INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

2021-2022

Automation of the INSA's rooms management

Github : https://github.com/zinebamegz/ProjetSOA

Students :

Zineb Amegouz

Solène Dumas-Grollier

Rebekka Lie

Département Génie Électrique et Informatique
135, Avenue de Rangueil
31077 Toulouse Cedex 4

1. User stories

In order to develop a full application to manage an INSA's room, we have divided each of the imagined scenarios in little user stories :

N°	User stories
1	**As a** user of the room *(student or teacher),* **I want** the lights to automatically be switched on when I walk into the room, **so** that no energy is wasted.
2	**As a** facility manager, **I want** the heating system to be turned on if the temperature is lower than 15°C and turned off if the temperature exceeds 25°C, **so** that the users don't get too cold or too hot.
3	**As a** covid responsible person, **I want** the ventilation system to be turned on automatically if the CO2 level in a room exceeds 30PPM, **so** that the air quality is at a healthy level.
4	**As a** student, **I want** to see which rooms are available and not, **so** that I know where to study.
5	**As a** covid responsible person, **I want** the windows to be opened every hour, **so** that fresh air is brought in on a frequent basis.
6	**As a** facility manager, **I want** to see which rooms are in use and not, **so** that I know what ventilation systems I can turn off.
7	**As a** facility manager, **I want** the heating system to be turned off on the nights and weekends **so** that it doesn't waste energy

In order to implement the three highlighted user stories we needed **6 microservices**: 3 for the sensors and 3 for the actuators, and **one microservice** to make them collaborate - Orchestrator. Since the project is on a local host (127.0.0.1) each microservice has a different port to be able to test them together.

Sensors(Port number) :

- Movement (8081), *Project Name* : MovSensor
- Temperature (8082), *Project Name* : TempSensor
- CO2(8083), *Project Name*: CO2Sensor

Actuators:

- Light (8092), *Project Name* : LightActuator
- Heating (8093), *Project Name* : HeaterActuator
- Ventilation (8094), *Project Name:* VentillationActuator

Controller :

- Orchestrator(8090), *Project Name* : Orchestrator

*This project is based on **SpringBoot** and **Java**.*

2. Realization of the User Stories

a. *User Story n°1 : Lights management*

The idea is to automatize the lights. The goal is to turn ON the lights when a movement is detected in the room. To do so, the movement sensor detects it thanks to a variable 'Motion' set to true. Then, the lights are turned ON with the variable 'State' set to true.

To realize this user story, I have implemented two microservices : ***LightActuator, MovSensor*** and the ***LightControl*** method inside the *Orchestrator*.

An example to simulate the automation of the lights, follow these steps :

1. Run the applications of the microservices *LightActuator, MovSensor, Orchestrator* as Java applications in Eclipse IDE

2. Check that the lights are off by one of these two URLs:
   a. http://localhost:8090/Orchestrator/LightControl
      Expected display : **Lights OFF - Movement not detected**
   b. http://localhost:8092/LightActuator/isON
      Expected display : **false**

3. Simulate a movement inside the INSA room by calling the method detect and giving it value=true, URL : http://localhost:8081/MovSensor/detect/true
   Expected display : **true**

4. Launch the LightControl to see if the lights turned on :
   http://localhost:8090/Orchestrator/LightControl
   Expected display : **Lights ON - Movement detected**

5. Another test to check the state of the lights can be done *(step 2)*

b. *User Story n°2 : Heating system control*

The orchestrator recovers the temperature from the temperature sensor in the room. If the temperature is under 15°C, the orchestrator turns ON the heating system.
If the temperature recovered exceeds 25°C, the orchestrator turns OFF the heating system.

To meet these specifications, I implemented two microservices (***TempSensor*** and ***HeaterActuator***) and a method in the *Orchestrator* microservice.

An example to simulate the automation of the heating system, follow these steps :

1. Run the applications of the microservices *TempSensor, HeaterActuator, Orchestrator* as Java applications in Eclipse IDE

2. Simulate the temperature in the room by putting a double value at the end of this URL: http://localhost:8082/TempSensor/setTemp/26.5/
   The heater will be turned on if the temperature in the room is lower than 15.0 °C and it will be turned off if the temperature is higher than 25.0 °C.
   Expected display : **26.5**

3. Run This URL to launch the automation of the heater:
   http://localhost:8090/Orchestrator/HeatingControl/
   Expected display : **Temperature in the room = 26.5 °C > 25.0 °C : too high - Heater turned OFF.**

4. You can check that the heater is turned ON or OFF with this command : http://localhost:8093/HeaterActuator/isActive/
   Expected display : **false**

c. *User Story n° 7 : Ventilation control*

The CO2 sensor measures the CO2 level of the room. If the CO2 level in the room exceeds 30 PPM, the ventilation is turned ON to ensure good air quality. If the CO2 level in the room is lower than 30 PPM, the ventilation is turned OFF.

In order to set up the functionalities above, I made the two microservices ***CO2Sensor*** and ***VentilationActuator***, and the **VentilationControl** in the microservice *Orchestrator.*

An example to simulate the automatic ventilation, follow these steps:

1. Run the applications of the microservices *CO2Sensor*, *VentilationActuator* and *Orchestrator* as Java applications in Eclipse IDE

2. Simulate the CO2 level in the room : http://localhost:8083/CO2Sensor/getCO2Value
   The value is 30 PPM by default. The CO2 value can be changed
   http://localhost:8083/CO2Sensor/setCO2Value/{CO2Value}
   The CO2 unit can also be simulated as described in the table below
   Expected display : **CO2 level set to 15PPM** (CO2Value=15)

3. Simulate the VentilationActuator in the room: get the ventilation state
   http://localhost:8094/VentilationActuator/getVentilationState or set the ventilation state: http://localhost:8094/VentilationActuator/setVentilationState/{activate}
   Expected display : **false** (activate=false)

4. Simulate the VentilationControl in the Orchestrator :
   http://localhost:8090/Orchestrator/VentilationControl
   Expected display : **Ventilation OFF - CO2 level is good**

d. Summary of all microservices and the respective possible commands

Microservice	port	Action	Associated URL
MovSensor	8081	Get the state of the sensor - if a movement is detected or not	http://localhost:8081/MovSensor/isIN
		Set or ignore a motion by putting a boolean at the end of the URL (true to detect and false in the other case)	http://localhost:8081/MovSensor/detect/{motion}
TempSensor	8082	Get the temperature in the room	http://localhost:8082/TempSensor/getTemp/
		Set the temperature /!\ manipulate double *To see if it works, do a getTemp (previous command).* *Useful to test our entire system.*	http://localhost:8082/TempSensor/setTemp/{TemperatureValue}
CO2Sensor	8083	Get CO2 level	http://localhost:8083/CO2Sensor/getCO2Value
		Set CO2 level (int) and display the CO2 level set and the current unit.	http://localhost:8083/CO2Sensor/setCO2Value/{CO2Value}
		Get CO2 Unit	http://localhost:8083/CO2Sensor/getCO2Unit
		Set CO2 Unit (String) and display the CO2 unit set.	http://localhost:8083/CO2Sensor/setCO2Unit/{unit}
LightActuator	8092	Get the state of the lights - true means they are ON	http://localhost:8092/LightActuator/isON
		Turn ON or OFF the lights and return the state they are in - state expects a boolean (true = ON, false = OFF)	http://localhost:8092/LightActuator/setLight/{state}
HeaterActuator	8093	Get if the heater is on (true) or off (false)	http://localhost:8093/HeaterActuator/isActive/
		Set the heater on (true) or off (false) depending on the given parameter	http://localhost:8093/HeaterActuator/putHeaterOnOff/true
		Get the temperature that is programmed to be reached by the heater	http://localhost:8093/HeaterActuator/getTemperatureToReach/
		Set the temperature to be reached /!\ manipulate double *To see if it works, do a getTemperatureToReach (previous command)*	http://localhost:8093/HeaterActuator/setTemperatureToReach/{TemperatureValue}
		Get ventilation state (boolean).	http://localhost:8094/VentilationActuator/getVentilationState

VentilationActuator	8094	Set ventilation state (boolean) and return the ventilation state set.	http://localhost:8094/VentilationActuator/setVentilationState/{activate}
Orchestrator	8090	Start the automatic management of the lights. The method gets the state of the sensor and if a movement is detected the lights are turned ON, OFF otherwise.	http://localhost:8090/Orchestrator/LightControl
		Start the automatic control of heating system, as described in the corresponding paragraph	http://localhost:8090/Orchestrator/HeatingControl/
		Start the automatic control of the ventilation system. The method gets the CO2 level and CO2 unit, and if it is more than 30PPM, the ventilation is turned on. Otherwise it is turned off.	http://localhost:8090/Orchestrator/VentilationControl

3. Agile method organization

In this project, we believe that following a **method agile** *(Scrum for example)* is the best thing to do. Because with this method we follow the iteration of conception, programming and test. With the test at the end of every iteration, it's easier to fix bugs or change what has been developed rather than waiting until the final product with other developing methods.

If we had to do this project with an agile method over **2 months**, we would have done **4 sprints of 2 weeks each**. We would have a list of prioritized objectives, product backlog, and two members would have had the role as the **scrum master** and **product owner**.

The *scrum master* would have been responsible for a smooth running of the sprints following the scrum method, and for reducing impediments and guiding the other team members for continuous improvement.

The *product owner* would have been responsible for the stakeholders and customers who would use the developed application.

We would have defined a sprint planning at the beginning of each sprint. We would have used a free version of the **JIRA** software for scrum setup.

As we would not be working full time every day on this project, daily scrum would not be relevant. Therefore we would do **20 minutes sprint reviews** every three days to see each other's progress, difficulties encountered, the remaining tasks, and to synchronize activities etc.

At the end of each sprint, we would have done a **sprint retrospective** to examine the work done, identify what went well and improvements, and create a plan for implementing the improvements.