

iHome: Smart Home Management as a Service in the Cloud

George Myrizakis



School of Electrical and Computer Engineering
Technical University of Crete
Greece

Committee:

Petrakis G.M. Euripedes, Professor (supervisor)
Samoladas Vasilis, Associate Professor
Sotiriadis Stelios, Assist. Professor, Birkberk, UL, UK

January 2019

Contents

1	Introduction	6
1.1	Problem Definition	6
1.2	Contribution	7
2	Related Work and Technologies - Tools	9
2.1	Smart Home systems	9
2.1.1	Amazon Alexa	9
2.1.2	Apple HomeKit	9
2.1.3	Google Home	10
2.1.4	Home Assistant	10
2.1.5	OpenHab	10
2.1.6	Pytotation	11
2.2	Openstack	11
2.3	FIWARE	12
2.4	Fiware Services	13
2.4.1	Publish/Subscribe Context Broker - Orion Context Broker	13
2.4.2	Identity Management (IdM) - Keyrock IdM	14
2.5	Service Oriented Architecture (SOA)	14
2.6	Web Services	14
2.7	REST	15
2.7.1	JSON	16
2.8	NoSQL Database - MongoDB	16
2.9	PHP	17
2.10	Java EE	17
2.11	Spring Boot	17
3	Requirement Analysis and System Design	18
3.1	Requirement Analysis	18
3.1.1	User Groups	18
3.1.2	Functional Requirements	18
3.1.3	Non-Functional Requirements	19
3.2	System Design - UML Diagrams	20
3.2.1	Use Case Diagrams	20
3.2.2	Activity Diagrams	24
3.2.3	System's flowchart	38
3.2.4	Class Diagram	40
3.2.5	Architecture Diagram	42

4	System Implementation	46
4.1	FIWARE Platform	46
4.1.1	Keyrock Identity Management	47
4.1.2	Orion Context Broker	51
4.1.3	Web Application(UI)	54
4.1.4	Application Logic	55
4.1.5	User Authorization Service	56
4.1.6	Complex Event Processing	57
4.1.7	History Database	61
4.1.8	Connectivity Service	62
4.1.9	Front-End User Authorization Service	62
4.2	REST Tables	62
4.2.1	Application Logic	63
4.2.2	User Identity Manager	65
4.2.3	Complex Event Processing	66
4.2.4	Publish/Subscribe Service	66
4.2.5	History Database	67
4.2.6	Connectivity Service	67
4.3	HTTPS	67
5	Performance Evaluation	70
6	Conclusion - Future Work	74
6.1	Conclusions	74
6.2	Future Work	75

Abstract

We present iHome, a smart home management service in the cloud. The service addresses the need of users to monitor and control their homes remotely provided that the home devices are “smart” themselves (i.e. they can be connected to the internet and operated remotely). Home devices transmit their identifier, measurements and status to a fog node and from there to the cloud. This information becomes available to registered users in the cloud based on subscriptions (i.e. to users authorized to review and respond to this information). User access rights are defined based on user roles (i.e. cloud administrators, home moderators and residents). Besides data publication and subscription services, an innovative feature of iHome, is a rule-based event management service which forwards alerts to subscribed users for responding to critical events (i.e. incidents of fire, malfunctioning appliances at home). iHome is implemented based on principles of Service Oriented Architecture design as a composition of RESTful services for the cloud side (back-end). Because a large number of smart devices for the front-end is not available to us, we decided to rely on software simulating the operation of smart devices at homes. This allowed us to shift focus from device specific functionality (e.g. IoT transmission protocols and vendor specific device functionality), to the design and implementation of the front and back-end solutions.

We run an exhaustive set of experiments using simulated (but realistic) data aiming to evaluate both, iHome response time and scalability. The results demonstrate that the system can handle up to a large number of users and data in real time.

ΠΕΡΙΛΗΨΗ

Παρουσιάζουμε το iHome, μια υπηρεσία διαχείρισης έξυπνου σπιτιού στο υπολογιστικό νέφος. Η υπηρεσία απευθύνεται στην ανάγκη των χρηστών να παρακολουθούν και να ελέγχουν απομακρυσμένα τα σπίτια τους, με προϋπόθεση ότι οι ίδιες οι συσκευές είναι και αυτές 'έξυπνες' (δηλαδή έχουν δυνατότητα σύνδεσης στο διαδίκτυο και απομακρυσμένης λειτουργίας). Οι συσκευές αυτές εκπέμπουν το μοναδικό αναγνωριστικό τους (ταυτότητα), τις μετρήσεις τους και την κατάστασή τους σε έναν κόμβο και από εκεί μεταφέρονται στο υπολογιστικό νέφος. Η πληροφορία αυτή γίνεται διαθέσιμη στους χρήστες οι οποίοι είναι συνδρομητές σε αυτή, μέσω του υπολογιστικού νέφους (σε χρήστες που έχουν εξουσιοδότηση να ελέγχουν και να ανταποκριθούν σε αυτήν την πληροφορία). Τα δικαιώματα πρόσβασης των χρηστών, ορίζονται ανάλογα με τον ρόλο τους (διαχειριστές υπηρεσίας, διαχειριστές σπιτιού, κάτοικοι). Εκτός από τις υπηρεσίες δημοσίευσης και συνδρομής, μια χαρακτηριστική λειτουργία του iHome είναι η υπηρεσία διαχείρισης συμβάντων βασισμένων σε κανόνες, η οποία προωθεί ειδοποιήσεις στους εγγεγραμμένους χρήστες ώστε να μπορούν να αντιδρούν σε κρίσιμα γεγονότα (περιστατικά πυρκαγιάς, δυσλειτουργία συσκευής στο σπίτι). Το iHome υλοποιήθηκε βασισμένο στις αρχές της Υπηρεσιοκεντρικής Αρχιτεκτονικής ως μια σύνθεση από RESTful υπηρεσίες στο υπολογιστικό νέφος. Λόγω του ότι δεν είχαμε πρόσβαση σε μεγάλο αριθμό έξυπνων συσκευών για την παραγωγή δεδομένων, αποφασίσαμε να βασιστούμε στην προσομοίωση αυτών. Έτσι, δε χρειάστηκε να δώσουμε έμφαση σε εξειδικευμένες λειτουργίες των συσκευών (π.χ. πρωτόκολλα μετάδοσης και μέθοδοι λειτουργίας συσκευών με βάση τον κατασκευαστή τους), επιτρέποντάς μας να δώσουμε βαρύτητα στον σχεδιασμό και στην υλοποίηση του συστήματος.

Προκειμένου να αξιολογήσουμε την απόδοση και την επεκτασιμότητα του iHome, εκτελέσαμε μια σειρά από απαιτητικά πειράματα, χρησιμοποιώντας προσομοιωμένα (αλλά ρεαλιστικά) δεδομένα. Τα αποτελέσματα έδειξαν πως το σύστημα μπορεί να διαχειριστεί μεγάλο αριθμό χρηστών και δεδομένων σε πραγματικό χρόνο.

Acknowledgments

I would like to express my gratitude to my supervisor, professor Euripides Petrakis for his valuable help and insightful comments. Also, I would like to thank the members of the laboratory for the excellent communication and collaboration, as well as my family and my friends for the continuous support they have given me throughout all these years.

Chapter 1

Introduction

Smart homes are the building blocks of smart cities. E-government, Green development and Energy Management policies for large residential areas may benefit from the evolution of smart home technologies. Smart policies and solutions at smart home level may influence policies applying at a larger scale (i.e. smart city level) and the reverse: Smart city policies may drive the development of smart home solutions for more efficient resource management (e.g. water, energy, internet etc.). Engineering and architectural solutions for smart homes are reviewed in [2, 4].

The idea of Internet of Things (IoT) combined with cloud computing, opens new horizons in the field of smart home technology. Smart home systems are now designed as two-fold solutions involving two interoperable and interconnected components: The IoT side (front-end) to support the network of home devices and their connection with the Internet (via some gateway or edge node) and, the cloud side (back-end) where home control functionality is implemented and where data are collected for permanent storage and analysis.

1.1 Problem Definition

The use of sensors installed in smart appliances and their capability for Internet connectivity provides significant benefits in applications areas that require fast and continuous monitoring from anywhere. In real-life smart home and smart city applications, huge amounts of data are collected and analyzed (e.g. improved home management solutions or better business policies can be designed based on the results of this analysis). These solutions have to be scalable (to deal with the ever-increasing number of smart homes and users, and finally, with the increased size of data), cost-effective, respond within reasonable time (e.g. considering the time constraints of each application) and, address concerns related to users privacy and data safety. While smart home solutions are deployed for the home and the cloud, security concerns arise at both places. Although cloud systems are considered to be more secure for deploying IoT applications, users and data are exposed to many risks as IoT at homes operates in the periphery of the cloud, it is open to many users and, as such, it is generally less protected than the cloud itself. The security concerns for the front-end part are addressed in [6]. In order to guarantee security at the back-end, data must

be transferred and stored securely to the back-end, user activities and system operations must be monitored at all times and, access to data and services must be allowed based on user roles and authorization [5].

The cloud is the ideal environment for implementing the back-end component of smart home applications due to reasons related to its affordability (no up-front investment, low operation costs), ease of deployment (taking advantage of IaaS and PaaS solutions already available in the market by many cloud vendors), low maintenance costs (i.e. easy software updates and enhancements), scalability (i.e. computational resources can be added on demand) and accessibility (i.e. smart home services can be accessed anytime from anywhere over the Web).

Cloud computing has some inherent disadvantages and limitations. Even though the cloud may offer virtually unlimited computational resources, internet bandwidth may impede application performance or, the cost of transferring and processing large amounts of data to the cloud call for hosting smart home services at the front-end side (rather than hosting all of them at the back-end side) or, closer to the place where data is generated (e.g. a server operating at home). The distance between the front and back-end is also the main reason for long delays in responding to critical events occurring at home. To address these limitations, the paradigm of fog computing emerged lately, starting from Cisco [1]. Fog computing can be assumed as an extension of cloud computing, bringing virtualized services closer to the edge of the network (e.g. closer to IoT devices). Fog brings the benefits of low latency (due to its close proximity with the IoT devices), location awareness and increased security, privacy and availability. Efforts to standardize architectures and platforms for extending the cloud to support functional edge nodes are currently underway [3].

1.2 Contribution

The concept of smart home is interesting and trendy¹. Existing solutions relying on the idea sensors and appliances connected to the Internet are now becoming available as commercial services² supporting functionalities ranging from home automation, intelligent home control, ambient or assisted living and health monitoring, to security control (e.g. anti-burglary systems). Off-the shelf smart devices with the desired functionality are currently becoming available in the market at affordable prices and can be integrated into smart home applications. An important feature of all systems (and also of iHome) is user interaction with a mobile device and intelligent user interfaces (using e.g. voice interfaces) over the Internet. Commercial applications promise fast responses, high availability and, (most important) high reliability. iHome is not intended to compete commercial solutions in terms of performance but rather, to show how a cost effective smart home system based on innovative services can be designed and deployed in the cloud and fog using well established, open-source technologies and principles of service oriented design.

iHome, is a service that runs in the cloud and the fog and implements a typical smart home scenario: Smart devices installed at home can transmit measurements or status information to gateways and from there to the cloud. To mitigate concerns in regards to data security and in regards to delays in

¹<https://www.qulix.com/industries/internet-of-things-apps/smart-home/>

²<https://thinkmobiles.com/blog/best-smart-home-apps/>

delivering large amounts of data to the cloud, services for the home are realized within a fog node installed at home. Access to this information is allowed only for users registered to iHome in the cloud based on their role and authorization. (i.e. cloud administrators, home administrators, residents). Permission to access homes is granted by cloud administrators. Home users (i.e. home administrators and residents) may access home information based on subscriptions. Besides measurements, alerts are a special type of information which is generated when events take place. These can be simple events (e.g. room temperature exceeds a threshold) or critical events (e.g. incidents of security breach or fire). The events are described in terms of rules involving measurements and threshold values.

The analysis of history (log) data collected from a large number of homes (eventually this information becomes big) can lead to important conclusions in relation to people habits and behavior, their needs, the causes of events or, in relation to energy consumption at peak and non-peak hours. The analysis of this information may provide the means for policy makers, energy managers to improve their plan for more efficient, safer and profitable management of a smart city.

Chapter 2

Related Work and Technologies - Tools

2.1 Smart Home systems

There are many smart home systems available to the consumers; commercial or open source, some of the most popular being Amazon's Alexa, Apple's HomeKit, Google's Assistant, Home Assistant, OpenHab, Pytomatic.

2.1.1 Amazon Alexa



Alexa is Amazon's digital assistant that's integrated into devices such as Amazon Echo (speaker - microphone). She can be connected with other compatible devices like thermostats, light switches TVs etc. These devices are controlled by issuing vocal commands to her. Also, she can try provide information when asked. Using the Alexa app, we can add devices that are connected on the same network.

2.1.2 Apple HomeKit



HomeKit enables devices that don't necessarily sync, to work with each other. Devices can be added to the system using QR and NFC and can be

controlled from the Home App via iOS devices (i.e. smartphone, HomePod etc.) that have Siri (equivalent of Alexa) integrated.

2.1.3 Google Home



Google Home uses Google Assistant as a controller, that is build in Android smartphones and smartwatches. Compatible devices can be controlled through it by issuing voice commands on an Android smartphone, smartwatch or smart speakers (Google Home, Home Mini, Home Max). Adding a device is achieved by scanning for compatible devices connected on the same network.

2.1.4 Home Assistant¹



Home Assistant is an open source, stand alone system that runs on Python 3 and does not depend on third-party cloud services. It is usually installed on a Raspberry Pi, lowering the cost of the setup. It supports many third-party components like communication protocols, automation platforms and voice assistants.

2.1.5 OpenHab²



OpenHab is an open source home automation system that is based on Java. It supports a wide number of third-party devices and multiple operating systems. It is designed to run on device bindings, making it totally modular. OpenHab supports almost all commercial brand protocols (i.e. Http, MQTT, Z-Wave etc.).

¹<https://www.home-assistant.io/>

²<https://www.openhab.org/>

2.1.6 Pytomaticion³



Pytomaticion is an open source home automation system, developed using the Python language. It's mostly used with the Raspberry Pi and it works as a microcontroller that can be managed via mobile applications and web servers. Like Home Assistant, Pytomaticion is not depended on the cloud and other third-party web services.

2.2 Openstack⁴



OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. An OpenStack deployment contains a number of components providing APIs to access infrastructure resources. The most popular components are the following⁵:

- **Horizon:** Provides a web-based interface for both cloud administrators and cloud tenants. Using this interface, administrators and tenants can provision, manage, and monitor cloud resources. The dashboard is commonly deployed in a public-facing manner with all the usual security concerns of public web portals.
- **Nova:** Provides on-demand access to compute resources by provisioning and managing virtual servers as well as services to support the management of virtual machine instances at scale, instances that host multi-tiered applications, dev or test environments, “Big Data” crunching Hadoop clusters, or high-performance computing.
- **Neutron:** Delivers networking as a service to cloud users, including the creation of private networks, IP address management, DNS, DHCP, load balancing and security groups. Also, neutron provides a framework for software defined networking (SDN) that allows for pluggable integration with various networking solutions.
- **Swift:** A highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply. The Object Storage service provides both a native API and an Amazon Web Services S3-compatible API. The service provides a high

³<http://www.pytomaticion.com/>

⁴<https://www.openstack.org/software/>

⁵<https://docs.openstack.org/security-guide/introduction/introduction-to-openstack.html>

degree of resiliency through data replication and can handle petabytes of data.

- **Glance:** Provides disk-image management services, including image discovery, registration, and delivery services to the Compute service, as needed. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image. VM images made available through Glance can be stored in a variety of locations from simple filesystems to object-storage systems like the OpenStack Swift project.
- **Keystone:** Shared service that provides API client authentication, service discovery, and distributed multi-tenant authorization by implementing OpenStack's Identity API. It supports LDAP, OAuth, OpenID Connect, SAML and SQL.
- **Cinder:** Virtualizes the management of block storage devices and provides end users with a self service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device. This is done through the use of either a reference implementation (LVM) or plugin drivers for other storage.

2.3 FIWARE⁶

FIWARE is an open middleware platform, driven and funded by the Future Internet Public-Private Partnership (FI-PPP) of the EU. It provides an enhanced Openstack-based cloud environment plus a rich set of open standard APIs and OSS components that enables and eases the full adoption of Future Internet technologies such as Internet of Things, Big Data Analytics, Service Marketplaces, Advanced multimedia, User Interaction and Privacy and Security Management⁷.

The FIWARE Community is an independent Open Community whose members are committed to materialize the FIWARE mission, that is: “to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors”. The FIWARE Community is not only formed by contributors to the technology (the FIWARE platform) but also those who contribute in building the FIWARE ecosystem and making it sustainable over time. As such, individuals and organizations committing relevant resources in FIWARE Lab activities or activities of the FIWARE Accelerator, FIWARE Mundus or FIWARE iHubs programmes are also considered members of the FIWARE community.

FIWARE Lab is a non-commercial sandbox environment where innovation and experimentation based on FIWARE technologies take place. Entrepreneurs and individuals can test the technology as well as their applications on FIWARE Lab, exploiting Open Data published by cities and other organizations. FIWARE Lab is deployed over a geographically distributed network of federated nodes leveraging on a wide range of experimental infrastructures.

⁶<https://www.fiware.org/>

⁷<https://fimac.m-iti.org/dse.php>

2.4 Fiware Services

Cloud providers offer catalogs of their services to their clients, which can be assembled together and with other third party components, in order to assist them in creating their own applications. FIWARE's catalog has a rich library of services (Generic Enablers) that enable the developers to implement functions such as connection with the Internet Of Things, Big Data Analysis, Context Management, User Authentication etc. FIWARE's GEs are open source and free of charge to its users and are derived in the following 7 categories:

- Advanced middleware and interfaces to Network and Devices
- Advanced Web-based User Interface
- Applications/Services and Data Delivery
- Cloud Hosting
- Data/Context Management
- Internet of Things Services Enablement
- Security

For our application, we used two GEs, Orion Context Broker and Keyrock IdM.

2.4.1 Publish/Subscribe Context Broker - Orion Context Broker⁸



Orion Context Broker is an implementation of the Publish/Subscribe Context Broker GE, providing the NGSI9 and NGSI10 interfaces. Using these interfaces, clients can do several operations:

- Register context producer applications, e.g. a temperature sensor within a room
- Update context information, e.g. send updates of temperature
- Being notified when changes on context information take place (e.g. the temperature has changed) or with a given frequency (e.g. get the temperature each minute)
- Query context information. The Orion Context Broker stores context information updated from applications, so queries are resolved based on that information.

Orion uses MongoDB to store context in the form of entities.

⁸<https://catalogue-server.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker>

2.4.2 Identity Management (IdM) - Keyrock IdM⁹



Identity Management covers a number of aspects involving users' access to networks, services and applications, including secure and private authentication from users to devices, networks and services, authorization and trust management, user profile management, privacy-preserving disposition of personal data, Single Sign-On (SSO) to service domains and Identity Federation towards applications. The Identity Manager is the central component that provides a bridge between IdM systems at connectivity-level and application-level. Furthermore, Identity Management is used for authorizing foreign services to access personal data stored in a secure environment. Hereby usually the owner of the data must give consent to access the data; the consent-giving procedure also implies certain user authentication.

2.5 Service Oriented Architecture (SOA)¹⁰

SOA is an application architecture that is based upon the deployment of services that communicate with each other.

Services are functions that are independent, self-contained with well-defined invocable interfaces, which can be called in defined sequences to form application processes¹¹. External components don't know nor care how services execute their function; the important thing is that they return the expected response. Services carry out functions such as validating customers, producing data etc and they transmit simple data as messages. Being invocable means that they can be called regardless of being deployed locally or remotely. They might be in the same application or in a totally different system and still be accessible. Also, the protocol that is used to effect the invocation or the components that are used to make the connection between services are irrelevant. Finally, SOA allows for service reuse so developing or updating an application from scratch is not necessary.

2.6 Web Services¹²

Web Services are the services that exchange data in a text-based format over HTTP protocol. They are self-contained, modular, distributed, dynamic applications that can be described, published, located or invoked over the network to create products and processes. These applications can be local, distributed or web-based. Web services are built on top of open standards such as TCP/IP,

⁹<https://catalogue-server.fiware.org/enablers/identity-management-keyrock>

¹⁰<https://www.service-architecture.com/articles/web-services/service-oriented-architecture-soa-definition.html>

¹¹Migrating to a service-oriented architecture - Kishore Channabasavaiah and Kerrie Holley, IBM Global Services, and Edward M. Tuggle, Jr., IBM Software Group

¹²https://www.tutorialspoint.com/webservices/what_are_web_services.htm

HTTP, Java, Python, HTML, JSON, XML etc. Applications written in various programming languages and running on various platforms can use web services to exchange data over the internet.

2.7 REST¹³

REpresentational State Transfer (REST) is a design architecture that defines a set of constraints to be used for creating web services (RESTful Web Services). These constraints are described below.

- **Client-Server architecture**

Separation of concerns is the principle behind the client-server constraints. Separating the user interface concerns from the data storage concerns improves the portability of the user interface across multiple platforms and improves scalability by simplifying the server components. This separation allows the components to evolve independently, supporting the internet-scale requirement of multiple organizational domains.

- **Stateless**

This constraint refers to the client-server interaction. Communication must be stateless, meaning that the requests that a client makes to a server must include all of the information necessary for the server to fulfill the request. The server doesn't store information from previous requests and on a failure, the client must send the request again. Statelessness improves visibility, scalability and reliability properties. In order to understand a request, a system must simply look at the request's data, thus improving visibility. As mentioned before, on a partial failure, recovery is easily achieved by resenting the request, improving the reliability of the system. Last, not storing state between requests means that the server does not need to manage resources across request, enabling him to quickly free resources and simplify its implementation, improving scalability. Of course, statelessness has some disadvantages such as the possible decrease of network performance due to the increase of repetitive data (i.e per-interaction overhead) sent in requests.

- **Uniform Interface**

Using a uniform interface between components is what distinguishes the REST architectural style from other network-based styles. This simplifies and decouples the system's architecture and improves the visibility of interactions, enabling each part to evolve independently. Resources are manipulated using CRUD operations. The main concepts of the uniform interface are:

- **Resource identification in requests.** In RESTful Web services, resources are identified in requests using URIs. They are separate from the representations that are returned to the client (i.e. Server can send data from a database as JSON).

¹³UNIVERSITY OF CALIFORNIA, IRVINE Architectural Styles and the Design of Network-based Software Architectures by Roy Thomas Fielding

- **Resource manipulation through representations.** The representation of a resource that a client receives contains information such as metadata, so the client can modify or delete the resource.
- **Self-descriptive messages.** Messages include information to help with the message processing.
- **Hypermedia as the engine of application state (HATEOAS¹⁴).** Servers provide hyperlinks dynamically to clients so they can discover all the available actions and resources they need, decoupling the client from the application URI structure so there is no need for the client to have hard-coded information for the structure of the application.

2.7.1 JSON¹⁵

The most common format for data exchange between modern web services is the JSON (JavaScript Object Notation) format. JSON is a lightweight, human readable file format for storing and transporting data as objects. An object is an unordered set of key-value pairs and it begins and ends with {} brackets. The left part of each field is the **Key** which is a unique string that is enclosed in quotes. The right part of each field is the **Value** which can be a string, number, array, boolean, null or another object. For example, we have an object "person" that has name and age attributes. The JSON object would be:

```
{name": "John", "age":20}
```

2.8 NoSQL Database - MongoDB¹⁶

A non-relational (NoSQL) database is a database that does not implement the table-key model that the relational databases use. These databases were designed to overcome the limitations of the relational databases in handling Big Data. Unlike relational databases which require to predefine a schema in order to form the structure of data, non relational databases have a dynamic schema for unstructured data, simplifying a possible change in the application's structure. Data in a non relational database are unstructured and can be stored as key-value pairs, documents and graphs faster than data in a SQL database and as most of the data that is generated today is unstructured, this is a very useful aspect.

Usually, SQL databases support vertical scalability, meaning that in order to cope with increasing load, the server can be supported by adding more CPU, RAM and Storage. On the other hand, NoSQL databases are horizontally scalable, meaning that we can add more servers to handle the load.

Some popular NoSQL databases are MongoDB, Cassandra, CouchDB, HBase.

¹⁴<https://en.wikipedia.org/wiki/HATEOAS>

¹⁵<https://www.json.org/>

¹⁶<https://www.mongodb.com/scale/what-is-a-non-relational-database>

```
Room Log:
{
  "homeId": "5bfc2052ea636e15c973a7bd",
  "roomId": "5c0281e37592c31084a217e1",
  "roomName": "kitchen",
  "floor": 1,
  "temperature": 17.5,
  "humidity": 95,
  "luminocity": 70,
  "lastUser": "home",
  "action": "update",
  "notificationType": "roomLog"
}
```

Figure 2.1: Example of MongoDB document - key values pairs

2.9 PHP

PHP is a server side scripting language that is designed for web development. Can be deployed on most web servers, many operating systems and platforms and can be used with many relational and non relational database management systems.

2.10 Java EE¹⁷

Java-EE is a Java based platform for server programming with specifications for enterprise features such as web services. Java EE includes several specifications that serve different purposes, like generating web pages, reading and writing from a database in a transactional way, managing servlets for HTTP requests, Java API for RESTful Web services and JSON Processing.

2.11 Spring Boot¹⁸

Spring is an open source application framework for the Java platform that provides many extensions for building web applications. Spring Boot is Spring's convention-over-configuration¹⁹ solution for creating stand-alone, production-grade Spring-based Applications that we can "just run". Spring Boot's main features are:

- Create stand-alone Spring applications
- Embedded Tomcat server
- Automatically configure Spring whenever possible
- Provide opinionated 'starter' Project Object Models (POMs) to simplify your Maven²⁰ configuration

¹⁷https://en.wikipedia.org/wiki/Java_platform,_Enterprise_Edition

¹⁸https://en.wikipedia.org/wiki/Spring_Framework

¹⁹https://en.wikipedia.org/wiki/Convention_over_configuration

²⁰https://en.wikipedia.org/wiki/Apache_Maven

Chapter 3

Requirement Analysis and System Design

Requirement analysis relates to describing the needs or the conditions that the system must meet in order to function properly (i.e. meet the expectations of its users). Requirements are distinguished in functional and non-functional.

3.1 Requirement Analysis

Functional requirements describe the needs of the users. However, not all system users apply the same operations or share the same view in regards to what functionality the system should support. In the following, the system functional requirements are examined separately per user group or category. There are also requirements that apply to all users henceforth referred to as General requirements.

3.1.1 User Groups

Our system supports three user types:

- **Cloud Moderator (Infrastructure Administrator):** There is only one infrastructure moderator who is responsible for the home and user management.
- **Home Moderator (Home Administrator):** Every home can have only one home moderator who is responsible for the room, device, user access and rule management.
- **Resident:** Every home can have none or more residents. Resident's device access is restricted and is controlled by the home moderator by granting or denying the resident operation permissions for a device group.

3.1.2 Functional Requirements

Functional requirements are the requirements that describe our system's behavior and they may differ between user groups.

General Functional Requirements

- **User Authentication:** All users must be registered with the cloud (FIWARE in our case) and have a username and password in order to log into the application.
- **User Authorization:** Access to services (e.g. system operations) is allowed only to authorized users. For example, a user must not be able to retrieve the room temperature of a home he doesn't belong to.

Cloud Moderator

- Home Insert/Edit/View/Delete
- Home Moderator Insert/Edit/View/Delete
- User Insert/Edit/View/Delete
- View Homes
- View Home Moderators/Residents
- View Home/Home Moderator/Resident History

Home Moderator

- Room Insert/Edit/View/Delete
- Device Insert/Operate/View/Delete
- Resident Access Control
- Rule Insert/Delete/View
- View Rooms
- View Devices
- View Users
- View Room/Device History

Resident

- View Rooms
- View Devices
- Operate Device

3.1.3 Non-Functional Requirements¹

Non-functional requirements are the requirements that specify criteria that can be used to judge that the system operates as expected.

¹https://en.wikipedia.org/wiki/Non-functional_requirement

Availability: It's important for a web application to be always online and available to the user. The user must be able to access it from anywhere as long as he has access to an internet connection and a web browser. We achieve that by having our application deployed on the cloud.

Security: It refers to the secure access of a user in the application, the protection of user's data from unauthorized users as well as the prevention of unauthorized access to the system.

Performance: Has to do with the response time of the system (for example how much time does it need for the lights to turn on once the users issues the command)

Scalability: The capability of the system to handle an increasing number of requests (horizontal, vertical scalability)

Disaster recovery: The application must be able to recover fast after a disaster (for example DDOS attack, server failure) and this can be achieved in a cloud environment by having backups of the service ready to be deployed.

Open source: The software is open for study, change or distributed according to any ones interests.

Usability: It refers to the ease of use by a customer. This is achieved by providing a user interface.

3.2 System Design - UML Diagrams²

In order to better understand the system's functionalities and their relation to the user groups, we will use UML diagrams. UML diagrams are graphical representations of the system's functions or the system itself.

3.2.1 Use Case Diagrams

Use case diagrams are used to describe a set of actions that someone can perform. Having defined the system's functional requirements for each one of the user groups, we created relative services that our system provides to them.

Cloud Moderator

For the cloud moderator, there are two services:

- User Management Service that includes:
Insert/Edit/Delete/View user
- Home Management Service that includes:
Insert/Edit/Delete/View home

²<https://www.uml-diagrams.org>

Apart from these two services, the Cloud Moderator can access the home and user logs. These logs contain information about the moment a home or a user was created, edited or deleted from the system and are retrieved from the history database. We can see these services in the diagram below

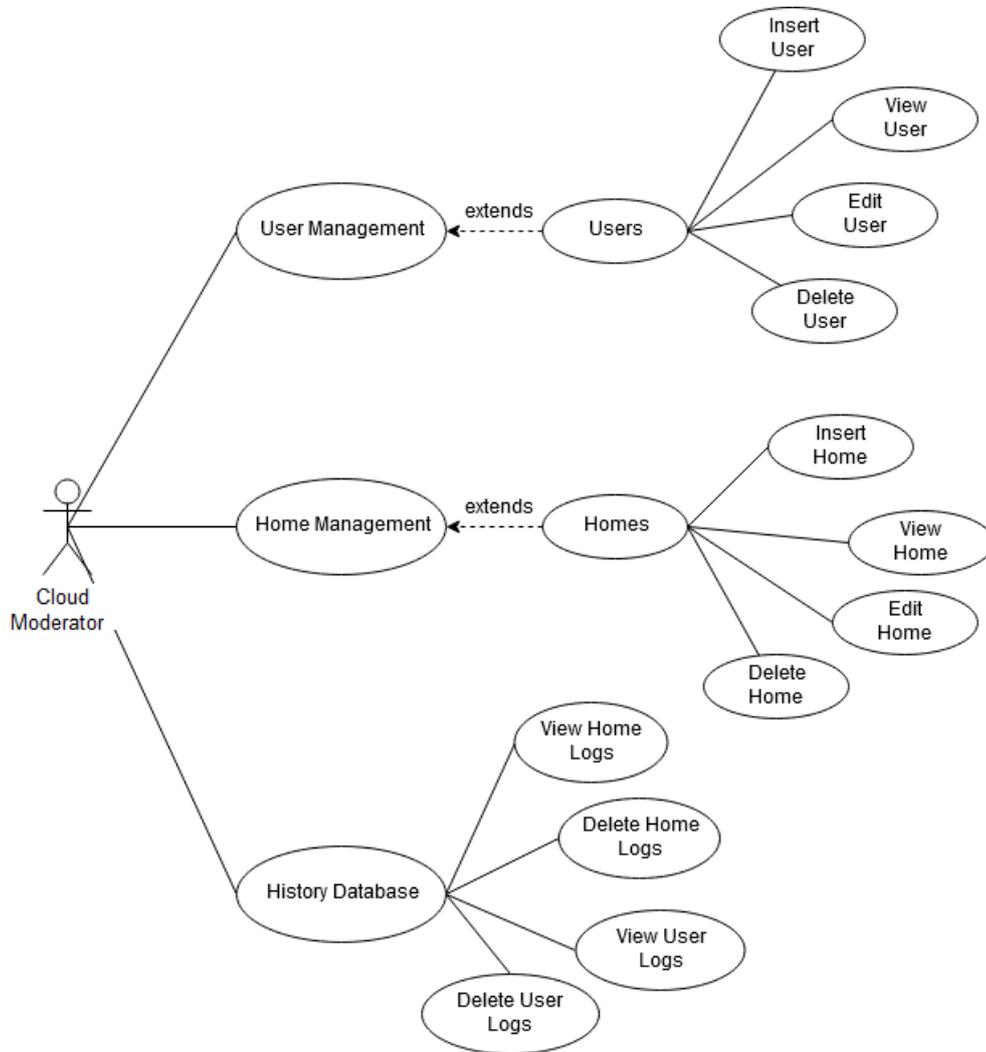


Figure 3.1: Cloud Moderator Use Case Diagram

Home Moderator

Likewise, we created the following four services for the Home Moderator:

- Room Management Service that includes:
Insert/Edit/Delete/View Room
- Device Management Service that includes:
Insert/Operate/Delete/View Device
- User Management Service that includes:
View/Edit Resident Access
- Rule Management Service that includes:
Insert/Delete/View Rule

In addition, Home Moderator has access to room and device logs, that are retrieved from the history database and contain information about the moment a room or a device was created, edited or deleted. Finally, he can receive notifications and alerts that are created on events such as the switch on of a device when a rule is triggered or a device failure. This mechanism will be further explained later in this chapter. The following diagram illustrates Home Moderator's available actions.

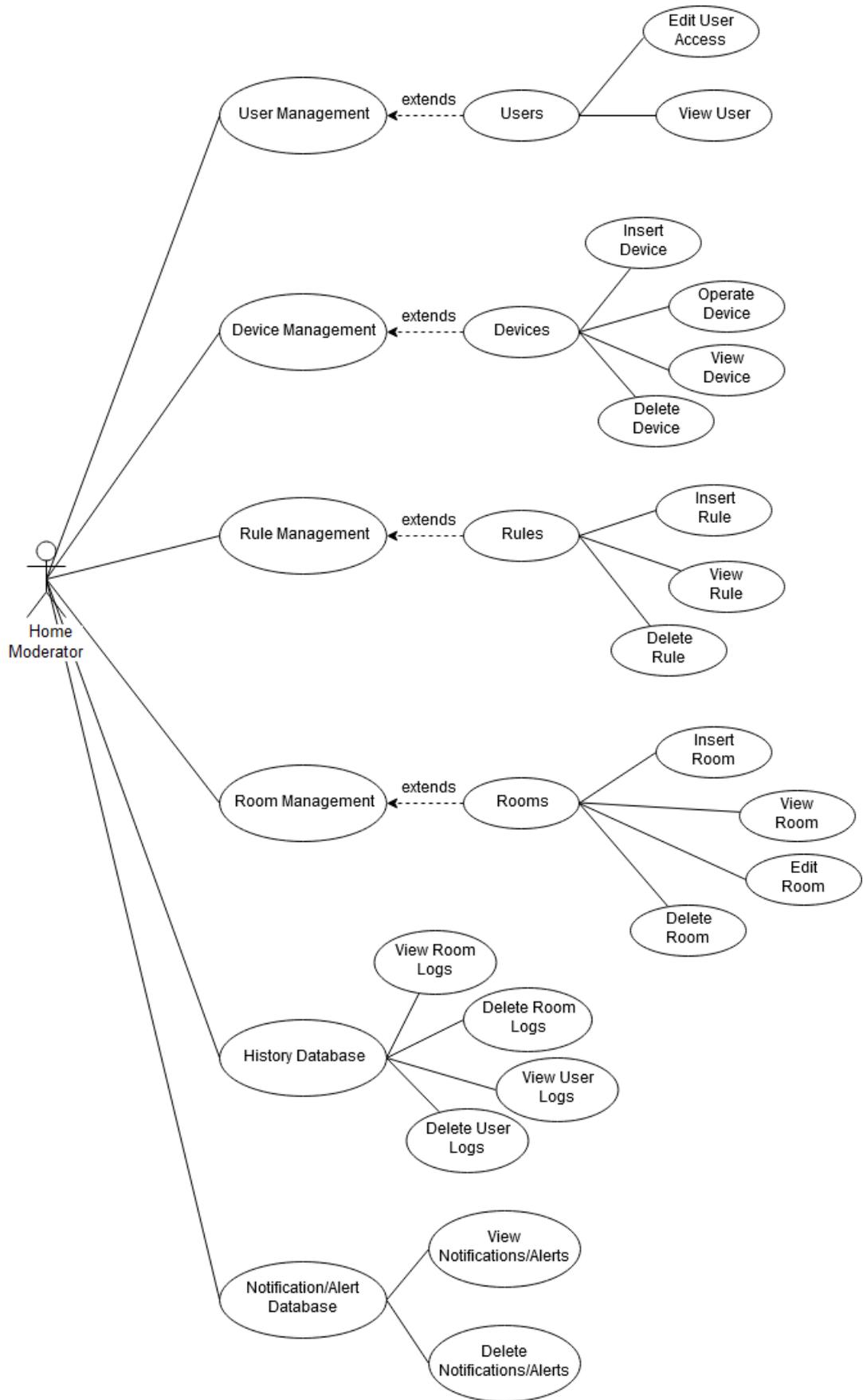


Figure 3.2: Home Moderator Use Case Diagram

Resident

Resident's operations are generally more restricted as he can only view the rooms and the devices of the home and operate the devices that the Home Moderator has given him access to. He can also view the logs of his device operations and retrieve notifications and alerts. That said, we can assume one service:

- Device Management Service that includes:
View/Operate Device

In figure 1.3 we see the operations that a Resident can execute.

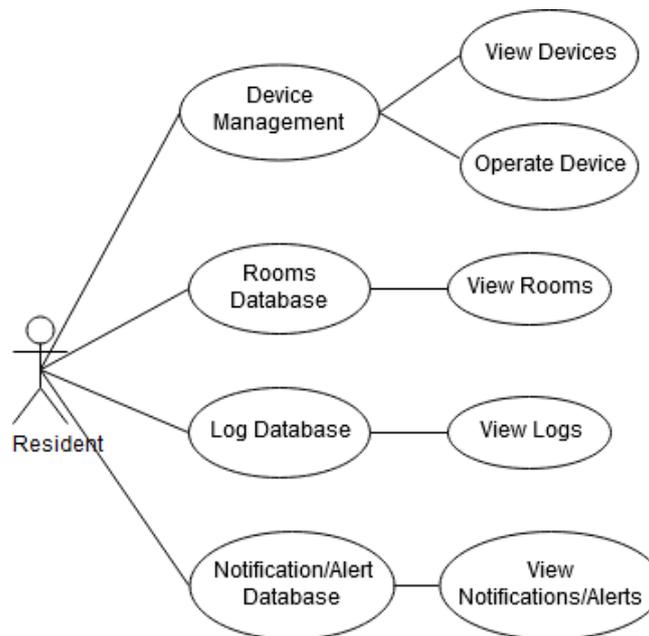


Figure 3.3: Resident Use Case Diagram

3.2.2 Activity Diagrams

Activity diagrams show us the steps and actions that can take place when an operation is executed. Again, we are separating them per user group.

Cloud Moderator

House Management Service

1. Cloud Moderator logs into the system.
2. Selects the Insert Home option from the UI. A form that must be filled with the home's attributes is displayed. If the form is submitted successfully, a home entity is created into the system.
4. Selects the Manage Homes option from the user interface and a table with all the system's homes appears. Three operations are available:

- 4.a. Edit Home: A form with the home's attributes filled is showed to Cloud Moderator. If he successfully submits the edited form, the home entity is updated.
- 4.b. View Home: Home's attributes are presented to the Cloud Moderator in a table form.
- 4.c. Delete Home: The home is deleted from the system along with it's Home Moderator, Residents, rooms, devices and rules.

The diagram below demonstrates the Home Management Service's operations; Insert/Edit/Delete/View Home.

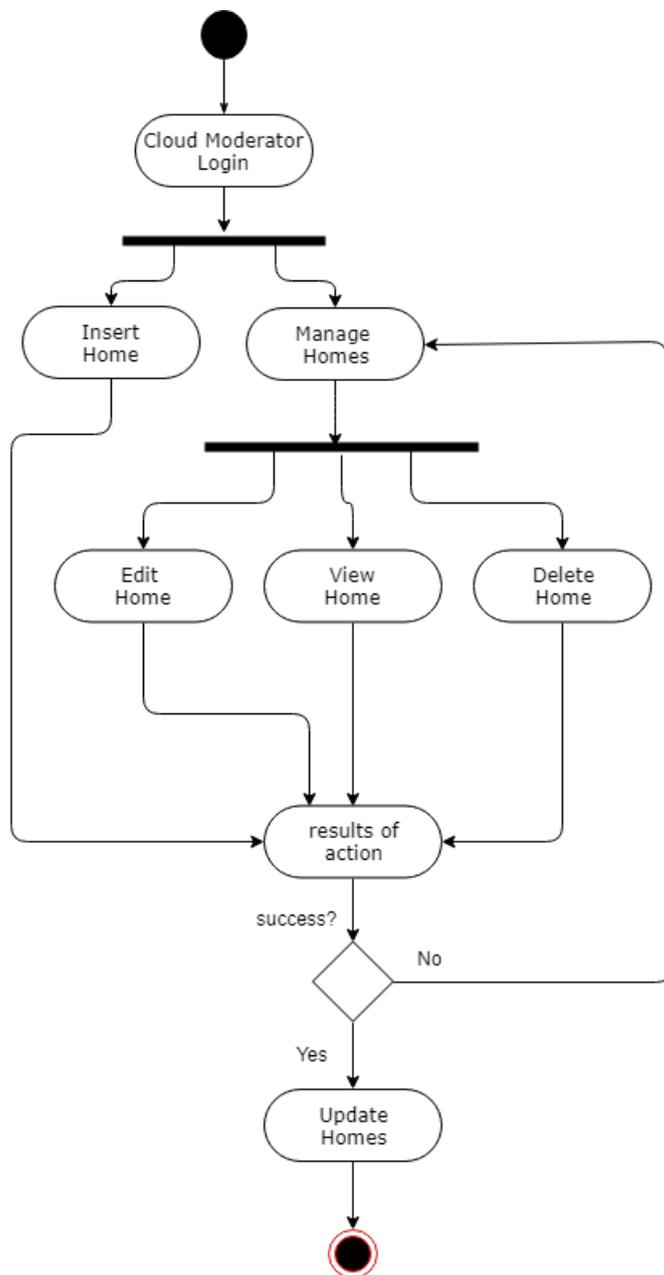


Figure 3.4: Cloud Moderator Home Management Activity Diagram

User Management Service

1. Cloud Moderator logs into the system.
2. Selects the Insert User option from the UI. A form that must be filled with the user's attributes as well as the home Id he belongs to, is presented to the Cloud Moderator. If the form is submitted successfully, a user entity

is created into the system. We should point that the username of a user must be the same as the username the user uses to login in FIWARE and we will explain the reason later.

4. Selects the Manage Home Moderator/Resident option from the user interface and a table with all the system's Home Moderators/Residents appears. Three operations are available:
 - 4.a. Edit User: A form with the user's attributes filled is showed to Cloud Moderator. If he successfully submits the edited form, the user entity is updated.
 - 4.b. View User: User's attributes are presented to the Cloud Moderator in a table form.
 - 4.c. Delete User: User is deleted from the system.

The User Management Service's operations are illustrated in the diagram below; Insert/Edit/Delete/View User.

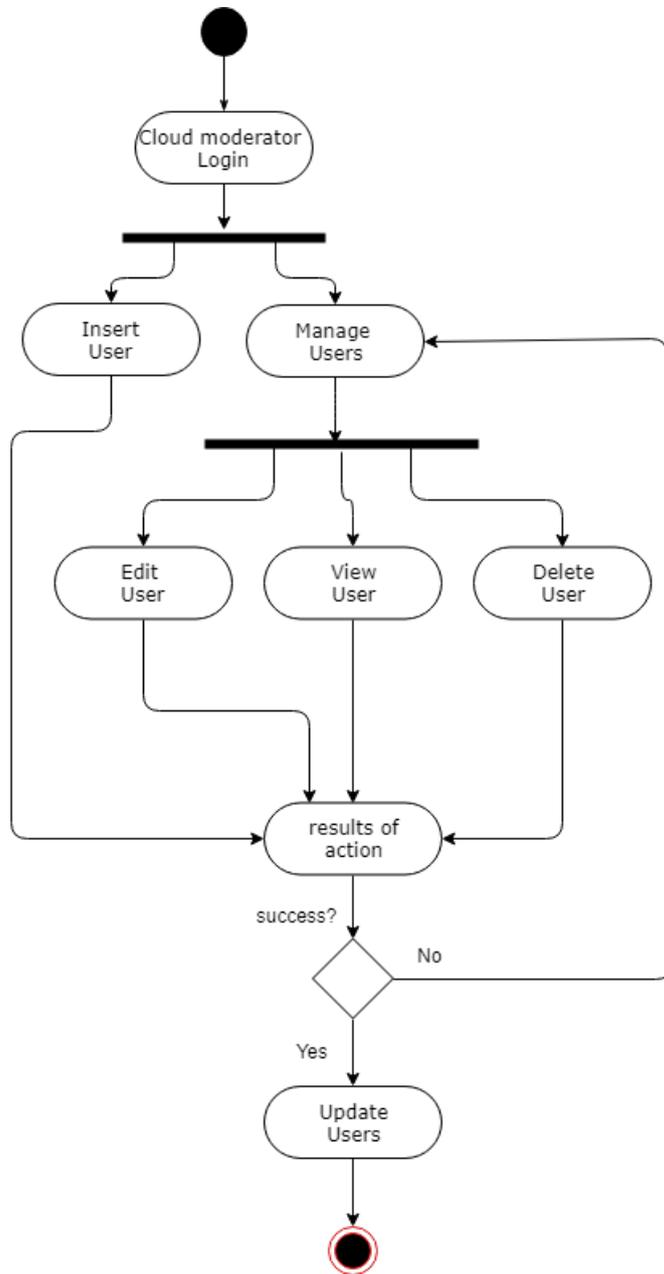


Figure 3.5: Cloud Moderator User Management Activity Diagram

Home Moderator

Room Management Service

1. Home Moderator logs into the system.
2. Selects the Insert Room option from the UI. A form that must be filled

with the room's attributes is displayed to him. If the form is submitted successfully, a room entity is created into the system.

4. Selects the Manage Room option from the user interface and a table with all the home's rooms appears. Three operations are available:
 - 4.a. Edit Room: A form with the room's attributes filled is showed to the Home Moderator. If he successfully submits the edited form, the room entity is updated.
 - 4.b. View Room: Room's attributes and devices are presented to the Home Moderator. From here, he can operate a device.
 - 4.c. Delete Room: Room is deleted from the system along with it's devices and rules.

In the following diagram we can see the Room Management Service's operations; Insert/Edit/Delete/View Room.

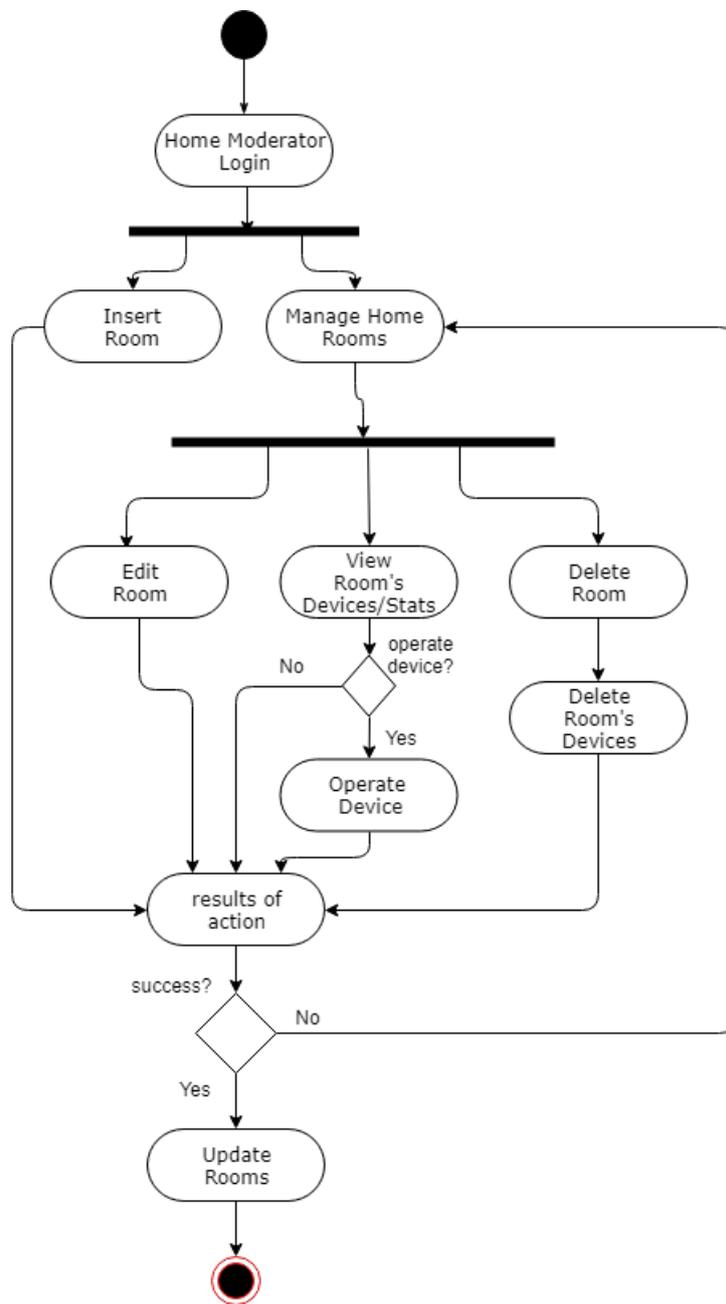


Figure 3.6: Home Moderator Room Management Activity Diagram

Home's User Management Service

1. Home Moderator logs into the system.
3. Selects the Manage Users option from the UI and a table with all the home's users appears. Two operations are available:

- 3.a. View User: Resident's attributes and access rights are presented to the Home Moderator.
- 3.b. Edit User Access Rights: Can grand or deny access for a device type to the Resident. (Figure 1.9)

The diagram below represents the Home's Resident Management Service operations; View/Edit User's Access Rights.

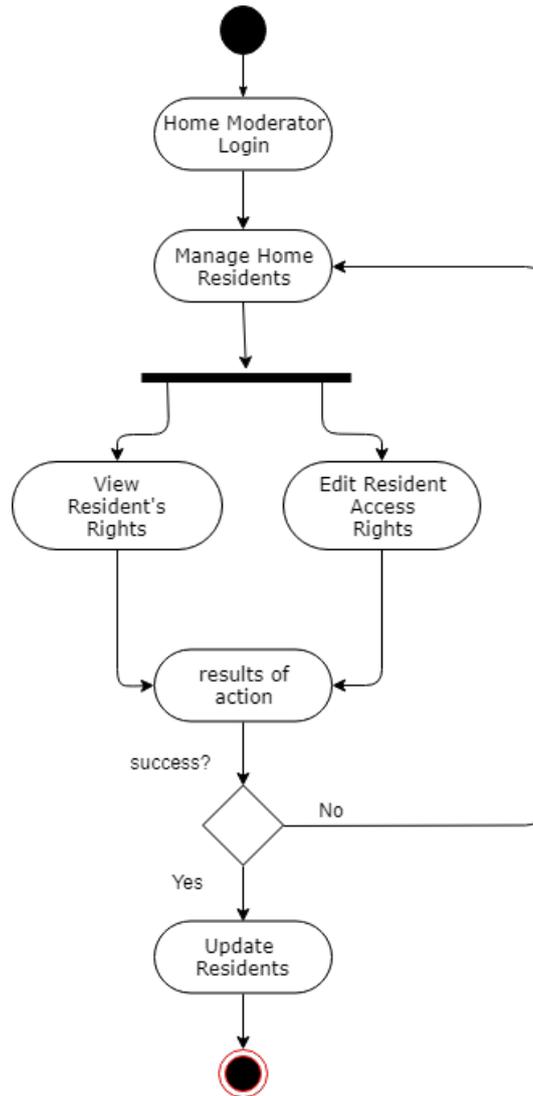


Figure 3.7: Home Moderator User Management Activity Diagram

In our system, every home's user has an "access number" as an attribute. This number, R, has as many digits as our system's device types which in the current state is nine. When a digit of this number is 1, it means that the user has access to the relative device type. When it's 0, user has no access. Home Moderators

have access to all devices (their access number is "11111111"). For better understanding, let's think of the following example. Assume that we have a room with three devices; an air conditioner, an oven and a lamp light. A Home Moderator, apart from his ability to insert, view or delete a device, can also operate any of them. On the other hand, a Resident has restricted access to the devices, which is controlled by the Home Moderator; he can allow or deny a Resident from operating a device type. In the image below, we see the access number of the home's Resident; all digits are 0. This means he has no access in any device type.

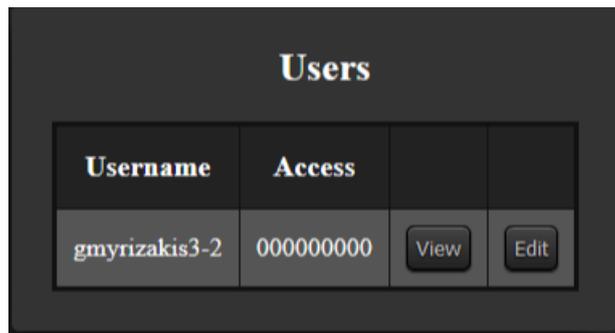


Figure 3.8: View Home's Users

Figure 1.9 shows us how the Home Moderator can give access to a Resident on a device group, by selecting the respective boxes.



Figure 3.9: Edit Home User's Rights

In the image bellow we see which digit of the number R corresponds a device type. R(8) is the left digit and R(0) the right.

R(8)	R(7)	R(6)	R(5)	R(4)	R(3)	R(2)	R(1)	R(0)
L	A	R	W	D	O	B	T	A
a	i	e	a	i	v	l	V	l
m	r	f	s	s	e	i		a
p		r	h	h	n	n		r
	C	i	i	w		d		m
L	o	g	n	a		s		
I	n	e	g	s				
g	d	r		h				
h	i	a	M	e				
t	t	t	a	r				
	i	o	c					
	o	r	h					
	n		i					
	e		n					
	r		e					

1 = Has access

0 = No access

Figure 3.10: Rights Access 9-digit number

So, taking into consideration the image above, in our example if Home Moderator selected the "Air Conditioner" and "Lamp Light" boxes, Resident's access number R, from "000000000" (figure 1.8) would have become "110000000". This number is checked every time a Resident tries to operate a device.

Device Management Service

1. Home Moderator logs into the system.
2. Selects the Insert Device option from the UI. A form that must be filled with the device's attributes is displayed. If the form is submitted successfully, a device entity is created into the system.
4. Selects the Manage Devices option from the user interface and a table with all the home's devices appears. Three operations are available:
 - 4.a. Operate Device: An interface of the device is shown to the Home Moderator (like a controller). If he successfully submits new operation data, the device entity is updated and a signal is send to the home.

4.b. View Device: Device's attributes are presented to the Home Moderator.

4.c. Delete Device: Device is deleted from the system.

The diagram below shows the Device Management Service's operations; Insert/Operate/Delete/View Device.

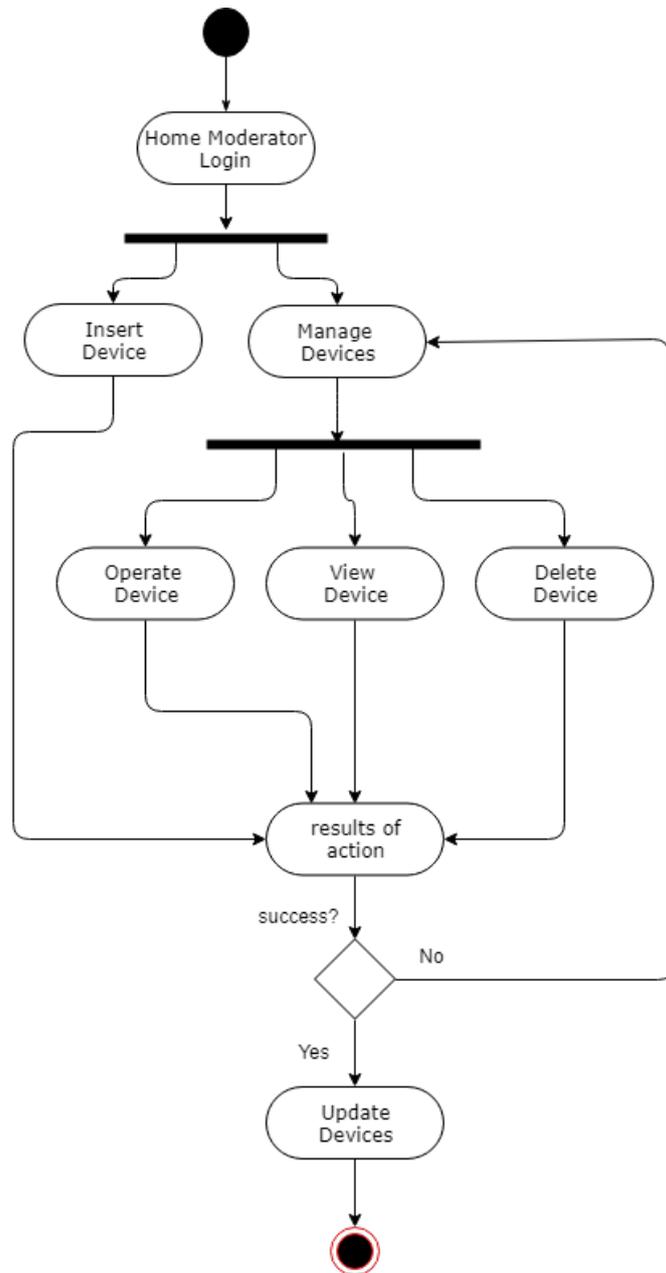


Figure 3.11: Home Moderator Device Management Activity Diagram

Rule Management Service

1. Home Moderator logs into the system.
2. Selects the Insert Rule option from the UI. First of all, the rule type must be chosen. Our system supports two rule types; condition rule and time rule. In order to create a condition rule, one or two conditions (AND/OR relation between them) must be set. When a time rule is to be created, the time of the execution must be set. In both cases, a device must be chosen to be operated when the rule is triggered. If the form is submitted successfully, a rule entity is created into the system.
4. Selects the Manage Rules option from the user interface and a table with all the home's rules appears. Two operations are available:
 - 4.a. View Rule: Rule's description is presented to the Home Moderator.
 - 4.b. Delete Rule: Rule is deleted from the system. Here we should point out that only conditional rules can be deleted.

In the following diagram we can see the Rule Management Service's operations; Insert/Delete/View Rule.

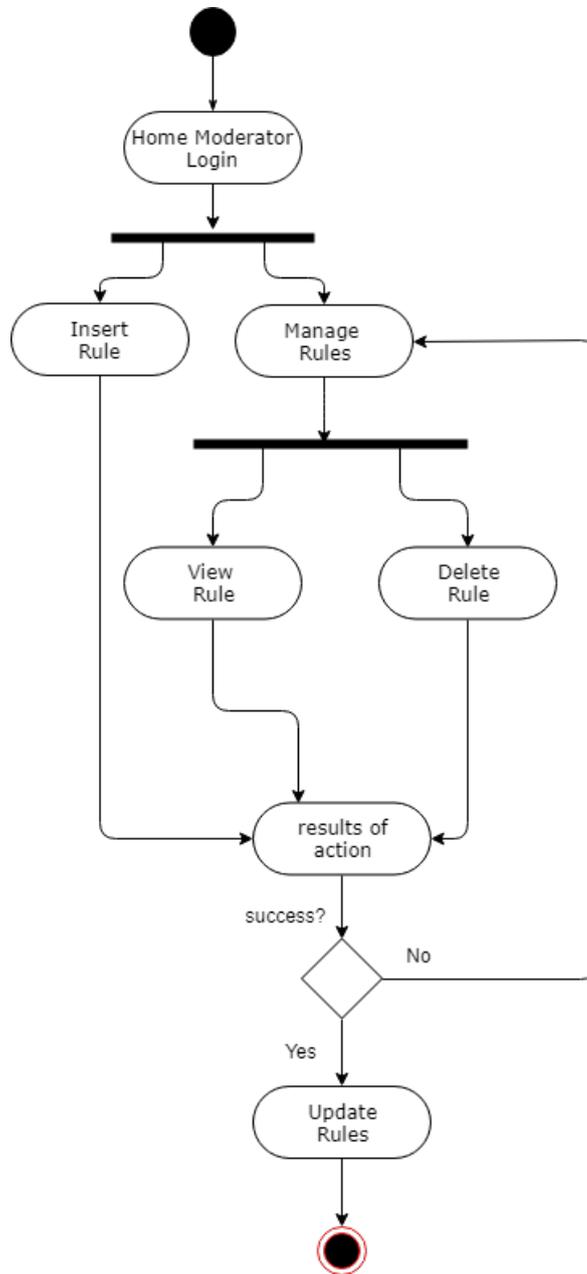


Figure 3.12: Home Moderator Rule Management Activity Diagram

Apart from the services that we presented above, Home Moderator has also access to the room and device logs that display the history of his actions and when a change in temperature, humidity and luminosity occurred.

Resident

Device Management Service

1. Resident logs into the system.
3. Selects the View Devices option from the user interface and a table with all the home's devices appears. Two operations are available:
 - 3.a. Operate Device: If the Resident has been granted authorization to operate the specific device type, an interface of the device is shown to the user (like a controller). If he successfully submits new operation data, the device entity is updated and a signal is send to the home.
 - 3.b. View Device: Device's attributes are presented to the Resident.

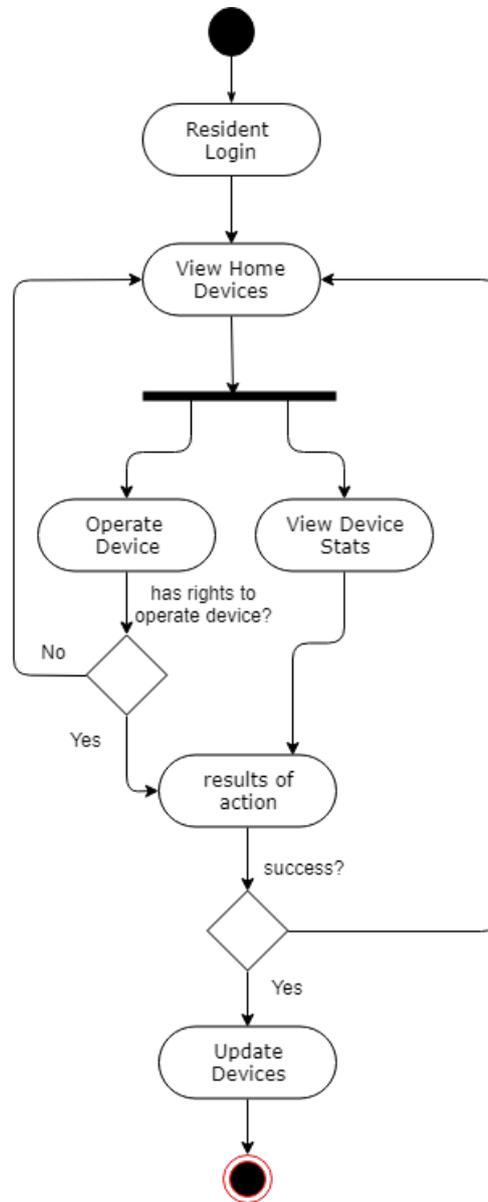


Figure 3.13: Resident Device Management Activity Diagram

Also, a Resident can view all the home's rooms and the log files that keep track of his device operation history.

3.2.3 System's flowchart

The system's overall functionality is illustrated in the diagram below. We can also see the input that is required from the insert and edit forms that the User Interface serves to the users.

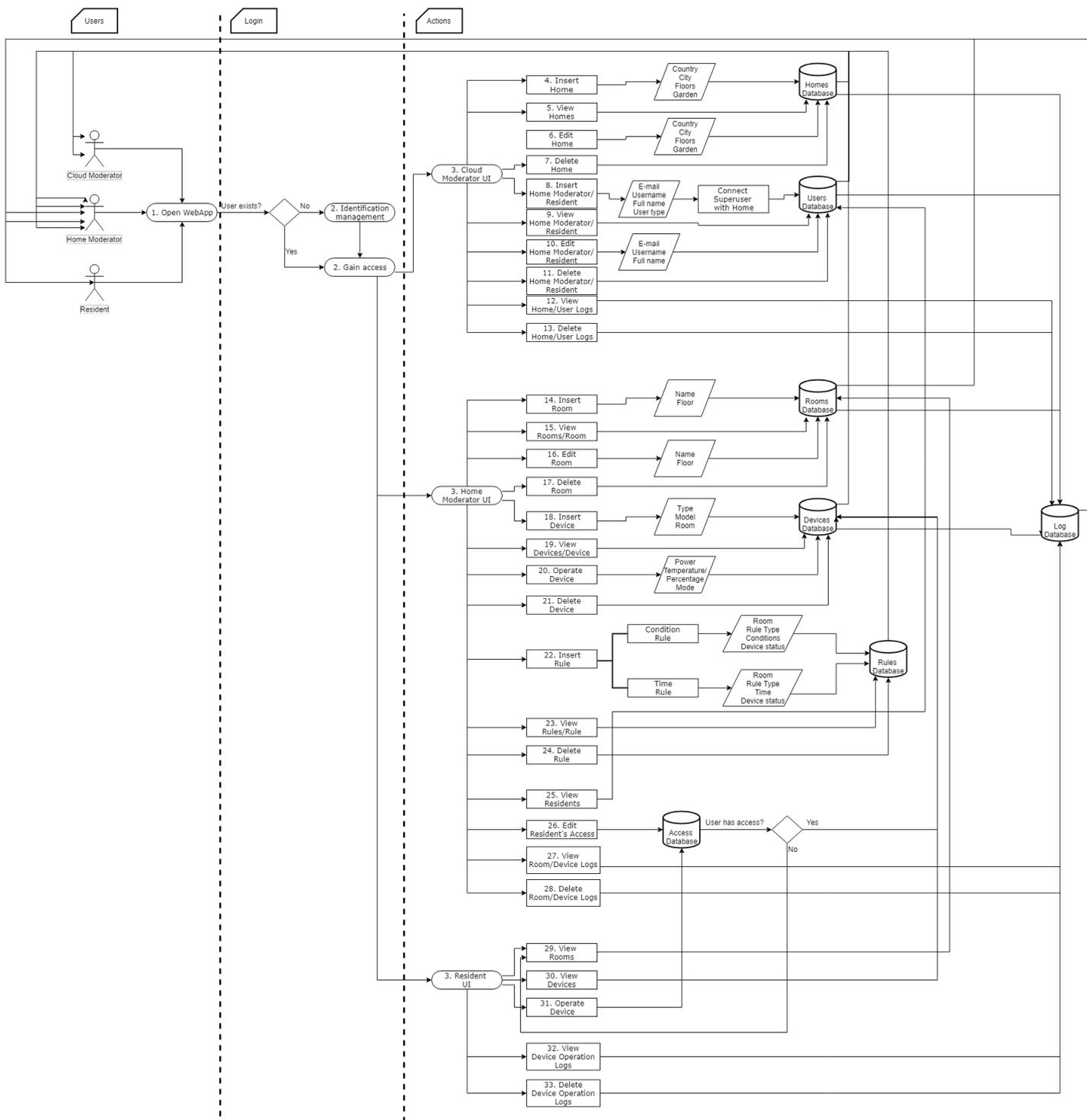


Figure 3.14: System Flowchart

3.2.4 Class Diagram

The UML Class Diagram describes the system's classes with their attributes as well as their functions and the relations between them. In the diagram below, we distinguish 10 main classes:

- **User:** A superclass that gets extended by the three user types our system supports; **Cloud Moderator**, **Home Moderator**, **Resident**. These classes extend user's e-mail, username, password, name, surname attributes and they defer in the methods they implement as well as the operation rights they have.
- **Cloud Moderator:** Is responsible for the home and user management of the system. Can insert/view/edit/delete homes and users as well as view/clear home or user logs.
- **Home Moderator:** Is responsible for his home management. Can insert/view/edit/delete rooms, devices, rules, change the Residents' device operation rights and view/clear room or device logs.
- **Resident:** Can view houses rooms and devices but can only operate devices that the Home Moderator gave him access to.
- **Home:** Contains users, rooms and devices.
- **Room:** Contains devices. Every room has three main attributes; temperature, humidity, luminosity.
- **Rule:** A superclass that gets extended by two rule types; conditional and time rule, with attributes: roomId, deviceId, mode, device temperature/percentage (for example switch on the air condition at 22oC or switch on a lamp light at 80
- **Conditional Rule:** As the name suggests, this rule is executed when one or two conditions are met.
- **Time Rule:** This rule is executed in a specific time instance.
- **Device:** There are nine device types that our system supports: Air Condition, Oven, Refrigerator, Washing Machine, Dishwasher, Lamp Light, Blinds, TV and Alarm.
- **Alert:** When a device malfunctions and it's status changes from normal to error, or if an alarm is triggered, an alert is been raised and send to the user.
- **Notification:** When a rule is executed, a notification is sent to the user.

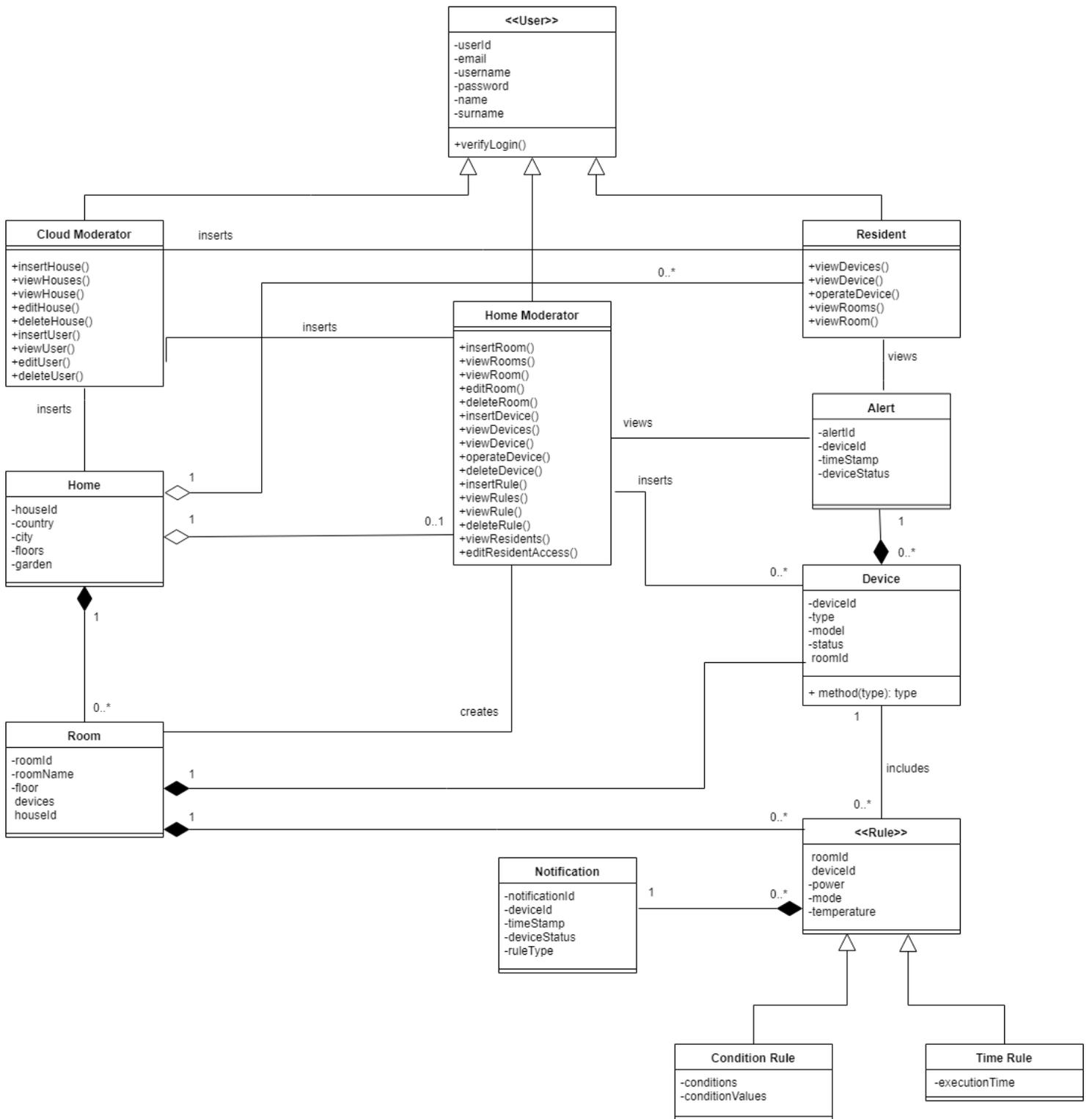


Figure 3.15: Class Diagram

3.2.5 Architecture Diagram

The Architecture Diagram illustrates the services that compose our system as well as the way they communicate.

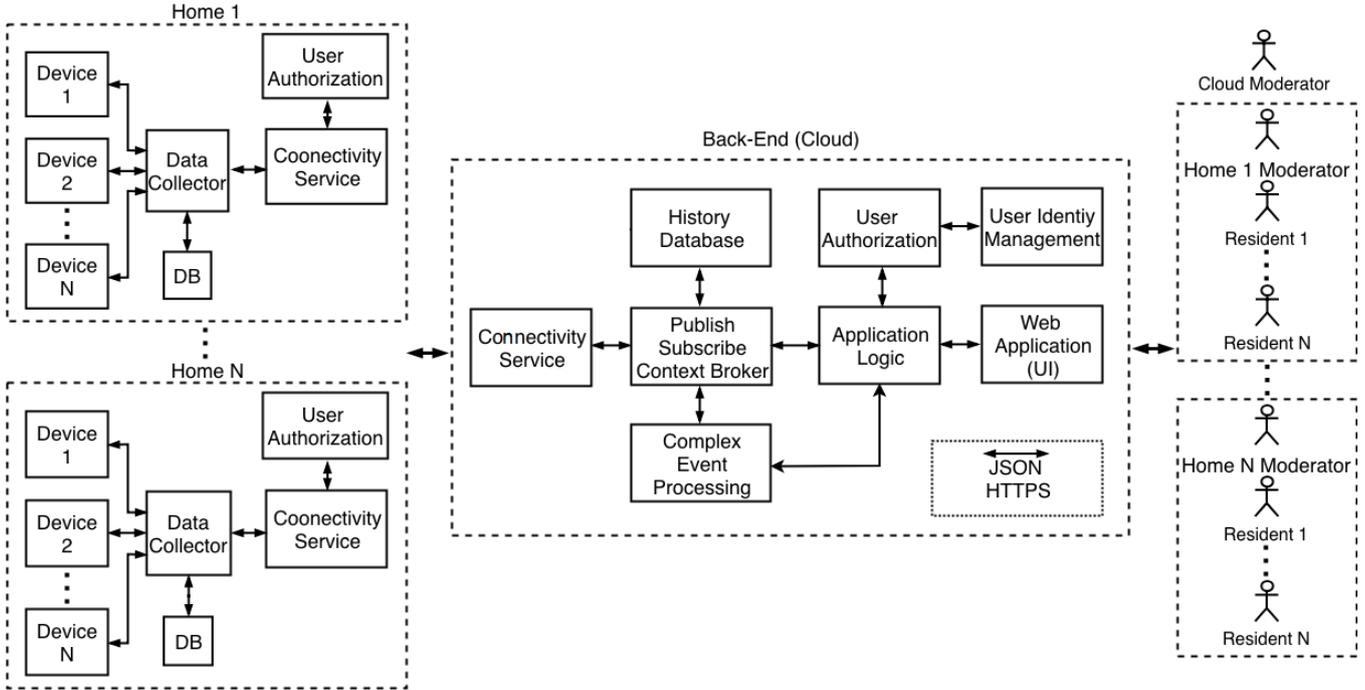


Figure 3.16: Architecture Diagram

Front-end

As front end we define the data producing mechanisms. They can be gateways, mobile devices, computers, sensors, a fog network etc. In our system we assume that data is produced by devices and sensors that are installed in homes and is send to the back-end with the use of a gateway. Each home contains rooms and each room contains devices. We consider that every room has a temperature, humidity and luminosity sensor and their measurements are send to the Data Collector. Also, each device produces its own data (power, operating temperature, mode etc) that is also send to the Data Collector. Every one minute, this gathered data is send to the back-end.

Data Collector: A gateway that gathers all the data the home’s devices and sensors produce as well as the data that’s been received from the back-end. It sends the collected sensor and device data to the back-end on a time period or distributes commands on the home’s devices. In our system’s implementation, we assume this time period to be one minute. In the data collector’s database, apart from all the collected data, the home’s authorized users ids are stored, in order to be used from the user authorization service as well as the SSL certificate for the HTTPS communication.

Connectivity Service

- Protocol adaptation: In the REST environment of our system, services communicate via HTTPS protocol. Home's devices on the other hand, use other protocols to communicate with the gateway or between them (WiFi, Bluetooth, ZigBee etc). So, in order for the data to be send from a device or a sensor to the back-end or vice versa and be accessible without causing interoperability problems, a protocol adaptation must be applied.
- Encryption/Decryption: The data received from the devices and sensors are encrypted with the use of a private ssl key. The data received from the back-end is decrypted with the back-end's public key.
- Connection establishment: After the protocol adaptation, an HTTPS connection is established between the front-end and the back-end.
- Data transfer: The encrypted data is transferred to or from the back-end

User Authorization This service's role is to prevent unauthorized users access into the home. Uses the OAuth 2 mechanism to confirm a user's authorization. It will be further explained in chapter 4.1.5.

Back-end

Back-end is responsible for managing, processing and storing the data that's been send from the front-end as well as controlling the operations of our system. The services that compose the back-end are:

Application Logic (Service Endpoint: <https://147.27.60.196:8448>)

The orchestrating service of the system. It controls the application's data flow and the operations' order of execution. When a request is generated by a user, the Application Logic decides to which service it will be forwarded to in order to be completed. When this happens, a response is returned that informs the Application Logic if the request was executed successfully or not. It is connected with the User Interface, User Authorization, Context Broker, Log Database and Connectivity services that will be described bellow. Also, it creates notifications when a rule is executed and alerts when a device presents a malfunction or an alarm triggers. These notifications and alerts are retrieved from the Web Application.

Use Identity Management (Endpoint: <https://account.lab.fiware.org>)

In our system we use a public instance of Fiware's Keyrock Identity Management. Keyrock IdM is responsible for user authentication and access authorization. First of all, a user, in order to use our application, must have an account in Fiware. After he creates the account, the Cloud Moderator must add him to the authorized user list of the application and define its role. Then, when a user tries to login to our application using the User Interface, he is redirected to Keyrock, where he inputs the username and password he uses to login to Fiware. If the login attempt succeeded and the user is authorized to use the application, he is redirected to the main page of the User Interface we created and he is provided with the OAuth2 token that was generated by Keyrock. This token is stored in the PHP session and is used as a header in the outgoing requests.

User Authorization This service ensures that every user action is "legal". When a user makes a request, he must be authorized to be able to execute it. For example, if a Home Moderator tries to view all homes of the system, his request must be rejected because this is Cloud Moderator's operation. This is achieved with the User Authorization service. Part of this service is a MongoDB database that contains all system's users with their attributes as well as their home's id. Once a request reaches Application Logic, the token is retrieved from it's header and its send to the User Authorization service. From there, it is send to Identity Management Service (Keyrock) who responds with the user's credentials. These credentials are used for the authorization. This will be fully explained in the next chapter.

Web Application (UI) (Service Endpoint: <https://147.27.60.196:80>)
Allows system's operation and control from the users through a user interface. Once a user logs into the system, the OAuth2 token that is created from Keyrock is saved into the user session. When a user uses the interface and a request is created (e.g. view room), this token is added to the request so the authorization service can use it and decide if the request will be forwarded or denied. Also, every one minute, Web Application asks the Application Logic for notifications or alerts that might have been produced and displays them to the users.

Publication/Subscription Service³ (Endpoint: <https://147.27.60.71:1026>)
This service mediates between consumer producers (devices, sensors) and the context consumer applications (e.g. user that reads a room's temperature using the User Interface). It's responsible for registering context producer applications (e.g. room temperature), updating context information (e.g. send updated temperature) and notifying when changes on context information takes place (e.g. change in room's temperature). For example, when a Home Moderator inserts a new room "kitchen" in his home, he issues a POST request that contains the room's data in JSON format and a new entity with type "room" is created into the Publication/Subscription Service. In addition, a subscription about this rooms is created. This subscription is used for the creation of the log entities and will be explained in the History Database service. When the user wants to retrieve data regarding this room, he issues a GET request to the Publication/Subscription Service and he responds by sending the specific entity in a JSON format. Every minute, each home sends it's rooms values to the back-end, Connectivity Service retrieves them and sends them to the Publication/Subscription Service. If they are different from the already existing ones, they are updated. For our system's needs, we used Fiware's Orion Context Broker as the publication/subscription service.

Complex Event Processing (CEP, Endpoint: <https://147.27.60.196:8888>)
Responsible for the conditional rules management. CEP receives the notifications that Orion Context Broker sends when one or more attributes change (e.g. temperature) and compare the notification's values with the values of the rule. If the conditions are met, CEP sends a request to Application Logic informing it to operate a device in a specific home. CEP will be fully explained in chapter 5.1.6.

³Services In Cloud And Fog Computing - Euripidis G.M. Petrakis

History Database (Service Endpoint: <https://147.27.60.196:7777>)

Our system keeps log entries for each user group on the following cases:

- Cloud Moderator
 - Home Insert/Edit/Delete
 - User Insert/Edit/Delete

 - Home Moderator
 - Room Insert/Edit/Delete/Attribute value change
 - Device Insert/Operate/Delete/Status change

- Resident
 - Device Operate

In the device logs are also stored the alerts that are generated when a device malfunctions (status different than Ok) or when an Alarm is triggered. All logs contain a timestamp of the action that occurred on a specific entity. This timestamp is extracted from the entity's objectId. An example of a log view can be seen in the figure below.



Time	Name	Floor				User	Action
Mon Nov 26 16:54:05 UTC 2018	living room	1	22	70	70	gmyrizakis2	INSERT
Mon Nov 26 16:54:18 UTC 2018	kitchen	1	22	70	70	gmyrizakis2	INSERT
Mon Nov 26 16:54:28 UTC 2018	bedroom	2	22	70	70	gmyrizakis2	INSERT

Figure 3.17: View Room Logs

As we mentioned above in the CEP description, when an entity is created, a subscription for this entity is created too. This way, when a change occurs, a notification is sent to the History Database service, where is stored as a log entity.

Connectivity Service (Service Endpoint: <https://147.27.60.196:8282>)

It's functionality is the same as the Front-End's home Connectivity Service.

Chapter 4

System Implementation

In this chapter we are going to further explain our system's components and services.

4.1 FIWARE Platform

In order to create our application, we used FIWARE as our cloud provider. Our system's back-end is implemented as a composition of eight main services (Figure 3.17):

1. User Identity Management
2. User Authorization Service
3. Application Logic
4. Web Application (UI)
5. Publish/Subscribe Service
6. Complex Event Processing
7. History Database
8. Connectivity Service

For the **back-end**, we used three virtual machines (VMs); two of them we created ourselves and the third is a shared VM installed in a remote FIWARE node. The first one, with floating IP 147.27.60.196 hosts all the services we created; the User Authorization, Application Logic, Web Application, Complex Event Processing, History Database and Connectivity services. The second one, with floating IP 147.27.60.71, hosts the publish/subscribe service.

For the implementation of User Identity Management and Publish/Subscribe service we used two services from the Fiware Catalog; Keyrock Identity Management and Orion Context Broker. The rest services, except the Web Application, were implemented with the use of JavaEE and the Spring Framework. Every Java service has embedded a Tomcat server and as its end-point we define a specific port in the VM. The Web Application service, created with PHP and

HTML, uses an Apache server and communicates with the Application Logic with CURL. The services' endpoints are presented in figure 4.1.

Regarding the front-end, we crated a VM with floating IP 147.27.60.61, that simulates the homes. It contains a MongoDB, where the devices and the rooms attributes are stored. The home's Connectivity Service that listens in port **8668**, retrieves all incoming data.

Service	Port
Web Application	80
Application Logic	8448
Complex Event Processing	8888
History Database	7777
Connectivity Service	8282

Figure 4.1: Services End-Points for 147.27.60.196

4.1.1 Keyrock Identity Management¹

Keyrock's IdM is the central component that provides a bridge between IdM systems at connectivity-level and application-level that manages:

- Users' and organizations' access to networks, services and applications by having a registered account in Keyrock
- Secure authentication from users to devices, networks, services
- Roles and permissions to manage authorization of users and organizations
- Foreign services authorization to access personal data
- Users' profiles
- Privacy-preserving disposition of personal data
- Single Sign-On (SSO) to service domains
- User authentication using their OAuth credentials (id and secret)

Keyrock complies with the OAuth2 standard described in RFC2616². In order to use our application, a user must be registered in Fiware lab and be authorized for the application.

¹<https://catalogue-server.fiware.org/enablers/identity-management-keyrock>

²<https://www.ietf.org/rfc/rfc2616.txt>

OAuth 2.0 ³

OAuth 2.0 is the industry-standard protocol for authorization and provides specific authorization flows for users, applications and devices. The main idea is to enable applications to obtain limited access to user accounts on a HTTP service such as Facebook, GitHub, our application without exposing the user's sensitive information. The OAuth roles are:

- Resource Owner: the user who authorizes an application to access their account.
- Client: The application that wants to access the user's account.
- Resource/Authorization Server: The resource server hosts the protected user account and the authorization server verifies the identity of the user, then issues access tokens to the application.

The figure below demonstrates how the OAuth roles interact with each other.

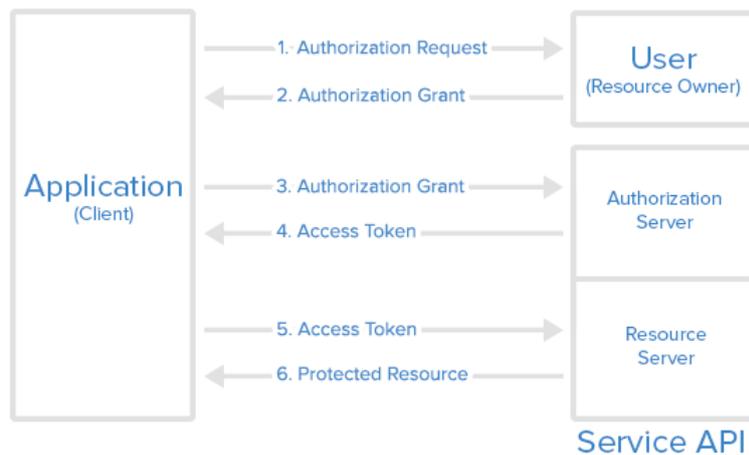


Figure 4.2: OAuth 2.0 Protocol Flow

³<https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>

In order to create our application, we first created an account in FIWARE Lab (Figure 5.2).

The screenshot shows the FIWARE Lab registration page. At the top is a navigation bar with the FIWARE Lab logo and links for Cloud, Store, Mashup, Data, Account, and Help&Info. The main content area is split into two columns. The left column features a 'FIWARE Lab' header, a brief description of the lab as a working instance for experimentation, and a 'Sign up' button. Below this are four teal buttons with white text and right-pointing arrows: 'Need Help?' (with a question mark icon), 'Our GEs' (with a document icon), 'FIWARE Lab nodes' (with a gear icon), and 'FIWARE Academy' (with a graduation cap icon). The right column is titled 'Registration' and contains a form with the following elements: a 'Username' text input field, an 'E-mail' text input field, a checkbox for 'I have Gravatar and want to use it for my avatar.', a 'Password' text input field with an eye icon, a 'Password (again)' text input field with an eye icon, a reCAPTCHA widget with the text 'I'm not a robot', a checkbox for 'I accept FIWARE Lab Terms and Conditions', and a dark blue 'Sign Up' button. At the bottom of the registration form are two links: 'Forgot password' and 'Didn't receive confirmation instructions?'.

Figure 4.3: Fiware Lab Sign Up Page

After that, we had to register our application by providing application's name, description, URL and callback URL (the url that the user is redirected after the OAuth flow is finished). We created the Smart Home application (Figure 5.3).

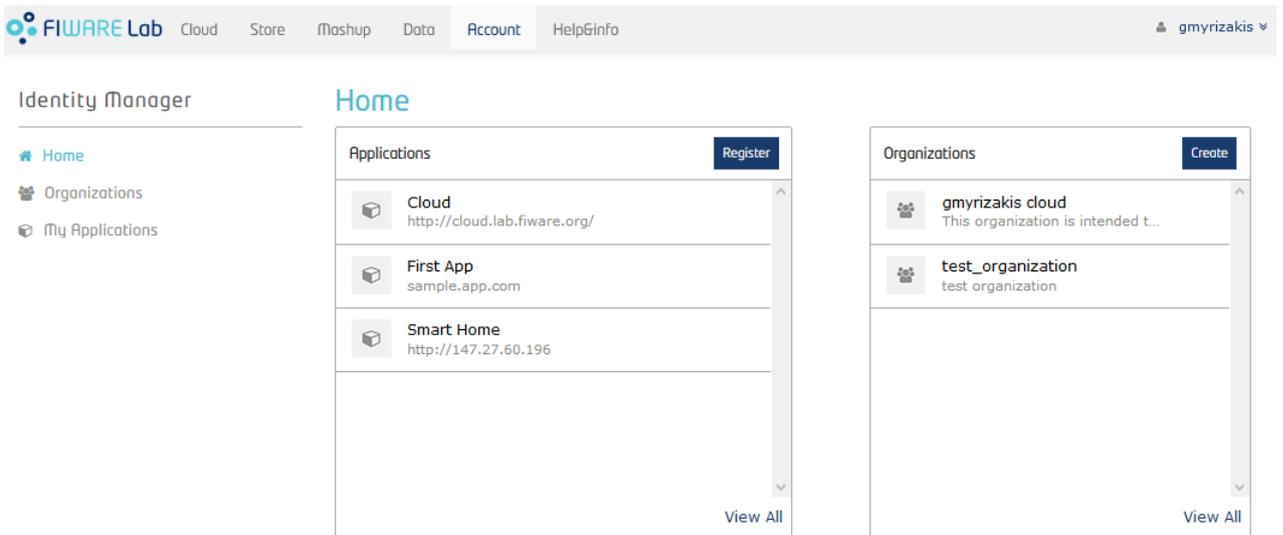


Figure 4.4: Fiware Lab Application Register

In order for a user to be able to use our application, he must be inserted to the application's Authorized Users list. We inserted more users and we provided them with a role. Our application supports three roles:

- Provider: Predefined by Fiware Lab and has all permissions on the application. We assigned this role to the Cloud Moderator.
- Purchaser: Predefined by Fiware Lab with permissions to get and assign all public application roles. In our system this role is assigned to Home Moderator.
- User: We created this role with the same permissions as the purchaser in order to distinguish the Resident from the Home Moderator.

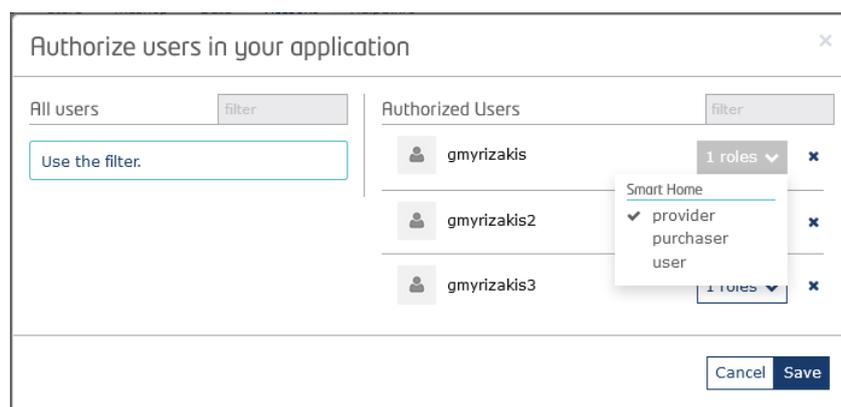


Figure 4.5: Application's Authorized User

After that, when a user wants to use our application, an OAuth 2 authentication procedure is taking place.

- User selects "Login With Fiware" button.
- An authorization code is requested from Keyrock by sending the application's client_id that is provided upon its registration (Figure 5.5).
- The authorization code is then send to Keyrock along with the header "Authorization: Basic base64(client_id:client_secret)" and the access token is issued.
- When the token is issued, it's stored in a PHP session and is used when a request is generated.

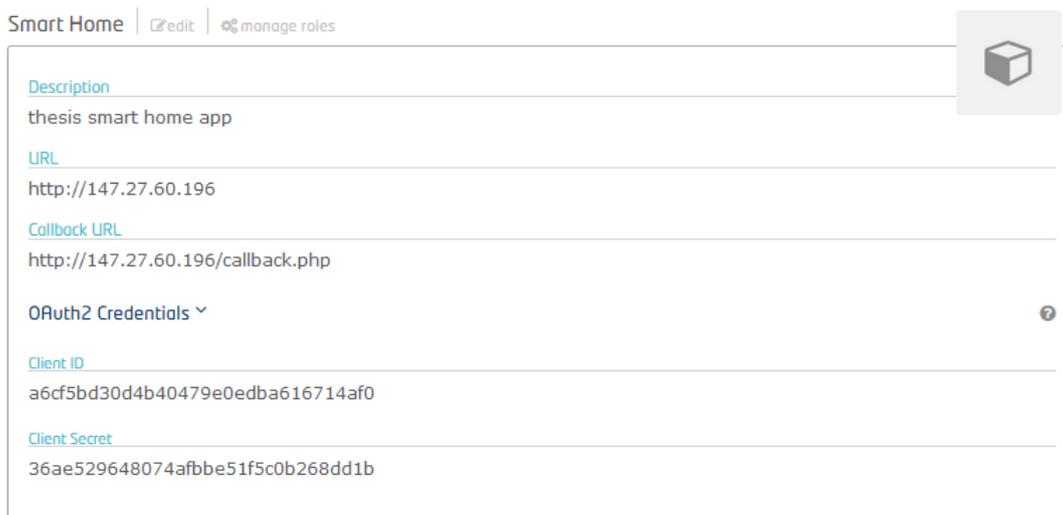


Figure 4.6: Application's OAuth2 Credentials

4.1.2 Orion Context Broker ⁴

Orion Context Broker allows the management of the entire lifecycle of context information including updates, queries, registrations and subscriptions. It is an NGSIv2 server implementation to manage context information and its availability. Using the Orion Context Broker, we are able to create context elements and manage them through updates and queries. In addition, we can subscribe to context information so when some condition occurs (e.g. the context elements have changed) we receive a notification. Orion's Context Broker basic operations are:

- GET-POST /v2/entities :Retrieve all entities-create entity.
- GET-[PUT-PATCH-POST]-DELETE /v2/entities/{entityId}: Retrieve-Update-Delete an entity.
- GET-PUT-DELETE /v2/entities/{entityId}/attrs/{attrName}: Retrieve-Update-Delete an attribute.

⁴<https://fiware-orion.readthedocs.io/en/master/>

- GET-PUT /v2/entities/{entityId}/attrs/{attrName}/value: Retrieve-Update attribute's value.
- GET-POST /v2/subscriptions: Retrieve all subscriptions-Create subscription.
- GET-DELETE /v2/subscriptions/{subscriptionId}: Retrieve-Delete subscription.

In our system, when an entity is created in Orion (e.g. room, user), a subscription on this entity is created too. When a change occurs to the entity, this subscription is notifying History Database service.

Orion Context Broker supports Entity Service Paths; hierarchical scopes, so entities can be assigned to a scope at creation time. Then, query and subscription can be scoped to locate entities in the corresponding scopes. In the figure below our system's service path is shown.

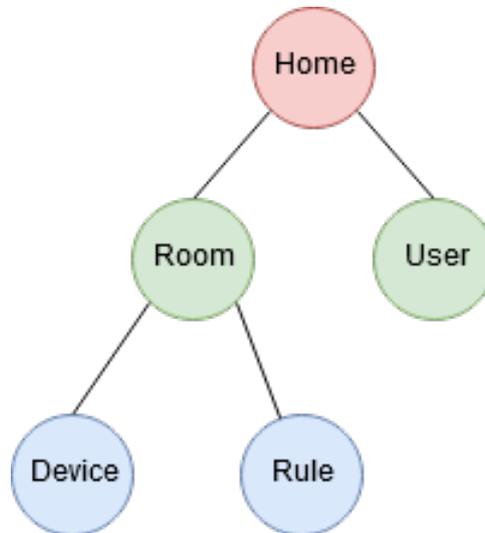


Figure 4.7: System's Service Path

This service path is used when a request is issued and is added in the header with the key "Fiware-ServicePath". In the following images we can see the data that is sent to Orion Context Broker when an entity is created, the subscription that is created and the data we receive when we issue a GET entity request. We can also observe that in the POST and GET room request the "Fiware-ServicePath: homeId" was added as a header in order to follow the system's service path.

```

Request:
POST https://147.27.60.71:1026/v2/entities \
-H "Fiware-ServicePath: 5bfc2052ea636e15c973a7bd"

Data:
{
  "id": "5c0281f07592c31084a217e2",
  "type": "room",
  "name": "kitchen",
  "floor": 1,
  "homeId": "5bfc2052ea636e15c973a7bd",
  "temperature": 22,
  "humidity": 70,
  "luminocity": 70
}

```

Figure 4.8: POST room JSON Data

```

Subscription:
{
  "description": "roomSub",
  "expires": "2020-04-05T14:00:00.00Z",
  "id": "5c0280ee925d56dd803c5255",
  "notification": {
    "attrs": [],
    "attrsFormat": "keyValues",
    "http": {
      "url": "https://147.27.60.196:7777/roomSubs"
    },
    "lastNotification": "2018-12-01T12:39:10.00Z",
    "lastSuccess": "2018-12-01T12:39:10.00Z",
    "timesSent": 1
  },
  "status": "active",
  "subject": {
    "condition": {
      "attrs": [
        "name",
        "floor",
        "temperature",
        "humidity",
        "luminocity"
      ]
    },
    "entities": [
      {
        "id": "5c0281f07592c31084a217e2",
        "type": "room"
      }
    ]
  }
}

```

Figure 4.9: Room Subscription

```

Request:
GET https://147.27.60.71:1026/v2/entities/5c0281f07592c31084a217e2 \
-H "Fiware-ServicePath: 5bfc2052ea636e15c973a7bd"

Response:
{
  "id": "5c0281f07592c31084a217e2",
  "type": "room",
  "name": {
    "metadata": {},
    "type": "Text",
    "value": "kitchen"
  },
  "floor": {
    "metadata": {},
    "type": "Number",
    "value": 1
  },
  "homeId": {
    "metadata": {},
    "type": "Text",
    "value": "5bfc2052ea636e15c973a7bd"
  },
  "temperature": {
    "metadata": {},
    "type": "Number",
    "value": 22
  },
  "humidity": {
    "metadata": {},
    "type": "Number",
    "value": 70
  },
  "luminocity": {
    "metadata": {},
    "type": "Number",
    "value": 70
  }
}

```

Figure 4.10: View Room JSON Data

An "?options=keyValues" parameter can be added at the end of the request URI and Orion will return the room's attributes in a simple format, as in figure 5.3.

Apart from Keyrock IdM and Orion Context Broker, the rest services were created by us.

4.1.3 Web Application(UI)

We created the Web Application(User Interface) service using PHP, HTML, CSS, Javascript, JQuery and AJAX tools. After successfully logging into the application, the token that's issued by Keyrock IdM is saved to the PHP session. All operations available to the user are displayed with a menu in the User Interface. When he selects an option from the menu, a request is generated. After the token is added to its header, it's send to the Application Logic. We used the CURL library of PHP in order to handle the requests. If the user is autho-

alized for that request, it is executed and a response is returned. A response is displayed using HTML and is managed by Javascript and CSS. For example, a Home Moderator wants to view a room. He logs into the system and chooses the Rooms - Manage Rooms option. A GET /homes/homeId/rooms request is issued and if it is executed successfully, a response with all home's rooms in JSON format is returned. Rooms are inserted in a HTML table and are formatted with CSS rules (Figure 5.10). Next to every room entity are three buttons that execute Javascript functions (View, Edit, Delete) on a click event. Home Moderator clicks the view button and again a GET /homes/homeId/rooms/roomId request is issued with response the room's attributes as well as its devices.

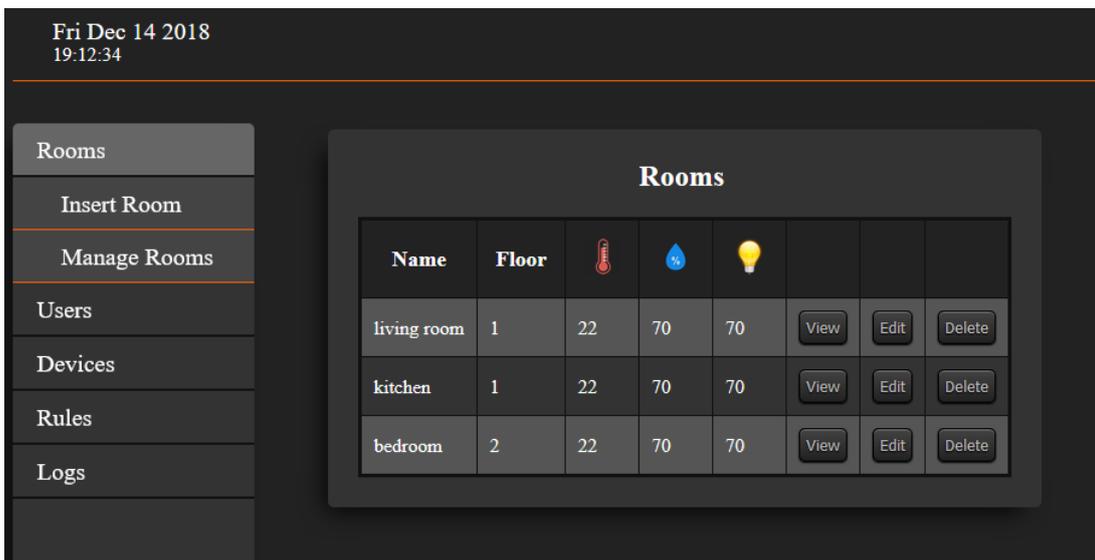


Figure 4.11: View Rooms

Web Application, with the execution of a jQuery ajax function, polls the Application Logic every one minute for notifications and alerts. If a notification/alert is created, it's written in a database and it's not deleted until the user clears it using the interface. **We created the rest of the system services using the Java's Spring Boot framework.**

4.1.4 Application Logic

Requests from the Web Application and Complex Event Processing are directed to the Application Logic. A Controller class is processing the requests by mapping them onto methods with the use of annotations. For example, the annotation "@RequestMapping (value = "/homes/{homeId}/rooms/{roomId}, method = GET)" above the method "public String viewRoom()", binds this method with the GET <https://147.27.60.196:8448/homes/5bfc2052ea636e15c973a7bd/rooms/5c0281e37592c31084a217e1> request that arrives to Application Logic. If the request is user generated, user's authorization to issue such a request must be first checked. This functionality is covered by the User Authorization service that is part of the Application Logic.

4.1.5 User Authorization Service

All requests that comes from the Web Application to the Application Logic, except the notification/alarm polling, are user issued. In order to ensure that each owner's data is safe into the application, we must make sure that no unauthorized user can access it. This is accomplished with the implementation of the User Authorization Service. Every user's request is relevant to his home and contains the homeId into its URI(e.g. Insert room, View Device etc). This means that we have to check that the user doesn't try to access another home's data. When the Cloud Moderator inserts a user in the system, he links him with a home. We created a MongoDB collection that contains all system's users. In the figure bellow we can see an example user entity.

```
User Entity
{
  "_id" : ObjectId("5bfc21ffea636e15c973a7c0"),
  "homeId" : "5bfc2052ea636e15c973a7bd",
  "email" : "gmyrizakis@hotmail.com",
  "username" : "gmyrizakis2",
  "firstName" : "George",
  "lastName" : "Myrizakis",
  "role" : "superUser",
  "access" : "111111111"
}
```

Figure 4.12: User Entity of Users Collection

We observe that the user's homeId is included in the user entity. Also, the "username" field contains the username that the user uses to log in the application. This is the user's id in Keyrock IdM and of course it's unique. When a request arrives to Application Logic, the User Authorization Service retrieves its token from the header and it sends it to Keyrock IdM requesting the user's credentials with the GET http://account.lab.fiware.org/user?access_token={accessToken} request. If the token is valid, user's information is returned. User Authorization Service gets the user's id and checks if the user exists in the Users collection. If he exists, it retrieves his homeId and compares it to the homeId that is in the request's path. If it's the same, the request is executed. If the token is not valid or the user's homeId is different that the request's homeId, the request is rejected. The following sequence diagram demonstrates the "GET room" example.

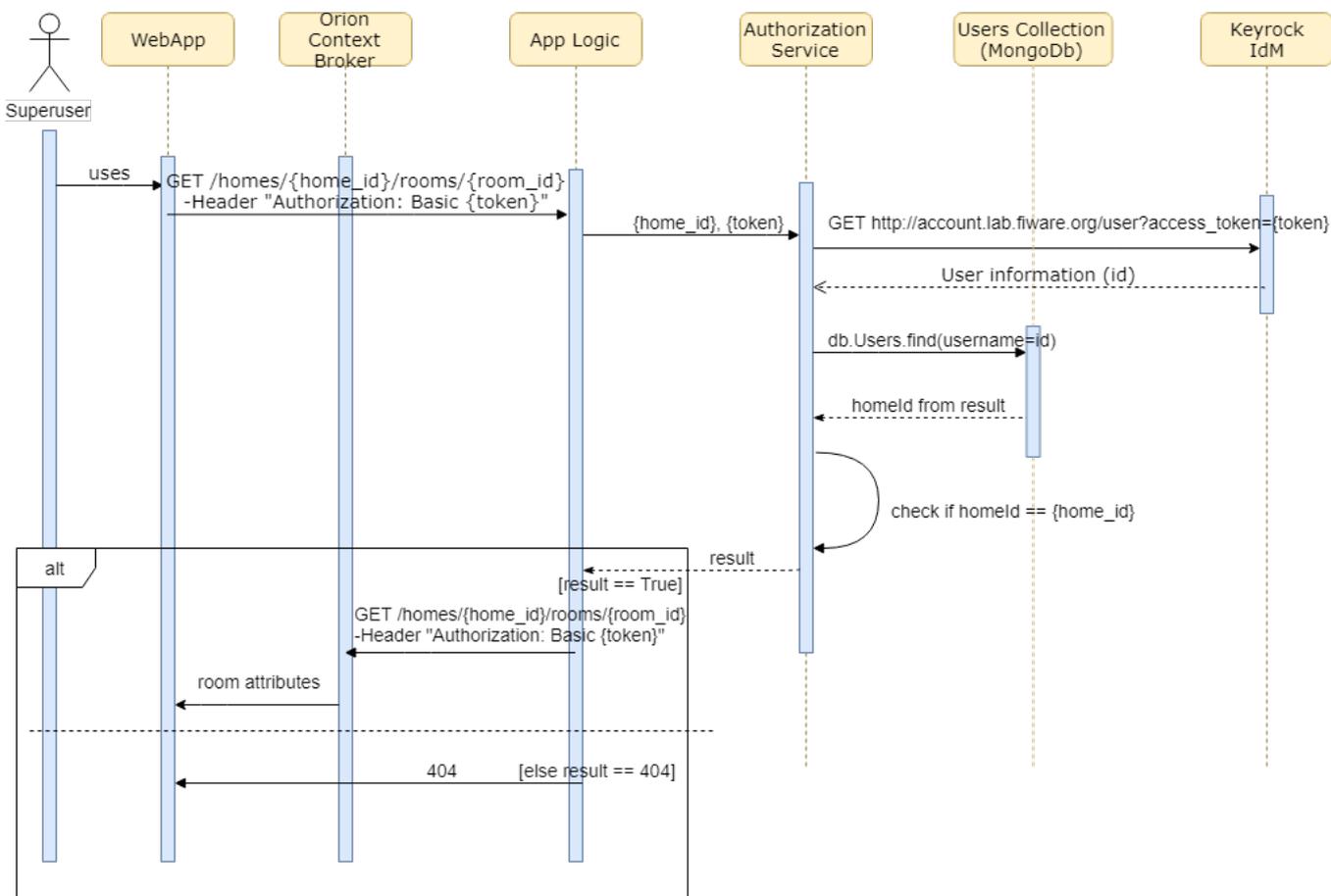


Figure 4.13: User Authorization Sequence Diagram - GET Room

4.1.6 Complex Event Processing

Responsible for the conditional rules management. The way we created CEP, a rule can have two conditions at max (e.g. *temperature* > 20 AND *humidity* > 90) and only one device can be operated per rule. When a rule regarding a room is created, a Context Broker subscription that contains the conditions, the conditions' values, the device that must be operated as well as the device operation values (e.g. mode, operation temperature), is created. A subscription is represented by JSON with the following fields (Figure 5.13):

- id: Subscription unique identifier that is automatically created at creation time.
- description: A free text that is inserted by the user to describe the subscription.
- expires: Expiration date of the subscription

- notification: An object that describes the notification to be send when the subscription is triggered. Includes the following sub fields
 - attrs: These are the room’s attributes that will be send with the notification. Being empty means that every attribute will be send. If for example, “temperature” was in attrs (“attrs”: [“temperature”]), only the temperature value would be send with the notification.
 - attrsFormat: How values are represented. If is set as keyValues, the notifications are send in a simple format (e.g. “temperature”:22).
 - http: Shows the URL that the notification will be send.
 - lastNotification: Timestamp of the last time the notification was sent.
 - lastSuccess: Timestamp of the last successful notification.
 - metadata: We use this field to store the rule information that will be retrieved by CEP. One condition rule metadata:
 - home Id
 - device Id
 - device Type
 - power: true = power on, false = power off
 - device operation temperature (e.g. air condition at 16oC)
 - mode (e.g. air condition in cool mode)
 - condition value (e.g. temperature 30)
 - condition operator (e.g. greater)
 - Two conditions rule metadata:
 - home Id
 - device Id
 - device Type
 - power: true = power on, false = power off
 - device operation temperature (e.g. air condition at 16oC)
 - mode (e.g. air condition in cool mode)
 - condition 1 value (e.g. temperature 30)
 - condition 2 value (e.g. humidity 90)
 - condition 1 operator (e.g. greater)
 - condition 2 operator (e.g. less)
 - conditions relation (AND/OR)
- Subject: contains the following sub fields
 - condition: The conditions that trigger the notifications. It contains the following sub field:
 - attrs: The names of the attributes that trigger the notification. The order of their appearance follows the order of the condition values in the metadata of the notification field.
 - entities: The Context Broker entity that the rule was created for. Has two sub fields:

- id: id of the entity
- type: type of the entity (e.g. room)
- throttling: Minimal period of time in seconds which must elapse between two consecutive notifications.

```

Rule Subscription:
{
  "description": "Condition Rule",
  "expires": "2020-04-05T14:00:00.00Z",
  "id": "5c0d70a6925d56dd803c525f",
  "notification": {
    "attrs": [],
    "attrsFormat": "keyValues",
    "http": {
      "url": "https://147.27.60.196:8888/eventHandler"
    },
    "lastNotification": "2018-12-09T19:44:38.00Z",
    "lastSuccess": "2018-12-09T19:44:38.00Z",
    "metadata": [
      "5bfc2052ea636e15c973a7bd",
      "5c0284177592c31084a217e4",
      "AirConditioner",
      "true",
      "24",
      "Heat",
      "18",
      "90",
      "less",
      "greater",
      "and"
    ],
    "timesSent": 1
  },
  "status": "active",
  "subject": {
    "condition": {
      "attrs": [
        "temperature",
        "humidity"
      ]
    },
    "entities": [
      {
        "id": "5c0281e37592c31084a217e1",
        "type": "room"
      }
    ]
  },
  "throttling": 5
}

```

Figure 4.14: Example Of a Two Conditioned Rule Subscription

Every time a change occurs in the room's attributes, a notification is sent to CEP that has two fields (Figure 5.14):

- data: contains the subscribed entity's id and type, as well as the attributes that changed and triggered the notification.
- subscriptionId

CEP, afterwards, using the subscriptionId from the notification, with a GET request on the Context Broker, retrieves the subscription (Figure 5.13) and compares the metadata attribute values with the attribute values of the notification.

```

Notification:
{
  "data": [
    {
      "id": "5c0281e37592c31084a217e1",
      "temperature": 17.5,
      "humidity": 95
    },
    {
      "type": "room"
    }
  ],
  "subscriptionId": "5c0d70a6925d56dd803c525f"
}

```

Figure 4.15: Example of Notification Send to CEP

If the conditions are satisfied, CEP sends a request to Application Logic to act according to the rule's instructions.

Example - Analyzing figure 5.13, the following "query" results. IF "temperature" is "less" than "18°C" "AND" "humidity" is "greater" than "90%" in the room with id:"5c0281e37592c31084a217e1" of home with id:"5bfc2052ea636e15c973a7bd" THEN "power ON" "Air Condition" with id:"5c0284177592c31084a217e4" and set it to work at "24°C" in "Heat" mode. We can see in figure 5.14 that the room's temperature is 17.5°C and humidity is 95%, thus, the conditions are satisfied. CEP then sends a request to Application Logic that informs it to switch on the air condition in the specific room on Heat mode, at 24°C. After that, if the device was operated successfully, Application Logic deletes the rule subscription and a notification is created saying that the rule was successfully executed. We can view this functionality in Figure 4.15.

Sequence Diagram-Complex Event Processing A sequence diagram demonstrates the sequence of interactions and messages that occur between the users and the objects in a system. In the figure below we can see the Complex Event Processing Service's (CEP) sequence diagram. Initially, Home Moderator uses the web application to create a rule. With an HTTPS Post call, the json data is passed from the UI to the Application Logic, which sends a subscription creation request to Orion Context Broker. When the home sends it's rooms values (temperature, humidity, luminosity) to the back end, CEP checks the rule from the Publish/Subscription Service and compares the home's values with the subscription's values. If the conditions are met, CEP sends a confirmation to Application Logic which sends a device operation request to the home.

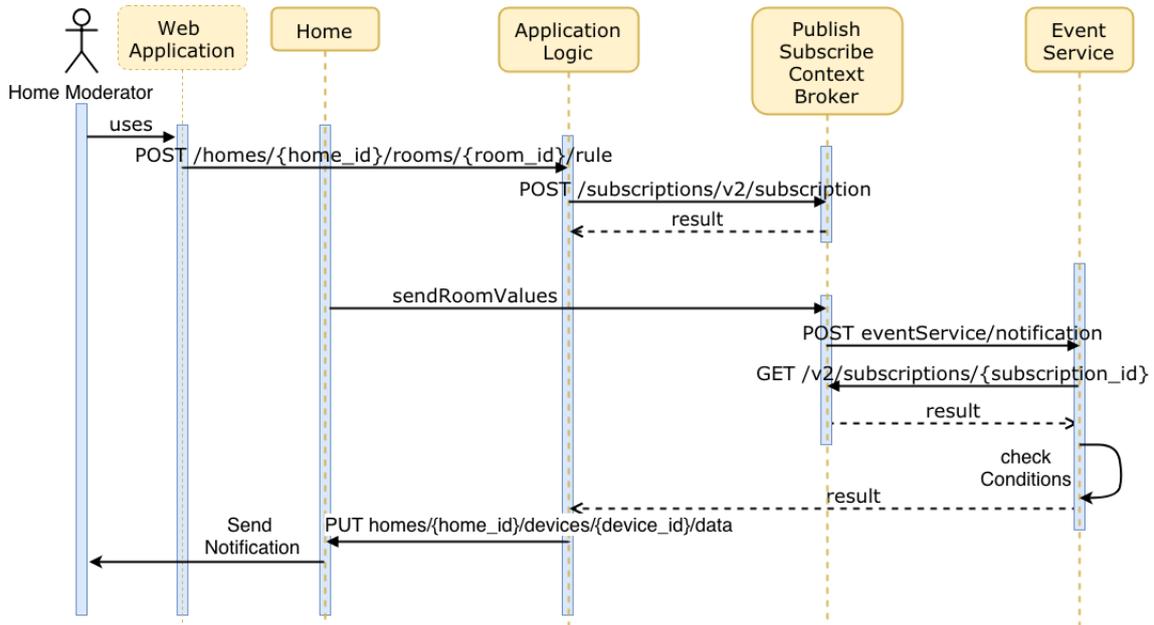


Figure 4.16: Complex Event Processing Sequence Diagram

4.1.7 History Database

We created this service using MongoDB in order to keep history of users' actions and data send from homes to the back-end. It listens for notifications Orion Context Broker sends and when it receives them, creates a log entity in the database. Logs are derived in categories and are accessible from specific user types. Generally, Orion sends notifications only when something changes to an entity. For example, when the temperature of a room changes, a notification will be send. On the other hand, if a home sends the temperature of the room and it's the same as the previous one, no change will take place. This means that a log entity will be created when something is: Created, edited, operated and deleted. The list bellow shows which log types are accessible from each user type.

- Cloud Moderator: Home, User Logs.
- Home Moderator: Room, Device Logs.
- Resident: Device Logs.

Logs keep information regarding the actor, the type of action, the time it took place as well as the entity it refers to. The following figure illustrates the form of a device log entry.

```
Room Log:
{
  "homeId": "5bfc2052ea636e15c973a7bd",
  "roomId": "5c0281e37592c31084a217e1",
  "roomName": "kitchen",
  "floor": 1,
  "temperature": 17.5,
  "humidity": 95,
  "luminocity": 70,
  "lastUser": "home",
  "action": "update",
  "notificationType": "roomLog"
}
```

Figure 4.17: Example of a Room Log entry

The "lastUser" field contains the user that issued the request. If it's "home", it means that the data was received from the front-end when the room or device data was updated. The action field can have the following values: insert, update, delete, operate and the notificationType field can be: homeLog, userLog, roomLog, deviceLog, alarmTrigger, deviceError. The last two types state the alarms and notifications that Application Logic creates when a home alarm is triggered or when a rule is executed.

4.1.8 Connectivity Service

Connectivity Service is the bridge between the back-end and the front-end. When a request that is destined for a home is generated from the Web Application, it ends up in Connectivity Service. There, the data is parsed in Json format (protocol adaptation) and then is encrypted with the use of an ssl certificate, to be send over HTTPS protocol. In our system that we have no real devices, data are simulated and are already in Json format, so this parsing isn't needed. Also, all services communicate with each other over HTTPS protocol so data is encrypted when it exits and decrypted when it enters each service.

4.1.9 Front-End User Authorization Service

It's really important for a home to be inaccessible from an unauthorized user. For example, a user that is not resident of a home must not be able to control a device of it. We created User Authorization service to filter the request and reject the unauthorized ones. As we saw in chapter 4.2.6, the Data Collector has stored in it's database the home's authorized users' ids. If a user makes a request, an OAuth2 token that is generated by FIWARE once he logs in to the system is added to the request's header. When it arrives to the home, User Authorization service receives this token and sends it to FIWARE's Keyrock Manager who is responding with the token's user credentials (id, email, role etc). User's id from the token is compared to the user ids that are stored into the Data Collector Database; if it matches with one of them, the request is executed or else, its rejected.

4.2 REST Tables

In this section we will present the REST API of each service.

4.2.1 Application Logic

The rest API of Home and User management services is displayed in the tables below.

- Cloud Moderator

Method	URL	Description
POST	/homes	Create new home
GET	/homes	View all homes
GET	/homes/{homeId}	View home with id={homeId}
PUT	/homes/{homeId}	Edit home with id={homeId}
DELETE	/homes/{homeId}	Delete home with id={homeId}

Figure 4.18: Home Management REST API

Method	URL	Description
POST	/superusers	Create new superuser
GET	/superusers	View all superusers
GET	/superusers/{superuserId}	View superuser with id={ superuserId}
PUT	/superusers/{superuserId}	Edit superuser with id= {superuserId}
DELETE	/superusers/{superuserId}	Delete superuser with id={ superuserId}
POST	/users	Create new user
GET	/users	View all users
GET	/users/{userId}	View user with id={userId}
PUT	/users/{userId}	Edit user with id={userId}
DELETE	/users/{userId}	Delete user with id={userId}

Figure 4.19: User Management REST API

- Home Moderator

Method	URL	Description
POST	/homes/{homeId}/rooms	Create new room in home with id = homeId
GET	/homes/{homeId}/rooms	View rooms of home with id=homeId
GET	/homes/{homeId}/rooms/{roomId}	View room with id={roomId} of home with id={homeId}
PUT	/homes/{homeId}/rooms/{roomId}	Edit room with id={roomId} of home with id={homeId}
DELETE	/homes/{homeId}/rooms/{roomId}	Delete room with id={roomId} of home with id={homeId}

Figure 4.20: Room Management REST API

Method	URL	Description
POST	/homes/{homeId}/devices	Insert new device in home with id=homeId
GET	/homes/{homeId}/devices	View devices of home with id=homeId
GET	/homes/{homeId}/rooms/{roomId}/devices	View devices of room with id=roomId that belongs to home with id=homeId
GET	/homes/{homeId}/rooms/{roomId}/devices/{deviceId}	View device of room with id=roomId that belongs to home with id=homeId
PUT	/homes/{homeId}/rooms/{roomId}/devices/{deviceId}	Operate device of room with id=roomId that belongs to home with id=homeId
DELETE	/homes/{homeId}/devices/{deviceId}	Delete device of home with id=homeId

Figure 4.21: Device Management REST API

Method	URL	Description
POST	/homes/{homeId}/rules	Create rule in home with id=homeId
GET	/homes/{homeId}/rules	View rules of home with id=homeId
GET	/homes/{homeId}/rules/{ruleId}	View rule with id=ruleId of home with id=homeId
DELETE	/homes/{homeId}/rules/{ruleId}	Delete rule with id=ruleId of home with id=homeId

Figure 4.22: Rule Management REST API

Method	URL	Description
GET	/homes/{homeId}/users	View users of home with id=homeId
GET	/homes{homeId}/users/{userId}	View user with id=userId of home with id=homeId
PUT	/homes/{homeId}/users/{userId}	Edit access rights of user with id=userId that belongs to home with id=homeId

Figure 4.23: User Management REST API

- Resident

Method	URL	Description
GET	/homes/{homeId}/devices	View devices of home with id=homeId
GET	/homes/{homeId}/rooms/{roomId}/devices	View devices of room with id=roomId that belongs to home with id=homeId
GET	/homes/{homeId}/rooms/{roomId}/devices/{deviceId}	View device of room with id=roomId that belongs to home with id=homeId
PUT	/homes/{homeId}/rooms/{roomId}/devices/{deviceId}	Operate device of room with id=roomId that belongs to home with id=homeId

Figure 4.24: Device Management REST API

4.2.2 User Identity Manager

Method	URL	Description
GET	/oauth2/authorize?response_type=code&client_id={clientID}&state=xyz&redirect_uri={redirectURI}	Request authorization code from keyrock. Client_id is provided by fiware upon application registration. Redirect URI is defined by the application creator and it points to the URI that the code will be send
POST	/oauth2/token	Request access token from Keyrock
GET	/user?access_token={accessToken}	Get user information by sending the accessToken that we received from Keyrock

Figure 4.25: Keyrock REST API

4.2.3 Complex Event Processing

Method	URL	Description
POST	/eventHandler	Notifications from Publish/Subscribe service are send here.
POST	/timeEvent	Create time rule

Figure 4.26: CEP REST API

4.2.4 Publish/Subscribe Service

Method	URL	Description
POST	/v2/entities	Create entity
GET	/v2/entities	Retrieve all entities
GET	/v2/entities/{entityId}	Retrieve entity with id=entityId
PUT/PATCH	/v2/entities/{entityId}	Update entity with id=entityId
DELETE	/v2/entities/{entityId}	Delete entity with id=entityId
GET	/v2/entities/{entityId}/attrs/{attrName}	Retrieve value of attribute with name=attrName of entity with id=entityId
PUT/PATCH	/v2/entities/{entityId}/attrs/{attrName}	Update value of attribute with name=attrName of entity with id=entityId
DELETE	/v2/entities/{entityId}/attrs/{attrName}	Delete value of attribute with name=attrName of entity with id=entityId

Figure 4.27: Entity Management API - Orion

Method	URL	Description
POST	/v2/subscriptions	Create subscription
GET	/v2/subscriptions	Retrieve all subscriptions
GET	/v2/subscriptions/{subId}	Retrieve subscription with id=subId
DELETE	/v2/subscriptions/{subId}	Delete subscription with id=subId

Figure 4.28: Subscription Management API - Orion

4.2.5 History Database

Method	URL	Description
POST	/logs/{logType}	Create log entry. LogType can be: homeLog, userLog, roomLog, deviceLog
POST	/alert	Create alert when an alarm triggers
POST	/notifications/rule	Create notification when a rule is executed
POST	/notifications/deviceStatus	Create notification when a device malfunctions

Figure 4.29: History Database API

4.2.6 Connectivity Service

Method	URL	Description
POST	/homes/{homeId}/rooms	Create room in home with id=homeId
PUT	/homes/{homeId}/rooms/{roomId}	Edit room with id=roomId of home with id=homeId
DELETE	/homes/{homeId}/rooms/{roomId}	Delete room with id=roomId of home with id=homeId
POST	/homes/{homeId}/devices	Insert new device in home with id=homeId
PUT	/homes/{homeId}/rooms/{roomId}/devices/{deviceId}	Operate device of room with id=roomId that belongs to home with id=homeId
DELETE	/homes/{homeId}/devices/{deviceId}	Delete device with id=deviceId of home with id=homeId
POST	/getRoomValues	Homes send their room values every one minute
POST	/getDeviceValues	Homes send their device values every one minute

Figure 4.30: Connectivity Service API

4.3 HTTPS ⁵

In order to secure data integrity in our system, the communication between services and between the front and back-end is implemented in HTTPS over SSL. HTTPS is an extension of the HTTP protocol and is used for secure

⁵<https://www.globalsign.com/en/ssl-information-center/>

communication. It provides data encryption and authenticated client - server connection by using trusted, signed certificates. SSL Certificates are small data files that bind a cryptographic key to an organization's detail and includes the domain name, server name or hostname as well as the organization's details (i.e. company name, email, location). Anyone can create an SSL Certificate, but in order to be valid and trusted must be issued from a trusted Certificate Authority (CA). CAs can be private companies to governments and the longer they are operational, the more their certificates are trusted. Before issuing a Digital Certificate, the CA conducts checks into the identity of the applicant (i.e. a domain validated SSL Certificate will have verified the ownership of the domain to be included in the certificate).

The certificate is composed from two keys; the private and the public. Once the certificate is installed on the server, an encrypted connection between the server and the client can be initiated, starting with an SSL handshake⁶. The steps for an HTTPS communication over SSL are the following:

- Server sends a copy of its public key to the client.
- Client creates a symmetric session key and encrypts it with the server's public key, then sends it to the server.
- Server decrypts the encrypted session key using its private key to get the symmetric session key.
- Server and client encrypt and decrypt all transmitted data with the symmetric session key. Only the server and the client know the symmetric session key that is used for that specific session. When a new connection is established, a new session key is created.

For the purpose of our application, we used Java's `keytool`⁷ command and created three self-signed certificates, one for every VM, using the RSA cryptographic algorithm. The private key of each certificate is stored in the corresponding VM and the public keys are stored in the trusted certificates database (`cacerts`) of the rest VMs. In figure 5.29 we see an example of a public key. In the Owner-Common Name field the domain name or the IP of the server must be inserted. In this example we use the floating IP of the VM, 147.27.60.196 because we don't have a DNS name. In the field "SubjectAlternativeName" we inserted as `DNSName` the "localhost" value and as `IPAddress` the VM's floating IP. This was necessary because in this VM are many services that communicate with each other using HTTPS and if the localhost DNS name was missing, the communication would be rejected.

⁶<https://www.digicert.com/ssl-certificate/>

⁷<https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html>

```

Alias name: appLogic3
Creation date: Dec 17, 2018
Entry type: trustedCertEntry

Owner: CN=147.27.60.196, OU=TUC, O=TUC, L=Chania, ST=Unknown, C=GR
Issuer: CN=147.27.60.196, OU=TUC, O=TUC, L=Chania, ST=Unknown, C=GR
Serial number: 139540e5
Valid from: Sun Nov 11 17:36:09 UTC 2018 until: Sat Feb 09 17:36:09 UTC 2019
Certificate fingerprints:
    MD5:  F6:89:05:1F:4B:22:25:77:9C:F7:61:35:8B:08:D5:73
    SHA1: 98:F7:B0:5F:8F:22:F1:F9:DE:16:51:7E:05:E7:3F:86:81:56:EE:B0
    SHA256: 21:7E:BC:81:A4:E6:3C:E3:B0:A1:D3:F6:B8:AD:24:FC:FB:3C:61:69:51:
59:2E:6A:04:71:73:42:13:0C:C5:16
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.17 Criticality=false
SubjectAlternativeName [
  DNSName: localhost
  IPAddress: 147.27.60.196
]

#2: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 01 D0 AE 20 46 FE D2 15  1B 48 9F 07 32 1C 5B 7A  ... F....H..2.[z
0010: 3F 0A 08 9D                               ?...
]
]

```

Figure 4.31: public key of ApplicationLogic VM certificate

Chapter 5

Performance Evaluation

We run an exhaustive set of experiments and we analyze the performance limits of the services running in the cloud. We study also system scalability (i.e. how system response time increases with the number of connected users). iHome is deployed in three Virtual Machines (VMs). The first two VMs are deployed in the FIWARE node of TUC (FIWARE is a federation of distributed cloud infrastructures running OpenStack). These are small flavor VMs with one virtual processor (x86_64 processor architecture, 2,800 Mhz, first level cache size 32 KB, second layer 4,096 KB cache size), 2,048 MB RAM, 20 GB hard drive capacity. Each VM runs Ubuntu Operating System 14.04 and Apache HTTP server. The first VM runs application logic, user authorization, EP, database and connectivity service. The second VM runs PS service alone. Finally, the third VM runs Keyrock IdM service which is provided by a shared VM installed in a remote FIWARE node (e.g. in Spain). All VMs may receive up to a large number of simultaneous requests as the number of users and requests increase. All measurements of time below account also for the time spent for the communication between VMs or between services within the same or different VMs.

For each experiment we report average response time (over 2,000 requests) for the most common operations. Each operation can be executed either sequentially (i.e. one after the other) or in parallel. We use ApacheBench¹ (Apache HTTP server benchmarking tool) to send multiple simultaneous requests to iHome. In ApacheBench we are opted to define the total number of requests and how many of them will be executed simultaneously. We report average response time for 1,000 requests executed in a sequence (concurrency = 1) and also for increasing values of concurrency (i.e. concurrency $\gg 1$). All service requests address the user authorization and Keyrock IdM service at the back-end which checks if the user or service issuing the request has the necessary access rights. Figures 5.1 - 5.6 summarize the performance of the following basic operations executed in iHome.

A home posts room values to the connectivity service (at the back-end) and published in PS service (i.e. the corresponding entities are updated) and the database. Users subscribing to this information get a notification about this change on the Web application (through application logic service). The EP service subscribes always to this information and triggers the execution of rules

¹<https://httpd.apache.org/docs/2.4/programs/ab.html>

that govern the monitoring of a home. If these values are within the pre-defined limits, no action is taken; otherwise (e.g. temperature exceeds a limit), EP service notifies the home moderator to take action.

A user requests room values from a home. The request is issued on the Web application. From there, it is forwarded to PS service which always holds the most recent values of all home entities. Finally, the Web application receives the requested information (again through application logic which is responsible for orchestrating operations running in iHome).

Insert a device at a home. The home moderator needs to start monitoring a new device in a home. The request is issued on the Web Application and its forwarded to application logic service. This operation will insert a new entity in PS service with the three default values (i.e. temperature, humidity and luminosity). The request is forwarded (through connectivity service) to the front-end. After decoding, user authorization service in the front-end will check if the user has the necessary access rights. If access is granted, the new device is created (data collector and database services are updated as well). The device is published at PS service (i.e. a new entity is created) and the home moderator can define users subscribing to the new PS entity. The user receives a notification upon successful completion of this process.

<i>A home posts room values to the cloud: POST localhost:8228/getRoomValues</i>						
<i>Concurrency</i>	1	50	100	150	200	300
<i>Time (ms)</i>	88.29	67.23	63.29	66.22	70.29	77.71
<i>CPU (%)</i>	97.94	99.43	98.86	98.14	96.24	99.39
<i>RAM (%)</i>	69.67	75.65	80.96	87.10	93.96	95.88

Figure 5.1: Performance of the first experiment

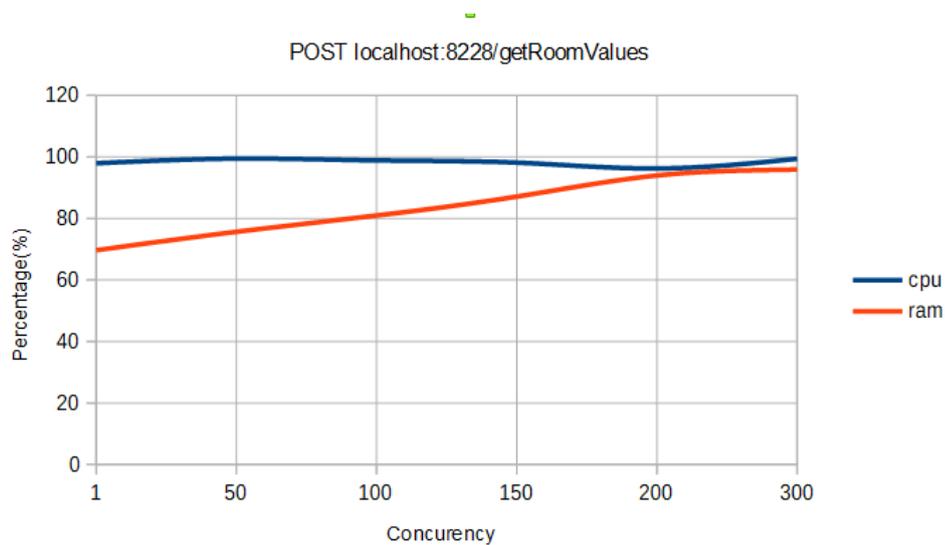


Figure 5.2: Resources utilization; first experiment

<i>Get room values from a home: GET localhost:8448/homes/homeId/rooms/roomId</i>						
Concurrency	1	50	100	150	200	300
Time (ms)	67.22	61.91	63.93	65.83	67.29	73.59
CPU (%)	36.16	74.96	78.26	77.66	73.08	68.63
RAM (%)	73.29	75.52	77.39	80.51	83.03	84.75

Figure 5.3: Performance of the second experiment

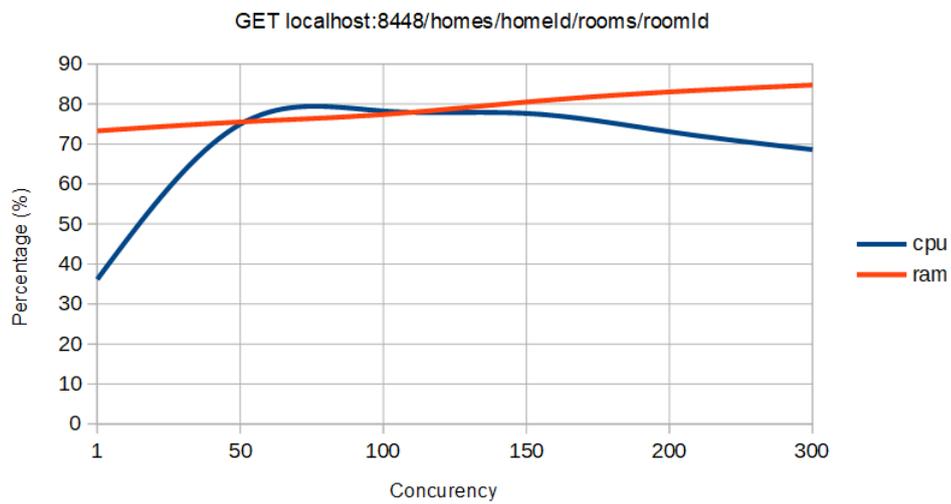


Figure 5.4: Resources utilization; second experiment

<i>Insert a new device at a home: POST localhost:8448/homes/homeId/devices</i>						
Concurrency	1	50	100	150	200	300
Time (ms)	370.5	100.1	101.3	103.1	104.4	109.8
CPU (%)	21.84	58.68	53.72	51.65	53.02	49.87
RAM (%)	66.43	68.96	70.74	72.86	74.90	77.76

Figure 5.5: Performance of the third experiment

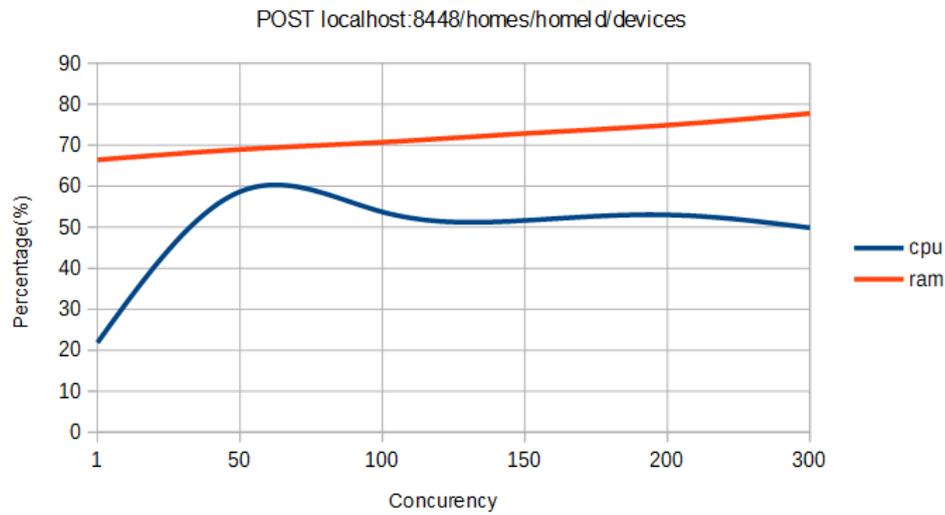


Figure 5.6: Resources utilization; third experiment

Response times improve with the simultaneous execution of requests (i.e. the Apache HTTP server switches to multitasking) reaching their lowest values for concurrency between 50 and 150. Even with concurrency = 300 the average execution time per request is close to real-time and resource usage is well below 100% in most cases.

iHome may produce big amounts of data and requests, requiring large processing capabilities which can surpass the capacities that our experimental system set-up is able to provide. In this iHome set-up, core services are implemented in three VMs. Presumably, overloading the VMs might occur in an application with a much larger number of concurrent users. An obvious solution to dealing with performance would be to employ additional VMs each running a single service (or a small group of services). Alongside, we can allocate additional VMs implementing the same service (or groups of services) thus having more than one VM sharing the load.

Chapter 6

Conclusion - Future Work

Leveraging PaaS functionality we show how a smart home management system is designed and implemented as a service in the cloud. iHome exhibits a highly modular SOA design based on micro-services running both, at a home (i.e. a fog node) and in the cloud allowing users to monitor and control their homes remotely in real-time.

6.1 Conclusions

The main goal of this thesis was to design and implement a service oriented architecture system for managing smart homes. During this process, we came to some conclusions.

Choosing a cloud environment for the development of a system proves to be very beneficial. A big variety of services, like the Orion Context Broker and the Keyrock Identity Manager, is provided to the users, making easier and faster the implementation and deployment of applications. Using virtual resources (i.e. CPU, RAM, Storage) and virtual machines, as well as having the ability to scale up or down your system according to the demands, cuts down the high cost of hardware and removes the need of complicated setup of networks and computing systems. Nowadays, almost everyone has access to internet through their smartphones, laptops etc. Connecting to a cloud application requires nothing more, and can be done from everywhere. Our system, provides a cloud application with no need for installation; just a device with a browser is enough, making it accessible from virtually everywhere.

Using service oriented architecture and RESTful Web services, we achieved the services isolation and modulation. As a result, communication between them is guaranteed even if they are deployed into different cloud environments. Also, by deploying our system in a virtualized environment, we are avoiding compatibility issues that could arise from hardware architecture.

iHome has been tested in a realistic scenario with up to three hundred concurrent users. The experimental results reveal that, iHome is capable of responding in real time even under heavy workloads. Besides being fast and modular, iHome innovative design relies on secure cloud services permitting access to services and information based on access rights and authorization (i.e. all services are protected by a security mechanism). At the same time, iHome implements

secure communication for data and services over HTTPS.

6.2 Future Work

As a future extension, iHome must be tested using a large number of physical devices and appliances installed at homes, instead of simulated data. A node like a gateway should be used, to collect all the data from these devices. After converting the device communication protocol (e.g. ZigBee) into a JSON format, data would be send to the back-end and vice versa.

Also, we could implement a better, reactive User Interface, that can run flawlessly on smartphones and computers and use technologies like Angular¹, to reverse some of the server's workload to the browsers. Another way would be to create a PC and mobile application, that would have to be installed and take the role of the gateway.

Concerning security issues, we could provide a proxy (FIWARE's PEP Proxy-Wilma) over our system, in order to enforce better access control to our application. Further, we should implement an authorization mechanism (i.e. mac address locking) for data transmitted from a home to the back-end.

A better implementation of the Complex Event Processing service could be achieved by supporting more simultaneous conditions (our system supports up to two conditions) as well as pattern detection.(i.e. events occur into a specified order in a specific time window).

Finally, implementation of data analytics functionality in a public cloud, in a use case with many homes connected to iHome, is also an interesting direction for future work.

¹<https://angular.io/>

Bibliography

- [1] F. Bonomi, R.A. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC'12)*, pages 13–16, Helsinki, Finland, 8 2012.
- [2] S. Ghosh. Smart homes: Architectural and engineering design imperatives for smart city building codes. In *Technologies of Smart-City Energy Security and Power (ICSESP'2018)*, pages 1–4, Bhubaneswar, India, March 2018.
- [3] OpenFog Architecture Working Group. Openfog architecture overview, February 2016.
- [4] L. Liu, Y. Liu, L. Wang, A. Zomaya, and S. Hu. Economical and balanced energy usage in the smart home infrastructure: A tutorial and new results. *IEEE Transactions on Emerging Topics in Computing*, 3(4):556–570, December 2015.
- [5] E. G.M. Petrakis, S. Sotiriadis, T. Soultanopoulos, P. Tsiachri Renta, R. Buyya, and N. Bessis. Internet of things as a service (itaas): Challenges and solutions for management of sensor data on the cloud and the fog. *Internet of Things*, 3–4(9):156–174, September 2018.
- [6] S. Rehman and V. Gruhn. An approach to secure smart homes in cyber-physical systems/internet-of-things. In *Software Defined Systems (SDS)*, pages 126–129, Barcelona, Spain, May 2018.