BEng Thesis

Reengineering and development of IoT Systems for Home Automation

Rafael Rocha

CISTER-TR-171204

Reengineering and development of IoT Systems for Home Automation

Rafael Rocha

*CISTER Research Centre

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8321159

E-mail:

http://www.cister.isep.ipp.pt

Abstract

With the increasing adoption of technology in today 19s houses, electricity is at an all-time highdemand. In fact, given the plethora of vital electricity-powered appliances used every day,such as refrigerators, washing machines, and so forth, it has been proven difficult to evenhandle all devices 19 electric consumption. To reduce consumption costs and turn it into a moremanageable process, the concept of flex-offers was created. A flex-offer is built aroundscheduling energy usage in conjunction with the prices of electricity, as provided by an energymarket. More specifically, a flex-offer is an energy consumption offer containing the user 19senergy consumption flexibility, which is sent to an entity called the Aggregator, whoaggregates together flex-offers from multiple parties, bargains with the energy market, andresponds to each flex-offer with a schedule that meets the lowest prices for consumption,while still satisfying the users 19 needs. By using flex-offers on a house 19s equipment, the idea ofFlexHousing was born. The aspired goal of the CISTER Research Center 19s FlexHousing projectis to deliver a platform where users can register their smart appliances, regardless of its brandand distributor, set up preferences for the devices 19 usage, and let the system manage theenergy consumption and device activation schedules based on the energy market prices.A previous project had already built a prototype of the FlexHousing system. Nevertheless, theoriginal platform had many limitations and lacked maturity from a software engineering pointof view, and the goal of this internship is to apply a reengineering process on the FlexHousingproject, while also adding new features to it. Thus, the project 19s domain model, its database,and class structures were altered to satisfy the new requirements. Furthermore, its webplatform was rebuilt from the ground up. Also, a new interface was developed to facilitatesupport for devices of different brands. As a proof of concept for the benefits provided by thisnew interface, a connection with a new device (Sonoff Pow) was also established. Moreover,a new functionality was developed to identify a device 19s type of appliance based on its energyconsumption, in other words, to specify if a device is, for instance, a refrigerator or not. Finally,another new feature was added in which, based on a device 19s type and its energy consumptionpattern, the flex-offer creation is automated, minimizing user input.As planned, the FlexHousing platform now supports multiple types of devices, and has asoftware interface to support more types in the future with minimal effort. The flex-offercreation process has been simplified and is now partially automated. Finally, the webplatform 19s UI has been updated, becoming more intuitive and appealing to the user.

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Reengineering and development of IoT Systems for Home Automation

CISTER - Research Centre in Real-Time and Embedded Computing Systems

2016 / 2017

1140329 Rafael Teles da Rocha

isep Instituto Superior de Engenharia do Porto

Reengineering and development of IoT Systems for Home Automation

CISTER - Research Centre in Real-Time and Embedded Computing Systems

2016 / 2017

1140329 Rafael Teles da Rocha

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Degree in Informatics Engineering

October 2017

ISEP Advisor: **Luis Lino Ferreira**

External Supervisors: **Michele Albano, José Bruno Silva**

«To my parents, for all the love and support they have given me »

Acknowledgments

First, I would like to thank my family for always supporting me and turning me into the person I am today. Without your help and love, I wouldn’t have reached this far in my journey.

I also want to thank professor Luis Ferreira, my supervisors Michele Albano and José Silva, and CISTER colleagues André Pedro, Pedro Santos, and Vincent Nelis for guiding me and assisting me along this project’s development. It was a wonderful learning experience that helped me become a more professional and pragmatic developer.

Furthermore, I have to thank everyone at CISTER and ISEP DEI for giving me the tools and opportunity to pursue my goals.

And last, but certainly not least, I want to thank my friends and colleagues at ISEP for working with me, sharing their knowledge with me, and helping me out when I most needed. I will always cherish those moments.

Rafael Rocha

Abstract

With the increasing adoption of technology in today's houses, electricity is at an all-time high demand. In fact, given the plethora of vital electricity-powered appliances used every day, such as refrigerators, washing machines, and so forth, it has been proven difficult to even handle all devices' electric consumption. To reduce consumption costs and turn it into a more manageable process, the concept of flex-offers was created. A flex-offer is built around scheduling energy usage in conjunction with the prices of electricity, as provided by an energy market. More specifically, a flex-offer is an energy consumption offer containing the user's energy consumption flexibility, which is sent to an entity called the Aggregator, who aggregates together flex-offers from multiple parties, bargains with the energy market, and responds to each flex-offer with a schedule that meets the lowest prices for consumption, while still satisfying the users' needs. By using flex-offers on a house's equipment, the idea of FlexHousing was born. The aspired goal of the CISTER Research Center's FlexHousing project is to deliver a platform where users can register their smart appliances, regardless of its brand and distributor, set up preferences for the devices' usage, and let the system manage the energy consumption and device activation schedules based on the energy market prices.

A previous project had already built a prototype of the FlexHousing system. Nevertheless, the original platform had many limitations and lacked maturity from a software engineering point of view, and the goal of this internship is to apply a reengineering process on the FlexHousing project, while also adding new features to it. Thus, the project's domain model, its database, and class structures were altered to satisfy the new requirements. Furthermore, its web platform was rebuilt from the ground up. Also, a new interface was developed to facilitate support for devices of different brands. As a proof of concept for the benefits provided by this new interface, a connection with a new device (Sonoff Pow) was also established. Moreover, a new functionality was developed to identify a device's type of appliance based on its energy consumption, in other words, to specify if a device is, for instance, a refrigerator or not. Finally, another new feature was added in which, based on a device's type and its energy consumption pattern, the flex-offer creation is automated, minimizing user input.

As planned, the FlexHousing platform now supports multiple types of devices, and has a software interface to support more types in the future with minimal effort. The flex-offer creation process has been simplified and is now partially automated. Finally, the web platform's UI has been updated, becoming more intuitive and appealing to the user.

Keywords (Theme):	Internet of Things, Flex-offer, Home automation, Energy saving, Smart Building.
Keywords (Technologies):	Java, PHP, REST, Arrowhead Framework, Laravel Framework.

Table of Contents

Table of Figures

Table of Tables

Notation and Glossary

API	Application Programming Interface. A set of clearly defined methods of communication between various software components.
Arrowhead Project	An international project, developed in partnership with CISTER, to address the technical and applicative challenges associated to cooperative automation.
BNearIT AB **(or BNearIT)**	A company focused on advanced systems development, modern software architectures, and service-based systems. BNeartIT is a partner of the Arrowhead Project.
CISTER	Research Centre in Real-Time and Embedded Computing Systems
CRUD	Create, Read, Update, and Delete. CRUD are the four basic functions of persistent storage [88].
DTO	A Data Transfer Object (DTO) is an object that is used to encapsulate data, and send it from one subsystem of an application to another.
IoT	Internet of Things. The inter-networking of physical devices, vehicles, buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data.
ISA	Intelligent Sensing Anywhere, a Portuguese IoT company for Oil&Gas market.
Java	General-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.
Multi-paradigm programming language	A programming language that supports more than one programming paradigm (i.e. object-oriented programming, functional programming, etc.).

MVC	Model–View–Controller. A software architectural pattern for implementing user interfaces on computers.
PHP	Server-side scripting language designed primarily for web development but also used as a general-purpose programming language.
REST	Representational state transfer or RESTful Web services are a way of providing interoperability between computer systems on the Internet.
SOA	Service-Oriented Architecture. A style of software design where services are provided to the other components by application components, through a communication protocol over a network.
System of Systems	An assortment of task-oriented or dedicated systems that merge their resources and capabilities together to create a new, more complex system which offers more functionality and performance than simply the sum of the constituent systems [80].
UI	User Interface
UML	Unified Modeling Language, a software modeling language.
VME	Virtual Market of Energy
VPN	Virtual Private Network
VPS	Virtual Power Solutions, a Portuguese energy solutions company related to ISA.
XMPP	Extensible Messaging and Presence Protocol

1 Introduction

This chapter consists of an overview of the project's context and its main goals.

1.1 Project Context

CISTER Research Center focuses its activity on the analysis, design and implementation of real-time and embedded computing systems. These embedded systems play an important role in IoT due to their unique characteristics and features such as real-time computing, low power consumption, low maintenance, and high availability.

In this context, CISTER has a project named FlexHousing that handles current IoT themes, such as home automation and smart device interoperability, with a focus on energy saving. However, considering the many limitations of the project's original prototype, it not only requires a reengineering process, but also the development of new features.

Taking on a project of this scale can greatly benefit one's research and engineering skills, while also allowing one to be at the forefront of the future of technology.

1.2 Project Overview

With the increasing adoption of technology in today's houses, electricity is at an all-time high demand. In fact, given the plethora of vital electricity-powered appliances used every day, such as refrigerators, lights, washing machines, water heaters, and so forth, it has been proven difficult to even manage all their operations, especially when it comes to electric consumption.

To solve this problem, several innovations were introduced. One of them is the concept of the Internet of Things (IoT), which turned itself into a key component of home automation and smart homes [1]. Home automation is automating the ability to control items around the house with a simple push of a button [1]. However, simple automation is not enough to effectively decrease energy costs, since prices in the energy market are constantly changing for any given hour.

Thus, to reduce consumption costs and turn it into a more manageable process, the concept of flex-offers was created [8]. A flex-offer is built around scheduling energy usage in conjunction with the prices of electricity, as provided by an energy market. More specifically, a flex-offer is an energy consumption offer containing the user's energy consumption flexibility, which is sent to an entity called the Aggregator, who aggregates together flex-offers from multiple parties, bargains with the energy market, and responds to each flex-offer with a schedule that meets the lowest prices for consumption, while still satisfying the users' needs. By using flex-offers on a house's equipment, the idea of FlexHousing was born.

FlexHousing (Fig. 1), project developed at CISTER and this internship's main theme, revolves around the implementation of a pilot capable of applying the Flex Offer concept to a real-life situation (using the Arrowhead framework), allowing control over the energy usage of a home's or building's appliances. The FlexHousing project is composed of two different applications: one is the FlexHousing Middleware, which communicates with devices, manages a database (which contains the registered users, houses, and devices), and provides its data as a RESTful service to web applications; the other is a web application, known as FlexHousing web platform, which serves as a gateway to the Middleware's data and services.

Fig. 1 – Visual representation of the FlexHousing project. [81]

Yet, since IoT is still at a very early stage, there aren't many standards when it comes to the process of controlling devices, leading to some companies using closed communication protocols for their own appliances. Some devices send their data to a service provider's servers and are controlled through a web or mobile application. This app connects to the service provider's cloud, sending commands and requesting the device's data from the company's servers. Whereas, other devices limit their access to the user's local network, meaning that the user must use the device's proprietary app while connected to his private network.

In this context, the goal of CISTER's FlexHousing platform is to deliver a platform where users can register their smart appliances, regardless of its brand and publisher, and manage their energy consumption by scheduling energy usage based on the energy market prices. However, the existing platform had many deficiencies: the web platform's UI could be heavily improved; it was developed to only function with devices from VPS and none more; the concept of multiple "Users" and "Houses" were not taken into account, meaning that, effectively, the system only supported one user with one house; the flex-offers had to be manually created through the user's input, meaning that a user without any background knowledge would be alienated; and its flex-offer implementation had dependencies to external servers.

Midway through development, this internship's project also built a proof of concept for a "client" (which cannot be identified in this report), namely, a multinational company in the field of household appliances and consumer electronics, who is interested in offering a service, whose basis can be built upon or based on the platform developed in this project. The client currently has a vision for an after-sales service which consists of a remote maintenance platform (for the purposes of this report, the client's envisioned platform will be called "DeviceFix"). Consequently, the client saw the Flex-offer concept as a feature that could add value to their DeviceFix platform. So, given that the FlexHousing platform was already at an advanced stage, the company proposed the merge of interests between both platforms, so that FlexHousing would serve as its customers' platform, and the company would have its own platform to manage its customers' appliances usage. As a result, the company wished for a platform that, by integrating with FlexHousing, displayed some statistical information about FlexHousing's users.

Therefore, the goal of this internship is to apply a reengineering process on the FlexHousing platform, while also adding new features to it. Thus, the project strives for six major goals:

- Rebuild and improve the FlexHousing web platform;

- Reengineer the connection to IoT devices, enabling compatibility with other types of appliances of different brands and manufacturers. This will also allow the FlexHousing project to connect to local generic devices, which in turn will generate a lesser dependency to third-party service providers;
- Add support for multiple “Users” and “Houses” in the FlexHousing Middleware;
- Add a feature that enables the automatic creation of Flex-offers, based on a device’s consumption pattern;
- Reengineer the FlexHousing Middleware so that its Arrowhead implementation can also function locally (in servers under CISTER/ISEP responsibility), without needing to connect to external servers;
- Develop a platform for company executives, which integrates with FlexHousing and displays some basic analyses of user data.

1.3 Organization Overview

CISTER (Research Centre in Real-Time and Embedded Computing Systems) is a Research Unit based at the School of Engineering (ISEP) of the Polytechnic Institute of Porto (IPP), Portugal. CISTER was, in 2004 and 2007, awarded the classification of Excellent in the FCT evaluations. CISTER has a strong and solid international reputation, built upon a robust track record of publications, a continuous presence on program and organizing committees of international top conferences [2].

CISTER — like its name entails — has been focusing its activity in the analysis, design and implementation of Real-Time and Embedded Computing Systems. CISTER has, mostly, provided advances in architectures for distributed embedded real-time systems, real-time wireless sensor networks, cyber-physical systems, middleware for embedded systems and on the usage of multicore processors on real-time systems [2].

1.4 Contributions from this project

The FlexHousing project presents a platform that offers not only the innovative flex-offer concept as a feature to minimize energy costs, but also the compatibility with different smart appliances, allowing its users to have the information to understand and know what is going on in their facility, this being either a house or an industrial site.

Furthermore, this platform could even be valuable to companies that want to better understand their energy usage and costs in real-time, identify malfunctioning equipment and overlay data sets to draw previously invisible insights that can increase efficiency and reduce operating costs. If being used commercially, the project could even offer data to businesses about their customer's device usage.

Ultimately, this project also serves as a proof of concept for a company who is interested in developing their own services that deal with IoT devices and energy management, and want to know all the potential setbacks, as well as opportunities, in developing one.

1.5 Report structure

This report is composed of four main chapters, each divided into sub-chapters:

- **2 Context:** This chapter focuses on explaining the problem the project is trying to solve, while also detailing the business areas where its results can be useful, and mention other projects or competing products that may have tackled the same issue or were important/influential for this project's development. More importantly, it also describes the proposed solution to the project's main problems;
    - **2.1 Main Problem:** This subchapter describes the problems that this project tries to solve;
    - **2.2 Business Areas:** This subchapter identifies and describes the business areas associated with the problem being addressed;
    - **2.3 State of the Art:** Based on the addressed problems, this subchapter identifies known components or approaches that contribute to the development of a viable solution;
    - **2.4 Solution Overview:** This subchapter outlines a solution for the project's main problems.
- **3 Work Environment:** This chapter describes how the work was planned, what kind of work methodology was used for development, all the attended meetings in the internship, and the technologies used to achieve the proposed solution;
    - **3.1 Work Methodology:** This subchapter documents the integration of version control systems into the working methodology, while also referring the development process, in this case, based on RUP;

  - **3.2 Project Planning:** This subchapter describes the project’s planning, identifying relevant tasks and sub-tasks;
  - **3.3 Meetings:** This subchapter mentions all project meetings that were important for the project’s development;
  - **3.4 Used Technologies:** This subchapter describes the chosen technologies for the project’s development.
- **4 Technical description:** This chapter specifies how the project was developed, its requirements, use cases, all the required diagrams for its features, the analysis of its overall structure, and the implementation of its more interesting and/or complex features;
  - **4.1 Requirements Engineering:** This subchapter delves into the project’s user roles, user stories, and its functional and non-functional requirements;
  - **4.2 Analysis:** This subchapter analyses the previous project’s domain model, while determining all the changes that must be made. Next, it presents the new domain model for this current project, and describes every conceptual class in the model;
  - **4.3 Design:** This subchapter describes the design documentation for every use case/feature in the project, while also specifying the changes made to the structure of the previous project’s database;
  - **4.4 Implementation:** This subchapter gives an in-depth description of the implementation of the more interesting and/or complex features developed in the project;
  - **4.5 Tests:** This subchapter describes the methods used to test the developed system, to ensure that all requirements are met and to guarantee accuracy and quality in the results presented by it.
- **5 Conclusions:** This chapter focuses on the final results of the project and whether it was a success or a failure. It also reviews all the encountered difficulties, and summarizes the final product’s advantages and its usefulness for the organization. Finally, the chapter ends by mentioning the overall project/internship experience, the training received from it, and the ease and difficulties experienced over time.

- **5.1 Report Summary:** This subchapter serves as a summary of the most important details focused on the previous chapters;
- **5.2 Accomplished Goals:** In this subchapter, for each objective presented in the introductory chapter, a degree of accomplishment is described for its implementation;
- **5.3 Additional work done:** This subchapter describes other minor works carried out during the project/internship and not part of the objectives or main work;
- **5.4 Limitations and future development:** This subchapter identifies the limitations of the developed project, making a self-critical analysis of the whole work, as well as extrapolating possible directions of future development;
- **5.5 Final Appreciation:** This subchapter provides a personal opinion on the whole internship and its project.

2 Context

This chapter will allow the reader to understand the origin and theoretical stand-point of the project, how the solution will be tackled and its design. Therefore, to understand the problem at hand, there needs to be a clear understanding of the context.

2.1 Main Problem

The main problem of this project is divided into two parts: apply a reengineering process on the original FlexHousing prototype, in order to solve its many limitations; add new features to the FlexHousing platform to both satisfy CISTER's original goal for the project and to meet the new requirements of the client's intended vision.

Regarding the reengineering process, we first need to check the original prototype's main limitations and identify their problems:

- The Middleware only functions with VPS/ISA devices and their REST API. To solve this problem, the solution must address a big hindrance in the IoT/Home Automation industry in general, which is the lack of data and device interoperability standards. Section 2.1.1 explains this problem in detail.
- The Middleware's Arrowhead implementation has dependencies to external servers when handling flex-offers. To develop a solution for this, the Arrowhead architecture and the Flex-offer energy flexibility framework must be understood. Section 2.1.2 focuses on these themes.
- The UI of the original prototype's web platform is not very appealing, nor practical, nor intuitive, and could be heavily improved. Section 2.1.3 addresses the original prototype's UI problems.

Concerning the addition of new features, the most important ones are the following:

- The original FlexHousing prototype required the flex-offers to be manually created through the user's input, which meant that a user without any background knowledge would be alienated. Section 2.1.4 goes more in-depth into this problem.
- The client requested a web platform for executives that, by integrating with FlexHousing, displays some basic analyses of the platform's users' data. Section 2.1.5 goes into more detail about this new platform.

2.1.1 Data and Device Interoperability

Currently, one of the biggest setbacks in IoT is the lack of standards when it comes to user data access and device management. Some brands offer access to their devices through a cloud service, requiring users to request data and send device commands to the company's servers, on the cloud, through a proprietary mobile/web app. In these cases, the IoT communication chain is comprised of three key components: sensors or actuators in a house, a gateway, a cloud infrastructure and an end-user application.

Typically, it begins with a sensor network, which consists of sensor nodes (low power devices) with different capabilities, spread over a physical location [3]. These nodes are linked to a gateway device – a more powerful node – that connects to the core network, and sends the sensors' data to the Service Provider's network cloud [3] [4]. Once the Service Provider gathers the data, it is then able to deliver it to the customer, be it through a standard web app, a smartphone app, or even directly into the customer's ERP/billing/customer care software [4]. This means that to access the sensors' data, the app must use the service provider's API.

This sequence of events can be easily understood through Fig. 2:

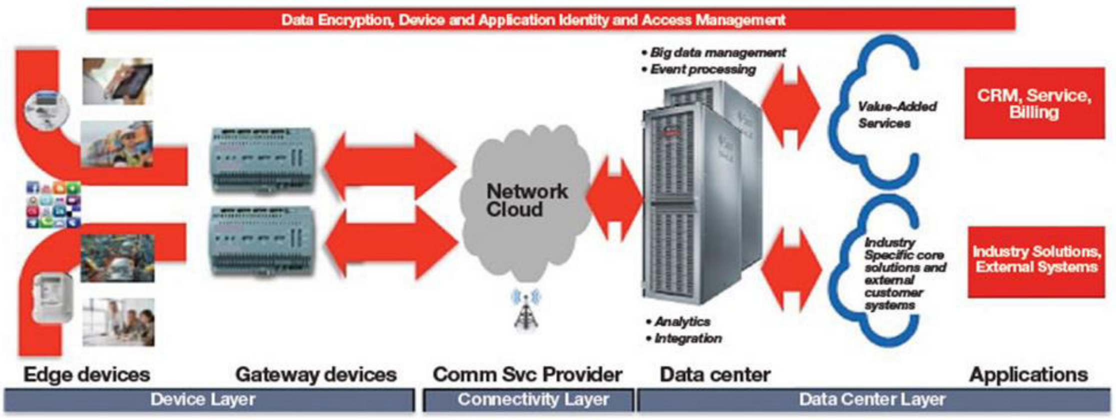


Fig. 2 – A cloud-based IoT value chain. [5]

As we can see, there is a big dependency to third-party service providers when it comes to the process of accessing the sensors' data. This can prove to be a big disadvantage to the app's user and developer, if, at any point, the service provider's servers are down.

On the other hand, other manufacturers offer direct access to their devices, but only through the user's local network, and by only using a proprietary mobile app.

Thus, as was previously explained in the project's goals, one of the main objectives is to enable both connection types from the application to a device, turning it into a flexible and generic platform. This will, in turn, provide easier access to data and improve device interoperability.

2.1.2 Arrowhead Implementation

Some of the components used in this project have been built upon the Arrowhead Framework, therefore this sub-chapter provides a concise description of how the Arrowhead framework is structured. Thus, an explanation of the Arrowhead framework will be given (sections 2.1.2.1 and 2.1.2.2), followed by the main problems to be tackled (section 2.1.2.3).

2.1.2.1 The Arrowhead Framework

The following explanation of the Arrowhead Framework [74] and its flex-offer implementation was mostly based on the papers *"Making System of Systems Interoperable - the Core Components of the Arrowhead Framework"* and "*Arrowhead Compliant Virtual Market of Energy*". Moreover, this report was given permission by their respective authors to use the information and text available in the papers to prove the project's concept and worth, provided that the original works and authors [6] [75] were properly referenced.

The objective of the Arrowhead Framework is to efficiently support the development, deployment and operation of interconnected, cooperative systems. It is based on the SOA philosophy. The building elements of the framework are systems that provide and consume services, and cooperate as systems of systems. With some commonly used systems, such as Service Discovery, Authorization and Orchestration services, it is possible to design and implement a minimal local automation cloud (Fig. 3).

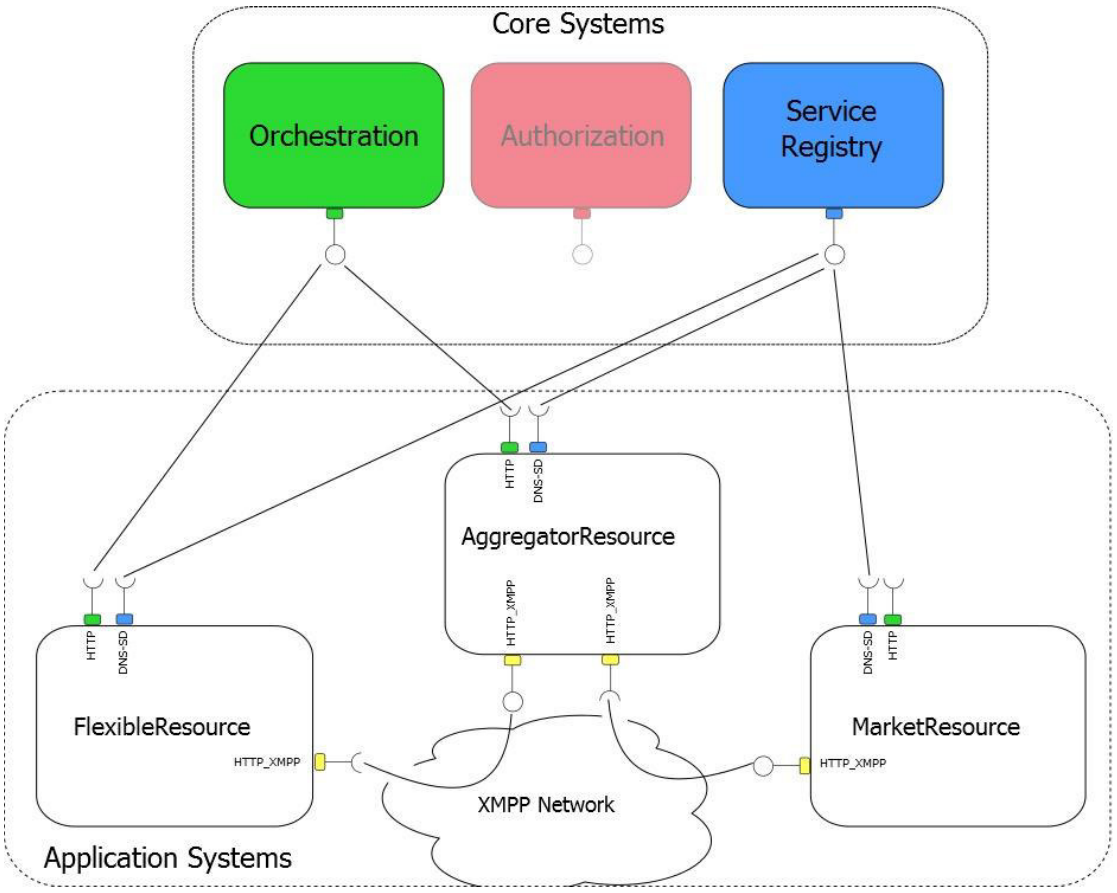


Fig. 3 – Arrowhead Overview [7]

This section outlines some of the core systems that are made available within the Arrowhead Framework.

2.1.2.1.1 Service Registry System

The Service Registry System keeps track of all active producing services within the network. It is used to ensure that all systems can find each other – even if endpoints are dynamically changed. It supports a service registry functionality based on DNS and DNS Service Discovery (DNS-SD); since the Arrowhead Framework is a domain-based infrastructure. All Systems within the network that have services producing information to the network shall publish its producing service within the Service Registry by using the Service Discovery service.

Within a system of systems, the Service Registry further supports system interoperability through its capability of searching for specific service producer features, i.e. an application service producer with a specific type of output. In short, it enables systems to publish their own application services and lookup others'.

2.1.2.1.2 Authorization System

The Authorization system stipulates that a service can only be accessed by an authorized consumer. It consists of two service producers and one service consumer and it maintains a list of access rules to system resources (i.e. services). The Authorization Management service provides the possibility to manage the access rules for specific resources. The Authorization Control service provides the possibility of managing the access for an external service to a specific resource. The system uses the Service Discovery service to publish all its producing services within the Service Registry system.

2.1.2.1.3 Orchestration System

The Orchestration system is a central component of the Arrowhead Framework and in any SOA-based architecture [76]. Orchestration is used to control how systems are deployed and in what way should they be interconnected. Orchestration in the context of SOA can be viewed as the system that supports the establishment of all other systems through providing coordination, control and deployment services.

In industrial applications, the use of SOA for massive distributed system of systems requires Orchestration. It is utilized to dynamically allow the re-use of existing services and systems to create new services and functionalities [77].

The application systems' services are initially seen as passive and being on standby. They are not connected at deployment or even during start-up of the system of systems. Their services can be managed to connect, or be connected to others – to fulfill a specific need.

The Arrowhead Framework currently supports REST-based Orchestration of services using for example REST or CoAP.

2.1.2.2 Energy Flexibility Framework

2.1.2.2.1 The Flex-Offer Concept

Flex-offer is the concept has been developed in the EU FP7 project MIRABEL [8]. It allows exposing demand and supply electric loads with associated flexibilities in time and amount for energy trading, load balancing, and other use-cases. Flex-offers are generic entities, and can accommodate various types of consumers (e.g., electric vehicles, heat pumps, household equipment, industry, etc.) and producers (decharging electric vehicles, solar panels, etc.). In its simplest form, a flex-offer specifies an amount of energy, a duration, an earliest start time, a latest end time, and a price, e.g., "I need 10 KWh over 2 hours between 1 AM and 5 AM, for a price of 0.35 DKK/kWh". A visual representation of this example is shown in Fig. 4:

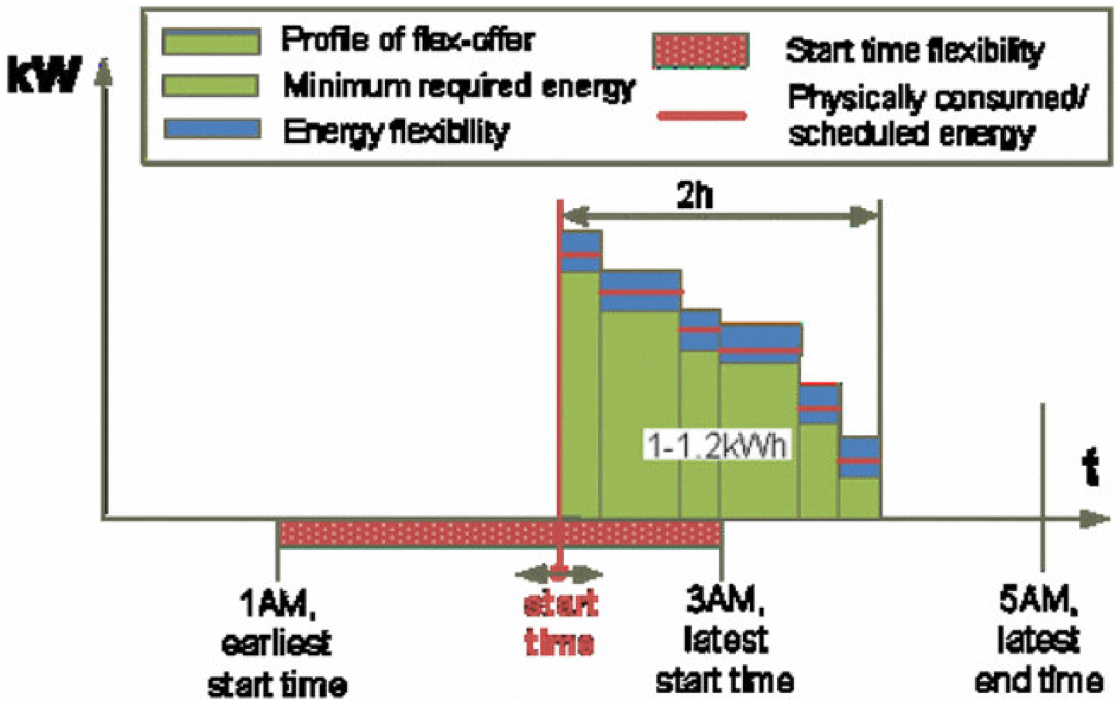


Fig. 4 – Flex-offer example [9]

Flex-offers can be aggregated and disaggregated (Fig. 5) irrespectively to a type of consumption or production they represent. Both aggregated and non-aggregated flex-offers can be mixed and dealt uniformly. A flex-offer can be seen as a kind of “option” that a consumer/producer puts out on a market. The flex-offer may be rejected, for example if the price is not right. If the option is accepted, the flex-offer is given an initial schedule, e.g., the flex-offer is scheduled at 2 AM, and the consumer control system is notified. On the simplest case, the schedule is carried out as specified. However, one of the strengths of the concept only comes into play when things do not go as foreseen, for instance due to a sudden drop in wind energy. In this case, the flex-offer can be rescheduled, shifted to 3 AM, when the wind has returned.

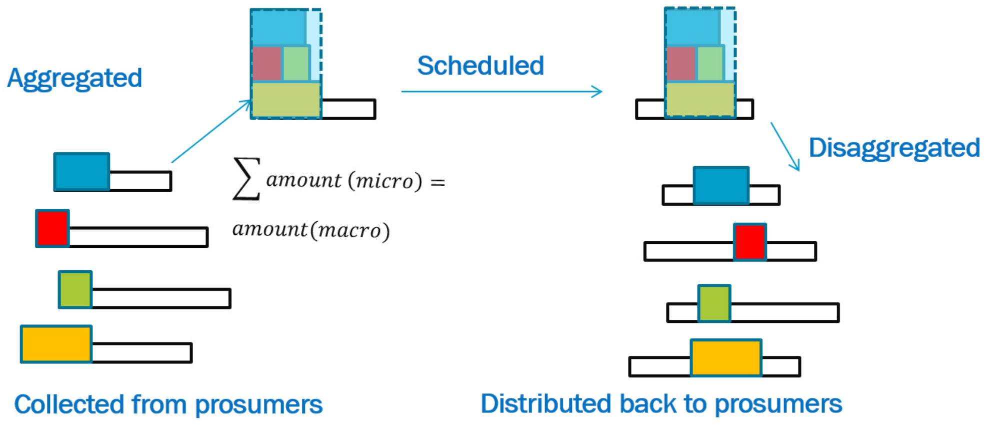

Aggregated
Scheduled
Disaggregated
$\sum amount\,(micro) = amount(macro)$
Collected from prosumers
Distributed back to prosumers

Fig. 5 – Flex-offer scheduling process

2.1.2.2.2 Virtual Market of Energy

For managing flex-offers, a Danish project named TotalFlex [10] proposed the use of the general Virtual Market of Energy system (see Fig. 6) that, by providing a set of Service Oriented Architecture (SOA) interfaces, interconnecting several (existing and new) European Electricity Market Actors.

Prosumers are entities, also named Distributed Energy Resources (DERs) that can both consume and produce electricity. Examples of Prosumers are residential houses, commercial buildings, manufacturing, and process industries. These generate flex-offers and consume schedules.

Aggregators are specialized entities capable of aggregating several (micro) flex-offers from Prosumers into larger (macro) flex-offers. It is also capable of disaggregating (macro) flex-offer schedules, e.g., received from the Electricity Market. An aggregator might be an integrated part of a Balance Responsible Party (BRP).

Balance Responsible Parties are European electricity market entities that secure the balance in a logical sub-domain within the grid, i.e. ensure that consumption is equal to production. It utilizes the aggregated flex-offers from Aggregators for an internal energy balancing and placing flex-offers on a so-called Flexibility Market for trading with other BRPs or Distribution System Operators (DSOs).

Distribution System Operators are entities responsible for uninterrupted supply of energy in the distribution grid. Flexibilities represented by flex-offers and offered on the market enable DSOs new ways to smoothen loads on the distribution grid by buying and then controlling the timing of loads.

Flexibility Market offers BRPs and DSOs the common place for trading flex-offers. It minimizes total costs by scheduling energy loads while respecting the constraints contained in the flex-offers (minimum/maximum power, earliest/latest start of energy consumption, etc.). Flexibility Market may also interface other traditional markets of energy.

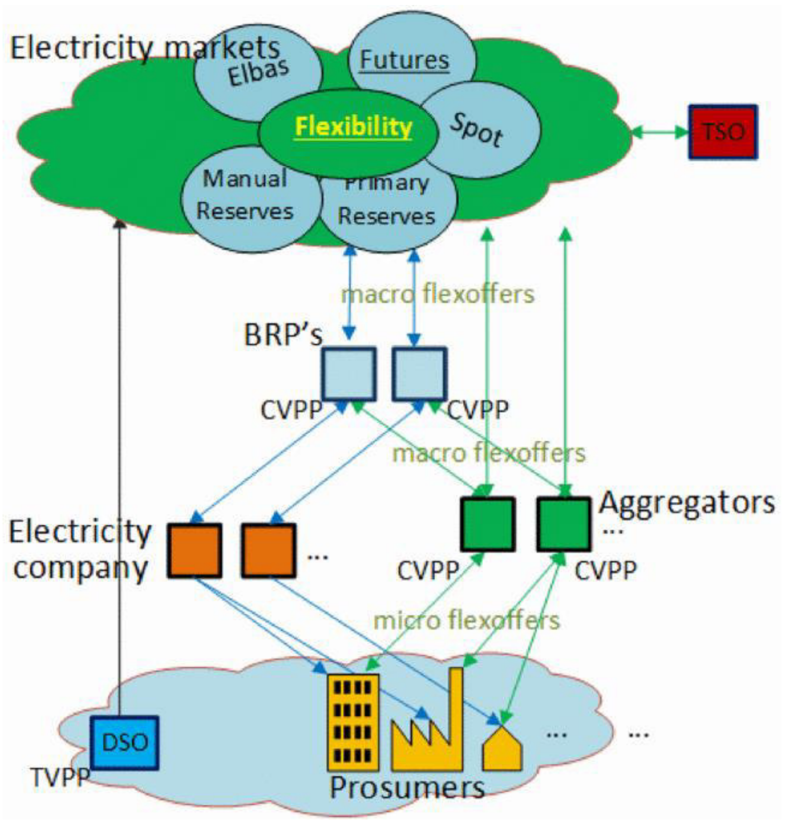


Fig. 6 – Virtual market of energy main actors and operations. [11]

2.1.2.2.3 System Architecture and Communication Interfaces

The block diagram in Fig. 7 introduces the main actors and shows the operation of Arrowhead's Virtual Market of Energy. The architecture described in this report is built upon the Arrowhead Common Framework. The Arrowhead Common Framework thus acts as an enabler for systems from different areas (e.g.: industrial automation, airplane maintenance, energy production, home automation, smart grids) to facilitate their interaction and exchange information. This multi-area approach can enable large savings in terms of energy, efficiency and interoperability.

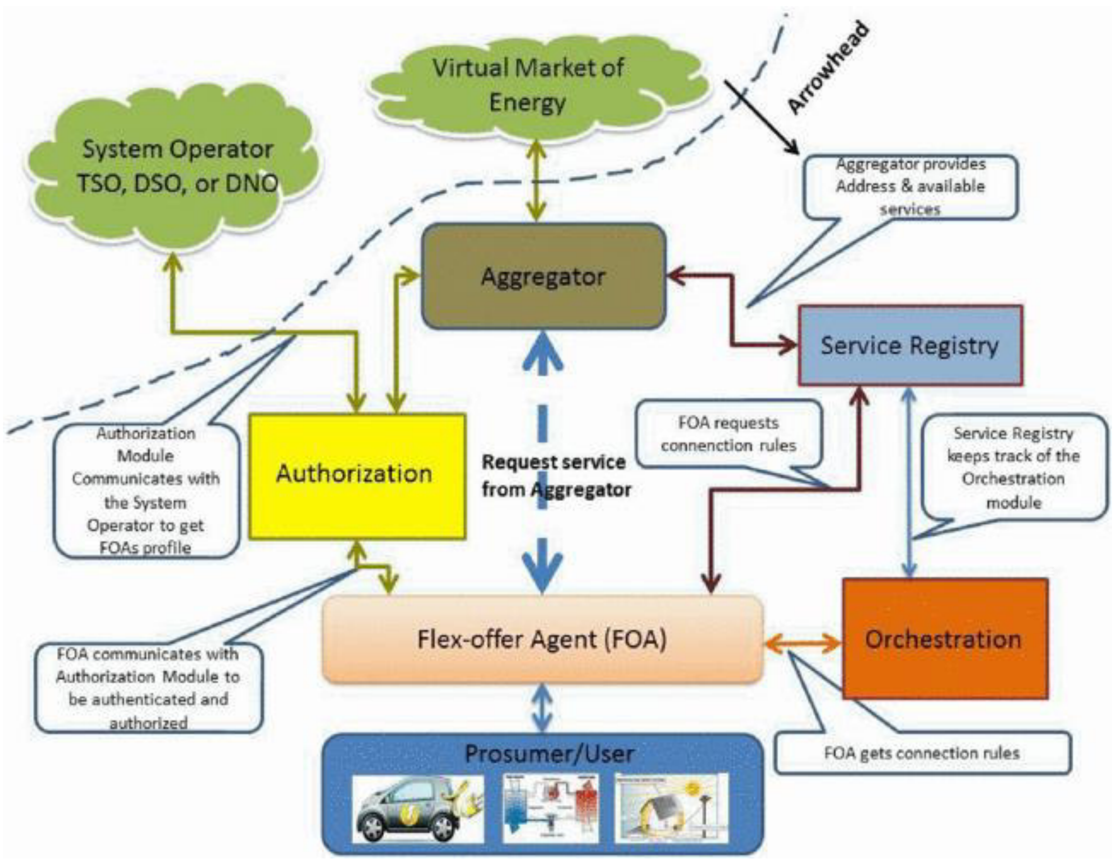

Virtual Market of Energy
Arrowhead
System Operator TSO, DSO, or DNO
Aggregator provides Address & available services
Aggregator
Service Registry
Authorization Module Communicates with the System Operator to get FOAs profile
Authorization
Request service from Aggregator
FOA requests connection rules
Service Registry keeps track of the Orchestration module
Flex-offer Agent (FOA)
Orchestration
FOA communicates with Authorization Module to be authenticated and authorized
Prosumer/User
FOA gets connection rules

Fig. 7 – High level architecture for the virtual market of energy. [12]

The described architecture is structured upon five modules, where three belong to the core Arrowhead framework – Service Registry, Authorization, and Orchestration, and the other two modules exchange business logic data – the Aggregator and the Flex-Offer Agent (FOA).

Flex-Offer Agents are basically software modules that offer the main functionalities to support the flex-offer concept. Its architecture already provides functionalities for getting information about the power consumption profile of specific devices, generate a flex-offer, and control the execution of a scheduled flex-offer. The design of this software also provides the necessary means to adapt to any platform by developing adequate interfaces among: 1) local and remote FOA modules; 2) with the controlled devices' hardware; 3) with other needed devices (e.g. a remote power meter) through a network; and 4) with external services (e.g. to obtain weather forecasts). Flex-Offer Agents are very flexible, its design allows for its total implementation to be running on a single device or distributed through several devices. As an example, the company providing the flex-offer service might give the user a specific hardware device, exclusively for the FOA, alternatively the FOA modules can be executed on existing devices on the prosumer premises. The main objective of the design being presented in this paper is to enable a high level of flexibility on the FOA implementation.

The Aggregators work by receiving flex-offers from FOAs, aggregating with flex-offers from other FOAs, into larger macro flex-offers and placing them to the Virtual Market of Energy. Note that only flex-offers larger than certain amount can be negotiated on existing electricity markets. Afterwards, the Aggregator receives a response from the Virtual market of Energy, disaggregates the response and sends a consumption schedule to the FOA. Several types of Aggregators might exist, and some Aggregators can be more specific for the control of electric motors while others can be more adequate for the control of heating systems. Additionally, choosing the most adequate Aggregator also depends on the geographical area.

To obtain the address of a proper Aggregator, a FOA uses the Service Registry module, to register itself, and the Orchestration module (to obtain an Aggregator that matches its criteria), both services are provided by the Arrowhead framework. The Service Discovery service has already two implementations, one using DNS Service Discovery (DNS-SD) [13] and another using Berkeley Internet Name Domain (BIND) [14].

The communication between the Aggregator and the FOA is implemented using XMPP, and exploits the existing Arrowhead framework services and their specific protocols to establish connection.

The main advantage of XMPP relies on its capabilities to support the Publish/Subscribe communication paradigm, which provides an asynchronous and highly scalable many-to-many communication model. The resulting decoupling between Publishers and Subscribers, in time, space and synchronization, simplifies the implementation of its associated software. Additionally, XMPP is also in a process of being standardized as a protocol for the control of Demand Response applications for OpenADR [15] and on the ISO/IEC/IEEE 21451–1-4 [16] standards.

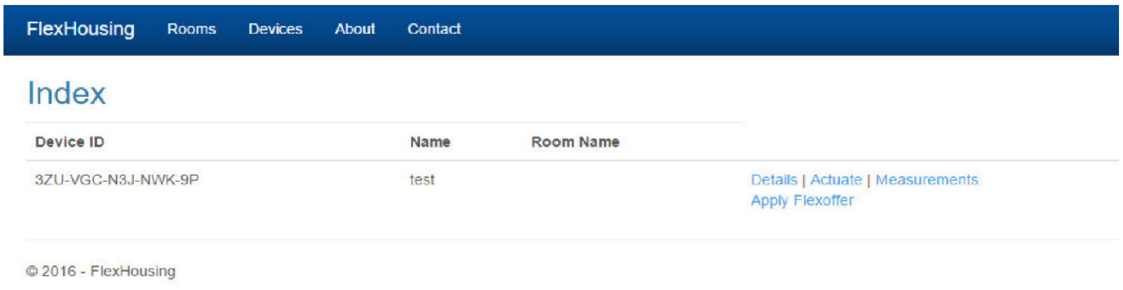
2.1.2.3 Communication between Arrowhead modules

Despite the previous version of the FlexHousing project already implementing the Arrowhead framework, the prototype had a severe dependency to servers, belonging to BNearIT, which were located in Denmark and offered a connection to the Aggregator module, the Service Registry, and the Virtual Market of Energy. For the FOA to communicate with the Aggregator, the machine running the prototype had to be connected to a VPN where the servers were found.

To prevent from situations where the VPN would be unavailable, it was decided that one of the major goals of the project had to be a local implementation and communication between the Aggregator module and the VME. With that, an XMPP server also had to be set up.

2.1.3 FlexHousing web platform’s user interface

As mentioned before, the UI of the original prototype’s web platform is not very appealing, nor elegant, nor intuitive, as seen in Figs. 8 and 9.

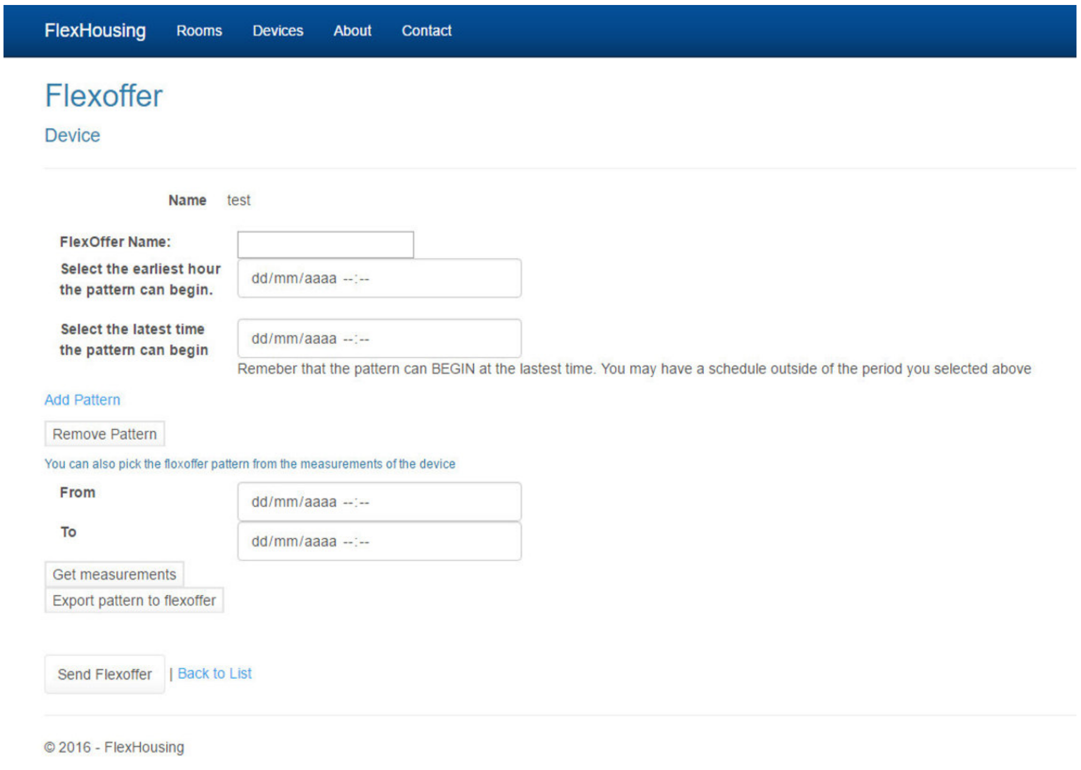


Index

Device ID	Name	Room Name	
3ZU-VGC-N3J-NWK-9P	test		Details | Actuate | Measurements Apply Flexoffer

© 2016 - FlexHousing

Fig. 8 – Device Index Page in the original FlexHousing prototype’s web platform



Flexoffer

Device

Name test

FlexOffer Name:

Select the earliest hour the pattern can begin. dd/mm/aaaa --:--

Select the latest time the pattern can begin dd/mm/aaaa --:--

Remeber that the pattern can BEGIN at the lastest time. You may have a schedule outside of the period you selected above

You can also pick the floxoffer pattern from the measurements of the device

From dd/mm/aaaa --:--

To dd/mm/aaaa --:--

© 2016 - FlexHousing

Fig. 9 – Flex-offer creation form in the original FlexHousing prototype’s web platform

The web platform should be able to easily convey all the necessary steps elegantly, without becoming bloated with various UI elements. Given that the concepts of energy management and flex-offer scheduling can be hard to grasp by an uninformed user, the UI should make the process more attractive, spontaneous, and easier to work with.

2.1.4 Specifying a flex-offer's energy consumption pattern

In the original prototype of FlexHousing platform, for the user to create a flex-offer and apply it to a device, the user would have to manually specify the energy consumption pattern of said device.

Naturally, if users do not know the device's usual consumption pattern, or do not want to be overly specific about the device's consumption, they will be confused and puzzled by the complexity of creating a flex-offer.

A solution to this issue is to develop a feature to make the system automatically create flex-offers, based on a device's energy consumption pattern.

2.1.5 Executives' platform

As mentioned before, the client proposed the idea of a platform for executives, to provide them some statistics about FlexHousing's users. These statistics could be the total amount of registered users, houses, and devices, or the number of times a brand of devices is used, or an average of the duration of use for every device brand, and so forth.

Additionally, another suggested feature for this platform was the ability to check the geolocation of every registered device.

2.2 Business areas

2.2.1 Smart Buildings, Smart Cities, and Smart Grids

The FlexHousing project, through the flex-offer concept, could have a profound impact on energy management as more intelligent devices are incorporated into buildings. As these devices can connect with each other while feeding data into analytics software, users gain a more complete picture of their energy usage. This insight might reveal that certain areas within a building are underutilized, so heating or cooling should only take place immediately before and during periods of occupancy [17].

Moreover, IoT devices could even start to communicate with external devices, such as a smart city's devices and the smart grid. These two areas are a hot topic in energy management throughout the last years. According to data published by the New Jersey Institute of Technology (NJIT), the smart city technology industry will generate revenues of more than

$27.5 billion by 2023. In addition, 88 cities worldwide will have adopted smart city technologies by 2025 [18].

To illustrate the advantages brought by smart cities, Fig. 10 describes some cases where a smart city system can improve a country's economics and sustainability.

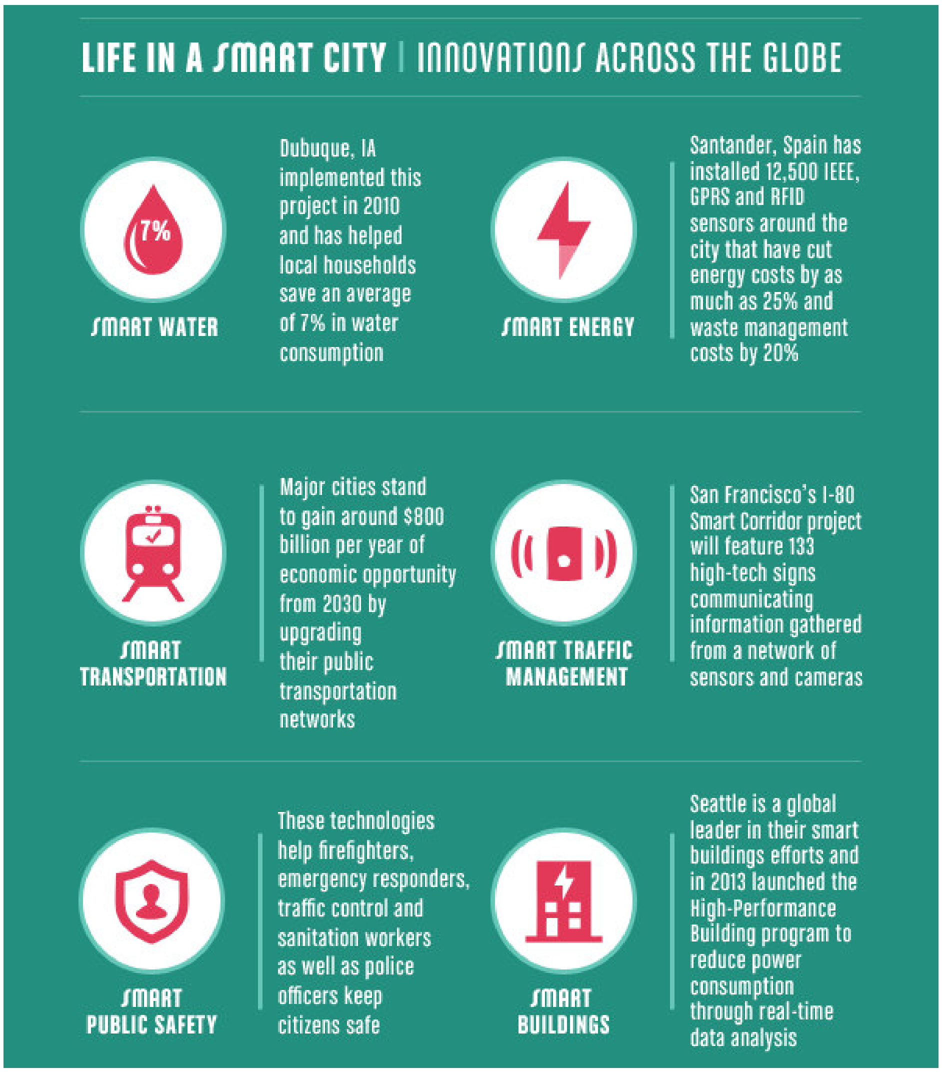

LIFE IN A SMART CITY | INNOVATIONS ACROSS THE GLOBE
7%
SMART WATER
Dubuque, IA implemented this project in 2010 and has helped local households save an average of 7% in water consumption
SMART ENERGY
Santander, Spain has installed 12,500 IEEE, GPRS and RFID sensors around the city that have cut energy costs by as much as 25% and waste management costs by 20%
SMART TRANSPORTATION
Major cities stand to gain around $800 billion per year of economic opportunity from 2030 by upgrading their public transportation networks
SMART TRAFFIC MANAGEMENT
San Francisco's I-80 Smart Corridor project will feature 133 high-tech signs communicating information gathered from a network of sensors and cameras
SMART PUBLIC SAFETY
These technologies help firefighters, emergency responders, traffic control and sanitation workers as well as police officers keep citizens safe
SMART BUILDINGS
Seattle is a global leader in their smart buildings efforts and in 2013 launched the High-Performance Building program to reduce power consumption through real-time data analysis

Fig. 10 – Innovations in sustainability through smart systems. [19]

In relation to Smart Grids, according to Chris King (Chief Regulatory Officer for eMeter, a Siemens Business), these empower consumers to save money by using less electricity or reducing peak consumption (studies indicate conservation of 5-10% and peak reduction of 10-20%). Furthermore, these can improve system reliability through reduced outages and faster restoration [20].

2.2.2 Energy analytics on businesses' equipment

In addition, areas like Retail, Industrial Production, Health Care, among others, can also benefit from this project's energy management solutions. For instance, smart devices and integrated systems can enable businesses to better understand their energy usage and costs in real-time, identify malfunctioning equipment and overlay data sets to draw previously invisible insights that can increase efficiency and reduce operating costs. In other words, an office building with smart lighting may look at the data and notice that conference room lighting is contributing to an unexpectedly high portion of the electricity bill. The company could then take steps such as installing sensors to activate the heating system in a conference room only when someone is in the room or create an intelligent schedule (in this case, a flex-offer schedule) that automatically turns the heating off when meetings are not scheduled. Without this type of data and reporting, companies lack the visibility required to realize efficiencies like this [17]. In summary, companies can employ energy analytics on their buildings to gather data and insights on their own operations.

2.2.3 Energy Markets

The energy markets consist of producers, transmitters and distributors, and consumers. The producers create the energy on power stations, which can operate on fossil or green origins. Furthermore, there are two types of electricity network: transmission and distribution [89]. Transmission networks carry electricity long distances around the country at high voltages. Distribution networks run at lower voltages and take electricity from the transmission system into homes and businesses [89].

According to the paper “Convergence to the European energy policy in European countries: Case Studies and Comparison” [21], society, as a consumer, is nowadays headed into becoming a low consumption economy, driven to use more competitive prices and greener energy. A European agreement of commitments known as “20/20/20” has set three new targets regarding energy for 2020:

- A minimum of 20% reduction in GHG emissions.
- 20% of energy production coming from renewable resources.
- 20% reduction in energy usage, by upping energy efficiency.

Since the focus of the FlexHousing project is about energy management, it could contribute to the goals of “20/20/20”.

Furthermore, by 2020, under EU legislation, 80% of consumers will need to have smart meters installed as part of a larger plan to help European nations meet energy-efficiency targets [21]. Once again, the FlexHousing project has the potential to help European nations to make profit of such infrastructure, when applied to consumers’ appliances.

2.3 State of the art

2.3.1 Flex-Offer-related Projects

The flex-offer concept used in this project has previously been explored in projects like MIRABEL (a European smart grid project that presents energy related data management solutions) [22], TotalFlex (a project under the ForskEL program – Energinet.dk's programme for supporting research and development within eco-friendly electricity production technologies) [10] [23], and, of course, Arrowhead [24].

Since this project uses Arrowhead as its framework, and because the report already goes in detail of Arrowhead’s specifics, there will be a bigger focus on the other two projects mentioned than on Arrowhead.

2.3.1.1 MIRABEL

The EU-funded research project MIRABEL (Micro Request-Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution) aims at developing a conceptual and infrastructural demand/supply response approach to enable a better utilization of renewable energy sources and a more flexible demand management. The core idea is that market players may express acceptable flexibilities for their energy demand and even specific supplies of so-called micro-request [25]. These micro-requests are called flex-offers. This system processes large amounts of flex-offers to balance electricity supply and demand in near real-time and thus supports the integration of non-schedulable renewable energy sources much better than earlier approaches [26].

Fig. 11 illustrates the advantages of the MIRABEL system. There, we see the energy consumption situation without and with the MIRABEL flexibility concept. The dark grey and shaded areas visualize the non-flexible and flexible demand respectively. The dotted line depicts the renewable energy production. With the help of the MIRABEL flex-offers, renewable energy sources can be better utilized by shifting energy demand through time to positions of large renewable production [25].

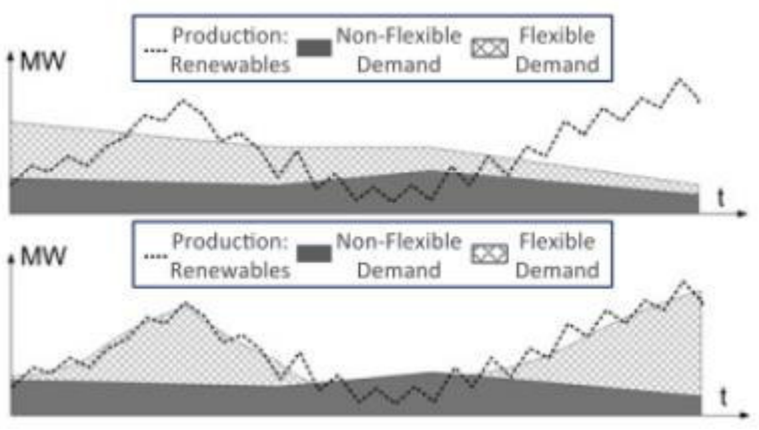


Fig. 11 – Balancing with and without the MIRABEL concept. [25]

Moreover, Fig. 12 exemplifies the lifecycle of a flex-offer from its inception to its execution and beyond. A flex-offer is submitted to the utility company and depending on its capacity, the utility company might accept or reject the flex-offer. In the case of acceptance, the utility company starts to schedule the flex-offer and, as time of execution approaches, assigns a fixed execution time slot. After the execution, the billing is conducted and depending on the benefit of the flex-offer for the utility company, an incentive is provided to the consumer, producer, or prosumer [25].

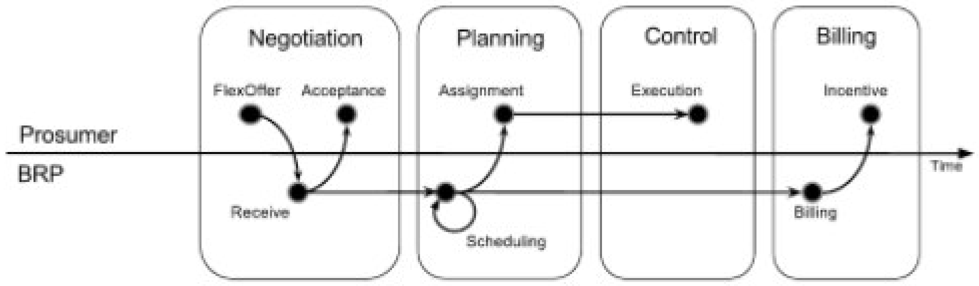


Fig. 12 – Flex-offer lifecycle. [25]

To realize the developed concepts, the MIRABEL project involves the design and implementation of an energy data management system, the EDMS. The EDMS exhibits a hierarchical architecture that is based on the hierarchy of the European electricity market. Each level of the hierarchy requires specific data in a certain granularity. This architecture can be seen in Fig. 13 [25].

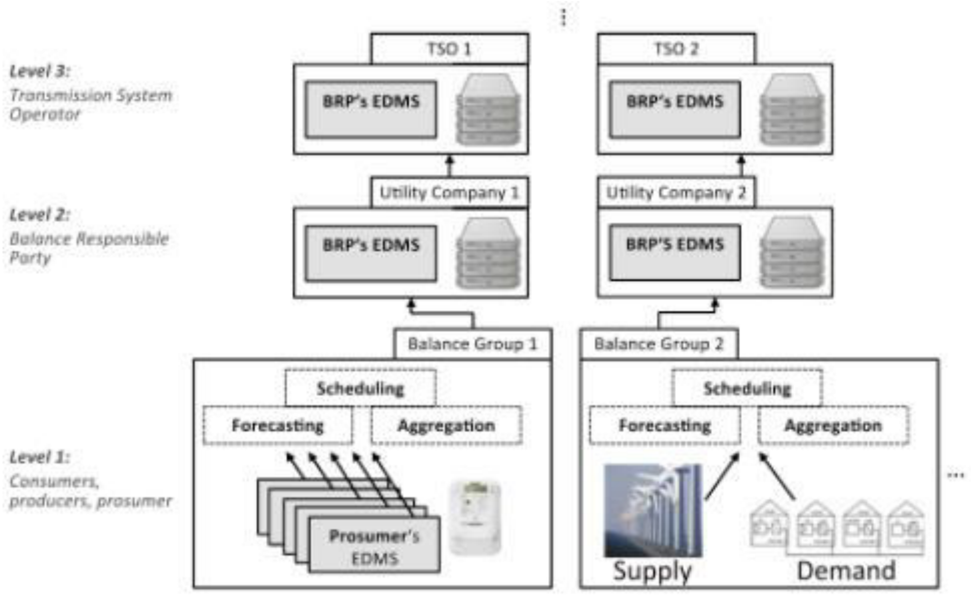


Fig. 13 – The EDMS of the MIRABEL project. [25]

Furthermore, the EDMS is designed to work as part of a single European electricity market, spanning the system over all European countries.

2.3.1.2 TotalFlex

The TotalFlex project aims to design a flexible, cost-effective power market system, which includes both flexible power consumption and flexible power production, all the while creating balance in the power distribution grid, making sure to avert bottlenecks and overloads [27].

TotalFlex implements a mechanism to express and utilize the notion of flexibility, using the concept of flex-offer proposed in the EU project MIRABEL. However, the vision of the TotalFlex project is that for users having a flex-offer contract with an energy supplier, their flexibility is not stated by the user, but instead predicted within the TotalFlex architecture based on past users' behavior [28].

Therefore, the TotalFlex project focuses on accurate flex-detection, flex-prediction, load-prediction, and automated generation of flex-offers. Flex-detection refers to the detection of available flexibility in device level energy consumption. Similarly, flex-prediction refers to the prediction of flexibility, for example, an EV with a max charging power level of 5kW is predicted to need 15kWh of energy with a time flexibility of 8 hours starting at 20:00 and ending at 04:00 of the next day. Finally, load-prediction refers to the prediction of aggregated and device level demand for the house, for instance, the predicted energy demand for house

X at 20:00 is 2kWh or the predicted energy demand for an oven in house X at 20:00 is 1.2 kWh [28].

2.3.1.3 Arrowhead

The Arrowhead project targets five business domains; Production (process and manufacturing), Smart Buildings and infrastructures, Electro mobility, Energy production and Virtual Markets of Energy. In these domains, there are several technological architectures used for implementing SOA solutions. One of the grand challenges of Arrowhead is to enable interoperability between systems that are natively based on different technologies. Naturally, one of its main objectives is to achieve that, thus keeping the advantages of SOA [29].

The Arrowhead framework (Fig. 14) includes a set of Core Services which can support the interaction between Application Services. The Core Services handle the support functionality within the Arrowhead framework to enable Application Services to exchange information.

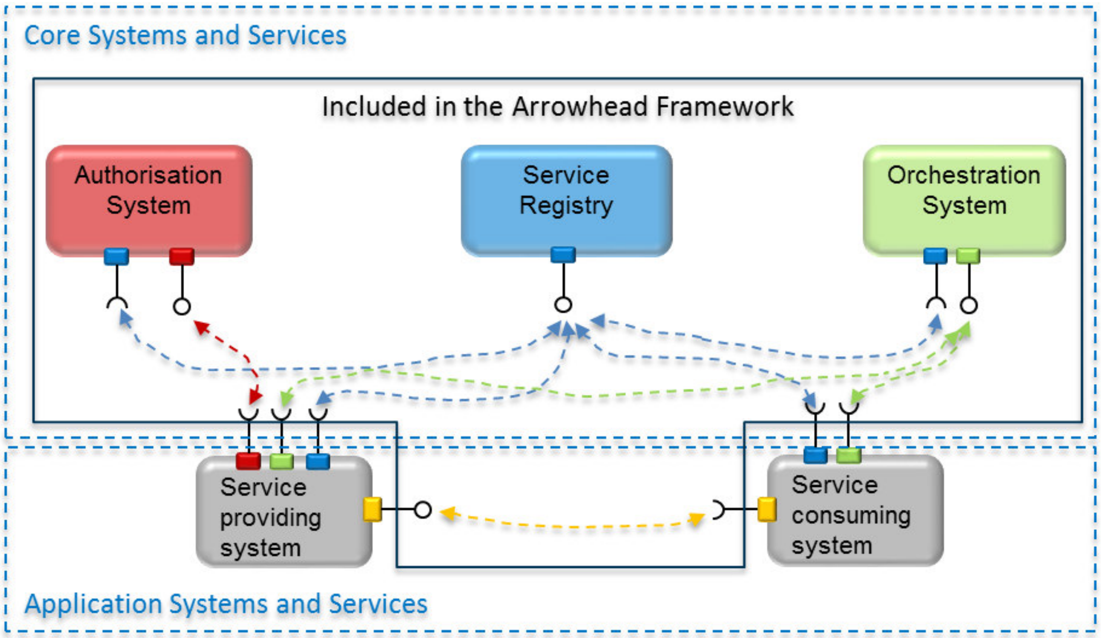


Fig. 14 – Arrowhead Framework System of Systems [30]

2.3.2 Home Automation platforms and Device interoperability

The home automation ecosystem has continually become more relevant and popular, with an increasing number of manufacturers developing devices for the IoT space. However, the smart home business is currently in a fractured state when it comes to compatibility between different third-party devices.

For starters, on Apple's end, there is HomeKit (Fig. 15): a framework for communicating with and controlling connected accessories in a user's home, built into the OS of any Apple device since iOS 8.1. HomeKit's goal is to give device makers a set of standards to build around. Users that comply with Apple's system will be able to enjoy seamless integration with Apple's mobile products, with other HomeKit-compatible gadgets, and with Siri, Apple's voice-activated AI assistant. Unfortunately, this is an approach that leaves out Android users, not to mention that many of those HomeKit gadgets won't work directly with non-HomeKit gadgets [31].

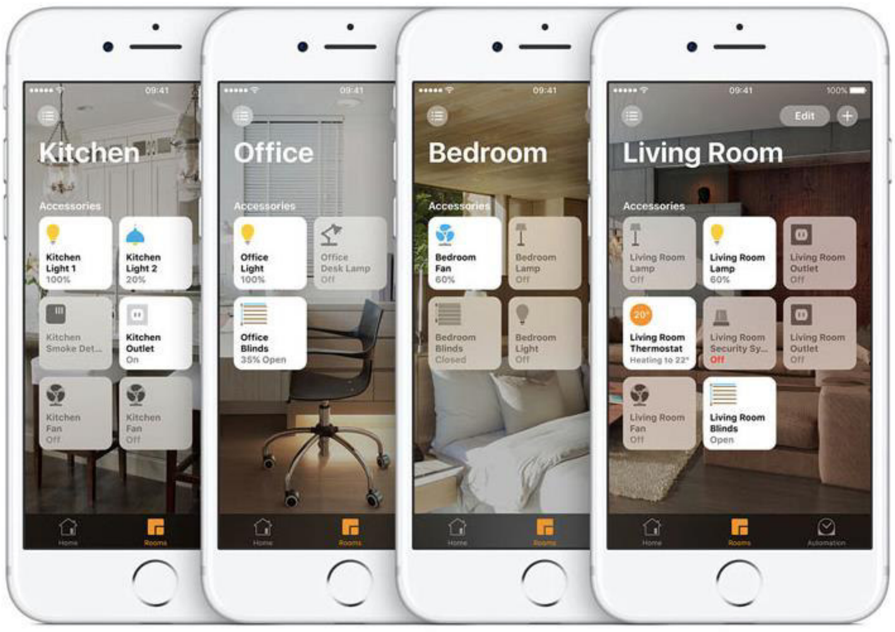


Fig. 15 – Apple HomeKit app UI [32]

From the Google-owned Nest Labs, there is the Nest Learning Thermostat, with more and more products joining up with it. Products willing to fall in line with Nest, and extend the thermostat's usefulness, get a share of its popularity, and many of them integrate directly into the Nest app (Fig. 16), which is available on iOS and Android. However, like with HomeKit, not every device is compatible with Nest [31].

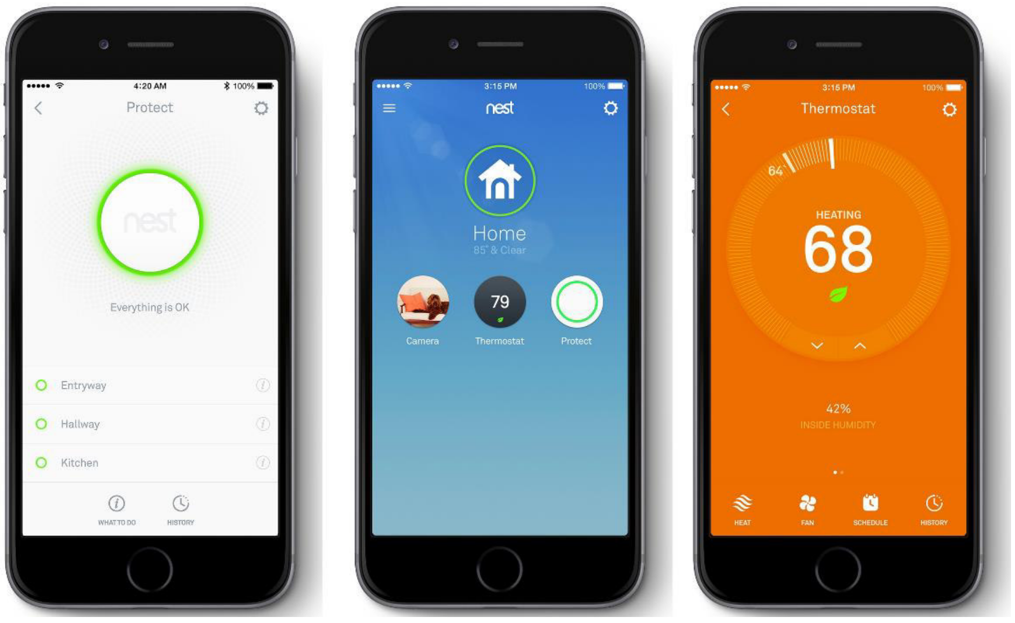


Fig. 16 – Nest app UI [33]

Aside from these two, there are also smaller control platforms like IFTTT, Insteon, SmartThings, among others. IFTTT, for instance, is a popular one. IFTTT is a free web-based service used to create chains of simple conditional statements, called applets (Fig. 17) [34]. An applet is triggered by changes that occur within other web services, from more social oriented ones such as Gmail, Twitter, LinkedIn, or Facebook, to more appliance specific web services like LG Washer, Whirlpool Refrigerator, WeMo Coffeemaker, WeMo Smart Plug, or Philips Hue lights. For example, by tapping into the Nest IFTTT service, we can sync up Nest devices with other IFTTT-compatible devices that wouldn't work directly with Nest otherwise [31]. However, IFTTT, like other platforms, only works with devices that are part of distributor partnerships.

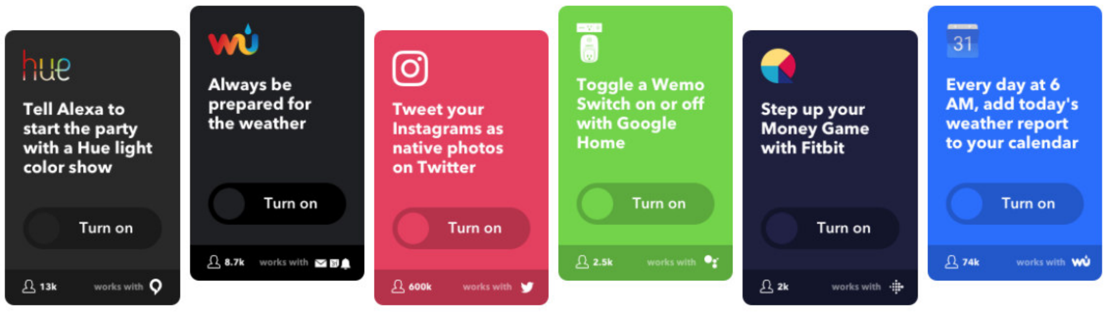

hue
Tell Alexa to start the party with a Hue light color show
Turn on
13k
works with
Always be prepared for the weather
Turn on
8.7k
works with
Tweet your Instagrams as native photos on Twitter
Turn on
600k
works with
Toggle a Wemo Switch on or off with Google Home
Turn on
2.5k
works with
Step up your Money Game with Fitbit
Turn on
2k
works with
31
Every day at 6 AM, add today's weather report to your calendar
Turn on
74k
works with

Fig. 17 – Examples of applets for different web services in IFTTT. [35]

Then there are also Amazon Echo and Google Home smart speakers, which offer voice-activated control of certain smart-home gadgets by way of AI assistants, Alexa and Google

Assistant (respectively). These are continuing to work with more and more devices, positioning these assistants as rivals to Apple's Siri.

Beyond all these platforms, many home automation devices like Portugal-based VPS's Cloogy (Fig. 18) or other low-cost devices can only be accessed by their own interfaces (website or mobile app), with no compatibility with other services.

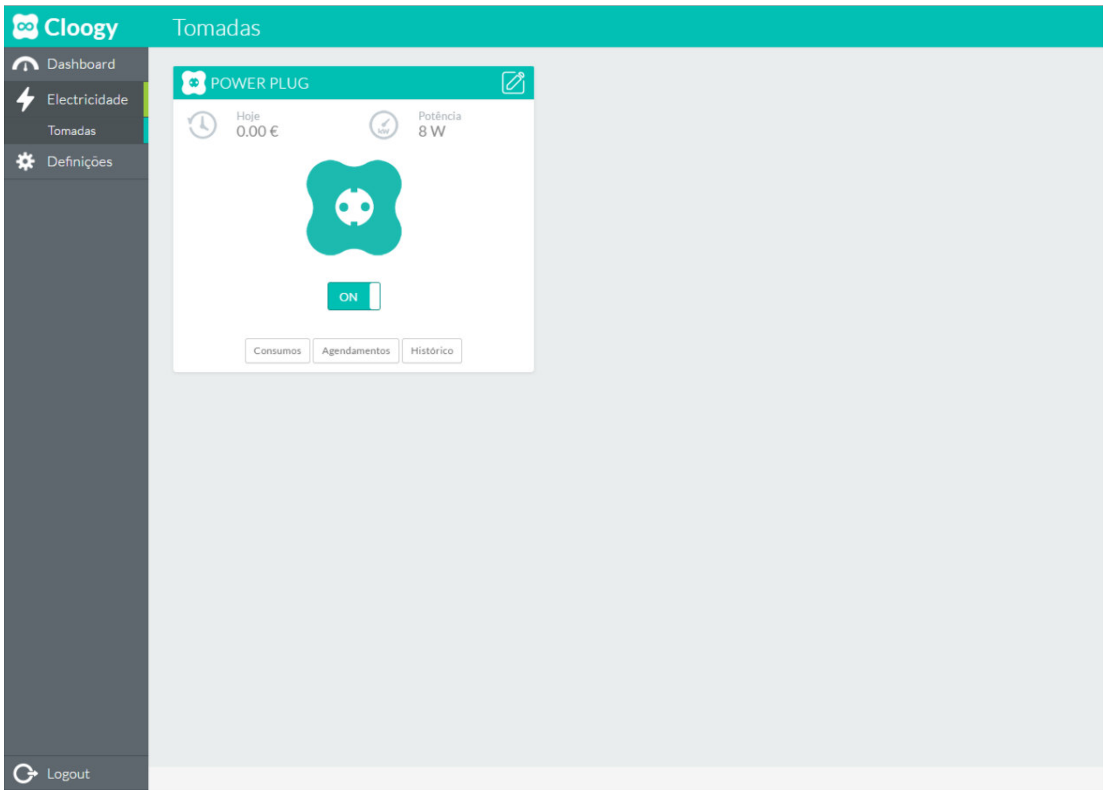


Fig. 18 – VPS's Cloogy UI.

Lastly, there are also many open-source platforms like OpenHAB or Home Assistant with an increasing number of compatible devices, and with options to run on everything, from an always-on personal computer to a Raspberry Pi. However, some of the device compatibility bindings could have been implemented by reverse engineering protocols, which might cause some legal risks when being used commercially [36].

OpenHAB (Fig. 19), written in Java, is designed to be device-agnostic while making it easier for developers to add their own devices or plugins to the system. It also ships iOS and Android apps for device control, as well as a design tools so you can create your own UI for your home system [37].

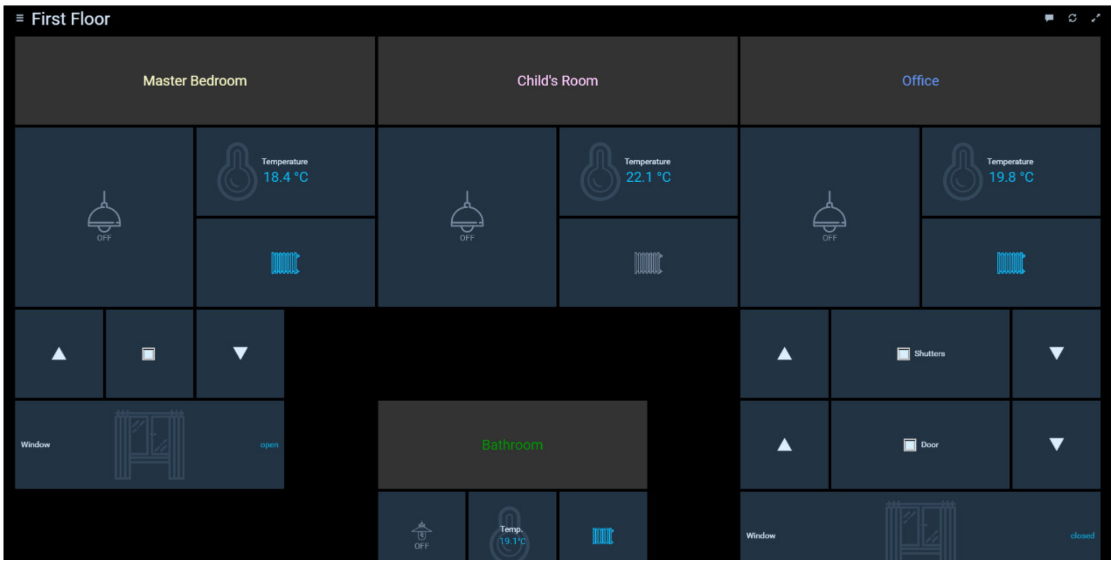


Fig. 19 – One of many UIs available in OpenHAB. [38]

Home Assistant (Fig. 20), running on Python 3, integrates with several open-source as well as commercial offerings, allowing the user to link, for example, IFTTT, weather information, or an Amazon Echo device, to controls from locks to lights to even a command line notifier [37].

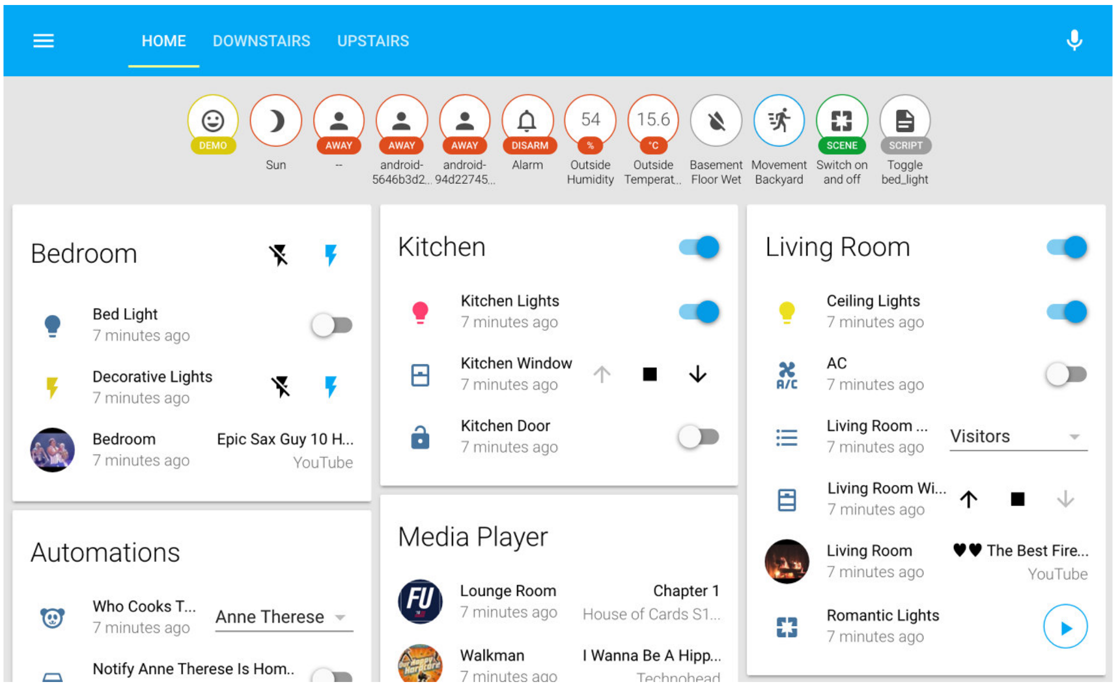


Fig. 20 – Home Assistant app UI. [39]

In conclusion, currently it isn't possible to have a unified setup that one could program from a single app or control from a single voice-control platform, thus no single home automation platform can claim to function with every device (at least not yet).

2.4 Solution Overview

2.4.1 Data and Device Interoperability

To handle the communication process between software applications and smart devices, APIs (Application Programming Interfaces) are the mainstay solution. In fact, APIs are strongly linked to IoT because they can securely expose connected devices to customers and other applications in an IT infrastructure [78]. The use of APIs makes working with IoT a more accessible experience, as software developers can work with connected “things” without needing to know its intricacies or protocol [79]. APIs solve this challenge successfully, since they abstract the “thing” and expose it as an interface.

However, there are numerous different types of APIs. In this case, several communication protocols like HTTP, MQTT, XMPP, could be a possible choice for the communication interface between the FlexHousing Middleware and the multiple types of devices. Since the original FlexHousing prototype already used the REST (HTTP) architectural style as a basis for device interactions, this groundwork was reused for the development of the new version of FlexHousing, so it could be expanded upon. Nevertheless, the use of other communication protocols is always a possibility.

Anyhow, REST, while just a concept and not a protocol, is the foundation of the most widely used form of API today [40]. REST is a good model for IoT because each device can easily make its state information available, and can standardize on a way to create, read, update, and delete that data [40]. It also does not maintain a constantly open connection, so it is very scalable [40].

Additionally, the past version of the FlexHousing Middleware would only be able to operate with devices from VPS, this version of the Middleware has been modified to easily support other types of devices without having to change the main source code.

To add support to another device, the new device only has to have API endpoints to these basic features: actuate and read consumption values. Through those endpoints, the Middleware will be able to connect to the device and carry out its methods.

Therefore, to implement and truly test this feature, two generic customizable switches (named Sonoff Pow), in conjunction with the Cloogy smart plug and transmitter, were used on multiple devices. This way, the Middleware can display its ability to handle both situations: a connection to a device via its service provider’s cloud; and direct access to a specific device in the local network. Regarding the Sonoff sensors, to acquire specific energy consumption data and provide them through a REST API, a custom firmware was developed and deployed into these switches.

So, given the proposed solution, the project's structure can be observed in the Component Diagram present in Fig. 21, followed by a description of every component in the diagram.

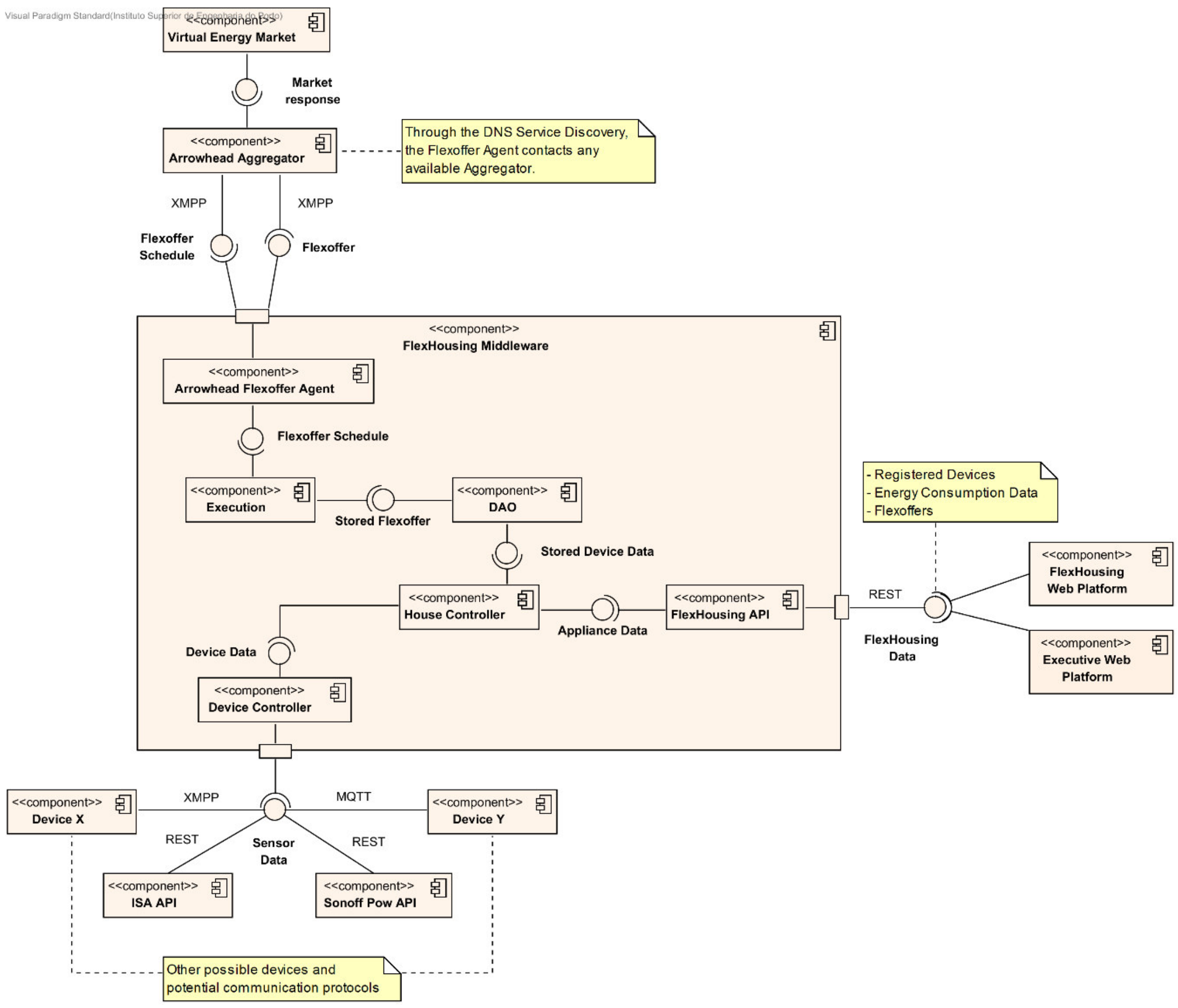

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
<<component>> Virtual Energy Market
Market response
<<component>> Arrowhead Aggregator
Through the DNS Service Discovery, the Flexoffer Agent contacts any available Aggregator.
XMPP
XMPP
Flexoffer Schedule
Flexoffer
<<component>> FlexHousing Middleware
<<component>> Arrowhead Flexoffer Agent
Flexoffer Schedule
<<component>> Execution
Stored Flexoffer
<<component>> DAO
Stored Device Data
<<component>> House Controller
Appliance Data
<<component>> FlexHousing API
Device Data
<<component>> Device Controller
- Registered Devices
- Energy Consumption Data
- Flexoffers
REST
FlexHousing Data
<<component>> FlexHousing Web Platform
<<component>> Executive Web Platform
<<component>> Device X
XMPP
MQTT
<<component>> Device Y
REST
Sensor Data
REST
<<component>> ISA API
<<component>> Sonoff Pow API
Other possible devices and potential communication protocols

Fig. 21 – Component Diagram of the FlexHousing project

- **FlexHousing Middleware:** This component is responsible for bridging the users' house devices with the Arrowhead Framework's flex-offer system, while also being able to perform operations like device actuations and acquiring their consumption data, and provide this information to the FlexHousing web platform.

  - **FlexHousing API:** Provides the system's data (registered devices, energy consumption data, flexoffers) to the FlexHousing web platform through a REST interface;
  - **House Controller:** Controls every house, including its associated rooms and devices, and provides their data to the FlexHousing API;
  - **Device Controller:** Manages the communication between the FlexHousing Middleware and the users' devices. Although this project will focus on communication via REST (HTTP), additional communication protocols could be implemented in the future;
  - **DAO:** Manages the database, storing every necessary data and providing it to other components in the system;
  - **Execution:** Carries out the system's automatic processes such as flexoffer emission and flexoffer schedule execution;
  - **Arrowhead Flexoffer Agent:** Offers the main functionalities to support the flex-offer concept, handling the generation of Flex-offers and sending them to the Aggregator via XMPP.
- **FlexHousing Web Platform:** Accesses the data from the FlexHousing Middleware (via REST) and presents it to the users;
- **Executive Web Platform:** Like the FlexHousing web platform, it accesses the data from the FlexHousing Middleware (via REST), however this data is specifically for company executives;
- **ISA API:** Provides data from all VPS devices' sensors via a RESTful API, through their remote servers;
- **Sonoff Pow API:** Provides the Sonoff sensor's data via a RESTful API (through the Sonoff's embedded web server) available in the local network;
- **Device X/Y:** Other potential devices with different communication protocols, that could be added in the future;
- **Arrowhead Aggregator:** Once given a flex-offer, this component handles the flex-offer's scheduling, based on the Virtual Energy Market's response;
- **Virtual Energy Market:** Handles the energy market prices, informing the Aggregator of their current state (through XMPP).

Furthermore, in view of the presented Component Diagram, a Deployment Diagram pertaining to the project's structure can be seen in Fig. 22.

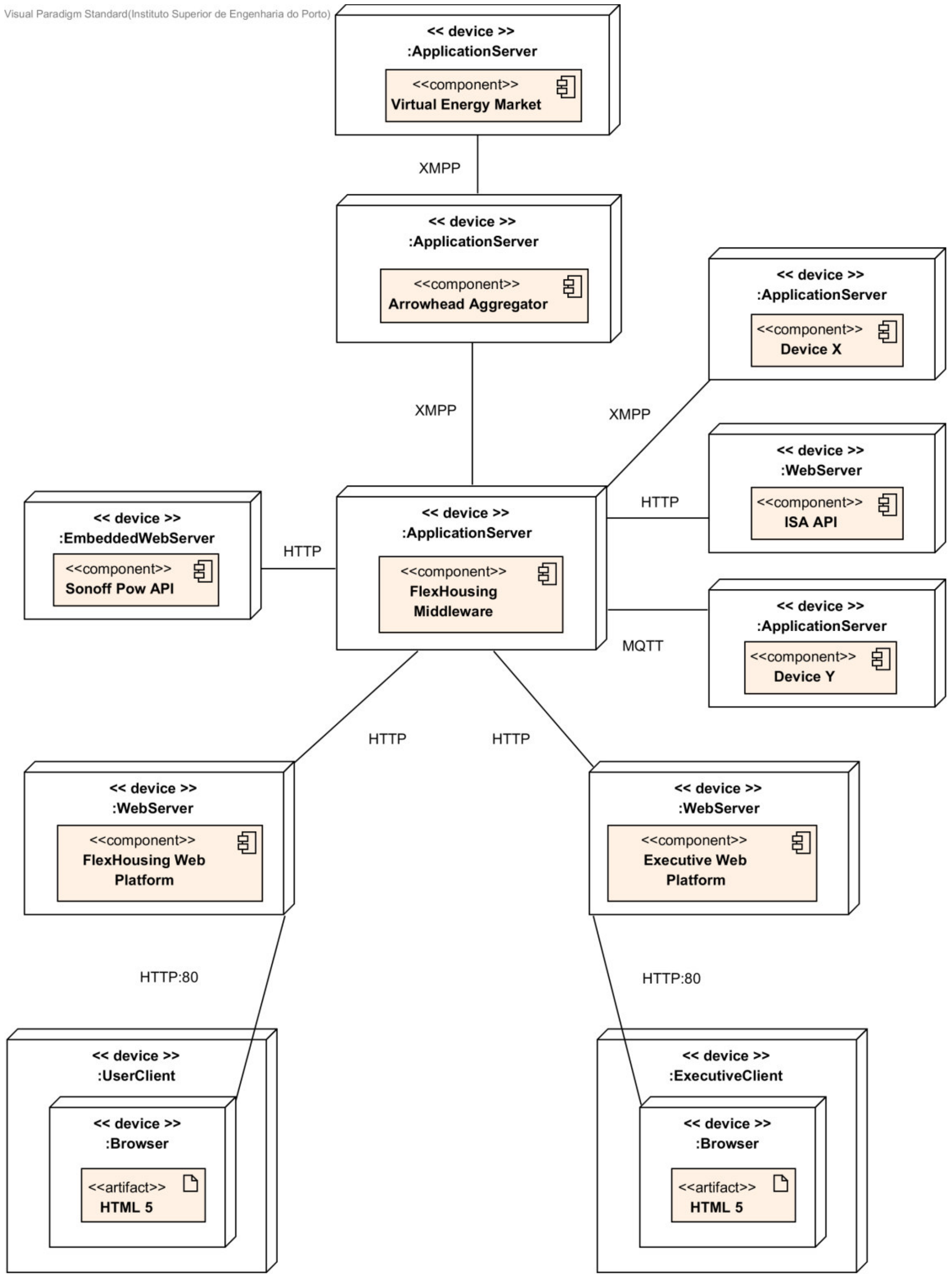

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
<< device >> :ApplicationServer
<<component>> Virtual Energy Market
XMPP
<< device >> :ApplicationServer
<<component>> Arrowhead Aggregator
XMPP
<< device >> :ApplicationServer
<<component>> Device X
XMPP
<< device >> :WebServer
<<component>> ISA API
HTTP
<< device >> :EmbeddedWebServer
<<component>> Sonoff Pow API
HTTP
<< device >> :ApplicationServer
<<component>> FlexHousing Middleware
<< device >> :ApplicationServer
<<component>> Device Y
MQTT
HTTP
HTTP
<< device >> :WebServer
<<component>> FlexHousing Web Platform
<< device >> :WebServer
<<component>> Executive Web Platform
HTTP:80
HTTP:80
<< device >> :UserClient
<< device >> :Browser
<<artifact>> HTML 5
<< device >> :ExecutiveClient
<< device >> :Browser
<<artifact>> HTML 5

Fig. 22 – Deployment Diagram of the FlexHousing project

2.4.2 Local communication between Arrowhead modules

As mentioned before, the past version of the project had a severe dependency on servers outside of Portugal (located on Denmark) that offered a connection to the Aggregator module, the Service Registry, and the Virtual Market of Energy (VME).

In order to break from that dependency, the solution was to locally implement the Aggregator and VME modules (even though the local VME only serves as a test module, since it just sends dummy data and not actual energy market data). Moreover, it was also necessary to install an XMPP server so that these modules could be able to communicate with each other and with the FlexHousing Middleware. A representation of the communication procedures between these modules can be viewed in Fig. 23:

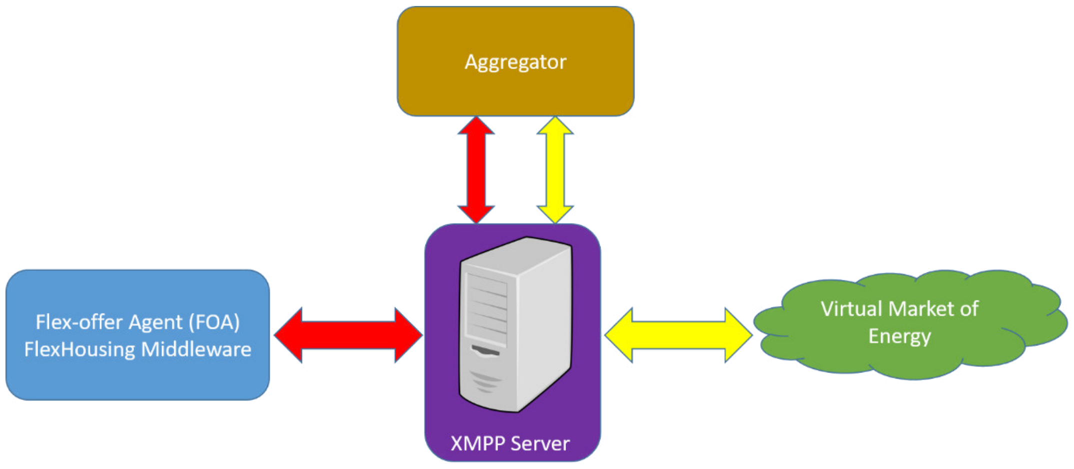


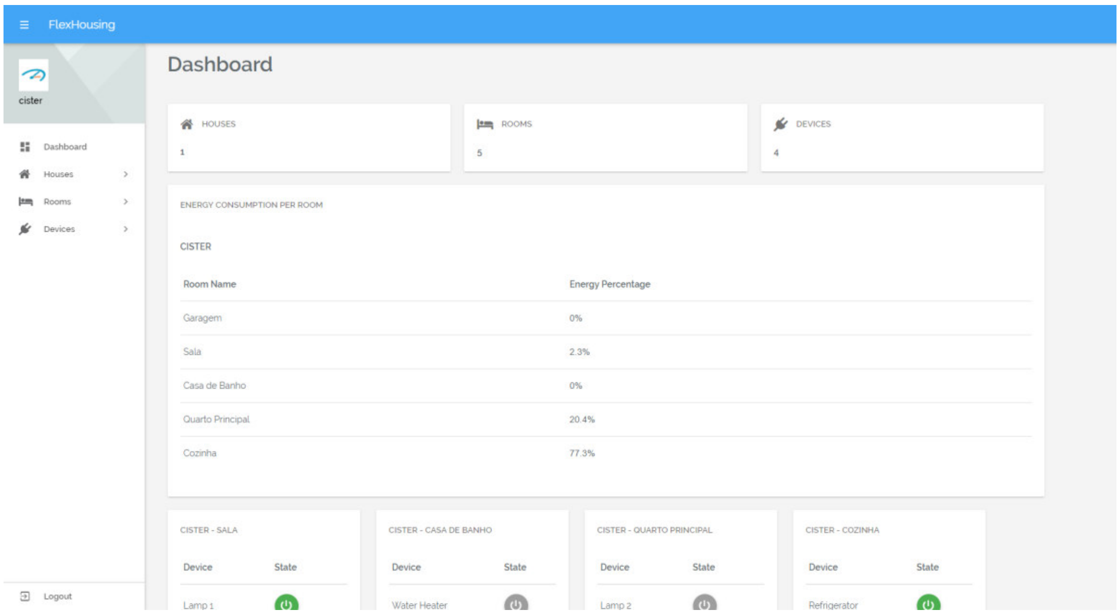
Fig. 23 – Communication between modules through an XMPP server.

Furthermore, this system should follow a configuration file (Appendix-A) that specifies the XMPP server’s hostname, port, resource, and service name, and each module’s XMPP client account’s ID and password.

2.4.3 FlexHousing web platform’s user interface

For the FlexHousing web platform’s UI and overall user experience, the focus is to make the device/energy management and the flex-offer creation process as appealing and intuitive as possible. To demonstrate this approach, this section presents a guided tour throughout the new platform.

First, after logging in, the user will be shown the Dashboard page (Fig. 24). Here, the user can check the current number of registered houses, rooms, and devices, check the energy consumption percentage of each room, and activate/deactivate a registered device.



Dashboard

HOUSES	ROOMS	DEVICES
1	5	4

ENERGY CONSUMPTION PER ROOM

CISTER

Room Name	Energy Percentage
Garagem	0%
Sala	2.3%
Casa de Banho	0%
Quarto Principal	20.4%
Cozinha	77.3%

CISTER - SALA

Device	State
Lamp 1	🟢

CISTER - CASA DE BANHO

Device	State
Water Heater	⚪

CISTER - QUARTO PRINCIPAL

Device	State
Lamp 2	⚪

CISTER - COZINHA

Device	State
Refrigerator	🟢

Fig. 24 – Dashboard page in the new FlexHousing web platform

Next, the user can then use the side-navigator (Fig. 25) to add a house, room, or device, as well as to check all the registered devices. As an example, Fig. 26 presents the “Add Device” form, while Fig. 27 shows the page displaying all registered devices.

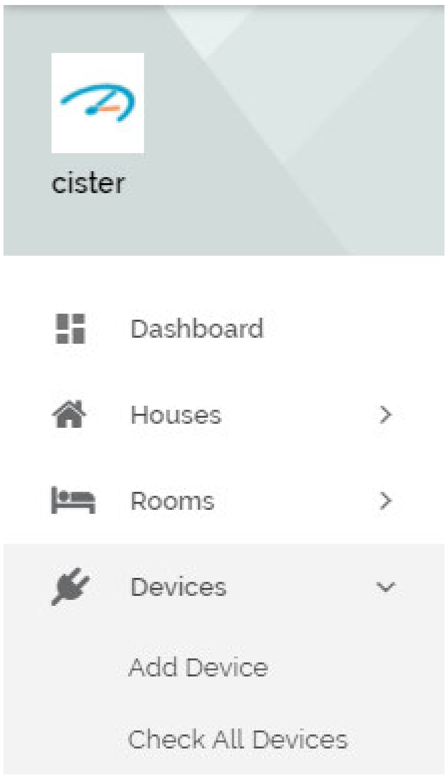


Fig. 25 – Side-Navigator in the new FlexHousing web platform

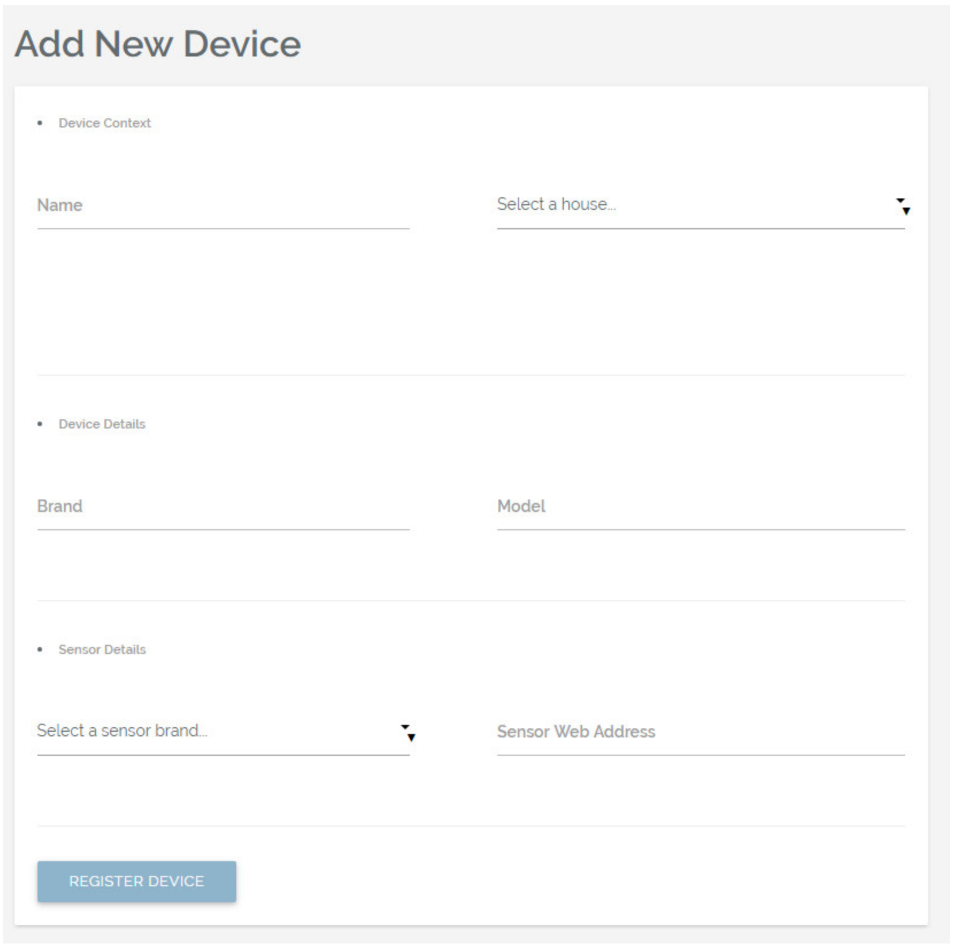


Fig. 26 – "Add Device" form in the new FlexHousing web platform

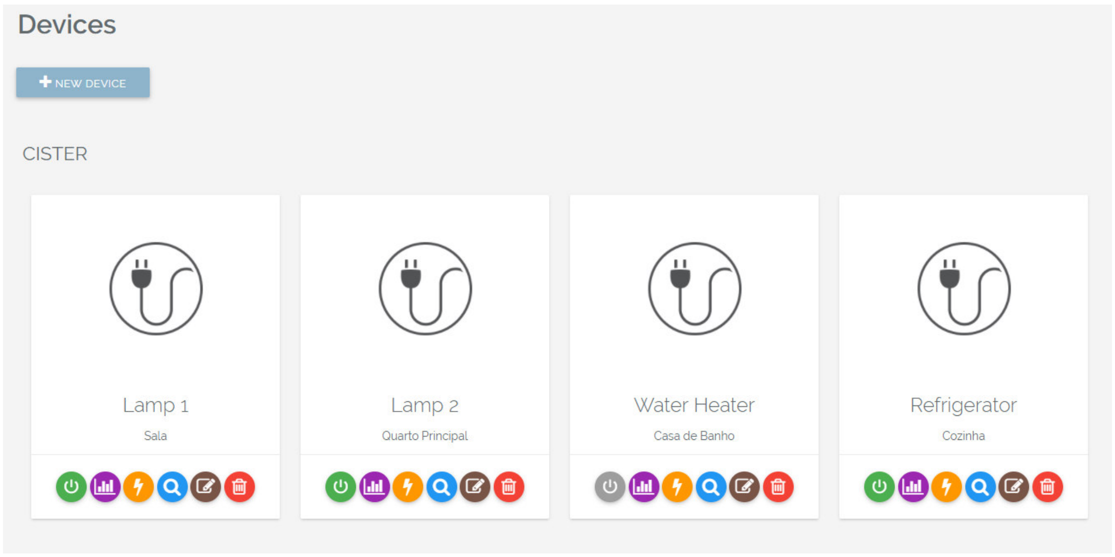


Fig. 27 – Device index page in the new FlexHousing web platform

When checking all devices, the user can select to create a flex-offer for the device. In the Flex-Offer creation form (Fig. 28), the schedule and the definition of the time window of a flex-offer is set by dragging and dropping the flex-offer event in a calendar with the mouse (Fig. 29).

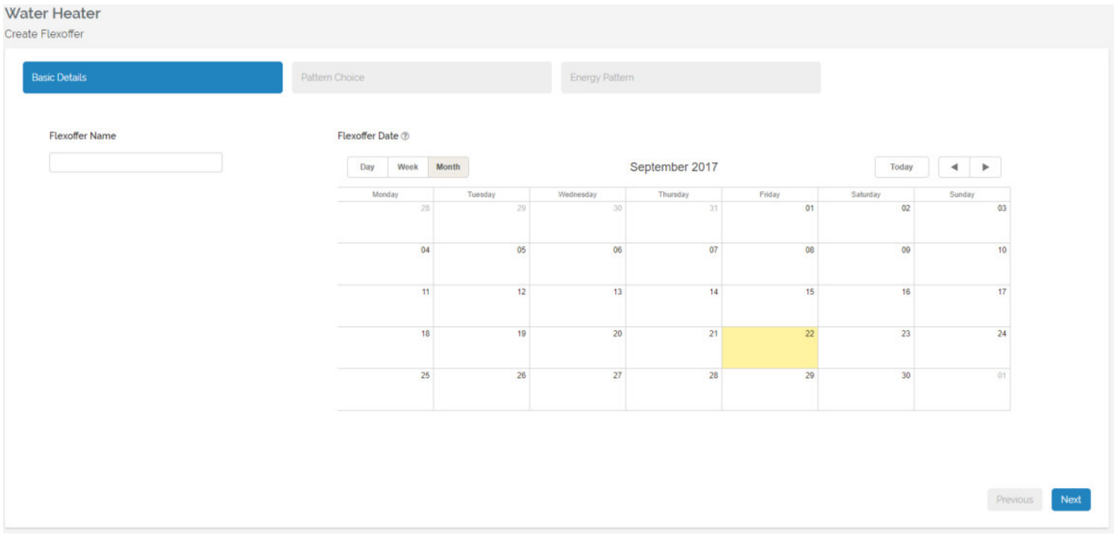


Fig. 28 – Flex-offer creation form in the new FlexHousing web platform

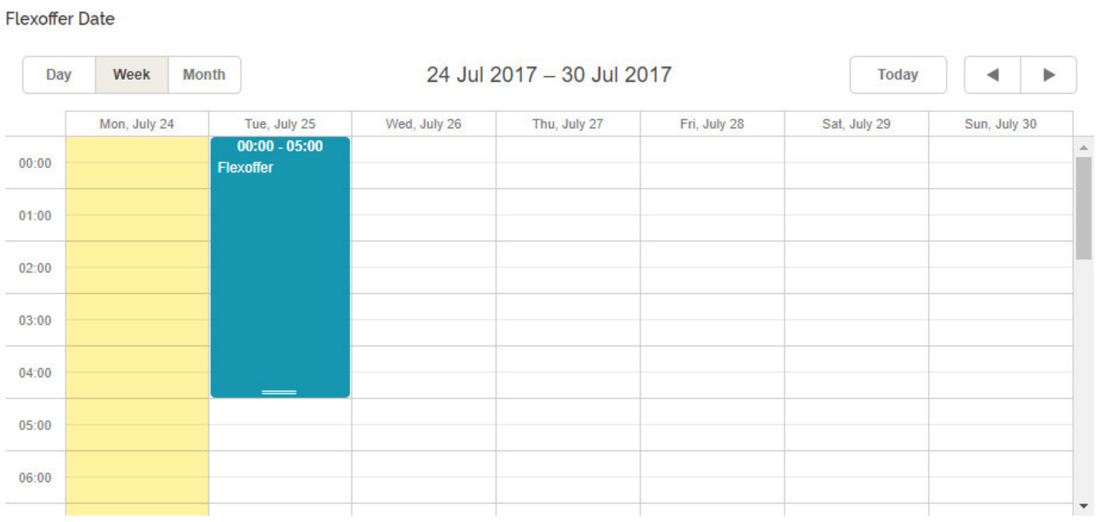


Fig. 29 – Scheduling a Flex-offer in the new FlexHousing web platform.

As for the creation of the energy consumption pattern for the flex-offer, the user has two options, a more experienced or expert user can do it manually or other users will do it automatically by running advanced pattern discovery algorithms.

By deciding to do it manually, the user can determine the duration of the pattern by clicking on a dropdown menu, and then define the energy consumption of each time segment by dragging the bars in a chart with the mouse (Fig. 30).

If doing it automatically, the user only has to specify the hour the device is usually activated, and then the system will present him with a flex-offer, which can be accepted, rejected or changed by the user (Fig. 31).

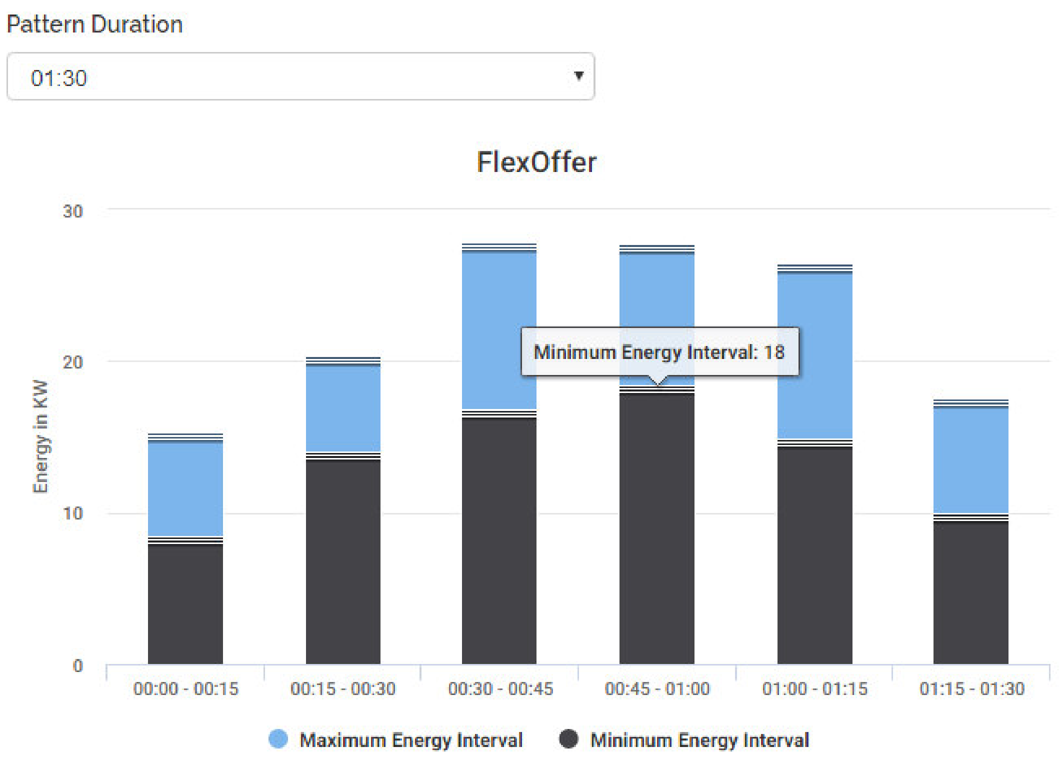

Pattern Duration
01:30

Fig. 30 – Manually creating the consumption pattern for a Flex-offer in the new FlexHousing web platform.

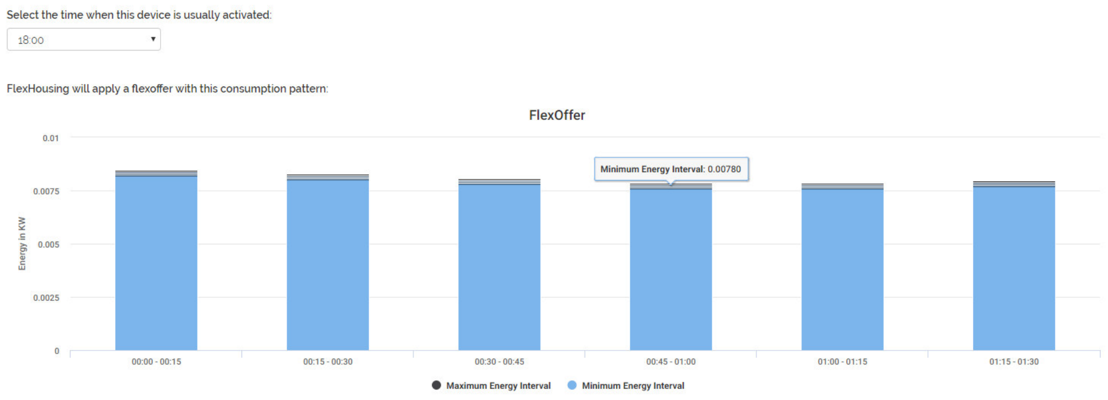

Select the time when this device is usually activated:
18:00
FlexHousing will apply a flexoffer with this consumption pattern:

Fig. 31 – Automatically creating the consumption pattern for a Flex-Offer in the new FlexHousing web platform.

Besides the Flex-offer creation, the user can also check the device's energy consumption (Fig. 32), and check a device's active flexoffer (Fig. 33) to see if it has had an effect in the device's consumption pattern.

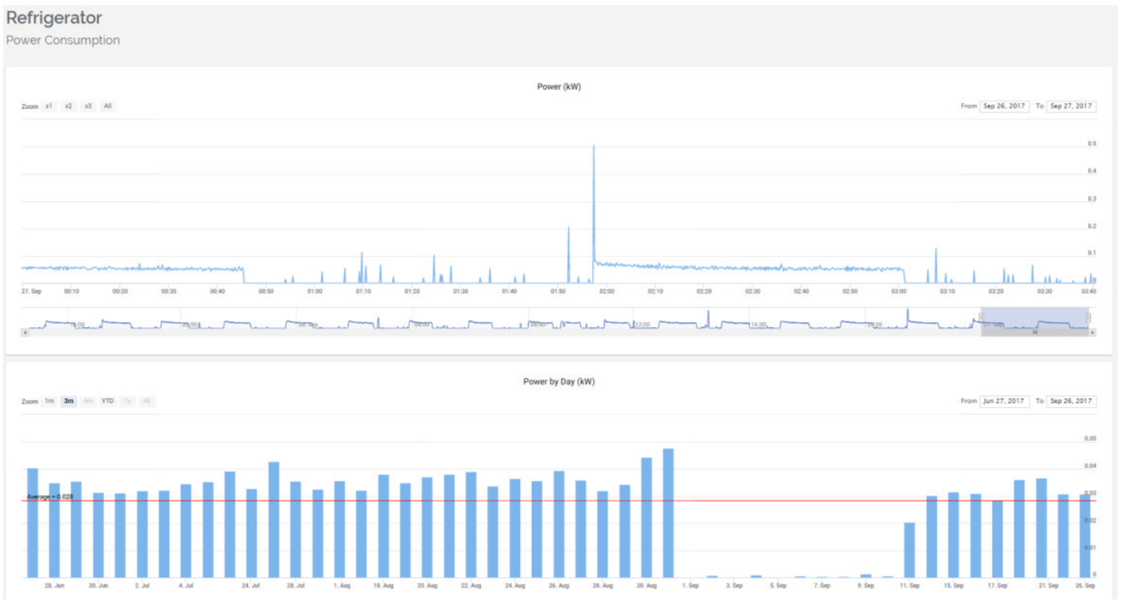


Fig. 32 – Checking a device's consumption in the new FlexHousing web platform

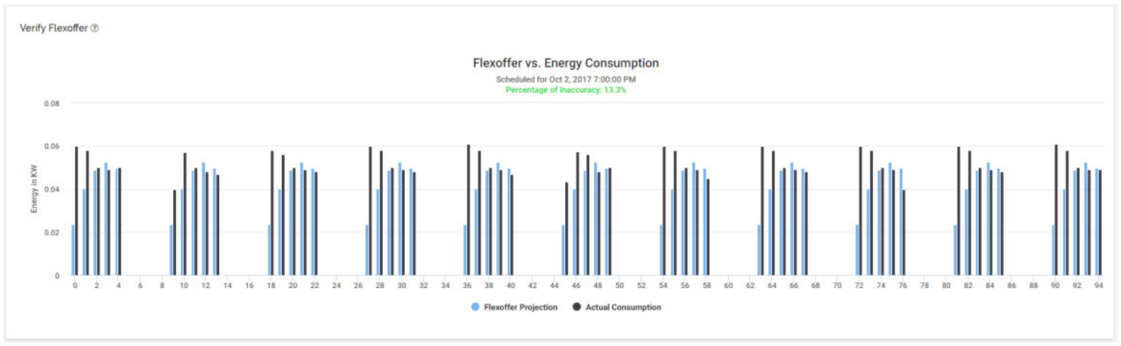

Verify Flexoffer
Flexoffer vs. Energy Consumption
Scheduled for Oct 2, 2017 7:00:00 PM
Percentage of Inaccuracy: 13.3%
Energy in KW
● Flexoffer Projection
● Actual Consumption

Fig. 33 – Verifying the effectiveness of a flexoffer in the new FlexHousing web platform

2.4.4 Specifying a flex-offer's energy consumption pattern

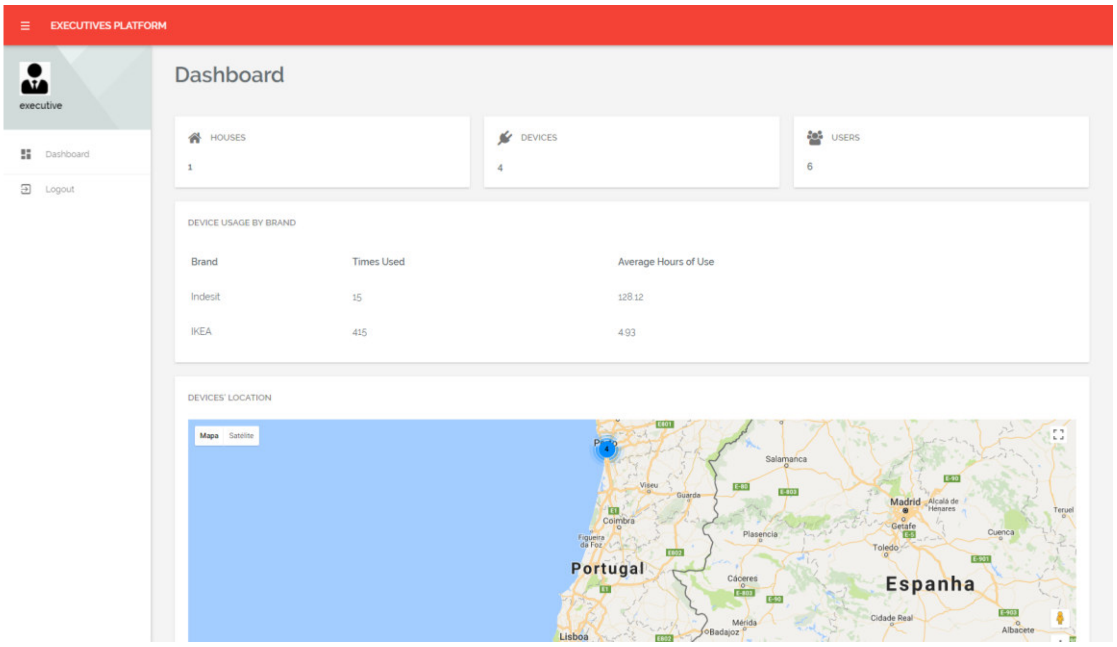
To solve the issue of users being discouraged due to the complex process of creating a flex-offer, it was decided to develop a feature to make the system automatically create the flex-offer's energy pattern, based on the device's consumption pattern.

To do so, certain algorithms from the paper "Generation and Evaluation of Flex-Offers from Flexible Electrical Devices" [65] will be used to implement a way to identify energy patterns. For the full explanation of these algorithms, section 4.4.4 in the Implementation chapter explains every step of the entire process.

2.4.5 Executives’ platform

For the executives’ platform (Fig. 34), the UI will be similar to FlexHousing’s with some slight changes. As mentioned before, the client was interested in seeing if it was possible to have a platform that could show some statistics and metadata about the registered users and devices.

Thus, the developed platform displays the total number of registered houses, devices, and users, while also showing the total number of times a device brand is used and its average hours of use. Since this project wants to provide proof of concepts for all interesting capabilities of a FlexHousing implementation, while the device geolocation feature was not part of a use case, a placeholder demonstration was added to the platform.



Dashboard

HOUSES
1

DEVICES
4

USERS
6

DEVICE USAGE BY BRAND

Brand	Times Used	Average Hours of Use
Indesit	15	128.12
IKEA	415	4.93

DEVICES' LOCATION

Fig. 34 – Executives’ platform UI

3 Work Environment

This chapter describes how the work was planned, what kind of work methodology was used for development, all the attended meetings in the internship, and the technologies used to achieve the proposed solution.

3.1 Work Methodology

3.1.1 Development Process

For the development of this project, an iterative work methodology similar to the Rational Unified Process (RUP) was used.

Essentially, RUP is a Software Engineering Process. According to IBM Rational Software Corporation, "it provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end-users, within a predictable schedule and budget" [41].

RUP provides the guidelines, templates and tool mentors necessary for a team to take full advantage of among others the following best practices [41]:

1. Develop software iteratively
2. Manage requirements
3. Use component-based architectures
4. Visually model software
5. Verify software quality
6. Control changes to software

According to the 4 phases of RUP, the project was distributed in the following way:

1. Inception – Research and code analysis. A study was employed to search for viable technologies to implement for the project.
2. Elaboration – Code design, such as documentation for functional and non-functional requirements.
3. Construction – Implementation of all the use cases, with continuous testing.

4. Transition – Some prototypes were developed and showcased to multiple stakeholders, resulting in updates, improvements, and the implementation of new features.

The development process can be described in Fig. 35. The horizontal axis represents time and shows the dynamic aspect of the process as it is enacted, and it is expressed in terms of cycles, phases, iterations, and milestones. The vertical axis represents the static aspect of the process: how it is described in terms of activities, artifacts, workers and workflows [41].

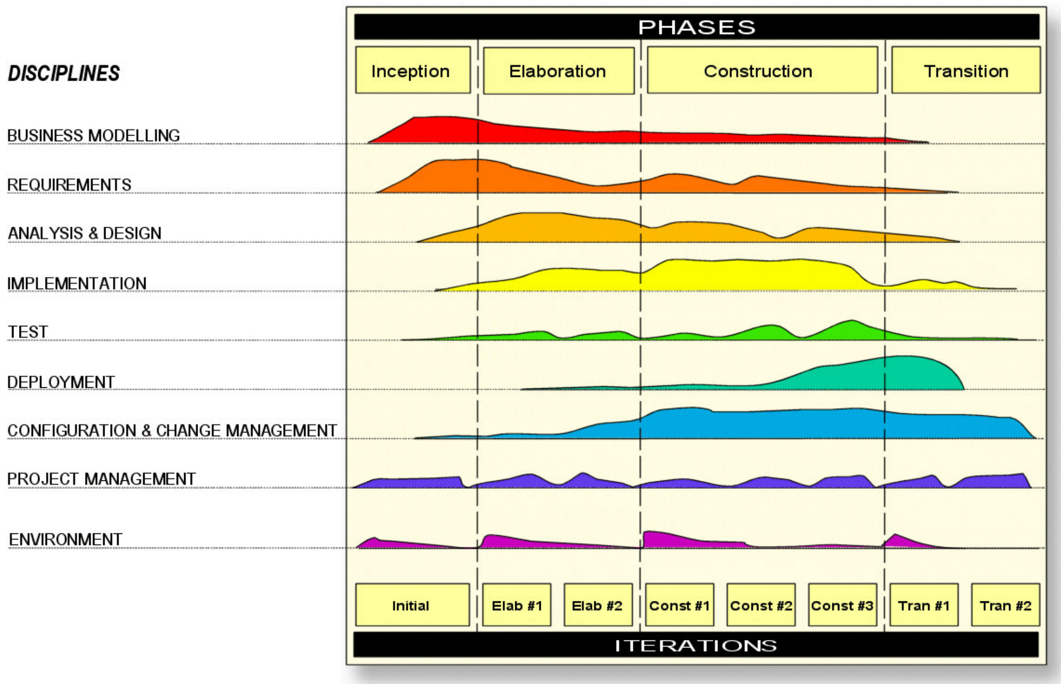


Fig. 35 – Typical RUP chart, showing how the development process is structured along two dimensions. [42]

3.1.2 Version Control

Version control systems are a category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members [43].

In the case of the development of this project, the tools used for version control were primarily Git, using Bitbucket as the web-based hosting service. Through this, Issues were employed to keep track of tasks, enhancements, and bugs for the project.

3.2 Project Planning

Given the investigative and incremental nature of this project, it suffered several changes in its requirements and planning throughout development, due to the shareholders' feedback. Nevertheless, Table 1 displays an approximation of its overall planning.

Table 1 – Project Planning

Tasks/Events	**Begin date**	**End date**
Investigation of the project's theme - IoT and Home Automation	17/02/17	22/02/17
Introduction to the FlexHousing project and Arrowhead Framework	23/02/17	03/03/17
Stakeholder Meetings and Requirements Engineering	06/03/17	17/03/17
Software Engineering - Requirements, Analysis, and Design	16/03/17	27/03/17
Development of the FlexHousing Web Platform	27/03/17	21/06/17
FlexHousing Web Platform Tests	27/03/17	21/06/17
Requirement Updates	05/04/17	24/04/17
Improvements and implementation of new features on the FlexHousing Middleware	05/04/17	25/07/17
FlexHousing Middleware Tests	05/04/17	25/07/17
Vacation Break	26/07/17	17/08/17
Remaining time to fulfill incomplete tasks	18/08/17	08/09/17
Project Report	23/02/17	08/09/17

3.3 Meetings

Table 2 – Project Meetings

Date	Participants	Location	Description
06/02/17	Rafael Rocha, Luis Lino Ferreira, José Bruno Silva	CISTER	Introduction and discussion about the project's theme.
10/02/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project's state of the art.
17/02/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano, José Bruno Silva	CISTER	Presentation by Rafael Rocha about Home Automation.
24/02/17	Rafael Rocha, Luis Lino Ferreira, Joss Santos	CISTER	Introduction to the FlexHousing project.
02/03/17	Rafael Rocha, Michele Albano	CISTER	Introduction to the Arrowhead Framework.
03/03/17	Rafael Rocha, Michele Albano	CISTER	Discussion about the project.
08/03/17	Rafael Rocha, Michele Albano	CISTER	Troubleshooting equipment (Cloogy and smart plug).
10/03/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	ISEP	Further discussion about the project.
27/03/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Project planning.
01/04/17	Rafael Rocha, José Bruno Silva	CISTER	Discussion about the project's implementation.
12/04/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano, José Bruno Silva	CISTER	Discussion about the project.

18/04/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano, José Bruno Silva	CISTER	Discussion about the project's status.
19/04/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Presentation by Rafael Rocha about the project's status.
24/04/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Discussion about the acquisition of new equipment.
05/05/17	Rafael Rocha, André Pedro, Pedro Santos	CISTER	Discussion about the implementation of the new equipment.
11/05/17	Rafael Rocha, Luis Lino Ferreira, André Pedro	CISTER	Discussion about the limitations of the new equipment.
15/05/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Discussion about the project's use cases.
17/05/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Discussion about the project report.
06/06/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano, André Pedro	CISTER	Discussion about the acquisition of new equipment.
12/06/17	Rafael Rocha, Michel Albano, José Bruno Silva	CISTER	Discussion about the implementation of the new equipment.
26/06/17	Rafael Rocha, Michel Albano, José Bruno Silva	CISTER	Discussion about the project's implementation.
28/06/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project's use cases.
29/06/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project's status.
08/07/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Discussion about the project's status.

11/07/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Presentation by Rafael Rocha about the project's status.
18/07/17	Rafael Rocha, Luis Lino Ferreira, Michele Albano	CISTER	Discussion about the project and the project report.
20/07/17	Rafael Rocha, José Bruno Silva	CISTER	Discussion about the project's implementation
21/07/17	Rafael Rocha, Michele Albano	CISTER	Discussion about algorithms to detect energy consumption patterns.
24/07/17	Rafael Rocha, Luis Lino Ferreira, Vincent Nelis	CISTER	Discussion about device type identification.
17/08/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project's status and the project report.
21/08/17	Rafael Rocha. Luis Lino Ferreira, Michele Albano, Vincent Nelis	CISTER	Discussion about device type identification and the implementation of data mining algorithms.
31/08/17	Rafael Rocha, Michele Albano	CISTER	Discussion about the project's status and the project report.
04/09/17	Rafael Rocha, Michele Albano	CISTER	Discussion about the project report.
15/09/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project report.
27/09/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project's status and the project report.
04/10/17	Rafael Rocha, Luis Lino Ferreira, Vincent Nelis	CISTER	Discussion about the device type identification section in the project report.
09/10/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project report.

11/10/17	Rafael Rocha, Luis Lino Ferreira	CISTER	Discussion about the project report.
12/10/17	Rafael Rocha, Michele Albano	CISTER	Discussion about the project report.

3.4 Used Technologies

This section will briefly mention every technology used along the project.

As mentioned before, the FlexHousing project is composed of two different applications: the FlexHousing Middleware, which communicates with a devices' or a service provider's API, manages the database, and provides its data as a RESTful service to web applications, and the FlexHousing web platform, which serves as a gateway to the Middleware's data and services.

Table 3 – Used Technologies

Technology	***What it is and where it was used***
Java	Java is an object-oriented programming language. The Middleware, and the local Aggregator and VME implementations developed in the Arrowhead project [44] are built in Java.
Jersey and Grizzly	These two Java libraries are used in the FlexHousing Middleware to consume resources from RESTful APIs, and to host the Middleware's own resources.
Apache Derby	An open source relational database implemented entirely in Java and available under the Apache License, Version 2.0. The Middleware's data persistence is guaranteed by this database.

ejabberd (XMPP Server)	XMPP is a communications protocol for message-oriented middleware based on XML, allowing for a secure message exchange between the FlexHousing Middleware and the other Arrowhead modules. *ejabberd* is a free XMPP application server, and was selected as the project's local XMPP server.
Laravel 5.4 (PHP)	Laravel is a free, open-source PHP web framework, intended for the development of web applications following the MVC architectural pattern. The web app's backend is built in PHP, using Laravel 5.4 as its framework.
HTML5, CSS3, JavaScript	The web app's frontend is built in HTML5, CSS3, and JavaScript, using the Materialize Framework.
Arduino (C/C++)	The Arduino language consists of a set of C/C++ functions. Arduino was used to develop a custom firmware for the Sonoff sensors.
Arrowhead Framework	The Middleware (which is a Flex-Offer Agent), uses the Service Registry and Orchestration modules, provided by the Arrowhead framework, to find the remote Aggregator's address.
Python	A multi-paradigm programming language. Python was used to develop scripts for identifying device types.

4 Technical Description

This chapter describes all the work done throughout the project's development, from requirements engineering, analysis and design, to implementation and tests.

4.1 Requirements Engineering

Software requirements are descriptions of features and functionalities of the requested system; therefore, these requirements convey the users' expectations of the final product. Some requirements can be obvious, known, or expected, while others can be hidden, unknown, or unexpected from the client's point of view [45].

The process of gathering these software requirements from the client, analyzing and documenting them, is known as requirements engineering.

4.1.1 User Roles

A user role is a collection of defining attributes that characterize a population of users and their intended interactions with the system [47]. These are roles that are significant to the success of a system and will occur over and over in the user stories that make up a system's product backlog [48].

Thus, the defined user roles for this project, and their respective interactions, are as follows:

User Role: End User

- Standard users that can only access their information and data.
- Can access the data of their houses, rooms, and devices through the FlexHousing web platform.
- Can check their devices' energy consumption through the FlexHousing web platform.
- Can turn their devices on and off through the FlexHousing web platform.
- Can create Flex-offers for their devices, manually or automatically, through the FlexHousing web platform.

User Role: Company Executive

- Users that can access an information summary of their clients' data.
- Can check the total number of registered users, devices, and houses.
- Can check the number of times a device brand was used and its average time of use.

4.1.2 User Stories

User stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system [49]. At times, there might exist a disconnection between what the user needs and what the system actually offers. This is the reason why requirements specification is so important to software development, and why user stories help align the system's features with the user's vision.

It should be noted that this project's user stories were made in conjunction with a mockup of the final application, hence their respective acceptance tests are so specific. Thus, this project allowed to define the following user stories and acceptance tests:

Table 4 - User Stories: End User

End User	***Acceptance Test***
I want to access the FlexHousing platform.	The user must access the platform’s web address and login (or register first, and then login).
I want to check my devices in their respective house and room.	The user must fill out the required forms for the device registry: first the house form, then the room form, and finally the device form, on the FlexHousing platform. The user must select the “Devices” option on the FlexHousing platform’s navigator panel.
I want to check if my devices are turned on or off.	The user must fill out the required forms for the device registry: first the house form, then the room form, and finally the device form, on the FlexHousing platform. The user must check the color on the device’s “On/Off” button in the “Devices” page, on the FlexHousing platform.
I want to control my devices remotely.	The user must fill out the required forms for the device registry: first the house form, then the room form, and finally the device form, on the FlexHousing platform. The user must click the device’s “On/Off” button in the “Devices” page, on the FlexHousing platform.
I want to check my devices’ energy consumption.	The user must fill out the required forms for the device registry: first the house form, then the room form, and finally the device form, on the FlexHousing platform.

	The user must click on the device’s “Consumption” button in the “Devices” page, on the FlexHousing platform.
I want to apply a flex-offer to one of my devices.	The user must fill out the required forms for the device registry: first the house form, then the room form, and finally the device form, on the FlexHousing platform. The user must click on the device’s “Create Flex-offer” button in the “Devices” page, on the FlexHousing platform. The user must fill out the form for the Flex-offer creation, on the FlexHousing platform.
I want to check my device’s active flexoffer and see if it was effective on the device’s consumption pattern.	The user must fill out the required forms for the device registry: first the house form, then the room form, and finally the device form, on the FlexHousing platform. The user must fill out the form for the Flex-offer creation, on the FlexHousing platform. The user must click on the device’s “Check Flex-offer” button in the “Devices” page, on the FlexHousing platform.

Table 5 - User Stories: Company Executive

Company Executive	***Acceptance Tests***
I want to check basic information about the system and a customer device usage analysis (e.g., number of times used, average hours of use, etc.).	The user must access the platform’s web address and login (or register first, and then login). The information will be displayed in the main page.

4.1.3 Functional Requirements

Functional requirements specify functionalities (use cases) that a system or system component must be able to perform [50].

In this case, the functional requirements of the project are as follows:

4.1.3.1 FlexHousing platform

1. The system should be able to handle multiple users and houses;
2. The system should be able to register users, and grant access to a user after they provide the correct username and password;
3. The system should be able to browse, read, edit, add, and delete houses (in other words, perform a CRUD operation);
    a. When filling out the form to register a house, the user should specify:
        i. House name;
        ii. House address.
4. The system should be able to browse, read, edit, add, and delete rooms (in other words, perform a CRUD operation);
    a. When filling out the form to register a room, the user should specify:
        i. Room name;
        ii. House where the room belongs to.
5. The system should be able to browse, read, edit, add, and delete devices (in other words, perform a CRUD operation);
    a. When filling out the form to register a device, the user should, at least, specify:
        i. Device name;
        ii. House where the device is located;
        iii. Room where the device is located;
        iv. Device brand;
        v. Device model;
        vi. Sensor brand;

vii. Sensor’s REST API address.

6. The system should be prepared for the addition of devices from different brands, using different communication protocols;
7. The system should be able to turn a device on and off;
8. The system should be able to register or access the device’s energy consumption data;
9. The system should be able to show the current energy consumption of a device;
10. The system should be able to connect to local or external Aggregator modules;
11. The system should be able to allow the user to create Flex-offers manually;
12. The system should be able to create Flex-offers automatically;
13. The system should be able to show a device’s active flexoffer and demonstrate if it was effective on the device’s consumption pattern;
14. The system should be able to identify the types of devices registered (wet-devices, refrigerators, and so on).

4.1.3.2 Executives’ platform

1. The system should be able to display the total number of registered users, devices and houses;
2. The system should be able to register and display the end users’ devices’ frequency of use and average time of use;

4.1.3.3 Custom switch (*Sonoff*)

1. The system should be able to turn the energy on and off;
2. The system should be able to acquire energy consumption data (Active Power, Voltage, Current);
3. The system should be able to provide energy consumption data via REST or other protocols.

4.1.4 Non-Functional Requirements

Non-functional requirements are not related to functional aspects of the software. They are implicit or expected characteristics of the system. Thus, the main non-functional requirements of the project are as follows:

4.1.4.1 Usability

Usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction [51].

This project's platforms must have an intuitive and appealing interface, using responsive layouts, to guarantee the adequate and useful presentation of information. More specifically, the FlexHousing platform must be able to adequately display information for both expert and inexperienced users, by being simple enough to use and understand while offering additional features for more in-depth information.

4.1.4.2 Performance

Performance pertains to the amount of work accomplished by a computer system [52].

Since this project's platforms aim to be interactive and could be used on mobile devices, which mostly use low-speed download-upload connections, certain precautions should be taken. For instance, images should be reduced in size and, if possible, converted to .jpg format, CSS should be minified, and requests to the server should mostly use AJAX technology, thus avoiding the entire page refresh.

Moreover, in the case of the FlexHousing platform, interaction with other devices should be seamless and as quick as possible, for instance, the response time when turning on/off a device should be less than one second.

4.1.4.3 Portability

Portability consists in the usability of the same software in different environments [53].

The portability of these platforms must be ensured, since they will be accessed through any device with an internet connection. For example, the project's platforms should use responsive layouts, which will in turn dynamically adapt their content to any device's display without the need to zoom.

4.1.4.4 Interoperability

Interoperability is a characteristic of a system, whose interfaces are completely understood, to work with other products or systems, at present or future, in either implementation or access, without any restrictions [54].

As mentioned before, interoperability is a big focus in the FlexHousing platform, since one of its goals is to develop a generic interface, able to work with different implementation modules for different devices.

4.2 Analysis

Domain Analysis is a process by which information used in developing software systems is identified, captured, and organized, producing a domain model with the purpose of making the information reusable when creating new systems [56]. A domain model is a conceptual model of the domain that incorporates both behavior and data [57]. It represents real-word concepts, and not software components [58].

Through the gathered functional requirements and use cases, and based on the previous project's domain model, it was possible to develop a domain model that captured the envisioned system.

Since these improvements are relevant to understanding the proposed solution, a domain analysis was made, and the changes resulting from it were documented.

4.2.1 Domain Model

As a starting point for the analysis, the previous project's domain model will be presented (Fig. 36). This will allow us to juxtapose it with the new requirements, and determine what can be kept, what must change, and what needs to be added.

Next, a domain model considering the new requirements will be displayed (with the new elements highlighted in red) in Fig. 37, as well as a detailed description of every domain object present in the new model.

4.2.1.1 Previous Domain Model

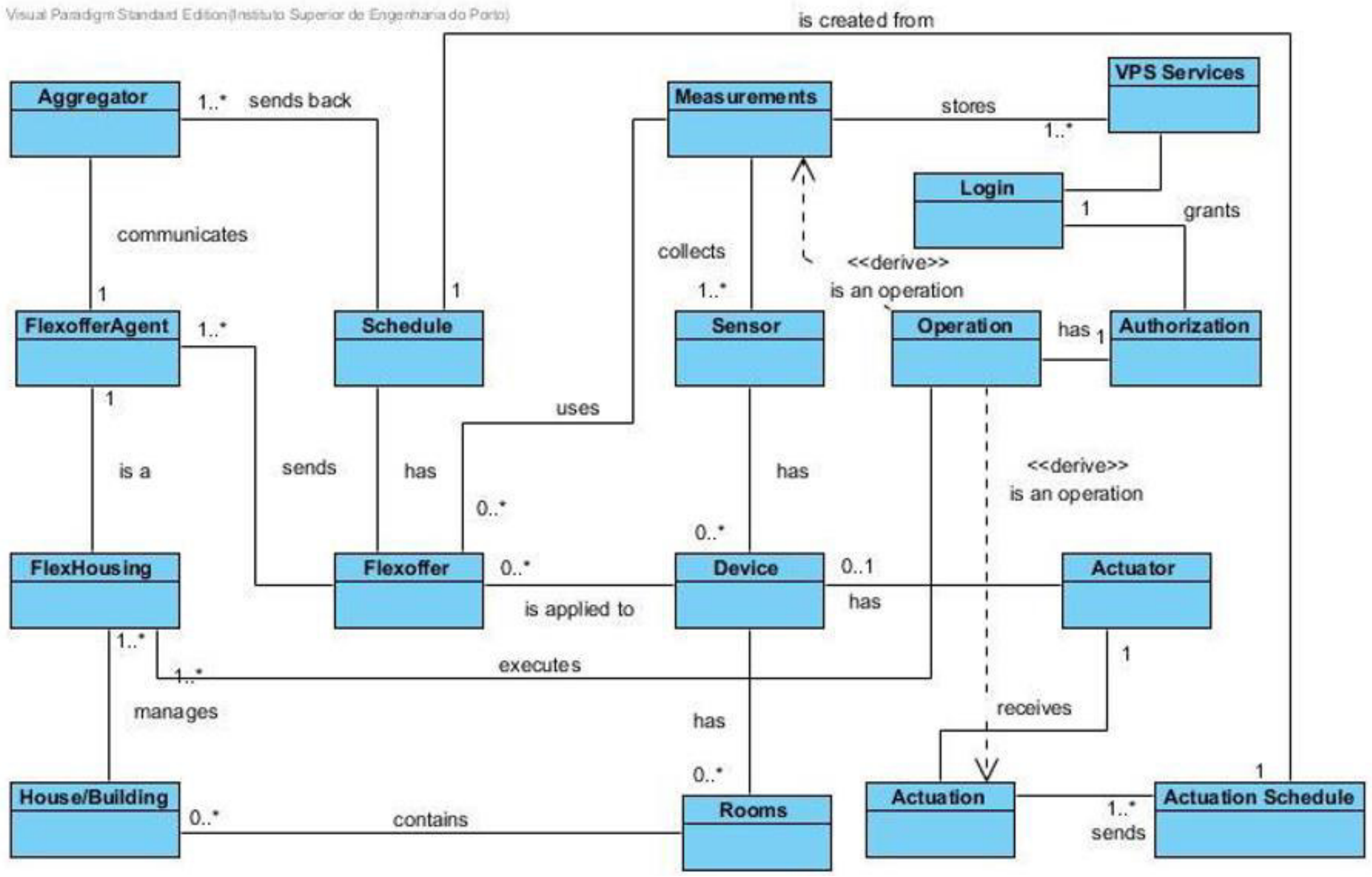


Fig. 36 – Previous project's domain model [59]

4.2.1.1.1 What can be kept

The flex-offer process (particularly the behavior between Flex-Offer Agent, Flex-offer, Measurements, Schedule, Actuation Schedule, and Actuation) can be reused, since it follows the established requirements. While the implementation of this feature did suffer some changes, the domain model itself does not merit any major tinkering on this front.

4.2.1.1.2 What must change

First, the previous project was focused on functioning only with VPS devices, which meant FlexHousing's Operations were tied to the VPS services' framework. Given that one of the project requirements specifies that the system should be prepared for the addition of other kind of devices from different brands, the process of requesting "Measurements" and sending "Actuations" to a "Device" must be abstracted.

In other words, the object "VPS Services" should implement the behavior specified by an interface named "Third-Party Service". This way, device operations (Measurements and Actuations) will be subject to different implementations of the "Third-Party Service" interface, depending on a given device.

Second, in the previous project, the concept of multiple "Users" or multiple "Houses" was not taken into account. Since one of the project requirements requests that the system can have multiple users, and these users can have multiple houses, the domain model will have to be changed to accommodate these features.

4.2.1.1.3 What needs to be added

Given that the system has to connect to local or external Aggregator and VME modules, the Virtual Market should not be omitted, and must be added as a domain object in the model.

Furthermore, since one of the project requirements specifies the existence of the "Company Executive" role, this role should also be added as a domain class. Moreover, because a project requirement stipulates that the system has to identify the type of a device, the Device class will also have a Device Type. Also, because of the added support for different kinds of device brands, it should be acknowledged that some device services do not store measurements. Thus, the FlexHousing system itself should be able to register measurements from devices that do not offer this feature. Lastly, since the FlexHousing system must verify the effectiveness of a flexoffer, this operation should also be displayed in the domain model.

4.2.1.2 New Domain Model

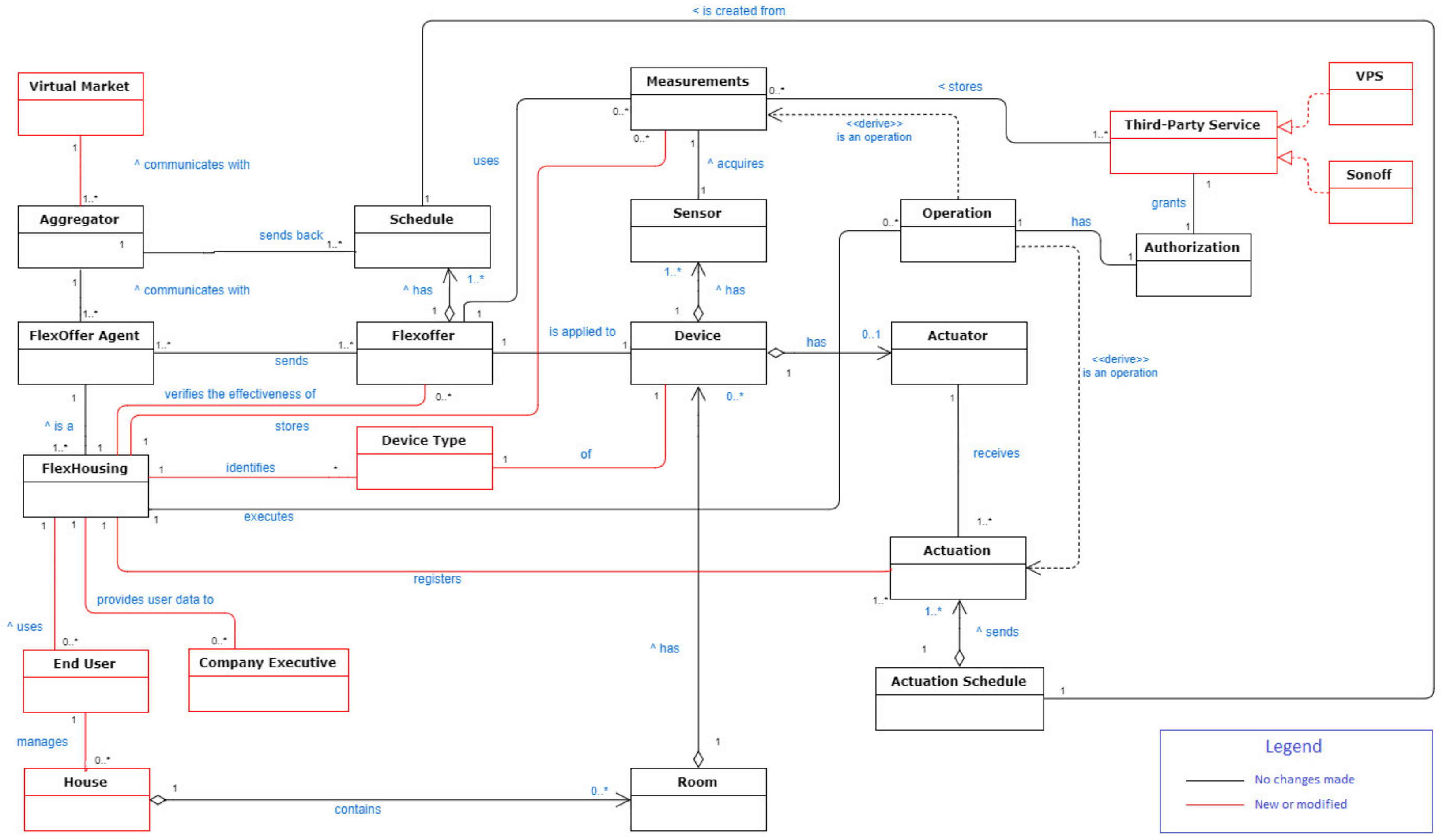


Fig. 37 – Project's new domain model

FlexHousing

This conceptual class represents the FlexHousing middleware. The FlexHousing middleware is responsible for bridging the users' houses, rooms and devices, with the Flex-Offer Agent, while also being able to perform operations like actuations and acquiring measurements.

Flex-Offer Agent

A Flex-Offer Agent is responsible for generating and sending the Flex-offers to the Aggregator. As explained before, Flex-Offer Agents are basically software modules that offer the main functionalities to support the flex-offer concept. In this regard, the FlexHousing middleware is effectively a Flex-Offer Agent.

Aggregator

The Aggregator, once having aggregated enough flex-offers from the Flex-offer Agents, will send these to the Virtual Market (of Energy). Afterwards, the Aggregator receives a response from the Virtual Market, disaggregates the response and sends a consumption schedule to the Flex-Offer Agent.

Virtual Market

The Virtual Market, after receiving a flex-offer larger than a certain amount, will then negotiate with existing electricity markets, and acquire the cheapest consumption schedule for these to be deployed. Finally, it then sends the schedule to the Aggregator.

End User and Company Executive

These conceptual classes represent the two types of user roles that will make use of the FlexHousing system. While the End User uses FlexHousing to manage his houses' appliances, the Company Executive uses it to analyze user data.

House

A House belongs an End User and contains Rooms. The House class will serve as a software class to store the Rooms belonging to the End User's house.

Room

A Room belongs a House and contains Devices. The Room class will serve as a software class to store the Devices belonging to the End User's room.

Device

A Device has one or more Sensors connected to it, so that the FlexHousing middleware can store the Device's energy consumption data (or other types of data in the future), and an Actuator, so that the FlexHousing middleware can turn the Device on and off.

The Device class will serve as a structure to store the device's name, brand and model, and its sensors.

Device Type

The Device Type class identifies what kind of device a Device is: either a wet-device, or a refrigerator, or an electric vehicle, and so forth.

Sensor

A Sensor has the purpose of acquiring a Device's Measurements, which, in this case, are energy consumption values. However, in the future, additional Sensors could be installed to obtain other types of Measurements.

Measurement

The Measurement class serves as a structure to store any kind of Measurement obtained by a Sensor. A Measurement is composed of three attributes: the type of measurement, the measurement value, and the date the measurement was taken. Currently, however, Measurements are only used to represent energy consumption values.

Actuator

The Actuator is a conceptual class that represents the Device's actuator, which has the responsibility of turning the Device on and off.

Actuation

The Actuation class represents the action of turning on and off a Device. This class serves as a structure to store the ID of the device the actuation is for, and the start time and end time of an actuation.

Operation

An Operation is only a conceptual class that represents the action of requesting Measurements from a Device or performing an Actuation on a Device. These types of operations may have to undergo an Authorization process, depending on the Third-Party Service that regulates the Device.

Third-Party Service

The Third-Party Service class depicts the service that controls the Device's smart plug/switch. This could be a service provider's API arranged by their servers, or it could just be an API available in the local network, provided by the smart plug/switch. Either way, these services may have different processes of executing the same operation.

Authorization

The Authorization class is a conceptual class that represents the authorization process granted by the Third-Party Service to perform operations. However, an Authorization process might not be required by the Third-Party Service.

Flex-offer

The Flex-offer class represents the flex-offers sent by the FlexHousing Middleware. This class can be used as a structure to store the start time and end time of the flex-offer, the minimum and maximum energy consumption for each time section, and its actuation schedule. At first, the flex-offer doesn't have a Schedule for the device actuations, until the Aggregator responds with a Schedule. Only then can it be applied to a device.

Schedule

The Schedule class represents the Flex-offer's schedule. At first, the flex-offer doesn't have a Schedule for the device actuations, until the Aggregator replies with one.

Actuation Schedule

The Actuation Schedule is a conceptual class created from the Flex-offer's Schedule, where it executes the scheduled actuations based on the time determined by the Schedule.

4.3 Design

After a careful analysis of the information gathered from the project requirements and domain model, comes the development process of designing a model of the software to be implemented.

In other words, the conceptual model is developed further into an object-oriented model using Object-Oriented Design (OOD) [60]. In OOD, the domain concepts in the analysis model are translated into software classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain [61].

In this case, the models designed in this section were ultimately implemented, with the project implementation being available on the Bitbucket repository.

4.3.1 Data structure

The database used in the previous project was an Apache Derby relational database. For the development of the current project, the same database was used, but its structure suffered some alterations because of the new requirements. This section will describe the schema for the new database.

First, the full database schema will be displayed in Fig. 38, with the new elements highlighted in red. Then, a brief explanation of these added elements will be given.

4.3.1.1 Database schema

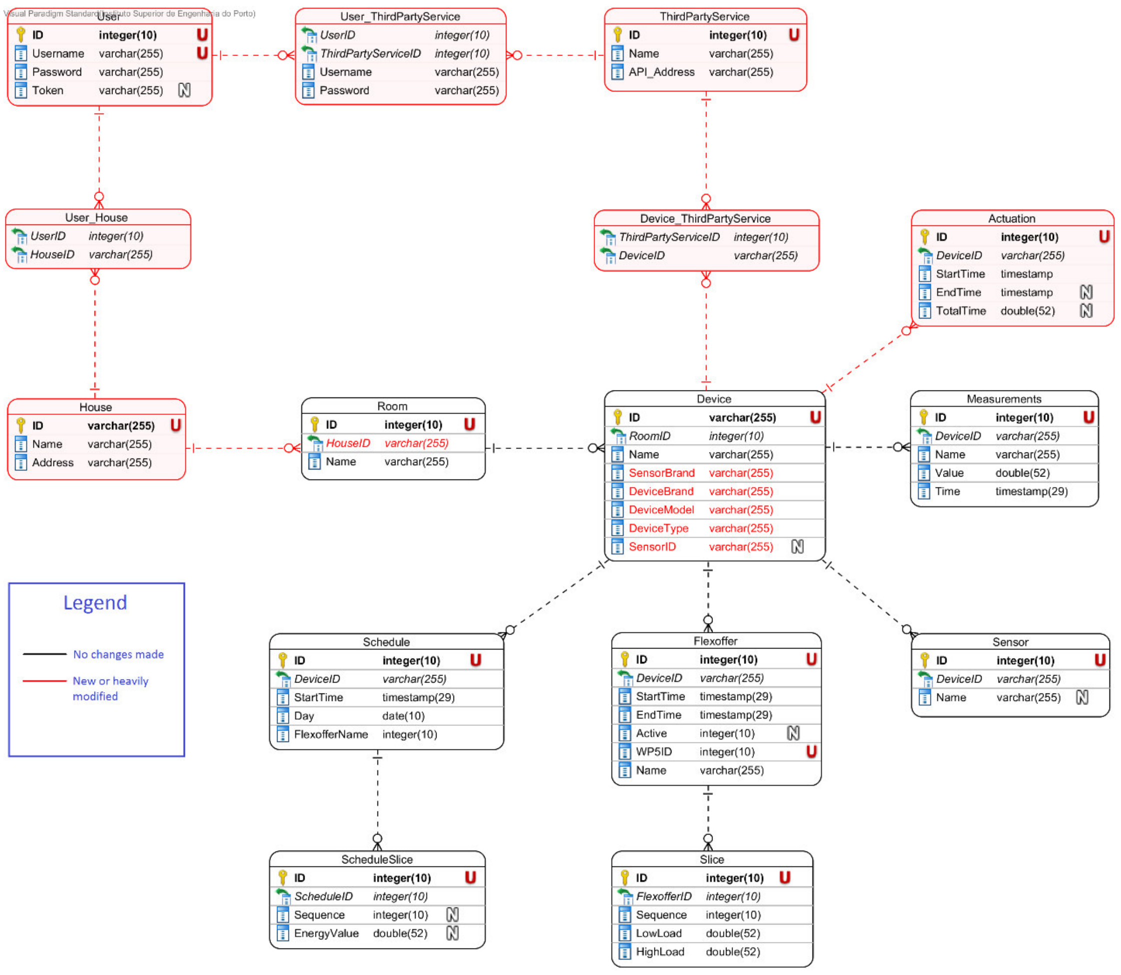


Fig. 38 – Database schema for the FlexHousing Middleware

4.3.1.2 Database Explanation

To address the requirements for multiple users and houses, the tables `User`, `House`, and `User_House` were created. Most of their attributes are self-explanatory, with the exception of the attribute `Token` in `User`, which serves as a randomly generated access token given to users when they log in, to control user access on the platform.

Furthermore, regarding the third-party service solution, the tables `ThirdPartyService`, `User_ThirdPartyService`, and `Device_ThirdPartyService` were conceived. The attribute `API_Address` stores the web address of the service's API, while `Username` and `Password` in `User_ThirdPartyService` pertain to the user's credentials of a specific service.

Lastly, to satisfy the requirements related to the Company Executive role, where the platform would present basic data about the End Users' device usage, some new attributes were added to the `Device` table, and the table `Actuation` was created, so that the system would be able to register every time a device is used (a device is considered "used" once when it is turned on, and then turned off, in this sequence).

4.3.2 Use Cases

Use cases are a list of actions or event steps, typically defining the interactions between a role (known in the UML as an actor) and a system, to achieve a goal [55].

Given the specified functional requirements, the following use cases were determined:

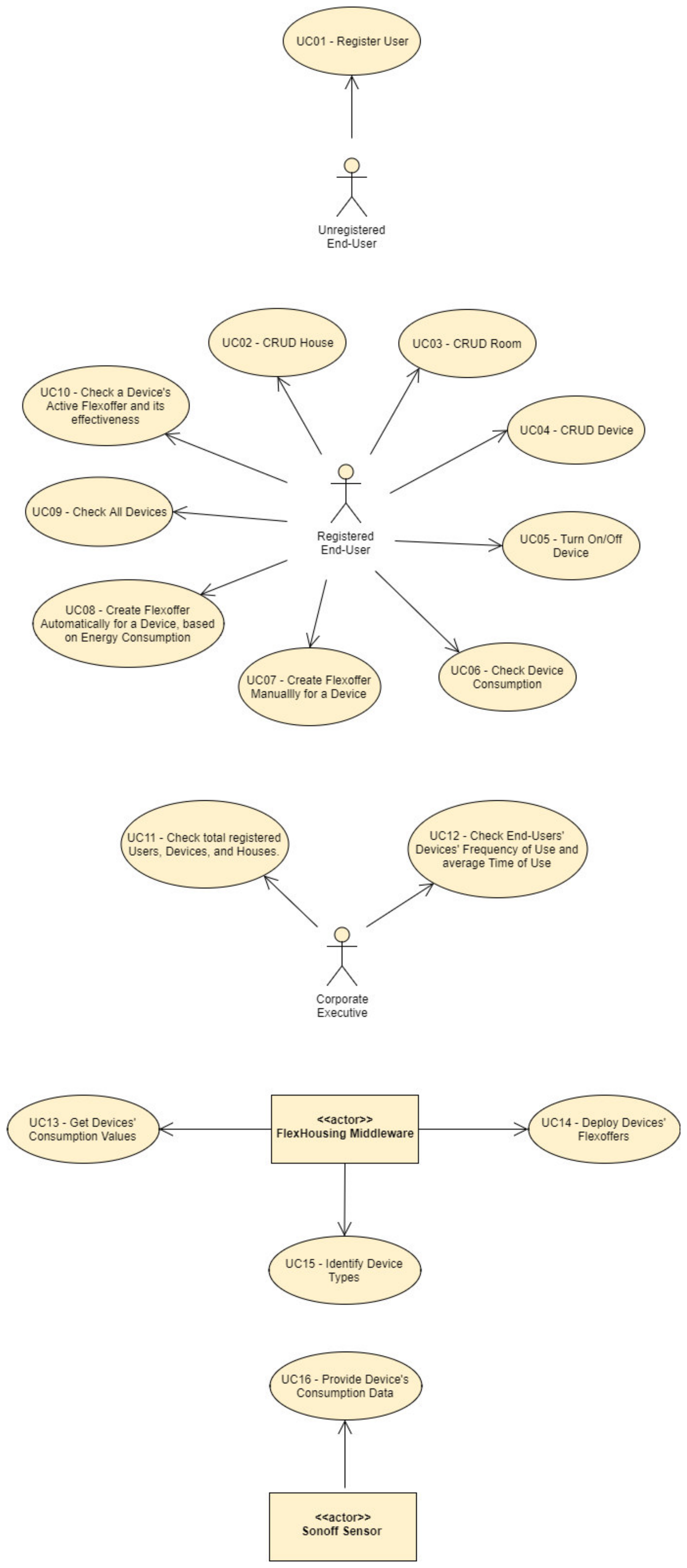

UC01 - Register User
Unregistered End-User
UC02 - CRUD House
UC03 - CRUD Room
UC10 - Check a Device's Active Flexoffer and its effectiveness
UC04 - CRUD Device
UC09 - Check All Devices
Registered End-User
UC05 - Turn On/Off Device
UC08 - Create Flexoffer Automatically for a Device, based on Energy Consumption
UC07 - Create Flexoffer Manuallly for a Device
UC06 - Check Device Consumption
UC11 - Check total registered Users, Devices, and Houses.
UC12 - Check End-Users' Devices' Frequency of Use and average Time of Use
Corporate Executive
UC13 - Get Devices' Consumption Values
<<actor>> FlexHousing Middleware
UC14 - Deploy Devices' Flexoffers
UC15 - Identify Device Types
UC16 - Provide Device's Consumption Data
<<actor>> Sonoff Sensor

Fig. 39 - Use Cases for the FlexHousing project

Next, all the use cases present in Fig. 34 will be specified. The specification will be carried out in detail, mentioning the pre- and postconditions, as well as the basic flow and sequence diagram of each operation. A sequence diagram is used primarily to show the interactions between objects in the sequential order that those interactions occur [64].

4.3.2.1 Use Case 01 – Register User

Table 6 - Use Case 01: Register User

Use Case 01 – Register User

Description	The user intends to register him/herself in the system, so that he/she can access the platform.
Actor(s)	Unregistered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*).
Postconditions	1. The user is registered in the system and allowed to access the platform, given he/she inputs the right credentials.
Basic Flow of Events	
Actions of the actor	1. Clicks the option "Register". 2. Inserts required information. 3. Clicks on the button "Register".
Actions of the system	1. Redirects to the user register form. 3. Validates if form was completed and registers user with the given information.

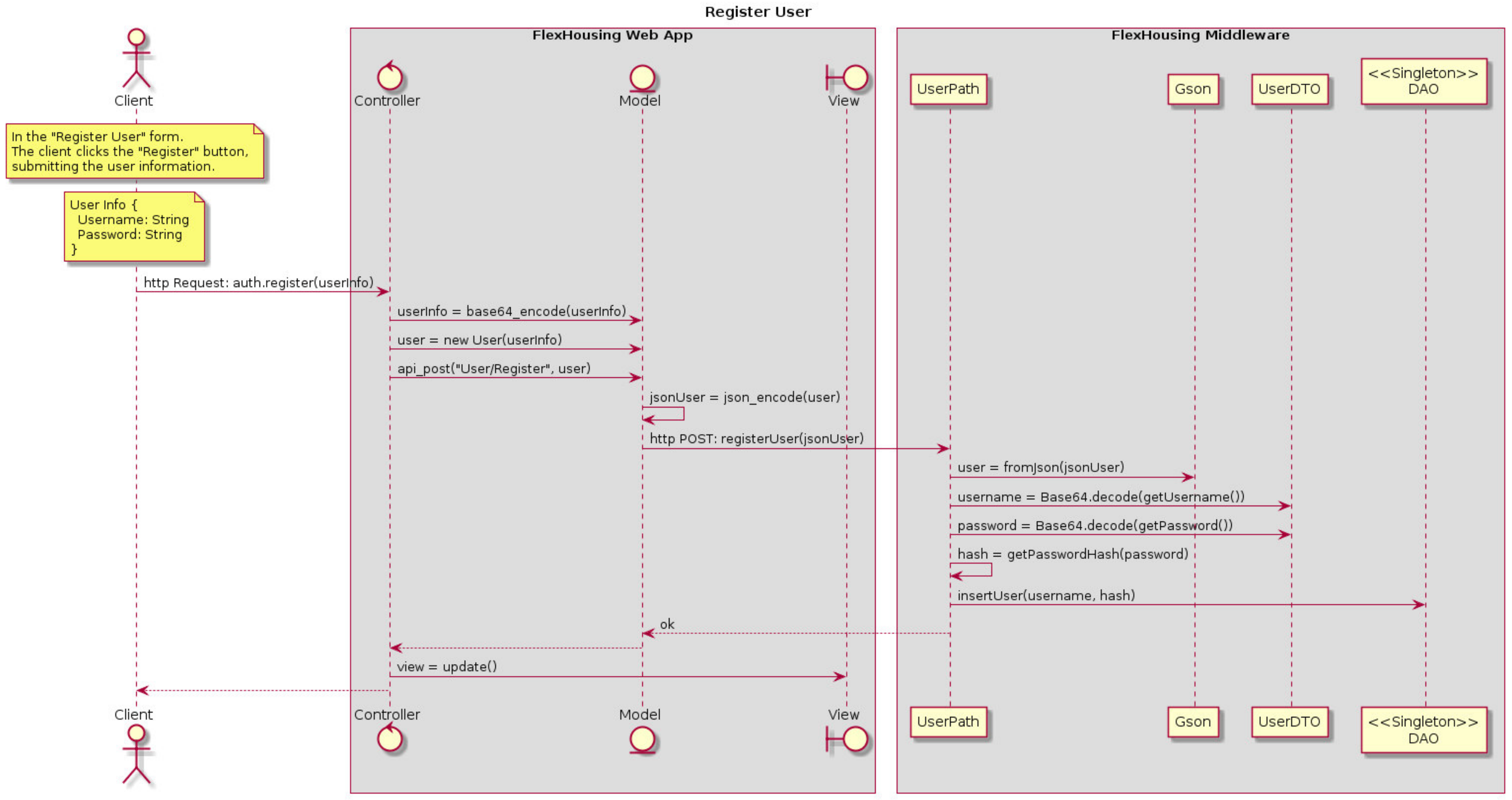

Register User
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
UserPath
Gson
UserDTO
<<Singleton>> DAO
In the "Register User" form.
The client clicks the "Register" button,
submitting the user information.
User Info {
Username: String
Password: String
}
http Request: auth.register(userInfo)
userInfo = base64_encode(userInfo)
user = new User(userInfo)
api_post("User/Register", user)
jsonUser = json_encode(user)
http POST: registerUser(jsonUser)
user = fromJson(jsonUser)
username = Base64.decode(getUsername())
password = Base64.decode(getPassword())
hash = getPasswordHash(password)
insertUser(username, hash)
ok
view = update()

Fig. 40 – Sequence Diagram: UC01 Register User

4.3.2.2 Use Case 02 – CRUD House

Table 7 - Use Case 02: CRUD House

Use Case 02 – CRUD House	
Description	The user intends to create/read/update/delete a house.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. For the Update process, a house must be registered in the system.
Postconditions	1. The house is created/read/updated/deleted successfully. If the house is deleted, all its registered rooms and devices are deleted as well.
Basic Flow of Events	
Actions of the actor	1. Clicks the option "Add/Check/Edit/Delete House". 2. If the option was a. Add/Edit, then the user inserts the required information and clicks on the button "Register/Edit House". b. Delete, then the user confirms the deletion process.
Actions of the system	1. If the selected option was a. Add/Edit, then the system redirects to the "Create/Edit House" form. b. Delete, then the system requests confirmation for the deletion process. c. Check, then the system redirects to the Houses index webpage.

	2. If the selected option was a. Add/Edit, then the system validates the information and registers/updates the house. b. Delete, then the system deletes the house, its rooms and devices.

4.3.2.2.1 UC02 – Create House

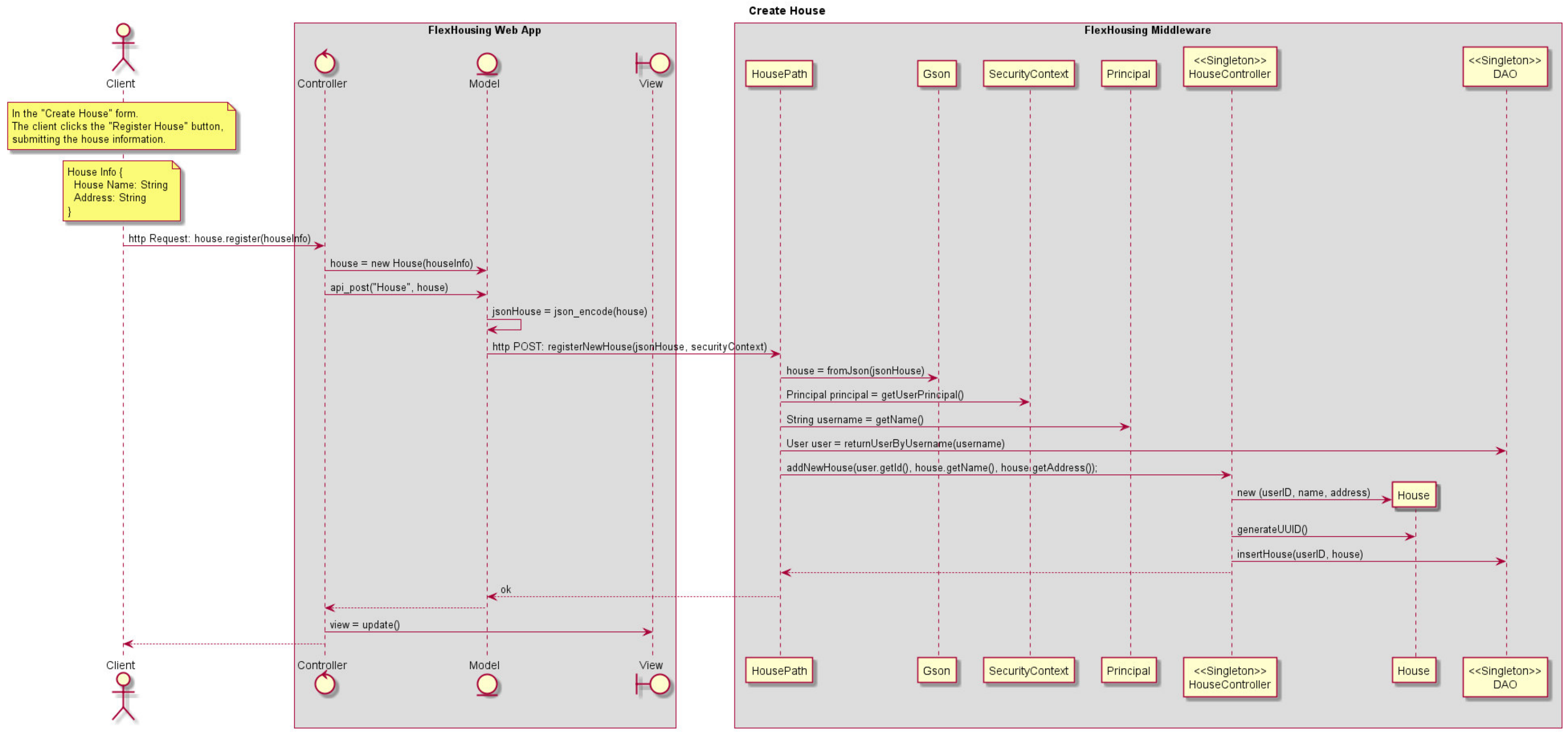

Create House
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
Gson
SecurityContext
Principal
<<Singleton>> HouseController
<<Singleton>> DAO
In the "Create House" form. The client clicks the "Register House" button, submitting the house information.
House Info { House Name: String Address: String }
http Request: house.register(houseInfo)
house = new House(houseInfo)
api_post("House", house)
jsonHouse = json_encode(house)
http POST: registerNewHouse(jsonHouse, securityContext)
house = fromJson(jsonHouse)
Principal principal = getUserPrincipal()
String username = getName()
User user = returnUserByUsername(username)
addNewHouse(user.getId(), house.getName(), house.getAddress());
new (userID, name, address)
House
generateUUID()
insertHouse(userID, house)
ok
view = update()

Fig. 41 – Sequence Diagram: UC02 Create House

4.3.2.2.2 UC02 – Read House

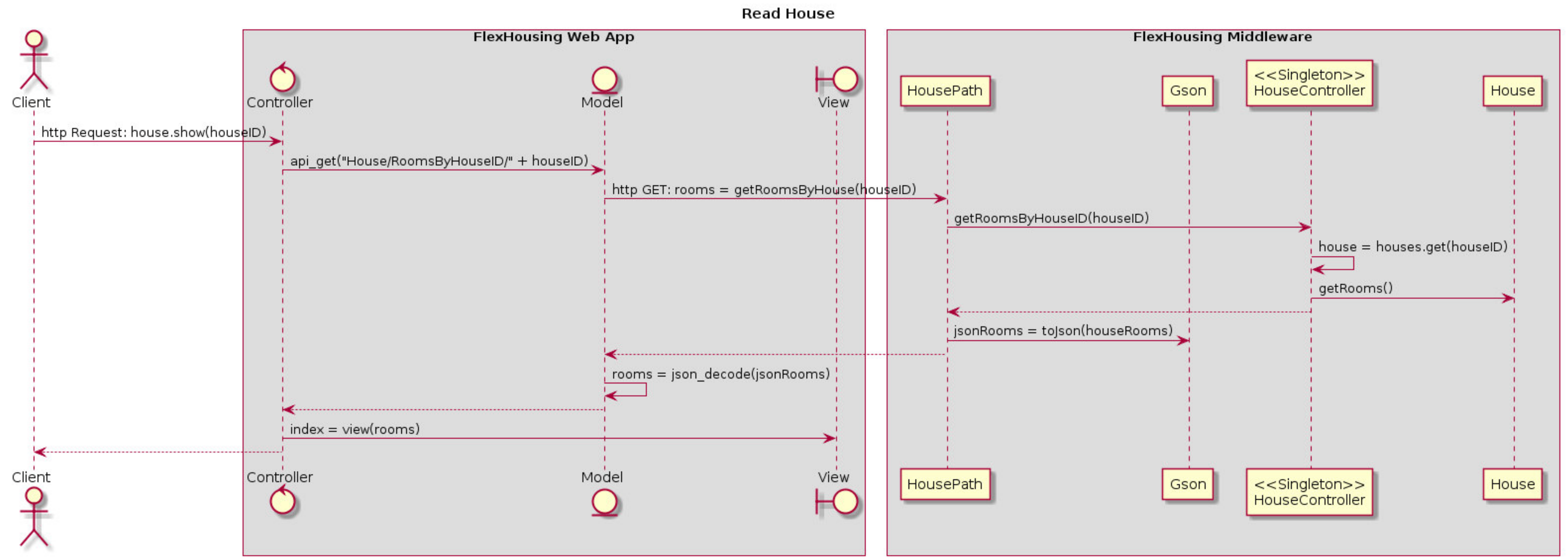

Read House
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
Gson
<<Singleton>> HouseController
House
http Request: house.show(houseID)
api_get("House/RoomsByHouseID/" + houseID)
http GET: rooms = getRoomsByHouse(houseID)
getRoomsByHouseID(houseID)
house = houses.get(houseID)
getRooms()
jsonRooms = toJson(houseRooms)
rooms = json_decode(jsonRooms)
index = view(rooms)

Fig. 42 – Sequence Diagram: UC02 Read House

4.3.2.2.3 UC02 – Update House

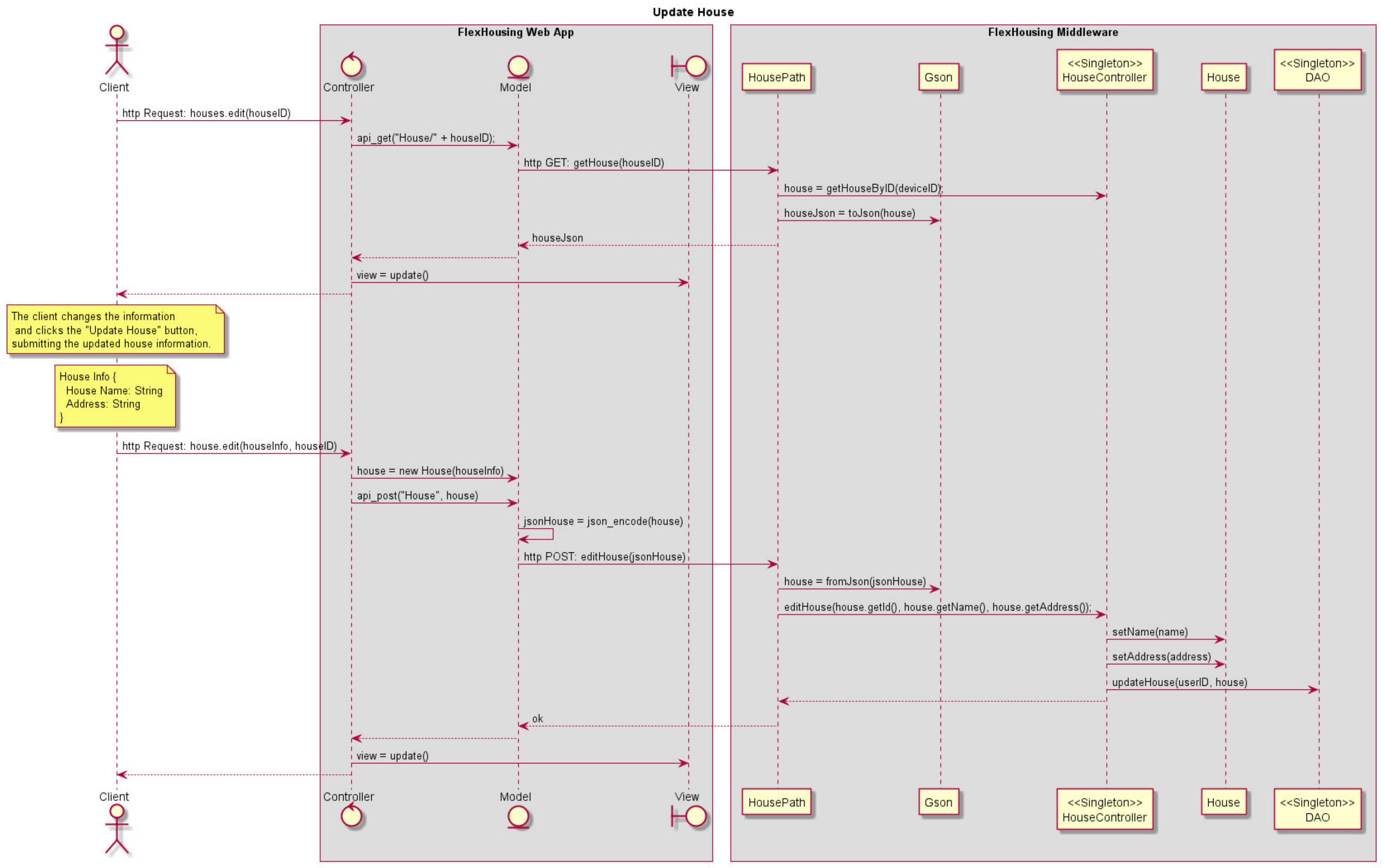

Update House
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
Gson
<<Singleton>> HouseController
House
<<Singleton>> DAO
http Request: houses.edit(houseID)
api_get("House/" + houseID);
http GET: getHouse(houseID)
house = getHouseByID(deviceID);
houseJson = toJson(house)
houseJson
view = update()
The client changes the information and clicks the "Update House" button, submitting the updated house information.
House Info { House Name: String Address: String }
http Request: house.edit(houseInfo, houseID)
house = new House(houseInfo)
api_post("House", house)
jsonHouse = json_encode(house)
http POST: editHouse(jsonHouse)
house = fromJson(jsonHouse)
editHouse(house.getId(), house.getName(), house.getAddress());
setName(name)
setAddress(address)
updateHouse(userID, house)
ok
view = update()

Fig. 43 – Sequence Diagram: UC02 Update House

4.3.2.2.4 UC02 – Delete House

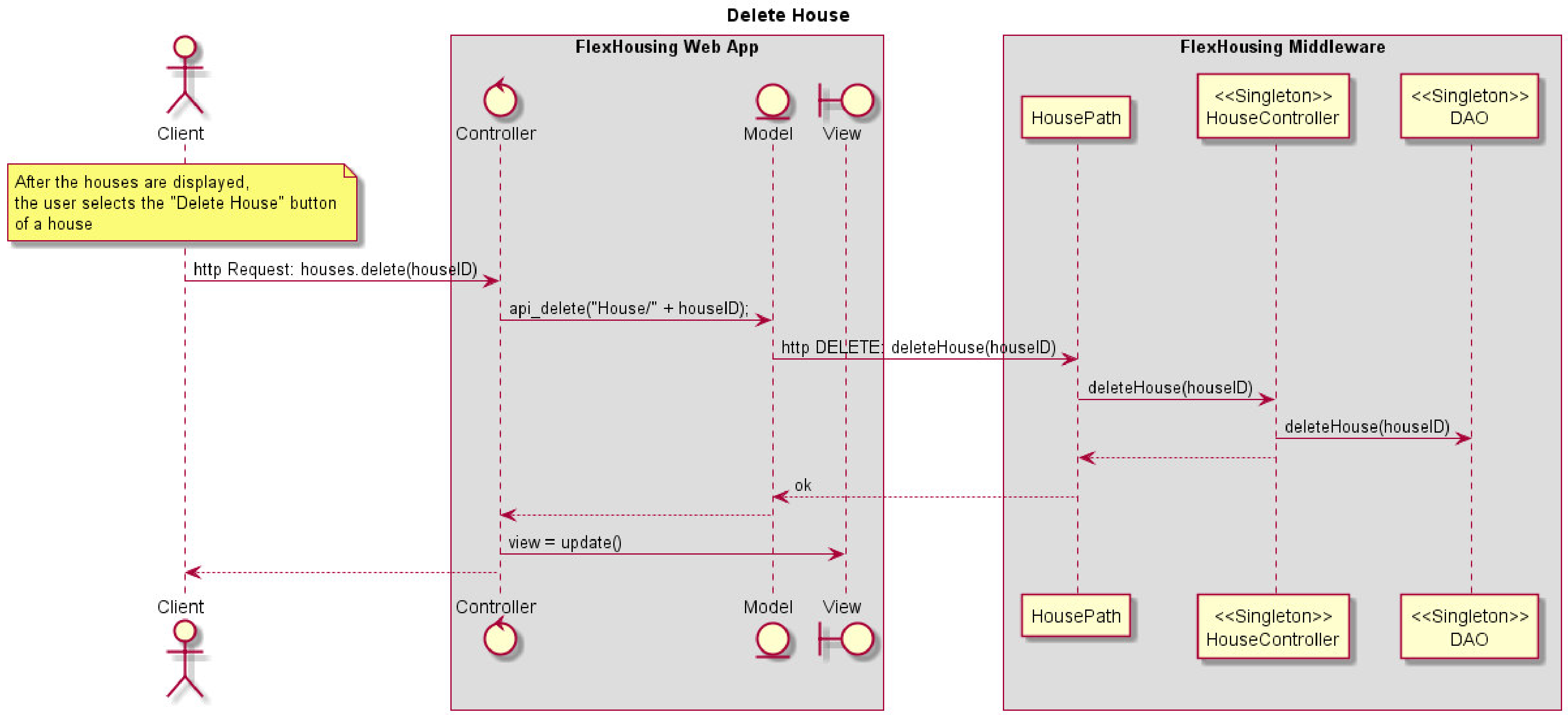

Delete House
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
<<Singleton>> HouseController
<<Singleton>> DAO
After the houses are displayed, the user selects the "Delete House" button of a house
http Request: houses.delete(houseID)
api_delete("House/" + houseID);
http DELETE: deleteHouse(houseID)
deleteHouse(houseID)
deleteHouse(houseID)
ok
view = update()

Fig. 44 – Sequence Diagram: UC02 Delete House

4.3.2.3 Use Case 03 – CRUD Room

Table 8 - Use Case 03: CRUD Room

Use Case 03 – CRUD Room	
Description	The user intends to create/read/update/delete a room.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house. 4. For the Update process, a room must be registered in the system.
Postconditions	1. The room is created/read/updated/deleted successfully. If the room is deleted, all its registered devices are deleted as well.
Basic Flow of Events	
Actions of the actor	1. Clicks the option "Add/Check/Edit/Delete Room". 2. If the option was a. Add/Edit, then the user inserts the required information and clicks on the button "Register/Edit Room". b. Delete, then the user confirms the deletion process.
Actions of the system	1. If the selected option was a. Add/Edit, then the system redirects to the "Create/Edit Room" form. b. Delete, then the system requests confirmation for the deletion process. c. Check, then the system redirects to the Rooms index webpage. 2. If the selected option was

	a. Add/Edit, then the system validates the information and registers/updates the room. b. Delete, then the system deletes the room and its devices.

4.3.2.3.1 UC03 – Create Room

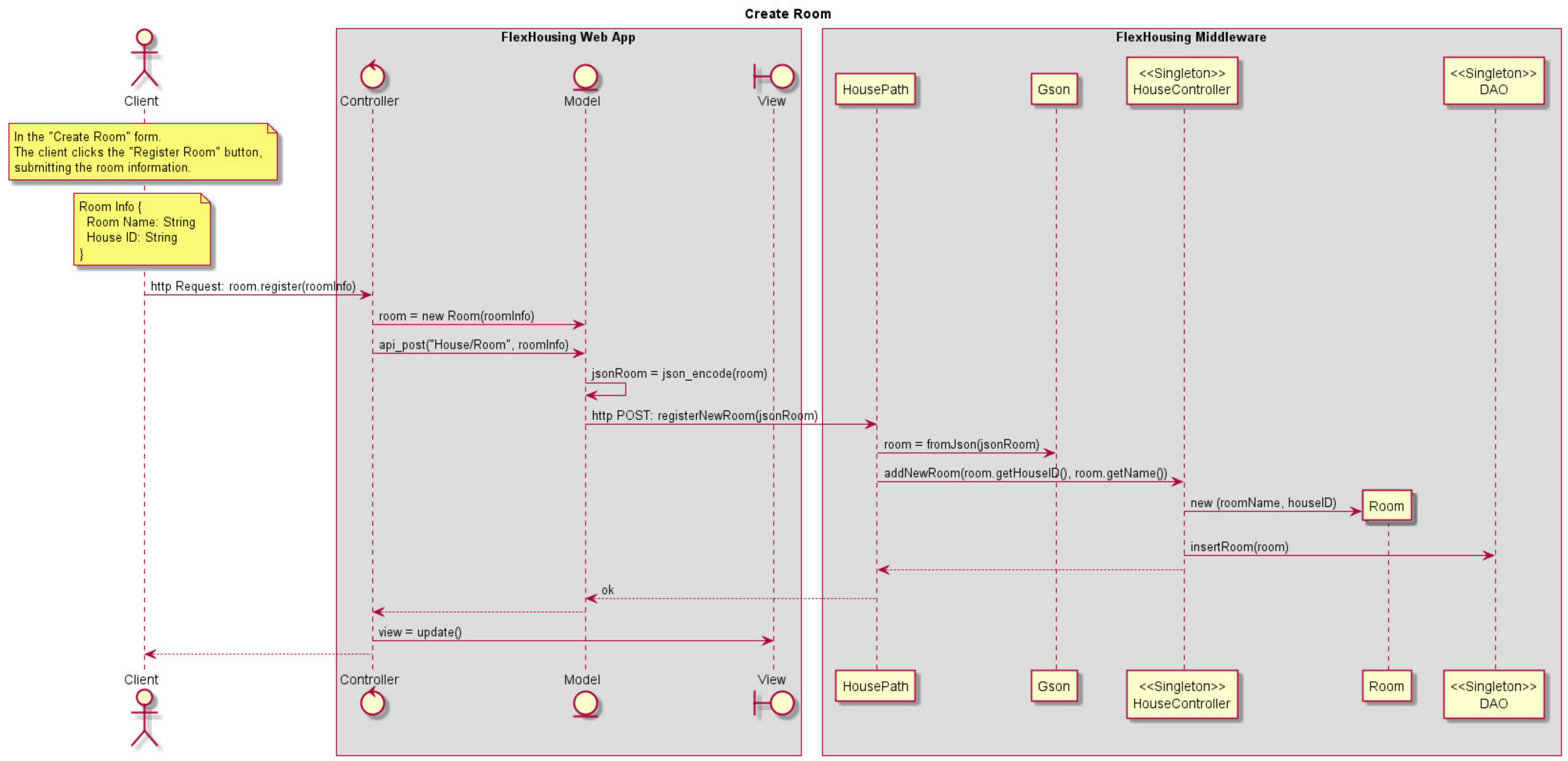

Create Room
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
Gson
<<Singleton>> HouseController
<<Singleton>> DAO
In the "Create Room" form. The client clicks the "Register Room" button, submitting the room information.
Room Info { Room Name: String House ID: String }
http Request: room.register(roomInfo)
room = new Room(roomInfo)
api_post("House/Room", roomInfo)
jsonRoom = json_encode(room)
http POST: registerNewRoom(jsonRoom)
room = fromJson(jsonRoom)
addNewRoom(room.getHouseID(), room.getName())
new (roomName, houseID)
Room
insertRoom(room)
ok
view = update()

Fig. 45 – Sequence Diagram: UC03 Create Room

4.3.2.3.2 UC03 – Read Room

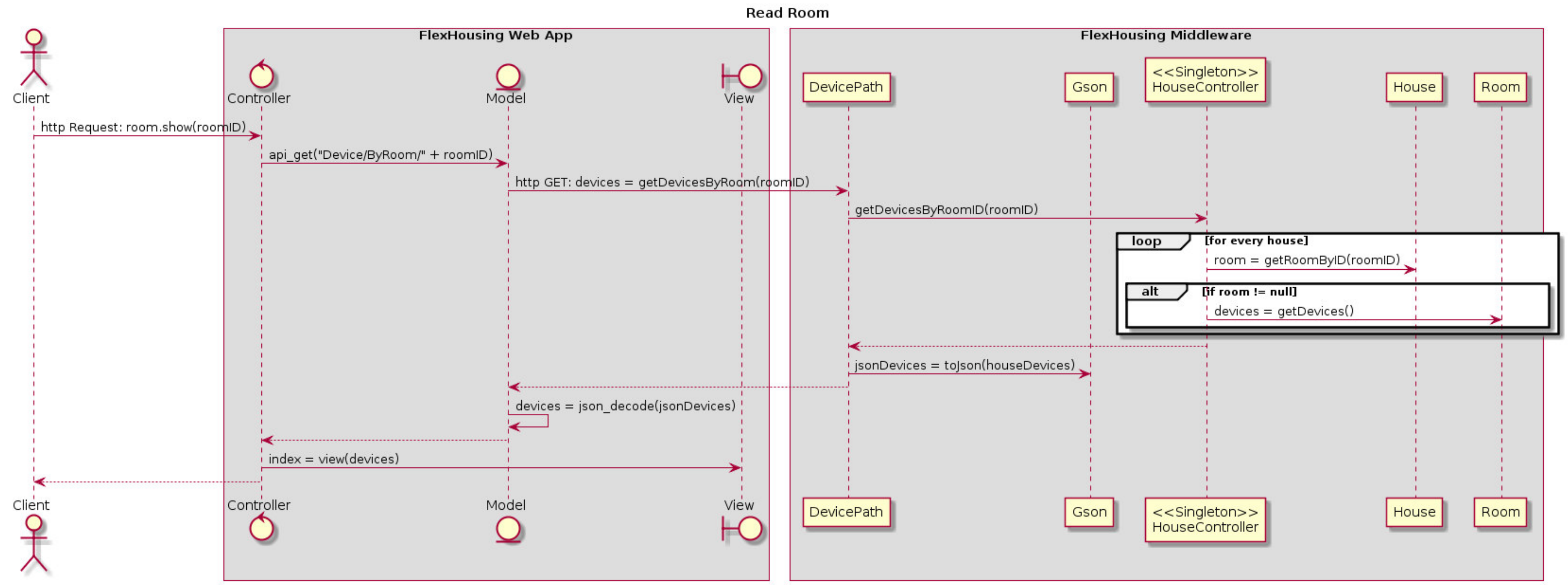
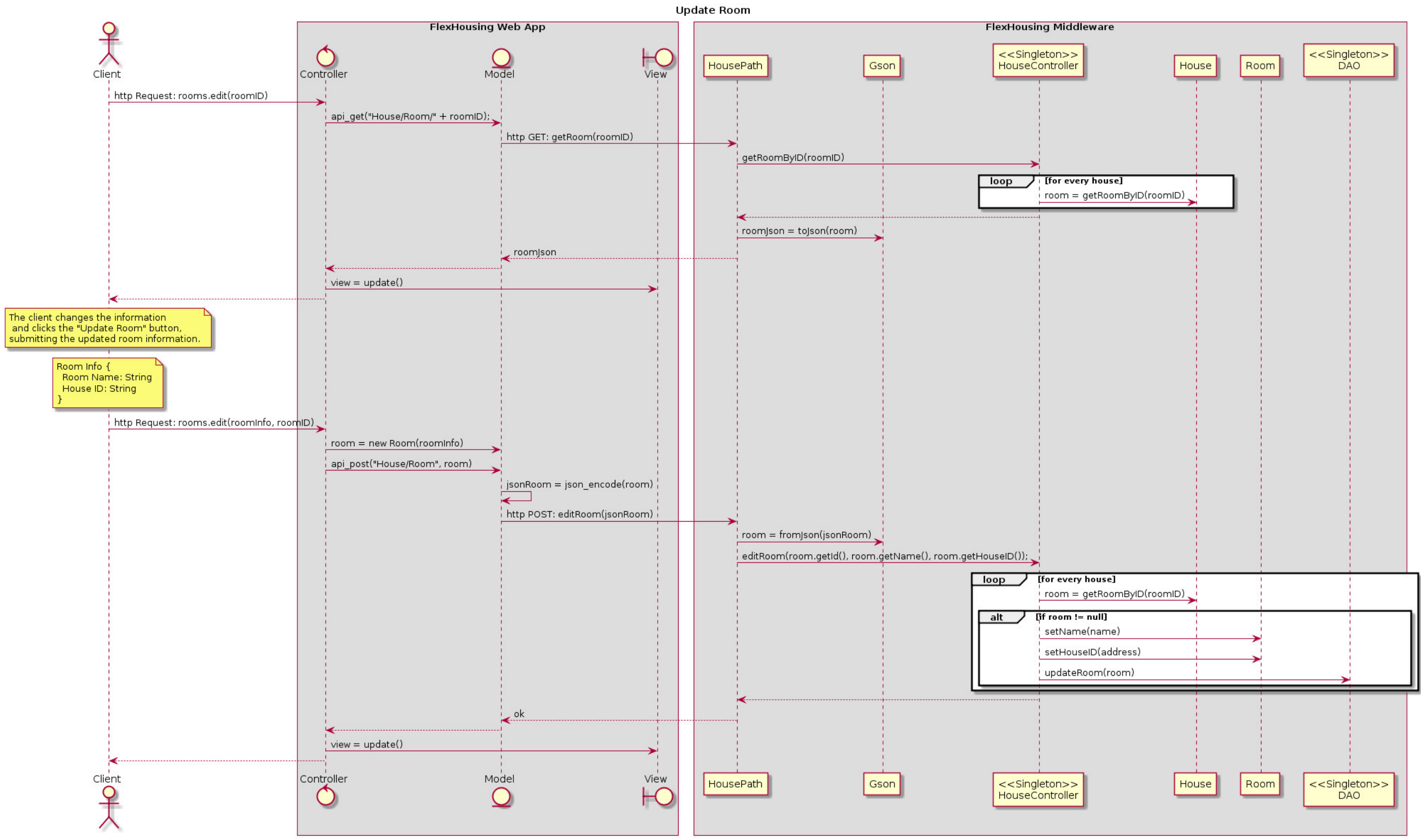

Read Room
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
DevicePath
Gson
<<Singleton>> HouseController
House
Room
http Request: room.show(roomID)
api_get("Device/ByRoom/" + roomID)
http GET: devices = getDevicesByRoom(roomID)
getDevicesByRoomID(roomID)
loop
[for every house]
room = getRoomByID(roomID)
alt
[if room != null]
devices = getDevices()
jsonDevices = toJson(houseDevices)
devices = json_decode(jsonDevices)
index = view(devices)

Fig. 46 – Sequence Diagram: UC03 Read Room

4.3.2.3.3 UC03 – Update Room



Update Room
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
Gson
<<Singleton>> HouseController
House
Room
<<Singleton>> DAO
http Request: rooms.edit(roomID)
api_get("House/Room/" + roomID);
http GET: getRoom(roomID)
getRoomByID(roomID)
loop [for every house]
room = getRoomByID(roomID)
roomJson = toJson(room)
roomJson
view = update()
The client changes the information and clicks the "Update Room" button, submitting the updated room information.
Room Info { Room Name: String House ID: String }
http Request: rooms.edit(roomInfo, roomID)
room = new Room(roomInfo)
api_post("House/Room", room)
jsonRoom = json_encode(room)
http POST: editRoom(jsonRoom)
room = fromJson(jsonRoom)
editRoom(room.getId(), room.getName(), room.getHouseID());
loop [for every house]
room = getRoomByID(roomID)
alt [if room != null]
setName(name)
setHouseID(address)
updateRoom(room)
ok
view = update()

Fig. 47 – Sequence Diagram: UC03 Update Room

4.3.2.3.4 UC03 – Delete Room

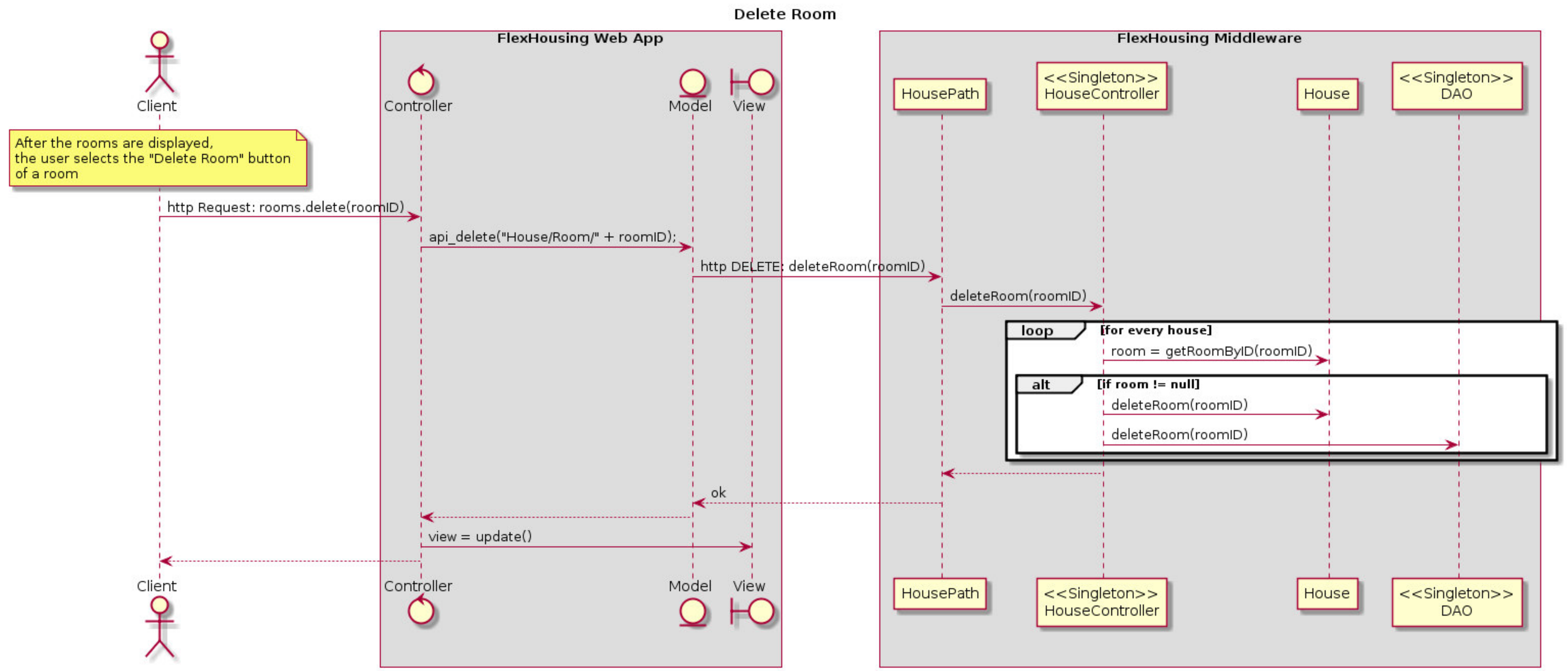

Delete Room
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
HousePath
<<Singleton>> HouseController
House
<<Singleton>> DAO
After the rooms are displayed, the user selects the "Delete Room" button of a room
http Request: rooms.delete(roomID)
api_delete("House/Room/" + roomID);
http DELETE: deleteRoom(roomID)
deleteRoom(roomID)
loop
[for every house]
room = getRoomByID(roomID)
alt
[if room != null]
deleteRoom(roomID)
deleteRoom(roomID)
ok
view = update()

Fig. 48 – Sequence Diagram: UC03 Delete Room

4.3.2.4 Use Case 04 – CRUD Device

Table 9 - Use Case 04: CRUD Device

Use Case 04 – CRUD Device	
Description	The user intends to create/read/update/delete a device.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house and room. 4. For the Update process, a device must be registered in the system.
Postconditions	1. The device is created/read/updated/deleted successfully. If the device is deleted, its registered energy consumption data is deleted as well.
Basic Flow of Events	
Actions of the actor	1. Clicks the option "Add/Edit/Delete Device". 2. If the option was a. Add/Edit, then the user inserts the required information and clicks on the button "Register/Edit Device". b. Delete, then the user confirms the deletion process.
Actions of the system	1. If the selected option was a. Add/Edit, then the system redirects to the "Create/Edit Device" form. b. Delete, then the system requests confirmation for the deletion process. 2. If the selected option was

	a. Add/Edit, then the system validates the information and registers/updates the device. b. Delete, then the system deletes the device and its registered energy consumption values.

4.3.2.4.1 UC04 – Create Device

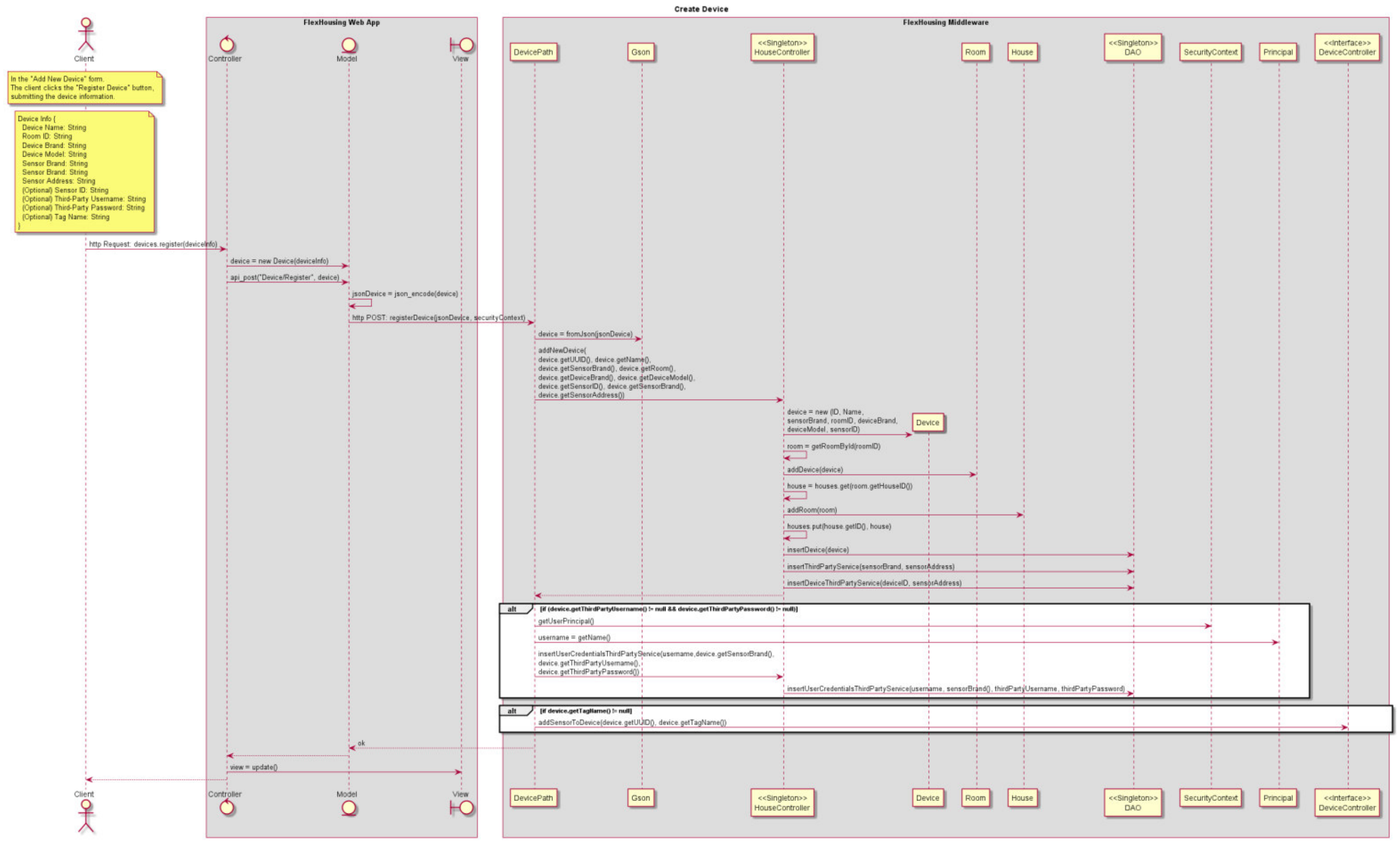

Create Device
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
DevicePath
Gson
<<Singleton>> HouseController
Device
Room
House
<<Singleton>> DAO
SecurityContext
Principal
<<Interface>> DeviceController
In the "Add New Device" form. The client clicks the "Register Device" button, submitting the device information.
Device Info { Device Name: String Room ID: String Device Brand: String Device Model: String Sensor Brand: String Sensor Brand: String Sensor Address: String (Optional) Sensor ID: String (Optional) Third-Party Username: String (Optional) Third-Party Password: String (Optional) Tag Name: String }
http Request: devices.register(deviceInfo)
device = new Device(deviceInfo)
api_post("Device/Register", device)
jsonDevice = json_encode(device)
http POST: registerDevice(jsonDevice, securityContext)
device = fromJson(jsonDevice)
addNewDevice(device.getUUID(), device.getName(), device.getSensorBrand(), device.getRoom(), device.getDeviceBrand(), device.getDeviceModel(), device.getSensorID(), device.getSensorBrand(), device.getSensorAddress())
device = new (ID, Name, sensorBrand, roomID, deviceBrand, deviceModel, sensorID)
room = getRoomById(roomID)
addDevice(device)
house = houses.get(room.getHouseID())
addRoom(room)
houses.put(house.getID(), house)
insertDevice(device)
insertThirdPartyService(sensorBrand, sensorAddress)
insertDeviceThirdPartyService(deviceID, sensorAddress)
alt
[if (device.getThirdPartyUsername() != null && device.getThirdPartyPassword() != null)]
getUserPrincipal()
username = getName()
insertUserCredentialsThirdPartyService(username,device.getSensorBrand(), device.getThirdPartyUsername(), device.getThirdPartyPassword())
insertUserCredentialsThirdPartyService(username, sensorBrand(), thirdPartyUsername, thirdPartyPassword)
alt
[if device.getTagName() != null]
addSensorToDevice(device.getUUID(), device.getTagName())
ok
view = update()

Fig. 49 – Sequence Diagram: UC04 Create Device

4.3.2.4.2 UC04 – Update Device

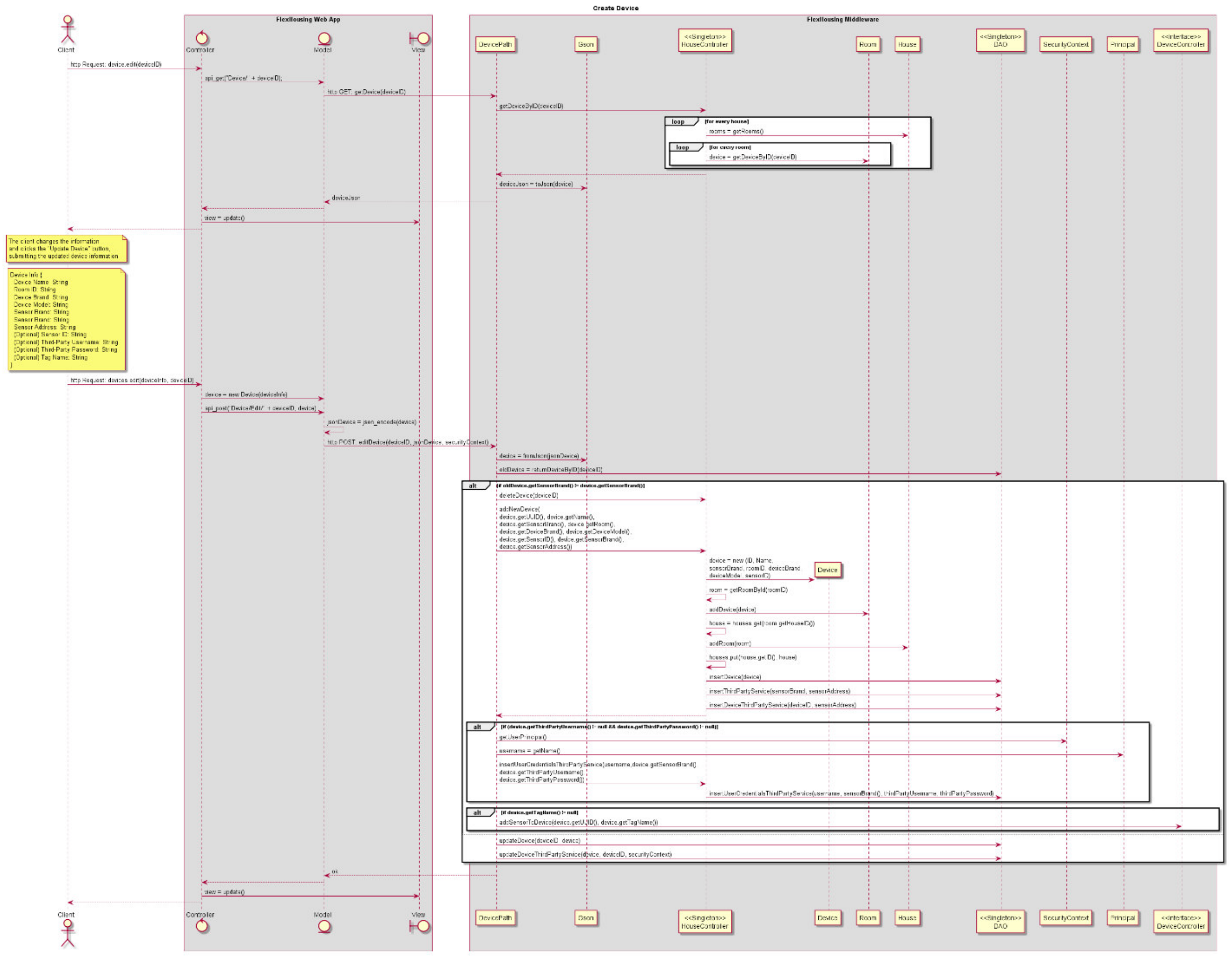


Fig. 50 – Sequence Diagram: UC04 Update Device

4.3.2.4.3 UC04 – Delete Device

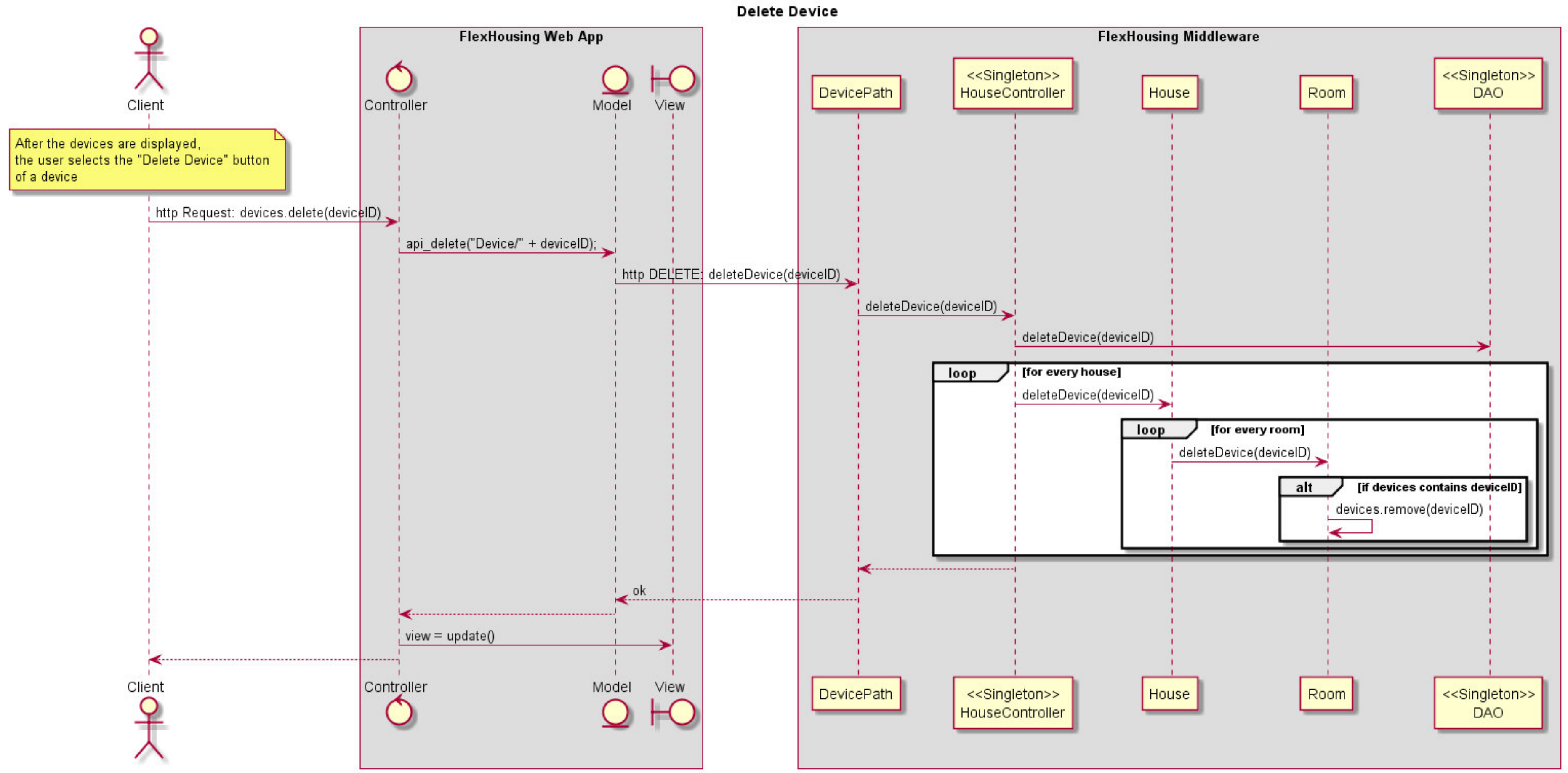


Fig. 51 – Sequence Diagram: UC04 Delete Device

4.3.2.5 Use Case 05 – Check All Devices

Table 10 – Use Case 05: Check All Devices

Use Case 05 – Check All Devices	
Description	The user intends to check all registered devices.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house and room.
Postconditions	1. All registered devices are successfully displayed.
Basic Flow of Events	
Actions of the actor	1. Selects the option "Check Devices".
Actions of the system	1. Redirects to the Devices index webpage and displays the registered devices.

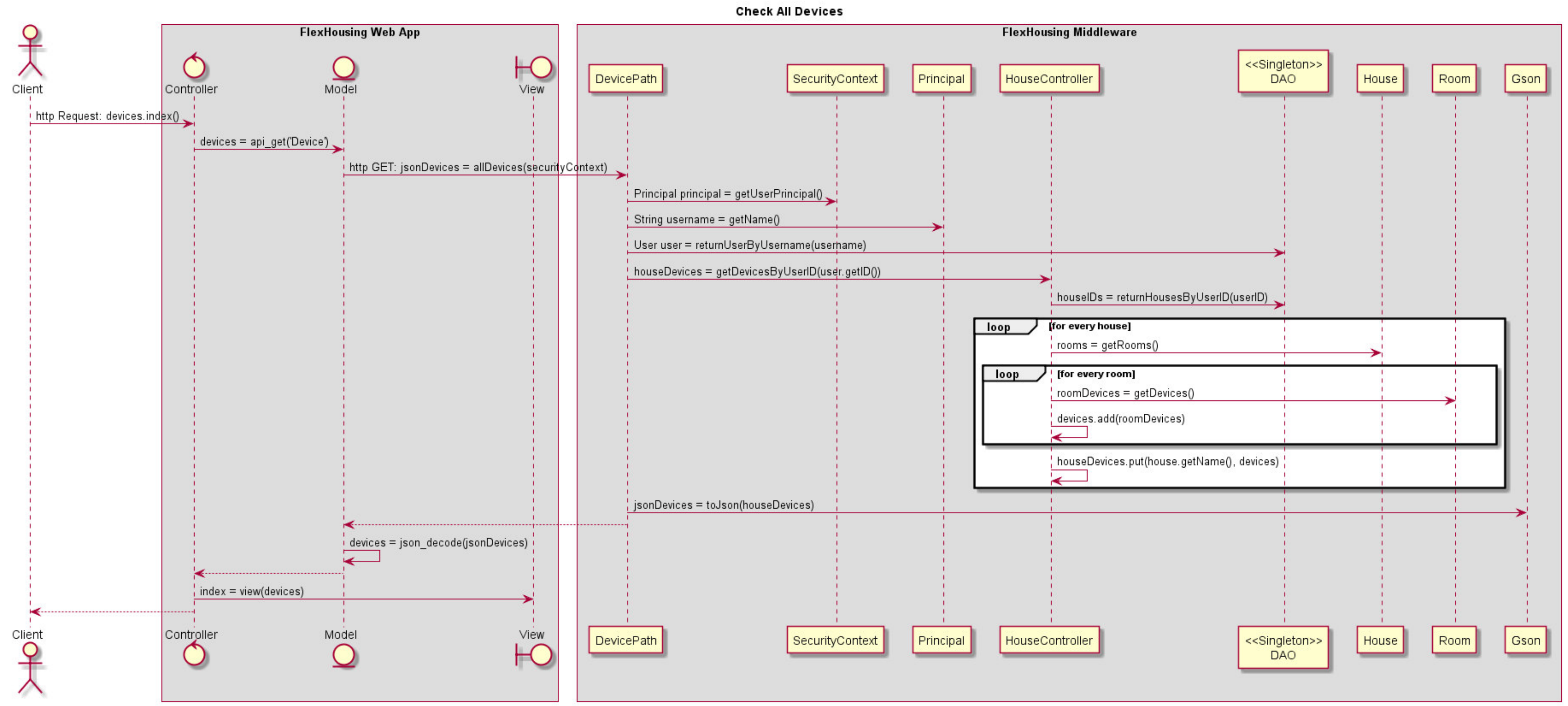

Check All Devices
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
DevicePath
SecurityContext
Principal
HouseController
<<Singleton>> DAO
House
Room
Gson
http Request: devices.index()
devices = api_get('Device')
http GET: jsonDevices = allDevices(securityContext)
Principal principal = getUserPrincipal()
String username = getName()
User user = returnUserByUsername(username)
houseDevices = getDevicesByUserID(user.getID())
houseIDs = returnHousesByUserID(userID)
loop [for every house]
rooms = getRooms()
loop [for every room]
roomDevices = getDevices()
devices.add(roomDevices)
houseDevices.put(house.getName(), devices)
jsonDevices = toJson(houseDevices)
devices = json_decode(jsonDevices)
index = view(devices)

Fig. 52 – Sequence Diagram: UC05 Check All Devices

4.3.2.6 Use Case 06 – Turn On/Off Device

Table 11 - Use Case 06: Turn On/Off Device

Use Case 06 – Turn On/Off Device	
Description	The user intends to turn on/off a device.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*“http://flexhousing.app”*). 2. Be authenticated in the system as a user. 3. Have a registered house, room, and device. 4. The device’s REST API address must be correct and accessible.
Postconditions	1. The device is turned on/off successfully.
Basic Flow of Events	
Actions of the actor	1. Selects the option “Check Devices”. 2. Clicks the button “Turn On/Off” of a specific device.
Actions of the system	1. Redirects to the Devices index webpage. 2. Sends a request to actuate the specified device, turning it on/off.

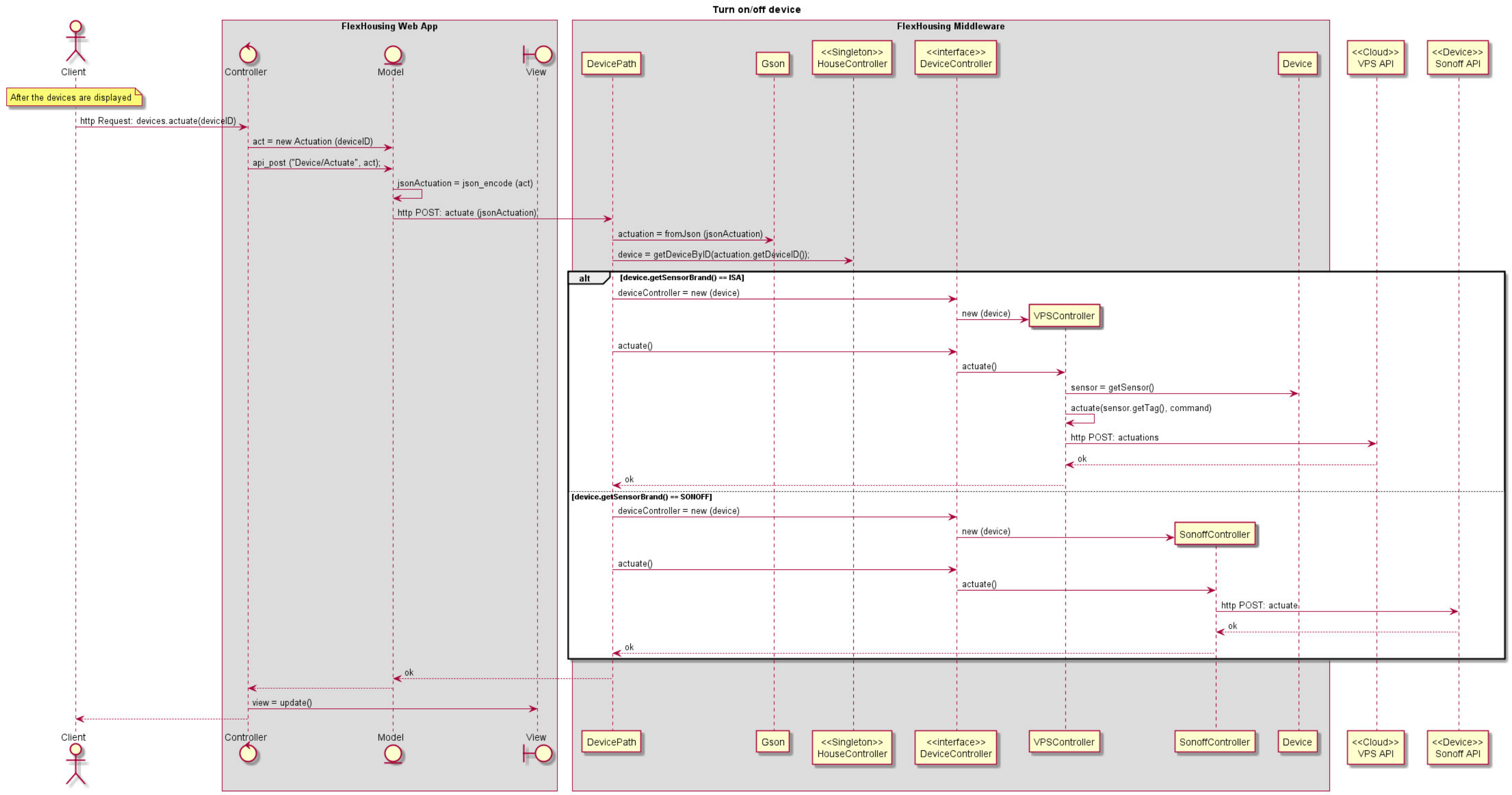

Turn on/off device
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
DevicePath
Gson
<<Singleton>> HouseController
<<interface>> DeviceController
Device
<<Cloud>> VPS API
<<Device>> Sonoff API
After the devices are displayed
http Request: devices.actuate(deviceID)
act = new Actuation (deviceID)
api_post ("Device/Actuate", act);
jsonActuation = json_encode (act)
http POST: actuate (jsonActuation)
actuation = fromJson (jsonActuation)
device = getDeviceByID(actuation.getDeviceID());
alt
[device.getSensorBrand() == ISA]
deviceController = new (device)
new (device)
VPSController
actuate()
actuate()
sensor = getSensor()
actuate(sensor.getTag(), command)
http POST: actuations
ok
ok
[device.getSensorBrand() == SONOFF]
deviceController = new (device)
new (device)
SonoffController
actuate()
actuate()
http POST: actuate
ok
ok
ok
view = update()

Fig. 53 – Sequence Diagram: UC06 Turn On/Off Device

4.3.2.7 Use Case 07 – Check Device Consumption

Table 12 - Use Case 07: Check Device Consumption

Use Case 07 – Check Device Consumption	
Description	The user intends to check a device's energy consumption data.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house, room, and device.
Postconditions	1. The device's energy consumption data is displayed in multiple charts, each with different granularity and functionality.
Basic Flow of Events	
Actions of the actor	1. Selects the option "Check Devices". 2. Clicks the button "Check Consumption" of a specific device. 3. Selects a date or time window (depending on the chart) to check the energy consumption data.
Actions of the system	1. Redirects to the Devices index webpage. 2. Sends a request to get the current energy consumption data of the specified device. Redirects the user to the Energy Consumption page and displays the charts. 3. Sends a request to get the data from the specified date or time window and updates the charts (based on the acquired data).

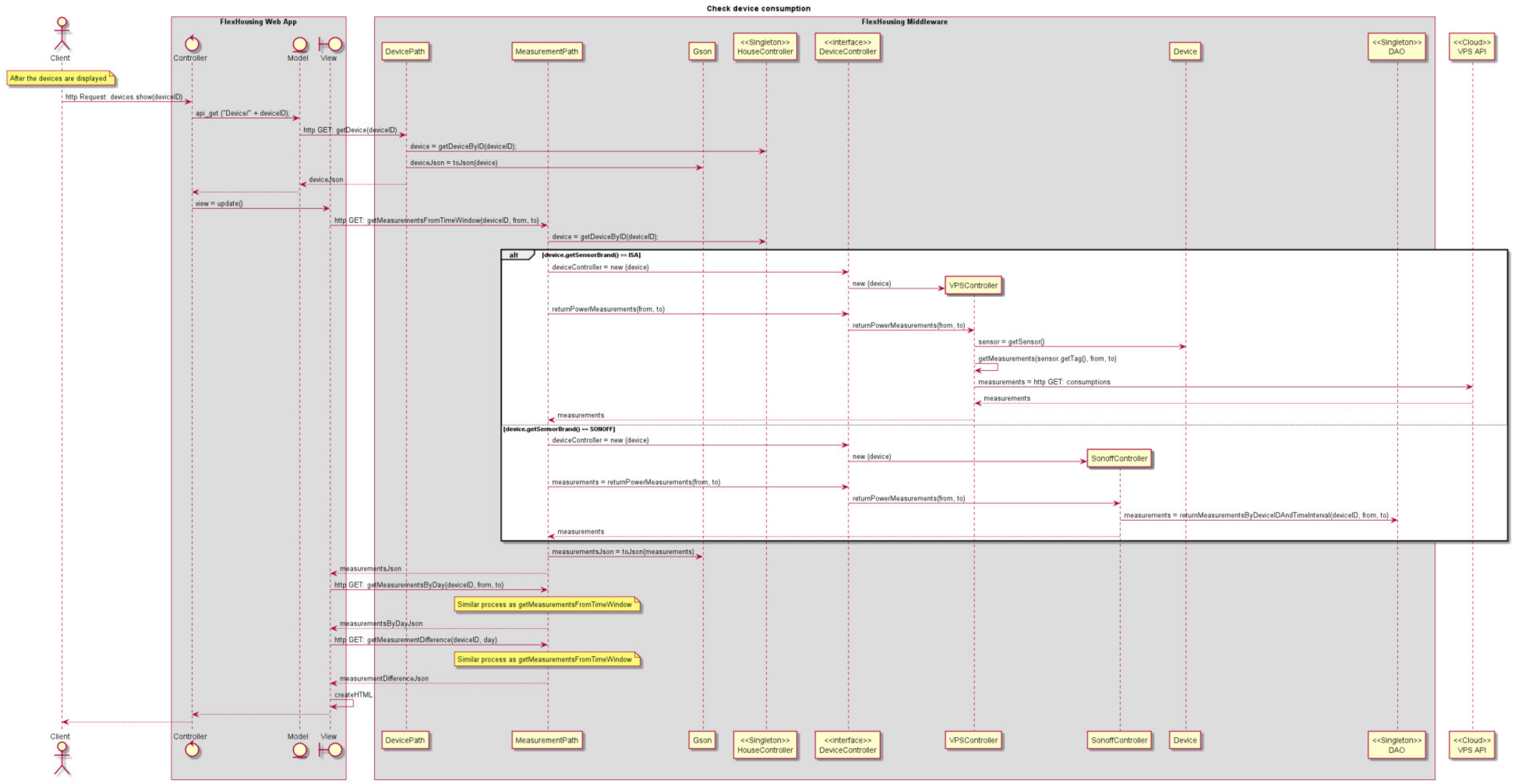

Check device consumption
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
DevicePath
MeasurementPath
Gson
<<Singleton>> HouseController
<<interface>> DeviceController
Device
<<Singleton>> DAO
<<Cloud>> VPS API
After the devices are displayed
http Request: devices.show(deviceID)
api_get ("Device/" + deviceID);
http GET: getDevice(deviceID)
device = getDeviceByID(deviceID);
deviceJson = toJson(device)
deviceJson
view = update()
http GET: getMeasurementsFromTimeWindow(deviceID, from, to)
device = getDeviceByID(deviceID);
alt
[device.getSensorBrand() == ISA]
deviceController = new (device)
new (device)
VPSController
returnPowerMeasurements(from, to)
returnPowerMeasurements(from, to)
sensor = getSensor()
getMeasurements(sensor.getTag(), from, to)
measurements = http GET: consumptions
measurements
measurements
[device.getSensorBrand() == SONOFF]
deviceController = new (device)
new (device)
SonoffController
measurements = returnPowerMeasurements(from, to)
returnPowerMeasurements(from, to)
measurements = returnMeasurementsByDeviceIDAndTimeInterval(deviceID, from, to)
measurements
measurementsJson = toJson(measurements)
measurementsJson
http GET: getMeasurementsByDay(deviceID, from, to)
Similar process as getMeasurementsFromTimeWindow
measurementsByDayJson
http GET: getMeasurementDifference(deviceID, day)
Similar process as getMeasurementsFromTimeWindow
measurementDifferenceJson
createHTML

Fig. 54 – Sequence Diagram: UC07 Check Device Consumption

4.3.2.8 Use Cases 08 & 09 – Create Flex-offer

4.3.2.8.1 Use Case 08 – Create Flex-offer Manually for a Device

Table 13 - Use Case 08: Create Flex-offer Manually for a Device

Use Case 08 – Create Flexoffer Manually for a Device	
Description	The user intends to create a flex-offer and apply it to a device.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house, room, and device.
Postconditions	1. The flex-offer is registered successfully.
Basic Flow of Events	
Actions of the actor	1. Selects the option "Check Devices". 2. Clicks the button "Create Flex-offer" of a specific device. 3. Fills out the Flex-offer form. 4. Selects the option to manually create the flexoffer. 5. Continues to fill out the Flex-offer form. 6. Clicks the button "Finish".
Actions of the system	1. Redirects to the Devices index webpage. 2. Redirects to the "Create Flex-offer" form webpage. 3. Validates the information being entered by the user. 4. Displays the "manual flex-offer creation" form. 5. Validates the information being entered by the user. 6. Registers the flex-offer.

4.3.2.8.2 Use Case 09 – Create Flex-offer Automatically for a Device, based on Energy Consumption

Table 14 – Use Case 09: Create Flex-offer Automatically for a Device, based on Energy Consumption

Use Case 09 – Create Flex-offer Automatically for a Device, based on Energy Consumption

Description	The user intends to create a flex-offer and apply it to a device.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house, room, and device.
Postconditions	2. The flexoffer is registered successfully.
Basic Flow of Events	
Actions of the actor	1. Selects the option "Check Devices". 2. Clicks the button "Create Flex-offer" of a specific device. 3. Fills out the Flex-offer form. 4. Selects the option to automatically create the flex-offer. 5. Continues to fill out the Flex-offer form. 6. Clicks the button "Finish".
Actions of the system	1. Redirects to the Devices index webpage. 2. Redirects to the "Create Flex-offer" form webpage. 3. Validates the information being entered by the user. 4. Displays the "automatic flex-offer creation" form. 5. Validates the information being entered by the user. 6. Registers the flex-offer.

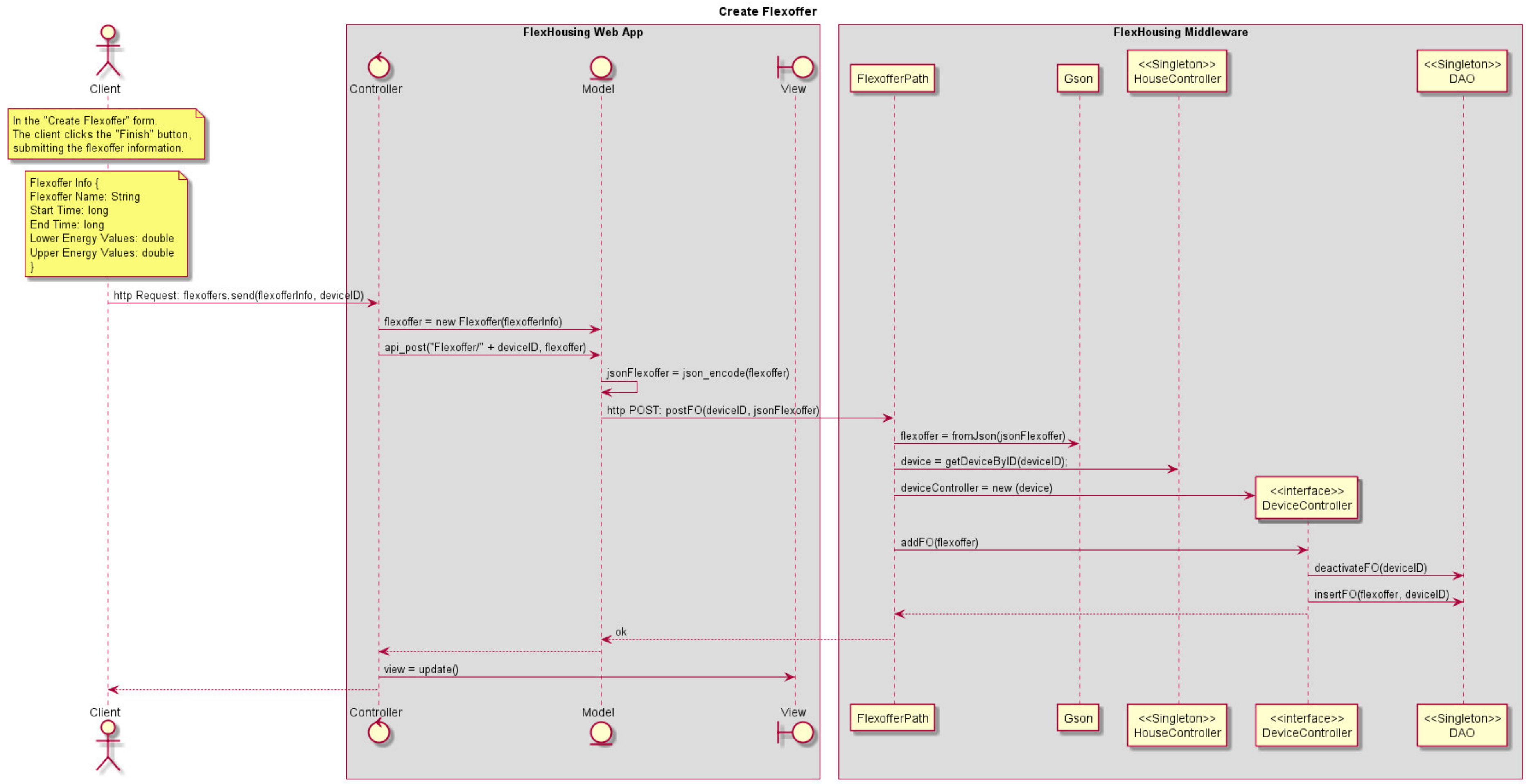

Create Flexoffer
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
FlexofferPath
Gson
<<Singleton>> HouseController
<<Singleton>> DAO
In the "Create Flexoffer" form. The client clicks the "Finish" button, submitting the flexoffer information.
Flexoffer Info { Flexoffer Name: String Start Time: long End Time: long Lower Energy Values: double Upper Energy Values: double }
http Request: flexoffers.send(flexofferInfo, deviceID)
flexoffer = new Flexoffer(flexofferInfo)
api_post("Flexoffer/" + deviceID, flexoffer)
jsonFlexoffer = json_encode(flexoffer)
http POST: postFO(deviceID, jsonFlexoffer)
flexoffer = fromJson(jsonFlexoffer)
device = getDeviceByID(deviceID);
deviceController = new (device)
<<interface>> DeviceController
addFO(flexoffer)
deactivateFO(deviceID)
insertFO(flexoffer, deviceID)
ok
view = update()

Fig. 55 – Sequence Diagram: UC08 & UC09 Create Flex-offer

4.3.2.9 Use Case 10 – Check a Device's active Flex-offer and its effectiveness

Table 15 – Use Case 10: Check a Device's active Flex-offer and its effectiveness

Use Case 10 – Check a Device's active Flex-offer and its effectiveness	
Description	The user intends to check a device's active flex-offer and see if it had an effect on the device's consumption pattern.
Actor(s)	Registered End User
Preconditions	1. Have access to the platform website (*"http://flexhousing.app"*). 2. Be authenticated in the system as a user. 3. Have a registered house, room, and device. 4. Have already created a flex-offer for a device.
Postconditions	1. The active flex-offer's information is displayed successfully.
Basic Flow of Events	
Actions of the actor	1. Selects the option "Check Devices". 2. Clicks the button "Check Flex-offer" of a specific device.
Actions of the system	1. Redirects to the Devices index webpage. 2. Redirects to the "Check Flex-offer" webpage. 3. Displays the active flex-offer, its schedule, and the mean absolute percentage error between the flex-offer's projection and the device's actual consumption.

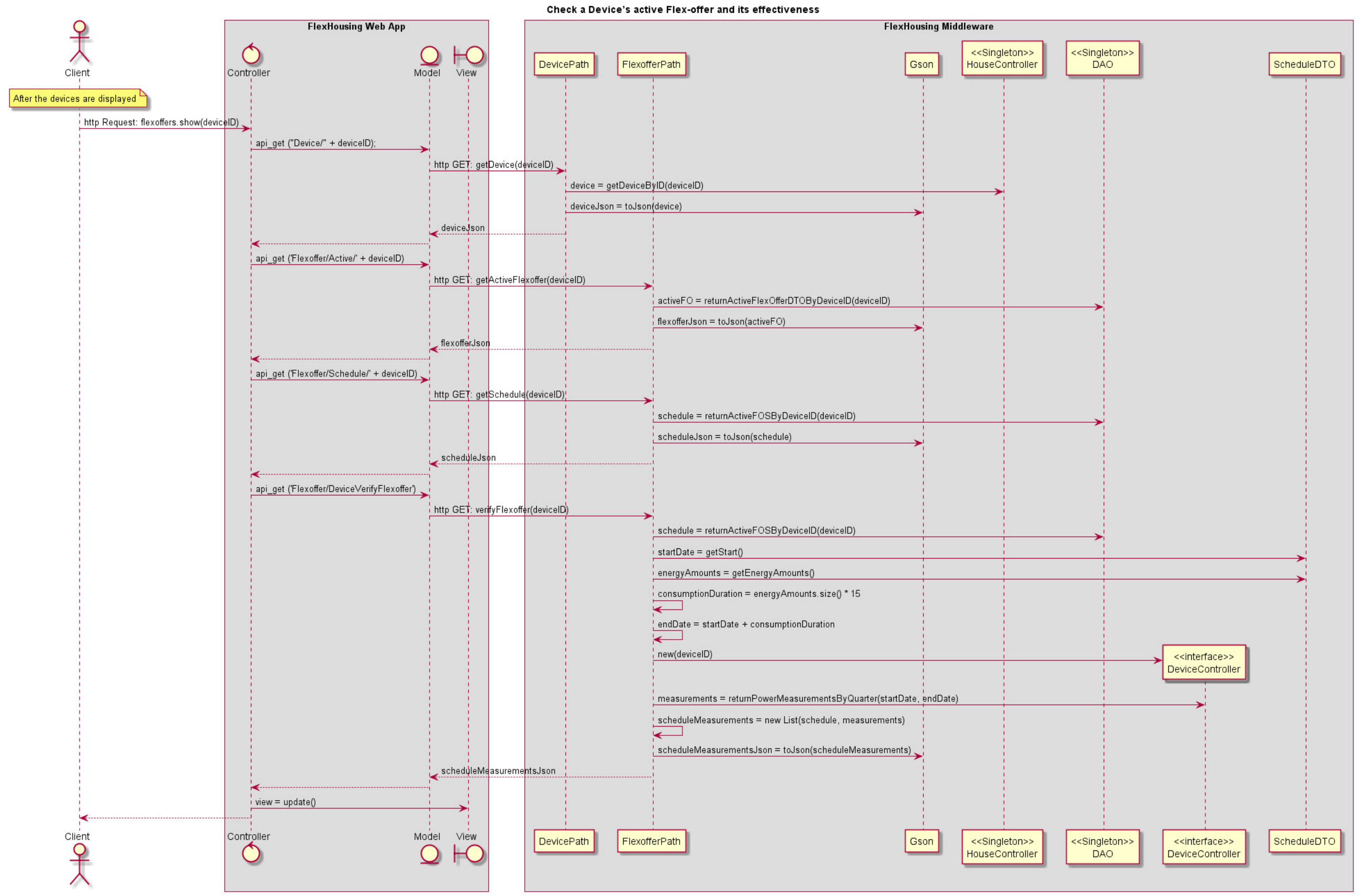

Check a Device's active Flex-offer and its effectiveness
FlexHousing Web App
FlexHousing Middleware
Client
Controller
Model
View
DevicePath
FlexofferPath
Gson
<<Singleton>> HouseController
<<Singleton>> DAO
ScheduleDTO
After the devices are displayed
http Request: flexoffers.show(deviceID)
api_get ("Device/" + deviceID);
http GET: getDevice(deviceID)
device = getDeviceByID(deviceID)
deviceJson = toJson(device)
deviceJson
api_get ('Flexoffer/Active/' + deviceID)
http GET: getActiveFlexoffer(deviceID)
activeFO = returnActiveFlexOfferDTOByDeviceID(deviceID)
flexofferJson = toJson(activeFO)
flexofferJson
api_get ('Flexoffer/Schedule/' + deviceID)
http GET: getSchedule(deviceID)
schedule = returnActiveFOSByDeviceID(deviceID)
scheduleJson = toJson(schedule)
scheduleJson
api_get ('Flexoffer/DeviceVerifyFlexoffer')
http GET: verifyFlexoffer(deviceID)
schedule = returnActiveFOSByDeviceID(deviceID)
startDate = getStart()
energyAmounts = getEnergyAmounts()
consumptionDuration = energyAmounts.size() * 15
endDate = startDate + consumptionDuration
new(deviceID)
<<interface>> DeviceController
measurements = returnPowerMeasurementsByQuarter(startDate, endDate)
scheduleMeasurements = new List(schedule, measurements)
scheduleMeasurementsJson = toJson(scheduleMeasurements)
scheduleMeasurementsJson
view = update()

Fig. 56 – Sequence Diagram: UC10 Check a Device's active Flex-offer and its effectiveness

4.3.2.10 Use Cases 11 & 12 – Get Device and User metadata

Table 16 – Use Case 11 & 12: Check total registered Users, Devices, and Houses & Check End Users' Devices' Frequency of Use and Average Time of Use

Use Case 11 & 12 – Check total registered Users, Devices, and Houses & Check End Users' Devices' Frequency of Use and Average Time of Use	
Description	The user intends to check the total number of registered users, devices, and houses, and check how many times a device brand is used, as well as its average time of use.
Actor(s)	Company Executive
Preconditions	1. Have access to the platform website (*"http://executives-data.app"*). 2. Be authenticated in the system as a user.
Postconditions	1. The requested information is displayed successfully.
Basic Flow of Events	
Actions of the actor	1. Accesses the platform and goes to the dashboard.
Actions of the system	1. Acquires the requested data and displays it in the dashboard.

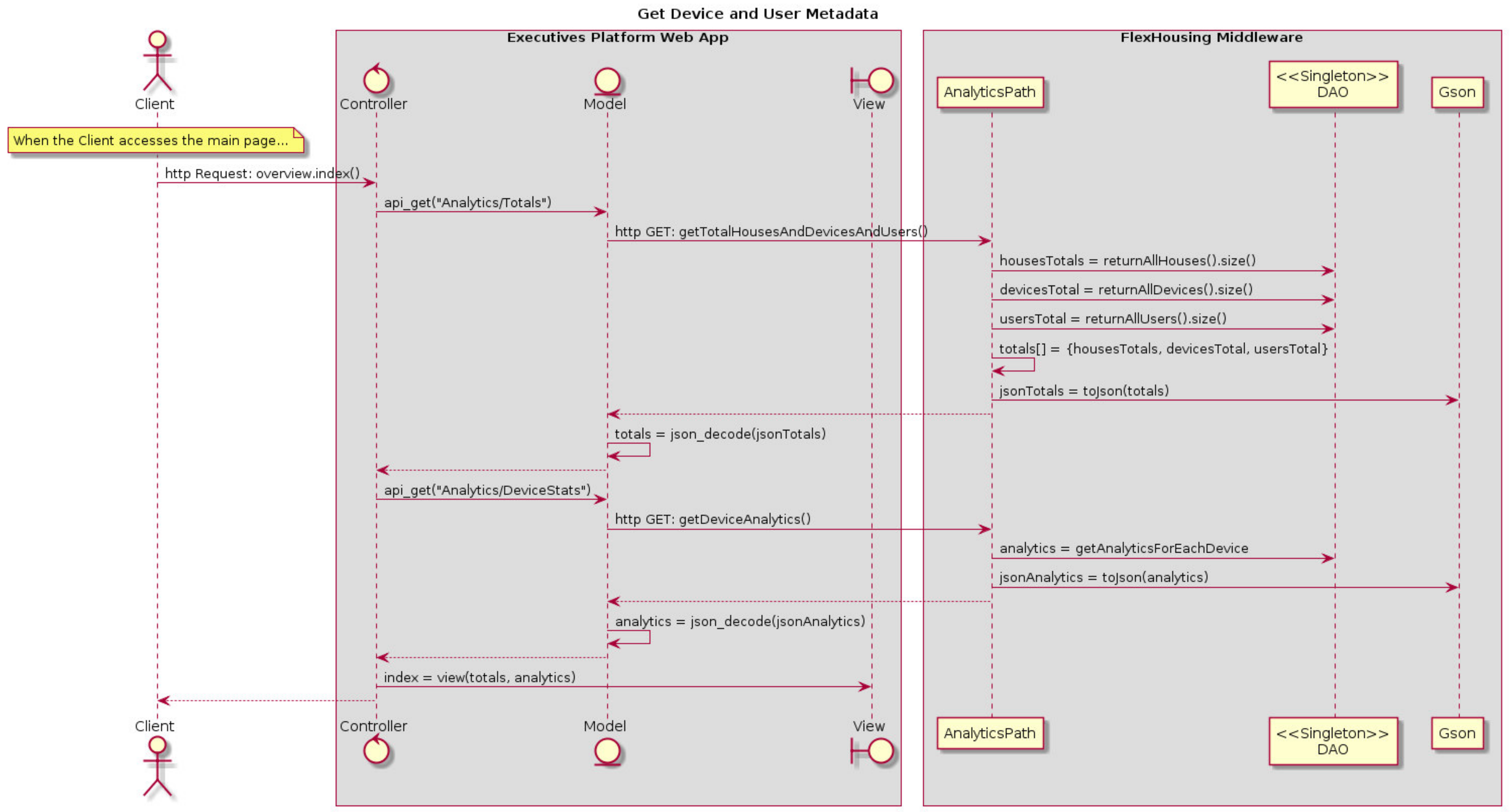

Get Device and User Metadata
Executives Platform Web App
FlexHousing Middleware
Client
Controller
Model
View
AnalyticsPath
<<Singleton>> DAO
Gson
When the Client accesses the main page...
http Request: overview.index()
api_get("Analytics/Totals")
http GET: getTotalHousesAndDevicesAndUsers()
housesTotals = returnAllHouses().size()
devicesTotal = returnAllDevices().size()
usersTotal = returnAllUsers().size()
totals[] = {housesTotals, devicesTotal, usersTotal}
jsonTotals = toJson(totals)
totals = json_decode(jsonTotals)
api_get("Analytics/DeviceStats")
http GET: getDeviceAnalytics()
analytics = getAnalyticsForEachDevice
jsonAnalytics = toJson(analytics)
analytics = json_decode(jsonAnalytics)
index = view(totals, analytics)

Fig. 57 – Sequence Diagram: UC11 & UC12 Get Device and User metadata

4.3.2.11 Use Case 13 – Get Devices' Consumption Values

Table 17 - Use Case 13: Get Devices' Consumption Values

Use Case 13 – Get Devices' Consumption Values	
Description	The system intends to periodically acquire its registered devices' energy consumption values and register them.
Actor(s)	FlexHousing Middleware
Preconditions	1. Have at least one device registered on the system. 2. The device's REST API address must be correct and accessible.
Postconditions	1. The device's energy consumption values are registered successfully.
Basic Flow of Events	
Actions of the actor	1. Requests energy consumption values from the device's REST API endpoint and registers them.
Actions of the system (Device)	1. Receives request and sends the requested data.

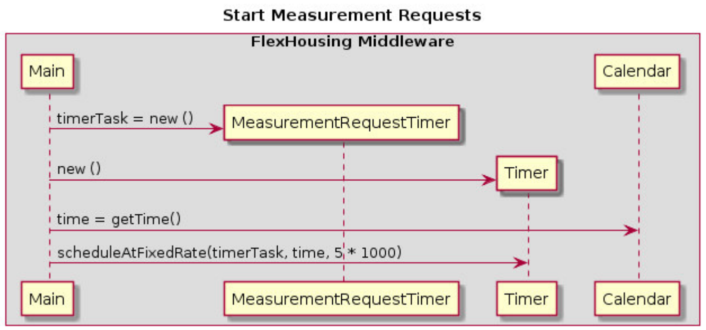


Fig. 58 – Sequence Diagram: UC11 Start Measurement Requests

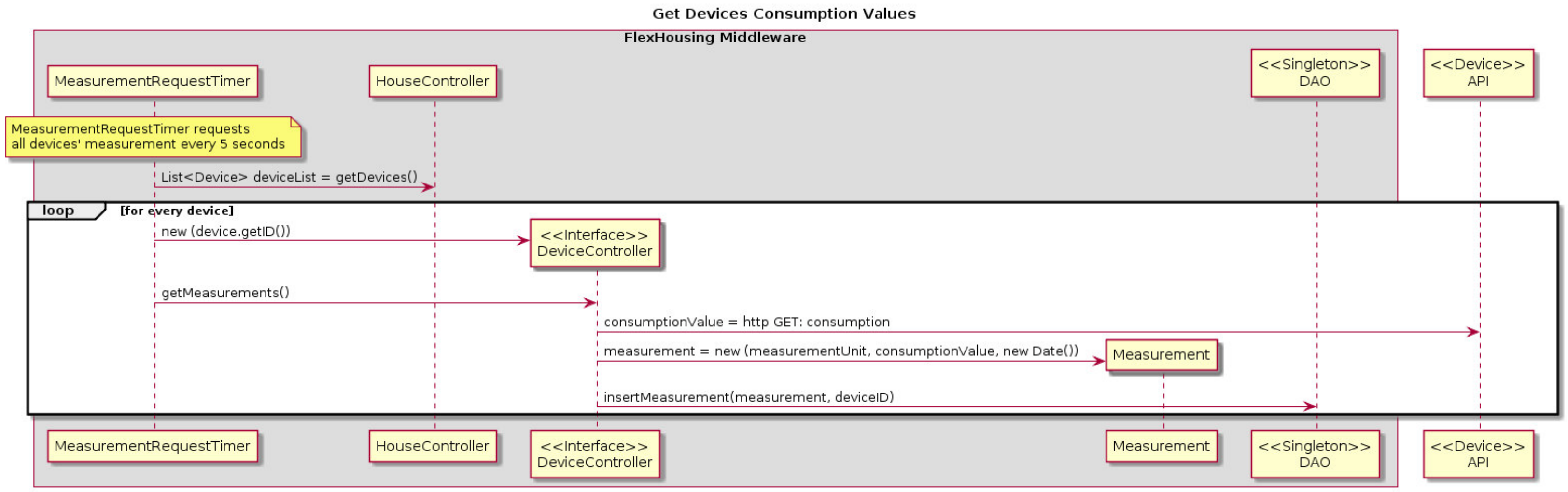

Get Devices Consumption Values
FlexHousing Middleware
MeasurementRequestTimer
HouseController
<<Singleton>> DAO
<<Device>> API
MeasurementRequestTimer requests all devices' measurement every 5 seconds
List<Device> deviceList = getDevices()
loop [for every device]
new (device.getID())
<<Interface>> DeviceController
getMeasurements()
consumptionValue = http GET: consumption
measurement = new (measurementUnit, consumptionValue, new Date())
Measurement
insertMeasurement(measurement, deviceID)
MeasurementRequestTimer
HouseController
<<Interface>> DeviceController
Measurement
<<Singleton>> DAO
<<Device>> API

Fig. 59 – Sequence Diagram: UC13 Get Devices Consumption Values

4.3.2.12 Use Case 14 – Deploy Device's Flex-offers

Table 18 - Use Case 14: Deploy Device's Flex-offers

Use Case 14 – Deploy Device's Flex-offers	
Description	The system intends to periodically deploy its registered flex-offers to their respective devices.
Actor(s)	FlexHousing Middleware
Preconditions	1. Have at least one device registered on the system. 2. Have at least one flex-offer registered on the system. 3. The device's REST API address must be correct and accessible.
Postconditions	1. The device's flex-offer is deployed successfully.
Basic Flow of Events	
Actions of the actor	1. Sends the device's flexoffer to the Aggregator module. Waits until it receives a schedule to begin the device's actuations 2. Actuates the device on the determined schedule.
Actions of the system (Aggregator)	1. Receives flex-offer, aggregates it with other flex-offers, and sends them to the VME module. Then, responds with a schedule to begin the actuations.

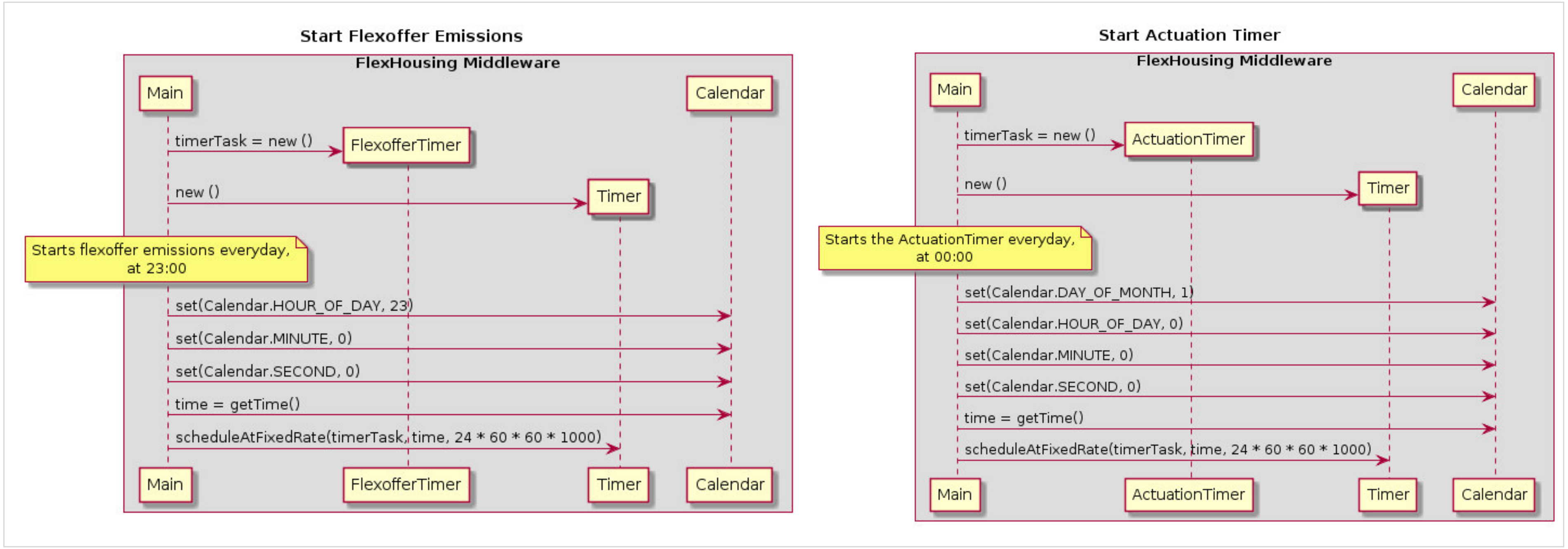

Start Flexoffer Emissions
FlexHousing Middleware
Main
FlexofferTimer
Timer
Calendar
timerTask = new ()
new ()
Starts flexoffer emissions everyday, at 23:00
set(Calendar.HOUR_OF_DAY, 23)
set(Calendar.MINUTE, 0)
set(Calendar.SECOND, 0)
time = getTime()
scheduleAtFixedRate(timerTask, time, 24 * 60 * 60 * 1000)
Start Actuation Timer
FlexHousing Middleware
Main
ActuationTimer
Timer
Calendar
timerTask = new ()
new ()
Starts the ActuationTimer everyday, at 00:00
set(Calendar.DAY_OF_MONTH, 1)
set(Calendar.HOUR_OF_DAY, 0)
set(Calendar.MINUTE, 0)
set(Calendar.SECOND, 0)
time = getTime()
scheduleAtFixedRate(timerTask, time, 24 * 60 * 60 * 1000)

Fig. 60 – Sequence Diagram: Left – UC14 Start Flex-offer Emissions; Right – UC14 Start Actuation Timer

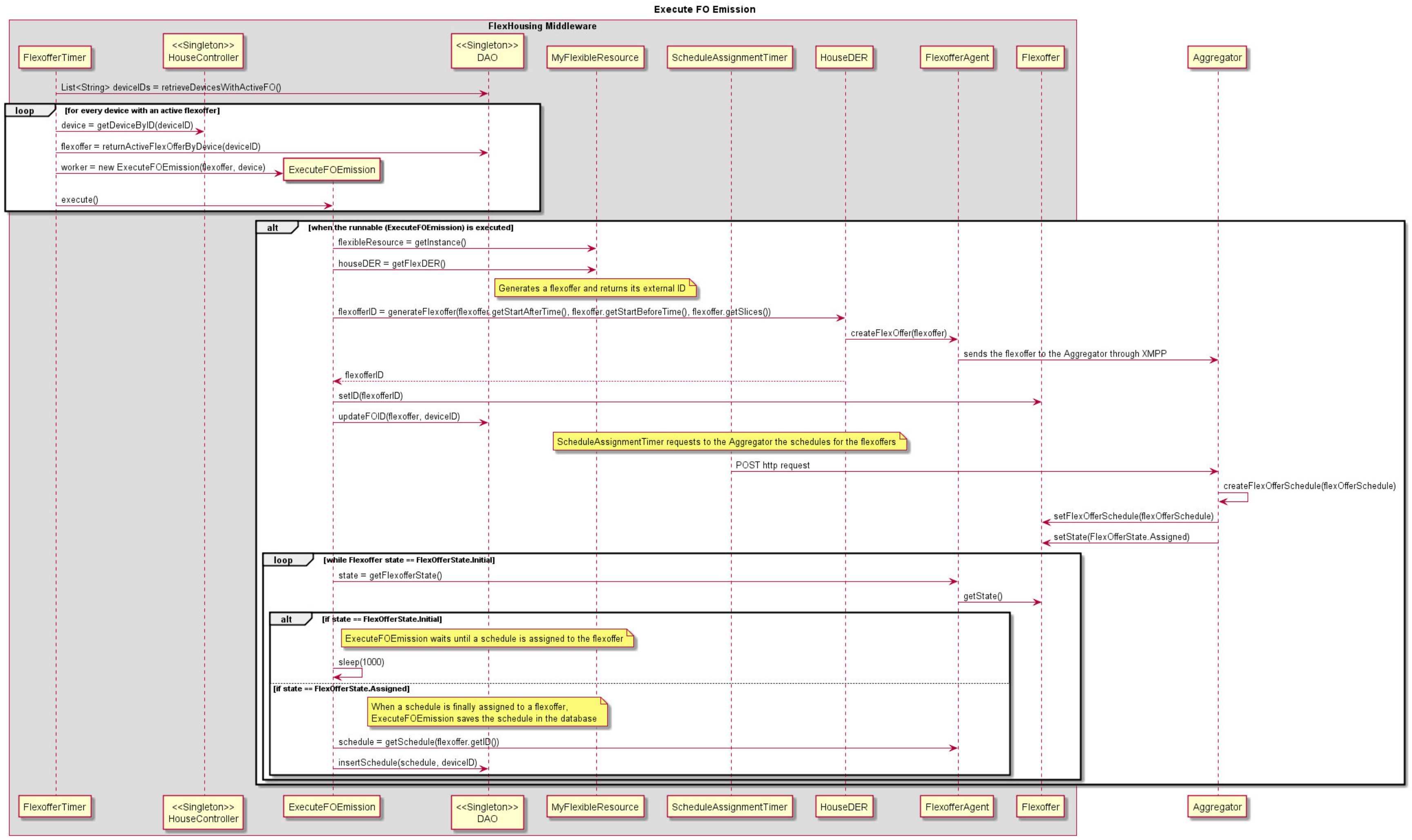

Execute FO Emission
FlexHousing Middleware
FlexofferTimer
<<Singleton>> HouseController
<<Singleton>> DAO
MyFlexibleResource
ScheduleAssignmentTimer
HouseDER
FlexofferAgent
Flexoffer
Aggregator
List<String> deviceIds = retrieveDevicesWithActiveFO()
loop [for every device with an active flexoffer]
device = getDeviceByID(deviceID)
flexoffer = returnActiveFlexOfferByDevice(deviceID)
worker = new ExecuteFOEmission(flexoffer, device)
ExecuteFOEmission
execute()
alt [when the runnable (ExecuteFOEmission) is executed]
flexibleResource = getInstance()
houseDER = getFlexDER()
Generates a flexoffer and returns its external ID
flexofferID = generateFlexoffer(flexoffer.getStartAfterTime(), flexoffer.getStartBeforeTime(), flexoffer.getSlices())
createFlexOffer(flexoffer)
sends the flexoffer to the Aggregator through XMPP
flexofferID
setID(flexofferID)
updateFOID(flexoffer, deviceID)
ScheduleAssignmentTimer requests to the Aggregator the schedules for the flexoffers
POST http request
createFlexOfferSchedule(flexOfferSchedule)
setFlexOfferSchedule(flexOfferSchedule)
setState(FlexOfferState.Assigned)
loop [while Flexoffer state == FlexOfferState.Initial]
state = getFlexofferState()
getState()
alt [if state == FlexOfferState.Initial]
ExecuteFOEmission waits until a schedule is assigned to the flexoffer
sleep(1000)
[if state == FlexOfferState.Assigned]
When a schedule is finally assigned to a flexoffer, ExecuteFOEmission saves the schedule in the database
schedule = getSchedule(flexoffer.getID())
insertSchedule(schedule, deviceID)

Fig. 61 – Sequence Diagram: UC14 Execute FO Emission

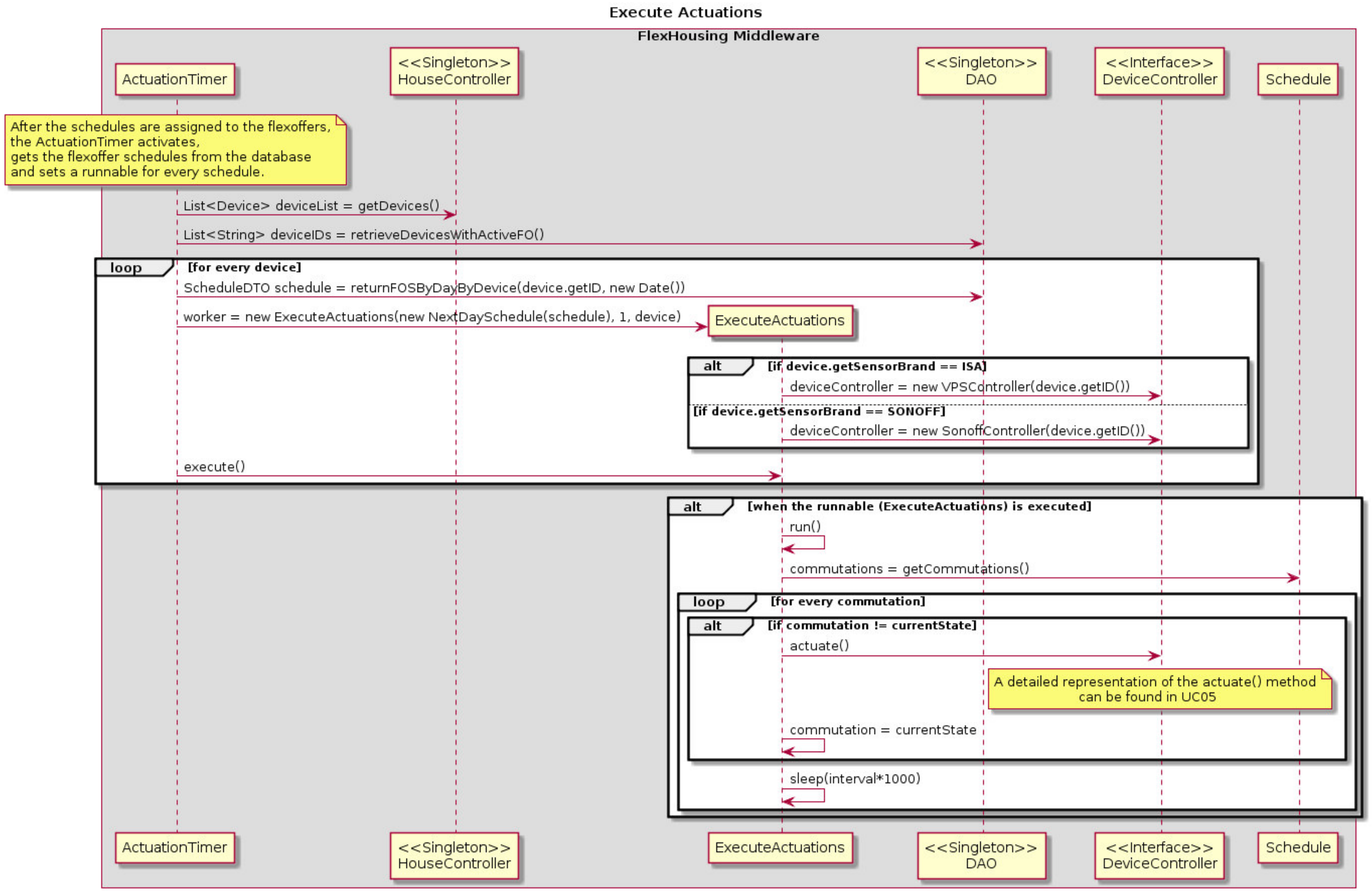

Execute Actuations
FlexHousing Middleware
ActuationTimer
<<Singleton>> HouseController
<<Singleton>> DAO
<<Interface>> DeviceController
Schedule
After the schedules are assigned to the flexoffers, the ActuationTimer activates, gets the flexoffer schedules from the database and sets a runnable for every schedule.
List<Device> deviceList = getDevices()
List<String> deviceIDs = retrieveDevicesWithActiveFO()
loop [for every device]
ScheduleDTO schedule = returnFOSByDayByDevice(device.getID, new Date())
worker = new ExecuteActuations(new NextDaySchedule(schedule), 1, device)
ExecuteActuations
alt [if device.getSensorBrand == ISA]
deviceController = new VPSController(device.getID())
[if device.getSensorBrand == SONOFF]
deviceController = new SonoffController(device.getID())
execute()
alt [when the runnable (ExecuteActuations) is executed]
run()
commutations = getCommutations()
loop [for every commutation]
alt [if commutation != currentState]
actuate()
A detailed representation of the actuate() method can be found in UC05
commutation = currentState
sleep(interval*1000)

Fig. 62 – Sequence Diagram: UC14 Execute Actuations

4.3.2.13 Use Case 15 – Identify Device Types

Table 19 – Use Case 15: Identify Device Types

Use Case 15 – Identify Device Types	
Description	The system intends to periodically identify the type of every registered device, based on their energy consumption values.
Actor(s)	FlexHousing Middleware
Preconditions	1. Have at least one device registered on the system. 2. Have enough energy consumption data to properly determine the device's type.
Postconditions	1. All devices will be identified with their correspondent type.
Basic Flow of Events	
Actions of the actor	1. Checks if it already has a trained model to predict the device's type. a. If not, then the actor creates and trains a model for the device type identification. 2. Identifies the device's type, through the trained model.

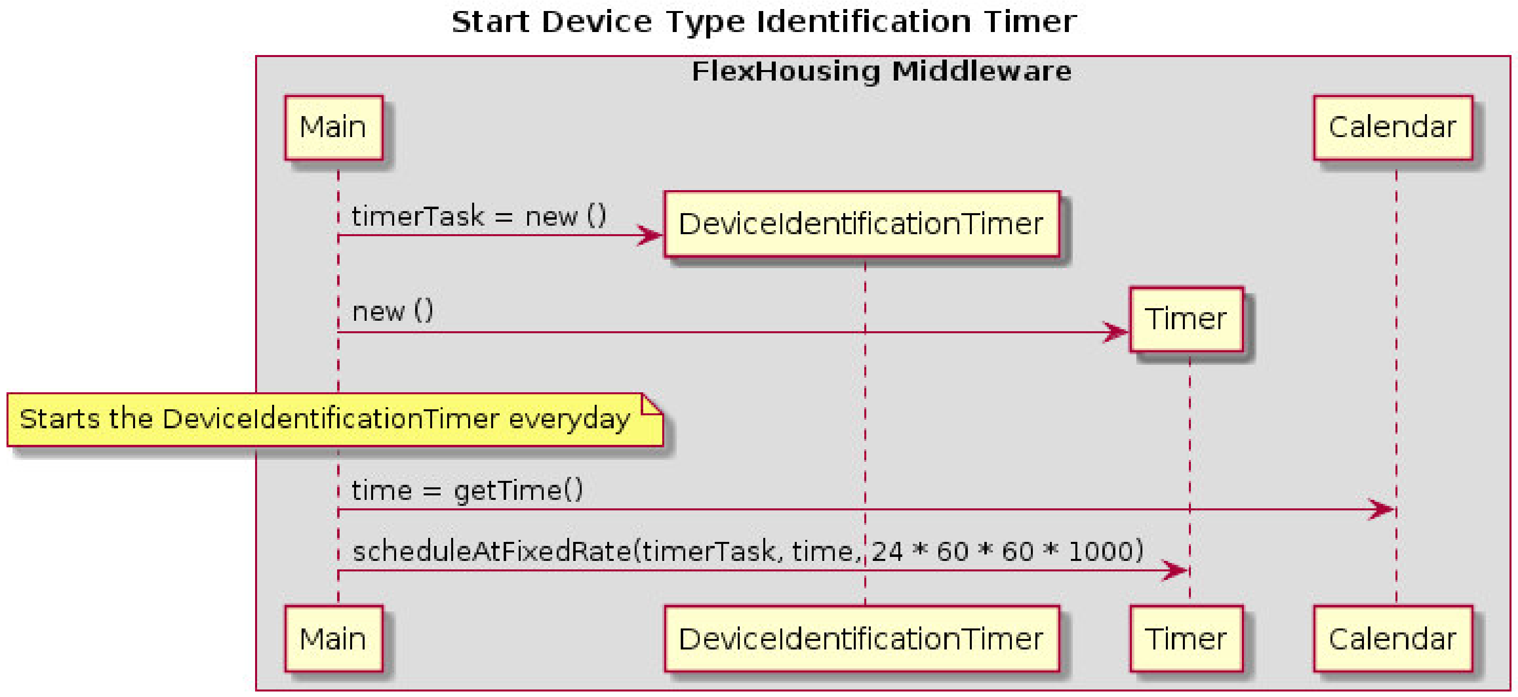

Start Device Type Identification Timer
FlexHousing Middleware
Main
Calendar
timerTask = new ()
DeviceIdentificationTimer
new ()
Timer
Starts the DeviceIdentificationTimer everyday
time = getTime()
scheduleAtFixedRate(timerTask, time, 24 * 60 * 60 * 1000)
Main
DeviceIdentificationTimer
Timer
Calendar

Fig. 63 – Sequence Diagram: UC15 Start Device Type Identification Timer

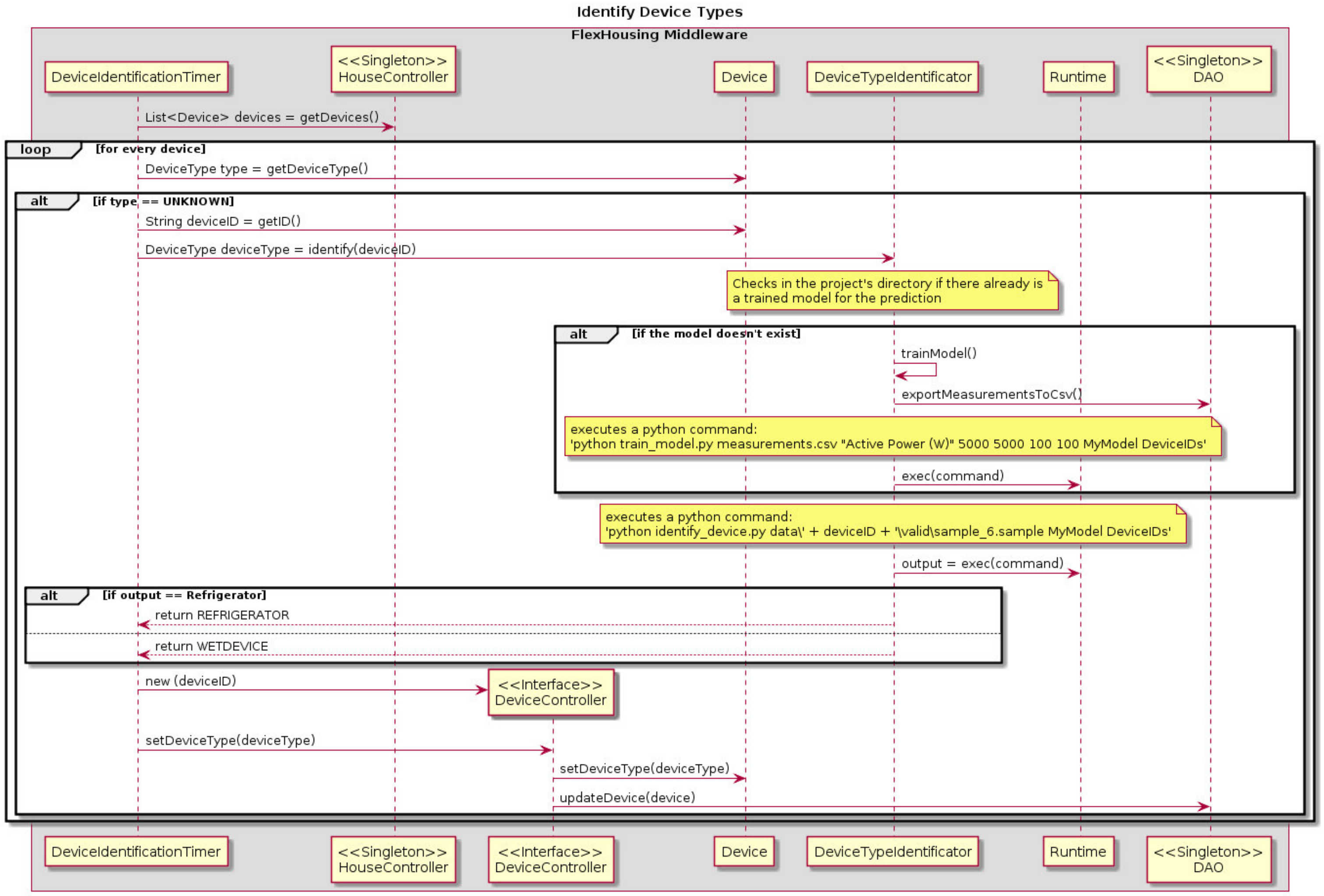

Identify Device Types
FlexHousing Middleware
DeviceIdentificationTimer
<<Singleton>> HouseController
Device
DeviceTypeIdentificator
Runtime
<<Singleton>> DAO
List<Device> devices = getDevices()
loop [for every device]
DeviceType type = getDeviceType()
alt [if type == UNKNOWN]
String deviceID = getID()
DeviceType deviceType = identify(deviceID)
Checks in the project's directory if there already is a trained model for the prediction
alt [if the model doesn't exist]
trainModel()
exportMeasurementsToCsv()
executes a python command: 'python train_model.py measurements.csv "Active Power (W)" 5000 5000 100 100 MyModel DeviceIDs'
exec(command)
executes a python command: 'python identify_device.py data\' + deviceID + '\valid\sample_6.sample MyModel DeviceIDs'
output = exec(command)
alt [if output == Refrigerator]
return REFRIGERATOR
return WETDEVICE
new (deviceID)
<<Interface>> DeviceController
setDeviceType(deviceType)
setDeviceType(deviceType)
updateDevice(device)

Fig. 64 – Sequence Diagram: UC15 Identify Device Types

4.3.2.14 Use Case 16 – Provide Device's Consumption Data

Table 20 - Use Case 16: Provide Device's Consumption Data

Use Case 16 – Provide Device's Consumption Data	
Description	The system intends to provide the current energy consumption values of a device via REST, updating its data every 2 seconds.
Actor(s)	Sonoff switch
Preconditions	----
Postconditions	2. The device's current energy consumption values are available via REST.
Basic Flow of Events	
Actions of the actor	3. Acquires energy consumption values from the device, every 2 seconds.
Actions of the system (Sensor)	1. Provides energy consumption values from the device, updating every 2 seconds.

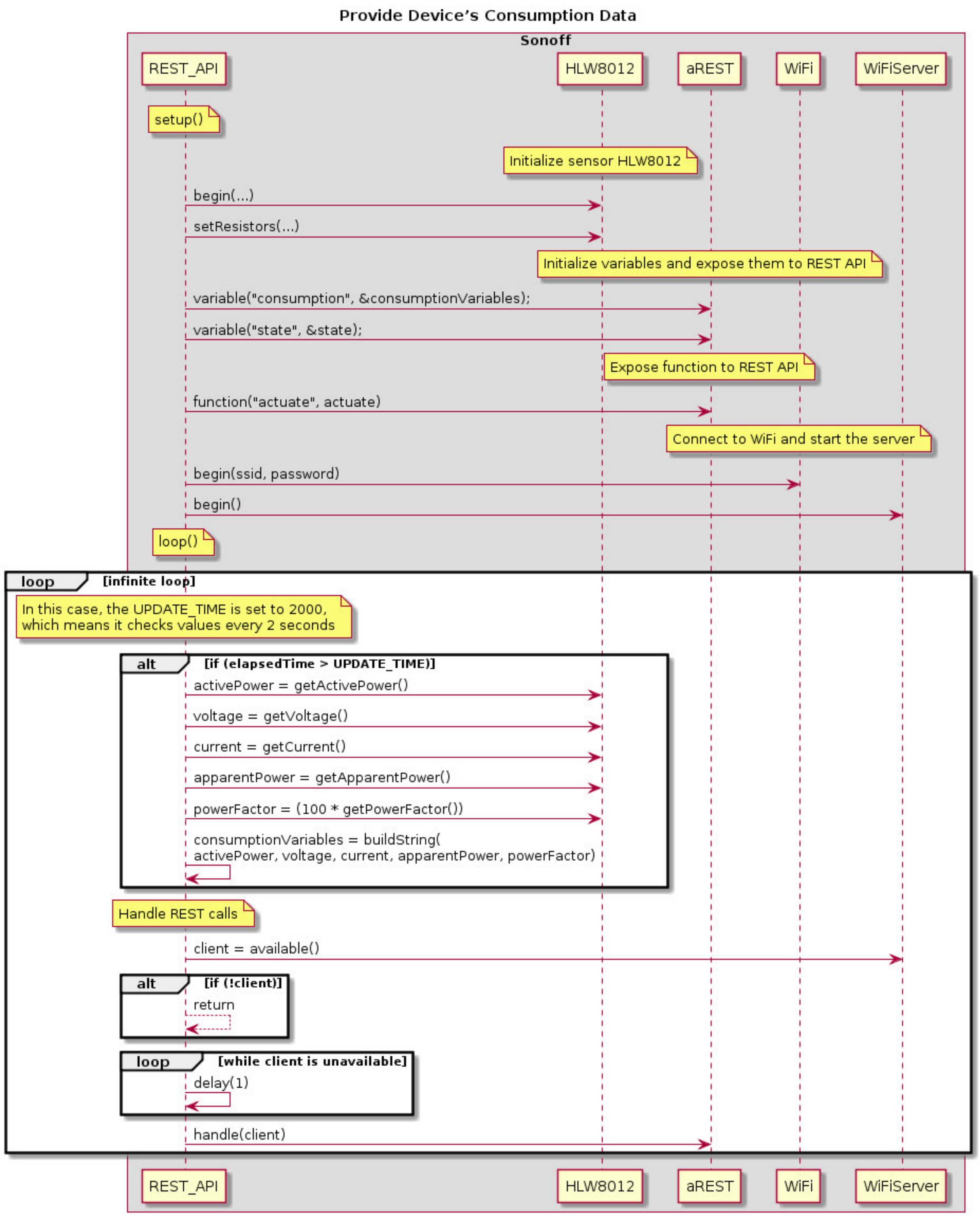

Provide Device's Consumption Data
Sonoff
REST_API
HLW8012
aREST
WiFi
WiFiServer
setup()
Initialize sensor HLW8012
begin(...)
setResistors(...)
Initialize variables and expose them to REST API
variable("consumption", &consumptionVariables);
variable("state", &state);
Expose function to REST API
function("actuate", actuate)
Connect to WiFi and start the server
begin(ssid, password)
begin()
loop()
loop [infinite loop]
In this case, the UPDATE_TIME is set to 2000, which means it checks values every 2 seconds
alt [if (elapsedTime > UPDATE_TIME)]
activePower = getActivePower()
voltage = getVoltage()
current = getCurrent()
apparentPower = getApparentPower()
powerFactor = (100 * getPowerFactor())
consumptionVariables = buildString(activePower, voltage, current, apparentPower, powerFactor)
Handle REST calls
client = available()
alt [if (!client)]
return
loop [while client is unavailable]
delay(1)
handle(client)

Fig. 65 – Sequence Diagram: UC16 Provide Device's Consumption Data

4.3.3 Class Diagram

A class diagram is a type of diagram that describes the structure of a system by showing the system's classes, their attributes, operations, and the relationships among objects [62].

In this section, a simplified class diagram of the FlexHousing Middleware will be presented (with the new classes and heavily altered ones highlighted in red) in Fig. 66, followed by a more detailed diagram and description of every package in the Middleware. Next, a class diagram of the FlexHousing web platform will also be displayed.

4.3.3.1 FlexHousing Middleware

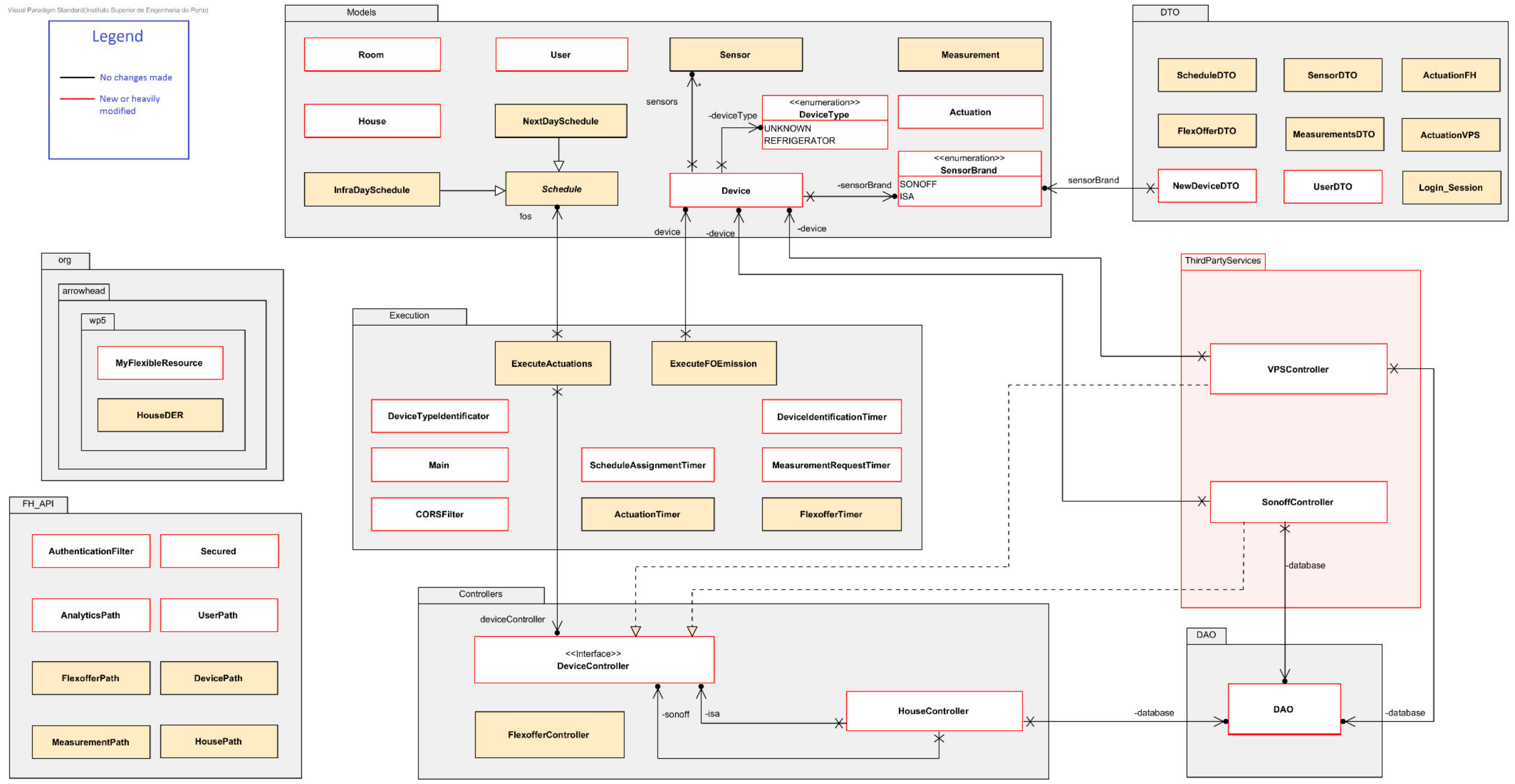


Fig. 66 – Simplified class diagram of the FlexHousing Middleware

4.3.3.1.1 Models

The Models package refers to the domain objects, in other words, the business layer containing all objects that model problem domain objects. These Model objects are data-centric classes that usually map roughly to the records of a corresponding database table and, thus, are often used as return values for Data Access Object (DAO) methods.

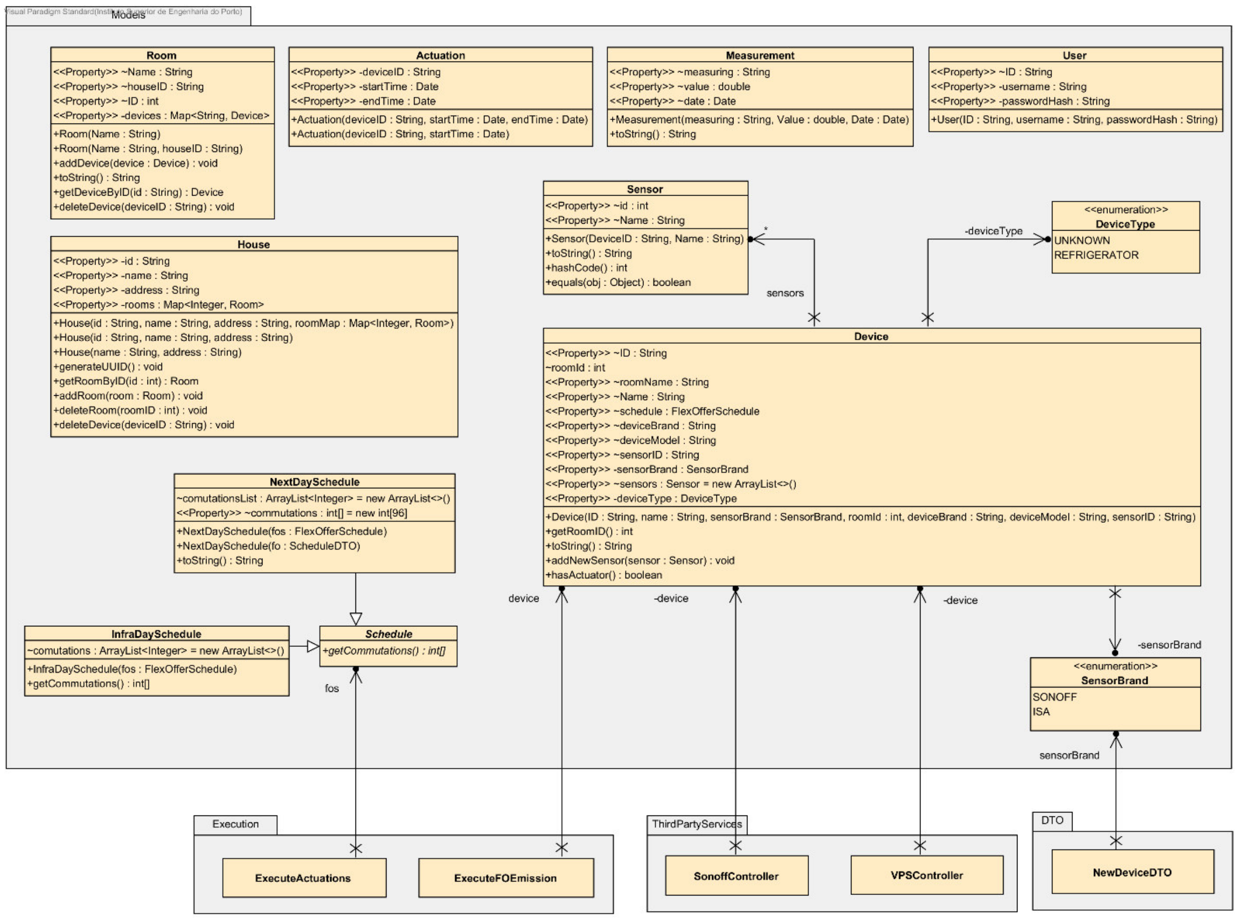

Models
Room
<<Property>> ~Name : String
<<Property>> ~houseID : String
<<Property>> ~ID : int
<<Property>> -devices : Map<String, Device>
+Room(Name : String)
+Room(Name : String, houseID : String)
+addDevice(device : Device) : void
+toString() : String
+getDeviceByID(id : String) : Device
+deleteDevice(deviceID : String) : void
Actuation
<<Property>> -deviceID : String
<<Property>> -startTime : Date
<<Property>> -endTime : Date
+Actuation(deviceID : String, startTime : Date, endTime : Date)
+Actuation(deviceID : String, startTime : Date)
Measurement
<<Property>> ~measuring : String
<<Property>> ~value : double
<<Property>> ~date : Date
+Measurement(measuring : String, Value : double, Date : Date)
+toString() : String
User
<<Property>> ~ID : String
<<Property>> -username : String
<<Property>> -passwordHash : String
+User(ID : String, username : String, passwordHash : String)
Sensor
<<Property>> ~id : int
<<Property>> ~Name : String
+Sensor(DeviceID : String, Name : String)
+toString() : String
+hashCode() : int
+equals(obj : Object) : boolean
<<enumeration>>
DeviceType
UNKNOWN
REFRIGERATOR
-deviceType
sensors
House
<<Property>> -id : String
<<Property>> -name : String
<<Property>> -address : String
<<Property>> -rooms : Map<Integer, Room>
+House(id : String, name : String, address : String, roomMap : Map<Integer, Room>)
+House(id : String, name : String, address : String)
+House(name : String, address : String)
+generateUUID() : void
+getRoomByID(id : int) : Room
+addRoom(room : Room) : void
+deleteRoom(roomID : int) : void
+deleteDevice(deviceID : String) : void
Device
<<Property>> ~ID : String
~roomId : int
<<Property>> ~roomName : String
<<Property>> ~Name : String
<<Property>> ~schedule : FlexOfferSchedule
<<Property>> ~deviceBrand : String
<<Property>> ~deviceModel : String
<<Property>> ~sensorID : String
<<Property>> -sensorBrand : SensorBrand
<<Property>> ~sensors : Sensor = new ArrayList<>()
<<Property>> -deviceType : DeviceType
+Device(ID : String, name : String, sensorBrand : SensorBrand, roomId : int, deviceBrand : String, deviceModel : String, sensorID : String)
+getRoomID() : int
+toString() : String
+addNewSensor(sensor : Sensor) : void
+hasActuator() : boolean
NextDaySchedule
~comutationsList : ArrayList<Integer> = new ArrayList<>()
<<Property>> ~commutations : int[] = new int[96]
+NextDaySchedule(fos : FlexOfferSchedule)
+NextDaySchedule(fo : ScheduleDTO)
+toString() : String
InfraDaySchedule
~comutations : ArrayList<Integer> = new ArrayList<>()
+InfraDaySchedule(fos : FlexOfferSchedule)
+getCommutations() : int[]
Schedule
+getCommutations() : int[]
fos
device
-device
-device
-sensorBrand
<<enumeration>>
SensorBrand
SONOFF
ISA
sensorBrand
Execution
ExecuteActuations
ExecuteFOEmission
ThirdPartyServices
SonoffController
VPSController
DTO
NewDeviceDTO

Fig. 67 – Class Diagram of Models package

Table 21 – Classes description of Models package

Class Name	***Description***
User	Represents the system user.
House	Represents a user's house.
Room	Represents a room of a user's house.
Device	Represents the monitored device, containing the device's details but also the sensors that are monitoring it.
Sensor	Representation of the sensors attached to a Device.
SensorBrand	Represents the brand of the sensors attached to the device. For the time being, these can only be ISA branded or Sonoff branded.
Schedule	Abstract class for the actuation schedule.
InfraDaySchedule	Implementation of the Schedule for same day flex-offers.
NextDaySchedule	Implementation of the Schedule for periodic flex-offers.
Measurement	Entity representing the data collected from the Sensors.
Actuation	Entity representing a device's actuation, registering an instance of use.
DeviceType	Represents the type of appliance the device is. For the time being, it can only be identified either as or not as a Refrigerator.

4.3.3.1.2 DTO

The DTO package, following the DTO pattern, acts as a layer between the domain objects and the API. It creates representational objects, originating from the ones in the Models package, but only containing relevant information for the operation it was requested for.

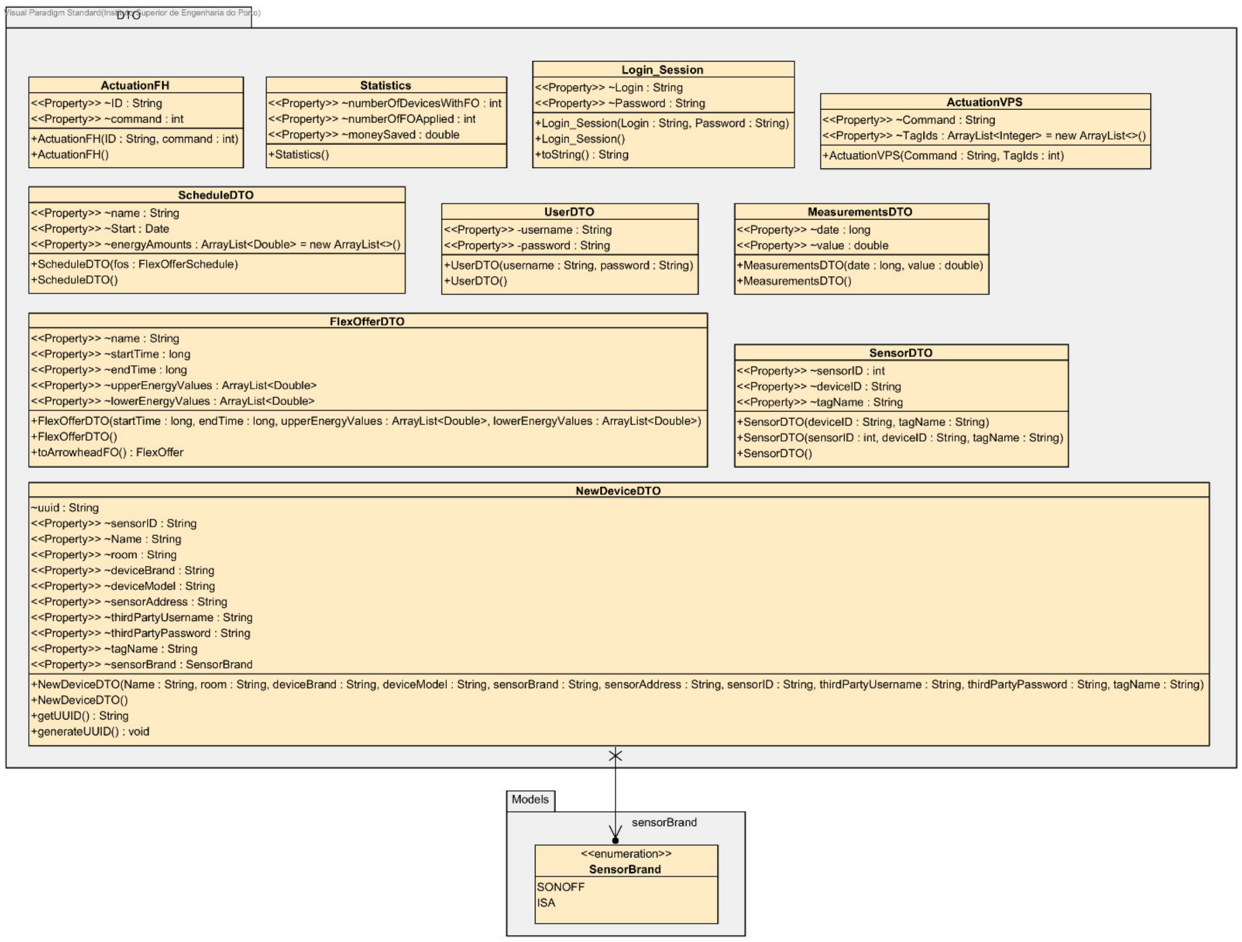

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
DTO
ActuationFH
<<Property>> ~ID : String
<<Property>> ~command : int
+ActuationFH(ID : String, command : int)
+ActuationFH()
Statistics
<<Property>> ~numberOfDevicesWithFO : int
<<Property>> ~numberOfFOApplied : int
<<Property>> ~moneySaved : double
+Statistics()
Login_Session
<<Property>> ~Login : String
<<Property>> ~Password : String
+Login_Session(Login : String, Password : String)
+Login_Session()
+toString() : String
ActuationVPS
<<Property>> ~Command : String
<<Property>> ~TagIds : ArrayList<Integer> = new ArrayList<>()
+ActuationVPS(Command : String, TagIds : int)
ScheduleDTO
<<Property>> ~name : String
<<Property>> ~Start : Date
<<Property>> ~energyAmounts : ArrayList<Double> = new ArrayList<>()
+ScheduleDTO(fos : FlexOfferSchedule)
+ScheduleDTO()
UserDTO
<<Property>> -username : String
<<Property>> -password : String
+UserDTO(username : String, password : String)
+UserDTO()
MeasurementsDTO
<<Property>> ~date : long
<<Property>> ~value : double
+MeasurementsDTO(date : long, value : double)
+MeasurementsDTO()
FlexOfferDTO
<<Property>> ~name : String
<<Property>> ~startTime : long
<<Property>> ~endTime : long
<<Property>> ~upperEnergyValues : ArrayList<Double>
<<Property>> ~lowerEnergyValues : ArrayList<Double>
+FlexOfferDTO(startTime : long, endTime : long, upperEnergyValues : ArrayList<Double>, lowerEnergyValues : ArrayList<Double>)
+FlexOfferDTO()
+toArrowheadFO() : FlexOffer
SensorDTO
<<Property>> ~sensorID : int
<<Property>> ~deviceID : String
<<Property>> ~tagName : String
+SensorDTO(deviceID : String, tagName : String)
+SensorDTO(sensorID : int, deviceID : String, tagName : String)
+SensorDTO()
NewDeviceDTO
~uuid : String
<<Property>> ~sensorID : String
<<Property>> ~Name : String
<<Property>> ~room : String
<<Property>> ~deviceBrand : String
<<Property>> ~deviceModel : String
<<Property>> ~sensorAddress : String
<<Property>> ~thirdPartyUsername : String
<<Property>> ~thirdPartyPassword : String
<<Property>> ~tagName : String
<<Property>> ~sensorBrand : SensorBrand
+NewDeviceDTO(Name : String, room : String, deviceBrand : String, deviceModel : String, sensorBrand : String, sensorAddress : String, sensorID : String, thirdPartyUsername : String, thirdPartyPassword : String, tagName : String)
+NewDeviceDTO()
+getUUID() : String
+generateUUID() : void
Models
sensorBrand
<<enumeration>>
SensorBrand
SONOFF
ISA

Fig. 68 – Class Diagram of DTO package

Table 22 – Classes description of DTO package

Class Name	***Description***
ActuationFH	Used to demand the actuation on a given device.
ActuationVPS	Object used by the VPSController for the actuations on devices.
Flex-offerDTO	Representation of the flex-offer only containing the fields configurable by the end user.
LoginSession	Object used by the VPSController to log into the VPS API.
MeasurementsDTO	Represents the relevant information gathered from the measurements received after a request for such on the VPSController and SonoffController.
NewDeviceDTO	Object containing the information of a new device to be added to the system.

ScheduleDTO	Contains the fields of the schedules that are relevant to the end user.
SensorDTO	Used when a request involving a sensor is received. Contains the sensor name and ID.
UserDTO	Object containing a user's credentials. Used in account registrations and logins.
Statistics	Object containing the statistical information of the system.

4.3.3.1.3 Execution

The Execution package contains all classes that execute the system's automatic processes, as well as the Main class that sets up the system itself (initializing all timer tasks and starting the HTTP server for the FlexHousing API).

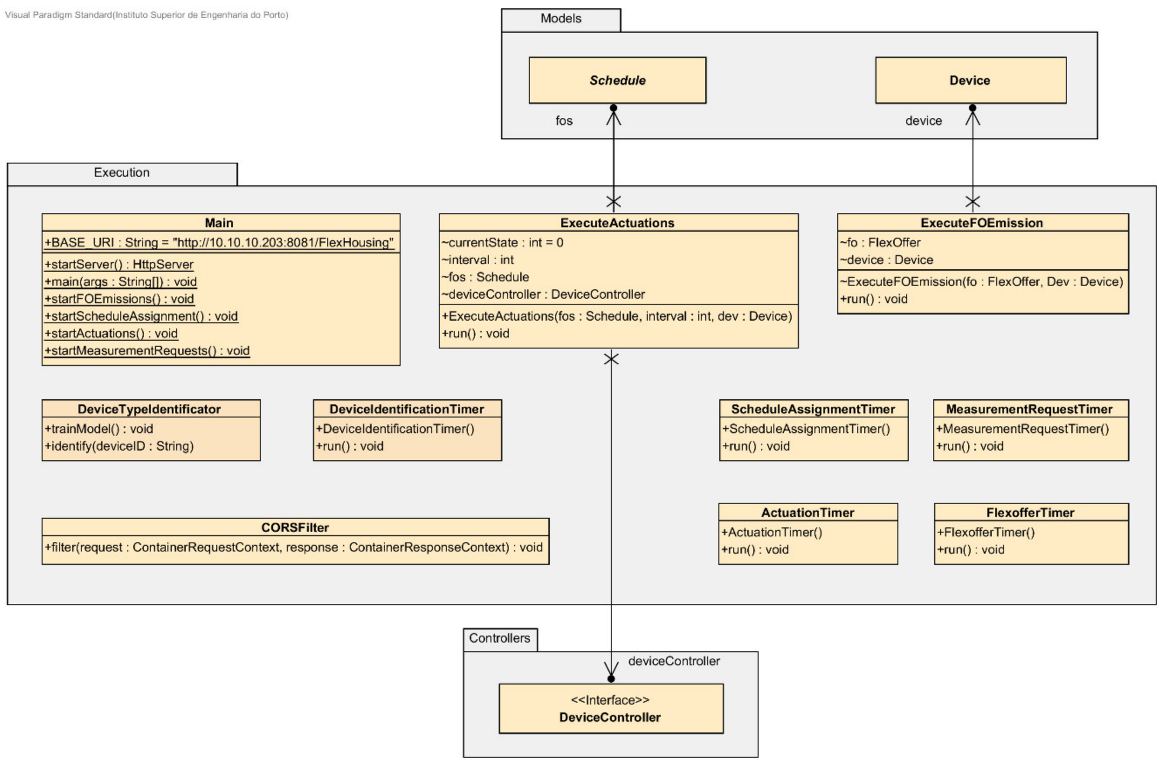

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
Models
Schedule
Device
fos
device
Execution
Main
+BASE_URI : String = "http://10.10.10.203:8081/FlexHousing"
+startServer() : HttpServer
+main(args : String[]) : void
+startFOEmissions() : void
+startScheduleAssignment() : void
+startActuations() : void
+startMeasurementRequests() : void
ExecuteActuations
~currentState : int = 0
~interval : int
~fos : Schedule
~deviceController : DeviceController
+ExecuteActuations(fos : Schedule, interval : int, dev : Device)
+run() : void
ExecuteFOEmission
~fo : FlexOffer
~device : Device
~ExecuteFOEmission(fo : FlexOffer, Dev : Device)
+run() : void
DeviceTypeIdentificator
+trainModel() : void
+identify(deviceID : String)
DeviceIdentificationTimer
+DeviceIdentificationTimer()
+run() : void
ScheduleAssignmentTimer
+ScheduleAssignmentTimer()
+run() : void
MeasurementRequestTimer
+MeasurementRequestTimer()
+run() : void
CORSFilter
+filter(request : ContainerRequestContext, response : ContainerResponseContext) : void
ActuationTimer
+ActuationTimer()
+run() : void
FlexofferTimer
+FlexofferTimer()
+run() : void
Controllers
deviceController
<<Interface>>
DeviceController

Fig. 69 – Class Diagram of Execution package

Table 23 – Classes description of Execution package

Class Name	***Description***
Main	Class executed when initiating the system. Starting point for every component.
FlexofferTimer	Class responsible for keeping track of the time of day for the emission of flex-offers.
ExecuteFOEmission	A runnable thread created for every flex-offer. Responsible for emitting the flex-offer, retrieve the schedules, and their respective persistence.
ActuationTimer	Class responsible for keeping track of the time of day for the creation of the actuations schedules.
ExecuteActuations	A runnable thread for the creation of the actuation schedules. Retrieves the schedule for the flex-offers and monitors the energy usage programed for the device.
ScheduleAssigmentTimer	Class responsible for requesting the Aggregator module a schedule assignment for the flex-offers sent.
MeasurementRequestTimer	Class responsible for requesting the current measurements of a device, for the cases where its service provider does not store measurements.
CORSFilter	Class responsible for dealing with Cross-Origin Resource Sharing (CORS).
DeviceTypeIdentificator	Class responsible for identifying the type of appliance a Device is.
DeviceIdentificatorTimer	Class responsible for executing DeviceTypeIdentificator every day.

4.3.3.1.4 org.arrowhead.wp5

The WP5 package, short for org.arrowhead.wp5, has Arrowhead's implementation of a Distributed Energy System (DER). WP5's responsibilities consist of emitting flex-offers, retrieving schedules, and handling the connection to the Flex-offer Services. The

MyFlexibleResource class is a singleton to make sure that the emission of the flex-offer originates for the same agent.

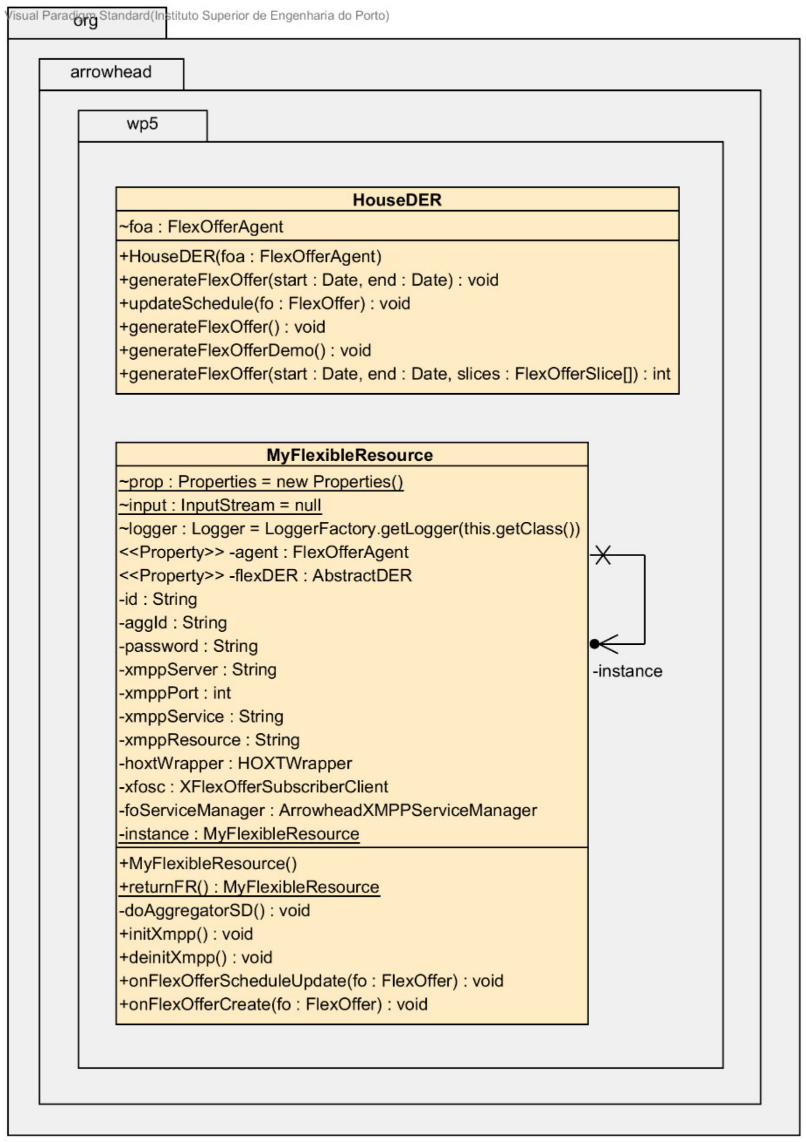

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
org
arrowhead
wp5
HouseDER
~foa : FlexOfferAgent
+HouseDER(foa : FlexOfferAgent)
+generateFlexOffer(start : Date, end : Date) : void
+updateSchedule(fo : FlexOffer) : void
+generateFlexOffer() : void
+generateFlexOfferDemo() : void
+generateFlexOffer(start : Date, end : Date, slices : FlexOfferSlice[]) : int
MyFlexibleResource
~prop : Properties = new Properties()
~input : InputStream = null
~logger : Logger = LoggerFactory.getLogger(this.getClass())
<<Property>> -agent : FlexOfferAgent
<<Property>> -flexDER : AbstractDER
-id : String
-aggId : String
-password : String
-xmppServer : String
-xmppPort : int
-xmppService : String
-xmppResource : String
-hoxtWrapper : HOXTWrapper
-xfosc : XFlexOfferSubscriberClient
-foServiceManager : ArrowheadXMPPServiceManager
-instance : MyFlexibleResource
+MyFlexibleResource()
+returnFR() : MyFlexibleResource
-doAggregatorSD() : void
+initXmpp() : void
+deinitXmpp() : void
+onFlexOfferScheduleUpdate(fo : FlexOffer) : void
+onFlexOfferCreate(fo : FlexOffer) : void
-instance

Fig. 70 – Class Diagram of org.arrowhead.wp5 package

Table 24 – Classes description of org.arrowhead.wp5 package

Class Name	***Description***
HouseDER	The HouseDER class extends the AbstractDER class. It contains an implementation of the generateFlexOffer() method, tailored to this system.

MyFlexibleResource	Entity responsible for the connection to the Flex-offer Services, containing methods for the XMPP connection and the Service Discovery for the Aggregator.

4.3.3.1.5 Controllers

The Controllers package includes the major controllers in the system. The controller pattern assigns the responsibility of dealing with system events to a class that represents the overall system or a use case scenario, in other words, a controller object is an object responsible for receiving or handling a system event. A controller should delegate the work that needs to be done to other objects; however, it should not do much work itself [63].

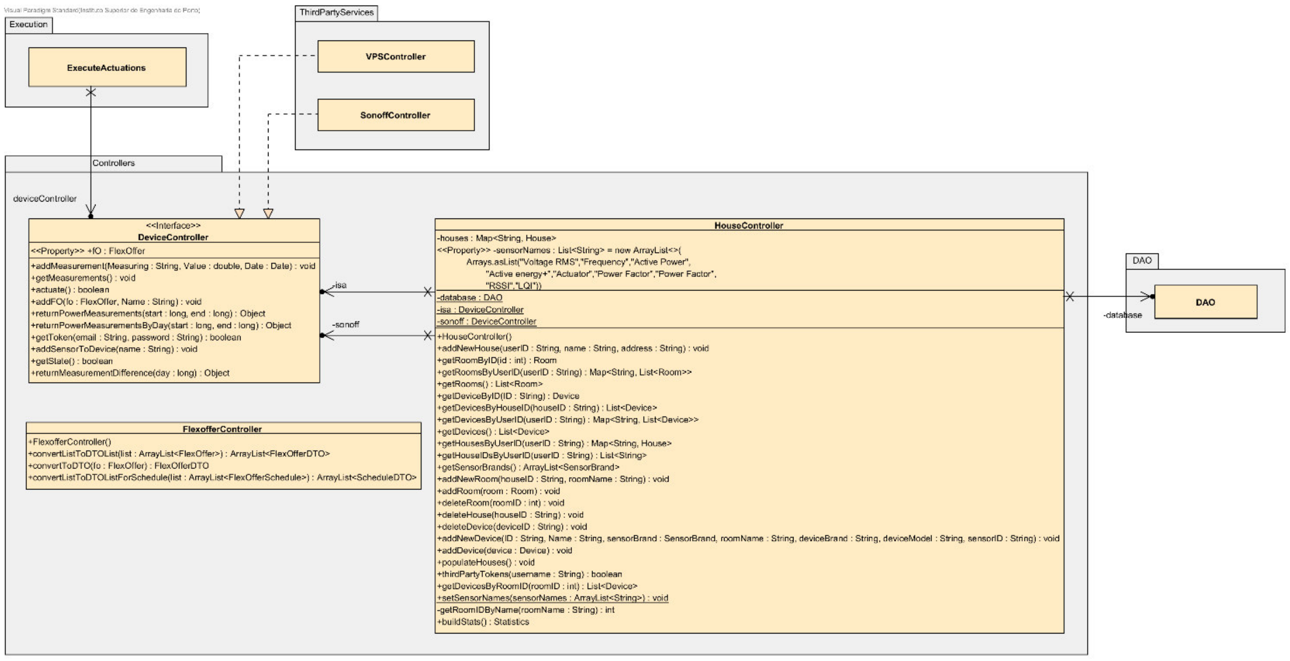

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
Execution
ExecuteActuations
ThirdPartyServices
VPSController
SonoffController
Controllers
deviceController
<<Interface>>
DeviceController
<<Property>> +fO : FlexOffer
+addMeasurement(Measuring : String, Value : double, Date : Date) : void
+getMeasurements() : void
+actuate() : boolean
+addFO(fo : FlexOffer, Name : String) : void
+returnPowerMeasurements(start : long, end : long) : Object
+returnPowerMeasurementsByDay(start : long, end : long) : Object
+getToken(email : String, password : String) : boolean
+addSensorToDevice(name : String) : void
+getState() : boolean
+returnMeasurementDifference(day : long) : Object
-isa
-sonoff
FlexofferController
+FlexofferController()
+convertListToDTOList(list : ArrayList<FlexOffer>) : ArrayList<FlexOfferDTO>
+convertToDTO(fo : FlexOffer) : FlexOfferDTO
+convertListToDTOListForSchedule(list : ArrayList<FlexOfferSchedule>) : ArrayList<ScheduleDTO>
HouseController
-houses : Map<String, House>
<<Property>> -sensorNames : List<String> = new ArrayList<>(
Arrays.asList("Voltage RMS","Frequency","Active Power",
"Active energy+","Actuator","Power Factor","Power Factor",
"RSSI","LQI"))
-database : DAO
-isa : DeviceController
-sonoff : DeviceController
+HouseController()
+addNewHouse(userID : String, name : String, address : String) : void
+getRoomByID(id : int) : Room
+getRoomsByUserID(userID : String) : Map<String, List<Room>>
+getRooms() : List<Room>
+getDeviceByID(ID : String) : Device
+getDevicesByHouseID(houseID : String) : List<Device>
+getDevicesByUserID(userID : String) : Map<String, List<Device>>
+getDevices() : List<Device>
+getHousesByUserID(userID : String) : Map<String, House>
+getHouseIDsByUserID(userID : String) : List<String>
+getSensorBrands() : ArrayList<SensorBrand>
+addNewRoom(houseID : String, roomName : String) : void
+addRoom(room : Room) : void
+deleteRoom(roomID : int) : void
+deleteHouse(houseID : String) : void
+deleteDevice(deviceID : String) : void
+addNewDevice(ID : String, Name : String, sensorBrand : SensorBrand, roomName : String, deviceBrand : String, deviceModel : String, sensorID : String) : void
+addDevice(device : Device) : void
+populateHouses() : void
+thirdPartyTokens(username : String) : boolean
+getDevicesByRoomID(roomID : int) : List<Device>
+setSensorNames(sensorNames : ArrayList<String>) : void
-getRoomIDByName(roomName : String) : int
+buildStats() : Statistics
DAO
DAO
-database

Fig. 71 – Class Diagram of Controllers package

Table 25 – Classes description of Controllers package

Class Name	***Description***
HouseController	Manages every house, and their corresponding room and device.

DeviceController	Responsible for every action on a Device object. As an interface, the DeviceController defines all the required methods to control a device.
FlexofferController	Allows flex-offer management. Has access to every flex-offer.

4.3.3.1.6 ThirdPartyServices

The ThirdPartyServices package contains the controllers corresponding to the service provider of each device. Every controller implements the DeviceController class, however these controllers have a specific implementation of each abstract method to their subsequent service. Essentially, this corresponds to the *Strategy* design pattern where each implementation is encapsulated in a separate (strategy) object.

In this case, there are two separate controllers for each implementation: one for VPS, and one for Sonoff.

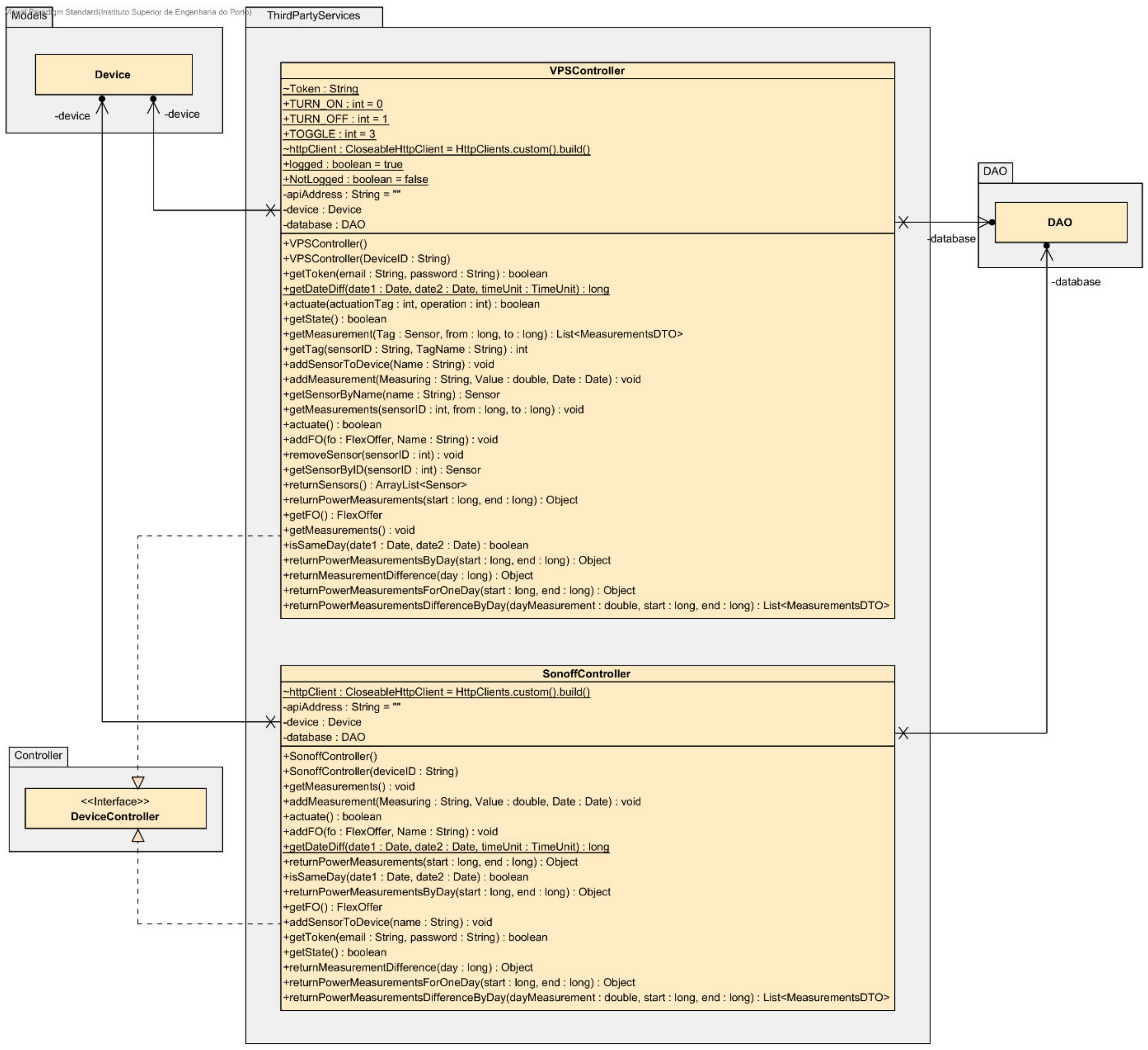

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
Models
Device
-device
-device
ThirdPartyServices
VPSController
~Token : String
+TURN_ON : int = 0
+TURN_OFF : int = 1
+TOGGLE : int = 3
~httpClient : CloseableHttpClient = HttpClients.custom().build()
+logged : boolean = true
+NotLogged : boolean = false
-apiAddress : String = ""
-device : Device
-database : DAO
+VPSController()
+VPSController(DeviceID : String)
+getToken(email : String, password : String) : boolean
+getDateDiff(date1 : Date, date2 : Date, timeUnit : TimeUnit) : long
+actuate(actuationTag : int, operation : int) : boolean
+getState() : boolean
+getMeasurement(Tag : Sensor, from : long, to : long) : List<MeasurementsDTO>
+getTag(sensorID : String, TagName : String) : int
+addSensorToDevice(Name : String) : void
+addMeasurement(Measuring : String, Value : double, Date : Date) : void
+getSensorByName(name : String) : Sensor
+getMeasurements(sensorID : int, from : long, to : long) : void
+actuate() : boolean
+addFO(fo : FlexOffer, Name : String) : void
+removeSensor(sensorID : int) : void
+getSensorByID(sensorID : int) : Sensor
+returnSensors() : ArrayList<Sensor>
+returnPowerMeasurements(start : long, end : long) : Object
+getFO() : FlexOffer
+getMeasurements() : void
+isSameDay(date1 : Date, date2 : Date) : boolean
+returnPowerMeasurementsByDay(start : long, end : long) : Object
+returnMeasurementDifference(day : long) : Object
+returnPowerMeasurementsForOneDay(start : long, end : long) : Object
+returnPowerMeasurementsDifferenceByDay(dayMeasurement : double, start : long, end : long) : List<MeasurementsDTO>
DAO
DAO
-database
-database
SonoffController
~httpClient : CloseableHttpClient = HttpClients.custom().build()
-apiAddress : String = ""
-device : Device
-database : DAO
+SonoffController()
+SonoffController(deviceID : String)
+getMeasurements() : void
+addMeasurement(Measuring : String, Value : double, Date : Date) : void
+actuate() : boolean
+addFO(fo : FlexOffer, Name : String) : void
+getDateDiff(date1 : Date, date2 : Date, timeUnit : TimeUnit) : long
+returnPowerMeasurements(start : long, end : long) : Object
+isSameDay(date1 : Date, date2 : Date) : boolean
+returnPowerMeasurementsByDay(start : long, end : long) : Object
+getFO() : FlexOffer
+addSensorToDevice(name : String) : void
+getToken(email : String, password : String) : boolean
+getState() : boolean
+returnMeasurementDifference(day : long) : Object
+returnPowerMeasurementsForOneDay(start : long, end : long) : Object
+returnPowerMeasurementsDifferenceByDay(dayMeasurement : double, start : long, end : long) : List<MeasurementsDTO>
Controller
<<Interface>>
DeviceController

Fig. 72 – Class Diagram of ThirdPartyServices package

Table 26 – Classes description of ThirdPartyServices package

Class Name	***Description***
VPSController	Handles every VPS related device. Builds HTTP requests aimed at the VPS API, and handles their responses.
SonoffController	Handles the Sonoff devices. Builds HTTP requests aimed at the Sonoff switch's API, and handles their responses.

4.3.3.1.7 FH_API

The FH_API package contains the definition of the services and resources hosted by the system's API through the HTTP server initialized in the Main class.

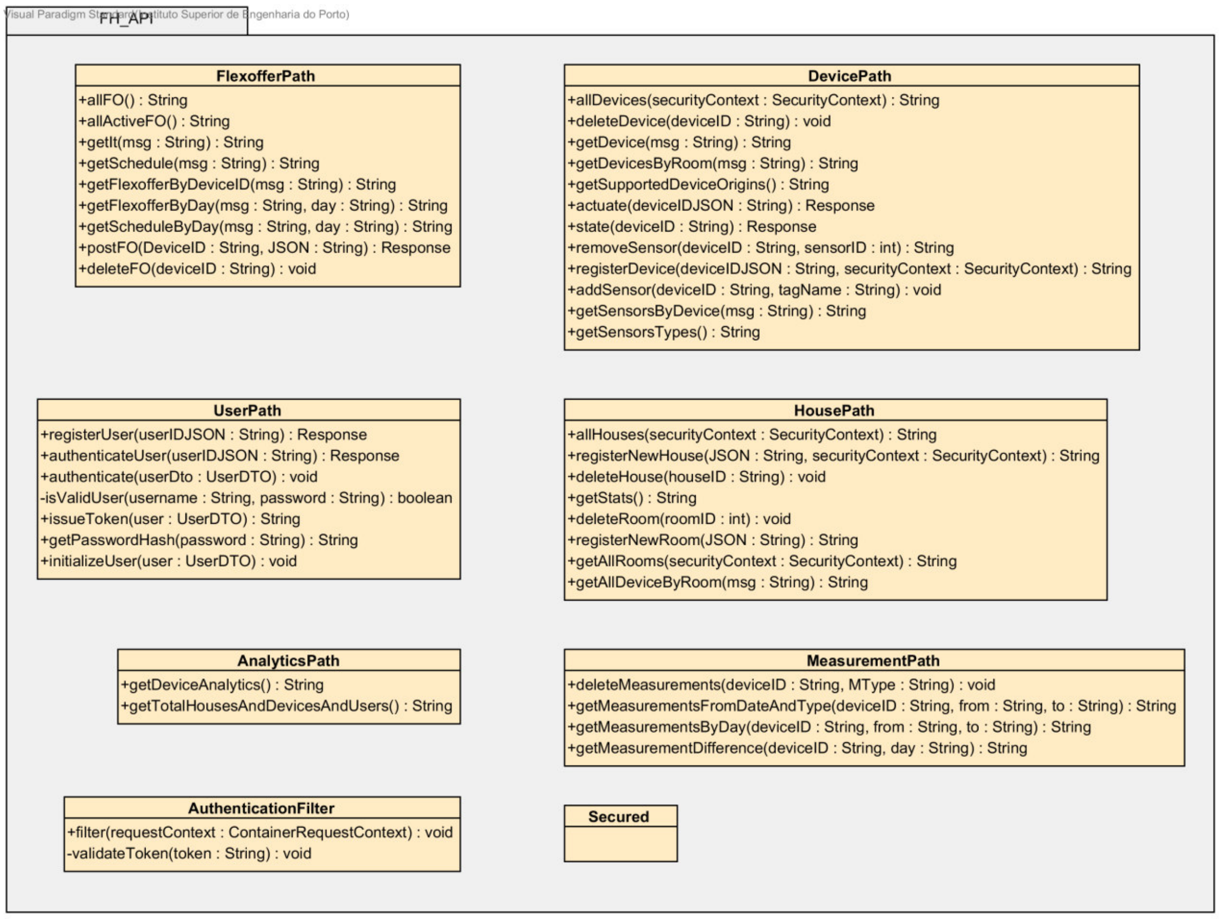

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
FH_API
FlexofferPath
+allFO() : String
+allActiveFO() : String
+getIt(msg : String) : String
+getSchedule(msg : String) : String
+getFlexofferByDeviceID(msg : String) : String
+getFlexofferByDay(msg : String, day : String) : String
+getScheduleByDay(msg : String, day : String) : String
+postFO(DeviceID : String, JSON : String) : Response
+deleteFO(deviceID : String) : void
DevicePath
+allDevices(securityContext : SecurityContext) : String
+deleteDevice(deviceID : String) : void
+getDevice(msg : String) : String
+getDevicesByRoom(msg : String) : String
+getSupportedDeviceOrigins() : String
+actuate(deviceIDJSON : String) : Response
+state(deviceID : String) : Response
+removeSensor(deviceID : String, sensorID : int) : String
+registerDevice(deviceIDJSON : String, securityContext : SecurityContext) : String
+addSensor(deviceID : String, tagName : String) : void
+getSensorsByDevice(msg : String) : String
+getSensorsTypes() : String
UserPath
+registerUser(userIDJSON : String) : Response
+authenticateUser(userIDJSON : String) : Response
+authenticate(userDto : UserDTO) : void
-isValidUser(username : String, password : String) : boolean
+issueToken(user : UserDTO) : String
+getPasswordHash(password : String) : String
+initializeUser(user : UserDTO) : void
HousePath
+allHouses(securityContext : SecurityContext) : String
+registerNewHouse(JSON : String, securityContext : SecurityContext) : String
+deleteHouse(houseID : String) : void
+getStats() : String
+deleteRoom(roomID : int) : void
+registerNewRoom(JSON : String) : String
+getAllRooms(securityContext : SecurityContext) : String
+getAllDeviceByRoom(msg : String) : String
AnalyticsPath
+getDeviceAnalytics() : String
+getTotalHousesAndDevicesAndUsers() : String
MeasurementPath
+deleteMeasurements(deviceID : String, MType : String) : void
+getMeasurementsFromDateAndType(deviceID : String, from : String, to : String) : String
+getMeasurementsByDay(deviceID : String, from : String, to : String) : String
+getMeasurementDifference(deviceID : String, day : String) : String
AuthenticationFilter
+filter(requestContext : ContainerRequestContext) : void
-validateToken(token : String) : void
Secured

Fig. 73 – Class Diagram of FH_API package

Table 27 – Classes description of FH_API package

Class Name	***Description***
FlexofferPath	Definition of the services and resources attached to the Flexoffer route.
DevicePath	Definition of the services and resources attached to the Device route.

UserPath	Definition of the services and resources attached to the User route.
HousePath	Definition of the services and resources attached to the House route.
AnalyticsPath	Definition of the services and resources attached to the Analytics route.
MeasurementPath	Definition of the services and resources attached to the Measurements route.
AuthenticationFilter	Implements the ContainerRequestFilter class, which allows it to handle a request. Verifies the access permissions for a user based on the username and password provided in the request.
Secured	Defines the name-binding annotation @Secured, used to decorate the AuthenticationFilter class, allowing the system to handle a request.

4.3.3.1.8 DAO

The DAO package acts as a layer between the database and the system, being responsible for interacting with the database through queries. The DAO is a singleton to insure concurrence and to establish only one connection to the database at any given moment.

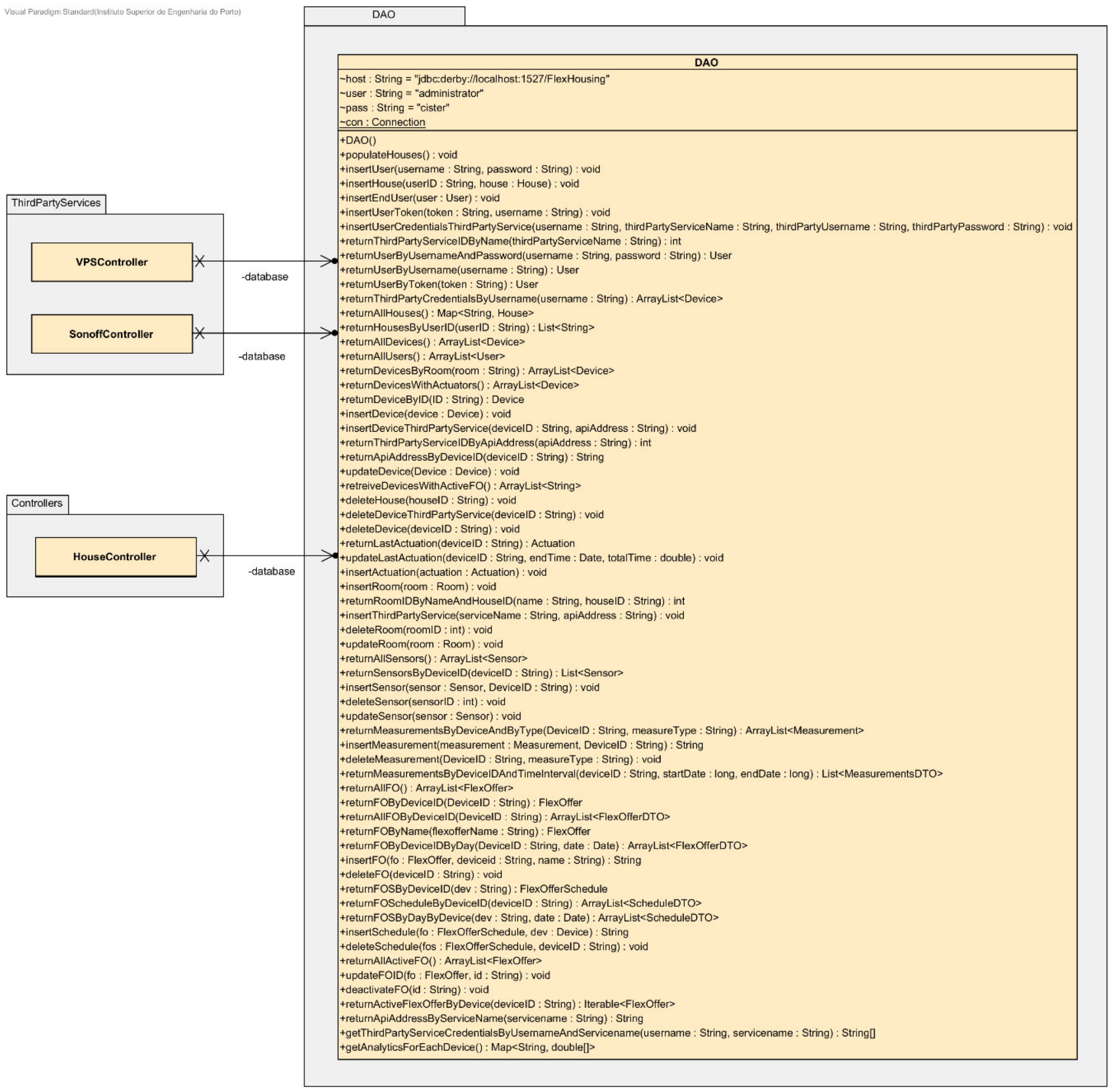

Visual Paradigm Standard(Instituto Superior de Engenharia do Porto)
DAO
DAO
~host : String = "jdbc:derby://localhost:1527/FlexHousing"
~user : String = "administrator"
~pass : String = "cister"
~con : Connection
+DAO()
+populateHouses() : void
+insertUser(username : String, password : String) : void
+insertHouse(userID : String, house : House) : void
+insertEndUser(user : User) : void
+insertUserToken(token : String, username : String) : void
+insertUserCredentialsThirdPartyService(username : String, thirdPartyServiceName : String, thirdPartyUsername : String, thirdPartyPassword : String) : void
+returnThirdPartyServiceIDByName(thirdPartyServiceName : String) : int
+returnUserByUsernameAndPassword(username : String, password : String) : User
+returnUserByUsername(username : String) : User
+returnUserByToken(token : String) : User
+returnThirdPartyCredentialsByUsername(username : String) : ArrayList<Device>
+returnAllHouses() : Map<String, House>
+returnHousesByUserID(userID : String) : List<String>
+returnAllDevices() : ArrayList<Device>
+returnAllUsers() : ArrayList<User>
+returnDevicesByRoom(room : String) : ArrayList<Device>
+returnDevicesWithActuators() : ArrayList<Device>
+returnDeviceByID(ID : String) : Device
+insertDevice(device : Device) : void
+insertDeviceThirdPartyService(deviceID : String, apiAddress : String) : void
+returnThirdPartyServiceIDByApiAddress(apiAddress : String) : int
+returnApiAddressByDeviceID(deviceID : String) : String
+updateDevice(Device : Device) : void
+retreiveDevicesWithActiveFO() : ArrayList<String>
+deleteHouse(houseID : String) : void
+deleteDeviceThirdPartyService(deviceID : String) : void
+deleteDevice(deviceID : String) : void
+returnLastActuation(deviceID : String) : Actuation
+updateLastActuation(deviceID : String, endTime : Date, totalTime : double) : void
+insertActuation(actuation : Actuation) : void
+insertRoom(room : Room) : void
+returnRoomIDByNameAndHouseID(name : String, houseID : String) : int
+insertThirdPartyService(serviceName : String, apiAddress : String) : void
+deleteRoom(roomID : int) : void
+updateRoom(room : Room) : void
+returnAllSensors() : ArrayList<Sensor>
+returnSensorsByDeviceID(deviceID : String) : List<Sensor>
+insertSensor(sensor : Sensor, DeviceID : String) : void
+deleteSensor(sensorID : int) : void
+updateSensor(sensor : Sensor) : void
+returnMeasurementsByDeviceAndByType(DeviceID : String, measureType : String) : ArrayList<Measurement>
+insertMeasurement(measurement : Measurement, DeviceID : String) : String
+deleteMeasurement(DeviceID : String, measureType : String) : void
+returnMeasurementsByDeviceIDAndTimeInterval(deviceID : String, startDate : long, endDate : long) : List<MeasurementsDTO>
+returnAllFO() : ArrayList<FlexOffer>
+returnFOByDeviceID(DeviceID : String) : FlexOffer
+returnAllFOByDeviceID(DeviceID : String) : ArrayList<FlexOfferDTO>
+returnFOByName(flexofferName : String) : FlexOffer
+returnFOByDeviceIDByDay(DeviceID : String, date : Date) : ArrayList<FlexOfferDTO>
+insertFO(fo : FlexOffer, deviceid : String, name : String) : String
+deleteFO(deviceID : String) : void
+returnFOSByDeviceID(dev : String) : FlexOfferSchedule
+returnFOScheduleByDeviceID(deviceID : String) : ArrayList<ScheduleDTO>
+returnFOSByDayByDevice(dev : String, date : Date) : ArrayList<ScheduleDTO>
+insertSchedule(fo : FlexOfferSchedule, dev : Device) : String
+deleteSchedule(fos : FlexOfferSchedule, deviceID : String) : void
+returnAllActiveFO() : ArrayList<FlexOffer>
+updateFOID(fo : FlexOffer, id : String) : void
+deactivateFO(id : String) : void
+returnActiveFlexOfferByDevice(deviceID : String) : Iterable<FlexOffer>
+returnApiAddressByServiceName(servicename : String) : String
+getThirdPartyServiceCredentialsByUsernameAndServicename(username : String, servicename : String) : String[]
+getAnalyticsForEachDevice() : Map<String, double[]>
ThirdPartyServices
VPSController
-database
SonoffController
-database
Controllers
HouseController
-database

Fig. 74 – Class Diagram of DAO package

Table 28 – Classes description of DAO package

Class Name	***Description***
DAO	Handles every database-related operation. Responsible for both storing and retrieving objects from the database.

4.3.3.2 FlexHousing Web Platform

Like the Middleware, the FlexHousing web platform was developed following the *single responsibility* principle, in order to avoid a class having more than one responsibility, unwanted coupling, resistance to change or incompatibilities when introducing code changes.

It should also be noted that inheritance was used for code reuse, and that this implementation did not cause any coupling problems.

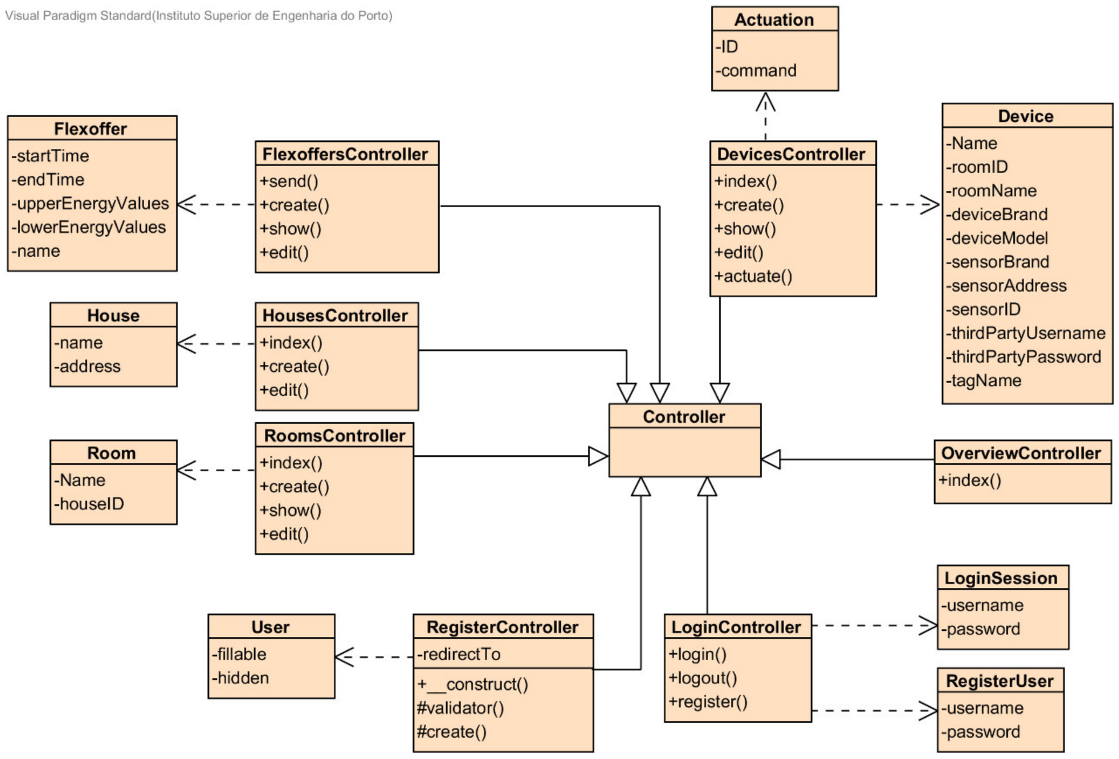


Fig. 75 – Class Diagram of FlexHousing Web Platform

The web platform's model classes are similar to the Middleware's in terms of their responsibilities and data structure. In regard to the controllers, most share the usual type of functions present in web apps: `index()`, `create()`, `show()`, and `edit()`.

4.3.3.3 Executives' Web Platform

The executives' platform is structured very similarly to the FlexHousing web platform, albeit much more simplified, since the only features required from it are accessing data from the FlexHousing Middleware. Thus, aside from the register and login pages, only the Overview page is needed.

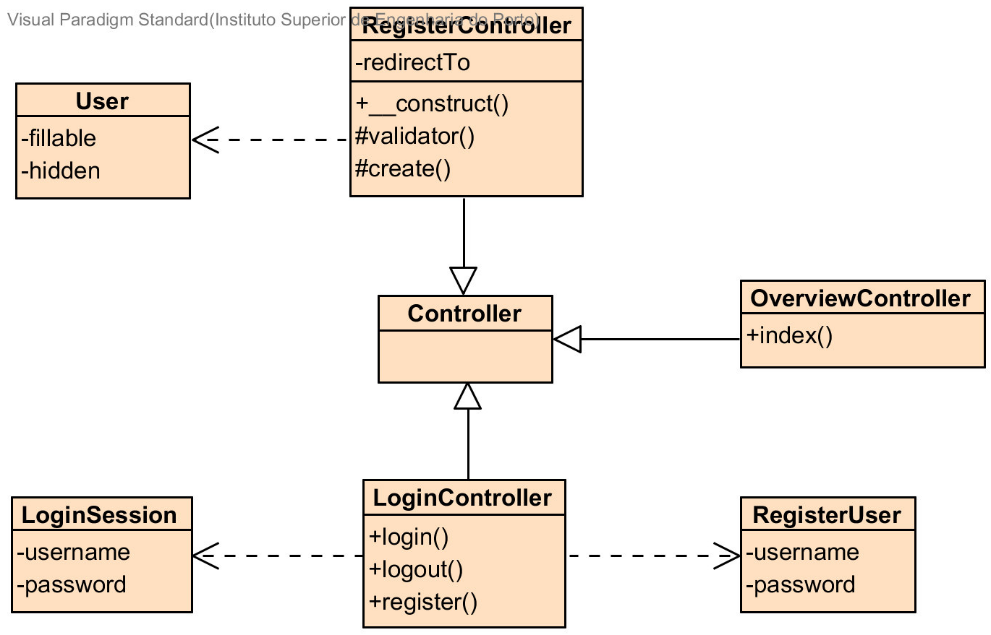


Fig. 76 – Class Diagram of Executives' Web Platfrom

4.4 Implementation

This section describes the implementation of the more interesting and/or complex features developed in the project, namely, the management of the two types of devices used in the project (VPS and Sonoff), the automatic creation of flex-offers, the identification of device types, the verification of the effectiveness of a flex-offer on a device's energy consumption pattern, and the setup of the FlexHousing system in a development environment.

4.4.1 Device Controller

As mentioned before, the previous version of the FlexHousing project was only developed to support VPS devices. This lead to its `DeviceController` being designed to exclusively operate with those devices' architecture.

To solve this problem, the *Strategy* design pattern was used. This meant that each implementation of a specific device (in this case, `VPSController` and `SonoffController`) must be encapsulated in a separate object, with the `DeviceController` serving as an interface for every device implementation.

Thus, the implementation of the `DeviceController` interface can be viewed in Fig. 77:

```
public interface DeviceController {

    public void addMeasurement(String Measuring, double Value, Date Date);

    public void getMeasurements();

    public boolean actuate();

    public void addFO(FlexOffer fo, String Name);

    public Object returnPowerMeasurements(long start, long end);

    public Object returnPowerMeasurementsByDay(long start, long end);

    public FlexOffer getFO();

    public boolean getToken(String email, String password);

    public void addSensorToDevice(String name);

    public boolean getState();

    public Object returnMeasurementDifference(long day);

    public Object returnPowerMeasurementsByQuarter(long start, long end);
}
```

Fig. 77 – Code Snippet: DeviceController Interface

4.4.2 VPS devices

Regarding the VPS devices, two different kinds were used in the project: a smart plug (Left part of Fig. 78) and a transmitter (Right part of Fig. 78). While the smart plug can read a device's consumed energy and turn on/off the device's power, the transmitter is only able to read a device's energy consumption.

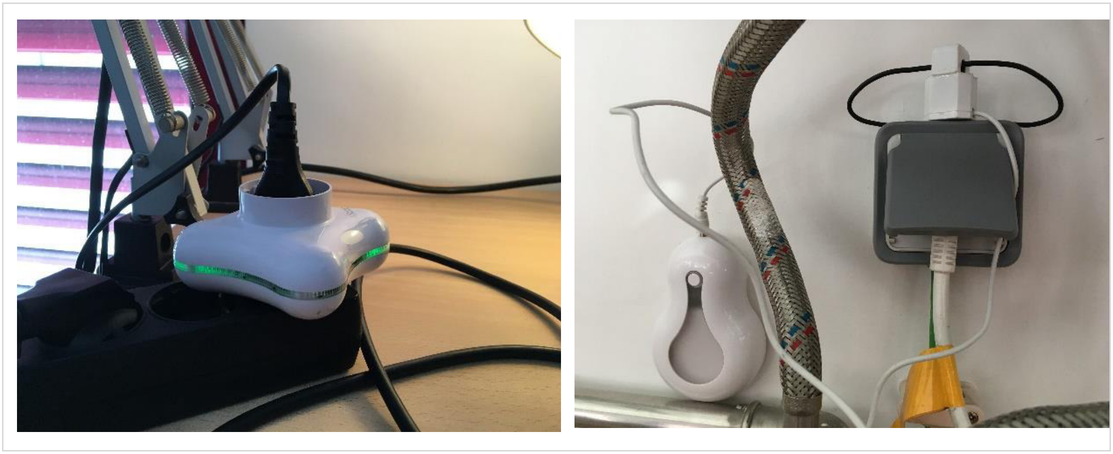


Fig. 78 – Left: VPS Smart Plug; Right: VPS Transmitter

These two devices send their data to a gateway named Cloogy (Fig. 79), which sends it to ISA's servers. Thus, for the FlexHousing platform to access these data, it must request it from ISA's web API.

Fig. 79 – VPS Cloogy

In the case of VPS devices, every sensor in it has an ID tag. For instance, the sensor "Actuator" (which turns the power on/off) has an ID, the same way the sensor "Active energy+" (which reads the energy consumption) has an ID. This means that, on ISA's web API, to request any kind of action from a device, the request must specify the ID tag of the device's sensor.

To demonstrate this, Fig. 80 displays a code snippet of `VPSController`'s implementation of the `returnPowerMeasurements` method:

```java
private static final String ADDRESS_MEASUREMENTS =
            "http://innov.isaenergy.pt:6600/api/1.4/consumptions/instant?from=";

private static final String SENSOR_MEASUREMENTS = "Active energy+";

@Override
public Object returnPowerMeasurements(long start, long end) {
    Sensor sensor = getSensorByName(SENSOR_MEASUREMENTS);
    return getMeasurement(sensor, start, end);
}

public List<MeasurementsDTO> getMeasurement(Sensor sensor, long from, long to){
        HttpGet getRequest = new HttpGet(
                ADDRESS_MEASUREMENTS + from + "&to=" + to + "&tags=[" + sensor.getId() +
"]");

        getRequest.addHeader(HttpHeaders.AUTHORIZATION, "ISA " + Token);
        HttpResponse response = null;
        try {
            response = httpClient.execute(getRequest);

        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
        Document doc = null;
        HttpEntity entity = response.getEntity();
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            doc = builder.parse(entity.getContent());

        } catch (ParserConfigurationException | IllegalStateException | SAXException e) {
            e.printStackTrace();
        } catch (IOException ex) {
            Logger.getLogger(VPSController.class.getName()).log(Level.SEVERE, null, ex);
        }
        Node list = doc.getFirstChild();
        List<MeasurementsDTO> measures = new ArrayList<>();
        NodeList measurements = list.getChildNodes();
        for (int i = 0; i < measurements.getLength(); i++) {
            long dateDTO = 0;
            double energyValueDTO = 0.0;
            NodeList nodes = (NodeList) measurements.item(i).getChildNodes();

            for (int temp = 0; temp < nodes.getLength(); temp++) {
                Node node = nodes.item(temp);
                Element eElement = (Element) node;

                if ("Date".equals(eElement.getNodeName())) {
                    dateDTO = Long.parseLong(eElement.getTextContent());
                }

                if ("Read".equals(eElement.getNodeName())){
                    energyValueDTO = Double.parseDouble(eElement.getTextContent());
                }
            }
            measures.add(new MeasurementsDTO(dateDTO, energyValueDTO));
        }

        return measures;
}
```

Fig. 80 – Code Snippet: returnPowerMeasurements in VPSController

4.4.3 Sonoff devices

In relation to the Sonoff devices (Fig. 81), these are cheap, generic, energy switches that, aside from switching the power on and off, and reading the current energy consumption, allow users to upload their own custom firmware on the switch's board. These boards are composed of a ESP8266 module (a low-cost Wi-Fi chip with full TCP/IP stack) to access the Wi-Fi network, and a HLW8012 current sensor to monitor the energy consumption.

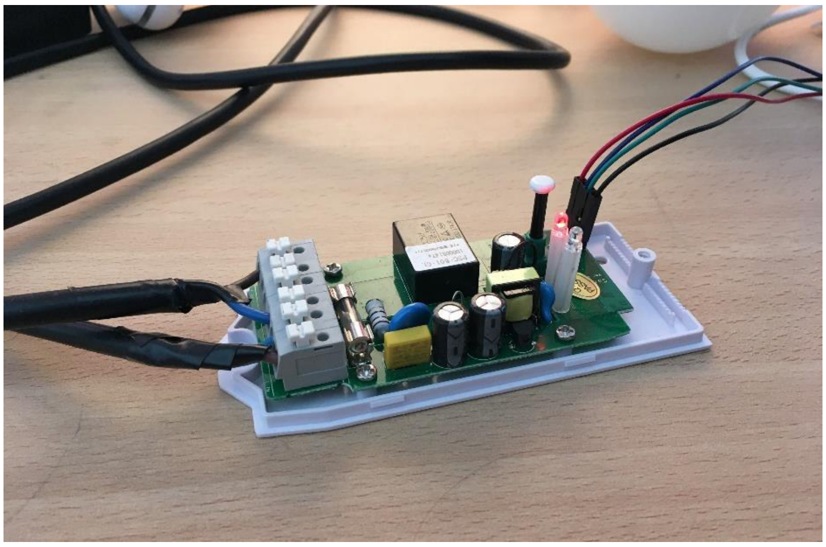


Fig. 81 – Sonoff Pow switch

The Sonoffs were primarily chosen for the project to help develop a direct connection from the FlexHousing platform to a different device in a local network, without having to request data to a third-party service. Given that the Sonoff switches' firmware can be entirely customized, this allows us to have full control of the sensor. Thus, a firmware for the Sonoffs was developed in Arduino (Appendix-B), to make the Sonoffs provide, through their REST API, their respective device's consumption values (updating every two seconds).

Since Sonoff devices don't send any data to external servers, the FlexHousing Middleware has to constantly request data from them (in which, every message has a length of around 128 bytes), every five seconds, and save it into its database. Whenever the FlexHousing web app requests data from the Middleware, the Middleware accesses its own database to deliver it.

To demonstrate this process, Fig. 82 displays a code snippet of `SonoffController`'s implementation of the `returnPowerMeasurements` method:

```
//  - SonoffController class -

@Override
public Object returnPowerMeasurements(long start, long end) {
    List<MeasurementsDTO> allMeasurements =
DAO.getInstance().returnMeasurementsByDeviceIDAndTimeInterval(this.device.getID(), start,
end);
    return allMeasurements;
}


// - DAO class -

public List<MeasurementsDTO> returnMeasurementsByDeviceIDAndTimeInterval
    (String deviceID, long startDate, long endDate) {
    List<MeasurementsDTO> allMeasurements = new ArrayList<>();
    String statement = "";
    PreparedStatement st = null;

    try {
        statement = "SELECT * FROM MEASUREMENTS "
                + "WHERE MEASUREMENTS.DEVICEID = ? "
                + "AND MEASUREMENTS.NAME = ? "
                + "AND MEASUREMENTS.TIME >= ? "
                + "AND MEASUREMENTS.TIME < ? "
                + "ORDER BY MEASUREMENTS.TIME";
        st = con.prepareStatement(statement);
        st.setString(1, deviceID);
        st.setString(2, "Energy (kWh)");
        st.setTimestamp(3, new java.sql.Timestamp(startDate));
        st.setTimestamp(4, new java.sql.Timestamp(endDate));
        ResultSet rs = st.executeQuery();
        while (rs.next()) {
            allMeasurements.add(
                    new MeasurementsDTO(rs.getTimestamp(4).getTime(),
                            rs.getDouble(3)));
        }
    } catch(SQLException err){
        System.out.println(err.getMessage());
    }

    return allMeasurements;
}
```

Fig. 82 – Code Snippet: returnPowerMeasurements in SonoffController

4.4.4 Automatic creation of a flex-offer's energy consumption pattern

For the automatic creation of a flex-offer, the system itself defines the device's energy consumption pattern based on its past consumption data. To do so, certain algorithms from the paper "Generation and Evaluation of Flex-Offers from Flexible Electrical Devices" [65] were used to implement a way to identify energy patterns. However, the algorithm for identifying an energy pattern depends on what kind of device it is.

4.4.4.1 Wet-devices

A device that has a certain daily routine (e.g. dishwashers, washing machines) is commonly known as wet-device. A wet device usually has a consistent activation hour. In this case, the system requests the user to input the hour when they typically turn it on.

With that predicted activation hour, the system goes through every consumption entry and tries to perform a Pattern Sequence Matching (PSM). The PSM is used to predict values for various attributes of FOs, e.g., the number of time slices, energy profile, etc.

First, all the changes in consumption values in the historical time series *X* are detected and are transformed into energy consumption patterns. Since a device activation causes a noticeable increase in power consumption, the PSM algorithm (Fig. 83) works under the premise that these patterns are correlated to the time of activation, e.g., a dishwasher activated at 20:00 always operates for two-time units and has an average energy profile of ⟨1.2,1⟩kWh [65].

Therefore, to estimate the energy profile for a predicted device activation at hour *h* of day *k*, the PSM first searches device activations triggered at hour *h* in the time series *X*. Then, for each activation the algorithm extracts the energy demand *et* for the duration of the device operation. This search outputs a set of indices of the device activation timestamps and profiles P = ⟨*p1*, ..., *pn*⟩, where each *pi* is an energy profile of the device activation at the timestamp *I* and *n* is the number of device activations at the hour *h* [65]. This algorithm returns an array of energy profiles for matching device activations.

```
                         Pattern Sequence Matching (PSM)

Input: = X – {e1, …, et} a time series.
         h – a predicted device activation hour.
Output: = P – a list of all demand patterns.
          I – a list of index for the patterns.
function demandPattern(X, h)
  P ← 0; p ← 0; active ← false
  for t ← 1 : T do
    if et >= thres then
      if t%24 = h then
        p ← p U {et};
        active ← true;
        I ← I U {t}
      else if active = true then
        p ← p U {et};
      endif
    else
      if active = true then
        P ← P U {p}
      endif
      p ← 0;
      active ← false
    endif
  endfor
  Return P, TimeDiffs
```

Fig. 83 – Algorithm: Pattern Sequence Matching (PSM) [66]

For further clarification of the PSM algorithm, Fig. 84 displays this process in a visual manner:

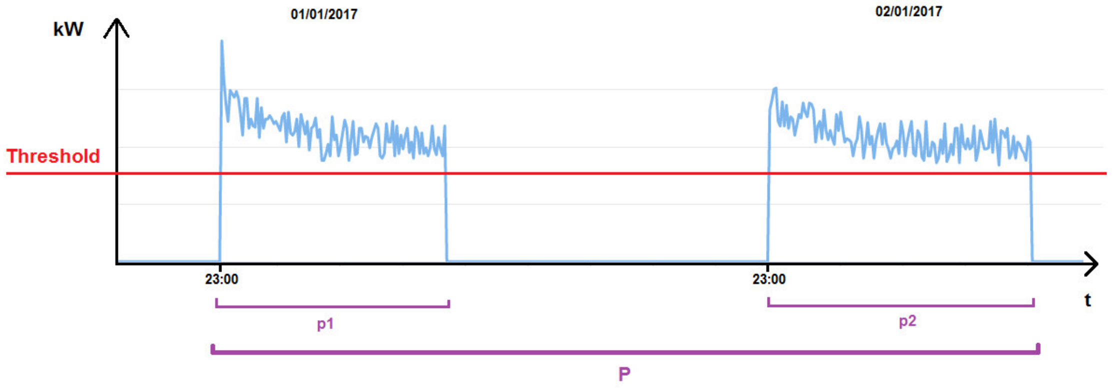


Fig. 84 – Visual sketch explaining the PSM algorithm

After collecting all energy profiles in *P*, for each respective time slice in every profile, the system calculates the average value and considers the result as the *emin* (minimum energy consumption) and *emax* (maximum energy consumption) for that time slice.

The algorithm for this estimation of the energy profile is displayed in Fig. 85:

```
Estimation of Energy Profile for a wet-device

Input: = P – the extracted demand patterns from PSM.
        d – an operation duration.
Output: = p – an energy profile for forecasted activation.
function estimateProfile-wet(P, d)
    p ← 0; n ← length of p
    for j ← 1 : d do
        e(min,j) ← $\frac{1}{n}\sum_{i=1}^{n} pi * e(min, j)$

        e(max,j) ← $\frac{1}{n}\sum_{i=1}^{n} pi * e(max, j)$

        sj ← [e(min,j), e(max,j)];
        p ← p U {sj}
    endfor
    Return p
```

Fig. 85 – Algorithm: Estimation of Energy Profile for a wet-device [67]

Finally, the result of this algorithm comes in the form of a model energy profile *p* that represents the usual consumption behavior of a device. However, this process only fits the situation of a wet-device, a device that only activates at a certain hour.

4.4.4.2 Refrigerators

If the device in question were to be a refrigerator, the wet-device process wouldn't work because the refrigerator is constantly activating and deactivating throughout the day, as seen in Fig. 86.

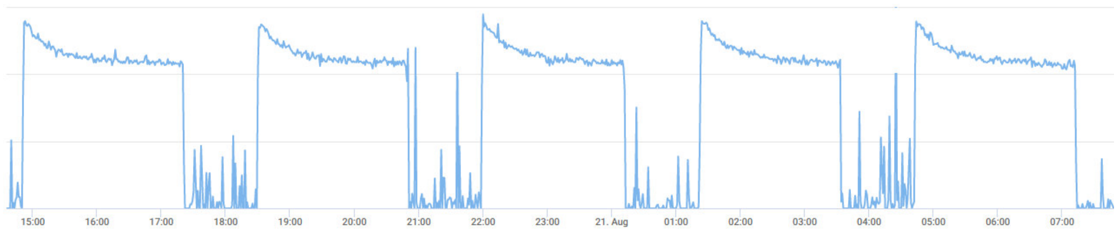


Fig. 86 – Energy consumption data from a refrigerator, measured by a Sonoff

So, for this case, the PSM algorithm was modified and new algorithms were developed to accommodate the situation.

First, looking at Fig. 87, there is a constant time segment of inactivity in between each activation of the refrigerator.

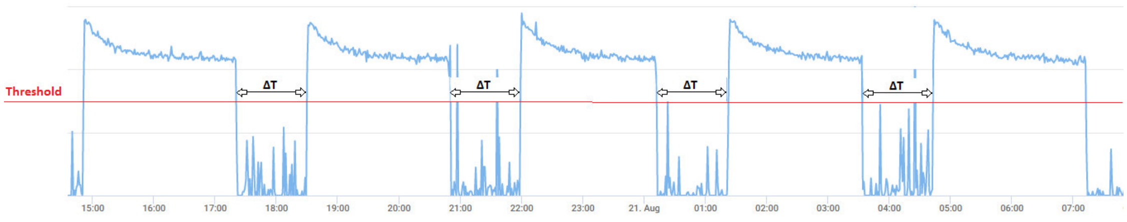


Fig. 87 – Constant time segment of inactivity in between each activation

If we were to calculate the average value of these time segments ($\overline{\Delta T}$) and use the PSM algorithm to determine the refrigerator's model energy profile (*p*), we can create an energy pattern for a full day, like so:

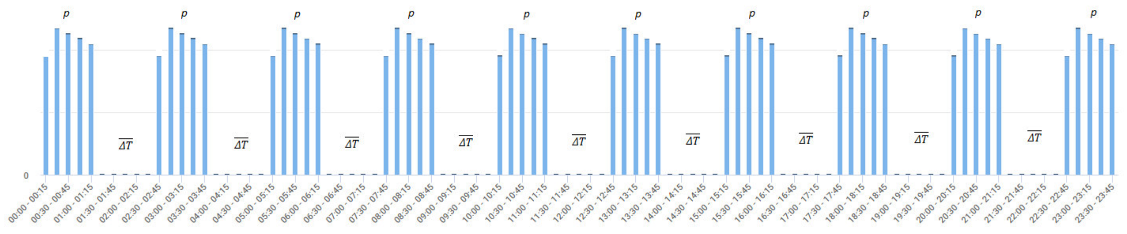


Fig. 88 – Energy consumption pattern of a refrigerator

Therefore, first we modify the PSM algorithm so we can get the time segments of inactivity:

```
                    Pattern Sequence Matching (PSM) for a refrigerator

**Input:** = X – {e1, …, et} a time series.
**Output:** = P – a list of all demand patterns.
              TimeDiffs – a list of inactive times (in seconds) in between patterns
**function** demandPattern(X)
   P ← 0; p ← 0; active ← false; measurementTimes ← 0
   **for** t ← 1 : T **do**
      **if** et >= thres **then**
         **if** active = false & measurementTimes length > 0 **then**
            index ← measurementTimes length - 1;
            diff ← (t - measurementTimes[index]) / 1000;
            timeDiffs ← timeDiff U {diff}
         **endif**
         measurementTimes ← measurementTmes U {t};
         p ← p U {et};
         active ← true
      **else**
         **if** active = true **then**
            P ← P U {p}
         **endif**
         p ← 0;
         active ← false
      **endif**
   **endfor**
   Return P, TimeDiffs
```

Fig. 89 – Algorithm: Pattern Sequence Matching (PSM) for a refrigerator

Then, we calculate the average value of the time segments to get the average time of inactivity, convert the time to minutes, and check how many 15-minutes slices it represents:

```
                    Calculate Time of Inactivity of a refrigerator

Input: = TimeDiffs – a list of inactive times (in seconds) in between patterns
Output: = timeSlices – number of slices the time of inactivity consists of.
function getTimeSlices(TimeDiffs)
    timeSlices ← 0; averageTimeBetweenPatterns ← 0; averageTimeInMinutes ← 0; sum ← 0
    for diff ← 1 : TimeDiffs do
        sum ← sum + diff
    endfor
    averageTimeBetweenPatterns ← sum / TimeDiffs length;
    averageTimeInMinutes ← averageTimeBetweenPatterns / 60;
    timeSlices ← averageTimeInMinutes / 15;
    Return timeSlices
```

Fig. 90 – Algorithm: Calculate Time of Inactivity of a refrigerator

Next, we use the same "Estimation of Energy Profile" algorithm to get the refrigerator's energy profile. Lastly, we can finally create the energy consumption pattern with the energy profile and the time slices:

```
                Create Energy Consumption Pattern of a refrigerator

Input: = p – an energy profile,
           timeSlices – number of slices the time of inactivity consists of.
Output: = Epattern – the energy consumption pattern of the refrigerator.
function createEnergyPattern(p, timeSlices)
    Epattern ← 0, index ← 0
    // a day is composed of 96 slices of 15 minutes
    while index <= 96 do
        for sj ← 1 : p do
            if index <= 96 do
                break
            endif
            Epattern ← Epattern U {sj[0]};
            index ← index + 1
        endfor
        for slice ← 1 : timeSlices do
            if index <= 96 do
                break
            endif
            Epattern ← Epattern U {0};
            index ← index + 1
        endfor
    endwhile
    Return Epattern
```

Fig. 91 – Algorithm: Create Energy Consumption Pattern of a refrigerator

4.4.5 Device Type Identification

Regarding the identification of device types, an interesting solution to this was to use data mining models to predict a device's type, by discovering patterns in its energy consumption values.

To do so, two Python scripts were developed by CISTER colleague Vincent Nelis: one (train_model.py) to create and train the data mining model; the other (identify_device.py) to identify the device's type, through the trained model. Essentially, this solution consists of a simple classifier based on a supervised machine-learning model. A more detailed explanation of the two scripts is as follows:

- train_model.py creates the model and trains it based on a labeled set of energy consumption traces (one trace for each device). That is, the script is fed with one consumption trace for each device (there is no constraints on the number of devices that the script must identify). Each trace is simply a sequence of tuples <timestamp, consumption> formatted in a 2-columns CSV file. In our experiments, the consumption of every device has been monitored by intervals of 5000 milliseconds, meaning that every trace contains at least 5000 data-points. The script starts by generating two datasets from the trace of each device: one for training and one for testing purposes. For each device, both datasets contain randomly picked fixed-length "slices" of the corresponding trace, i.e. every sample in both datasets is a cropped portion (selected randomly and of fixed length) of the consumption trace of the device. The number of samples in both the training and testing datasets, as well as the length of each sample, are user-defined parameters given as input to the script. After generating these two datasets from every input consumption trace, all the training sets are merged into a single labeled set (i.e. every sample is labeled with the name of the device it comes from) and that aggregated set is used to train a classifier and save its parameters into a file. We used the model from the pyAudioAnalysis Python library [87]. The trick to be able to use that simplified library was to treat every randomly generated sample of the training sets as a WAV sound (the library is designed to classify sounds).
- identify_device.py is much simpler that the first script. It loads the parameters of the model (computed in the previous step using train_model.py) and then, given a trace to be identified, it outputs the predicted name of the device. Once again, the model parameters are loaded and used automatically by the pyAudioAnalysis library that

outputs a prediction in a single line of python code. It turned out that this library that was initially designed to classify sounds work remarkably well for recognizing energy consumption traces as long as they are saved into WAV files.

With these two scripts, the FlexHousing Middleware must export the device's consumption data to a CSV file (in this case, "measurements.csv") and run the scripts with Python. The first one is run like so:

```
python train_model.py measurements.csv "Active Power (W)" 5000 5000 100 100 MyModel DeviceIDs
```

The parameters for train_model.py are, respectively:

- `raw_data_file` (measurements.csv): The file that contains the traces;
- The Measure of Interest ("Active Power (W)"): The name of the column that must be retrieved in the raw data file;
- `sample_size` (5000): Number of consecutive timestamps in each generated sample;
- `n_train_samples_per_device` (5000): Number of samples generated in the training set for each device;
- `n_test_samples_per_device` (100): Number of samples generated in the testing set for each device;
- `n_valid_samples_per_device` (100): Number of samples generated in the validation set for each device;
- `model_file` (MyModel): The file in which the model will be saved;
- `device_ids_file` (DeviceIDs): The file in which the device IDs are saved.

The script, running for about 15 minutes or more (depending on the machine running it), will create the following files:

- `MyModel.*`: A few files describing the model after training;
- `DeviceIDS`: The file in which the device IDs are saved;
- The folder data, with one subfolder for each device, and 3 subfolders for each: `train`, `test`, and `valid`:
    - The `train` folder contains the 5000 samples generated to train the model;

- The `test` folder holds the 100 samples generated to test the model. The tests are carried out at the end of the execution of the script;
- The `valid` folder comprises of some samples that are generated to test the next script “identify_device.py”.

Afterwards, to identify a specific device, the Middleware runs the second script like so:

```
python identify_device.py data\\{device ID}\\valid\\sample_6.sample MyModel DeviceIDs
```

The inputs are, respectively:

- The sample file to identify. This file is a trace that holds 5000 rows and 2 columns (timestamps and active power). Any sample in the `valid` folder is adequate for the identification;
- The model to load for the prediction;
- The list of devices.

The output consists of the model’s prediction of the device’s type, which the Middleware will then store it in its database.

4.4.6 Verifying a Flex-offer’s effect on a device’s consumption pattern

In relation to UC10, the system must display how effective a device’s active flex-offer was, relative to the device’s energy consumption pattern.

To do so, once the system has both the Flex-offer projected energy values and the device’s actual consumption values, it uses a mean percent error formula to measure the size of the inaccuracies between these two data sets in percentage terms.

At first, the Mean Absolute Percent Error (MAPE) [82; 83] was chosen as the solution to measure forecast errors. MAPE (Fig. 92) is the average absolute percent error for each time period or forecast minus actuals divided by actuals.

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|Forecast_t - Actual_t|}{Actual_t}$$

Fig. 92 – The MAPE formula

However, MAPE has been argued to be asymmetric, in which it puts a heavier penalty on forecasts that exceed the actual than those that are less than the actual [84]. Furthermore, with zeros or near-zeros, MAPE can give a distorted picture of error. The error on a near-zero item can be infinitely high, causing a distortion to the overall error rate when it is averaged in [85].

For forecasts of items that are near or at zero volume, Symmetric Mean Absolute Percent Error (SMAPE) is a better measure [86]. SMAPE (Fig. 93) is a modified MAPE in which the divisor is the sum of forecasts and actuals.

$$SMAPE = \frac{100}{n} \sum_{t=1}^{n} \frac{|Forecast_t - \ Actual_t|}{Forecast_t + Actual_t}$$

Fig. 93 – The SMAPE formula

Thus, the SMAPE formula was ultimately chosen for the calculation of the percentage of inaccuracy between the estimated pattern in the active flexoffer and the actual consumption pattern of the device.

4.4.7 FlexHousing System Setup

Regarding the setup of the FlexHousing system on a development environment, some specifics must be considered. Appendix-C presents a setup guide that goes into detail about these particularities.

4.5 Tests

This section describes the methods used to test the FlexHousing system (both Middleware and Web App), to ensure that all requirements are met and to guarantee accuracy and quality in the results presented by it.

For this project, three different levels of software testing were used: unit testing, integration testing, and acceptance testing.

4.5.1 Unit Tests

Unit testing concentrates on testing the internal processing logic of an application’s components. For this project, these tests were applied on the FlexHousing Middleware’s domain objects, which are the classes located in the Models package.

Fig. 94 depicts a code snippet of the Models unit tests, using Junit. This snippet shows some tests done to assess the relationships between the classes House, Room, and Device, and to check if the actuation schedule is created correctly when the flex-offer schedule is received.

```
public class ModelsTest {

    private static Device device;
    private static Room room;
    private static House house;
    private static ArrayList<Double> upper, lower;

    @Before
    public void setUp() {
        this.device = new Device("testID", "Test Device", SensorBrand.SONOFF,
                0, "Test Brand", "Test Model", "");
        this.room = new Room("Test Room");
        this.room.setID(0);
        this.house = new House("testID", "Test House", "Test Address");

        upper = new ArrayList<>();
        upper.addAll(Arrays.asList(
                100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0));

        lower = new ArrayList<>();
        lower.addAll(Arrays.asList(
                100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0,100.0,0.0));
    }

    /**
     * Assert that a Room adds a Device correctly.
     */
    @Test
    public void ensureRoomAddsDeviceCorrectly() {
        this.room.addDevice(this.device);
        Device otherDevice = this.room.getDeviceByID("testID");
        assertEquals(this.device, otherDevice);
    }

    /**
     * Assert that a House adds a Room correctly.
     */
    @Test
    public void ensureHouseAddsRoomCorrectly() {
        this.house.addRoom(this.room);
        Room otherRoom = this.house.getRoomByID(0);
        assertEquals(this.room, otherRoom);
    }

    /**
     * Assert that a Room deletes a Device correctly.
     */
    @Test
    public void ensureRoomDeletesDeviceCorrectly() {
        this.room.deleteDevice("testID");
        Device noDevice = this.room.getDeviceByID("testID");
        assertEquals(null, noDevice);

        // added the device again for the next test
        this.room.addDevice(this.device);
    }

    /**
```

```
     * Assert that a House deletes a Device correctly.
     */
    @Test
    public void ensureHouseDeletesDeviceCorrectly() {
        this.house.deleteDevice("testID");
        Device noDevice = this.room.getDeviceByID("testID");
        assertEquals(null, noDevice);
    }

    /**
     * Assert that a House deletes a Room correctly.
     */
    @Test
    public void ensureHouseDeletesRoomCorrectly() {
        this.house.deleteRoom(0);
        Room noRoom = this.house.getRoomByID(0);
        assertEquals(null, noRoom);
    }

    /**
     * Assert that NextDaySchedule registers commutations correctly.
     */
    @Test
    public void ensureNextDayScheduleRegistersCommutationsCorrectly() {
        Date date = new Date();
        Calendar cal = Calendar.getInstance();
        cal.setTime(date);

        cal.add(Calendar.HOUR_OF_DAY, 1);
        cal.set(Calendar.MINUTE, 0);
        Date start = cal.getTime();

        cal.add(Calendar.HOUR_OF_DAY, 4);
        Date end = cal.getTime();
        FlexOfferDTO fodto = new FlexOfferDTO(start.getTime(), end.getTime(),
                upper, lower);
        FlexOffer flexoffer = fodto.toArrowheadFO();
        FlexOfferSchedule flexofferSchedule = new FlexOfferSchedule(flexoffer);

        NextDaySchedule nextDay = new NextDaySchedule(flexofferSchedule);
        int[] commutations = nextDay.getCommutations();

        boolean turnsOnCorrectly = false;

        int turnOn = 0;

        for (int commutation : commutations) {
            if (commutation == 1) {
                turnOn++;
            }
        }

        if (turnOn == 6) {
            turnsOnCorrectly = true;
        }

        assertTrue(turnsOnCorrectly);
    }
}
```

Fig. 94 – Code Snippet: Models Unit Tests

4.5.2 Integration Tests

Upon conclusion of unit testing, the modules are to be integrated, which leads to integration testing. The purpose of integration testing is to verify the functionality and reliability between the integrated modules [68]. There are two types of integration testing: top-down integration and bottom-up integration.

In this case, top-down integration was used, allowing to quickly find errors and failures in high-level logic and data flow. Consequently, to simulate the behavior of lower-level modules, stubs must be used [69] [70]. Thus, to replicate the database, a mockup database was created through Apache Derby's in-memory database facility [71] (a feature specifically for testing and developing applications, in which the database resides completely in main memory, and not in the file system).

Fig. 95 shows a code snippet of the Middleware integration tests, using JUnit. This snippet shows a few tests done to assess the relationships between House, Room, and Device in the database.

```
public class MiddlewareIntegrationTest {

    private HouseController controller;
    private Device device, otherDevice;
    private Room room, otherRoom;
    private House house, otherHouse;
    private Map<String, House> housesMap;
    private User user;
    private DAO testDAO;

    @Before
    public void setUp() throws SQLException {
        this.testDAO = DAO.getInstance("test");

        Connection con = this.testDAO.getConnection();

        // Creating the necessary tables
        Statement sta = con.createStatement();
        sta.executeUpdate(
          "create table HOUSE\n" +
            "(\n" +
            "  ID VARCHAR(255) primary key not null,\n" +
            "  NAME VARCHAR(255) not null,\n" +
            "  ADDRESS VARCHAR(255) not null\n" +
            ")"
        );
        System.out.println("HOUSE Table created.");

        …

        sta.close();

        this.controller = new HouseController(this.testDAO);

        this.testDAO.insertUser("testUser", "testPassword");
        this.user = this.testDAO.returnUserByUsername("testUser");

        this.device = new Device("testID1", "Test Device 1", SensorBrand.SONOFF,
                0, "Test Brand1", "Test Model1", "");

    }

    /**
     * Assert a house, its room and devices are added and removed correctly.
     */
    @Test
    public void assertHouseAndRoomAndDeviceAreAddedAndRemovedCorrectly() {

        // Add house and room
        this.controller.addNewHouse(this.user.getID(), "New Test House",
                "Test Address");

        Map<String, House> houses = this.controller.
```

```
                getHousesByUserID(this.user.getID());

        boolean hasHouse = false;

        if (houses != null) {
            hasHouse = true;
        }

        String houseID = "";

        for (Map.Entry<String, House> entry : houses.entrySet()) {
            if (entry.getValue().getName().equals("New Test House")) {
                houseID = entry.getKey();
                break;
            }
        }

        this.controller.addNewRoom(houseID, "New Test Room");

        int roomID = -1;
        roomID = this.controller.
                getRoomIDByName("New Test Room");

        boolean hasRoom = false;

        if (roomID != -1) {
            hasRoom = true;
        }

        this.room = this.controller.getRoomByID(roomID);
        this.room.addDevice(this.device);

        List<Device> devices = this.controller.getDevices();

        boolean hasDevice = false;

        if (!devices.isEmpty()) {
            hasDevice = true;
        }

        // Delete house and room
        this.controller.deleteHouse(houseID);

        houses = null;
        houses = this.controller.
                getHousesByUserID(this.user.getID());

        boolean hasHouseBeenRemoved = false;

        if (houses != null) {
            hasHouseBeenRemoved = true;
        }

        roomID = -1;
        roomID = this.controller.
                getRoomIDByName("New Test Room");

        boolean hasRoomBeenRemoved = false;
        if (roomID == -1) {
            hasRoomBeenRemoved = true;
        }

        devices = this.controller.getDevices();

        boolean hasDeviceBeenRemoved = false;

        if (devices.isEmpty()) {
            hasDeviceBeenRemoved = true;
        }

        boolean hasFunctionedCorrectly = hasHouse & hasRoom & hasDevice &
                hasHouseBeenRemoved & hasRoomBeenRemoved & hasDeviceBeenRemoved;

        assertTrue(hasFunctionedCorrectly);
        }
    }
```

Fig. 95 – Code Snippet: Middleware Integration Tests

4.5.3 Acceptance Tests

Acceptance testing consists of a testing technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users [72]. Therefore, these tests focus on visible actions with user inputs and system outputs.

In this case, the required features translate into use cases. Thus, for every use case, an acceptance test was developed. These tests were performed on the FlexHousing web application, using Laravel Dusk (a browser automation and testing API, based on the open source tools ChromeDriver and Facebook Php-webdriver [73]).

Table 29 – Acceptance Test: UC01 Register User

Feature:	1 *Register User*		
Scenario	**Test**	**Expected result**	**Validation**
The unregistered End-User registers into the system.	Fill out the user creation form and submit.	End-User can now login in FlexHousing.	Success

```
public function testRegisterSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/')
                ->assertSee('FlexHousing')
                ->clickLink('Register')
                ->type('#name','cister')
                ->type('#password','123456789')
                ->type('#password-confirm','123456789')
                ->press('Register')
                ->waitForText('FlexHousing')
                ->assertSee('FlexHousing');
    });
}

public function testLoginSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/')
                ->assertSee('FlexHousing')
                ->type('#username','cister')
                ->type('#password','123456789')
                ->press('Login')
                ->waitForText('Dashboard')
                ->assertSee('Dashboard');
    });
}
```

Fig. 96 – Code Snippet: Acceptance Test of UC01 Register User

Table 30 – Acceptance Test: UC02 CRUD House

Feature:	2 *CRUD House*		
Scenario	**Test**	**Expected result**	**Validation**
The End-User registers a house.	Fill out the house creation form and submit.	The house is now registered in the system.	Success
The End-User edits a house.	Change values in the fields of the house form and submit.	The house details are updated.	Success
The End-User deletes a house.	Click the “Remove House” button of the respective house.	The house, its rooms, and devices are removed from the system.	Success

```
public function testAddHouseSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/houses/create')
                ->waitForText('Add New House')
                ->type('#houseName','Test House')
                ->type('#houseAddress','Test Address')
                ->click('#register')
                ->waitForText('Houses')
                ->assertSee('Test House');
    });
}

public function testRemoveHouseSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/houses')
                ->waitForText('Test House')
                ->click('[title="Delete Test House"]');

        $browser->driver->switchTo()->alert()->accept();
    });
}
```

Fig. 97 – Code Snippet: Acceptance Test of UC02 CRUD House

Table 31 – Acceptance Test: UC03 CRUD Room

Feature:	3 *CRUD Room*		
Scenario	**Test**	**Expected result**	**Validation**
The End-User registers a room.	Fill out the room creation form and submit.	The room is now registered in the system.	Success
The End-User edits a room.	Change values in the fields of the room form and submit.	The room details are updated.	Success
The End-User deletes a room.	Click the "Remove Room" button of the respective room.	The room and its devices are removed from the system.	Success

```
public function testAddRoomSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/rooms/create')
                ->waitForText('Add New Room')
                ->type('#roomName','Test Room')
                ->script("document.getElementById('houseList').value = 'Test House';");

        $browser->click('#register')
                ->waitForText('Rooms')
                ->assertSee('Test Room');
    });
}

public function testRemoveRoomSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/rooms')
                ->waitForText('Test House')
                ->click('[title="Delete Test Room"]');

        $browser->driver->switchTo()->alert()->accept();
    });
}
```

Fig. 98 – Code Snippet: Acceptance Test of UC03 CRUD Room

Table 32 – Acceptance Test: UC04 CRUD Device

Feature:	4 *CRUD Device*		
Scenario	**Test**	**Expected result**	**Validation**
The End-User registers a device.	Fill out the device creation form and submit.	The device is now registered in the system.	Success
The End-User edits a device.	Change values in the fields of the device form and submit.	The device details are updated.	Success
The End-User deletes a device.	Click the “Remove Device” button of the respective device.	The device is removed from the system.	Success

```
public function testAddDeviceSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/devices/create')
                ->waitForText('Add New Device')
                ->type('#deviceName','Test Device')
                ->script("document.getElementById('houseList').value = 'Test House';");

        $browser->script("document.getElementById('Test House').style.display = 'true';");

        $browser->script("document.getElementById('Test House rooms').value = 'Test
Room';");

        $browser->type('#deviceBrand','Brand Test')
                ->type('#deviceModel','Test Model')
                ->script("document.getElementById('sensorBrandList').value = 'SONOFF';");

        $browser->type('#sensorAddress','http://172.16.0.253/')
                ->press('#register')
                ->waitForText('Devices')
                ->assertSee('Test Device');
    });
}

public function testRemoveDeviceSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/devices')
            ->waitForText('Devices')
            ->click('a.removeDevice');

        $browser->driver->switchTo()->alert()->accept();
    });
}
```

Fig. 99 – Code Snippet: Acceptance Test of UC04 CRUD Device

Table 33 – Acceptance Test: UC05 Turn On/Off Device

Feature:	5 *Turn On/Off Device*		
Scenario	**Test**	**Expected result**	**Validation**
The End-User clicks on the "On/Off" button of a device.	Click on the "On/Off" button of a device.	The device is turned on/off.	Success

```
public function testActuateDeviceSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/devices')
            ->waitForText('Devices')
            ->click('a.actuate') // turn on
            ->pause('3000')
            ->click('a.actuate'); // turn off
    });
}
```

Fig. 100 – Code Snippet: Acceptance Test of UC05 Turn On/Off Device

Table 34 – Acceptance Test: UC06 Check Device Consumption

Feature:	6 *Check Device Consumption*		
Scenario	**Test**	**Expected result**	**Validation**
The End-User clicks on the “Consumption” button of a device.	Click on the “Consumption” button of a device.	The device’s energy consumption history is displayed	Success

```
public function testCheckDeviceConsumptionSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/devices')
            ->waitForText('Devices')
            ->click('a.checkConsumption')
            ->waitFor('#containerEnergy')
            ->waitFor('#containerEnergyByDay')
            ->waitFor('#containerEnergyDifferenceByDay');
    });
}
```

Fig. 101 – Code Snippet: Acceptance Test of UC06 Check Device Consumption

Table 35 – Acceptance Test: UC07 Create Flex-offer Manually for a Device

Feature:	7 *Create Flex-offer Manually for a Device*		
Scenario	**Test**	**Expected result**	**Validation**
The End-User manually creates a Flex-offer for a device.	Fill out the flex-offer creation form, manually define the energy consumption pattern, and submit.	The flex-offer is created and registered in system.	Success

Table 36 – Acceptance Test: UC08 Create Flex-offer Automatically for a Device

Feature:	8 *Create Flex-offer Automatically for a Device, based on Energy Consumption*		
Scenario	**Test**	**Expected result**	**Validation**

The End-User automatically creates a Flex-offer for a device.	Fill out the flex-offer creation form, let the system define the energy consumption pattern, and submit.	The flex-offer is created and registered in system.	Success

```
public function testCreateFlexofferSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/devices')
                ->waitForText('Devices')
                ->click('a.createFlexoffer')
                ->waitFor('#scheduler_here')
                ->type("#flexofferName", "Flexoffer Test");

        $browser->script('
            scheduler.addEvent({
                start_date: "16-06-2019 00:00",
                end_date:  "16-06-2019 23:00",
                text:    "Flexoffer"
            });
        ');

        $browser->click("Next")
            ->radio('patternChoice', 'automaticPattern')
            ->select('activationHour', '12:00')
            ->click("Finish")
            ->assertSee('Devices');
    });
}
```

Fig. 102 – Code Snippet: Acceptance Test of UC08 Create Flex-offer Automatically for a Device

Table 37 – Acceptance Test: UC09 Check total registered Users, Devices, and Houses & UC10 Check End-Users' Devices' Frequency of Use and average Time of Use

Feature:	9 *Check total registered Users, Devices, and Houses.* 10 *Check End-Users' Devices' Frequency of Use and average Time of Use.*		
Scenario	**Test**	**Expected result**	**Validation**
The Company Executive checks the Executive's platform.	Login in the Executive's platform and check the main page.	The total registered Users, Devices, and Houses are displayed in the main page, as well as the *Frequency of Use and average Time of Use*.	Success

```
public function testLoginSuccessfully()
{
    $this->browse(function (Browser $browser) {
        $browser->visit('/')
                ->assertSee('Executive platform')
                ->type('#username','executive')
                ->type('#password','executive')
                ->clickLink('Login')
                ->assertSee('Overview');
    });
}
```

Fig. 103 – Code Snippet: Acceptance Test of UC09 Check total registered Users, Devices, and Houses & UC10 Check End-Users' Devices' Frequency of Use and average Time of Use

5 Conclusions

This chapter recaps the most relevant points of this work, describes the end-results obtained from the project's development, mentions additional work done, explains the project's limitations and future improvements, and, lastly, gives a final appreciation of the project and internship.

5.1 Report summary

This project's goals were to improve CISTER's FlexHousing system on multiple aspects. First, a full reengineering process of the FlexHousing system was performed, to make it more usable, stable, maintainable and extendable. Moreover, new features were developed.

As explained in section 1.2, the Flex-offer concept consists in the exposure of the users' electrical power consumption flexibility to the energy market. An energy consumption offer containing the user's consumption flexibility, in time and power, is sent to an aggregator, which responds with a schedule that meets the best prices (lowest price) for consumption, while still satisfying the users' needs.

Furthermore, the FlexHousing project consists of a pilot capable of applying the flex-offer concept to a real-life situation (supported by the Arrowhead framework), allowing control over the energy usage of home appliances. FlexHousing is composed of two different applications: one is the FlexHousing Middleware, which communicates with devices, manages a database (which contains the registered users, houses, and devices), and provides its data through a RESTful service to web applications; the other is a web application, known as FlexHousing web platform, which serves as a gateway to the Middleware's data and services. This project also serves as a proof of concept for a multinational company that is interested in the concept of the project's platform infrastructure to support the maintenance of home appliances at their costumers' houses.

The goals of the project are to:

- Rebuild and improve the FlexHousing web platform;
- Reengineer the connection to IoT devices, enabling compatibility with other types of appliances of different brands and manufacturers;

- Reengineer the FlexHousing Middleware (add support for multiple “Users” and “Houses”, and modify its Arrowhead implementation so that it can also function locally);
- Add a feature that automates the flex-offer creation, based on the device’s energy consumption, minimizing user input;
- Add the functionality of verifying the execution of a flex-offer;
- Develop a platform for company executives that integrates with FlexHousing and displays some basic data analyses of user data.

In the analysis and design phase (sections 4.2 and 4.3), the project’s domain model, database structure, and class diagram were altered to satisfy the new requirements gathered in the requirements engineering stage (section 4.1).

In the implementation phase (section 4.4), to support different types of devices, a generic interface for device modules was implemented (section 4.4.1). Then, the existing module for the VPS devices in the Middleware was altered in order to implement it (section 4.4.2). Next, to connect to a new device that communicates through the local network, a generic customizable switch, named Sonoff Pow, was chosen (section 4.4.3). To acquire specific energy consumption data and provide them through a REST API, a custom firmware (Appendix-B) was developed and deployed into the Sonoffs. After that, a module for it was developed and implemented in the Middleware.

In relation to the automatic creation of flex-offers (section 4.4.4), some of the algorithms published in [65] were used to implement a way to identify energy patterns. However, the algorithm for identifying an energy pattern depends on what kind of device it is. Therefore, different algorithms were developed for different kinds of devices (in this case, one for wet-devices and the other for refrigerators).

Regarding the Arrowhead implementation (section 2.4.2), the past version of the project had a severe dependency on servers running in Denmark, which also supported the connection to the Aggregator module, the Service Registry, and the Virtual Market of Energy. The solution to this was to locally implement the Aggregator and VME modules. Moreover, it was also necessary to install an XMPP server so that these modules could be able to communicate with each other and with the FlexHousing Middleware, through XMPP. Furthermore, the system must follow a configuration file (Appendix-A) that specifies the XMPP server’s hostname, port, resource, and service name, and each module’s XMPP client account’s ID and password.

In the testing phase (section 4.5), three different levels of software testing were used: unit testing, integration testing, and acceptance testing.

5.2 Accomplished goals

In this section, a degree of accomplishment is specified for each objective presented in the introduction chapter.

Table 38 – Accomplished goals

Goal	Degree of Accomplishment
Rebuild and improve the FlexHousing web platform.	Complete
Develop a generic interface for future device implementations.	Complete
Add support for a new device that communicates through a local network, and not through external servers.	Complete
Add support for multiple “Users” and “Houses” in the FlexHousing Middleware.	Complete
Develop a feature for automatic creation of Flex-offers for wet-devices, based on their consumption patterns.	Complete
Develop a feature for automatic creation of Flex-offers for refrigerators, based on their consumption patterns.	Complete
Reengineer the FlexHousing Middleware so that its Arrowhead implementation can also function locally, without needing to connect to external servers.	Complete
Develop a platform for company executives that integrates with FlexHousing and displays some basic analyses of user data.	Complete

5.3 Additional work done

During the project's development, a contribution was made for the (yet to be published) paper "*FlexHousing: FlexOffer concept for the energy manager*" [90]. Additionally, some presentations and context diagrams were made for meetings with the customers.

5.4 Limitations and future development

Although the project achieved all the initially planned goals, as well as those that have been added during development, it is possible to enumerate some limitations that may arise, along with a few solutions and new features that could be implemented in the future.

First, although the web platform's frontend has been heavily improved, some further enhancements could be made to enrich the User Experience. For instance, the process of registering a new device on the platform requires a great deal of user input, when it should be a simpler procedure. Required details like the device sensor API's web address or the device sensors' brand are specifics that the user shouldn't have to know or find out. One solution for this problem would be the use of QR codes for each device sensor, where one QR code would contain all the details and metadata of a specific device sensor. If a device only communicates through the local network, then the Middleware would search for its IP.

Second, when creating a new flex-offer on the web platform, the flex-offer is always scheduled to be activated every day in the time window specified by the user. This could be a problem if, for example, the user only wants it to run once, or once every two days, or once a week, and so forth. As such, the platform should support and allow different scheduling periods specified by the user.

Third, while the concept of multiple "Users" and "Houses" was implemented in the FlexHousing system, there is no functionality in the web platform for one user to add another user to their house. Although it is completely possible to do so directly in the database, in practical situations, this currently isn't conceivable through the web platform. The reason for this is because it would require more development time to implement a system where:

- A user selects an already registered house, and requests to have permission to access it; the house's owner would then be notified that someone wants to access their house and would allow or deny access permission;
- A house owner would search for specific users registered in the system, and allow them to have access permission.

In a future iteration though, this feature could be certainly achievable.

Fourth, although HTTP/REST was the chosen communication method between the FlexHousing Middleware and its registered devices, in the case of devices like Sonoff, where their data has to be stored in the Middleware's database, a more ideal communication protocol could be AMQP using the *publish/subscribe* pattern. In a REST environment, the Middleware must create a thread for each device, continually requesting data from each one's REST API. Alternatively, in a AMQP environment with *publish/subscribe*, the devices, as publishers, would send their data through a message broker to their subscribers, in this case, the Middleware. Thus, in a situation where there are hundreds of registered devices, the Middleware wouldn't have to request data from each one, instead it would receive all messages from a single broker.

Finally, the idea of detecting equipment malfunctions through energy consumption was considered, since it was also suggested by the client. However, its implementation was not carried through, because not only can it prove difficult to establish through energy consumption data that a malfunction has actually happened, but abnormal consumption data also wouldn't be proof enough for some situations. A good example for these kinds of situations would be for a refrigerator. A refrigerator's energy consumption hinges on the ambient temperature of its surroundings: if it's cold, the refrigerator won't consume much energy; if it's hot, then the refrigerator will have to consume more energy than usual. To establish that a refrigerator is malfunctioning, then the best approach would be to install another sensor, specifically, a temperature sensor. This way, the system could match consumption values with temperature values (through their measurements' timestamps) and determine if there are any long-term discrepancies between the two. If so, then a malfunction may be likely.

5.5 Final appreciation

Considering the web platforms developed and the Middleware reengineering, the documentation produced, the deadlines related to planning, and the new features requested, this project can be considered a success. All client meetings, where a new iteration of the FlexHousing project is presented, were met with satisfaction and furthered the client's interest in the subject.

However, while the presented solution reflects a good understanding of the main subject, there are obvious improvements to be made, as suggested in section 5.4.

On a more personal note, working at CISTER on a proof of concept for a big multinational company was a great learning experience. The work environment is very flexible, any requests for equipment or other demands are met within a short time, and all the people working at CISTER are very friendly and always able to help. From working with multicultural teams to learning different technologies in IoT, this internship has contributed to an incredible experience that helps one develop their technical and social skills.

6 Bibliography

1. Griffith, E., & Colon, A. (2017, May 08). The Best Smart Home Devices of 2017. Retrieved February 17, 2017, from http://www.pcmag.com/article2/0,2817,2410889,00.asp.

2. CISTER Research Unit. (n.d.). Retrieved February 17, 2017, from http://www.cister.isep.ipp.pt/info/

3. P., Barnaghi. (2016, July 2). Internet of Things: Concepts and Technologies [Scholarly project]. In SlideShare. Retrieved February 17, 2017, from https://pt.slideshare.net/PayamBarnaghi/internet-of-things-concepts-and-technologies

4. Automatic Meter Reading. (n.d.). Retrieved February 17, 2017, from http://www.isasensing.com/solutions/automatic-meter-reading/

5. T., Angelucci. (2014, July). A Typical IoT Value Chain [Digital image]. Retrieved February 18, from http://rtcmagazine.com/articles/view/103677

6. L. L., Ferreira, L., Siksnys, P., Pedersen, P., Stluka, C., Chrysoulas, T. L., Guilly, T., Pedersen. (2015). Arrowhead compliant virtual market of energy (Barcelona, Spain, 2014). Barcelona. Retrieved February 18, from http://ieeexplore.ieee.org/document/7005193/

7. Le Guilly, Thibaut, et al. "An Energy Flexibility Framework on The Internet of Things." The Success of European Projects using New Information and Communication Technologies (2016): 17-37

8. M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Siksnys, and T. Tusar, "Data management in the mirabel smart grid system," in EnDM, 2012, pp.95-102.

9. Flex-offer example [Digital image]. (2015, January 12). Retrieved February 23, 2017, from http://ieeexplore.ieee.org/ielx7/6994138/7005023/7005193/html/img/7005193-fig-1-large.gif

10. TotalFlex. (n.d.). Retrieved February 23, 2017, from http://www.totalflex.dk/

11. Virtual market of energy main actors and operations [Digital image]. (2015, January 12). Retrieved February 23, 2017, from http://ieeexplore.ieee.org/ielx7/6994138/7005023/7005193/html/img/7005193-fig-2-large.gif

12. High level architecture for the virtual market of energy [Digital image]. (2015, January 12). Retrieved February 23, 2017, from http://ieeexplore.ieee.org/ielx7/6994138/7005023/7005193/html/img/7005193-fig-3-large.gif

13. E. Guttman, "Autoconfiguration for IP Networking: Enabling Local Communication", IEEE Internet Computing 5 (3), 2001, pp. 81-86.

14. D. B. Terry, M. Painter, D. W. Riggle and S. Zhou, "The Berkeley Internet Name Domain Server", Proceedings of USENIX Summer Conference, Salt Lake City, Utah, 1984, pp. 23-31.

15. OpenADR Alliance. (n.d.). Retrieved from http://www.openadr.org/

16. "ISO/IEC/IEEE P21451-1-4 Standard for a Smart Transducer Interface for Sensors, Actuators, and Devices based on the eXtensible Messaging and Presence Protocol (XMPP) for Networked Device Communication," Available online: Http://wiki.xmpp.org/web/Tech/IoT.pages-Sensei, accessed April 2014.

17. Pieper, C. (n.d.). How the Internet of Things Intersects with Energy Management. Retrieved March 5, 2017, from https://www.artisenergy.com/blog/how-the-internet-of-things-intersects-with-energy-management

18. A., Willis. (November 13). The Benefits of Becoming a Smart City - Infographic. Retrieved March 5, 2017, from https://datafloq.com/read/the-benefits-of-becoming-a-smart-city/1644

19. Big Data and the IoT: The Future of the Smart City. (n.d.). Retrieved March 5, 2017, from http://graduatedegrees.online.njit.edu/resources/mscs/mscs-infographics/big-data-and-the-iot-the-future-of-the-smart-city/

20. Smart Grid Watch Team. (n.d.). Smart Grid Watch. Retrieved March 05, 2017, from https://blogs.siemens.com/en/smart-grid-watch.entry.html/1782-smart-grid-benefits-for-consumers-service-providers.html

21. C., Teixeira, M., Albano, A., Skou, L. P., Dueñas, F., Antonacci, R., Ferreira, . . . S., Scalari. (2014). CONVERGENCE TO THE EUROPEAN ENERGY POLICY IN EUROPEAN COUNTRIES: CASE STUDIES AND COMPARISON.

22. Castellanos, M., Dayal, U., & Rundensteiner, E. A. (2013). Enabling Real-Time Business Intelligence 6th International Workshop, BIRTE 2012, Held at the 38th International Conference on Very Large Databases, VLDB 2012, Istanbul, Turkey, August 27, 2012, Revised Selected Papers. Berlin, Heidelberg: Springer Berlin Heidelberg.

23. TotalFlex demonstration. (n.d.). http://smart-cities-centre.org/wp-content/uploads/Per-Pedersen.pdf

24. Arrowhead — Ahead of the future. (n.d.). Retrieved March 12, 2017, from http://www.arrowhead.eu/

25. Dannecker, L. (2015). Energy time series forecasting: efficient and accurate forecasting of evolving time series from the energy domain. Wiesbaden: Springer Vieweg. Retrieved March 12, 2017, from https://books.google.pt/books?id=ufhUCgAAQBAJ&pg=PA41&lpg=PA41&dq=mirabel+flex-offer&source=bl&ots=P0xuTSOwDE&sig=ALBoGLO43RI_9h6-lX-Z7NHPviw&hl=pt-PT&sa=X&ved=0ahUKEwitmoT-8ZTUAhUJVxoKHWfRD6UQ6AEIMzAC#v=onepage&q=mirabel%20flex-offer&f=true

26. A., Doms, Z., Marinzek, & T. B., Pedersen. (2013). MIRABEL - Efficiently managing more renewable energy using explicit demand and supply flexibilities (Doctoral dissertation, Aalborg University, 2013). Berlin. Retrieved March 12, 2017, from http://vbn.aau.dk/files/160236283/MIRABEL_Efficiently_managing_more_renewable_energy_using_explicit_demand_and_supply_flexibilities.pdf

27. TotalFlex. (n.d.). Retrieved March 19, 2017, from http://neogrid.dk/portfolio/totalflex-new/

28. Woon, W. L., Aung, Z., Kramer, O., & Madnick, S. (2017). Data Analytics for Renewable Energy Integration: 4th ECML PKDD Workshop, DARE 2016, Riva del Garda, Italy, September 23, 2016, Revised Selected Papers. Cham: Springer International Publishing. Retrieved March 19, 2017, from https://books.google.pt/books?id=scSPBQAAQBAJ&pg=PA2&lpg=PA2&dq=total+flex+energy&source=bl&ots=_xDwT386jc&sig=MjKfEdOxU35lPnsmVJKyGaObH-0&hl=pt-

PT&sa=X&ved=0ahUKEwjIxN2dgJXUAhUB2BoKHVULB6YQ6AEIOTAD#v=onepage&q=total%20flex%20energy&f=false

29. Arrowhead framework — Arrowhead. (n.d.). Retrieved March 19, 2017, from http://www.arrowhead.eu/about/arrowhead-common-technology/arrowhead-framework/
30. Arrowhead Framework [Digital image]. (n.d.). Retrieved March 19, 2017, from http://www.arrowhead.eu/wp-content/uploads/2014/10/arrowhead-framework.png
31. Ry Crist October 14, 2015 5:00 AM PDT @rycrist. (2015, October 14). A smart home divided: Can it stand? Retrieved March 26, 2017, from https://www.cnet.com/news/a-smart-home-divided-can-it-stand/
32. Apple HomeKit App [Digital image]. (n.d.). Retrieved March 26, 2017, from http://dispatchweekly.com/wp-content/uploads/2016/09/Apple-iOS-10-HomeKit-App5.png
33. Nest [Digital image]. (n.d.). Retrieved March 26, 2017, from http://media.idownloadblog.com/wp-content/uploads/2015/06/Nest-5.0-for-iOS-iPhone-screenshot-001.jpg
34. IFTTT. (2017, May 25). Retrieved May 26, 2017, from https://en.wikipedia.org/wiki/IFTTT
35. IFTTT. (n.d.). Retrieved May 26, 2017, from https://ifttt.com/
36. Kreuzer, K. (2015, August 15). Re: Openhab for business purposes? [Web log comment]. Retrieved April 2, 2017, from https://community.openhab.org/t/openhab-for-business-purposes/1460/2
37. Baker, J., (Red Hat). (2016, March 29). 5 open source home automation tools. Retrieved April 2, 2017, from https://opensource.com/life/16/3/5-open-source-home-automation-tools
38. HABPanel. (n.d.). Retrieved April 2, 2017, from http://demo.openhab.org:8080/habpanel/index.html#/view/first-floor
39. [Digital image]. (n.d.). Retrieved from https://www.mysensors.org/uploads/57be15b86b0aea1b61746265/394/homeassistant_devices.png

40. Alexandra. (2016, September 30). IoT is eating the world: APIs and REST – Alexandra – Medium. Retrieved April 2, 2017, from https://medium.com/@AlexandraBowen/iot-is-eating-the-world-apis-and-rest-9e0321bc6cbf

41. Rational Unified Process: Best Practices for Software Development Teams [PDF]. (1998). IBM Rational Software Corporation. https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

42. [Digital image]. (n.d.). Retrieved April 17, 2017, from http://wm2info.com.br/wp-content/uploads/2012/01/RUPSummaryDiag.gif

43. Atlassian. (n.d.). What is version control | Atlassian Git Tutorial. Retrieved April 17, 2017, from https://www.atlassian.com/git/tutorials/what-is-version-control

44. Lawrizs. (n.d.). Lawrizs/ARROWHEAD_VME. Retrieved February 24, 2017, from https://github.com/lawrizs/ARROWHEAD_VME

45. Software Requirements. (n.d.). Retrieved May 1, 2017, from https://www.tutorialspoint.com/software_engineering/software_requirements.htm

46. Sommerville, Ian (2009). Software Engineering (9th ed.). Addison-Wesley. ISBN 978-0-13-703515-1.

47. Cohn, M. (2004, May 21). Telling Stories and User Role Modeling. Retrieved May 1, 2017, from http://www.informit.com/articles/article.aspx?p=170964

48. Cohn, M. (2014, July 1). Adding Decorated User Roles to Your User Stories. Retrieved May 1, 2017, from https://www.mountaingoatsoftware.com/blog/adding-decorated-user-roles-to-your-user-stories

49. Cohn, M. (n.d.). User Stories and User Story Examples by Mike Cohn. Retrieved May 1, 2017, from https://www.mountaingoatsoftware.com/agile/user-stories

50. LBushkin. (2013, May 10). What is functional and non functional requirement? [Online forum comment]. Retrieved May 7, 2017, from https://stackoverflow.com/questions/16475979/what-is-functional-and-non-functional-requirement

51. Ergonomic Requirements for Office Work with Visual Display Terminals, ISO 9241-11, ISO, Geneva, 1998.

52. Computer performance. (2017, August 31). Retrieved August 31, 2017, from https://en.wikipedia.org/wiki/Computer_performance

53. Software portability. (2017, August 06). Retrieved August 29, 2017, from https://en.wikipedia.org/wiki/Software_portability

54. Definition of Interoperability. (n.d.). Retrieved August 29, 2017, from http://interoperability-definition.info/en/

55. Use case. (2017, August 22). Retrieved August 22, 2017, from https://en.wikipedia.org/wiki/Use_case

56. W. Tracz. Domain analysis working group report. In First International Workshop on Software Reusability, 1991.

57. Fowler, Martin. Patterns of Enterprise Application Architecture. Addison Wesley, 2003, p. 116.

58. G., Sunyé. (2015, April 13). Domain Analysis [Scholarly project]. In SlideShare. Retrieved May 15, 2017, from https://pt.slideshare.net/sunye/domain-analysis

59. Santos, J. (2016). Creation of a pilot for the FlexOffer concept. 55. Retrieved from http://www.cister.isep.ipp.pt/docs/creation_of_a_pilot_for_the_flexoffer_concept/1256/view.pdf

60. Object Oriented Design. (2017, August 15). Retrieved August 17, 2017, from https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_design.htm

61. Object Oriented Design. (2017, August 15). Retrieved August 17, 2017, from https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_design.htm

62. Class diagram. (2017, August 24). Retrieved August 27, 2017, from https://en.wikipedia.org/wiki/Class_diagram

63. GRASP (object-oriented design). (2017, March 06). Retrieved June 16, 2017, from https://en.wikipedia.org/wiki/GRASP_(object-oriented_design)#Controller

64. Bell, D. (2004, February 16). The sequence diagram. Retrieved July 17, 2017, from https://www.ibm.com/developerworks/rational/library/3101.html

65. B. Neupane, L. Siksnys, T. Pedersen. (2017). Generation and Evaluation of Flex-Offers from Flexible Electrical Devices. Retrieved July 21, from http://dl.acm.org/citation.cfm?id=3077850

66. B. Neupane, L. Siksnys, T. Pedersen. (2017). Generation and Evaluation of Flex-Offers from Flexible Electrical Devices. 12. Retrieved July 21, from http://dl.acm.org/citation.cfm?id=3077850

67. B. Neupane, L. Siksnys, T. Pedersen. (2017). Generation and Evaluation of Flex-Offers from Flexible Electrical Devices. 5. Retrieved July 21, from http://dl.acm.org/citation.cfm?id=3077850

68. Integration Testing. (2017, August 15). Retrieved August 17, 2017, from https://www.tutorialspoint.com/software_testing_dictionary/integration_testing.htm

69. Integration Testing Tutorial: Big Bang, Top Down & Bottom Up. (n.d.). Retrieved August 17, 2017, from https://www.guru99.com/integration-testing.html

70. Stub. (2017, August 15). Retrieved August 17, 2017, from https://www.tutorialspoint.com/software_testing_dictionary/stub.htm

71. Using in-memory databases. (n.d.). Retrieved August 17, 2017, from https://db.apache.org/derby/docs/10.11/devguide/cdevdvlpinmemdb.html

72. Acceptance Testing. (2017, July 23). Retrieved August 20, 2017, from https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm

73. Otwell, T. (n.d.). Browser Tests (Laravel Dusk). Retrieved August 18, 2017, from https://laravel.com/docs/5.4/dusk

74. SOA4D Forge: Arrowhead Framework: Source Code Repository for Arrowhead Framework. (n.d.). Retrieved September 15, 2017, from https://forge.soa4d.org/scm/?group_id=58

75. P., Varga, F., Blomstedt, L. L., Ferreira, J., Eliasson, M., Johansson, J., Delsing, I., Martinez de Soria. (2016, August 28). Making System of Systems Interoperable - the Core Components of the Arrowhead Framework. Retrieved February 18, from http://www.arrowhead.eu/wp-content/uploads/2013/03/Arrowhead_core_Elsevier-cr2.pdf

76. T. Erl, SOA Principles of Service Design (The Prentice Hall Service Oriented Computing Series from Thomas Erl), Prentice Hall PTR, Upper Saddle River, NJ, USA, 2007.

77. K. Nagorny, R. Harrison, A. Colombo, G. Kreutz, A formal engineering approach for control and monitoring systems in a service-oriented environment, in: IEEE International Conference on Industrial Informatics (INDIN), 2013, pp. 480–487.

78. APIs for Internet of Things (IoT). (n.d.). Retrieved April 2, 2017, from http://www.axway.com/en/enterprise-solutions/api-management/api-internet-of-things-iot

79. Rele, A. (2015, August 26). How APIs Unlock Value from the IoT. Retrieved April 2, 2017, from https://apigee.com/about/blog/digital-business/how-apis-unlock-value-iot

80. Dersin, P. (2014, October 15). Systems of Systems. Retrieved September 21, 2017, from http://rs.ieee.org/tech-activities/77-systems-of-systems

81. "Flat design style modern vector illustration concept of smart..". Adapted from Flat design style modern vector illustration concept of smart.., by bloomua. Retrieved from http://s3-us-west-2.amazonaws.com/simplicitywebstorage/wp-content/uploads/2017/06/05094407/smarty-home-1.jpg

82. Stellwagen, E. (n.d.). Forecasting 101: A Guide to Forecast Error Measurement Statistics and How to Use Them. Retrieved September 28, 2017, from http://www.forecastpro.com/Trends/forecasting101August2011.html

83. Mean absolute percentage error. (2017, July 27). Retrieved September 28, 2017, from https://en.wikipedia.org/wiki/Mean_absolute_percentage_error#Alternative_MAPE_definitions

84. Tim. (2017, May 9). Is MAPE a good error measurement statistic? And what alternatives are there? [Online forum comment]. Retrieved September 28, 2017, from https://stats.stackexchange.com/questions/280464/is-mape-a-good-error-measurement-statistic-and-what-alternatives-are-there

85. Mean Absolute Percent Error (MAPE). (n.d.). Retrieved September 28, 2017, from http://www.vanguardsw.com/business-forecasting-101/mean-absolute-percent-error/

86. Symmetric Mean Absolute Percent Error (SMAPE). (n.d.). Retrieved September 28, 2017, from http://www.vanguardsw.com/business-forecasting-101/symmetric-mean-absolute-percent-error-smape/

87. Giannakopoulos, T. (n.d.). pyAudioAnalysis. Retrieved August 25, 2017, from https://github.com/tyiannak/pyAudioAnalysis

88. Heller, M. (2007, January 29). REST and CRUD: the Impedance Mismatch. Retrieved October 09, 2017, from https://www.infoworld.com/article/2640739/application-development/rest-and-crud--the-impedance-mismatch.html

89. The energy market explained. (5, March 2017). Retrieved October 10, 2017, from http://www.energy-uk.org.uk/energy-industry/the-energy-market.html

90. Santos, J., Albano, M., Ferreira, L.L., Silva, J., Rocha, R., Olsen, P., Matos, L. (2017). FlexHousing: FlexOffer concept for the energy manager. Manuscript in preparation.

7 Appendixes

7.1 Appendix-A – Configuration Properties file for XMPP communication

```
aggregatorid=aggregator
aggregatorpassword=aggregator

flexoffermanagerid=admin
flexoffermanagerpassword=password

marketid=market
marketpassword=market

xmpphostname=192.168.60.110
xmppport=5222
xmppresource=demo
xmppservicename=localhost
```

Fig. 104 – Configuration Properties file (config.properties) for XMPP communication

7.2 Appendix-B – Sonoff Pow Custom Firmware

```
// Import required libraries
#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <aREST.h>
#include "HLW8012.h"

#define SERIAL_BAUDRATE                 115200

// GPIOs
#define RELAY_PIN                       12
#define SEL_PIN                         5
#define CF1_PIN                         13
#define CF_PIN                          14

// Check values every 2 seconds
#define UPDATE_TIME                     2000

// Set SEL_PIN to HIGH to sample current
// This is the case for Itead's Sonoff POW, where a
// the SEL_PIN drives a transistor that pulls down
// the SEL pin in the HLW8012 when closed
#define CURRENT_MODE                    HIGH

// These are the nominal values for the resistors in the circuit
#define CURRENT_RESISTOR                0.001
#define VOLTAGE_RESISTOR_UPSTREAM       ( 5 * 470000 ) // Real: 2280k
#define VOLTAGE_RESISTOR_DOWNSTREAM     ( 1000 ) // Real 1.009k

HLW8012 hlw8012;
int gpio13Led = 13;
int gpio12Relay = 12;

// Create aREST instance
aREST rest = aREST();

// WiFi parameters
const char* ssid = "CISTER";
const char* password = "2ae7a525d4882ed2a8ee1890968932f6";

// The port to listen for incoming TCP connections
#define LISTEN_PORT           80

// Create an instance of the server
WiFiServer server(LISTEN_PORT);

// Variables to be exposed to the API
int activePower = 0;
int voltage = 0;
double current = 0;
int apparentPower = 0;
double powerFactor = 0;
String consumptionVariables = "";
String state = "Off";

// Auxiliary variable
int auxInt = 0;
double auxDouble = 0;

// Define the number of samples to keep track of.  The higher the number,
// the more the readings will be smoothed, but the slower the output will
// respond to the input.  Using a constant rather than a normal variable lets
// use this value to determine the size of the readings array.
const int numReadings = 30;

int readings[numReadings];      // the readings from the analog input
int readIndex = 0;              // the index of the current reading
int total = 0;                  // the running total
int average = 0;                // the average

// Declare functions to be exposed to the API
```

```
int ledControl(String command);
int actuate(String param);


void unblockingDelay(unsigned long mseconds) {
    unsigned long timeout = millis();
    while ((millis() - timeout) < mseconds) delay(1);
}

void calibrate() {
    // Let's first read power, current and voltage
    // with an interval in between to allow the signal to stabilise:

    hlw8012.getActivePower();

    hlw8012.setMode(MODE_CURRENT);
    unblockingDelay(2000);
    hlw8012.getCurrent();

    hlw8012.setMode(MODE_VOLTAGE);
    unblockingDelay(2000);
    hlw8012.getVoltage();

    // Calibrate using a 60W bulb (pure resistive) on a 230V line
    hlw8012.expectedActivePower(60.0);
    hlw8012.expectedVoltage(230.0);
    hlw8012.expectedCurrent(60.0 / 230.0);

    // Show corrected factors
    //Serial.print("[HLW] New current multiplier : ");
Serial.println(hlw8012.getCurrentMultiplier());
    //Serial.print("[HLW] New voltage multiplier : ");
Serial.println(hlw8012.getVoltageMultiplier());
    //Serial.print("[HLW] New power multiplier   : ");
Serial.println(hlw8012.getPowerMultiplier());
    //Serial.println();

}

void setup(void)
{
  // Start Serial port and clean garbage
  Serial.begin(SERIAL_BAUDRATE);
  Serial.println();
  Serial.println();

  // Close the relay to switch on the load
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, HIGH);
  pinMode(CF_PIN, INPUT_PULLUP);

  // Initialize HLW8012
  // void begin(unsigned char cf_pin, unsigned char cf1_pin, unsigned char sel_pin,
unsigned char currentWhen = HIGH, bool use_interrupts = false, unsigned long pulse_timeout
= PULSE_TIMEOUT);
  // * cf_pin, cf1_pin and sel_pin are GPIOs to the HLW8012 IC
  // * currentWhen is the value in sel_pin to select current sampling
  // * set use_interrupts to false, we will have to call handle() in the main loop to do
the sampling
  // * set pulse_timeout to 500ms for a fast response but losing precision (that's ~24W
precision :( )
  hlw8012.begin(CF_PIN, CF1_PIN, SEL_PIN, CURRENT_MODE, false, 500000);

  // These values are used to calculate current, voltage and power factors as per
datasheet formula
  // These are the nominal values for the Sonoff POW resistors:
  // * The CURRENT_RESISTOR is the 1milliOhm copper-manganese resistor in series with the
main line
  // * The VOLTAGE_RESISTOR_UPSTREAM are the 5 470kOhm resistors in the voltage divider
that feeds the V2P pin in the HLW8012
  // * The VOLTAGE_RESISTOR_DOWNSTREAM is the 1kOhm resistor in the voltage divider that
feeds the V2P pin in the HLW8012
  hlw8012.setResistors(CURRENT_RESISTOR, VOLTAGE_RESISTOR_UPSTREAM,
VOLTAGE_RESISTOR_DOWNSTREAM);
```

```
  //calibrate();

  // Init variables and expose them to REST API
  consumptionVariables = "";
  rest.variable("consumption",&consumptionVariables);
  rest.variable("state",&state);

  // Function to be exposed
  rest.function("led",ledControl);
  rest.function("actuate",actuate);

  // Give name & ID to the device (ID should be 6 characters long)
  rest.set_id("1");
  rest.set_name("esp8266");

  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.println(WiFi.localIP());

    // initialize all the readings to 0:
  for (int thisReading = 0; thisReading < numReadings; thisReading++) {
    readings[thisReading] = 0;
  }

  checkState();
}

void loop() {

    static unsigned long last = millis();

    // This UPDATE_TIME should be at least twice the minimum time for the current or
voltage
    // signals to stabilize. Experimentally that's about 1 second.
    if ((millis() - last) > UPDATE_TIME) {

        last = millis();

        // subtract the last reading:
        total = total - readings[readIndex];
        // read from the sensor:
        readings[readIndex] = hlw8012.getActivePower();
        // add the reading to the total:
        total = total + readings[readIndex];
        // advance to the next position in the array:
        readIndex = readIndex + 1;

        // if we're at the end of the array...
        if (readIndex >= numReadings) {
          // ...wrap around to the beginning:
          readIndex = 0;
        }

        // calculate the average:
        average = total / numReadings;

        // Active power
        Serial.print("[HLW] Active Power (W)    : ");
        activePower = hlw8012.getActivePower();
        Serial.println(activePower);
        // Average Active power
        Serial.print("[HLW] Average Active Power (W)    : ");
        Serial.println(average);

        // Voltage
        Serial.print("[HLW] Voltage (V)         : ");
        voltage = hlw8012.getVoltage();
        Serial.println(voltage);

        // Current
        Serial.print("[HLW] Current (A)         : ");
        current = hlw8012.getCurrent();
        Serial.println(current);

        // Apparent Power
        Serial.print("[HLW] Apparent Power (VA) : ");
        apparentPower = hlw8012.getApparentPower();
        Serial.println(apparentPower);

        // Power Factor
        Serial.print("[HLW] Power Factor (%)    : ");
        powerFactor = (int) (100 * hlw8012.getPowerFactor());
        Serial.println(powerFactor);

        consumptionVariables = "";
        consumptionVariables += "Active Power (W)=";
        consumptionVariables += activePower;
        consumptionVariables += ",";
        consumptionVariables += "Average Active Power (W)=";
        consumptionVariables += average;
        consumptionVariables += ",";
        consumptionVariables += "Voltage (V)=";
        consumptionVariables += voltage;
        consumptionVariables += ",";
        consumptionVariables += "Current (A)=";
        consumptionVariables += current;
        consumptionVariables += ",";
        consumptionVariables += "Apparent Power (VA)=";
        consumptionVariables += apparentPower;
        consumptionVariables += ",";
        consumptionVariables += "Power Factor (%)=";
        consumptionVariables += powerFactor;

        Serial.println();

        // When not using interrupts we have to manually switch to current or voltage
monitor
        // This means that every time we get into the conditional we only update one of
them
        // while the other will return the cached value.
        hlw8012.toggleMode();
    }

    checkState();

    // Handle REST calls
    WiFiClient client = server.available();
    if (!client) {
      return;
    }
    while(!client.available()){
      delay(1);
    }
    rest.handle(client);
}

// Custom function accessible by the API
int ledControl(String command) {

  // Get state from command
  int state = command.toInt();

  digitalWrite(6,state);
  return 1;
```

```
}

// Custom function accessible by the API
int actuate(String param) {

  int command = param.toInt();

  // Turn Off
  if (command == 0) {
    digitalWrite(gpio13Led, HIGH);
    digitalWrite(gpio12Relay, LOW);
    state = "Off";
    return 0;

  // Turn On
  } else if (command == 1) {
    digitalWrite(gpio13Led, LOW);
    digitalWrite(gpio12Relay, HIGH);
    state = "On";
    return 1;

  // Actuate
  } else if (command == 2) {
    digitalWrite(gpio13Led, !digitalRead(gpio13Led));
    digitalWrite(gpio12Relay, !digitalRead(gpio12Relay));
    checkState();
    return 2;
  }

  return 3;
}

void checkState(){
  if (digitalRead(gpio13Led) == LOW && digitalRead(gpio12Relay) == HIGH){
    state = "On";
  } else if (digitalRead(gpio13Led) == HIGH && digitalRead(gpio12Relay) == LOW){
    state = "Off";
  }
}
```

Fig. 105 – Code: Sonoff Pow Custom Firmware

7.3 Appendix-C – FlexHousing System Setup Guide

7.3.1 FlexHousing Middleware

To run the FlexHousing Middleware, it is recommended to use a Java IDE (the NetBeans IDE was the one chosen for the development of this project) and run the Apache Derby database "FlexHousing".

In case you want the Middleware to communicate with the BNearIT servers to access an Aggregator that links to the real Virtual Market of Energy (VME), please read section *7.3.1.1 Remote Access*.

If you wish to use the local Aggregator and VME modules and make the Middleware connect to those, please read section *7.3.1.2 Local Access*.

7.3.1.1 Remote Access

First, you will have to connect to BNearIT's VPN (for more details, visit the link https://forge.soa4d.org/svn/arrowhead/WP5/FlexTutorial/arrowhead-benearit.html):

- IP: 77.53.53.44
- Port: 45678
- Username: guest
- Password: Guest5678&&

Next, you will have to configure the Middleware's XMPP client settings. To do so, you have to edit the properties file (full name: *config.properties*) in the *resources* folder located in the *FlexHousing* project.

In this case, you should focus on the `flexoffermanager` and `xmpp` properties. The following info should be useful for the XMPP configuration:

- The `flexoffermanagerid` property refers to the XMPP server's account username/id for the Middleware;
- The `flexoffermanagerpassword` property refers to the XMPP server's account password for the Middleware;
- The `xmpphostname` property refers to the XMPP server's IP address or DNS;
- The `xmppport` property refers to the XMPP server's listening port;
- The `xmppresource` property must be the same as the Aggregator's;

- The `xmppservicename` property refers to the XMPP domain.

7.3.1.2 Local Access

First, you need to set up an XMPP server:

- Create three accounts: one for the VME, one for the Aggregator, and one for the FlexofferAgent (the Middleware);
- The current XMPP server (running on Ubuntu 16.04 LTS) is using ejabberd for the server and has these three accounts registered:
  - ID: market; Password: market
  - ID: aggregator; Password: aggregator
  - ID: admin; Password: password

Next, you must configure the XMPP client settings in the MarketManager and AggregatorManager classes (both are in the Wp5 project), and in the FlexofferManager class (located in the FlexHousing project).

While you could manually edit these three classes, the recommended way is to edit the properties file (full name: *config.properties*) placed in the Wp5 and FlexHousing project folders. Either way, the following info should be useful for the XMPP configuration:

- These three classes use the Smack library as their XMPP client service;
- Important things to take note:
  - The `id` properties refer to the XMPP server's account username/id;
  - The `password` properties refer to the XMPP server's account password;
  - The `xmpphostname` property refers to the XMPP server's IP address or DNS;
  - The `xmppport` property refers to the XMPP server's listening port (on ejabberd, the port is 5222);
  - The service name refers to the XMPP domain;
    - A jabber account is identified like this: username@domain. For instance, the Aggregator account is identified as aggregator@localhost;
  - The `xmppresource` property must be the same for all classes;

- It is recommended to set *Security Mode* to disabled to avoid any authorization "annoyances";
- The FlexofferManager must identify the AggregatorManager (`aggregatorid` variable), and the AggregatorManager must identify the MarketManager (`marketid` variable).

7.3.2 FlexHousing Web Platform

To run the FlexHousing web platform, it is necessary to install Laravel 5.4 or higher and the Laravel Homestead virtual machine (for more details, visit the link https://laravel.com/docs/5.4/installation).

After the Laravel installation, you should look into the `.env` file located in the FlexHousing web platform folder, and check if the value of the `API_URL` property corresponds to the FlexHousing Middleware's REST API web address.

Lastly, input the command "`homestead up`" in your terminal and the FlexHousing web platform shall then be available through the web address "*flexhousing.app*".