Electrical and Computer Engineering Department - INSA Toulouse
5th year Innovative Smart Systems

Room Management for INSA buildings

Service Oriented Architecture Project

Authors:
Maxime Arens
Vincent Erb
Marine Péfau

Tutor:
Ghada Gharbi
`ghada.gharbi@laas.fr`

December 19, 2019

Contents

1. Introduction

The objective of this project is to develop an application for managing smart rooms on the INSA campus. The idea is to imagine different sensors and actuators and to link them in multiples scenarios that are close to what could happen in real life. We chose to study our own building: the GEI and two rooms.
This project has two main objectives. First, to make the link between the Service Oriented Architecture concepts we learned and the OM2M platform. We used OM2M to simulate our sensors and actuators, in a way that is really close to how actual sensors behave on the platform. Second, it is a group project that was a good opportunity for us to use the Agile method, especially with the Icescrum managing tool.
In the first part, we will approach the main conceptions points, with the OM2M tree, the architecture diagram, and implementation choices. Then, we will dive deeper into the implementation and technical challenges that ensued. Finally, we will presents our results and talk about how we managed the project using the Agile method, going over each sprint.

2. Conception

Before going ahead and starting to implement everything, we spent a lot of time planning the conception, first for the OM2M tree, and then for the services and the implementation architecture. In this chapter, you will find the three main conception diagrams along with explanations of our main choices.

2.1 OM2M architecture

We decided to focus on a single building, the GEI. We have an Infrastructure Node (IN) representing the building itself, then two Middle Nodes (MN) representing the two smart rooms we will be working on.
In each room, an Application Entity (AE) represents a sensor/actuator, with a Container (CNT) associated with it to hold all the Content Instances (CIN).

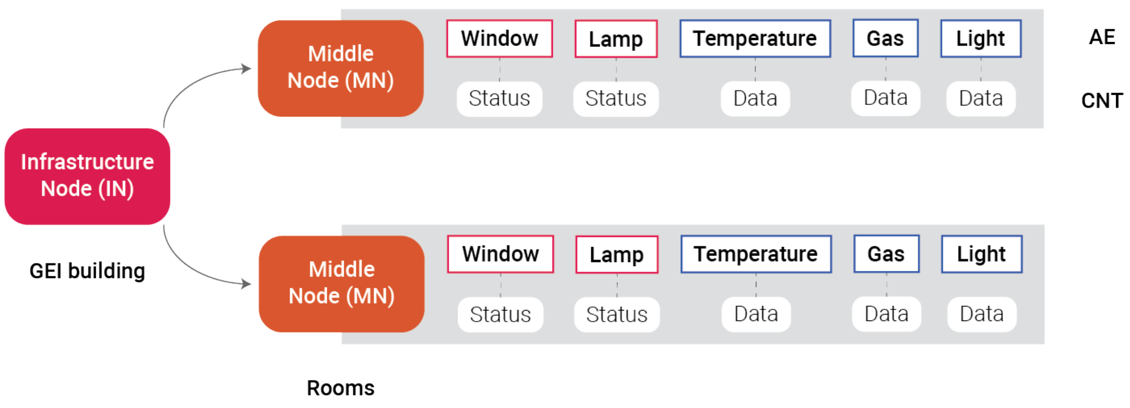


Figure 2.1 – OM2M Architecture diagram

Note: There are multiple AEs for Windows and Lamps in each room, we just represented one in the diagram for clarity reasons.

2.2 Web services architecture

Once the OM2M architecture was defined, we could start to think about which services we were going to need for each OM2M Entity. We decided to go with one web service per sensor or actuator, and then offer the same services for every sensor and every actuator.
We will get into more detail about the services methods provided by the Web Services in the Implementation chapter.

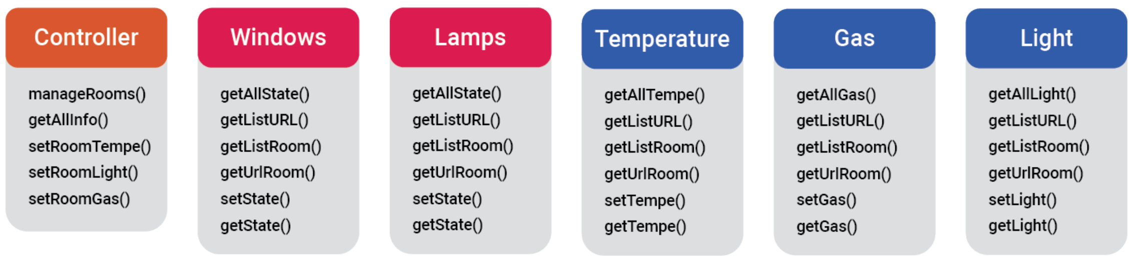


Figure 2.2 – Web Services Architecture diagram

2.3 Implementation architecture

Now that we defined the abstract architecture of the project, we needed to work on the actual code components that we were going to use.
There are two main parts:

- The main Java project, regrouping all the web services along with the main controller service, and a scheduler class that periodically calls the controller. We were aware that it is unrealistic to have all the services in the same project, because we cannot run them independently. However, we have written the code as is the services were independent, never calling functions directly but always with REST methods. We chose to do this because it was very hard to share code on Github with many java projects, as we had many libraries and imports involved in each project.
- The front-end, consisting in a dashboard made with Java Servlet Pages (JSP), and a Javascript managing script.

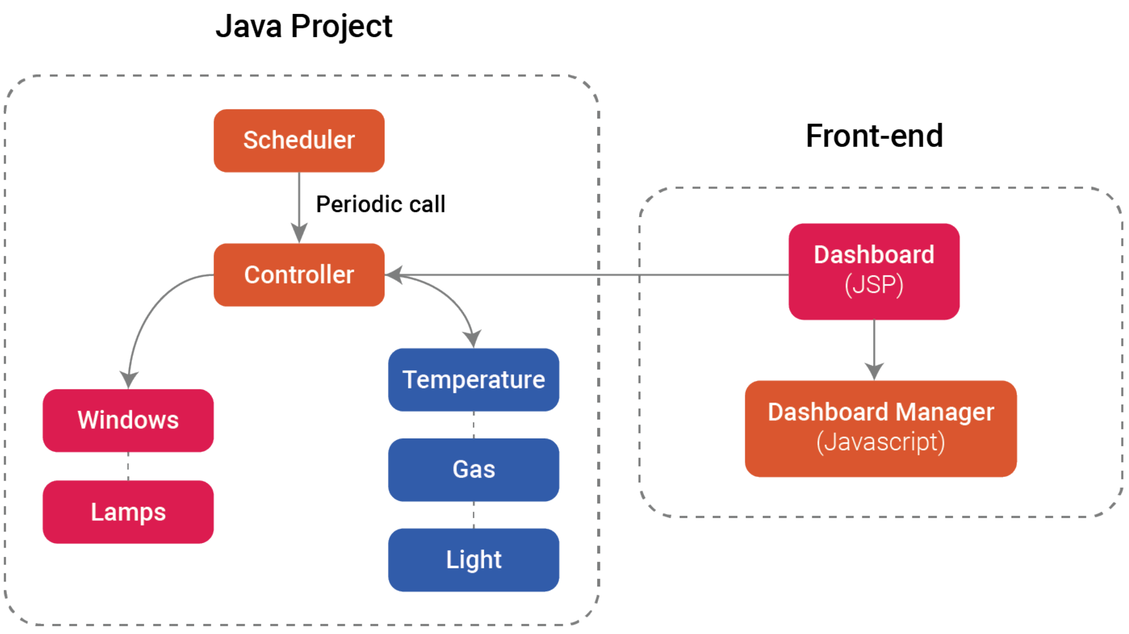


Figure 2.3 – Implementation Architecture diagram

3. Implementation

3.1 Web services

We implemented our web services using the classic Java REST library Jersey. We mainly used the usual methods GET and POST along with the OM2M Java library to create Content Instances. We chose to send the data in JSON format.
We can split the web services we developed in two categories:

- Sensors, with the gas sensor, the temperature sensor and the light intensity sensor.
- Actuators, with the windows and the lamps.

Each sensor or actuator is represented by a package, and a single class containing all the methods. The two categories have some common general methods, but also differences in the services they offer, and we will go over them in the following sections.

3.1.1 General methods

There are three methods that are deployed in the same way on every service, whether it is a sensor or an actuator:

- **getListRoom**: this method returns the IDs of all the rooms in the building.
- **getUrlRoom**: using the result from the previous function, this method returns an array containing the full URL of every room in the building.
- **getListUrl**: looks into every room from the URL given by the previous function, returns all the URLs of the sensors or actuators of the class it is in. For example, in the Windows class, this method returns all windows URLs.

3.1.2 Specific methods

The three following methods are implemented in both the sensor classes and the actuator classes, but work a little differently.

- **setData**: takes as input the URL of a sensor of actuator, and POSTS a OM2M Content Instance in the appropriate Container. In the case of actuators, it will be used by the controller for the scenarios. In the case of sensors, it should not be used because in real life, sensors generate their own data. However, as we are not using actual sensors and instead simulating them, we need to be able to set their values.
- **getData**: takes as input the URL of a sensor of actuator, and returns its value, whether it is a numerical value for a sensor, or a 0/1 boolean value for the actuator states.
- **getOutsideTemp**: Exclusive to the Temperature web service, this method calls an exterior web service on the web to get the outside temperature in Toulouse.
- **getAllData**: this method is called with no argument, and uses all the previous method to return an array of tuples, containing all the information about a sensor class.

  The tuples look like this: [**URL of sensor, Room ID, value**]

With all theses methods, both the controller and the front-end dashboard have everything they need to run our scenarios.

3.2 Controller and scheduler

The controller has two jobs because it serves as a link between the front-end dashboard and the whole service architecture by running the scenarios.

3.2.1 Back-end services

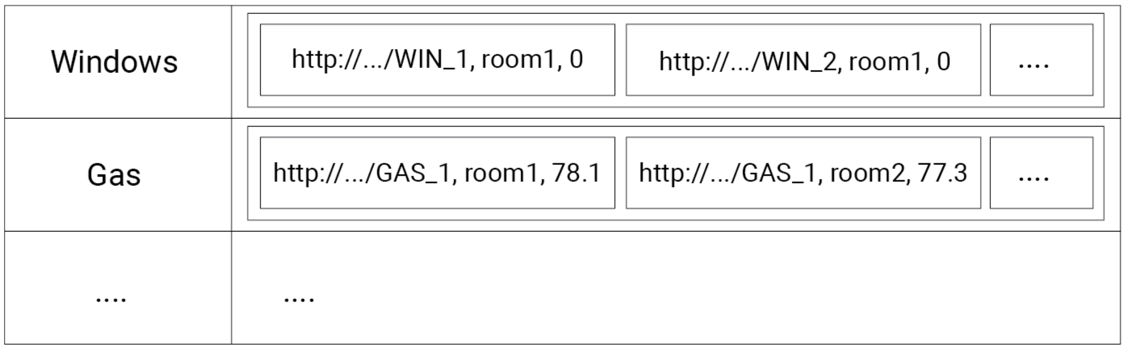
The main method implemented is **manageRooms**. It calls the method manageScenario1 and manageScenario2, and is the main method responsible in ensuring that the scenario conditions are respected.
It is in this method that we call the sensor web services to get their information and run verifications on them. If they validate certain conditions, we will call the set methods of the appropriate actuator services to fit the scenario.

3.2.2 Scheduler

The scheduler is a local Java class, that simply calls the manageRooms method of the controller periodically. It is independant from the rest, and the architecture works on its own without it, with the exception that the scenarios are not run.
To implement it, we used the **TimerTask** Java library to help us create a controllerTask, then run it.

3.2.3 Front-end services

As we said previously, the controller is the only services that acts as a link with the front-end dashboard. That is why it provides a method **getAllInfo**, that returns structured data on all the sensors and actuators of the system at a given point in time.
The data structure is pretty complicated: it is a map with the sensor/actuator type as key, and array of tuples (from the getAllData method) as value.



Windows	http://.../WIN_1, room1, 0	http://.../WIN_2, room1, 0	....
Gas	http://.../GAS_1, room1, 78.1	http://.../GAS_1, room2, 77.3	....
....	....		

Figure 3.1 – Data structure for the getAllInfo method

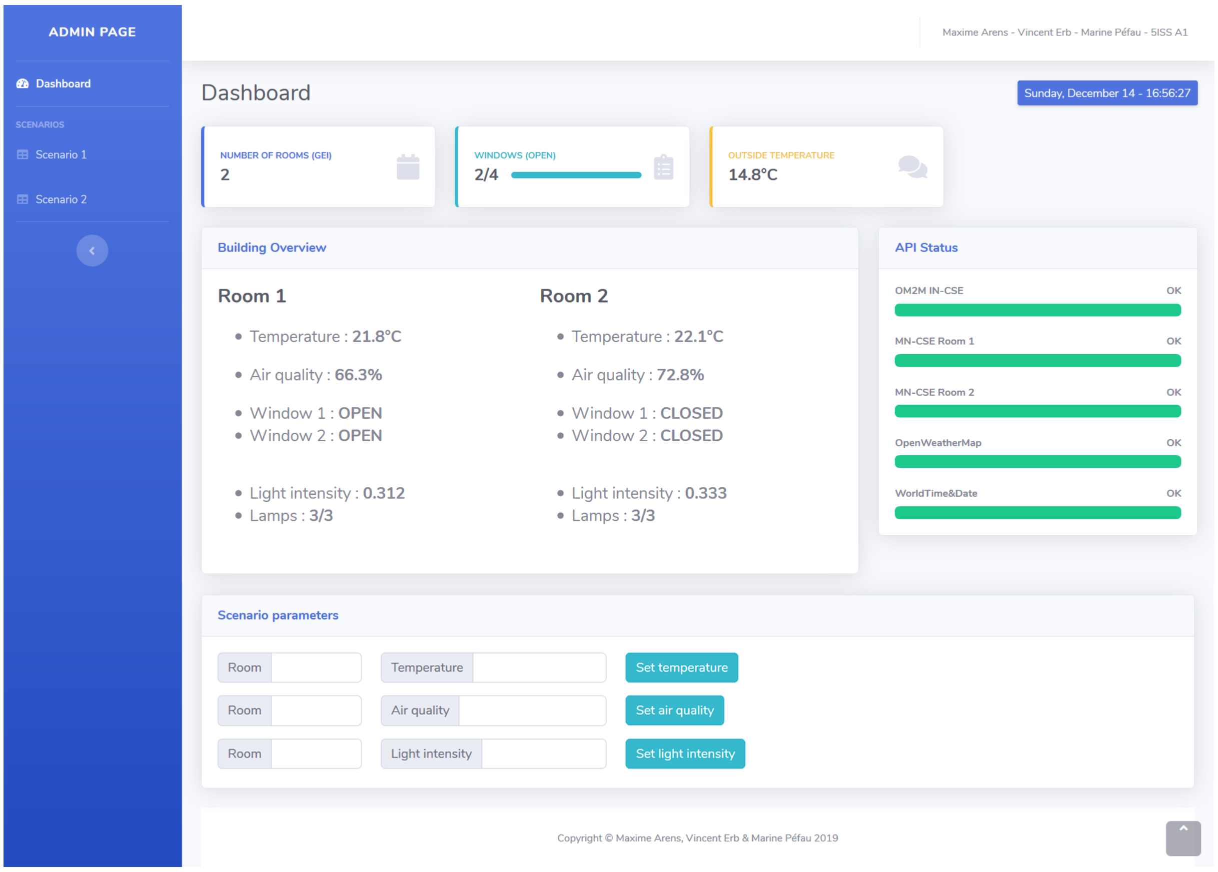
3.3 Front-end dashboard

The dashboard has been made using Java Servlet Pages (JSP), a classic javascript script using GET methods to call the getAllInfo method from the controller. It uses bootstrap and the theme SBAdmin2 for the CSS theming.
You can find more information here:
`https://startbootstrap.com/themes/sb-admin-2/`

4. Results and instructions

4.1 Dashboard

The main HCI (Human-computer interaction) is the dashboard. It is accessible in any browser and displays all the information about the entire system.



Maxime Arens - Vincent Erb - Marine Péfau - 5ISS A1

Dashboard

Sunday, December 14 - 16:56:27

NUMBER OF ROOMS (GEI) 2

WINDOWS (OPEN) 2/4

OUTSIDE TEMPERATURE 14.8°C

Building Overview

Room 1
- Temperature : 21.8°C
- Air quality : 66.3%
- Window 1 : OPEN
- Window 2 : OPEN
- Light intensity : 0.312
- Lamps : 3/3

Room 2
- Temperature : 22.1°C
- Air quality : 72.8%
- Window 1 : CLOSED
- Window 2 : CLOSED
- Light intensity : 0.333
- Lamps : 3/3

API Status

OM2M IN-CSE	OK
MN-CSE Room 1	OK
MN-CSE Room 2	OK
OpenWeatherMap	OK
WorldTime&Date	OK

Scenario parameters

Room	Temperature	Set temperature
Room	Air quality	Set air quality
Room	Light intensity	Set light intensity

Copyright © Maxime Arens, Vincent Erb & Marine Péfau 2019

Figure 4.1 – Screenshot of the full dashboard

On it, you can see multiple things:

- At the top right corner, the current date and time of the system, useful for one of the scenarios.
- At the top, some card with general informations like the number of rooms or number of windows open.
- In the center, the overview of all the sensor and actuators values. This is updated every 10 seconds.
- On the right, sliders verifying that the web services are reachable.
- At the bottom, you can set the value of the sensors directly for the user interface. This is useful to demonstrate that the scenarios are working.
- In the left bar, there are two scenario buttons that display a popover describing the functioning of the scenario.

4.2 Scenarios

We implemented two complex scenarios, involving more than one type of sensor or actuator:

- The first scenario is the following. If the temperature of a room is 5°C lower than the outside temperature, open the window to bring in the heat. If it is 5°C hotter, close the windows to keep the heat. If air quality is below 70 percent, open the windows regardless. This scenario involves the temperature sensor, the gas sensor and the windows actuator. It was challenging to ensure that the gas condition is prioritized over the temperature condition.
- The second scenario is the following. If the light sensor value drops below 0.4 in a room, turn on all the lights of the room. If it's above this value, turn them on. After 8PM and before 8AM, close every window and turn off every lamp. This scenario involves the light sensor, the current value of time, the windows actuator and the lamp actuator. The biggest challenge was using time, we finallly decided to use the actual machine time, even though it meant that we couldn't demonstrate this part of the scenario depending on the time of the test.

4.3 Instructions to run the project

4.3.1 OM2M

In order to generate the OM2M tree to work with our project, you have to run the the script **script.bat** in the OM2M folder. It only works on windows though, so if you are on linux, you can just launch the in script in the in folder and the mn scripts in the mn folders directly.
Then, to fill the OM2M nodes, you have to use **Postman** and import the collection **First Scenario.postman_collection.json**. It will create AEs CNTs, and a CIN for every AE.

4.3.2 Java

To run the web project, you simply have to import in eclipse the Java project **RoomManagement**. Then, run it on a server (Tomcat) and everything is online. You can go to the dashboard which is situated at the URL: `http://localhost:8484/RoomManagement/dashboard`

In order to run the scenarios, you have to start the scheduler. To do so, just find the class **Scheduler.java** in the project, and run it as a java application.

For troubleshooting about running the project, feel free to contact us at this address: erb@etud.insa-toulouse.fr

5. Project management

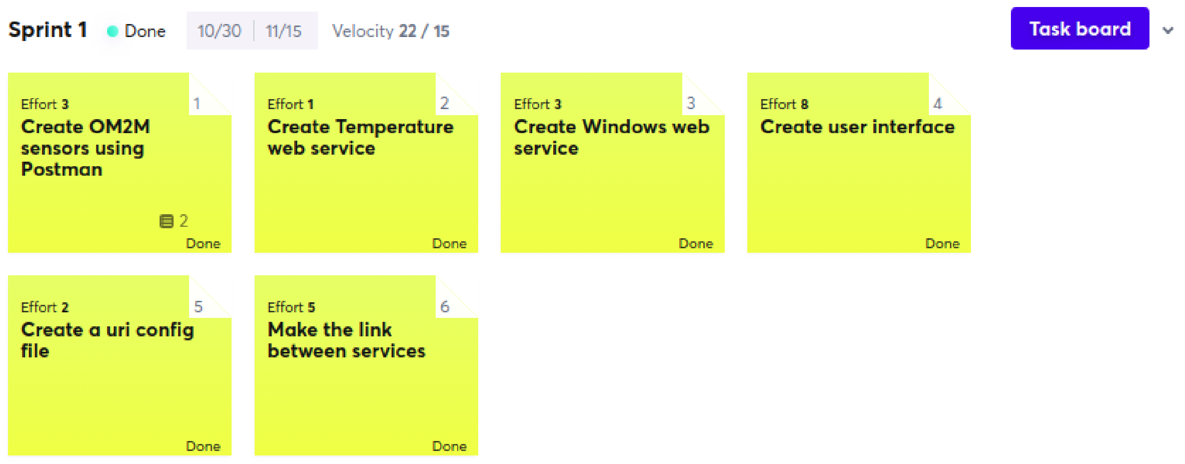
5.1 Management method choices

To manage this project, we used iceScrum, that allows us to use the AGILE method to organize the team. This method relies on the collaboration and the autonomy of the team. With this method, the first thing to do is to identify the time we have for our project and divide it in sprints. The idea is to have objectives for each sprints and to achieve them at the end of each one. For our project, we decided to divide our work in three sprints. Indeed, we saw three main parts : the first one being to create a first, simple and working scenario, the second one to create others and manage them and finally finalize the project and do the report.

5.2 Sprints review

What is interesting with AGILE method is to separate the work in different sub-tasks that are more independent. That allows us to see the different steps we have to follow in order to complete our project. To do that, we defined different tasks for every sprint in iceScrum and we identify the complexity of each one with the parameter "effort". We create then sub-tasks that we can share between the collaborators.

5.2.1 Sprint 1



Sprint 1 ● Done 10/30 | 11/15 Velocity 22 / 15

Task board

Effort 3 — 1 — Create OM2M sensors using Postman — 2 — Done

Effort 1 — 2 — Create Temperature web service — Done

Effort 3 — 3 — Create Windows web service — Done

Effort 8 — 4 — Create user interface — Done

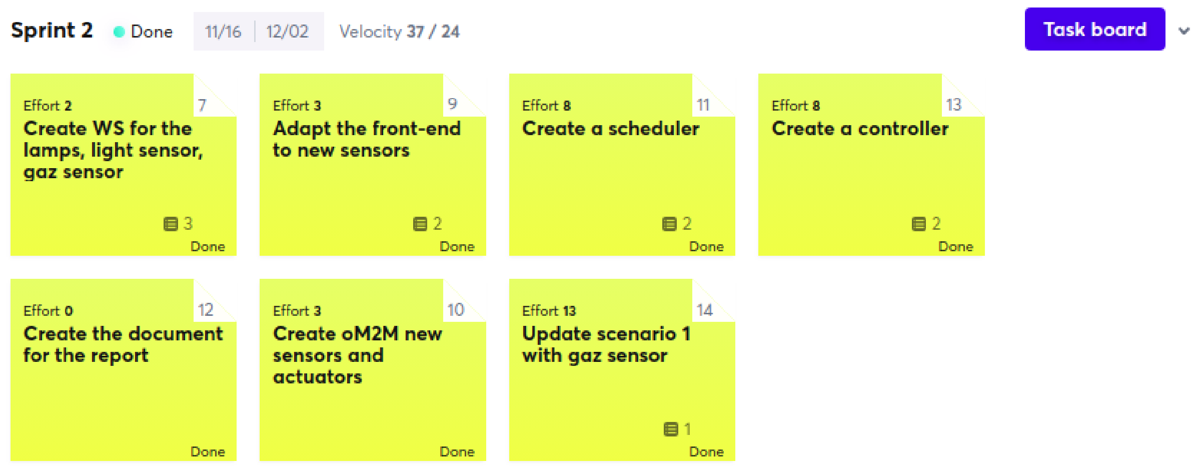
Effort 2 — 5 — Create a uri config file — Done

Effort 5 — 6 — Make the link between services — Done

Figure 5.1 – Backlog of sprint 1 : Tasks and efforts

The first sprint was maybe the more complicated because we had to define our architecture and understand all the technical concepts of the project. We created simple tasks in the backlog and we made sure that there were done at the end of the sprint. Every person of the project took care of one of the task. However, to use this tool more efficiently, we maybe had to be more precise and more clear on the different tasks to do and that is what we improved for the second sprint.

5.2.2 Sprint 2



Sprint 2 Done 11/16 12/02 Velocity 37 / 24

Effort 2 — 7 — Create WS for the lamps, light sensor, gaz sensor — 3 — Done

Effort 3 — 9 — Adapt the front-end to new sensors — 2 — Done

Effort 8 — 11 — Create a scheduler — 2 — Done

Effort 8 — 13 — Create a controller — 2 — Done

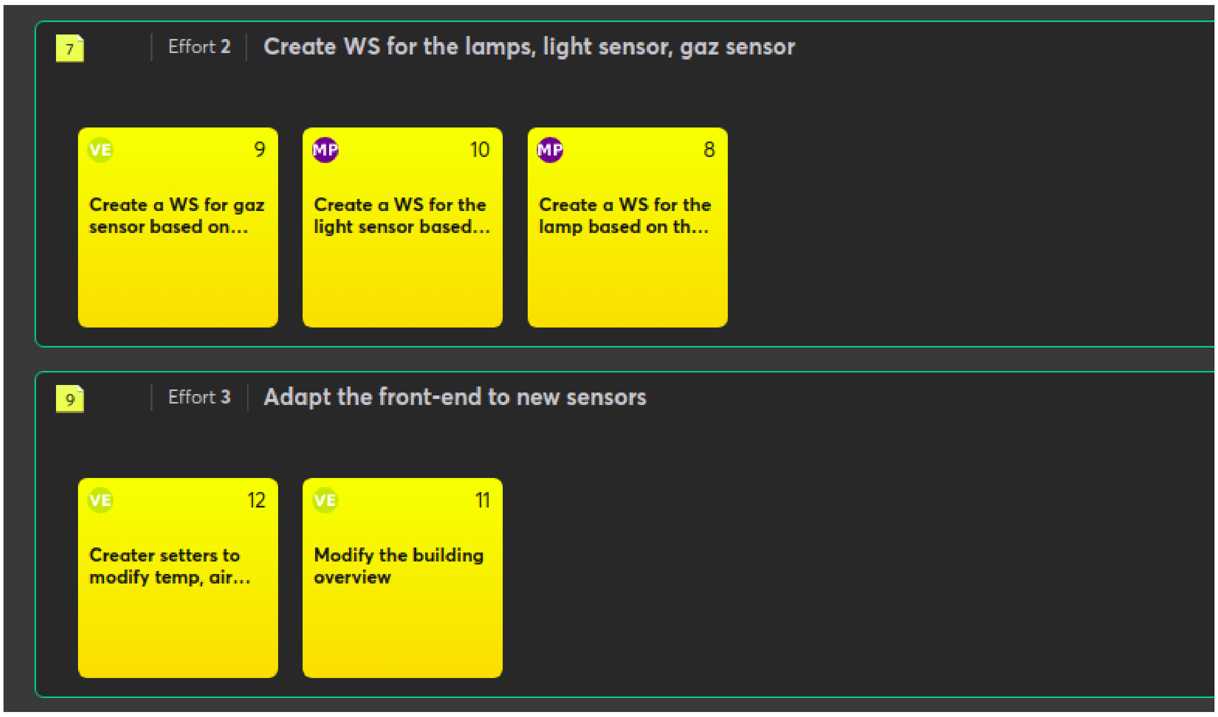
Effort 0 — 12 — Create the document for the report — Done

Effort 3 — 10 — Create oM2M new sensors and actuators — Done

Effort 13 — 14 — Update scenario 1 with gaz sensor — 1 — Done

Figure 5.2 – Backlog of sprint 2 : Tasks and efforts

For this second sprint, we had a clearer understanding of the project as we were developing it. We created adapted backlogs and we precised them by different tasks that we shared between us. During this sprint, it was important to communicate and give updates to the other members of the group because the tasks were linked one to each other. It was also important to work together at a same rhythm so that every member of the group could develop his part without being blocked.



7 — Effort 2 — Create WS for the lamps, light sensor, gaz sensor

VE 9 — Create a WS for gaz sensor based on…

MP 10 — Create a WS for the light sensor based…

MP 8 — Create a WS for the lamp based on th…

9 — Effort 3 — Adapt the front-end to new sensors

VE 12 — Creater setters to modify temp, air…

VE 11 — Modify the building overview

Figure 5.3 – Extract of task board of Sprint 2

5.2.3 Sprint 3

Sprint 3 was the end of our project and the tasks mainly concerned the realisation of the report. We also wanted to correct some details on the dashboard and on the controller.

5.3 Key takeaways

During this project, we separated the tasks in three main parts. Marine was in charge of creating the different web services we needed, Maxime created the controller and scheduler for the different scenarios and Vincent was in charge of the front-end and all the link with the back-end. With that method, we discovered new needs that we did not anticipated at the beginning and that allowed us to not have too much delays on what we wanted to do. Every person in our group was implicated and did his part, according to what he could do in terms of skills and what was more interesting for him regarding his background.

6. Conclusion and feedback

This project was a good way for us to get better with multiple skills. First, on a technical standpoint, it allowed us to use the concepts we learned in the Service Oriented Architecture class on a big project, where we could really see the perks of using them. We also were able to use the OM2M platform with a real complex project, which we hadn't had the chance to do before.
On a project management standpoint, we were able to utilize the Agile method on a pretty long project of about three months, so we could really use sprints and reviews to track our progress and adjust the development of the project.

For some feedback about the project and the course in general, we felt that maybe the TDs didn't prepare us enough for the technicality of the project, especially for AE or GP people with less advanced programming skills. Maybe focusing more on one technology instead of doing a sweep of everything that exists (like BPEL which is a pretty dated technology) would be better.
Regarding the project management,we thought that Agile method and Scrum are tools that are more interesting for a project that involves more collaborators. It seems less interesting to use it for a work in pairs as it was meant at first.