Management of INSA Rooms Lab Report

Service Oriented Architecture & Software Engineering, ISS 2023

Cyprien Heusse, Emily Holmes, Aude Jean-Baptiste, Romain Moulin
GitHub - GitHub Organization (for CD)

Conception and technological choices

We are trying to manage and automate various actuators in INSA classrooms. We are working with the following sensors and actuators, each connected to a room. The architecture is as follows:

- Each sensor continuously sends in its data.
- A microservice handling all sensors of the same type receives the data in the form of REST messages.
- If the value is above or below a threshold, or indicates a significant change, the microservice sends this information to the supervisor.
- The supervisor microservice handles the logic between various sensors of the same room and, if necessary, sends a command to the room actuator.
- The supervisor microservice also interacts with a front microservice. It stores in a database all previous messages, so the logs may be viewed on a webpage if requested.

This can be represented visually on the following Figure 1:

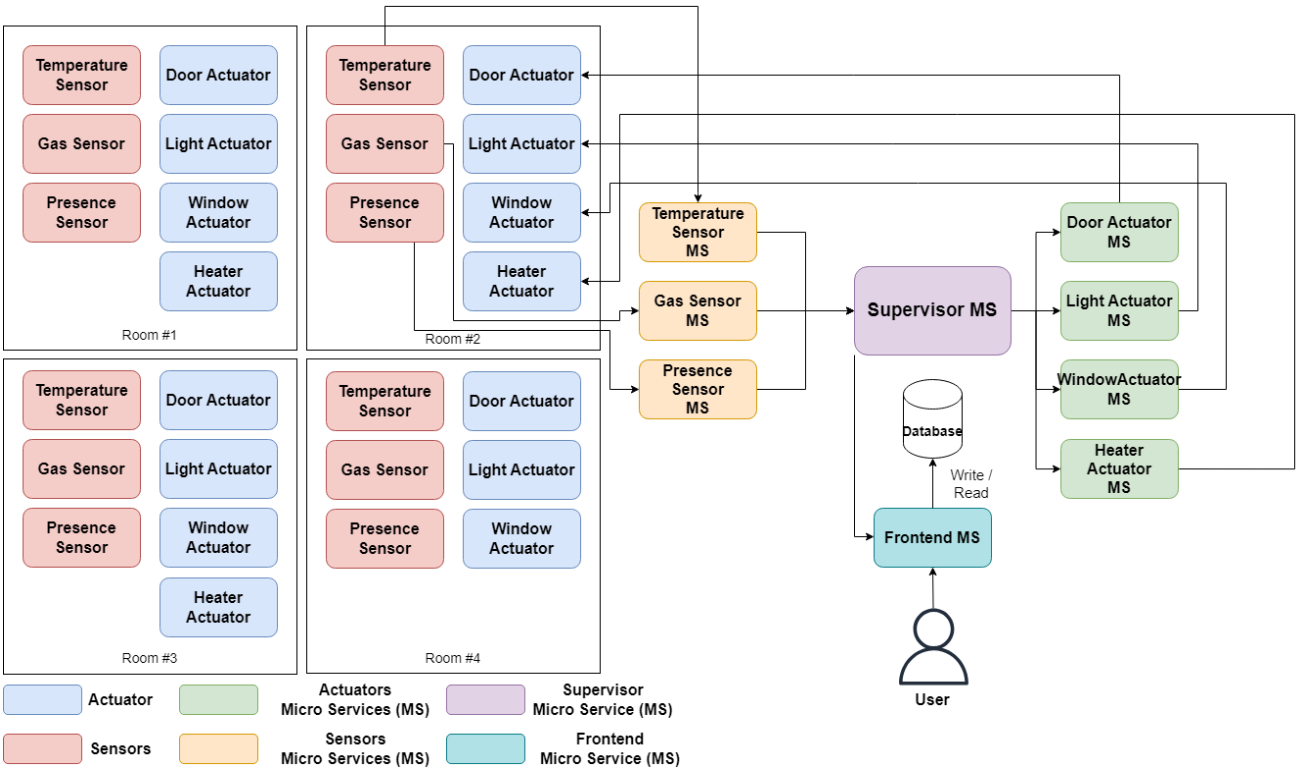

Figure 1: Diagram of the MS architecture

Technologically, we are choosing to develop the application using node.js to handle the various microservices. The services will communicate using the REST API, as per microservices standards.

Project management and tracking

For this project, our team chose to use Jira and SCRUM as the framework. Sprints were 1 to 2 weeks long to match with the lab sessions and to adjust to our time schedules on a given week. Story points were not attributed as we felt that it would be a hindrance. This is a short project so we wanted to optimize the time available to us without the need to add meetings.

We created tickets for each given sprint.

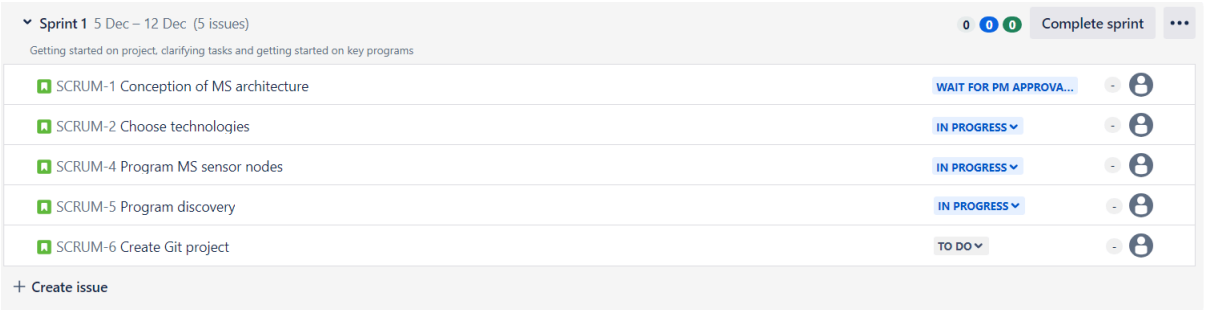

Figure 2: Screenshot of our tickets

A custom field “Assignees” was created to be able to affect multiple people to the same ticket (as we would often work in pairs). Each ticket figured a clear indication of what

had to be delivered to consider the ticket done. A “wait for PM approval” tab was created. Although we do not have a Product Manager (or PO, Product Owner), it meant we had to look at the deliverables together before clearing the ticket.

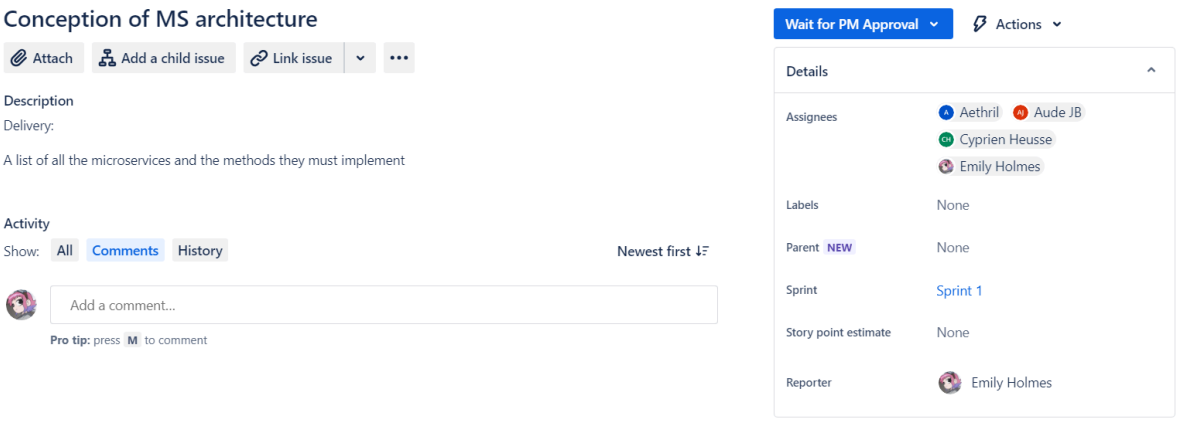

Figure 3: Detailed view of a ticket

Technical Integration

As we do not really process sensors and actuators for this project, we decided to simulate the behavior of such devices by implementing a REST POST interface to our Sensors Micro Services (MS). That way, we would be able to send data to the microservices and start the whole process of detection and correction of abnormal situations.

The behavior of our sensor MS consists of a simple REST POST interface that can possibly filter some special events (for example the temperature MS only uses that data when it reaches a threshold). When those events are detected, they send the data to a supervisor through a POST interface.

After that the supervisor MS collects all the special events from the different sensors, stores these events in a database and takes action depending on the state of all the different sensors of a room. These actions take the form of sending orders to the actuators through another POST interface. If the supervisor took an action, it stores what it did in the database.

The final MS is the frontend MS which is basically a web server that the user is available to GET. When it receives a request, it looks in the database and displays all the events that were stored.

We stored all the actions that our supervisor can take depending on the state of our sensors in the following table:

	Temperature	Gas	Presence	Actuator State	
1	Low	No gas	No presence	Window : Closed	Door: Closed
				Heater : ON	Light : OFF
2	Low	No gas	Presence	Window: Closed	Door: Opened
				Heater: ON	Light: ON
3	Low	Gas	No presence	Window: Opened	Door: Opened
				Heater: ON	Light: OFF

4	Low	Gas	Presence	Window: Opened	Door: Opened
				Heater: ON	Light: ON
5	High	No Gas	No Presence	Window: Opened	Door: Closed
				Heater: OFF	Light: OFF
6	High	No Gas	Presence	Window: Opened	Door: Closed
				Heater: OFF	Light: ON
7	High	Gas	No Presence	Window: Opened	Door: Opened
				Heater: OFF	Light: OFF
8	High	Gas	Presence	Window: Opened	Door: Opened
				Heater: OFF	Light: ON
9	Normal	No Gas	No Presence	Window: Closed	Door: Closed
				Heater: OFF	Light: OFF
10	Normal	No Gas	Presence	Window: Closed	Door: Closed
				Heater: OFF	Light : ON
11	Normal	Gas	No Presence	Window: Opened	Door: Opened
				Heater: OFF	Light: OFF
12	Normal	Gas	Presence	Window: Opened	Door: Opened
				Heater : OFF	Light: OFF

Table 1: Table of action of the supervisor

Continuous Integration (CI)

In a desire to streamline development and to mimic real software engineering as closely as possible, we implemented CI in the project. Specifically, we used GitHub Actions and Mocha, a Node.js test framework to automate integration.

Functions from the program had to be exported in order to be tested with a test script. Mocha allows us to write tests and their expected behavior. The use of Github Actions runs these test scripts whenever we push onto the repository. An example of this can be found below.

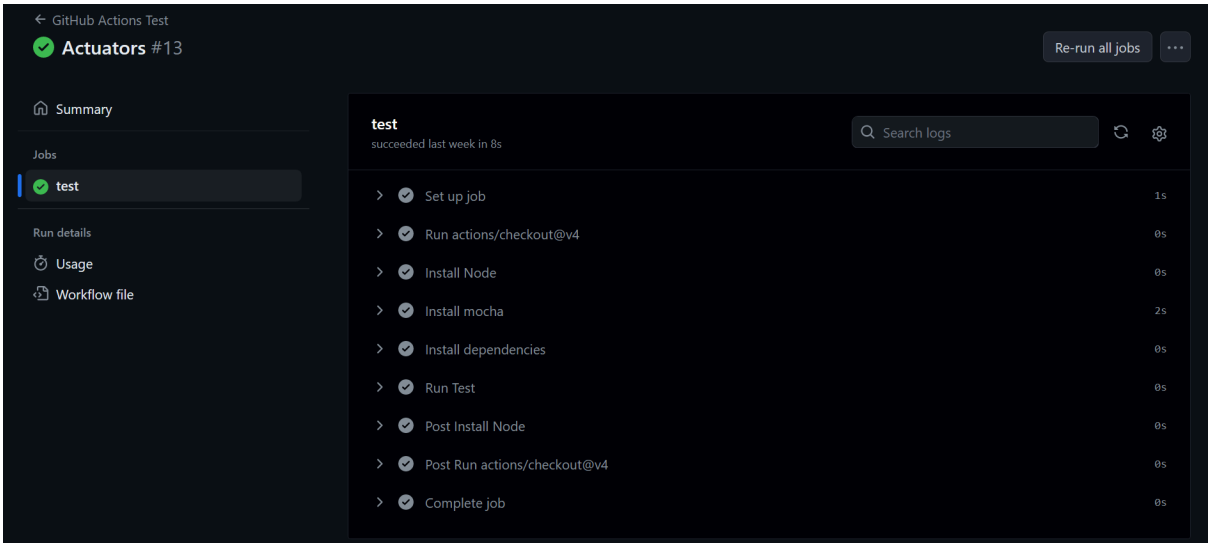

Figure 4: Github Actions executed on each push

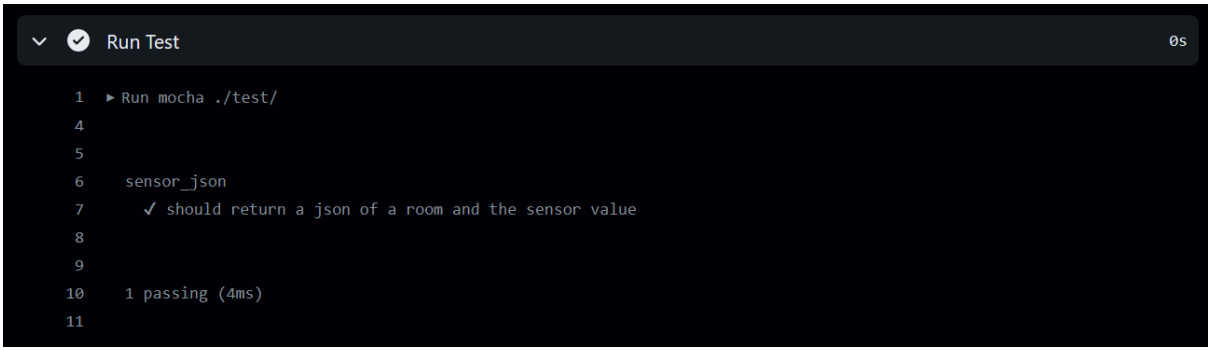

Figure 5: Details of the first test

Continuous Deployment (CD)

To be able to manage deployments efficiently, we created a new GitHub workspace. In this workspace, each service has its own git repository. Each service has a dev and a prod branch. When working on code, the developer should always be in the dev branch. Once the new code has been approved by the Product Manager (and the team), the dev branch is merged to the prod branch which is where a GitHub action is triggered to automatically deploy the update to Azure.
When updating the Discovery, all other services need to be restarted for the network to work as a whole.

Frontend

Finally, a simple frontend interface was made to track all the changes in the system as per Figure 1. For example, we log which actuator in which room triggered which action.

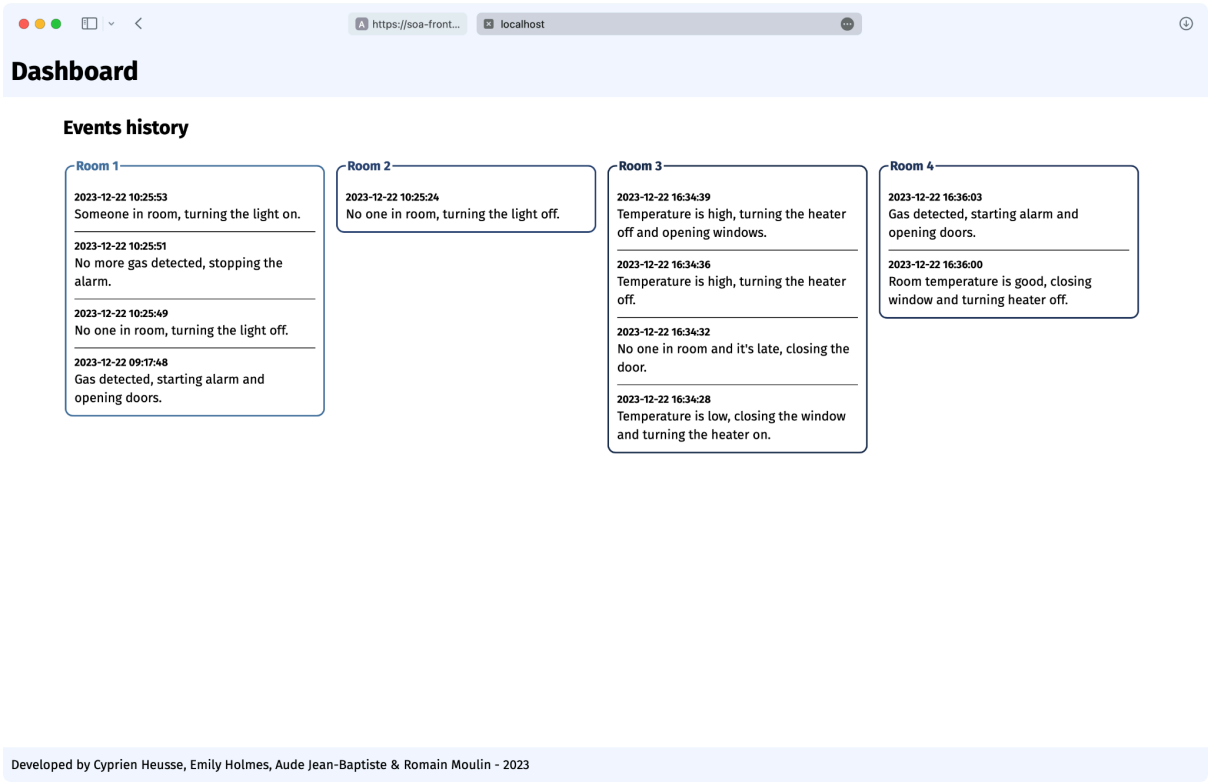

Figure 6: Frontend dashboard

Conclusion

We have successfully conceived a system to automate the management of INSA rooms following a microservice architecture. The project features CI and CD thanks to GitHub Actions and Microsoft Azure and was led with a SCRUM framework.