HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

ins institute for networked solutions

IoT - Provisioning and Management

Bachelorarbeit FS 2017

Abteilung Informatik
Hochschule für Technik Rapperswil

Autoren:	Andreas Stalder, David Meister
Betreuer:	Prof. Beat Stettler, Urs Baumann
Gegenleser:	Prof. Dr. Olaf Zimmermann
Experte:	Michael Schneider
Projektpartner:	INS Institute for Networked Solutions
Datum:	16. Juni 2017

Aufgabenstellung

Das "Internet of Things" (IoT) ist gerade am entstehen. In verschiedensten Projekten installiert die Schweizer Wirtschaft Tausende von Sensoren, die alles mögliche detektieren, messen, überwachen und die Resultate in die Cloud schicken. ABER: schon nach kurzer Zeit stehen diese Firmen vor der Frage, wie eine solche Flut von Geräten provisioniert, administriert, überwacht, troubleshooted, updated und sicher "retired" werden kann.

Diese Arbeit soll den Stand der Entwicklungen in diesem Umfeld aufzeigen, bestehende Lösungen evaluieren und einen eigenen Prototypen entwickeln. In einem ersten Schritt sollen bestehende Cloud Lösungen (Amazon, Azure, Google etc.) untersucht und daraus eine Zielarchitektur definiert werden. Zudem stellt sich die Frage, wie wir mit der fehlenden physical Security umgehen.

Vorgehen:

- Analyse von Cloud IoT Frameworks (Google, Azure, Amazon usw.)
- Kennenlernen einer Auswahl von IoT Sensoren
- Security-Analyse und Konzept
- Entwicklung einer skalierbaren Management-Architektur
- Bau eines Prototypen

Prof. Beat Stettler

Urs Baumann

Abstract

Das Internet of Things erfreut sich immer grösserer Beliebtheit. Unternehmensanwendungen mit einer grossen Anzahl Sensoren werden in Zukunft stark zunehmen. Bereits bei herkömmlichen Computersystemen stellt das Management eine grosse Herausforderung dar. IoT Devices dürften potenziell in einer noch deutlich grösseren Anzahl verbreitet sein als herkömmliche Geräte. Unternehmen benötigen Lösungen, um eine regelrechte Flut von neuartigen Devices effizient und sicher administrieren zu können. In dieser Arbeit werden wichtige Management-Aufgaben wie beispielsweise Bootstrapping, Konfiguration, Updates und Monitoring für IoT Systeme untersucht.

Nach einer umfassenden Einarbeitung in IoT Technologien und Management-Aufgaben mussten geeignete Kommunikationsprotokolle untersucht werden. Es stellte sich heraus, dass herkömmliche Protokolle wie HTTP im IoT Umfeld schlecht geeignet sind, da sie häufig nur über eingeschränkte Ressourcen wie Bandbreite und Rechenleistung verfügen. Das von der Open Mobile Alliance (OMA) entwickelte Lightweight Machine-to-Machine (LwM2M) Protokoll verwendet unterhalb das Constrained Application Protocol (CoAP). CoAP arbeitet im Gegensatz zu HTTP über UDP und besitzt im Allgemeinen einen geringeren Protokoll Overhead, weshalb es sich im IoT Umfeld besser eignet. LwM2M wurde entwickelt, um standardisierte Zugriffe auf Device Informationen zu ermöglichen. Als nächstes wurde eine geeignete Architektur für eine Management Webapplikation evaluiert.

Aus dieser Bachelorarbeit ist die Management Webapplikation ”Smartmanager” entstanden. Das Back-End wurde mit dem Java Spring Framework erstellt. Der LwM2M Server wurde mit der Eclipse Leshan Library implementiert, als Datenbank wurde MongoDB verwendet. Mit dem ”Smartmanager” können LwM2M Clients administriert werden und die integrierte Gruppenverwaltung ermöglicht Devices in selbst angelegte Hierarchien zu strukturieren. Ausserdem können Konfigurationen bestehend aus unterschiedlichen Schreiboperationen erstellt -, welche wiederum auf einzelne Geräte oder ganze Gruppen geschrieben werden können. Dank der Verwendung des LwM2M Protokolls können eine grosse Anzahl an Management-Aufgaben ausgeführt werden. Es bleibt zu hoffen, dass sich LwM2M zukünftig als Standard für das IoT Device Management durchsetzen wird um die einheitliche Verwaltung verschiedenster Geräte zu ermöglichen.

Management Summary

Ausgangslage

Internet of Things wird in Unternehmen immer wichtiger. Mit Hilfe von Sensoren werden bei IoT-Anwendungen reale Objekte miteinander verbunden. Bewährte Konzepte der Informatik werden angewandt, um intelligente Systeme von vernetzten Objekten zu erschaffen. In der Industrie werden bereits heute eine grosse Anzahl unterschiedlicher Sensoren eingesetzt. Vieles scheint bei IoT auf den ersten Blick bekannt, es gibt jedoch einige technische und organisatorische Unterschiede zu beachten. IoT-Devices sind verglichen mit herkömmlichen PCs und Notebooks in vielerlei Hinsicht eingeschränkt. Sie verfügen beispielsweise über weniger Ressourcen wie Rechenleistung, Speicher und Netzwerkbandbreite. Des Weiteren werden Sensoren in einer sehr viel grösseren Anzahl als herkömmliche Geräte vorhanden sein. Zukünftig werden ganze Geschäftsprozesse von einer funktionierenden IoT-Infrastruktur abhängig sein, weshalb Sicherheit immer wichtiger wird.

In einem Firmenumfeld sollten Sensoren und IoT-Devices effizient und einheitlich ausgerollt werden. Weitere Verwaltungsaufgaben wie Patching, Monitoring oder Troubleshooting müssen auch im IoT-Umfeld berücksichtigt werden. Ziel dieses Projekts ist eine umfassende Analyse über wichtige Aspekte im Bezug auf Device Management im IoT-Umfeld. Bestehende Ansätze und Lösungen sollen analysiert und erarbeitet werden, um einen geeigneten Prototypen erstellen zu können.

Vorgehen/Technologien

Zu Beginn hat sich das Projektteam ein Basisverständnis im IoT-Umfeld verschafft. Wichtige Aspekte des Device Managements wurden erarbeitet und aufgezeigt. Damit viele Geräte effizient verwaltet werden können, muss eine IoT-Management Applikation Funktionen für automatisches und sicheres Provisioning, Konfigurationsmanagement, Updating und Monitoring bieten.

Kommunikation über Netzwerke ist, wie der Name verrät, bei Internet of Things ein zentrales Thema. In dieser Arbeit wird dem Leser eine Übersicht über die IoT-Architektur, sowie über Kommunikationsmodelle vermittelt. Aufgrund der geänderten Anforderungen sind neue Protokolle entstanden. Es hat sich gezeigt, dass für Geräte mit eingeschränkten Ressourcen Alternativen gewählt werden sollten. Die Open Mobile Alliance (OMA) hat mit dem ”Lightweight Machine-to-MachineProtokoll (LwM2M) ein massgeschneidertes Protokoll für Device Management im IoT-Umfeld entwickelt. Bereits existierende Libraries für Implementierungen des LwM2M Stacks konnten für dieses Projekt verwendet werden.

Eine breite Security-Analyse hat die Notwendigkeit von Sicherheitsüberlegungen gezeigt. Gegenüber bestehenden Infrastrukturen könnten zukünftige IoT-Systeme weitaus anfälliger werden. Ausfälle und Kompromittierungen solcher Systeme könnten drastischere Auswirkungen haben als bei heutigen Systemen.

Aufgrund der Erkenntnisse hat das Projektteam Anforderungen an die Management Applikation definiert. Eine Architektur wurde erarbeitet und geeignete Technologien ausgewählt. Der Prototyp hat gezeigt, dass die gewählte Lösung mit einem Java-basierten LwM2M Server funktioniert. Durch das Web-Front-End können Benutzer über einen gängigen Webbrowser auf die Applikation zugreifen.

Ergebnisse

Mit dem ”Smartmanager” ist ein funktionsfähiger Prototyp einer IoT-Management-Applikation entstanden. Durch den Einsatz des LwM2M Protokolls können theoretisch viele unterschiedliche IoT-Devices verwaltet werden. Der Benutzer hat die Möglichkeit, Konfigurationen für Devices anzulegen und diese zu verteilen. Die implementierte Gruppenverwaltung für Devices ermöglicht den Aufbau einer gewünschten Struktur. Die durch LwM2M unterstützen Operationen ”Read”, ”Write” und ”Execute” können sowohl einzeln auf Devices ausgeführt-, als auch auf Gruppenebene ausgeführt werden. Neue Devices können über die Discovery-Funktion dem System hinzugefügt werden. Dabei können initiale Konfigurationen und Gruppenzugehörigkeit bestimmt werden.

Ausblick

IoT-Device-Management wird für viele Unternehmen ein Thema werden. Es ist erfreulich, dass die OMA mit LwM2M ein Protokoll entwickelt hat, um eine einheitliche Managementlösung für IoT-Devices zu ermöglichen. Es bleibt zu hoffen, dass viele Hersteller im IoT-Umfeld dieses Protokoll implementieren werden.

Wie zu erwarten war, konnten in diesem Projekt zeitlich nicht alle wichtigen Aspekte berücksichtigt werden. Für die produktive Nutzung des Tools könnten noch wichtige Sicherheitsfeatures und automatisiertes Bootstrapping implementiert werden.

Inhaltsverzeichnis

II. Anhang

Teil I.

Technischer Bericht

1. Einführung in Internet of Things

1.1 Übersicht

Das Internet der Dinge unterscheidet sich in einigen Aspekten vom klassischen Internet. End-Benutzer haben über sogenannte Terminals wie Laptops oder Smartphones über die globale Internet Infrastruktur kommuniziert [1]. Diese Terminals wurden meist von Benutzern eingeschaltet, benutzt und wieder ausgeschaltet. Damit Geräte mit dem Internet auf sinnvolle Art und Weise kommunizieren konnten, war eine manuelle Tätigkeit von Benutzern notwendig [2]. Beispiele dafür sind das Abrufen von E-Mails, Surfen im Web, Streaming von Videos oder Spielen von Online Games [1].

Mit „Internet of Things“ (IoT) wird eine andere Philosophie verfolgt. Es gibt keine einheitliche Definition und Abgrenzung von IoT. Grundsätzlich versucht man Objekte und Gegenstände, welche im klassischen Sinne des Internets nicht berücksichtigt wurden, ans Netz anzuschliessen. Mit minimalen menschlichen Eingriffen sollen diese Geräte Daten sammeln, austauschen und aufgrund von Software und Algorithmen Entscheidungen treffen [3]. Man spricht im Zusammenhang von „Things “ auch von „Smart Devices“ oder „Smart Objects“.

1.1.1 Smart Objects

Smart Objects oder auch „Things“ ergänzen das herkömmliche Internet um eine Vielzahl neuartiger Teilnehmer. Man ist versucht, die mit dem Internet erschaffene virtuelle Welt mit Objekten der tatsächlichen „echten“ Welt zu verbinden. Der Begriff „Smart“ ist seit der Erscheinung des iPhones weltweit bekannt. Er beschreibt die Fähigkeit eines Objekts mit dem Internet zu kommunizieren.

Während Smartphones oder Smart-TVs noch als herkömmliche Internet Terminals angesehen werden können, so erweitern die Smart Objects das bisherige Internet um eine neue Art von Teilnehmer. Smart Objects lassen sich wie folgt beschreiben:

- haben eine physikalische Repräsentation mit Eigenschaften wie Form und Grösse
- haben Mindestmass an Kommunikationsfunktionalitäten wie Request/Reply
- besitzen eine UID (unique identifier)
- haben mindestens einen Namen und eine Adresse
- besitzen ein Mindestmass an Rechenfähigkeiten
- besitzen Sensoren, um physikalische Erscheinungen wie Druck, Licht, Temperatur, etc. zu messen

Der letzte Punkt in der oberen Definition beschreibt den tatsächlichen Unterschied zu herkömmlichen Devices im Internet. Konzeptionell liegt bei IoT der Fokus mehr auf Daten und Informationen von physikalischen Objekten als bei Punkt-zu-Punkt Kommunikation von Terminals [1].

1.2 Einsatzgebiete

Das ”Internet of Things” hat viele Einsatzmöglichkeiten und wir stehen erst am Anfang. Es werden immer neue Einsatzbereich entdeckt und vorhandene optimiert und erweitert. Um die Einsatzmöglichkeiten aufzuzeigen, wird hier das Beispiel ”Gesundheitsvorsorge” genauer gezeigt.

1.2.1 Allgemein

Die Gesundheitsvorsorge ist ein riesiger Bereich, welche alle Menschen betrifft und enorme Kosten verursacht. Heutzutage wird noch sehr viel manuell erledigt. In naher Zukunft wird sich das ändern und das Internet der Dinge wird immer wichtiger, um die bestmögliche Behandlung jedes Patienten sicherzustellen.

1.2.2 Im Alltag

Selbstüberwachung In den letzten Jahren ist die Selbstüberwachung zu einem riesigen Thema geworden. Firmen wie Apple oder Fitbit haben diesen Bereich stark gefördert. So gibt es heute schon für mehrere Körperdatensensoren, die alles überwachen. Ein Beispiel dafür sind die Smartwatches. Diese können bereits heute den Puls rund um die Uhr überwachen oder sie zählen die Schritte mit. So kann jeder sein eigenes Fitnessprofil von sich erstellen. Ein weiteres Beispiel findet man bei Patienten, welche Medikamente verabreicht bekommen. Da Medikamente häufig vergessen werden, gibt es kluge Medikamentenspender, welche den Patienten benachrichtigen, wenn dieser die Medikamente nicht genommen hat.

Fernüberwachung Neben der Selbstüberwachung gibt es auch noch die Fernüberwachung. Dabei wird der Patient vom Pflegepersonal überwacht und bei kritischen Situationen kann sofort gehandelt werden. Für Rollstuhlfahrer oder Senioren gibt es zum Beispiel ein Falldetektor. Falls der Patient umfällt, reagiert der Sensor und meldet diesen Unfall direkt an das Pflegepersonal. So kann schnell reagiert und dem Patienten geholfen werden. Auch das Überwachen der Patientenwerte kann so über das Internet sichergestellt werden. Durch eine kontinuierliche Abfrage der Werte, ist die Reaktionszeit enorm geringer als beim manuellen Messen. Hierfür benötigt man Pflegepersonal vor Ort.

1.2.3 Im Spital

Patientenzimmer Durch die komplette Überwachung des Patienten in seinem Zimmer, verkürzt sich die Reaktionszeit enorm. Der Patient wird zwar schon überwacht, aber durch die Anbindung an das Internet wären weitere Verbesserungen möglich. So kann man bei gewissen Problemen nicht nur das Pflegepersonal alarmieren, sondern zum Beispiel auch direkt benötigtes Personal aus anderen Spitälern anordnen. Auch könnte man die Messdaten aus der Fernüberwachung mit der aus dem Patientenzimmer verknüpfen und so ein besseres Patientenbild erstellen.

Infrastruktur In Spitälern werden viele verschiedenen Geräte eingesetzt, welche momentan noch nicht miteinander kommunizieren. Durch die Verknüpfung der Geräte könnte man die Daten verschiedener Geräte miteinander kombinieren und vergleichen, um ein besseres Bild erstellen zu können. Oder auch die Wartung würde viel einfacher werden. Aus der Ferne werden alle Geräte inventarisiert, gewartet und überwacht. So wird bei einem Fehler sofort der Techniker gerufen, damit ein defektes Gerät sofort wieder Einsatzbereit gemacht wird.

2. IoT-Device-Management

In Zukunft ist eine stark ansteigende Anzahl an IoT-Devices zu erwarten. Laut der International Data Corporation (IDC) dürften im Jahre 2020 in etwa 30 Milliarden Devices weltweit verbunden sein [4]. Unternehmen könnten potenziell mehrere Tausend Sensoren für ihre Zwecke einsetzen. Bereits bei herkömmlichen Computersystemen und Servern stellt das Management eine grosse Herausforderung dar. IoT-Devices dürften potenziell in einer sehr viel grösseren Anzahl verbreitet sein als herkömmliche Geräte. Herausforderungen wie die Heterogenität, Verteilung und Security verschärfen sich mit der stetig wachsenden Anzahl an Geräten.

IoT-Devices durchleben in ihrem Lebenszyklus verschiedene Stadien. Ein Device-Management Tool soll die Administration in jeder dieser Phasen unterstützen.

2.1 Device Lifecycle

Der Lebenszyklus eines IoT-Devices könnte beispielsweise aus folgenden fünf Phasen bestehen.

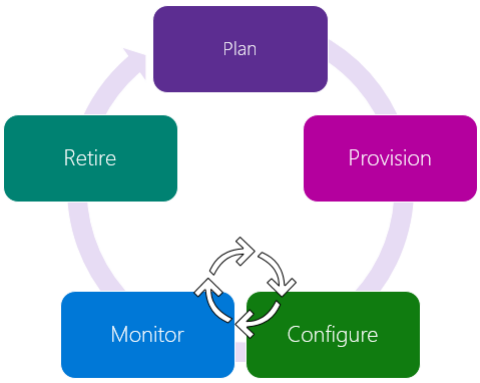


Abbildung 2.1.: IoT-Management Azure [5]

Plan In der Planungsphase möchte man aufgrund von vorliegenden Daten eine Veränderung am System vornehmen. Um fundierte Entscheide in der Planungsphase zu ermöglichen werden Daten und Messwerte von Devices benötigt. Je einfacher diese Daten zugänglich-, respektive abfragbar sind, desto qualitativ hochwertiger und exakter kann geplant werden.

Provision Neue Geräte müssen vor der produktiven Nutzung bereitgestellt werden. Dieser Prozess kann mehrere Personen und Aufgaben involvieren. Grundsätzlich werden neue Geräte in ein bestehendes System eingebunden oder ein komplett neues System aufgebaut.

Configure Damit ein Device in den vorgesehenen Zustand versetzt werden kann, benötigt es eine Konfiguration. Bei einer grossen Anzahl Devices empfiehlt es sich diesen Prozess bestmöglichst zu automatisieren.

Monitor In der Monitoringphase sollen die Zustände der Devices überwacht werden. Ziel ist es, die Funktionalität und Korrektheit des angestrebten Verhaltens sicherzustellen. Dies wird mittels periodischer Abfragen oder Observations sichergestellt.

Retire Am Ende des Lebenszyklus sollen Geräte geordnet aus dem System entfernt werden. Dabei gilt es den Prozess bestmöglich zu automatisieren und allfällige Vorschriften betreffend Datensicherheit und Datenschutz zu beachten. Andere Systeme wie das Inventar könnten ebenfalls in diesem Prozess beteiligt sein.

2.2 Device Management Aufgaben

2.2.1 Provisionierung und Authentisierung

Bevor neue Devices produktiv genutzt werden können, müssen sie entweder manuell oder automatisch provisioniert werden. Beim Device Provisioning muss ein Gerät in einen Zustand gebracht werden, in dem es für eine oder mehrere vorgesehene Personen erreichbar respektive verfügbar ist.

Notwendige Schritte So unterschiedlich die Geräte im Netzwerk und Internet auch sein können, so haben sie abstrakt gesehen die selben Schritte zu durchlaufen um für die produktive Nutzung Verfügbar zu werden:

- ein Device muss physisch am vorgesehenen Ort platziert werden
- die Stromversorgung (Stromnetz, Batterie, Akku) für das Gerät muss sichergestellt werden
- das Device muss gestartet werden
- eine Initialkonfiguration muss geladen werden
- die Netzwerk- respektive Internetverbindung muss hergestellt werden

Ab dem Zeitpunkt der Erreichbarkeit des Netzwerks unterscheiden sich manuelles und automatisches Provisioning. Beim manuellen Provisioning greift der User oder Administrator entweder direkt über ein grafisches User Interface auf das Gerät zu und nimmt weitere Einstellungen vor oder er verbindet sich über das Netzwerk auf eine Benutzeroberfläche des Geräts. Beim automatischen Provisioning meldet sich das Device über das Netzwerk an einem Controller, respektive einer Management-Instanz.

Authentisierung Bei einem Eintritt in ein bestehendes System, in diesem Falle die Bereitstellung eines neuen Devices, muss sich das Gerät am System authentisieren. Aus Sicherheitsgründen möchte man sicherstellen, dass die Zugriffe in das System kontrolliert werden. Es gibt verschiedene Möglichkeiten der Authentisierung:

- Authentisierung durch Wissen (z.B. Passwort, PIN, Challenge)
- Authentisierung durch Besitz (z.B. RFID-Chip, Zertifikat, One Time Pin)
- Authentisierung durch biometrische Merkmale (z.B. Fingerprint, Handvene)

Nicht alle Verfahren eignen sich für die Authentisierung von Devices. Ein Gerät selbst verfügt über keine biometrischen Merkmale, höchstens der Benutzer, welcher versucht das Gerät bereitzustellen. Ein Shared Secret in Form eines Passworts oder eine Chain-of-Trust mit Zertifikaten eignen sich für die Device Authentisierung besser.

IoT-Provisioning IoT-Devices sollten vor der produktiven Nutzung in einem definierten Prozess bereitgestellt werden. IoT-Devices befinden sich häufig nicht in einer standardisierten Büroumgebung wie zum Beispiel Notebooks und Drucker. Mögliche Standorte sind Lagerhallen, Produktionsstätten oder auch andere Orte im Freien. Oft ist es nicht möglich solche Geräte an das Stromnetz anzuschliessen, deshalb werden diese über Batterien oder einen Akku gespiesen. Die Netzwerk- und Internetverbindung wird über ein Wireless-Mesh-Netzwerk realisiert. In einem ersten Schritt muss ein Device mit einem Netzwerk verbunden werden. Es ist weitgehend bekannt, dass eine grosse Vielfalt an Devices existieren. Manche von ihnen enthalten ein einfaches User Interface, andere nicht. So unterscheidet sich das Ausrollen von IoT-Geräten von herkömmlichen PCs und Laptops indem man nicht direkt über das geräteeigene UI Einstellungen vornehmen kann. Ein Management von IoT-Devices muss also unterschiedliche Anforderungen abdecken. Es könnten sowohl Geräte im herkömmlichen Sinne über eine direkte Internetverbindungen provisioniert werden, als auch neuartige Geräte, welche an ein Low-Power & Lossy Network (LLN) angeschlossen sind.

Ein neues IoT-Device im System kontaktiert entweder seinen lokalen Gateway oder direkt einen Server über TCP/IP. Das System kann nun das neue Device aufnehmen und weitere Aufgaben wie beispielsweise das Laden der Konfiguration oder das Speichern von Geräteinformationen veranlassen. Netzwerke mit IoT-Devices sind sehr dynamisch. Es werden häufig neue Teilnehmer aufgenommen oder alte Teilnehmer entfernt. Tools, welche diese Prozesse automatisieren sind stark gefragt. Devices sollen sich selbständig an Netzwerken anmelden können und sich bei zuständigen Servern für weitere Anweisungen registrieren. Solche Discovery Mechanismen sind für zukünftige Anwendungen essentiell [6].

Mit dem Prozess der automatischen Registrierung an Netzwerken und Systemen sind Fragen der Sicherheit verbunden. Man möchte selbstverständlich die Zugriffe in Netzwerke und Systeme korrekt authentisieren und autorisieren. Sicheres Bootstrapping von IoT-Devices umfasst neben Authentisierung und Autorisierung auch die Übertragung von Security Parametern für die Ausführung von vertrauenswürdigen Operationen [7].

2.2.2 Konfiguration

Konfigurationsmanagement Konfigurationsmanagement beschäftigt sich mit der Erstellung, Verteilung, Versionierung und Sicherung der Konfigurationen von unterschiedlichen Devices. Durch eine Konfiguration wird das Verhalten eines Geräts bestimmt. Die grösste Schwierigkeit bei den Konfigurationen liegt in der Heterogenität. Unterschiedliche Hersteller verwenden für ihre Geräte unterschiedliche Konfigurationen. Selbst Devices derselben Hersteller benötigen oft eine andere Konfiguration. Im Kern lassen sich diese Tatsachen nicht ändern, man kann lediglich versuchen eine möglichst generische Basiskonfiguration für eine Gruppe von Devices bereitzustellen, um die manuellen Tätigkeiten zu minimieren.

Automatische Konfiguration Eine manuelle Konfiguration von sämtlichen IoT-Devices scheint nicht zeitgemäss. So müsste vor jeder Auslieferung das Gerät mit allen Besonderheiten vorinstalliert werden. Stattdessen sollte eine möglichst generische Konfiguration automatisch geladen werden (Zero Touch Provisioning). Bei Bedarf könnte die Konfiguration remote angepasst werden [8]. Auf diese Weise würden beispielsweise in einem WSN von IoT-Devices sämtliche Sensorgeräte initial mit derselben Konfiguration gestartet werden. Möchte man punktuell eine Veränderung an einem Gerät vornehmen, so wäre dies Remote möglich, da mit der generischen Initialkonfiguration Connectivity und Managability sichergestellt würden.

Softwareverteilung Nebst dem erstmaligen Laden und Anpassen der Konfigurationen von Devices müssen auch deren Soft-, respektive Firmware verwaltet werden. Bei notwendigen Patches und Upgrades sollten diese von Remoteseite aus provisioniert werden können. So können bei bekannt werdenden Sicherheitslücken eine grosse Anzahl Geräte mit Updates versorgt werden. Die Schwierigkeit liegt darin, die möglicherweise grosse Datenmenge der Soft- oder Firmwares auf Geräte in LLNs zu übertragen.

Backup und Restore Backup und Restore der Devicekonfigurationen sind wichtige Aufgaben im Device-Management. Die Sicherungsvorgänge müssen entweder automatisch über ein Scheduling, oder manuell von einer Person vorgenommen werden können. Eine manuelle Sicherung ist vor allem bei einer einmaligen Anpassung an der Konfiguration notwendig.

Die Aufgaben des Konfigurationsmanagements sind immens wichtig. Bei fehlerhaften Konfigurationen oder Ausfällen respektive Defekten von Devices drohen kostspielige Downtimes. Es wäre gar möglich, dass eine ganze Produktionskette für längere Zeit ausfällt. Weitgehend automatisierte Prozesse bei Konfigurationen, sei es das Laden oder das Sichern, stellen einen enormen Mehrwert für den Betrieb von IoT-Devices dar.

2.2.3 Monitoring und Diagnose

Monitoring allgemein Das Hauptziel des Monitorings ist das proaktive Überwachen der Devices. Dadurch verringert sich nicht nur die Zeit, die benötigt wird um den Fehler zu erkennen, sondern auch die benötigte Reparaturzeit. Ein weiteres Ziel ist das Erkennen von Mustern. So können all die gesammelten Daten zusammengefügt- und die Muster analysiert werden, Dies führt zu einer besseren Früherkennung. Zukünftige Probleme können so besser erkannt- und schneller behoben werden. [9]

Monitoring im IoT-Bereich Im Bereich IoT gibt es spezielle Anforderungen an das Monitoring. Durch die vielen Devices und die dynamischen Netzwerke muss das Monitoring flexibel sein. Täglich werden neue Device eingeführt und alte entsorgt. Nicht nur der Austausch von von Geräten ist mühselig, sondern auch die möglichen Standortwechsel. Die Devices bewegen sich zum Teil und sind so in verschiedenen Netzwerken.

Schlussendlich unterscheidet sich auch die Art der Überwachung. IoT-Monitoring ist nicht mit herkömmlichem Monitoring gleichzustellen. Zum Beispiel bei Serverfarmen sind Auslastung, verfügbarer Speicher wichtig. Bei IoT-Devices sind andere Faktoren wie Verfügbarkeit und Software-Versionen wichtiger. Auch die gewonnen Daten können wichtig sein. Falls zum Beispiel bei Temperaturmessungen arktische Kälte anstelle von sommerlichen Temperaturen gemessen werden, ist das sicher auch ein Indiz für Fehlverhalten.[10]

2.2.4 Maintenance und Update

Maintenance und Update Allgemein Unter Maintenance versteht man die Wartung und das dazugehörige Updaten der Devices. Bei einer kleinen Anzahl von Geräten funktioniert dies vielleicht noch gut mit einer von Hand geführten Liste, aber bei Hunderten oder gar Tausenden von Geräten wird es schnell unübersichtlich. Daher wird ein Maintanence-Management benötigt.

Ein Bestandteil des Maintenance-Managements ist das Asset-Management. Beim Asset-Management werden alle Gerätedaten, Handbücher, Garantien und andere relevante Daten des entsprechenden Geräts gespeichert.[11] So hat man eine zentrale Verwaltung aller wichtigen Daten und kann effizienter arbeiten.

Heutige Geräte werden mit einem Softwarestand ausgeliefert, bei denen noch Fehler vorhanden sein könnten. Daher veröffentlichen Gerätehersteller ständig neue Updates und Patches, um neue Funktionen und Sicherheitsupdates einzuspielen. Ohne ein Update-Management wäre diese Aufgabe für so viele Geräte unmöglich. Daher ist eine zentrale Verwaltung unbedingt nötig.

Maintenance und Update im IoT-Bereich Da es bei IoT-Geräten häufig um Sensoren handelt, welche auch im freien Gelände platziert werden können, ist ein Wartungsmanagement sehr wichtig. Solche Sensoren sind der Umwelt erschwerten Bedingungen ausgesetzt als normale PCs oder Netzwerkgeräte.

Ohne ein ausgereiftes Management, kann diese Menge aber nicht gewartet werden. So soll das Management eine Empfehlung geben, wann die Batterie auszutauschen ist, oder wann der Sensor nicht mehr die gewünschte Genauigkeit liefert. So weiss der Techniker genau, wann welche Sensoren/Geräte ausgetauscht werden müssen.

Ein weiteres Problem könnten allfällig grosse Distanzen zu Sensoren betragen. Der Sensor befindet sich vielleicht mehrere Kilometer entfernt an schwer erreichbaren Orten.

IoT-Geräte sollte jahrelang im Einsatz sein. Doch Hersteller könnten Konkurs gehen oder pflegen ein Produkt nicht weiter. Nun kann es zum Problem kommen, wenn ein Sensor keine Updates mehr erhält und schwerwiegende Sicherheitslücken vorhanden sind, vor solchen Szenarien sollte man gewarnt werden.

Durch die vielen Software Updates bei der hohen Anzahl an Geräten, ist man ohne automatische Softwareverteilung überfordert. Von Hand kann diese hohe Anzahl an Geräte kaum bewältigt werden. Mit einer Softwareverteilung würde man den gewünschten Softwarestand freigeben, die Managementumgebung verteilt diese automatisch auf die jeweiligen Sensoren. Bei Problemen wird der Administrator benachrichtigt und kann eingreifen.

Bei der grossen Anzahl von Updates darf das Netzwerk nicht vernachlässigen werden. Bedenkt man, dass IoT-Netzwerke für geringere Bandbreite ausgelegt sind, muss unbedingt auf die Datenmenge geachtet werden. Bei vielen Geräten könnte ein solches Netzwerk überlastet werden wenn man allen Geräten gleichzeitig ein Update schickt.

3. Kommunikation

3.1 Architektur

Man könnte ein IoT-System in vier wichtige Gruppen unterteilen: [12]

- Dinge (things)
- das lokale Netzwerk
- das Internet
- Back-End Services (z.B. Cloud Services)

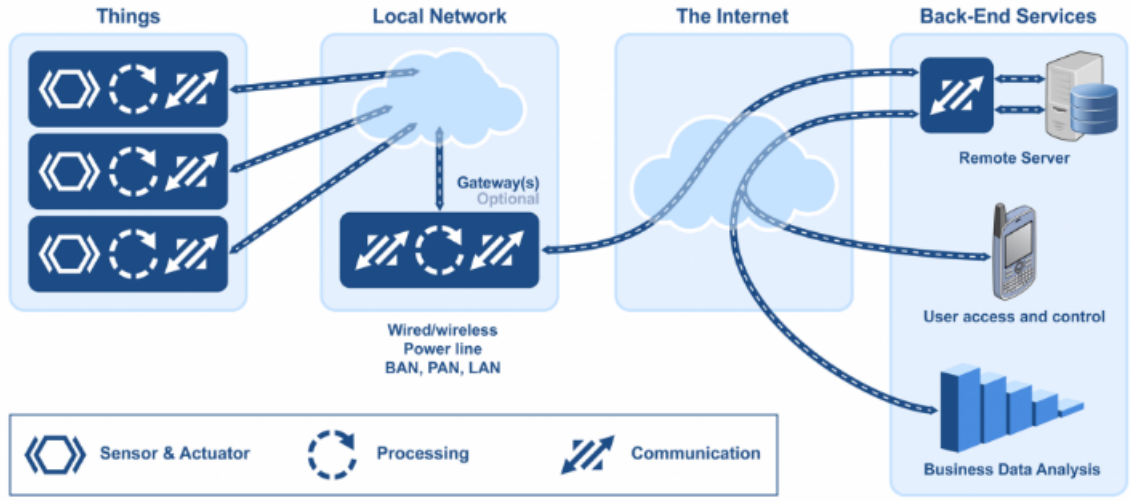


Abbildung 3.1.: IoT-Systemübersicht[13]

Grundsätzlich scheint die Architektur vertraut. Smart Objects kommunizieren über ein lokales Netzwerk mit Diensten im Internet.

Bisher konnten Geräte wie Laptops, PCs und Smartphones beinahe einheitlich mit dem Internet verbunden werden; entweder verkabelt über Ethernet oder drahtlos über ein lokales WLAN oder mobile Netze wie UMTS und LTE. Die Endgeräte verfügten jeweils über viel Rechenleistung, Speicher und ein leistungsfähiges Betriebssystem mit einem vollständig implementierten TCP/IP Stack.

In einem IoT-System muss man von einer grossen Anzahl an Geräten mit Sensoren ausgehen. Manchmal verfügen diese Geräte über eine extrem niedrige Bandbreite, wenig Speicher und Rechenleistung [14].

Die Kommunikation erfolgt oft nicht vertikal der Architektur, sondern auch horizontal auf derselben Ebene. Sensordevices können beispielsweise miteinander kommunizieren oder Cloud Services Daten der Sensoren untereinander austauschen.

3.1.1 Wireless Sensor Netzwerke

Bei einer Vielzahl von verbundenen Sensoren, welche über einen grossen Bereich verstreut sind, bietet sich ein Wireless Sensor Network (WSN) an. In einem WSN werden die Sensoren nicht direkt mit dem Internet verbunden. Die Daten werden drahtlos von Teilnehmer zu Teilnehmer versendet. Muss ein Datenpaket in ein entferntes Netzwerk wie das Internet, so wird ein Gateway oder Edge Node benötigt.

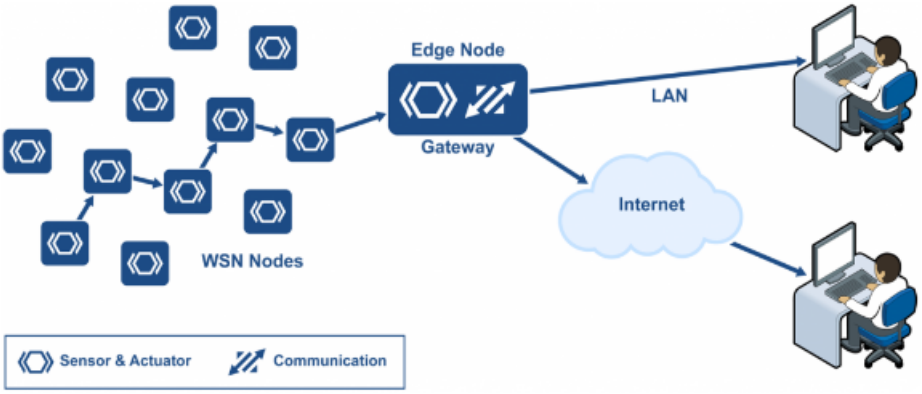


Abbildung 3.2.: IoT-WSN[15]

WSN Nodes sind typischerweise günstig im Einkauf. Sie können mit sehr wenig Leistung betrieben werden, dies ermöglicht den Batteriebetrieb. Durch diese Eigenschaften können WSN Nodes einfach, schnell und in sehr grosser Anzahl bereitgestellt werden.

3.2 Kommunikationsmodelle

Internet of Things verbindet Objekte aus der realen Welt miteinander. Um Objekte aus der Realität in die virtuelle Welt zu transformieren, werden Sensoren verwendet. Es gilt nun, diese Sensoren mit dem Internet zu verbinden.

Um unterschiedliche Bedürfnisse abzudecken, sind verschiedene Arten der Kommunikation entstanden. Die mit Sensoren ausgestatteten Geräte können sich in ihrer Weise, mit dem Internet zu kommunizieren stark unterscheiden.

3.2.1 Device-to-Device

Beim Device-to-Device Kommunikationsmodell kommunizieren mehrere Teilnehmer direkt miteinander (Peer-to-Peer). In diesem Szenario kommunizieren unterschiedliche Glühbirnen drahtlos mit einem Lichtschalter. Denkbar wären sämtliche Anwendungsgebiete aus dem „Smart Home“-Bereich. Kommunikation mit dem Internet ist nicht zwingend notwendig. Eine grosse Herausforderung besteht darin, dass mehrere Teilnehmer unterschiedlicher Hersteller miteinander interagieren können. Dazu müssen die Teilnehmer denselben Protokoll-Stack implementieren.

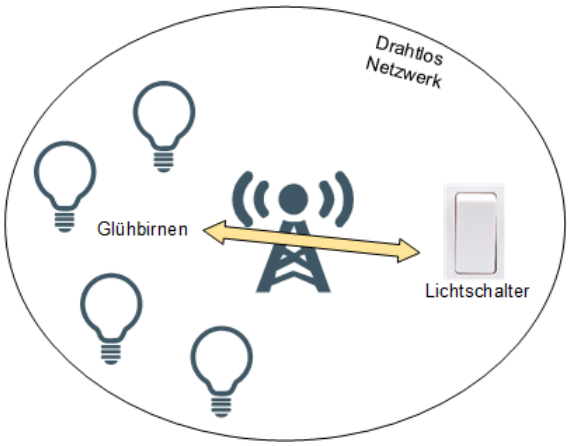


Abbildung 3.3.: Device-to-Device Kommunikation

3.2.2 Device-to-Cloud

Die Gerätehersteller bieten für ihre End-User Cloud-Dienste im Internet an. Die Sensorgeräte kommunizieren direkt End-to-End über TCP/IP mit dem jeweiligen Cloud-Dienst. Die Benutzer können über eine Mobile App oder eine Webseite auf die jeweiligen Sensordaten zugreifen. Häufig wird aufgrund proprietärer Kommunikationsprotokolle ein Vendor-lock-in betrieben. Dies erschwert die Interoperabilität von Sensoren unterschiedlicher Hersteller [3].

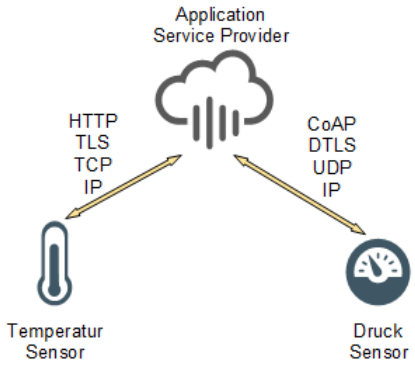


Abbildung 3.4.: Device-to-Cloud Kommunikation

3.2.3 Device-to-Gateway

Anstatt einer Ende-zu-Ende Kommunikation zwischen Sensoren und Servern wird in diesem Modell ein Gateway zwischen diesen Komponenten eingesetzt. Sensoren kommunizieren somit nicht direkt mit einem Server. Auf diese Art und Weise können eine grosse Anzahl Sensoren mit Internetdiensten verbunden werden ohne dass die Sensoren selbst über einen direkten Internetzugriff verfügen. Der Gateway muss somit über eine Schnittstelle verfügen, damit Dienste im Internet indirekt mit den Sensoren kommunizieren können. Aus Sicht des Diensts ist es irrelevant, wie der Gateway mit den Sensoren kommuniziert.

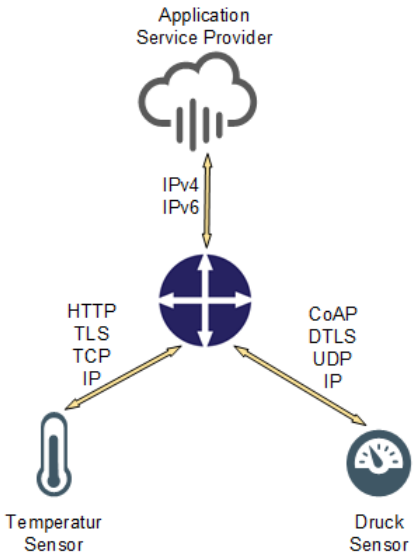


Abbildung 3.5.: Device-to-Gateway Kommunikation

3.2.4 Back-End Data-Sharing

Sobald sich die Sensordaten auf einem Server befinden, können diese auf bekannte Weise anderen zur Verfügung gestellt werden. Beispielsweise könnte man den Zustand eines Sensors als JSON Objekt über eine REST-Schnittstelle abfragen. Ein weiterer Serviceprovider muss somit nicht mehr direkt mit den Sensoren kommunizieren. Da Sensordevices oft über limitierte Möglichkeiten verfügen, grössere Datenmengen bereitzustellen, verringert man mit dieser Art der Kommunikation die Anzahl Abfragen auf den Sensordevices.

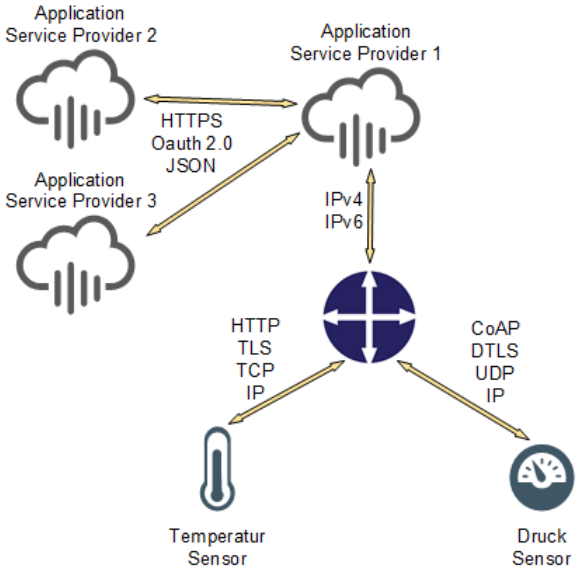


Abbildung 3.6.: Backend-Data-Sharing Kommunikation

3.3 IoT-Kommunikationsprotokolle

Seit der Entstehung des Internets werden für unterschiedliche Aufgaben Kommunikationsprotokolle entwickelt. Eine grosse Herausforderung war stets die Interoperabilität zwischen Geräten unterschiedlicher Hersteller. Standardisierungsgremien wie die International Standards Organization (ISO) und die Internet Engineering Task Force (IETF) haben in den vergangenen Jahrzehnten Richtlinien und Standards von Kommunikationsprotokollen veröffentlicht.

Mit zunehmender Popularität des Internets der Dinge sind eine unüberschaubare Menge an proprietären und offenen Kommunikationsprotokollen entstanden. In der Geschichte des Internets hat sich gezeigt, dass sich langfristig nur offene Protokolle durchsetzen werden [16], proprietäre Protokolle hingegen werden aufgrund der fehlenden Interoperabilität niemals eine breite Verwendung finden.

3.3.1 Anforderungen

Bereits heute zeichnen sich die populärsten IoT-Kommunikationsprotokolle ab. Um zu verstehen weshalb-, und vor allem in welchen Szenarien welches Protokoll eingesetzt wird respektive werden sollte, muss man sich mit den unterschiedlichen Anforderungen vertraut machen.

Die typischen bekannten Fragen nach der Verbreitung/Unterstützung und Skalierbarkeit stellen sich auch hier. Ebenfalls muss auf die mutmassliche Datenmenge geachtet werden. So dürfte eine vergleichsweise hohe Datenmenge für Geräte, welche über einen direkten, verkabelten Internetzugang verfügen, kein Problem darstellen, während Geräte in einem Mesh-betriebenen WSN wohl über deutlich weniger Bandbreite verfügen dürften.

Weitere Anforderungen wären Realtime Kommunikation, Stromverbrauch, Sicherheit und Network Address Translation (NAT) [17].

3.3.2 Request/Response

Request/Response ist das wohl bekannteste Pattern. Ein Client fordert mittels eines Requests eine Response von einem Service an. Der Service hört auf einkommende Requests, verarbeitet diese und antwortet (Response) den aufrufenden Clients. Request/Response Kommunikation skaliert schlecht, deshalb sollte beim Versenden einer Meldung an viele Teilnehmer auf ein anderes Pattern zurückgegriffen werden.

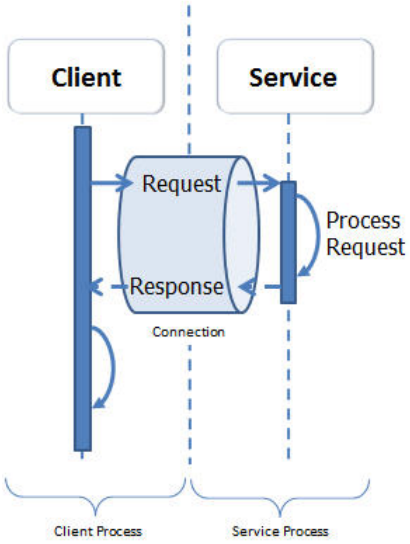


Abbildung 3.7.: Request/Response Kommunikation [18]

HTTP In den 1990er Jahren wurde HTTP verwendet um statische HTML-Dokumente übers Internet abzurufen. Bis heute hat sich die grundsätzliche Funktion von HTTP nicht verändert. HTTP bietet über seine Methoden ein umfangreiches Interface für die Request/Response Kommunikation zwischen Clients und Servern über das Internet.

Aufgrund seiner hohen Verbreitung, Standardisierung und Unterstützung ist HTTP auch im IoT-Umfeld beliebt. Für fast jede Programmiersprache und Laufzeitumgebung existieren Libraries, was das Entwickeln sehr angenehm macht. HTTP eignet sich jedoch nicht für alle Anwendungsfälle im IoT-Bereich. Für jeden Request wird der gesamte HTTP (und darunterliegende) Header benötigt. Zusätzlich ist das Protokoll textbasiert, was mit dem zusätzlichen, grossen Overhead eine erhebliche Datenmenge bedeuten könnte. Für Endgeräte an Mobilen Netzwerken könnte dies ungeeignet sein [17].

In der Version 1, respektive 1.1 gibt es mit HTTP keine Möglichkeit, echte Push-Meldungen zu versenden. Bei Push-Meldungen sendet der Server eine Response (besser: Nachricht) an den Client ohne vorgängigen Request. In der Version 2 von HTTP sind echte Push-Meldungen vorgesehen, jedoch gibt es wenige Implementation und Erfahrungswerte damit.

CoAP Das Constrained Application Protocol (CoAP) implementiert wie HTTP das Request/Response Pattern [17]. Mit CoAP existiert ein massgeschneidertes IoT-Protokoll, welches nach dem REST Paradigma konzipiert wurde. HTTP ist schwergewichtig, hat einen grossen Overhead und generiert damit hohe Datenmengen. Ausserdem ist vor jeder Session den für TCP benötigten 3-Way Handshake nötig.

CoAP wurde entwickelt, um diesen Schwächen von HTTP entgegenzuwirken. Bei sogenannten Low-Power and Lossy Networks (LLNs) sind die Nodes im Vergleich zu herkömmlichen Computersystemen sehr eingeschränkt, was die Verwendung von HTTP schwierig gestaltet. CoAP bietet grundsätzlich folgende Features:[19]

- Request/Response Kommunikation zwischen Endpoints
- Discovery von Services und Ressourcen
- URIs und Media Types
- Kompatibilität mit HTTP
- Multicast Support
- sehr kleiner Overhead (Header von 4 Byte)
- implementiert das Observer Design Pattern
- UDP als Transportprotokoll
- Asynchroner Nachrichtenaustausch

CoAP kann Peer-to-Peer zwischen Devices eingesetzt werden, aber auch zwischen Device und Service oder zwischen Device und einem Proxy.

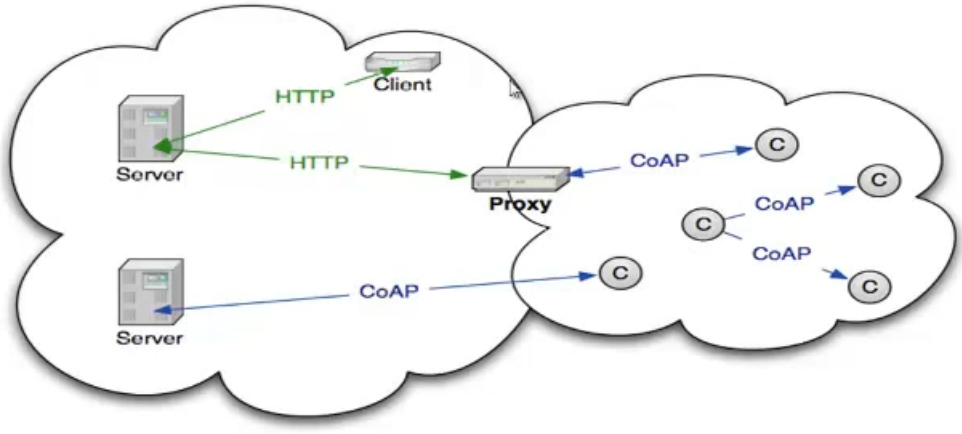


Abbildung 3.8.: CoAP Architektur [20]

Durch die massgeschneiderten Features für IoT wird CoAP hauptsächlich in WSNs eingesetzt [17].

3.3.3 Publish/Subscribe

Beim Publish/Subscribe Pattern gibt es einen Sender (Publisher) und einen Empfänger (Subscriber). Der Empfänger hört auf gewisse Themen (Topics). Der Sender kategorisiert seine Nachrichten in Themen und sendet diese zu den jeweiligen Empfängern. Sender und der Empfänger wissen nichts voneinander, sie senden oder hören nur im Netzwerk, ob eine für sie interessante Nachricht angekommen ist. Durch die einfache Verknüpfung von Publisher und Subscriber, eignet sich dieses Verfahren sehr gut im IoT-Bereich. Die Sensoren sind die Publisher, sie liefern zum Beispiel Temperaturdaten in die richtige Kategorie. Alle Server/Clouddienste, welche sich für Temperaturdaten interessieren, können auf diese Kategorie hören. Durch die einfache Handhabung skaliert dieses Pattern sehr gut.

In der folgenden Grafik sieht man das Publish and Subscribe Pattern. Der Publisher hat eine ”Address Changed” Message in den Channel geschickt. Nun erhalten alle Subscriber, welche dem Channel folgen, diese Nachricht und verarbeiten sie.

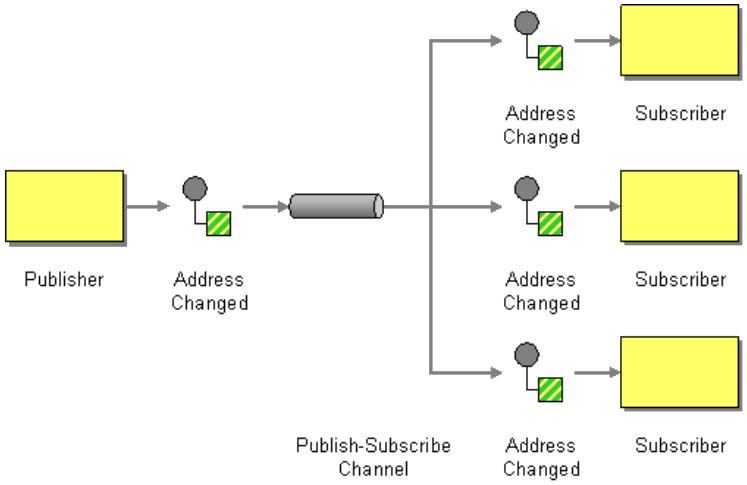


Abbildung 3.9.: Publish and Subscribe Pattern[21]

MQTT MQTT (Message Queue Telemetry Transport) ist ein von IBM entwickeltes Protokoll. Es ist ein speziell für IoT entwickeltes Protokoll um Machine-to-Machine Kommunikation herzustellen. Bei dem Protokoll wurde speziell auch ein schlankes Design geachtet. So ist die kleinst mögliche Nachricht gerade mal 2 Byte gross.

Der Hauptverwendungszweck von MQTT ist vor allem der Austausch von Daten zwischen Geräten und Server (D2S).[22] Da das Protokoll für die D2D Konnektivität entwickelt wurde, wird es auch in diesem Bereich eingesetzt. Die Vorteile liegen in Netzen mit vielen kleinen Geräten, welche wenig kommunizieren.[23]

Bei MQTT wird ein Broker verwendet. Der Sensor ”published” seine Daten mit einem Topic und den Daten an den Broker. Dieser sendet die Nachricht an Subscriber des jeweiligen Topics.

AMQP AMQP (Advanced Message Queuing Protocol) ist ein bekanntes und viel eingesetztes Protokoll. Das binäre Netzwerkprotokoll wird von vielen grossen Firmen[23] entwickelt, wie zum Beispiel Microsoft oder auch Cisco. Seit 2010 ist die Version 1.0 als aktueller Standard im Einsatz.

AMQP wird vorallem im Server zu Server Bereich eingesetzt (S2S).[22] Daher ist es in Bereichen einzusetzen, in der die Geschwindigkeit und der Prozessor nicht relevant sind. Zusätzlich wird es in Bereichen eingesetzt, in dem eine Nachricht nur von A nach B gesendet werden soll und man keine Nachricht verlieren möchte.

Die Sensoren senden die Nachrichten an einen Message Broker, welcher die Nachricht in die richtige Queue schiebt. Nun können sich die Dienste an der Queue anmelden und die Nachrichten konsumieren. Dabei gibt es die Möglichkeit Topics zu setzen, um Kategorien einzuführen. Es können jeder Nachricht auch noch Attribute hinzugefügt werden, wie zum Beispiel Name oder Durability. Durch den grossen Funktionsumfang des Protokolls ist es natürlich auch schwieriger einzurichten und die minimale Paketgrösse wächst damit auch. Die kleinstmögliche Paketgrösse ist 60 Byte.

XMPP XMPP (Extensible Messaging and Presence Protocol) wurde speziell für das Internet der Dinge erweitert, um den Anforderungen gerecht zu werden. XMPP gibt es schon seit mehreren Jahren und wurde in vielen Chatprogrammen eingesetzt. Auch heute findet man das Protokoll beim Facebook-Messenger wieder. Mit der IoT-Erweiterung/Anpassung will man nun nicht mehr Menschen miteinander verknüpfen, sondern Dinge.

XMPP ist ein hervorragendes Protokoll um Geräte mit Menschen zu verbinden. Dies ist eine Spezialform des Geräte zu Server Pattern (D2S). [22] XMPP sollte dann verwendet werden, wenn die Geschwindigkeit und der Prozessor nicht wichtig sind, wenn das Gerät immer verbunden sein soll und wenn nur wenige Konnektivitätspunke in einem grossen Bereich vorhanden sind.[23]

Bei XMPP wird kein Broker, sondern ein zentraler Server verwendet. Dieser soll allerlei verschiedene Geräte miteinander verbinden. Dieses Protokoll wird häufig für das Remote Management von Konsumergeräten verwendet.

3.4 LwM2M

3.4.1 Einführung

LwM2M ist ein speziell für IoT entwickeltes Management- und Messaging-Protokoll. Wie es der Name schon verrät, handelt es sich um ein M2M - Machine-to-Machine - Protokoll. Bei LwM2M wird ein Client-Server-Modell umgesetzt. Dabei werden alle Daten per Request angefragt und als Response beantwortet.

Entwickelt und standardisiert wurde dieses durch die Open Mobile Alliance, kurz OMA. Die OMA ist ein Verbund aus Firmen, welche vroallem in der Mobilfunkindustrie tätig sind, wie zum Beispiel Nokia, Motorola oder auch viele grosse Mobilfunkprovider.

3.4.2 Protokollstack

LwM2m ist ein Applikationslayer Protokoll und hat als Basis das CoAP Protokoll. Alle LwM2M Befehle werden umgewandelt, damit diese über CoAP zum Geräte gesendet werden. Dabei werden nicht alle Features von CoAP eingesetzt und unterstützt. CoAP ist wie auch LwM2M speziell für M2M (Maschine-to-Machine) Applikationen entwickelt worden.

Unterhalb von CoAP gibt es drei Varianten, die am gebräuchlichsten sind. UDP, UDP mit DTLS und SMS. Verwendet man nur UDP wird keine Verschlüsselung verwendet. Alle Daten werden in Klartext übertragen. Um eine sichere Verbindung herzustellen muss DTLS eingerichtet werden.

Unterhalb der Transportschicht können wieder viele verschiedene Technologien eingesetzt werden. Dies ist unter anderem IPv4, IPv6, 6LoWPAN oder auch ZigBee IP.

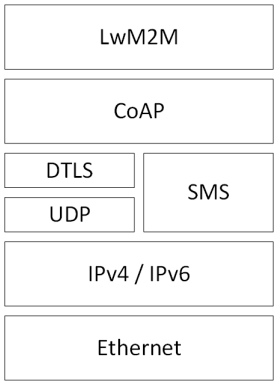


Abbildung 3.10.: LwM2M Stack

3.4.3 Client-Server Model

Die drei zentralen Bestandteile des Protokolls sind der Bootstrap-Server, der LwM2M-Server und der LwM2M-Client. Der Bootstrap-Server ist für den Erstkontakt sowie die Erstkonfiguration zuständig. Die restliche Kommunikation geschieht zwischen dem LwM2M-Server und dem LwM2M-Client. Ohne einen Bootstrap-Server kommunizieren die Geräte direkt mit dem LwM2M-Server.

Client Der Client kann viele verschiedene Formen annehmen. Ein Beispiel wäre eine Linux Installation auf einem Raspberry Pi, welche einen Java LwM2M-Client gestartet hat. An dem Raspberry Pi kann man nun mehrere Sensoren anschliessen und diese über den Client abfragen. Client Umsetzungen gibt es dabei in den Programmiersprachen C und Java.

Server Die Serverumsetzung ist der Clientumsetzung sehr ähnlich. Auf einem Server wird eine Instanz gestartet und diese wartet auf Client, welche sich Registrieren möchten. Da die Serverimplementierung als Libraries in den Programmiersprachen C und Java verfügbar ist, gibt es viele Möglichkeiten den Server zu deployen. So kann er auf einer Windows- sowie auch unter Linux gestartet werden.

Management Server Der Management-Server wird für die Verwaltung von einem oder mehreren LwM2M-Server verwendet. Alle registrierten LwM2M-Clients werden im Management-Server aufgelistet und können durch diesen angepasst werden.

Der LwM2M-Server kann direkt oder über ein Web API angesprochen werden. Je nach Implementierung gibt es beide Varianten. So könnte man mehrere Serverinstanzen zu einem Management Server hinzufügen, um alles zentral zu Verwalten. Möchte man nun ein Gerät managen, schickt man einen Befehl an den zuständigen LwM2M-Server und dieser leitet den Befehl nun an das Device weiter. Die Antwort wird danach bis zum Management Server zurückgegeben und ausgewertet.

Bootstrap-Server Um ein ”factory bootstrap” zu vermeiden, hat LwM2M einen Bootstrap-Server. Dieser ist für die Erstkonfiguration zuständig und macht die Geräte variabler einsetzbar. Auf der Grafik ist dieser nicht erfasst worden.

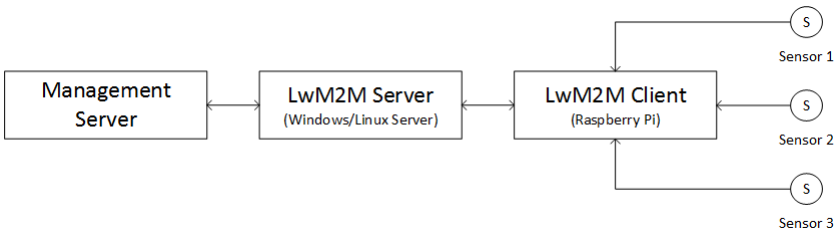


Abbildung 3.11.: Client-Server-Model

3.4.4 Object Model

Eine klare Stärke des LwM2M-Protokolls sind die Object Models. Durch die Object Models werden alle Informationen eines Devices in einem strukturierten Modell abgelegt, damit ein geregelter Zugriff auf alle Daten entsteht. Es wird zwischen Objekt, Instanz und Ressource unterschieden. Eine Ressource ist zum Beispiel der Temperaturwert eines Temperatursensors. Eine zusätzliche Ressource könnte die dazugehörige Temperatureinheit sein. Mehrere solche Ressourcen werden zu einem Object Model zusammengefasst. Ein Device kann aber auch mehrere Temperatursensoren besitzen und hat so mehrere Instanzen des gleichen Objects.

Um nun die Information zugreifbar zu machen, bekommt sie eine URL mit den oben beschriebenen Angaben. Diese kann wie folgt aussehen:

/ObjectID/InstanceID/ResourceID

/3/0/13

Abbildung 3.12.: Object Model URL

Object Jedes Object besitzt eine eindeutige Identifikationsnummer. Diese liegt zwischen 0-32768 und ist folgendermassen verteilt:

- 0-1023: OMA-Label
- 1024-2047: Reserviert für zukünftige Benutzung
- 2048-10240: Registrierungen der Partnerfirmen
- 10241-32768: Individuelle Registrierungen durch Firmen oder Personen

Wenn man ein neues Modell erstellen möchte, kann man dieses im ”LWM2M Management Object Editor” auf der OMA-Seite erfassen. Pro neuem Modell muss man eine Object ID, Namen, ObjectVersion, LWM2MVersion, Object URN, Instances und Mandatory festlegen. Mit ”Instances” gibt man an, ob ein Object pro Device mehrmals vorhanden sein darf.

Instances Wenn ein Object mehrmals vorhanden sein kann, wie zum Beispiel ein Temperaturobjekt, gibt es die Möglichkeit der Multi-Instanz. Der mittlere Teil der URL nimmt dadurch nicht nur den Wert 0 an, sondern eine beliebige ID pro Instanz.

Resource Dem Object werden zum Schluss noch die Ressourcen hinzugefügt. Diese besitzen auch eine eindeutige Identifikationsnummer von 0 bis 32768. Hier gibt es folgende Unterteilung:

- 0-2047: Common Ressources
- 2048-26240: Reusable Ressources
- 26241-32768: Private Ressources

Jede Ressource hat vordefinierte Felder. Dazu gehören Name, Operations, MultipleInstances, Mandatory, Type, RangeEnumeration, Units und Description. Durch all diese Felder kann eine Ressource genau beschrieben werden. Nicht alle Felder sind Pflicht. Wichtig sind vor allem ID, Name und Operations und Type. Durch die Operations gibt man an, was mit dieser Ressource genau gemacht werden kann. Hier kann man zwischen Read, Write, Read-Write und Execute wählen. Mit dem Type Feld gibt man den Type Informationstyp an, wie zum Beispiel String, Float oder Date.

Durch all diese standardisierten Objekte und Ressourcen können alle Informationen eines Gerätes beschrieben werden.

Beispiel XML Hier sieht man ein Beispiel von dem Object Model ”Device” mit der Ressource ”0 - Manufacturer”. Das Object Model besitzt noch weitere Ressourcen, die andere Informationen eines Devices definieren.

```
<LWM2M>
  <Object">
    <Name>Device</Name>
    <Description1><![CDATA[ This LWM2M Object ... ]]></Description1>
    <ObjectID>3</ObjectID>
    <ObjectURN>TBD</ObjectURN>
    <MultipleInstances>Single</MultipleInstances>
    <Mandatory>Mandatory</Mandatory>
    <Resources>
      <Item ID="0">
        <Name>Manufacturer</Name>
        <Operations>R</Operations>
        <MultipleInstances>Single</MultipleInstances>
        <Mandatory>Optional</Mandatory>
        <Type>String</Type>
        <RangeEnumeration />
        <Units/>
        <Description><![CDATA[ Human rea... ]]</Description>
      </Item>
      <Item>...</Item>
    </Resources>
  </Object>
</LWM2M>
```

Beispiel Client In der untenstehenden Grafik sind Ressourcen eins Beispielclient ersichtlich. Dies könnte eine kleine LED-Lampe sein, welche jedes LED separat steuern lässt. Dazu wurden die Objekte 3 und 3311 verwendet, welche bereits vorgefertigt bereitstehen. Da es mehrere LEDs existieren, werden auch mehrere Light-Control-Instanzen benötigt. Jede Instanz steuert so eine einzelne LED. Neben diesen drei Objekten könnten noch weitere vorhanden sein, wie zum Beispiel Firmware oder andere Sensorobjekte.

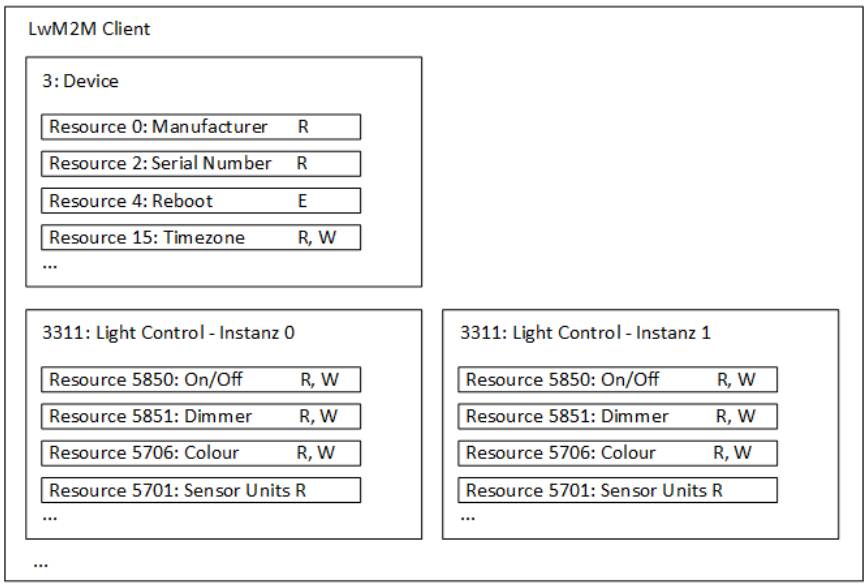


Abbildung 3.13.: LwM2M Stack

3.4.5 Features und Funktionen

LwM2M bietet vier Funktionen Bootstraping, Registration, Object / Ressource Access und Reporting, um mit einem Device zu kommunizieren oder dieses zu konfigurieren. Es müssen aber nicht immer alle Funktionen eingesetzt werden. So kann auf Bootstraping und Reporting verzichtet werden, falls jedes Device von Hand vorkonfiguriert wird.

Bootstrapping - Allgemein Wie bereits erwähnt, ist ”factory bootstrapping” möglichst zu vermeiden. Durch die statisch hinterlegten Schlüssel und Server-URLs müsste man bei jeder Anpassung der Daten physischen Zugriff auf das Device haben. Dieser Ansatz kann in kleinen und nahe beieinander liegenden Umgebungen noch funktionieren, bei einer grosseren Anzahl Devices ist dies aber nicht mehr praktikabel. Daher hat das LwM2M-Protokoll einen Bootstrap-Vorgang definiert.

Der Hersteller muss für das Bootstrapping jedem Gerät ein Endpoint-Name, sowie eine Bootstrap-URL hinterlegen. Alle anderen Informationen werden beim Starten vom Bootstrap-Server bezogen. Wichtige Einstellungen, welche der Bootstrap-Server verteilen kann, sind unter anderem:[24]:

- Schlüssel und Zertifikate für DTLS
- Server URL
- SMS Security Parameter
- Kommunikationsparameter
- Access Control Lists

Durch den Einsatz eines Bootstrap-Servers kann man sich die Management-Arbeit erleichtern und hat ein einheitlicheres und besser verwaltetes Netzwerk von LwM2M-Server und Client.

Bootstrapping - Ablauf Sobald ein Device gestartet wird, überprüft es seine Einstellungen. Es gibt zwei Zeitpunkte, an denen das Device einen Bootstrap-Server kontaktiert. Der erste ist, sobald das Device nur Bootstrap-Server Daten besitzt und keine LwM2M-Server Daten hinterlegt hat.

- die Authentisierung fehlgeschlagen
- der LwM2M-Server gibt keine Antwort

Sobald einer dieser Punkte zutrifft, meldet sich das Device wieder beim Bootstrap-Server, um eine neue Konfiguration zu erhalten.

In der folgenden Grafik sieht man den Ablauf des Bootstrappings.

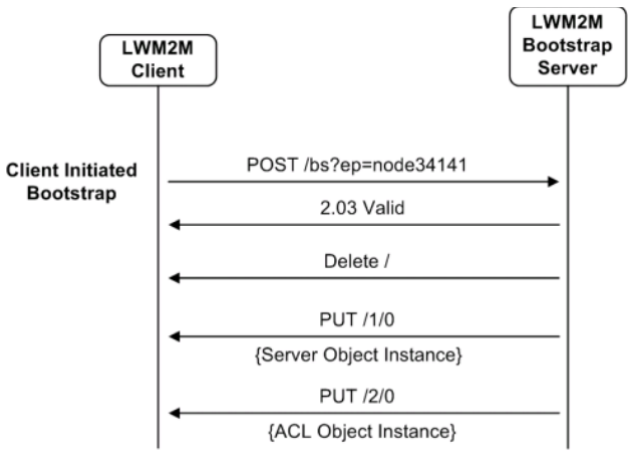


Abbildung 3.14.: LwM2M Bootstraping[25]

Als ersten Schritt sendet das Device eine Anfrage an den Bootstrap-Server. In dieser Anfrage gibt der Client seinen Endpoint-Namen an. Durch diesen Namen weiss der Bootstrap-Server, welche Konfiguration für dieses Gerät vorgesehen ist. Der Server antwortet nun mit ”Valid” oder ”Invalid” . Durch einen Delete-Befehl auf die Root-URL löscht es alte Konfigurationen. Danach werden durch einzelne PUT-Befehle alle wichtigen Daten auf das Gerät geschrieben.

Wenn der Server eine neue Konfiguration, wie zum Beispiel neues Schlüsselmaterial, verteilen möchte, löscht der LwM2M-Server durch einen Delete-Befehl die Root-URL auf dem Client. Dadurch meldet sich der Client wieder beim Bootstrap-Server, um die neuen Daten zu erhalten, dies nennt man den Server initiierten Bootstrap-Vorgang.

Zero Touch Provisioning Seit dem 26. Dezember 2016 gibt einen Internet-Draft mit dem Namen ”DHCPv6 Options for LWM2M bootstrap information”. Dieser Draft beschreibt ein Mechanismus, bei welchem die LwM2M-Bootstrap Informationen durch eine neue DHCPv6-Server Option verteilt wird. Durch diesen Mechanismus wäre ein Zero Touch Provisioning umsetzbar und man müsste keine weitere Einstellungen vornehmen. Sobald sich die Client beim DHCP melden, erhalten sie alle wichtigen Informationen um sich selbst zu konfigurieren oder die restlichen Konfigurationen vom richtigen Bootstrap-Server zu erhalten.

https://tools.ietf.org/html/draft-nalluri-dhc-dhcpv6-lwm2m-bootstrap-options-02

Registration - Allgemein Wenn dem Client ein gültiger und erreichbarer LwM2M-Server hinterlegt wurde, beginnt die Phase der Registrierung. Bei der Registrierung wird dem Server bekanntgeben, welche Funktionen und Informationen der Client besitzt. Dies beinhaltet die Object Model URLs, Registrierungszeitpunkt, IP-Adresse, Port und noch weitere Angaben, die der Server von jedem Client benötigt. All diese Angaben werden vom Client immer wieder aktualisiert und an den Server gesendet. Dies geschieht meistens mehrmals pro Minute.

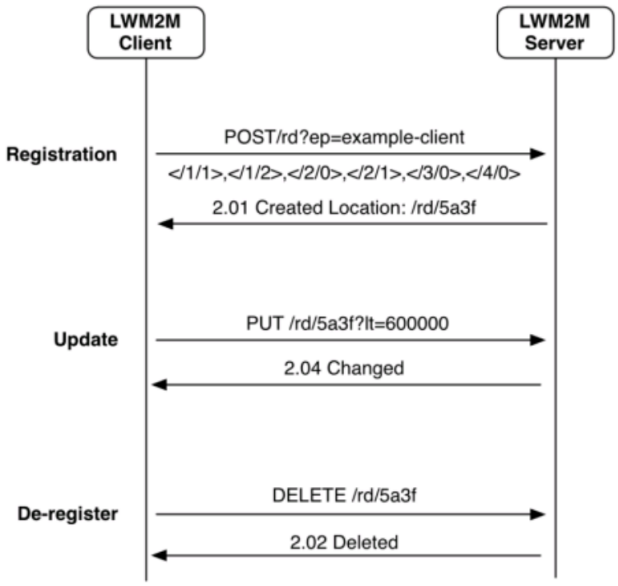


Abbildung 3.15.: LwM2M Registrierung[25]

Registration - Ablauf Die Registrierung wird immer vom Client aus initiiert. Dieser sendet seinem hinterlegtem LwM2M-Server einen POST-Request mit allen relevanten Daten. Sobald diese gesendet wurden, wartet der Client auf eine Antwort. Der Server checkt nun diese Daten und checkt auch gleichzeitig, ob der Client sich richtig authentifiziert hat. Sobald alle Daten korrekt sind, legt der Server eine Registrierung an.

Nach einer gewissen Zeit meldet der Client ein Update an den Server. Durch einen PUT-Request auf die für ihn hinterlegte URL kann der Client die angepassten Daten an den Server senden. Wird ein Client heruntergefahren deregistriert sich dieser automatisch bei seinen Servern. Dazu sendet der Client ein Delete-Befehl auf seine URL auf dem Server und erhält ”Deleted” als Antwort.

Object / Ressource Access - Allgemein Nach dem Registrieren kann der Server auf alle für ihn freigegebenen Daten zugreifen. Dies geschieht über vier definierten Anfragen Read, Write, Execute und Observe. Um neue Daten auf den Client zu senden, erlauben die Write und Execute Anfragen zusätzliche Daten. All diese Anfragen an den Client werden vom LwM2M-Server in ein CoAP Paket umgewandelt und so an den Client gesendet.

Object / Ressource Access - Ablauf Alle vier Varianten laufen sehr ähnlich ab und der Initiator ist immer der Server. Der Client kann keine Reads oder Writes an den Server senden.

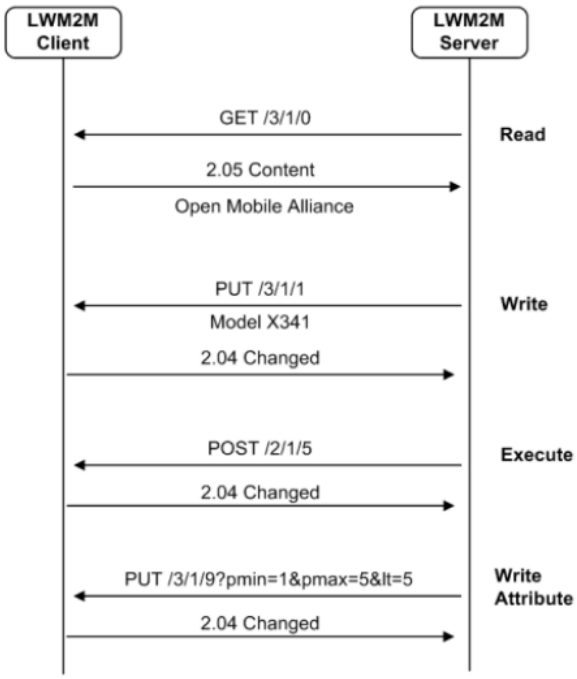


Abbildung 3.16.: LwM2M Ressource Access[25]

In der Grafik sind drei Beispielanfragen ersichtlich.

Die erste Anfrage ist ein Read-Request. Der Server sendet ein GET-Request auf die gewünschte Ressource, in diesem Beispiel ist dies 3/1/0 und wartet auf die Response. Der Client überprüft nun seine ACLs und wenn der Server berechtigt ist, sendet der Client eine Response mit dem Content zurück. Wenn die Ressource nicht vorhanden-, oder der Server nicht berechtigt ist, erhält der Servier eine Fehlermeldung als Response. Dies könnte zum Beispiel ”Not Found” sein, falls die Ressource nicht vorhanden ist.

Der Write-Request ist sehr ähnlich aufgebaut. Anstelle eines GET-Requests sendet der Server ein PUT-Request. In diesem Request wird wieder die gewünschte Ressource-URL angesprochen und die zu schreibenden Daten werden direkt mit dem Request mitgesendet.

Auch der Execute-Befehl funktioniert ähnlich wie die anderen zwei Requests. Anstelle von GET- oder PUT-Request wird hier ein POST-Request verwendet.

Reporting - Allgemein LwM2M bietet einen Observe-Befehl an. Dieser wird mit dem Reporting Interface beschrieben. Reporting bietet dabei die drei Funktionen, Observe, Notify und Cancel Observation an. Bei jeder Zustandsänderung der Ressource meldet sich der Client beim Server und sendet die neuen Daten. Der Server kann so stetig wechselnde Daten sehr einfach überwachen.

Reporting - Ablauf Im Grunde wird hier ein normales Observer-Pattern umgesetzt. Der LwM2M-Server ist der Observer und der Client ist das Subject. Initiiert wird dieser Vorgang immer vom LwM2M-Server aus.

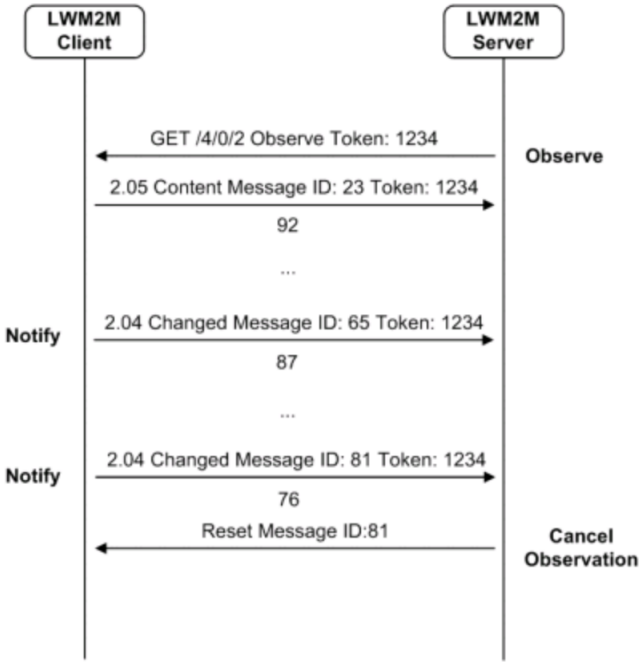


Abbildung 3.17.: LwM2M Reporting[25]

Um eine Ressource zu überwachen, sendet der Server ein GET-Request an die gewünschte Resource-URL mit einem Observe-Token. Direkt nach dem Anmelden des Observers meldet der Client den aktuellen Wert der Ressource mit einer Content Message ID und dem Observer Token.

Bei jeder Änderung der Ressource erstellt der Client wieder ein neuer Response mit einer Changed Message ID und dem Token und sendet diese an alle Observer zurück.

Möchte der Server die Ressource nicht länger Überwachen, sendet der Server ein Reset zurück. Dieser Reset beinhaltet die letzte Changed Message ID. Hier im Beispiel sieht man dies ganz am Schluss der Grafik. So meldet sich der Server ab und erhält keine weiteren Notifikationen.

3.4.6 Fazit

Denkt man an die vielen verschiedenen Protokolle und deren Management-Funktionen, wie umständlich Kommunikation mit Devices werden könnte. Durch LwM2M hat man einen durchdachten Standard, welche das Management enorm vereinfacht. Durch die eindeutigen XML-Definitionen ist jede Ressource genau spezifiziert.

Ein weiterer Vorteil des LwM2M-Protokolls sind die vielen Client- und Serverumsetzungen. Es gibt bereits mehrere in Java und C geschriebene Libraries, welche diesen Protokoll bereits umsetzen.

Bei all den positiven Argumenten muss aber bedenkt werden, das sich LwM2M noch mitten in der Entwicklung befindet. Momentan gibt es noch eine überschaubare Anzahl an Unternehmen, welche die Entwicklung solcher IoT-Devices vorantreibt. Wie bei vielen anderen Protokollen muss sich erst noch zeigen, ob sich LwM2M als Standard für den Bereich Management durchsetzen wird.

4. IoT-Security

In diesem Kapitel sollen wichtige Aspekte der Security im IoT-Umfeld erarbeitet werden. Verschiedene Bereiche in der Architektur und deren Herausforderungen bezüglich Security erfordern unterschiedliche Massnahmen. Es wird versucht, zentrale Aspekte hervorzuheben, eine umfassende Behandlung ist in diesem Rahmen jedoch nicht möglich.

4.1 Einführung

Sicherheit ist im IoT-Bereich in den letzten Jahren immer wichtiger geworden. Viele Hersteller wollen ihre Produkte schnellstmöglich auf den Markt bringen und beachten Sicherheitsanforderungen zu wenig. Laut einer Studie von HP kamen bei den meisten Produkten schwerwiegende Sicherheitsbedenken auf.[26]

Die Vielfalt an Geräten bringt eine grosse Angriffsfläche mit sich. Verantwortliche müssen bei jedem Hersteller umfassende Sicherheitsanalysen durchführen. Im Gegensatz zu traditionellen Firmennetzwerken sind Netzwerke mit IoT-Devices schwieriger abzusichern. Bisher haben fast ausschliesslich Menschen die Kommunikation in Netzwerken verursacht. Maschine-zu-Maschine (M2M) Kommunikation ist in IoT-Netzwerken zentral. Menschen kommunizieren nur in seltenen Fällen (Konfiguration, Fehlerbehandlung, etc.) direkt mit den End-Devices. Die IoT-Devices haben dennoch ständige Kommunikation über das Netzwerk oder gar das Internet. Eine Kompromittierung dieser Systeme ist also theoretisch möglich.

4.1.1 Folgen

Die Vergangenheit hat gezeigt, dass Sicherheitsvorfälle schwerwiegende Folgen haben können. Reputationsverluste, Datenverluste, Industriespionage oder Systemausfälle durch Denial-of-Service (DoS) können Unternehmen beträchtlichen finanziellen Schaden zufügen. Wie seit einigen Jahren bekannt ist, haben selbst Regierungen von Weltmächten wie China, Russland oder die USA Interesse an der illegalen Beschaffung von Daten und Informationen. Die Motivation und technischen Fähigkeiten von Angreifern auf IT-Systeme sind sehr unterschiedlich. Ebenso unterscheiden sich die Sicherheitsbedürfnisse von Unternehmen stark.

Durch Entwicklungen im IoT-Umfeld muss damit gerechnet werden, dass IT-Systeme noch viel weiter als heute in Unternehmensprozessen eingebunden werden. Mit der Vision der Industrie 4.0 werden ganze Geschäftsprozesse vollautomatisiert und manuelle Tätigkeiten werden weitestgehend eliminiert. Wenn man von automatisierten Produktionsstätten ausgeht, kommen einige Gefahren zum Vorschein. Die Konkurrenz könnte beispielsweise durch gezielte Attacken auf ein Fertigungssystem Fehler einprogrammieren, welche zu grossen Rückrufaktionen und Reputationsverlusten führen könnten. Nochmals eine Stufe gefährlicher sind Attacken, welche Leib und Leben gefährden können. Denkt man an den medizinischen Bereich, könnten Patientensysteme, welche für die Medikation von Patienten zuständig sind, übernommen werden.

Es ist also ersichtlich, dass zu den herkömmlichen finanziellen Schäden bei IoT-Systemen Gefährdungen für Leib und Leben existieren können. Gesetzgeber wie auch Ingenieure müssen sich mit diesen wichtigen Herausforderungen intensiv befassen.

4.1.2 Grobübersicht

Um eine möglichst übersichtliche Vorstellung von IoT-Security zu erhalten, werden vier grobe Architekturbereiche einzeln behandelt. Jeder dieser Bereiche birgt eigene Gefahren und benötigt unterschiedliche Massnahmen um die Informationssicherheit zu gewährleisten.

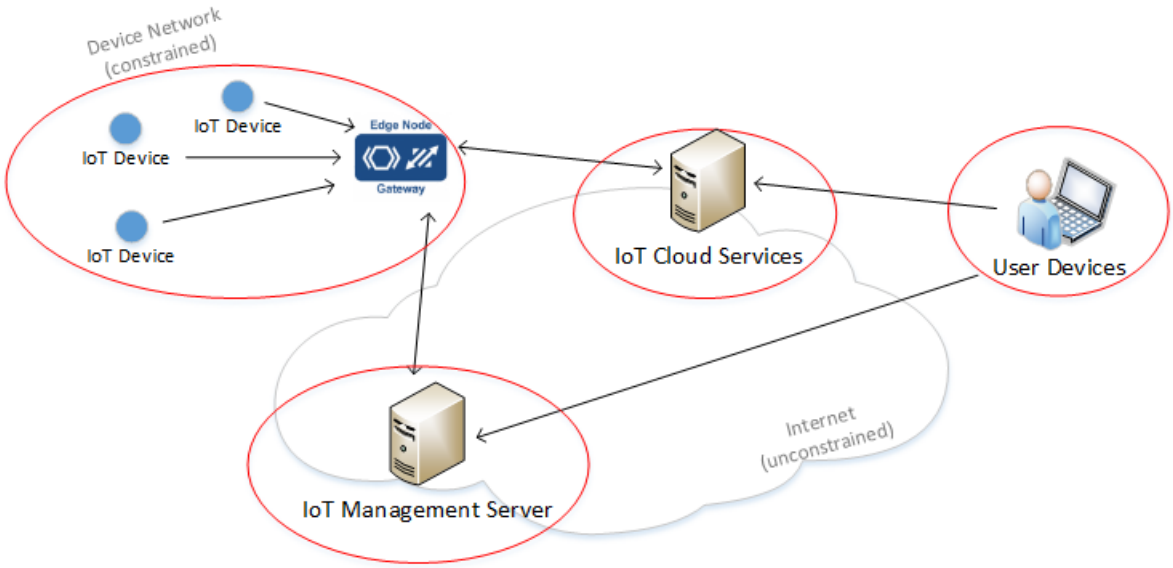


Abbildung 4.1.: Sicherheitszonen

Device-Networks Device-Netzwerke beinhalten die IoT-Devices. Diese können sehr unterschiedlich aussehen. Denkbar wären Wireless Sensor Netzwerke (WSN), es könnten aber auch ”herkömmliche”, verkabelte Netzwerke sein. Die Heterogenität ist sehr gross, da nicht nur viele unterschiedliche Hersteller existieren, sondern auch die Arten der Kommunikation und die verwendeten Protokolle sich unterscheiden.

IoT-Cloud-Services IoT-Devices selbst kommunizieren häufig über Cloud-Services im Internet. Entweder stellen die Hersteller der Devices selbst Cloud-Services zur Verfügung, oder die Unternehmen entwickeln eigene Applikationen. Auf der einen Seite kommunizieren die Server mit den IoT-Devices oder deren Gateways, auf der anderen Seite werden die Services von End-Usern selbst benutzt. Sensordaten könnten auch über bereitgestellte APIs von anderen Cloud-Services konsumiert werden.

User-Devices Benutzer selbst kommunizieren in den seltensten Fällen direkt mit IoT-Devices sondern über bereitgestellte Cloud Dienste. Häufige Devices sind PCs, Laptops und Mobilgeräte wie Smartphones oder Tablets.

IoT-Management-Services Unternehmen möchten ihre IoT-Devices über einen zentralen Service verwalten. Häufig liefern Hersteller eigene Management Software, diese beschränken sich aber oft auf die Verwaltung von Geräten derselben Hersteller.

4.2 Informationssicherheit

Nach dem Parker'schen Hexad befasst sich die Informationssicherheit grundlegend mit sechs Bereichen: [27]

Vertraulichkeit (Confidentiality) Die Information muss geheim bleiben und darf von Unbefugten nicht einsehbar sein. Die Vertraulichkeit werden zum einen mittels kryptografischen Funktionen sichergestellt, zum anderen durch Berechtigungen. Informationen werden verschlüsselt, die vorgesehenen Entitäten sind im Besitz der Schlüssel um an die Informationen zu gelangen. Verschlüsseln von Informationen wird seit Tausenden von Jahren verwendet. Es existieren unterschiedliche Algorithmen und Schlüssellängen. Verwendete Techniken sollten regelmässig auf deren Aktualität und Sicherheit überprüft werden.

Besitz oder Kontrolle (Possession or Control) Wenn ein Unbefugter in den Besitz oder die Kontrolle der Information bekommt so muss nicht zwangsweise die Vertraulichkeit verletzt sein. Wird beispielsweise ein Laptop oder eine Kreditkarte gestohlen, so erhält der Dieb diesen Zugriff, obwohl er nie vorgesehen wurde. Um sich vor diesen Gefahren zu schützen, empfiehlt sich eine Multi-Faktor Authentisierung, bei der Besitz und geheimes Wissen benötigt wird.

Integrität (Integrity) Bei der Einhaltung der Integrität möchte man die unbefugte oder unbemerkte Veränderung der Daten verhindern. Integritätschecks werden meistens mit kryptografischen Hashfunktionen durchgeführt. Dabei wird mittels einer mathematischen Einwegfunktion ein sogenannter Hashwert einer Eingangsinformation erstellt. Dieser Hashwert ist praktisch einmalig und schwierig reproduzierbar, weshalb bei einer Veränderung der Eingangsinformation ein anderer Hashwert resultiert.

Echtheit (Authenticity) Die Echtheit einer Information ist in vielen alltäglichen Bereichen wichtig. So möchte man sicherstellen, dass eine erhaltene Rechnung auch wirklich von der Firma stammt, von welcher man die Leistung bezogen hat. Bei E-Mails und dem zugrunde liegenden SMTP-Protokoll sind Probleme der Authentizität vielen Personen bekannt. Die Kryptografie hat Verfahren für digitale Signaturen hervorgebracht. Mit einem geheimen Schlüssel kann eine Information signiert werden, der dazugehörige öffentliche Schlüssel dient zur Verifikation der signierten Nachricht.

Verfügbarkeit (Availability) Die Sicherheit ist ebenfalls nicht gewährleistet, wenn Informationen nicht verfügbar sind. Durch Denial-of-Service (DoS) Attacken, aber auch durch schlechte Planung oder unvorhergesehenen Ausfällen können Verfügbarkeitsprobleme auftreten. Durch Hochverfügbarkeitscluster können Ausfälle einzelner Knoten aufgefangen werden und die Wahrscheinlichkeit von Ausfällen drastisch reduziert werden.

Nützlichkeit (Utility) Als Beispiel für die Verletzung dieses Aspekts kann man sich am besten ein vergessenes Passwort vorstellen. Sämtliche anderen Sicherheitsaspekte sind erfüllt, die Information ist trotzdem nicht zugänglich, da der benötigte Schlüssel (das Passwort) fehlt. Solche Fälle können durch Key-Recovery Mechanismen abgefangen werden. [28]

4.3 Device-Networks

In diesem Unterkapitel werden Gefahren und Herausforderungen in Netzwerken mit IoT-Devices beschrieben.

4.3.1 Device-Security

Software Versionen Ein wichtiger Aspekt für Unternehmen ist die Aktualität der verwendeten Software Versionen auf ihren IoT-Devices. Wie heute üblich, wird Software oft unfertig, mit Bugs und fehlenden Features ausgeliefert. Nach der Installation muss also schon fest mit kommenden Updates gerechnet werden. Die Unternehmen müssen also über ihre Softwarestände in den IoT-Systemen im Bilde sein und über die Möglichkeit von zeitnahen Updates verfügen.

Physischer Zugriff Wie bei vielen anderen Geräten sollte auch bei IoT-Devices der physische Zugang nur autorisierten Personen vorbehalten sein. Sonst könnte beispielsweise ein Sensor manipuliert werden, der falsche Daten liefert.

Authentifizierung und Autorisierung Da ein IoT-Device über das Internet kommuniziert, sollten Zugriffe authentifiziert und autorisiert werden. Durch unbefugten Zugriff können entweder potenziell sensible Sensordaten gelesen werden. Ausserdem besteht die Möglichkeit, einem Botnetz beizutreten, was mit geschützten Zugriffen teilweise entschärft werden kann. Es sollten ausserdem Mechanismen vorgesehen werden, welche zuständige Personen bei zu häufigen Fehlversuchen benachrichtigen, da potenziell eine Attacke vorliegen könnte.

4.3.2 Netzwerksicherheit

Bootstrapping In der Bootstrapping-Phase betritt ein neues IoT-Device ein bestehendes Netzwerk. Eine Security-Association (SA) zwischen dem IoT-Device und dem Netzwerk, resp. deren Devices muss erstellt werden. Als Betreiber des Netzwerks muss man sicherstellen, dass nur vorgesehene Devices beitreten können, deshalb ist eine Authentisierung sehr wichtig. Das IoT-Device muss im Gegensatz dem Netzwerk vertrauen. Ein Server kann über Protokolle wie beispielsweise EAP neue Devices authentisieren.

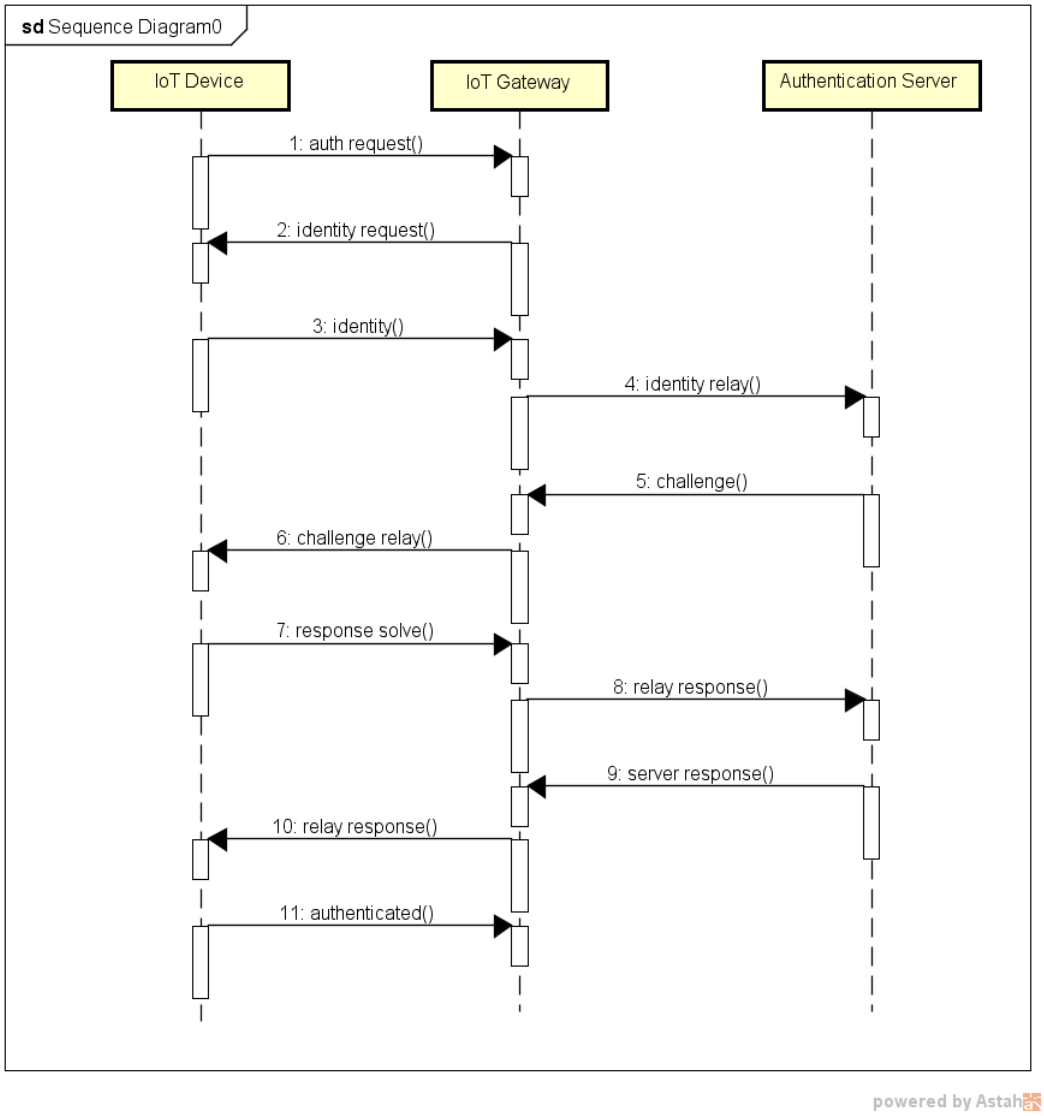


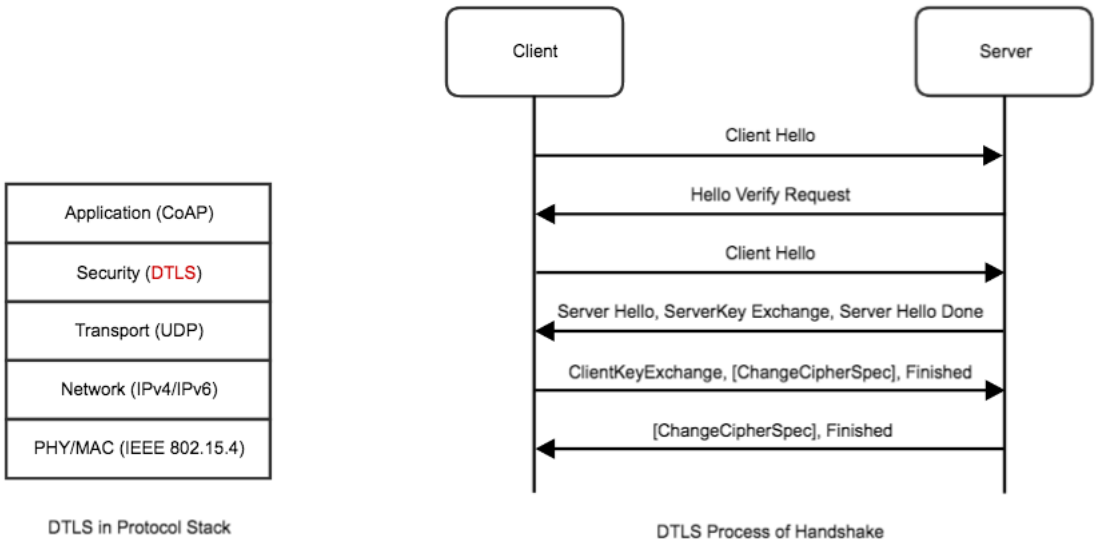
Abbildung 4.2.: Schema möglicher Deviceauthentisierung

Im Gegensatz zu herkömmlichen Geräten, bei denen die Identität beispielsweise durch die Eingabe eines Passworts oder der Verteilung von Zertifikaten mittels einer PKI festgestellt werden kann, kommen bei IoT-Geräten erschwerende Hürden hinzu. [7] So könnten Devices beispielsweise über keine direkten Eingabemöglichkeiten-, oder zu wenig Rechenleistung für herkömmliche asymmetrische Kryptografie verfügen.

Verschlüsselung Weder das HTTP-, noch das CoAP-Protokoll haben eine eigene Verschlüsselung des Payloads vorgesehen. Seit langer Zeit wird für HTTP SSL, respektive TLS verwendet, um die Kommunikation zwischen zwei Devices zu verschlüsseln. TLS verschlüsselt die gesamten Pakete in der Applikationsschicht, Protokolle unterhalb der Applikationsschicht sind weiterhin im Klartext verfügbar.

IoT-Devices arbeiten häufig mit CoAP anstatt HTTP. Wie in Kapitel 3.3.2 beschrieben, arbeitet CoAP im Gegensatz zu HTTP mit UDP anstelle von TCP. TLS selbst benötigt jedoch TCP als Transportprotokoll, weshalb sich dieses Verfahren also nicht für CoAP eignet.

Aus diesen Gründen musste ein neues Verschlüsselungsprotokoll DTLS (Datagram Transport Layer Security) entwickelt werden, welches "TLS" über UDP ermöglicht. Die Unterschiede von TLS zu DTLS sind gering. Einfach betrachtet erledigen die Protokolle die gleichen Aufgaben.



[29]

Abbildung 4.3.: DTLS Sequenz Diagramm

4.4 Cloud-Services

In diesem Unterkapitel werden sicherheitsrelevante Aspekte für IoT-Cloud-Dienste behandelt. Neben den behandelten Themen müssen viele weitere Aspekte, welche für übrige Internetdienste gelten auch eingehalten werden.

4.4.1 Bedeutung

IoT-Cloud-Dienste kommunizieren mit Sensoren und empfangen deren Daten. IoT-Cloud-Applikationen steuern ganze Geschäftsprozesse, deshalb sind sie in diesem Umfeld oft businesskritisch. Cloud-Services kommunizieren auf der einen Seite mit IoT-Devices (M2M-Kommunikation), auf der anderen Seite mit End-User Devices.

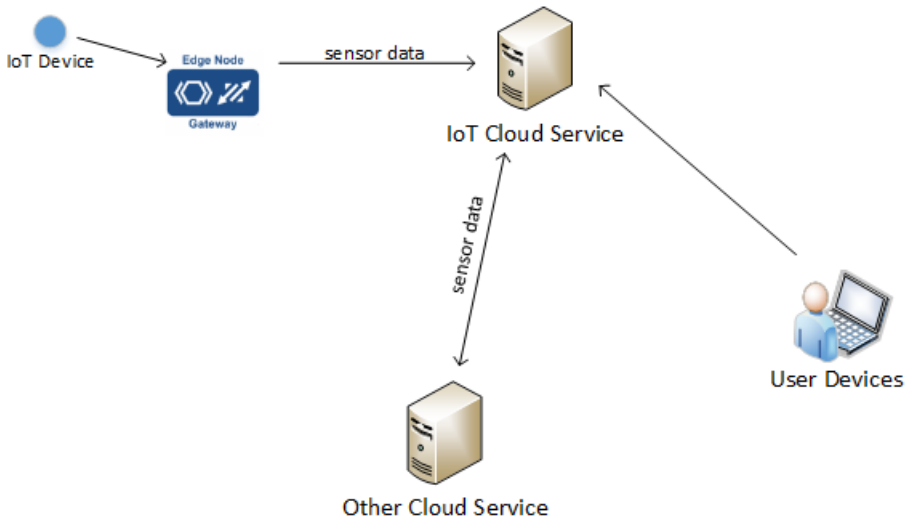


Abbildung 4.4.: IoT-Cloud-Service Übersicht

Wie in der Einführung bereits erwähnt, können die Folgen von Ausfällen je nach Applikation sehr unterschiedlich sein. Eintrittswahrscheinlichkeiten und Schadensausmasse müssen in einer Risikoanalyse bewertet werden, damit geeignete Massnahmen getroffen werden können. So benötigen Patientensysteme in Spitälern weitaus mehr Sicherheitsmassnahmen als Pflanzenüberwachungssysteme in Gewächshäusern.

4.4.2 API-Security

Kommunikation und Austausch von Daten erfolgen über bereitgestellte APIs. Sämtliche API Zugriffe müssen zuerst authentifiziert werden, ansonsten können möglicherweise geheime Daten abgefragt-, oder gar fehlerhafte und schädliche Inputs eingegeben werden. Mittels Berechtigungen sollte festgelegt werden, wer auf welche Daten zugreifen darf.

4.4.3 Korrektheit

Je nach Sensitivität der Cloud-Applikation müssen Softwareentwickler ein erhöhtes Augenmerk auf die Korrektheit der Software legen. Die Vergangenheit hat gezeigt, dass Softwarefehler wie falsches Exception Handling oder Race Conditions weitreichende Folgen haben können. Hängt beispielsweise eine ganze Produktionskette von der Cloud-Applikation ab, so könnte ein Absturz innert kürzester Zeit einen sehr hohen finanziellen Schaden bedeuten.

4.4.4 Verfügbarkeit

Wie bei herkömmlichen Servern muss auch bei Applikationsserver im IoT-Bereich auf die Verfügbarkeit geachtet werden. Es gibt potenziell viele Vorfälle, welche die Verfügbarkeit beeinträchtigen könnten. Als wirksamste Methode empfiehlt sich eine redundante Auslegung der wichtigsten Komponenten.

4.5 User-Devices

In diesem Unterkapitel werden Gefahren und Massnahmen für End-User Devices behandelt. Über Geräte PCs, Notebooks, Tablets oder Smartphones werden IoT-Services konsumiert. Für Unternehmen stellen sich seit dem ”bring-your-own-device”-Zeitalter die Herausforderung, dass nicht nur User Devices im Firmenbesitz auf interne Anwendungen zugreifen. Network-Access-Control-Systeme überprüfen diverse Attribute wie beispielsweise Virendefinitionen von Devices an einem Enforcement-Point, bevor in den geschützten Bereich zugegriffen werden kann. Grundsätzlich unterscheiden sich die empfohlenen Security-Massnahmen für User-Devices für IoT-Anwendungen nicht von herkömmlichen Systemen, da potenzielle Schadensausmasse jedoch höher sind, sollten empfohlene Massnahmen konsequent umgesetzt werden.

4.5.1 Client-Applikationen

Unterschiedliche Arten von Client-Applikationen sind für IoT-Anwendungen denkbar. Herkömmlich installierte Desktop-Programme für Windows-Betriebssysteme verlieren an Bedeutung. Web- und Mobil-Applikationen haben in den letzten Jahren stark zugenommen.

Ein Benutzer muss sich an der Client-Anwendung authentisieren. Wie hinlänglich bekannt, muss die Authentifizierung an einem vorgesehenen Server stattfinden. Für sensitive Anwendungen sollten Multi-Faktor-Authentifizierungslösungen eingesetzt werden. Neuere Mobilgeräte und Notebooks verfügen häufig über Fingerabdruckscanner, es wären aber auch SMS-Tokens oder Smartcards als weitere Möglichkeit neben Passwörtern denkbar.

Mobile Geräte werden oft verloren oder gestohlen. Sobald eine Session beendet wurde, sollte eine erneute Authentifizierung verlangt werden. Es sollte ein möglichst kurzes Session-Timeout gewählt werden, damit beim Vergessen des Logouts fremde Personen keinen Zugriff erhalten können.

Viele sicherheitsrelevante Überprüfungen von Anwendungen wie Input-Validierung müssen serverseitig implementiert werden, da man clientseitige Security-Checks leicht umgehen kann. Es gibt jedoch Angriffe wie Phishing oder Social Engineering, welche selbst serverseitig nicht abgefangen werden können. Gegen solche Angriffe schützt man sich am besten mit Benutzerschulungen.

4.6 Management-Server

In diesem Abschnitt wird Security für Management Server behandelt. Ziel dieses Projektes ist ein Prototyp eines Management Servers zu erstellen. Es ist wichtig, bereits vorgängig Security fest einzuplanen, da die Kosten für Security-Vorfälle mit der Zeit zunehmen.

4.6.1 Bedeutung

Management Server sind für Unternehmen sicherheitskritische Systeme. Sie gelten als besonders schützenswert, da viele Applikationen und ganze Geschäftsprozesse abhängig sein könnten. Ist der IoT-Management-Server kompromittiert, so erhält der Angreifer eine Komplettübersicht des gesamten IoT-Systems eines Unternehmens. Als Angreifer könnte man dann beispielsweise versucht sein, Devices herunterzufahren, fehlerhafte Konfigurationen einzuspielen, oder sensitive Informationen zu gewinnen.

4.6.2 User Interface

Es muss die Frage gestellt werden, ob Zugriffe aus dem Internet notwendig sind. Allenfalls könnten externe Benutzer über ein Virtual Private Network (VPN) zugreifen. Angreifer aus dem Internet hätten somit keinen direkten Zugriff ausserhalb des Firmennetzwerks.

Da sensitive Informationen wie Kennwörter und weitere Daten übertragen werden, sollte der Netzwerkverkehr (auch firmenintern) verschlüsselt werden. Wird zum Beispiel ein Web-Interface verwendet, so sollte jeglicher Verkehr über das TLS gesicherte HTTPS-Protokoll geschehen. Unverschlüsselte Verbindungen dürfen nicht erlaubt werden.

Inputvalidierung ist bei Applikationen enorm wichtig. Sicherheitsrelevante Validierungen müssen zwingend serverseitig implementiert werden. Durch saubere Input Validierungen werden häufige Angriffe wie Code Injection, XSS und CSRF Attacken verhindert.

4.6.3 Devicekommunikation

Der Management Server kommuniziert über eigene Sessions mit IoT-Devices. Meldet sich ein Device beim Management Server, so muss dieser zuerst überprüfen, ob das Device berechtigt ist, um mit dem Management Server zu kommunizieren. Auf dem Server könnte beispielsweise ein Passwort hinterlegt werden, welches neuen Devices bekannt sein muss.

Man könnte auf dem Server auch vertrauenswürdige Root Zertifikate hinterlegen. So könnten Clients, welche Zertifikate von einem dieser vertrauten Root Zertifikate ausgestellt erhalten haben auf den Management Server zugreifen. Der lokale Bootstrap-Server könnte solche Clientzertifikate an die IoT-Devices ausstellen.

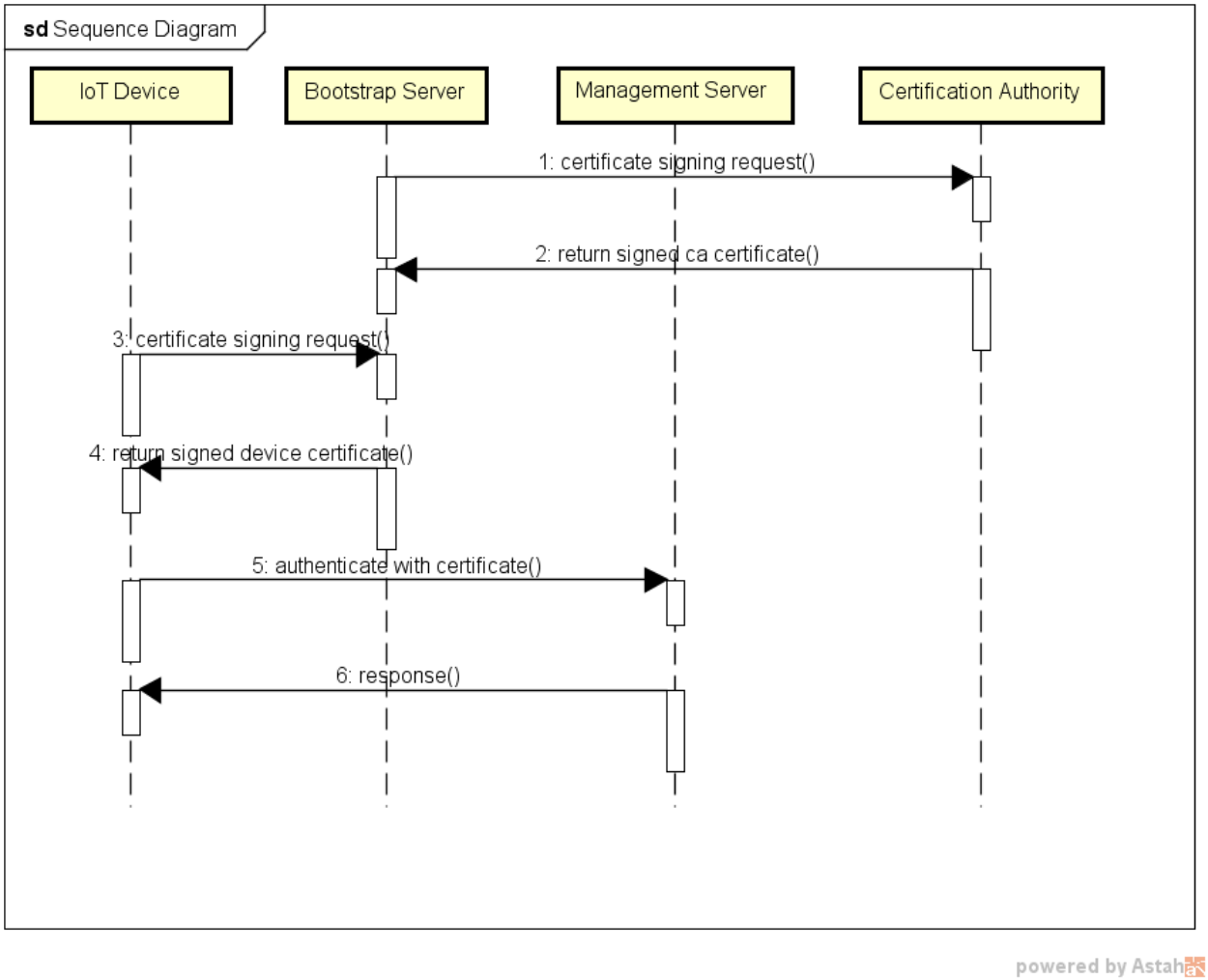


Abbildung 4.5.: Authentisierung mit Zertifikaten

IoT-Devices übermitteln unter Umständen sensitive Daten an den Management Server. Die Kommunikation muss deshalb verschlüsselt werden.

4.6.4 Server

Die Management Applikation sollte über ein sicheres Authentifizierungsverfahren verfügen. Für kritische Applikationen lohnt sich eine Zwei-Faktor-Authentifizierung. Viele Benutzer verwenden überall die selben Kennwörter oder schreiben sie sich auf, weshalb ein Verlust des Kennworts nicht sehr unwahrscheinlich ist. Durch eine zweite Authentifizierungshürde verbessert sich die Sicherheit markant.

Bereitgestellte APIs müssen geschützt sein. Nur authentifizierte Benutzer sollten zugreifen dürfen. Die Kommunikation über diese APIs muss gesichert sein, dass keine schädlichen Daten übertragen werden können.

Auch für Management Server ist die Verfügbarkeit wichtig. Hardwarekomponenten sollten redundant vorhanden sein. Der Applikationsserver sollte regelmässig gesichert sein, sodass im Totalausfall durch einen Restore die Downtime möglichst kurz gehalten wird.

5. Requirements

5.1 Allgemeine Beschreibung

5.1.1 Produktperspektive

Mit Internet of Things sind eine Vielzahl neuartiger Devices entstanden. Während in herkömmlichen Netzwerken hauptsächlich Personal Computer, Notebooks, Server usw. verwaltet werden mussten, so bringen IoT-Devices den IT-Abteilungen neue Herausforderungen. Zum einen dürfte die Anzahl Geräte gegenüber herkömmlichen Computer deutlich ansteigen, zum anderen sind IoT-Devices in Sachen Funktionalität und Rechenleistung, sowie auch der Netzwerkbandbreite deutlich beschränkt.

Mit diesem Projekt soll ein Prototyp einer Management Applikation bereitgestellt werden, um eine grosse Anzahl unterschiedlicher IoT-Devices administrieren zu können.

5.1.2 Produktfunktionen

Die Applikation soll den Benutzern erlauben, IoT-Geräte zu verwalten. Die Aufgaben reichen vom Discovery von Devices über die Konfigurationsverwaltung und Softwareverteilung bis zu Backup und Restore. Ausserdem sollen Management-relevante Kommandos auf Devices ausgeführt- und Security Aspekte beachtet werden. Die Details zu den Produktfunktionen sind den Use Cases zu entnehmen.

5.1.3 Benutzer Charakteristik

Zielpersonen der Applikation sind Betreiber von IoT-Devices. Dies können im Enterprise Umfeld IT-Mitarbeiter in operationeller Funktion-, oder auch Softwareentwickler für IoT-Applikationen sein. Heimanwender können bei entsprechenden Kenntnissen ebenfalls zur Zielgruppe gehören. Es werden solide Grundkenntnisse in TCP/IP Netzwerken sowie Verständnis der verwendeten IoT-Architekturen und Devices vorausgesetzt. Es werden ausserdem Grundkenntnisse des LwM2M-Protokolls empfohlen.

5.2 Use Cases

5.2.1 Use Cases Diagramm

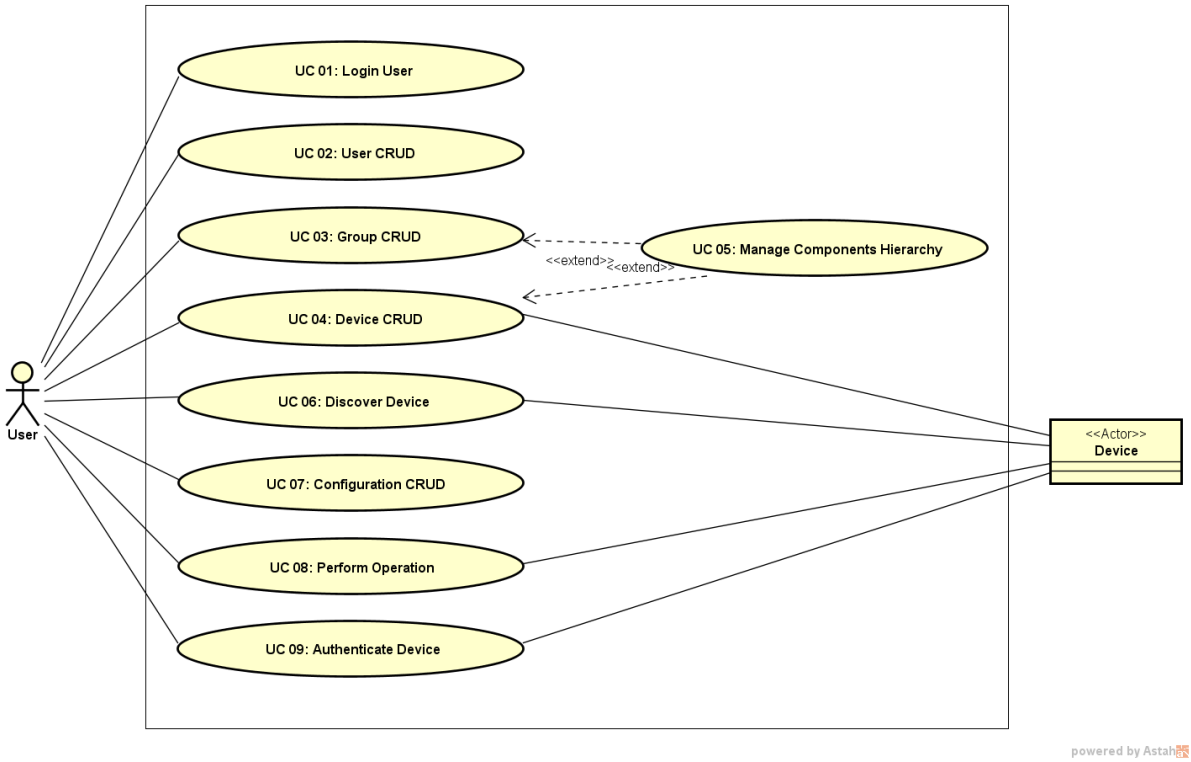


Abbildung 5.1.: Use Case Diagramm

5.2.2 Aktoren

Der Benutzer der Applikation ist in diesem System der einzige primäre Aktor. Dieser bewirtschaftet die Applikation und verwaltet alle Devices und Benutzer. Als sekundärer Aktor werden die einzelnen Devices gezählt.

5.2.3 Beschreibungen (Casual)

Login User

ID	01
Name	Login User
Beschreibung	Der Benutzer loggt sich in die Applikation ein.
Preconditions	• Applikation gestartet • Benutzer im System angelegt
Postconditions	• Benutzer kann auf dem Management Startseite interagieren.
Main Success Scenario	1. Loginseite erscheint beim Start der Applikation 2. Benutzer gibt Benutzername und Passwort ein 3. Benutzer wählt „Login“. 4. Benutzer ist eingeloggt.
Extensions	4.a Fehlermeldung bei falschem Login erscheint.

User CRUD

ID	02
Name	User CRUD
Beschreibung	Benutzerverwaltung der Applikation
Preconditions	• Applikation gestartet • Benutzer ist registriert • Benutzer ist eingeloggt
Postconditions	• Änderungen gespeichert
Main Success Scenario	**Create:** 1. UC 01: Benutzer registrieren **Read:** 1. Benutzer lässt Userdaten anzeigen **Update:** 1. Benutzer lässt Userdaten anzeigen 2. Benutzer verändert Attribute 3. Benutzer speichert Änderungen **Delete:** 1. Benutzer wird gelöscht
Extensions	-

Group CRUD

ID	03
Name	Group CRUD
Beschreibung	Gruppenverwaltung der Applikation
Preconditions	• Applikation gestartet • Benutzer eingeloggt
Postconditions	• Falls ein Gerät gefunden wird, wird es angezeigt
Main Success Scenario	**Create:** 1. Benutzer wählt ”Add New Group”. 2. Benutzer erfasst Name für Gruppe. **Read:** 1. Benutzer navigiert zur Gruppennavigation. 2. Benutzer wählt eine Gruppe zur Anzeige aus 2. Gruppendetails werden angezeigt **Update:** 1. Benutzer navigiert zur Gruppennavigation. 2. Benutzer passt Gruppeninformationen an. **Delete:** 1. Benutzer navigiert zur Gruppennavigation. 2. Benutzer wählt ”Delete Group”. 3. Gruppe wird vom System gelöscht.
Extensions	-

Device CRUD

ID	04
Name	Device erfassen
Beschreibung	Der Benutzer möchte ein Device hinzufügen.
Preconditions	• Applikation gestartet • Benutzer eingeloggt • Device ist registriert (UC06)
Postconditions	• Device in Datenbank gespeichert (Attribut added = true)
Main Success Scenario	**Create:** 1. UC 06: Discover Device **Read:** 1. Benutzer navigiert zur Geräteübersicht 2. Device details werden angezeigt 3. Benutzer wählt Objekt und klickt ”Read”. **Update:** 1. Benutzer navigiert zur Geräteübersicht 2. Benutzer passt Geräteinformationen an. **Delete:** 1. Benutzer navigiert zur Geräteübersicht 2. Benutzer wählt „delete“. 3. Gerät wird vom System gelöscht.
Extensions	-

Manage Components Hierarchy

ID	05
Name	Manage Components Hierarchy
Beschreibung	Gruppenhierarchie verwalten
Preconditions	• Applikation gestartet • Benutzer ist eingeloggt
Postconditions	• Änderungen gespeichert
Main Success Scenario	1. UC 03: Group CRUD (Create) 2. Benutzer wählt ”Group Members” 3. Benutzer fügt Kinds-Komponenten hinzu 4. Benutzer speichert Änderungen
Extensions	2.a Benutzer wählt ”Group Memberships” 3.a Benutzer verändert Eltern-Gruppe 2.b Benutzer wählt ”Add New Child Group” 3.b Benutzer gibt neuer Gruppenname ein

Discover Device

ID	06
Name	Discover Device
Beschreibung	Der Benutzer möchte ein- oder mehrere Devices finden. Endpunkt des Devices ist dem Benutzer unbekannt. Gefundene Devices sollen dem Benutzer aufgelistet werden.
Preconditions	• Applikation gestartet • Benutzer eingeloggt
Postconditions	• Gefundene Devices werden dem Benutzer angezeigt
Main Success Scenario	1. Benutzer öffnet Device Discovery 2. System listet eingegangene Anfragen von Devices auf
Extensions	-

Configuration CRUD

ID	07
Name	Configuration CRUD
Beschreibung	Konfigurationsverwaltung der Applikation
Preconditions	• Applikation gestartet • Benutzer eingeloggt
Postconditions	-
Main Success Scenario	1. Benutzer wählt ”create new configuration” 2. Benutzer stellt Konfiguration zusammen 3. Benutzer speichert Konfiguration
Extensions	4. Fehlermeldung wird angezeigt

Perform Operation

ID	08
Name	Perform Operation
Beschreibung	Operationen (Read, Write, Execute) auf Device(s) ausführen
Preconditions	• Applikation gestartet • Benutzer ist eingeloggt • Device(s) erfasst
Postconditions	-
Main Success Scenario	1. Benutzer selektiert betreffendes Device oder Gruppe 2. Operation (Read, Write, Execute) wird gewählt 3. Operation wird an Device/Gruppe gesendet 4. Resultat wird angezeigt.
Extensions	-

Authenticate Device

ID	09
Name	Authenticate Device
Beschreibung	Benutzer verwaltet Authentifizierung der Devices (nicht implementiert)
Preconditions	• Applikation gestartet • Benutzer eingeloggt
Postconditions	-
Main Success Scenario	1. Benutzer definiert Root Zertifikat(e) für Devices 2. Device baut Verbindung zum Server auf 3. Device authentisiert sich mittels Zertifikat 4. System prüft ausstellendes Zertifikat 5. System gewährt Zugriff
Extensions	5.a System verhindert Zugriff

5.3 Nichtfunktionale Anforderungen

In diesem Kapitel werden die nicht funktionalen Anforderungen des Projekts gemäss ISO/IEC 9126 Norm behandelt. Es werden für das Projekt wichtige Merkmale wie Interoperabilität, Effizienz, Benutzbarkeit und Sicherheit werden aufgeführt.

5.3.1 Interoperabilität

Bei Interoperabilitäten muss auf drei unterschiedliche Bereiche geachtet werden. Zum einen müssen IoT-Devices mit dem Server kommunizieren können, zum anderen die Benutzer selbst. Zusätzlich muss die Applikation auf unterschiedlichen Betriebssystemen installiert werden können.

- IoT-Devices mit installiertem LwM2M-Client (Version 1.0) müssen unterstützt werden
- Clients mit Webbrowsern (Google Chrome >Version 50, Mozilla Firefox >Version 45) müssen unterstützt werden
- Die Applikation soll auf aktuellen Windows-Betriebssystemen (> Windows 7) und aktuellen Linux Betriebssystemen lauffähig sein.

5.3.2 Effizienz

Durch die vielseitigen Aufgaben muss auf die Parallelität geachtet werden. Es werden durchaus zeitintensive Tasks ausgeführt, welche potenziell die Applikation über längere Zeit blockieren könnten.

- Mindestens fünf Benutzer sollen gleichzeitig angemeldet sein können
- Die Antwortzeiten vom Server an Clients dürfen unterschiedlich sein
- Ist keine Kommunikation mit dem IoT-Device nötig, so muss der Server dem Client innerhalb einer Sekunde (exkl. Round-Trip-Time) antworten
- Muss ein Server vor der Antwort mit einem IoT-Device kommunizieren, so kann keine Antwortzeit garantiert werden. Spätestens nach 1 Minute muss dem Benutzer ein Fehler angezeigt werden
- Mindestens 95% aller Anfragen an den Server müssen korrekt (erwartetes Ergebnis) beantwortet werden
- Die gestellten Anforderungen müssen bei einer Last von 1000 Devices, 200 Gruppen und 5 gleichzeitigen Benutzern erfüllt werden

5.3.3 Benutzbarkeit

Die Benutzbarkeit soll aufgrund benötigter Zeit für Use-Cases gemessen werden [30]. Der Benutzer spielt Use Cases mit Hilfe einer Anleitung durch, dabei wird die benötigte Zeit bis zur erfolgreichen Durchführung gemessen. Es wird zudem angenommen, dass die Testperson die in Kapitel 5.1.3 gestellten Anforderungen erfüllt.

Use Case	**Zeit**
UC 01: Login User	<10s
UC 02: User CRUD	<20s
UC 03: Group CRUD	<20s
UC 04: Device CRUD	<180s
UC 05: Manage Components Hierarchy	<60s
UC 06: Discover Device	<60s
UC 07: Configuration CRUD	<120s
UC 08: Deploy Configuration	<30s
UC 09: Perform Operation	<120s
UC 10: Authenticate Device	<180s

5.3.4 Wartbarkeit

Sämtliche Teile der Software sollen möglichst modular und lose gekoppelt aufgebaut werden. Eine Management-Applikation für IoT könnte potenziell sehr umfangreich sein und eine Weiterentwicklung muss in Betracht gezogen werden. Auch muss der gesamte Codeumfang verständlich sein oder mit allfälligen Kommentaren/Dokumentationen beschrieben sein.

- Eigenständiger Client- und Serverkomponente
- Testbarkeit von Code (lose Kopplung, Inversion of Control, etc.)

5.3.5 Sicherheit

Da die Applikation über das Internet erreichbar ist, muss die Applikationssicherheit gewährleistet sein. Eine Kompromittierung der Management Plattform könnte die gesamte IoT-Landschaft eines Unternehmens beeinträchtigen.

- Benutzer kommunizieren verschlüsselt mit dem Server
- Nur authentifizierte Benutzer können die Applikation benutzen
- Das Session-Timeout beträgt maximal 15 Minuten
- IoT-Devices kommunizieren verschlüsselt mit dem Server
- Nur authentifizierte IoT-Devices können mit dem Management Server kommunizieren

6. Technologie Evaluation

In diesem Kapitel werden relevante Technologien und Frameworks, welche das Front-End, Back-End, sowie die Datenbank betreffen, evaluiert. Für jeden dieser Bereiche wurden Kriterien erarbeitet. Die Auswahl geschieht aufgrund der Bewertung anhand dieser Kriterien.

6.1 Back-End

Die Entscheidung des Frameworks für das Back-End ist von grosser Tragweite. Heutzutage existieren einige beliebte Frameworks um Web Applikationen zu erstellen. Die zu verwendende Programmiersprache ist bei vielen Frameworks ebenfalls unterschiedlich und sollte berücksichtigt werden.

6.1.1 Kriterien

- Open-Source
- Gute Dokumentation und Unterstützung durch die Community
- Einfache Bedienung und schnelle Einarbeitung
- Performance / Skalierbarkeit
- Libraries für IoT-Kommunikationsprotokolle
- Kenntnisse der Programmiersprache

Node.js 	Node.js ist eine Plattform für serverseitiges JavaScript. Das I/O-Handling ist event-driven und nicht-blockierend. Jeder Node.js Prozess ist single-threaded. Node.js hat viele Erweiterungsmodule wie z.B. Express um Web Applikationen zu erstellen. Für Web Applikationen ist Node.js mit Express besonders gut geeignet, da auf Client- und Serverseite mit JavaScript gearbeitet werden kann. Die Anzahl möglicher Verbindungen könnten trotz Single-Threading gut verarbeitet werden da die Anfragen asynchron mit einem Callback an die Devices gesendet werden, der aktive Thread würde somit nicht blockieren. **Vorteile** • Einfacher Einstieg • Weite Verbreitung und gute Community-Unterstützung • Gute Skalierbarkeit (non-blocking) • Grundkenntnisse beim Entwicklerteam vorhanden • Gleiche Programmiersprache beim Front- und Backen **Nachteile** • Kein Multi-Threading • Interpretierte Skriptsprache • Schwache Objektorientierung (kein Polymorphismus, Duck-Typing, etc.)

Spring MVC 	Spring MVC wird verwendet um dynamische Webseiten mit Java Servlets zu erstellen. Dependency Injection und aspektorientierte Programmierung sind ebenfalls unterstützt. Konfigurationen über XML und Java Annotations unterstützen den Entwickler beim Erstellen von MVC Web-Applikationen. Die Library-Unterstützung für IoT-spezifische Kommunikationsprotokolle wie beispielsweise CoAP sind bei Java EE hervorragend. Java unterstützt sowohl Multi-Threading, als auch non-blocking I/O. Gleichzeitige Verbindungen zu Devices könnten in eigene Threads ausgelagert werden, um nicht das gesamte Back-End zu blockieren. **Vorteile** • Weite Verbreitung und gute Community-Unterstützung • Gute Skalierbarkeit (Multi-Threading) • Gute Java Programmierkenntnisse beim Entwicklerteam vorhanden • Kompilierte Sprache • Starke Objektorientierung (Polymorphismus, Type-Safety, etc.) • Dependency-Injection Unterstützung **Nachteile** • Umgang mit Spring MVC müsste vom Entwicklerteam erlernt werden • Unterschiedliche Sprachen für Front- und Back-End

6.1.2 Skalierbarkeit

Da potenziell viele Devices über die Applikation verwaltet werden könnten ist die Skalierbarkeit von grosser Bedeutung. Die Art der Verbindungen unterscheiden sich. Während die Anzahl gleichzeitiger Verbindungen über das Front-End zum Server (concurrent users) sehr gering sein wird, besteht die Möglichkeit, dass hunderte oder gar tausende IoT-Devices verbunden sind.

Es besteht jedoch nicht die Notwendigkeit, dass während der Nutzungsdauer alle diese Verbindungen aufgebaut und offen gehalten werden. Es ist damit zu rechnen, dass niemals mehr als 100 gleichzeitige Verbindungen zu unterschiedlichen Devices existieren.

6.1.3 Entscheidung

Die Entscheidung fiel auf Spring MVC. Eine Implementation mittels Node.js und Express wäre ebenfalls denkbar, da alle Anforderungen und Restriktionen erfüllt werden könnten. Auch andere Frameworks wie Django oder Ruby on Rails würden ebenfalls in Frage kommen, wurden aber nicht berücksichtigt da keine oder nur kaum Vorkenntnisse beim Entwicklerteam vorhanden waren.

Ausschlaggebend für die Auswahl von Spring MVC ist die zugrundeliegende Programmiersprache Java. Die Unterstützung von Klassen, Vererbung und Polymorphie sowie die grosse Unterstützung von Tools und Plugins ermöglichen eine angenehmere und übersichtlichere Entwicklung. Durch die Unterstützung mittels XML Konfigurationen und Java Annotations werden die Entwickler mit Spring MVC hervorragend unterstützt.

6.2 Front-End

Das Front-End der Applikation muss dem Benutzer auf eine einfache und übersichtliche Weise die Interaktion ermöglichen. Es sollen einerseits neue Geräte hinzugefügt werden, andererseits bestehende Geräte verwaltet oder überwacht werden. Potenziell könnten eine grosse Anzahl solcher IoT-Devices vorliegen, weshalb diese übersichtlich gruppiert werden sollen.

6.2.1 Kriterien

- Webbasierter Zugriff
- Responsive Web Design für unterschiedliche Bildschirmgrössen
- Einfache Bedienung und schnelle Einarbeitung
- Gute Dokumentation und Unterstützung durch die Community

6.2.2 Varianten

Bei der Variantenauswahl wurden drei sich stark unterscheidende Ansätze gewählt. Grundsätzlich wären weitere Frameworkalternativen zu Bootstrap und Angular.js denkbar.

Hardcoding	Hardcoding des Frontends bringt wenige Vorteile. Sämtlicher HTML, CSS und JavaScript Code müsste von Hand geschrieben werden. Ein selber erstelltes responsives Layout mit CSS ist zwar möglich, aber aufwändig. Es müsste somit viel Zeit in das Layout und das Styling für ein ansprechendes Ergebnis investiert werden. Eine Cross-Browser Unterstützung ist ebenfalls schwierig.
Twitter Bootstrap	Bootstrap von Twitter ist ein weit verbreitetes Framework für Webseiten. Responsive Layouts für unterschiedliche Bildschirmgrössen werden unterstützt. Elemente wie Buttons, Drop-Down Menüs und ähnliches werden von Bootstrap auf sehr einfache Weise bereitgestellt. Die JavaScript Funktionalitäten beschränken sich auf die Anzeige von UI Elementen. Falls JavaScript für die Programmierung der Applikation verwendet wird sollte ein weiteres Framework dafür verwendet werden.
Angular.js	Angular.js ist ein bekanntes Framework für Single Page Applications (SPA) im MVC Stil. Angular.js ist TypeScript 2 basiert, hat Dependency Injection Mechanismen eingebaut, unterstützt 2-way Databinding und Unit Testing. Die Softwarearchitektur ist Client-konzentriert, MVC passiert im Browser.

6.2.3 Entscheidung

Durch die einfache und schnelle Entwicklungsmöglichkeiten wurde Bootstrap gewählt. Angular.js hat klare Vorteile bei der Erstellung von Single Page Applications. Nach jetzigen Erkenntnissen sind jedoch keine aufwändigen Funktionen clientseitig vorgesehen, Data-Binding scheint zum heutigen Zeitpunkt nicht notwendig.

Dem Projektteam ist Angular.js unbekannt und es müsste mit einer langen Einarbeitungszeit und anfänglich schlechter Code-Qualität gerechnet werden. Falls zu einem späteren Zeitpunkt weitere Anforderungen und Funktionen vorgesehen werden, wäre eine „echte“Single Page Application mit Angular.js eine echte Alternative und könnte die „einfache“Variante mit Bootstrap von Twitter ergänzen oder ablösen.

6.3 Datenbanktechnologie

Für die Persistierung der Objekte stehen zwei grundlegende Verfahren zur Verfügung. SQL (Structured Query Language) war über einen langen Zeitraum die Standardtechnologie für die Persistierung von Datenobjekten, mit NoSQL Technologien ist jedoch eine starke Alternative mit einigen Vorteilen gegenüber der herkömmlichen SQL Technologie entstanden.

6.3.1 SQL

SQL basiert auf einem relationalen Datenmodell. Die Objekte werden auf Tabellen abgebildet. Eine Tabelle stellt somit ein Typ- und die Columns Felder dar. Mit SQL-Datenbanken wird ein stark normalisiertes Datenmodell ohne Redundanzen angestrebt. Mittels referenzieller Integrität (Fremdschlüsselwerte in Tabelle X müssen als Primärschlüsselwerte in Tabelle Y vorhanden sein) wird eine starke Datenkonsistenz gewährleistet. Ein Datensatz kann daher nur gelöscht werden, wenn keine Referenzen darauf zeigen. Mittels Joins von Tabellen können zusammenhängende Datensätze mit einem einzelnen Query abgefragt werden.

Ein grosser Nachteil bei SQL ist das strikte Datenschema. Die Tabelle gibt genau vor, welche Columns für das Objekt existieren (müssen). Möchte man weitere Werte respektive Columns hinzufügen, so müsste das gesamte Schema der Tabelle angepasst werden (altering).

6.3.2 NoSQL

Mit NoSQL werden die Daten als Key-Value Paare in einem strukturierten Format gespeichert. Häufig ist dies JSON (JavaScript Object Notation). Eine Collection steht für einen Typ von Daten und enthält entweder Key-Value Paare oder weitere Collections. Beziehungen zu anderen Objekten können auf zwei verschiedene Arten erfolgen; entweder als embedded Collection (Denormalisierung) oder mittels Referenz auf ein anderes Objekt.

Ein grosser Vorteil ist das flexible Datenmodell. Wenn man sich beispielsweise ein persistiertes Device-Objekt anschaut, so könnte dies wie folgt aussehen:

```
{
    "_id":{
        "id": "58ecfe09beedc31188fc0281",
        "name":  "Device XY",
        "protocolType": "HTTP",
        "authType": "BASIC_AUTH",
        "endpoint": "http://some.domain.com/path",
        "username": "admin",
        "password": "1234"
        }
}
```

Falls ein weiteres Attribut gespeichert werden müsste, ist dies bei NoSQL nicht weiter problematisch. Die Datenbank speichert die Objekte auch mit anderen Attributen resp. Key-Value Paaren.

NoSQL selbst unterteilt sich in verschiedene Datenmodelle. [31]

Key-Value	Leistung: hoch Skalierbarkeit: hoch Flexibilität: hoch Komplexität: keine Funktionalität: keine Durch die Einfachheit des zugrundeliegenden Datenmodells lassen sich Datensätze schnell (in O(1)) abfragen. Komplexere Abfragen über Verknüpfung der Objekte sind deutlich langsamer. Bei einer hohen Frequenz von Abfragen und Speicherung von Key-Value Tupeln ist diese Form von Datenbank geeignet.
Spaltenorientiert	Leistung: hoch Skalierbarkeit: hoch Flexibilität: mittel Komplexität: gering Funktionalität: minimal Das zugrunde liegende Datenmodell basiert auf der Speicherung der Spaltenwerte. Während normalerweise Datensätze auf einer Zeile mit ihren unterschiedlichen Spalten abgelegt werden, so wird bei dieser Art zuerst eine komplette Spalte gespeichert bevor mit Werten aus einer anderen Spalte fortgefahren wird.
Dokumentenorientiert	Leistung: hoch Skalierbarkeit: hoch Flexibilität: hoch Komplexität: gering Funktionalität: gering Dokumente bilden das Schema einer Entität. Eine dokumentenbasierte Datenbank kann mehrere solcher Dokumente enthalten. Dokumente sind quasi das Gegenstück zu Tabellen in einer relationalen Datenbank. Ein Dokument liegt meist im JSON Format vor und hält seine Objekte eingebettet.
Graphbasiert	Leistung: unterschiedlich Skalierbarkeit: unterschiedlich Flexibilität: hoch Komplexität: hoch Funktionalität: gem. Graphentheorie Eine Graphendatenbank speichert Knoten und Kanten. Knoten sind Entitäten und Kanten deren Beziehungen. Dieses Datenmodell eignet sich für die Modellierung und Speicherung von Netzwerken wie beispielsweise Social Media.

6.3.3 Entscheidung

NoSQL wurde als geeignetere Variante für die Speicherung der Objekte ausgewählt. Bei vielen unterschiedlichen Devicetypen muss davon ausgegangen werden, dass Änderungen respektive Ungleichheiten am Schema auftauchen werden. Ein starres, relationales Modell würde dies stark erschweren, da jedes mal mittels „ALTER TABLE“das Schema geändert- und bestehende Datensätze angepasst werden müssten.

Als NoSQL Implementation wurde „mongoDB“ausgewählt. MongoDB ist eine dokumentenorientierte Datenbank, welche Objekte im BSON Format abspeichert. Die Entitäten werden in sogenannten „Collections“abgelegt. Attribute der Collections können einzeln abgefragt werden. Die Struktur der Collections ist flexibel, Attribute von Objekten innerhalb der Collections dürfen sich unterscheiden. MongoDB ist die zurzeit wohl am weitest verbreitete NoSQL Implementation, weshalb eine grosse Community und umfangreiche Online-Hilfen verfügbar sind.

7. Architektur

In diesem Kapitel wird die Software Architektur erarbeitet. Es soll sowohl die Problemdomäne abstrakt mittels Domänenmodell analysiert werden, als auch ein Klassendesign mittels Schichtendiagramm erarbeitet werden.

7.1 Systemübersicht

In der folgenden Abbildung ist das System auf hoher Abstraktionsstufe zu sehen. Die Applikation ist über einen Web Browser bedienbar. Auf dem Management Server werden verschiedenartige IoT-Devices verwaltet. Der Management Server kommuniziert über TCP/IP mit den Devices.

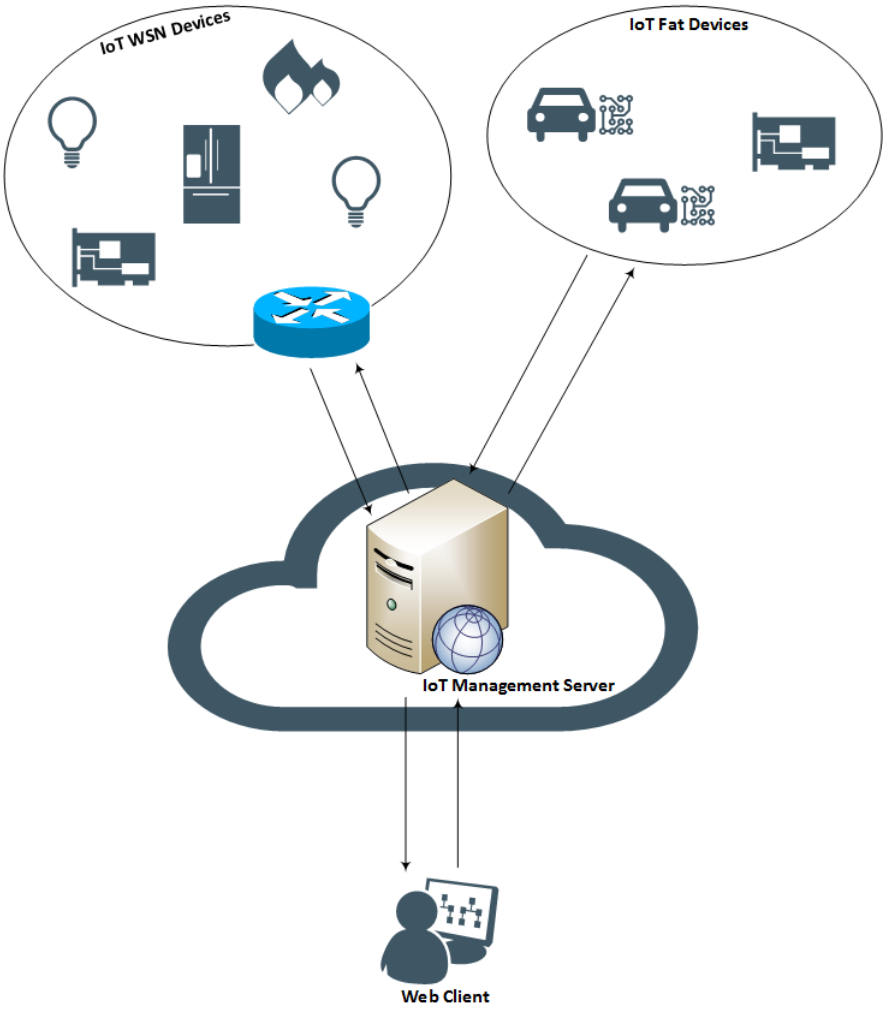


Abbildung 7.1.: Systemübersicht

7.2 Klassenstruktur

7.2.1 Klassendiagramm

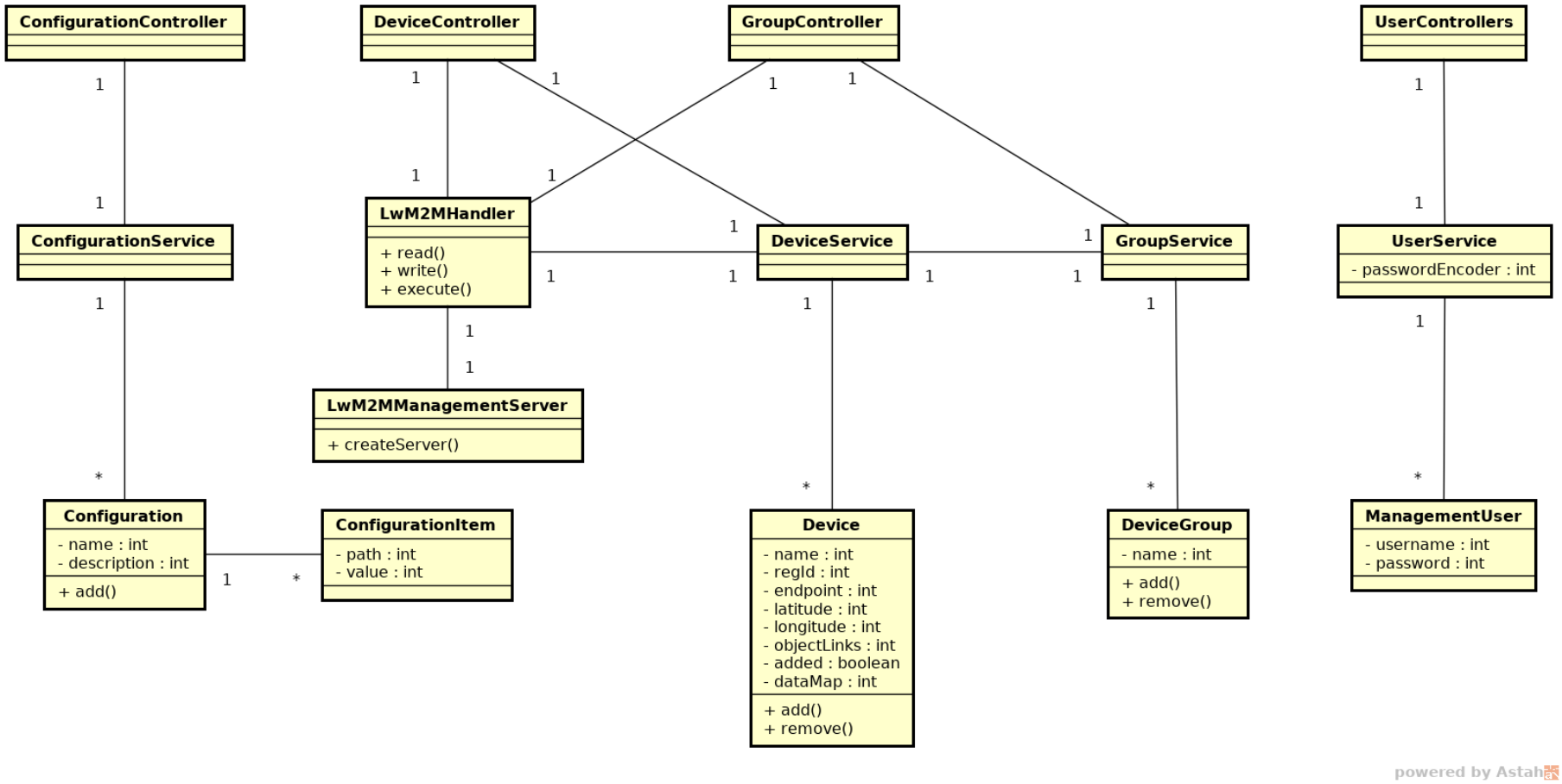


Abbildung 7.2.: Klassendiagramm

7.3 Logische Architektur

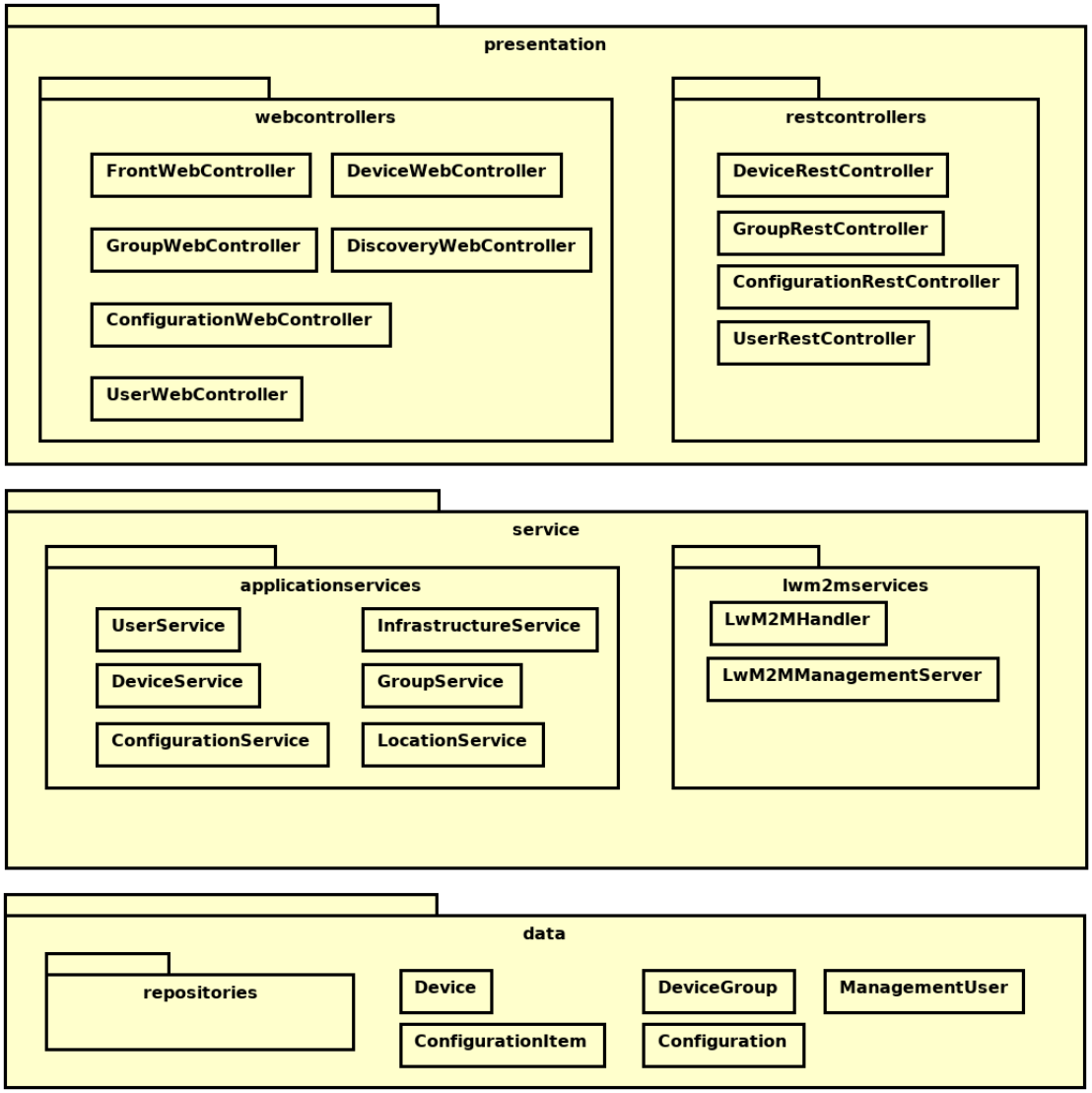


Abbildung 7.3.: Logische Architektur

7.3.1 Presentation Layer

Im Presentation-Layer gibt es Web-Controller und Rest-Controller. Die Web-Controller liefern die HTML-Inhalte aus. Die Rest-Contoller liefern Daten im JSON-Format oder es werden Methoden aus dem Service Layer gestartet, um die Daten zu verarbeiten.

Packagename	Beschreibung
webcontrollers	Alle HTML-Inhalte werden durch die Web-Controller ausgeliefert. Dazu werden die Daten aus dem Service-Layer geholt.
restcontrollers	Die Rest-Controller führen Methoden auf dem Service-Layer aus und geben die Resultate als JSON zurück an den Client.

Packagestruktur

Klassenname	Beschreibung
FrontWebController	Beinhaltet die Login, Logout und Home Routen.
GroupWebController	Liefert alle für Groups relevanten HTML-Inhalte aus.
DeviceWebController	Liefert alle für Devices HTML-Inhalte aus.
DiscoveryWebController	Beinhaltet die Discovery Routen.
ConfigurationWebController	Beinhaltet die für die Configuration wichtigen Routen.
UserWebController	Liefert die User HTML-Inhalte aus.
DeviceRestController	Führt Service-Layer Methoden aus und liefert JSON Daten.
GroupRestController	Führt Service-Layer Methoden aus und liefert JSON Daten.
ConfigurationRestController	Führt Service-Layer Methoden aus und liefert JSON Daten.
UserRestController	Führt Service-Layer Methoden aus und liefert JSON Daten.

Klassenstruktur

Schnittstellen Der Presentation-Layer hat eine Schnittstelle zum Service-Layer. Alle Daten werden über den Service-Layer geholt und an den Service-Layer gegeben. So ist eine saubere Trennung zwischen Daten und Methoden sichergestellt.

7.3.2 Service-Layer

Der Service-Layer beinhaltet die Klassen, welche für die Verarbeitung der Daten zuständig ist. Hier werden alle Devices erfasst, verbunden und verwaltet. Dazu wird jedes gefundene Device auf der Datenbank hinterlegt. Bei einem Read, Write oder Execute wird das Device aus dem Data Layer geholt und mit den neuen Daten aktualisiert. Dabei wird hier zwischen Applikation-Services und LwM2M-Services unterschieden.

Packagename	Beschreibung
applicationservice	Dieses Package beinhaltet die Klassen, welche die Devices oder Groups erstellen, anpassen usw.
lwm2mservices	Dieses Package beinhaltet alle Kommunikationsrelevanten Klassen, um mit den Devices Daten auszutauschen.

Packagestruktur

Klassenname	Beschreibung
DeviceService	Holt und speichert die Devices von dem Data-Layer und verarbeitet die Änderungen
GroupService	Zugriff auf den Data-Layer. Liest und schreibt alle Groups.
ConfigurationService	Zugriff auf den Data-Layer. Liest und schreibt alle Configurationen.
UserService	Checkt bei den Logins und verarbeitet die User Schreib- und Lesevorgänge.
LocationService	Liest die Locations aus den Devices und gibt sie dem Controller weiter.
InfrastructureService	Infrastruktur-Methoden, wie zum Beispiel Server-Uptime.
LwM2MHandler	Beinhaltet alle Kommunikation zwischen Devices und dem Management Server.
LwM2MManagementServer	Der Server, welche die Befehle aufs Netzwerk schickt und empfängt.

Klassenstruktur

Schnittstellen Der Service-Layer hat eine Schnittstelle zum Data-Layer. Durch den Service-Layer werden die Datenobjekte erstellt, bearbeitet und ausgewertet. Der Presentation-Layer muss immer über den Service-Layer, um eine saubere Abtrennung der Schichten zu gewährleisten.

7.3.3 Data-Layer

Der Data-Layer beinhaltet alle Datenobjekte, welche vom laufenden Programm benötigt werden. Diese werden von hier in die Datenbank geschrieben. Das Repository bietet die Datenbankmethoden an, um die Daten in die Datenbank zu speichern.

Packagename	Beschreibung
repositories	Zugriffsmethoden auf die Datenbank. Eigene sowie durch die Library ausgelieferte Methoden.

Packagestruktur

Klassenname	Beschreibung
Device	Device ist eine Datenklasse, welche alle Daten von einem Device beinhaltet.
DeviceGroup	In der Datenklasse DeviceGroup, werden alle Gruppen verwaltet, damit das Composite-Pattern umgesetzt werden kann.
ManagementUser	Datenklasse für die Loginbenutzer.
Configuration	Configuration Datenklasse, welche eine List von ConfigurationItems beinhaltet.
ConfigurationItem	Datenklasse für die einzelnen ConfigurationItems.

Klassenstruktur

Schnittstellen Da der Data-Layer die unterste Schicht ist, gibt es nur eine Verbindung zur Datenbank.

7.4 Deployment

Die Applikation hat eine typische 3-Tier Architektur. Es wird ein Serverseitiges Templating verwendet. Alle Daten werden daher auf dem Server verarbeitet und als HTML an den Client gesendet. Die Daten werden in einer Datenbank persistiert.

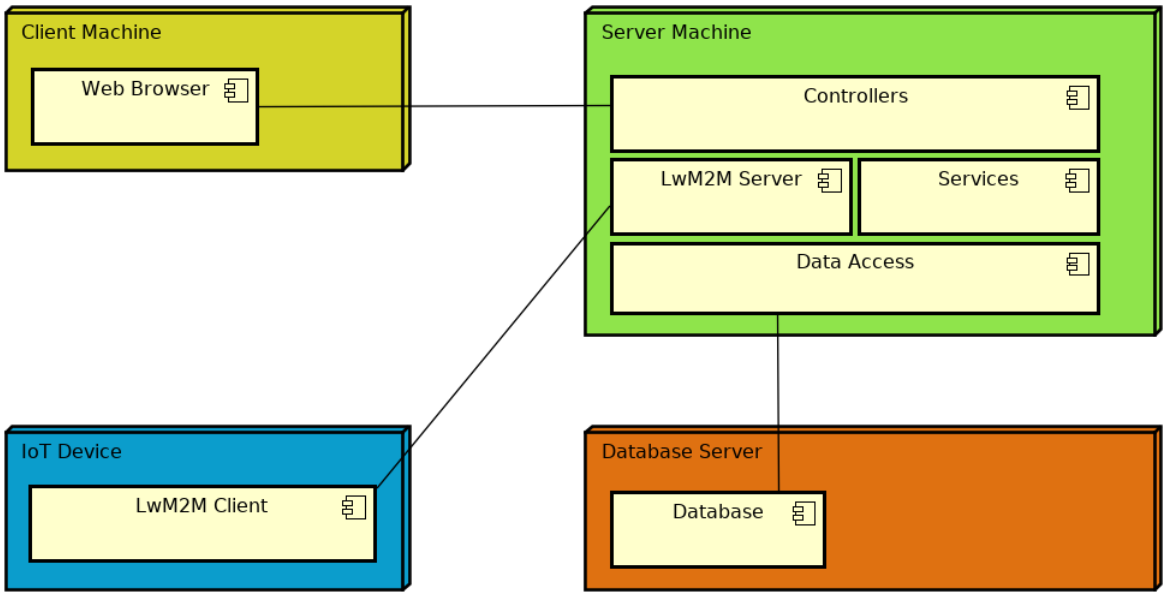


Abbildung 7.4.: Deployment

7.5 User Interface Entwurf

7.5.1 Design

Das Projekt erhält eine möglichst einfach gehaltene Benutzeroberfläche. Durch die farbliche Trennung von Funktionen und fixen Bereichen für die Daten, soll die Oberfläche einfach und intuitiv gestaltet werden. Zusätzlich wird die Oberfläche responsiv aufgebaut, damit sie an unterschiedlichen Bildschirmgrössen verwendbar wird.

Es wurden Mock-ups anhand von einfachen Handskizzen erstellt, um das aussehen und die Funktionen der Benutzeroberfläche zu planen.

7.5.2 Ansichten

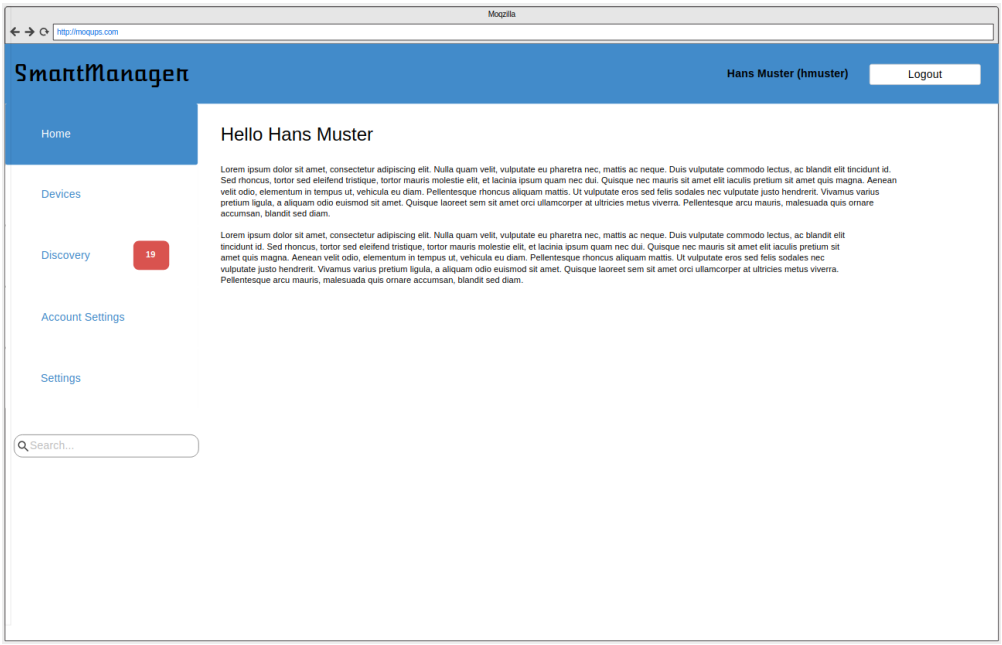


Abbildung 7.5.: Startseite

Startseite Die Startseite zeigt dem Benutzer Hilfestellungen an und führt den Benutzer durch die ersten Schritte. Auf der Linken Seite befindet sich das Menu und eine Suche. Oben Rechts findet der Benutzer den Loginbereich und die Benutzerangaben.

Beim Menu gibt es statische, sowie dynamische Einträge. Der Discoveryeintrag ist ein dynamischer Eintrag und zeigt die Anzahl gefundenen Geräte an.

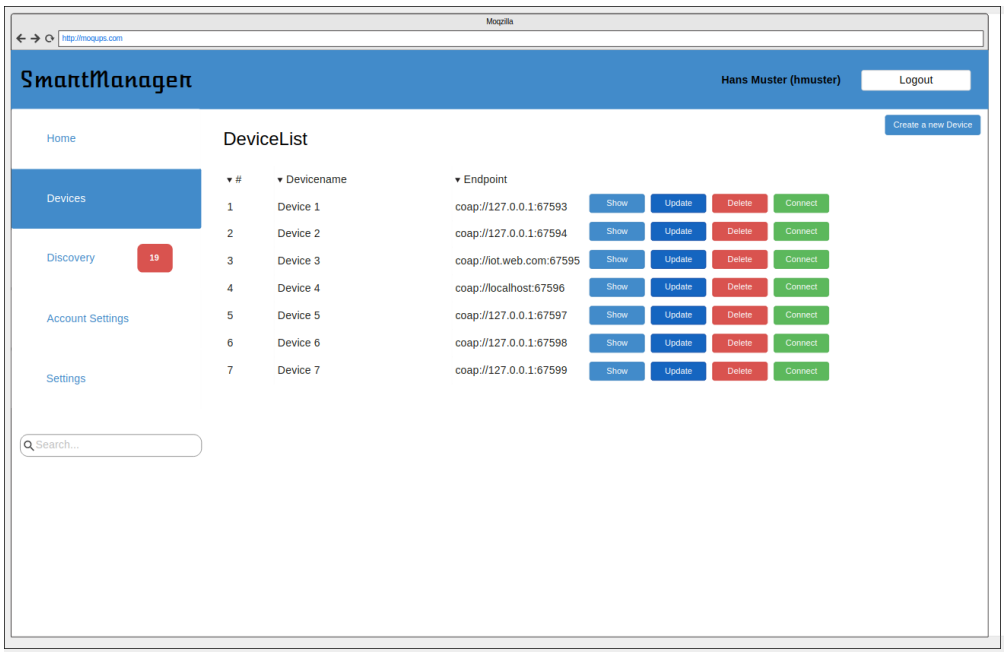


Abbildung 7.6.: Device List

Device List In der Device Liste werden alle Devices angezeigt. Dazu wird zu jedem Device die jeweiligen Möglichkeiten als Button eingeblendet. So kann man ein Device betrachten, Anpassen, Löschen und sich mit dem Gerät verbinden. Zusätzlich werden die Daten, wie zum Beispiel Endpoint angezeigt.

Oben rechts befindet sich der Button, um weitere Devices zu erstellen.

Show Zeigt die Details zu einem Device an

Update Führt zum Device Update Formular, damit das Device angepasst werden kann.

Delete Button, der das Device aus der Applikation löscht

Connect Button, um sich mit dem Gerät zu verbinden.

Create a new Device Führt zur Create Device Ansicht.

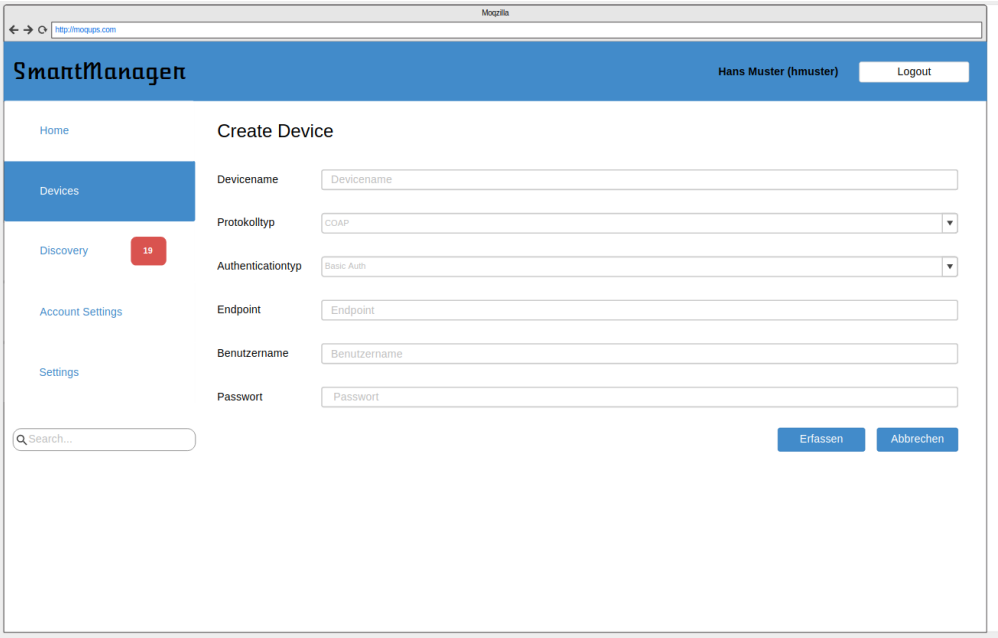


Abbildung 7.7.: Create Device

Create Device Bei der Create Device Ansicht kann ein neues Device erstellt werden. Dadurch wird das Device manuell erfasst und in der Datenbank abgelegt. Dies kann gemacht werden, wenn das Device nicht via Discovery gefunden worden ist.

Je nach Combobox Auswahl passt sich das Formular an, um zum Beispiel auch Zertifikate hinterlegen zu können.

Erfassen / Update Erstellt oder passt ein Device an

Abbrechen Bricht den Device Create ab und führt zur Device List Ansicht

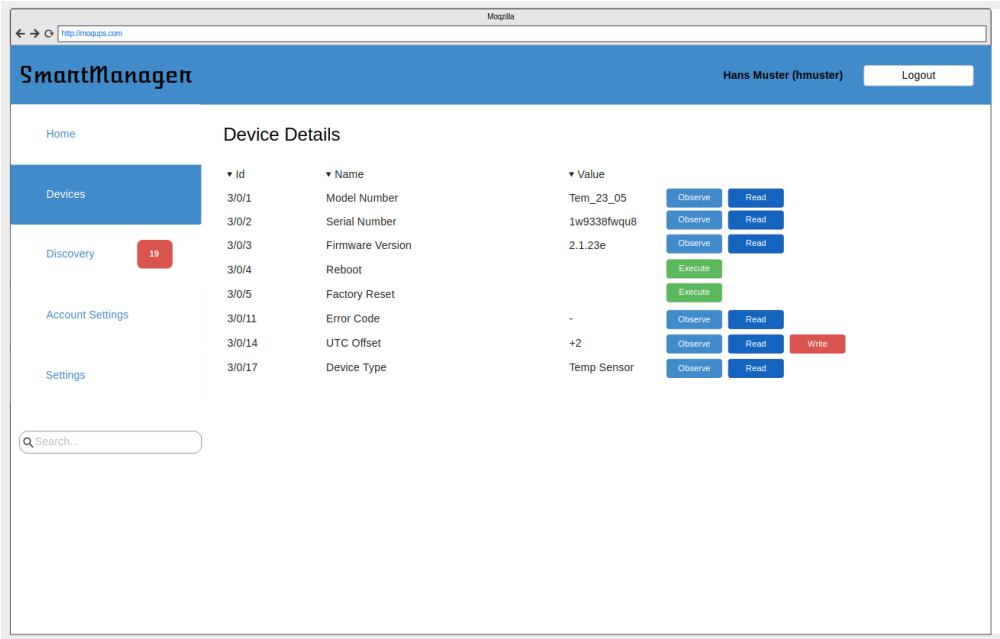


Abbildung 7.8.: Device Details

Device Details Alle wichtige Informationen eines Devices werden in dieser Übersicht angezeigt. Zu Jeder Ressource wird die Id, der Name und den aktuellen Wert angezeigt. Je nach Ressource passen sich die Möglichen Aktionen an. Dabei gibt es Observe, Read, Write und Execute. Bei Write wird ein zusätzliches Eingabefeld eingeblendet, um den gewünschten Wert eintragen zu können.

Observe Startet ein Observer für die gewünschte Ressource.

Read Liest die Ressource vom Device

Write Schreibt den Wert in die Ressource und aktualisiert den Eintrag in der Datenbank

Execute Führt einen Befehl auf dem Device aus

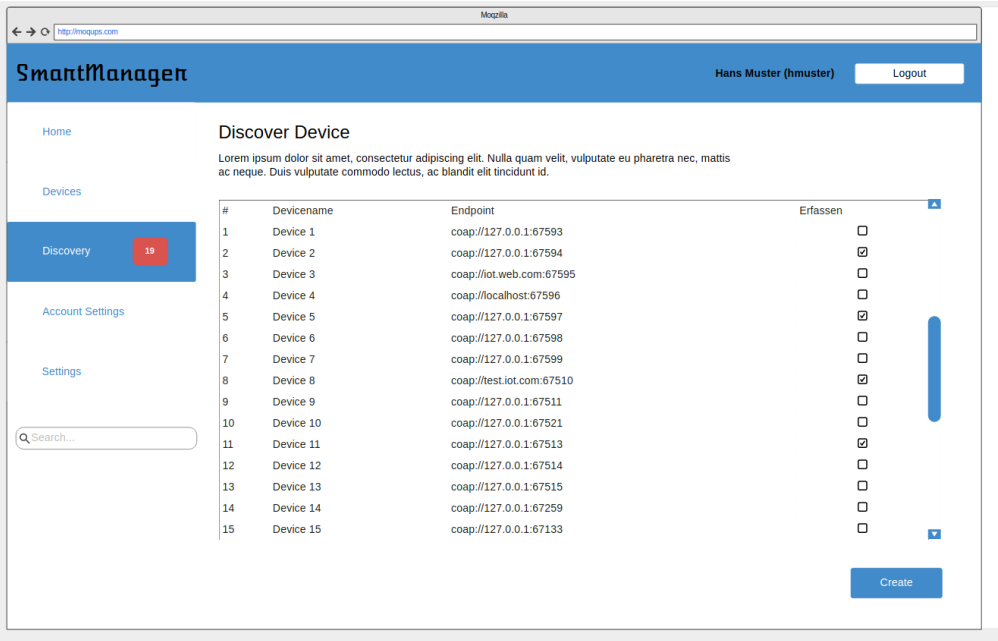


Abbildung 7.9.: Device Details

Discover Device Alle durch das Discovery gefundenen Devices werden in dieser Übersicht aufgelistet. Dazu werden direkt die wichtigen Daten angezeigt und man hat die Möglichkeit das gewünschte Device zur Applikation hinzuzufügen. Die Anzahl gefundenen Devices werden auf der Linken Seite im Menu angezeigt und passt sich dynamisch an.

Create Erstellt für die ausgewählten Devices ein Deviceobjektauf der Datenbank

8. Realisierung

In diesem Kapitel werden die Realisierungen der Komponenten beschrieben. Es wird versucht, Konzepte und Problemlösungen zu erklären. Vereinzelt werden auch Codeausschnitte gezeigt, sofern diese relevant für das Verständnis sind.

8.1 User Interface

In diesem Abschnitt wird das User Interface gezeigt. Man erhält einen guten Überblick über Funktionalitäten und Aufbau des Front-Ends.

8.1.1 Änderungen

Der erste Prototyp der Applikation wurde gemäss Mock-ups im Kapitel 7.5 erstellt. Beim Requirements- und Architekturreview am 24.04.2017 haben die Teammitglieder unter anderem Änderungen am Erscheinungsbild beschlossen. Durch den Einsatz des LwM2M Servers haben sich auch Anpassungen an der Seitenstruktur ergeben.

Anbei die UI-relevanten Anpassungen:

- Hauptnavigation von linker Seite an den oberen Bildschirmrand verschoben
- Entfernen der Accountsettings aus der Hauptnavigation
- Hinzufügen des Links ”Configurations” in die Hauptnavigation
- Verzicht von blauer Farbe am oberen Bildschirmrand, stattdessen schlichteres grau
- Begrüssungstext am Homebildschirm entfernt, stattdessen Dashboard mit wichtigen Informationen hinzugefügt
- Hinzufügen einer Navigation für Devices und Gruppen auf der ”Devices” Seite
- Das ”Create Device” Formular wurde entfernt
- Der ”Observe”-Button unter ”Devices” wurde entfernt, da diese Funktion nicht implementiert wurde
- Dem ”Discovery” wurden Drop-Down Menüs für Konfigurationen und Gruppen hinzugefügt

8.1.2 Allgemein

In diesem Unterabschnitt werden die Konzepte erklärt, mit welchen das User Interface realisiert wurde. Das Webinterface wurde HTML, CSS und JavaScript erstellt und als unterstützende Library wurde Twitter Bootstrap verwendet.

Java Server Pages JSP Sämtlicher HTML Code ist in Java Server Pages (JSP) abgelegt. JSP Seiten werden auf dem Server in Java Servlets kompiliert.

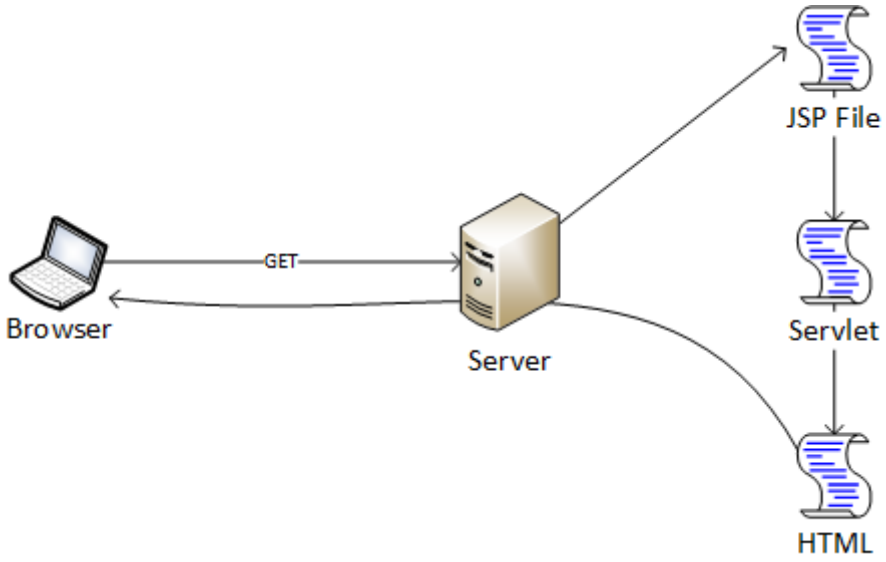


Abbildung 8.1.: Java Server Pages

Der Client erhält über eine Web-URL vom Server generiertes HTML.

Java Server Pages Standard Tag Library JSTL Mittels JSTL werden in dieser Applikation folgende Aufgaben erfüllt:

- HTML Fragmente in bestehende Container laden
- if/else Abfragen für Darstellungen
- Loop durch Collections im Modell für die Darstellung von mehreren Objekten

In JSP Seiten wird JSTL mittels folgender Code-Zeile verwendet:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

Ein Loop durch eine Collection mit Hilfe von JSTP sieht folgendermassen aus:

```
<c:forEach var="collection" items="${collection}">
</c:forEach>
```

Seitenlayout Das Seitenlayout ist grundsätzlich sehr einfach. Die Navigation erfolgt am oberen Rand der Seite. Der gesamte Seiteninhalt befindet sich im Content-Bereich.

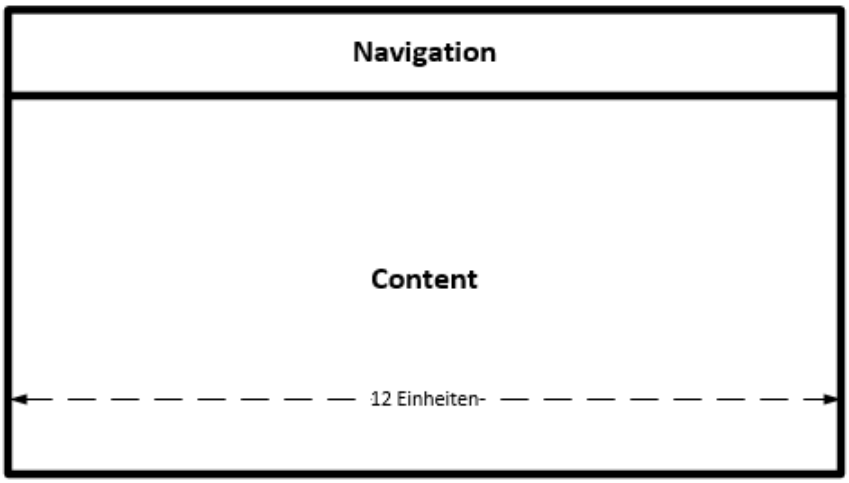


Abbildung 8.2.: Seitenlayout Allgemein

Das Grid-System von Bootstrap teilt den sichtbaren Bereich in 12 Spalten auf. Jede Spalte besetzt immer $\frac{1}{12}$ der gesamten Fenstergrösse. Bootstrap verwendet folgende CSS Media-Queries für unterschiedliche Bildschirmtypen:

- Small Devices (min-width: 576px)
- Tablets (min-width: 768px)
- Desktops (min-width: 992px)
- Large Desktops (min-width: 1200px)

Sämtlichen div-Containern wird angegeben, wieviele Spalten in welcher Bildschirmgrösse sie besetzen. Mit Bootstrap könnte dies folgendermassen aussehen:

```
<div class="col-lg-6 col-md-6 col-sm-12 col-xs-12">
</div>
```

An diesem Beispiel besetzt dieser Div-Container auf mittelgrossen und grossen Desktops die Hälfte-, auf Tablets und Smartphones die gesamte Bildschirmbreite. Somit wird ein responsives Layout umgesetzt, dass die Applikation sowohl von mobilen-, als auch von Desktop Devices gut nutzbar ist.

Navigation Bootstrap liefert eine ”default-navbar”. Sämtliche Navigationslinks (Home, Device, Discovery, Configurations, User Panel) befinden sich in der Navigationsleiste. Die Navigationsleiste besetzt in jeder Bildschirmgrösse die gesamte Breite.

```
<div class="col-lg-12 col-md-12 col-sm-12 col-xs-12">
</div>
```

Abbildung 8.3.: Navigationsleiste gross

Da diese Navigationsleiste auf Smartphones schwierig zu bedienen ist, wird auf der kleinsten Bildschirmgrösse anstatt der normalen Links in der Navigation ein Hamburger-Button verwendet. Anfangs war dieser Button stark umstritten, mittlerweile wird er häufig eingesetzt, weshalb jedem Benutzer klar sein sollte, dass sich dahinter weitere Elemente befinden.

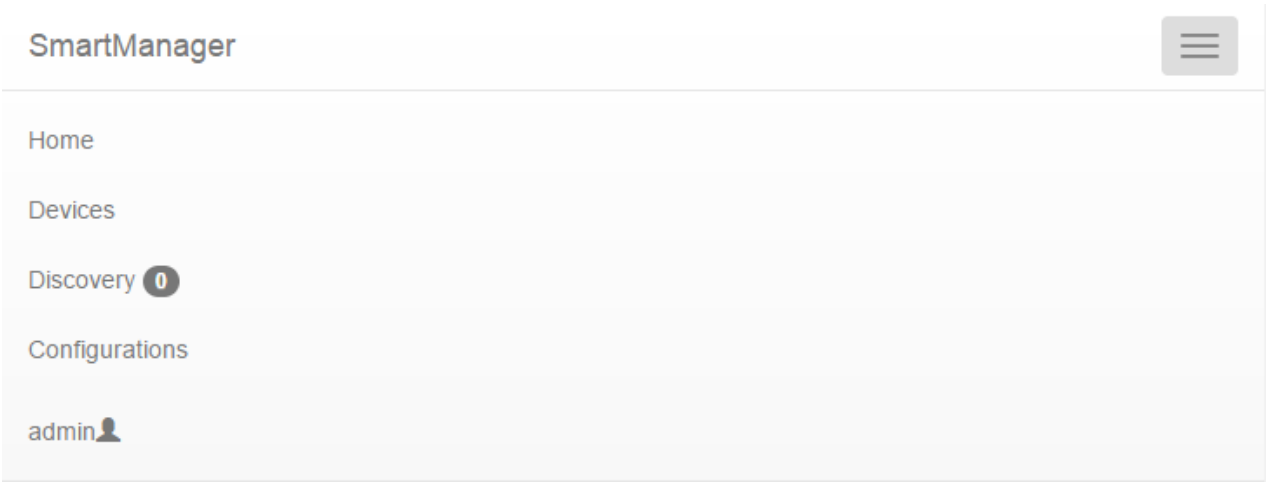


Abbildung 8.4.: Navigationsleiste xs

Die Navigationsleisten sind auf jeder Seite am oberen Bildschirmrand fixiert, sodass bei allfälligem Scrollen die Navigation trotzdem sichtbar bleibt.

Damit duplizierter Code auf jeder Seite verhindert wird, wird der gesamte HTML Code für die Navigation in ein eigenes JSP File ”menuFragment.jsp” ausgelagert. Der Inhalt des JSP Files wird mittels

```
<jsp:include page="../../views/menuFragment.jsp" />
```

serverseitig ins HTML gerendert.

Content Der gesamte Content Bereich ist auf jeder Seite unterschiedlich gestaltet. Die gesamte Breite von 12 Spalteneinheiten kann verwendet werden.

Asynchronous JavaScript and XML AJAX Damit bei der Kommunikation mit dem Server das User Interface nicht blockiert wird, werden bei sämtlichen Server-Calls AJAX verwendet. AJAX versendet Nachrichten vom Client an den Servern asynchron. Im ”success” Attribut kann eine Callback-Funktion hinterlegt werden, welche bei erfolgreicher Antwort ausgeführt wird.

In dieser Applikation werden AJAX-Calls nach folgendem Schema ausgeführt:

```
$.ajax({
  type : "GET/POST/DELETE",
  url : ctx + "/resource/method",
  success : function() {
    /*some callback code...*/
  },
  error : function(xhr, ajaxOptions, thrownError) {
    window.location.href = ctx + "/";
    alert(thrownError);
  }
});
```

Falls der Server mit einem Fehler antwortet, wird die Callback-Funktion im ”error”-Attribut ausgeführt. Der Benutzer wird dabei auf die Hauptseite umgeleitet und erhält eine Error-Meldung angezeigt.

Bootbox An verschiedenen Stellen auf der Seite werden Pop-ups benötigt. In diesem Projekt wird die JavaScript Library ”Bootbox” von Bootboxjs.com verwendet.

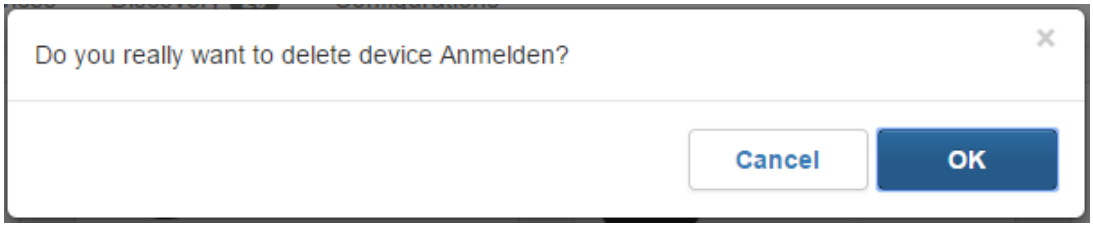


Abbildung 8.5.: Bootbox

Der JavaScript Code für die Erstellung und Verwendung der Bootbox ist leicht verständlich.

```
bootbox.confirm({
  size : "large",
  title : "some title",
  message : content,
  callback : function(ok) {
    if (ok) {
      /*save, redirect etc.*/
    }
  }
});
```

Als ”message”-Attribut kann ein String oder auch HTML Code eingefügt werden. Beim ”Callback”-Attribut kann eine Callback-Funktion hinterlegt werden. Diese wird ausgeführt, wenn das Pop-up Fenster geschlossen wird. Die if-Abfrage stellt sicher, dass der darinliegende Code nur ausgeführt wird, wenn ”OK” auf geklickt wird.

Google Maps API An verschiedenen Stellen auf der Seite werden Devices auf Google Maps Karten angezeigt. Dies gibt dem Benutzer eine schnelle Übersicht der Lokalitäten der IoT-Devices.

Über die JavaScript-Funktion ”initMap()” wird die (leere) Map geladen. Danach werden über die Funktion ”getLocations()” die Locations der benötigten IoT-Devices vom Server abgefragt. Über die ”insertLocations()”-Funktion werden die Punkte auf der Map eingezeichnet.

8.1.3 Home

Die Home-Seite dient als Einstiegspunkt und Übersichtsseite. Dem Benutzer sollen Informationen über die Applikation und die verwaltete Infrastruktur angezeigt werden.

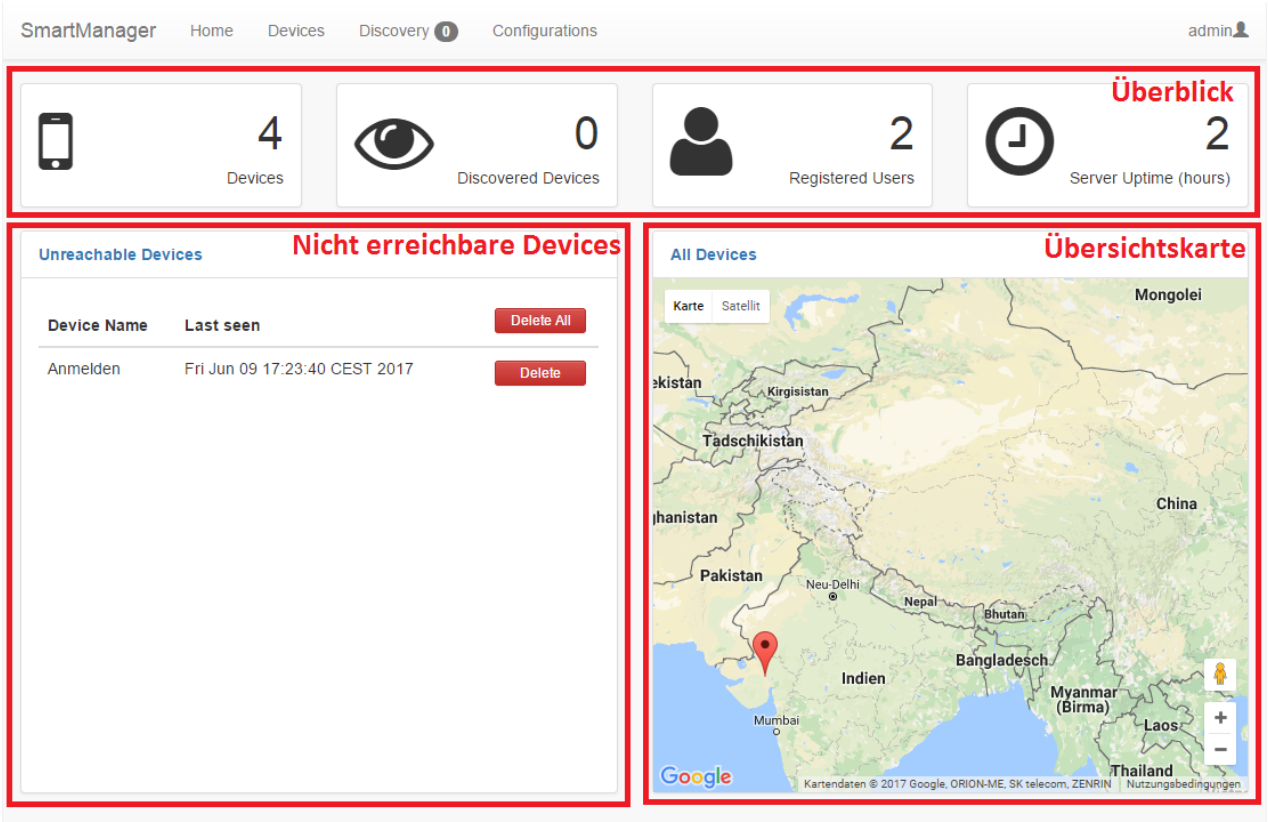


Abbildung 8.6.: Dashboard

Überblick In diesem Bereich werden dem Benutzer folgende vier Informationen auf den ersten Blick angezeigt:

- Gesamtzahl erfasster Devices im System
- Anzahl Devices im Discovery (noch nicht verwaltet)
- Anzahl registrierter Benutzer
- Server Uptime in Stunden

Nicht erreichbare Devices Sind Devices nicht mehr erreichbar, muss dies dem Benutzer mitgeteilt werden. Hat sich ein Device mehr als 30 Minuten nicht mehr beim Server gemeldet, so erscheint es in dieser Tabelle. Klickt der Benutzer auf ”Delete” respektive ”Delete All”, so werden die JavaScript Funktionen ”deleteUnreachableDevice(id, name)” und ”deleteAllUnreachableDevices()” ausgeführt. Über eine AJAX-Anfrage werden die entsprechenden Serverpfade aufgerufen.

Übersichtskarte Auf dieser Übersichtskarte werden alle im Device gespeicherten Locations von IoT-Devices angezeigt.

8.1.4 Discovery

Auf der Discovery Seite sieht der Benutzer alle registrierten Devices. Sämtliche Devices registrieren sich am LwM2M Server. Sind sie vom Benutzer noch nicht hinzugefügt worden, so erscheinen sie auf dieser Seite.

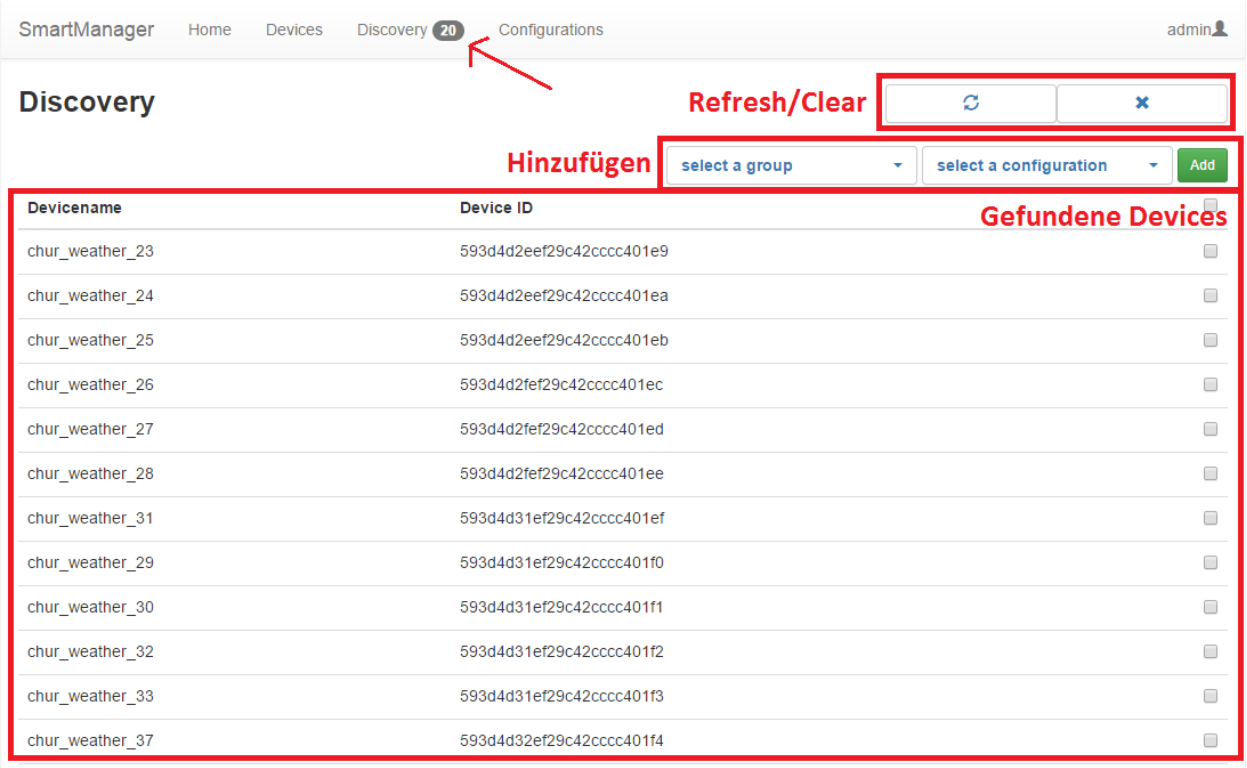


Abbildung 8.7.: Discovery

Gefundene Devices Ist ein Device vom Benutzer noch nicht hinzugefügt worden, besitzt das Attribut ”added” den Wert ”false”. Nachdem der Benutzer das Device hinzugefügt hat, wird dem Device das Attribut ”added” auf ”true” gesetzt und erscheint deshalb nicht mehr.

Hinzufügen Beim ”Add”-Button werden die selektierten Devices hinzugefügt. Es besteht die Möglichkeit, beim Hinzufügen eine Gruppe oder eine Initiale Konfiguration zu setzen.

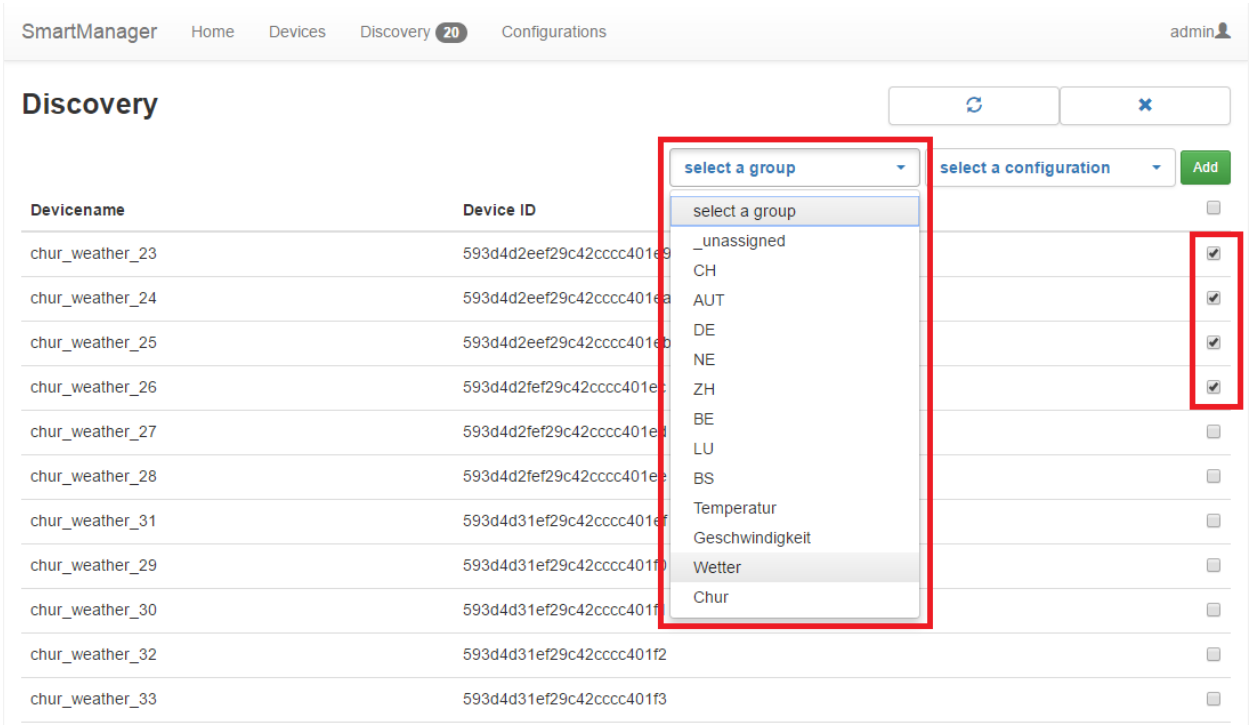


Abbildung 8.8.: Discovery Gruppenauswahl

Die selektierten Werte in den Drop-Down-Listen werden dem Server beim Hinzufügen des Devices übergeben.

Refresh/Clear Mit dem Refresh Button wird die Seite neu geladen, damit frisch registrierte Devices angezeigt werden. Der Clear Button leert die Liste von Devices. Melden sich die Devices zu einem späteren Zeitpunkt nochmals, so tauchen sie in dieser Ansicht wieder auf.

8.1.5 Configuration

Benutzer können auf dieser Seite eigene Konfigurationen erstellen. Sämtliche schreibbaren LwM2M-Objekte können gesetzt werden. Nach der Erstellung einer Konfiguration kann sie auf neue Devices oder eine Devicegruppe geschrieben werden.

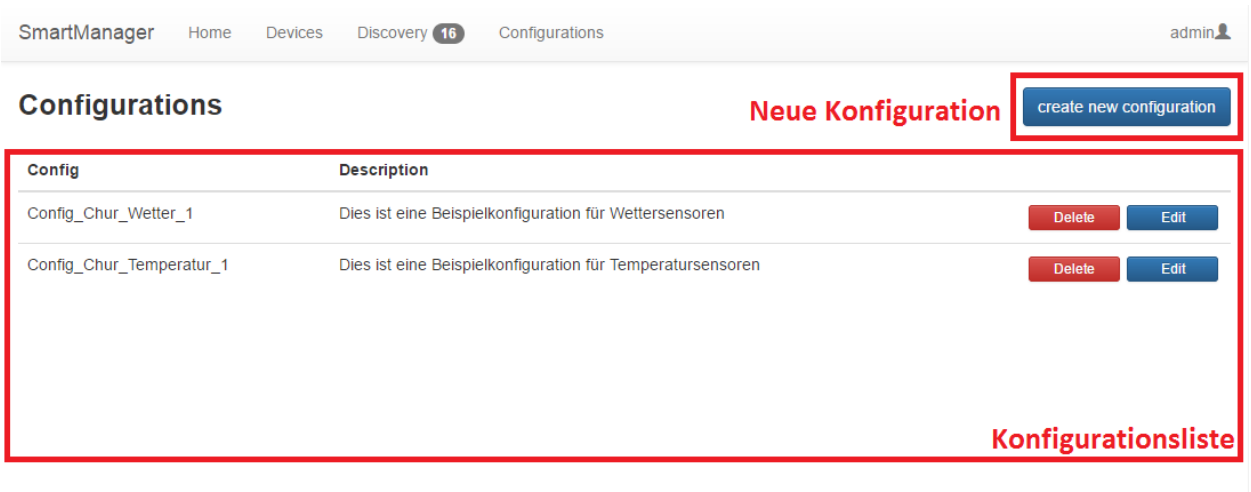


Abbildung 8.9.: Configuration Übersicht

Konfigurationslisten Sämtliche erstellten Konfigurationen werden hier aufgelistet. Eine Konfiguration kann einzeln gelöscht und editiert werden.

Neue Konfiguration Mit dem ”create new configuration” Button öffnet sich ein neues Pop-up, in welchem der Benutzer eine neue Konfiguration erstellt.

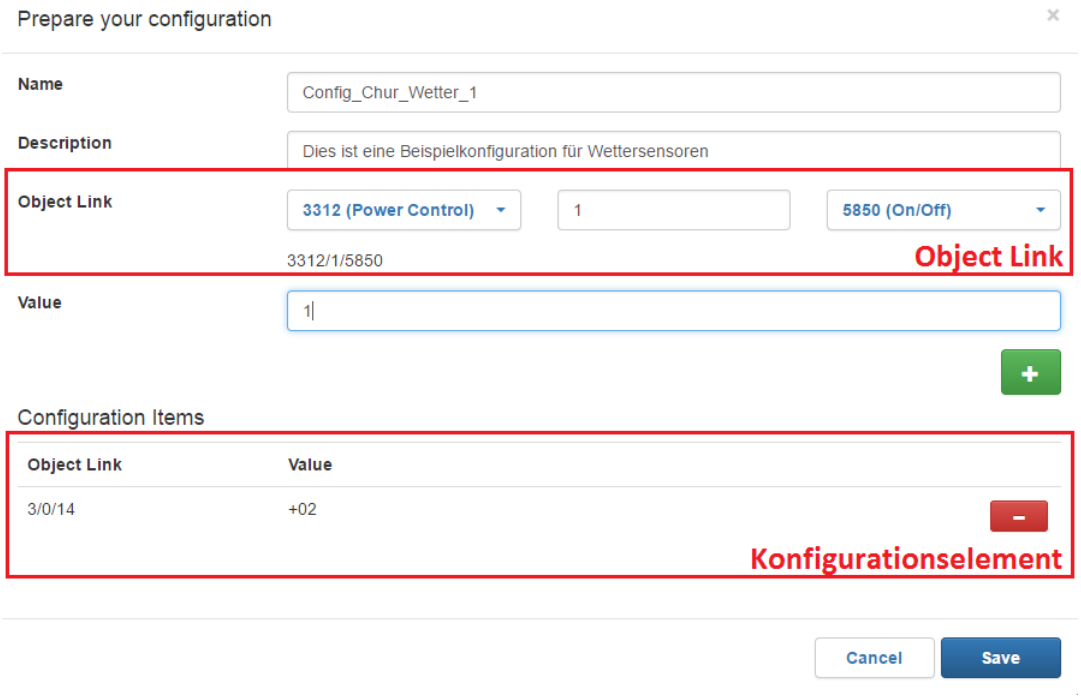


Abbildung 8.10.: Configuration Neu

Sowohl Name, als auch Description können frei gewählt werden. Beim selektieren der Object ID (linkes Drop-Down Menü) wird eine Ajax Anfrage an den Server gesendet. Als Antwort erhält man informationen über die beiden anderen Felder. Sind Multi-Instanzen nicht erlaubt, so bleibt das Zahlenfeld ausgegraut, die Daten für das rechte Drop-Down werden dynamisch gerendert, sind also abhängig von der links gewählten Object-ID. Das Value Feld kann frei beschrieben werden. Serverseitig werden Überprüfungen des Typs, sowie Character Escaping durchgeführt.

Nachdem der Benutzer den Plus-Button geklickt hat, erscheinen Object Link und Value als Configuration Item in einer Liste und es kann mit weiteren Einstellungen fortgefahren werden. Sobald der Benutzer ”Save” drückt, wird clientseitig ein JSON-formatiertes Konfigurationsobject erstellt und an den Server gesendet.

```
[
  "Config_Chur_Temperatur_1",
  "Dies ist eine Beispielkonfiguration fuer Temperatursensoren",
  {
  "Object Link": "3305/0/5806",
  "Value": "5.5382"
  },
  {
  "Object Link": "3201/0/5750",
  "Value": "I/O"
  }
]
```

8.1.6 Devices und Groups

Benutzer können auf dieser Seite einzelne Devices und Gruppen verwalten. Sie ist also sozusagen das Kernstück der Applikation, da viele Use-Cases in diesem Bereich abgedeckt werden.

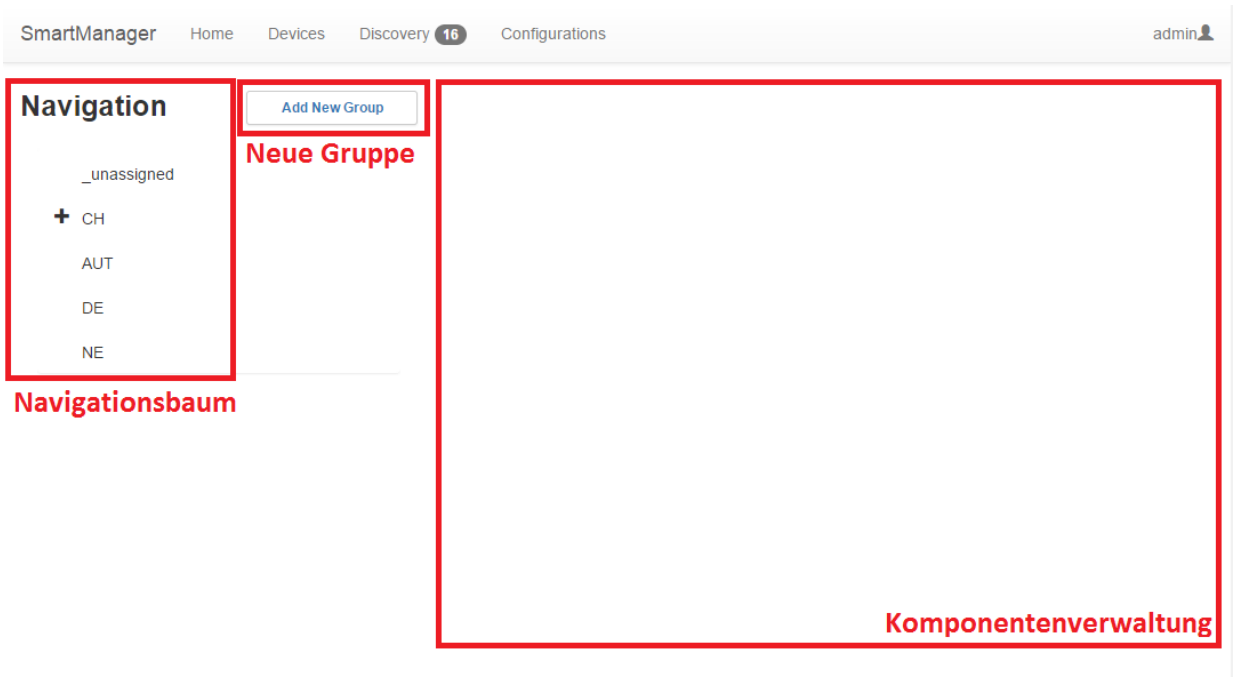


Abbildung 8.11.: Devices

Komponentenverwaltung Der Bereich der Komponentenverwaltung ist zu Beginn leer. Der Benutzer muss zuerst eine Gruppe oder ein Device selektieren, damit der entsprechende Inhalt geladen wird. Es gibt zwei verschiedene Komponentenverwaltungen, die Device- und die Gruppenverwaltung. Nachdem im Navigationsbereich link eine Gruppe/Device gewählt wurde, wird vom Server ein HTML Fragment angefordert und dargestellt.

Navigationsbaum Auf der linken Seite befindet sich der Navigationsbaum. Darin befinden sich sämtliche Gruppen und Devices hierarchisch angeordnet.

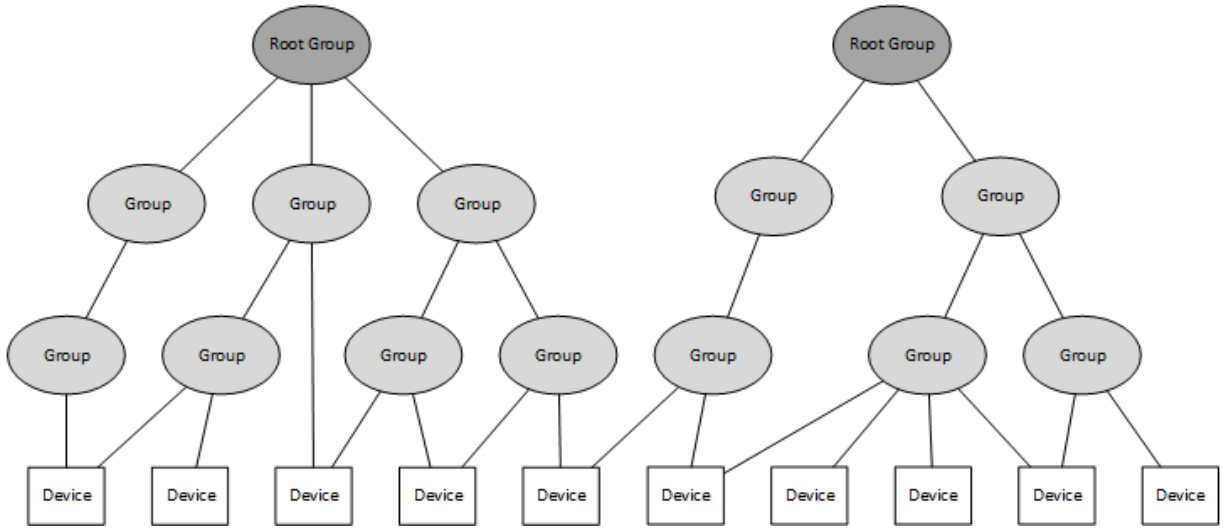


Abbildung 8.12.: Komponentenbaum

Gruppen dürfen beliebig tief verschachtelt werden, sind jedoch einzigartig und haben höchstens eine direkte Elterngruppe. Es dürfen beliebig viele Root Gruppen bestehen. Diese erscheinen in der Navigation zuoberst (ohne aufklappen zu müssen), ansonsten unterscheiden sie sich nicht von ”normalen” Gruppen.

Devices entsprechen den Blattknoten, sie dürfen keine Kindsknoten enthalten. Devices dürfen aber bei beliebig vielen Gruppen als Kindsknoten existieren, somit hat der Benutzer maximale Freiheiten beim Verwalten seiner Devices.

Der Navigationsbaum wurde mit Hilfe der gijgo Library implementiert. Gijgo ist eine JavaScript (jQuery) Library, welche aus Baumstrukturen aufklappbare Navigationen erstellt.

```
$('#tree').tree(
  {
    primaryKey : 'id',
    uiLibrary : 'bootstrap',
    dataSource : ctx + '/groups/getAll'
  });
```

Die Verwendung der Library ist sehr simpel. Auf einem selektierten Div-Container kann die Funktion tree() ausgeführt werden. Als Datenquelle wird eine URL angegeben. Diese API liefert die Baumstruktur im JSON Format, welche von der Gijgo-Library direkt interpretiert wird.

Anbei ist ein einfaches Beispiel einer Komponentenhierarchie.

```
[
  {
    "id": "groups/593e584eed204e174c2a0cc4",
    "text": "ZH",
    "children":
      [
        {
        "id": "devices/593e5845ed204e174c2a0cae",
        "text": "chur_weather_23"
      },
      {
        "id": "devices/593e5845ed204e174c2a0caf",
        "text": "chur_weather_24"
      }
    ]
  }
]
```

8.1.7 Devices

Wählt man in der Navigation ein Device an, so wird eine Anfrage für das HTML Fragment für Devices gestartet.

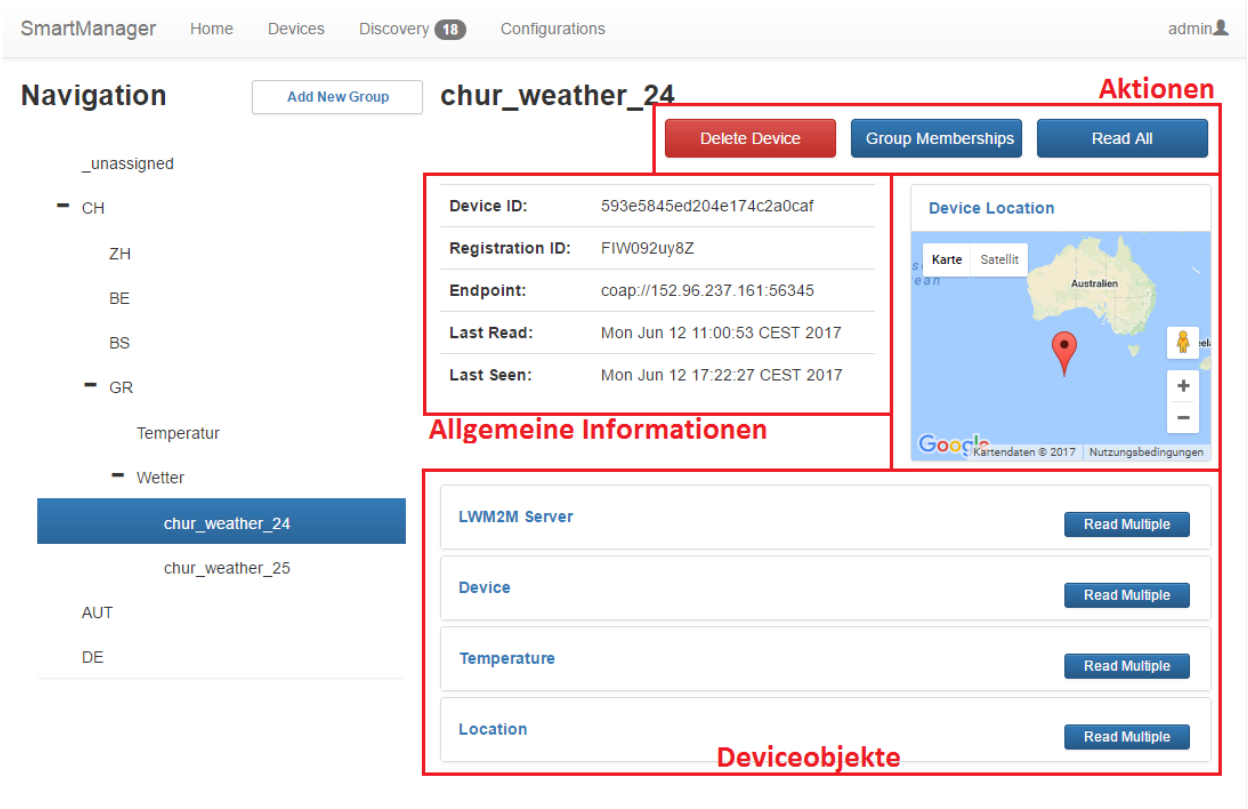


Abbildung 8.13.: Übersicht einzelnes Device

Aktionen Mit ”Delete Device” wird das Device komplett von der Datenbank entfernt. Mit ”Group Memberships” kann das Device unterschiedlichen Gruppen zugewiesen werden. ”Read All” führt ein Lesevorgang über sämtliche verfügbaren Objekte (siehe weiter unten).

Allgemeine Informationen Im Bereich der allgemeinen Informationen erhält der Benutzer nützliche Device-Details angezeigt, ebenfalls wird mittels Google Maps API die Location des Devices automatisch angezeigt.

Deviceobjekte Die Deviceobjekte sind standardmässig zugeklappt. Die einzelnen Objekte sind jeweils eine aufklappbare Panels. Bei der Registration meldet ein Device dem LwM2M Server, welche Objekte es unterstützt. Die Panels werden, abhängig der unterstützten Objekte, dynamisch erzeugt. Mit dem ”Read Multiple” Button werden sämtliche Felder des Objekts gelesen.

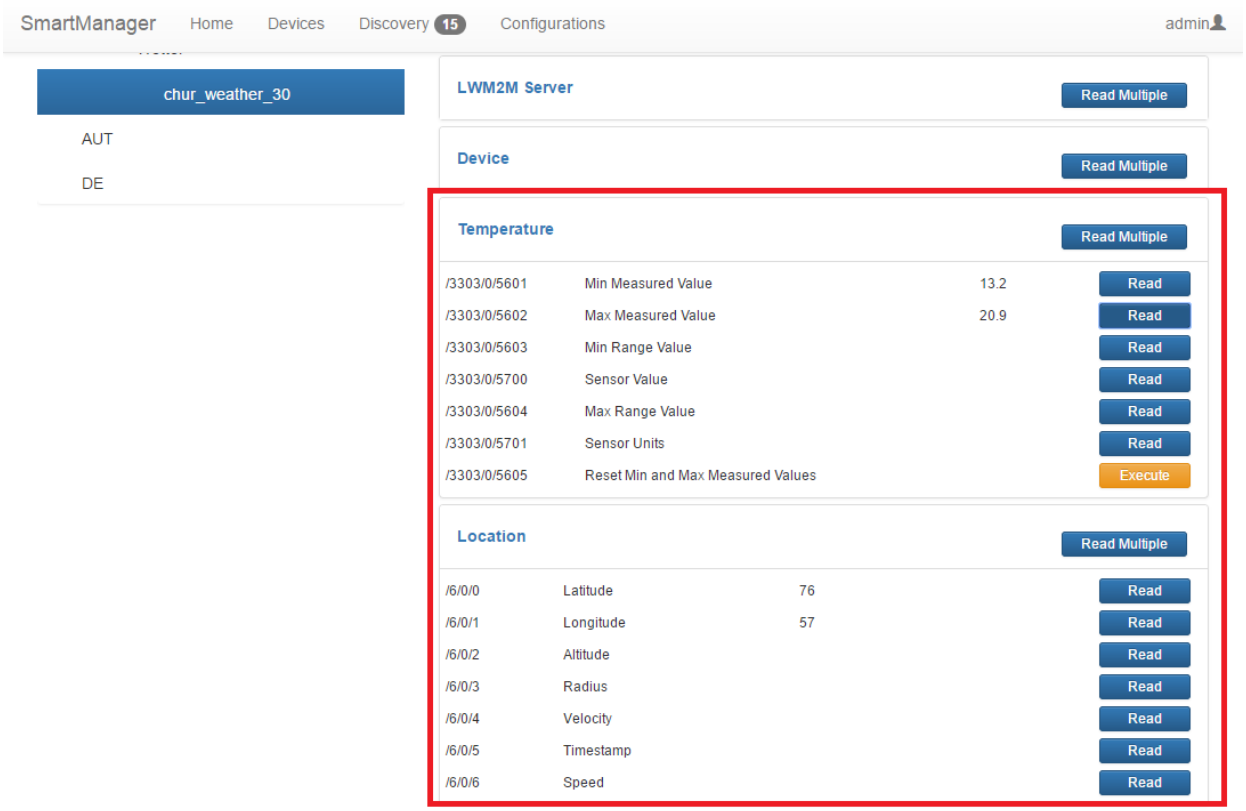


Abbildung 8.14.: Anzeige Objektdetails

Sobald der Benutzer ein Panel anklickt, werden die darunterliegenden Attribute in einer dynamisch generierten Tabelle angezeigt. Zu Beginn sind noch keine Werte vorhanden. Es muss entweder pro Feld einzeln-, oder über den ”Read Multiple”- respektive ”Read All”-Button vom Device gelesen werden.

Auf der rechten Seite werden Buttons für die unterstützten Aktionen angezeigt. Jede Instance ID hat in der XML-Spezifikation hinterlegt, welche Aktionen verfügbar sind.

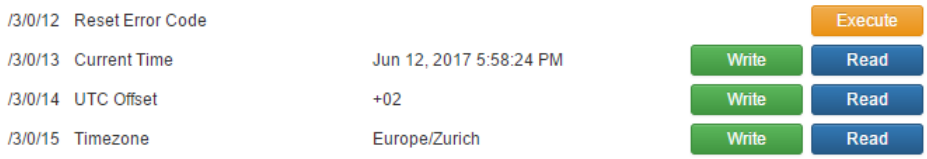


Abbildung 8.15.: RWE Buttons

8.1.8 Groups

In der Gruppenansicht stehen dem Benutzer viele Funktionen für die Verwaltung von Devices und Groups zur Verfügung. In der Map-Übersicht sind alle Devices ersichtlich.

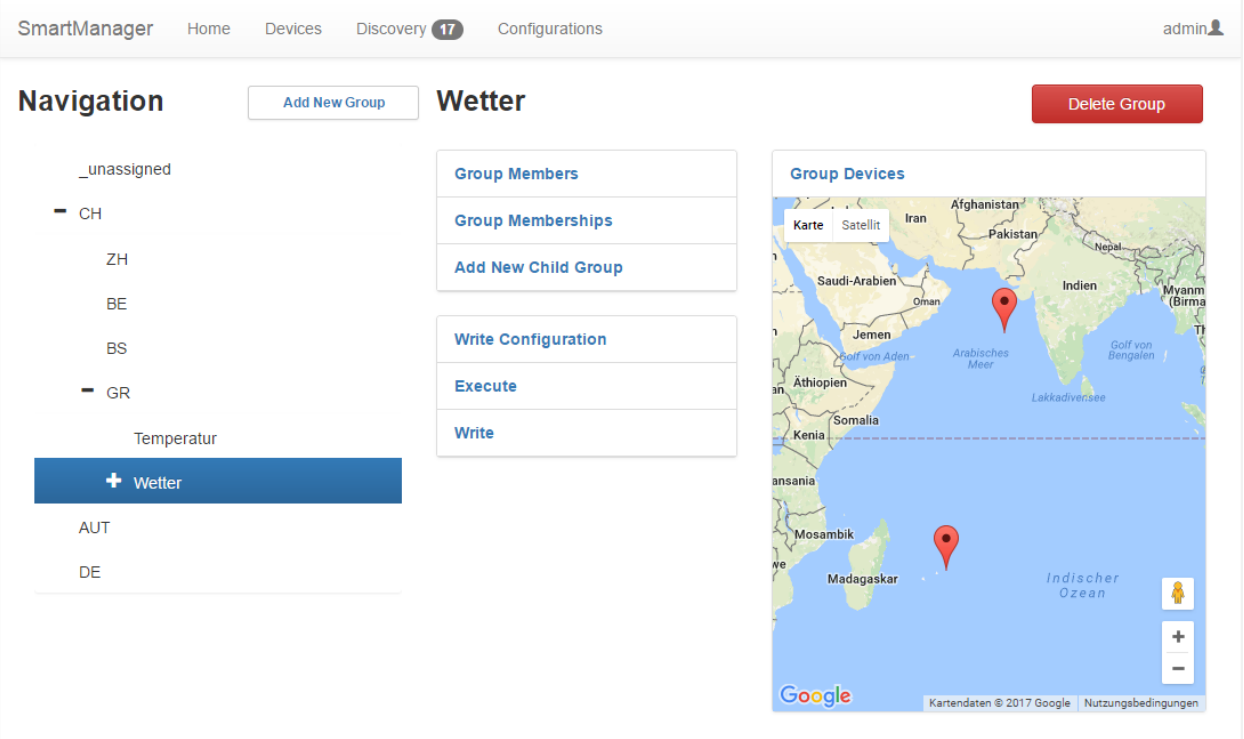


Abbildung 8.16.: Gruppenansicht

Group Members Gruppenmitglieder können unter ”Group Members” verwaltet werden.

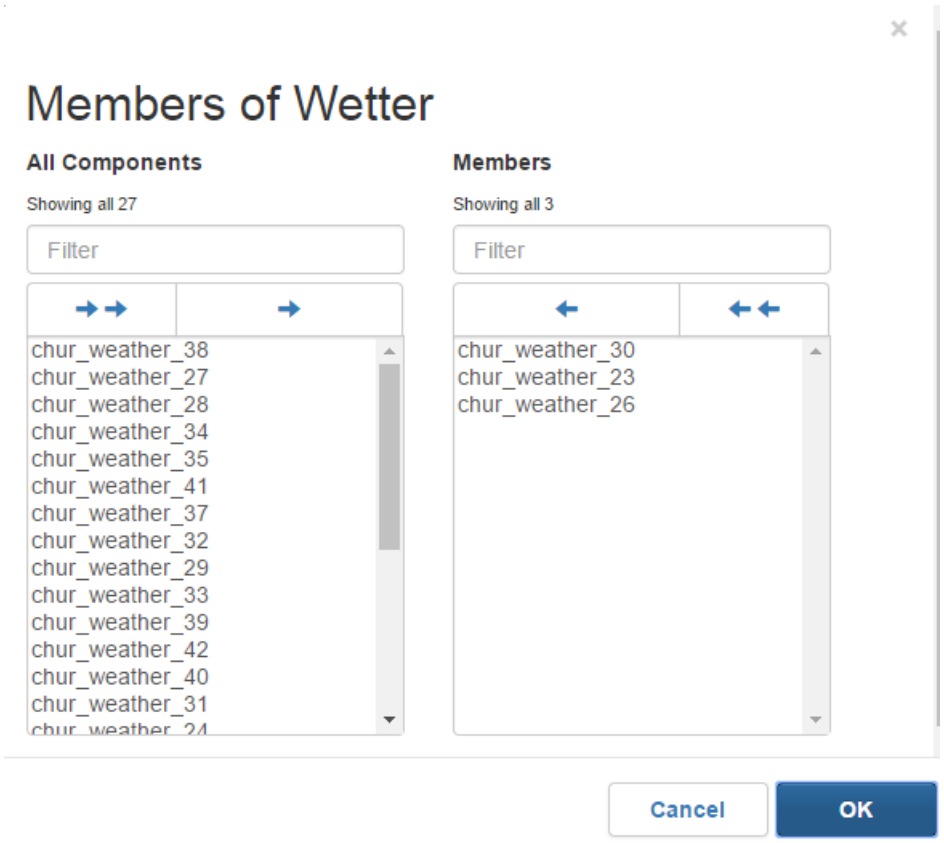


Abbildung 8.17.: Gruppenmitglieder

Sowohl Devices, als auch Gruppen werden auf der linken Seite angezeigt, diese können hinzugefügt werden. Die Liste wird mittels Ajax-Anfrage vom Server abgerufen. Es wird ein Delta zwischen der Menge aller Komponenten und den aktuellen Mitgliedern erstellt.

Auf der rechten Seite sind die Mitglieder aufgelistet, sobald ”OK” gedrückt wird, wird eine Liste mit den angepassten Mitgliedern an den Server gesendet.

Diese Verwaltung wurde mit Hilfe der Bootstrap-Duallist-Box Library implementiert.

Group Memberships Die Mitgliedschaft der Gruppe wird über diesen Button verwaltet. Eine Gruppe kann nur über eine Gruppenzugehörigkeit gleichzeitig verfügen. Die Mitgliedschafsverwaltung wird ebenfalls über die Bootstrap-Duallist-Box Library implementiert.

Add New Child Group Mit dieser Funktion kann der Gruppe eine neue Kindsgruppe hinzugefügt werden. Serverseitig wird überprüft, ob der Gruppenname bereits existiert.

Write Configuration Zuvor erstellte Konfigurationen können auf Gruppenebene verteilt werden. Alle definierten Schreiboperationen werden auf alle Nachfahren verteilt. Nach erfolgter Konfiguration erhält der Benutzer eine Response vom Server und wird über den Erfolg der Schreiboperationen benachrichtigt. Die Response wird auf dem Server generiert und dem Client im HTML Format ausgeliefert.

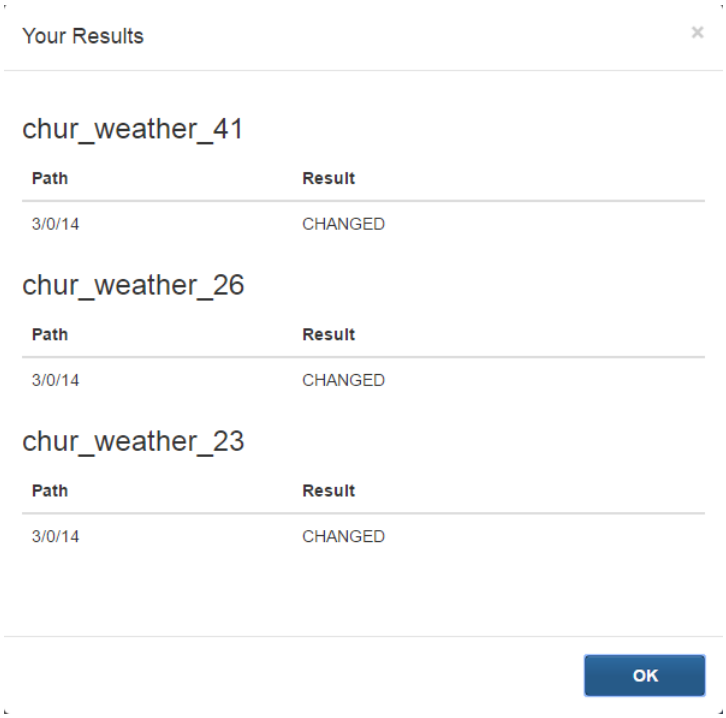


Abbildung 8.18.: Konfigurationsantwort

Execute Execute Funktionen wie beispielsweise einen Neustart können auf Gruppenebene ausgeführt werden. Betroffen sind sämtliche Nachfahren der Gruppe. Execute Funktionen definieren sich in den XML-Models mit der Operation ”E”.

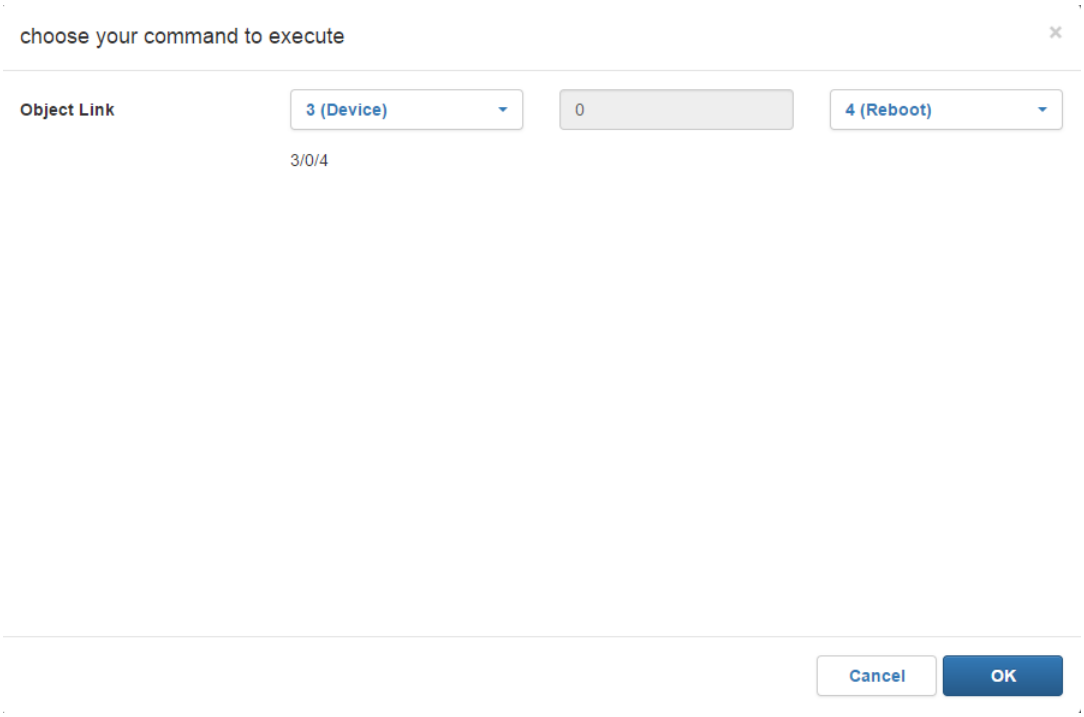


Abbildung 8.19.: Execute

Nach der Auswahl des linken Felds (Object ID) wird eine Anfrage an den Server gesendet. Das mittlere und das rechte Feld werden aufgrund der Response vom Server dynamisch generiert.

Write Die Write Funktion ist identisch zu der Write Configuration Funktion, mit dem einzigen Unterschied, nur ein Konfigurationselement zu senden. Es muss somit nicht eine Konfiguration erstellt werden, der zu beschreibende Wert kann direkt definiert werden.

8.1.9 User Management

In der Benutzerverwaltung kann der eingeloggte User Änderungen an benutzerspezifischen Attributen vornehmen. Momentan kann der Administrator lediglich neue Benutzer hinzufügen und bestehende Benutzer löschen. Änderungen am Passwort können alle Benutzer vornehmen.

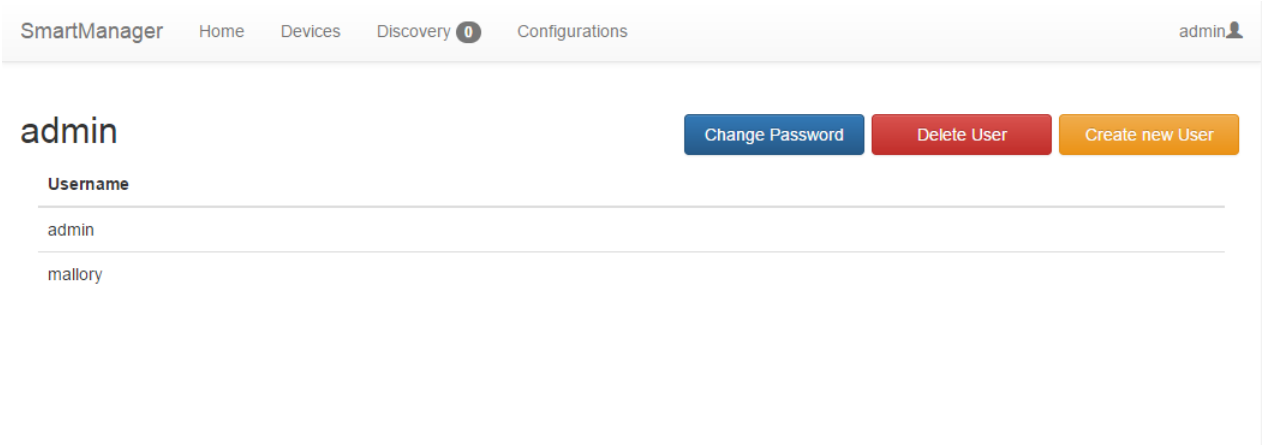


Abbildung 8.20.: Benutzerverwaltung Admin

8.2 Controllers

In diesem Abschnitt wird die Implementiert der einzelnen Controller beschrieben sowie eine Erklärung der verwendeten Spring Funktionen.

8.2.1 Allgemein

Mit den Controllern werden alle Routen und Web API Methoden definiert. Für dies wurde das Spring Framework mit den Libraries spring-webmvc (v4.3.7) und spring-web (v4.3.7) verwendet. Im Controller wurden daher nur noch die Daten aus dem Service-Layer abgefragt, mit der View verbunden und an den Client zurückgesendet.

8.2.2 Verwendete Annotations

@Controller Die @Controller Annotation ist die Spring Bezeichnung für einen Web Controller. Diese Annotation wird direkt über dem Klassennamen platziert. Hier sieht man das Beispiel des DeviceWebController:

```
@Controller
@RequestMapping("/devices")
public class DeviceWebController {
...
}
```

Der Controller führt die zur Route passenden Methoden aus, fragt die Daten vom Service Layer ab, speichert diese in das Model und gibt die View mit den Daten zurück. Dabei ist die Rückgabe immer der Name der View und wird als String an das Spring Framework übergeben.

Dies ist ein Beispiel der Route /groups/id. Als erster Schritt wird die Gruppe aus dem Service-Layer abgerufen und dem Modell hinzugefügt. Danach gibt man im Return-Wert die gewünschte View-Datei an. Hier im Beispiel ist dies die ”groupFragment.jsp” Datei.

```
@RequestMapping(value = "/{id}", method = RequestMethod.GET)
public String showGroupDetails(Model model, @PathVariable("id") String id) {
  model.addAttribute("group", groupService.getGroup(id));
  return "devices/groupFragment";
}
```

@RestController Die zweite Controllerart ist der ”RestController”. Auch diese Annotation wird direkt oberhalb des Klassennamens platziert. Im Gegensatz zur @Controller Annotation gibt der RestController Daten im JSON-Format zurück.

Die Rückgabewerte werden von Spring automatisch in das JSON-Format umgewandelt und an den Client gesendet. Dies setzt voraus, dass der Rückgabetyp entweder eine get-Methode auf alle Felder besitzt oder ein einfacher Datentyp wie zum Beispiel String ist. An diesem Beispiel ist ersichtlich, wie eine Methode des RestControllers aussehen könnte. Es werden alle Gruppen aus dem Service-Layer abgerufen und im JSON-Format an den Client gesendet.

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
public List<DeviceGroup> getGroupList() {
  return groupService.getAllGroups();
}
```

Das durch Spring generierte JSON-Objekt sieht wie folgt aus:

```
[
    {
        "id": "593e57293f56e959e84339f9",
        "name": "_unassigned",
        "children": []
    },
    {
        "id": "593e828a3f56e96a8b665fa8",
        "name": "CH",
        "children": [
            {
                "id": "593e82903f56e96a8b665fa9",
                "name": "Luzern",
                "children": []
            }
        ]
    },
    {
        "id": "593e82903f56e96a8b665fa9",
        "name": "Luzern",
        "children": []
    }
]
```

@RequestMapping Durch das RequestMapping wird die Route und die Methode angegeben, auf welche der Controller reagieren soll. Diese Annotation kann bei der Klasse, sowie bei jeder Methode angegeben werden. Gibt man das RequestMapping bei der Klasse an, gilt dieses für alle weiteren Methoden in der Klasse.

Hier ist ersichtlich wie die Route ”/users/delete” mit der Request-Methode ”POST” definiert wird.

```
@RestController
@RequestMapping("/users")
public class UserRestController {

  @RequestMapping(value = "/delete", method = RequestMethod.POST)
  public String deleteUser(@RequestParam("username") String username) {}
  ...
}
```

@PathVariable Um einer Route eine Pfadvariable zu hinterlegen, gibt es die ”@PathVariable” Annotation. Diese wird in der Parameterliste der Methode eingefügt, um die Werte von der Route zu erhalten.

In diesem Beispiel wurde eine Pfadvariable ”Id” definiert. Um die Variable in der Methode benutzen zu können, wird die Annotation @PathVariable eingefügt. Dadurch nimmt Spring den Id-Teil der URL und wandelt es in diesem Beispiel in einen String um.

```
@RequestMapping(value = "/{id}/delete", method = RequestMethod.DELETE)
public void removeGroup(@PathVariable("id") String id) {
  groupService.deleteGroup(id);
}
```

@RequestParam Um Request Parameter abzufangen, gibt es die @RequestParam Annotation. Durch diese können beispielsweise Formulardaten abgefangen- und der Methode zur Verfügung gestellt werden. Dazu werden alle zu erwartenden Request-Parameter als Variable in der Parameterliste der Methode angegeben.

In diesem Beispiel wird eine Liste von Strings übergeben. Wird alles korrekt übergeben, werden die erhaltenen JSON-Daten in das gewünschte Format umgewandelt.

```
@RequestMapping(value = "/{id}/removeFromGroups", method = RequestMethod.POST)
public void removeFromGroups(@PathVariable("id") String id, @RequestParam("value")
    List<String> value) {
  groupService.removeDeviceFromGroups(value, id);
}
```

8.2.3 Übersicht aller Routen

URI	Method
/	GET
/login	GET
/logout	GET
/users	POST
/users/add	POST
/users/delete	POST
/users/checkUser	POST
/users/userAddFragment	GET
/users/userDeleteFragment	GET
/users/userEditFragment	GET
/users/id/edit	POST
/discovery	GET
/discovery/clean	GET
/configurations	GET
/configurations/add	POST
/configurations/createConfigurationFragment	GET
/configurations/delete	POST
/configurations/id/editConfigurationFragment	GET
/devices	GET
/devices/add	POST
/devices/deleteAll	DELETE
/devices/locations/dashboard	GET
/devices/id	GET
/devices/id/changeMembership	POST
/devices/id/delete	DELETE
/devices/id/memberships	GET
/devices/id/removeFromGroups	POST
/devices/id/locations/mapType	GET
/devices/id/read/objectId	GET
/devices/id/execute/objectId/objectInstanceId/resourceId	GET
/devices/id/write/objectId/objectInstanceId/resourceId	POST
/devices/id/read/objectId/objectInstanceId/resourceId	GET
/groups/add	POST

URI	Method
/groups/getAll	GET
/groups/list	GET
/groups/executeCommandToChildsFragment	GET
/groups/writeCommandToChildsFragment	GET
/groups/writeConfigToChildsFragment	GET
/groups/id	GET
/groups/id/add	POST
/groups/id/changeMembers	POST
/groups/id/changeMembership	POST
/groups/id/delete	DELETE
/groups/id/members	GET
/groups/id/memberships	GET
/groups/id/writeChildDevices/objectId/objectInstanceId/resourceId	GET
/groups/objectId/executeToChildren	GET
/groups/objectId/multiInstance	GET
/groups/objectId/writeToChildren	GET
/groups/id/executeChildDevices/objectId/objectInstanceId/resourceId	GET
/groups/id/writeChildDevices/objectId/objectInstanceId/resourceId	GET

Tabelle 8.1.: **Web API**

8.3 Services

In diesem Abschnitt wird die Implementierung aller Service-Klassen beschrieben und erklärt. Dazu werden Codebeispiele gezeigt, welche eine komplexere Logik besitzen.

8.3.1 Allgemein

Der Service-Layer ist das Kernstück der Web-Applikation. Die Services verbinden den LwM2M-Server, den Data-Layer und den Controller-Layer miteinander. Der Controller muss immer über den Service-Layer kommunizieren, damit eine klare Trennung der Schichten sichergestellt werden kann.

In dieser Schicht wurden keine speziellen Libraries verwendet. Die meisten Methoden greifen auf die darunter liegenden Repositories zu und löschen-, finden- oder fügen neue Objekte ein. Die komplexeren Methoden, welche mehr Logik beinhalten, werden hier aufgeführt und kurz erklärt.

8.3.2 @Service

Jede Service-Klasse wird mit der @Service Annotation deklariert. Dies ist für Dependency Injection wichtig, damit die Service-Klassen gefunden werden, ohne jede Klasse einzeln als Bean zu definieren.

Diese Annotation gehört bei jeder Service-Klasse vor den Klassennamen.

```
@Service
public class DeviceService {
...
}
```

8.3.3 DeviceService

In der DeviceService-Klasse beinhaltet alle Methoden, welche für die Device-Klasse benötigt werden. Die meisten Methoden sind Getter oder führen einfache Datenbankabfragen aus. Bei vielen Methoden wird eine Datenbankabfrage gemacht und durch gewisse Checks abgesichert, bevor die Daten in der Datenbank gespeichert oder an den Controller zurückgegeben werden. Komplexe Methoden sind nur sehr wenige vorhanden. Die Methoden, welche mehr Logik beinhalten, werden hier aufgeführt.

Read-, Write-, Execute-Methoden Damit der Controller-Layer immer über den DeviceService auf den LwM2M-Server zugreifen muss, sind diese Methoden im DeviceService hinterlegt. Die DeviceService-Klasse ist dabei für Abfragen auf Devices zuständig. Mit der ”saveMultipleValueToDevice”-Methode wird auch bei jedem Aufruf der akutelle Wert im Device gespeichert. Hier sieht man das Read Beispiel.

```
public ReadResponse read(String id, int objectId) {
  Device device = getDevice(id);

  ReadResponse response = lwM2MHandler.read(device, objectId);

  if (response != null) {
    Device dev = saveMultipleValueToDevice(response, device, objectId);
    updateDevice(dev);
  }
  return response;
}
```

Add- und Remove im Management Damit ein Device nach einem Discovery aufgenommen wird, können Gruppe und Konfiguration gesetzt werden. Damit ein Device ohne Angaben immer in mindestens einer Gruppe ist, wird dieses implizit in die ”_unassigned”-Gruppe verschoben.

Ob ein Device im Management angezeigt wird, entscheidet das Feld ”Added”. Dieses wird beim hinzufügen auf ”true” gesetzt und beim Entfernen wieder auf ”false”. So ist ein Device daher schon bei der Discovery in der Datenbank hinterlegt.

```
public void addToManagement(String[] deviceIds, String groupId, String configId) {
  DeviceGroup group;
  if (groupId.equals("_unassigned")) {
    group = groupRepo.findByName("_unassigned");
  } else {
    group = groupRepo.findOne(groupId);
  }

  for (String id : deviceIds) {
    Device device = deviceRepo.findOne(id);

    if (!configId.equals("none")) {
      configurationService.writeConfigurationToDevice(id, configId);
    }
    device.setAdded(true);

    deviceRepo.save(device);
    groupService.addDeviceToGroup(group.getId(), id);
  }
}
```

8.3.4 Change Membership

Eine der komplizierteren Methoden ist die ChangeMembership-Methode. Durch diese wird ein Device aus mehreren Gruppen entfernt und wieder hinzugefügt. Dazu muss jedes mal geprüft werden, ob das Device bereits in einer anderen Gruppe vorhanden ist, oder ob es am Schluss ohne zugehörige Gruppe in der Datenbank vorzufinden ist. Dies wird durch eingebaute Checks vermieden und die Devices kommen mindestens in die _unassigned-Gruppe.

```
public void changeMembership(String id, JSONArray value) {
  List<DeviceGroup> preGroups = groupService.listAllGroupsForGroup(id);
  List<DeviceGroup> postGroups = convertJSONArrayToList(value);
  Device device = getDevice(id);

  saveGroupDifference(preGroups, postGroups, device);

  DeviceGroup unassigned = groupService.findByName("_unassigned");
  if (!postGroups.isEmpty() && postGroups.contains(unassigned)) {
    groupService.removeDeviceFromGroup(unassigned.getId(), id);
  }
  if (postGroups.isEmpty()) {
    groupService.addDeviceToGroup(unassigned.getId(), id);
  }
}
```

8.3.5 GroupService

Der GroupService beinhaltet sämtliche Logik, welche die Gruppenverwaltung betrifft. Durch die vielen Möglichkeiten Gruppen zu erstellen, zu verschachteln und neu zu verknüpfen sind die Methoden des GroupServices häufig komplex. Dies hat leider auch die Folge, das es manchmal viele Datenbankabfragen benötigt werden. Sehr viele Methoden sind wie beim DeviceService wieder Getter oder sonstige normalen Datenbankabfragen.

AddGroupToGroup Die AddGroupToGroup Methode ist für die Applikation die Aufwendigste. Es werden sehr viele Datenbankabfragen und Checks ausgeführt, da es keine Inkonsistenten geben darf. Bei jeder Gruppe wird überprüft, ob die gewünschte Child-Gruppe bereits als Vorfahren vorhanden ist. Auch die Parent-Gruppe muss diesen Check ausführen, da es sonst zu zirkulären Abhängigkeiten kommen kann. Auch darf eine Gruppe nur einmal als Child-Gruppe vorhanden sein. Dies wird auch jedes mal überprüft.

```
private void addGroupToGroup(String parent, String child) {
  DeviceGroup parentGroup = groupRepo.findOne(parent);
  DeviceGroup childGroup = groupRepo.findOne(child);

  if (childGroup.getName().equals("_unassigned")  ||
    parentGroup.getName().equals("_unassigned") ||
    isAncestor(parentGroup.getName(), childGroup.getName()) ||
    isAncestor(childGroup.getName(), parentGroup.getName())) {
      return;
  }

  DeviceGroup oldParentGroup = groupRepo.findByChildrenId(new ObjectId(childGroup.
      getId()));

  if (oldParentGroup != null) {
    oldParentGroup.remove(childGroup);
    groupRepo.save(oldParentGroup);
  }
  parentGroup.add(childGroup);
  groupRepo.save(parentGroup);
  groupRepo.save(childGroup);
}
```

findAllChildren Eine weitere wichtige Methode im GroupService ist die findAllChildren Methoden. Diese wird bei jedem rekursiven Ausführen von Methoden auf eine Gruppe ausgeführt. Möchte man zum Beispiel eine Konfiguration auf einer Gruppe der obersten Stufe ausführen, müssen zuerst alle unteren Devices gefunden werden. Dazu werden alle Child-Gruppen der gewünschten Gruppe gesucht und in jeder Gruppe werden alle Devices gesammelt. Diese werden dem Controller dann als Liste zurückgegeben, damit dieser Methoden mit diese Liste ausführen kann.

```
public List<Device> findAllChildren(String id) {
  List<Device> allChildrenDevices = new ArrayList<>();

  DeviceGroup parentGroup = groupRepo.findOne(id);
  List<String> childrenGroup = groupRepo.findAllChildren(parentGroup.getName());

  childrenGroup.add(parentGroup.getName());

  for (String name : childrenGroup) {
    DeviceGroup group = groupRepo.findByName(name);

    for (DeviceComponent device : group.getChildren()) {
      if (device instanceof Device) {
        allChildrenDevices.add((Device) device);
      }
    }
  }
  return allChildrenDevices;
}
```

8.3.6 ConfigurationService

Der ConfigurationService beinhaltet all die Methoden, welche für die Konfiguration wichtig sind. Diese sind vor allem Add-, Save-, Remove- und Get-Methoden. Da diese nicht weiter kompliziert oder speziell sind, werden sie hier nicht genauer dokumentiert.

8.3.7 UserService

Mit dem UserService werden nur sehr wenige Methoden implementiert. Diese sind sehr einfach und werden hier nicht speziell dokumentiert. Neben Add- und Remove-Methoden gibt es einfache Methoden um Passwörter und Usernamen zu überprüfen.

8.3.8 LocationService

Der LocationService bietet keine speziellen Methoden. Es sind im Grunde alles nur Getter, welche die Location der einzelnen Device, Gruppen oder aus allen Devices zurückgibt.

8.3.9 InfrastructureService

Der InfrastructureService bietet Methoden für den Start der Applikation an. Diese Methoden werden bei jedem Start ausgeführt und sorgen dafür, dass immer ein Admin-User, sowie die _unassigned Gruppe vorhanden ist. Zusätzlich werden alle Devices entfernt, welche bei der letzten Benutzung gefunden und nie verwendet wurden. So hat man bei jedem Neustart der Applikation einen sauberen Start und keine Altlasten in der Datenbank.

```
public void startUpClean() {
  if (!groupRepo.existsByName("_unassigned")) {
    DeviceGroup unassigned = new DeviceGroup("_unassigned");
    groupRepo.save(unassigned);
  }
  if (!managementUserRepository.existsByUsername("admin")) {
    ManagementUser admin = new ManagementUser("admin", passwordEncoder.encode("
        adminadmin"));
    managementUserRepository.save(admin);
  }
    deviceRepo.removeDeviceByAddedIsFalse();
}
```

8.4 LwM2M-Server

In diesem Abschnitt wird erklärt wie der LwM2M-Server von Eclipse Leshan eingesetzt wird und wie dieser gestartet- und an eine Adresse gebunden wird.

8.4.1 Allgemein

Damit sich die LwM2M-Clients bei dem Management-Tool melden können, wird ein LwM2M-Server benötigt. In diesem Projekt wurde der Leshan-Server in der Version 1.0.0 verwendet. Diese LwM2M Implementation beinhaltet eine Server- sowie eine Clientumsetzung des LwM2M-Protokolls. Entwickelt wird diese Library von dem Eclipse Project. Leshan baut auf der CoAP-Implementation Californium auf. Diese wurde von der ETH Zürich entwickelt und gehört inzwischen auch zum Eclipse Project.

8.4.2 Leshan Server

Adresse und Port Der LwM2M-Server wird mit der Klasse ”LwM2MManagementServer” erstellt. Als Default wird die Adresse ”127.0.0.1” und der Port 5853 verwendet. Diese Angaben können aber vor dem Start in der LwM2MManagementServer-Klasse angepasst werden, damit der Server eine andere Adresse oder einen anderen Port erhält.

Object Models Jeder Leshan-Server benötigt Object Model Definitionen. Diese werden als File im XML Format im Projektordner abgelegt und danach als Fileobjekt eingebunden. Die Variable resource gibt dabei den Pfad zu den Models an:

```
private Resource resource = new ClassPathResource("ch/hsr/smartmanager/resources/
    models/");
```

Jedes Object Model, welches sich in diesem Pfad befindet, wird vom Server geladen und dargestellt. Benötigt man weitere Object-Modelle, hinterlegt man diese in diesem Ordner und startet die Applikation neu.

Create Methode Für das erstellen des Servers wurde der ”LeshanServerBuilder” verwendet. Dies ist eine einfache Möglichkeit einen Server zu erstellen und diesen danach zu starten.

Hier ist die Implementation der createServer-Methode ersichtlich. Als ersten Schritt fügt man eine Adresse und einen Port hinzu. Nun können die oben erwähnten Object Models geladen und hinterlegt werden. Diese werden von dem LwM2mModelProvider verwaltet.

Diese Angaben reichen bereits um einen Standardserver zu erstellen. Es wird noch ein RegistrationListener hinzugefügt. Der Server wird nun einem TaskExecuter übergeben, um ihn zu starten.

```
@PostConstruct
public void createServer() {
  LeshanServerBuilder builder = new LeshanServerBuilder();

  builder.setLocalAddress(address, port);

  builder.setEncoder(new DefaultLwM2mNodeEncoder());
  LwM2mNodeDecoder decoder = new DefaultLwM2mNodeDecoder();
  builder.setDecoder(decoder);

  File file;
  try { file = resource.getFile(); }
  catch (IOException e) { file = null; }

  models.addAll(ObjectLoader.load(file));
  LwM2mModelProvider modelProvider = new StaticModelProvider(models);
  builder.setObjectModelProvider(modelProvider);

  this.server = builder.build();

  server.getRegistrationService().addListener(
    registrationListenerImpl.getRegistrationListener()
  );

  serverTaskExecutor.doIt(this.server);
}
```

8.4.3 RegistrationListener

Der LeshanServer hat einen Registrierungsdienst, welcher auf neue Anfragen hört. Um einen Zugriff auf diesen zu erhalten, muss man einen neuen RegistrationListener implementieren. Dieser erweitert den Standardlistener und gibt dadurch Zugriff auf die Registred-, Unregistred- und Update-Methoden. So kann das Gerät direkt nach der Registrierung im Management erfasst werden.

Hier sieht man die einfache Umsetzung des in Smartmanager eingesetzten RegistrationListener. Einzige Funktion ist das Hinzufügen oder Anpassen der Registrierung.

```
@Service
public class RegistrationListenerImpl {

  @Autowired
  private DeviceService deviceService;

  public RegistrationListener getRegistrationListener() {

    return new RegistrationListener() {
      @Override
      public void registered(Registration registration) {
        updateOrAddDevice(registration);
      }

      @Override
      public void unregistered(Registration registration, Collection<Observation>
          observerColl) {}

      @Override
      public void updated(RegistrationUpdate registrationUpdate, Registration
          registration) {
        updateOrAddDevice(registration);
      }

    };
  }
}
```

8.4.4 Server TaskExecutor

Um den Server zu starten, wird ein TaskExecutor vom Spring Framework verwendet. Dazu wird ein Thread erzeugt und in diesem wird der Server durch server.start() gestartet. Der Thread wird danach dem TaskExecutor übergeben und gestartet.

```
public class ServerTaskExecutor {
  private class StartTask implements Runnable {

    private final LeshanServer server;

    public StartTask(LeshanServer server) {
      this.server = server;
    }
    public void run() {
      server.start();
    }
  }

  private TaskExecutor taskExecutor = new SimpleAsyncTaskExecutor();
  public void doIt(LeshanServer server) {
    taskExecutor.execute(new StartTask(server));
  }
}
```

8.4.5 LwM2MHandler

Mit dem LwM2MHandler wird die gesamte Kommunikation zwischen den LwM2M-Clients und dem LwM2M-Server durchgeführt. Dazu wurde eine Read-, Write- und Execute-Methode erstellt. Diese Methoden erstellen den passenden Request und übergeben diesen dem Server. Der Server schickt diesen Response an den Client und wartet, bis eine Response zurück kommt. Diese Response wird danach durch die Applikation verarbeitet.

Hier ist die Read-Methode ersichtlich. Die Write- und Execute-Methoden sind sehr ähnlich, es wird jeweils nur andere Requests und Responses verwendet. Mit der im Device hinterlegten Registration-Id wird die richtige Registration auf dem LwM2M-Server gefunden. Dieser sendet danach den Request an den Device und erwartet die Response. Bei einem Fehler wird eine Response mit einer Fehlermeldung erzeugt. Der Request wird danach in die Service-Klasse zurückgegeben.

```
public ReadResponse read(Device device, int objectId) {
  LeshanServer server = lwM2MManagementServer.getServer();

  Registration registration = server.getRegistrationService().getById(device.getRegId
      ());
  if (registration == null) {
    return new ReadResponse(ResponseCode.NOT_FOUND, null, "Device is not reachable");
  }

  ReadRequest request = new ReadRequest(objectId);
  ReadResponse response;

  try {
    response = server.send(registration, request);
  } catch (InterruptedException e) {
    response = new ReadResponse(ResponseCode.NOT_FOUND, null, "Device is not reachable
        ");
  }

  return response;
}
```

Die anderen Methoden sind einfache Getter, welche Object Models zurückgeben oder sehr triviale Methoden, welche selbsterklärend sind.

8.5 Datenbank

In diesem Abschnitt wird die Anbindung der Datenbank, sowie die die Umsetzung der Repositoriers gezeigt. Auch wird erklärt, wie Custom Abfragen implementiert und ausgeführt werden.

8.5.1 Allgemein

Als Datenbank wurde eine MongoDB in der Version 3.4.3 verwendet. Zusätzlich wurden die Pakete mongo-java-driver (v3.4.2), spring-data-jpa (v3.4.2) spring-data-mongodb (v1.11.1) sowie spring-data-commons (v1.13.1) verwendet. Dadurch konnten wir uns sehr viel Codearbeit ersparen, da wir so schon vorgefertigte Repositories eingebunden haben.

Die Datenbank wurde bei der Implementierung lokal gehostet, könnte aber auch leicht auf eine Cloudumgebung ausgelagert werden. Die Datenbank beinhaltet vier Collections: Device, DeviceGroup, ManagementUser und Configuration. Zusätzlich zu jeder Collection gibt es ein Interface mit den angebotenen Repository-Methoden.

8.5.2 Anbindung

Für die Anbindung der Datenbank wurde das Spring Framework verwendet. In der der Datei ”smartmanager-servlet.xml” wurden dazu folgendes Bean und der MongoDB Eintrag hinterlegt:

```
<mongo:db-factory id="mongoDbFactory" client-uri="mongodb://localhost/smartmanager" />

<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate">
  <constructor-arg name="mongoDbFactory" ref="mongoDbFactory" />
</bean>
```

Diese Angaben reichen dem Spring Framework bereits aus, um die Datenbankverbindung herzustellen. Möchte man nun den Pfad der Datenbank auf einen Cloud-Service wechseln, gibt man einfach eine andere Client-Uri an.

8.5.3 Repositories

Spring unterstützt bereits vorgefertigte Repositories. Um diese einzubinden, muss man in der ”smartmanager-servlet.xml” einen ”mongo:repositories”-Eintrag hinterlegen. In diesem wird das Package, in welchem die Repository-Klassen und Interfaces liegen, hinterlegt.

```
<mongo:repositories base-package="ch.hsr.smartmanager.data.repositories" />
```

Library Methoden Pro Datenklasse, welche gespeichert werden soll, wird ein Interface erstellt. Dieses Interface extended das Interface MongoRepository. Dadurch erhält man viele Methoden, wie zum Beispiel findOne() oder save() und muss diese nicht selbst implementieren. Dieses Interface muss nach einem Schema benannt werden. Und zwar beginnt sie immer mit dem Datenklassenname und wird mit ”Repository” erweitert. Hier sieht man ein Beispiel des DeviceRepository. Wichtig ist dabei die Annotation @Repository, damit Spring dieses Interface findet.

```
@Repository
public interface DeviceRepository extends MongoRepository<Device, String>,
    DeviceRepositoryCustom {

  List<Device> findByAdded(boolean added);
  boolean existsByName(String name);
  Device findByName(String name);
}
```

Möchte man weitere Methoden zur Verfügung stellen, können diese im Interface erfasst werden. Dabei können viele Methoden ohne Implementierung erstellt werden. Dazu nimmt man die existierenden Methoden wie zum Beispiel ”find” und hängt das gewünschte Feld als Namen an. Zum Beispiel ”findByName” oder ”findbyLastUpdate” generiert automatisch eine Methode, welche die Datenbank nach dem Datenklassenfeld ”Name” oder ”LastUpdate” durchsucht. So können schon sehr viele Methoden einfach und schnell angeboten werden. Ist das angegebene Feld nicht in der Datenklasse vorhanden, gibt Spring einen Fehler aus und startet nicht.

Es gibt bei den selbst erstellten Methoden aber noch Einschränkungen. Es können keine RemoveBy... Methoden erstellt werden. Für diese muss man immer eine Custom-Implementation erstellen.

Custom Methoden Um eigene Methoden anzubieten, kann man eine Custom-Implementation erstellen. Für dies wird ein weiteres Interface und eine weitere Klasse benötigt. Hier ist die Namensgebung wieder entscheidend, da die Klassen sonst nicht gefunden werden.

Zuerst wird ein Interface mit <Datenklassennamen>RepositoryCustom als Namen erstellt. Hier ist wieder die Annotation @Repository notwendig. Dies ist ein Beispiel des DeviceRepositoryCustom-Interfaces. Alle benötigten Methoden werden hier erfasst.

```
@Repository
public interface DeviceRepositoryCustom {

  void removeDeviceByAddedIsFalse();
  void removeDeviceByName(String name);
}
```

Danach erstellt man eine Klasse mit dem Namen <Datenklassennamen>RepositoryImpl und implementiert die einzelnen Methoden. Dazu verwendet man die Klasse MongoTemplate, um die vorgefertigten Methoden, wie zum Beispiel ”Remove” oder ”Save”, einsetzen zu können. Durch die Query-Klasse kann man jede Abfrage abbilden und so nach spezielleren Kriterien filtern, anstelle von nur einzelnen Feldern.

Hier sieht man ein Beispiel Query, welche alle Devices entfernt, bei denen Added False ist.

```
public class DeviceRepositoryImpl implements DeviceRepositoryCustom {

  @Autowired
  private MongoTemplate mongoTemplate;

  @Override
  public void removeDeviceByAddedIsFalse() {
    Query query = new Query();
    query.addCriteria(Criteria.where("added").is(false));
    mongoTemplate.remove(query, Device.class);
  }
  ...
}
```

8.5.4 Collections

Jede zu speichernde Datenklasse muss mit der Annotation @Document erstellt werden. Eine Collection mit dem Klassennamen wird dadurch automatisch erzeugt.

Zusätzlich wird eine Id und ein leerer Default Konstruktor benötigt. Die Id benötigt auch eine @Id Annotation und sollte wenn möglich den Datentyp String oder ObjectId haben. Es existieren noch weitere Annotations, wie beispielsweise @Indexed, aber es diese wurden bei diesem Projekt nicht verwendet.

```
@Document
public class Device implements DeviceComponent {

  @Id
  private String id;

  private String name;
  private String regId;
  ...

  public Device() {}
```

8.5.5 Datenbankobjekt

In der Datenbank wird jede Datenklasse als JSON-Objekt abgespeichert. Dieses beinhaltet alle Felder der Klasse, sowie eine generierte Id und eine Klassendefinition. Dadurch kann Spring das Objekt sehr einfach wieder zu einem Java-Objekt umwandeln und zur Verfügung stellen.

Hier ist ein Device ersichtlich, wie es auf der Datenbank gespeichert sein könnte. Alle Datumsformate werden automatisch in ”ISODate” umgewandelt und alle Ids werden von ”String” in ObjectIds umgewandelt. Alle anderen Typen sind nativ unterstützt und müssen nicht weiter umgewandelt werden.

```
{
  "_id" : ObjectId("593ab77b3f56e9309bbb4cf1"),
  "_class" : "ch.hsr.smartmanager.data.Device",
  "name" : "chur_weather_104",
  "regId" : "tEBwKboGWD",
  "endpoint" : "coap://127.0.0.1;42941",
  "lastUpdate" : ISODate("2017-06-09T14:58:03.228Z"),
  "latitude" : "44.0",
  "longitude" : "-30.0",
  "lastRegistrationUpdate" : ISODate("2017-06-09T15:38:34.117Z"),
  "objectLinks" : [ "/1/0", "/3/0", "/3303/0", "/6/0" ],
  "added" : true,
  "dataMap" : {  }
}
```

8.6 Login

In diesem Abschnitt wird der komplette Login- und Logout-Vorgang erklärt. Zusätzlich wird die HTTPS Implementation und die Routenzugriffsrestriktionen gezeigt.

8.6.1 Allgemein

Für das Login und die HTTPS-Umleitungen wurden die zwei Libraries spring-security-crypto (v4.2.2) und spring-security-oauth2 (v2.1.0) verwendet. Bei der Crypto-Library wurde der Password-Encoder verwendet, um alle Passwörter sicher in der Datenbank abzuspeichern. Die Oauth2-Library wurde für die HTTPS-Umleitung und für das Loginformular verwendet.

8.6.2 Security Einstellungen

Alle Einstellungen im Bereich Security werden in der Datei ”security-config.xml” konfiguriert. Durch den Namespace ”sec” werden die Routen abgesichert und die Login-Form definiert.

Hier sieht man einen Ausschnitt aus der security-config.xml Datei. Für jeden Eintrag werden Pattern, Access- und Require-Channel definiert. Das Pattern bestimmt die Route, welche definiert werden soll. Dazu kann man eine fixe Route setzen, wie zum Beispiel ”/”. Wenn man auch noch Subrouten ansprechen möchte, kann man dies mit einem ”*” für eine tiefere Stufe oder mit ”**” für eine beliebig tiefe Route. Mit dem Access Parameter kann der Zugriff gesteuert werden. mit ”permitAll” kann jeder Benutzer auf die Route zugreifen. Möchte man die Route durch das Login absichern, gibt man ”isAuthenticated()” an. Diese Methode ist im Spring Framework implementiert worden und checkt bei jedem Aufruf, ob der Benutzer berechtigt ist. Mit ”requires-channel” gibt man an, ob die Route per HTTP- oder HTTPS erreichbar ist.

Man sieht das nur die /login und /logout für jeden aufrufbar ist. Alle anderen Routen sind durch das Login gesperrt. Die zwei letzten Zeilen definieren die Login- und Logout-URL.

```
<sec:http use-expressions="true">
  <sec:intercept-url pattern="/login" access="permitAll" requires-channel="https"/>
  <sec:intercept-url pattern="/logout" access="permitAll" requires-channel="https"/>

  <sec:intercept-url pattern="/" access="isAuthenticated()" requires-channel="https"
      />
  <sec:intercept-url pattern="/*" access="isAuthenticated()" requires-channel="https"
      />

  ...

  <sec:form-login login-page="/login" authentication-failure-url="/login?error=true"
      />
  <sec:logout logout-success-url="/logout" />
</sec:http>
```

Ein Authentication-Manager wird zusätzlich zu den Routen Einstellungen benötigt. Dieser ist für die Überprüfung und Speicherung des Passworts zuständig. Als Authentication-Provider gibt man seine Service-Klasse an, welche für die Logins zuständig ist. Damit das Passwort nicht in Klartext gespeichert wird, wird ein PasswortEncoder konfiguriert. Dieser fügt das Passwort einen Salt-Wert hinzu und speichert einen Hash-Wert davon.

Hier sieht man die Umsetzung der Security Einstellungen. Als Authentication-Provider wurde der ”userService” verwendet und als PasswordEncoder der ”BCryptPasswordEncoder” von dem Spring Framework.

```
<sec:authentication-manager>
    <sec:authentication-provider
      user-service-ref="userService">
      <sec:password-encoder ref="passwordEncoder"></sec:password-encoder>
    </sec:authentication-provider>
  </sec:authentication-manager>

  <bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.
      BCryptPasswordEncoder"></bean>

  <bean id="userService" class="ch.hsr.smartmanager.service.applicationservices.
      UserService"></bean>
```

8.6.3 Authentication-Provider

Damit Spring die Logindaten überprüfen kann, muss der User-Service das Interface UserDetailsService implementieren. Dies heisst es wird eine loadUserbyUsername-Methode benötigt. Spring ruft bei jedem Login diese Methode auf, holt den Benutzer von der Datenbank und checkt die Username/Password-Kombination. Der Rest muss nicht implementiert werden, denn Spring erledigt die restlichen Schritte ohne weitere Implementierungen.

```
@Override
public UserDetails loadUserByUsername(String username) throws
    UsernameNotFoundException {
  ManagementUser user = userRepository.findByUsername(username.toLowerCase());
  if (user == null) {
    throw new UsernameNotFoundException(username);
  } else {
    return new User(user.getUsername(), user.getPassword(), new ArrayList<>());
  }
}
```

8.6.4 Login Formular

Für das Login wurde eine neue ”login.jsp”-Datei erstellt. Diese enthält das Loginformular und als Action die Standardroute von Spring, für die Loginüberprüfung. Wenn man in der security-config.xml Datei nichts weiteres eingestellt hat, müssen alle Felder wie im Beispiel benannt werden. Auch die Action der Form muss auf j_spring_security_check zeigen, damit im Spring Framework die richtige Methode ausgeführt wird. Hier sieht man das vereinfachte JSP-File der Loginseite.

```
<form role="form" name='f' action='${pageContext.request.contextPath}/
    j_spring_security_check' method='POST'>
  <input type="text" name="j_username" value='' placeholder="Username" required
     autofocus>
  <input type="password" name="j_password" placeholder="Password" value="">
  <button name="submit" type="submit" value="Login">Sign in</button>
</form>
```

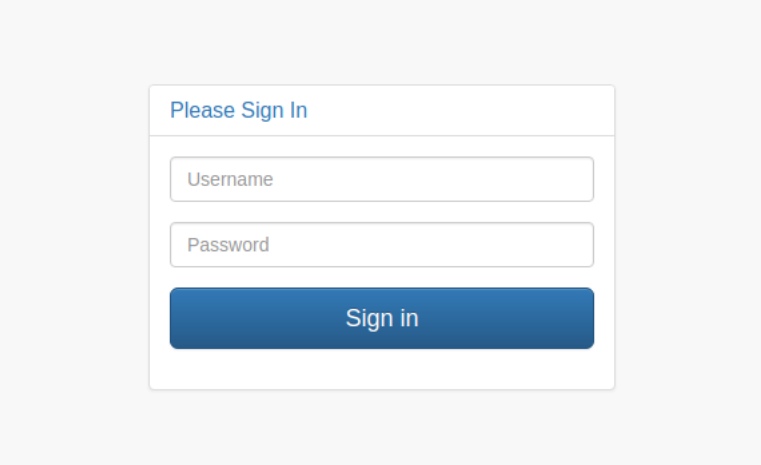


Abbildung 8.21.: Login Formular

8.7 Demo LwM2M-Client

In diesem kurzen Abschnitt wird der eingesetzte Democlient erklärt. Dieser wird bei der Schlusspräsentation sowie bei der Ausstellung eingesetzt.

8.7.1 Allgemein

Wie auch der Server, ist der LwM2M-Client mit der Leshan-Library eingesetzt. Bei der Leshan-Library gibt es bereits einen Demo-Client. Dieser wurde übernommen und für die Demoumgebung angepasst.

8.7.2 Aufbau

Als Hardware wird ein Raspberry Pi 3 verwendet. Auf dem Gerät wird ein schlankes Linux installiert sowie das Java Runtime Environment. Weitere Software wird nicht benötigt. Der Leshan-Democlient kann von Github bezogen werden. Der Client bekommt eine angepasste Location sowie die Definitionen für den Temperatursensor.

Temperatursensor Bei dem Demo-Client wird ein Temperatursensor verwendet. Dieser wird wie folgt angeschlossen. Die Sensoren werden wie folgt an das Geräte angeschlossen:

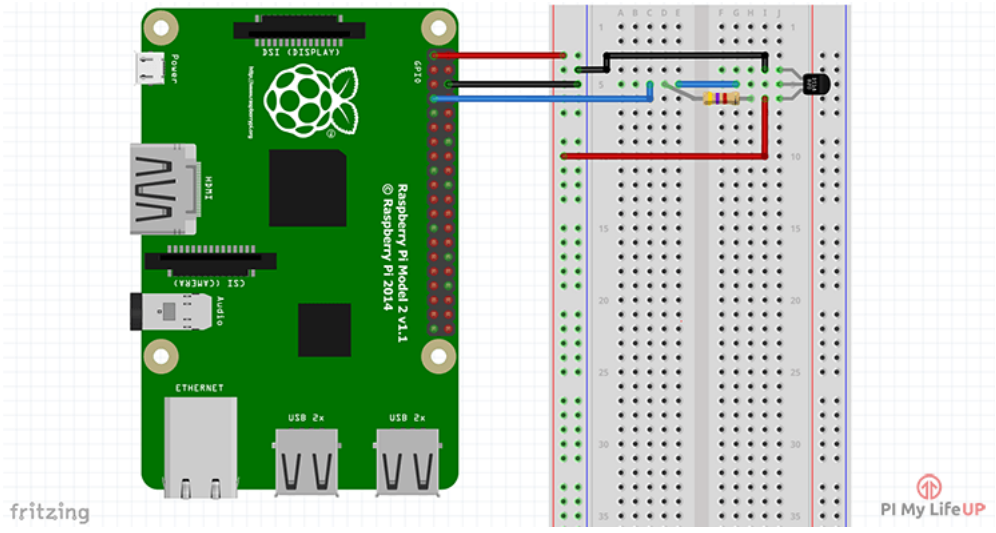


Abbildung 8.22.: Temperatur-Device Schema[32]

8.8 Verwendete Libraries und Software

In diesem Kapitel werden alle verwendeten Libraries und Software mit den dazugehörigen Lizenzen aufgelistet.

8.8.1 Back-End

Alle im Back-End verwendeten Libraries sind hier aufgelistet. Beim Spring Framework wurden mehrere Libraries verwendet. Diese sind nicht alle separat aufgelistet.

Library	**Lizenz**
Spring Framework	Apache 2.0
Javax.servlet – Servlet-Api	CDDL 2
Javax.servlet – Jstl	CDDL 2
Javax.servlet – Jsp	CDDL 2
Leshan	EDL 1.0 / EPL 1.0
Mongo-java-driver	Apache 2.0
org.json	JSON
hamcrest	BSD 2-clause
JUnit	EDL 1.0 / EPL 1.0
json-path	Apache 2.0

8.8.2 Front-End

Dies ist eine Auflistung der im Front-End verwendeten CSS und Javascript Libraries und die dazugehörigen Lizenzen.

Library	**Lizenz**
jQuery	MIT License
Bootstrap	MIT License
Bootbox	MIT License
Font Awesome	MIT License
gijgo	MIT License
MetisMenu	MIT License
Prettify	MIT License
sb-admin-2	MIT License
Bootstrap Dual Listbox	Apache License 2

8.8.3 Entwicklungssoftware

Alle für die Entwicklung verwendeten Tools und Software wird hier aufgelistet.

Software	**Beschreibung**
Eclipse	IDE für die Entwicklung des Back-End
java	Programmiersprache, welche im Back-End einsetzt worden ist
Git	Versionskontrollsystem
Maven	Softwareprojekt Management Tool
MongoDB	dokumentenorientierte NoSQL-Datenbank,

8.9 Testing

In diesem Kapitel ist das Testing der Applikation beschrieben. Auf Unit-Tests wurde verzichtet, da ohne aufwendiges Refactoring der bestehende Code sehr schwierig zu testen ist. Stattdessen werden Integrationstests verwendet, um das Zusammenspiel der Klassen über die verschiedenen Softwareschichten zu testen.

Verwendete Libraries Für die Tests wurde neben Junit auch die Libraries von dem Spring Framework verwendet. Das Spring Framework erstellt die Mocks der Web-Umgebung und sorgt dafür, dass alle Einstellungen übernommen werden.

Damit die Testumgebung auf alle Beans der Produktiven-Umgebung zugreifen kann, werden bei jeder Test-Klasse folgende Annotations hinzugefügt:

```
@WebAppConfiguration
@ActiveProfiles("dev")
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"file:WebContent/WEB-INF/smartmanager-servlet.xml"})
```

Durch @ActiveProfiles ”dev” werden zu den Standard-Beans zusätzlich die mit profile=”dev” bezeichneten Beans geladen. Die produktive Umgebung benutzt weiterhin nur die Beans ohne profile=”dev” Angabe. So kann man sicher sein, dass die Testumgebung nicht die Produktionsumgebung verändert. Bei der Datenbank ist dies wichtig, da man nicht mit der produktiven Datenbank testen sollte.

Hier sieht man ein Beispiel, wie die Testdatenbank angebunden wird und wie man das profile Attribute setzt.

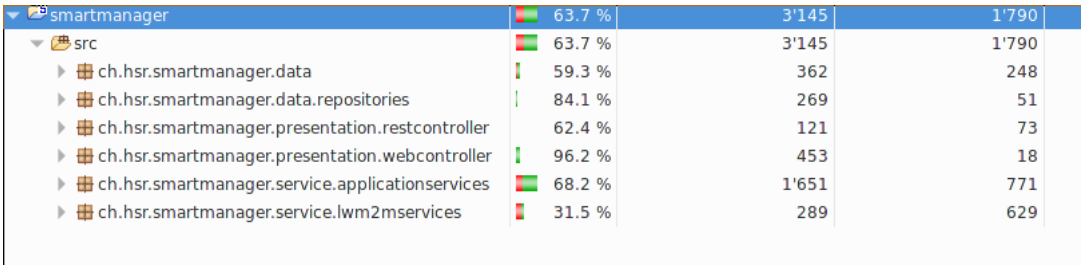
```
<beans profile="dev">
  <mongo:db-factory id="mongoDbFactory" client-uri="mongodb://localhost/test" />
</beans>
```

Beispieltest Bei jedem Test wird die MockMvc-Klasse verwendet. Diese schickt Anfragen an den Webserver und wartet auf den Response. Mit ”.andExpect” kann dieser Response nun geprüft werden. In diesem Beispiel wird die JSON-Response auf den Inhalt von Feldern geprüft.

```
@Test
public void listAllGroups() throws Exception {
  mockMvc.perform(get("/groups/list").principal(principal).accept(MediaType.
      APPLICATION_JSON))
    .andExpect(jsonPath("$[*].name", hasItem("group1")))
    .andExpect(jsonPath("$[*].name", hasItem("subgroup1")))
    .andExpect(jsonPath("$[*].name", hasItem("subgroup2")));
}
```

Viele Methoden konnten nicht nur durch die ”.andExpect”-Variante getestet werden. Diese wurden mit den von Junit angebotenen Methoden getestet.

8.9.1 Testabdeckung



smartmanager	63.7 %	3'145	1'790
src	63.7 %	3'145	1'790
ch.hsr.smartmanager.data	59.3 %	362	248
ch.hsr.smartmanager.data.repositories	84.1 %	269	51
ch.hsr.smartmanager.presentation.restcontroller	62.4 %	121	73
ch.hsr.smartmanager.presentation.webcontroller	96.2 %	453	18
ch.hsr.smartmanager.service.applicationservices	68.2 %	1'651	771
ch.hsr.smartmanager.service.lwm2mservices	31.5 %	289	629

Abbildung 8.23.: Testabdeckung

Durch die durchgeführten Integrationstests wurde eine Testabdeckung von 63.7% erreicht. Da es mit der Testumgebung nicht möglich war, einen LwM2M-Client zu simulieren, könnten viele Methoden nicht getestet werden. Das heisst alle read, write und execute Routen konnten nicht getestet werden. Deshalb ist die Testabdeckung beim lwm2mservice-Package so gering.

9. Ergebnisse

9.1 Analyse

In der Analyse wurde eine breite Einarbeitung in das IoT-Themengebiet erarbeitet. Es hat sich gezeigt, dass sich IoT-Devices in vieler Hinsicht von herkömmlichen PCs und Notebooks unterscheiden. Es sind neue Protokolle entstanden, um den eingeschränkten Verhältnissen im IoT-Umfeld gerecht zu werden.

Wichtige Aufgaben des Device Managements wurden in Kapitel 2 erarbeitet. Durch die grosse Anzahl an neuen Sensoren ist Automatisierung gefragter denn je. Wichtige Prozesse wie Bootstrapping, Monitoring, Updating und Konfiguration müssen im IoT-Bereich skalierbar aufgebaut sein.

Mit LwM2M ist ein massgeschneidertes IoT-Devicemanagement-Protokoll entstanden. Durch die Funktionen Bootstrapping, Registration, Resource-Access und Reporting können viele wichtige Device Management Aufgaben mit diesem Protokoll erledigt werden. Vor allem der standardisierte Ressourcenzugriff hat sich als wertvoll erwiesen. Es muss somit nicht mehr auf devicespezifische Implementierungen geachtet werden.

Die Security Analyse hat gezeigt, dass zukünftig die Bedeutung der Sicherheit noch weiter steigen wird. Die unterschiedlichen Bereiche im IoT-Umfeld wurden analysiert und wichtige, sicherheitsrelevante Aspekte erarbeitet. Zu den bestehenden Gefahren von herkömmlichen IT-Infrastrukturen kommen neue Herausforderungen. Lösungsansätze für sicheres Bootstrapping und Authentisierung über Zertifikate wurden erarbeitet.

9.2 Smartmanager

Mit dem ”Smartmanager” ist ein funktionsfähiger Prototyp einer IoT-Management-Applikation entstanden. Als Basis für den LwM2M Server wurde die Eclipse Leshan Library verwendet. Das Web-Back-End wurde in Java mit dem Spring Framework realisiert.

Durch den Einsatz des LwM2M Protokolls können theoretisch viele unterschiedliche IoT-Devices verwaltet werden. Das IoT-Device muss dabei einen LwM2M Client installiert haben. Die Kommunikation mit LwM2M Clients gestaltet sich durch das standardisierte Ressourcenmodell leicht. Vom Client unterstützte Objekte und Ressourcen werden in dieser Applikation dynamisch angezeigt.

LwM2M unterstützt mit ”Read”, ”Write”, ”Execute” und ”Observe” vier verschiedene Operationen. Die ersten drei Operationen sind implementiert, die ”Observe”-Funktion ist nicht verfügbar. Benutzer können Konfigurationen für Devices anlegen. Konfigurationselemente sind dabei Ressourcen, welche die LwM2M Write-Operation unterstützen. Benutzer haben die Möglichkeit, eine Gruppenstruktur für die Organisation zu erstellen. Somit können Devices in unterschiedliche Gruppen verteilt werden. Auf einer Discovery-Seite sind alle Devices ersichtlich, welche sich beim Management Server registriert haben und vom Benutzer noch nicht hinzugefügt worden sind. Beim Hinzufügen eines Devices kann vom Benutzer eine Gruppe und eine Konfiguration gesetzt werden.

Durch die implementierte Gruppenstruktur können viele IoT-Geräte effizient verwaltet werden. Möchte der Benutzer Aktionen auf vielen IoT-Devices vornehmen, so kann er Konfigurationen, Write- und Execute-Operationen auf eine Gruppe anwenden. Detaillierte Feedbacks über vorgenommene Aktionen wurden implementiert, sodass der Benutzer über den Erfolg informiert wird.

10. Schlussfolgerungen

10.1 Ergebnisbewertung

Positives

- Erkenntnisreiche Analysen für IoT
- Grosser Funktionsumfang dank LwM2M Ressourcenmodell
- Potenziell breite Unterstützung für Devices dank standardisiertem LwM2M Protokoll
- Konfigurationen für Devices
- Hierarchische Gruppenverwaltung
- Ausführung von Operationen (Read, Write, Execute) auf Gruppenebene ermöglichen effizientes Management

Negatives

- Viele wichtige Sicherheitsaspekte nicht implementiert (Deviceauthentisierung, Verschlüsselung des CoAP Verkehrs, etc.)
- Testing zu wenig ausgereift (Unit Tests, Systemtests)
- Einige Teile des Codes müssten refactored werden (schwierig testbar)

10.2 Ausblick

IoT-Devicemanagement wird für viele Unternehmen ein Thema werden. Es ist erfreulich, dass die OMA mit LwM2M ein Protokoll entwickelt hat, um eine einheitliche Managementlösung für IoT-Devices zu ermöglichen. Es bleibt zu hoffen, dass viele Hersteller im IoT-Umfeld dieses Protokoll implementieren werden.

Mit dem Smartmanager konnte ein funktionsfähiger Prototyp eines Device Management Servers entwickelt werden. Wie zu erwarten war, konnten in diesem Projekt zeitlich nicht alle wichtigen Aspekte berücksichtigt werden. Für die produktive Nutzung des Tools müssten wichtige Sicherheitsfeatures und Bootstrapping implementiert werden.

Das Projektteam empfiehlt für die Weiterentwicklung folgende Massnahmen:

- Code Refactoring der gesamten Applikation für eine bessere Testbarkeit
- Performanceanalysen und gegebenfalls Leistungsoptimierungen
- Entkopplung des LwM2M Servers vom Management Server (eigenes Servlet)
- Implementierung eines Bootstrap Servers (Eclipse Leshan Library)
- Implementierung von Security-Features (CoAP Verschlüsselung und sicheres Bootstrapping)
- Verwendung eines JavaScript Frameworks für ein moderneres User Interface

Teil II.

Anhang

1. Projektplan

1.1 Projektübersicht

In diesem Projekt soll der Stand der Entwicklungen im Bereich „Internet of Things“ aufgezeigt werden. Die verschiedenen Arten der eingesetzten Sensoren sollen ermittelt- und die Einsatzgebiete untersucht werden. Heutzutage werden in der Industrie bereits verschiedenartige Sensoren eingesetzt. Die Anzahl der eingesetzten Sensoren steigt drastisch. Schon bald stellt sich die Frage, wie man mit der steigenden Anzahl Sensoren deren Management realisieren soll.

1.1.1 Zweck und Ziel

Die Bachelorarbeit soll den Nachweis der Problemlösungsfähigkeit unter Anwendung ingenieurmässiger Methoden nachweisen. Entsprechend verfügt die Arbeit über einen konzeptionellen, theoretischen und einen praktischen Anteil.

1.1.2 Projektorganisation

Vorname	Name	E-Mail
Andreas	Stalder	astalder@hsr.ch
David	Meister	dmeister@hsr.ch

Das Projekt wird von Prof. Beat Stettler und Urs Baumann betreut und benotet.

1.2 Management Abläufe

1.2.1 Zeitbudget

Der Projektstart ist am Montag, dem 20. Februar 2017.
Die Projektdauer beträgt 17 Wochen, und das Projektende ist am Freitag, dem 16. Juni 2017.

Während diesen 17 Wochen sind 360 Arbeitsstunden pro Projektmitglied eingeplant. Das entspricht pro Mitglied eine Arbeitszeit von ca. 22 Stunden pro Woche. Dies ergibt einen totalen Aufwand von ca. 720 Stunden.

Die wöchentliche Arbeitszeit von 22 Stunden kann bei Verzug oder bei unerwarteten Problemen auf maximal 30 Stunden erhöht werden.

Es sind gegenwärtig keine Absenzen während dieser Zeit geplant.

1.2.2 Projektphasen

Das Projekt wird in fünf Phasen unterteilt: Initialisierung, Analyse, Design, Realisierung und Abschluss.

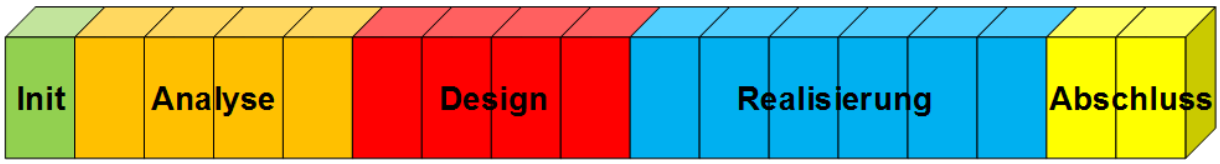


Abbildung 1.1.: Projektphasen

1.2.3 Meilensteine

Das Projekt beinhaltet insgesamt vier Meilensteine.

Meilenstein	Beschreibung	Datum
MS1	Anforderungen und Scope definiert	26.03.2017
MS2	Architektur und Design beschrieben	23.04.2017
MS3	Software fertiggestellt, Codefreeze	04.06.2017
MS4	Arbeitsabgabe	16.06.2017

Tabelle 1.1.: **Projekt Meilensteine**

1.2.4 Iterationen

Die Dauer eines Iterationszyklus beträgt jeweils eine Woche.

Iteration	Inhalt	Start	Ende
Initialisierung 1	Kickoff Meeting, Projektplanung, Infrastruktur	20.02.2017	26.02.2017
Analyse 1	IoT-Analyse Allgemein	27.02.2017	05.03.2017
Analyse 2	IoT-Analyse Allgemein	06.03.2017	12.03.2017
Analyse 3	Sensoren Analyse technisch & Evaluation	13.03.2017	19.03.2017
Analyse 4	Requirements definieren	20.03.2017	26.03.2017
Design 1	Architektur beschreiben, Risikoanalyse	27.03.2017	02.04.2017
Design 2	Prototyp programmieren	03.04.2017	09.04.2017
Design 3	Prototyp programmieren	10.04.2017	16.04.2017
Design 4	Requirements- und Architektur Review	17.04.2017	23.04.2017
Realisierung 1	Implementation	24.04.2017	30.04.2017
Realisierung 2	Implementation	01.05.2017	07.05.2017
Realisierung 3	Implementation	08.05.2017	14.05.2017
Realisierung 4	Implementation	15.05.2017	21.05.2017
Realisierung 5	Implementation	22.05.2017	28.05.2017
Realisierung 6	Refactoring / Bugfixing	29.05.2017	04.06.2017
Abschluss 1	Dokumentation	05.06.2017	11.06.2017
Abschluss 2	Dokumentation	12.06.2017	16.06.2017

Tabelle 1.2.: **Projekt Iterationen**

1.2.5 Arbeitspakete

Name	Inhalt	Iteration	Wer	Soll	Ist
Initialisierung					
Kickoff-Meeting	Allgemeine Besprechungen zum Projektstart	Initialisierung 1	Alle	1	1
Dokumenterstellung	Erstellung LaTeX Vorlagen	Initialisierung 1	Alle	4	5
Projektplan	Zeitplanung, Phasen, Meilensteine	Initialisierung 1	Alle	5	6
Einrichtung Projektmanagement Software	Installation und Einrichtung Jira	Initialisierung 1	Alle	3	5
Analyse					
Einarbeitung IoT-Allgemein	IoT-Übersicht erarbeiten und dokumentieren	Analyse 1/2	dm	25	32
Einarbeitung Sensortypen	Sensortypen recherchieren und dokumentieren	Analyse 1/2	as	15	12
Einarbeitung Kommunikation	IoT-relevante Kommunikationsbereiche analysieren und dokumentieren	Analyse 1/2	Alle	40	44
Einarbeitung Management	Device Management in IoT	Analyse 3	Alle	40	52
Requirements	Funktionale Anforderungen (Use Cases) und Nichtfunktionale Anforderungen	Analyse 4	Alle	40	29
Design					
Architekturübersicht	Übersichtsverschaffung, Grobkonzept	Design 1	dm	8	6
Domain Model	Erstellung des Domänenmodells	Design 1	Alle	8	8
Schichtenarchitektur	Erstellung der Schichtenarchitektur, Packages schnüren	Design 1	as	8	6
Auswahl Frameworks	Frameworks für Front- & Backend sowie Datenbank analysieren und auswählen	Design 1-4	Alle	60	76
Prototyp	Erstellung eines Prototypen über alle Softwareschichten	Design 2-4	Alle	90	128
Realisierung					

Name	Inhalt	Iteration	Wer	Soll	Ist
Anpassungen Architektur	Anpassungen Architektur gem. Reviews	Realisierung 1	Alle	30	22
Erstellung Views	Seitenaufbau mit verschiedenen Views aufgebaut	Realisierung 1	Alle	10	14
Gruppenverwaltung implementieren	Navigationsbaum, Gruppen CRUD etc.	Realisierung 2-4	Alle	53	58
Deviceverwaltung implementieren	Anzeige, Kommunikation	Realisierung 2-4	Alle	47	37
Discovery implementieren	Implementieren Device Discovery	Realisierung 3, 5	Alle	14	16
Konfigurationsverwaltung implementieren	Implementieren Konfigurationsverwaltung	Realisierung 4-5	Alle	27	26
Login + User Management implementieren	Login + User Management implementieren implementieren	Realisierung 5	Alle	10	13
Dashboard implementieren	Login + User Management implementieren implementieren	Realisierung 5-6	Alle	18	19
Devicemap implementieren	Implementierung Google Map	Realisierung 5	Alle	4	6
Testing	Integrationstests	Realisierung 6, Abschluss 1	Alle	16	15
Refactoring	Struktur- und Codeanpassungen	Realisierung 6, Abschluss 1	Alle	32	36
Bugfixing	Beheben von Softwarefehlern	Realisierung 6	Alle	8	18
Testclient aufsetzen	LwM2M Testclient aufbauen	Realisierung 6	Alle	4	5
Abschluss					
Schlussbericht erstellen	Schlussbericht erstellen und abgeben	Abschluss 1-2	Alle	100	138
Anleitungen erstellen	Benutzer- und Installationsanleitung erstellen	Abschluss 2	Alle	12	9

Tabelle 1.3.: **Arbeitspakete**

1.2.6 Teammeetings

Besprechungen finden dreimal wöchentlich jeweils an den vorgesehenen Arbeitstagen statt. Besprechungen dauern in der Regel 10-15 Minuten. Es wird das weitere Vorgehen, sowie durchgeführte Arbeiten, fällige Arbeiten und auftretende Probleme besprochen. Weiter werden Arbeitspakete verteilt, damit beide Projektmitglieder wissen was zu tun ist.

1.2.7 Meeting mit Betreuern

Die Meetings mit den Betreuern finden jeden Freitag um 14:00 Uhr statt. Die Meetings werden mit den Betreuern Prof. Beat Stettler und Urs Baumann in ihrem Büro durchgeführt. Die Meetings dauern normalerweise zwischen 30-60 Minuten.

1.3 Qualitätsmassnahmen

1.3.1 Versionierung

Wie die Dokumentation wird auch der Sourcecode mit git versioniert und auf GitHub abgelegt. Es wird darauf geachtet, möglichst häufig auf den Stamm zu commiten.

1.3.2 Reviews

Regelmässige Reviews sind in einem iterativen Vorgehen unerlässlich. Die getätigte Arbeit muss ständig abgeglichen und in Frage gestellt werden. Aus Kosten-Nutzen Sicht sind Reviews das effektivste Mittel um die geforderte Qualität zu erreichen.

In diesem Projekt werden drei verschiedene Arten von Reviews durchgeführt. Zum einen sind dies regelmässige Code Reviews, zum anderen sind dies Requirement- und Architekturreviews.

Bei den regelmässigen Code Reviews wird besonders auf die gewählten Namen (Packages, Klassen, Methoden, Variablen), die Verständlichkeit vom Code und Code Smells geachtet.

Bei den Requirement Reviews wird sichergestellt, dass man den Wünschen des Auftraggebers entsprechend entwickelt. Die Frage nach dem „Was“ wird erneut gestellt und somit sichergestellt, dass man beim Projektende nicht ein qualitativ hochwertiges Produkt entwickelt hat, welches aber nicht die Wünsche des Auftraggebers abdeckt.

Beim Architektur Review wird besonders auf die nicht-funktionalen Anforderungen (NFA) geachtet. Diese wirken sich in den meisten Fällen auf die gewählte Architektur aus. Die gewählte Architektur muss mit den NFAs verträglich sein. Wenn zu spät im Projekt bemerkt wird, dass die Architektur geändert werden muss, kann dies sehr aufwändig sein.

1.3.3 Code Metriken

Code Metriken zeigen mögliche Fehler oder Schwachstellen im entwickelten Code auf. Es wird grundsätzlich zwischen statischen- und dynamischen Metrik Tools unterschieden. Bei den dynamischen Metrik Tools wird der Code ausgeführt. Beispiele wären Unit Tests und die dazugehörige Coverage. Bei den statischen Analysetools wird der Code nicht ausgeführt. Ein Beispiel wäre Checkstyle. Dieses Tool überprüft vor allem die Einhaltung von Style Richtlinien.

1.4 Risikomanagement

1.4.1 Risiken

Nr	Titel	Beschreibung	max. Schaden [h]	Eintrittswahrscheinlichkeit	Gewichteter Schaden	Vorbeugung	Verhalten bei Eintitt
R1	Netzwerkstabilität	Netzwerkqualität reicht nicht aus um benötigte Daten zu übertragen	11	10%	1	keine Vorbeugung möglich	Einschränkungen für gewisse Funktionen definieren
R2	Performance	Applikation läuft langsam	22	10%	2	in Architektur berücksichtigen	Fehlersuche, Architektur Review
R3	Backup	Sichern von IoT Devices an Remote Destination funktioniert nicht	22	20%	4	optionales Feature, keine Vorbeugung vorgesehen	Einschränkungen für bestimmte Devicetypen definieren
R4	Implementation von Kommunikationsstandards	Kommunikation zu IoT Devices ist nicht möglich (Protokolle, Standards, etc.)	44	25%	11	Prototyp bis 16.04. klärt	Einschränkungen für Standards definieren
R5	Device Authentifizierung	Authentifizierung am Device (Password, Zertifikat) funktioniert nicht	44	25%	11	Prototyp bis 16.04. klärt	Zertifikatauthentifizierung optional
R6	Device Discovery	Automatisches Discovery von IoT Devices funktioniert nicht	44	40%	18	Prototyp bis 16.04. klärt	nur manuelles Hinzufügen von Devices hinzufügen
R7	Provisioning	Probleme bei der Verteilung von Software oder Konfigurationen an Devices	88	40%	35	Prototyp bis 16.04. klärt	Kompensation durch Überzeit, soweit als möglich implementieren
R8	Einarbeitung in neue Frameworks	Die ausgesuchten Frameworks benötigen mehr Zeit in der Einarbeitung als geplant	88	50%	44	Prototyp bis 16.04. klärt	Kompensation durch Überzeit
Summe			**363**		**126**		

Abbildung 1.2.: Risikoanalyse

1.4.2 Umgang mit Risiken

Die im Dokument aufgeführten Risiken sind in der Zeitplanung nicht speziell vorgesehen. Falls beim Eintreten eines geplanten Risikos ein erhöhter Zeitbedarf entsteht, so muss dies mit hoher Wahrscheinlichkeit mit Mehrarbeit der Teammitglieder kompensiert werden. Falls die nötige Mehrarbeit ausserhalb der Möglichkeiten liegt, so muss in Absprache aller Teammitglieder mit dem Betreuer nach einer anderer Lösung (z.B. Einschränkung von Programmfeatures, etc.) gesucht werden.

2. Zeitauswertung

Die Bachelorarbeit wird mit 12 ECTS Punkten honoriert. Pro ECTS Punkt wird mit einem Aufwand von ca. 30 Stunden gerechnet. Daraus ergibt sich für die Bachelorarbeit ein geplanter Zeitaufwand von 360 Stunden pro Teammitglied, also 720 Stunden Gesamtaufwand. Mit den erstellen Arbeitspaketen wurde ein Aufwand von 732 Stunden geschätzt. Die Endabrechnung zeigt einen ungefähren Zeitaufwand von 842 Stunden. Daraus resultiert ein Gesamtaufwand von 421 Stunden pro Teammitglied. Die geforderten 360 Stunden pro Teammitglied wurden damit deutlich überschritten.

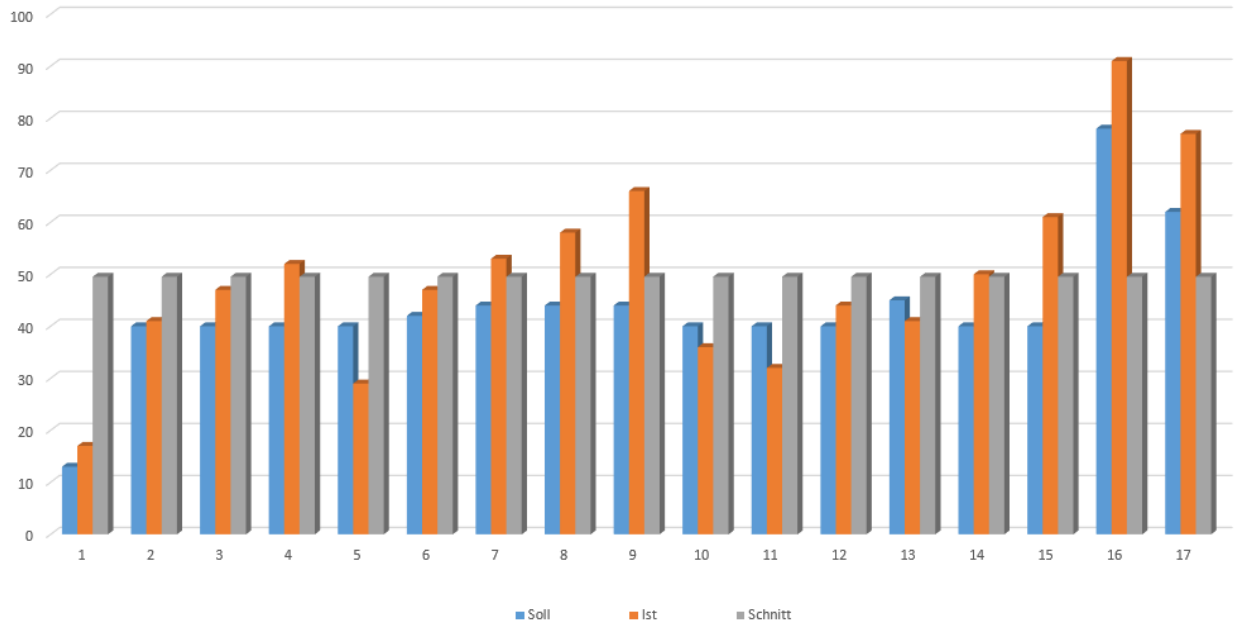


Abbildung 2.1.: Arbeitsstunden wöchentlich

Die Zeitverteilung pro Kalenderwoche ist unterschiedlich. Die Aufwände im Programmierbereich waren oft deutlich höher als angenommen. In den Wochen 6-9 (Erstellung Prototyp) und 14-17 (Fertigstellung Software, Abschlussbericht) sind die Differenzen am höchsten. Dies ist einerseits mit teils geringen Vorkenntnissen, aber andererseits auch mit hohem Implementationsaufwand zu rechtfertigen.

Durch die teils geringen Vorkenntnisse musste verhältnismässig viel Aufwand in Analyse und Prototyping gesteckt werden. $\frac{1}{3}$ Aufwand für die Realisierung scheint daher eher gering.

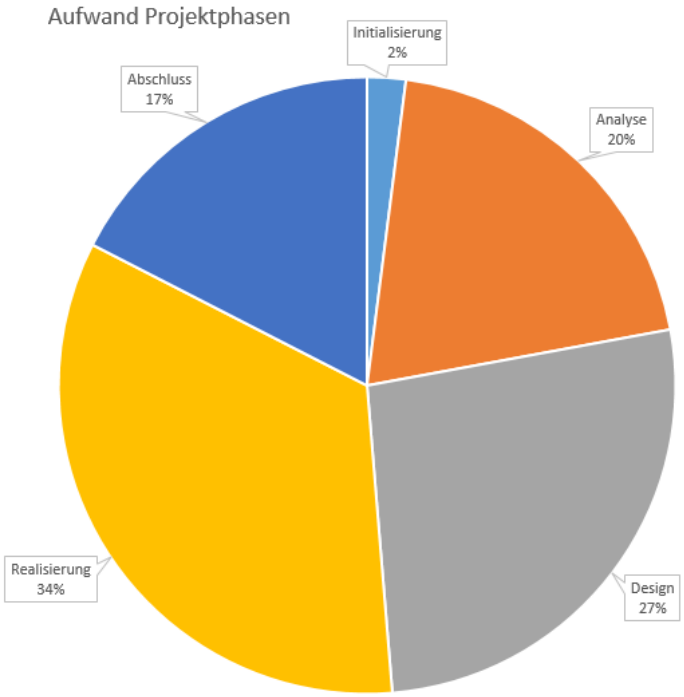


Abbildung 2.2.: Aufwände pro Projektphase

3. Installationsanleitung

Dies ist die Installationsanleitung für den Smartmanager. Der Smartmanager unterstützt eine Installation unter Linux sowie auch unter Windows. Ein Deployment in einer PaaS wird momentan noch nicht unterstützt.

3.1 Benötigte Software

Für die Installation der Applikation werden folgende Softwares benötigt:

- Java SE Runtime Environment Version 8
- Tomcat Version 8
- Maven
- git

3.2 Zertifikat Variante 1: PKI

Keystore erstellen Als ersten Schritt erstellt man mit dem Keytool einen Keystore. Diese benötigt einen Alias und einen Pfad, indem er abgelegt wird. Dieser Pfad wird später für die Applikationskonfiguration benötigt.

Nun wird man nach einem Passwort gefragt. Dieses wird auch für die spätere Applikationskonfiguration benötigt.

```
[andreas@archbook ~]$ keytool -genkey -alias youralias -keyalg RSA -keystore /tmp/.keystore
Keystore-Kennwort eingeben:
Neues Kennwort erneut eingeben:
Wie lautet Ihr Vor- und Nachname?
  [Unknown]:
```

Danach gibt man alle wichtig Informationen zum Zertifikat an. Diese Informationen müssen alle stimmen, da diese an die ”Certificate Authority” weitergegeben werden. Dies kann am Schluss mit ”Ja” bestätigt werden.

```
Wie lautet Ihr Vor- und Nachname?
  [Unknown]:  www.mydomain.ch
Wie lautet der Name Ihrer organisatorischen Einheit?
  [Unknown]:  Muster Organisation
Wie lautet der Name Ihrer Organisation?
  [Unknown]:  Muster Organisation
Wie lautet der Name Ihrer Stadt oder Gemeinde?
  [Unknown]:  Rapperswil
Wie lautet der Name Ihres Bundeslands?
  [Unknown]:  St. Gallen
Wie lautet der Ländercode (zwei Buchstaben) für diese Einheit?
  [Unknown]:  CH
Ist CN=www.mydomain.ch, OU=Muster Organisation, O=Muster Organisation, L=Rapperswil, ST=St. G
allen, C=CH richtig?
  [Nein]:
```

Zum Schluss wird nun noch ein Passwort verlangt, welches für dieses Zertifikat gilt. Dies muss das gleiche Passwort wie für den Store sein, da Tomcat sonst nicht auf das Zertifikat zugreifen kann.

CSR erstellen Als nächsten Schritt wird der Certificate Signing Request erstellt. Dazu gibt man folgenden Befehl ein:

```
[andreas@archbook ~]$ $JAVA_HOME/bin/keytool -certreq -keyalg RSA -alias youralias -file you
rcertificatname.csr -keystore /tmp/.keystore
Keystore-Kennwort eingeben:
Schlüsselkennwort für <youralias> eingeben
[andreas@archbook ~]$
```

Dieses Zertifikat kann man nun bei einer ”Certificate Authority” bestätigen lassen und erhält danach die benötigten Zertifikate.

Zertifikate importieren Die erhaltenen Zertifikate werden nun importiert. Zuerst wird das Root-Zertifikat importiert.

```
[andreas@archbook ~]$ $JAVA_HOME/bin/keytool -import -alias root -keystore /tmp/.
keystore -trustcacerts -file smartmanager.cer
Keystore-Kennwort eingeben:
```

Danach import man das erhaltene Zertifikat und gibt die Passwörter ein. Durch das ist der Keystore vorbereitet.

```
[andreas@archbook ~]$ $JAVA_HOME/bin/keytool -import -alias youralias -keystore
/tmp/.keystore -file mycert.cer
Keystore-Kennwort eingeben:
```

3.3 Zertifikat Variante 2: Self-Signed

Anstelle einer PKI kann man auch einfach ein Self-Signed Zertifikat erstellen. Durch diesen Befehl ist dies Möglich:

```
[andreas@archbook ~]$ keytool -genkey -keyalg RSA -alias selfsigned -keystore .k
eystore
Keystore-Kennwort eingeben:
Wie lautet Ihr Vor- und Nachname?
  [Unknown]:  www.mydomain.ch
Wie lautet der Name Ihrer organisatorischen Einheit?
  [Unknown]:  Muster Organisation
Wie lautet der Name Ihrer Organisation?
  [Unknown]:  Muster Organisation
Wie lautet der Name Ihrer Stadt oder Gemeinde?
  [Unknown]:  Rapperswil
Wie lautet der Name Ihres Bundeslands?
  [Unknown]:  St. Gallen
Wie lautet der Ländercode (zwei Buchstaben) für diese Einheit?
  [Unknown]:  CH
Ist CN=www.mydomain.ch, OU=Muster Organisation, O=Muster Organisation, L=Rappers
wil, ST=St. Gallen, C=CH richtig?
  [Nein]:  Ja

Schlüsselkennwort für <selfsigned> eingeben
        (RETURN, wenn identisch mit Keystore-Kennwort):
Neues Kennwort erneut eingeben:
[andreas@archbook ~]$
```

Dies reicht aus, damit Tomcat auf das Zertifikat zugreifen kann.

3.4 Smartmanager

Smartmanager herunterladen Das Projekt befindet sich auf folgendem URL: https://github.com/BA-Smartmanager/smartmanager

Durch ”git clone https://github.com/BA-Smartmanager/smartmanager” kann nun das Repo heruntergeladen werden.

Smartmanager konfigurieren Im erhaltenen Ordner smartmanager befindet sich nun der gesamte Source-Code. Um Einstellungen vorzunehmen, öffnet man die Datei ”smartmanager/WebContent/WEB-INF/smartmanager-configuration.xml”

In dieser Datei kann die LwM2M-Server Adresse und die MongoDB-URL anpassen. Weitere Einstellungen sind nicht nötig.

```
<property name="address" value="127.0.0.1" />
<property name="port" value="5683"></property>
<mongo:db-factory id="mongoDbFactory" client-uri="mongodb://localhost/smartmanager" />
```

Builden Um die Software zu builden, gibt im Ordner ”smartmanager” ”mvn clean install” ein. Diese generiert ein Ordner target und darin befindet sich eine .war-Datei. Diese wird später in den Tomcat-Ordner kopiert. Am besten benennt man diese .war-Datei in ”smartmanager.war” um, da dieser Name die URL direkt beeinflusst.

3.4.1 Tomcat

Konfigurieren Im Tomcat Installationsverzeichnis befindet sich im Ordner ”conf” die Datei server.xml. In dieser sucht man nach den Connector Tags und fügt folgende hinzu.

```
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="
    8443" />
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
  maxThreads="150" SSLEnabled="true" scheme="https" secure="true"
  clientAuth="false" sslProtocol="TLS" keystoreFile="</yout/keystore/path/.keystore>"
  keystorePass="<Your Password>">
</Connector>
```

Dabei ist wichtig, dass der Pfad zum Keystore, sowie das dazugehörige Passwort richtig angegeben werden. Sollten noch weitere Connector Einträge wie folgender vorhanden sein (Nur die mit Port 8080):

```
<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="
    8443" />
```

können diese gelöscht werden.

War-Datei in Tomcat einfügen Die vorher erstellte .war-Datei wird nun kopiert und in den ”/webapps/”-Ordner im Tomcat Installationsverzeichnis eingefügt.

Tomcat starten Zum Schluss wird nun Tomcat gestartet. Dies kann man durch die im Tomcat Installationsverzeichnis vorhandenen Skripts erledigen. Zu finden sind diese im ”bin”-Ordner

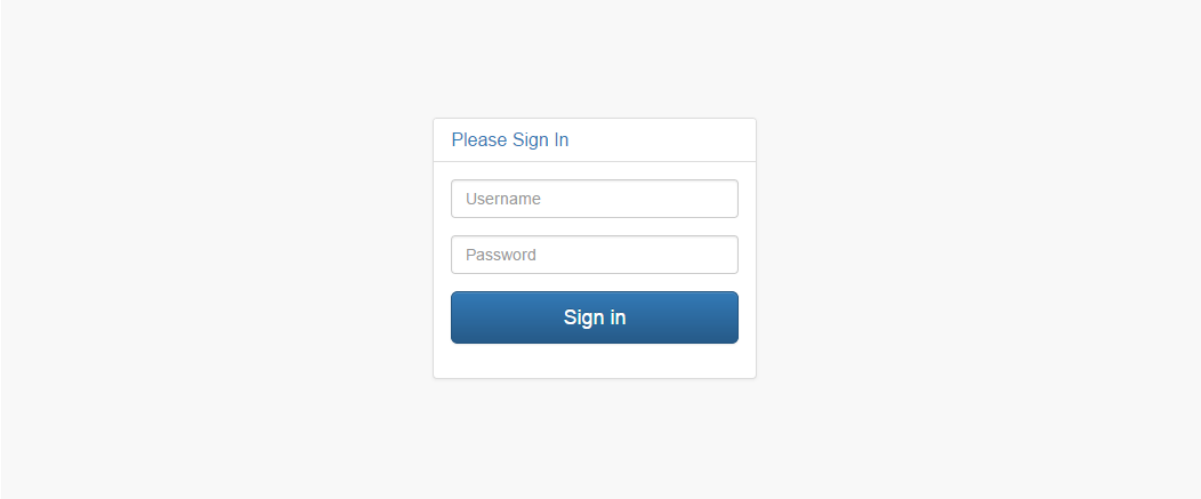
4. Benutzeranleitung

Diese Benutzeranleitung dient der leichten Erlernbarkeit der Software. Für Benutzer ohne Kenntnisse des Lightweight M2M (LwM2M) Protokolls empfiehlt sich die Spezifikationen des Ressourcenmodells (http://openmobilealliance.org/wp/OMNA/LwM2M/LwM2MRegistry.html) als Nachschlagewerk. Für das Grundlegende Verständnis von LwM2M kann das Kapitel 3.4 konsultiert werden. Die Benutzeranleitung ist inhaltlich nach den implementierten Use Cases (siehe Kapitel 5.2) aufgebaut.

Voraussetzung für die Nutzung ist eine erfolgreiche Installation des Smartmanagers (siehe Kapitel 3).

4.1 Benutzerlogin

- Der Smartmanager kann unter folgender URL benutzt werden: https://server.url:8443/smartmanager/
- Auf der Startseite erscheint ein Loginfenster. Bevor die Applikation benutzt werden kann, muss ein gültiges Login getätigt werden.

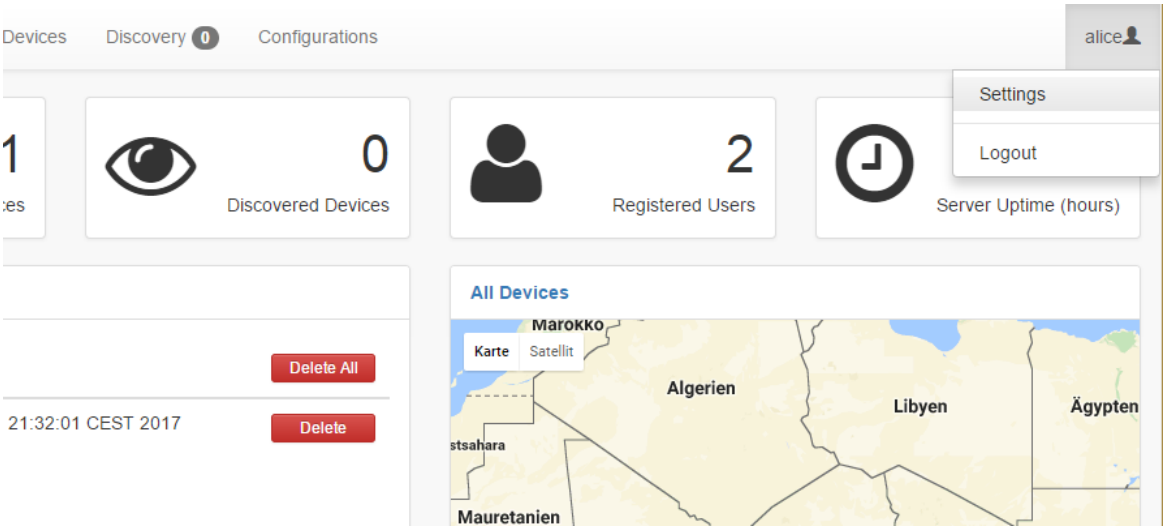


- Sofern kein Login vorliegt, muss der Administrator konsultiert werden, welcher Benutzername und Passwort aushändigen kan
- Das Standard-Login für den Administrator ist Username: admin, Password: admin

4.2 Benutzerverwaltung

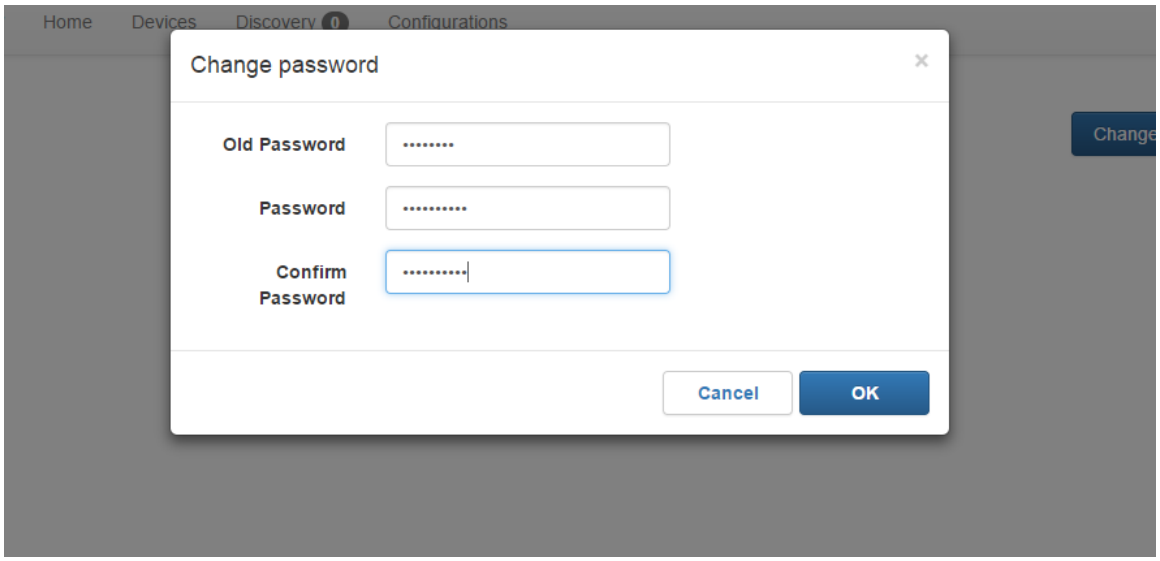
Die Benutzerverwaltung unterscheidet sich für Benutzer und den Administrator. Benutzer können zur Zeit lediglich ihr Kennwort zurücksetzen. Der Administrator kann sein eigenes Kennwort zurücksetzen, neue Benutzer anlegen und Benutzer löschen.

Die Benutzerverwaltung kann mit einem Klick auf den eigenen Benutzernamen am oberen rechten Bildschirmrand geöffnet werden.

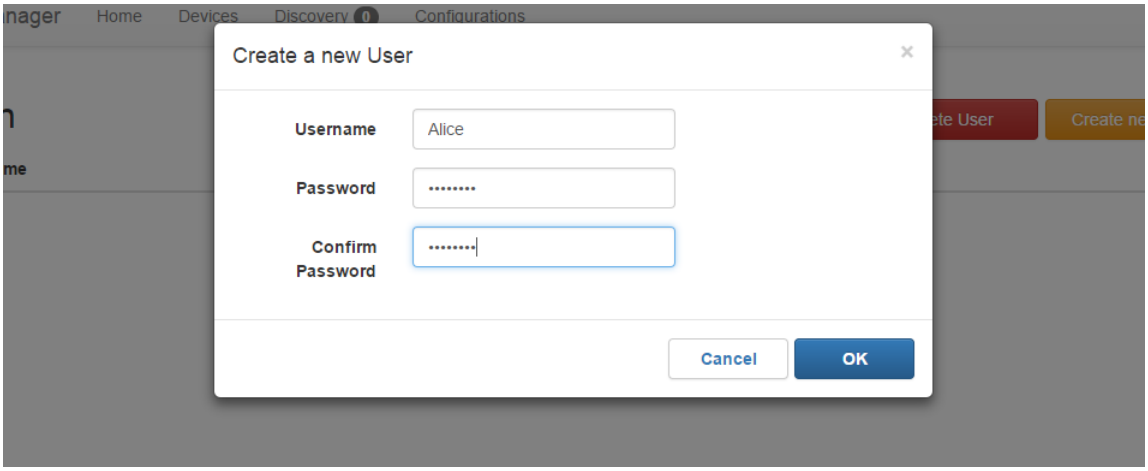


Eigenes Kennwort ändern

- Mit einem Klick auf ”Change password” öffnet sich ein neues Fenster
- Das bestehende Kennwort muss einmal- und das neue Kennwort zweimal eingegeben werden

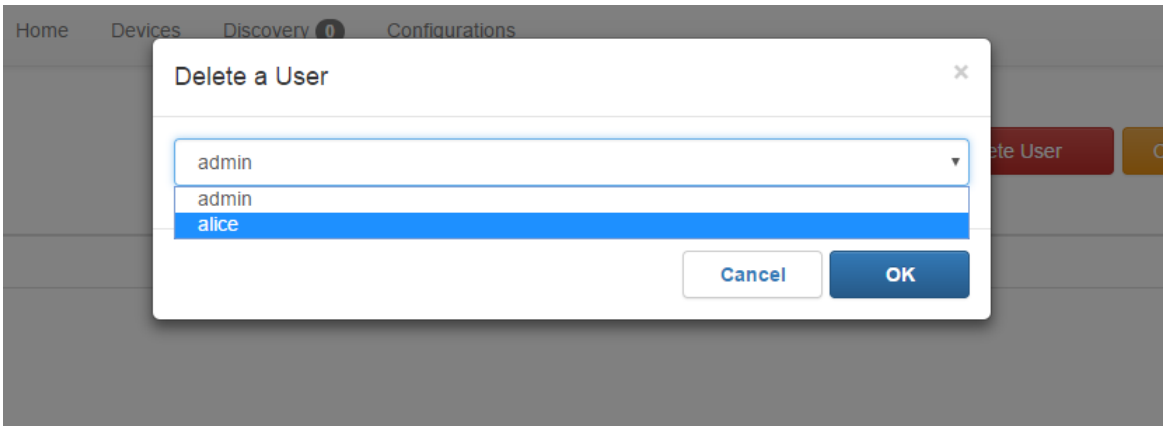


Neuen Benutzer erstellen Neue Benutzer können nur als Administrator erstellt werden. Sobald man ”Create new User” klickt, öffnet sich ein neues Fenster.



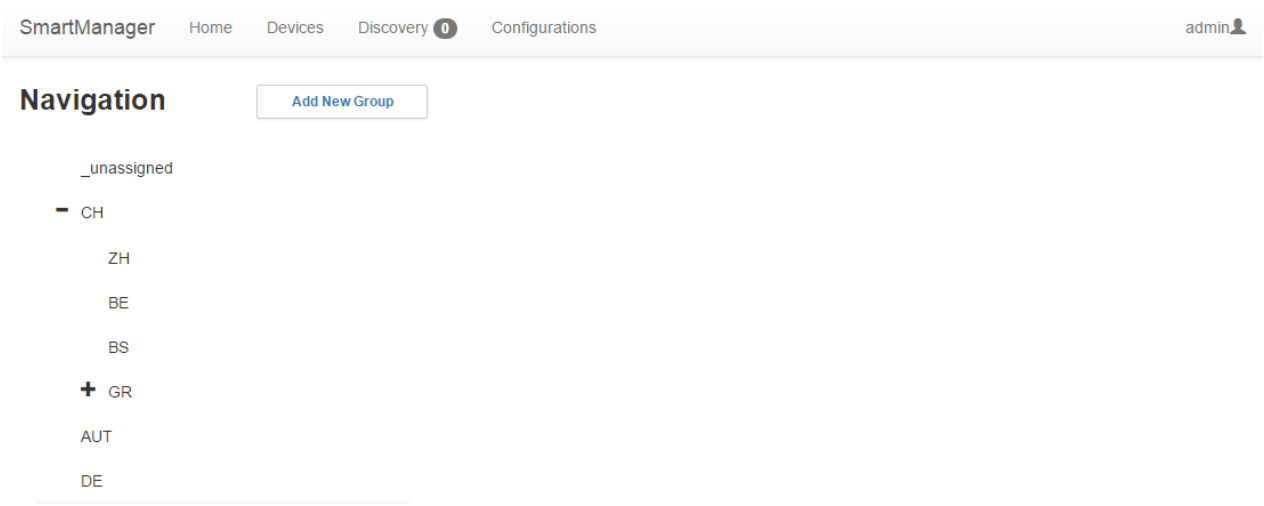
Der Benutzername muss dabei aus mindestens 4 Zeichen bestehen.

Benutzer löschen Benutzer können unter ”Delete User” gelöscht werden. Im Drop-Down Menü kann der entsprechende Benutzer gelöscht werden. Es ist nicht möglich, den Admin zu löschen.

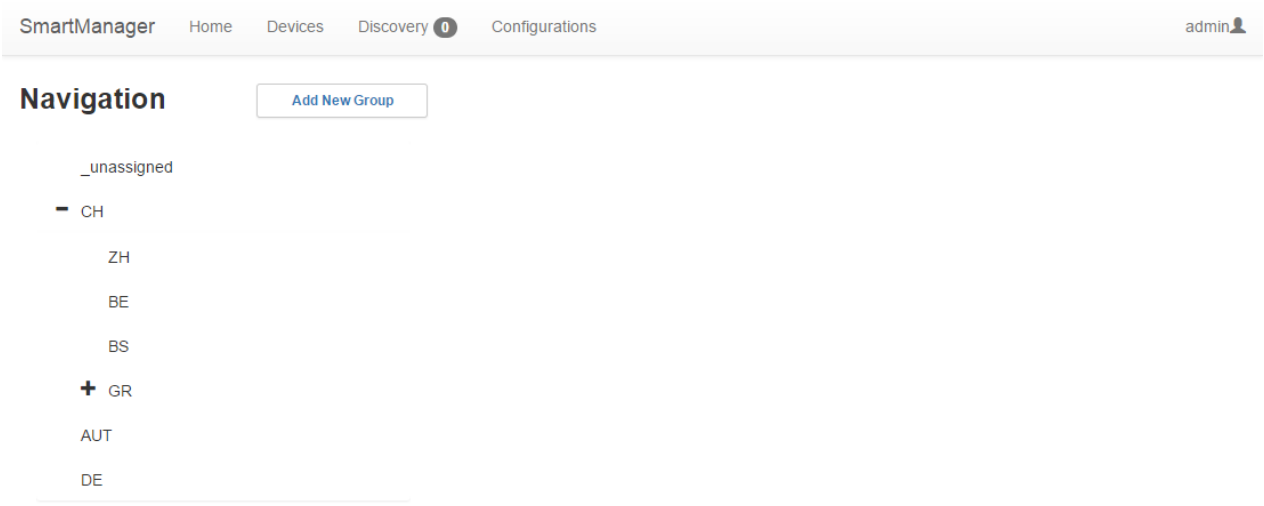


4.3 Gruppenverwaltung

Auf der ”Devices”-Seite können Gruppen verwaltet werden. Auf der linken Seite befindet sich der Navigationsbaum.

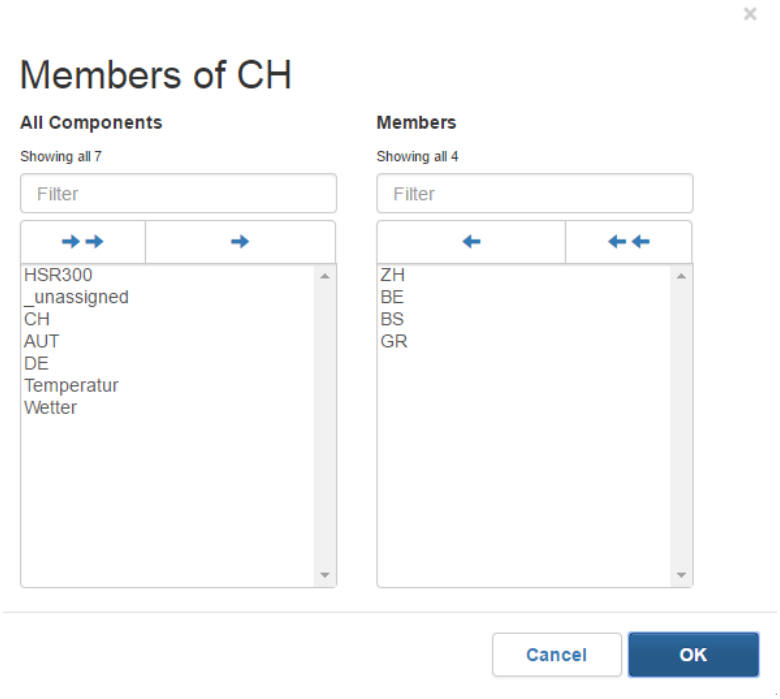


Mit einem Klick auf ”Add New Group” wird eine neue Gruppe auf höchster Ebene angelegt. Einzig der Name der neuen Gruppe muss einzigartig sein, ansonsten existieren keine Einschränkungen. Sobald man in der Navigation eine Gruppe anklickt, erscheint die Gruppenansicht im Hauptfenster.



Neue Kindsgruppe hinzufügen Unter ”Add New Child Group” kann der Gruppe eine neue Kindsgruppe hinzugefügt werden. Einzig der Name darf nicht mehrfach vorkommen, ansonsten existieren keine Einschränkungen.

Gruppenmitglieder verwalten Unter ”Group Members” können die direkten Kindskomponenten (Devices und Gruppen) hinzugefügt- oder entfernt werden.



Auf der linken Seite sind sämtliche Komponenten enthalten, welche noch nicht Mitglieder der Gruppe sind, auf der rechten Seite sind alle Mitglieder. Bei zuvielen Einträgen kann unter ”Filter” nach dem Namen gesucht werden.

Folgende Regeln gelten für das Setzen von Gruppenmitgliedern:

- Devices können in mehreren Gruppen Mitglied sein
- Gruppen können nur in einer Gruppe Mitglied sein
- Wird eine Gruppe als Mitglied gesetzt, so wird sie bei einer anderen Gruppe als Mitglied entfernt
- Ein direkter oder indirekter Vor- oder Nachfahre der Gruppe kann nicht als Mitglied hinzugefügt werden
- die Gruppe ”_unassigned” kann nicht als Mitglied hinzugefügt werden

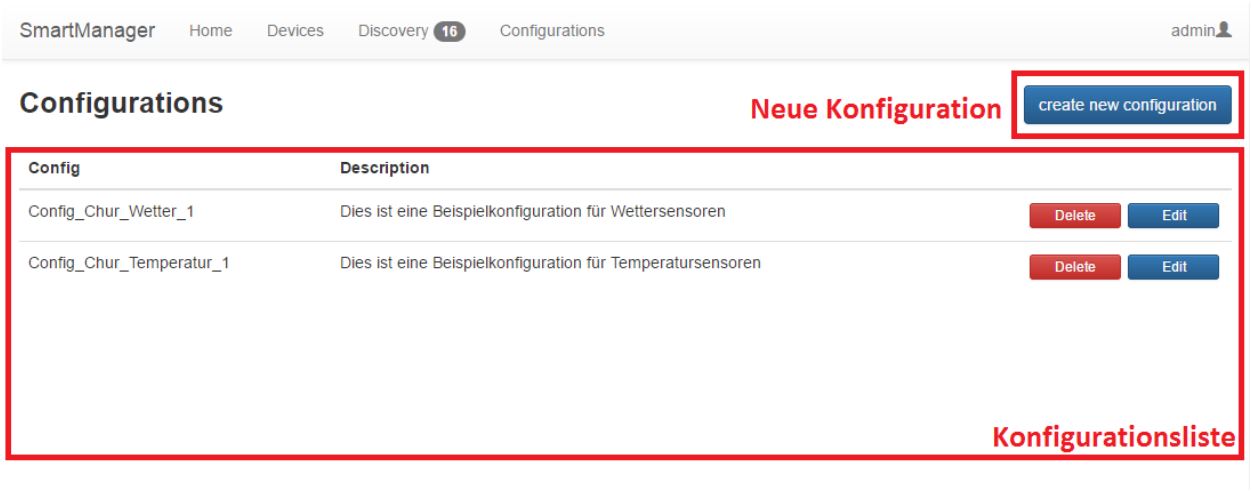
Gruppenmitgliedschaften verwalten Unter ”Group Memberships” kann die direkte Elterngruppe hinzugefügt- oder entfernt werden.

Folgende Regeln gelten für das Setzen von Gruppenmitgliedschaften:

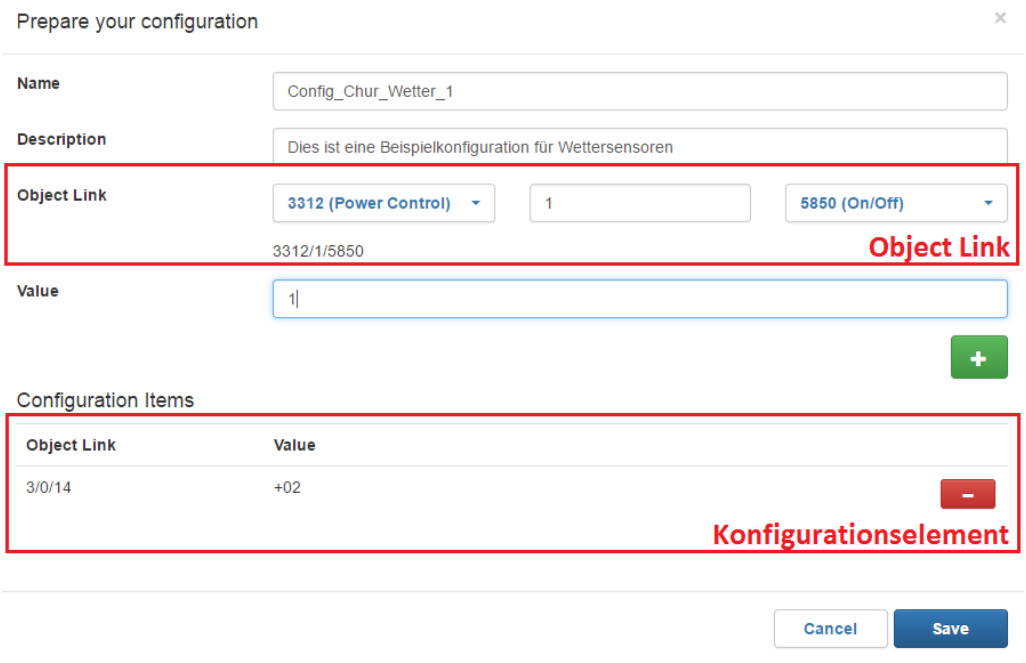
- Eine Gruppe kann genau eine Elterngruppe haben
- Die Gruppe ”_unassigned” kann nicht als Elterngruppe gesetzt werden
- Direkte- oder indirekte Nachfahren können nicht als Elterngruppe gesetzt werden

4.4 Konfigurationen verwalten

Auf der ”Configurations”-Seite kann man eigene Konfigurationen erstellen, bearbeiten und löschen.



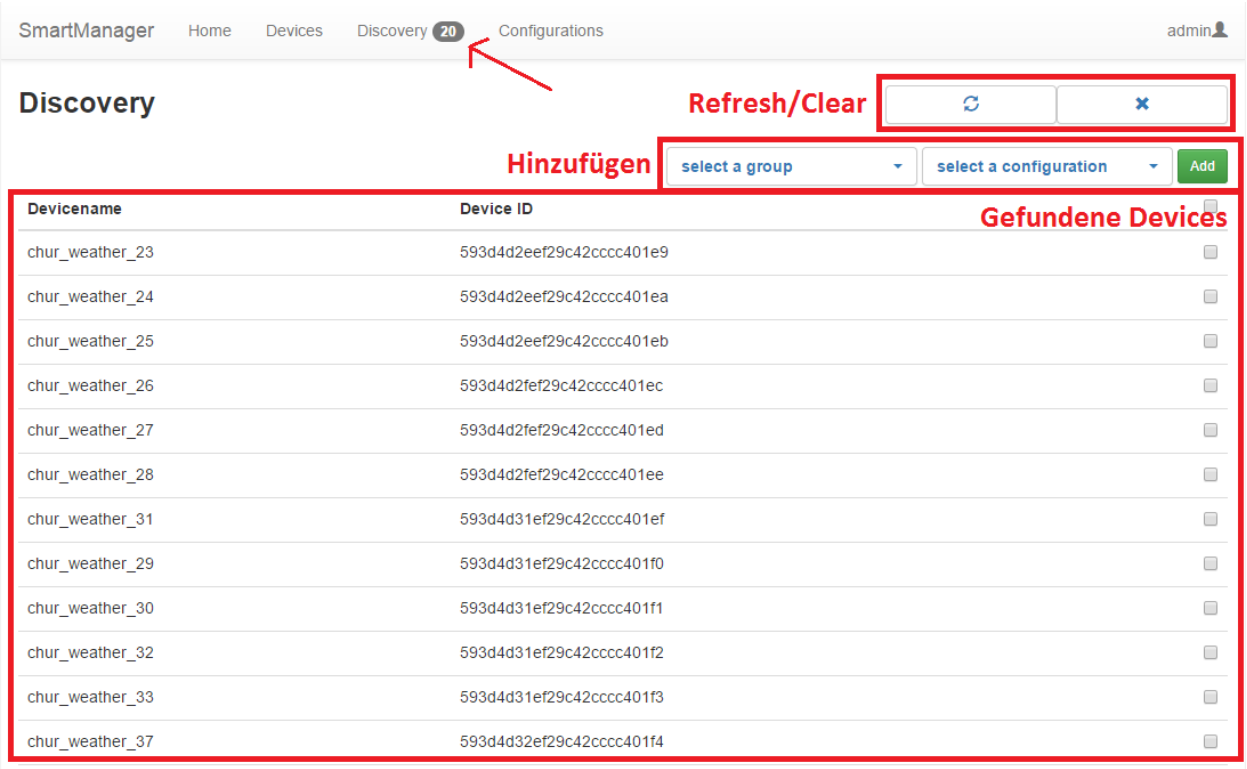
Konfiguration erstellen Unter ”create new configuration” erstellt man eine neue Konfiguration.



- Im Namensfeld kann ein beliebiger Name der Konfiguration eingegeben werden
- Im Descriptionfeld kann eine beliebige Beschreibung eingegeben werden
- Die Object Link Zeile wählt die zu beschreibende Ressource aus. Es werden dabei nur Ressourcen angezeigt, welche die Operation ”Write” unterstützen
- Im Value Feld wird der Wert für die Ressource eingetragen. Gegebenenfalls sollte die Spezifikation des Ressourcenmodells konsultiert werden
- Unten sind die Konfigurationselemente tabellarisch ersichtlich

4.5 Device Discovery

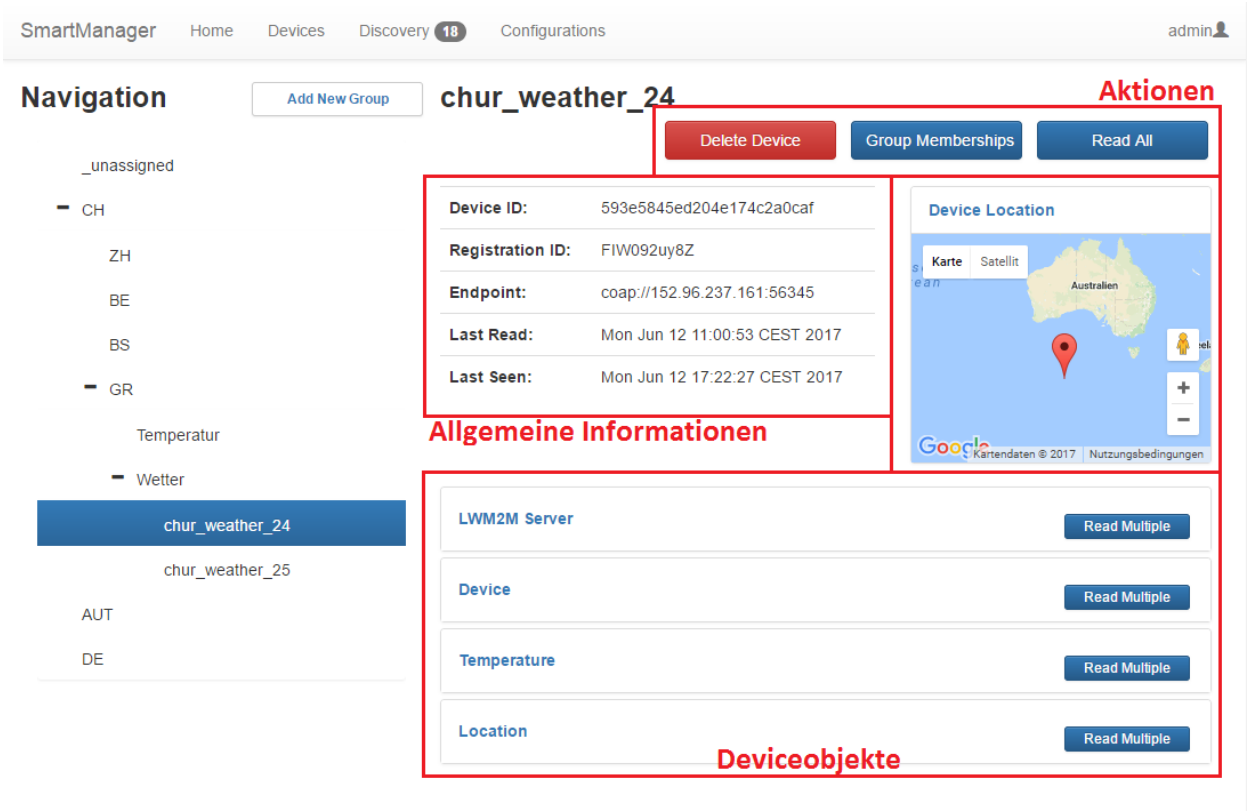
Für das Discovery müssen sich Devices am Server registrieren. Auf der ”Discovery”-Seite sind registrierte- und noch nicht hinzugefügte Devices tabellarisch aufgelistet.



Sobald man ein- oder mehrere Devices selektiert hat, kann man mit dem ”Add”-Button das Device hinzufügen. Wenn man möchte, kann man eine Gruppenmitgliedschaft setzen und eine initiale Konfiguration schreiben. Wird keine Gruppe gewählt, so werden neue Devices automatisch der Gruppe ”_unassigned” hinzugefügt.

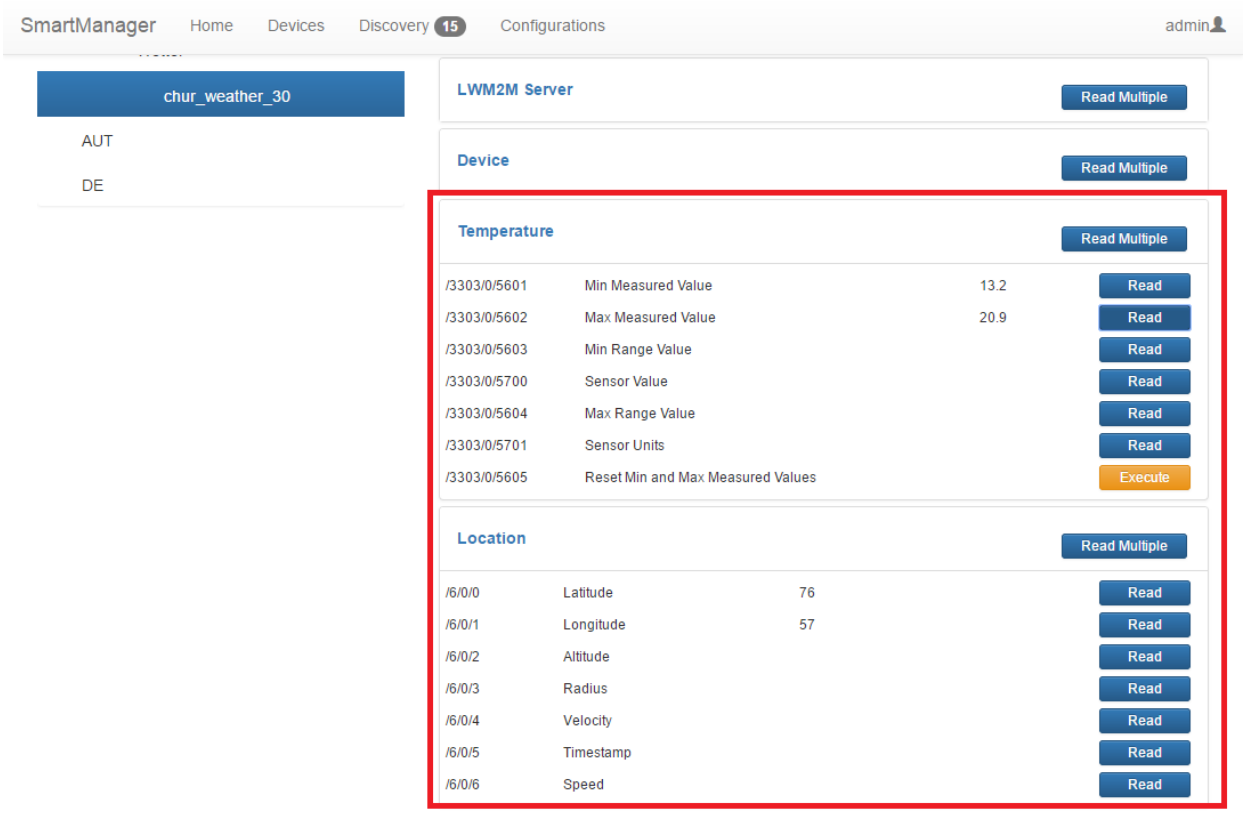
4.6 Device verwalten

Sobald ein Device hinzugefügt wurde, erscheint es im Navigationsbaum. Wählt man es an, öffnet sich die Geräteübersicht.



Für Devices können Aktionen wie Gruppenmitgliedschaften, Löschen und Lesen ausgeführt werden. Im Bereich der allgemeinen Informationen ist Nützliches auf einen Blick zusammengefasst.

Deviceobjekte Im Bereich der Deviceobjekte sind die Deviceattribute verborgen. Klickt man auf ein Objekt, so öffnet sich eine Tabelle mit sämtlichen verfügbaren Ressourcen.



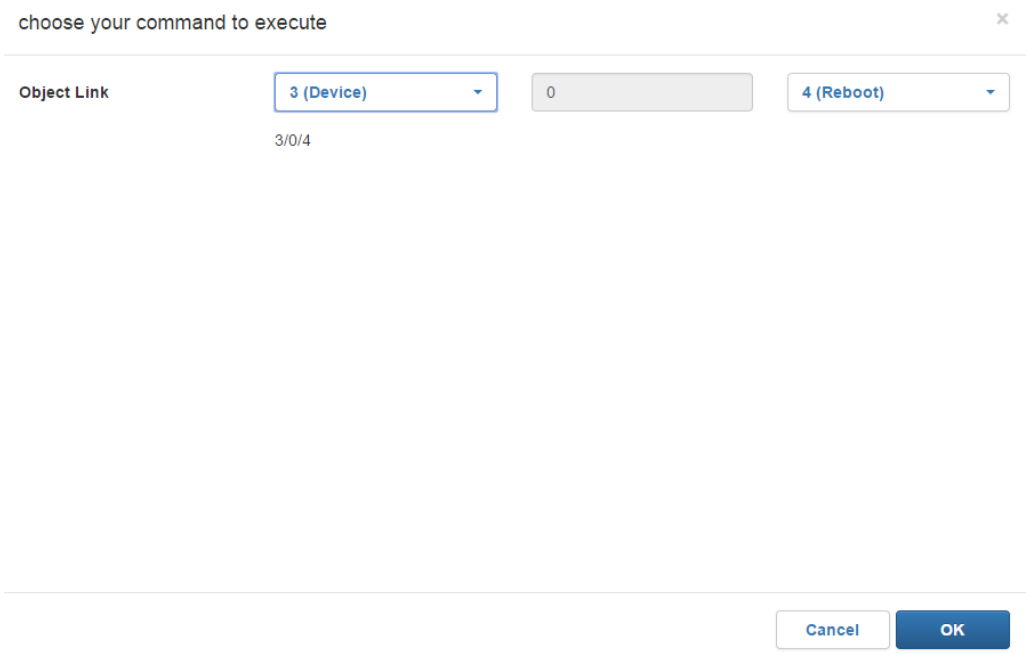
Falls man Probleme hat, die einzelnen Zeilen zu interpretieren, sollte die Spezifikation des Ressourcenmodells konsultiert werden. Auf der rechten Seite sind Buttons mit den verfügbaren Aktionen vorhanden.

4.7 Operationen auf Devices ausführen

LwM2M unterstützt die vier Operationen ”Read”, ”Write”, ”Execute” und ”Observe”. Die ”Oberserve”-Funktion ist noch nicht verfügbar. Diese Operationen sind jeweils für eine Ressource definiert und können einzeln ausgeführt werden.

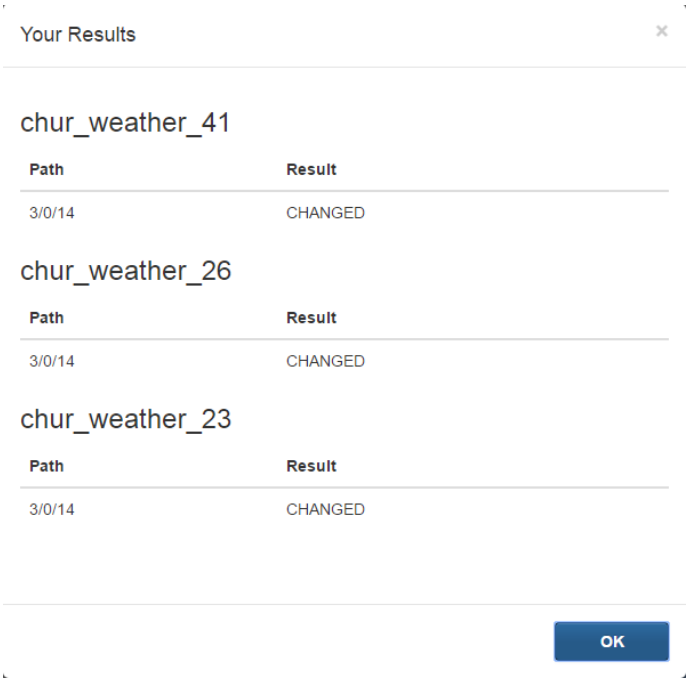
In dieser Applikation ist es ebenfalls möglich, Operationen auf Gruppenebene auszuführen, damit effizient eine grosse Anzahl Devices administriert werden können.

Execute Wählt man in der Gruppenansicht ”Execute”, so kann man für sämtlichen Nachfahren der Gruppe eine Aktion wie Beispielsweise einen Reboot durchführen.



Write Configuration Bestehende Konfigurationen können auf eine ganze Gruppe geschrieben werden. Unter ”Write Configuration” kann im Drop-Down die gewünschte Konfiguration ausgewählt werden.

Resultate Nachdem Schreib- oder Execute-Operationen auf Devices ausgeführt wurden, werden serverseitig Resultate generiert. Somit erhält man einen Überblick über den Erfolg der getätigten Operationen.



5. Glossar

LwM2M	Lightweight Machine-2-Machine Protokoll
Industrie 4.0	Vierte industrielle Revolution. Vernetzung von Maschinen, Sensoren und Menschen.
SNMP	Simple Network Management Protocol
DHCPv6	Dynamic Host Configuration Protocol in der IPv6 Variante
ZigBee IP	Auf IPv6 basierendes IP-Protokoll für drahtlose Mesh-Netzwerke.
6LoWPAN	IPv6 over Low power Wireless Personal Area Network
URN	Uniform Resource Name
RFID	Radio-frequency identification
Certificate authority	Zertifizierungsstelle, welche digitale Zertifikate herausgibt
CSR	Certificate Signing Request - digitaler Antrag um ein Zertifikat zu erhalten
PKI	Public-Key-Infrastruktur
Zero Touch Provisioning	Automatische Konfiguration von Geräten, ohne menschlichen Einfluss
Duck-Typing	Programmierkonzept, bei dem die Methoden oder Attribute den Typ der Klasse bestimmt. Ohne Duck-Typing wird das Objekt über die Klasse definiert.
Leshan	LwM2M-Server und -Client Implementation in Java. Entwickelt wird diese Library von der Eclipse Community.
XSS	Cross-Site-Scripting
CSRF	Cross-Site-Request-Forgery
DoS	Denial of Service. Dies ist die Nichtverfügbarkeit eines Dienstes.
EAP	Extensible Authentication Protocol
ACL	Access Control List
TLS	Transport Layer Security
DTLS	Datagram Transport Layer Security
SSL	Secure Sockets Layer

6. Abbildungsverzeichnis

7. Quellenverzeichnis

[1] Miorandi, Daniele ; Sicari, Sabrina ; Pellegrini, Francesco D. ; Chlamtac, Imrich: *Internet of things: Vision, applications and research challenges.* http://www.sciencedirect.com/science/article/pii/S1570870512000674, 2012

[2] Radovici, Alexandru: *Introduction to the Internet of Things Summer School.* https://www.youtube.com/watch?v=G4-CtKkrOmc, 2015

[3] Rose, Karen ; Eldridge, Scott ; Chapin, Lyman: *THE INTERNET OF THINGS: AN OVERVIEW.* https://www.internetsociety.org/sites/default/files/ISOC-IoT-Overview-20151221-en.pdf, 2015

[4] Corporation, International D.: *Connecting the IoT: The Road to Success.* http://www.idc.com/infographics/IoT, 2015

[5] Microsoft: *Overview of device management with IoT Hub.* https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-management-overview, 2017

[6] Guinard, Dominique ; Trifa, Vlad ; Karnouskos, Stamatis ; Spiess, Patrik ; Savio, Domnic: *Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services.* http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5416674, 2010

[7] Heer, Tobias ; Garcia-Morchon, Oscar ; Hummen, René ; Keoh, Sye L. ; Kumar, Sandeep S. ; KlausWehrle: *Security Challenges in the IP-based Internet of Things.* https://link.springer.com/article/10.1007/s11277-011-0385-5, 2011

[8] Weber, John: *Fundamentals of IoT device management.* http://iotdesign.embedded-computing.com/articles/fundamentals-of-iot-device-management/, 2016

[9] Pardey, John: *Netzwerkmanagement.* http://www.it-administrator.de/themen/netzwerkmanagement/-grundlagen/111034.html, 2012

[10] Kumbhar, Sheetal: *Getting ready for IoT monitoring.* http://www.iot-now.com/2016/11/17/55128-getting-ready-for-iot-monitoring, 2016

[11] Unknown: *Computerized Maintenance Management System.* https://de.wikipedia.org/wiki/Computerized_Maintenance_Management_System, 2015

[12] micrium.com: *IoT Networks.* https://www.micrium.com/iot/devices/, 2017

[13] micrium.com: *Devices & Networks.* https://www.micrium.com/wp-content/uploads/2014/03/internet-of-things.png, 2017

[14] Vasseur, JP: *The Internet of Things: Architecture and Protocols.* https://www.youtube.com/watch?v=co2MLqkJVXs, 2014

[15] micrium.com: *Devices & Networks.* https://www.micrium.com/wp-content/uploads/2014/03/wireless-sensor-network.png, 2017

[16] Obermaier, Dominik: *Internet der Dinge: Leichtgewichtiges Messaging.* https://www.informatik-aktuell.de/betrieb/netzwerke/internet-der-dinge-protokolle-verfahren-und-integration-von-mqtt.html, 2014

[17] Obermaier, Dominik: *IoT-Protokolldschungel – Ein Wegweiser.* https://www.informatik-aktuell.de/betrieb/netzwerke/iot-protokolldschungel-ein-wegweiser.html, 2015

[18] servicedesignpatterns.com: *Request/Response.* http://www.servicedesignpatterns.com/Images/-

RequestResponse.jpg, 2011

[19] IETF: *The Constrained Application Protocol (CoAP).* https://tools.ietf.org/html/rfc7252, 2014

[20] Shelby, Zach: *Constrained Application Protocol (CoAP) Tutorial.* https://www.youtube.com/watch?v=4bSr5x5gKvA, 2014

[21] Hohpe, Gregor ; Woolf, Bobby: *Enterprise Integration Patterns.* http://www.enterpriseintegrationpatterns.com/img/PublishSubscribeSolution.gif, 2011

[22] Schneider, Stan: *Understanding The Protocols Behind The Internet Of Things.* http://electronicdesign.com/iot/understanding-protocols-behind-internet-things, 2013

[23] Schneider, Stan: *Understanding The IoT Protocols.* https://de.slideshare.net/RealTimeInnovations/iot-34485340, 2014

[24] Vermillard, Julien: *Bootstrapping device security with Lightweight M2M.* https://medium.com/@vrmvrm/device-key-distribution-with-lightweight-m2m-36cdc12e5711, 2015

[25] Hung, Le K.: *Overview OMA Lightweight M2M Protocol.* https://lehunguit.wordpress.com/2015/08/03/overview-lightweight-machine-to-machine/, 2015

[26] Orebaugh, Angela: *Internet der Dinge: Was zu tun ist, um IoT-Security Realität werden zu lassen.* http://www.searchsecurity.de/meinung/Internet-der-Dinge-Was-zu-tun-ist-um-IoT-Security-Realitaet-werden-zu-lassen, 2015s

[27] Pender-Bey, Georgie: *The Parkerian Hexad.* http://cs.lewisu.edu/mathcs/msisprojects/papers/-georgiependerbey.pdf, 2013

[28] unknown: *Parkerian Hexad.* https://en.wikipedia.org/wiki/Parkerian_Hexad, 2016

[29] Developers, ARTIK C.: *DTLS CoAP.* https://developer.artik.cloud/images/docs/artik-cloud/connect-the-data/coap-dtls-diagram.png, unknown

[30] Flavio Reinhard, Ken S.: *QuiXilver Mobile Client.* https://eprints.hsr.ch/371/1/BA_ohne_Anhang.pdf, 2014

[31] unknown: *NoSQL.* https://de.wikipedia.org/wiki/NoSQL, 2017

[32] Gus: *Raspberry Pi Temperature Sensor.* https://pimylifeup.com/wp-content/uploads/2016/03/Raspberry-Pi-Temperature-Sensor-Diagram-v2.png, 2016