Project: Automatic management of INSA's rooms

Git : https://github.com/Assianguyen/SOA_Project.git

Agathe Lièvre
Assia Nguyen

5ISS

2021/2022

I. Introduction

This project is set in the context of the Service based Architecture and Software Engineering course. We developed a Web application (Proof-of-Concept) that automatically manages an INSA's room . For that, we had to create microservices (software services) that used simulated sensors. Depending on the value we retrieve from them, actuators or alarms can be triggered.

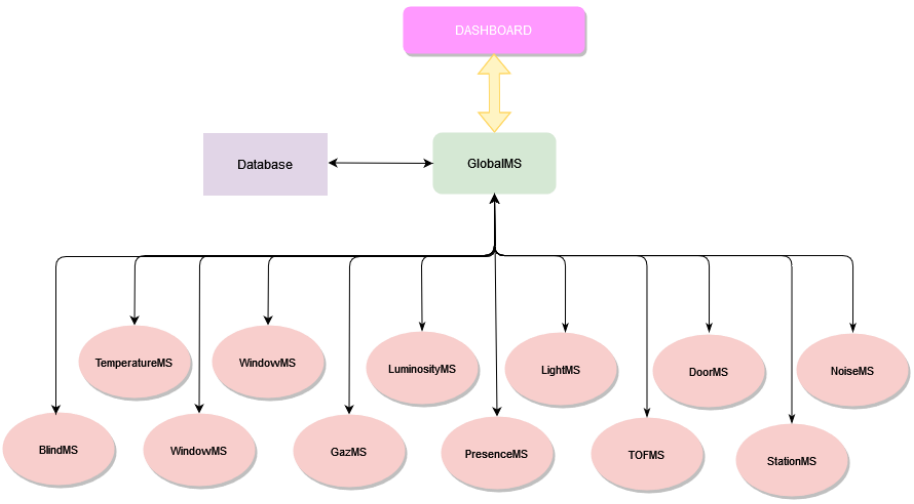
II. Scenarios

We first had to create different scenarios to implement in our web application. Here is the list of the ones we decided on :

Scenario	Sensors used (microservice)	Actuators used (microservice)
If the outside temperature is lower than the indoor temperature and the outside temperature is between 18 and 27°, the windows should be opened automatically	Temperature (TemperatureMS)	Window (WindowMS)
If the outside luminosity is high enough the light inside of the room are turned off	Luminosity (LuminosityMS)	Light (LightMS)
[COVID-19]If the CO2 level of the room is over 800 ppm, the doors and windows are opened automatically and an alarm is triggered	Gaz (GazMS)	-Window (WindowMS) -Door (DoorMS)
[COVID-19]If there are too many people in a room, the doors and windows are opened automatically and an alarm is triggered	Presence detector (PresenceMS)	-Window (WindowMS) -Door (DoorMS)
If an object belonging to INSA leaves the room and is too far away from its intended position, an alarm is triggered	TOF (TOFMS)	X
If the noise level in the room is too high, an alarm is triggered	Noise (NoiseMS)	X
If the current time is outside working hours and there is nobody in the room, the doors, the windows and blinds are closed, the light and the stations are turned off	X	-Door (DoorMS) -Window (WindowMS) -Blind(BlindMS) -Light(LightMS) -Station (StationMS)
If the emergency button inside the room is activated, an alarm is triggered	X	- Button (GlobalMS)

III. Microservice architecture

For each actuator we used, we created an associated class in the model package of the microservice. For every microservices, we created a Ressource java file in the controller package. We have a microservice (GlobalMS) that manages and supervises all the other microservices and generates the response to the scenarios. It is the only microservice where the emergency button is used since it is triggered on the node-red interface.



List of API endpoints:

- GET "http://localhost:8080/gei/auto”
- GET "http://localhost:8080/gei/run”
- POST "http://localhost:8080/gei/seuil/{sensor}?value={int value}”
- POST "http://localhost:8080/gei/button/{actuator}?value={bool value}”
- POST "http://localhost:8080/gei/order/{actuator}?value={int value}”
- GET "http://localhost:[port]/[device]/{id}”
- GET "http://localhost:[port]/[device]/status/{id}”
- POST "http://localhost:[port]/[device]/order/{id}?value={[v]}”
- PUT "http://localhost:[port]/[device]/{id}”
- GET "http://localhost:[port]/[sensor]/value”

Microservice	[port]	[device]/[sensor]	[v]
BlindMS (don’t have GET "http://localhost:port/device/status/{id}”)	8081	[d] blind	int value
DoorMS	8082	[d] door	bool value
GazMZ	GET “http://localhost:8083/gaz/CO” GET “http://localhost:8083/gaz/CO2”		
LightMS	8084	[d] light	bool value
LuminosityMS	8085	[s] luminosity	
NoiseMS	8086	[s] noise	
PresenceMS	8087	[s] presence	
StationMS	8088	[d] station	bool value
TemperatureMS	8089	[s] temperature	
TOFMS	8090	[s] tof	
WindowMS	8091	[d] window	bool value

IV. Agile method and continuous integration

The management method we used was the agile method. We used the JIRA tool to schedule the different objectives and tasks throughout the project. To do so, we divided our project into 3 sprints. Here are the main tasks of each sprint

1st sprint: Invent the scenarios, create the architecture,create the Classes associated with the actuators as well as the Ressources.

2nd sprint: Implement and automate most of the scenarios on the GlobalMS

3rd sprint: Finish the scenarios left and create the node-red interface, write the report

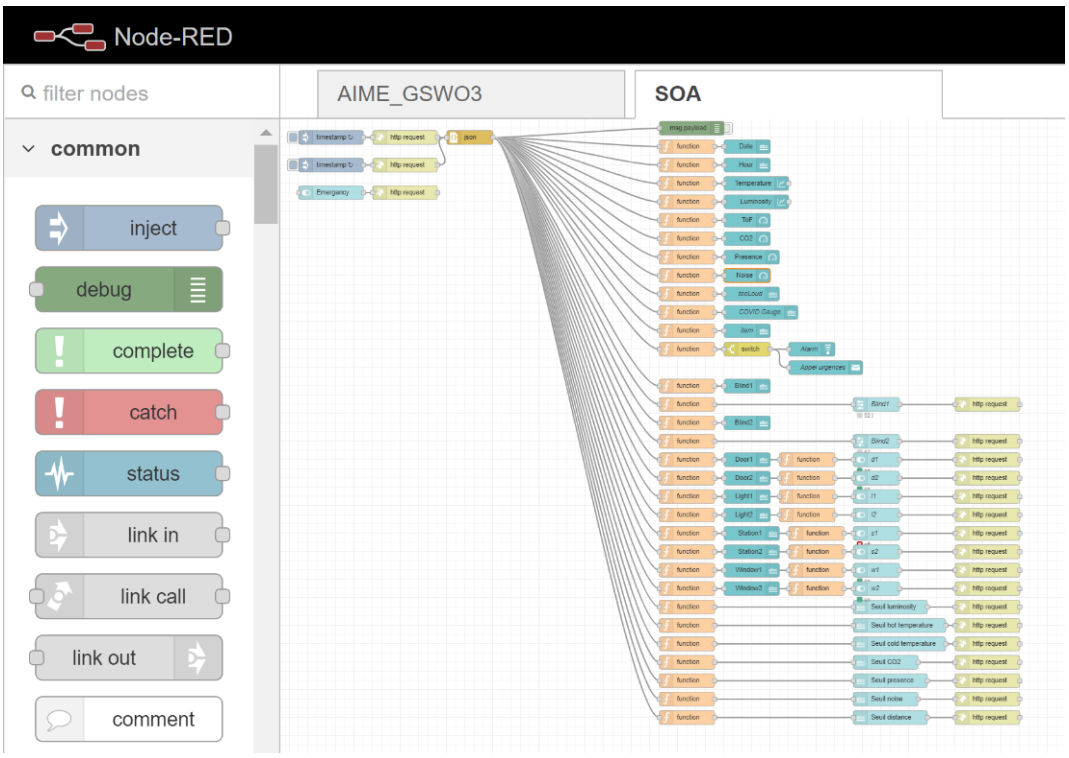
A screenshot of the JIRA interface can be found in the appendix.

This part allowed us to work with a real agile tool that can be really useful to us in our future careers.

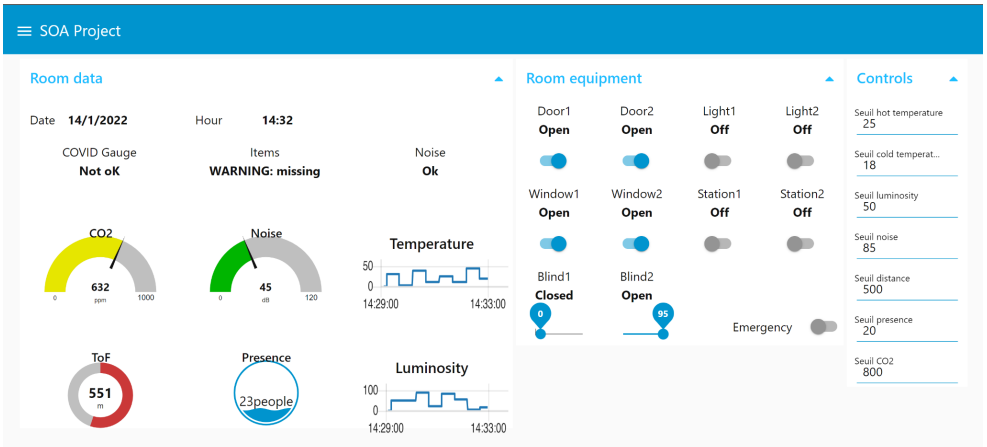
We also implemented a Continuous Integration thanks to GitHub Actions. First, we tried using Jenkins but we were unsuccessful, so we switched to GitHub Actions. We did not have time to implement a lot of tests. We created a workflow that builds our Java projects with Maven, each time we push our code in our repository. It allowed us to see if our projects were buildable, meaning there were no errors in our code. We could have implemented other workflows to test our functions and see if they give the expected results.

V. Node-red interface

We used Node-RED to create our dashboard. The figure below represents our flow (you can also find it on our GitHub repository):



With a ‘http request’ node, we send GET requests to our microservice ‘GlobalMS’. Every 30 seconds, we send a GET request to do an auto run of the global microservice, i.e. simulating the data recovery from the sensors and playing the scenarios. Every 1 second, we send a GET request to run the global microservice to update the data if the user used an actuator or changed the threshold. You can see our dashboard in the following figure:



The dashboard is composed of 3 sections:

- room data. This section displays the data received from the sensors and the state of the Covid gauge, if some items are missing and if the noise level is not too loud.
- room equipment. This section displays all the equipment present in the room. The switches and sliders allow the user to open/turn on or close/turn off the equipment.
- controls. This section displays the thresholds for all our scenarios. The user can change them here if he wants.

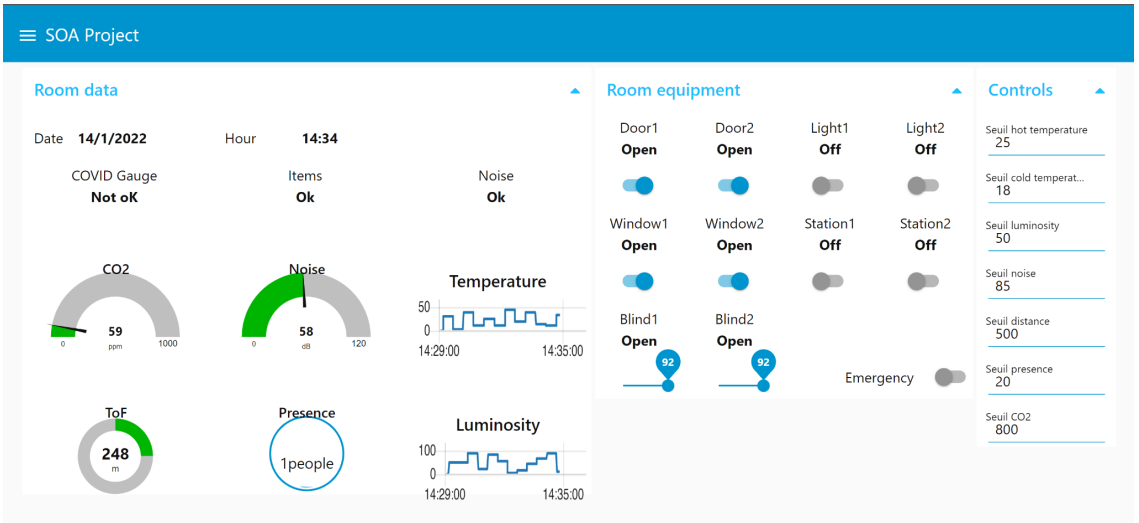
In the appendix, you can see the dashboard in two different states. In the first picture, everything is ok (except the COVID gauge), you can see the states displayed on the dashboard. In the second picture, someone pushes the emergency button and you can see the notification on the bottom right corner. You can also see what happens when the thresholds are exceeded.

VI. Conclusion

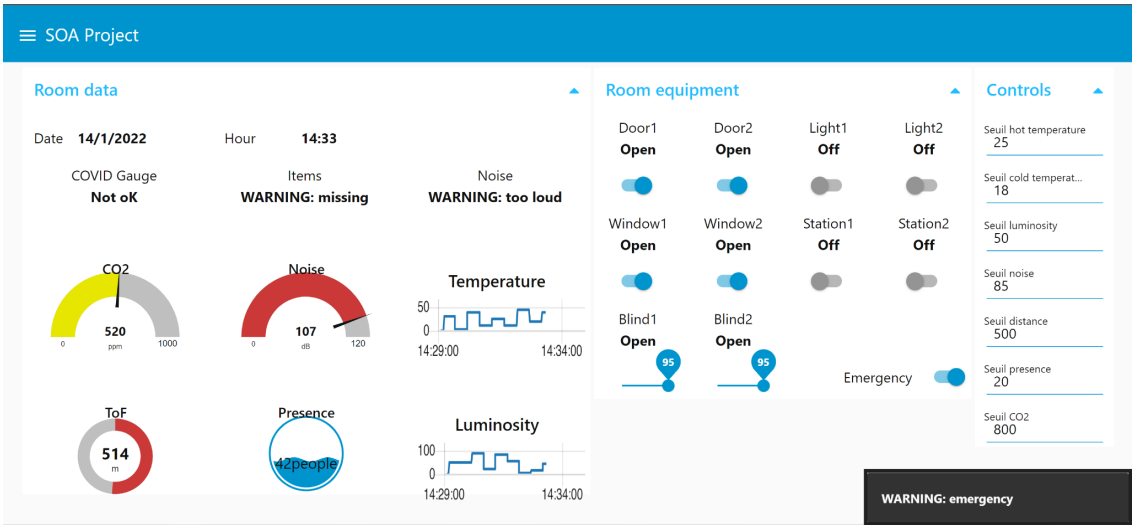
In this project, we were able to use many notions we had learned in the course of the UF. We were able to apply theory into an interesting application.

VII. Appendix

Dashboard - status Ok:



Dashboard - emergency:



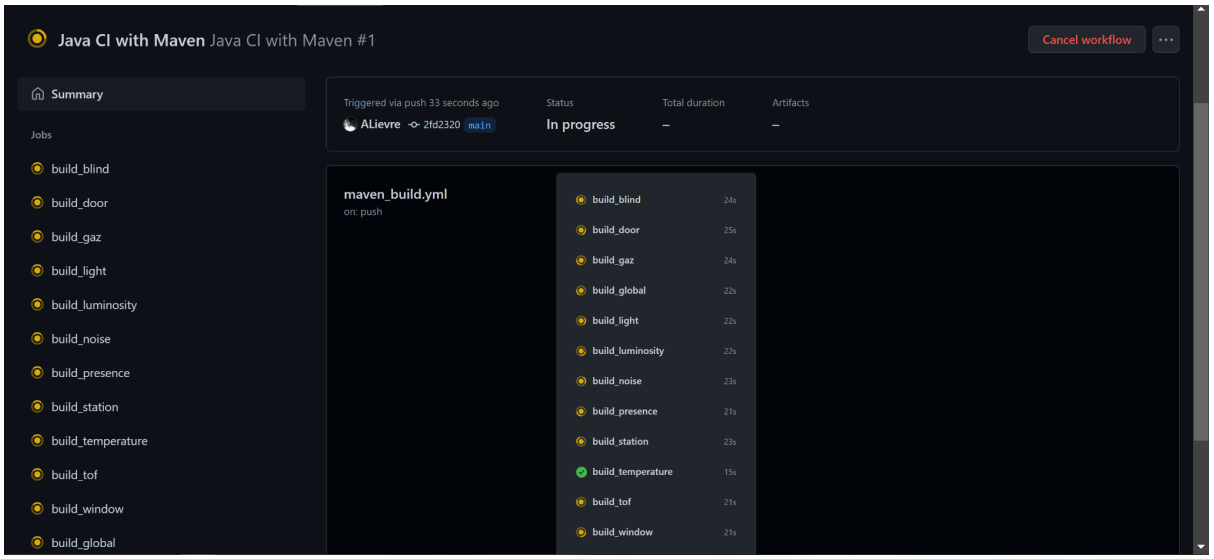
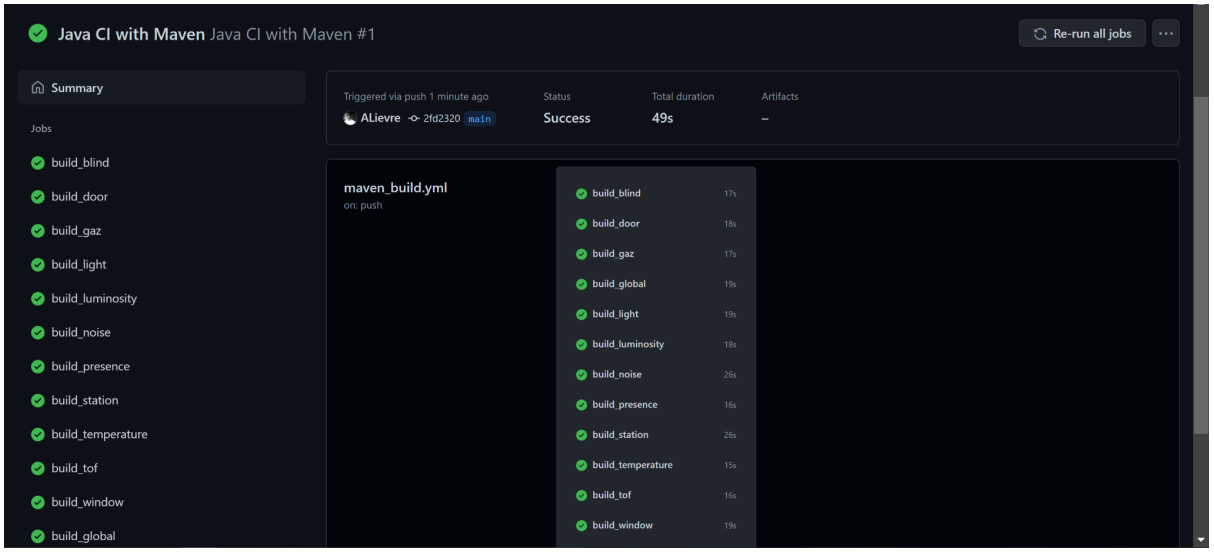
Jira: sprint 3

Tickets terminés

Afficher dans le navigateur de tickets

Clé	Résumé	Type de ticket	Epic	État	Responsable	Story points
AMIR-73	Implémentation de l'ensemble des scénarios	Story		TERMINÉ(E)		-
AMIR-37	Automatisation des microservices dans GlobalMS	Story		TERMINÉ(E)	AL	-
AMIR-36	Créer ressources	Story		TERMINÉ(E)		-
AMIR-6	Créer interface	Story		TERMINÉ(E)	AL	-

GitHub Actions: workflow in progress



GitHub Actions: workflow successful