HOCHSCHULE HEILBRONN
TECHNIK WIRTSCHAFT INFORMATIK

Software Engineering Bachelor

Embedded Systems SS2016

Hausautomatisierung mit ZigBee

Projektdokumentation

Eugen Schlecht, Florens Hückstädt, Florian Wohlgemuth, Patrick Wohlgemuth

Inhaltsverzeichnis

Abkürzungsverzeichnis

AT Mode Transparent Mode

API Mode Application Programming Interface Mode

Rx-Schnittstell Receiver-Schnittstelle

Tx-Schnittstell Transmitter-Schnittstelle

1 Einführung

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

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.

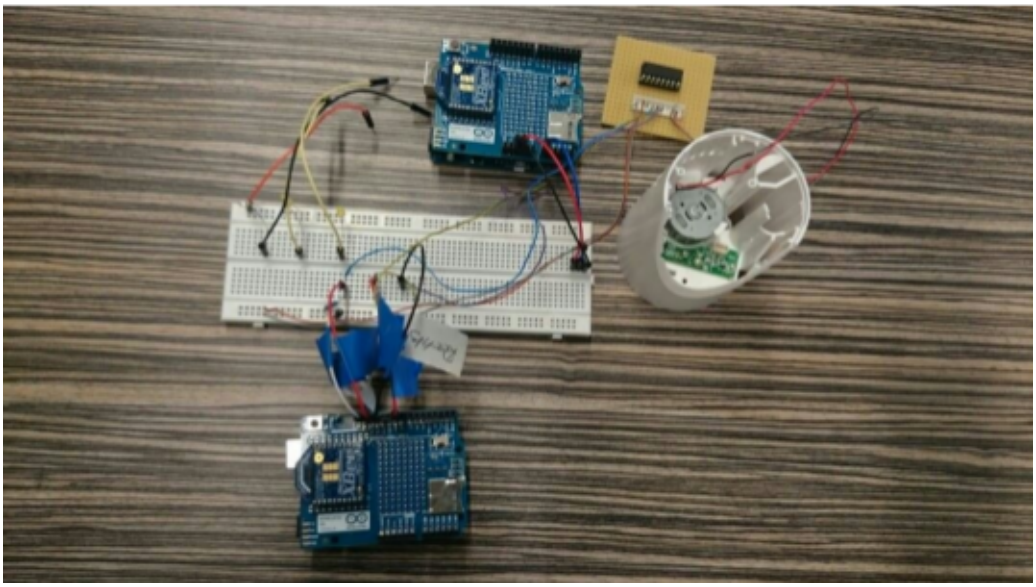

Abbildung 1: Projektaufbau der Vorgruppe aus dem Sommersemester 2015

2.1 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 getätigt 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 Vorteile[1]

- 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.

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 veringerten Komplexität der Boards können rasch Ergebnisse erzielt werden. Dies ist bei der Evaluierung von Konzepten hilfreich, bevor sie auf dem Zielgerät entwickelt werden.

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

2.2 Eingesetzte Hardware

Eine genau Liste der eingesetzen 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 sinnvole 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.

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-Adress 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 Thema Hausautomatisierung mit ZigBee. Der Stand der Vorgruppe ist ein durchgeführter Proof-Of-Concept mit Entwicklung eines prototypischen Systems zur Steuerung eines Heizungsventils. Die Absicht der Weiterführung des Themas ist Verarbeitung der gewonnen 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 inform eines Use-Case-Diagramms ist in Abbildung 2 dargestellt. Im Folgendem werden die Teilziele bzw. die Gütekriterien des geplanten Systems erläutert.

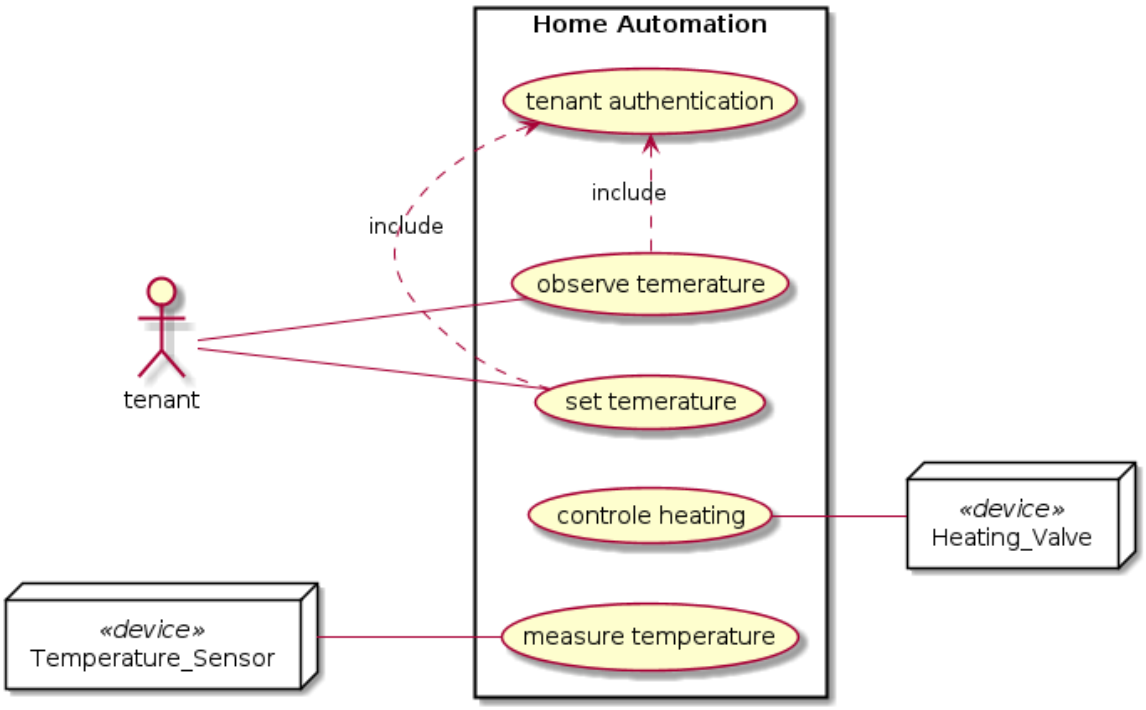

Abbildung 2: Use-Case-Diagramm Heizungssteuerung

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 Hausautomatisierung 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.

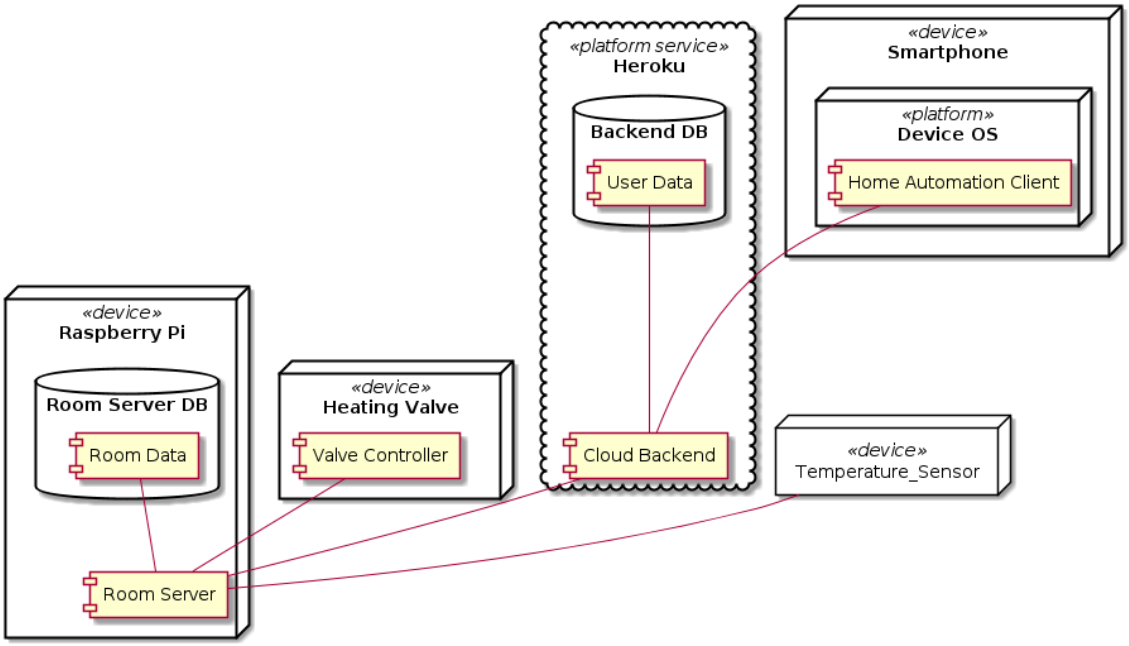

Abbildung 3: Sestemarchitektur

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.

4.1 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 verwalten 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 Unterunterabschnitt 4.1.2). Das API wird in Unterunterabschnitt 4.1.1 beschrieben. Der Vorgang des Deployments wird in Unterunterabschnitt 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.1 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}`.

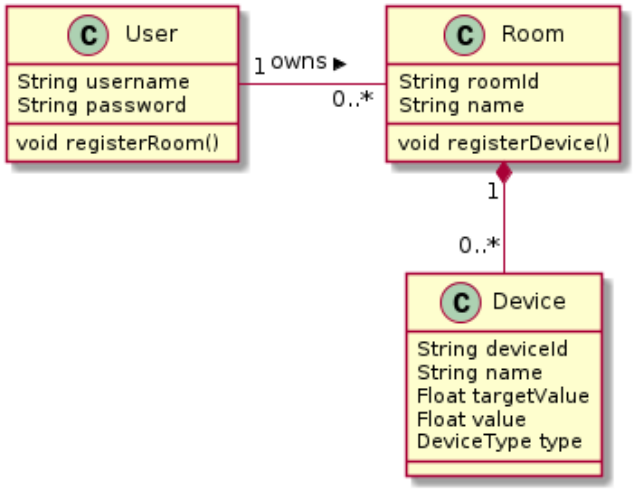

Abbildung 4: Klassendiagramm Hausautomatisierung

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

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"
}
```

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:
    eyJpZCI6MTEsInVzZXJuYW1lIjoidXNlciIsImV4cGlyZXMiOjE0NjI5NjkONzIwMTgsInJvbGVzIjpbIlVTRVIiXX0=.
    LDd74G0yJAuQChYR9P0AOmThfyl0GGlY19u2DcTcXyQ=
Content-Length: 0
Date: Sun, 01 May 2016 12:24:32 GMT
Via: 1.1 vegur
```

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

Bla bla bla.

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
```

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 auschließlich ein git push auf das Heroku Repository auszuführen[2]:

```
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 Unterunterabschnitt 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 Unterunterabschnitt 4.2.4). Zur

[2] https://devcenter.heroku.com/articles/deploying-spring-boot-apps-to-heroku

Kommunikation mit ZigBee-fähigen Geräten wird ein generisches Protokoll verwendet (Unterunterabschnitt 4.2.5). 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 Abschnitt 4.2.1 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 erleutert.

Raspberry Pi

Temperatursensor

XBee USB Module

Status-LED

4.2.2 Software

4.2.3 Schnittstellen

Der Raumserver bietet mehrere Schnittstellen. Eine Übersicht ist in Abbildung 5 dargestellt. Dabei wird ein Heizungsventil beipielhaft als Endgerät angegeben. Der Raumserver kommuniziert mit dem Cloud Backend über die bereitgestellte REST API (siehe Unterunterabschnitt 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 Unterunterabschnitt 4.2.4).

Die entgegengesetzte Kommunikation findet über das Messaging-Verfahren AMQP[3] 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-Fromat. Siehe dazu folgendes Beispiel:

```
{
 "deviceId": "123",
 "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 Nachrichten werden an die entsprechenden Endgeräte über das ZigBee-Protokoll weitergeleitet (siehe dazu Unterunterabschnitt 4.2.6).

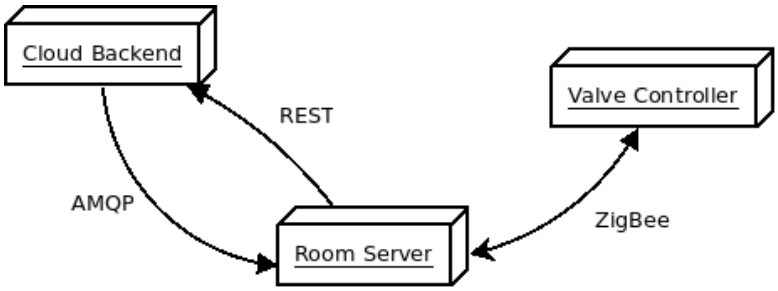

Abbildung 5: Schnittstellen des Raumservers

Innerhalb der ZigBee-Kommunikation nimmt der Raumserver die Rolle des Coordinator[4] ein und die Geräte wie zum Beispiel das Heizungsventil die Rolle des End Device (siehe Abbildung 6).

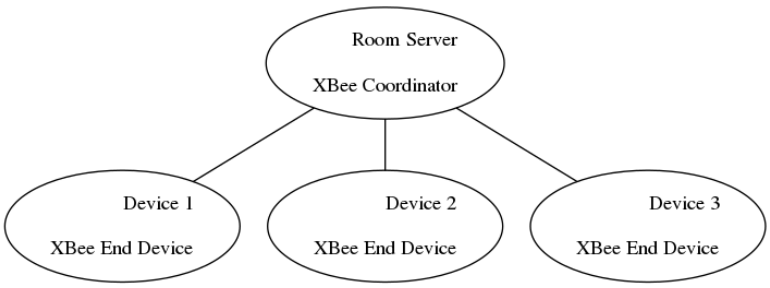

Abbildung 6: ZigBee-Netzwerk mit Raumserver und Geräten

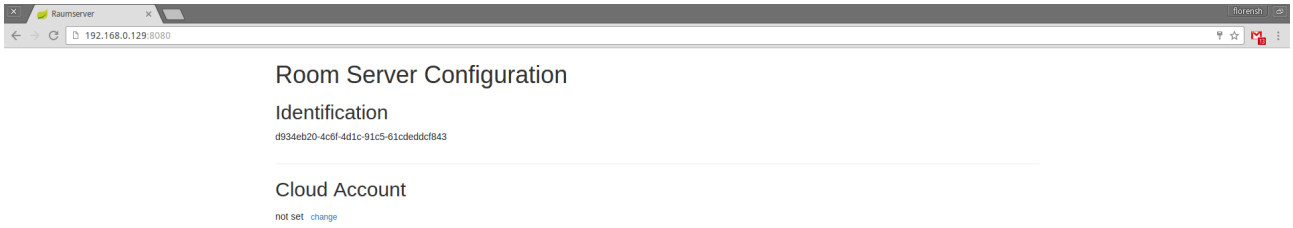

Abbildung 7: Konfigurationsseite des Raumservers

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

Tabelle 1: Komanndotypen

4.2.4 Konfiguration

4.2.5 Nachrichtenprotokoll

4.2.6 Kommunikation über ZigBee

4.2.7 Inbetriebnahme

Inatallation

alsdjflakjsdf

[3]https://www.amqp.org/
[4]https://en.wikipedia.org/wiki/ZigBee

Gerätetypen	**Bezeichnung**
01	Heizung
02	Schalter

Tabelle 2: Gerätetypen

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 3: Beispiel einer Kommunikation zwischen Raumserver und Heizung

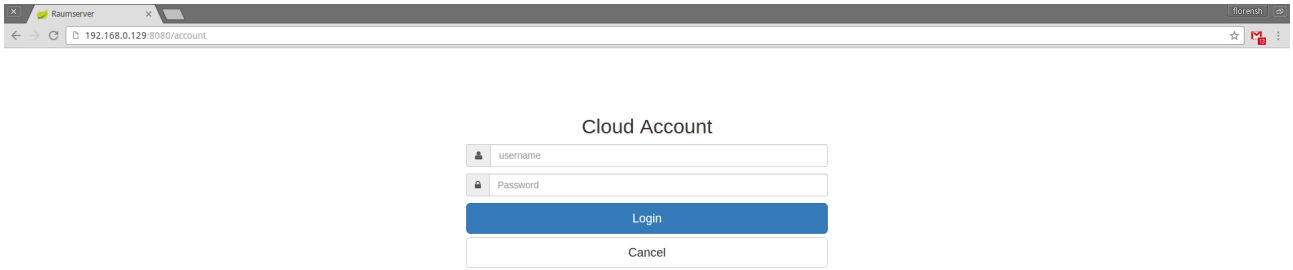

Abbildung 8: Eingabemaske für den Cloud Account

Starten und Stoppen

lajd lajsdf laj d

Logging

alsdjfla sd

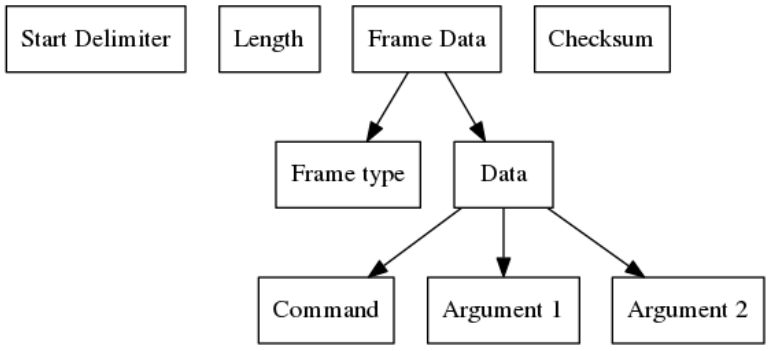

Abbildung 9: Raumserver XBee-Nachrichtenstruktur

4.3 Ventilsteuerung

Bla Bla Bla

4.3.1 Prototyp

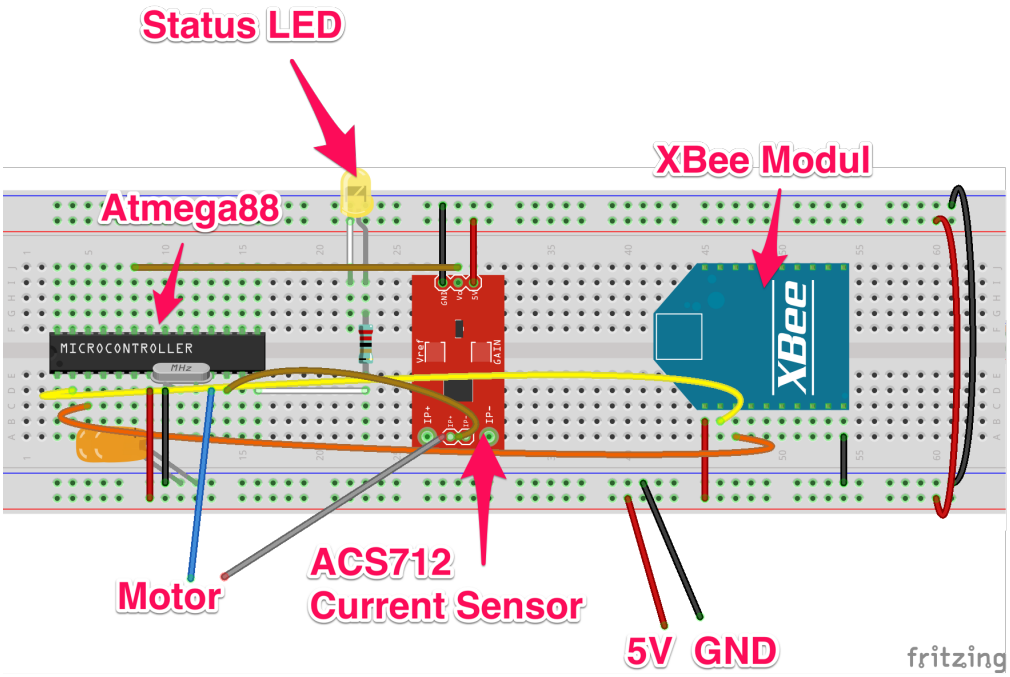

Abbildung 10: Prototyp Ventilsteuerung

4.3.2 Gehäuse

bla bla bla

bla bla bla

bla bla bla

bla bla bla

4.4 Frontend

Bla Bla Bla.

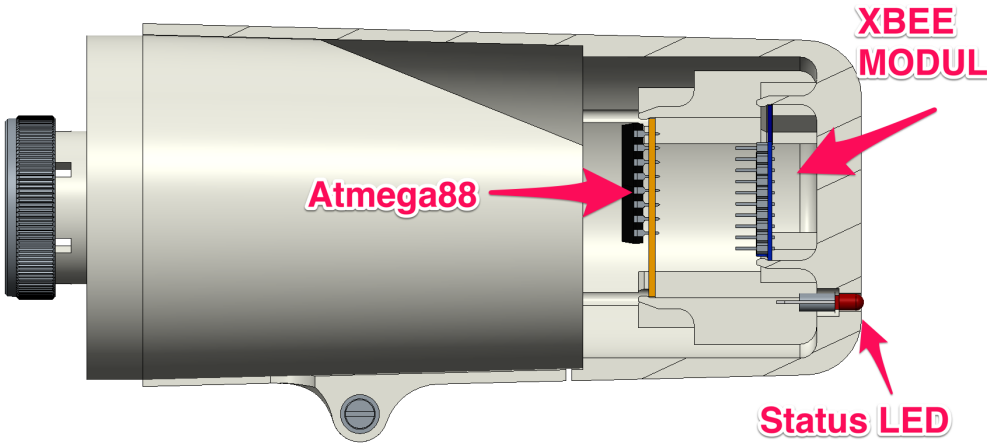

Abbildung 11: Gehäuseentwurf Querschnitt

5 Beispielhafte Kommunikation über das gesamte System

6 Projektorganisation

Abbildung 12: Gehäuseentwurf gerendert

Abbildung 13: Gehäuse gefertigt und montiert

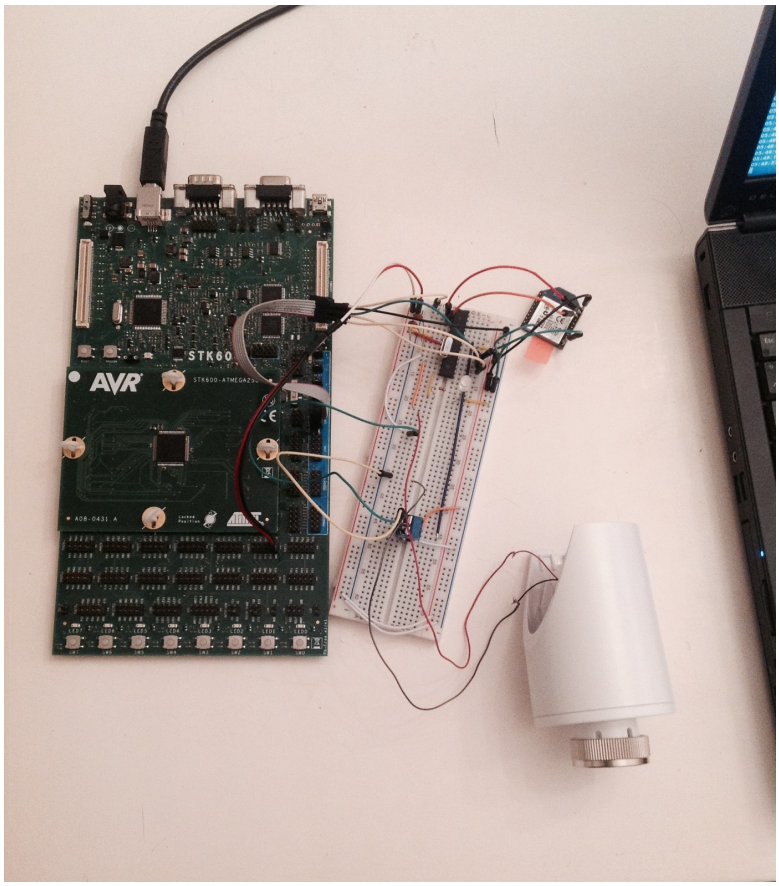

Abbildung 14: Aufspielen der Software auf den Atmega88

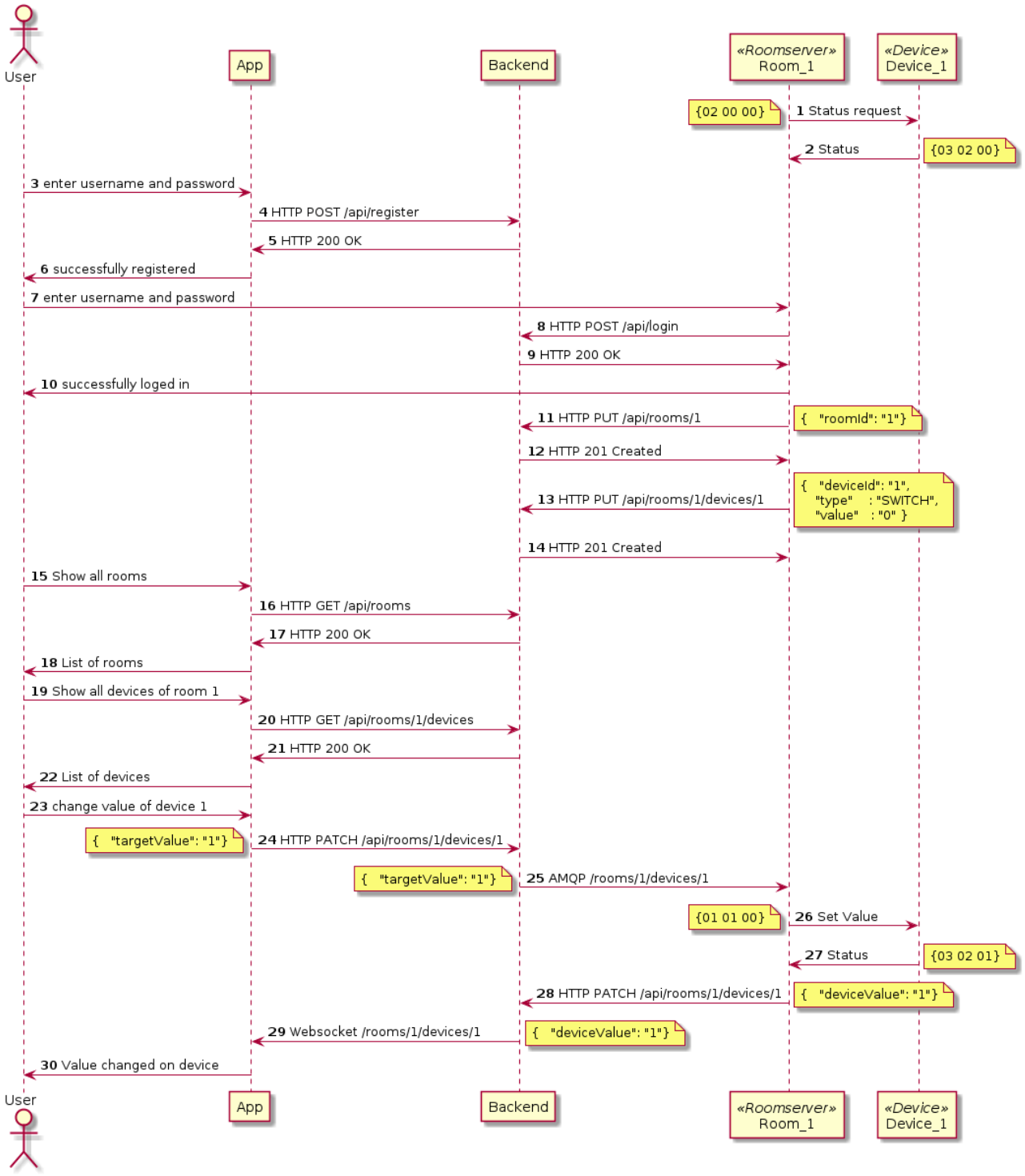

Abbildung 15: Beispielhafte Kommunikation über das gesamte System hinweg.

A Anhang

A.1 API Dokumentation

A.1.1 API Aufruf: Registrieren eines neuen Benutzers

Bezeichnung	Registrieren eines neuen Benutzers
URL	`/api/register`
Methode	POST

Parameter	`{`
	`"username":[String],`
	`"password":[alphanumeric]`
	`}`

Erfolgsantwort	`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`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/register' -i -X POST -H 'Content-Type:`
	`application/json' -d`
	`'{"username":"testuser", "password": "testpassword"}'`

A.1.2 API Aufruf: Als registrierter Benutzer anmelden

Bezeichnung	Als registrierter Benutzer anmelden
URL	`/api/login`
Methode	POST

Parameter	`{`
	`"username":[String],`
	`"password":[alphanumeric]`
	`}`

Erfolgsantwort	`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...`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/login' -i -X POST -H 'Content-Type:`
	`application/json' -d`
	`'{"username":"user", "password": "user"}'`

A.1.3 API Aufruf: Einen Raum registrieren

Bezeichnung	Einen Raum registrieren
URL	`/api/user/rooms/:id`
Methode	PUT
URL-Parameter	**erforderlich:** `id=[String]` Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843

Parameter	`{`
	`"name":[String]`
	`}`

Erfolgsantwort	`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`

Beispiel	`$ 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"}'`

A.1.4 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:** `id=[alphanumeric]` Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843

Erfolgsantwort	`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`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843' -i -X DELETE -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJuYW1l...'`

A.1.5 API Aufruf: Einen spezifischen Raum abrufen

Bezeichnung	Einen spezifischen Raum abrufen
URL	`/api/user/rooms/:id`
Methode	GET
URL-Parameter	**erforderlich:** `id=[alphanumeric]` Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843

Erfolgsantwort	`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"`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJu...'`

A.1.6 API Aufruf: Eine Liste aller Räume abrufen

Bezeichnung	Eine Liste aller Räume abrufen
URL	`/api/user/rooms`
Methode	GET

Erfolgsantwort	`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"}]`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZ67KDv...'`

A.1.7 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	`{`
	`"type":["HEATING" | "LIGHT"],`
	`"name":[String]`
	`}`

Erfolgsantwort	`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`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843/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"}'`

A.1.8 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	`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"}`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843/devices/a614eb20-4c6f-4d1c-91c5-61cdeddcf843' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsInVzZXJuYW1lIjoi...'`

A.1.9 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	`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`

Fehlerantwort	`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`

Beispiel	`$ 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...'`

A.1.10 API Aufruf: Alle Geräte eines Raumes auflisten

Bezeichnung	Alle Geräte eines Raumes auflisten
URL	`/api/user/rooms/:id/devices`
Methode	GET
URL-Parameter	**erforderlich:** `id=[alphanumeric]` Beispiel: id=d934eb20-4c6f-4d1c-91c5-61cdeddcf843

Erfolgsantwort	`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"}]`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61cdeddcf843/devices' -i -H 'X-AUTH-TOKEN: eyJpZCI6MTEsIyu2ZOL67KDvf1RPcmfC8dA...'`

A.1.11 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	`{`
	`"targetValue":[numeric],`
	`"value":[numeric]`
	`}`

Erfolgsantwort	`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`

Fehlerantwort	`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`

Beispiel	`$ curl 'http://localhost:8080/api/user/rooms/d934eb20-4c6f-4d1c-91c5-61 cdeddcf843/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"}'`