

Die folgenden Schritte werden durch das Installations-Script durchgeführt:

* Installation aller benötigten Dependencies, die über die offizielle Paketverwaltung verfügbar sind
* Download und Installation des Node.JS-Servers
* Installation des Forever-Moduls
* Download und Installation des MongoDB-Ports für den Raspberry Pi
* Download und Kompilieren von Julius
* Klonen des Git-Repositories der Studienarbeit
* Installieren der npm-Module
* Download des VoxForge Akustik-Modells
* Anlegen von Autostart-Scripts für die MongoDB und die Anwendung in Verbindung mit Forever

Nach dem Durchlauf des Scripts muss der Raspberry Pi neu gestartet werden. Danach startet die Anwendung automatisch mit dem System.

Eine ausführliche Beschreibung aller Schritte, die auch als Anleitung zum manuellen Durchführen der Installation dient, ist im Git-Repository unter *docs/setup/raspberry.md* enthalten. Das Script selbst ist in Anhang C aufgeführt (Seite 68).

# Entwicklung des Frontends

Das Frontend ist die Schnittstelle zwischen dem Backend und dem Benutzer. Sie stellt dem Benutzer die im Backend erzeugten Funktionen zur Verfügung und ist für den Datenaustausch zwischen Eingabe und Umsetzung zuständig. Dabei soll der Zugriff sowohl vom Desktop-PC, Laptop als auch vom Tablet-PC und Smartphone möglichst benutzerfreundlich sein.

Dieses Kapitel widmet sich den Herausforderungen der Entwicklung einer plattformunabhängigen Web-Applikation. Dabei wird zunächst die Auswahl der Software thematisiert, bevor auf das Designkonzept und die letztendliche Implementierung eingegangen wird.

## Softwareentscheidungen

Dieses Kapitel beschäftigt sich mit der Auswahl der genutzten Software zur Implementierung der Applikation.

### WebApp

Die Entscheidung, welche Art von mobiler Applikation gebaut werden soll, spielte sich zwischen zwei Umsetzungsansätzen ab. Zum einen gibt es die native App, welche betriebssystemspezifisch programmiert wird und einen direkten Zugriff auf die Hardware des Smartphones oder Tablets ermöglicht. Durch das Attribut "plattformunabhängig" wurde dieser erste Ansatz allerdings direkt verworfen und die zweite Möglichkeit, die WebApp als Lösung definiert. Die sich hier bietenden Vorteile durch die Nutzbarkeit auf nahezu allen internetfähigen Endgeräten und die Tatsache, dass das Endprodukt dieser Entwicklung keinen Zugriff auf smartphone- oder tabletspezifische Hardware benötigt, bestätigen die Entscheidung.

Zur Entwicklung der WebApp wird die plattformunabhängige Dokumentbeschreibungssprache HyperText Markup Language 5, abgekürzt auch HTML5, verwendet, welche den strukturellen Aufbau der WebApp übernimmt. Das Design wird von Cascading Style Sheets 3 oder CSS3 übernommen. Diese Trennung zwischen Struktur und Design wird durch die steigende Komplexität dringend benötigt und stellt heutzutage einen Klassiker in der Webentwicklung dar. Um die Möglichkeiten von statischem HTML zu erweitern wird Javascript verwendet. Dabei kann die Benutzung von Frameworks die Entwicklung deutlich beschleunigen, weil sie oft schon Funktionalitäten mitliefern. Hier sollte man allerdings die Einarbeitungszeit in ein neues Framework nicht unterschätzen, wobei diese Einarbeitungszeit nicht nur für ein Projekt zu rechnen ist, denn das erlernte Framework kann ja auch in anderen Projekten, dann ohne Einarbeitung, eingesetzt werden. In den folgenden Kapiteln werden die zwei für dieses Projekt ausgewählten Frameworks näher betrachtet.

### jQuery Mobile

jQuery Mobile ist ein Framework, welches auf jQuery aufbaut und von sehr vielen Browsern und Systemen unterstützt[[24]](#footnote-24) wird. jQuery ist eine Javascript-Bibliothek, welche Funktionen zur vereinfachten Entwicklung mit Javascript bereitstellt.

jQuery Mobile ist schnell zu erlernen und durch die vordefinierten Designs der UI-Elemente leicht anwendbar. Ein weiterer Vorteil ist die Plattform- und Geräteunabhängigkeit. jQuery Mobile ist ein Javascript-Framework, dass vor allem die Entwicklung von mobilen Benutzerschnittstellen vereinfacht. Das heißt es können keine Funktionen geschrieben werden, die sich um das Abspeichern von clientseitigen Daten auf dem Server konzentrieren, sondern lediglich das Design der WebApp wird verändert.[[25]](#footnote-25) jQuery Mobile baut auf jQuery auf, deshalb muss nicht nur das jQuery Mobile-Framework in den Header der HTML Seite eingebaut werden, sondern auch die jQuery-Bibliothek. Neben der Javascript-Datei müssen auch diverse CSS-Dateien im Header aufgenommen werden, die beispielsweise die typischen jQuery Mobile-Icons und Designthemen enthalten. Über ein Online-Tool namens Themeroller[[26]](#footnote-26) können die Designthemen farblich personalisiert und danach als \*.css exportiert werden.

jQuery Mobile arbeitet nach dem Prinzip des Progressive Enhancements. Dieses Prinzip verfolgt das Ziel eine Website für jeden Browser zur Verfügung zu stellen, egal wie alt er ist. Das heißt, dass auch Browser, die CSS und Javascript nicht unterstützen auf den Inhalt und die Grundfunktionalität zugreifen können.

Es gibt einige User Interface-Komponenten, die jQuery Mobile zur Verfügung stellt. Dazu gehören auch:

* Toolbars
* Buttons
* Events
* Icons
* Slider
* Formularelemente

Darüber hinaus stellt jQuery Mobile verschiedene Touch-Gesten zur Verfügung, wie zum Beispiel:

* Tap   
  Ein kurzer Tap auf den Bildschirm startet ein Event.  
  Beispiel: Seitenwechsel
* Taphold  
  Nur wenn der Benutzer ein paar Sekunden auf den Bildschirm tippt wird das Event ausgelöst.   
  Beispiel: Öffnen eines Kontext-Menüs
* Swipeleft/Swiperight  
  Wird durch ein Wischen von rechts nach links/ links nach rechts über den Bildschirm ausgelöst.  
  Beispiel: vorwärts oder rückwärts blättern in einem Ebook

Die Navigation in jQuery Mobile wird zudem über AJAX geregelt. Dabei ist zu beachten, dass nur der Inhalt der neuen HTML-Seite ersetzt wird. Dieser Inhalt wird durch das Attribut *data-role=“page“* gekennzeichnet. [[27]](#footnote-27)

### AngularJs

AngularJs ist ebenfalls ein Javascript-Framework, welches als Open Source-Projekt von Google entwickelt wird. Es gibt drei Prinzipien nach denen AngularJs verfährt:

* Model-View-Controller   
  Die UI, die Daten und die Applikationslogik werden voneinander getrennt geregelt.
* Data Binding  
  Hier handelt es sich um bidirektionales Data-Binding, bei welchem die Daten auf dem Server und in der UI jederzeit synchronisiert sind.
* Dependency Injection   
  Die Dependency Injection entzieht einem Objekt die Kontrolle über das Instanziieren einer Funktion.

HTML dient Angular als Template-Sprache und erweitert es mit eigenen HTML Tags & Attributen. Angular setzt sich aus mehreren Komponenten zusammen, dazu gehören auch die Controller. Ein Controller kann als Bindeglied zwischen dem Model und der View, also zwischen den Daten und der Oberfläche bezeichnet werden. Er enthält die Applikationslogik und sorgt dafür, dass die Informationen aus dem Model im eigenen Scope der UI zur Verfügung steht. Der Scope ist eine Art clientseitiges Informationsbecken, über das die Benutzeroberfläche ihre Daten bezieht.

Eine weitere Komponente ist die Direktive. Eine Direktive kann als ein HTML-Element in die normale HTML-Seite eingebaut werden und erstellt dann ein Objekt nach einem in der Direktive definiertem Bauplan. Dabei können auch Funktionen definiert werden, wie zum Beispiel das Verhalten bei der Initialisierung oder beim Klick auf das Element. Eine Direktive besteht immer aus einem HTML-Template und selbst definierten Funktionen oder Verhalten.

Ein Service ist ebenfalls eine Angular-Komponente, die ähnlich wie die Controller Logik und Daten zur Verfügung stellt. Allerdings handelt es sich hier um Applikationsübergreifende Logik, wie zum Beispiel den $route-Service, welcher abhängig von der URL die Webseite wechselt.

Alle Komponenten in Angular können vom Benutzer selbst gebaut werden und bieten dadurch einen großen Spielraum.[[28]](#footnote-28)

## Layout und Design

Im folgenden werden die Konzepte vorgestellt, nach denen die WebApp umgesetzt werden soll.

### Responsive Webdesign

Responsive Webdesign ist ein neues Modewort in der Branche der Webentwicklung geworden. Es bedeutet, dass eine Website nicht nur für den alltäglichen Gebrauch auf dem Desktop-PC angepasst ist, sondern auch für alle anderen internetfähigen Geräte, wie zum Beispiel für das Smartphone oder den Tablet-PC. Man könnte responsives Webdesign also auch als eine Anforderung bezeichnen, die aus dem technischen Fortschritt heraus entstanden ist.

Ziel ist es, unabhängig von der Bildschirmgröße des Endgeräts den Benutzer den größtmöglichen Nutzen einer Website schnell und einfach zu ermöglichen. Es gibt verschiedene Lösungsansätze, um dieser Anforderung gerecht zu werden und das Ziel zu erfüllen.

Zum einen können für verschiedene Endgeräte jeweils spezifische Webseiten entwickelt werden, diese Art der Umsetzung nennt man allerdings nicht responsive Design, sondern mobile Webseite, was oft miteinander vermischt wird. Eine mobile Webseite ist ganz genau auf die Anforderungen des Endgeräts und auf die Bedürfnisse des Benutzers angepasst und es müssen keine Kompromisse eingegangen werden. Der Nachteil daran ist allerdings, dass die Entwicklungskosten- sowie der Wartungsaufwand sich auf das doppelte erhöhen.

Bei einer responsiv gestalteten Webseite wird nur ein Template benötigt, welches dann über reaktionsfähiges Design und ein fließendes Layout realisiert werden kann. Die Umsetzung kann dabei mit Hilfe von CSS3 Media Queries erfolgen. Diese Media Queries sind zusätzliche CSS-Regeln, die abhängig von der Bildschirmgröße des Endgeräts in Kraft treten und die verschiedenen HTML-Elemente optisch verändern. Wichtig dabei ist, dass es unabhängig von der Größe eine möglichst übersichtliche Webseite bleiben soll und das der Benutzer sich unabhängig von der Darstellung auf der Webseite zurechtfindet. Es darf nicht das Gefühl vermittelt werden, dass der Benutzer auf einer völlig anderen Webseite ist. Deshalb sollte ein gewisser Wiedererkennungswert vorhanden sein. [[29]](#footnote-29)

### Mobile First

Mobile First ist ein Konzept zur Entwicklung von responsiven Webseiten. Dieses Konzept beschränkt sich allerdings auf die Herangehensweise und konzentriert sich weniger auf die technische Umsetzung der Webseite.

Es geht darum sich gedanklich auf Bildschirme mit einer sehr niedrigen Auflösung und geringen technischen Möglichkeiten zu konzentrieren. Dabei wird das Wesentliche der Webseite fokussiert und es kann besser analysiert werden, was der Benutzer wirklich braucht und welche Funktionalitäten und Information eine hohe Priorität bekommen sollten, beziehungsweise welche weggelassen werden könnten. Laut Luke Wroblewski bietet das "Mobile First"-Konzept sogar den großen Vorteil, dass durch diesen Ansatz ein schmales Design entwickelt wird, welches bei Übertragung auf größere Bildschirme ein durchweg smartes Layout produziert.[[30]](#footnote-30)

Der Verzicht auf unwesentliche Informationen auf kleinen Bildschirmen wird zum Vorteil gemacht und auch für große Bildschirme übernommen. Es bleibt eine übersichtliche und leicht bedienbare Designlösung, bei der die Menge an Informationen und Funktionalitäten auf ein Mindestmaß beschränkt wurde. So wird das Ausmaß der Webseite verkleinert und nicht jeder verfügbare Pixel mit Informationen gefüllt. Das Risiko eine Webseite zu voll zu packen oder den Benutzer mit zu viel Informationen zu überfordern wird damit verringert.

Natürlich ist dieses Konzept sehr idealistisch und das Ergebnis hängt stark von der Interpretation des Entwicklers ab. Bei Anwendung dieses Konzepts kann die Entwicklung aber in die richtige Richtung gedrängt werden. Das heißt, selbst wenn das Design auf dem Smartphone nicht genau die gleichen Funktionalitäten enthält wie auf dem Desktop-PC, kann das Gesamtdesign smart und zweckerfüllend sein.

### Layoutkonzept

Die Webseite der SmartLights soll responsiv aufgebaut und nach dem Mobile First-Konzept entwickelt werden. Da zunächst von der kleinsten Bildschirmgröße ausgegangen werden soll, wurde dafür eine Breite von 300-550px definiert. In diesen Bereich liegen normale bis große Smartphones und auch kleine Tablets. Durch die geringe Breite, sollten nur sehr kleine Inhalte nebeneinander angezeigt werden, beispielsweise kleine Buttons. Da bei einer geringen Breite nur wenig Inhalt pro Zeile dargestellt werden kann, müssen die Inhalte untereinander positioniert werden. Um dabei schnelle Seitenwechsel zu ermöglichen, wird ein fixierter Kopfbereich eingefügt, der die Navigationsleiste enthält. Dies ist vor allem bei einem Layout wichtig, bei dem viel gescrollt werden muss. Durch eine fixierte Navigationsleiste muss der Benutzer für einen Seitenwechsel nicht wieder zum Beginn einer Seite zurückscrollen.

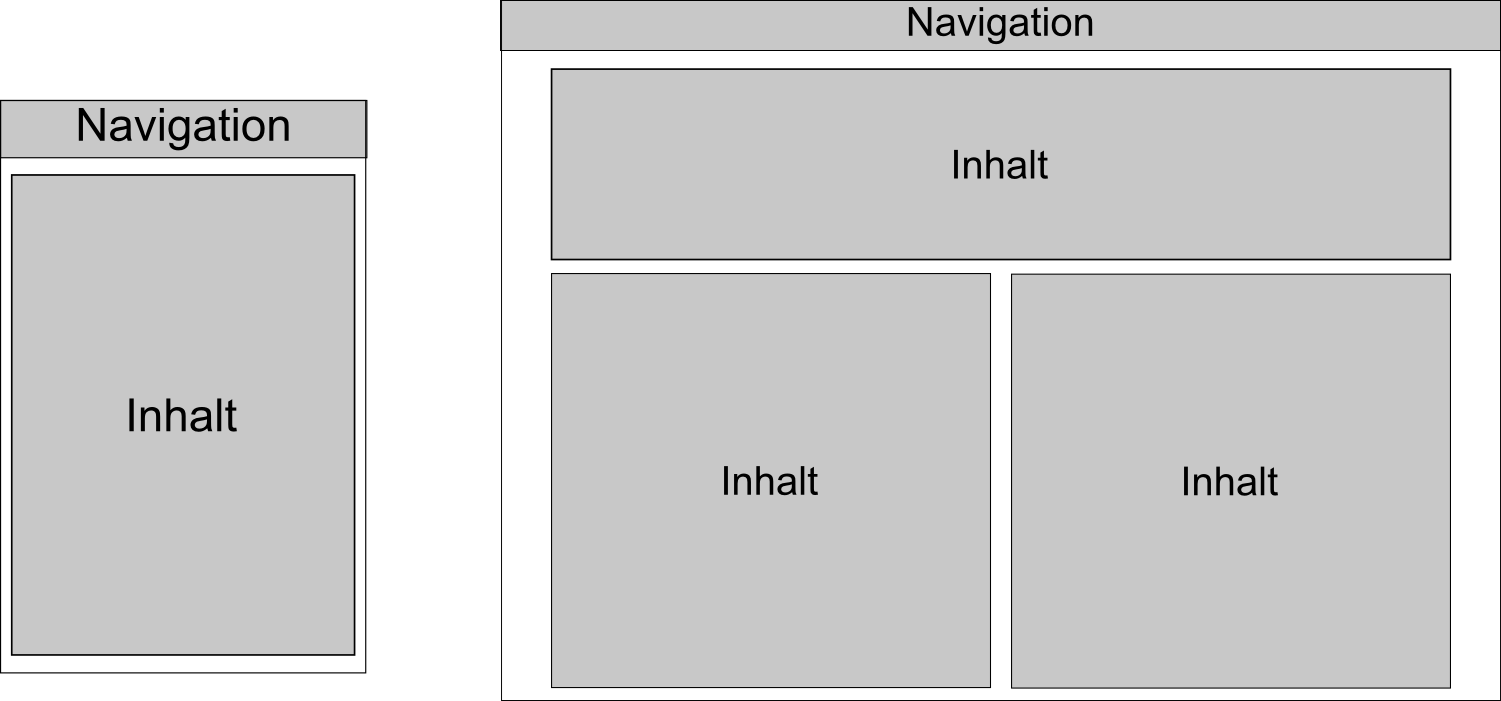


Abbildung : Erstentwürfe für Smartphone und Tablet

Für die nächstgrößeren Endgeräte wie beispielsweise Tablets oder kleine Desktop-PCs, die eine Breite von 550-1100px besitzen, kann die Anzeige der Informationen mehr in die Breite ausgedehnt werden. Um den möglichen Platz allerdings nicht mit unnötigen Informationen zu überfluten, bleibt die Navigation weiterhin im fixierten Kopfbereich. Es entsteht durch die erhöhte Breite die Möglichkeit Inhalte flexibler anzeigen zu lassen, beispielsweise über die gesamte Breite hinweg oder nur über einen Teil davon.

Für alle Endgeräte über 1100px Breite wird keine weitere Differenzierung vorgenommen, da ganz nach dem Prinzip von Herrn Luke Wroblewski vorgegangen wird: nur weil es mehr Platz gibt, müssen nicht mehr Informationen platziert und damit der Bildschirm komplett gefüllt werden. So bleibt die Webseite weiterhin übersichtlich und klar strukturiert. Die Navigationsleiste kann allerdings von der Kopfzeile in eine Liste links verschoben und vergrößert dargestellt werden. Dies ist auch für die Benutzerfreundlichkeit wichtig.

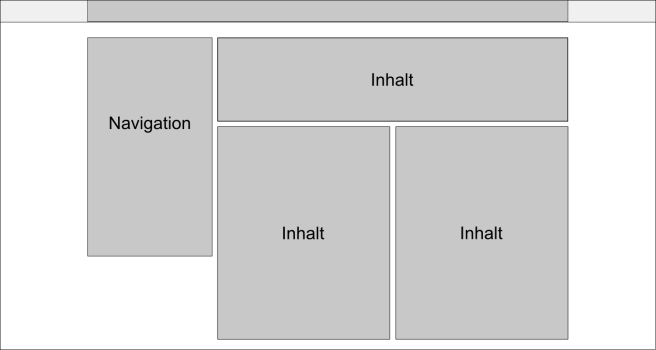


Abbildung : Erstentwurf >1100px Breite

### Designkonzept

Grundsätzlich wird ein flaches transparentes Design gewählt. Je nach logischem Zusammenhang werden verschiedene Elemente optisch gruppiert, sodass dem Benutzer die Anwendung der WebApp leichter fällt. Diese optische Gruppierung geschieht oftmals durch einen fortlaufenden Hintergrund für mehrere zusammenhängende Funktionen.

Der Hintergrund der gesamten WebApp wird mit einem leichten Farbverlauf durchzogen, welcher nicht nur die Transparenz der einzelnen daraufliegenden Elemente hervorheben soll, sondern auch durch einen dunklen Farbton beruhigend wirkt und sich von den bunten Farben der Lampen abhebt. Das Farbschema ist relativ schlicht gehalten, damit die Farben der einzelnen Lampen, die teilweise angezeigt werden, ausreichend zur Geltung kommen und der Benutzer nicht mit zu vielen Reizen auf einmal konfrontiert wird.

Die Icons und Buttons, sowie die Schrift werden in einem dunklen Rotton realisiert. Auf den weiß-transparenten Elementen bietet dieser Farbton einen guten Kontrast.

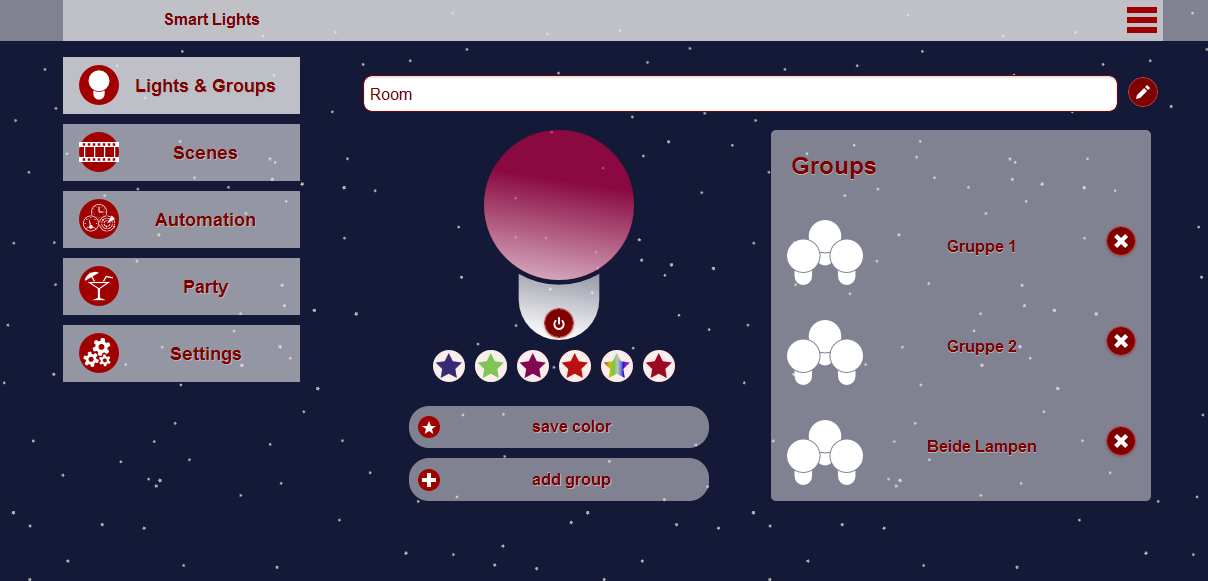


Abbildung : Veranschaulichung des Designkonzepts anhand der light.html

## Implementierung

Nun wird die genaue Umsetzung der theoretischen Überlegungen vorgestellt.

### Javascript-Frameworks

Um beide Javascript-Frameworks zu benutzen, muss eine weitere Javascript-Datei als Verbindungsglied eingebunden werden. Es wurde bereits ein Adapter für die Versionen 1.0.6 von Angular, 1.8+ von jQuery und 1.3.1 von jQuery Mobile entwickelt. Dieser konnte von Git (*https://github.com/opitzconsulting/jquery-mobile-angular-adapter)* bezogen werden und nach einigen Codekürzungen für die Version 1.3.0 von Angular, 1.10.2 von jQuery und 1.4.0 von jQuery Mobile kompatibel gemacht werden. Die Frameworks verfolgen bei manchen Problemstellungen unterschiedliche Lösungsansätze. Die Lösungsansätze behindern sich teilweise gegenseitig, wie beispielsweise beim Thema Routing in AngularJs. Das ist das Hauptproblem, welches den Adapter überhaupt notwendig macht.

Durch den Adapter konnte allerdings eine andere Art der Controllerarchitektur ausprobiert werden: der "SharedController". Die Controller wurden je nach ihrem Inhalt in verschiedene Javascript-Dateien gegliedert, um eine gewisse Übersichtlichkeit für den Entwickler zu bewahren. Das erklärt beispielsweise die Existenz der LightAndGroup, Party, Scene und Speech Controller neben dem MainController. Der MainController wird durch das Attribut *ngm-shared-controller="sharedScope:MainCtrl"* in jeder Seite eingebunden und ist ein SharedController, der von allen Seiten eingebunden wird und seinen Scope damit allen Seiten zur Verfügung stellt. Der MainController verwendet zudem noch den $rootScope, welcher im Grunde auch ein geteilter Scope ist, allerdings entfällt hier die Möglichkeit zu entscheiden, welcher Controller den Zugriff darauf haben darf und welcher nicht. Der $rootScope steht als großer Rahmen um alle anderen Scopes herum.

Alle Controller binden zusätzlich den StateManager ein, der als Service den aktuellen Status des Servers mit dem des Clients synchronisiert und mit Fehlermeldungen umgeht. Dazu gehört zum Beispiel auch die Verbindung zur Bridge, das Login-Management und die generelle Verbindung des Sockets.IOs zum Backend. Ein weiterer Service ist dann noch der Socket.IO Service, welcher sicherstellt, dass eine Verbindung zum Server hergestellt wird.

### Grundstruktur

Durch das jQuery Mobile Element *data-role="page"* wird der Beginn des Inhalts der HTML-Seite gekennzeichnet. Als nächstes inneres Element wäre da der Kopf der Seite, welcher durch das Attribut *data-role="header"* markiert wird.

Jeder Header besitzt zum einen die Desktop-Ansicht, die lediglich eine Überschrift und die Möglichkeit zum Logout gibt.

<img src="./css/images/icons/icon-menu-logout.png"

ng-click="sharedScope.user.logout()"

ng-show="sharedScope.state.user.loginRequired">

Abbildung : Einbinden des Logout Icon

Dieses Stück Code, zeigt, wie das Logout-Icon als HTML-Element eingefügt und direkt mit Bedingungen und Aktionen über AngularJs verknüpft wird. Bei nicht gesetztem Passwort muss und kann kein Logout erfolgen und somit wird der Button in diesem Fall nicht benötigt, also auch nicht eingeblendet. Über das *ng-show* ist an die Variable *loginRequired* gebunden und nur wenn diese auf *true* gesetzt ist, wird der Button auch angezeigt.

Zum anderen gibt es den Tablet- oder Smartphonemodus, der bei einer Pixelbreite von 1100px oder kleiner eintritt. Hier wird ein Navigationsmenü aus Icons angezeigt, welches den schnellen Seitenwechsel erlaubt.

mobile navileiste.png

Abbildung :Navigationsmenü für Tablet und Smartphone

Die Responsivität der Seite wurde in diesem Fall durch zwei kurze CSS-Klassen gelöst, die je nach Bildschirmgröße ihre Attributwerte verändern. Beispielsweise soll die Navigationsleiste, die nur im Tablet- oder Smartphonemodus im Header erscheinen soll, sonst nicht in der Form auftreten. Für genau diese Fälle, in denen ganze Elemente verschwinden müssen, wurde die CSS Klasse *tablet\_show* geschrieben. Für alle Bildschirmgrößen über 1100px enthält die Klasse *tablet\_show* das Attribut *display:none*. Sobald die Bildschirmgröße allerdings darunter fällt, so überschreibt das Attribut *display:block* das vorherige und die Navigationsleiste wird angezeigt. Das gleiche Prinzip wird auch beim normalen Desktop-Header angewandt, allerdings genau anders herum mit der Klasse *tablet\_hide*.

.tablet\_hide{

display: block;

}

@media screen and (max-width:1100px) {

.tablet\_hide{

display: none;

}

}

Abbildung : CSS-Klasse in Abhängigkeit von der Bildschirmgröße

Der zweite Teil der Seite wird mit dem Start-Attribut *data-role="content"* angekündigt. Diesem wird zusätzlich zum normalen sharedController noch ein weiterer Controller übergeben, damit der Zugriff auf die spezifischen Funktionen, die im Inhalt benötigt werden ohne Beeinträchtigung funktioniert. Der grundsätzliche Aufbau des Inhalts wurde in mehrere Container unterteilt, damit es trotz vieler Elemente eine übersichtliche Seite bleibt. Dazu gehört zunächst der Hauptcontainer, welcher im gleichen HTML-Element definiert wird wie das eröffnende data-Attribut.

Der so genannte *content\_container* umschließt alle Inhalte und ist auf eine maximale Breite von 1100px beschränkt.

.content\_container{

max-width: 1100px;

margin: auto;

text-align: center;

}

Abbildung : CSS-Klasse des inhaltumfassenden Containers

Er ordnet sich automatisch zentriert an und lässt sich wieder in zwei Untercontainer aufbrechen (siehe Abbildung 19). Der erste Untercontainer befindet sich auf der linken Seite und dient lediglich der Navigationsdarstellung im Desktopmodus. Aus diesem Grund ist eine maximale Breite von 250px mehr als ausreichend. Dieser Container soll durch Scrollen nicht verschwinden und bekommt daher zusätzlich das Attribut *fixed*.

.content\_left\_subcontainer{

max-width:250px;

float: left;

position: fixed;

}

Abbildung : CSS-Klasse des Navigationscontainers

In allen anderen Modi wird der linke Untercontainer nicht angezeigt, sondern nur der zweite, welcher sich auf der rechten Seite befindet. Die maximale Breite des rechten Untercontainers liegt bei 800px und reicht trotzdem zum Anzeigen aller wichtigen Informationen aus, da nach dem Mobile-First Konzept gearbeitet wurde und es leichter ist eine Webseite von 300px Breite auf 800px Breite zu erweitern.

.content\_right\_subcontainer{

max-width: 800px;

width:100%;

margin: auto;

float: right;

}

Abbildung :CSS-Klasse des internen Inhaltscontainers

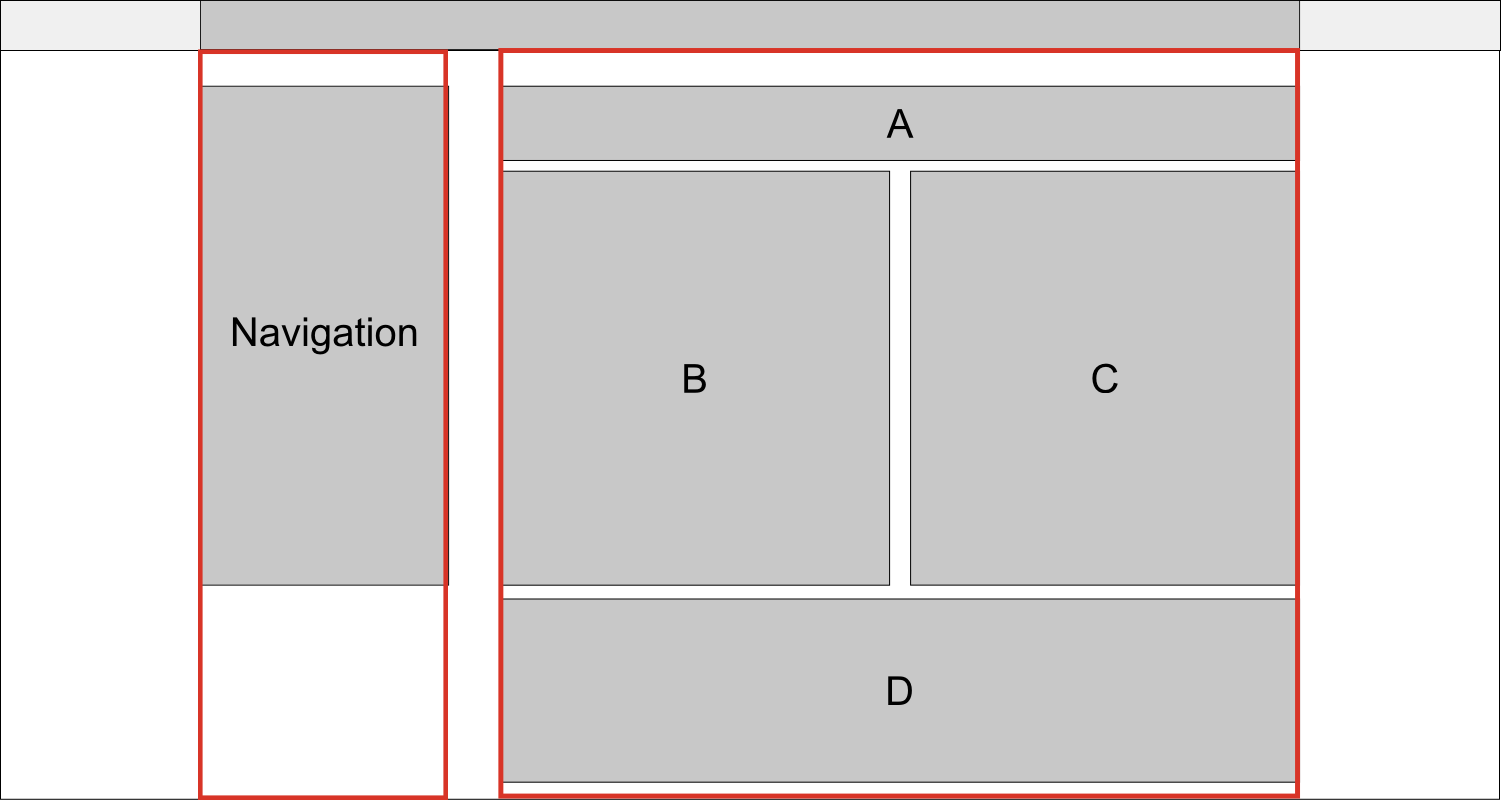


Abbildung 19: Containereinteilung im Desktopmodus

Um die Möglichkeiten der deutlich breiteren Seite bei einem großen Tablet und Desktop-PC auch zu nutzen, wurden die Inhalte in verschiedene Gruppen zusammengefasst, die sich bei Veränderung des Browserfensters oder des Endgeräts fließend umeinander bewegen. Es gibt vier CSS-Klassen, die sich unterschiedlich im rechten Untercontainer verhalten. Zunächst wären da die Klassen A und D, welche die gesamte zur Verfügung stehende Breite des rechten Containers nutzen.

.subcontent\_a{

width: 100%;

overflow:hidden;

padding-top:10px;

padding-bottom:10px;

}

.subcontent\_d{

width: 100%;

overflow:hidden;

}

Abbildung : CSS-Klassen der Untercontainer A und D

Die Höhe dieser Klassen hängt allein vom Inhalt ab, welcher je nach Seite variiert. Der Unterschied zwischen den beiden Klassen liegt im Abstand, den Klasse A von der Kopfzeile haben sollte. Für Klasse D muss der Abstand zum darüber liegenden Element nicht so groß sein. Die Klasse B und C ermöglich das vertikale Aufteilen des rechten Untercontainers in zwei weitere Teile und somit eine konzentriertere Darstellung des Inhalts.

Da bei der Tabletansicht der einzige Unterschied darin liegt, dass die Navigationsleiste in den Header wandert um Platz zu sparen und der linke Untercontainer komplett verschwindet, bleibt genug Platz, um den rechten Untercontainer genau so anzuzeigen wie auf dem Desktop. Dies ist nicht nur programmiertechnisch ein sehr geringer Aufwand, sondern auch für die Benutzerfreundlichkeit ein Gewinn, da der Benutzer sich nur in einem kleinen Punkt umgewöhnen muss. Bei der Smartphoneansicht ist das Ganze schon etwas komplexer, da hier die Container B und C nicht nebeneinander angezeigt werden können. Die Lösung lässt sich allerdings durch CSS3 Media-Queries herbeiführen: Bei einer Breite unter 550px nehmen auch die Elemente B und C die gesamte zur Verfügung stehende Breite ein und platzieren sich untereinander.

.subcontent\_b{

width: 49%;

overflow:hidden;

float: left;

}

.subcontent\_c{

width: 49%;

overflow:hidden;

float: right;

}

@media screen and (max-width:550px) {

.subcontent\_b{

width: 100%;

}

.subcontent\_c{

width: 100% ;

float:left;

padding-top: 5px;

}

}

Abbildung : CSS-Klassen Untercontainer B und C im Vergleich Smartphone und normale Ansicht

Auch das Element D bekommt in dieser Ansicht eine neue Position: es wird als eine Art Überlagerung dargestellt, das heißt es legt sich über alle anderen Elemente und kann dann bearbeitet werden. Dies wird allerdings nicht durch eine Änderung der CSS-Klasse in den Media Queries hervorgerufen, sondern muss durch eine Codeduplizierung gelöst werden. Es gibt zwei Implementierungen des Inhalts von Untercontainer D, einmal die Implementierung für die Desktop- und Tabletansicht und zum anderen die Implementierung für die Smartphoneansicht. Auch wenn Codeduplikate keine saubere Lösung darstellen, ist es hier gut einsetzbar, da sich nicht nur die CSS-Eigenschaften der HTML-Elemente ändern, sondern ganze HTML-Elemente hinzukommen und verschwinden. Somit ist ein Teil des Codes zwar dupliziert, allerdings wirklich nur ein kleiner Teil. Diese Art von Überlagerung wird nur bei so genannten Untermenüs eingesetzt, auf welche später noch eingegangen wird.

Der Vorteil der sich durch unterschiedliche Darstellungen des Untercontainers D bietet ist, dass der Benutzer nicht lange Scrollen muss um an die gewünschten Informationen oder Funktionalitäten zu kommen. Der Nachteil neben dem Codeduplikat ist, dass der Benutzer mit diesem Verhalten nicht rechnen wird und der Wiedererkennungswert ist leicht gefährdet. Zudem ist diese Lösung nur empfehlenswert, wenn zusätzliche Informationen erfragt werden.

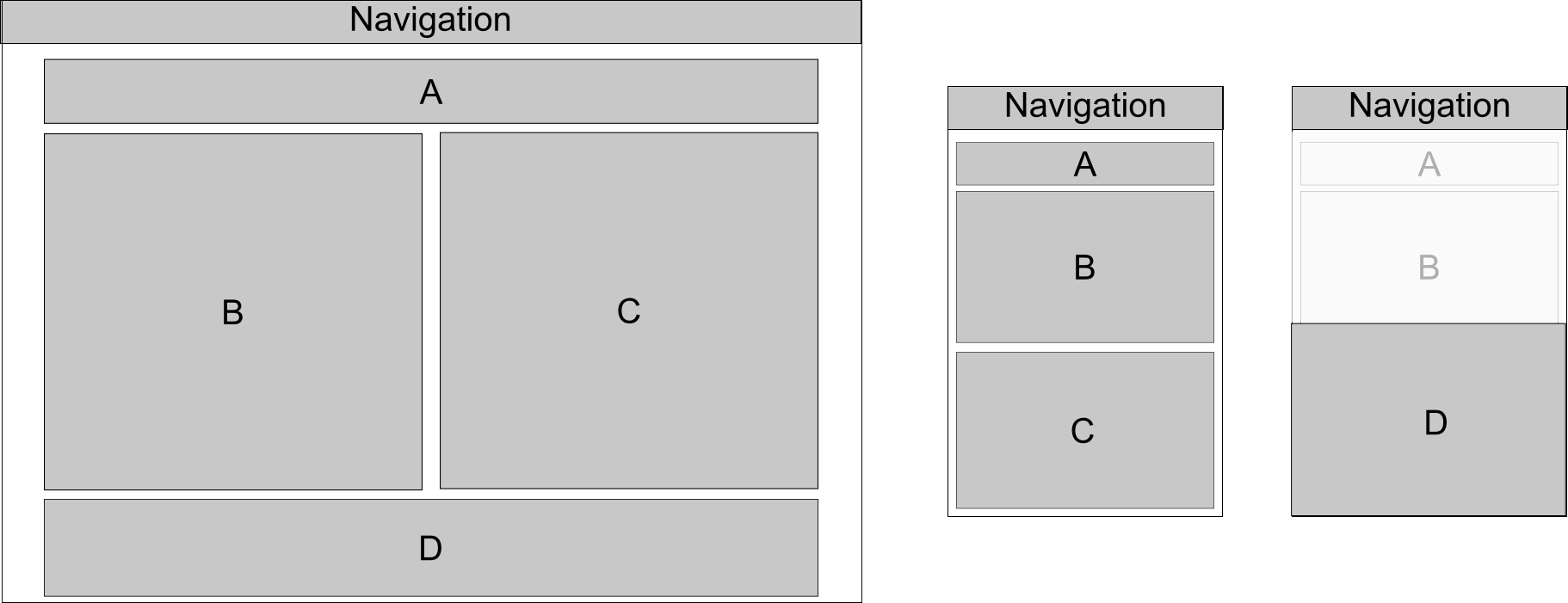


Abbildung : Containereinteilung im für Tablet und Smartphone

### Implementierungskonzept

Dieses Kapitel stellt verschiedene Anwendungsfälle und Implementierungen an Beispielen vor.

#### Controller

Eine Startseite für eine geringe Größe anzufertigen nimmt nicht viel Zeit in Anspruch, da bereits ein einfaches Menü in Form einer Liste reicht, um dem Benutzer verschiedene Aktionsmöglichkeiten aufzuzeigen.

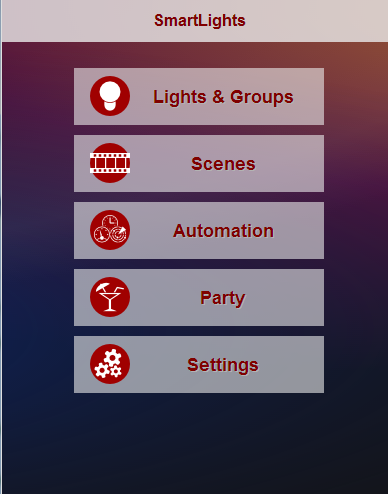


Abbildung : Startseite für Tablet und Smartphone

Da für die Desktopansicht eine Liste mit fünf Menüpunkten zu leer ist und viel Platz ungenutzt bleibt, wurde hier eine Weiterleitung zum nächst sinnvollen Menüpunkt hergestellt. Diese Weiterleitung wurde mit einem minimalistischen Controller implementiert. Ein Controller wird durch das Attribut *ng-controller="IndexCtrl"* für einen HTML-Abschnitt definiert. Der Inhalt des zugewiesenen Controllers wird bei jedem Aufruf der Seite ausgeführt.

*controller*('IndexCtrl', [function() {

if(*window*.innerWidth > 1100){

*window*.location.href = 'lightandgroup.html';

}

}])

Abbildung : Implementierung des IndexCtrl-Controllers

Das heißt, es reicht in diesem Controller zu testen, wie groß der Bildschirm ist und bei Bedarf eine Weiterleitung auf eine andere Seite festzulegen, sollte die Bildschirmgröße die eines Tablets überschreiten. So kann auch ohne direkte Anwendung von responsivem Design eine Webseite an die Gegebenheiten angepasst werden. Diese Vorgehensweise wird auch beim Menüpunkt Settings angewandt, damit keine nur teils gefüllte Seite in der Desktopansicht auftritt und der vorhandene Platz genutzt wird. Der Vorteil für den Benutzer ist ein geringerer Klickaufwand, den er betreiben muss um zu seiner Zielseite zu gelangen.

#### Angular Attribute und DataBinding

Auf der Übersichtsseite der Lichter und Gruppen werden einem die angeschlossenen Lampen angezeigt. Dazu wird das Angular-Attribut *ng-repeat* genutzt. Das Attribut benötigt ein Array oder ein Objekt, welche es dann nacheinander durchgeht und jeweils angeforderte Daten zurückgibt. In diesem Beispiel werden die Lampen aufgelistet und das *ng-repeat*-Attribut gibt zu jeder Lampe, zum einen den Schlüssel des Objekts und zum anderen das Objekt der Lampe selbst zurück. Über Data-Binding wird der Name der Lampe angezeigt. Data-Binding wir in Angular über doppelt geschweifte Klammern gekennzeichnet.

<div ng-repeat="(id, light) in sharedScope.state.lights">

<a href="light.html?id={{id}}">

<div class="light\_wrapper\_69width">

<div hueper-color="sharedScope.state.lights[id].state"></div>

<div class="light\_template">

<img src="./css/images/transparent\_light\_69width.png">

</div>

</div>

{{light.name}}

</a>

</div>

Abbildung : Implementierung eines ng-repeat

Durch Klick auf eine Lampe oder deren Namen kommt man in das Lampenmenü. Die Information, für welche Lampe das Lampenmenü geöffnet werden soll, wird über das Anhängen des Lampenschlüssels in der URL geregelt. Dies geschieht ebenfalls über Data-Binding. In Zeile 4 des Codebeispiels (Abbildung 25) wird zudem eine Direktive angewandt, um die aktuelle Farbe der gerade im Objekt durchlaufenden Lampe zu bekommen. Die Direktive *hueperColor* hat kein HTML-Template und wird auf frei definierbare Tags angewandt, wie im Beispiel hier, kann ein *div* damit manipuliert werden. Die Direktive überwacht den Lampenstatus, des im HTML-Code mitgegebenen Models und gibt dem HTML-Tag, in dem es eingebunden wurde die Lampenfarbe als Hintergrundfarbe. Sollte der Effekt „Colorloop“ aktiviert sein, so wird entsprechend ein Hintergrundbild mit Farbverlauf eingefügt. Da die Anzeige der Lampenfarbe als Hintergrund mehrfach gebraucht wird, lohnt es sich diese als Direktive zu schreiben. Ein weiterer Vorteil, neben der Wiederverwendbarkeit, ist die Übersichtlichkeit des HTML-Codes.

#### Untermenü und Submenu

Im Szenenmenü können Szenen direkt gelöscht werden. Als Sicherheitsfrage öffnet sich im Untercontainer D (der im Kapitel 4.3.2 bereits angesprochen wurde) ein so genanntes Untermenü, welches den Benutzer nochmals fragt, ob wirklich die Szene gelöscht werden soll. Erst bei erneutem Bestätigen des Löschvorgangs wird die Funktion *remove* aus dem *SceneCtrl* aufgerufen. Diese Funktion schickt einen Befehl an das Backend, die Szene mit der mitgegebenen ID zu löschen. Ist dies geschehen, so wird das Untermenü wieder ausgeblendet.

remove: **function**(id) {

socket.emit(**'scene.delete'**, id);

**delete** $scope.**state**.**scenes**[id];

$scope.sharedScope.**submenu**.closeSubmenu();

},

Abbildung 26: remove-Funktion aus dem SceneCtrl

Das Ein- und Ausblenden von Untermenüs wird durch die Submenu-Funktion gesteuert. Es gibt zwei Funktionen, die das Öffnen und das Schließen der Untermenüs veranlassen.

$scope.**submenu** = {

**visible**:{},

openSubmenu: **function**(menuname){

$scope.**submenu**.**visible** = {};

**if**(**angular**.isArray(menuname)){

**angular**.forEach(menuname, **function**(value){

$scope.**submenu**.**visible**[value] = **true**;

});

}**else**{

$scope.**submenu**.**visible**[menuname] = **true**;

}

},

closeSubmenu: **function**(){

$scope.**submenu**.**visible** = {};

}

};

Abbildung : Implementierung der Submenu Funktionen

Ein Submenu ist ein HTML-Abschnitt, der nicht ohne Aktivierung angezeigt wird. Der äußerste HTML-Tag des Submenu enthält das *ng-if* Attribut, welches einen Wert enthalten muss, der auf *true* gesetzt ist, um angezeigt zu werden.

<div ng-if="sharedScope.submenu.visible.deleteScene"> … </div>

Abbildung : Einsatz eines Submenu

Das heißt, beim Klick auf das Löschen-Symbol in der Szenenübersicht wird der Wert von *sharedScope.submenu.visible.deleteScene* auf *true* gesetzt und die HTML-Tags, die das *ng-if* enthalten werden samt Inhalt angezeigt. Das Submenu ist, vom Layout her, für die Desktop- und Tabletansicht umgesetzt, wie auch die Untermenüs A, B und C. In der Smartphoneansicht öffnet sich das neue Feld als Überlagerung über der aktuellen Ansicht. Der Hintergrund wird leicht ausgegraut und das neue Feld wird so hervorgehoben.

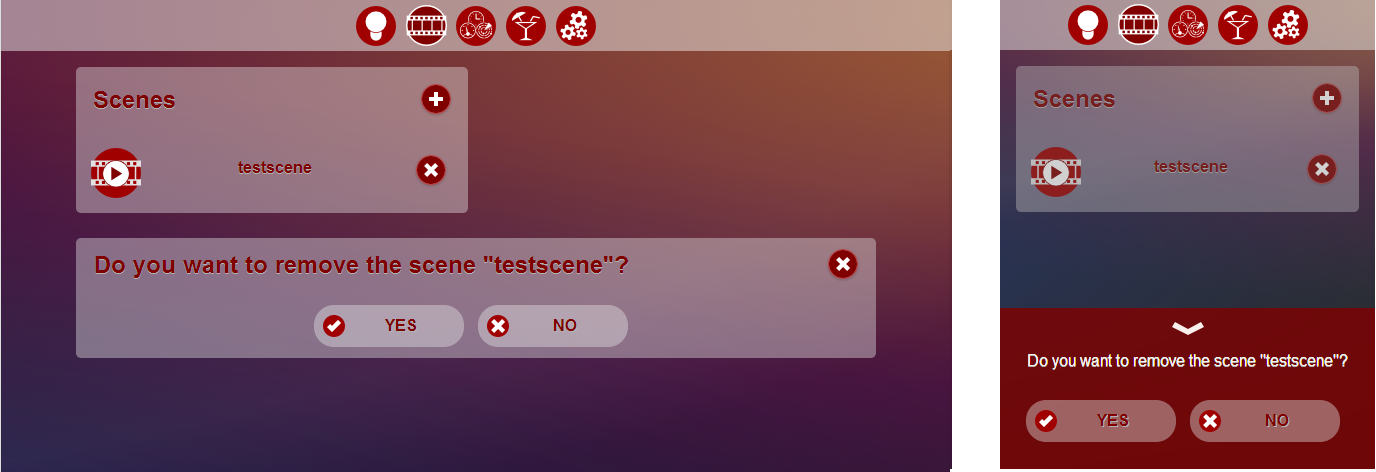


Abbildung : Vergleich Tablet und Smartphone Submenu

Zur Umsetzung dieses zweiten Layouts wurden in den äußeren Tag des Submenu nicht nur die Desktop-/Tabletansicht implementiert, sondern zusätzlich die Smartphoneansicht. Beide Ansichten werden von Containern umrahmt, welche über die CSS-Klassen *mobile\_show* und *mobile\_hide* jeweils entsprechend ein- oder ausgeblendet werden.

<div ng-if="sharedScope.submenu.visible.deleteScene">

*<!-- desktop/tablet view -->*

<div class="lightgroup-all-container transparent\_bg mobile\_hide"> … </div>

*<!-- mobile view -->*

<div class="overlap\_content mobile\_show"> … </div>

</div>

Abbildung : verschiedene Submenu Implementierungen

Die Funktionsweise der Klassen kann analog zu den CSS-Klassen *tablet\_show* und *tablet\_hide (siehe auch* Abbildung 15*)* gesehen werden, der einzige Unterschied liegt in der Media Query, in der sich das display-Attribut ändert. Die CSS-Klasse tablet\_\* wird bereits bei einer Bildschirmgröße von 1100px und kleiner verändert, wobei das display-Attribut der mobile\_\* Klasse erst bei 550px und kleiner modifiziert wird.

#### Direktiven

Bei einigen Funktionalitäten, die das Frontend zur Verfügung stellt, besteht die Möglichkeit eine Farbe auszuwählen, meist in einem umfassenden Menü, in dem nicht nur die Farbe selbst eingestellt werden kann, sondern auch die Helligkeit und ein Effekt.

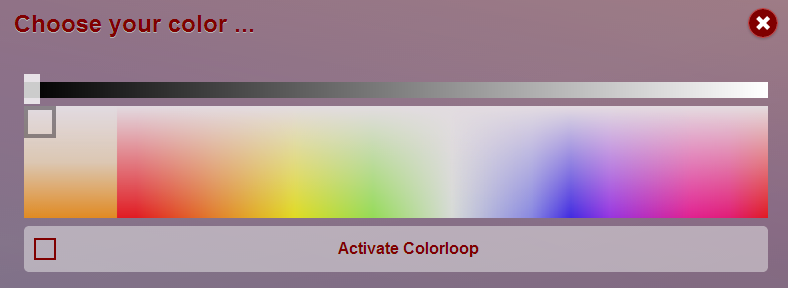


Abbildung : Farbauswahl Direktive

Diese Auswahl wird über drei Direktiven implementiert, die die Lesbarkeit des HTML-Codes deutlich besser lesbar machen. Dieses Stück Code (Abbildung 32) zeigt besonders gut, wie unterschiedlich die Direktiven implementiert sein können. Die Grundstruktur ist jedoch bei allen gleich: es gibt ein Model, auf das sich die Direktive bezieht. Beim *hueperSlider* ist die Helligkeit das Model, dessen Wert direkt angezeigt wird und sich mit dem Bewegen des Sliders verändert. Der *hueperColorpicker* bezieht seine Werte aus dem Status einer Lampe. Da dieser Status aus bis zu neun Werten bestehen kann, ist es deutlich weniger kompliziert eine Direktive zu implementieren, der das gesamte Objekt mitgegeben werden kann. So bleibt der komplexe Teil im Code der Direktive und erleichtert das Verständnis des HTML-Codes.

<div hueper-slider model="sharedScope.state.lights[helpers.urlId].state.bri"

change="lights.state(helpers.urlId, { bri: value })" min="0" max="254"

step="5">

</div>

<div hueper-colorpicker state="sharedScope.state.lights[helpers.urlId].state"

change="lights.state(helpers.urlId, state)">

</div>

<div hueper-checkbox="sharedScope.state.lights[helpers.urlId].state.effect"

change="lights.state(helpers.urlId, {effect : value})" trueval="colorloop"

falseval="none" >

Activate Colorloop

</div>

Abbildung : Aufruf der Farbauswahl-Direktiven

Die *hueperCheckbox* bezieht sich im Beispiel hier auf eine Variable im Lampenstatus, den Effekt. Die Direktive besteht aus einem Template, welches das Layout und Design der Checkbox beschreibt. Im Template wird das Angular-Attribut *ng-show* verwendet, um den Haken in der Checkbox ein- und auszublenden. Da nicht alle Variablen mit zwei möglichen Inhalten diese auch true und false nennen, weil diese Bezeichnungen zum Beispiel nicht auf den Variablennamen oder das Attribut passen. Gibt es die Möglichkeit den Wert für *true* und *false* auch auf andere Bezeichnungen zu setzen. Die Übersetzung der Bezeichnung „colorloop“ auf den Wert *true* wird in der Direktive übernommen. So kann die Direktive auch auf Model angewendet werden, die kein *true* und *false* benutzen. Die *hueperCheckbox* kann auch mit Arrays arbeiten und liefert *true* zurück, wenn ein Objekt in einem Array enthalten ist und *false*, wenn es das nicht ist.

Das Attribut *ng-click*, das oft benutzt wird, um eine Funktion beim Klick auf ein HTML-Element aufzurufen, wird bei einer Direktive nicht benötigt, weil es die Möglichkeit hat intern zu definieren, was bei einem Mausklick oder eben Fingertap auf die Direktive passieren soll. Am Beispiel der *hueperCheckbox* lässt sich dies gut zeigen, da bei einem Klick nur der Status der Direktive von *true* auf *false* oder umgekehrt geändert werden soll und keinen komplexen Funktionsablauf hat. Zunächst wird geprüft, ob es sich beim Model um ein Array oder eine normale Variable handelt. Bei einer Variable kann direkt der neue Wert gesetzt werden. Sollte es sich aber um ein Array handeln, so muss das Model dem Array entweder hinzugefügt oder gelöscht werden.

elm.on('click', function(){

scope.$apply(function(){

var id = *parseInt*(scope.hueperCheckbox);

if(scope.list === undefined){

if(scope.hueperCheckbox === trueVal){

scope.hueperCheckbox = falseVal;

}

else{

scope.hueperCheckbox = trueVal;

}

}

else{

if(scope.list.indexOf(id) > -1){

scope.list.splice(scope.list.indexOf(id), 1);

}

else{

scope.list.push(id);

}

}

scope.change({value: scope.hueperCheckbox});

});

});

Abbildung : Auszug aus der hueperCheckbox-Direktive

#### Services

Es gibt zwei essentielle Services, die für die Benutzung der WebApp eine große Rolle spielen. Da wäre zum einen der Socket-Service, welcher sich um die Verbindung mit dem Backend über Socket.IO kümmert.

Der zweite wichtige Service ist der StateManager-Service. Dieser besitzt ein clientseitiges Datenmodell, welches durch Nachrichten vom Backend den Lampenstatus synchronisiert. Beim DataBinding ist die View mit dem vom StateManager bereitgestellten Model verknüpft. Sollte also das Modell des StateManagers mal nicht auf dem aktuellen Status sein, so zeigt auch die View falsche Informationen an.

# Ausblick

Die Studienarbeit bietet vielfältige Möglichkeiten, die implementierte Anwendung in verschiedene Richtungen weiterzuentwickeln.

Durch die Unabhängigkeit von Backend und Frontend ist es möglich, parallel mehrere verschiedene Web-Frontends zu betreiben. So könnten etwa spezielle Frontends optimiert für Desktop-Computer oder Smart-TVs entworfen werden, die zusätzlich zu dem bereits entwickelten Frontend eingesetzt werden können.

Mit der REST-Schnittstelle des Backends ist die Möglichkeit vorhanden, die Anwendung auch durch andere Software oder auch Netzwerk-fähige Hardware fernsteuern zu können. Denkbar wären etwa folgende Szenarien:

* WLAN-Lichtschalter, Dimmer oder Farbregler
* Einbinden in bestehende Heimautomatisierungs-Systeme
* Kontrolle durch Licht-Steuersysteme wie DMX
* Native Apps für spezielle Geräte wie z.B. Smartwatches

Durch die flexible Architektur des Automatisierungs-Bereichs können ohne großen Aufwand neue Trigger, Bedingungen und Aktionen hinzugefügt werden. Dies eröffnet vielfältige Möglichkeiten zum Erweitern des Systems:

* Ausstatten des Arduino mit weiteren Sensoren und Umwelt-Messgeräten
* Ausdehnen auf andere Bereiche als nur das Steuern von Philips Hue-Leuchten bis; durch Relais-Steuerungen könnten etwa beliebige Geräte ein- und ausgeschaltet werden

Durch das Einbinden weiterer Controller kann die bestehende Funktionalität des Backends ebenfalls sehr einfach erweitert werden.

# Fazit

Die geschaffene Software erweitert den bestehenden Funktionsumfang der Philips Hue-Bridge um viele nützliche Aspekte. Einerseits kann sie nun plattformunabhängig von verschiedenen Geräten aus gesteuert werden, andererseits bietet sie vielfältige Möglichkeiten zur Automatisierung und erweiterten Steuerung.

Die Kombination aus einem Node.JS-Server mit einer MongoDB als Datenbank und einer Socket.IO-Schnittstelle zum Frontend erwies sich als durchweg geeignet für die Entwicklung des Backends. Damit können bei geringem Ressourcenverbrauch verschiedene Peripheriegeräte und Zusatzprogramme in Echtzeit überwacht und gesteuert werden.

Die MongoDB als Datenbanksystem ist für die Benutzung auf dem Raspberry Pi eher problematisch, da sie die ARM-Plattform nicht von Haus aus unterstützt und man somit auf Portierungen Dritter angewiesen ist, die oft auf veralteten Versionen aufbauen. Auch die Eignung für den Einsatz auf 32bit-Systemen ist auf kleine Datenmengen beschränkt.

Der Raspberry Pi und der Arduino haben beide die Schwäche, dass sie über eine relativ geringe Rechenleistung verfügen. Dies schränkt etwa die Möglichkeiten bei der Spracherkennung auf dem Raspberry Pi sowie der Takterkennung auf dem Arduino erheblich ein. Hier könnten für ein zukünftiges Projekt eventuell leistungsfähigere Plattformen wie etwa der ODROID U3[[31]](#footnote-31) mit Multicore-Prozessor gewählt werden. Ebenso könnten dadurch die Funktionsbereiche des Raspberry Pi und des Arduino gebündelt werden, wenn eine geeignete Plattform gefunden werden kann, die ausreichend durch Peripherie und Sensoren erweiterbar ist. Eine mögliche Plattform, die diesen Ansatz verfolgt, ist beispielsweise der UDOO[[32]](#footnote-32).

AngularJS hat gerade in Verbindung mit dem Socket.IO-Backend große Stärken, da durch das Data Binding die Änderungen des Backends in Echtzeit auf der Oberfläche angezeigt werden können. Durch das MVC-Prinzip ist eine strikte Trennung von Logik und Darstellung möglich. Zusätzlich kann es die Entwicklung der Applikation erheblich beschleunigen, da dem Entwickler viel Arbeit abgenommen wird.

jQuery Mobile ist vor allem dazu geeignet, mit geringem Aufwand Oberflächen zu entwickeln, bei denen es mehr auf die Funktionalität als auf das Design ankommt. Dafür bietet es konsistente UI-Elemente und ein anpassbares Farbschema. Möchte man allerdings vom vorgegebenen Design abweichen, schränkt es den Entwickler oft ein oder resultiert in einem erhöhten Arbeitsaufwand.

Vor allem die Verbindung zwischen zwischen AngularJS und jQuery Mobile bereitet größere Probleme. Der dafür nötige Adapter beschränkt den Funktionsumfang von AngularJS und verhindert etwa das Benutzen des Routings. Da er nicht mehr aktiv weiterentwickelt wird, ist auch nicht garantiert, dass zukünftige Versionen beider Frameworks noch ohne weiteres zusammenarbeiten werden. AngularJS und jQuery Mobile sind beide für sich genommen sehr nützliche Frameworks, der gemeinsame Einsatz ist allerdings eher nicht ratsam. Eine Alternative für jQuery Mobile zum Gestalten der Oberfläche wäre beispielweise das Bootstrap-Framework[[33]](#footnote-33). Dieses baut im Wesentlichen auf CSS auf und lässt sich daher gut mit AngularJS benutzen. Für interaktive Komponenten wie Dropdown-Menüs und Tab-Interfaces stellt das Angular UI-Projekt spezielle AngularJS-Direktiven bereit[[34]](#footnote-34).

Generell wurden beim Entwickeln des Frontends gute Erfahrungen mit dem Mobile First-Ansatz und Responsive Design im Allgemeinen gemacht. Wichtig hierbei ist vor allem, dass das Konzept bereits zumindest in groben Zügen vorhanden ist, bevor man mit dem Implementieren beginnt.

# Quellenverzeichnis

Adafruit. *Adafruit\_NFCShield\_I2C.* 2013. https://github.com/adafruit/Adafruit\_NFCShield\_I2C.

—. *PN532 RFID/NFC Shield.* 2014. https://learn.adafruit.com/adafruit-pn532-rfid-nfc.

Amarra, Arthur. *Speech Recognition Using The Raspberry Pi.* 26. Mai 2012. http://www.aonsquared.co.uk/raspi\_voice\_control.

Angular-UI Team. *UI Bootstrap.* 2014. http://angular-ui.github.io/bootstrap/ (Zugriff am 06. Juni 2014).

Arduino. *Arduino.* 2014. http://arduino.cc/.

—. *Arduino IDE installation Guide (Windows).* 2014. http://arduino.cc/en/Guide/Windows.

—. *Arduino Uno.* 2013. http://arduino.cc/en/Main/ArduinoBoardUno.

—. *Install on Linux Mint.* 11. Februar 2013. http://playground.arduino.cc/Linux/Mint.

—. *Language Reference.* 2014. http://arduino.cc/en/Reference/HomePage.

Bagwell, Chris. *SoX.* 2013. http://sox.sourceforge.net/.

Banzi, Massimo. *Getting Started with Arduino (2nd Edition).* O'Reilly Media, 2011.

Bauer, Ellen. *elma studio.* 14. November 2011. http://www.elmastudio.de/webdesign/webdesign-goes-mobile-first-eine-kleine-einfuhrung-zum-neuen-webdesign-trend/ (Zugriff am Mai 2014).

Buus, Mathias. *network-address.* 2013. https://github.com/mafintosh/network-address.

Carnegie Mellon University. *CMU Sphinx - Speech Recognition Toolkit.* 2014. http://cmusphinx.sourceforge.net/.

Express.js. *Express.js.* 2014. http://expressjs.com/.

—. *Express.js 3.x documentation.* 2014. http://expressjs.com/3x/api.html.

Firtman, Maximiliano. *jQuery Mobile Up and Running.* USA: O'Reilly Media, 2012.

Git. *Git.* 2014. http://git-scm.com/.

Google Inc. *AngularJS.* 2014. http://angularjs.org/.

—. *AngularJS 1.2.16 documentation.* 2014. http://code.angularjs.org/1.2.16/docs/api.

Hackhappy. *How to use a Raspberry Pi to create a wireless to wired network bridge.* 2013. http://hackhappy.org/uncategorized/how-to-use-a-raspberry-pi-to-create-a-wireless-to-wired-network-bridge/.

Hardkernel co., Ltd. *ODROID-U3.* 2014. http://www.hardkernel.com/main/products/prdt\_info.php?g\_code=G138745696275 (Zugriff am 06. Juni 2014).

Herbison, Tom. *Adding an Audio Input Device.* 11. Februar 2013. http://asliceofraspberrypi.blogspot.de/2013/02/adding-audio-input-device.html.

JetBrains s.r.o. *WebStorm.* 2014. https://www.jetbrains.com/webstorm/.

Joyent, Inc. *NodeJS.* 2014. http://nodejs.org/.

—. *NodeJS v0.10.26 Maual & Documentation.* 2014. http://nodejs.org/dist/v0.10.26/docs/api/.

Knight, Kayla. *Responsive Web Design: what it is and how to use it.* 12. Januar 2011. http://www.smashingmagazine.com/responsive-web-design-guidelines-tutorials/#a2 (Zugriff am Mai 2014).

Koninklijke Philips Electronics N.V. *Meet hue.* 2014. http://meethue.com.

—. *Philips Hue API.* 2013. http://developers.meethue.com/.

Kyoto University; Nagoya Institute of Technology. *Open-Source Large Vocabulary CSR Engine Julius.* 2014. http://julius.sourceforge.jp/en\_index.php.

LearnBoost. *Mongoose.* 2014. http://mongoosejs.com/.

—. *Mongoose documentation.* 2014. http://mongoosejs.com/docs/.

—. *Socket.IO.* 2013. http://socket.io/.

Lyon, Gordon. *Nmap.* 2013. http://nmap.org/.

McLean, Ken. *VoxForge - Free Speech Recognition.* 2014. http://www.voxforge.org/.

—. *VoxForge - Task Grammar.* 2014. http://www.voxforge.org/home/dev/acousticmodels/windows/create/htkjulius/tutorial/data-prep/step-1.

Mintert, Stefan. „Das gute Buch.“ *iX Magazin für professionelle Informationstechnik*, Mai 2014: 114-119.

MongoDB, Inc. *MongoDB.* 2014. http://www.mongodb.org/.

—. *The MongoDB 2.4 Manual.* 2013. http://docs.mongodb.org/v2.4/.

Morin, Brice. *MongoDB binaries for Raspberry Pi.* 2013. https://github.com/brice-morin/ArduPi/tree/master/mongodb-rpi.

Murray, Peter. *node-hue-api.* 2013. https://github.com/peter-murray/node-hue-api.

Navarrete, Jason. *The SoX of Silence.* 25. August 2009. http://digitalcardboard.com/blog/2009/08/25/the-sox-of-silence/.

npm, Inc. *npm.* 2014. https://www.npmjs.org/.

Patin, Frédéric. *Beat Detection Algorithms .* 07. Juni 2003. http://www.gamedev.net/page/resources/\_/technical/math-and-physics/beat-detection-algorithms-r1952.

PingBin. *How To: WiFi your Raspberry Pi.* 23. Dezember 2012. http://pingbin.com/2012/12/setup-wifi-raspberry-pi/.

Raspberry Pi Foundation. *Raspberry Pi.* 2014. http://www.raspberrypi.org/.

Reuter, S. *node-speakable.* 2013. https://github.com/sreuter/node-speakable.

Rhodes, Michael. *ipv4-range.* 2013. https://github.com/michaelrhodes/ipv4-range.

Robbins, Charlie. *Forever.* 2014. https://github.com/nodejitsu/forever.

Rüedlinger, Matthias. *Raspberry Pi and Node.JS: Basic Setup.* 31. März 2013. http://blog.rueedlinger.ch/2013/03/raspberry-pi-and-nodejs-basic-setup/.

SECO USA Inc. *UDOO.* 2014. http://www.udoo.org/ (Zugriff am 06. Juni 2014).

Seeed Technology Inc. *Seeedstudio Grove Base Shield.* 19. November 2013. http://www.seeedstudio.com/wiki/index.php?title=Grove\_-\_Base\_Shield\_V1.3&oldid=50946.

—. *Seeedstudio Grove Light Sensor.* 30. Dezember 2013. http://www.seeedstudio.com/wiki/index.php?title=Grove\_-\_Light\_Sensor&oldid=66065.

—. *Seeedstudio Grove PIR Motion Sensor.* 30. Dezember 2013. http://www.seeedstudio.com/wiki/index.php?title=Grove\_-\_PIR\_Motion\_Sensor&oldid=66057.

—. *Seeedstudio Grove Sound Sensor.* 04. September 2013. http://www.seeedstudio.com/wiki/index.php?title=Grove\_-\_Sound\_Sensor&oldid=37564.

The jQuery Foundation. *jQuery.* 2014. http://jquery.com/.

—. *jQuery Mobile.* 2014. http://jquerymobile.com/.

The Thing System, Inc. *node-arp-a.* 2014. https://github.com/TheThingSystem/node-arp-a.

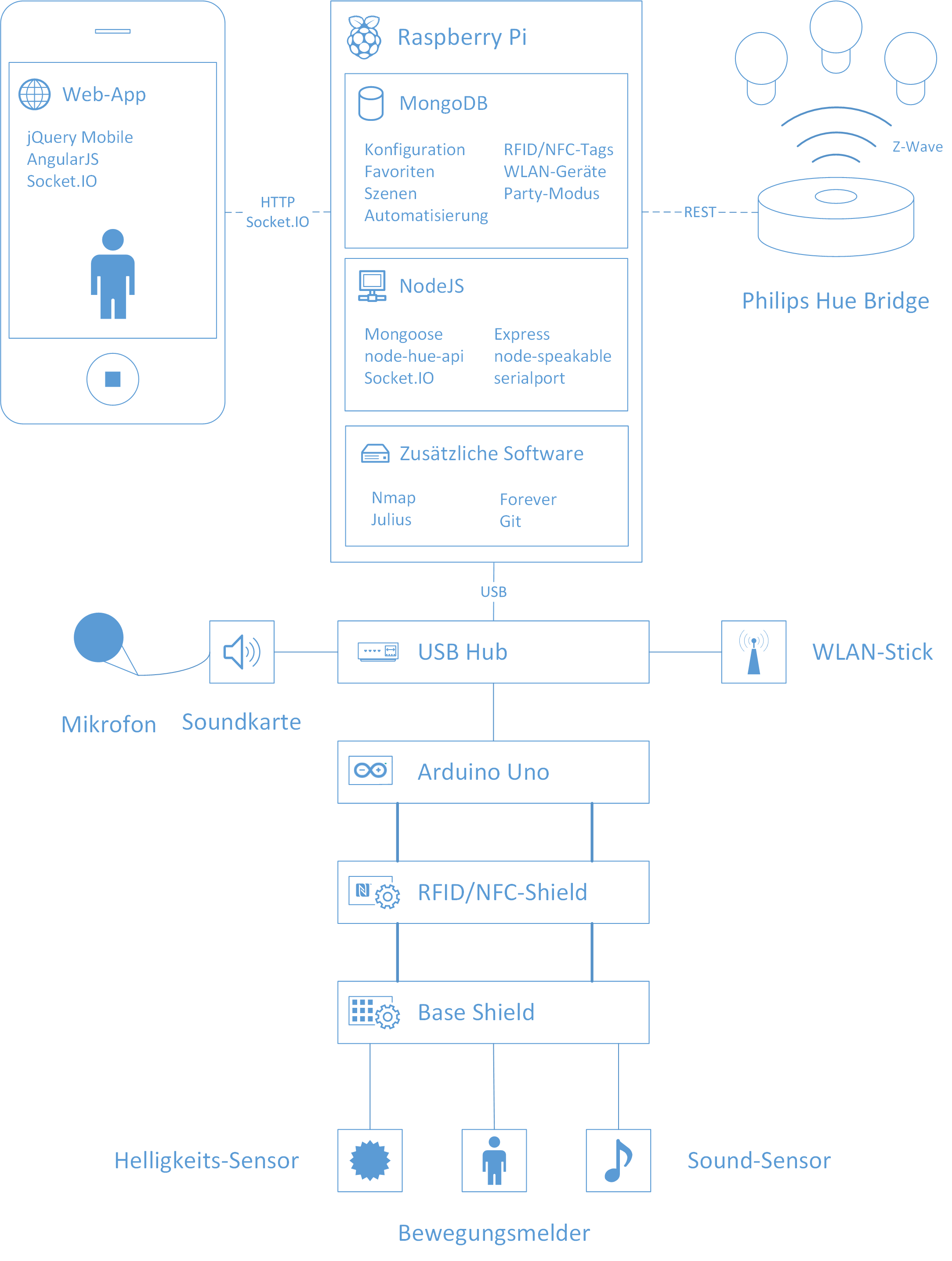
Twitter, Inc. *Bootstrap.* 2014. http://getbootstrap.com/ (Zugriff am 06. Juni 2014).

Wahlin, Dan. *AngularJS in 60 Minutes.* Wahlin Consulting, 2013.

Williams, Chris. *node-serialport.* 2014. https://github.com/voodootikigod/node-serialport.

Zelisko, Daniel. *node-ping.* 2014. https://github.com/danielzzz/node-ping.

# Anhang A: Architektur-Übersicht



# Anhang B: Hardware

Preise ungefähr; Stand Ende 2013

**Philips Hue Starter Pack** – 200€

Website: <http://meethue.com/>

Online-Shop: <http://www.reichelt.de/iPad-App-faehiges-Zubehoer/PHILIPS-HUE-SP/3/index.html?&ACTION=3&LA=446&ARTICLE=135653&GROUPID=6332&artnr=PHILIPS+HUE+SP>

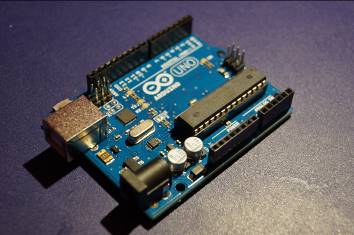
**Raspberry Pi** – 40€



Website: <http://raspberrypi.org>

Online-Shop: <http://www.exp-tech.de/Mainboards/raspberry-pi.html>

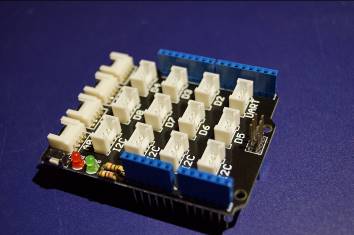
**Arduino Uno** – 24€



Website: <http://arduino.cc/en/Main/arduinoBoardUno>

Online-Shop: <http://www.exp-tech.de/Mainboards/Arduino-Uno-R3.html>

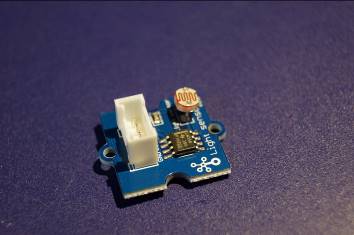
**Seeedstudio Grove Base Shield** – 9€



Website: <http://www.seeedstudio.com/wiki/Grove_-_Base_Shield_V1.3>

Online-Shop: <http://www.exp-tech.de/Shields/Grove-Base-Shield-V1-3.html>

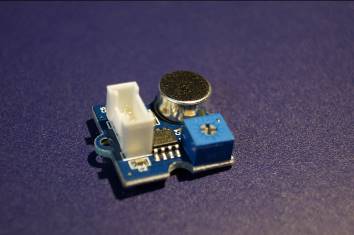
**Seeedstudio Grove Lichtsensor** – 3€



Website: <http://www.seeedstudio.com/wiki/Grove_-_Light_Sensor>

Online-Shop: <http://www.exp-tech.de/Sensoren/Seeed-Studio-Grove-Lichtsensor.html>

**Seeedstudio Grove Sound-Sensor** – 5€



Website: <http://www.seeedstudio.com/wiki/Grove_-_Sound_Sensor>

Online-Shop: <http://www.exp-tech.de/Sensoren/Seeed-Studio-Grove---Sound-Sensor.html>

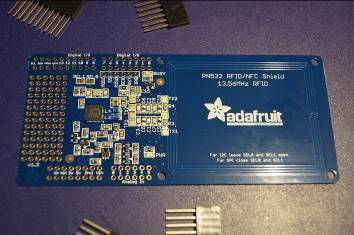
**Seeedstudio Grove PIR Motion Sensor** – 8€



Website: <http://www.seeedstudio.com/wiki/Grove_-_PIR_Motion_Sensor>

Online-Shop: <http://www.exp-tech.de/Sensoren/Seeed-Studio-Grove---PIR-Motion-Sensor.html>

**Adafruit PN532 NFC/RFID Controller Shield** – 37€



Website: <http://learn.adafruit.com/adafruit-pn532-rfid-nfc>

Online-Shop: <http://www.exp-tech.de/Shields/Adafruit-PN532-NFC-RFID-Controller-Shield-for-Arduino.html>

**LogiLink Aktiver USB Hub 4-Port** – 7€



Online-Shop: [http://www.reichelt.de/USB-Hubs/LOGILINK-UA0085/3/index.html?&ACTION=3&LA=446&ARTICLE= 139191&GROUPID=6103&artnr=LOGILINK+UA0085](http://www.reichelt.de/USB-Hubs/LOGILINK-UA0085/3/index.html?&ACTION=3&LA=446&ARTICLE=139191&GROUPID=6103&artnr=LOGILINK+UA0085)

**LogiLink USB-Soundkarte** – 5€



Online-Shop: [http://www.reichelt.de/Soundkarten/LOGILINK-UA0053/3/index.html?&ACTION=3&LA=446&ARTICLE= 132569&GROUPID=6186&artnr=LOGILINK+UA0053](http://www.reichelt.de/Soundkarten/LOGILINK-UA0053/3/index.html?&ACTION=3&LA=446&ARTICLE=132569&GROUPID=6186&artnr=LOGILINK+UA0053)

**Speedlink FAMA Notebook Microphone** – 6€

Online-Shop: <http://www.alternate.de/Speedlink/FAMA_Notebook_Microphone,_Mikrofon/html/product/1105006/>?



**Edimax USB WLAN-Stick** – 9€

Online-Shop: <http://www.reichelt.de/?ARTICLE=99944>

**SanDisk SDHC-Card 8 GB** – 9€



Online-Shop: [http://www.reichelt.de/SD-Karten/SDSDU-008G-U46/3/index.html?&ACTION=3&LA=2&ARTICLE= 124258&GROUPID=4800&artnr=SDSDU-008G-U46](http://www.reichelt.de/SD-Karten/SDSDU-008G-U46/3/index.html?&ACTION=3&LA=2&ARTICLE=124258&GROUPID=4800&artnr=SDSDU-008G-U46)

**USB-B auf USB-A-Adapterkabel** – 1€

Online-Shop: <http://www.reichelt.de/USB-Kabel/AK-672-HSF-0-5/3/index.html?&ACTION=3&LA=2&ARTICLE=63613&GROUPID=6099&artnr=AK+672%2FHSF-0%2C5>

**Transparentes Raspberry Pi-Gehäuse** – 10€

Online-Shop: <http://www.exp-tech.de/Zubehoer/Gehaeuse/Adafruit-Pi-Case-Enclosure-for-Raspberry-Pi-Model-A-or-B.html>

**Netzteil für den Raspberry Pi** – 9€

Online-Shop: <http://www.exp-tech.de/Zubehoer/Netzteil/Micro-USB-Netzteil-5V-DC-1200mA-fuer-Raspberry-Pi.html>

**4-Pin Verbinderkabel** – 5€

Online-Shop: <http://www.exp-tech.de/Shields/Seeed-GROVE-System/Grove---Universal-4-Pin-Buckled-50cm-Cable--5-PCs-pack-.html>

**Arduino Stackable Header Kit** – 2€

Online-Shop: <http://www.exp-tech.de/Zubehoer/Steckverbinder/Arduino-Stackable-Header-Kit-R3.html>

**RFID-Tags** – 1€

Online-Shop: <http://www.reichelt.de/Zutrittskontrollsysteme/GRAND-MF-1-S50/3/index.html?&ACTION=3&LA=446&ARTICLE=124820&GROUPID=3510&artnr=GRAND+MF+1+S50&SEARCH=mifare>

Gesamt-Summe ca. 390€

# Anhang C: Raspberry Pi Setup-Script

#!/bin/bash

*# root check*

if [ "$(whoami)" != "root" ]; then

echo "This script must be run with root privileges!"

echo "Try sudo $0"

exit 1

fi

cd /home/pi

mkdir /var/log/hue

chown pi /var/log/hue

*# Dependency installation*

apt-get -y install git python build-essential sox nmap alsa-tools alsa-oss flex zlib1g-dev libc-bin libc-dev-bin python-pexpect libasound2 libasound2-dev cvs

*# NodeJS setup*

mkdir /opt/node

wget http://nodejs.org/dist/v0.10.26/node-v0.10.26-linux-arm-pi.tar.gz

tar xvzf node-v0.10.26-linux-arm-pi.tar.gz

cp -r node-v0.10.26-linux-arm-pi/\* /opt/node

rm -f -r node-v0.10.26-linux-arm-pi

*# Create symlinks for PATH and root access*

ln -s /opt/node/bin/node /usr/bin/node

ln -s /opt/node/bin/npm /usr/bin/npm

ln -s /opt/node/lib /usr/lib/node

*# Forever setup*

/usr/bin/npm install -g forever

*# MongoDB setup*

mkdir /opt/mongo

git clone https://github.com/brice-morin/ArduPi.git

cp -r ArduPi/mongodb-rpi/mongo/\* /opt/mongo

rm -f -r ArduPi

chmod +x /opt/mongo/bin/\*

mkdir /data

mkdir /data/db

chown pi /data/db

*# add NodeJS and MongoDB to PATH, set ALSADEV for Julius speech recognition*

sed -i '/^export PATH/c NODE\_JS\_HOME="/opt/node"\nexport ALSADEV="plughw:1,0"\nPATH="$PATH:$NODE\_JS\_HOME/bin:/opt/mongo/bin/"\nexport PATH' /etc/profile

*# preserve environment variables when starting NodeJS with sudo*

echo 'Defaults env\_keep += "ALSADEV NODE\_ENV"

' *> /etc/sudoers.d/hue*

chmod 0440 /etc/sudoers.d/hue

echo '#!/bin/bash

### BEGIN INIT INFO

# Provides: MongoDB

# Required-Start: $remote\_fs $syslog

# Required-Stop: $remote\_fs $syslog

# Default-Start: 2 3 4 5

# Default-Stop: 0 1 6

# Short-Description: MongoDB Autostart

# Description: MongoDB Autostart

### END INIT INFO

NAME="MongoDB"

EXE=/opt/mongo/bin/mongod

PARAM="--dbpath /data/db"

USER=pi

OUT=/var/log/hue/mongod.log

ERR=/var/log/hue/mongod.error.log

LOCK=/data/db/mongod.lock

PIDFILE=/var/run/mongod.pid

if [ "$(whoami)" != "root" ]; then

echo "This script must be run with root privileges!"

echo "Try sudo $0"

exit 1

fi

case "$1" in

start)

if [ -s $LOCK ]; then

echo "repairing MongoDB state"

rm $LOCK

sudo -u $USER $EXE $PARAM --repair >> $OUT 2>>$ERR

fi

echo "starting $NAME: $EXE $PARAM"

sudo -u $USER $EXE $PARAM >> $OUT 2>>$ERR &

echo $! > $PIDFILE

;;

stop)

echo "stopping $NAME"

kill $(cat $PIDFILE)

;;

restart)

$0 stop

$0 start

;;

\*)

echo "usage: $0 (start|stop|restart)"

esac

exit 0

' *> /etc/init.d/mongod*

chmod 755 /etc/init.d/mongod

update-rc.d mongod defaults

*# Julius setup*

cd /home/pi

sudo -u pi cvs -z3 -d:pserver:anonymous@cvs.sourceforge.jp:/cvsroot/julius co julius4

cd julius4

JULIUSCFLAGS="-O2 -mcpu=arm1176jzf-s -mfpu=vfp -mfloat-abi=hard -pipe -fomit-frame-pointer"

sudo -u pi CFLAGS="$JULIUSCFLAGS" ./configure --with-mictype=alsa

sudo -u pi CFLAGS="$JULIUSCFLAGS" make

CFLAGS="$JULIUSCFLAGS" make install

*# Project setup*

cd /home/pi

sudo -u pi git clone https://github.com/SBejga/hueper.git

cd hueper/nodejs

sudo -u pi /usr/bin/npm install

*# download VoxForge acoustic model*

cd /home/pi/hueper/julius

wget http://www.repository.voxforge1.org/downloads/Nightly\_Builds/AcousticModel-2014-05-15/HTK\_AcousticModel-2014-05-15\_16kHz\_16bit\_MFCC\_O\_D.tgz

sudo -u pi mkdir acoustic\_model\_files

sudo -u pi tar xvfz HTK\_AcousticModel-2014-05-15\_16kHz\_16bit\_MFCC\_O\_D.tgz -C acoustic\_model\_files

rm HTK\_AcousticModel-2014-05-15\_16kHz\_16bit\_MFCC\_O\_D.tgz

cd /home/pi

echo '#!/bin/bash

### BEGIN INIT INFO

# Provides: Forever NodeJS Hue

# Required-Start: $remote\_fs $syslog

# Required-Stop: $remote\_fs $syslog

# Default-Start: 2 3 4 5

# Default-Stop: 0 1 6

# Short-Description: Forever NodeJS Hue Autostart

# Description: Forever NodeJS Hue Autostart

### END INIT INFO

NAME="Forever NodeJS"

EXE=/usr/bin/forever

SCRIPT=/home/pi/hueper/nodejs/server.js

USER=pi

OUT=/var/log/hue/forever.log

if [ "$(whoami)" != "root" ]; then

echo "This script must be run with root privileges!"

echo "Try sudo $0"

exit 1

fi

case "$1" in

start)

echo "starting $NAME: $EXE $PARAM"

sudo -u $USER $EXE start -a -l $OUT $SCRIPT

;;

stop)

echo "stopping $NAME"

sudo -u $USER $EXE stop $SCRIPT

;;

restart)

$0 stop

$0 start

;;

\*)

echo "usage: $0 (start|stop|restart)"

esac

exit 0

' *> /etc/init.d/forever-hue*

chmod 755 /etc/init.d/forever-hue

update-rc.d forever-hue defaults

*# finished*

echo "Installation complete. Please reboot your device."

# Anhang D: Frontend

Entwürfe

Screenshots

…

1. (Koninklijke Philips Electronics N.V. 2014) [↑](#footnote-ref-1)
2. (Koninklijke Philips Electronics N.V. 2013) [↑](#footnote-ref-2)
3. (Raspberry Pi Foundation 2014) [↑](#footnote-ref-3)
4. (Arduino 2014) [↑](#footnote-ref-4)
5. (Adafruit 2014) [↑](#footnote-ref-5)
6. (Seeed Technology Inc. 2013) [↑](#footnote-ref-6)
7. (Hackhappy 2013) [↑](#footnote-ref-7)
8. https://github.com/SBejga/hueper [↑](#footnote-ref-8)
9. (Koninklijke Philips Electronics N.V. 2013) [↑](#footnote-ref-9)
10. (Joyent, Inc. 2014) [↑](#footnote-ref-10)
11. (MongoDB, Inc. 2014) [↑](#footnote-ref-11)
12. (Morin 2013) [↑](#footnote-ref-12)
13. (npm, Inc. 2014) [↑](#footnote-ref-13)
14. (Kyoto University; Nagoya Institute of Technology 2014) [↑](#footnote-ref-14)
15. (Bagwell 2013) [↑](#footnote-ref-15)
16. (Lyon 2013) [↑](#footnote-ref-16)
17. (Git 2014) [↑](#footnote-ref-17)
18. (McLean, VoxForge - Free Speech Recognition 2014) [↑](#footnote-ref-18)
19. (Carnegie Mellon University 2014) [↑](#footnote-ref-19)
20. (Arduino 2014) [↑](#footnote-ref-20)
21. (Banzi 2011) [↑](#footnote-ref-21)
22. (Patin 2003) [↑](#footnote-ref-22)
23. (Arduino 2013) [↑](#footnote-ref-23)
24. (Firtman 2012) [↑](#footnote-ref-24)
25. (Mintert 2014) [↑](#footnote-ref-25)
26. http://themeroller.jquerymobile.com/ [↑](#footnote-ref-26)
27. (The jQuery Foundation 2014) [↑](#footnote-ref-27)
28. (Google Inc. 2014) [↑](#footnote-ref-28)
29. (Knight 2011) [↑](#footnote-ref-29)
30. (Bauer 2011) [↑](#footnote-ref-30)
31. (Hardkernel co., Ltd. 2014) [↑](#footnote-ref-31)
32. (SECO USA Inc 2014) [↑](#footnote-ref-32)
33. (Twitter, Inc. 2014) [↑](#footnote-ref-33)
34. (Angular-UI Team 2014) [↑](#footnote-ref-34)