Software engineering

Lucas Baclé, Clément Géhin, 5ISS A1
January 2021

Introduction

During this class, we had to manage a software development project using the Scrum agile methodology from specifications to deployment. In this regard, we have built a room management system for schools. We used continuous integration platform Jenkins to build, test and validate our software.

Scenario

The temperature sensor data of given rooms is retrieved periodically. We want to maintain an inside temperature between 18°C and 23°C. To ensure this, we remotely open and close the windows of each individual room based on the outside and inside temperatures.

The final sources are available in the following GitHub repository :
https://github.com/lucasbacle/insa-room-management

Design

Architecture

We have decided early on that sensors would periodically transmit data. This has the main advantage of allowing a loose coupling between the room management cloud application and data acquisition. This also avoids idle listening and wasted energy consumption on the sensors side. Based on this assumption we decided to store sensor data on our private “cloud”. This would require both a database management system (DBMS) and a way to read and write this data.

As we have never used NoSQL before, we selected MongoDB as our DBMS for education purposes. Since this class followed the service-oriented architecture one, we designed a REST API using the Spring Boot java framework. A draft of its documentation is attached at the end of this report.

With this decoupled architecture we have then been able to develop both a distinct logging web application and a periodic room management logic. As seen in Figure 1, both cloud-based softwares consume the REST API.

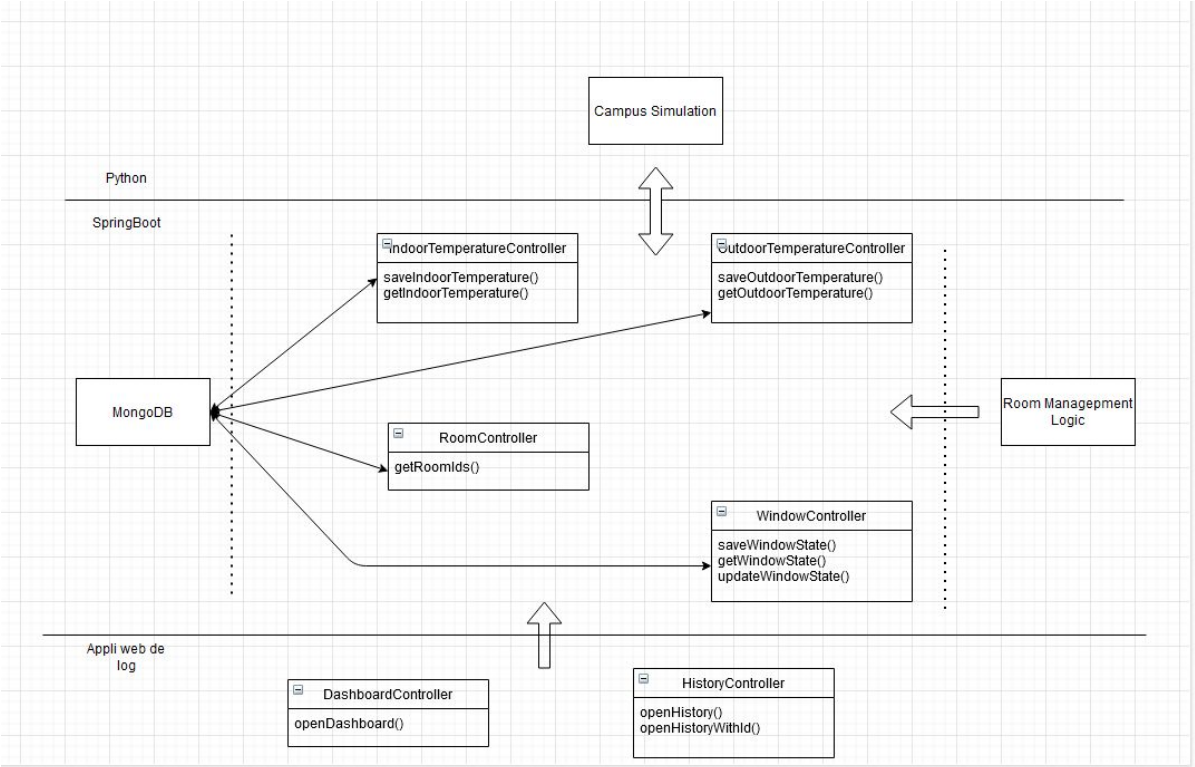


Figure 1 : general architecture of our system

REST API

We developed a REST API using a controller for each type of data. All controllers are more or less working in the same way. When a GET request is received, the corresponding controller method will retrieve data from our database and return a list of our model structure. For example, a GET request on /campus/rooms/2/temperature will return the list of temperatures from room number two.
In the same way, when a POST request is received, the corresponding controller method will parse the body of the request (JSON format) and store the extracted data in our database.
Finally, the PUT request sends an update to the campus sensor system to change the state of the windows of a room (open/close).

Web application

Alongside our REST API we developed an independent web application to display the data. The web application is using GET requests from our API to retrieve the data it needs. The application is composed of two pages : a dashboard and a history.

The dashboard always displays the latest temperature and window state for each room and for outside. It automatically refreshes itself every 30 seconds to keep the data up to date. The dashboard is completely dynamic, which means that if you add or remove a room, data will be displayed or not without changing the source code.

The history displays all the temperatures registered up to 720 (half a day) for a given room or for the outside. A combobox is used to select which history to display. Once again the history is fully dynamic, new rooms will be added automatically and old rooms removed.

To link the frontend and backend and be able to display the data we used Thymeleaf. Thymeleaf is integrated at spring boot and permits to create a model in the backend that will be used by the frontend. We can then display and use the data in html code as well as in javascript code. We also use the JavaScript library Chart.js to display our data on a graphic on the history page.

Room management logic

```
@Autowired
private RoomController roomController;

@Autowired
private WindowController windowController;

@Autowired
private IndoorTemperatureController indoorTemperatureController;

@Autowired
private OutdoorTemperatureController outdoorTemperatureController;

@Scheduled(fixedRate = 5000)
public void windowAutomation() {
    try {
        List<Integer> roomIds = roomController.getRoomIds();
        LocalTime limitTime = LocalTime.parse("21:00");
        LocalTime currentTime = LocalTime.now();
        int upperLimit = 23;
        int lowerLimit = 18;
        for(int roomId : roomIds) {
            if(currentTime.isAfter(limitTime)) {
                windowController.updateWindowState(String.valueOf(roomId), String.valueOf(false));
            }
            else {
                List<Temperature> indoorTemperatureList = indoorTemperatureController.GetIndoorTemperature(String.valueOf(roomId));
                List<Temperature> outdoorTemperatureList = outdoorTemperatureController.GetAllOutdoorTemperature();
                if ((indoorTemperatureList.get(indoorTemperatureList.size()-1).getValue() > upperLimit) && (outdoorTemperatureList.get(outdoorTemperatureList.size()-1).getValue() < upperLimit)) {
                    windowController.updateWindowState(String.valueOf(roomId), String.valueOf(true));
                }
                else if ((indoorTemperatureList.get(indoorTemperatureList.size()-1).getValue() < lowerLimit) && (outdoorTemperatureList.get(outdoorTemperatureList.size()-1).getValue() > lowerLimit)) {
                    windowController.updateWindowState(String.valueOf(roomId), String.valueOf(true));
                }
                else {
                    windowController.updateWindowState(String.valueOf(roomId), String.valueOf(false));
                }
            }
        }

    }
```

Figure 1 : room management logic

This listing presents our room management logic. Implemented as a separate spring boot component, this is in fact a Scheduled task that is running periodically. We directly use the different controllers because we developed this part inside of the same java project, but we also could have used RestTemplate to achieve the same results.

Campus simulator

Using Python 3, we developed a basic simulation software that can interact with our REST API. This software simulates a given amount of rooms in a campus and periodically sends sensor data to our cloud application populating our database. If the windows are opened, the room temperature adapts to the outdoors, else the temperature slowly rises because of human activities. Finally, a simple REST API permits to change the windows state.

Scrum process

A big part of this project was our management methodology. As introduced earlier in this report, we used the Scrum methodology. Vastly documented, we assume that the reader knows its basic principles and will only discuss our usage. We divided the project into 2 sprints of 2 weeks each.

Sprint 1

In our first sprint, we wanted to create the base of our project for the web application but also for oneM2M. So, we decided to put in place the oneM2M structure. We also developed the API controllers of temperature and created our MongoDB database.

Sprint 2

In our second sprint, we started to create the frontend of our web application and linked it to the backend. We then added the final controllers, for the rooms and windows. Finally, we decided, in a very agile way, to discard oneM2M in favor of a custom built python 3 campus simulator.

Continuous Integration & Deployment

We implemented unit tests to ensure that each route of our REST API presented the right behaviour. For each URI we tested what we obtained when called. If the result matches what we expected then the test is passed successfully. The tests are independent of the production data and are replayed every time an update is posted.

Using Jenkins, we created a new item, linked it with our local git repository and configured it to trigger a new build at each commit. To run the build we chose to go with the manual way by executing a windows batch script. Because our git repo contains two different folders, we first need to navigate into the java project to build it with maven.

Figure 2 : Jenkins build job configuration

We considered deploying this project on a cloud PaaS like Heroku, and even managed to deploy a simple Spring Boot prototype using GitHub Actions but sadly didn't have the time to do it with the final project.

Conclusion

We really enjoyed practicing our service-oriented architecture design knowledge and going a bit further than expected during this project. We acquired quite numerous skills from NoSQL concepts to continuous development tools.
We have however had a hard time understanding the OneM2M standard because of its generic nature. Even though this project made its working principles clearer, we decided not to use it.
Even if we are proud of our work, there still is room for improvement though - got the joke ?. With a better understanding of OneM2M we may have architected our system around it differently.

Annex 1 - REST API

Campus

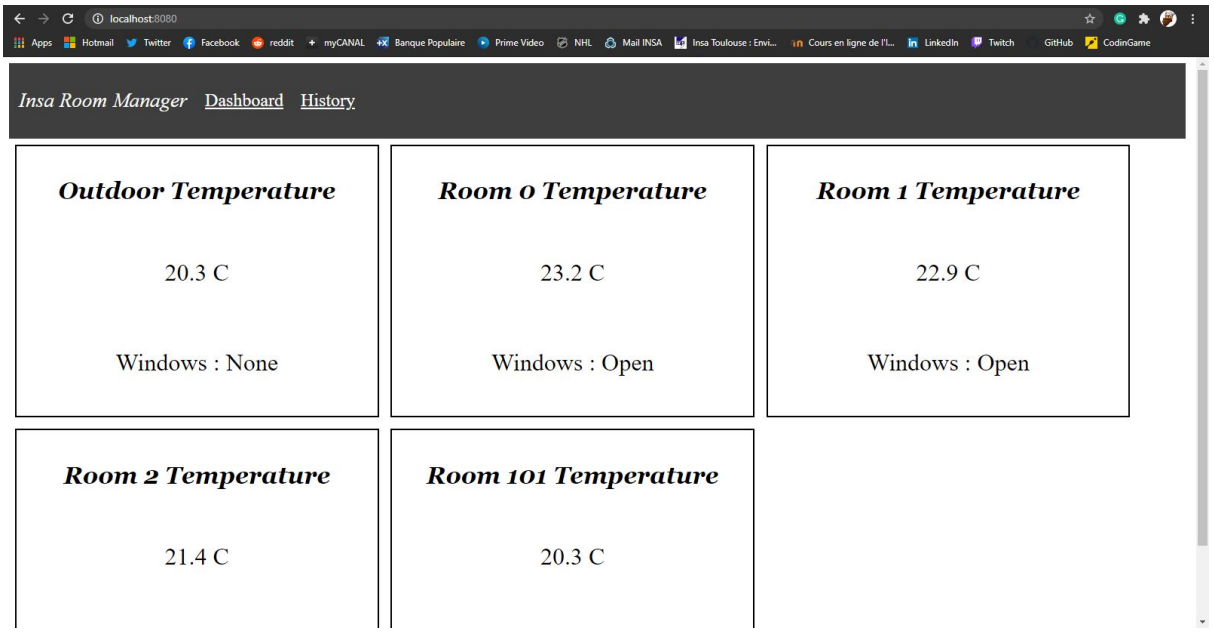
POST - /campus/temperature : store outdoor temperature
GET - /campus/temperature : retrieve outdoor temperature

Rooms

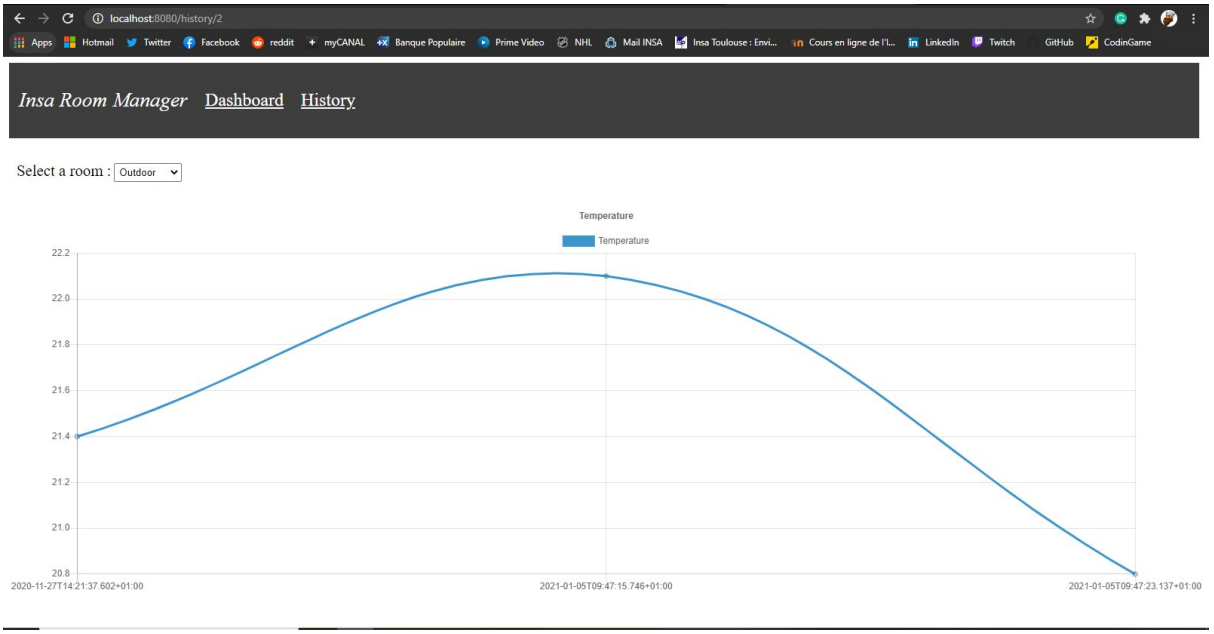
GET - /campus/rooms : retrieve registered room ids
POST - /campus/rooms/{id}/temperature : store room temperature
GET - /campus/rooms/{id}/temperature : retrieve room temperature

POST - /campus/rooms/{id}/windows : store current window state
GET - /campus/rooms/{id}/windows : retrieve current window state
PUT - /campus/rooms/{id}/windows : modify window state

Annex 2 - Screenshot of the web logging app



Dashboard web page



History web page