

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 23. Mai 2014

Michael Strobel

Stuttgart, den 23. Mai 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](#_Toc388646905)

[Kurzbeschreibung 3](#_Toc388646906)

[Abbildungsverzeichnis 6](#_Toc388646907)

[Tabellenverzeichnis 7](#_Toc388646908)

[Abkürzungsverzeichnis 8](#_Toc388646909)

[1 Aufgabenstellung 10](#_Toc388646910)

[2 Hardware 11](#_Toc388646911)

[2.1 Auswahl der Komponenten 11](#_Toc388646912)

[2.1.1 Raspberry Pi und Peripherie 11](#_Toc388646913)

[2.1.2 Arduino und Peripherie 12](#_Toc388646914)

[2.2 Zusammenbau und Installation 13](#_Toc388646915)

[2.2.1 Philips Hue-Leuchten 13](#_Toc388646916)

[2.2.2 Aufsetzen des Raspberry Pi 14](#_Toc388646917)

[2.2.3 Zusammenbau des Arduino 16](#_Toc388646918)

[3 Entwicklung des Backends 17](#_Toc388646919)

[3.1 Funktionsumfang 17](#_Toc388646920)

[3.1.1 Funktionen der Hue-Bridge 17](#_Toc388646921)

[3.1.2 Favoriten und Szenen 18](#_Toc388646922)

[3.1.3 Automatisierung 18](#_Toc388646923)

[3.1.4 Party-Modus 19](#_Toc388646924)

[3.1.5 Weitere Funktionen des Backends 20](#_Toc388646925)

[3.1.6 Funktionen des Arduino 20](#_Toc388646926)

[3.2 Auswahl der Software 20](#_Toc388646927)

[3.2.1 Grundbausteine 20](#_Toc388646928)

[3.2.2 Module und Erweiterungen 21](#_Toc388646929)

[3.2.3 Zusätzliche Software 22](#_Toc388646930)

[3.3 Architektur 23](#_Toc388646931)

[3.4 Implementierung 24](#_Toc388646932)

[3.4.1 Grundstruktur 24](#_Toc388646933)

[3.4.2 Socket.IO-Verbindung 25](#_Toc388646934)

[3.4.3 Netzwerk-Geräte-Erkennung 27](#_Toc388646935)

[3.4.4 Sprachsteuerung 27](#_Toc388646936)

[3.4.5 Implementierung des Arduino 29](#_Toc388646937)

[3.5 Installation 30](#_Toc388646938)

[3.5.1 Einrichten der Entwicklungsumgebung 30](#_Toc388646939)

[3.5.2 Deployment auf dem Raspberry Pi 31](#_Toc388646940)

[4 Entwicklung des Frontends 32](#_Toc388646941)

[4.1 Auswahl der Software 32](#_Toc388646942)

[4.2 Layout und Design 32](#_Toc388646943)

[4.3 Implementierung 32](#_Toc388646944)

[5 Ausblick 34](#_Toc388646945)

[6 Fazit 35](#_Toc388646946)

[Quellenverzeichnis 36](#_Toc388646947)

[Anhang A: Architektur-Übersicht 39](#_Toc388646948)

[Anhang B: Hardware 40](#_Toc388646949)

[Anhang C: Frontend 44](#_Toc388646950)

# Abbildungsverzeichnis

[Abbildung 1: Raspberry Pi Soundkarten-Konfiguration 15](#_Toc388646951)

[Abbildung 2: Raspberry Pi WLAN-Konfiguration 15](#_Toc388646952)

[Abbildung 3: Backend-Code zum Einbinden der Controller 24](#_Toc388646953)

[Abbildung 4: Grundaufbau für Backend-Controller am Beispiel der Szenenverwaltung 25](#_Toc388646954)

[Abbildung 5: Event-Listener im Backend 25](#_Toc388646955)

[Abbildung 6: Dummy-Socket-Implementierung im REST-Controller 27](#_Toc388646956)

[Abbildung 7: Installation der Entwicklungsumgebung unter Linux 30](#_Toc388646957)

[Abbildung 8: Aufrufen des Installations-Scripts für den Raspberry Pi 31](#_Toc388646958)

# Tabellenverzeichnis

[Tabelle 1: Mögliche Optionen der Automatisierung 19](#_Toc388646959)

[Tabelle 2: Zentrale npm-Module des Backends 22](#_Toc388646960)

# Abkürzungsverzeichnis

**API** Application Programming Interface

**ARP** Address Resolution Protocol

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

**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

**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, die 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. 40). Eine Gesamt-Architekturübersicht zum Zusammenspiel von Hardware- und Software-Komponenten ist in Anhang A: Architektur-Übersicht (S. 39) 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 Busse 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 eine der Tasten 1-4 gewählt 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 „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 einfach 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
* Einmalige zeitgesteuerte Kommandos
* 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 implementiert.

### Favoriten und Szenen

Die Philips Hue-Leuchten verfügen über eine große Anzahl einstellbarer Farben, Helligkeiten und eine 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 eine Szene speichern. Wendet man sie an, nehme alle betroffenen Leuchte ihren jeweiligen 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; 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. 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) [[12]](#footnote-12). 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[[13]](#footnote-13) ist ein Programm zur kontinuierlichen Spracherkennung basierend auf einem Akustik-Modell, einem Vokabular sowie einer Grammatik
* SoX[[14]](#footnote-14) 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[[15]](#footnote-15) 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[[16]](#footnote-16) 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[[17]](#footnote-17) 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[[18]](#footnote-18) 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[[19]](#footnote-19). 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.

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 nach dem folgenden Prinzip:

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 durch einen festgelegten Wert geteilt, um Speicherplatz zu sparen, 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).

Um zu erkennen, ob die aktuelle 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. 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[[20]](#footnote-20):

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

sudo usermod -aG dialout username

Abbildung 7: 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 8: Aufrufen des Installations-Scripts für den Raspberry Pi

Die folgenden Schritte werden durch das Installations-Script durchgeführt:

* Aktualisieren der bereits installierten Programme auf die neuste verfügbare Version
* 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.

# Entwicklung des Frontends

Grob: Frontend soll Zugriff auf Backend-Funktionen von Smartphones und Tablets erlauben; cross-platform

…

Begründung Warum Angular? jQueryMobile?

## Auswahl der Software

Web: HTML5, CSS, JS; von Express zusammen mit dem Backend ausgeliefert

jQuery / jQuery Mobile

AngularJS

…

## Layout und Design

Bedienkonzept: Navigation, Overlays

Farbschema, flach, transparent

Icons

…

## Implementierung

Adapter: Kompatibilität AngularJS 1.3 und jQM 1.4 (generell: warum braucht man den)

sharedController

Direktiven: Slider, Colorpicker, Farben…

Socket- und stateManager-Services

Controller-Architektur

Parameter-Übergabe

Overlays

…

# 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

Was würden wir anders machen, was würden wir beibehalten?

* Leistungsfähigere Plattform als Raspberry Pi?
* Datenbank mit nativer ARM-Unterstützung + 32bit
* Bootstrap statt jQM? (kein Adapter, Routing möglich)

Was haben wir gelernt?

…

# 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.

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/.

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.

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.

Hackhapy. *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/.

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/.

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.

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/.

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/.

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/.

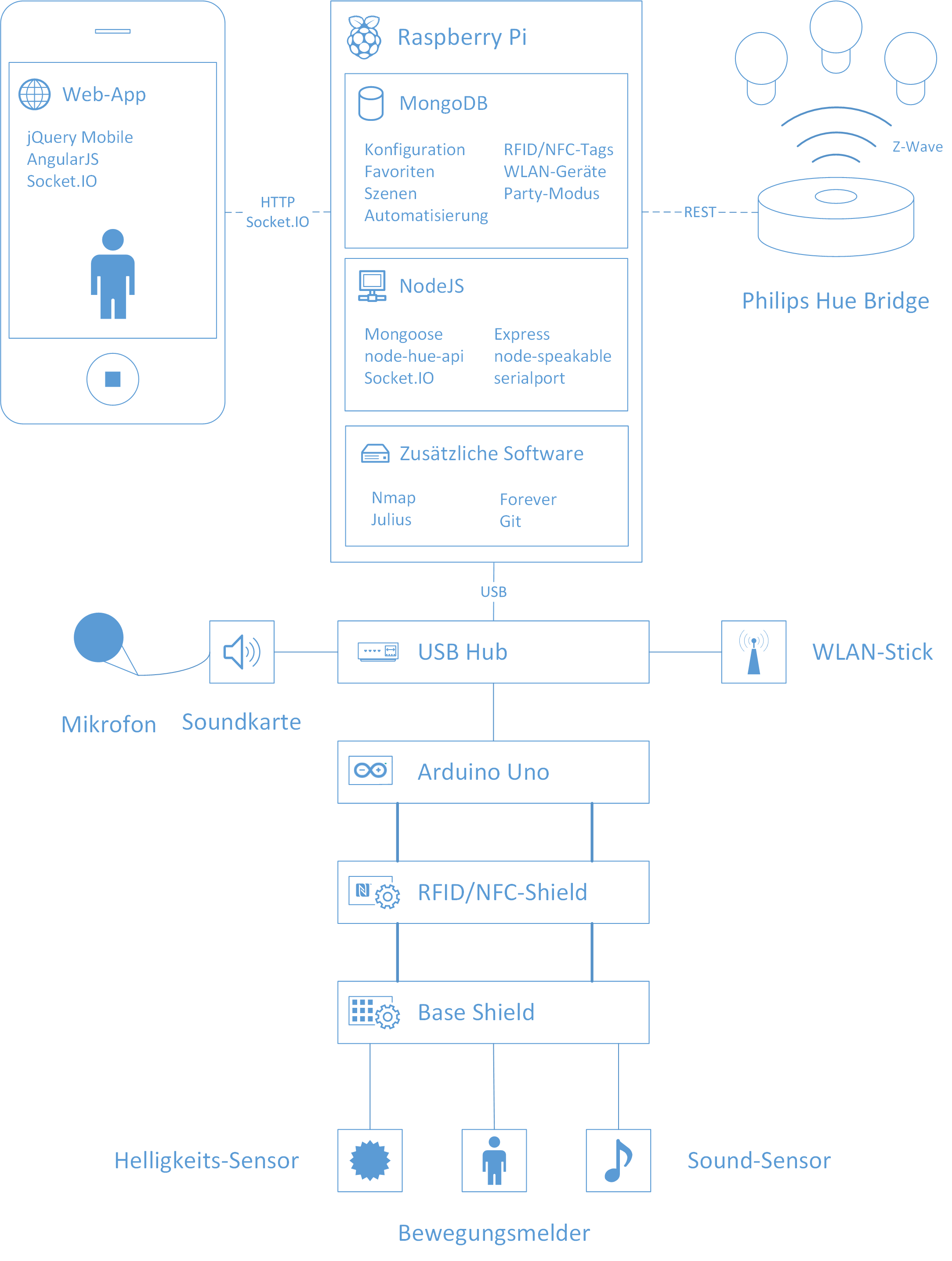
—. *jQuery Mobile.* 2014. http://jquerymobile.com/.

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

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>

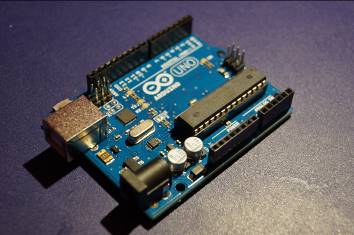
**Raspberry Pi** – 40€



Website: <http://raspberrypi.org>

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

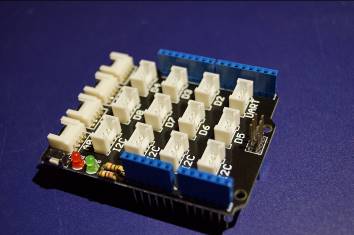
**Arduino Uno** – 24€



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

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

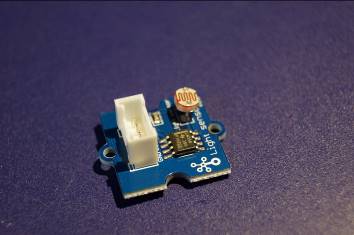
**Seeedstudio Grove Base Shield** – 9€



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>

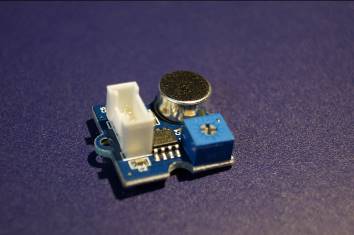
**Seeedstudio Grove Lichtsensor** – 3€



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>

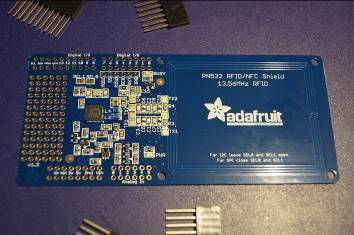
**Seeedstudio Grove PIR Motion Sensor** – 8€



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: 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. (Hackhapy 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. (npm, Inc. 2014) [↑](#footnote-ref-12)
13. (Kyoto University; Nagoya Institute of Technology 2014) [↑](#footnote-ref-13)
14. (Bagwell 2013) [↑](#footnote-ref-14)
15. (Lyon 2013) [↑](#footnote-ref-15)
16. (Git 2014) [↑](#footnote-ref-16)
17. (McLean, VoxForge - Free Speech Recognition 2014) [↑](#footnote-ref-17)
18. (Carnegie Mellon University 2014) [↑](#footnote-ref-18)
19. (Arduino 2014) [↑](#footnote-ref-19)
20. (Arduino 2013) [↑](#footnote-ref-20)