Service Oriented Architecture Report

Mathilde Cornille | Clément Delobel | Vincent Laurens 5 ISS A1

Table of Contents

Introduction

Context

First, we will defined the scenario we chose to implement. Secondly, we present the architecture of the project and we explain how the different services are communicating. Finally, you will learn more about how we divided the work thanks to an Agile method in a third part.

Project overview

For this SOA Project, we decided to work on three different scenarios:

- The first scenario address security issues in the building : we want to trigger an alarm if a presence is detected in the room between 10pm and 7am. To do so, we just need an alarm and a presence sensor.
- The aim of the second scenario is to control air temperature. If the inside temperature is between 18 and 27°C and higher than the outside temperature we want to open the windows, so the room can be ventilated. This scenario uses a temperature sensor for the inside temperature and a window position sensor. The outside temperature is collected thanks to a weather API : name of the API.
- The last scenario is an historic which store all new values collected by differents sensors.

I. Project architecture

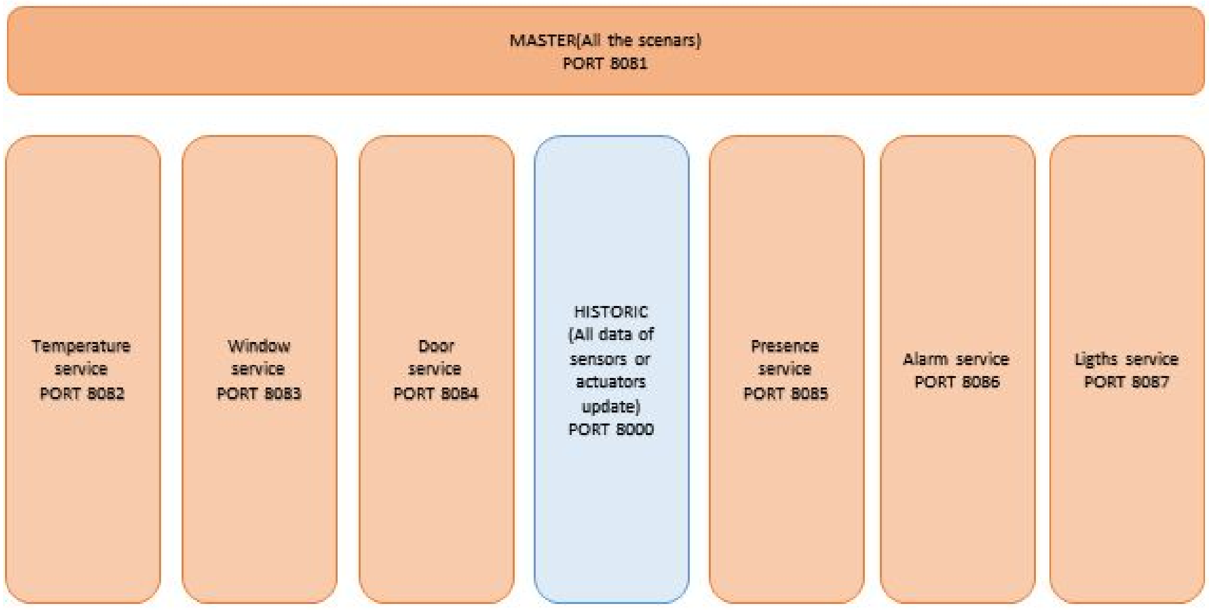


Figure 1 : Services architecture

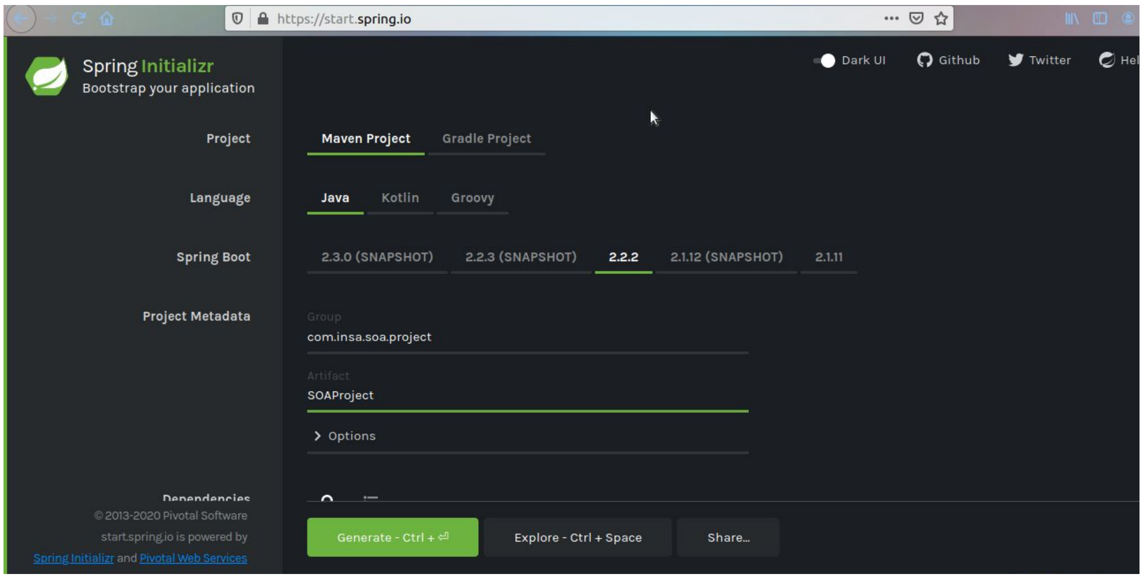
We developed seven services orchestrated by a master. The first operation we did was to map an application port to each services. Indeed, this port added to the local IP address of computer will be part of URI which is the address to interrogate service.

We choose to increment port of oneM2M (8080) each time, as described on figure 1. Historic is totally different because it constitute a monitoring service of our application and we would like to isolate it from other, especially if we anticipate the scale up/down of our application.

<u>Creation of the microservices using Spring Boot</u>

At we beginning, we wanted to create a REST API. But we know it can be time consuming to do it from scratch as you need to set up the apache server and all its dependencies with correct parameters and software version to avoid any unexpected behaviour. So, we decided some teacher advice and with the background of the lasts lab exercices to use Spring boot. What is Spring boot?

Spring boot is a bootstrap service which generate all dependencies for a new java project. We can either create a JAVA project, a maven project or a gradle project. When we have some experience on that kind of project. So, some tools try to help us define all dependencies needed for developing an application and Spring boot is one of them.



Project: Maven Project

Language: Java

Spring Boot: 2.2.2

Project Metadata

Group: com.insa.soa.project

Artifact: SOAProject

Figure 2 : Spring Boot configuration

source : https://start.spring.io/

<u>**Implementation with Eclipse.**</u>

Once a Spring boot zip file is generated by using the Spring Boot website, we simply have to import it on Eclipse as a maven project. Then, we can modify some parameter in the application.properties file to change the port on which the service will be deployed. As we want to create several services, the port definition is primordial. Then, to deploy a service, you only have to run it as a Java application, the bootstrap services configured it correctly to avoid any dependency issue.

<u>**Architecture of a microservice**</u>

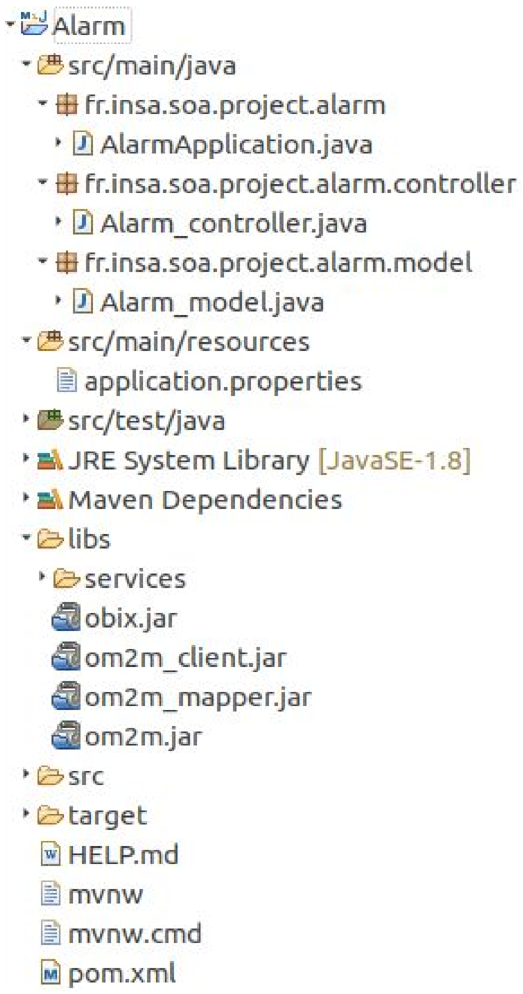
As each service is done through a microservice, we chose a standard format for each one. This is composed of three compulsory packages based on Model, View, Controller as described below. Notice that we don't need a View package in our case because the Java service don't process any print on a UI. Our microservice is also composed of java libraries generated from JAVA obix project and onem2m_client and mapper projects.
Common packages:

- Root package, named according to the name of the service. It contains the Application.java file, generated by Spring Boot, that is used when starting the service.
- Model package. It contains the java class of the service and defines all attributes and methods that characterize the service. It is used in the controller file.
- Controller package. It contains the Resource file that will establish the service to the given address (using @GetMapping or @POSTMapping function). It can be considered as the intelligence of the service as it will instantiate the different classes defined in the other packages, use their function and attributes according to the chosen use case.

For each microservice, we create libs of oneM2M mapper and client that we developed during labs and also a lib for obix object. We import these libraries in order each of our service to be deployed stand-alone.

- Obix : standard for RESTful Web Services-based interfaces to building control systems. oBIX is about reading and writing data over a network of devices using XML and URIs, within a framework specifically designed for building automation.
- oneM2M_client : library composed of client class developed during labs
- oneM2M_mapper : library composed of mapper class developed during labs

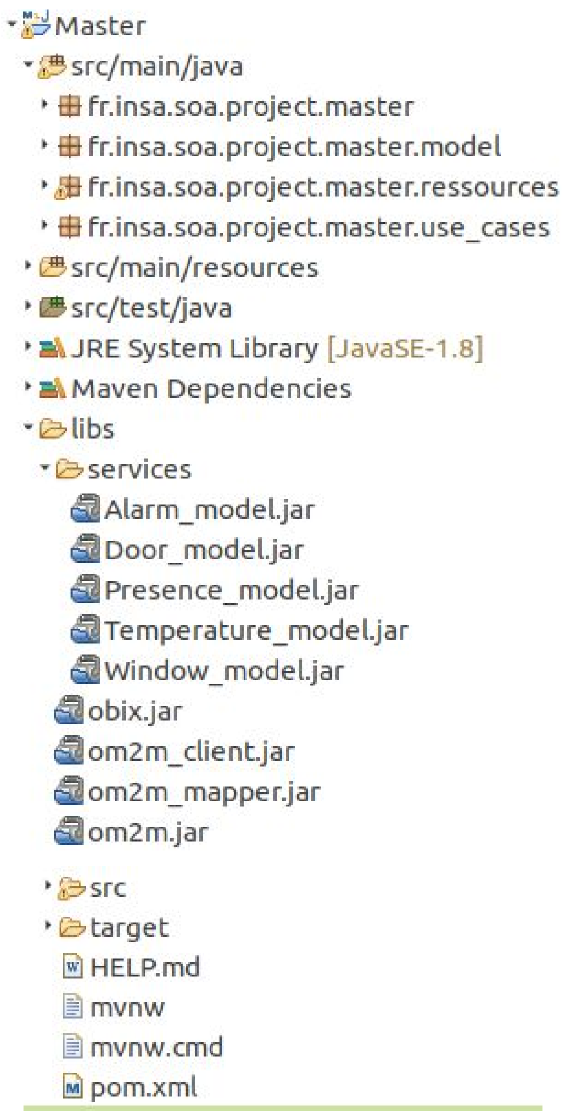
We can have a picture of what we describe by observing next figure which is an eclipse treemap snapshot.



```
Alarm
  src/main/java
    fr.insa.soa.project.alarm
      AlarmApplication.java
    fr.insa.soa.project.alarm.controller
      Alarm_controller.java
    fr.insa.soa.project.alarm.model
      Alarm_model.java
  src/main/resources
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  libs
    services
    obix.jar
    om2m_client.jar
    om2m_mapper.jar
    om2m.jar
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

Figure 3 : Eclipse treemap

Only the Master service as a quite different architecture as it does not have any sensor or actuator and contains the different use cases in the ressources package. This service will link all the others services togethers. By consequence, in order the master to be deployed stand-alone like all other services, we import libs oneM2M_client, oneM2M_mapper, obix, then we create in libs a folder which contains libs generated with model class of each services. Why? Because the master need to create an instance of each service to send him a rest get or post request.



```
Master
  src/main/java
    fr.insa.soa.project.master
    fr.insa.soa.project.master.model
    fr.insa.soa.project.master.ressources
    fr.insa.soa.project.master.use_cases
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  libs
    services
      Alarm_model.jar
      Door_model.jar
      Presence_model.jar
      Temperature_model.jar
      Window_model.jar
    obix.jar
    om2m_client.jar
    om2m_mapper.jar
    om2m.jar
    src
    target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

Figure 3 : Treemap of master service

<u>**Creation and Interaction with OM2M architecture.**</u>

To begin, we decided to generate oneM2M architecture with Postman web client. Then, to be more efficient, we used a bash script sending curl commands. We develop mechanism in our backend application to read value or status of the last content instance (CIN) for each sensor and actuator container. These CIN contain wanted data. We simulated the first data for each sensor and actuator. After, if the scenario leads to a change of value, we send an HTTP POST to create a new CIN.

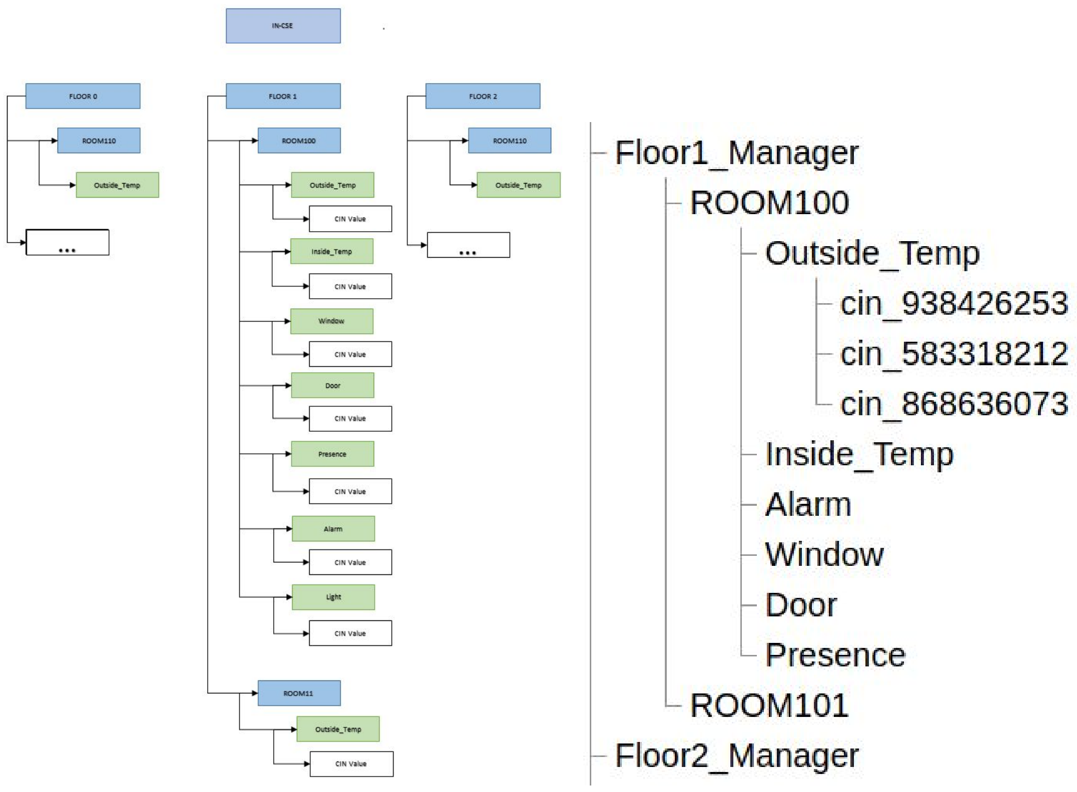


Figure 4 : oneM2M architecture

<u>Implementation of an interface with Node-Red</u>

To have a graphic user interface (GUI) to interact with the different use cases and create some simulations, we decided to implement a Node-Red application. As the services are available at a given address and port, we only have to use a HTTP GET block to retrieve the data offered by the service.

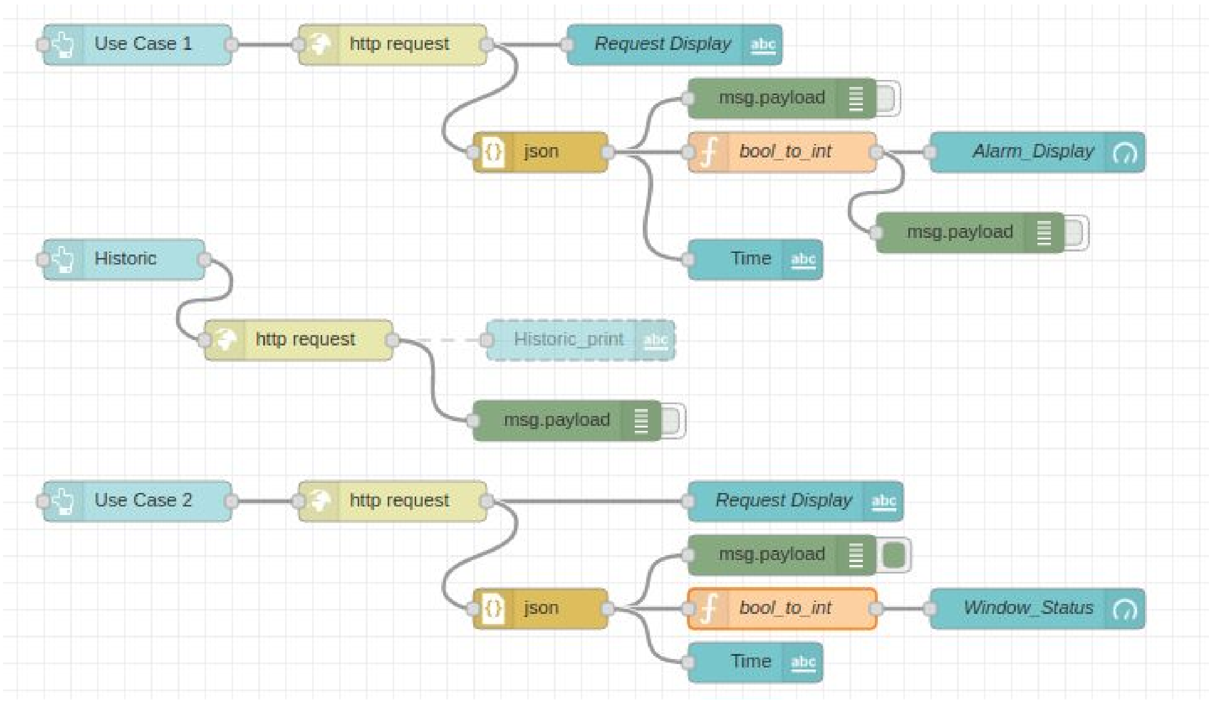


Figure 5 : Node Red interface to test API

Then depending on the type of data that was retrieved, we parse it, store it, use it with graphical elements and/or simply display it. We used given interface node such as gauges or graphs to have a truly user friendly interface on the Node-Red dashboard. For example, we used a gauge with the boolean value of alarm status to know whether the alarm is turned on or not: gauge filled in red or in white.

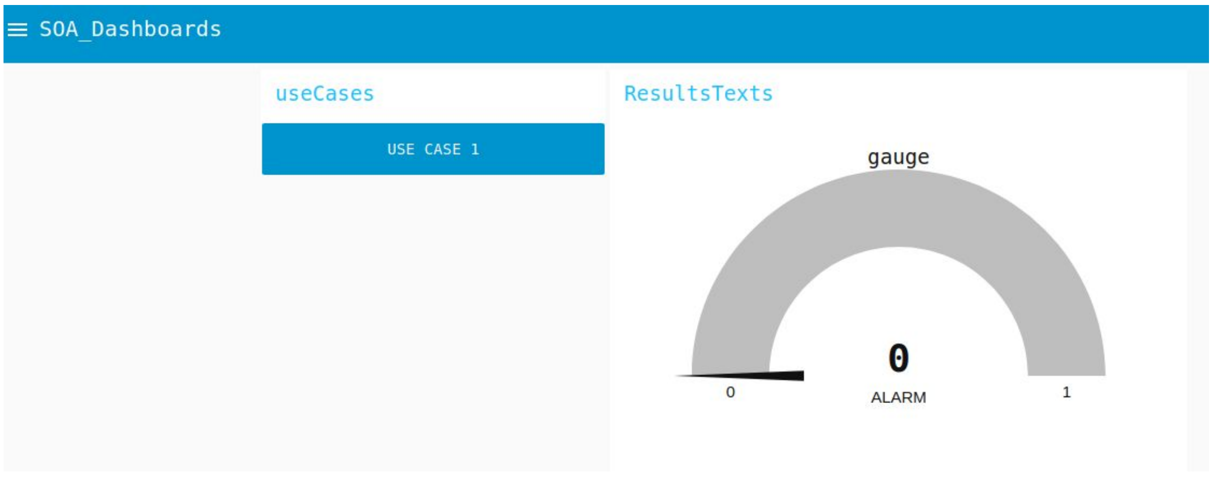
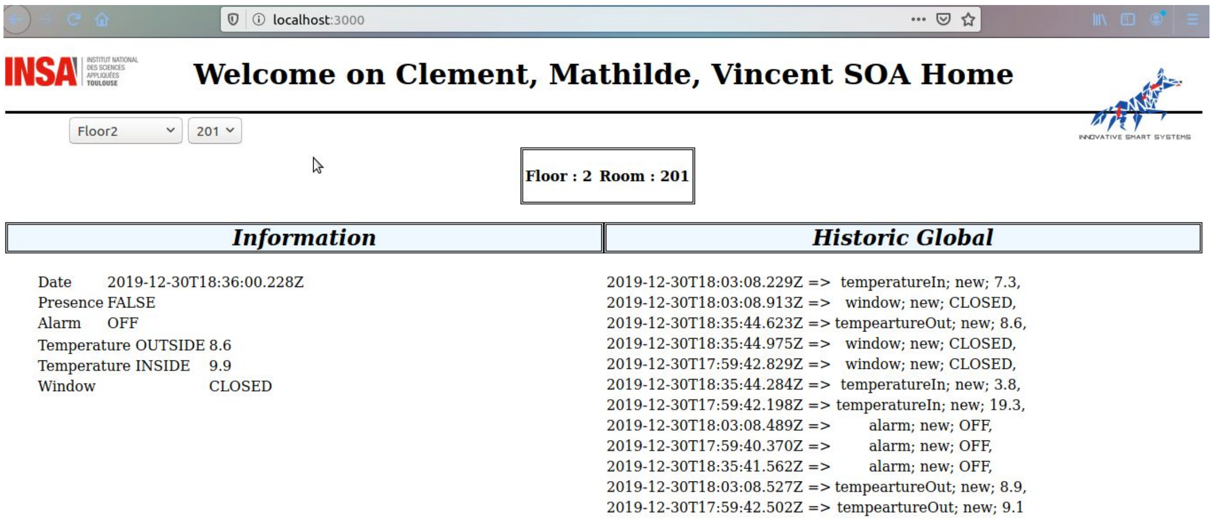


Figure 6 : Dashboard

Even if this Node-Red interface was easy and fast to implement, it was really limited in term of freedom of development, adaptability and personalization. That's why we created a second interface using simple web languages as html/CSS/JS.

<u>Second interface using Html/CSS/JS</u>

This interface allow to show information of sensors and actuator of a room on a floor. Currently, the management of rooms and floors is static because the architecture is fixed as described on the scheme of figure 2. Furthermore, the architecture oneM2M is generated by a script which allow to create always the same architecture. So we didn't focus on the development of an automatic and auto-filled menu. To do so, we should modify the API to manage rooms. For the tested we generate two floor and one room on each floor.



```
Date        2019-12-30T18:36:00.228Z
Presence FALSE
Alarm       OFF
Temperature OUTSIDE 8.6
Temperature INSIDE    9.9
Window                       CLOSED
```

```
2019-12-30T18:03:08.229Z =>  temperatureIn; new; 7.3,
2019-12-30T18:03:08.913Z =>   window; new; CLOSED,
2019-12-30T18:35:44.623Z => tempeartureOut; new; 8.6,
2019-12-30T18:35:44.975Z =>   window; new; CLOSED,
2019-12-30T17:59:42.829Z =>   window; new; CLOSED,
2019-12-30T18:35:44.284Z =>  temperatureIn; new; 3.8,
2019-12-30T17:59:42.198Z => temperatureIn; new; 19.3,
2019-12-30T18:03:08.489Z =>        alarm; new; OFF,
2019-12-30T17:59:40.370Z =>        alarm; new; OFF,
2019-12-30T18:35:41.562Z =>        alarm; new; OFF,
2019-12-30T18:03:08.527Z => tempeartureOut; new; 8.9,
2019-12-30T17:59:42.502Z => tempeartureOut; new; 9.1
```

Figure 7 : Web interface

II. Use Case 1

<u>Description of the use case</u>

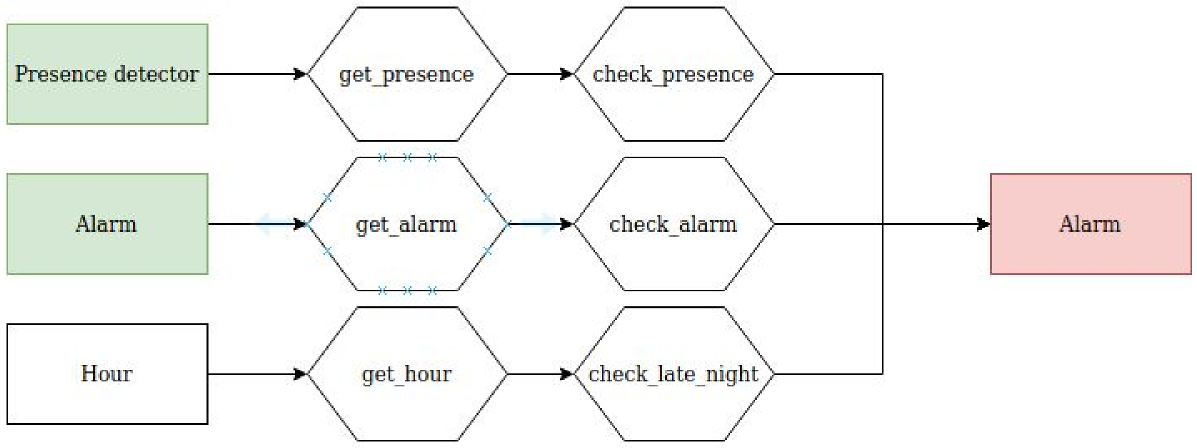


Figure 8 : Description of use case 1 microservices interactions

Notice that Alarm and Presence Detector (Presence) are services rather than Hour is a function.

The objective of the first use case is to detect an abnormal intrusion in the building. It means that we check, using presence sensors, if a person is detected in a room out of the normal hour of working. More precisely, if the presence sensor detects something between 11pm and 7pm, it will send a notification to the alarm manager and trigger it.

Different services needed

To implement this use case, we need different services. The first one is the one that manage the presence sensor. It is used to retrieve the data of the sensor on OM2M and to post the data and the information of the sensor on a specific address and port number. The second service it the one to manage the alarm. This service is used to provide the actual status of the alarm. In this specific case, it is to avoid triggering the alarm if it is already turned on. This could also be used in other services or other ways, to call the police for example.

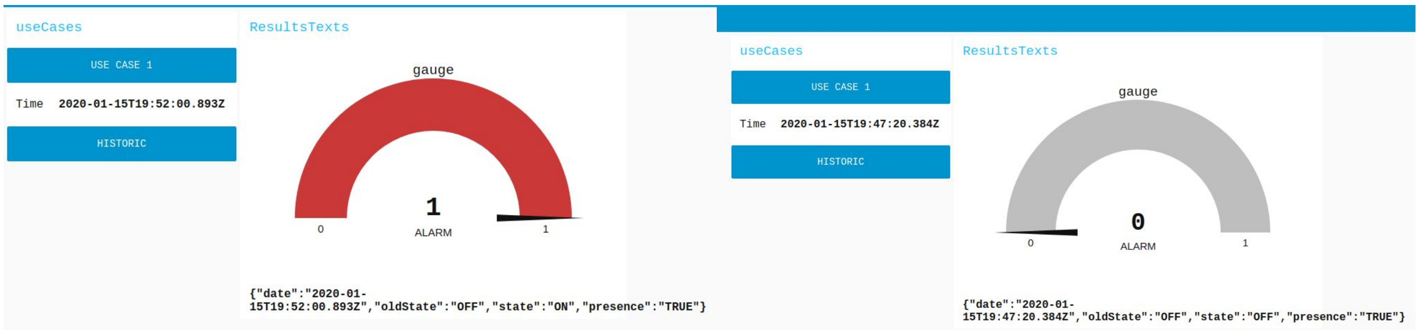
The last service is the master one, it is the intelligence of the architecture. It will get both value of the presence and the alarm service and, according to the actual hour, it will send a "true" value to the OM2M architecture that pilot the alarm to turn it on or do nothing.

DIAGRAMME DE FONCTIONNEMENT DU SERVICE

Figure 9 : ???

Example of a simulation

For example, is a presence is detected at 2:53am and the alarm activation status is "false", the master will post a data containing value "true" to the OM2M architecture.



Time 2020-01-15T19:52:00.893Z

```
{"date":"2020-01-
15T19:52:00.893Z","oldState":"OFF","state":"ON","presence":"TRUE"}
```

Time 2020-01-15T19:47:20.384Z

```
{"date":"2020-01-
15T19:47:20.384Z","oldState":"OFF","state":"OFF","presence":"TRUE"}
```

Figure 10 : Dashboard for use case 1

In the first figure we can see that we are not between 10PM to 7AM and we have someone in the room so the alarm remain OFF.

In the second case, we modify API for the test to modify hour range where the presence is forbidden in the room. We define 7PM because we would like to show our use case is functioning and we see that it work because the new state is ON and the Jauge become red.

Technology and tools

To run this use case, we used several tools and libraries such as *OM2M client* and *obix.* Of course we also used a microservice architecture and tools such as Node-Red but this tools are common to all use cases and are presented in the architecture part. The OM2M client is used to manage the architecture with a java application. With functions like *client.retrieve* we can get the data from OM2M.

As this function returns a json string that contains an Obix object, we parsed it and used the Obix library to extract the wanted data. Once the data is extracted, the values are given as attribute of the service class.

III. Use Case 2

Description of the use case

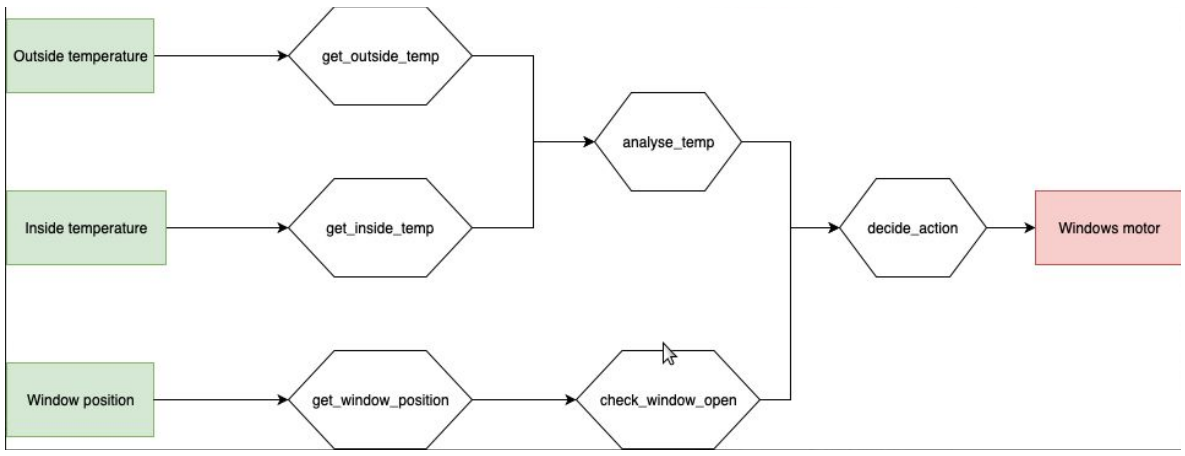


Figure 11 : Description of use case 2 microservices interactions

In this use case we retrieve outside temperature and inside temperature from sensors and then we compare values. If the outside temperature is highest than inside temperature and the window is closed so we open it. At contrary, if outside temperature is lower than inside temperature and the window is open we close it.

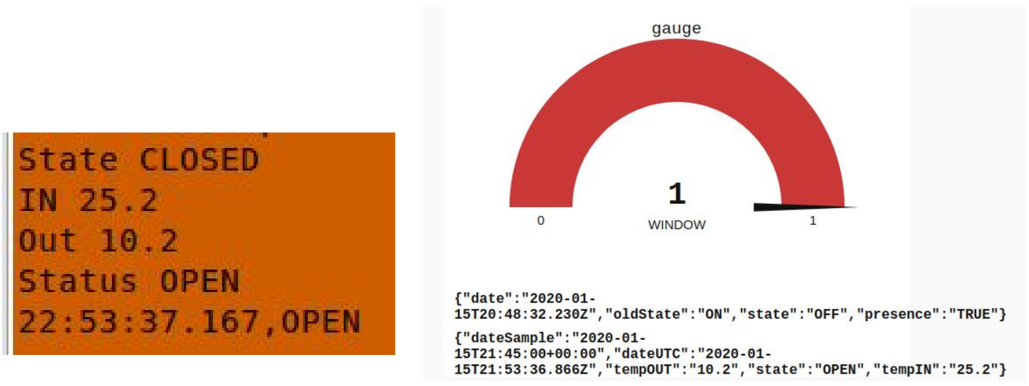
Different services needed

For this use case we have implement the analysis on master service. The master service create an instance of temperature service: Outside_Temp which will retrieve outside temperature from meteo toulouse website and we used data from station of Carmes (https://data.toulouse-metropole.fr/api/records/1.0/search/?dataset=28-station-meteo-carme

s&lang=fr&sort=heure_de_paris&fbclid=IwAR3mFob-pcISDNpt1-nnUAtJs8IEZmnsnu8VbkjXwqK_yzAN74_diGnPzKc) and which will create an instance "Inside_Temp" of temperature service which will randomly generate a inside temperature.
Then the maser service will compare value retrieved by the two instances of sensors and change state of the actuator Window if it is necessary as describe in the previous section.

Example of a simulation

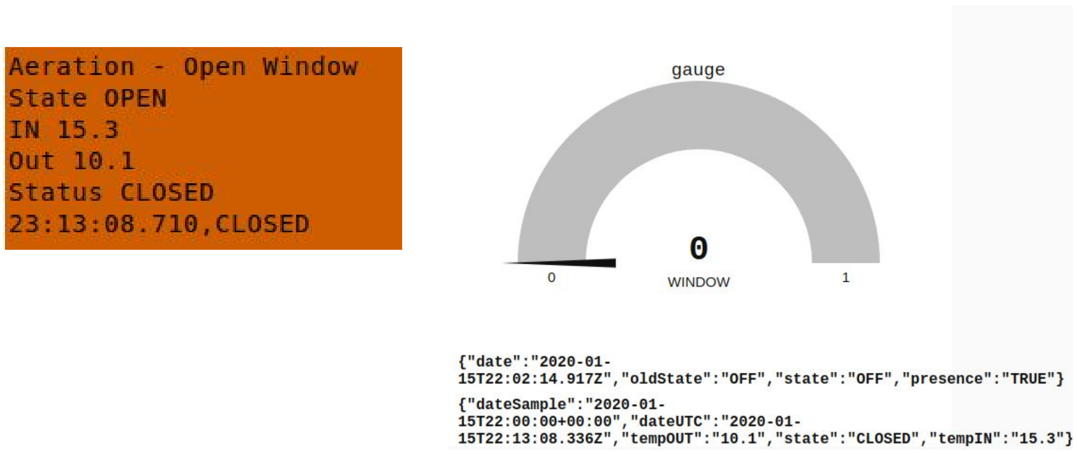
In an example we launch we can see that the window remains closed when outside temperature is lower than inside temperature.



```
State CLOSED
IN 25.2
Out 10.2
Status OPEN
22:53:37.167,OPEN
```

```
{"date":"2020-01-15T20:48:32.230Z","oldState":"ON","state":"OFF","presence":"TRUE"}
{"dateSample":"2020-01-15T21:45:00+00:00","dateUTC":"2020-01-15T21:53:36.866Z","tempOUT":"10.2","state":"OPEN","tempIN":"25.2"}
```

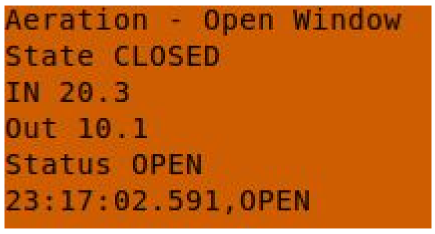
Figure : Test opening window



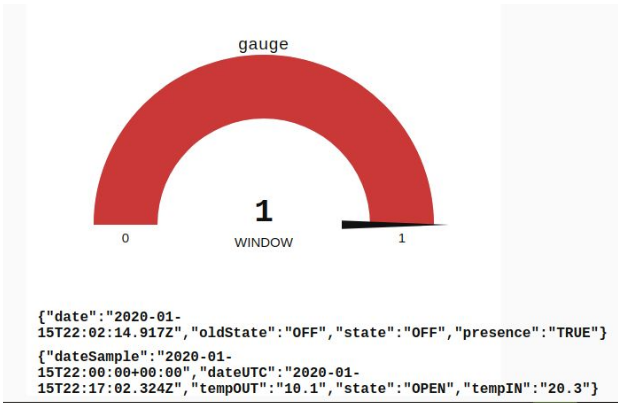
```
Aeration - Open Window
State OPEN
IN 15.3
Out 10.1
Status CLOSED
23:13:08.710,CLOSED
```

```
{"date":"2020-01-15T22:02:14.917Z","oldState":"OFF","state":"OFF","presence":"TRUE"}
{"dateSample":"2020-01-15T22:00:00+00:00","dateUTC":"2020-01-15T22:13:08.336Z","tempOUT":"10.1","state":"CLOSED","tempIN":"15.3"}
```

Figure 12: Test open window we close because the temperature inside is not between 18°C and 27°C

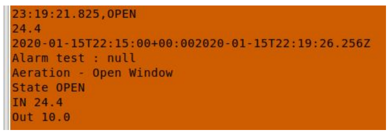


```
Aeration - Open Window
State CLOSED
IN 20.3
Out 10.1
Status OPEN
23:17:02.591,OPEN
```

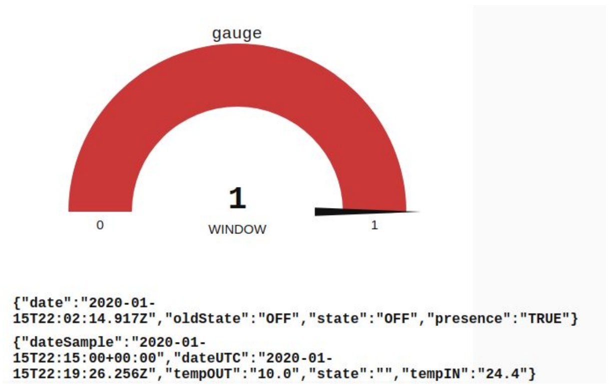


```
{"date":"2020-01-15T22:02:14.917Z","oldState":"OFF","state":"OFF","presence":"TRUE"}
{"dateSample":"2020-01-15T22:00:00+00:00","dateUTC":"2020-01-15T22:17:02.324Z","tempOUT":"10.1","state":"OPEN","tempIN":"20.3"}
```

Figure 13: Window closed we open because the temperature inside is between 18°C and 27°C and the inside temperature is higher than the outside temperature.



```
23:19:21.825,OPEN
24.4
2020-01-15T22:15:00+00:002020-01-15T22:19:26.256Z
Alarm test : null
Aeration - Open Window
State OPEN
IN 24.4
Out 10.0
```



```
{"date":"2020-01-15T22:02:14.917Z","oldState":"OFF","state":"OFF","presence":"TRUE"}
{"dateSample":"2020-01-15T22:15:00+00:00","dateUTC":"2020-01-15T22:19:26.256Z","tempOUT":"10.0","state":"","tempIN":"24.4"}
```

Figure 14: Window is open but remain open because the inside temperature is between 18°C and 27°C and the inside temperature is higher than outside so we can get fresh air.

<u>Technology and tools</u>

In this use case we use obix to format xml object in order onem2m functions (create and mapper) can write data on onem2m content instance using URI of the ressource.

IV. Historic

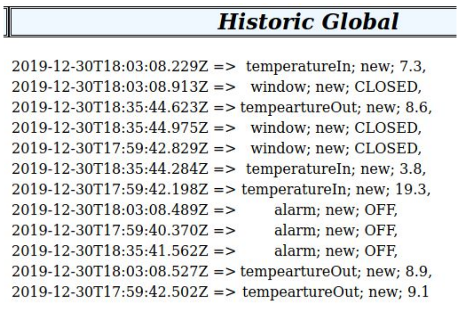
<u>Description of the use case</u>

This service is deployed in master in uses cases. Indeed, when a service is called to add data on oneM2M infrastructure, it will call historic methode POST which will generate a log adding time, information on the action nature realized and service which call historic.Then the log generated is added to a log object table. Then when we call “localhost:8000/historic” URL the api will generate a get to the service historic which will return a JSONObject containing a JSONArray with all the logs collected.
An improvement of this service will be to make an historic filtered by floor and even by room.

Different services needed

This service is independent from the other services because it’s unique role is to monitor all the actions done by all services.

Example of a simulation



Historic Global

```
2019-12-30T18:03:08.229Z =>  temperatureIn; new; 7.3,
2019-12-30T18:03:08.913Z =>   window; new; CLOSED,
2019-12-30T18:35:44.623Z => tempeartureOut; new; 8.6,
2019-12-30T18:35:44.975Z =>   window; new; CLOSED,
2019-12-30T17:59:42.829Z =>   window; new; CLOSED,
2019-12-30T18:35:44.284Z =>  temperatureIn; new; 3.8,
2019-12-30T17:59:42.198Z => temperatureIn; new; 19.3,
2019-12-30T18:03:08.489Z =>        alarm; new; OFF,
2019-12-30T17:59:40.370Z =>        alarm; new; OFF,
2019-12-30T18:35:41.562Z =>        alarm; new; OFF,
2019-12-30T18:03:08.527Z => tempeartureOut; new; 8.9,
2019-12-30T17:59:42.502Z => tempeartureOut; new; 9.1
```

We can see that some action have been done and are stored by Historic service and return the list of logs.

Technology and tools

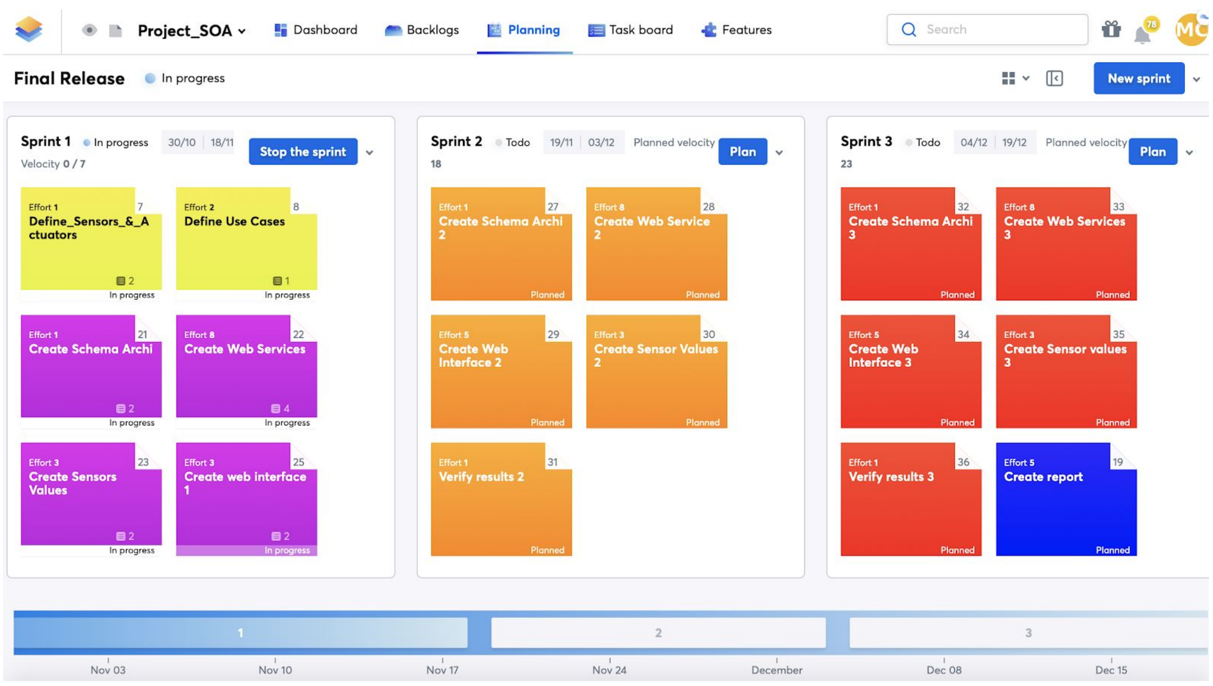
This service use mainly package org.JSON to parse data in JSON. This state is important because on the front-end is easier to parse a JSONObject and print the data on the html interface.

V. Project organisation

IceScrum

For the management of this project we worked with an Agile method. To do so, we used IceScrum. IceScrum is an online tool which allows us to define precisely our objectives. We can define stories and features and planify them in differents sprints. Each task defined is attributed to one (or more) member of the team.

Thanks to IceScrum we can define a planning with precise deadlines, and see the evolution of the project.



Final Release · In progress

Sprint 1 · In progress · 30/10 | 18/11 · Velocity 0 / 7
- Effort 1 · 7 · Define_Sensors_&_Actuators · 2 · In progress
- Effort 2 · 8 · Define Use Cases · 1 · In progress
- Effort 1 · 21 · Create Schema Archi · 2 · In progress
- Effort 8 · 22 · Create Web Services · 4 · In progress
- Effort 3 · 23 · Create Sensors Values · 2 · In progress
- Effort 3 · 25 · Create web interface 1 · 2 · In progress

Sprint 2 · Todo · 19/11 | 03/12 · Planned velocity 18
- Effort 1 · 27 · Create Schema Archi 2 · Planned
- Effort 8 · 28 · Create Web Service 2 · Planned
- Effort 5 · 29 · Create Web Interface 2 · Planned
- Effort 3 · 30 · Create Sensor Values 2 · Planned
- Effort 1 · 31 · Verify results 2 · Planned

Sprint 3 · Todo · 04/12 | 19/12 · Planned velocity 23
- Effort 1 · 32 · Create Schema Archi 3 · Planned
- Effort 8 · 33 · Create Web Services 3 · Planned
- Effort 5 · 34 · Create Web Interface 3 · Planned
- Effort 3 · 35 · Create Sensor values 3 · Planned
- Effort 1 · 36 · Verify results 3 · Planned
- Effort 5 · 19 · Create report · Planned

1 · 2 · 3
Nov 03 · Nov 10 · Nov 17 · Nov 24 · December · Dec 08 · Dec 15

Figure 15 : IceScrum initial planning

For this SOA project we decided to define three sprints. One for each scenario. The first sprint last longer as we need to chose the architecture and to discover how to implement the services and their interaction. Next sprints are simpler as we already know how the architecture works, we just need to collect the data by applying the same methodology to others sensors or via an API.

The use of IceScrum, or more generally of an agile method, should be a very useful planification technique for longer projects. Here, we did not need such a method as the project was really short. We spent a lot of time planning the stories and features but as some courses were moved it was quite hard to follow them. We didn't really update the IceScrum on time. However, we have worked as we were expecting, use case by use case.

GitHub

In order to manage version of our code we use git (https://github.com/clement-dlbl/SOA_Project) . We used an official git flow discribe by the official Atlassian bitbucket website : https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow
That a git flow used by majority of IT companies.
We decide to create 2 main branches: master which represent code pushed in production, develop which represent code to test and validate before to create a release and deploy it on production.

Then we created copies from develop branch when we would like to develop a new feature. At the end of feature development we tested our code create a release branch which is a copy of develop branch at a given commit. Create a release branch is important if we want not to block development of new features and furthermore if we see bugs on release code version we can make an hotfix to solve it before pushing into production (MIP).

<u>**Review of the project**</u>

So in this project, we tried to share the work to do between each member of the group. We try to load balance the load of work to do depending on technical background. Indeed, it was easy for IT students.

Conclusion

In this project we learn to develop a service application based on microservices. That kind of application is widely used on companies because it is easier to develop, it allow scale up and scale down. We can say that is the most suitable application architecture when we deploy application on a cloud infrastructure.

We learnt to deploy a service oriented application layer using a default template Model View Controller and using JAVA language. We also learn to create interaction between JAVA api and front html interface developed using web basic languages.

This project gave us the opportunity to work with agile management methods which are often use in IT projects. It was for the majority of us the first time we worked in that configuration and it demand some adaptation. Otherwise, when we will meet a project manage with that method we will able to interact with team members and be efficient.