5A - ISS

January 2021

Automatic management of INSA's rooms

For "Service based Architecture and Software Engineering"

By Léo POTIERS and Paul SERONIE-VIVIEN

Introduction

During a large period of the semester, we have been introduced to service architecture and software engineering. The goal of the project that will be presented in this report is to put into application the different notions that we have been taught. To be clearer, our mission was to develop a Web application that is responsible for automating INSA's rooms with features such as closing windows, turning on lights or turning off the heat. Our application must be based on a Restful architecture and we needed to use data collected by sensors and take decisions after analysing them.

The main difference from a project to another is the scenario the application is trying to answer to. In our case, we decided to orient to the actual sanitary crisis of COVID-19 and provide a service supposed to answer the new problematics it implies. Our main mission is to monitor the amount of people in INSA's rooms and to assure a good aeration between the different uses.

This report will be divided in two main parts. The first one treats the specifications of our application and what are the key points we wanted to cover. The second part explains the whole implementation process and what technical decisions we took.

Project planning

The first part of this project was to clearly define what functionalities our application was supposed to provide. To do that, we used the Scrum method with the help of JIRA, which is a platform dedicated to project management.

User stories

The first step, when it came to planning our project, was to determine our user stories. These are made to answer three basic questions: who, what and why. Who does the functionality help? What does it do? Why does it need to do this? After a certain period of brainstorming, we achieved to determine five user stories that would become the real body of our project. These are listed below.

- ❖ **US1:** As an administrator, I want to be able to count how many people are in a specific room to monitor its use and its aeration.
- ❖ **US2:** As an administrator, I want to ensure that the room's door stays closed during the whole aeration to be sure to provide a perfect process.
- ❖ **US3:** As an administrator, I want to control the opening of the windows after the room was used to start an aeration process.
- ❖ **US4:** As an administrator, I want to control the closing of the windows after the room was ventilated and ready to be used again.
- ❖ **US5:** As an administrator, I want an alarm to ring whenever the room is too crowded regarding the new sanitary recommendations, until the room is less full.

Tasks

After defining our user stories, we needed to list the different tasks that would allow us to meet our specifications. Basically, it was an establishment of what would be coded later and what user stories each of those tasks would serve. All these tasks are referenced to their User Story (US) and are listed below.

- ❖ Room's Flux: We need to simulate the evolution of the number of people in the room, and thus continuously. **(US1)**

- Crowdedness Detection: A signal must be sent whenever there are too many persons in the room. **(US1)**
- Automatic Closing of the Door: The door must close as soon as the room is left empty to let the aeration process start. It must stay closed during the whole time. **(US2)**
- Door State: We need to get the state of the door (open/close) at any time. **(US2)**
- Windows State: We need to get the state of the windows (open/close) at any time. **(US3)**
- Automatic Opening of the Windows: Once the door is shut, the windows must open automatically to begin the aeration process. **(US3)**
- Automatic Closing of the Windows: After the aeration process period is over, the windows should close automatically. **(US4)**
- Alarm Trigger: Whenever the room's maximum threshold is crossed, an alarm should start ringing until the room's population diminished. **(US5)**

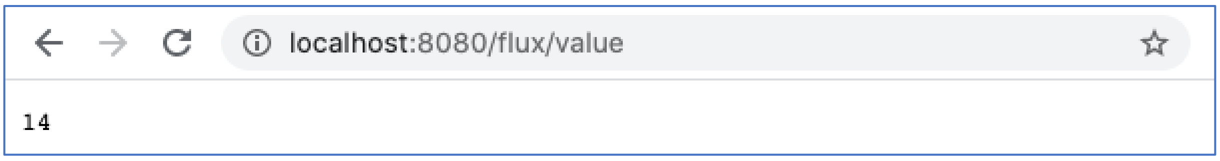
Sprints

Finally, we had to group all the tasks into sprints. Those can be assimilated to what we projected to do between two practice sessions, until the next meeting with a teacher that could help us overcome difficulties. In the end, we did not follow them perfectly since our time evaluation was quite unprecise. Initially, US1, US2 and US3 was in a first sprint and the rest in a second one. A JIRA screen of our sprints is available in appendices.

Implementation of the architecture

In order to create our architecture, we deployed online various web services that exchange data from simulated sensors and command simulated actuators.

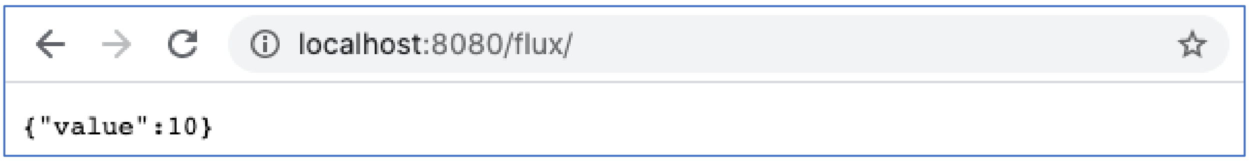
The first web service we deployed was *ServiceFlux*, that publishes on the port 8080 of localhost a random number of students in the class between 0 and 25. This service simulates the data that could be returned by a laser sensor that could be fixed at the entrance of the room, or of a neural network trained to count the number of people in a camera video flow. Once the web service is deployed, the number of people in the class is accessible at the following address: **http///localhost:8080/flux/value** as it is shown on the figure 1:



```
14
```

Figure 1: Simulation of the people flow by the web service *ServiceFlux*

The raw data is accessible at the address: **http///localhost:8080/flux/:**

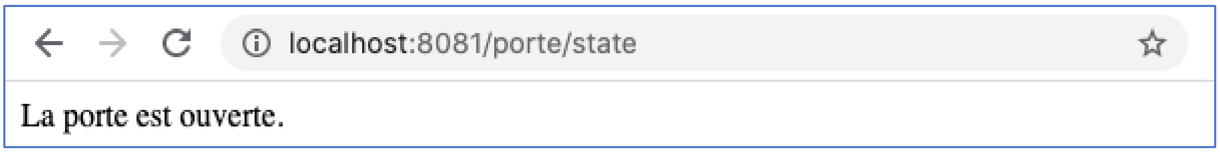


```
{"value":10}
```

Figure 2: Simulation of the raw people flow by the web service *ServiceFlux*

Now that the people flow is generated and accessible on the web, we need to create two other web services that will enable the access to the state of the door and of the windows. By this, we are simulating a sensors that are transmitting if the door and windows are closed and open, and the actuators to command automatically their state. Initially, the door is open and the windows are closed because there is no aeration process.

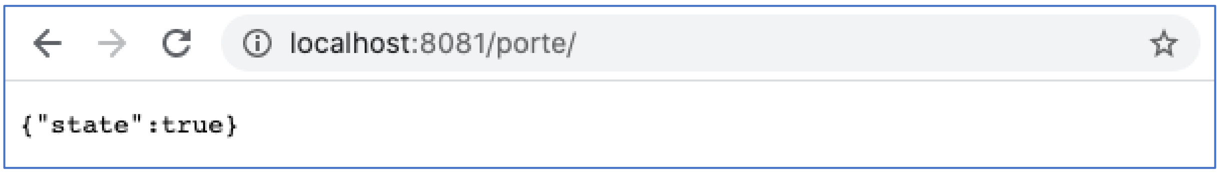
The Web Service for the door is called *EtatPorte* and publishes the state of the door at the address **http:// localhost:8081/porte/state**:


localhost:8081/porte/state

La porte est ouverte.

Figure 3: Simulation of the state of the door by the web service *EtatPorte*

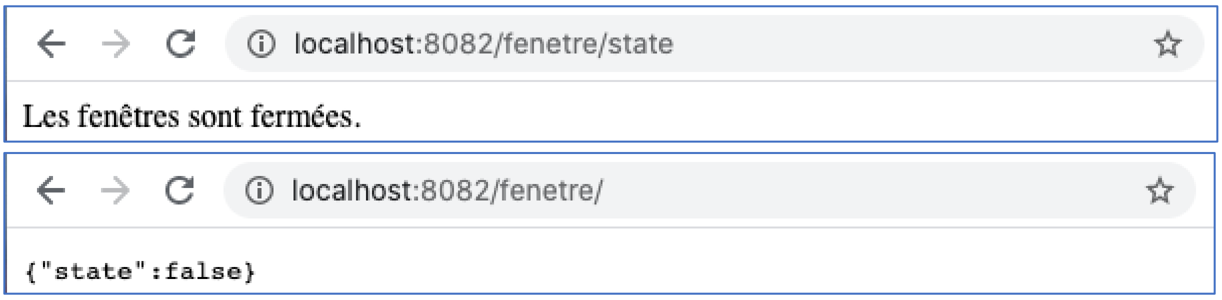
As previously, we can find the raw data on the topic **/porte/:**


localhost:8081/porte/

```
{"state":true}
```

Figure 4: Simulation of the raw state of the door by the web service *EtatPorte*

We designed the same web service for the window of the room with a web service called *EtatFenetre*, that publishes on **http://localhost:8082/fenetre/:**


localhost:8082/fenetre/state

Les fenêtres sont fermées.

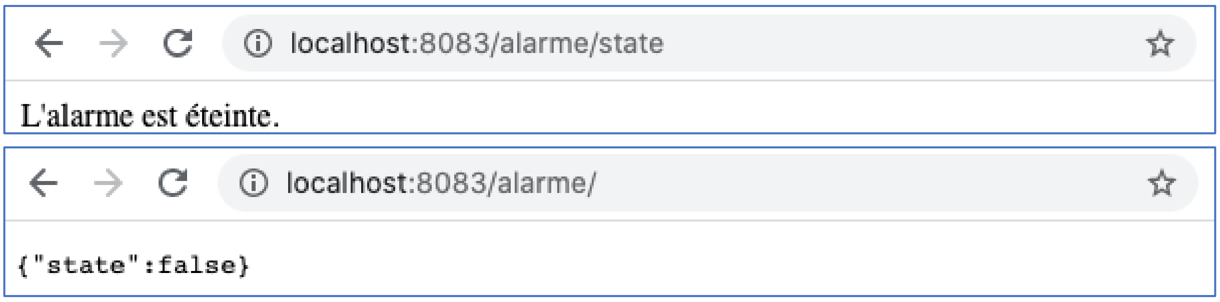
localhost:8082/fenetre/

```
{"state":false}
```

Figure 5: Simulation of the state of the window by the web service *EtatFenetre, analyzed and raw*

We created another web service called *Alarme* in order to alert the users that the maximal capacity of the room is reached, for our study we fixed this maximal capacity to 25 but this could be adapted to the size of the room.

Similarly, to what we have presented before for the other web services, the data are accessible on the address **http://localhost:8083/alarme/**:


localhost:8083/alarme/state

L'alarme est éteinte.

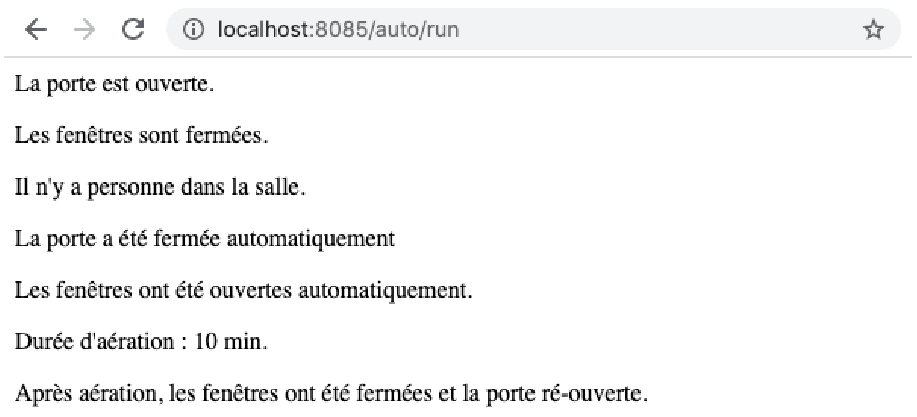
localhost:8083/alarme/

```
{"state":false}
```

Figure 6: Simulation of the state of the alarm by the web service *EtatFenetre, analyzed and raw*

Finally, we created a main web service called *Automation* which subscribes to all data published on previous web services and that commands appropriate action. This web service is accessible at **http://localhost:8085/auto/run**.We defined three scenarios depending on the number of students in the room published by *ServiceFlux*:

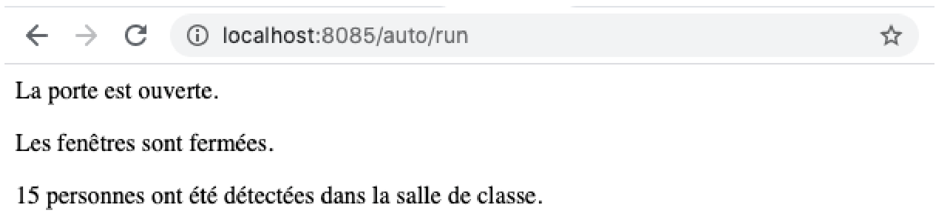
- **Aeration scenario:** Happens after every use of the room, when there is no students detected in the room. For 10 minutes, the door is automatically closed, and the windows are being opened in order to start the aeration process.



```
La porte est ouverte.

Les fenêtres sont fermées.

Il n'y a personne dans la salle.

La porte a été fermée automatiquement

Les fenêtres ont été ouvertes automatiquement.

Durée d'aération : 10 min.

Après aération, les fenêtres ont été fermées et la porte ré-ouverte.
```

Figure 7: Aeration scenario on the Automation web service

- **Normal use of the room scenario:** When the number of detected students is in] 0 ; 25 [, the door is open, there is no ventilation process so the windows are closed,



```
La porte est ouverte.

Les fenêtres sont fermées.

15 personnes ont été détectées dans la salle de classe.
```

Figure 8: Normal use of the room scenario on the Automation web service

- **Emergency scenario:** When the number of students in the room is at 25, then the alarms turns on until the number of students decreases.

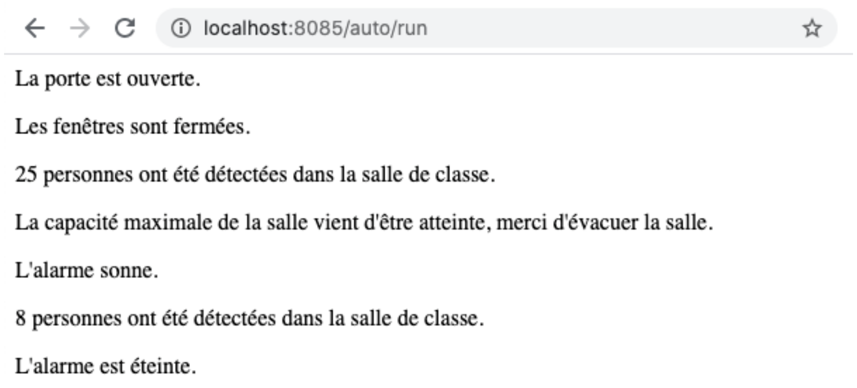

```
localhost:8085/auto/run

La porte est ouverte.

Les fenêtres sont fermées.

25 personnes ont été détectées dans la salle de classe.

La capacité maximale de la salle vient d'être atteinte, merci d'évacuer la salle.

L'alarme sonne.

8 personnes ont été détectées dans la salle de classe.

L'alarme est éteinte.
```

Figure 9: Emergency scenario on the Automation web service

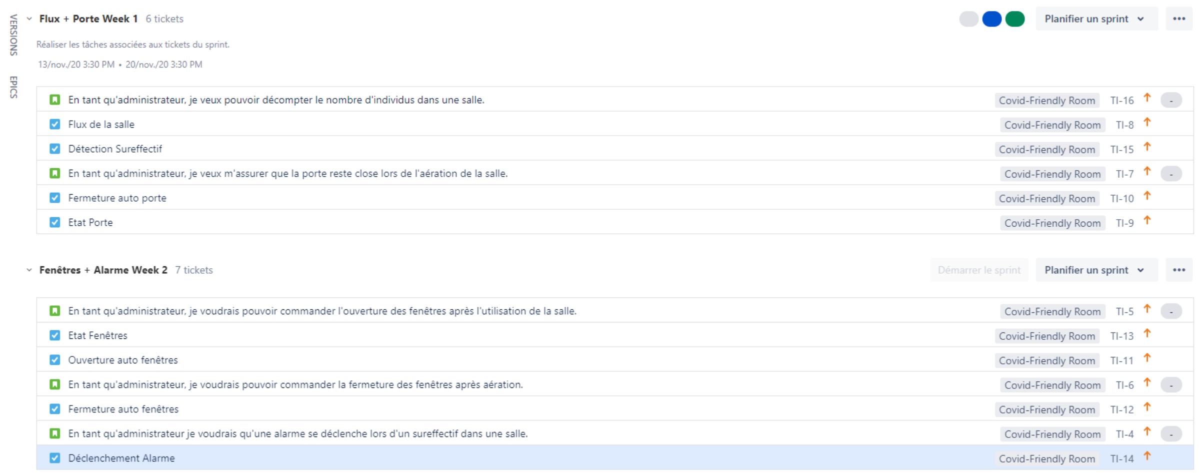
Conclusion

During this project, we were able to imagine an application that suited our will and ideas. In this strange period, sanitary speaking, we really took at heart designing a solution that fitted needs that are new to everyone but that could totally be part of our daily life for a certain time.

Using the Scrum method was new to both of us, as well as using JIRA, but in the end, it really helped us fixing the boundaries of our project. There is no doubt that it helped us optimizing our thought process.

When it came to the implement part, it was key for us to understand and make work several Web services. In the end, we were really satisfied about all the interactions we were able to design between our different services. Seeing how the whole system was reacting using some prints on a web page was already quite unique for us. Due to technical issues on both of our machines and a lack of time at the end of the sessions, we were not able to able to deploy our application using OM2M. It was frustrating not crossing the final line with the project, but we are still happy about our work and the result.

Appendices



Flux + Porte Week 1 6 tickets

Réaliser les tâches associées aux tickets du sprint.

13/nov./20 3:30 PM • 20/nov./20 3:30 PM

En tant qu'administrateur, je veux pouvoir décompter le nombre d'individus dans une salle.	Covid-Friendly Room	TI-16	-
Flux de la salle	Covid-Friendly Room	TI-8	
Détection Sureffectif	Covid-Friendly Room	TI-15	
En tant qu'administrateur, je veux m'assurer que la porte reste close lors de l'aération de la salle.	Covid-Friendly Room	TI-7	-
Fermeture auto porte	Covid-Friendly Room	TI-10	
Etat Porte	Covid-Friendly Room	TI-9	

Fenêtres + Alarme Week 2 7 tickets

En tant qu'administrateur, je voudrais pouvoir commander l'ouverture des fenêtres après l'utilisation de la salle.	Covid-Friendly Room	TI-5	-
Etat Fenêtres	Covid-Friendly Room	TI-13	
Ouverture auto fenêtres	Covid-Friendly Room	TI-11	
En tant qu'administrateur, je voudrais pouvoir commander la fermeture des fenêtres après aération.	Covid-Friendly Room	TI-6	-
Fermeture auto fenêtres	Covid-Friendly Room	TI-12	
En tant qu'administrateur je voudrais qu'une alarme se déclenche lors d'un sureffectif dans une salle.	Covid-Friendly Room	TI-4	-
Déclenchement Alarme	Covid-Friendly Room	TI-14	

Figure 1: Sprints planning on JIRA