HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

CLOUD GUARD

IoT Management

Studienarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Herbstsemester 2017

Autoren:	Elias Brunner, Arooran Thanabalasingam
Betreuer:	Prof. Beat Stettler
Projektpartner:	CloudGuard Software AG, Zürich
Experte:	Michael Schneider

1 Aufgabenstellung

Die Aufgabenstellung wurde so unter https://avt.hsr.ch ausgeschrieben und entsprechend als Zielvorgabe angenommen.

1.1 Ausgangslage

IoT ist aktuell in aller Munde. Jedoch stehen Firmen oft vor dem Problem, dass IoT meistens eine grosse Masse an Geräten mit sich bringt. Standards zu diesem Thema sind noch jung und weisen noch einige Problemstellen auf.

1.2 Aufgabe

In dieser Arbeit soll eine Systemarchitektur erarbeitet werden, mit Hilfe derer mehrere tausend Geräte konfiguriert und überwacht werden können. Die Kommunikation zwischen den IoT Geräten und dem Server soll über das Standardprotokoll CoAP (http://coap.technology/) geschehen.

1.3 Erwartete Resultate

Es wird eine Java basierende Webanwendung erwartet, die für die Verwaltung der IoT Geräte eingesetzt werden kann. Für jedes Gerät bzw. Gruppe von Geräten kann eine Konfiguration auf dem Server hinterlegt werden, welche, sobald das Gerät online verfügbar ist, auf dem Device appliziert wird. Weiter können Geräte überwacht und in einem bestimmten Zyklus abgefragt werden. Die Authentisierung der verbundenen Geräte geschieht über Zertifikate, die während dem Bootstrapping-Prozess automatisch enrollt werden (https://tools.ietf.org/html/draft-ietf-acme-acme-07).

Ort, Datum	Unterschrift
.....	..... Prof. Beat Stettler

2 Abstract

Das Thema Internet of Things hat in den letzten Jahren stark an Beliebtheit gewonnen. Da erstaunt es nicht, dass viele Unternehmen auf diesen Zug aufspringen wollen und Produkte auf den Markt bringen, die in diesem Bereich angesiedelt sind. Entsprechend wird es in der Zukunft immer wichtiger sein, die Vielzahl an Sensoren mit einer Applikation verwalten zu können.

Doch bereits heutzutage kann das Verwalten von ein paar hunderten Computer in einem Unternehmen eine Herausforderung darstellen. IoT Geräte werden aber tendenziell in einer noch grösseren Stückzahl auftreten. Da verwundert es nicht, dass die Unternehmen daran interessiert sind, Lösungsansätze zu haben, die mit dieser regelrechten Flut an IoT Geräten umgehen können. Diese Studienarbeit befasst sich daher mit Aspekten wie Bootstrapping, Konfiguration und Monitoring von IoT Geräten.

Zunächst wurde daher das Constrained Application Protocol (CoAP), welches als Kommunikationsprotokoll verwendet werden sollte, genauer angeschaut und auch mit anderen Protokollen verglichen. Dabei stellte sich heraus, dass CoAP speziell für den Einsatz mit IoT Geräten konzipiert wurde. Im Gegensatz zu z.B. HTTP läuft CoAP über UDP und benötigt nicht so viel Overhead, was zu kleineren Paketen führt. CoAP ist zudem Datagramm basiert und könnte auch über SMS oder andere paketbasierten Kommunikationsprotokollen verwendet werden.

Basierend auf den gewonnenen Erkenntnissen wurde dann ein Konzept für das Bootstrapping, Konfigurieren und Monitoring der einzelnen IoT Geräte ausgearbeitet.

Anschliessend wurde eine geeignete Architekturlandschaft entwickelt. Der Backend Server wurde mit dem Play Framework in Scala geschrieben und bildet mit der Californium Bibliothek von Eclipse die Schnittstelle zu den IoT Geräten. Als Datenbank wurde MongoDB verwendet. Um mit dem Backend Server zu interagieren, wurde Node.js als Frontend - Webserver verwendet.

Mit der IoT Management Software konnte eine Applikation realisiert werden, bei der zunächst IoT Geräte anhand eines Lieferscheins importiert werden können. Sobald die Geräte online sind, melden sie sich beim Backend Server und tragen dort ihre Ressourcen ein. Zudem können Gruppen und Domains angelegt werden, was das Verwalten für eine grosse Anzahl an Geräten vereinfacht.

3 Inhaltsverzeichnis

4 Management Summary

4.1 Ausgangslage

In den letzten Jahren hat das Internet der Dinge engl. Internet of Things stark an Beliebtheit und an Zuwachst gewonnen [1]. Der Begriff IoT ist sehr dehnbar und es gibt keine einheitliche Definition dazu. Generell werden dabei allerdings Geräte mit Zugang zum Internet bezeichnet [2], welche in der Regel in bestimmten Abständen Messdaten an einen Server senden oder auch Anweisungen entgegennehmen können.
Es erstaunt daher auch nicht, dass viele Firmen interessiert sind, solche Geräte im eigenen Unternehmen einzusetzen, um Gebäude oder Produktionsstrassen zu überwachen oder gar zu steuern. Schon jetzt gibt es daher viele Unternehmen, die eine Vielzahl von unterschiedlichen IoT-Geräten einsetzten.

Es ist allerdings zu beachten, dass sich IoT-Geräte im Vergleich mit gewöhnlichen Computern in vielen Bereichen unterscheiden und sehr eingeschränkt sind. Sie sind beispielsweise nicht so leistungsstark oder verfügen über weniger Speicherkapazität und haben oftmals eine geringe Netzwerkbandbreite, über welche sie kommunizieren können. Auf der anderen Seite müssen IoT-Geräte in einer sehr viel grösseren Anzahl aufgesetzt, konfiguriert und überwacht werden als gewöhnliche Computer. Trotz der enormen Anzahl von IoT-Geräten, müssen diese einheitlich verwaltet und überwacht werden können.

Diese Studienarbeit soll sich daher mit Aspekten wie Bootstrapping, Konfiguration und Monitoring von IoT Geräten befassen, um einen entsprechenden Prototypen zu präsentieren.

4.2 Vorgehen

Um sich zunächst in das Thema IoT einzuarbeiten, wurden zu Beginn verschiedene Protokolle analysiert und das im Projekt verwendete Constrained Application Protocol (CoAP) mit den anderen Protokollen verglichen. Zudem wurde eine Problemanalyse der vorliegenden Probleme erstellt.
Desweitern wurden Anforderungen an die IoT Management Applikation ausgearbeitet und in der Anforderungsspezifikation zusammengetragen.

Basierend auf den gewonnenen Erkenntnissen und Analysen wurde dann ein Konzept mit Lösungsansätzen für das Bootstrapping, Konfigurieren und Monitoring der einzelnen IoT Geräte ausgearbeitet.

In einem nächsten Schritt wurde eine geeignete Architekturlandschaft entwickelt.

Anhand der erarbeiteten Konzepte, Anforderungen, Überlegungen zur Architektur oder sonstigen Analysen wurde ein Prototyp entwickelt. Durch den Prototypen konnte gezeigt werden, dass die ausgearbeiteten Lösungsansätze funktionieren.

4.3 Ergebnisse

Mit der IoT Management Software konnte ein funktionierender Prototyp für das Verwalten einer grossen Anzahl von IoT-Geräten realisiert werden. Durch den Einsatz von CoAP ist es möglich Geräte von unterschiedlichen Herstellern zu verwalten und zu überwachen. Durch das Importieren eines Lieferscheines können neue IoT-Geräte automatisch in der IoT Management Applikation erstellt werden. Sobald diese dann online sind, werden ihre Sensoren dem Server bekannt gegeben, welche ab diesem Zeitpunkt auch überwacht werden können.

Der Administrator hat zudem die Möglichkeit Gruppen/Domains und Modelle zu erstellen, um seine gewünschte Hierarchie aufzubauen. Somit wird das Verwalten für eine grosse Anzahl von IoT-Geräten stark vereinfacht.

5 Technischer Bericht

5.1 Einleitung

Unter dem Begriff IoT-Geräte werden grundsätzlich alle Geräte zusammengefasst, welche sich mit dem Internet verbinden können [2]. Die IoT-Geräte haben in den letzten Jahren stark an Beliebtheit gewonnen [1]. IoT-Geräte haben enormes Potenzial, aber sie bringen auch grosse Gefahren mit sich.

Eine Gefahr wäre, dass das Gerät bei der Kommunikation über das Internet durch eine Drittperson gekapert werden könnte und für Botnets missbraucht werden könnte. Ein möglicher Grund dafür wäre, dass die aufgebaute Verbindung über das Internet schlichtweg ungeschützt ist. Weitere Ursachen für eine solche Übernahme wäre, dass die laufende Software auf diesen Geräten voller Sicherheitslücken ist und nicht mehr weiter gepflegt wird [3]. Deshalb ist die Verwaltung und Wartung dieser Geräte sehr wichtig. Aber die grosse Anzahl und die grosse Vielfalt von IoT-Geräten stellt sich als eine schwierige Herausforderung dar.

Das Konzept für das Verwalten von IoT-Geräten wurde aus der Perspektive vom Industriepartner Cloudguard erstellt. Cloudguard ist ein Dienstleister, welcher unteranderem von anderen Unternehmen beauftragt wird, deren IoT-Geräte zu verwalten. Trotzdem können die entstandenen Lösungskonzepte für jedes andere Unternehmen oder jede andere Organisation problemlos verwendet werden.

5.2 Übersicht

Im Kapitel Domainanalyse wird auf die eigentliche Problemstellung des Projektes eingegangen. Das Kapitel wird in drei Abschnitte unterteilt. Im ersten Abschnitt, der Problemanalyse, werden die einzelne Problematiken genauer analysiert und beschrieben. Im zweiten Abschnitt geht es um die Technologieanalyse. Dabei werden die vorhandenen Technologien analysiert und untereinander verglichen. Es werden Vor- und Nachteile der einzelnen Technologien aufgezeigt. Eine ausführliche Geräteanalyse wird im dritten Abschnitt gemacht. Dort werden die verschiedenen Merkmale der IoT-Geräte analysiert und die Besonderheiten der einzelnen Geräte aufgezeigt.

Im Kapitel Anforderungsspezifikation werden die Annahmen und Einschränkungen des Projektes beschrieben. Ausserdem wird die Zielgruppe, für welche die Applikation gedacht ist definiert. Ebenfalls werden die funktionalen Anforderungen sowie die nichtfunktionalen Anforderungen konkretisiert. Dort befindet sich auch das Use Case Diagramm, welches einen Überblick über die funktionalen Anforderungen bietet.

Im Kapitel Konzept wird die Planung als Ablaufdiagramm dargestellt. Es stellt einen systematischen und chronologischen Ablauf von Anfang bis Schluss dar. Zudem wird auch textuell beschrieben was genau Schritt für Schritt abläuft, wer mit wem interagiert und was alles wann passiert. Ausserdem befindet sich hier auch das Domain Model. Im Abschnitt Erweiterungen werden einzelne Vorschläge für eine Erweiterung des angedach-

ten Konzeptes in Bezug auf das Netzwerk vorgestellt und warum diese von Vorteil sein könnten.

Im Kapitel Systemarchitektur werden die einzelnen Komponenten des Systems genau beschrieben. In Form des Deployment Diagramms werden die Lokalitäten von Komponenten dargestellt und dabei werden die Protokolle, welche zwischen den Komponenten verwendet wird, aufgezeigt. Desweiteren werden die ausgewählten Frameworks für die jeweiligen Tiers beschrieben. Mit dem Technologie Stack werden die eingesetzten Technologien übersichtlich dargelegt. Ausserdem wird die logische Architektur in Form von Klassendiagrammen dargestellt.

5.3 Domainanalyse

5.3.1 Einführung

5.3.1.1 Übersicht

Im Abschnitt Problemanalyse wird auf die eigentliche Problemstellung des Projektes eingegangen. Was ist das eigentliche Ziel der Arbeit und welche Probleme gibt es, dieses Ziel zu erreichen.

Im Abschnitt Technologieanalyse werden die verschiedenen Technologien analysiert und deren Vor- und Nachteile für eine mögliche Umsetzung aufgelistet.

Im Abschnitt Geräteanalyse werden die Merkmale von Gerät unter die Lupe genommen. Was gibt es bei den einzelnen Geräten zu beachten bzw. was könnten die Schwierigkeiten für eine mögliche Umsetzung darstellen.

5.3.2 Problemanalyse

In diesem Abschnitt wird auf die eigentliche Problemstellung bzw. das Ziel des Projektes eingegangen. Dabei werden einzelne Problematiken genauer analysiert und beschrieben.

5.3.2.1 Allgemeines Ziel

In dieser Arbeit soll eine Systemarchitektur erarbeitet werden, mit Hilfe derer mehrere tausend Geräte konfiguriert und überwacht werden können. Die Kommunikation zwischen den IoT Geräten und dem Server soll über offene Standardprotokolle geschehen.

Es soll eine Webanwendung erstellt werden, die für die Verwaltung der IoT Geräte eingesetzt werden kann. Für jedes Gerät bzw. Gruppe von Geräten kann eine Konfiguration auf dem Server hinterlegt werden, welche, sobald das Gerät online verfügbar ist, auf dem Device appliziert wird. Weiter können Geräte überwacht und in einem bestimmten Zyklus abgefragt werden. Die Authentisierung der verbundenen Geräte geschieht über Zertifikate, die während dem Bootstrapping-Prozess automatisch enrollt werden.

5.3.2.2 NAT-Problematik

Ein grosses Problem stellt die Problematik von NAT für das Erreichen des Ziels der Arbeit dar. Da ein Netzwerk gegen aussen also gegen das Internet meist nur eine öffentliche IP-Adresse hat, werden eingehende Verbindungen via Router an das richtige Endgerät gesendet. Das bedeutet allerdings dass der Router vorgängig wissen muss, von welchem Endgerät die Verbindung kam und kann die eingehende Verbindung dann via Port (PAT) dem richtigen Endgerät bzw. der internen IP-Adresse zuweisen.

Im Umkehrschluss bedeutet das, dass unsere Verwaltungsapplikation keine direkte Verbindung zu einem IoT-Gerät hinter einem NAT-Router herstellen kann. Die Verbindung

muss vom IoT-Gerät ausgehen. Nur so kann eine Kommunikation zwischen IoT-Gerät und Verwaltungstool sichergestellt werden.

5.3.2.3 Konfigurationsstruktur

Jeder Hersteller von IoT-Geräte legt eine eigene Struktur für Konfigurationsdaten fest. Dadurch ist die Darstellung von Konfigurationsdaten in einem GUI in einem heterogenen Umfeld unmöglich. Ein weiteres Problem ist, falls wir ein Konfigurationstemplate für eine Gerätekategorie wie Router erstellen, kann man es nicht auf Einsatzgeräte, welche von anderen Herstellern sind, anwenden.

5.3.2.4 Hierarchie

Eine weitere Problematik stellt die Tatsache dar, dass man in der Verwaltungssoftware nicht vorgängig eine vordefinierte Hierarchie einbetten kann. Durch die verschiedenen Anwendungsmöglichkeiten von IoT-Geräten kann z.B. einen Einteilung in Funktionen gemacht werden. Man kann sich aber als Administrator der Software genauso gut für die ortsabhängige Einteilung z.B. in Gebäude 1, Gebäude 2, etc. entscheiden. Das Einzige was sicher ist, ist die Unterteilung in Nutzer bzw. die Firmen.

5.3.2.5 Modell

Unabhängig von der hierarchischen Einteilung in der Verwaltungssoftware muss es möglich sein, über alle Hierarchiestufen die entsprechenden IoT-Geräte zu selektieren. Zum Beispiel möchten man alle Router finden, welche eine Firma in der Verwaltungssoftware hat. Eine hierarchische Einteilung ist hier also nicht genügend.

5.3.3 Technologieanalyse

In diesem Abschnitt werden die schon vorhandenen Technologien analysiert und untereinander verglichen. Es werden Vor- und Nachteile der einzelnen Technologien aufgezeigt.

5.3.3.1 Anforderung an Protokoll

Beim Entwerfen von IoT Lösungen treten am häufigsten die folgenden Bedenken, gemäss dem IoT Developer Survey 2017 [4], auf.

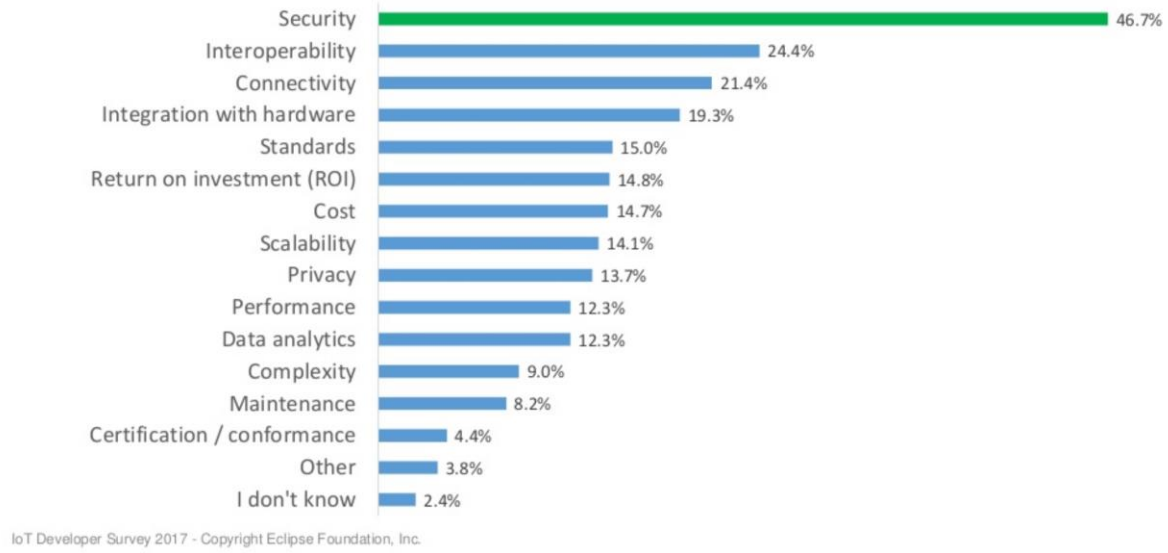


Abbildung 1 Bedenken [4]

Anhand der oben aufgelisteten Bedenken, werden die folgenden Anforderungen an das Protokoll formuliert. Es wird erwartet, dass bei dem anzuwendenden Protokoll die Security eine Kernfunktionalität ist. Ein weiterer Punkt ist, dass das Protokoll mit anderen Anwendungsprotokollen wie z.B. HTTP reibungslos zusammenarbeiten kann. Der Verbindungsaufbau soll einfach ermöglicht werden. Das Protokoll sollte zudem ein offener Standard sein.

Gerade in der IoT-Welt ist es wichtig, dass das verwendete Protokoll auch auf leistungsschwachen Sensoren läuft. Da in einem heterogenen Geräte-Umfeld unterschiedlichste Content-Formate verwendet werden, sollte das Protokoll Content-Aushandlung unterstützen. Das verwendete Protokoll soll ein angenehmes Paradigma verwenden, so dass die Entwicklung beschleunigt und die Wartung erleichtert wird. Zudem soll das Protokoll die Skalierbarkeit erleichtern.

5.3.3.2 Umfrage 2017

Gemäss dem IoT Developer Survey 2017 [4] werden folgende Messaging Standards benutzt.

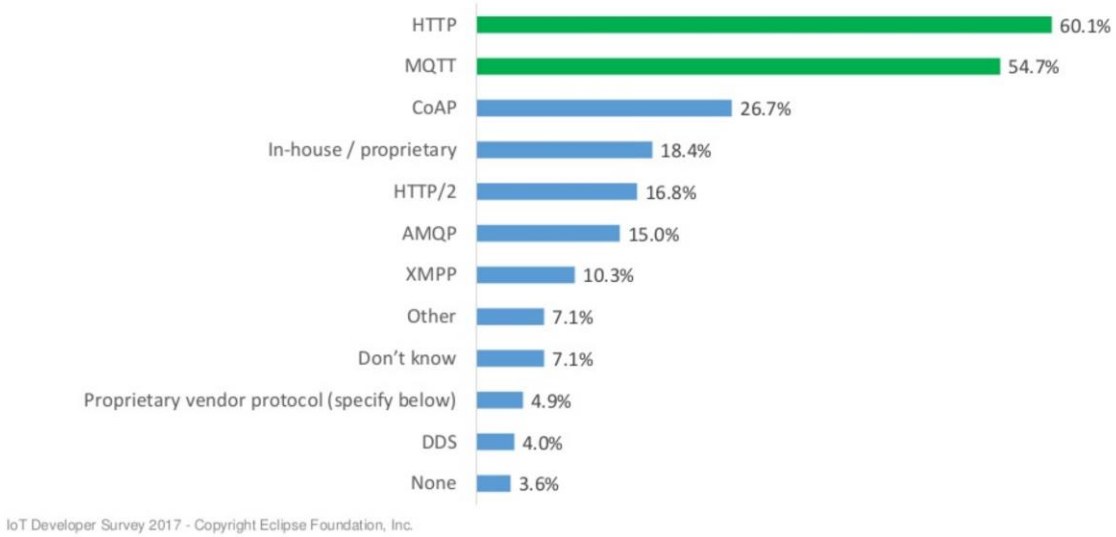


Abbildung 2 Messaging Standards bei IoT [4]

5.3.3.3 HTTP - Hypertext Transfer Protocol

HTTP 1.1 ist ein flexibles, unkompliziertes Übertragungsprotokoll. Es hat eine riesige Entwicklerbasis. Aber die HTTP Fehler, welche Webentwickler jahrelang verärgert haben, können eine noch grössere Auswirkung auf IoT-Entwickler haben. Beispielsweise wird Plain Text Strings ohne Komprimierung für den HTTP Header verwendet, wodurch das Netzwerkprotokoll aufgebläht wird.

Die ursprüngliche HTTP-Spezifikation wurde auf der Idee kurzlebiger Netzwerkverbindungen entworfen. Der Client öffnet eine Verbindung und verlangt eine Seite. Dann liefert der Server die Seite und die Verbindung wird geschlossen. Aber heutzutage ruft eine durchschnittliche Webseite über ein Dutzend Ressourcen gleichzeitig ab. HTTP 1.1 führte einige Funktionen ein, um Verbindungen offen zu halten und für kurze Zeit wiederverwendbar zu machen. Aber im Wesentlichen bleibt HTTP auf kurzlebige Verbindungen konzentriert.

Wenn der Netzwerkaspekt eines IoT-Geräts berücksichtigt wird, dann ist ein Verbindungsaufbau insbesondere mit TLS-Aushandlung teuer in Bezug auf Energieverbrauch und Zeit. Jede zusätzliche Verbindung bringt einen erheblichen Rechenverbrauch mit sich. Wiederholtes Öffnen von "teuren" Netzwerkverbindungen ist eine unnötige Belastung für das Gerät. Das HTTP Payload Modell eignet sich hervorragend für IoT aber ein besseres Protokoll würde die Sicherheit und die Übertragungsgrösse optimieren. [5]

5.3.3.4 MQTT - Message Queue Telemetry Transport

MQTT ist ein Publish/Subcribe Nachrichtenprotokoll für leichte M2M Kommunikationen. Es wurde ursprünglich von IBM entwickelt, nun ist es aber ein offener Standard. MQTT basiert auf einem Client/Server Modell und es ist nachrichtenorientiert

Jeder Sensor ist ein Client und verbindet sich via TCP zu einem Server, welcher als Broker bekannt ist. Jede Nachricht ist ein diskreter Daten-Chunk, welcher opak für den Broker ist. Jede Nachricht wird zu einem sogenannten Topic veröffentlicht. Jeder Client darf mehrere Topics abonnieren. All jene Clients, welche ein Topic abonniert haben, erhalten alle Nachrichten, welche zu diesem Topic veröffentlicht werden.

Jeder MQTT Client muss TCP unterstützen und muss typischerweise die Verbindung zum Broker ständig offen halten. In manchen Umgebungen, wo hoher Paketverlust besteht oder Computing Resources knapp sind, ist dies ein Problem. MQTT Nachrichten können für jeden Zweck verwendet werden aber alle Clients müssen das Nachrichtenformat im Voraus kennen, um zu kommunizieren. [6]

5.3.3.5 CoAP - Constrained Application Protocol

CoAP ist ein Dokument-Übertragungsprotokoll wie HTTP. Im Gegensatz zu HTTP wurde CoAP für Bedürfnisse von Constrained Devices entworfen. CoAP Pakete sind viel kleiner als HTTP TCP Flows. Bitfields und Mappings von Strings zu Integers werden intensiv genutzt, um den Speicherbedarf zu minimieren. Pakete sind einfach zu generieren und sie können an Ort und Stelle geparsed werden, ohne extra RAM von den Constrained Devices zu konsumieren.

CoAP läuft über UDP. Clients und Server kommunizieren über verbindungslose Datagramme. Retries und Reordering sind im Applikations -Stack implementiert. Dadurch wird der Bedarf für TCP überflüssig und es ist auch möglich ohne TCP das volle IP Networking für kleine Microcontroller anzuwenden. In CoAP kann UDP Broadcast und Multicast für die Adressierung gebraucht werden.

Auch CoAP verfolgt das Client/Server Modell. Clients machen Request zum Server und der Server senden Responses zurück. CoAP wurde entworfen, um via HTTP und dem RESTful Paradigma durch einfache Proxys miteinander zu interagieren. Weil CoAP datagramm-basierend ist, kann es auf SMS oder auf anderen paketbasierten Kommunikationsprotokollen verwendet werden. CoAP bietet eine eingebaute Unterstützung für Content Aushandlung und Discovery an. [6]

5.3.3.6 Vergleich

Alle drei Anwendungsprotokolle sind offene Standards und laufen über das Internet-protokoll.

HTTP

Vorteile	Nachteile
• Paradigma REST • TLS • Content Aushandlung • ausgereifter und stabiler Standard	• grosse Header • nur TCP

MQTT

Vorteile	Nachteile
• kleine Header • TLS • ausgereifter und stabiler Standard	• Paradigma Publish/subscribe • nur TCP • Die Verbindung zum Broker muss offen gehalten werden. • Content Format muss im Voraus bekannt sein

CoAP

Vorteile	Nachteile
• Paradigma REST • kleine Header • DTLS • UDP basierend • Content Aushandlung • Es kann mit HTTP reibungslos in-teragieren.	• Manche Komponenten sind noch im Zustand des Entwurfes im Standard.

5.3.3.7 Entscheidung

MQTT und CoAP bieten optimierte Paketgrössen für den Einsatz im IoT-Umfeld an. Aber nur CoAP hat einen optimalen Verbindungsaufbau, welcher durch UDP ermöglicht wird. CoAP verwendet das Paradigma “REST”, mit welchem wir schon vertraut sind. Deshalb haben wir uns für CoAP Protokoll entschieden.

5.3.4 Geräteanalyse

In diesem Abschnitt werden die verschiedenen Merkmale von IoT-Geräten analysiert. Welche Arten von Geräten gibt es überhaupt und auf was muss bei den einzelnen Geräten geachtet werden. Ein Gerät kann mehrere Merkmale haben, kann nur einer Klasse angehören.

5.3.4.1 Merkmale

Die Merkmale repräsentieren die besonderen Bedürfnisse eines Gerätes.

5.3.4.1.1 Moving

Es kann durchaus vorkommen, dass IoT-Geräte in Fahrzeugen wie Busse, Autos oder sonstige fahrbare Fahrzeuge eingebaut werden können. Daher ist es auch möglich, dass sich ein Gerät z.B. für einige Minuten nicht bei der Verwaltungssoftware meldet, da es sich vielleicht in einem Tunnel oder in einer Tiefgarage befindet. Hier stellt sich die Frage, ab wann wird ein Alarm ausgelöst.

5.3.4.1.2 Behind-NAT

Wie schon im Abschnitt NAT-Problematik erwähnt, gibt es häufig IoT-Geräte, welche sich hinter einem NAT-Router befinden. Die Kommunikation zu diesen Geräten kann nur vom Gerät selbst ausgehend aufgebaut werden, da sonst der NAT-Router nicht weiss, an welches Endgerät er die eingehende Verbindung weiterleiten soll.

Mehr dazu ist im Abschnitt Problemanalyse zu finden.

5.3.4.1.3 Battery-Powered

Es wäre auch denkbar, dass IoT-Geräte ausschliesslich mittels einer Batterie mit Strom versorgt werden. Hier stellen sich Fragen wie die Batterie geschont werden kann. Oder sollte die Batterie zu Neige gehen, wie das IoT-Gerät das der Verwaltungssoftware mitteilen kann.

5.3.4.1.4 Wireless

Kabellose Übertragung wird öfters bevorzugt, wenn die geografische Lage nicht via Kabel erschlossen werden kann oder die Kosten für Kabelverlegung eingespart wird. Die Zuverlässigkeit von kabellosen Geräten ist viel niedriger als mit kabelgebundenen Geräten, weil die Datenpakete öfters verloren gehen können. Die Geschwindigkeit von Datenübertragung ist entsprechend auch langsamer.

Gemäss dem IoT Developer Survey 2017 [4] werden mehr Wi-Fi als Ethernet verwendet.

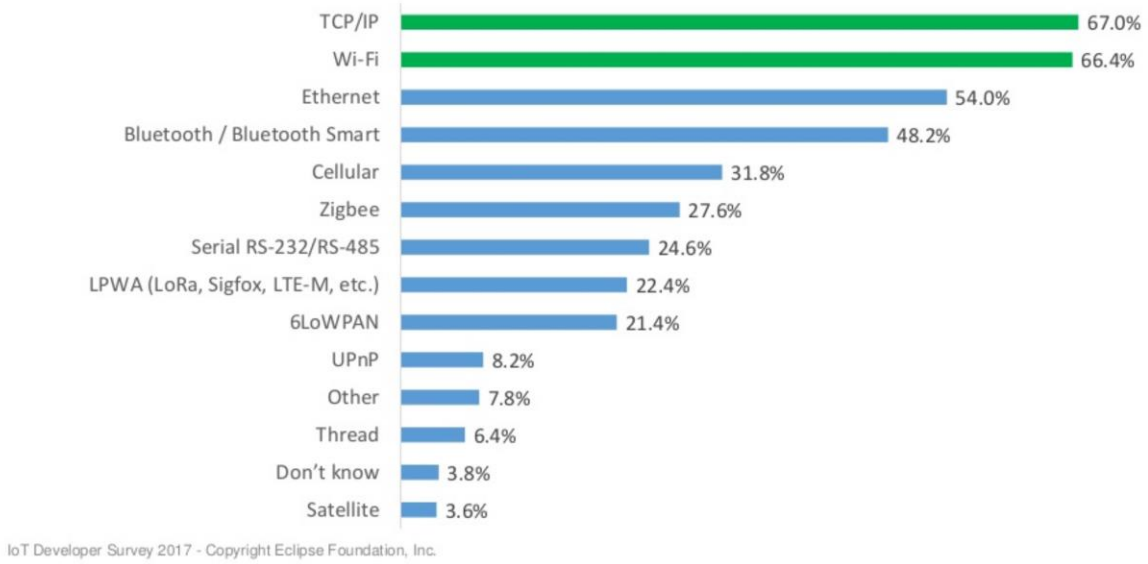


Abbildung 3 Übertragungsart [4]

5.3.4.2 Klassen

IETF hat mit RFC 7227 [7] CoAP Terminologie die IoT-Geräte anhand ihrer Leistungsfähigkeit in Klassen eingeteilt.

Name	Data size (bsp RAM)	Code size (bsp Flash)
Class 0	< 10 KiB	< 100 KiB
Class 1	~ 10 KiB	~ 100 KiB
Class 2	~ 50 KiB	~ 250 KiB

5.3.4.2.1 Class 0

Die Class 0 Geräte sind sehr klein und leistungsschwach. Sie sind sehr stark in der Memory und Processing Fähigkeiten eingeschränkt, so dass in den meisten Fällen die Ressourcen nicht ausreichen, um direkt ins Internet über eine sichere Art und Weise zu kommunizieren. Mit Hilfe vom grösseren Geräte wie Proxies können die Class 0 Geräte sicher ins Internet kommunizieren.

5.3.4.2.2 Class 1

Die Class 1 Geräte sind ziemlich in ihrer Processing Fähigkeit eingeschränkt, so dass sie nicht zu anderen Internet Nodes kommunizieren können, wenn sie den vollen Protokoll Stack mit HTTP und TLS verwenden. Jedoch sind sie in der Lage, speziell entworfen Protokoll Stacks (bsp CoAP mit UDP) zu verwenden und dadurch können die Class 1 Geräte ohne Hilfe von Proxys an Unterhaltungen umfangreicheren teilnehmen.

5.3.4.2.3 Class 2

Die Class 2 Geräte sind weniger eingeschränkt und sie unterstützen denselben Protokoll Stack wie ein herkömmliches Notebook. Aber die Geräte können von leichtgewicht- und energieeffizienten Protokollen profitieren. Durch weniger Verbrauch von Netzwerkressourcen lässt es mehr Ressourcen für die Applikation übrig. Auf der Class 2 Geräteebene könnten durch die Nutzung von speziellen Protokoll Stacks die allgemeinen Kosten reduziert werden und die Interoperabilität gesteigert werden.

5.3.4.3 Beispiele von Geräte

- **Wifi Router in einem Bus**
  Das IoT-Gerät gehört zu Class 2, weil die modernen Router leistungsfähig sind. Jedes Mobile-Netz wie 4G ist hinter einem NAT, da es mehr Smartphones gibt als öffentliche IPv4 Adressen. Der Bus ist ständig in Bewegung. Das heisst, es kann zu Verbindungsunterbrüchen kommen. Deshalb hat das IoT-Gerät die Merkmale Behind-NAT und Moving.

- **Temperatur Sensor am Dach**
  Das IoT-Gerät wird der Class 0 zugeordnet, da es sehr stark in der Rechenleistung eingeschränkt ist, um die Batterielaufzeit zu steigern. Weil das IoT-Gerät draussen angemacht wird, ist es batteriebetrieben und via WiFi erreichbar. Jeder Haushalt ist hinter einem NAT. Deshalb hat das IoT-Gerät die Merkmale Behind-NAT, Battery-Powered und Wireless.

5.4 Anforderungsspezifikation

5.4.1 Einführung

5.4.1.1 Übersicht

Im Abschnitt Allgemeine Beschreibung sind unter anderem Annahmen und Einschränkungen des Projektes beschrieben sowie die Zielgruppe für welche die Applikation gedacht ist. In den weiteren Abschnitte folgen die funktionalen Anforderungen sowie die nicht-funktionalen Anforderungen.

Das Use Case Diagramm im selbigen Abschnitt bietet eine komplette Übersicht für die funktionalen Anforderungen an das System. Die im Diagramm enthaltenen Use Cases sind auch schriftlich detailliert beschrieben.

5.4.2 Allgemeine Beschreibung

5.4.2.1 Produkt Perspektive

Die IoT Management Applikation ist eine Webanwendung, die es ermöglicht, mehrere tausend IoT-Geräte abzufragen, zu inventarisieren und generell zu verwalten. Ähnliche Produkte welche schon vergleichbare Funktionen anbieten, sind meist nur sehr eingeschränkt, d. h. nur auf ganz bestimmte IoT-Geräte oder Geräte von bestimmten Herstellern beschränkt. Diese Applikation soll offen gestaltet sein, sodass jedes IoT-Gerät einfach und unkompliziert eingebunden werden kann.

5.4.2.2 Produkt Funktion

Eine Low-Level Beschreibung des Funktionsumfangs der Applikation kann im Projektplan gefunden werden. Die Funktionalität wird durch die Anforderungen genauer beschrieben.

5.4.2.3 Benutzer Charakteristik

Zur der Zielgruppe gehören primär Firmen, welche mehrere tausende IoT-Geräte verwalten müssen. Diese Firmen arbeiten auch mit unterschiedlichen IoT-Geräten von unterschiedlichen Herstellern.

5.4.2.4 Einschränkungen dieser Arbeit

Die IoT Management Applikation ist eine Webplattform und setzt dementsprechend eine Internetverbindung voraus. Die Plattform läuft auf einer Serverinfrastruktur der Hochschule Rapperswil, ist aber so ausgelegt, dass Sie auf Servern von jedem Cloud-Anbieter laufen kann. Die Webplattform wird primär für aktuelle Browser und Browser-Versionen optimiert. Weitere Einschränkungen funktioneller sowie nicht-funktioneller Art sind in diesem Dokument zu finden.

Das gesamte Projekt beschränkt sich primär darauf, dass IoT-Geräte auf dem Managementserver eröffnet werden können und einer Domain/Group zugeordnet werden kön-

nen. Des Weiteren soll für die verschiedenen Geräte eine Konfiguration erstellt werden können. Das konkrete und erfolgreiche Einspielen auf den IoT-Geräten ist allerdings nicht mehr Bestandteil dieser Arbeit. Zum Projekt gehört aber noch die Möglichkeit, dass sich die IoT-Geräte in regelmässigen Abständen beim Managementserver melden und allfällige Tasks entgegennehmen. Eine Multiuser-Benutzung wird in dieser Arbeit nicht angestrebt. Generell konzentriert sich die Arbeit auf das Problem der IoT-Geräte Verwaltung.

5.4.2.5 Annahmen

Es wird davon ausgegangen, dass die IoT-Geräte mit einer CoAP Schnittstelle bestückt sind und entsprechend auf die jeweilig vorhandenen Sensoren programmiert sind. Um die Echtheit bzw. die richtige Zugehörigkeit zu einer Domäne zu gewähren, werden Seriennummern und Public-Keys eingesetzt. Für die Änderung an der Konfiguration wird angenommen, dass wenn man ein Konfigurationsfile sendet, dieses dann auf dem Gerät entsprechend richtig umgesetzt wird. Für die restlichen Teile des Projekts wurden keine weiteren Annahmen getroffen.

5.4.3 Use Cases

In diesem Abschnitt werden die funktionalen Anforderungen detailliert aufgeführt.

5.4.3.1 Aktoren & Stakeholder

Aktoren	Beschreibung und Ziele
Administrator	Ein Administrator verwaltet die Geräte.
IoT-Geräte	IoT-Geräte interagiert mit Server.

5.4.3.2 Architektur

Wegen der NAT (Network Address Translation) Problematik im Hinblick auf die Verbindung zu einem IoT-Gerät, welches sich in einem anderen Netzwerk als die IoT Management Applikation befindet, geht die Verbindung immer vom IoT-Gerät zum Server aus.

5.4.3.3 Use Case Diagramm

Abbildung 4 Use Case Diagramm

5.4.4 Beschreibungen (Brief)

5.4.4.1 UC-1 Ressource auslesen

Das IoT-Gerät sendet in regelmässigen Abständen die Werte eines Sensors. Der Administrator kann den zuletzt bekannten Wert sehen.

5.4.4.2 UC-2 keep-alive senden

Das IoT-Gerät sendet in regelmässigen Abständen keep-alive Signale, um dem Server zu melden, dass es noch online ist.

5.4.4.3 UC-3 Gerät offline

Der Administrator wird informiert, sobald das Gerät offline ist.

5.4.4.4 UC-4 Konfigurationstemplate erstellen

Der Administrator erstellt für ein bestimmtes Modell ein Konfigurationstemplate. Dieses Template wird mit Variablen versehen, so dass es für mehrere hundert IoT-Geräte mit dem gleichen Modell spezifiziert werden kann.

5.4.4.5 UC-5 Konfiguration spezifizieren

Der Administrator wählt die gewünschten IoT-Geräte aus. Die ausgewählten IoT-Geräte müssen alle vom gleichen Modell sein. Im nächsten Schritt wird das entsprechende Template ausgewählt und im letzten Schritt wird ein gewisser Bereich für z.B. IPs zugewiesen. Aus diesem Bereich spezifiziert der Server die konkrete IP für die Konfiguration des einzelnen IoT-Gerätes.

5.4.4.6 UC-6 Konfiguration einspielen

Der Administrator hinterlegt den Task “neue Konfiguration einspielen” auf dem Server. Das IoT-Gerät fragt in regelmässigen Abständen den Server ab. Sobald für das anfragende IoT-Gerät Tasks verfügbar sind, sendet der Server diese Tasks zum IoT-Gerät.

5.4.4.7 UC-7 Firmware einspielen

Der Administrator hinterlegt den Task “neue Firmware einspielen” auf dem Server. Das IoT-Gerät fragt in regelmässigen Abständen den Server ab. Sobald für das anfragende IoT-Gerät Tasks verfügbar sind, sendet der Server diese Tasks zum IoT-Gerät.

5.4.4.8 UC-8 Ressource eintragen

Das IoT-Gerät trägt seine Ressourcen beim Server ein.

5.4.4.9 UC-9 Domain/Group verwalten (CRUD)

Der Administrator verwaltet Domains und Groups. Als Domäne wird z.B. ein Kunde bezeichnet und als Group wird ein Ort oder eine Funktion bezeichnet. Prinzipiell kann der Administrator frei wählen wie er die Domain/Group bezeichnen will. Das Prinzip dient hauptsächlich zur hierarchischen Unterteilung innerhalb der IoT Management Applikation. Diese hierarchische Unterteilung kann beliebig tief sein und ermöglicht eine logische Einordung der Ressourcen.

5.4.4.10 UC-10 Modell verwalten (CRUD)

Der Administrator verwaltet Modelle. Jedes IoT-Gerät besitzt nur ein Modell. So kann sichergestellt werden, dass über alle Domänen mit Geräten mit diesem Modell z.B. Updates gemacht werde können. So ist man nicht an die Domain limitiert.

5.4.4.11 UC-11 Lieferschein importieren

Der Administrator importiert den Lieferschein, damit werden die IoT-Geräte in der Verwaltungssoftware erstellt. Bei diesem Vorgang wird auch gleich der Public Key zu jedem IoT-Gerät importiert. Die eröffneten Geräte gelangen zunächst in eine Domain “Im Lager”, welche der CloudGuard AG zugeordnet ist. Von dort aus können die Geräte den entsprechend richtigen Groups zugeordnet werden.

5.4.4.12 UC-12 IoT-Geräte an Domain/Group/Modell zuordnen

Nach dem erfolgreichen Import des Lieferscheins, sind die IoT-Geräte in der Domain “Im Lager” der CloudGuard zugeordnet. Der Administrator von CloudGuard AG kann nun von dort aus die Geräte den richtigen Domains zuordnen. Sind die Geräte der richtigen Domain zugeordnet, kann der Administrator, welche dann für diese Domain zuständig ist, die Geräte noch entsprechend der richtigen Group und dem richtigen Modell zuordnen.

5.4.5 Beschreibungen (Fully-dressed)

5.4.5.1 UC-1 Ressource auslesen

Goal	Der Administrator liest den zuletzt bekannten Wert aus
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Ihm ist es möglich, den zuletzt gelieferten Wert der einzelnen Sensoren anzusehen
Preconditions	Das IoT-Gerät ist online
Postconditions	Der Administrator weiss den zuletzt bekannten Wert des Sensors
Main Success Scenario	1. Alle Ressourcen senden in einem bestimmten Intervall ihre Werte an den Server. 2. Der Server trägt sich den Wert ein. 3. Der Administrator wählt ein Gerät bzw. eine Ressource aus. 4. Ihm wird der zuletzt bekannte Wert angezeigt.
Alternativer Flow	keiner
Technology and Data variation List	Die Werte können unterschiedliche Bedeutungen haben, je nach Sensor. Daher gibt es hier eine grosse Variation von Daten, welche beachtet werden muss.
Frequency of Occurrence	manchmal
Open Issues	keine

5.4.5.2 UC-2 keep-alive senden

Goal	Das IoT-Gerät sendet keep-alives an den Server
Level	User Goal
Primary Actor	IoT-Gerät
Trigger	Jederzeit
Stakeholders and Interests	**IoT-Gerät**: Es teilt dem Server mit, dass es noch online ist
Preconditions	Das IoT-Gerät ist in der Verwaltungssoftware eingetragen
Postconditions	Das IoT-Gerät hat ein keep-alive Signal abgesetzt
Main Success Scenario	1. Das IoT-Gerät sendet ein keep-alive Signal 2. Der Server erhält das keep-alive Signal. 3. Der Server stellt den Timer zurück 4. Schritte 1-3 wiederholen sich in einem bestimmten Intervall.
Alternativer Flow	1. Das IoT-Gerät kann kein keep-alive Signal senden, weil es in einem Tunnel ist. 2. Der Server erhält das keep-alive Signal nicht 3. Der Timer für das IoT-Gerät läuft aus 4. Use Case 3 wird ausgeführt.
Technology and Data variation List	keine
Frequency of Occurrence	häufig
Open Issues	keine

5.4.5.3 UC-3 Gerät offline

Goal	Der Administrator wird informiert, wenn ein IoT-Gerät offline ist
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Er wird informiert, dass das Gerät offline ist und kann entsprechende Massnahmen einleiten
Preconditions	Der UC2 ist im alternativen Ablauf abgelaufen
Postconditions	Der Administrator wurde informiert.
Main Success Scenario	1. Der Administrator wird mittels einem Status informiert, dass ein Gerät offline ist.
Alternativer Flow	Der Administrator wird per SMS oder EMail informiert
Technology and Data variation List	Ein SMTP oder SIM Karte müsse beim Alternativen Ansatz verfügbar sein
Frequency of Occurrence	niedrig
Open Issues	keine

5.4.5.4 UC-4 Konfigurationstemplate erstellen

Goal	Der Administrator erstellt ein Konfigurationstemplate
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Er erstellt pro Modell ein Konfigurations-template, um später alle IoT-Geräte mit diesem Modell mit einer Konfiguration auszustatten.
Preconditions	Das gewünschte Modell existiert
Postconditions	Das Konfigurationstemplate wurde erstellt
Main Success Scenario	1. Der Administrator erstellt das Konfigurations-template
Alternativer Flow	keiner
Technology and Data vari-ation List	keine
Frequency of Occurrence	niedrig
Open Issues	keine

5.4.5.5 UC-5 Konfiguration spezifizieren

Goal	Der Administrator erstellt für IoT-Geräte eine Konfiguration.
Level	User Goal
Primary Actor	Administrator
Trigger	bei Konfigurationsänderungen
Stakeholders and Interests	**Administrator**: Er will Änderungen der Konfiguration für mehrere IoT-Geräte gleichzeitig vornehmen.
Preconditions	Die IoT-Geräte sind auf dem Server eingetragen. Es gibt für die ausgewählten IoT-Geräte ein Konfigurationstemplate.
Postconditions	Das Konfigurationsfile ist erstellt und ein Task für die entsprechenden IoT-Geräte ist angelegt
Main Success Scenario	1. Der Administrator wählt die gewünschten IoT-Geräte aus, auf der eine Änderung der Konfiguration vorgenommen werden soll. Es dürfen nur IoT-Geräte vom gleichen Modell ausgewählt werden. 2. Der Server zeigt mögliches Template an. 3. Der Administrator gibt einen Bereich für die Variablen z.B. für IP-Adressen an. 4. Die Verwaltungssoftware erstellt für die ausgewählten Geräte die Konfigurationsdatei. 5. Der Server hinterlegt für jedes der ausgewählten IoT-Geräte einen entsprechenden Task.
Alternativer Flow	keinen
Technology and Data variation List	Bei falschen Eingaben muss eine entsprechende Meldung erscheinen.
Frequency of Occurrence	Bei jeder Änderung der Konfiguration
Open Issues	keine

5.4.5.6 UC-6 Konfiguration einspielen

Goal	Der Administrator gibt die neueste Konfiguration für IoT-Gerät frei.
Level	User Goal
Primary Actor	Administrator
Trigger	Konfigurationspezifizierung
Stakeholders and Interests	**Administrator**: Er will Änderungen der Konfiguration für alle gewünschten IoT-Geräte vornehmen.
Preconditions	Die IoT-Geräte sind auf dem Server eingetragen. Es gibt eine spezifizierte Konfiguration für das ausgewählte IoT-Gerät.
Postconditions	Das IoT-Gerät hat die neue Konfiguration eingespielt.
Main Success Scenario	1. Der Administrator hinterlegt den neuen Task “Konfiguration einspielen” auf dem Server. 2. Das IoT-Gerät sendet in regelmässigen Abständen keep-alive Signale. Bei dieser Gelegenheit schaut der Server nach, ob es Tasks für dieses IoT-Gerät zum Abarbeiten hat. 3. Falls ja, liefert der Server die Konfigurationsdatei mit dem Request “Konfiguration einspielen”. 4. Das IoT-Gerät antwortet mit einer Acknowledge Response. 5. Das IoT-Gerät spielt sich die Konfiguration ein.
Alternativer Flow	Sollte es keine offenen Tasks für das IoT-Gerät haben, wird bei der nächsten Gelegenheit nochmals nachgefragt. Es passiert also nichts
Technology and Data variation List	Dieser Mechanismus wird durch das CoAP Protokoll unterstützt
Frequency of Occurrence	manchmal
Open Issues	keine

5.4.5.7 UC-7 Firmware einspielen

Goal	Der Administrator gibt die neueste Firmware frei
Level	User Goal
Primary Actor	IoT-Gerät
Trigger	Neue Firmware verfügbar
Stakeholders and Interests	**Administrator**: Er will ein Upgrade der Firmware für alle entsprechenden IoT-Geräte vornehmen.
Preconditions	Die IoT-Geräte sind auf dem Server eingetragen. Neue Firmware steht zur Verfügung.
Postconditions	Die IoT-Geräte haben das Upgrade eingespielt.
Main Success Scenario	1. Der Administrator hinterlegt neuen Task “Firmware einspielen” auf dem Server. 2. Das IoT-Gerät sendet in regelmässigen Abständen keep-alive Signale. Bei dieser Gelegenheit schaut der Server nach, ob es Tasks für dieses IoT-Gerät zum Abarbeiten hat. 3. Falls ja, liefert der Server die neueste Firmware mit dem Request “Firmware einspielen” 4. Das IoT-Gerät antwortet mit einer Acknowledge Response . 5. Das IoT-Gerät spielt sich die Firmware ein.
Alternativer Flow	Sollte es keine offenen Tasks für das IoT-Gerät haben, wird bei der nächsten Gelegenheit nochmals nachgefragt. Es passiert also nichts
Technology and Data variation List	Dieser Mechanismus wird durch das CoAP Protokoll unterstützt
Frequency of Occurrence	manchmal
Open Issues	keine

5.4.5.8 UC-8 Ressource eintragen

Goal	Das IoT-Gerät trägt seine Ressourcen beim Server ein
Level	User Goal
Primary Actor	IoT-Gerät
Trigger	Erster Kontakt mit Server
Stakeholders and Interests	**Administrator**: Er kann später auf den Sensor zugreifen Er möchte alle Ressource von IoT-Geräten kennen.
Preconditions	Die Domain/Group ist vorhanden.
Postconditions	IoT-Gerät ist bei der richtigen Domain und bei der richtigen Group eingeordnet
Main Success Scenario	1. Das IoT-Gerät registriert seine Ressourcen beim Server unter dem entsprechend zuvor eröffneten Gerät (UC11). Dieses kann mittels der Seriennummer ermittelt werden.
Alternativer Flow	Sollte das IoT-Gerät noch nicht eröffnet sein, wird versucht, die Ressourcen zu einem späteren Zeitpunkt nochmals einzutragen.
Technology and Data variation List	Die erforderlichen Angaben können via CoAP Protokoll übermittelt werden.
Frequency of Occurrence	Bei jedem erstmaligen Anmelden
Open Issues	keine

5.4.5.9 UC-9 Domain/Group verwalten

Goal	Der Administrator verwaltet seine Domains/Groups
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Er kann mittels Domains/Groups eine hierarchische, für ihn logische Ordnung anlegen
Preconditions	keine
Postconditions	Die neu erzeugte Domain/Group wird in der Hierarchie am richtigen Ort eingegliedert
Main Success Scenario	1. Der Administrator erzeugt eine Group. 2. Der Administrator erzeugt eine weitere Group. Er kann sich nun entscheiden zwischen: a. setze ich die Group eine Hierarchiestufe unter eine andere ein. b. setzte ich die Group auf derselben Hierarchiestufe, wie eine andere Group ein. 3. Schritt 2 wiederholt sich bis man genügend Groups hat.
Alternativer Flow	keiner
Technology and Data variation List	Wird mit einer Liste auf dem Domain bzw. Group Objekt realisiert
Frequency of Occurrence	manchmal
Open Issues	keine

5.4.5.10 UC-10 Modell verwalten

Goal	Der Administrator verwaltet seine Modells
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Er ist nicht an die hierarchische Ordnung gebunden. Er kann seine IoT-Geräte mit Modells versehen und hat somit eine hierarchieunabhängige Klassifizierung.
Preconditions	keine
Postconditions	Das neu erzeugte Modell kann für eine hierarchieunabhängige Klassifizierung verwendet werden.
Main Success Scenario	1. Der Administrator erstellt ein Modell. 2. Der Administrator weist jedem IoT-Gerät ein gewünschtes Modell zu.
Alternativer Flow	keiner
Technology and Data variation List	keine
Frequency of Occurrence	manchmal
Open Issues	keine

5.4.5.11 UC-11 Import Lieferschein

Goal	Der Administrator importiert den Lieferschein
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Indem er die Möglichkeit hat, den Lieferschein zu importieren und anhand der Seriennummer und dem Public Keys ein IoT-Gerät von der Verwaltungssoftware eröffnen zu lassen, kann auf eine manuell Eröffnung verzichtet werden.
Preconditions	Lieferschein wurde zugesendet, die Domain “Im Lager” existiert
Postconditions	Die IoT-Geräte wurden in der Verwaltungssoftware eröffnet und sind in der Domain “Im Lager” aufgelistet
Main Success Scenario	1. Der Administrator importiert den Lieferschein in der Verwaltungssoftware. 2. Anhand der Seriennummer und dem Public Keys wird ein IoT-Gerät erstellt. 3. Das IoT-Gerät wird der Domain “Im Lager” zugeordnet. 4. Schritt 2 und 3 wird wiederholt bis es keine IoT-Geräte mehr zu eröffnen gibt.
Alternativer Flow	keiner
Technology and Data variation List	Der Lieferschein wird als Excel Datei (CSV) zugestellt.
Frequency of Occurrence	Bei Erhalt von Lieferscheinen
Open Issues	keine

5.4.5.12 UC-12 Ordnet IoT-Geräte Domain / Group / Modell zu

Goal	Der Administrator ordnet ein IoT-Gerät einer Domain, Group und/oder Modell zu
Level	User Goal
Primary Actor	Administrator
Trigger	Jederzeit
Stakeholders and Interests	**Administrator**: Damit die IoT-Geräte hierarchisch eingeordnet werden können, ordnet der Administrator die IoT-Geräte entsprechend ein.
Preconditions	Die gewünschten Domains, Groups und Modelle existieren
Postconditions	Die IoT-Geräte wurden den gewünschten Domains, Groups oder Modelle zugeordnet.
Main Success Scenario	1. Der Administrator (CloudGuard) ordnet ein IoT-Gerät der richtigen Domain z.B. PostAuto Schweiz AG zu. 2. Der Administrator dieser Domain (PostAuto Schweiz AG) ordnet das Gerät einer gewünschten Gruppe seiner Domain zu. 3. Anschliessend weist er dem IoT-Gerät auch noch ein Modell zu.
Alternativer Flow	keiner
Technology and Data variation List	keine
Frequency of Occurrence	Bei Erhalt von Lieferscheinen
Open Issues	keine

5.4.6 Weitere Anforderungen

5.4.6.1 Skalierbarkeit

Eine wichtige Anforderung im nicht-funktionalen Bereich stellt die Skalierbarkeit dar. Es muss möglich sein, die Software bei entsprechend hoher Belastung nach oben zu skalieren.

5.4.6.2 IoT-Geräteunterstützung

Die Software muss in der Lage sein, IoT-Geräte von verschiedensten Herstellern oder Marken zu managen. Egal wie das IoT-Gerät ausgestattet ist (mit welchen Sensoren), die Verwaltungssoftware ist für alle Geräte und entsprechend deren Sensoren geeignet und unterstützt diese in all deren Funktionen.

5.4.7 Qualitätsmerkmale

Die nachfolgenden Qualitätsmerkmale richten sich nach der Checkliste des Standards ISO 9126 [8]. Auf einzelne Punkte des Standards wurde verzichtet, da sie keine grössere Bedeutung für das vorliegende Projekt haben.

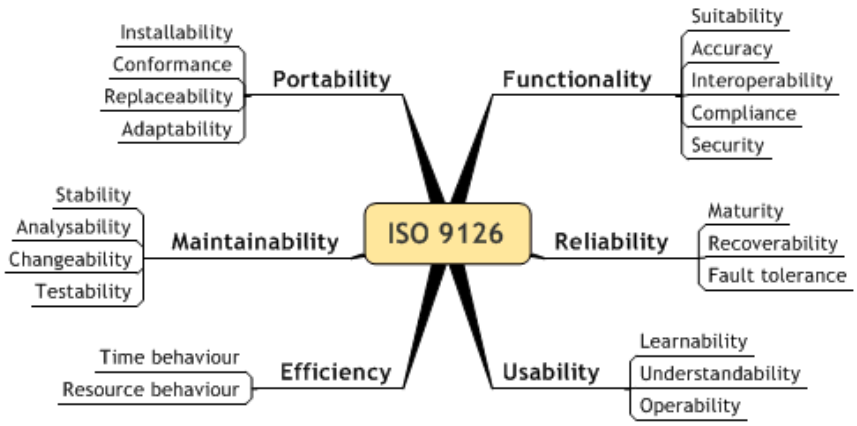


Abbildung 5 Standard ISO 9126 [8]

5.4.7.1 Funktionalität (functionality)

5.4.7.2 Angemessenheit (suitability)

Änderungen des Funktionsumfangs sind mit dem gesamten Team abzuklären bzw. zu diskutieren und zu genehmigen. Abweichungen des Funktionsumfangs des IoT Management Systems, welche nicht in der Anforderungsspezifikation definiert wurden, dürfen maximal 5% der gesamten Entwicklungszeit beanspruchen.

5.4.7.3 Interoperabilität (interoperability)

Die eigentliche IoT Management Software besteht aus einer REST-API. So ist es auch in Zukunft möglich eine andere Front-End Technologie einzusetzen oder die REST-Schnittstelle andern zur Verfügung zu stellen.

5.4.7.4 Sicherheit (security)

Die Kommunikation zwischen den IoT-Geräten und der REST-Schnittstelle soll via DTLS [9] gesichert sein.

5.4.7.5 Fehlertoleranz (fault tolerance)

Fehleingaben müssen in allen Formularen vorgängig validiert werden und bei einer Fehleingabe wird dem User eine verständliche Fehlerausgabe präsentiert. Diese Notwendigkeit ist vor allem für die Erstellung von Konfigurationsfiles von Bedeutung.

5.4.7.6 Benutzbarkeit (usability)

5.4.7.7 Verständlichkeit (understandability)

Die IoT Management Suite muss so konzipiert und aufgebaut sein, dass für den User eine einfache und intuitive Benutzung möglich ist. Der Aufwand für die Einarbeitung soll aber nicht länger als einer halben Stunde überschreiten.

5.4.7.8 Erlernbarkeit (learanability)

Der Einstieg für neue User soll durch das Konzept der intuitiven Bedienung erleichtert werden.

5.4.7.9 Bedienbarkeit (operability)

Mit genügend starken Kontrasten wird sichergestellt, dass Überschriften und Text sich gut voneinander unterscheiden und man die Texte auch noch mit einer Entfernung von einem Meter lesen kann. Das gesamte Frontend muss zudem auch über Touchscreens und/oder Tastatur bedient werden können.

5.5 Konzept

5.5.1 Einführung

5.5.1.1 Übersicht

Im Abschnitt „Beschreibung des Ablaufs“ wird das Konzept als Ablaufdiagramm dargestellt. Es stellt einen systematischen und chronologischen Ablauf von Anfang bis Schluss dar. Zudem wird auch textuell beschrieben was genau Schritt für Schritt abläuft, wer mit wem interagiert und was alles wann passiert.

Im Abschnitt Erweiterungen wird speziell nochmals auf das Netzwerk eingegangen. Es werden einzelne Vorschläge für eine Erweiterung des angedachten Konzeptes in Bezug auf das Netzwerk vorgestellt und warum diese von Vorteil sein könnten. Auch wird eine Erweiterung angesprochen, welche beim Erfassen der IoT-Geräte in die Management Software zum Zuge kommen könnte (Regelbasierte Zuordnung der IoT-Geräte). Alle diese Erweiterungen werden allerdings in dieser Arbeit nicht weiter verfolgt.

5.5.2 Ablaufdiagramm

Dieser Abschnitt zeigt das angestrebte Konzept bzw. den angestrebten Ablauf von Beginn, also vom Zeitpunkt bei dem sich eine Firma entscheidet IoT-Geräte für einen bestimmten Zweck zu verwenden, bis zum Zeitpunkt an dem diese dann vollumfänglich konfiguriert und einsatzbereit sind.

Das gesamte Ablaufdiagramm sowie der Beschreibungstext wurde für ein konkretes Beispiel der PostAuto Schweiz AG geschrieben. So ist sichergestellt, dass ein konkretes Anwendungsbeispiel vorliegt und man sich ein wenig einfacher den Ablauf vorstellen kann. Zur besseren Übersicht wurden aus dem gesamten Ablauf einzelne Teilabschnitte bzw. Diagramme gemacht.

5.5.2.1 Annahmen

Folgende Annahmen werden getroffen, um das Konzept nicht unnötig kompliziert zu machen. In den Use Cases, welche in der Anforderungsspezifikation beschrieben sind, wird allerdings auf diese Annahmen eingegangen und ein alternativer Ablauf aufgezeigt, sollten diese Annahmen nicht zutreffen. Dieser alternative Ablauf wurde hier wie geschrieben vernachlässigt.

Es wird ausserdem davon ausgegangen, dass wenn ein IoT-Gerät sich das erste Mal mit einem Netzwerk verbindet, es eine IP-Adresse vom lokalen DHCP Server bekommt. Nur so ist es überhaupt in der Lage über das Internetprotokoll zu kommunizieren.

5.5.3 Beschreibung des Ablaufs

Der gesamte Ablauf startet damit, dass sich die PostAuto Schweiz AG entscheidet Router in ihren Bussen einzusetzen. Sie wollen ihren Kunden damit ein frei zugängliches WLAN in den Bussen anbieten.

5.5.3.1 Bestellung

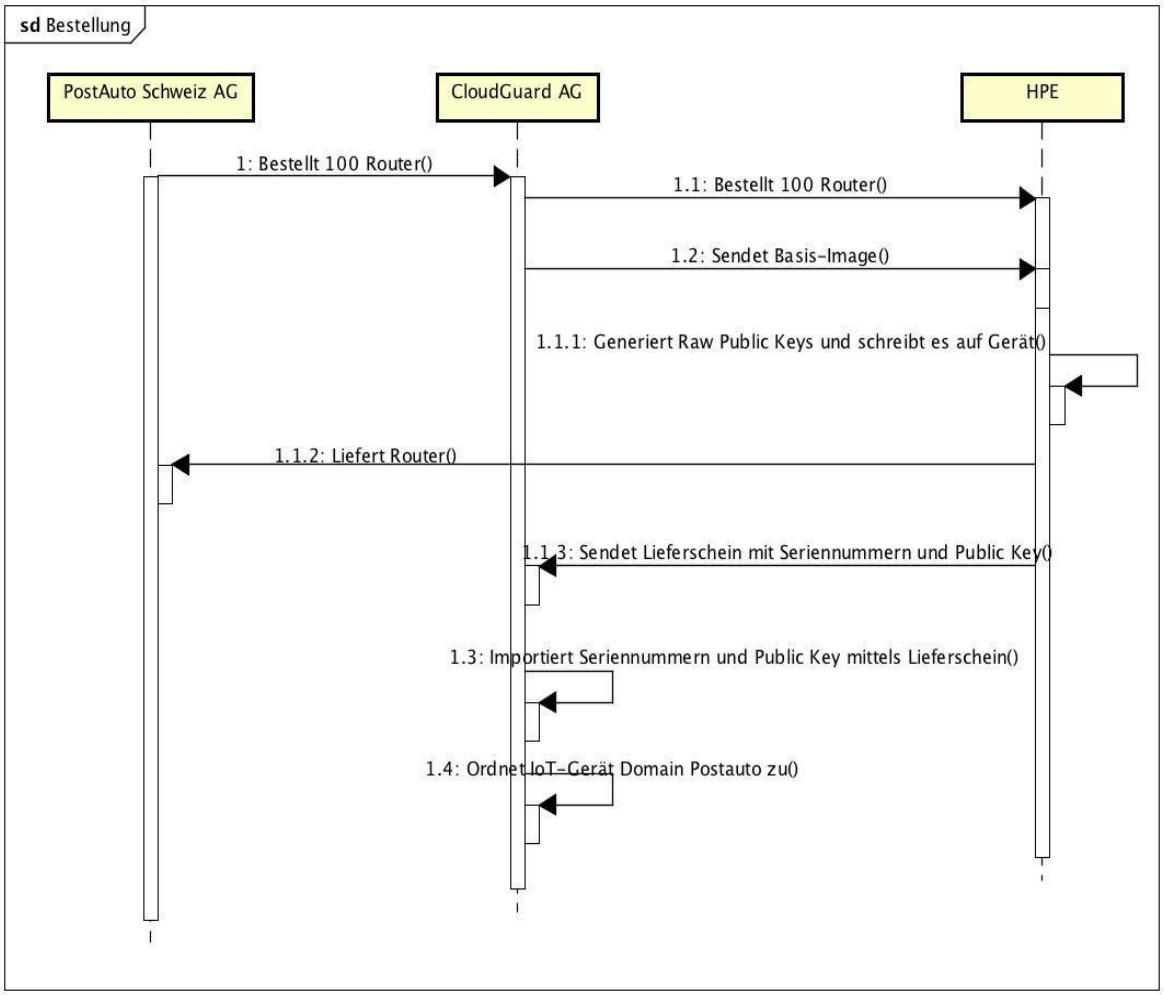


Abbildung 6 Ablauf Bestellung der IoT Geräte

- Sie kontaktieren also die CloudGuard AG und möchten, dass CloudGuard ihnen diese Router verwaltet.
- Da CloudGuard selbst keine Router herstellt, gibt sie einen Auftrag an z.B. HPE, welche ihnen die Router liefern soll.
- Gleichzeitig sendet CloudGuard ein Basis-Image mit der Adresse des Management Servers an HPE.
- Dieses Image wird nun auf allen bestellten Routern installiert.
- Als nächstes erstellt HPE für jedes Gerät ein Raw Public Key und lädt diese auf das Gerät. Diese Raw Public Key wird für die DTLS - Verbindung benötigt.
- Die hergestellten Geräte werden der PostAuto Schweiz AG zugesendet.
- Mit dem Lieferschein erhält die CloudGuard AG eine Liste mit Seriennummern der Geräte. Im Lieferschein enthalten sind zudem die Public Keys jedes einzelnen IoT-Gerätes.
- Die CloudGuard AG importiert die Seriennummern und die Public Keys in die Management Software. Daraus werden die IoT-Geräte in der Software erstellt und werden einer Domain (“An Lager”) der CloudGuard zugeordnet.
- Der Administrator bei der CloudGuard AG kann nun die neu erstellen IoT-Geräte der Domain PostAuto zuweisen.

5.5.3.2 Konfiguration

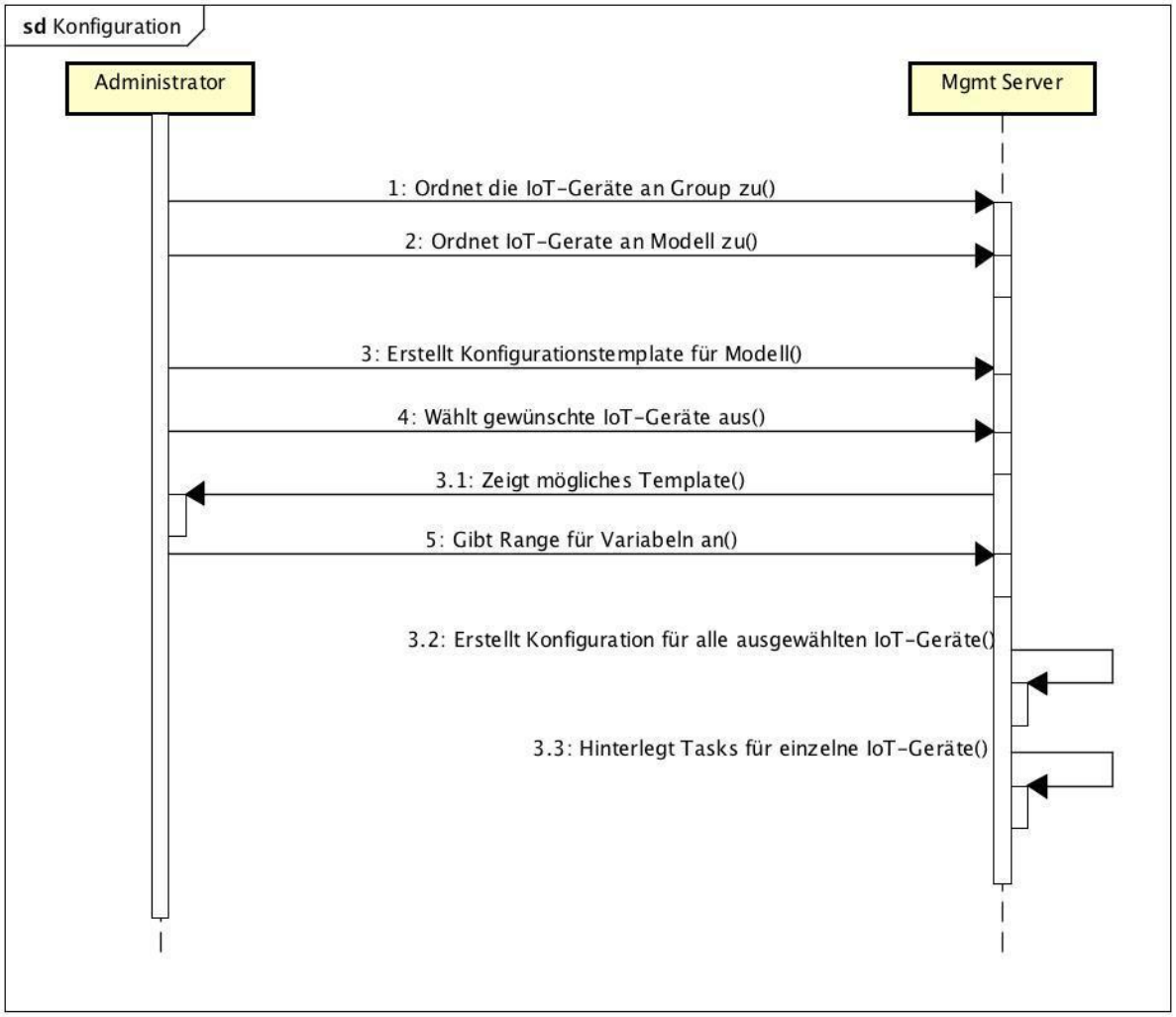


Abbildung 7 Ablauf Konfiguration der IoT Geräte

- Die PostAuto Schweiz AG möchte gerne anhand Region gruppieren, so würden die Groups Zürich, St. Gallen und Aargau bestehen. Entsprechend teilt der Administrator der PostAuto AG die Router den einzelnen Groups zu.
- Die Router werden anhand der erhaltenen Seriennummern einem bestimmten Modell zugewiesen. Also weist der Administrator diese entsprechend zu.
- Der Administrator erstellt ein Konfigurationstemplate.
- Der Administrator wählt alle gewünschten IoT-Geräte aus, welche er mit der gewünschten Konfiguration ausstatten will.
- Der Management Server zeigt dem Administrator alle möglichen Templates, welche für die IoT-Geräte zur Verfügung stehen.
- Der Administrator gibt daraufhin einen Bereich für die Variablen im Template ein.
- Der Management Server erstellt daraufhin alle Konfigurationen für die ausgewählten IoT-Geräte mit konkreten Angaben.
- Der Management Server hinterlegt den Task “Konfiguration einspielen” für jedes einzelne IoT-Gerät.

5.5.3.3 Inbetriebnahme

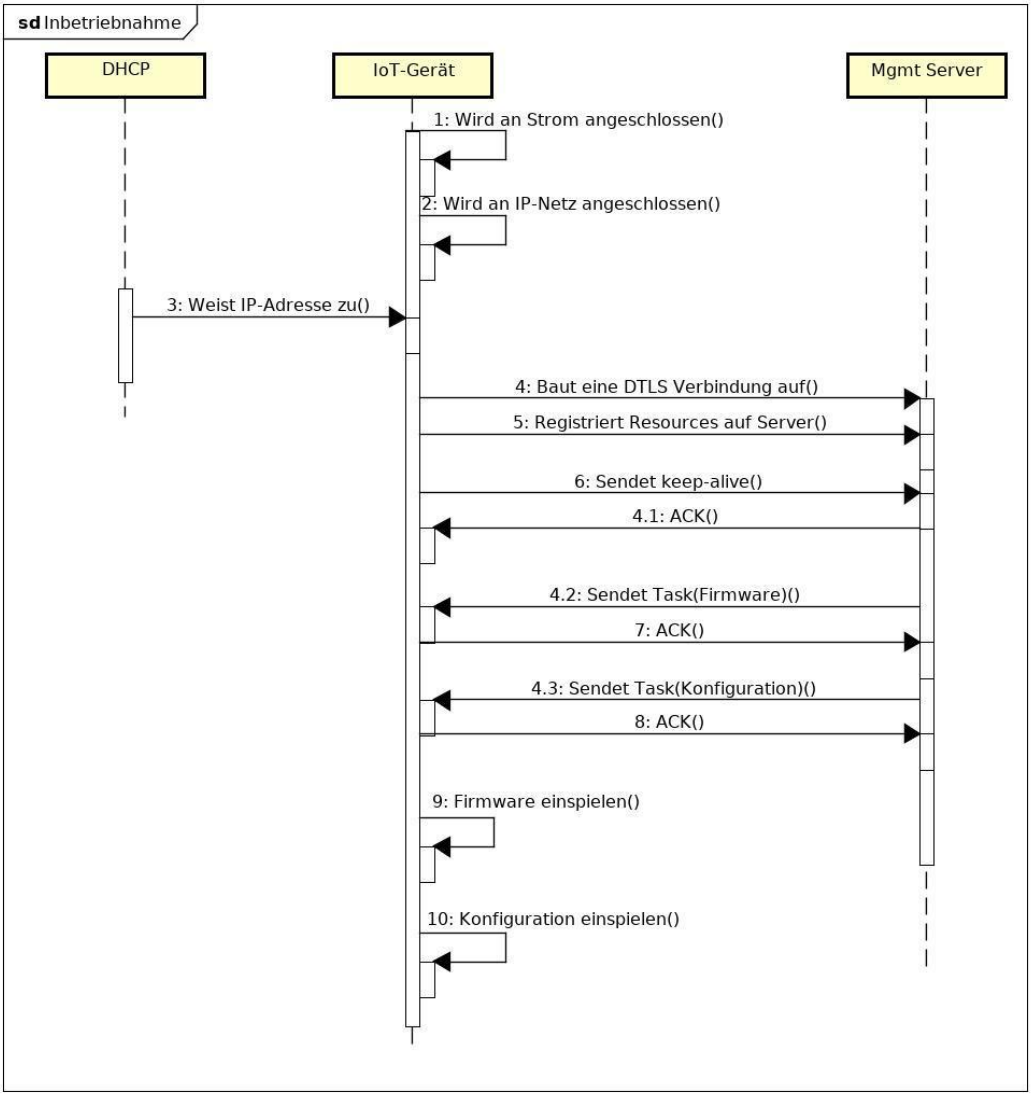


Abbildung 8 Ablauf Inbetriebnahme

- Die Router sind nun bei der PostAuto Schweiz AG angekommen und werden nun an den Strom angeschlossen.
- Als nächstes wird eine Verbindung zum lokalen Netzwerk hergestellt, indem die Router per LTE an das mobile Datennetz angeschlossen werden.
- Als nächstes wird eine DTLS-Verbindung zum Management Server aufgebaut.
- Im nächsten Schritt registriert das IoT-Gerät seine Ressourcen beim Management Server.
- Nach dem Registrieren sendet das IoT-Gerät in regelmässigen Abständen automatisch keep-alive Signale an den Management Server.
- Wenn der Administrator bei Konfigurationsprozess eine neue Firmware auf dem Management Server für das IoT-Gerät hinterlegt hat, dann sendet der Server jetzt den Task “Firmware einspielen”.
- Falls der Administrator bei Konfigurationsprozess eine konkrete Konfiguration für das IoT-Gerät auf dem Server hinterlegt hat, dann sendet der Server den Task “Konfiguration einspielen”.
- Die Firmware bzw. die Konfiguration wird daraufhin in das IoT-Gerät eingespielt.

5.5.3.4 Betrieb

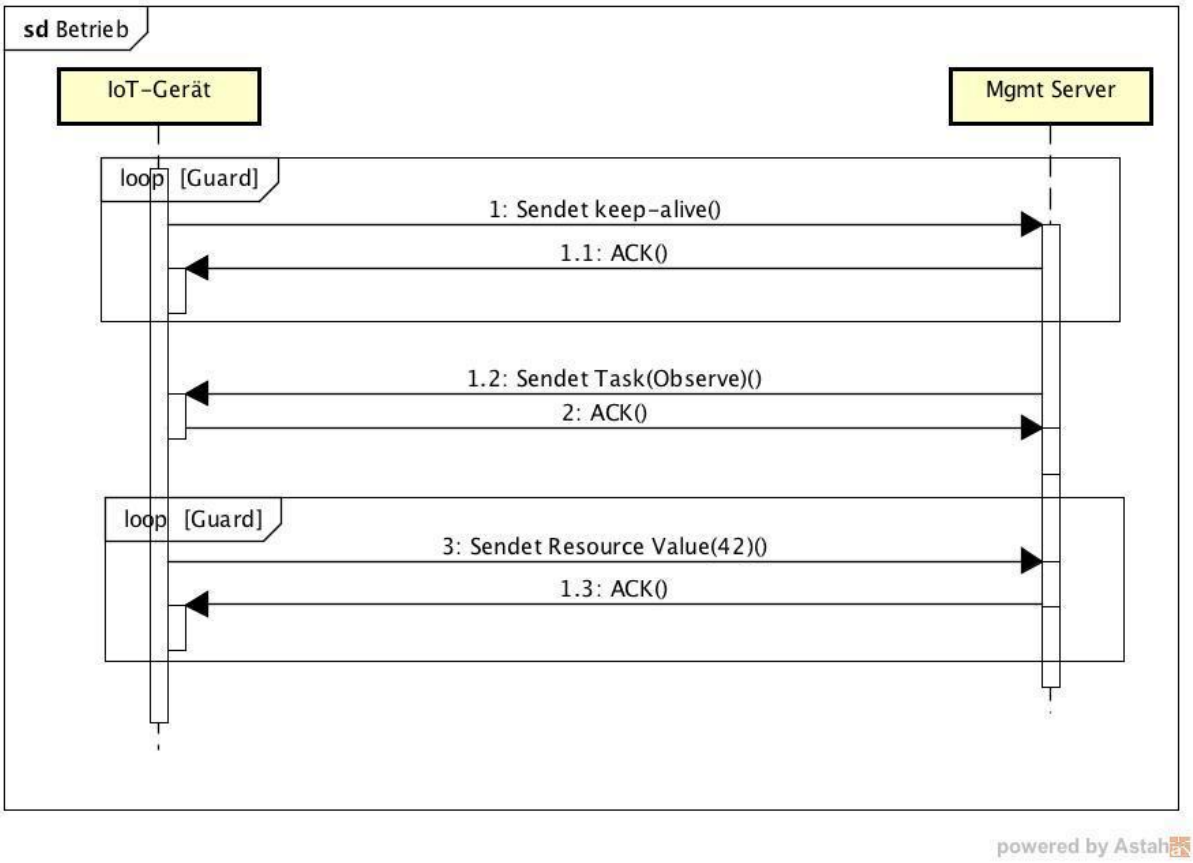


Abbildung 9 Ablauf Betrieb

- In regelmässigen Abständen liefert das IoT-Gerät nun Angaben, ob es noch erreichbar ist. Es sendet also Keep-alive Signale.
- Bei diesem Vorgang schaut der Server, ob er Tasks hat, welche das IoT-Gerät abarbeiten muss.
- Sollte dies der Fall sein, wird der Task “Observe Resource CPU-Auslastung” an das IoT-Gerät gesendet.
- Ab nun sendet das IoT-Gerät den aktuellen Value der Ressource in regelmässigen Abständen.

5.5.4 Lösungsansätze für Problematiken

5.5.4.1 NAT-Problematik

Um das NAT Problem zu lösen, müssen die Verbindungen vom IoT-Gerät ausgehend zum Managementserver aufgebaut werden. Nur so kann sichergestellt werden, dass der Server anschliessend auch auf diese Anfrage antworten kann.

5.5.4.2 Konfigurationsstruktur

Die Problematik bei einer Vielzahl von verschiedenen IoT-Geräten ist, dass jeder Hersteller seine eigene, leicht unterschiedliche Struktur der Konfiguration hat. Damit nun aber z.B. über ein Web Frontend eine Konfiguration für alle (verschiedene) Geräte gemacht werden kann, wird hier eine unabhängige Struktur benötigt.

Um das Problem der herstellerabhängigen Konfigurationsstruktur zu lösen, wird also eine herstellerunabhängige Konfigurationsstruktur eingeführt. In Template wird zunächst nur festgelegt, welcher Datentyp für ein gewisses Attribut angegeben werden kann. Danach kann dieses Template bzw. die Attribute des Templates mit einem Range z.B. für IPs ausgestattet werden. Zu Ende wird mittels Mapping ein konkreter Wert in die herstellerabhängige Konfigurationsstruktur eingesetzt. Am Schluss erhält man eine Konfigurationsdatei welche der Herstellerstruktur entspricht.

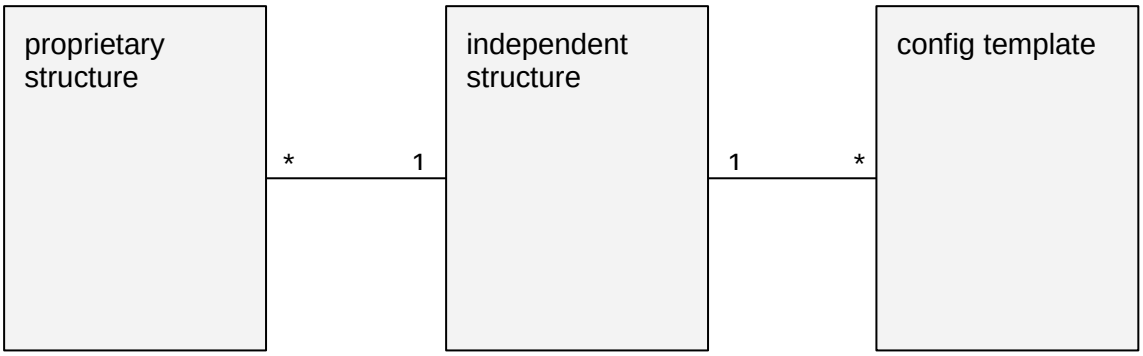


Abbildung 10 Konfigurationsstruktur

5.5.4.3 Hierarchie

Die Hierarchie ist unabhängig von ortsabhängigen oder funktionalen Einträgen. Das heisst es ist völlig dem Administrator der Software überlassen, wie er die hierarchische Einteilung vornimmt. Zudem ist die Hierarchie beliebig tief verschachtelbar. Es ist z.B. möglich in einer ersten Einteilung nach Orten (Gebäude 1, Gebäude 2, etc.) zu unterscheiden und in einer zweiten Einteilung (innerhalb der ersten Einteilung) nach Funktionen zu unterscheiden.

Bei den Kunden, welche die Software verwenden, spricht man von Domains während man für die darunterliegenden Hierarchiestufen von Groups spricht.

Folgende Hierarchien sind möglich:

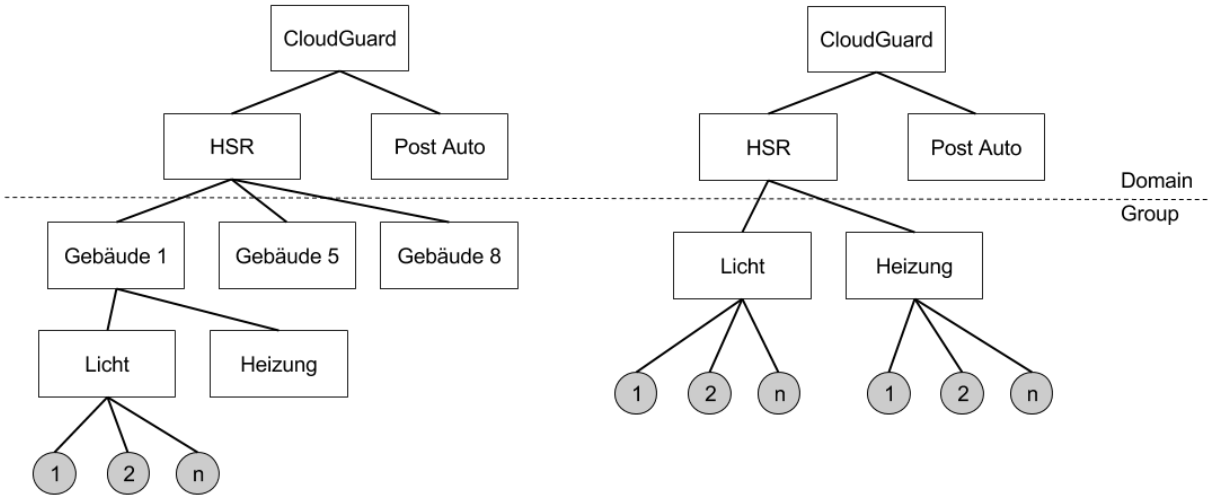


Abbildung 11 Hierarchie

5.5.4.4 Modell

Damit eine von der Hierarchie unabhängige Unterscheidung gemacht werden kann, werden Modelle eingesetzt. Unter einem Modell versteht man im Prinzip die Einteilung zwischen verschiedenen Gerätetypen.

5.5.4.5 Domain Model

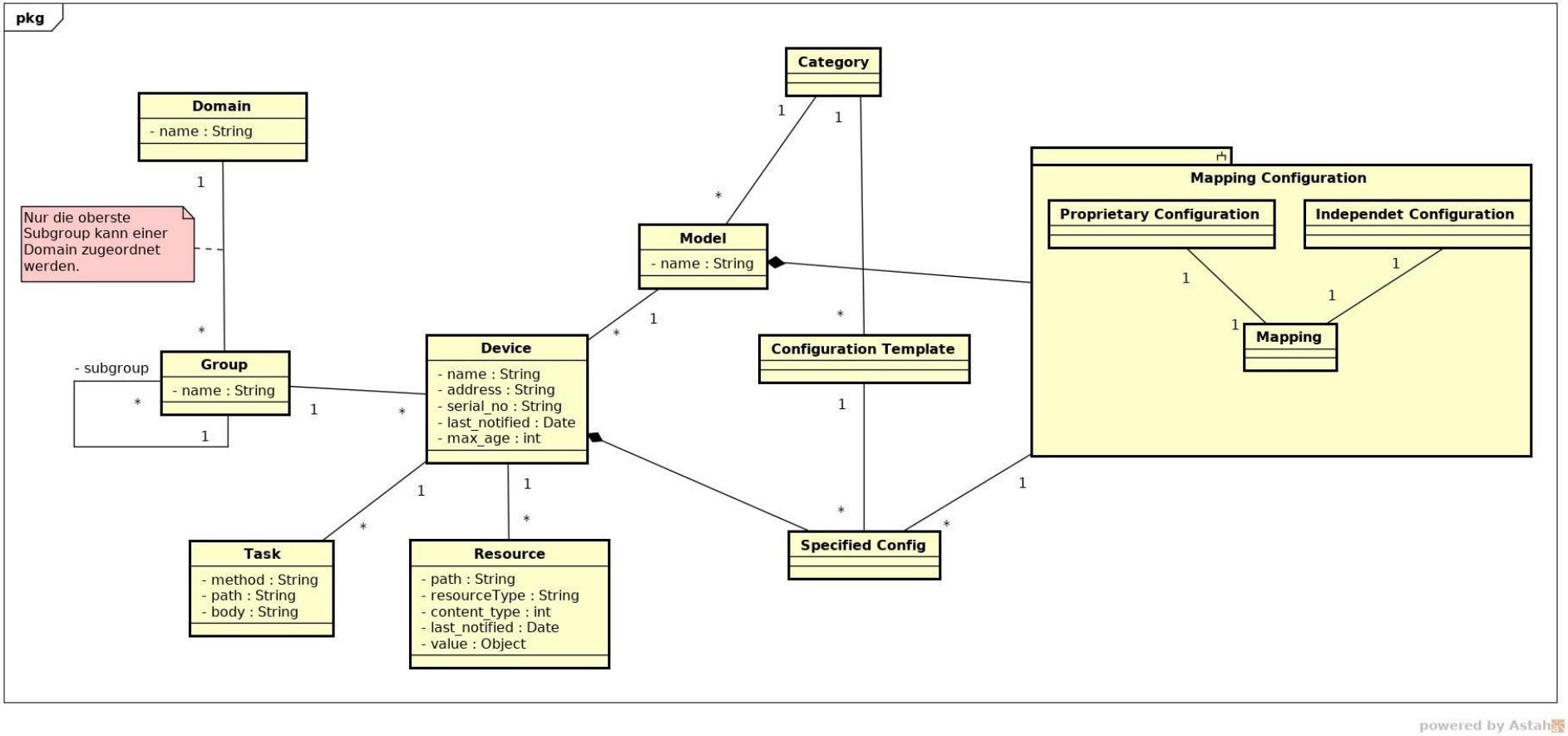


Abbildung 12 Domain Model

Der Kern des Domain Modells stellt das Device und dessen Resources dar. Ein Device ist zudem einer Group zugeordnet (siehe Abschnitt Hierarchie). Diese Group kann eine Subgroup einer andern Group sein, kann aber auch direkt einer Domain untergeordnet sein. Allerdings ist dabei zu beachten, dass nur die oberste Group einer Domain zugeordnet werden kann.

Neben der Zuordnung zur Group/Domain, besitzt das Device eine weitere Zuordnung zu einem Model. Das Model (z.B. Netgear X5400j) ist wiederum mit einer Category (z.B. Router) verknüpft.

Anhand der Category können die richtigen Configuration Templates gewählt werden. Jedes Model hat ein bestimmtes Mapping zwischen unserer “Independet Configuration” und der pro Model einzigartigen “Proprietary Configuration”. Nun wird zwischen den, im Configuration Template, eingetragenen Ranges/Werten und der Mapping Configuration ein Specified Config für jedes Device erstellt.

Um die Konfigurationen auf die Devices zu bringen, werden Tasks eingesetzt. Das bedeutet, dass es auf dem Management Server eine Queue pro Device gibt, welche die anstehenden Tasks beinhaltet bis sich das Device das nächste Mal meldet. Ist dies der Fall, werden die Tasks an das Device gesendet und dort entsprechend prozessiert.

5.5.5 Erweiterungen

In diesem Abschnitt werden einzelne Erweiterungen für das vorliegende Konzept präsentiert. Es wird beschrieben, was deren Vorteile sind, wenn man sich entscheidet diese einzusetzen. Diese Erweiterungen werden in diesem Projekt allerdings nicht weiter verfolgt.

5.5.5.1 Regelbasierte Zuordnung der IoT-Geräte

Eine weitere Möglichkeit, IoT-Geräte den entsprechenden Domains oder Groups zuzuordnen, könnten Regeln sein. Es wäre also vorstellbar, dass man als Administrator Regeln definierten könnte, anhand derer die IoT-Geräte dann automatisch beim ersten Kontakt mit der IoT Management Software den entsprechenden Domains/Groups zugeordnet werden. Mit einer regelbasierten Zuordnung wäre es also z.B. möglich, dass alle Geräte mit einer Seriennummer zwischen 300 000 - 400 000 automatisch der Domain “HSR” und der Group “Gebäude 1” / “Licht” zugeordnet werden.

5.5.5.2 Proxy

Mit CoAP besteht die Möglichkeit, einen Proxy für die Verbindung zwischen IoT-Gerät und Server einzubeziehen. Der Vorteil eines Proxys liegt darin, dass IoT-Geräte mit dem Proxy kommunizieren und dieser dann die Angaben an die Verwaltungssoftware weiterleitet. So können aus z.B. sechs Verbindungen zwischen Proxy und IoT-Geräten eine Verbindung vom Proxy zur Verwaltungssoftware gemacht werden. Dies verringert die Verbindungen zum Verwaltungsserver drastisch.

Ein weiterer Vorteil besteht darin, dass der Proxy eine Konvertierung von HTTP nach CoAP macht. So kann zwischen dem Managementserver und dem Proxy HTTP verwendet werden und wischen den IoT-Geräten und dem Proxy CoAP. Die Batterielaufzeit von IoT-Geräten könne erhöht werden, indem die Ressourcen auf dem Proxy zwischengespeichert werden. Ansonsten müsste das IoT-Gerät bei jeder Abfrage aus dem “Schlaf” erwachen, was der Batterielaufzeit schadet.

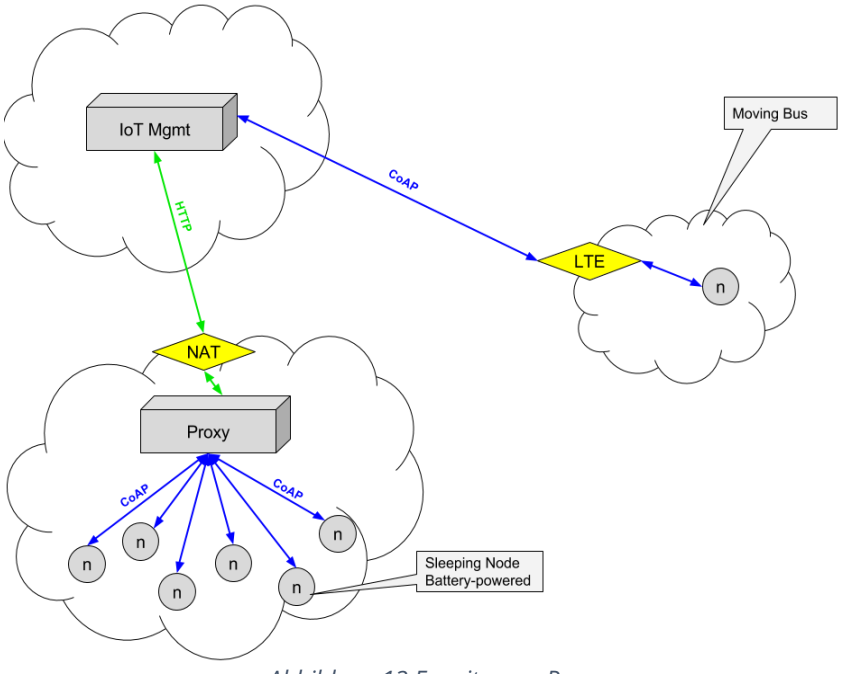


Abbildung 13 Erweiterung Proxy

5.5.5.3 SMS

Im Fall der PostAuto AG sind die IoT-Geräte über das LTE Netz also das Mobilfunknetz mit dem Managementserver verbunden. Es würde nun die Möglichkeit bestehen, ein SMS an das IoT-Gerät zu senden, so dass dieses “erwacht” und eine Verbindung zum Managementserver aufbaut. So kann eine Art Erzwingung für einen Verbindungsaufbau gemacht werden.

5.6 Systemarchitektur

5.6.1 Systemübersicht

Die Systemübersicht gibt einen groben Überblick über die einzelnen Komponenten des Systems. Nachfolgend sind die einzelnen Komponenten ein wenig genauer beschrieben. Im nächsten Abschitt wird genauer auf die physische Architektur bzw. das Deployment-Diagramm eingegangen. Dort werden die einzelnen Komponenten noch ausführlicher beschrieben.

Generell läuft unsere Applikation auf folgendem virtuellen Server der HSR:

- sinv-56025.edu.hsr.ch

Da wir nur diesen einen Webserver zur Verfügung haben, befinden sich auf diesem Server ein Webserver, ein Backend - Server welches als REST-Schnittstelle umgesetzt ist und eine Datenbank. Alle drei Komponenten sind notwendig, damit die Benutzer mit dem Server über das Web Frontend interagieren können, die IoT-Geräte über die REST-Schnittstelle ihre Daten, etc. abfragen oder senden können und alle notwendigen Informationen am Ende in der Datenbank gespeichert werden können.

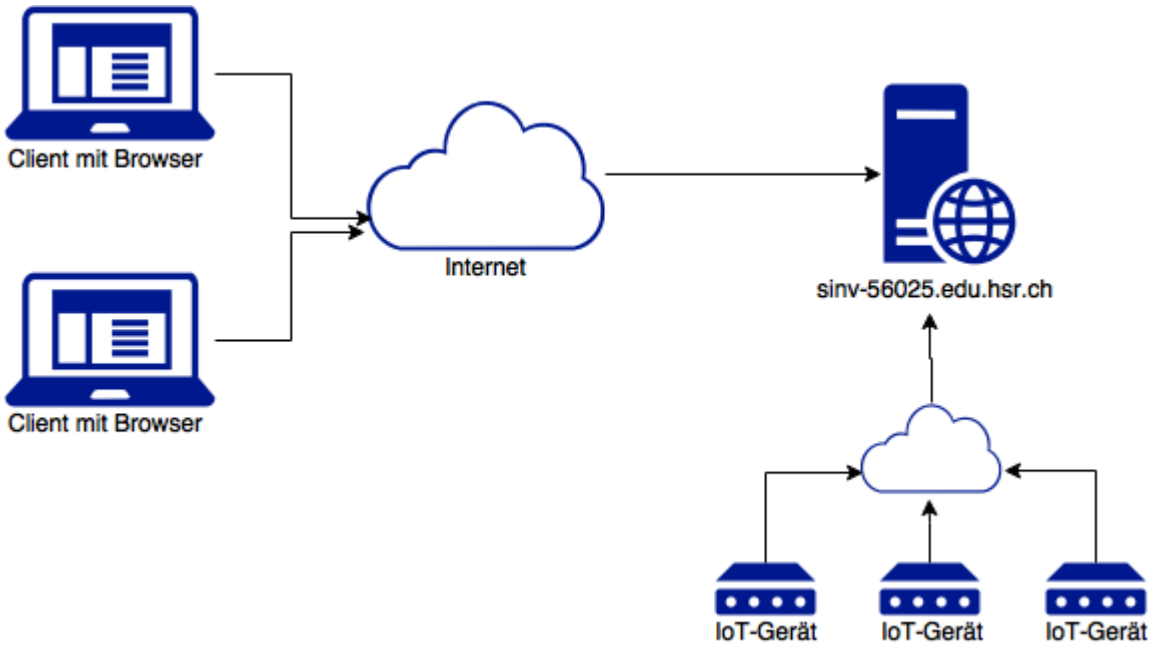


Abbildung 14 Systemübersicht

5.6.1.1 Web - GUI

Die Weboberfläche dient hauptsächlich zur Interaktion zwischen den Clients und dem eigentlichen Backend - Server. Die Clients greifen über das Internet mittels eines Browsers auf das Web - GUI zu. Auf dem Web - GUI können sie dank der grafischen Oberfläche relativ leicht mit dem Backend - Server interagieren und z.B. Informationen der IoT-Geräte ablesen.

5.6.1.2 Backend - Server

Der Backend - Server, welches als REST-Schnittstelle umgesetzt ist, dient als Vermittler zwischen der Weboberfläche und der Datenbank als auch zwischen den IoT-Geräten und der Datenbank. Hier ist die Businesslogik des gesamten Systems untergebracht.

5.6.1.3 Datenbank

Die Datenbank selbst befindet sich auch auf dem sinv-56025.edu.hsr.ch Server. In der Datenbank werden alle notwendigen Informationen über die IoT-Geräte gespeichert.

5.6.1.4 IoT-Gerät

Die einzelnen IoT-Geräte melden sich über ein Netzwerk, sei es das Internet oder z.B. das mobile Datennetz, bei dem Backend - Server. Da sich dort die gesamte Logik befindet, wird anhand der vorliegenden Situation, das IoT-Gerät entsprechend prozessiert und/oder die Informationen entsprechend weiter verarbeitet.

5.6.2 Physische Architektur

Das IoT Management System läuft auf einem virtuellen Server (sinv-56025.edu.hsr.ch) der Hochschule Rapperswil. Auf diesem Server ist ein Ubuntu als Betriebssystem installiert. Da wir nur diesen einen Server zur Verfügung haben, werden alle benötigten Subsysteme des IoT Management Systems auf diesem Server gehostet.

Nachfolgend werden die einzelnen Komponenten genau beschrieben bzw. im Deployment Diagramm aufgezeigt.

5.6.2.1 Deployment Diagramm

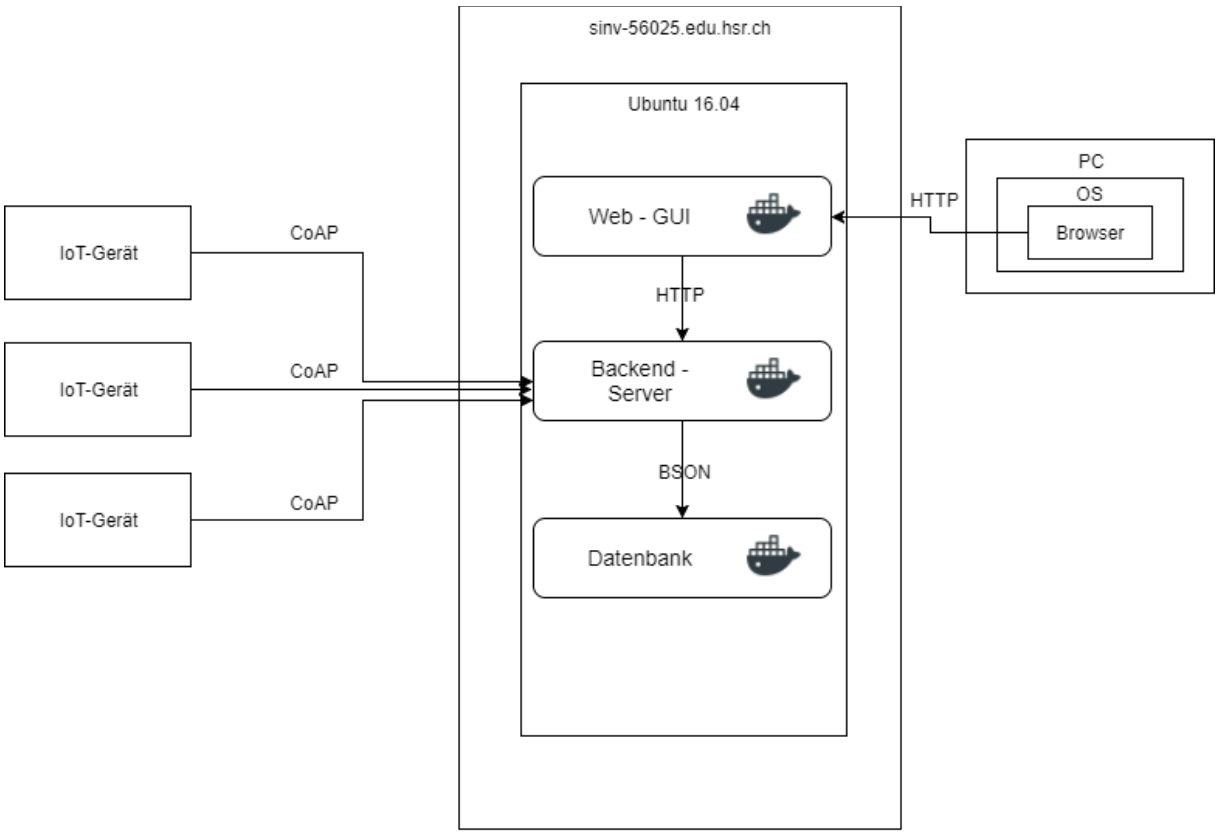


Abbildung 15 Deployment Diagramm

Auf dem Server, welcher von der Hochschule zur Verfügung gestellt wird, ist zunächst ein Ubuntu als Betriebssystem installiert. Damit die einzelnen Komponenten sauber voneinander getrennt sind, wird Docker eingesetzt. Die einzelnen Komponenten laufen daher als Docker Container auf dem physischen Server. Die grösste Last wird auf dem Backend - Server durch die enorme Anzahl von IoT-Geräten erwartet. Der Einsatz von Docker soll daher vor allem helfen, eine gute Skalierung der einzelnen Komponenten zu gewähren. Mehr dazu im Abschnitt Docker.

Die IoT-Geräte kommunizieren aber grundsätzlich via dem CoAP Protokoll mit dem Backend - Server. Anfragen vom Browser an das Web - GUI wie auch Anfragen vom Web - GUI an das Backend - Server werden über das HTTP - Protokoll übertragen. Die Verbindung vom Backend - Server zur Datenbank wird mittels dem ReactiveMongo Treiber bzw. BSON [10] als Protokoll übermittelt.

5.6.3 Web - GUI

5.6.3.1 Webserver

Für das Web - GUI bzw. die Weboberfläche verwenden wir Node.js [11] als Webserver. Node.js ist in JavaScript geschrieben und bietet eine ressourcensparende Architektur, die eine besonders grosse Anzahl an gleichzeitig bestehender Netzwerkverbindungen ermöglicht. Node.js steht unter der MIT-Lizenz [12]. Dies bedeutet, dass eine Wiederverwendung der unter der Lizenz stehenden Software erlaubt ist. Dies gilt für Software, deren Quelltext frei einsehbar ist, als auch für Software, deren Quelltext nicht frei einsehbar ist [12].

Express.js [13] ist ein serverseitiges Webframework, welches Node.js um weitere Werkzeuge erweitert. Mit dessen Hilfe kann die Entwicklung von Webanwendungen einfacher und schneller gestaltet werden. Wie Node.js ist auch Express.js in JavaScript geschrieben und steht unter der MIT-Lizenz [12].

5.6.3.2 Entscheidung

Für uns war der Entscheid Node.js/Express.js für die Umsetzung des Web - GUIs zu verwenden sehr schnell klar, da wir diese Technologien in der Schule gelernt haben und seitdem gute Erfahrungen gesammelt haben.

5.6.4 Backend - Server

5.6.4.1 Anforderungen

Da wir von einen Moment zum anderen viele neue IoT-Geräte verwalten müssen, erwarten wir ein Framework, welches auf Skalierung ausgelegt ist. Dieses Framework sollte stabil und ausgereift sein. Ausserdem muss es auf einer JVM laufen. Das Framework muss unter der Open Source Lizenz veröffentlicht sein. Dazu soll es die Integrierung in Docker ermöglichen, so dass das Deployment vereinfacht wird. Rapid Prototyping sollte durch das Framework ermöglicht werden, so dass die Entwicklung durch schnelle Feedbacks beschleunigt wird.

5.6.4.2 Play

Das Play Framework [14] ist ein, unter der Apache 2 Lizenz stehendes Web Application Framework. Es erleichtert das Erstellen von Webanwendungen und folgt dem MVC-Pattern. Es ermöglicht auf einfache Art und Weise eine RESTful API zu schreiben. Play basiert auf einer zustandslosen und schlanken Architektur. Da es auf Akka aufgebaut ist, nützt es eine vollständig asynchrones I/O. Ausserdem bietet Play minimalen Ressourcenverbrauch. Die Produktivität von Entwicklung wird zum Teil durch sofortige Code-Aktualisierung verbessert, was kein Republishing oder Restarting erfordert. Code, welcher nicht kompiliert werden konnte, wird zudem im Browser angezeigt und man weiss als Entwickler sofort auf welcher Zeile man nachbessern muss.

Seit der Version 2.0 des Play Frameworks ist der Framework Core in Scala geschrieben und als Build Tool wird SBT [15] verwendet. Für das Testen stehen verschiedene Test Frameworks sowohl für Java als auch für Scala zur Verfügung. Mittels Scalatest oder JUnit können Unit Tests für beide Sprachen geschrieben werden. Es ist auch möglich Tools wie scoverage für die Code Coverage zu verwenden. Die Ausführung der Tests sowie Generierung von Test Reports wird SBT verwendet.

5.6.4.3 Entscheidung

Da wir bereits in unserem Engineering Projekt mit dem Play Framework gute Erfahrungen gesammelt haben, entscheiden wir uns das Play Framework auch für diese Arbeit zu verwenden.

5.6.4.4 Programmiersprache

Die Programmiersprache Scala [16] wurde in der ETH Lausanne entwickelt und erstmals im Jahre 2004 veröffentlicht. Der Name leitet sich von Scalable Language ab. Gegensatz zu Java, welche eine sehr verbose Sprache ist, lassen sich Anweisung mit Scala sehr kompakt ausdrücken. Dazu vereint Scala objektorientierte und funktionale Programmierung. Deshalb wird oftmals Scala als “besseres Java” bezeichnet [17].

Bei der Kompilierung wird in Scala geschrieben Quellcode zu Java Bytecode umgewandelt. Weil Scala auf JVM läuft, ist es eine plattformunabhängige Sprache. Scala ist mit Java kompatibel, das heisst Libraries, welche in beiden Sprachen geschrieben sind, können in Scala oder in Java Quellcode eingebunden werden. Daher steht auch für Scala eine riesige Auswahl von Java Libriaries zur Verfügung [16].

Play 2 wurde in Scala geschrieben. Dennoch lässt sich Play 2 weiterhin in Java oder in Scala nutzen. Aber falls man sich für Java entscheidet, dann wird man häufig mit Scala Idiomen konfrontiert. Von Java Entwicklungsperspektive ist dies daher sehr irritierend. Wir möchten von Scalas kompakter Schreibweise profitieren und möchten das Play Framework reibungslos nutzen. Deshalb entscheiden wir das Play Framework in Scala zu nutzen [18].

5.6.4.5 Libraries

5.6.4.5.1 ReactiveMongo

Für die Datenbank Verbindung stehen viele Treiber zur Verfügung. MongoDB selbst stellt für Scala einen offiziellen Treiber zur Verfügung. Dann gibt es andere Treiber wie ReactiveMongo [19]. ReactiveMongo ist auf Akka aufgebaut, was eine gute Voraussetzung ist, dass es optimal mit dem Play Framework interagieren kann. Dazu kommt noch, dass ReactiveMongo ein gut gepflegter Treiber ist [20].

5.6.4.5.2 Californium

Die Auswahl von CoAP Intefaces ist verglichen mit HTTP Interfaces eingeschränkt. Leider gibt es noch keine direkte Scala Implementation von CoAP. Da es möglich ist, Java Libraries reibungslos in Scala einzubinden, werden auch Java Libraries berücksichtigt. Es gibt zwei serverseitige CoAP Interfaces in Java. Der erste Kandidat heisst nCoap [21], welcher auf dem Netty Framework basiert. Leider unterstützt nCoAP die DTLS-Verschlüsselung nicht und fällt als verwendbare Library weg. Der zweite Kandidat heisst Californium [22], welches von Eclipse entwickelt wird. Da am Schluss nur Californium für uns übrig bleibt, wird Californium verwendet.

5.6.4.5.3 Swagger

Mit Swagger [23] können die REST-APIs sprachunabhängig beschrieben und dokumentiert werden. Dies ermöglicht es Menschen, die Funktionen des Web Services ohne Zugriff auf den Quellcode zu entdecken und zu verstehen. Ein Benutzer kann das Restful API verstehen und mit diesem interagieren. Durch das Verwenden von Swagger wird das Raten, wie man mit dem Server interagieren muss, entfernt. Deshalb nutzen wir die offizielle Swagger-Play Erweiterung [24].

5.6.4.6 Lizenz-Kompatibilität

Der Prototyp wird mit der Apache 2.0 Lizenz veröffentlicht. In der nachfolgenden Tabelle wird aufgezeigt, ob die Lizenzen der Software Komponenten mit Apache 2.0 kompatibel sind. Gemäss Apache Legal [25] ist Apache 2.0 mit Eclipse Distribution 1.0 kompatibel. Gemäss WhiteSource [26] sind BSD und MIT mit Apache 2.0 kompatibel.

Software	Lizenz	kompatibel
Play	Apache 2.0	✓
ReactiveMongo	Apache 2.0	✓
Californium	Eclipse Distribution 1.0 Eclipse Public 1.0	✓
Swagger-Play	Apache 2.0	✓
Guice	Apache 2.0	✓
Scalatestplus	Apache 2.0	✓
Mockito-core	MIT	✓
Jackson-dataformat-csv	Apache 2.0	✓
Scalaz-core	BSD	✓

5.6.5 Datenbank

5.6.5.1 Anforderungen

Das Datenbanksystem kann entweder vertikal oder horizontal skaliert werden. Unter vertikaler Skalierung wird das Aufrüsten vorhandener Server durch bessere Hardwarekomponenten verstanden. Unter horizontaler Skalierung wird die Erhöhung der Anzahl der verfügbaren Server verstanden. Dabei müssen die neuen Server nicht leistungsfähig sein. Bei horizontal skalierten Systemen liegt der Vorteil in der Verfügbarkeit und Ausfalltoleranz. Deshalb erwarten wir, dass die Datenbank horizontal skaliert.

Da wir keine Anforderung haben, dass eine externe Abteilung die Anwendungsdaten auswerten muss, wird auf die Normalisierung verzichtet. Dadurch wird das sogenannte Impedance Mismatch [27] vermieden, welche bei der Speicherung von objektorientierten Daten in relationalen Datenbanken auftritt. Mit anderen Worten erwarten wir eine Datenbank, welche die Speicherung von einzelne Objekte auf natürliche Art und Weise erlaubt. Dadurch wird die Produktivität der Entwicklung gesteigert.

5.6.5.2 MongoDB

MongoDB [28] ist eine dokumentenorientierte NoSQL-Datenbank. Der Name leitet sich von huMONGOus ab, was so viel bedeutet wie gigantisch [29]. Die Datenbank selber wird unter der GNU Affero General Public Lizenz veröffentlicht und die offiziellen Treiber für diese Datenbank werden unter der Apache Lizenz veröffentlicht. MongoDB gehört zu den populärsten NoSQL Datenbanken. Entsprechend gibt es eine grosse Gemeinschaft und eine grosse Anzahl von Tools für MongoDB. Deshalb entscheiden wir uns für MongoDB.

5.6.6 Pattern

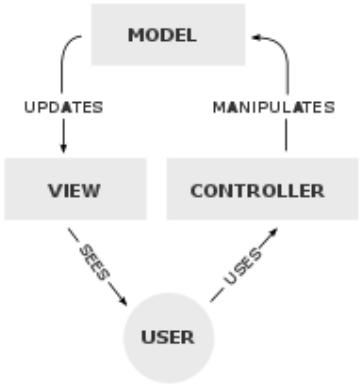


Abbildung 16 MVC Model [45]

Das MVC Pattern [30] oder model-view-controller Pattern ist ein Softwarearchitektur Pattern, welches traditionell bei der Implementierung von Anwendungen mit einer Benutzeroberfläche zur Anwendung kommt. Es wird heutzutage in vielen populären Programmiersprachen wie Java, C#, Ruby, PHP und vielen anderen out-of-the-box unterstützt und angewendet.

Die Idee hinter dem model-view-controller Pattern liegt darin, dass die Applikation in drei verbundene Teile (model, view, controller) getrennt wird. Mit dieser Vorgehensweise kann sichergestellt werden, dass die interne Repräsentation bzw. Verarbeitung der Informationen vollkommen getrennt ist von der Art und Weise wie die Informationen dem User auf der Benutzeroberfläche dargestellt werden. Es entkoppelt also die Verarbeitung der internen Objekte/Informationen komplett von der Darstellung, womit eine hohe Kohäsion und tiefe Kopplung angestrebt wird. Daraus resultiert, dass der Code der einzelnen Teile einfacher und effizienter wiederverwendet werden kann. Zudem ist eine parallele Programmierung an den einzelnen Teilen einfacher zu bewerkstelligen.

5.6.6.1 Komponenten

5.6.6.1.1 Model

Das Model ist die zentrale Komponente des MVC Patterns und steuert die Logik der Applikation unabhängig von der Benutzeroberfläche. Es steuert direkt die Daten, Logik und oder Regeln, für welches es selbst verantwortlich ist.

5.6.6.1.2 View

Die View ist im Prinzip nur verantwortlich für die Repräsentation der Informationen, welche sich im Model befinden. Es ist durchaus möglich, dass mehrere Views für ein Model zur Verfügung stehen.

5.6.6.1.3 Controller

Wie der Name schon sagt, kontrolliert der Controller das Model bzw. sendet die gewünschten Informationen, welche z.B. in der View eingegeben wurden an das Model.

5.6.6.2 Entscheidung

Sowohl das Play Framework wie auch Nodejs bzw. Express unterstützen das MVC Pattern out-of-the-box. Da wir, wie oben erwähnt, mit dem Play Framework und Node.js und somit mit dem MVC Pattern schon gute Erfahrungen gemacht haben, werden wir auch dieses Pattern mit seinen Vorteilen wiederverwenden.

5.6.7 Technologie Stack

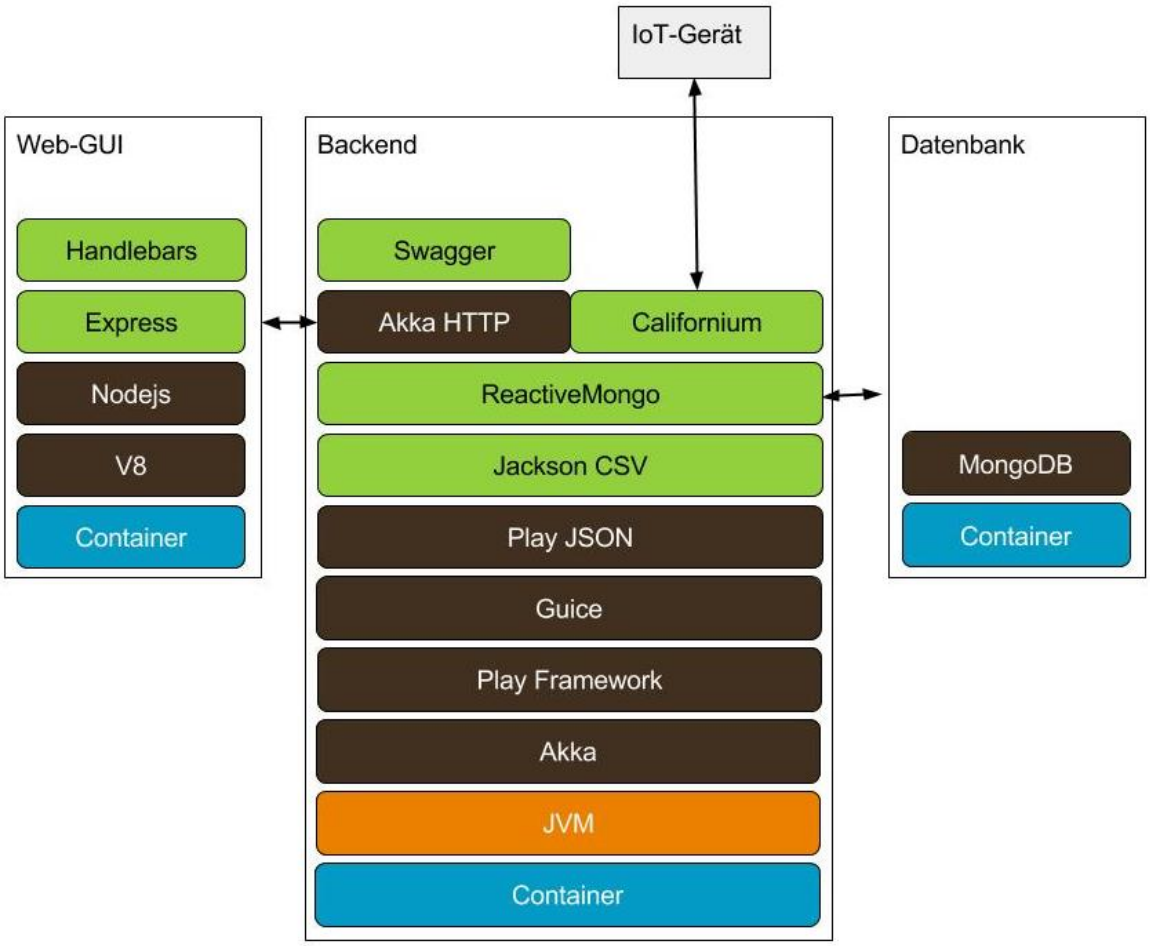


Abbildung 17 Technologie Stack

Der Technologie Stack stellt prinzipiell dasselbe dar, wie das Deployment Diagramm. Entsprechend werden auch dieselben Protokolle zwischen den einzelnen Containern verwendet. Allerdings wurde auf die Beschriftung verzichtet, da hier der Fokus auf den eingesetzten Technologien liegt.

Schwarz bedeutet, dass die Komponente vom Framework standardmässig gegeben ist. Grün bedeutet, dass die Komponente von uns aus erweitert wird. Alle 3 Tiers laufen in unterschiedlichen Containern. Mit Hilfe von Docker Compose werden die Container orchestriert.

Beim Web-GUI Tier verwenden wir Nodejs als Plattform. Diese Plattform läuft auf der JavaScript Laufzeitumgebung namens V8, welches ursprünglich für Google Chrome entwickelt wurde. Wir verwenden Express als Web Framework. Dazu verwenden wir Handlebars als Template Engine.

Beim Datenbank Tier wird einfach ein MongoDB Container gebraucht.

Beim Backend Tier wird ein JVM Container benötigt. Play ist auf der Akka Plattform aufgebaut. Für die Runtime Dependency Injection wird standardmässig Googles Guice verwendet. Als JSON Library wird Play JSON verwendet. Grundsätzlich ist Play JSON nur ein Wrapper um die bekannte Jackson Library. Um CSV Files zu parsen, wird Jacksons CSV Library eingesetzt. Beim Datenbanktreiber wird ReactiveMongo verwendet, weil ReactiveMongo auch auf Akka aufgebaut ist. Dadurch ist garantiert, dass der Datenbanktreiber reibungslos mit Play interagiert. Vor Play 2.6 wurde Netty für die HTTP Kommunikation verwendet. Aber ab Play 2.6 wird standardmässig Akka HTTP verwendet. Akka HTTP ist auch besser geeignet, weil es auf der Akka Plattform aufgebaut ist. Für die CoAP Kommunikation wird Californium von Eclipse genutzt. Um eine Dokumentation für unsere Restful API zur Verfügung zu stellen, wird Swagger verwendet. [31]

5.6.8 Version

Die meiste Software befolgen eine semantische Versionierung [32]. Das bedeutet, dass die erste Zahl der Version die Major Version repräsentiert, die zweite Stelle die Minor Version und die dritte Stelle die Patch Version angibt.

Entsprechend fixieren wir die Major und Minor Version. Die Patch Version lassen wir frei oder anders gesagt, die Software sollte jeweils auf die aktuellste Patch Version aktualisiert werden.

Bei Erweiterungen wie Swagger-Play wurde darauf geachtet, dass es mit Play kompatibel ist. Auf Ebene vom OS ist es prinzipiell nicht wichtig welches Betriebssystem verwendet wird. Es ist allerdings von Vorteil, dass ein GNU/Linux OS verwendet wird, weil Docker native auf GNU/Linux OS läuft.

Tier	Software	Version (MAJOR.MINOR.PATCH)
Web-GUI	Node.JS	6.11.x
	Express	4.16.x
	Handlebars	4.0.x
Backend	Java	8.x
	Scala	2.12.x
	Play	2.6.x
	ReactiveMongo	0.12.x
	Californium	1.0.x
	Swagger-Play	1.6.x
Datenbank	MongoDB	3.4.x
OS	Ubuntu	16.04 LTS

5.6.9 Docker

Docker [33] ist eine unter der Open-Source stehende Software, welche verwendet wird, um mit Hilfe von Virtualisierungen ganze Applikationen in sogenannten Containern zu isolieren und somit vom Host System zu trennen. Prinzipiell ist Docker auf die Virtualisierung auf Linux ausgerichtet. Mit Hilfe von Virtual Machines kann Docker aber auch auf Windows oder auf Mac OS verwendet werden.

Ein Docker Container ist also sowas wie eine Mini-VM. Der Unterschied zu einer echten VM besteht darin, dass ein Docker Container viel schlanker und kleiner ist als eine echte VM. Das liegt daran, dass der Container immer noch die Ressourcen des Host-OS verwendet. Im Container selbst sind lediglich alle notwendigen Libraries, welche für die "containerisierte" Applikation benötigt werden und natürlich die Applikation selbst. Die getrennte Kapselung vom darunterliegenden System garantiert aber eine portable und lauffähige Umgebung.

Die Verteilung und das Administrieren von einzelnen Containern auf verschiedenen Hosts kann mit Docker-swarm [34] umgesetzt werden. Swarm erlaubt es, verschiedene Host zu einem Cluster zusammenzufügen. So kann eine Vielzahl an Containern auf beliebig viele physische sowie virtuelle Hosts verteilt werden. Die Steuerung des Clusters geschieht durch eine, als "master" bezeichnete Maschine.

Damit Docker weiss, welche Maschine wie mit anderen Kommuniziert oder welcher Container überhaupt auf die Maschine geladen werden muss, gibt es das sogenannte Docker-Compose.yml File. In diesem File können komplexe Strukturen aus mehreren Container und Netzwerken elegant verwaltet werden. Das Compose-File enthält unter anderem folgende Informationen:

- Images oder Dockerfiles
- Portmapping
- Umgebungsvariablen
- Abhängigkeiten der Container

Der grosse Vorteil von Docker-Compose in Kombination mit Swarm ist das automatische Deployment und eine einfache Skalierung. Sobald der Aufbau auf dem Master mittels "docker-compose up" abgeschlossen ist, werden die Container auf die einzelnen Hosts verteilt und können dann mit "docker-compose scale" ohne grösseren Aufwand skaliert werden.

Da in unserem Projekt die Skalierung eine wichtige Rolle spielt, setzen wir auf diese beiden Technologien. Zu beachten ist jedoch, dass wir nur einen physischen Server zur Verfügung haben und wir daher nur virtuelle Hosts in den Swarm-Verbund joinen können.

Abschliessend kann aber gesagt werden, dass Docker eine gute Basis für eine Skalierung bietet. Der gesamte Swarm sowie die Container sind komplett unabhängig von der darunterliegenden Hardware und die Verteilung sowie die Skalierung sind elegant mit dem Compose File umgesetzt. Docker selbst ist als Open Source Projekt entstanden und entwickelt sich auch ständig weiter. Da wir auch schon einzelne Erfahrungen mit Docker gesammelt haben, sehen wir Docker als geeignet an, um es vor allem im Bereich Skalierung einzusetzen.

5.6.10 Logische Architektur

Das IoT Management System besitzt zwei Subsysteme mit diversen Subpackages, um die Funktionalität zur Verfügung zu stellen. Dies sind zum einen die REST-Schnittstelle und zum anderen die Webapplikation. In den nachfolgenden Abschnitte werden diese Subsysteme detaillierter beschrieben und auf die Funktionen der einzelnen Klassen eingegangen.

5.6.10.1 Web - GUI

Die Webapplikation implementiert die Benutzeroberfläche als Webseite. Sie dient hauptsächlich der Interaktion mit dem Benutzer. Im Hintergrund greift sie auf den Backend - Server zu, um die notwendigen Daten oder Prozesse zu erhalten bzw. abzuwickeln.

5.6.10.1.1 Klassenstruktur

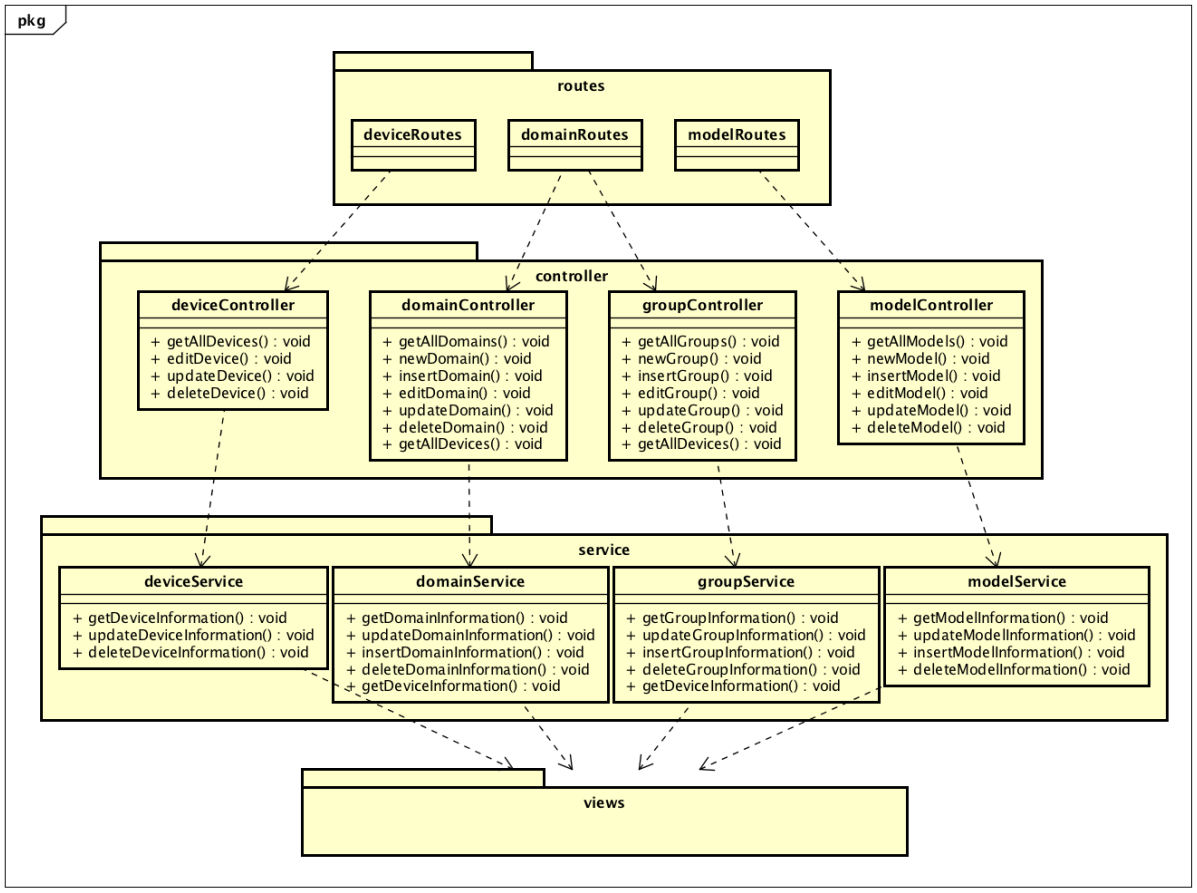


Abbildung 18 Klassenstruktur WEB

Generell kann man bei der Struktur des Web – GUIs sagen, dass es sich am Model-View-Controller Pattern [30] orientiert. Zunächst weisen die verschiedenen Routen, welche über den Browser eingegeben werden, zu den entsprechenden Controllern bzw. zu einer Methode im Controller. Die Controller-Methoden selbst, leiten dann zur richtigen Service-Methode weiter.

In der Service Methode wird ein Request mit den Angaben für URL, Method, Header und z.T. auch Body erstellt und an den Backend – Server geschickt. War der Request erfolg-

reich, wird eine Callback-Methode aufgerufen, welche die gewünschte View rendert und im Browser darstellt.

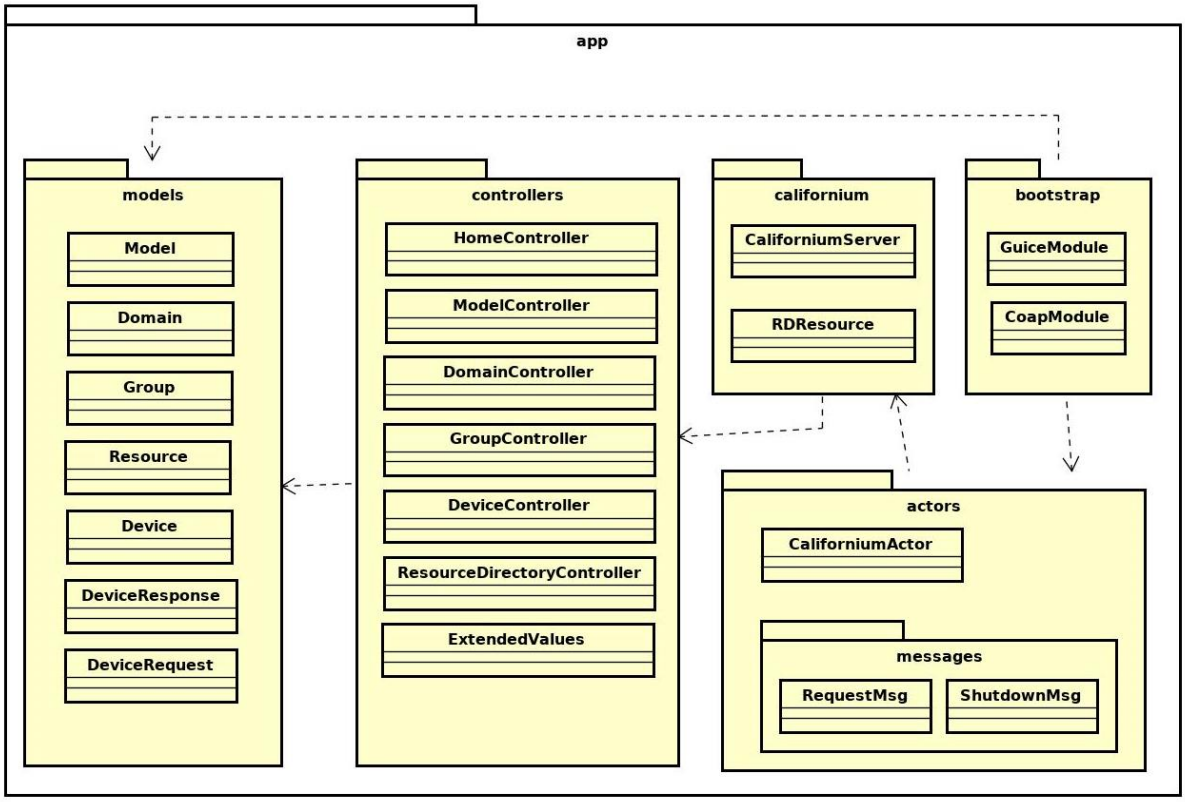
Auf die einzelnen Views sowie die Routen wurde hier bewusst verzichtet, um eine besser Übersicht zu gewähren.

Packages	Beschreibung
routes	Die Klassen in diesem Package sind dafür zuständig, dass die einkommenden Aufrufe an die richtigen Controller weitergeleitet werden. Dabei wird zwischen deviceRoutes, domainRoutes und modelRoutes unterschieden. Die domainRoutes sind insofern speziell, dass sie auch Aufrufe zum groupController weiterleiten. Dies ist der Fall, weil die Gruppen von den Domänen abhängen.
controller	In den Controller-Klassen wird hauptsächlich gesteuert, welcher Service bzw. welche Service-Methode ausgeführt wird. Des Weiteren wird eine Methode definiert, welche dann auf den Services als Callback aufgerufen wird.
service	Die Service-Klassen erzeugen einen Request mit URL, Method, Header und Body. Dieser Request wird dann an den Backend – Server gesendet. Bei erfolgreicher Ausführung wird die im Controller definierte Callback-Methode ausgeführt.
views	Die Views beschreiben mittels HTML und Handlebars Syntax, was auf der Seite dargestellt werden soll und wie sie aussieht.

5.6.10.2 Backend - Server

Der Backend - Server bildet das Verbindungsstück zwischen der Webapplikation und der Datenbank bzw. zwischen den IoT-Geräten und der Datenbank. Hier ist die eigentliche Logik des gesamten Systems implementiert.

5.6.10.2.1 Klassenstruktur



Packages	Beschreibung
models	Die Model-Klassen geben die Struktur der Objekte vor. Ausserdem sind sie auch für die korrekte Datenpersistenz verantwortlich.
controllers	Diese Klassen sind zuständing, dass die eingehenden Request richtig verarbeitet werden. Hier befindet sich die gesamte Businesslogik.
californium	Im Californium-Package werden alle Klassen bezüglich dem CoAP Protokoll verwaltet.
bootstrap	Diese Module werden beim Aufstarten ausgeführt. Sie sind vorallem für die Dependency Injection und den Lifecycle des CoAP Servers verantwortlich.
actors	Die Aktoren und ihre Messages werden hier verwaltet.

5.7 Realisierung

In diesem Abschnitt wird aufgezeigt, was die jeweiligen Layers anbieten. Es werden keine Quellcodes gezeigt. Falls man dennoch am Quellcode interessiert ist, kann dieser unter folgendem Link gefunden werden:

https://gitlab.com/iotmgmt/

5.7.1 Datenmodellierung

Die folgenden Modelle entsprechen der Datenstruktur, welche vom Backend - Server zurückgegeben bzw. erwartet wird. Diese Modelle sind, bis auf ein paar Kleinigkeiten, fast dieselben wie die effektiven Datenbank-Modelle. Die Objekte werden mittels Referenzen miteinander verknüpft.

5.7.1.1 Model

Alle Felder sind erforderlich.

```
{
    "_id": "netgear_500x",
    "description": "unreliable, slow router",
    "category": "router"
}
```

5.7.1.2 Domain

Alle Felder sind erforderlich.

```
{
    "_id": "hsr",
    "description": "HSR Hochschule für Technik Rapperswil"
}
```

5.7.1.3 Group

Das Feld parent ist optional. Die restlichen Felder sind jedoch erforderlich.

```
{
    "_id": ".building1.floor2.room262",
    "domain": "hsr",
    "description": "the best room",
    "parent": ".building1.floor2"
}
```

5.7.1.4 Device

Für das Erstellen von Devices sind nur die folgenden drei Felder erforderlich.

```
{
    "modelNo": "netgear_500x",
    "serialNo": "MNBNN4555220012",
    "domain": "stock"
}
```

Das Feld _id wird automatisch von der Datenbank gesetzt. Die Timestamp Felder wie „created“ und „lastNotified“ werden mittels der Java 8s Instant-Klasse erzeugt und geben den Timestamp im Standard ISO-8601 aus [35]. Das Z am Ende bedeutet, dass es in der Zulu Zeitzone repräsentiert ist, was der UTC±0 Zeitzone entspricht [36]. Daher muss beim Interpretieren von Timestamps eine Stunde dazu gerechnet werden. Das Feld „maxAge“ gibt den Intervall von Keepalive in Sekunden an. Der Public Key ist in base64 codierter Form bei dem Feld „publicKey“ hinterlegt.

Beim Resource Objekt ist nur das Feld „path“ erforderlich. Die Felder „resourceType“, „interface“ und „contentType“ gehören zu den CoRE Link Attributen [37]. Content Type 50 bedeutet, dass die Resource im JSON Format zurückgegeben wird [38].

```
{
    "_id": "5a267395acc4819bdae5f728",
    "name": "IoT-Device105",
    "address": "192.186.1.10:5683",
    "modelNo": "netgear_500x",
    "serialNo": "MNBNN4555220012",
    "domain": "hsr",
    "group": ".building1.floor2",
    "created": "2017-12-14T15:28:43.228Z",
    "lastNotified": "2017-12-14T15:49:20.298Z",
    "maxAge": 60,
    "publicKey": "-----BEGIN PUBLIC KEY-----
                MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4IjNf
                 -----END PUBLIC KEY-----",
    "status":"offline",
    "resources": [
        {
            "path": "/dev/mainlight",
            "resourceType": "lightsensor",
            "interface": "sensor",
            "contentType": 50,
            "lastNotified": "2017-12-14T15:49:20.298Z",
            "value": "3000 Lux"
        }
    ]
}
```

5.7.2 Backend

Das Backenend bietet HTTP Ressourcen für die Frontend Anwendung und CoAP Ressourcen für IoT-Geräte an. Dabei befolgt es die herkömlichen REST-API Konventionen. Zusätzlich wird bei der Verwendung der Methode POST, PUT oder DELETE das Objekt als Response zurückgegeben. Der Content-Type „application/json“ wird als Response zurückgegeben und wird entsprechend auch als Request erwartet. Im folgenden Abschnitt werden die Endpunkte der beiden Protokolle beschrieben.

5.7.2.1 HTTP Ressourcen

Models
Alle CRUD Operationen werden für Models unterstützt.

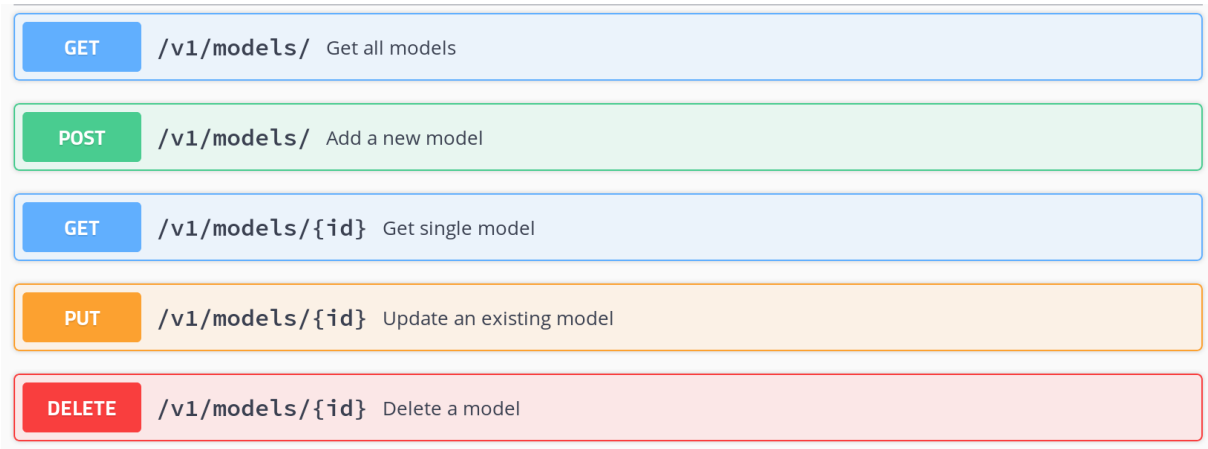


Abbildung 19 Ressourcen von Models

Domains
Alle CRUD Operationen werden für Domains unterstützt.

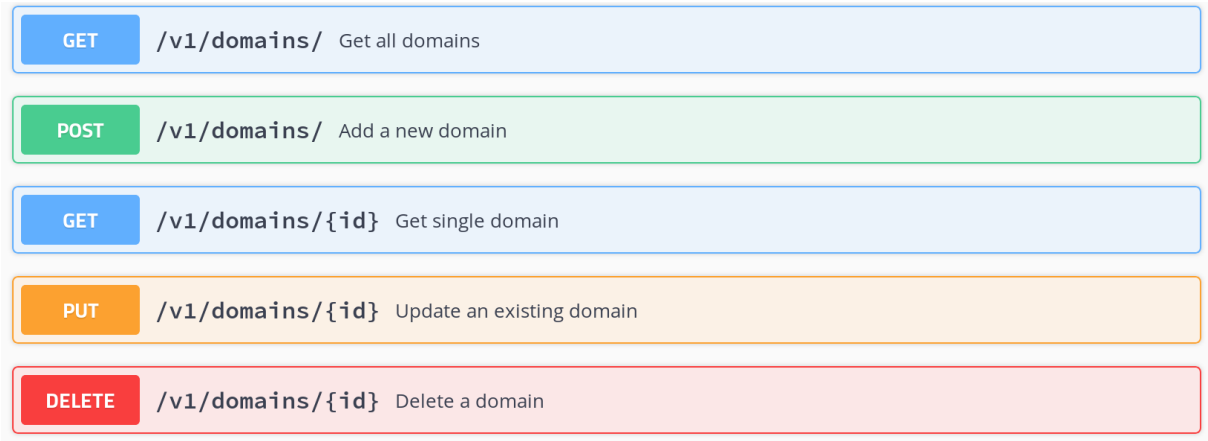


Abbildung 20 Ressourcen von Domains

Groups

Alle CRUD Operationen werden für Groups unterstützt.

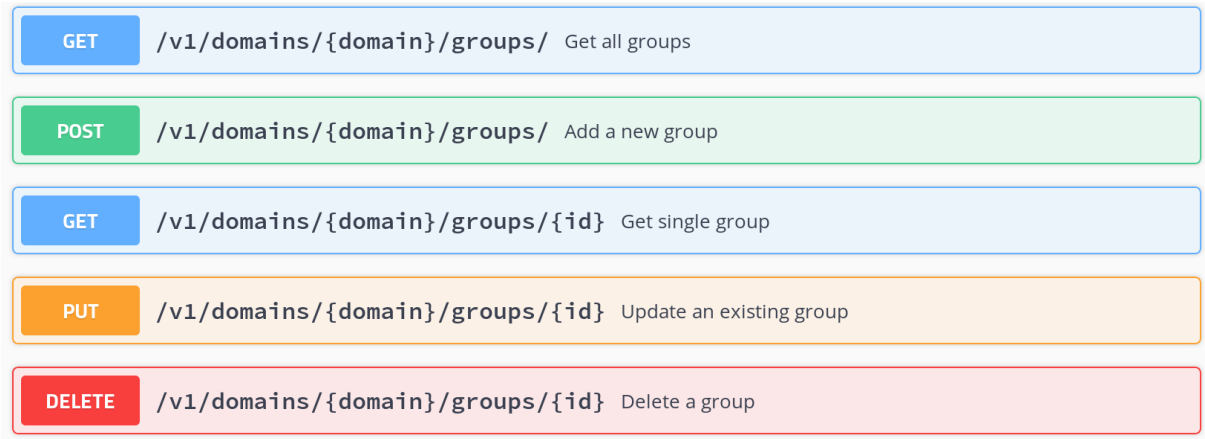


Abbildung 21 Ressourcen von Groups

Devices

Alle CRUD Operationen werden für Devices unterstützt. Zudem wird die Methode PATCH unterstützt, dadurch können auch einzelne Felder von Devices an das Backend zugesendet werden. Ausserdem können alle Devices von einer spezifischen Domain oder von einer spezifischen Group mittels GET abgefragt werden.

Abbildung 22 Ressourcen von Devices

Dann gibt es noch ein speziellen Endpunkt „/devices/import/“, welche für den Import von Lieferscheinen gedacht ist. Der Content-Type muss „multipart/form-data“ sein. Der Key des Files muss „data“ genannt werden. Es wird ein CSV File erwartet. Dabei wird die Seriennummer (sn) in der erste Spalte, die Modellnummer (mn) in der zweite Spalte und der Public Key (key) in der dritte Spalte erwartet. Das File muss wie gefolgt aufgebaut sein:

```
sn,mn,key
100000001,STOR, "-----BEGIN PUBLIC KEY-----
                 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4IjNf
                -----END PUBLIC KEY-----"
100000002,STOR, "-----BEGIN PUBLIC KEY-----
                 MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4IjNf
                -----END PUBLIC KEY-----"
```

5.7.2.2 CoAP Ressourcen

Resource Directory

Der erste Endpunkt wird automatisch von CoAP unterstützt. Dadurch können alle verfügbaren Ressourcen abgefragt werden. Beim zweite Endpunkt können andere IoT-Geräte alle eingetragen Devices abfragen. Beim dritten Endpunkt sind die Query Strings „sn“ und „mn“ erforderlich. Der Query String „ep“ ist optional. Eine Voraussetzung des dritten Endpunktes ist, dass die SerienNr und ModellNr bereits im Backend erfasst sein müssen. Mit diesem Endpunkt können die IoT-Geräte ihre Ressourcen registrieren lassen. Falls das IoT-Gerät eine Resource mit Resource Type „heartbeat“ hat, dann wird diese Resource nach erfolgreicher Registrierung automatisch vom Backend observiert. Dadurch wird eine Art von Keepalive Signal für das Monitoring bewerkstelligt.

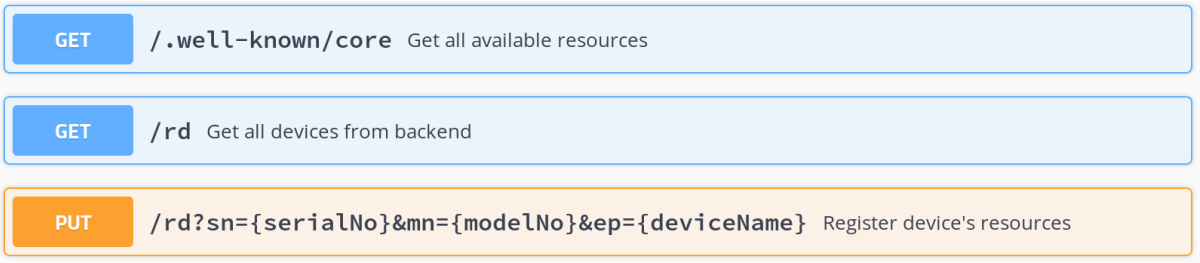


Abbildung 23 Ressourcen von CoAP

5.7.3 Frontend

Das Web GUI wurde mit Hilfe von Bootstrap v3 umgesetzt.

5.7.3.1 Layout

Das Layout ist in zwei Bereiche unterteilt. Die lila-farbene Elemente ist für die Navigation gedacht. Die grauen Elemente sind dem Inhalt gewidmet. Zudem ist das Layout responsiv. Entsprechend wurde es wie folgt gestaltet:

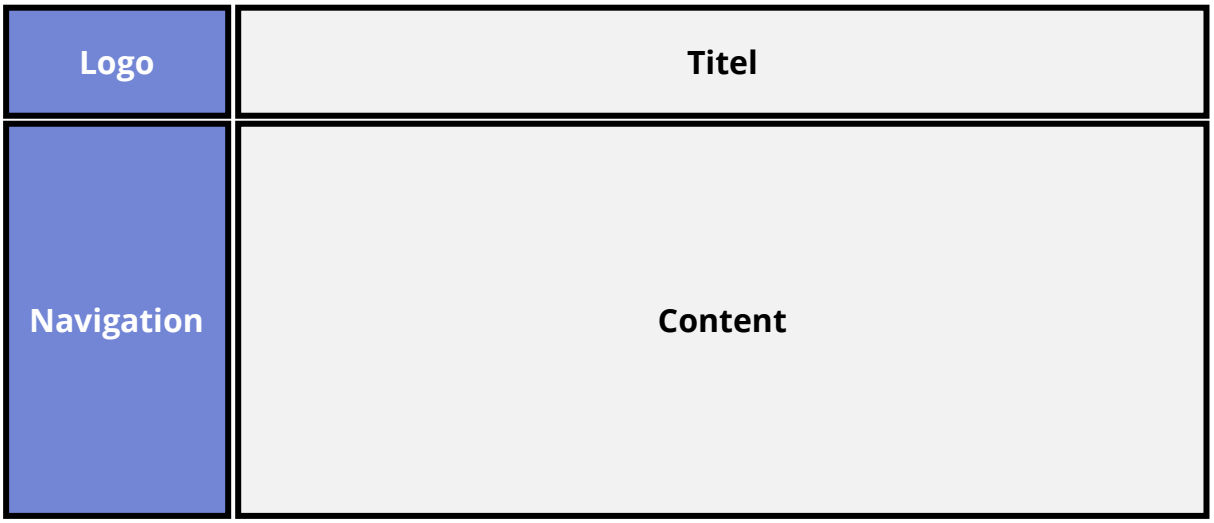


Abbildung 24 Web GUI Layout

5.7.3.2 Navigation

Die Hierarchie der Navigation ist folgendermassen aufgebaut:

- IoT Devices
- Import IoT Devices
- Models
    - All Models
    - New Models
- Domains
    - New Domain
    - All Domains
        - show Devices
        - show Groups
            - show Devices
            - new Group

5.7.3.3 Übersicht Geräte

Wenn auf den Menueintrag „IoT Devices“ geklickt wird, dann erscheint folgende Ansicht. Unterhalb des Titels kann die Anzahl sichtbarer Einträge geändert werden. Standardmässig werden 10 Einträge angezeigt. Unten rechts befindet sich die Seitennavigation.

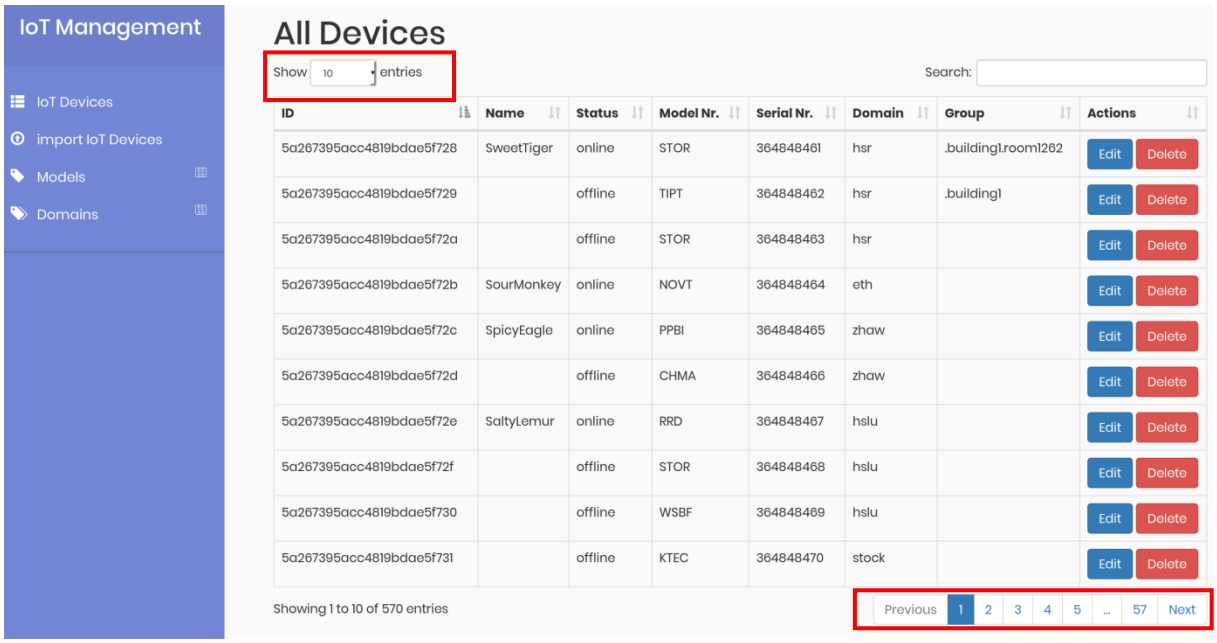


Abbildung 25 UI - All Devices - Big Menu

Mit einem Klick auf das Logo bzw. den Namen „IoT Management“, wird das Navigations-Element schmaller. Standardmässig wird in der Tabelle nach ID aufsteigend sortiert aber mit einem Klick auf einer der Spaltenüberschriften kann nach einem anderem Kriterium entweder aufsteigend oder absteigend sortiert werden.

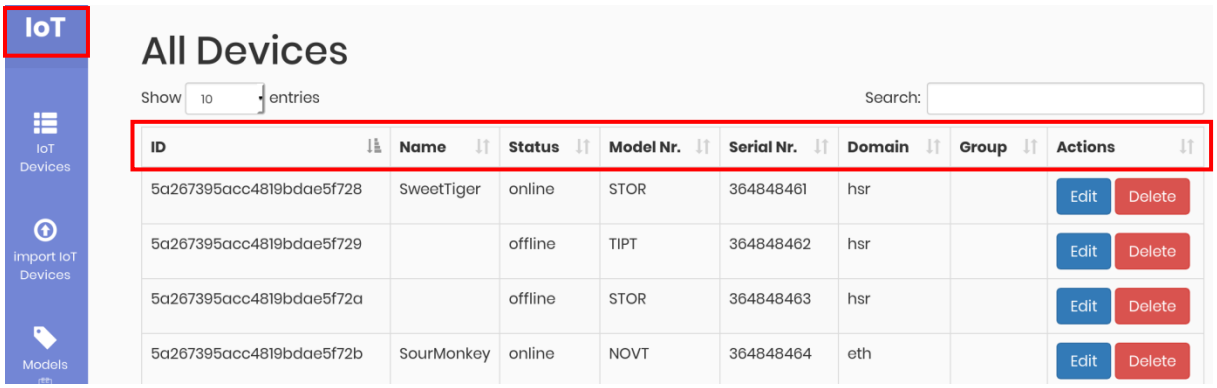


Abbildung 26 UI - All Devices - Small Menu

Oben rechts befindet sich das Suchfeld. Unten links wird beim Footer anzeigt, wie viele Einträge zu einem Suchkriterium gefunden wurden.

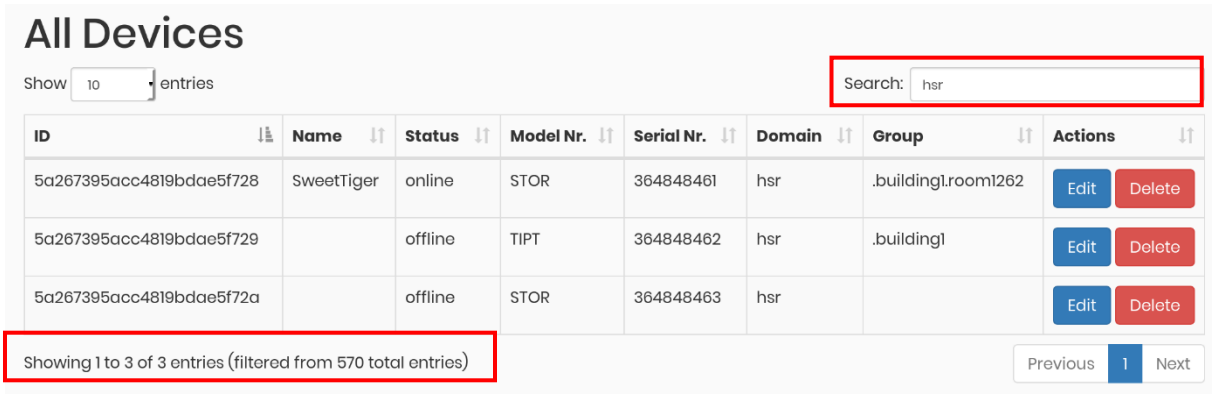


Abbildung 27 UI - All Devices - Suchergebnisse

5.7.3.4 Ansicht einzelnes Gerät

Die Felder, welche nicht verändert werden dürfen sind deaktiviert und daher werden sie grau hinterlegt. Mit dem grünen Button „Submit“ werden die Änderungen vorgenommen. Mit dem roten Button „Cancel“ werden die Änderungen bei den Feldern wieder zurückgesetzt. Die verknüpften Felder, wie Domain, haben eine Dropdown Liste.

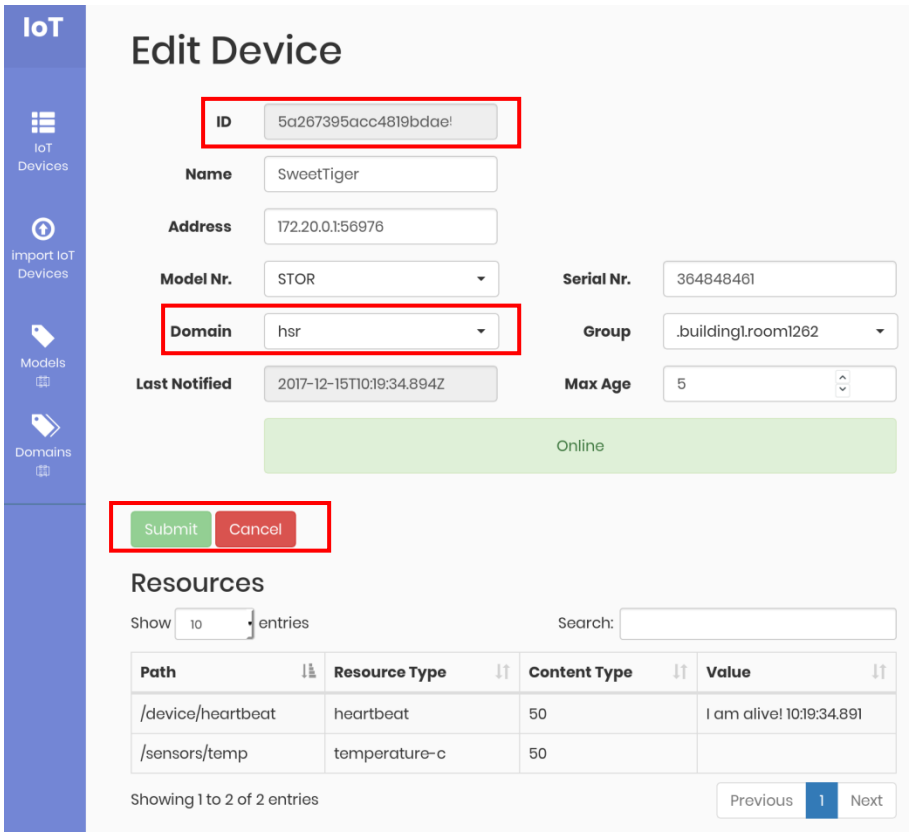


Abbildung 28 UI - Edit Device

Falls die Änderungen erfolgreich waren, dann wird eine Nachricht wie folgt gezeigt.

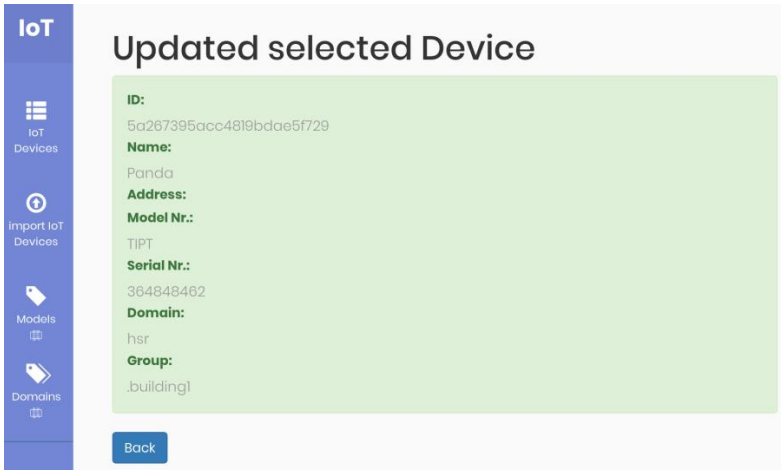


Abbildung 29 UI - successfully updated Message

5.7.3.5 Ansicht Import

Für den Import steht nur ein Feld zur Verfügung. Damit kann der Pfad des CSV Files angegeben werden.

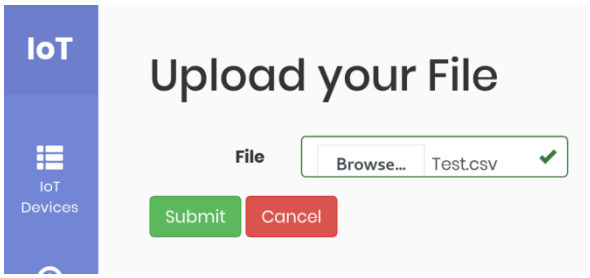


Abbildung 30 UI - Import Ansicht

5.7.3.6 Übersicht Models

Bei der Übersicht aller Modelle ist die Handhabung dieselbe wie bei „All Devices“.

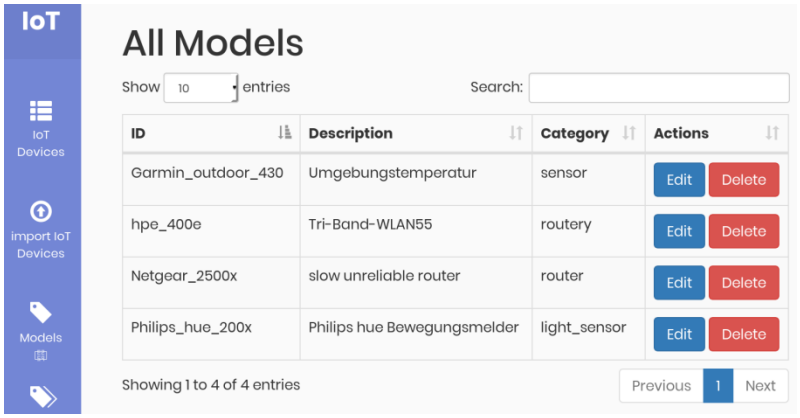


Abbildung 31 UI - All Models

5.7.3.7 Übersicht Domains

Bei dieser Ansicht stehen mehrere Optionen zur Verfügung. Die erste Option ist „show Groups“. Da die Groups abhängig von Domains sind, muss zuerst in die „All Domains“ Ansicht gewechselt werden, bevor die Groups der jeweiligen Domaine angezeigt werden können. Die zweite Option ist „show Devices“, damit können alle IoT-Geräte einer Domain aufgelistet werden.

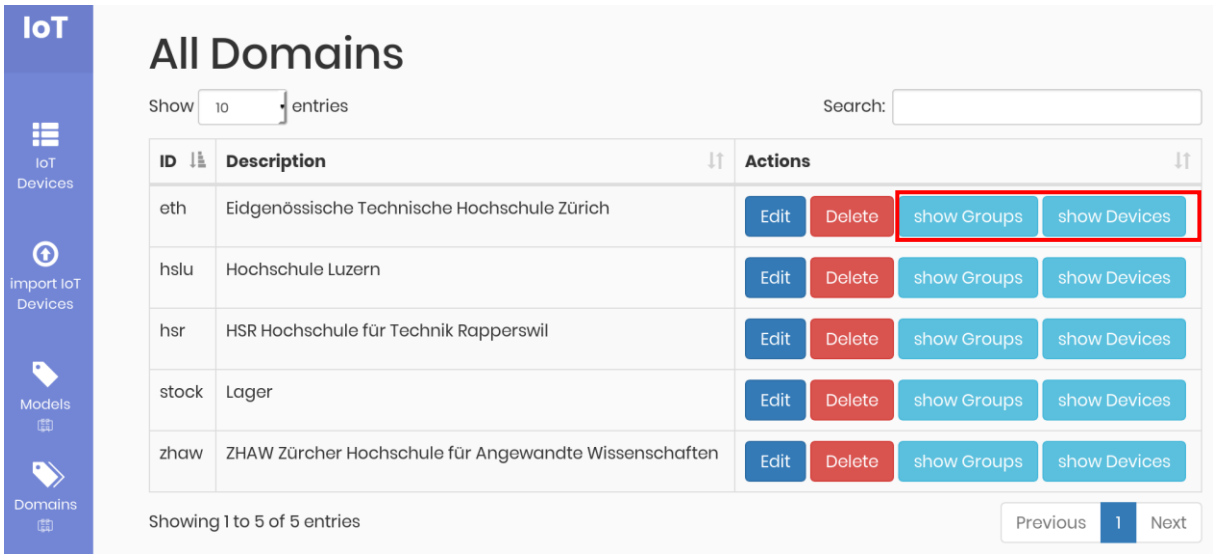


Abbildung 32 UI - All Domains

Wenn auf „show Devices“ der Domain „hslu“ geklickt wird, werden daraufhin alle IoT-Geräte von „hslu“ aufgezeigt.

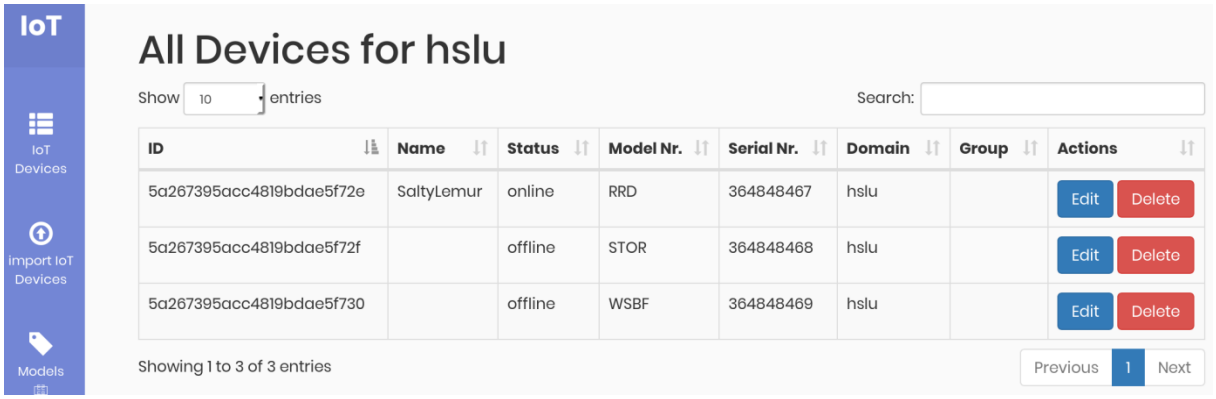


Abbildung 33 UI - All Devices for a Domain

5.7.3.8 Übersicht All Groups

Falls auf „show Groups“ der Domain „hsr“ geklickt wird, dann werden alle Groups von „hsr“ aufgelistet. Dabei stehen wieder zwei Optionen zur Verfügung. Die erste Option ist „show Devices“, damit können alle IoT-Geräte einer Group gezeigt werden. Die zweite Option ist der grüne Button in der oberen rechten Ecke. Dadurch kann eine neue Group für die ausgewählte Domain erstellt werden.

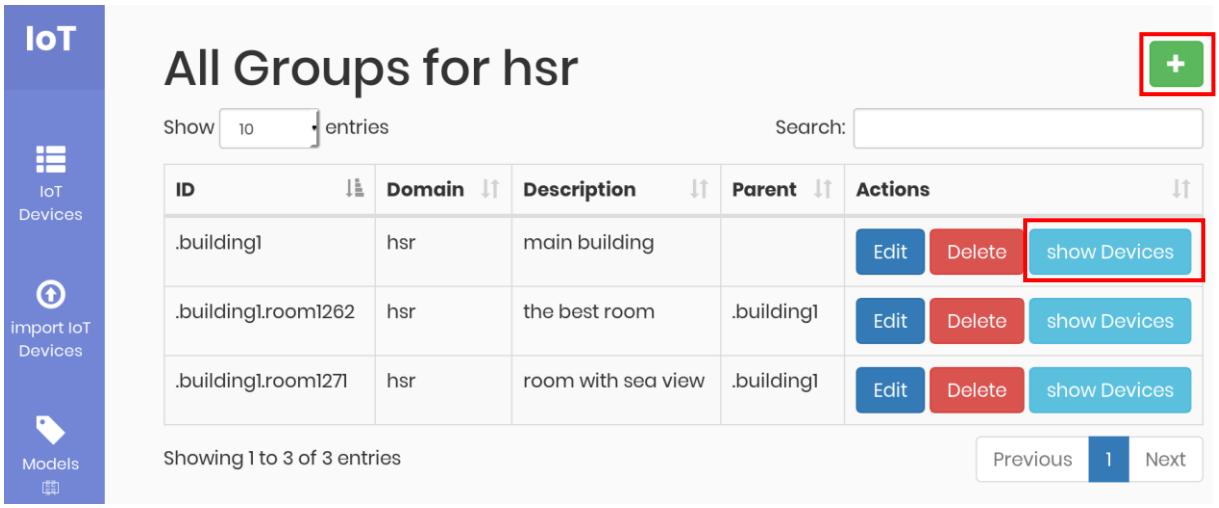


Abbildung 34 UI - All Groups

Wenn auf „show Devices“ der Group „.building1“ geklickt wird, werden daraufhin alle IoT-Geräte der Group „.building1“ und von allen Subgroups der Group „.building1“ aufgelistet.

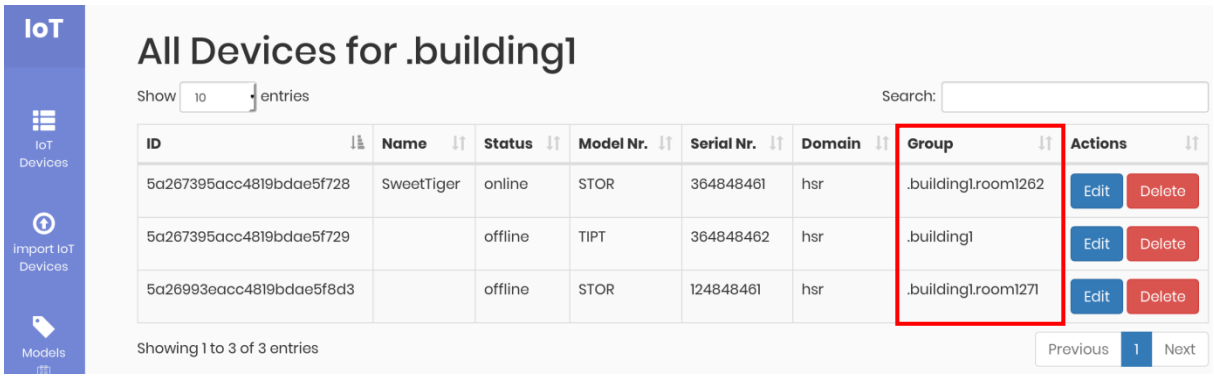


Abbildung 35 UI - All Devices for a Group

5.8 Ergebnisse

Das Verwalten von IoT-Geräten wurde in drei Aspekten eingeteilt. Diese Aspekte sind Inventar, Monitoring und Konfiguration. Da es während dem Projekt immer wieder neue Herausforderungen zu bewältigen gab, konnten am Schluss nur noch die zwei Aspekte Inventar und Monitoring vollständig implementiert werden. Das entspricht genau zwei Dritteln der Use Cases. In den folgenden Absätzen werden die zwei grössten Hürden genauer geschildert.

Am Anfang der Contruktion Phase stellte sich der Datenbanktreiber ReactiveMongo als grosse Hürde heraus, weil einerseits ReactiveMongo kaum dokumentiert ist und andereseits die vorhandene Dokumentation für Scala Experten ausgerichtet ist. Aber zum Glück gab es andere Nutzer, welche kleine Beispiele auf GitHub zur Verfügung gestellt haben. Dennoch war ReactiveMongo die richtige Entscheidung, weil dieser Treiber zu den wenigen Ausnahmen von Datenbanktreiber gehört, welcher nicht blockierend ist. [39] Dieser Punkt ist vorallem hinsichtlich der Skalierbarkeit wichtig.

Als Basis für den CoAP Endpunkt wird die Eclipse Californium Library verwendet. Wie sich gegen Ende der Construktion Phase herausgestellt hat, ist Californium, abgesehen von Kommentaren in Quellcode, gar nicht dokumentiert. Californium bietet zwar eine handvoll Beispielen auf Github an. Diese Beispiele sind ausreichend für einfache Verbindungen aber für eine DTLS-Verbindung sind die Beispiele unzureichend. Erst beim Durchlesen vom Quellcode des Eclipse Leshan Projekts auf Github, waren die ersten Schritte mit DTLS machbar. Trotz dieser Bemühungen konnte keine funktionierende DTLS Verbindung implementiert werden.

Dennoch konnte am Schluss ein funktionierender Prototyp für das Verwalten einer grossen Anzahl von IoT-Geräten realisiert werden. Das Frontend wurde mit NodeJS umgesetzt und das Backend wurde mit dem Play Framework in Scala realisiert.

Durch die Verwendung von CoAP ist es möglich verschiedenste IoT-Geräte zu verwalten und zu überwachen. Das IoT-Gerät muss einfach einen CoAP Endpunkt installiert haben und die Adresse des Backends muss als Resource Directory vorkonfiguriert sein. Beim Aufstarten trägt das IoT-Gerät seine Ressourcen dann beim Backend Server ein. Anschliessend überwacht der Backend Server das IoT-Gerät, indem das IoT-Gerät in regelmässigen Abständen ein Keepalive an das Backend sendet.

Der Administrator kann Domains & Groups erstellen. Damit lässt sich eine hierarchische Struktur definieren. Später können die IoT-Geräte an der Domain/Group zugeordnet werden. Somit wird das Verwalten für eine grosse Anzahl von IoT-Geräten stark vereinfacht. Ausserdem kann ein CSV File, welches einen Lieferschein repräsentiert, importiert werden. Dies ermöglicht ein gleichtzeitiges Erfassen von mehreren Geräten im System.

5.9 Schlussfolgerungen

5.9.1 Ergebnisbewertung

Aus zeitlichen Gründen konnten am Schluss folgende Features nicht mehr implementiert werden.

- Konfigurationstemplate erstellen
- Konfiguration spezifizieren
- Mapping zwischen unabhängiger und proprietärer Konfigurationsstruktur
- Queue für Tasks
- DTLS - Verbindung

Dank einer längeren Elaboration Phase konnte eine tiefere Einarbeitung ins Thema IoT gemacht werden, was zu einem besseren Verständnis über die Besonderheiten und Bedürfnisse von IoT-Geräten geführt hat. CoAP ein Protokoll, welches sich sehr gut für die Kommunikation mit IoT-Geräten eignet, weil es auf die Besonderheiten von IoT-Geräten eingeht. Aber die Anzahl von CoAP Libraries ist beschränkt und bei dieser beschränkten Auswahl unterstützen nur noch wenige Libraries die DTLS Verbindung. Dennoch war die Einarbeitung in CoAP wertvoll, weil in Zukunft sicherlich eine grössere Auswahl von CoAP Libraries zur Verfügung stehen werden.

Am Schluss konnte ein funktionstüchtiger Prototyp abgeliefert werden. Dieser Prototyp deckt die Aspekte Inventar und Montoring vollständig ab. Solange die Verbindung vom IoT-Gerät zum Backend initiert wird, klappt auch die NAT Traversierung. Abgesehen vom CSV File Import, sind alle Zugriffe auf dem Backend nicht-blockierend, was zugunsten der Skalierung kommt. Ausserdem konnte beim Architekturtest bestätigt werden, dass sich das Backend einwandfrei skalieren lässt. Der Architekturtest befindet sich im Anhang.

Die Programmiersprache Scala war für beide Teammitglieder eine neue Sprache. Trotzdem verlief der Einstieg in Scala problemlos, weil einerseits der Backend Entwickler die Funktionale Progammierung aus dem Modul „Programmiersprachen und formale Methoden“ kennt und er andereseits die modernen Sprachkonzepte, wie Pattern Matching von anderen Sprachen bereits kennt.

Das Play Framework war bereits aus dem Engineering Projekt bekannt, aber erst mit Scala wird das volle Potential von Play ausgeschöft. Bei der Verwendung von Futures muss man zum Beispiel am Schluss normalerweise warten, wenn man mit dem Resultat weiterarbeiten möchte. Aber in Scala kann man dank der funktionalen Programmierung elegant weiterarbeiten, ohne auf das Resultat warten zu müssen. Auf diese Weise ergibt sich eine saubere, nicht-blockiendere Ausführung.

Falls der Prototyp im produktiven Einsatz verwendet werden sollte, dann müssen folgende Punkte unbedingt beachtet werden. Zuerst muss eine DTLS Verbindung implementiert werden. Auf dem HTTP Endpunkt vom Backend ist sehr empfehlenswert TLS zu aktiviern. Das REST API vom Backend hat keine Implementation für Authorization und somit ist es gegen Fremdzugriffe ungeschützt. Deshalb muss noch ein Authorization-Verfahren implementiert werden.

5.9.2 Ausblick

Da die Zeit von Anfang an limitiert war und Integrationstests grundsätzlich aufwändiger aufzusetzen sind, wurde kein Wert auf automatisierte Integrationstests gelegt. Was sich bei einer retrospektivischen Betrachtung als eine schlechte Entscheidung herausgestellt hat. Weil einerseits durch das Mocken der Datenbank geht die effektive Überprüfung verloren, ob die Daten wirklich in die Datenbank geschrieben werden. Andereseits sind die Integrationstests langlebiger als simple Unittests. Deshalb wird empfohlen, zuerst saubere Integrationstests zu schreiben und diese Tests mit der echten Datenbankverbindung zu automatisieren.

Für eine Weiterentwicklung wird empfohlen, zuerst die ursprünglich geplanten Features noch zu implementieren und anschliessend die neuen Features zu implementieren. Daher wird das folgendene Vorgehen vorgeschlagen:

1. Queue für Tasks implementieren
2. DTLS Verbindung implementieren
3. unabhängige Konfigurationsstruktur definieren
4. Verwaltung von Konfigurationstemplate einbauen
5. Automatisiertes Mapping einer unabhängigen zu einer proprietären Konfigurationsstruktur einbauen
6. Spezifizierung für Konfiguration zu implementieren

7. IoT-Gerät offline überarbeiten
8. Android / iOS App entwicklen

Falls ein IoT-Gerät offline geht, ist dies aktuell so implementiert, dass man selber nachschauen muss. Idealerweise sollte aber das Backend den Benutzer darüber informieren. Eine einfache Option wäre, dass das Backend eine Email versendet. Eine andere Option wäre es in Form von WebSockets zu lösen. Beispielsweise ruft Benutzer Bob die Seite über das IoT-Gerät Tempy auf. Falls Tempy offline geht, dann würde das Backend die Seite automatisch aktualiseren. Auf der neugeladenen Seite wird entsprechend signalisert, dass Tempy nun offline ist. Die offengehaltene Seite beobachtet sozusagen das gewünschte IoT-Gerät und wird über alle Änderungen von dem IoT-Geräte informiert.

So dass die IoT-Geräte bequem von Smartphone verwaltet werden können, kann eine Android / iOS App entwickelt werden. Da der Backend - Server bereits eine REST API anbietet, muss diesbezüglich nichts mehr am Backend verändert werden.

5.10 Literaturverzeichnis

[1] «Gartner,» 7 2 2017. [Online]. Available: https://www.gartner.com/newsroom/id/3598917. [Zugriff am 7 12 2017].

[2] «Heise Online,» 07 12 2017. [Online]. Available: https://www.heise.de/thema/Internet-der-Dinge. [Zugriff am 07 12 2017].

[3] H. Böck, «Golem,» 26 9 2016. [Online]. Available: https://www.golem.de/news/ddos-das-internet-of-things-gefaehrdet-das-freie-netz-1609-123454.html. [Zugriff am 12 12 2017].

[4] «Slideshare,» 27 10 2017. [Online]. Available: https://de.slideshare.net/IanSkerrett/iot-developer-survey-2017.

[5] «Dzone,» 26 10 2017. [Online]. Available: https://dzone.com/articles/json-http-and-the-future-of-iot-protocols.

[6] «eclipse,» 26 10 2017. [Online]. Available: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php.

[7] «IETF,» 25 10 2017. [Online]. Available: https://tools.ietf.org/html/rfc7228#section-3.

[8] «ISO 9126,» 25 10 2017. [Online]. Available: http://bit.ly/2m8k43H.

[9] «Wikipedia,» 25 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/Datagram_Transport_Layer_Security.

[10] «bsonspec,» 25 10 2017. [Online]. Available: http://bsonspec.org/.

[11] «Wikipedia,» 27 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/Node.js.

[12] «Wikipedia,» 28 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/MIT-Lizenz.

[13] «Wikipedia,» 28 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/Express.js.

[14] «Play Framework,» Lightbend, 28 10 2017. [Online]. Available: https://www.playframework.com/.

[15] «SBT,» 25 10 2017. [Online]. Available: https://www.scala-sbt.org/1.x/docs/index.html.

[16] «Wikipedia - Scala,» [Online]. Available: https://en.wikipedia.org/wiki/Scala_(programming_language). [Zugriff am 15 10 2017].

[17] A. Alexander, «Alvin Alexander,» 10 September 2016. [Online]. Available: https://alvinalexander.com/scala/using-scala-as-a-better-java.

[18] Kapep, «Stackoverflow,» 23 September 2012. [Online]. Available: https://stackoverflow.com/questions/12552210/play-2-0java-vs-play-2-0scala.

[19] «ReactiveMongo,» 25 10 2017. [Online]. Available: http://reactivemongo.org/releases/0.12/documentation/bson/overview.html.

[20] W. Ptak, 17 03 2016. [Online]. Available: https://tech.evojam.com/2016/03/17/mongodb-scala-drivers-microbenchmark-reactivemongo-vs-scalajava-driver/.

[21] «Github,» 25 10 2017. [Online]. Available: https://github.com/okleine/nCoAP.

[22] «Eclipse,» 25 10 2017. [Online]. Available: http://www.eclipse.org/californium/.

[23] «Swagger,» 25 10 2017. [Online]. Available: https://swagger.io/.

[24] «Github,» 29 10 2017. [Online]. Available: https://github.com/swagger-api/swagger-play/tree/master/play-2.6/swagger-play2.

[25] «Apache,» 27 10 2017. [Online]. Available: https://www.apache.org/legal/resolved.html.

[26] R. Sass, «WhiteSource,» 6 August 2015. [Online]. Available: https://www.whitesourcesoftware.com/whitesource-blog/top-10-apache-license-questions-answered/?utm_content=Is-the-MIT-license-compatible-with-the-Apache&utm_medium=social&utm_source=quora&utm_term=blog-top-10-apache-license-questions-answered.

[27] M. Fowler, «Youtube - GOTO 2012,» 19 2 2013. [Online]. Available: https://www.youtube.com/watch?v=ql_g07C_Q5I.

[28] «Wikipedia,» 25 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/MongoDB.

[29] «Mongodb,» 27 10 2017. [Online]. Available: https://www.mongodb.com/de.

[30] «Wikipedia,» 25 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/Model_View_Controller.

[31] «Play Framework - API Documentation,» Lightbend, [Online]. Available: https://www.playframework.com/documentation/2.6.x/ScalaHome.

[32] «Semver,» 27 10 2017. [Online]. Available: http://semver.org/.

[33] «Wikipedia,» 27 10 2017. [Online]. Available: https://de.wikipedia.org/wiki/Docker_(Software).

[34] «Doubleslash,» 27 10 2017. [Online]. Available: https://blog.doubleslash.de/docker-swarm-container-orchestration/.

[35] Oracle, «Java Documentation,» [Online]. Available: https://docs.oracle.com/javase/8/docs/api/java/time/Instant.html#toString--. [Zugriff am 15 12 2017].

[36] Sogger, «Stackoverflow,» 14 3 2012. [Online]. Available: https://stackoverflow.com/questions/9706688/what-does-the-z-mean-in-unix-timestamp-120314170138z. [Zugriff am 15 12 2017].

[37] Z. Shelby, «RFC6690,» IETF, August 2012. [Online]. Available: https://tools.ietf.org/html/rfc6690. [Zugriff am 15 Dezember 2017].

[38] Z. Shelby, K. Hartke und C. Bormann, «RFC7252,» IETF, Juni 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252. [Zugriff am 10 Dezember 2017].

[39] «Play Framework - ThreadPools,» [Online]. Available: https://www.playframework.com/documentation/2.6.x/ThreadPools. [Zugriff am 10 12 2017].

[40] «Scala-lang,» 27 10 2017. [Online]. Available: https://docs.scala-lang.org/style/.

[41] «Github,» 27 10 2017. [Online]. Available: https://github.com/airbnb/javascript.

[42] «all-free-download.com,» 01 12 2017. [Online]. Available: http://all-free-download.com/free-vector/download/firefox-extension-html-validator-icons-clip-art_24427.html. [Zugriff am 01 12 2017].

[43] «docs.docker.com,» 15 12 2017. [Online]. Available:

https://docs.docker.com/engine/installation/. [Zugriff am 15 12 2017].

[44] «CoAP,» 27 10 2017. [Online]. Available: http://coap.technology/.

[45] «Wikimedia,» 28 10 2017. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/200px-MVC-Process.svg.png.

[46] «all-free-download.com,» 01 12 2017. [Online]. Available: http://all-free-download.com/free-vector/download/tools-icons_311023.html. [Zugriff am 01 12 2017].

5.11 Abbildungsverzeichnis

5.12 Glossar

B

BSON *Binary JSON*

C

CoAP *Constrained Application Protocol*
CPU *Central Processing Unit*

D

DTLS *Datagram Transport Layer Security*

G

GUI *Graphical User Interface*

H

HTTP *Hypertext Transfer Protocol*

I

IoT *Internet of Things*
IP *Internet Protocol*

M

MVC *Model-View-Controller*

N

NAT *Network Address Translation*

P

PAT *Port Address Translation*

R

REST *Representational State Transfer*

T

TCP *Transmission control Protocol*
TLS *Transport Layer Security*

U

UDP *User Datagram Protocol*

V

VM *Virtual Machine*

6 Anhänge

6.1 Projektplan

6.1.1 Einführung

6.1.1.1 Zweck

Dieses Dokument beschreibt den Projektplan des Projektes «IoT Management». Es beinhaltet die Planung, die Organisation sowie weitere Aspekte und liefert damit eine Übersicht über das Projekt. Dieses Dokument dient als Grundlage für den Verlauf des Projektes.

6.1.1.2 Gültigkeitsbereich

Der Gültigkeitsbereich erstreckt sich über die gesamte Dauer des Projektes. Der Zeitraum geht vom 19. September 2017 bis zum 22. Dezember 2017. Das Projekt findet im Rahmen des Moduls «Studienarbeit» im Herbstsemester 2017 statt.

6.1.2 Projekt Übersicht

IoT ist aktuell in aller Munde. Jedoch stehen Firmen oft vor dem Problem, dass IoT meistens eine grosse Masse an Geräten mit sich bringt. Standards zu diesem Thema sind noch jung und weisen noch einige Problemstellen auf.

6.1.2.1 Zweck und Ziel

In dieser Arbeit soll eine Systemarchitektur erarbeitet werden, mit Hilfe derer mehrere tausend Geräte konfiguriert und überwacht werden können. Die Kommunikation zwischen den IoT Geräten und dem Server soll über das Standardprotokoll CoAP (http://coap.technology/) geschehen.

Es wird eine Java basierende Webanwendung erwartet, die für die Verwaltung der IoT Geräte eingesetzt werden kann. Für jedes Gerät bzw. Gruppe von Geräten kann eine Konfiguration auf dem Server hinterlegt werden, welche, sobald das Gerät online verfügbar ist, auf dem Device appliziert wird. Weiter können Geräte überwacht und in einem bestimmten Zyklus abgefragt werden. Die Authentisierung der verbundenen Geräte geschieht über Zertifikate, die während dem Bootstrapping-Prozess automatisch enrollt werden (https://tools.ietf.org/html/draft-ietf-acme-acme-07).

6.1.2.2 Annahmen und Einschränkungen

Das Projekt ist auf die Dauer des Herbstsemester 2017 begrenzt (bis 22. Dezember 2017). Pro Projektmitglied soll das Projekt mit 240 Arbeitsstunden (gesamthaft 480 Stunden) realisiert werden.

6.1.3 Projektorganisation

In unserem Projekt arbeiten wir in einer flachen Organisationsstruktur, wobei die wesentlichen Entscheide, an den wöchentlichen Meetings, im ganzen Projektteam und/oder mit dem Dozenten / Businesspartner getroffen werden. An den Meetings getroffene Entscheidungen werden in Protokollen dokumentiert. Die Projektmitglieder sind innerhalb des Teams gleichgestellt.

6.1.3.1 Organisationsstruktur

Arooran Thanabalasingam
Developer

Elias Brunner
Developer

6.1.3.2 Ansprechpartner

Im Projekt „IoT Management“ sind folgende Ansprechpartner vorhanden.

Name	Funktion
Beat Stettler	Betreuung und Bewertung des Projektes
Michael Schneider	Businesspartner

6.1.4 Management Abläufe

6.1.4.1 Aufwandsberechnung

Das Projekt IoT Management wurde zu Beginn des Herbstsemesters am Dienstag, 19. September gestartet. Es endet mit dem Ende des Herbstsemesters, welches am Freitag, dem 22. Dezember ist. Somit ergeben sich folgende Kennzahlen.

Start	19. September 2017
Ende	22. Dezember 2017
Teamgrösse	2 Personen
Zu leistende Stunden pro Person	240 Stunden
Stunden gesamt	480 Stunden

Die Planung ist darauf ausgelegt, dass das Projekt innerhalb der 480 Stunden zu bewältigen ist. Bleibt am Ende Zeit über, werden optionale Features implementiert und als zusätzliche Funktionalität ergänzt.

6.1.4.2 Zeitliche Planung

Eine ausführliche Iterationsplanung mit den dazugehörigen Milestones befindet sich auf GitLab unter Issues/Milestones[1]. Die aufgewendeten Zeiten für ein Issue werden auf toggl.com[2] aufgeschrieben. Da wir mit einer agilen Entwicklung arbeiten, wird die Planung während dem Projekt laufend aktualisiert und den aktuellen Gegebenheiten angepasst. Hier werden die Iterationen und Meilensteine dennoch aufgelistet wobei diese in chronologischer Reihenfolge geordnet sind.

6.1.4.3 Übersicht der Meilensteine im Detail

KW	SW	Iteration	Meilenstein
38	1	Inception 26. September 2017	MS1 - Projektplan eingereicht Projektplan erstellt und eingereicht, Datenablage erstellt
39	2	E1 10. Oktober 2017	MS2 - Anforderungen, Domainanalyse und Use Cases fertig Alle relevanten Anforderungen wurden zusammengetragen. Zudem wurden die Use Cases in einer fully-dressed Form geschrieben und abgegeben
40	3		
41	4	E2 24. Oktober 2017	MS3 – Konzept fertig Anhand der Anforderungen der Domiananalyse und der Use Cases wurde ein Konzept erarbeitet.
42	5		
43	6	E3 07. November 2017	MS4 – Systemarchitektur und Domain Modell fertig Anhand der gesammelten Anforderungen und Überlegungen wurde eine Systemarchitektur und ein Domain Modell erarbeitet.
44	7		
45	8	C1 21. November 2017	MS5 - Prototyp erstellt Bestellungsprozess funktioniert.
46	9		
47	10	C2 05. Dezember 2017	MS6 - Betriebsprozess funktioniert IoT-Geräte senden keep-alives. Der Status des IoT-Gerätes kann ermittelt werden.
48	11		
49	12	C3 12. Dezember 2017	MS7 - Applikation fertig Die Applikation wurde einem End-to-End Test unterzogen und funktioniert. Erster Entwurf für das

[1] https://gitlab.com/iotmgmt
[2] https://www.toggl.com/app/timer

			Enddokument kann gezeigt werden.
50	13	Transition 19. Dezember 2017 22. Dezember 2017	MS8 - Enddokument abgegeben Abstract und alle notwendigen Dokumente abgegeben

6.1.4.4 Besprechungen

Einmal in der Woche trifft sich das ganze Projektteam miteinander um sich über den aktuellsten Stand des Projekts auszutauschen, Fragen zu klären, Probleme anzugehen und die nächsten Schritte zu definieren. Diese wöchentlichen Meetings finden jeden Dienstagnachmittag jeweils um 15:00 Uhr statt.

Während diesen Meetings wird zum Teil auch unser Businesspartner vertreten sein, um unsere Fortschritte zu sehen. Über jede Besprechung wird Protokoll geführt. Dies mit dem Ziel, die Entscheidungen festzuhalten und Missverständnisse zu vermeiden. Die Protokolle werden in der gemeinsamen Dateiablage gespeichert.

6.1.5 Risikomanagement

Projekt:		**IoT Management**					
Erstellt am:		**09.11.2017**					
Autor:		**Elias Brunner**					
Gewichteter Schaden:		**61.5**					
Nr	**Titel**	**Beschreibung**	**max. Schaden [h]**	**Eintrittswahrscheinlichkeit**	**Gewichteter Schaden**	**Vorbeugung**	**Verhalten beim Eintreten**
R1	Anforderungen	Stark wechselnde Anforderungen	20	30%	6	Genaue Definition während der Elaboration und gute Planung der Iterationen	Änderungen einschätzen, überprüfen und eventuell vornehmen
R2	Managment Tool	Arbeiten mit GitLab zeitaufwändiger als erwartet da noch nie verwendet	20	5%	1	Früh genug mit GitLab vertraut machen	Zusätzliche interne Schulung und üben
R3	Kommunikation	Gegenseitige Kommunikation innerhalb des Projekteteams oder zwischen Dozent/Businesspartner ist schwierig	15	30%	4.5	Frühzeitig abklären und genau mitteilen, wer was übernimmt	Zusätzliche Meetings um Ungenauigkeiten abzuklären
R4	Komplexität	Die gewünschte Funktionalität des Projekt ist zu hoch und schwierig zu erreichen	50	40%	20	Genaue Abschätzung der geplanten Funktionalität und der benötigten Zeit	Funktionalitätsumfang anpassen und verringern
R5	Systemdesign	Gewählten Tools für das Systemdesign entsprechen nicht den Vorstellungen	10	10%	1	Über die verwendeten Tools informieren, Tools verwenden die bekannt sind	Zusätzliche interne Schulung
R6	Entwicklungsumgebung	Kompatibilität der Entwicklungsumgebung oder Kentnisse derselben nicht aussreichend	40	5%	2	Auswahl der Umgebung für alle Mitglieder in Ordnung	Aushilfe bei Problemen, zusätzlich informieren
R7	Qualität	Code Guidelines, Qualitätsmanagment werden nicht eingehalten	20	10%	2	Guidelines einhalten, Kontrolle bei Code-Reviews	Mehr Reviews und Gespräch mit Entwicklern
R8	Dokumentation	Erstellte Arbeiten werden nicht gut genug dokumentiert oder Detailierungsgrad ist nicht ausreichend	20	30%	6	Alles verständlich Dokumentieren und kommunizieren	Sensibilisierung der Mitglieder für Dokumentation
R9	Zusammenführen der Komponente	Einzelne Komponenten des Projekts sind nicht kompatibel	10	10%	1	Genaue Planung der Arbeit, logische Abfolge der Entwicklung	Zusätzliche informieren über Kompatibilität
R10	Know-How	Fehlendes Know-How in den gewählten Programmiersprachen oder der eingesetzten Technologien	30	30%	9	Know-How vertiefen über verwendete Sprachen und Umgebungen	Know-How aufbauen
R11	Zeitlicher Aufwand	Interne Aufwandschätzung für einzelne Arbeitspakete sind zu optimistisch gerechnet	30	30%	9	Genauere Aufwandschätzung während dem agilen Vorgehen (Construction) im Projekt	Funktionalitätsumfang anpassen und verringern
Summe			**265**		**61.5**		

6.1.5.1 Umgang mit Risiken

Um auch auf unbekannte Risiken vorbereitet zu sein, wurde am Ende des Projektes eine Reserve eingeplant. Zudem haben Sie alle Teilnehmer bereit erklärt ihr Engagement punktuell zu erhöhen, falls die Situation dies erfordern würde. Diese Erhöhung sollte jedoch nur Phasenweise sein und in einer folgenden Phase kompensiert werden.

6.1.6 Arbeitspakete

Die Arbeitspakete werden direkt in GitLab verwaltet. Dies macht es möglich die Pakete später zu verfeinern, verschieben, neu zu schätzen und Iterationen zuzuordnen. Unten werden nur die wichtigsten Arbeitspakete aufgelistet und repräsentieren bei weitem nicht die gesamte Anzahl an erledigten Arbeitspaketen während der gesamten Durchführung der Arbeit.

Allerdings ist an dieser Stelle anzumerken, dass die Zeiterfassung extern auf toogl.com[3] gemacht wurde. Die genaue Auswertung der aufgewendeten Zeiten für die einzelnen Arbeitspakete stellte sich dabei als sehr schwierig heraus. Der Grund dafür liegt in den unterschiedlich benannten Arbeiten für die Arbeitspakete.

Die Zugriffsinformationen für die Arbeitspakete sind hier notiert.

Allgemein	https://gitlab.com/iotmgmt/docs/boards
Konstruktion	https://gitlab.com/iotmgmt/rest-api/boards https://gitlab.com/iotmgmt/web/boards https://gitlab.com/iotmgmt/iot-device

Die Zeiterfassung auf den Arbeitspaketen erfolgt in folgenden Labels. Andernfalls wird die Zeitauswertung zu detailliert und verliert damit an Mehrwert.

- Anforderungen
- Design
- Programmierung
- Testen & QS
- Projekt-Management
- Infrastruktur & Deployment
- Einarbeitung in Technologien
- Dokumentation
- Bugs

[3] https://www.toggl.com/app/timer

Phase	Name	Meilenstein	Soll [h]	Ist [h]
Inception	Projektplan erstellen	MS1	8	10
Elaboration	Anforderungsspezifikation erstellen	MS2	16	30
	Technologieanalyse erstellen	MS2	8	8
	Domainanalyse erstellen	MS2	16	20
	Konzept erstellen	MS3	32	36
	Systemarchitektur erstellen	MS4	32	40
Construction	Prototyp erstellen	MS5	16	26
	Domain/Group verwalten	MS5	16	26
	Modell verwalten	MS5	16	25
	Ressource eintragen	MS5	16	2
	importiert Lieferschein	MS5	16	24
	ordnet IoT-Geräte Domain/Group/Modell zu	MS5	16	25
	keep alive senden	MS6	16	5
	Geräte offline	MS6	16	10
	Testdokumentation erstellen	MS7	16	3
Transition	Finales Dokument fertigstellen	MS8	32	25

6.1.7 Zeitauswertung

Für die Studienarbeit, mit ihren 8 ECTS Punkten, wird mit einem zeitlichen Aufwand von 240 Stunden pro Teammitglied gerechnet. Pro ECTS Punkt sind also mindestens 30 Stunden zu leisten. Daraus ergibt sich ein Gesamtaufwand von min. 480 Stunden für das gesamte Projektteam.

Wie in der Endabrechnung zu sehen ist, wurde der Zeitaufwand leicht überschritten. Insgesamt wurden 544 Stunden für das Projekt „IoT Management“ aufgewendet.

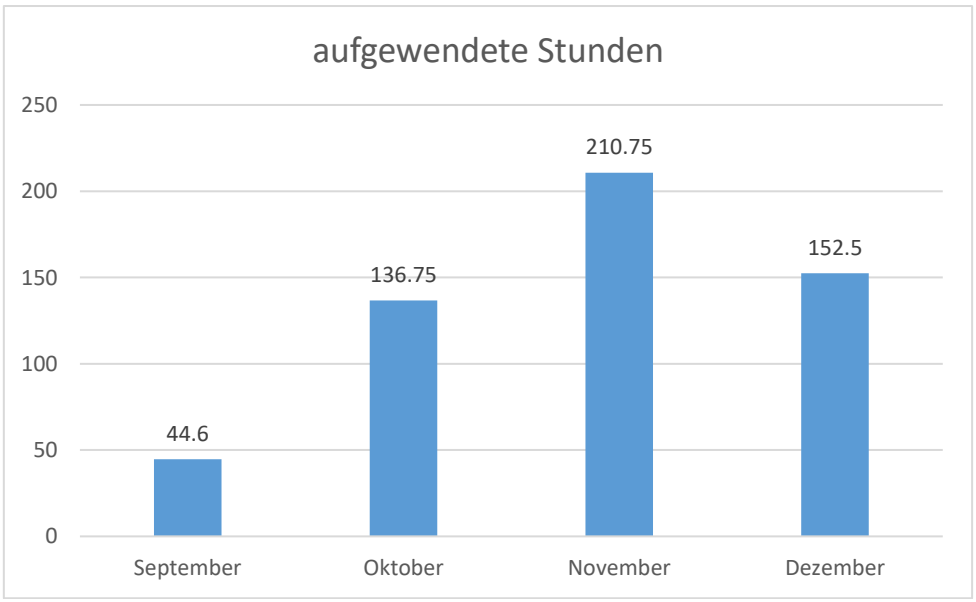


Abbildung 36 aufgewendete Stunden pro Monat

Klar zu erkennen ist allerdings, dass im Monat November am meisten Zeit investiert wurde. In diesem Monat wurde vorallem Zeit für die Implementierung investiert, welche sich als zeitintensiver herausstellte als zunächst gedacht.

Die Auswertung anhand der verwendeten Labels macht diesen Umstand noch deutlicher. Mit 173.5 Stunden macht die Programmierung über ein Drittel aller aufgewendeten Stunden aus. Auch die 134.5 Stunden, welche für die Anforderungen bzw. jeglicher Analysen benötigt wurden, schlagen mit fast 30% zu buche.

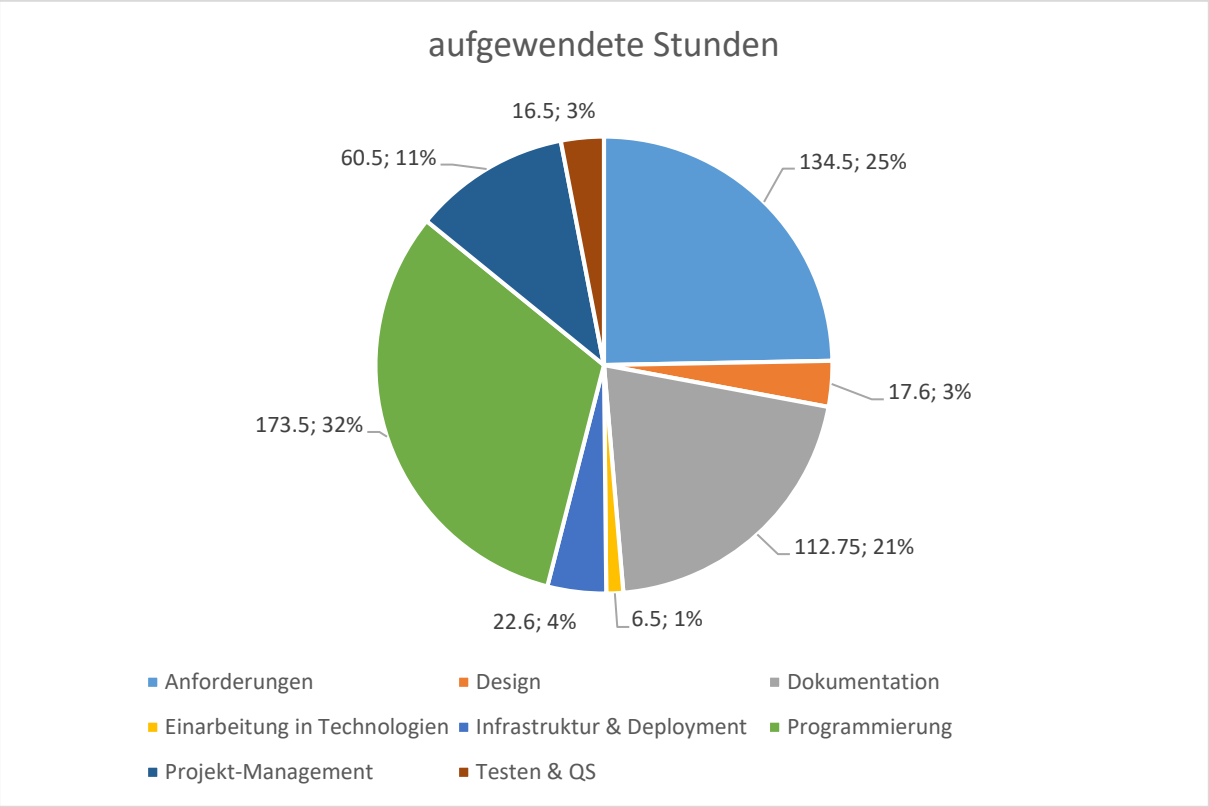


Abbildung 37 aufgewendete Stunden pro Label

6.1.8 Infrastruktur

Alle Mitglieder des Projekts arbeiten mit ihren eigenen Geräten. Diese erstrecken sich von Fedora über Windows bis hin zu MacOS. Neben der Hauptarbeit werden wir die persönlichen Laptops auch dazu nutzen, das Web GUI der Applikation zu testen.

Die Entwicklungsumgebungen wurden so gewählt, dass Sie auf jedem Betriebssystem (Mac, Linux, Windows) eingesetzt werden können. Die Entwicklungswerkzeuge unterscheiden sich je nach Entwicklungsbereich. Nachstehende Tabelle gibt Auskunft über die verschiedenen Bereiche.

Entwicklungsbereich	Eingesetzte Werkzeuge
Backend	IntelliJ IDEA[4]
Web-GUI	WebStorm von JetBrains[5]

Zur Verwaltung des Codes nutzen wir öffentliche GitLab-Repositories, die Dokumente werden in Google Drive verwaltet und die Kommunikation abseits der HSR Anwesenheit erfolgt über einen Whats-App Chat.

6.1.9 Qualitätsmassnahmen

Das Endprodukt dieses Projektes soll von möglichst hoher Qualität sein. Es wurden folgende Massnahmen getroffen um diese Qualität auch zu erreichen.

Massnahme	Zeitraum	Ziel
Meeting im Team und mit Betreuer	Jede Woche	Projektstand aufzeigen, allfällige Probleme möglichst früh erkennen.
Code Reviews	Bei jedem Pull Request	Die Qualität des Codes wird durch die Einhaltung der Code Style Guidelines verbessert.

6.1.9.1 Dokumentation

Die gesamten Dokumente des Projektes werden in einem gemeinsamen Google Drive Ordner abgelegt. Dort drin befinden sich jederzeit die aktuellsten Versionen der Dokumente, auch die Autoren oder Editoren der jeweiligen Dokumente sind anhand der Versionierung ersichtlich.

Vor jedem Meilenstein werden die Dokumente einer Review unterzogen, um allfällige Fehler oder fehlende Daten zu finden und zu beheben. Jedes Teammitglied hat nach jeder Änderung die Änderungsgeschichte im Dokument nachzuführen, so dass verfolgt werden kann, wer was wann angepasst hat.

[4] https://www.jetbrains.com/idea/
[5] https://www.jetbrains.com/webstorm/?fromMenu

6.1.9.2 Projektmanagement

Als Projektmanagementsystem kommt auch hier GitLab zum Einsatz. Darin werden alle Issues und Arbeitspakete erfasst und die aufgewendete Zeit pro Vorgang darauf verbucht. Zu jederzeit sollte das GitLab den aktuellen Stand des Projektes widerspiegeln können.

Die GitLab-Instanz kann über https://gitlab.com/iotmgmt erreicht werden. Jedes Teammitglied hat seinen persönlichen Zugang. Die Repositories sind öffentlich. Daher wird kein Login benötigt.

6.1.9.3 Entwicklung

Der Entwicklungscode wird in öffentlichen GitLab-Repositories unter unserer Gruppe https://gitlab.com/iotmgmt gehalten. Für alle einzelnen Teile des Projekts gibt es ein eigenes Repository.

Reponame	Link
docs	https://gitlab.com/iotmgmt/docs
rest-api	https://gitlab.com/iotmgmt/rest-api
web	https://gitlab.com/iotmgmt/web
iot-device	https://gitlab.com/iotmgmt/iot-device

6.1.9.4 Vorgehen

Jedes Teammitglied verfügt über eine lokale Kopie der Repositories von GitLab. Für jede Aufgabe/Issue von GitLab wird ein eigener Branch erstellt, unter Angabe der Nummer. Darin werden die Änderungen für diese Aufgabe vorgenommen. Die Änderungen sollen mit sinnvollen und präzisen Commit-Notizen festgehalten werden. Um ein Tracking der Änderung möglichst effizient zu gestalten, gilt es möglichst früh, möglichst viel zu commiten, sowie diese sinnvoll zu benennen.

6.1.9.5 Unit Testing

Grundsätzlich sollten Unit Tests für folgende Komponenten erstellt werden.

- Services
- Businesslogic
- Basisfunktionalitäten

6.1.9.6 Code Style Guidelines

Für den Scala-Code wird die Styleguide von Scala selbst verwendet [40]. Beim NodeJS kommt der Styleguide von AirBNB zum Einsatz [41].

6.1.9.7 Komponententests

Zu den Klassen in der «Businesslogic» werden Unit Tests geschrieben. Diese werden vor jedem Commit durchgeführt. Der Code darf nur commited werden, wenn alle Tests grün sind.

6.1.9.8 Architekturtest & Systemtest

Am Ende der Projektarbeit wird ein umfangreicher Systemtest durchgeführt. Nach der Durchführung aller Tests werden noch vorhandene Fehler behoben. Da es kaum möglich sein wird, hunderte IoT-Geräte physisch zum Testen zur Verfügung zu haben, wird hier auf Docker mit IoT-Geräten als Docker Containern ausgewichen. Das Testing findet kontrolliert mit Testabläufen und entsprechenden Protokollen statt.

6.2 Testspezifikation

6.2.1 Voraussetzungen und Vorbereitungen

Für die Durchführung des Tests muss das komplette System “IoT Management“ installiert sein.

Es wird vorausgesetzt, dass die aktuellsten Versionen des Codes vorhanden, installiert und konfiguriert sind. Die Schritte dazu sind in der Installationsanleitung zu finden.

6.2.1.1 Legende der Icons

ID	Beschreibung	Icon
1	Bedingung erfüllt	 *Abbildung 38 OK [42]*
2	Bedingung generell erfüllt aber z.B. Benutzerführung ungeeignet (Fehlermeldung vorhanden aber zu generisch) oder Teilaspekte nicht ganz erfüllt	 *Abbildung 39 Achtung [42]*
3	Bedingung nicht erfüllt	 *Abbildung 40 NOK [42]*

6.2.2 Architekturtest

Der Architekturtest soll vor allem dazu dienen, die Teile der IoT Management Applikation zu testen, welche mit Unit Tests oder mit dem Systemtest nicht gut oder gar nicht getestet werden können.

Dazu zählt insbesondere der Teil der Kommunikation der IoT-Geräte zur Applikation. Da die IoT-Geräte später nicht im selben Netz wie die IoT Management Applikation sein werden, macht es keinen Sinn diese z.B. mit Fake-Objekten zu faken und so die Testfälle der Use Cases, welche weiter unten beschrieben werden, durchzuführen.

Viel besser ist es daher, wenn ein Test unter möglichst realen Bedingungen gemacht wird. Zu diesem Zweck, wird ein externer Server ausserhalb des Netzes, in welchem sich der IoT Management Server befindet, aufgestellt. Auch hier wird ein Ubuntu mit Docker aufgesetzt. Mit Hilfe von Docker Swarm und dem Compose File kann eine Vielzahl von Containern, welche die IoT-Geräte simulieren sollen, auf einfache Art und Weise erstellt werden. Mit dieser Strategie ist eine möglichst gute Annäherung an die Realität gewährleistet ohne dass hunderte von realen IoT-Geräten verwendet werden müssen.

Neben dem Aspekt der NAT Problematik, soll in diesem Test daher auch untersucht werden, wie sich unsere Applikation mit einer möglichst grossen Anzahl an IoT-Geräten verhält.

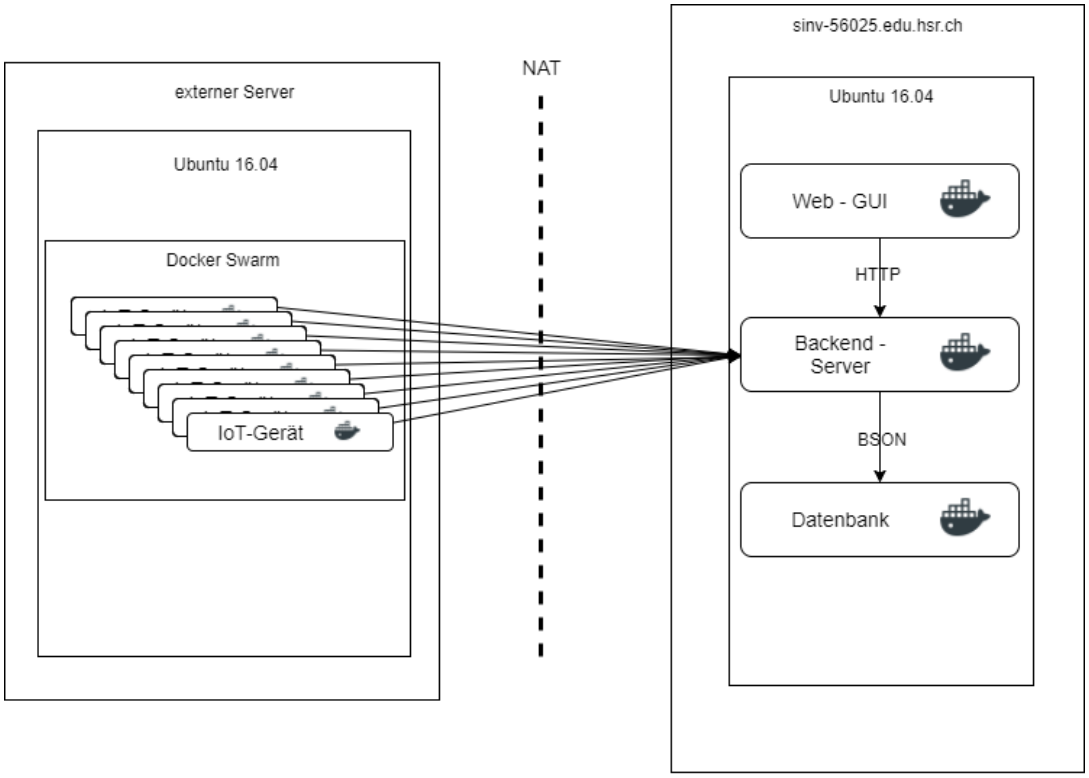


Abbildung 41 Architekturtest Setup

6.2.2.1 Angaben zum Server

Die Applikation wird unter folgenden Bedingungen auf dem Server der HSR (sinv-56025.edu.hsr.ch) getestet:

Betriebssystem	Ubuntu 16.04
CPU	1 Core max. 2.2 GHz
RAM	1 GB
HD	15 GB
Keep-alive Intervall	5 Sek

6.2.2.2 Testfälle

Der Architekturtest wurde auf der aktuellsten Version des Systems am 07.12.2017 durchgeführt.

Test ID	Anzahl IoT Geräte	Auslastung CPU Backend – Server [%]	Auslastung RAM Backend – Server [MB]	Erfüllt
1	1	0-1	215	
2	25	1-8	210	
3	100	3-10	202	
4	200	4-12	212	
5	500	5-12	233	
6	1000	Konnte nicht geprüft werden	Konnte nicht geprüft werden	
7	2000	Konnte nicht geprüft werden	Konnte nicht geprüft werden	

6.2.2.3 Ergebnisse

Mit dem Architekturtest konnte einerseits gezeigt werden, dass sich die IoT-Geräte beim Backend – Server melden können, auch wenn sich dieser nicht im selben Netz wie die IoT-Geräte befinden. Dies wurde mit dem ersten Testfall mit nur einem IoT-Gerät getestet und beweist somit, dass die NAT Problematik erfolgreich bewältigt werden konnte.

Andererseits konnte mit dem Architekturtest auch gezeigt werden, dass der Backend – Server bei einer Anzahl von 500 IoT-Geräten nicht übermässig ausgelastet ist. Die angegebenen Zahlen in der Tabelle wiederspiegeln die absoluten Höchstwerte. Die meiste Zeit schwankte die Auslastung der CPU eher im mittleren Bereich.

Da allerdings nur ein externer Server für das Simulieren der IoT-Geräte zur Verfügung stand, konnten nicht mehr als 500 Geräte simuliert werden, da dieser bei dieser Anzahl an Docker-Containern an seine Leistungsgrenzen kam.

Mit den vorliegenden Testresultaten, kann jedoch davon ausgegangen werden, dass der Backend – Server mit grosser Wahrscheinlichkeit 3000-5000 IoT-Geräte bewältigen kann. Wenn man dann noch bedenkt, dass der Backend – Server während der Testphase auf einem virtuellen Server mit nur einem Prozesserkern lief und die IoT-Geräte alle 5 Sekunden ein keep-alive Signal sendeten, könnte mit einem entsprechend stärkeren Ser-

ver und längeren keep-alive Abständen wahrscheinlich noch bessere Resultate erzielt werden.

Bei diesem Test wurden gute Ergebnisse erzielt und es konnte ein erster Eindruck für die Skalierbarkeit der Applikation gewonnen werden.

6.2.3 Systemtest

Der Systemtest orientiert sich grundsätzlich an den verschiedenen Use Cases, welche in der Anforderungsspezifikation definiert wurden. Nachfolgend sind die einzelnen Testfälle beschrieben. Der Systemtest selbst wurde allerdings in einer chronologisch logischen Abfolge durchgeführt. Daher befinden sich die Testfälle nicht in derselben Reihenfolge wie die Use Cases in der Anforderungsspezifikation.

An dieser Stelle ist allerdings noch zu vermerken, dass bei der Durchführung des Systemtest die NAT Problematik nicht berücksichtigt wurde. Dies wurde mit dem Architekturtest getestet. Der Systemtest wurde auf der aktuellsten Version des Systems am 01.12.2017 durchgeführt.

6.2.3.1 UC-9 Domain/Group verwalten

ID	Beschreibung	Erfüllt
9.1	Der Administrator erstellt eine neue Domain, indem er auf „New Domain“ klickt, eine ID und Beschreibung eingibt und auf „Submit“ klickt	
9.2	Die neu erstellte Domain erscheint in der Liste aller Domains	
9.3	Die Domain kann mit einem Klick auf „Edit“ editiert werden	
9.4	Die Domain kann mit einem Klick auf „Delete“ gelöscht werden	
9.5	Der Administrator erstellt eine neue Group, indem er bei einer Domain auf „show Groups“ klickt und anschliessend auf das grüne Plus oben rechts.	
9.6	Er gibt anschliessend eine ID, die übergeordnete Domain, eine Beschreibung und allenfalls die übergeordnete Parent-Group ein. Ist dies erledigt, klickt er auf „Submit“	
9.7	Die neu erstellte Group erscheint in der Liste aller Groups dieser Domain	
9.8	Die Group kann mit einem Klick auf „Edit“ editiert werden	
9.9	Die Group kann mit einem Klick auf „Delete“ gelöscht werden	

9.10	Sowohl die Domains wie auch die Groups können durchsucht werden	

6.2.3.1.1 Bemerkungen

ID	Beschreibung
9.6	Es kann keine Group mit derselben ID zweimal erstellt werden auch wenn sich diese nicht unter der gleichen Domain befinden.
9.8	Der Back Button führt direkt wieder zu allen Domains. Schön wäre es, wenn er wieder zur entsprechenden Domain führt.
9.9	Der Back Button führt direkt wieder zu allen Domains. Schön wäre es, wenn er wieder zur entsprechenden Domain führt.

6.2.3.2 UC-10 Modell verwalten

ID	Beschreibung	Erfüllt
10.1	Der Administrator erstellt ein neues Modell, indem er auf „New Model“ klickt, eine ID, eine Beschreibung und eine Category eingibt und auf „Submit“ klickt	
10.2	Das neu erstellte Model erscheint in der Liste aller Models	
10.3	Das Modell kann mit einem Klick auf „Edit“ editiert werden	
10.4	Das Modell kann mit einem Klick auf „Delete“ gelöscht werden	
10.5	Alle Modelle können durchsucht werden	

6.2.3.2.1 Bemerkungen

ID	Beschreibung

6.2.3.3 UC-11 Import Lieferschein

ID	Beschreibung	Erfüllt
11.1	Der Administrator wählt den Menüpunkt „import IoT Devices“ aus	
11.2	Der Administrator wählt den gewünschten Lieferschein aus und klickt anschliessend auf „Submit“	
11.3	Die IoT-Geräte werden importiert und in der Liste aller Geräte angezeigt	
11.4	Die IoT-Geräte werden der Domain „Stock“ zugeordnet	

6.2.3.3.1 Bemerkungen

ID	Beschreibung
11.3	Die Weiterleitung zur Liste aller Geräte funktioniert nur teilweise.

6.2.3.4 UC-12 Ordnet IoT-Geräte Domain / Group / Modell zu

ID	Beschreibung	Erfüllt
12.1	Der Administrator wählt den Menüpunkt „IoT Devices“ aus und ihm werden alle IoT-Geräte angezeigt	
12.2	Mit einem Klick auf „Edit“ kann ein IoT-Gerät editiert werden	
12.3	Der Administrator von CloudGuard kann die gewünschte Domain eintragen	
12.4	Der Administrator von PostAuto Schweiz kann für das Gerät einen Namen, Adresse, Group, Model Nr. und Max Age vergeben	

6.2.3.4.1 Bemerkungen

ID	Beschreibung
12.4	Die Zuweisung eines Modells ist nicht notwendig, da dies beim Import anhand des Lieferscheins zugeordnet wurde
12.4	Die Max Age hat noch keinen Einfluss auf die tatsächliche Max Age von CoAP. Kann aber bei einer Weiterentwicklung dazu verwendet werden.

6.2.3.5 UC-8 Ressource eintragen

ID	Beschreibung	Erfüllt
8.1	Das IoT-Gerät trägt seine Ressourcen beim Server unter dem beim Import erstellten Device ein	

6.2.3.5.1 Bemerkungen

ID	Beschreibung

6.2.3.6 UC-2 keep-alive senden

ID	Beschreibung	Erfüllt
2.1	Die Heartbeat Ressourcen eines IoT-Gerätes senden in einem bestimmten Intervall ihre Werte an den Server	
2.2	Der Server erhält das keep-alive Signal	
2.3	Der Server trägt sich die übertragene Zeitangabe ein	

6.2.3.6.1 Bemerkungen

ID	Beschreibung

6.2.3.7 UC-1 Ressource auslesen

ID	Beschreibung	Erfüllt
1.1	Alle Ressourcen eines IoT-Gerätes senden in einem bestimmten Intervall ihre Werte an den Server	
1.2	Der Server trägt sich den letzten bekannten Wert in die Datenbank ein	
1.3	Der Administrator wählt das gewünschte IoT-Gerät aus der Liste aller Geräte aus	
1.4	Der Server zeigt den letzten bekannten Wert unter der entsprechenden Ressource an	

6.2.3.7.1 Bemerkungen

ID	Beschreibung
1.1	Die Max Age, welche im Web GUI eingetragen werden kann, hat noch keinen Einfluss auf die tatsächliche Max Age von CoAP. Kann aber bei einer Weiterentwicklung dazu verwendet werden.

6.2.3.8 UC-3 Gerät offline

ID	Beschreibung	Erfüllt
3.1	Der Administrator wählt das gewünschte IoT-Gerät aus	
3.2	Falls das Gerät online ist, wird ein grüner Balken angezeigt, welcher signalisiert, dass das Device online ist	
3.3	Falls das Gerät offline ist, wird ein roter Balken angezeigt, welcher signalisiert, dass das Device offline ist	
3.4	In der Liste aller Geräte wird der Status aller Geräte aufgelistet	

6.2.3.8.1 Bemerkungen

ID	Beschreibung
3.2, 3.3	Die Aktualisierung kann bis zu 20 Sek. dauern

6.2.3.9 UC-4 Konfigurationstemplate erstellen

ID	Beschreibung	Erfüllt
4.1	Der Administrator erstellt ein Konfigurationstemplate für ein gewisses Modell	

6.2.3.9.1 Bemerkungen

ID	Beschreibung

6.2.3.10 UC-5 Konfiguration spezifizieren

ID	Beschreibung	Erfüllt
5.1	Der Administrator wählt die gewünschten IoT-Geräte aus, auf der eine Änderung der Konfiguration vorgenommen werden soll.	
5.2	Der Server zeigt mögliches Template an	
5.3	Der Administrator gibt einen Range für die Variablen z.B. für IP-Adressen an	
5.4	Der Server erstellt für die ausgewählten Geräte die Konfigurationsdatei	
5.5	Der Server hinterlegt für jedes der ausgewählten IoT-Geräte einen entsprechenden Task	

6.2.3.10.1 Bemerkungen

ID	Beschreibung

6.2.3.11 UC-6 Konfiguration einspielen

ID	Beschreibung	Erfüllt
6.1	Der Administrator hinterlegt den neuen Task “Konfiguration einspielen” auf dem Server	
6.2	Da sich das IoT-Gerät in regelmässigen Abständen meldet, schaut der Server nach, ob es den Tasks „Konfiguration einspielen“ für dieses IoT-Gerät zum Abarbeiten gibt	
6.3	Ist dies der Fall, liefert der Server die Konfigurationsdatei mit dem Request “Konfiguration einspielen”.	
6.4	Das IoT-Gerät antwortet mit einer Acknowledge Response	
6.5	Das IoT-Gerät spielt sich die Konfiguration ein	

6.2.3.11.1 Bemerkungen

ID	Beschreibung

6.2.3.12 UC-7 Firmware einspielen

ID	Beschreibung	Erfüllt
6.1	Der Administrator hinterlegt den neuen Task “Firmware einspielen” auf dem Server	
6.2	Da sich das IoT-Gerät in regelmässigen Abständen meldet, schaut der Server nach, ob es den Tasks „Firmware einspielen“ für dieses IoT-Gerät zum Abarbeiten gibt	
6.3	Ist dies der Fall, liefert der Server die Firmware mit dem Request “Firmware einspielen”.	
6.4	Das IoT-Gerät antwortet mit einer Acknowledge Response	
6.5	Das IoT-Gerät spielt sich die Firmware ein	

6.2.3.12.1 Bemerkungen

ID	Beschreibung

6.3 Installationsanleitung

6.3.1 Installation von Docker

Um das IoT Management System zu installieren, muss zunächst Docker auf ihrem PC oder Server installiert sein. Die dafür notwendigen Schritte sind der Anleitung von Docker selbst zu entnehmen [43].

Die Installation des IoT Management Systems kann anschliessen auf zwei Arten gemacht werden. Es wird empfohlen, die automatische Installation mittels dem docker-compose File durchzuführen. Alternativ kann aber auch die manuelle Installation durchlaufen werden.

6.3.2 Automatische Installation

1. *sa_db:*
   volumes:
   /to/my/folder:data/db:Z

Ändern Sie im docker-compose.yaml File unter sa_db -> volumes den Pfad auf ein Verzeichnis Ihrer Wahl. Es ist lediglich der Teil vor dem ersten Doppelpunkt zu ändern.

1. *docker-compose up*

Startet das IoT Management System

2. *docker run -e IOT_DEV_PORT=5683 -e IOT_DEV_URL="coap://[1]:5683/rd?sn=[2]&mn=[3]&ep=[4]" --name iot_device registry.gitlab.com/iotmgmt/iot-device*

Startet ein Docker Container mit dem Namen iot_device und folgenden Umgebungsvariabeln:

IOT_DEV_PORT: Port welcher von CoAP verwendet wird.

IOT_DEV_URL:
[1] entspricht der IP des sa_api Containers.
[2] entspricht der serial nr, für das IoT Device.
[3] entspricht der model nr, für das IoT Device.
[4] entspricht dem Namen (Endpoint) für das Device und kann frei gewählt werden.

Achtung: Angaben [2] und [3] müssen mit den Angaben auf dem Lieferschein übereinstimmen.
Beispiel - Kommando:
docker run -e IOT_DEV_PORT=5683 -e IOT_DEV_URL="coap://172.17.0.1:5683/rd?sn=100000501&mn=STOR&ep=Bobby" --name iot_device registry.gitlab.com/iotmgmt/iot-device

6.3.3 Manuelle Installation

1. *docker network create serverNet*

Richtet ein separates Netzwerk ein.

2. *docker run -v /home/my/path/:/data/db:Z --name sa_db --net=serverNet -d library/mongo:3.4.10*

Startet ein Docker Container mit dem Namen sa_db. **Achtung**: Ändern Sie beim Attribut –v den Pfad auf ein Verzeichnis Ihrer Wahl. Es ist lediglich der Teil vor dem ersten Doppelpunkt zu ändern.

3. *docker run -d -p 5683:5683/udp -p 9000:9000 -e IOT_BACK_DB=mongodb://sa_db:27017/coap --name sa_api --net=serverNet*

Startet ein Docker Container mit dem Namen sa_api.

4. *docker run -p 80:3000 --name=sa_web --net=serverNet -e IOT_WEB_URL=http://sa_api:9000 -e IOT_WEB_PORT=9000 registry.gitlab.com/iotmgmt/web*

Startet ein Docker Container mit dem Namen sa_web.

5. *docker run -e IOT_DEV_PORT=5683 -e IOT_DEV_URL="coap://[1]:5683/rd?sn=[2]&mn=[3]&ep=[4]" --name iot_device registry.gitlab.com/iotmgmt/iot-device*

Startet ein Docker Container mit dem Namen iot_device und folgenden Umgebungsvariabeln:

IOT_DEV_PORT: Port welcher von CoAP verwendet wird.

IOT_DEV_URL:
[1] entspricht der IP des sa_api Containers.
[2] entspricht der serial nr, für das IoT Device.
[3] entspricht der model nr, für das IoT Device.
[4] entspricht dem Namen (Endpoint) für das Device und kann frei gewählt werden.

Achtung: Angaben [2] und [3] müssen mit den Angaben auf dem Lieferschein übereinstimmen.

Beispiel - Kommando:
docker run -e IOT_DEV_PORT=5683 -e IOT_DEV_URL="coap://172.17.0.1:5683/rd?sn=100000501&mn=STOR&ep=Bobby" --name iot_device registry.gitlab.com/iotmgmt/iot-device