

Software Engineering Bachelor
Embedded Systems SS2016

Hausautomatisierung mit ZigBee

Projektdokumentation

Eugen Schlecht, Florens Hückstädt, Florian Wohlgemuth, Patrick Wohlgemuth

Inhaltsverzeichnis

Inhaltsverzeichnis

Abbildungsverzeichnis	III
Abkürzungsverzeichnis	IV
1 Einführung	1
2 Ausgangssituation	1
2.1 Konzepte	2
2.2 Eingesetzte Hardware	3
2.3 Sinnvolle Ergänzungen	3
3 Zieldefinition	4
3.1 Erweiterbarkeit	5
3.2 Benutzerfreundlichkeit	5
3.3 Sicherheit	5
4 Architektur und Umsetzung	5
4.1 Cloud Backend	6
4.1.1 API	6
4.1.2 Messaging	9
4.1.3 Build und Deployment	10
4.2 Raumserver	11
4.2.1 Hardware	12
4.2.2 Schnittstellen	13
4.2.3 Konfiguration	14
4.2.4 Nachrichtenprotokoll	14
4.2.5 Kommunikation über ZigBee	15
4.2.6 Inbetriebnahme	16
4.2.7 Regulierung der Raumtemperatur	18
4.3 Ventilsteuierung	19
4.3.1 Prototyp	19
4.3.2 Gehäuse	20
4.3.3 Software	21
4.4 Frontend	23
5 Integrated Sensors	24
5.1 Verwendete Sensoren	24
5.2 Temperatursensor	24
5.2.1 Sunfounder - Analog Temperatur Sensor Module	24
5.2.2 Seeedstudio - Temperatur Sensor Daten	25
5.2.3 Problemstellung	26

Inhaltsverzeichnis

5.3	Stromsensor	27
5.4	Im Projekt Hausautomatisierung	28
6	Fazit	28
A	Anhang	i
A.1	Beispielhafte Kommunikation über das gesamte System	i
A.2	Schaltplan	ii
A.3	Platine	iii
A.4	Screenshots vom Frontend	iv
A.4.1	Web Frontend: Ein neuer Raum wurde registriert	v
A.4.2	Web Frontend: Übersicht der Geräte eines Raums	vi
A.4.3	Web Frontend: Lichtschalter	vii
A.4.4	Web Frontend: Heizung	viii
A.4.5	Web Frontend: Ändern der Temperatur	ix
A.4.6	Konfigurationsseite des Raumservers	x
A.4.7	Aufspielen der Software auf den Atmega88	xii
A.5	API Dokumentation	xiii
A.5.1	API Aufruf: Registrieren eines neuen Benutzers	xiii
A.5.2	API Aufruf: Als registrierter Benutzer anmelden	xiv
A.5.3	API Aufruf: Einen Raum registrieren	xv
A.5.4	API Aufruf: Einen Raum von der Datenbank entfernen	xvi
A.5.5	API Aufruf: Einen spezifischen Raum abrufen	xvii
A.5.6	API Aufruf: Eine Liste aller Räume abrufen	xviii
A.5.7	API Aufruf: Ein neues Gerät registrieren	xx
A.5.8	API Aufruf: Ein spezifisches Gerät abrufen	xxi
A.5.9	API Aufruf: Ein Gerät aus der Datenbank entfernen	xxii
A.5.10	API Aufruf: Alle Geräte eines Raumes auflisten	xxiii
A.5.11	API Aufruf: Verändern der Gerätedaten	xxv

Abbildungsverzeichnis

Abbildungsverzeichnis

1	Projektaufbau der Vorgruppe aus dem Sommersemester 2015	1
2	Use-Case-Diagramm Heizungssteuerung	4
3	Sestemarchitektur	6
4	Klassendiagramm Hausautomatisierung	7
5	AMQP Konfigurationsseite	10
6	Raspberry Pi mit Status-LED auf Steckbrett	12
7	Schnittstellen des Raumservers	14
8	ZigBee-Netzwerk mit Raumserver und Geräten	14
9	Raumserver XBee-Nachrichtenstruktur	15
10	Prototyp Ventilsteuerung	19
11	Gehäuseentwurf Querschnitt	20
12	Gehäuseentwurf gerendert	20
13	Gehäuse gefertigt und montiert	21
14	Kommunikationsschnittstellen	23
15	Sunfounder - Analog Temperatur Sensor Module	24
16	Seeedstudio - Temperatur Sensor Daten	25
17	Temperatursensor am Pi	26
18	Temperatursensor am Pi mit Kabel	26
19	Beispielhafte Kommunikation über das gesamte System hinweg.	i
20	Schaltplan	ii
21	Platine	iii
22	Web Frontend: Übersicht der Räume	iv
23	Web Frontend: Ein neuer Raum wurde registriert	v
24	Web Frontend: Übersicht der Geräte eines Raums	vi
25	Web Frontend: Lichtschalter	vii
26	Web Frontend: Heizung	viii
27	Web Frontend: Ändern der Temperatur	ix
28	Konfigurationsseite des Raumservers	x
29	Eingabemaske für den Cloud Account	xi
30	Aufspielen der Software auf den Atmega88	xii

Abkürzungsverzeichnis

AT Mode	Transparent Mode
API Mode	Application Programming Interface Mode
Rx-Schnittstell	Receiver-Schnittstelle
Tx-Schnittstell	Transmitter-Schnittstelle
GPIO-Pins	General-Purpose-Input/output-Pins
ISP-Schnittstelle	In-System-Programmierung-Schnittstelle
PID-Regler	Proportional–Integral–Derivative Controller
AMQP	Advanced Message Queuing Protocol

1 Einführung

1 Einführung

Im Zuge der Vorlesung Embedded Systems SS2016 des Studiengangs Software Engineering Bachelor wird ein Projekt im Bereich der Hausautomatisierung durchgeführt. Thema des Projekts ist die Konzeption und Erstellung einer Heizungssteuerung. Dabei wird an einem im Sommersemester 2015 durchgeführten Projekt angeknüpft.

Die Use-Cases des Hausautomatisierungssystems sind allgemein betrachtet, das Lesen und Setzen von Werten für Geräte des häuslichen Gebrauchs. Darunter fallen zum Beispiel Lichtschalter, Türöffner oder wie in unserem konkreten Fall eine Heizung. Die Schwierigkeit ein solches System zu konzipieren liegt darin mit vielen verschiedenartigen Endgeräten umgehen zu können und gleichzeitig eine intuitiv bedienbare Software zu liefern. Im Idealfall muss der Benutzer das Gerät, dass er in die Hausautomatisierungsumgebung integrieren möchte nur einschalten und das Gerät macht sich in der Umgebung selbstständig bekannt und kann unmittelbar angesteuert werden.

Ziel des Projekts ist die Konzipierung und Entwicklung eines solchen Systems und die Erstellung einer Heizungssteuerung, die sich wie beschrieben einfach in die Hausautomatisierungsumgebung integrieren lässt. Dabei werden teilweise bereits existierende Konzepte der Vorgruppen übernommen.

2 Ausgangssituation

Es handelt sich bei dem Projekt Hausautomatisierung mit ZigBee um ein Thema, dass über mehrere Semester von unterschiedlichen Studentengruppen weitergeführt wird. Bei der Übernahme im Sommersemester 2016 konnte an einem Stand vom Sommersemester 2015 angeknüpft werden. Im folgenden Kapitel wird dieser Stand analysiert.

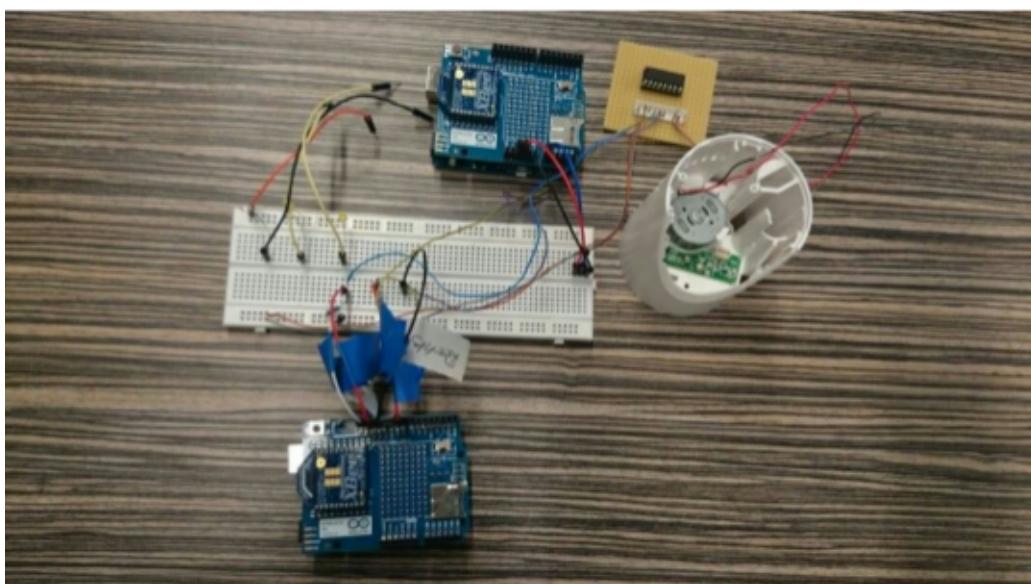


Abbildung 1: Projektaufbau der Vorgruppe aus dem Sommersemester 2015

2 Ausgangssituation

2.1 Konzepte

In diesem Abschnitt werden die Konzepte betrachtet und beurteilt, die bei der Vorgruppe erdacht und umgesetzt wurden. Positive sowie negative Aspekte werden dabei kurz erläutert.

REST API

Sämtliche Kommandos an die Endgeräte können über eine RESTful API getätigter werden. Der Einsatz einer API fördert die Erweiterbarkeit des Systems. So können beispielsweise unterschiedliche Frontends entwickelt werden ohne Geschäftslogik redundant zu programmieren.

ZigBee-Kommunikation im AT-Modus

Die Kommunikation im Netzwerk findet im Transparent Mode ([AT Mode](#)) statt. Dabei werden empfangene Daten von der Receiver-Schnittstelle ([Rx-Schnittstell](#)) direkt an die Transmitter-Schnittstelle ([Tx-Schnittstell](#)) weitergeleitet. Damit eine Kommunikation mit mehreren Teilnehmern möglich ist werden alle Nachrichten an die Broadcast-Adresse gesendet. Dadurch ist es notwendig, dass die Adressierung im Protokoll des Hausautomatisierungssystems abgehandelt werden muss. Darüberhinaus sind keine Empfangsbestätigungen im [AT Mode](#) implementiert.

Alternativ zum [AT Mode](#) kann der Application Programming Interface Mode ([API Mode](#)) verwendet werden. Dieser bringt folgende Vorteile¹

- Ändern von Parametern ohne den Command Mode zu aktivieren
- Anzeigen von RSSI und Absenderadresse
- Empfangen von Übertragungsbestätigungen

Generisches Protokoll zwischen Server und Endgeräten

Es wurde ein simples Protokoll erarbeitet, das es möglich macht, dass unterschiedliche Geräte gesteuert werden. Das Protokoll kommt dabei mit drei Befehlen aus:

- Statusanfrage
- Statusantwort
- Einen Wert setzen

Die Werte, die gesetzt werden können, haben dabei abhängig vom jeweiligen Gerät unterschiedliche Bedeutungen. So ist eine 1 bei einem Lichtschalter An. Bei einem Heizungsventil würde dies eine Öffnung um 1% bewirken.

¹http://knowledge.digi.com/articles/Knowledge_Base_Article/What-is-API-Application-Programming-Interface-Mode-and-how-does-it-work

2 Ausgangssituation

Das Protokoll ist jedoch in der Eigenschaft limitiert, dass sich Geräte nicht selbstständig beim Server registrieren können. Eine händische Eingabe des Benutzers ist erforderlich, der über die API den Typ des Geräts festlegt. Hier wäre eine Erweiterung des Protokolls sinnvoll, so dass die Geräte dem Server ihren Typ mitteilen können.

Prototypische Entwicklung mit Arduino

Sowohl Heizungssteuerung als auch Temperatursensor wurden jeweils auf einem Arduino Uno entwickelt. Aufgrund der verringerten Komplexität der Boards können rasch Ergebnisse erzielt werden. Dies ist bei der Evaluierung von Konzepten hilfreich, bevor sie auf dem Zielgerät entwickelt werden.

2.2 Eingesetzte Hardware

Eine genau Liste der eingesetzten Hardware kann der Dokumentation der Vorgruppe entnommen werden. Aufgrund des prototypischen Charakters des Projekts bis zu diesem Zeitpunkt war die Entwicklung auf Basis des Arduino Unos sinnvoll. Nach Beendigung der Arbeit der Gruppe in diesem Semester soll ein Stand erreicht sein, der dem eines einsetzbaren Produkts nahe kommt. Dies erfordert den Einsatz kompakter Hardware, die im Fall des Heizungsventils zum Beispiel in einem Gehäuse untergebracht werden kann. Ein Aufbau des Prototypensystems vom Sommersemester 2015 kann in [Abbildung 1](#) betrachtet werden.

2.3 Sinnvolle Ergänzungen

Neben der Beurteilung der Konzepte in [Unterabschnitt 2.1](#) werden in diesem Abschnitt sinnvolle Ergänzungen dazu erläutert. Zusammen mit den Gedanken aus [Unterabschnitt 2.1](#) sollen diese in die Weiterentwicklung einfließen.

Entwicklung eines Frontends

Die Steuerung der Geräte über direkte API-Aufrufe ist für Benutzer des Hausautomatisierungssystems nicht konfortabel. Die Entwicklung eines Frontends ist sinnvoll. Dabei sollte die Einsatzmöglichkeit auf verschiedenen Geräten bedacht werden.

Implementierung einer Authentifizierungs- und Autorisierungsmöglichkeit

Das System im aktuellen Stand ist nicht gegen unberechtigten Zugriff abgesichert. Eine Authentifizierungs- und Autorisierungsmöglichkeit ist für den Einsatz unter reellen Bedingungen zwingend erforderlich.

3 Zieldefinition

Zugriff auf Geräte über das Internet

Der Server in der jetzigen Ausführung ist nur über das LAN erreichbar. Für eine Erreichbarkeit über das Internet wäre es notwendig, dass der Benutzer über eine feste IP-Adresse verfügt. Besser wäre hier die Schaffung eines Cloud Backends, das mit dem Heimserver kommunizieren kann. Dies ermöglicht auch die Verwaltung mehrerer Gebäude über eine zentrale Plattform.

3 Zieldefinition

Gegenstand des Projekts ist die Weiterführung des über mehrere Semester bearbeitete Themas Hausautomatisierung mit ZigBee. Der Stand der Vorgruppe ist ein durchgeföhrter Proof-Of-Concept mit Entwicklung eines prototypischen Systems zur Steuerung eines Heizungsventils. Die Absicht der Weiterführung des Themas ist Verarbeitung der gewonnenen Erkenntnisse und die Entwicklung eines einsetzbaren Hausautomatisierungssystems. Der Aspekt der Einsetzbarkeit ist dabei so definiert, dass das System Gütekriterien aktueller Produkte im Bereich der Hausautomatisierung entspricht. Als exemplarisches Endgerät, das über das System angesteuert werden kann, bleibt weiterhin das Heizungsventil im Fokus. Der Anspruch an dieses ist nun die Kapselung der Elektronik in ein geschlossenes Gehäuse, so dass ein Anschluss an einen Heizkörper möglich ist. Ein Überblick des Systems informiert eines Use-Case-Diagramms ist in [Abbildung 2](#) dargestellt. Im Folgendem werden die Teilziele bzw. die Gütekriterien des geplanten Systems erläutert.

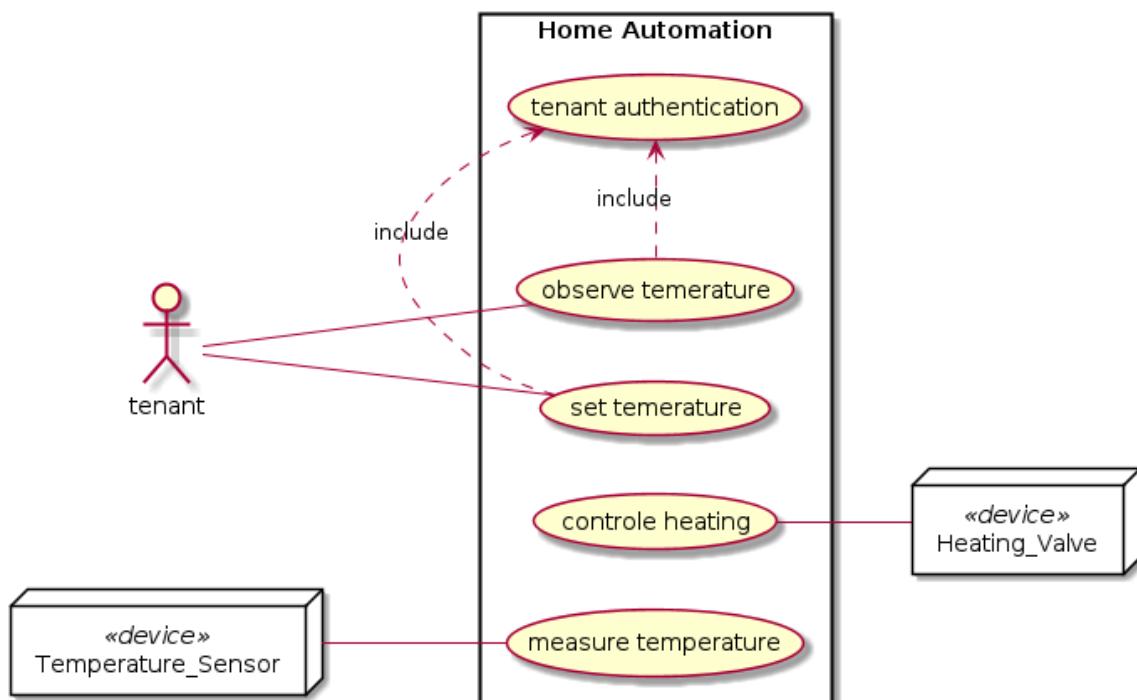


Abbildung 2: Use-Case-Diagramm Heizungssteuerung

4 Architektur und Umsetzung

3.1 Erweiterbarkeit

Die Erweiterbarkeit des Systems beinhaltet zum einen die Möglichkeit der Ansteuerung vielfältiger Endgeräte wie zum Beispiel Lichtschalter, Türöffner, Jalousien. Die Erweiterbarkeit in diesem Sinne erfordert generische Ansätze in den Bereichen der Kommunikation, Datenhaltung und Darstellung. Zum anderen wird Erweiterbarkeit verstanden als Möglichkeit das Softwaresystem ohne großen Aufwand um neue Anforderungen, wie zum Beispiel die Entwicklung von Benutzeroberflächen für verschiedene Endgeräte, zu ergänzen.

3.2 Benutzerfreundlichkeit

Die Bedienung, Installation und Erweiterung des Systems soll für den Benutzer möglichst einfach erlernbar sein. Die Benutzeroberflächen sollen dabei an Benutzer angepasst werden, die über keine internen Systemkenntnisse verfügen.

3.3 Sicherheit

Das System soll gegen unbefugten Zugriff von außen abgesichert sein. Komponenten der Hausautomation dürfen nicht von Unberechtigten gesteuert werden können.

4 Architektur und Umsetzung

Im Nachfolgenden werden die entwickelten und eingesetzten Lösungen zu den im [Abschnitt 3](#) beschriebenen hardware- und softwarespezifischen Anforderungen erläutert. [Abbildung 3](#) zeigt die grobe Systemarchitektur der entwickelten Lösung.

Das System besteht im wesentlichen aus vier Komponenten:

- **Endgerät:** Zum Beispiel Heizung, Schalter, Türöffner
- **Raumserver:** Steuert Endgerät über das ZigBee-Protokoll
- **Cloud Backend:** Bietet eine RESTful API zur Verwaltung der Räume und Geräte
- **Web Frontend:** Grafische Oberfläche zur Kommunikation mit dem Cloud Backend

Die einzelnen Bestandteile des Systems werden in den folgenden Unterkapiteln genau beschrieben.

4 Architektur und Umsetzung

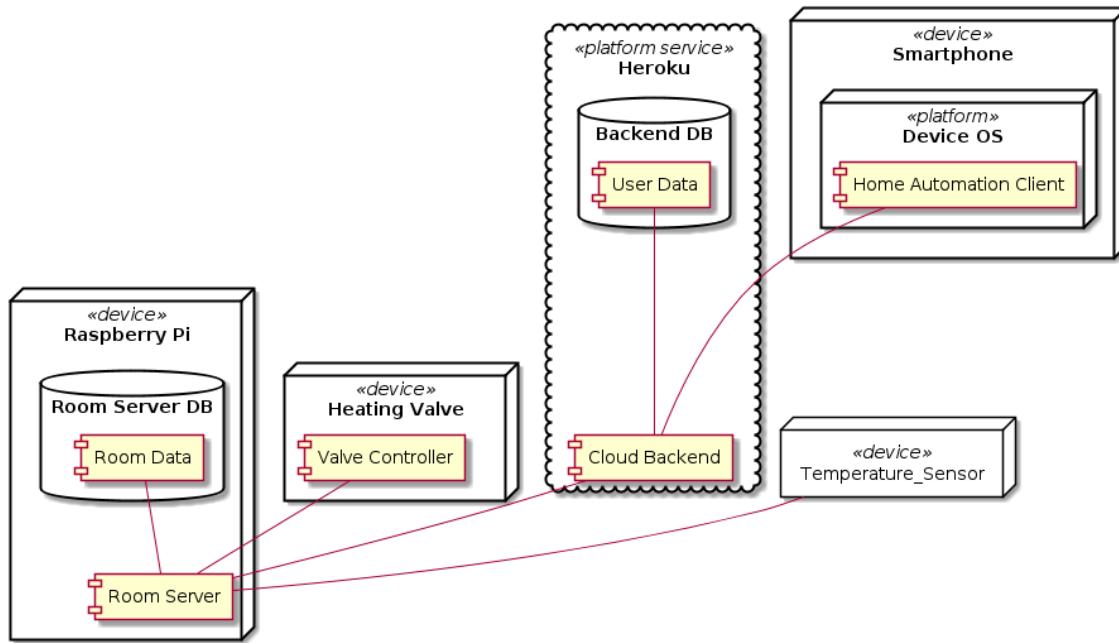


Abbildung 3: Systemarchitektur

4.1 Cloud Backend

Das Cloud Backend ist eine in Java geschriebene Anwendung, die über das Internet erreichbar ist und ein RESTful API bietet. Über das API können Räume, Geräte und Benutzerdaten verwaltet werden. Darunter fällt beispielsweise das Registrieren von Räumen und Geräten für einen bestimmten Benutzer. Darüberhinaus können Sollwerte der Geräte verändert werden. Die Veränderung wird dem tatsächlichen Gerät bzw. dem Room Server, der es verwaltet, über ein Messaging-Verfahren mitgeteilt (siehe dazu [Unterabschnitt 4.1.2](#)). Das API wird in [Unterabschnitt 4.1.1](#) beschrieben. Der Vorgang des Deployments wird in [Unterabschnitt 4.1.3](#) erläutert.

4.1.1 API

Das API ist als RESTful Web Service realisiert. Ressourcen können über HTTP mit den Methoden `GET`, `POST`, `PUT`, `PATCH` und `DELETE` angesprochen werden. Eine genaue Auflistung der API-Methoden ist im Anhang im [Unterabschnitt A.5](#) zu finden.

Schema

Aufgrund des Klassenmodells der Domäne Hausautomatisierung, welches in [Abbildung 4](#) dargestellt ist, ergeben sich API Endpoints mit folgender Struktur: `/user/rooms/{roomId}/devices/{deviceId}`.

Sämtlicher Zugriff auf das API erfolgt über HTTPS und kann über <https://enigmatic-waters-31128.herokuapp.com> erreicht werden.

4 Architektur und Umsetzung

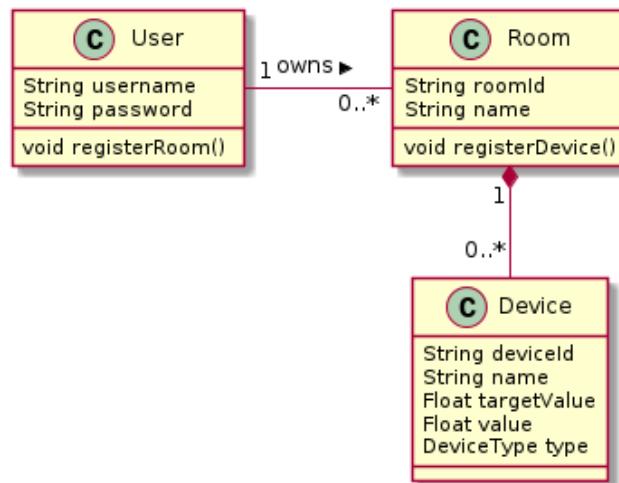


Abbildung 4: Klassendiagramm Hausautomatisierung

Eine beispielhafte Anfrage ist die Ausgabe des Raumes mit der Identifikation `d934eb20-4c6f-4d1c-91c5-61cdeddcf843`. Dies kann mit einem GET-Request auf folgende URL erreicht werden:

<https://enigmatic-waters-31128.herokuapp.com/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843>.

Das Cloud Backend liefert auf diese Anfrage beispielhaft folgende Antwort:

```

HTTP/1.1 200 OK
Connection: keep-alive
Server: Apache-Coyote/1.1
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, PUT, PATCH, GET, OPTIONS, DELETE
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: x-requested-with, content-type, X-AUTH-TOKEN
Access-Control-Expose-Headers: X-AUTH-TOKEN
X-Content-Type-Options: nosniff
X-Xss-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 01 May 2016 10:57:10 GMT
Via: 1.1 vegur

{
  "roomId": "d934eb20-4c6f-4d1c-91c5-61cdeddcf843",
  "name": "Wohnzimmer"
}
  
```

4 Architektur und Umsetzung

Sowohl beim Abfragen einer Liste als auch beim Abfragen einzelner Ressourcen beinhaltet die Antwort alle Attribute der Ressource.

Authentifizierung

Anfragen, die eine Authentifizierung erfordern, geben im Fehlerfall als Antwort `403 Forbidden` zurück. Zur Authentifizierung muss im Header das Feld `X-AUTH-TOKEN` gesetzt sein. Das entsprechende Token wird nach einer erfolgreichen Login-Anfrage erhalten.

Authentifizierung mit einem Token

```
$ curl 'https://enigmatic-waters-31128.herokuapp.com' -i -H 'X-AUTH-TOKEN: TOKEN'
```

Login

Zum Erhalt eines Tokens muss eine Loginanfrage gestellt werden. Diese enthält die Attribute `username` und `password`.

```
$ curl 'https://enigmatic-waters-31128.herokuapp.com/api/login' -i -X POST -H 'Content-Type: application/json' -d '{"username": "foo", "password": "bar"}'

HTTP/1.1 200 OK
Connection: keep-alive
Server: Apache-Coyote/1.1
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, PUT, PATCH, GET, OPTIONS, DELETE
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: x-requested-with, content-type, X-AUTH-TOKEN
Access-Control-Expose-Headers: X-AUTH-TOKEN
X-Content-Type-Options: nosniff
X-Xss-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-Auth-Token:
eyJpZCI6MTEsInVzZXJuYW1lIjoidXNlcjIwMTgsInJvbGVzIjpBI1VTRVIiXX0=.Ldd74G0yJAuQChYR9P0AOmThfy1OGG1Y19u2DcTcXyQ=
Content-Length: 0
Date: Sun, 01 May 2016 12:24:32 GMT
Via: 1.1 vegur
```

4 Architektur und Umsetzung

Das Token aus dem Response-Header-Feld `X-AUTH-TOKEN` muss bei zukünftigen Anfragen enthalten sein.

Fehlgeschlagener Login

Loginanfragen mit ungültigen Benutzerdaten werden mit `401 Unauthorized` beantwortet.

```
$ curl 'https://enigmatic-waters-31128.herokuapp.com/api/login' -i -X POST -H 'Content-Type: application/json' -d '{"username": "foo", "password": "bar"}'

HTTP/1.1 401 Unauthorized
Connection: keep-alive
Server: Apache-Coyote/1.1
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: POST, PUT, PATCH, GET, OPTIONS, DELETE
Access-Control-Max-Age: 3600
Access-Control-Allow-Headers: x-requested-with, content-type, X-AUTH-TOKEN
Access-Control-Expose-Headers: X-AUTH-TOKEN
X-Content-Type-Options: nosniff
X-Xss-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
Content-Type: application/json; charset=UTF-8
Transfer-Encoding: chunked
Date: Sun, 01 May 2016 12:30:03 GMT
Via: 1.1 vegur

{
  "timestamp":1462105803298,
  "status":401,
  "error":"Unauthorized",
  "message":"Authentication Failed: Bad credentials",
  "path":"/api/login"
}
```

4.1.2 Messaging

Beim Messaging kommt das Advanced Message Queuing Protocol ([AMQP](#)) in der Kommunikation von Cloud Backend zu Raumserver zum Einsatz. Dabei wird für jeden Raumserver eine eigene Queue eingerichtet. Siehe dazu [Abbildung 5](#).

4 Architektur und Umsetzung

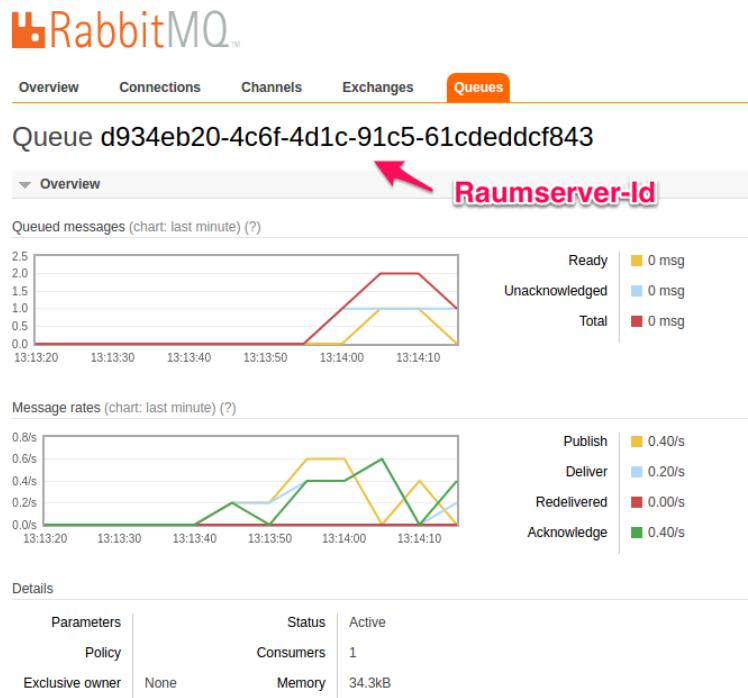


Abbildung 5: AMQP Konfigurationsseite

4.1.3 Build und Deployment

Zum Erstellen der Serverpaketierung kann Gradle verwendet werden. Da das Projekt den Gradle-Wrapper enthält muss Gradle nicht auf dem System installiert sein. Der Befehl zum Erstellen der Serverpaketierung lautet wie folgt:

```
gradlew build
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:findMainClass
:asciidoctor UP-TO-DATE
:jar
:bootRepackage
:assemble
:check
:build

BUILD SUCCESSFUL
```

4 Architektur und Umsetzung

Der Server kann dann als Java-Programm gestartet werden:

```
java -jar build/libs/backend-0.1.0.jar
```

Deployment auf Heroku

Zur Bereitstellung des Servers bei dem Cloud-Dienstleister Heroku ist ausschließlich ein git push auf das Heroku Repository auszuführen²:

```
git push heroku master
Initializing repository, done.
Counting objects: 110, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (87/87), done.
Writing objects: 100% (110/110), 212.71 KiB | 0 bytes/s, done.
Total 110 (delta 30), reused 0 (delta 0)

-----> Java app detected
-----> Installing OpenJDK 1.8... done
-----> Installing Maven 3.3.3... done
-----> Executing: mvn -B -DskipTests=true clean install
[INFO] Scanning for projects...
...
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 11.417s
[INFO] Finished at: Thu Sep 11 17:16:38 UTC 2014
[INFO] Final Memory: 21M/649M
[INFO] -----
-----> Discovering process types
Procfile declares types -> web
```

Die Anwendung kann anschließend unter folgender URL erreicht werden:

```
https://enigmatic-waters-31128.herokuapp.com
```

4.2 Raumserver

Der Raumserver ist eine Java-Anwendung zur Steuerung von Geräten über ZigBee. Seine Befehle nimmt er über eine Message-Queue entgegen (siehe [Unterabschnitt 4.1.2](#)). Diese kann vom Benutzer über die Hausautomatisierungs-API vom Cloud Backend angesprochen werden. Der Raumserver bietet darüberhinaus eine Webseite zur Konfiguration (siehe [Unterabschnitt 4.2.3](#)). Zur

²<https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku>

4 Architektur und Umsetzung

Kommunikation mit ZigBee-fähigen Geräten wird ein generisches Protokoll verwendet ([Unterunterabschnitt 4.2.4](#)). Der Raspberry Pi dient als Plattform für die Software. Am Raspberry Pi ist ein Temperatursensor verbaut, der vom Raumserver verwendet wird um die Raumtemperatur zu regulieren.

4.2.1 Hardware

In folgendem Abschnitt wird auf die hardwarespezifischen Besonderheiten des Raumservers eingegangen. Dabei wird in [Abschnitt 4.2.1](#) auf die Plattform eingegangen auf der die Java-Anwendung ausgeführt wird. [Abschnitt 4.2.1](#) beschreibt den Temperatursensor, der auf dem Pi verbaut ist. In ?? wird auf die Hardware eingegangen, die verwendet wird um über das ZigBee-Protokoll mit den Endgeräten zu kommunizieren. Abschließend wird in [Abschnitt 4.2.1](#) die Bedeutung der LED-Anzeigen erläutert.

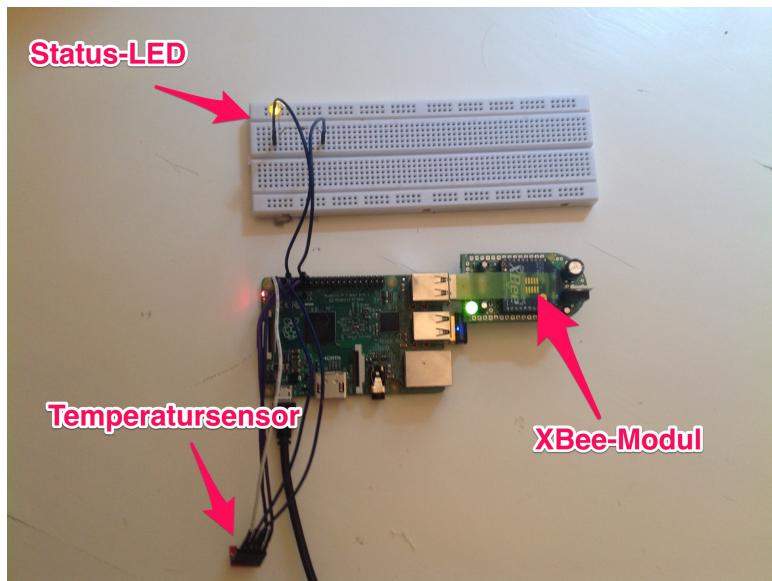


Abbildung 6: Raspberry Pi mit Status-LED auf Steckbrett

Raspberry Pi

Es wird das Raspberry Pi 2 Model B verwendet. Über die General-Purpose-Input/output-Pins ([GPIO-Pins](#)) ist weitere Hardware verbunden. Diese ist:

- **Temperatursensor LM75**
- **XBee Series 2 über USB-Adapterplatine**
- **LED**

In den weiteren Abschnitten wird die angeschlossene Hardware und deren Verwendung beschrieben.

4 Architektur und Umsetzung

LED	Bedeutung
Aus	Der Raumserver ist nicht aktiv. Dies deutet auf ein Problem hin.
Dauerhaft an	Der Raumserver ist aktiv.
Blinken in langsamer Frequenz	Der Raumserver befindet sich in der Initialisierungsphase
Blinken in schneller Frequenz	Der Raumserver empfängt oder sendet Nachrichten

Tabelle 1: Signale der Status-LED

Temperatursensor

Siehe dazu [Abschnitt 5](#).

Status-LED

Die Status-LED signalisiert wichtige Ereignisse des Raumservers. Die Signale sind in [Tabelle 1](#) beschrieben.

4.2.2 Schnittstellen

Der Raumserver bietet mehrere Schnittstellen. Eine Übersicht ist in [Abbildung 7](#) dargestellt. Dabei wird ein Heizungsventil beispielhaft als Endgerät angegeben. Der Raumserver kommuniziert mit dem Cloud Backend über die bereitgestellte REST API (siehe [Unterabschnitt 4.1.1](#)) mittels HTTPS. Über diesen Kanal werden vorwiegend Registrierungen von Endgeräten und Werteänderungen übertragen. Damit der Raumserver mit dem Cloud Backend kommunizieren kann muss er sich bei diesem authentifizieren. Dazu muss der jeweilige Cloud Account des Benutzers über die Konfigurationsseite des Raumservers eingetragen werden (siehe [Unterabschnitt 4.2.3](#)).

Die entgegengesetzte Kommunikation findet über das Messaging-Verfahren AMQP³ statt. Die Adressierung der Message Queues geschieht über die eindeutige ID der Raumserver. Die Queues sind persistent. Das heißt für den Fall, dass der Raumserver nicht erreichbar ist, bleiben die Nachrichten bestehen. Der Nachrichteninhalt ist im JSON-Format. Siehe dazu folgendes Beispiel:

```
{  
  "deviceId": "d934eb20-4c6f-4d1c-91c5-61cdeddcf843",  
  "targetValue: "20.5"  
}
```

Nachrichten beschränken sich auf das Setzen von Werten für bestimmte Endgeräte. Das jeweilige Endgerät wird über die im JSON mitgelieferte deviceId identifiziert. Beim Raumserver eingehende

³<https://www.amqp.org/>

4 Architektur und Umsetzung

Nachrichten werden an die entsprechenden Endgeräte über das ZigBee-Protokoll weitergeleitet (siehe dazu [Unterunterabschnitt 4.2.5](#)).

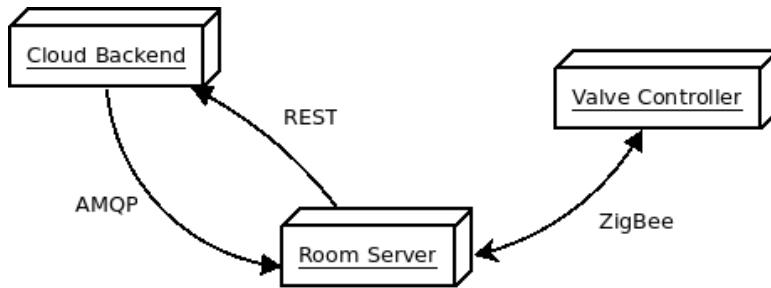


Abbildung 7: Schnittstellen des Raumservers

Innerhalb der ZigBee-Kommunikation nimmt der Raumserver die Rolle des Coordinator⁴ ein und die Geräte, wie zum Beispiel das Heizungsventil, die Rolle des End Device (siehe [Abbildung 8](#)).

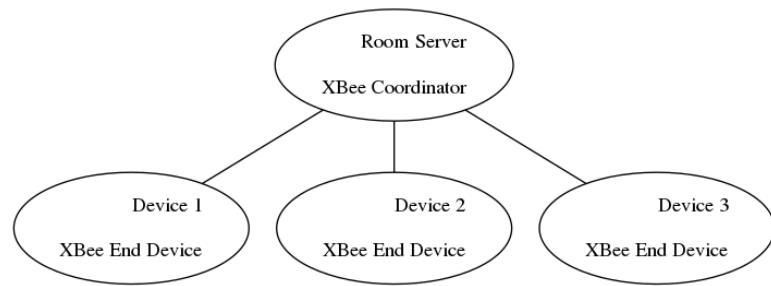


Abbildung 8: ZigBee-Netzwerk mit Raumserver und Geräten

4.2.3 Konfiguration

Die Konfigurationsseite (siehe [Abbildung 28](#)) des Servers kann über den Browser aufgerufen werden. Die Seite zeigt die eindeutige Identifikation des Servers an und bietet eine Maske zur Eingabe des Cloud Accounts. Die Eingabe des Accounts ist notwendig, damit der Raumserver einem Benutzer zugeordnet werden kann. Sobald der Account eingestellt ist registriert sich der Server selbst beim Backend (siehe [Abbildung 23](#)). Nun können Raumserver und Backend bidirektional kommunizieren.

4.2.4 Nachrichtenprotokoll

Das Protokoll zum Austauschen von Kommandos zwischen Raumserver und Endgeräten sieht Nachrichten mit fester Länge von drei Bytes vor (siehe [Abbildung 9](#)). Ein Byte für den jeweiligen Kommandotyp und 2 Bytes für Argumente. Die Argumente unterscheiden sich in Abhängigkeit vom Kommandotyp. Die verschiedenen Kommandotypen können der [Tabelle 2](#) entnommen werden.

⁴<https://en.wikipedia.org/wiki/ZigBee>

4 Architektur und Umsetzung

CMD	Bezeichnung	ARG1	ARG2
01	Wert setzen	Wert	
02	Statusanfrage		
03	Status	Gerätetyp	Wert

Tabelle 2: Kommandotypen

Gerätetypen	Bezeichnung
01	HEATING
02	SWITCH

Tabelle 3: Gerätetypen

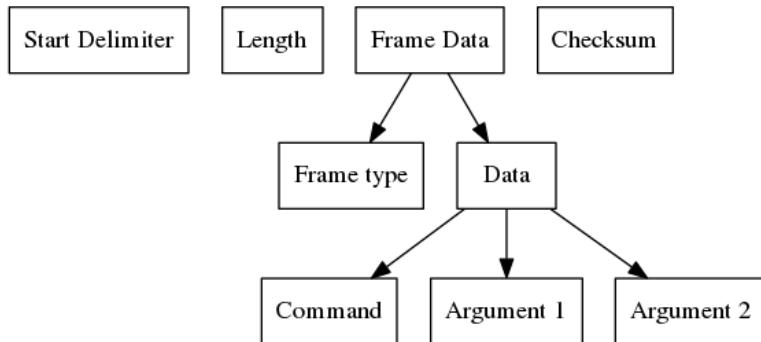


Abbildung 9: Raumserver XBee-Nachrichtenstruktur

Das Setzen eines Wertes 1 erfolgt beispielsweise über die Nachricht 01/01/00. Beim Rücksenden des Status wird sowohl der aktuelle Wert angegeben als auch der Gerätetyp. Die Schlüssel der verschiedenen Gerätetypen kann der Tabelle 3 entnommen werden.

In der Tabelle 4 ist ein beispielhafter Kommunikationsablauf dargestellt. Dabei sendet der Raumserver einem Endgerät eine Statusabfrage. Nach Erhalt der Statusantwort ist der Typ bekannt. Es handelt sich um eine Heizung. Der Raumserver reguliert die Raumtemperatur und sendet dazu der Heizung den notwendigen Öffnungsgrad des Ventils.

4.2.5 Kommunikation über ZigBee

Im Gegensatz zur Vorgruppe findet die XBee-Kommunikation aktuell im **API Mode** statt. Die Vorteile wurden in Abschnitt 2.1 erläutert. Eine offizielle Java Library wird von DIGI zur Verfügung gestellt⁵.

⁵<http://www.digi.com/blog/community/official-xbee-java-library/>

4 Architektur und Umsetzung

CMD	ARG1	ARG2
Statusanfrage		
02	00	00
Statusantwort: Gerät ist Heizung mit Wert 0 =>geschlossen		
03	01	00
Heizung 5% öffnen		
01	05	00
Statusanfrage		
02	00	00
Statusantwort: Gerät ist Heizung mit Wert 5 =>5% geöffnet		
03	01	05

Tabelle 4: Beispiel einer Kommunikation zwischen Raumserver und Heizung

4.2.6 Inbetriebnahme

In diesem Abschnitt werden die notwendigen Schritte zum Installieren, Starten und Stoppen der Software auf dem Raspberry Pi erklärt. Im [Abschnitt 4.2.6](#) wird beschreiben wie auf dem Raspberry Pi die Log-Dateien des Raumservers gelesen werden können.

Installation

Der Raumserver kann mit gradle gebaut werden: `gradle build`

Anschließend kann die jar-Datei auf den Raspberry Pi hochgeladen werden:

```
scp roomserver-0.1.0.jar pi@172.20.10.5:/home/pi
```

Die IP-Adresse des Raumservers kann über das Cloud Backend ermittelt werden:

```
https://enigmatic-waters-31128.herokuapp.com/dev/roomserver
```

```
{"ip": "172.20.10.5"}
```

Die IP-Adresse wird vom Raspberry Pi beim Start über ein Skript an das Cloud Backend übermittelt:

4 Architektur und Umsetzung

```
#!/bin/bash
# send pi info script

echo '{"ip":'`LANG=C ifconfig \
| grep "inet addr:" \
| grep -v "127\.[0-9]+\.[0-9]+\.[0-9]+"\+' \
| cut -d":" -f2 \
| cut -d" " -f1 \
| head -n1'"'}`' \
| curl 'https://enigmatic-waters-31128.herokuapp.com/dev/roomserver' -i -X PUT -H 'Content-Type: application/json' -d @- > /dev/null
```

Starten und Stoppen

Grundsätzlich startet die Anwendung automatisch sobald der Raspberry Pi eingeschalten wurde. Dies wird mit einem entsprechenden Eintrag in crontab realisiert. Ein weiterer Cronjob prüft jede Minute ob der Raumserver noch läuft und startet gegebenenfalls erneut.

Der Server kann zusätzlich manuell gestartet werden: Per SSH mit dem Server verbinden: `ssh pi@raspberrypi`

Starten:

```
sudo /home/pi/roomserver.sh start
```

Stoppen:

```
sudo /home/pi/roomserver.sh stop
```

Neustarten:

```
sudo /home/pi/roomserver.sh restart
```

Logging

Der Raumserver schreibt seine Logstatements in folgende Datei:

```
/srv/logs/roomserver.log
```

Zum mitverfolgen der Logdatei kann folgender Befehl benutzt werden:

```
tail -f /srv/logs/roomserver.log
```

Im Folgendem wird ein Beispiel einer Logausgabe dargestellt:

```
22:23:17.566 - Starting Application on raspberrypi with PID 2603 (/home/pi/roomserver-0.1.0.jar
started by root in /home/pi)
22:23:17.620 - The following profiles are active: prod
22:24:23.001 - [/dev/ttyS0 - 9600/8/N/1/N] Opening the connection interface...
```

4 Architektur und Umsetzung

```
22:24:23.100 - [/dev/ttyS80 - 9600/8/N/1/N] Connection interface open.  
22:24:51.570 - Started Application in 97.26 seconds (JVM running for 103.278)  
22:24:53.216 - Discovery process started  
22:24:53.425 - Device discovered: 0013A20040AFBEAD -  
22:24:59.293 - Discovery process finished successfully - 1 devices in network  
22:24:59.416 - Sending status request to 0013A20040AFBEAD  
22:24:59.418 - sending message to 0013A20040AFBEAD >> 2/0/0  
22:24:59.961 - [/dev/ttyS80 - 9600/8/N/1/N] Data received from 0013A20040AFBEAD >> 33 2F 31 2F 35.  
22:24:59.979 - Message is 3/1/5  
22:24:59.981 - cmd is: 3  
22:24:59.982 - arg1 is: 1  
22:24:59.983 - arg2 is: 5  
22:24:59.985 - Received status from 0013A20040AFBEAD: [HEATING 5]  
22:24:59.998 - 0013A20040AFBEAD is unknown, registering to backend  
23:19:51.558 - Received device update [id: d934eb20-4c6f-4d1c-91c5-61cdeddcf843_HEATING / target  
value: 30.0]  
23:19:51.576 - calibrating heating [target: 30.0 / current: 28.0]  
23:19:51.580 - sending message to 0013A20040AFBEAD >> 1/10/0  
  
23:19:51.706 - [/dev/ttyS80 - 9600/8/N/1/N] Data received from 0013A20040AFBEAD >> 33 2F 31 2F 31  
30.  
23:19:51.711 - Message is 3/1/10  
23:19:51.713 - cmd is: 3  
23:19:51.715 - arg1 is: 1  
23:19:51.717 - arg2 is: 10  
23:19:51.719 - Received status from 0013A20040AFBEAD: [HEATING 10]  
23:19:53.054 - sending temperature to backend
```

4.2.7 Regulierung der Raumtemperatur

Die Heizungssteuerung ist so entwickelt, dass die Werte, die sie empfängt als prozentualen Öffnungsgrad interpretiert. Das Prozentuale Öffnen der Heizung ist wichtig um ein konstantes Raumklima zu erreichen.

In professionellen Systemen kommt hier ein so genannter Proportional–Integral–Derivative Controller ([PID-Regler](#)) zum Einsatz. Die Implementierung eines solchen Reglers war für dieses Projekt geplant, konnte aber aus zeittechnischen Gründen nicht mehr umgesetzt werden. Für nachfolgende Gruppen wäre hier ein Ansatzpunkt Verbesserungen an dem System vorzunehmen. Eine Java-Library für den PID-Algorithmus ist auf Github⁶ zu finden.

⁶<https://github.com/ebisi/pid4j>

4 Architektur und Umsetzung

4.3 Ventilsteuering

Die Ventilsteuering besteht aus einem Mikrokontroller, der den Öffnungsgrad des Ventils durch die Ansteuerung des Ventilmotors einstellen kann. Der Öffnungsgrad des Ventils wird vom Raumserver über das ZigBee-Protokoll übermittelt. Zur ZigBee-Kommunikation verwendet die Ventilsteuering ebenso wie der Raumserver das XBee 2 Module. Die Ventilsteuering wurde zunächst als Prototyp auf einem Steckbrett entwickelt. Dies ist in [Unterunterabschnitt 4.3.1](#) genauer beschrieben. [Unterunterabschnitt 4.3.2](#) beschäftigt sich mit der Entwicklung eines Gehäuses für die Ventilsteuering. In [Unterunterabschnitt 4.3.3](#) wird beispielhaft der Algorithmus für das Initialisieren und Empfangen von Nachrichten inform eines Struktogramms gegeben.

4.3.1 Prototyp

Für den Prototypen wurde zunächst vollständig auf dem Stk600 programmiert. Zu einem späteren Zeitpunkt wurde die Zielhardware gewechselt. Dabei wurde der Atmega88 über die In-System-Programmierung-Schnittstelle ([ISP-Schnittstelle](#)) des Stk600s mit der Software bespielt (siehe [Abbildung 30](#)). Abbildung 10 zeigt den Prototyp auf dem Steckbrett. Dieser enthält dieselben Komponenten wie die spätere Platine.

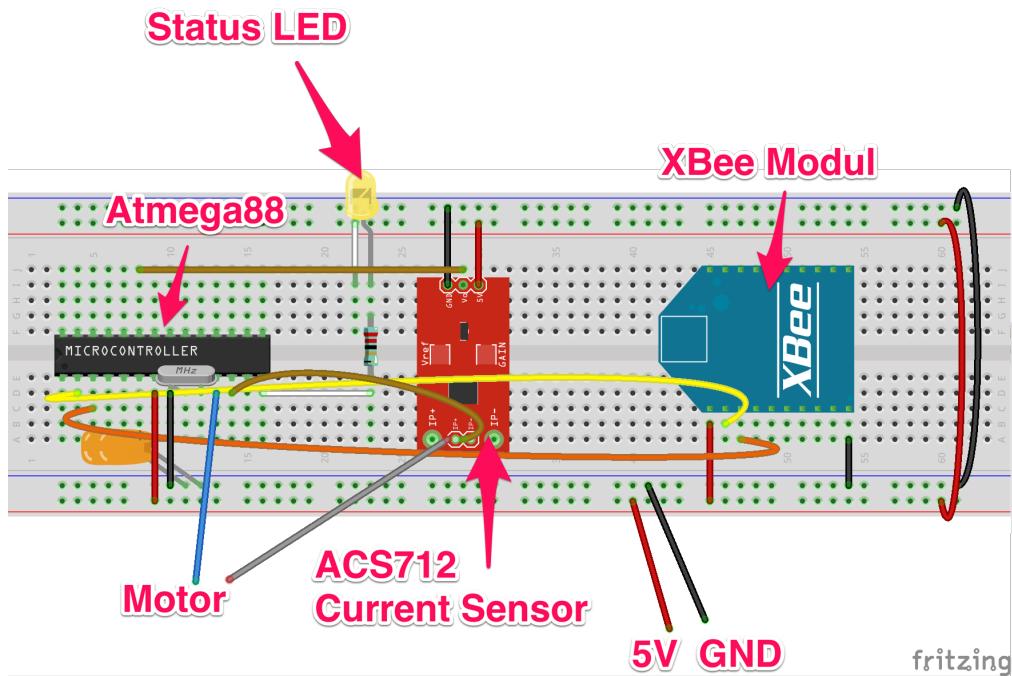


Abbildung 10: Prototyp Ventilsteuering

4 Architektur und Umsetzung

4.3.2 Gehäuse

Das Gehäuse wurde mit CAD entworfen und von Shapeways⁷ gefertigt. Bei dem Entwurf waren die final verwendeten Komponenten teilweise unbekannt. Daher musste der Platz großzügig dimensioniert werden, um so eine gewisse Freiheit für weitere Komponenten zu schaffen. Dies konnte erreicht werden indem das XBee-Modul unterhalb der Platine platziert wurde. Dafür musste dann in der horizontalen Ebene zusätzlicher Platz vorgesehen werden.

Sowohl Platine als auch XBee-Modul können in das Gehäuse gesteckt werden, sodass keine zusätzliche Befestigung notwendig ist.

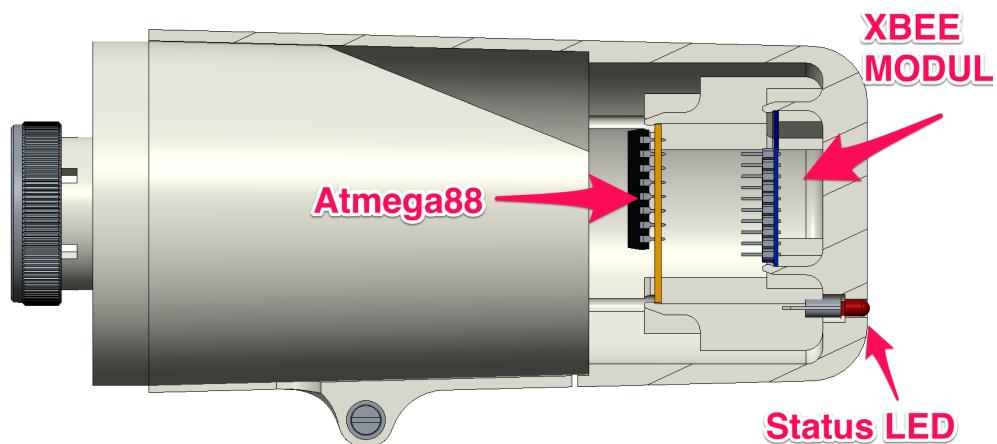


Abbildung 11: Gehäuseentwurf Querschnitt



Abbildung 12: Gehäuseentwurf gerendert

⁷<http://www.shapeways.com/>

4 Architektur und Umsetzung



Abbildung 13: Gehäuse gefertigt und montiert

4.3.3 Software

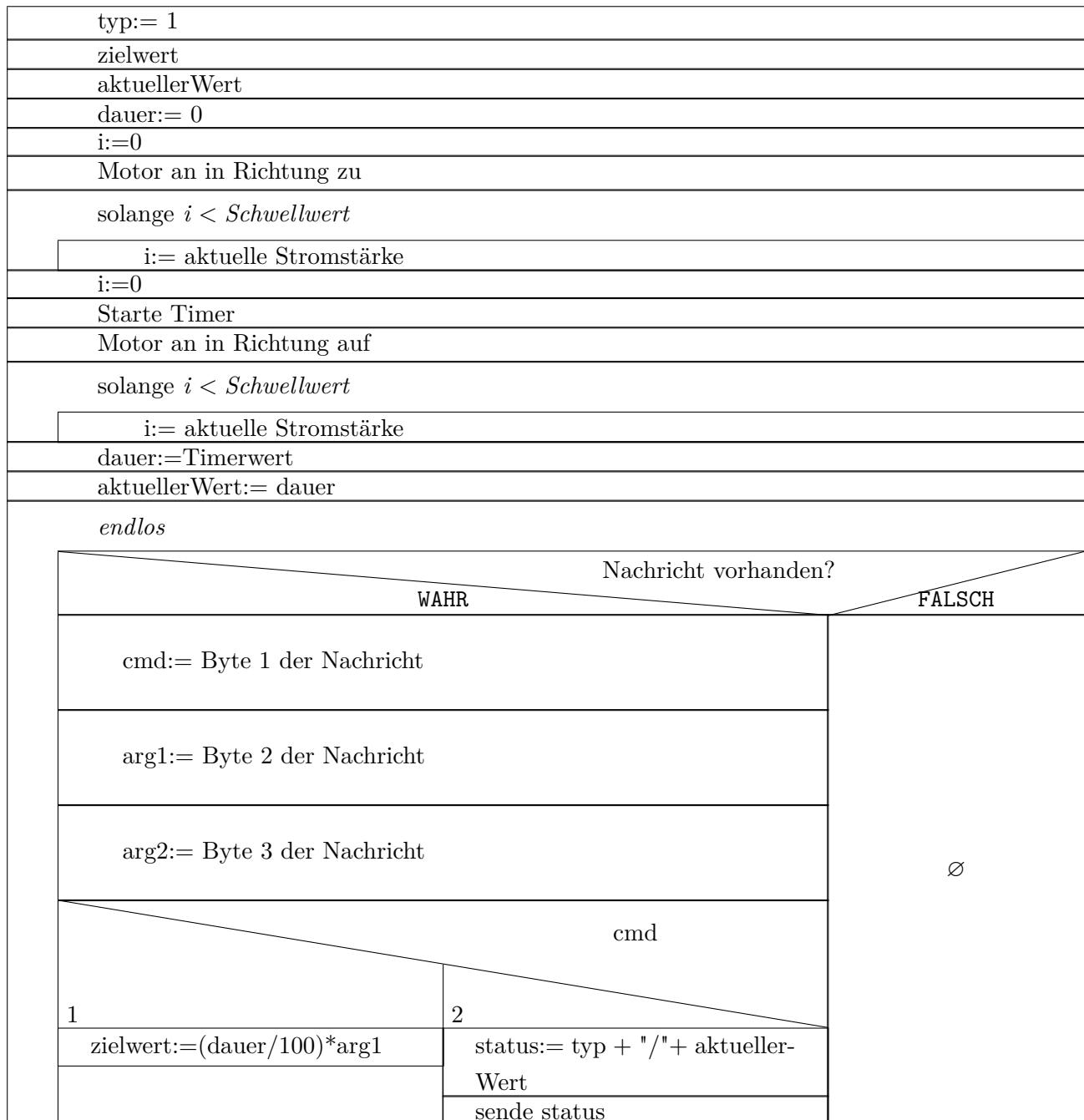
Die Steuerung des Ventils besteht im wesentlichen aus einer Initialisierungsphase, in der der Zeitbedarf für das Öffnen bzw. Schließen des Ventils gemessen wird, dem Empfangen von ZigBee-Nachrichten und dem Einstellen des gewünschten Öffnungsgrads des Ventils.

In der Initialisierungsphase wird sich zu Nutzen gemacht, dass die Stromstärke beim Blockieren des Motors ansteigt. Die aktuelle Stromstärke wird über den Stromsensor eingelesen. Sobald die Dauer des Öffnens bekannt ist, kann das Ventil prozentual eingestellt werden.

Im folgenden Struktogramm wird die Initialisierungsphase und das Setzen des neuen Zielwerts beschrieben. Der Einfachheit halber wurde das Öffnen bzw. Schließen aufgrund des Zielwerts nicht abgebildet.

Die eigentliche Programmcode ist in C geschrieben und der Dokumentation beigefügt.

4 Architektur und Umsetzung



4 Architektur und Umsetzung

4.4 Frontend

Das Frontend ist eine mit AngularJS entwickelte Webanwendung. Aufgrund des Responsive Designs passt sich die Anwendung dem jeweiligen Endgerät an. Dadurch ist es möglich die Hausautomatisierung über den Computer, Tablet und Handy zu steuern.

Aufgrund der Eigenschaft einer Single-Page-Application ist das Benutzererlebnis aufgrund der schnellen Reaktion besonders gut.

Durch die Verwendung von Websockets werden Werteänderungen direkt zum Browser gepusht, so dass kein GUI-Refresh erforderlich ist. Dies kommt bei folgenden Funktionen zum Einsatz:

- Registrierte Geräte und Räume erscheinen automatisch
- Die aktuelle Temperatur wird automatisch auf der Oberfläche angepasst
- Das Temperaturverlaufsdiagramm wird automatisch aktualisiert
- Veränderungen, die auf anderen Geräten vorgenommen werden, werden auf der eigenen Oberfläche automatisch angepasst

Eine Darstellung der Kommunikationsschnittstellen des Frontends ist in [Abbildung 14](#) zu sehen.

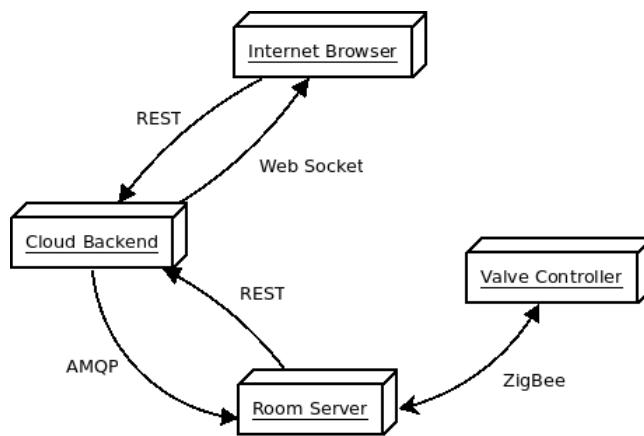


Abbildung 14: Kommunikationsschnittstellen

Bei der Gestaltung des Frontends wurde besonderen Wert auf Einfachheit gelegt. Neben der Darstellung von Räumen und Geräten verfügt die Anwendung noch über zwei Oberflächen zum Anmelden und Registrieren. Beim Login wird ein vom Backend erhaltenes Token im Local Storage abgelegt und bei weiteren Anfragen verwendet. Da die Screenshots ([Unterabschnitt A.4](#)) selbsterklärend sind wird auf eine genauere Erklärung der einzelnen Oberflächen verzichtet.

5 Integrated Sensors

5.1 Verwendete Sensoren

Im Projekt werden zwei Sensoren benötigt. Zum einen ein Temperatursensor und zum anderen ein Strommesser. Der Temperatursensor wurde an dem Raspberry-Pi angeschlossen, um die Temperatur in einem Raum messen zu können. Der Stromsensor wurde an den Mikrocontroller angeschlossen, um feststellen zu können wann der Motor am Ende seiner Umdrehungsmöglichkeit angekommen ist.

5.2 Temperatursensor

Um die Temperatur messen zu können, wird ein Temperatursensor benötigt. Wenn ein Temperatursensor an einen Mikrokontroller angeschlossen werden soll, wird ein analoger Temperatursensor benötigt, der die Temperatur beispielsweise in eine Spannung oder einen Strom umwandelt. Des Weiteren ist einen AD-Wandler nötig, der das Signal digitalisiert. Dieser kann auf dem Sensor oder dem Mikrocontroller integriert sein.

Temperatursensoren gibt es in verschiedenen Varianten. Vom temperaturabhängigen Widerstand bis zum fertig abgeglichenen All-in-one-Bauteil mit digitalem Ausgang.

Im Projekt wurden zwei verschiedene Sensoren getestet.

5.2.1 Sunfounder - Analog Temperatur Sensor Module

Beschreibung

Der Temperatursensor von Sunfounder weist eine hohe Genauigkeit der Messung durch eine große Reichweite auf, außerdem besitzt er eine gute Stabilität sowie eine starke Überladungskapazität. Der Sensor kann analoge und digitale Signale gleichzeitig ausgeben (siehe Abbildung 15).



Abbildung 15: Sunfounder - Analog Temperatur Sensor Module

5 Integrated Sensors

Technische Daten

- NTC Thermistor
- LM393
- Potentiometer
- PCB 2.3 x 2.3 cm
- Betriebsspannung: 3.3V-5V

5.2.2 Seeedstudio - Temperatur Sensor Daten

Beschreibung

Der Temperatursensor von Seeedstudio besitzt einen Delta-Sigma-Analog-Digital Converter. Durch den I2C Port kann der Sensor mit dem Raspberry Pi kommunizieren, dabei hat jedes angeschlossene Gerät eine eigene Adresse (siehe [Abbildung 16](#)).



Abbildung 16: Seeedstudio - Temperatur Sensor Daten

Technische Daten

- Temperaturbereich von -25°C 100°C
- PCB 1.3 x 1.0 cm
- Betriebsspannung: 3.3V-5V
- unterstützt Raspberry Pi und Arduino
- kompatibel mit Raspberry Pi A+,B,B+/2
- I2C Port

5 Integrated Sensors

- LM75 IC

5.2.3 Problemstellung

Wurde der Temperatursensor direkt auf dem Pi geschlossen, gab dieser zunächst falsche Temperaturwerte zurück (siehe [Abbildung 17](#)).



Abbildung 17: Temperatursensor am Pi

Diese entstanden jedoch nicht durch eine falsche Pin-Belegung oder durch ungenaues Messen des Sensors, sondern daran, dass sich die Temperatur durch die Hardware des Pi im Laufe des Betriebs erhöhte und eine hohe Wärme abgab. Diese erwärmte Luft im direkten Umfeld des Pi's sorgte dafür, dass zu dieser Zeit der Wärmewert der Luft so hoch war, dass dieser die Messwerte verfälschte.

Abhilfe wurde dadurch geschaffen, dass der Sensor in einem Bereich in dem die erwärmte Luft durch die Pi Hardware nicht mehr relevant war ausgelagert wurde ([Abbildung 18](#)).



Abbildung 18: Temperatursensor am Pi mit Kabel

5 Integrated Sensors

5.3 Stromsensor

Beschreibung

Der ACS712 ist ein Hall-Effekt Sensor. Das heißt; wird der Sensor von einem Strom durchflossen erzeugt es in seinem Magnetfeld eine Ausgangsspannung die proportional zum Produkt aus magnetischer Feldstärke und Strom ist (das nennt man den Hall-Effekt). Der ACS712 braucht eine eigene Stromverbindung, damit er den Betrieb aufnehmen kann. Dazu besitzt er einen VCC-Pin und einen GND-Pin, welche jeweils außen sind. Um den eingehenden Strom über die oberen Kontakte auslesen zu können, ist der Out-Pin in der Mitte zuständig. Der Wert der über den Out-Pin übermittelt wird ist analog, d.h. er gibt die Werte zwischen 0-1023 aus, da die Ausgangsspannung wie oben beschrieben proportional zum Produkt aus magnetischer Feldstärke und Strom ist, liegt der Wert im Bereich zwischen 518-522, da der Sensor temperaturabhängig ist und einen Offset haben kann.

Technische Daten

- Geräuscharmer analoger Signalweg
- Bandbreite wird über den FILTER-Pin eingestellt
- 5 μ s Reaktion zum Eingangsstrom • 80 kHz Bandbreite
- Abweichungsanfälligkeit 1.5% at TA = 25°C
- Geringer Platzbedarf, low-profile SOIC8 Gehäuse
- 1.2 m Ω interner Leistungswiderstand
- 2.1 kVRMS minimale Isolationsspannung an den Pins 1-4 auf die Pins 5-8
- 5.0 V Einzelbetrieb
- 66 bis 185 mV/A Ausgangsempfindlichkeit
- Ausgangsspannung proportional zum Fluss AC oder DC
- Stabile Ausgangsoffsetspannung
- Magnet Hysterese knapp bei Null

6 Fazit

5.4 Im Projekt Hausautomatisierung

Verwendung: Das 5a Range Current Sensor Module Acs712 Module wird in unserem Projekt dazu verwendet, um die Stromspannung die der Motor bekommt / abgibt zu messen. Wenn der Motor auf einen Widerstand trifft, steigt die Stromstärke. Diese wird im Atmega-88 ausgelesen und ausgewertet, übersteigt die Stromstärke einen festgelegten Schwellenwert, wird die Stromzufuhr für den Motor abgestellt. Damit die Stromstärke ausgehend vom Motor ausgelesen werden kann, ist der Sensor mit dem Motor in Reihe geschaltet.

6 Fazit

Durch die Bearbeitung des Themas Hausautomatisierung mit ZigBee der Studengruppe im Sommersemester 2016 konnte ein ein Projektstand erreicht werden, der dem Ziel eine einsetzbare Plattform zu schaffen ein großes Stück näher gekommen ist. Es wurde eine REST API geschaffen, die es ermöglicht Räume und Geräte zu verwalten. Auf dem Raspberry Pi wurde eine Software installiert, die bidirektional mit dem Backend und den Endgeräten kommunizieren kann. Die ZigBee-Kommunikation findet im [API Mode](#) statt. Dadurch können unterschiedliche Endgeräte direkt adressiert werden ohne im Broadcast-Modus zu arbeiten. Des Weiteren existiert ein optisch ansprechendes Frontend, dass diversen Anzeigegeräten automatisch anpasst. Für die Kommunikation innerhalb des gesamten Systems wurden moderne Technologien verwendet wie zum Beispiel [AMQP](#) oder Web Sockets. Das System ist über das Internet erreichbar und ist durch einen Authentifizierungs- und Autorisierungsmechanismus abgesichert. Für das Ventil wurde ein zweckmäßiges Gehäuse gefertigt, dass die Hardware in sich kapselt. Das Ventil kann prozentual geöffnet werden. Dies ermöglicht den Einsatz eines [PID-Regler](#) zum Einstellen eines konstanten Raumklimas.

Der Algorithmus für den [PID-Regler](#) kann von einer Nachfolgegruppe einfach in die Raumserversoftware integriert werden. Auf der Hausautomatisierungsplattform können ebenfalls neue Hardwareprojekte aufbauen. So ist es möglich, dass zusätzliche Endgeräte entwickelt werden, die das hier konzipierte Protokoll unterstützen.

A Anhang

A Anhang

A.1 Beispielhafte Kommunikation über das gesamte System

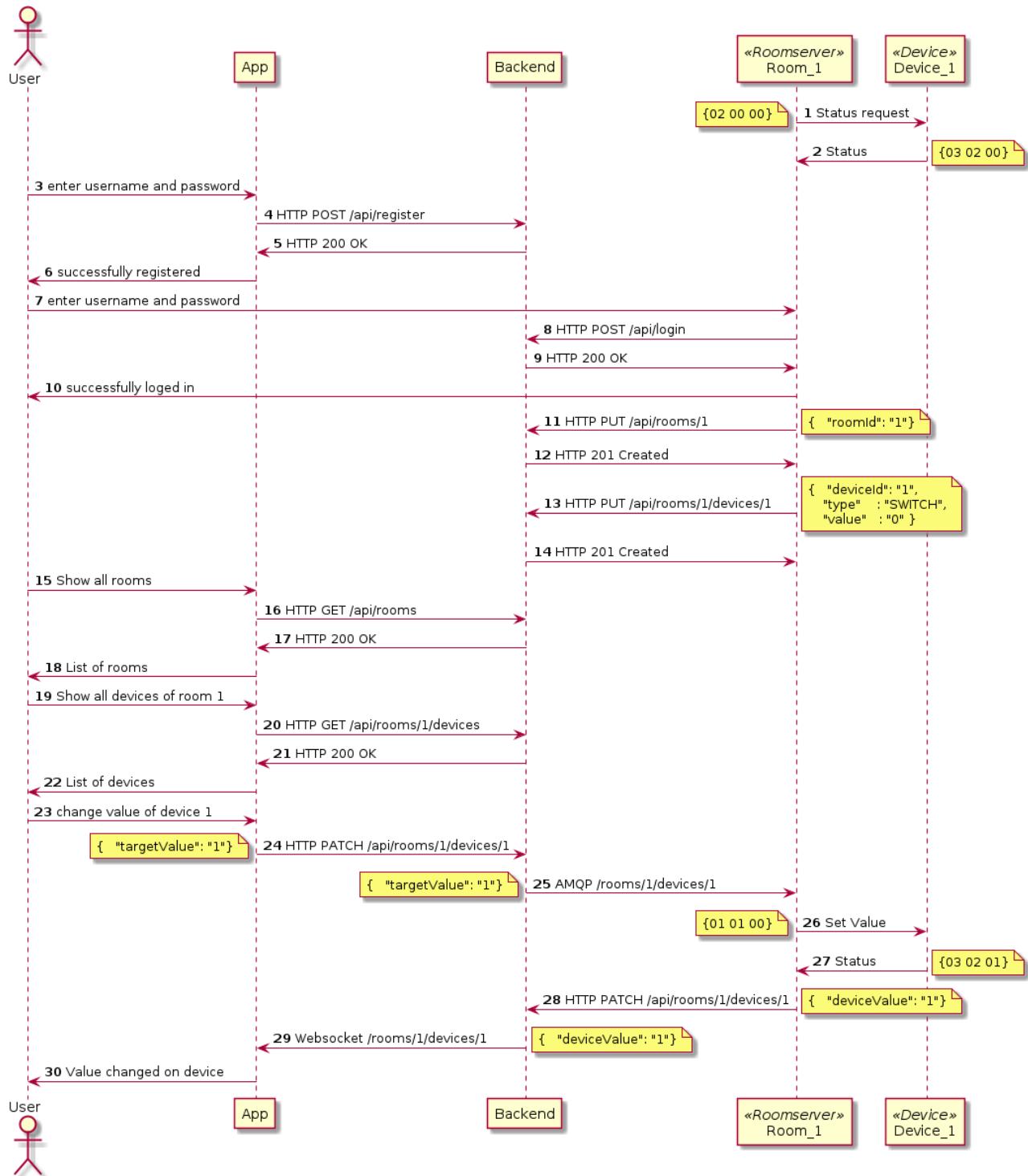


Abbildung 19: Beispielhafte Kommunikation über das gesamte System hinweg.

A Anhang

A.2 Schaltplan

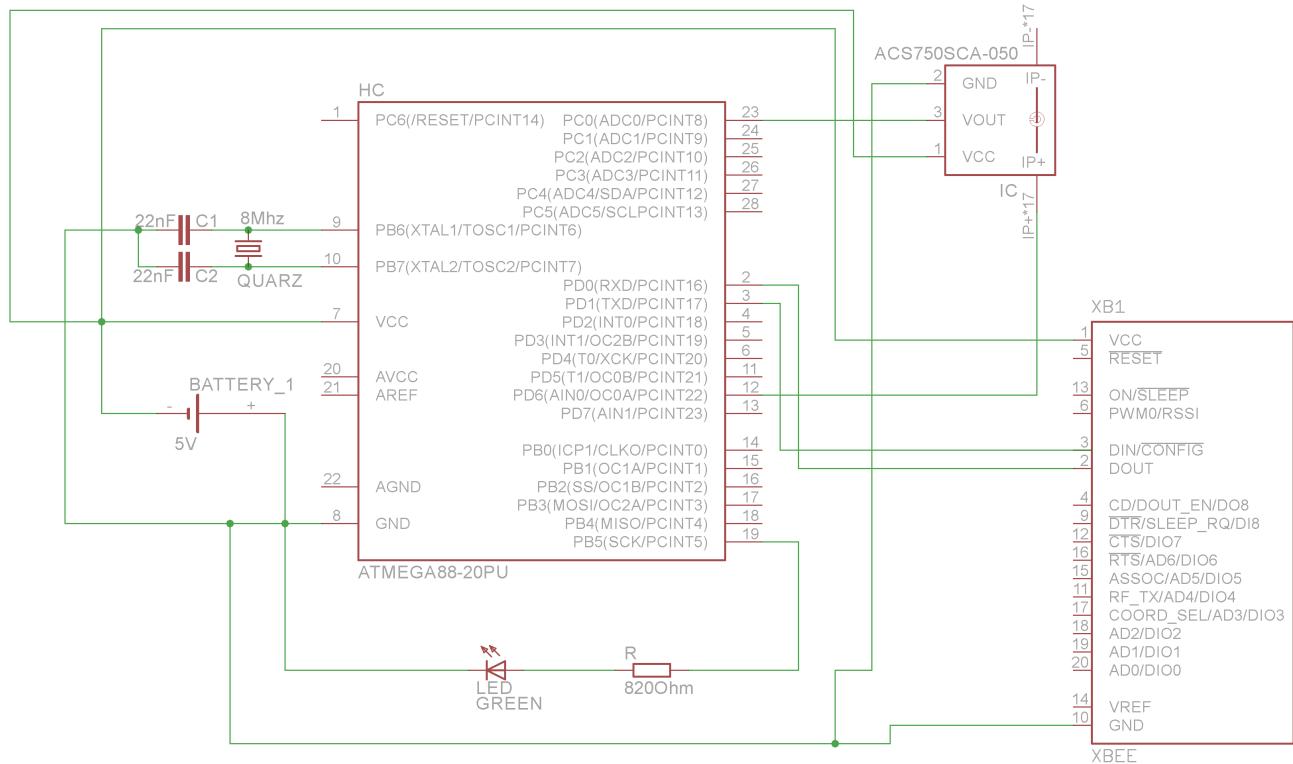


Abbildung 20: Schaltplan

A Anhang

A.3 Platine

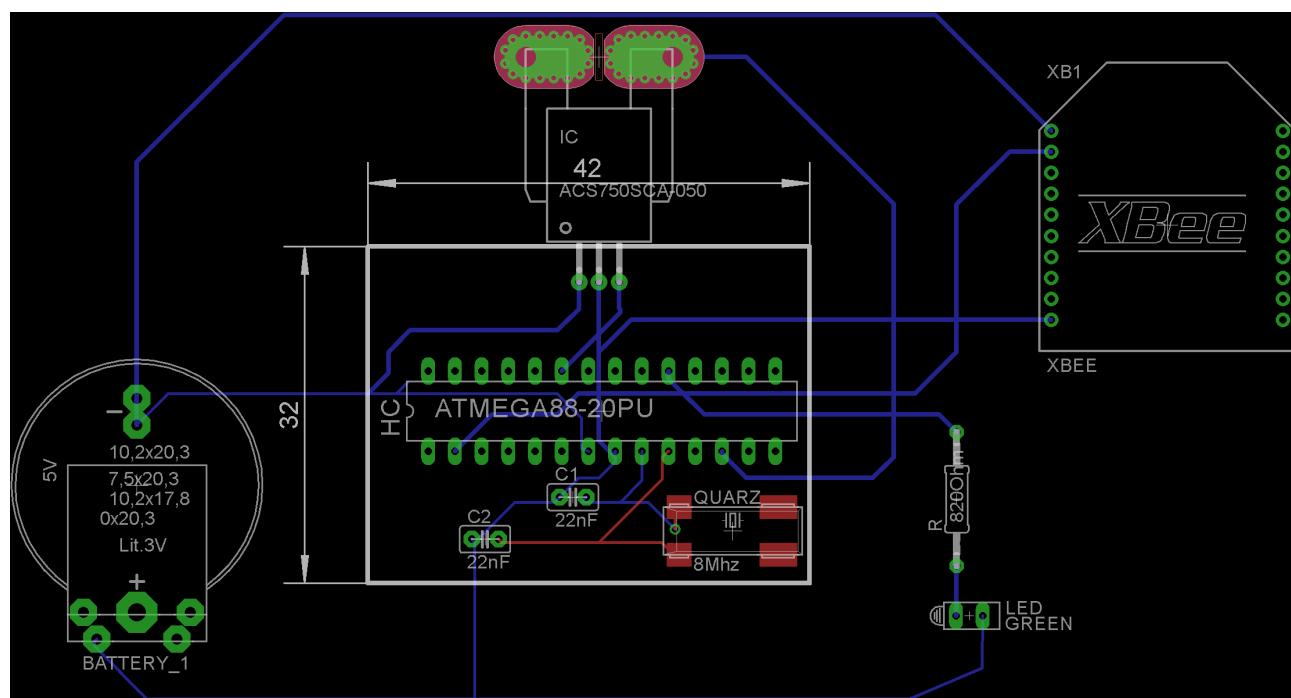


Abbildung 21: Platine

A Anhang

A.4 Screenshots vom Frontend

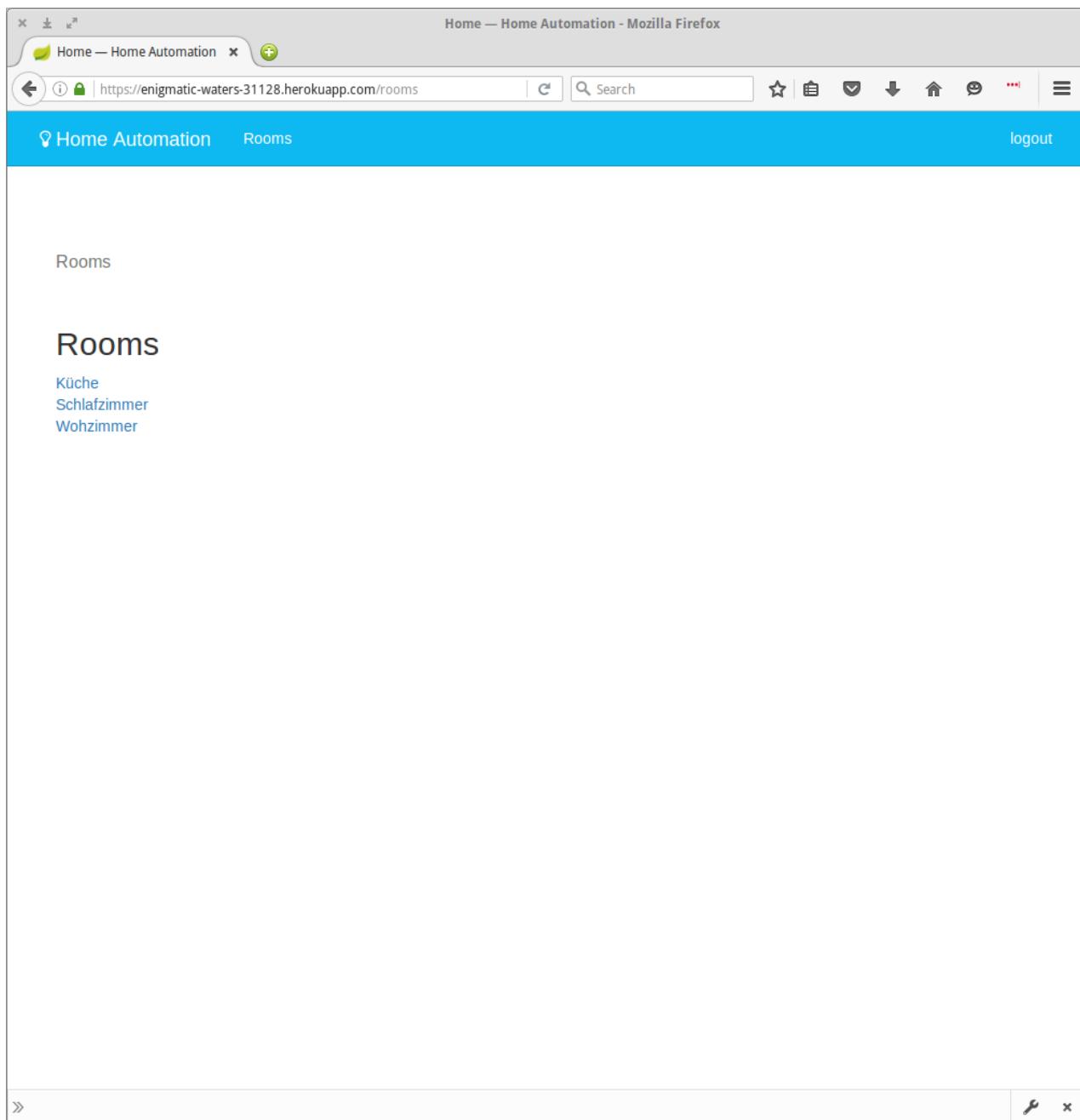


Abbildung 22: Web Frontend: Übersicht der Räume

A Anhang

A.4.1 Web Frontend: Ein neuer Raum wurde registriert

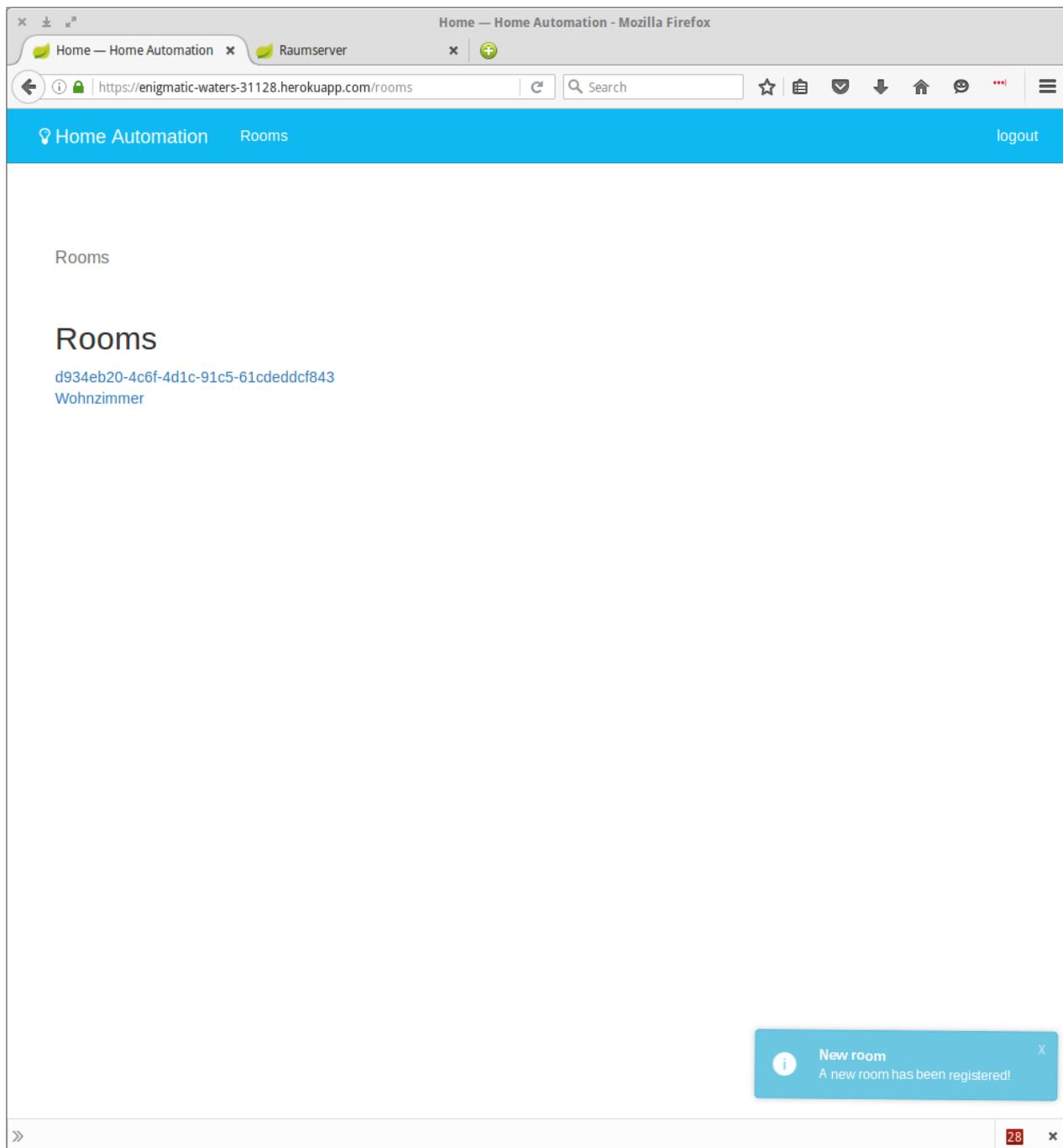


Abbildung 23: Web Frontend: Ein neuer Raum wurde registriert

A Anhang

A.4.2 Web Frontend: Übersicht der Geräte eines Raums

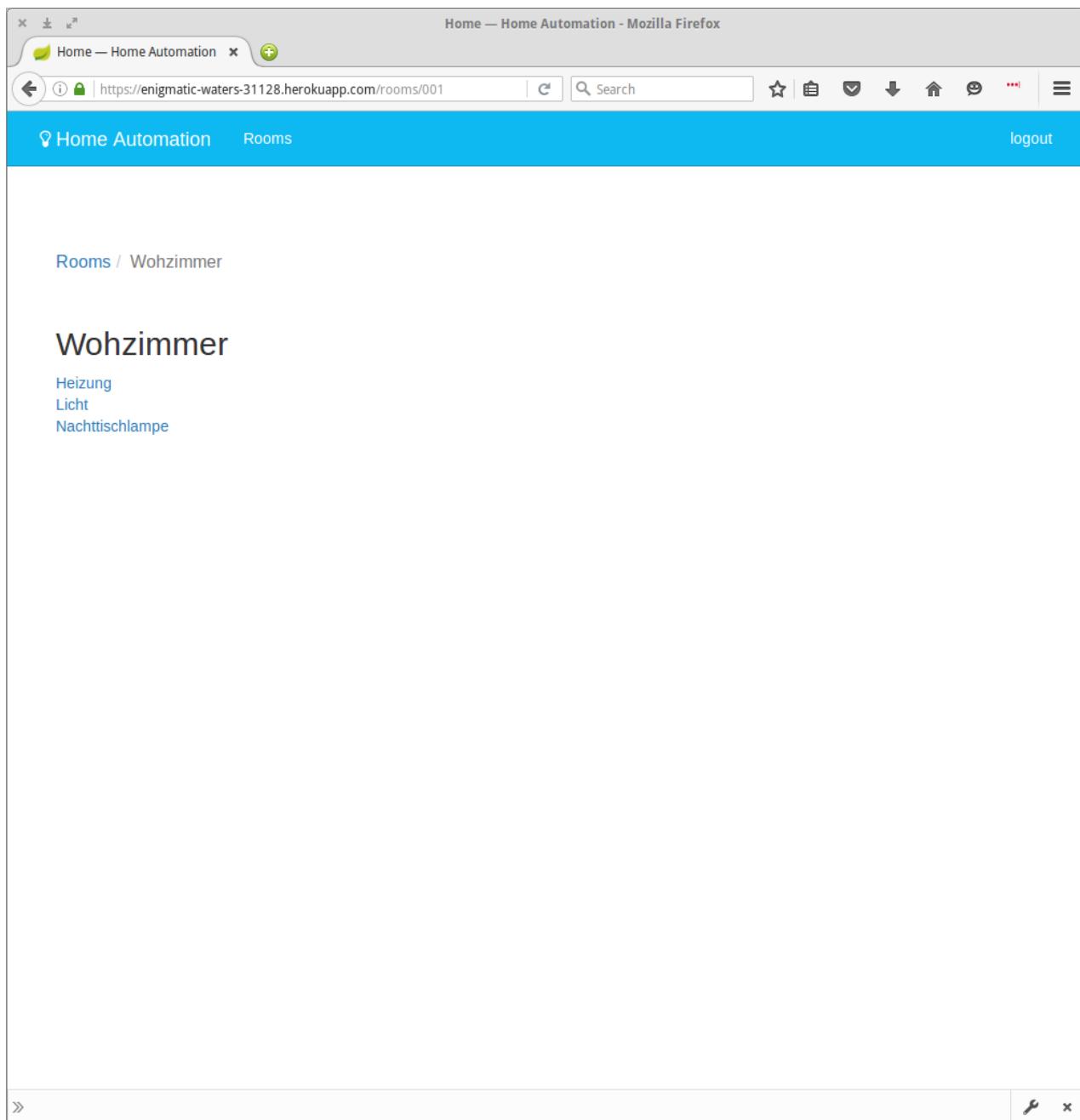


Abbildung 24: Web Frontend: Übersicht der Geräte eines Raums

A Anhang

A.4.3 Web Frontend: Lichtschalter

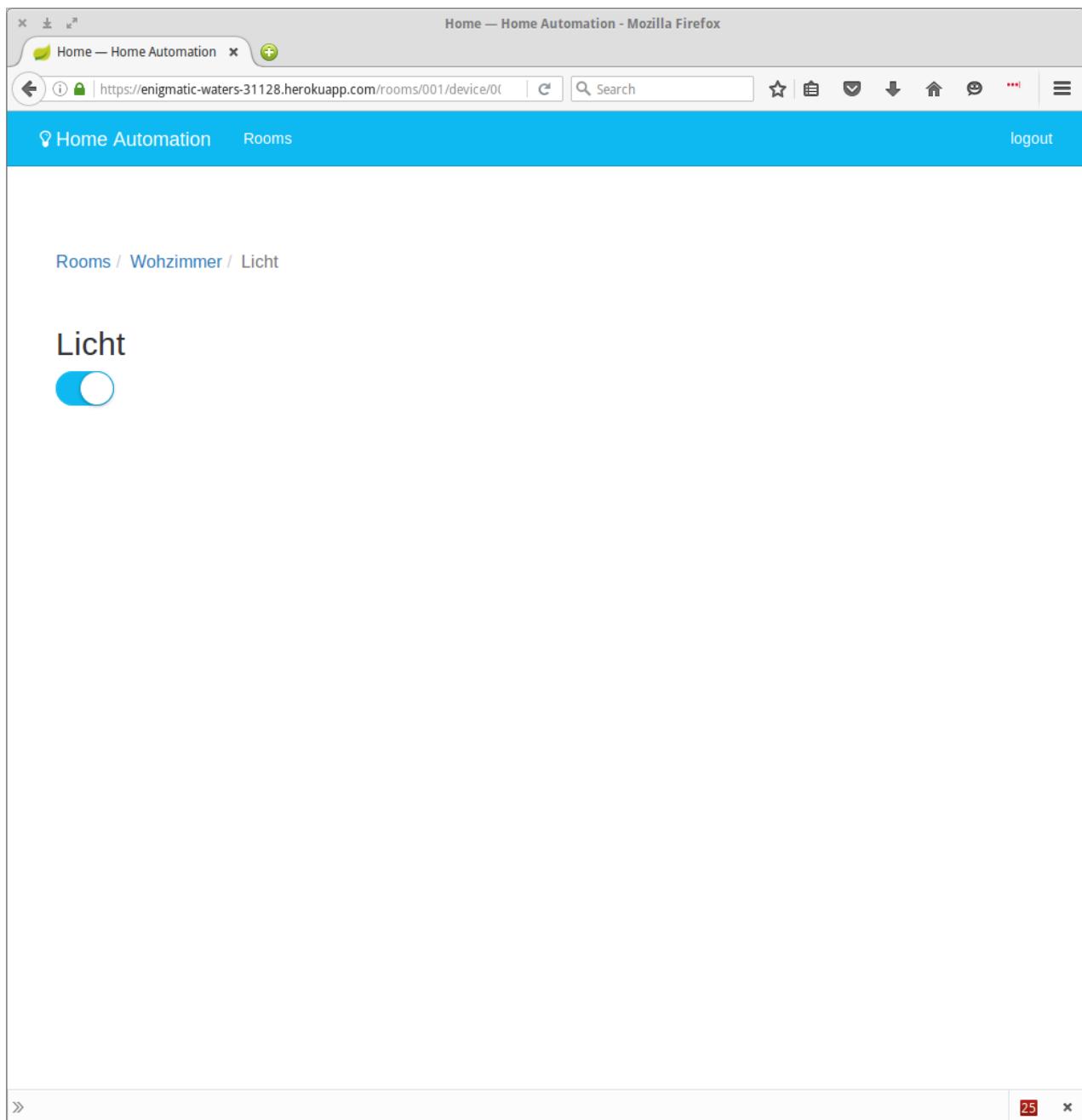


Abbildung 25: Web Frontend: Lichtschalter

A Anhang

A.4.4 Web Frontend: Heizung

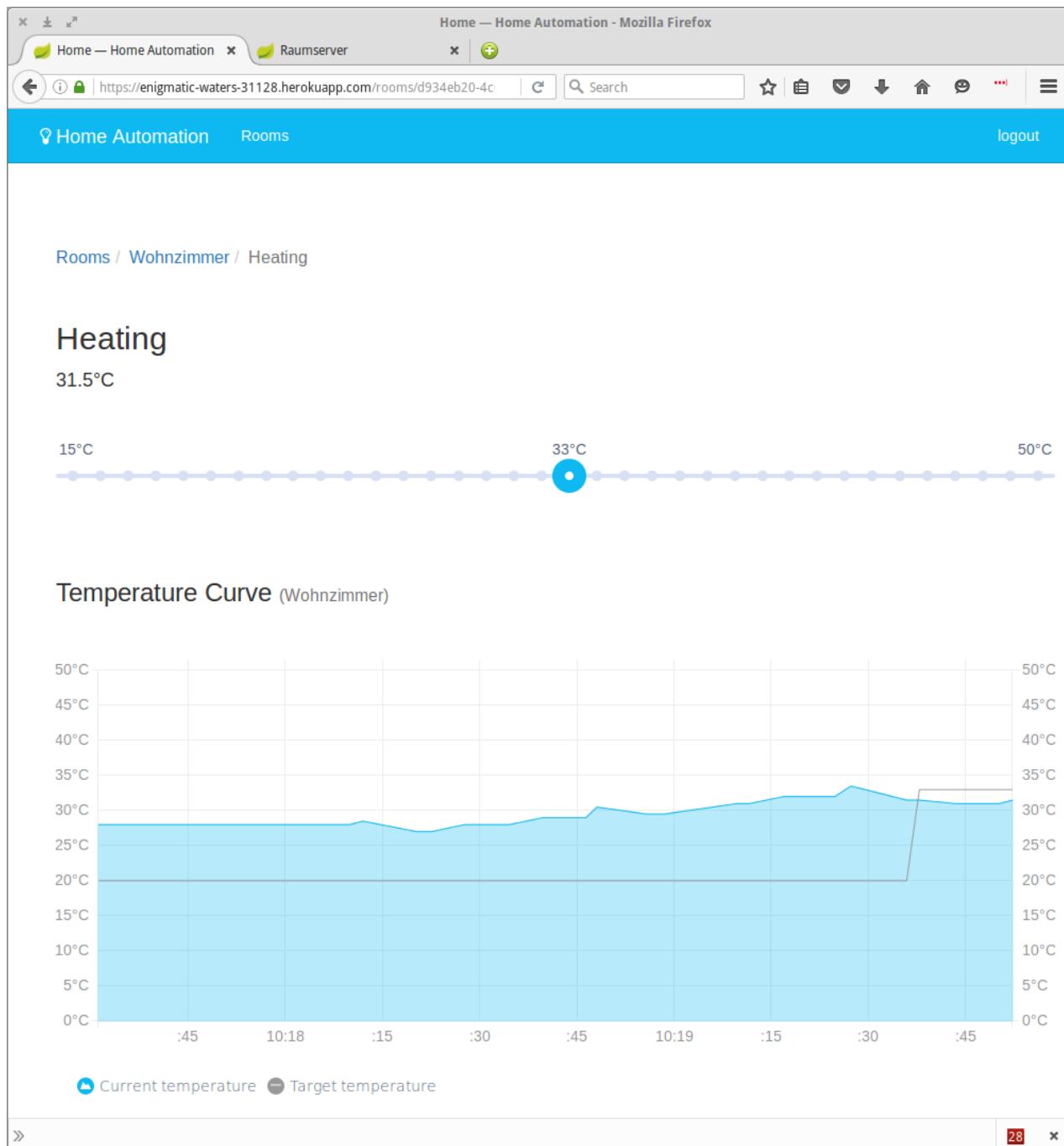


Abbildung 26: Web Frontend: Heizung

A Anhang

A.4.5 Web Frontend: Ändern der Temperatur

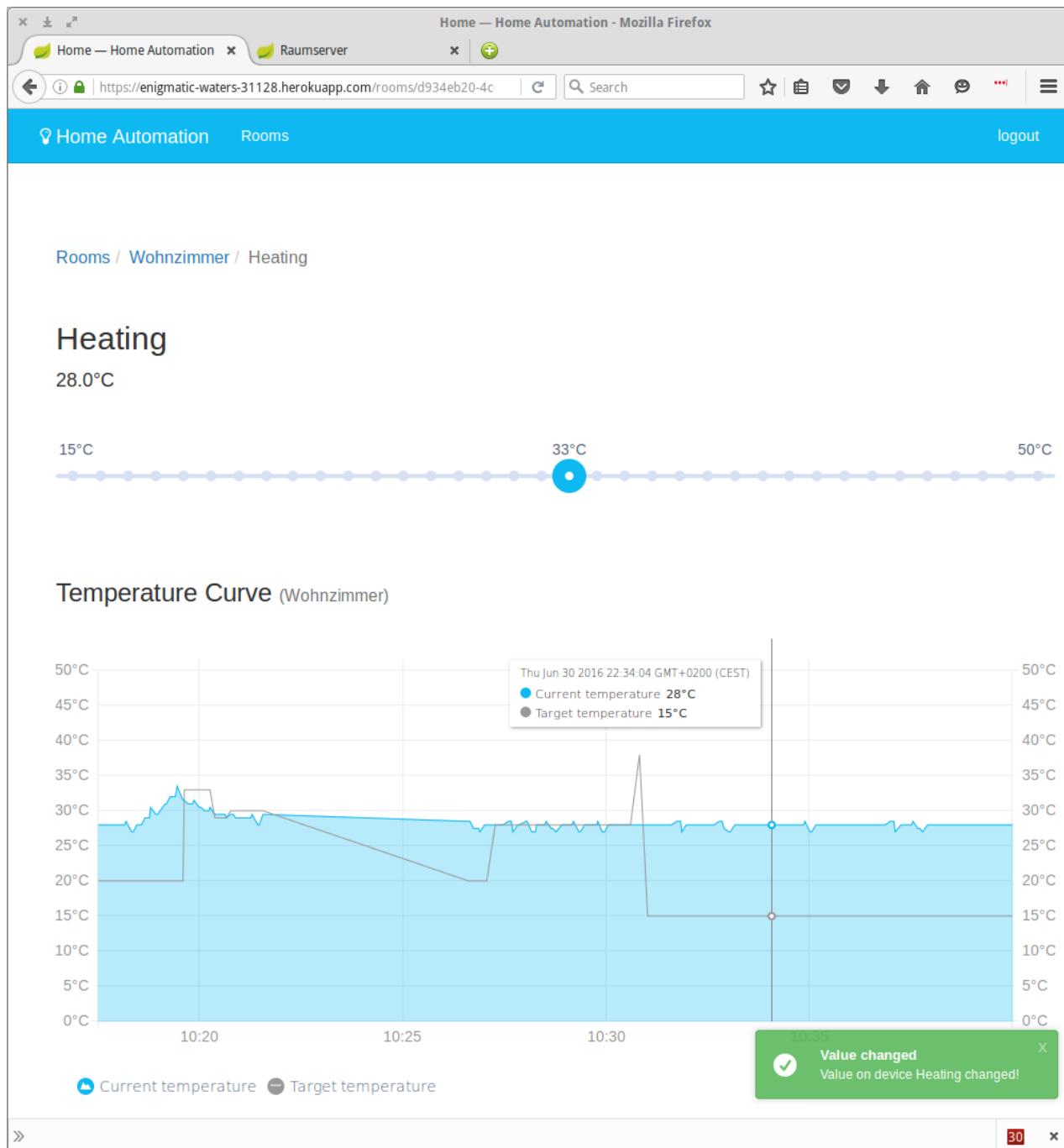


Abbildung 27: Web Frontend: Ändern der Temperatur

A Anhang

A.4.6 Konfigurationsseite des Raumservers

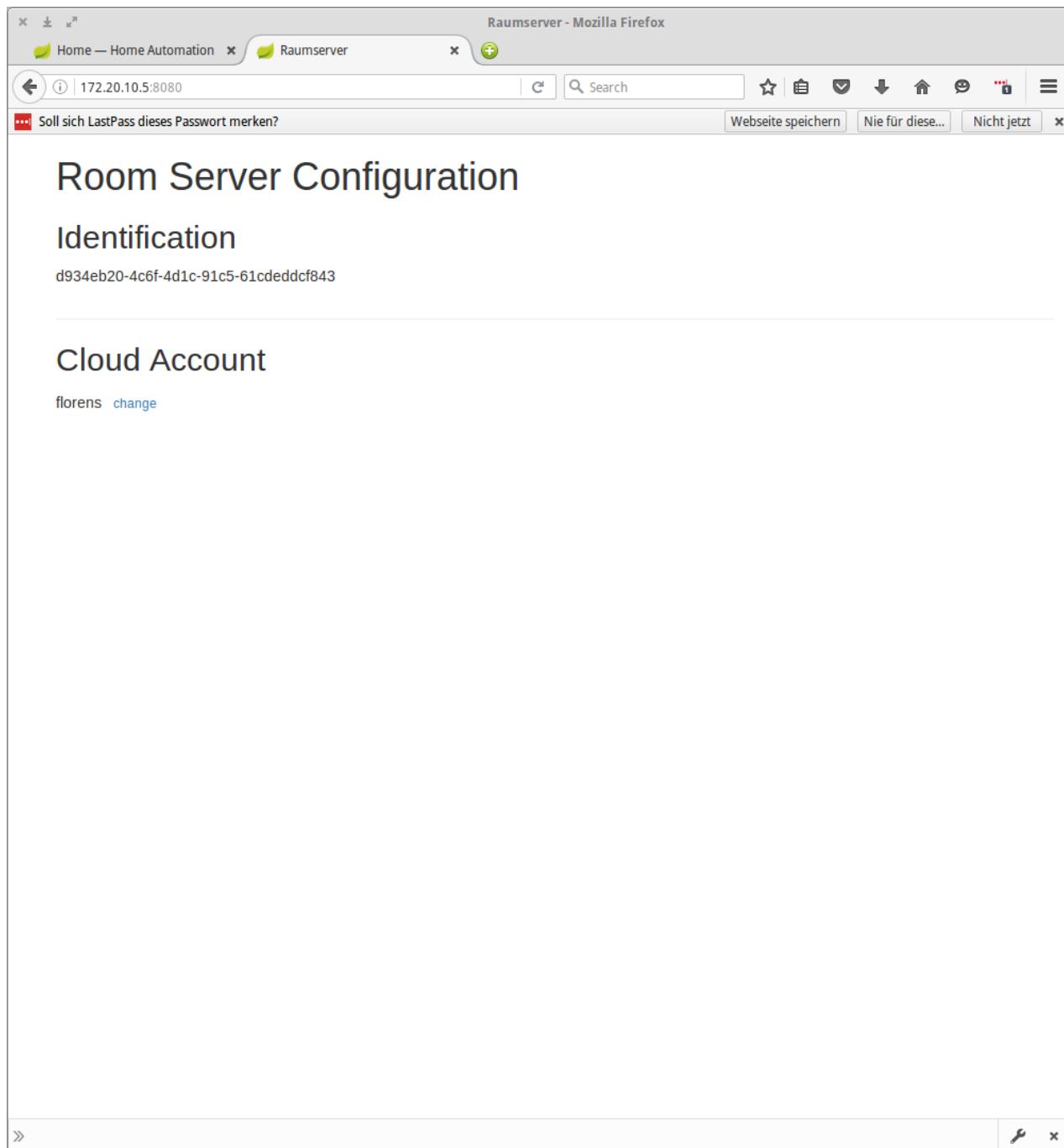


Abbildung 28: Konfigurationsseite des Raumservers

A Anhang

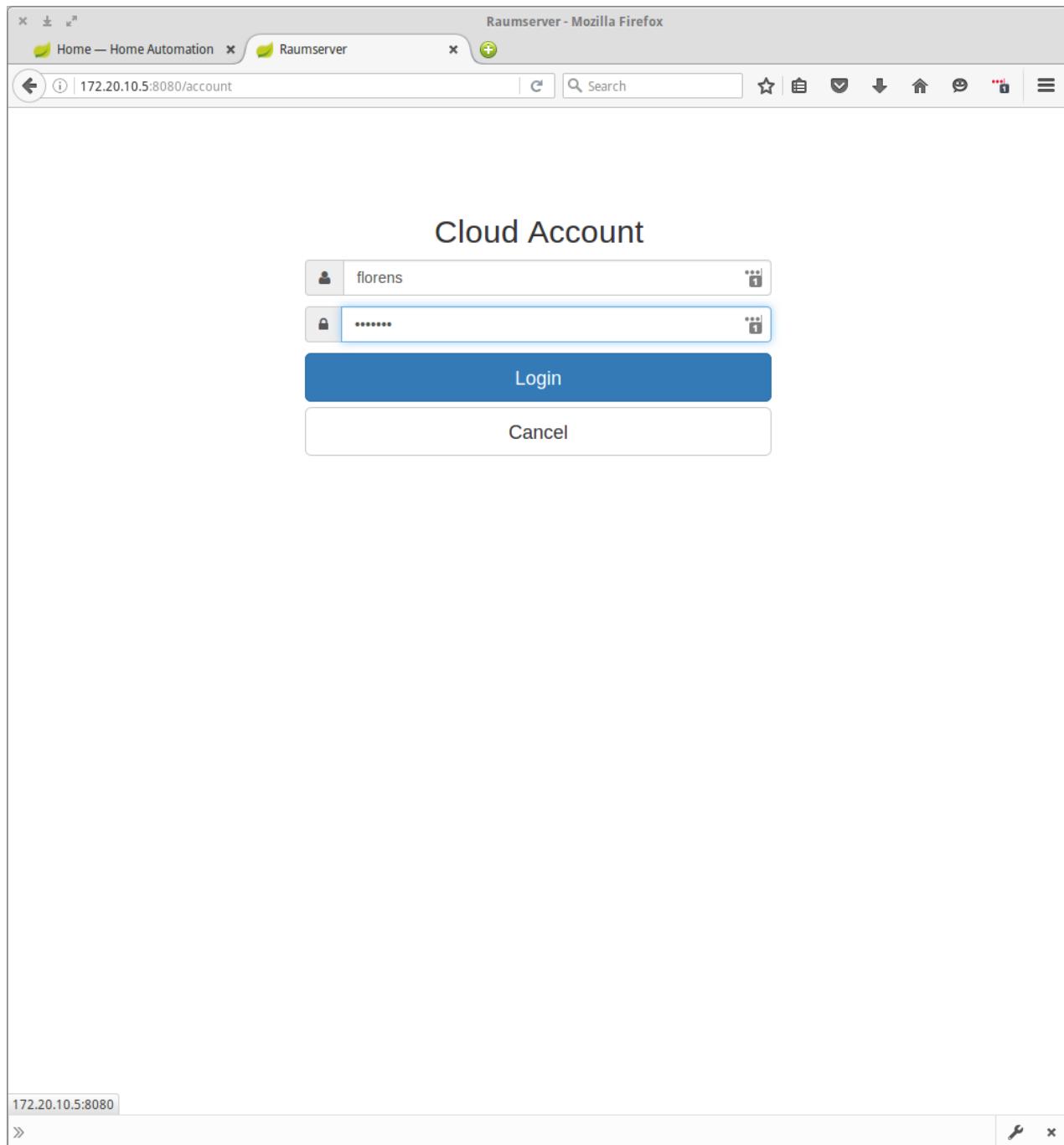


Abbildung 29: Eingabemaske für den Cloud Account

A Anhang

A.4.7 Aufspielen der Software auf den Atmega88

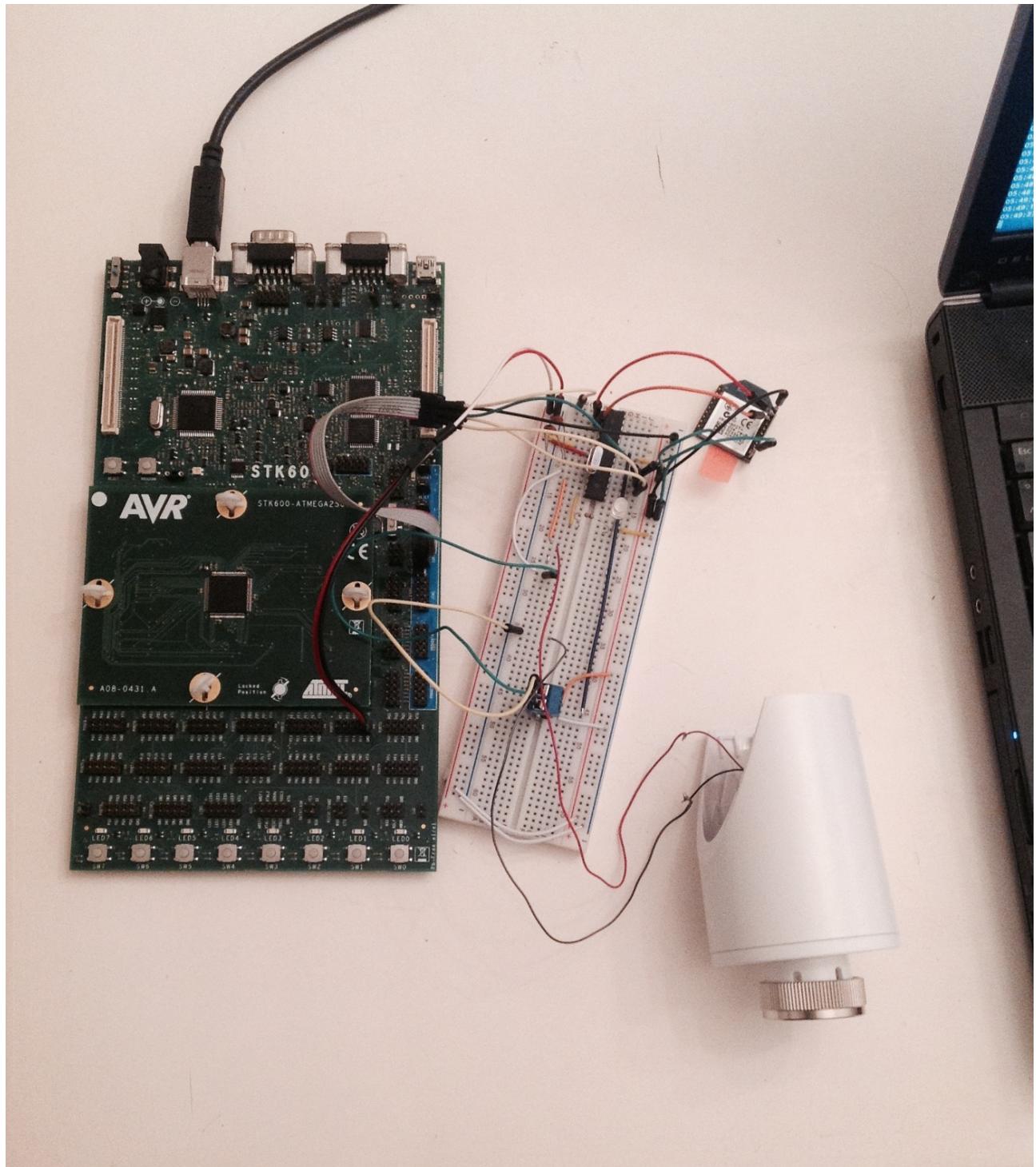


Abbildung 30: Aufspielen der Software auf den Atmega88

A.5 API Dokumentation

A.5.1 API Aufruf: Registrieren eines neuen Benutzers

Bezeichnung	Registrieren eines neuen Benutzers
URL	/api/register
Methode	POST
Parameter	<pre>{ "username": [String], "password": [alphanumeric] }</pre>
Erfolgsantwort	<pre>HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY X-AUTH-TOKEN: eyJpZCI6MTIsInVz... Content-Type: application/json; charset=UTF-8 Content-Length: 72 user successfully registered</pre>
Fehlerantwort	<pre>HTTP/1.1 422 Unprocessable Entity X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 17 password to short</pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/register' -i -X POST -H 'Content-Type: application/json' -d '{"username": "testuser", "password": "testpassword"}'</pre>

A Anhang

A.5.2 API Aufruf: Als registrierter Benutzer anmelden

Bezeichnung	Als registrierter Benutzer anmelden
URL	/api/login
Methode	POST
Parameter	<pre>{ "username": [String], "password": [alphanumeric] }</pre>
Erfolgsantwort	<pre>HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY X-AUTH-TOKEN: eyJpZCI6MTEsInVz...</pre>
Fehlerantwort	<pre>HTTP/1.1 401 Unauthorized X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY</pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/login' -i -X POST -H 'Content-Type: application/json' -d '{"username": "user", "password": "user"}'</pre>

A Anhang

A.5.3 API Aufruf: Einen Raum registrieren

Bezeichnung	Einen Raum registrieren
URL	/api/user/rooms/:id
Methode	PUT
URL-Parameter	erforderlich: <code>id=[String]</code> Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843
Parameter	<pre>{ "name": [String] }</pre>
Erfolgsantwort	<pre> HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 28 room successfully registered </pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843' -i -X PUT -H 'Content-Type: application/json' -H 'X-AUTH- TOKEN: eyJpZCI6MTEsInVzZXJuYW...' -d '{"name": "Wohnzimmer"}'</pre>

A Anhang

A.5.4 API Aufruf: Einen Raum von der Datenbank entfernen

Bezeichnung	Einen spezifischen Raum von der Datenbank entfernen
URL	/api/user/rooms/:id
Methode	DELETE
URL-Parameter	erforderlich: <code>id=[alphanumeric]</code> Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843
Erfolgsantwort	<pre>HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 25 room successfully removed</pre>
Fehlerantwort	<pre>HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 14 room not found</pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843' -i -X DELETE -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJuYW1l...'</pre>

A Anhang

A.5.5 API Aufruf: Einen spezifischen Raum abrufen

Bezeichnung	Einen spezifischen Raum abrufen
URL	/api/user/rooms/:id
Methode	GET
URL-Parameter	erforderlich: <code>id=[alphanumeric]</code> Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843
Erfolgsantwort	<pre>HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: application/json; charset=UTF-8 Content-Length: 69 "roomId": "d934eb20-4c6f-4d1c-91c5-61cdeddcf843", "name": "Wohnzimmer"</pre>
Fehlerantwort	<pre>HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY</pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJu...'</pre>

A Anhang

A.5.6 API Aufruf: Eine Liste aller Räume abrufen

Bezeichnung	Eine Liste aller Räume abrufen
URL	/api/user/rooms
Methode	GET
Erfolgsantwort	<pre>HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: application/json; charset=UTF-8 Content-Length: 71 [{"roomId": "d934eb20-4c6f-4d1c-91c5-61cdeddcf843", "name": "Wohnzimmer"}]</pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/user/rooms' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZ67KDv...'</pre>

A Anhang

A.5.7 API Aufruf: Ein neues Gerät registrieren

Bezeichnung	Ein neues Gerät registrieren
URL	/api/user/rooms/:roomId/devices/:deviceId
Methode	PUT
URL-Parameter	erforderlich: roomId=[alphanumeric] Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843 deviceId=[alphanumeric] Beispiel: id=a934eb20-4c6f-4d1c-91c5-61cdeddcf843
Parameter	<pre>{ "type": ["HEATING" "LIGHT"], "name": [String] }</pre>
Erfolgsantwort	<pre>HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 17 device registered</pre>
Fehlerantwort	<pre>HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 14 room not found</pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843/devices/a614eb20-4c6f-4d1c-91c5-61cdeddcf843' -i -X PUT -H 'Content-Type: application/json' -H 'X-AUTH-TOKEN: eyJpZCI6MT...' -d '{"deviceId": "a614eb20-4c6f-4d1c-91c5-61cdeddcf843", "name": "Licht", "type": "LIGHT"}'</pre>

A Anhang

A.5.8 API Aufruf: Ein spezifisches Gerät abrufen

Bezeichnung	Ein spezifisches Gerät abrufen
URL	/api/user/rooms/:roomId/devices/:deviceId
Methode	GET
URL-Parameter	erforderlich: roomId=[alphanumeric] Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843 deviceId=[alphanumeric] Beispiel: id=a934eb20-4c6f-4d1c-91c5-61cdeddcf843
Erfolgsantwort	<pre> HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: application/json; charset=UTF-8 Content-Length: 81 {"deviceId": "a614eb20-4c6f-4d1c-91c5-61cdeddcf843", "name": "Licht", "type": "LIGHT"} </pre>
Fehlerantwort	<pre> HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY </pre>
Beispiel	<pre> \$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843/devices/a614eb20-4c6f-4d1c-91c5-61cdeddcf843' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJuYW1lIjoi...' </pre>

A Anhang

A.5.9 API Aufruf: Ein Gerät aus der Datenbank entfernen

Bezeichnung	Ein spezifisches Gerät aus der Datenbank entfernen
URL	/api/user/rooms/:roomId/devices/:deviceId
Methode	DELETE
URL-Parameter	erforderlich: roomId=[alphanumeric] Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843 deviceId=[alphanumeric] Beispiel: id=a934eb20-4c6f-4d1c-91c5-61cdeddcf843
Erfolgsantwort	<pre> HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 27 device successfully removed </pre>
Fehlerantwort	<pre> HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 16 device not found </pre>
Beispiel	<pre> \$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843/devices/a614eb20-4c6f-4d1c-91c5-61cdeddcf843' -i -X DELETE - H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJuYW1...' </pre>

A Anhang

A.5.10 API Aufruf: Alle Geräte eines Raumes auflisten

Bezeichnung	Alle Geräte eines Raumes auflisten
URL	/api/user/rooms/:id/devices
Methode	GET
URL-Parameter	erforderlich: <code>id=[alphanumeric]</code> Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843
Erfolgsantwort	<pre> HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY Content-Type: application/json; charset=UTF-8 Content-Length: 83 [{"deviceId": "a614eb20-4c6f-4d1c-91c5-61cdeddcf843", "name": "Licht", "type": "LIGHT"}] </pre>
Fehlerantwort	<pre> HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Pragma: no-cache Expires: 0 X-Frame-Options: DENY </pre>
Beispiel	<pre> \$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843/devices' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsIyu2Z0L67KDvf1RPcmfC8dA...' </pre>

A Anhang

A.5.11 API Aufruf: Verändern der Gerätedaten

Bezeichnung	Verändern von Gerätedaten
URL	/api/user/rooms/:roomId/devices/:deviceId
Methode	PATCH
URL-Parameter	erforderlich: roomId=[alphanumeric] Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843 deviceId=[alphanumeric] Beispiel: id=a934eb20-4c6f-4d1c-91c5-61cdeddcf843
Parameter	<pre>{ "targetValue": [numeric], "value": [numeric] }</pre>
Erfolgsantwort	<pre> HTTP/1.1 200 OK X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 17 device updated </pre>
Fehlerantwort	<pre> HTTP/1.1 404 Not Found X-Content-Type-Options: nosniff X-XSS-Protection: 1; mode=block Cache-Control: no-cache, no-store, max-age=0, must-revalidate Expires: 0 X-Frame-Options: DENY Content-Type: text/plain; charset=UTF-8 Content-Length: 14 room not found </pre>
Beispiel	<pre>\$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843/devices/a614eb20-4c6f-4d1c-91c5-61cdeddcf843' -i -X PATCH -H 'Content-Type: application/json' -H 'X-AUTH-TOKEN: eyJpZCI6MT...' -d '{"deviceId": "a614eb20-4c6f-4d1c-91c5-61cdeddcf843", "targetValue": "0"}'</pre>