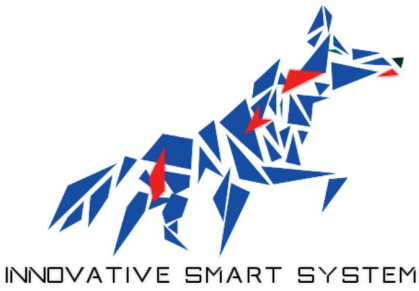



INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Service Based Architecture and Software Engineering:
# Automatic Management of INSA's rooms

Authors:
Julien Chouvet
Jean Baptiste Laffosse

Tutor:
Guillaume Garzone
garzone@laas.fr

# Summary

# Introduction

Within the course Service based Architecture and Software Engineering, we apply our knowledge on a Service Oriented Architecture (SOA) practical application: develop a web application to manage the INSA's rooms. The main goal of the application is to automatically close or open the windows, the doors, turn on or off the lights, switch on or off the alarm. All along the project, we used the AGILE Scrum to schedule all the main steps and sprints.

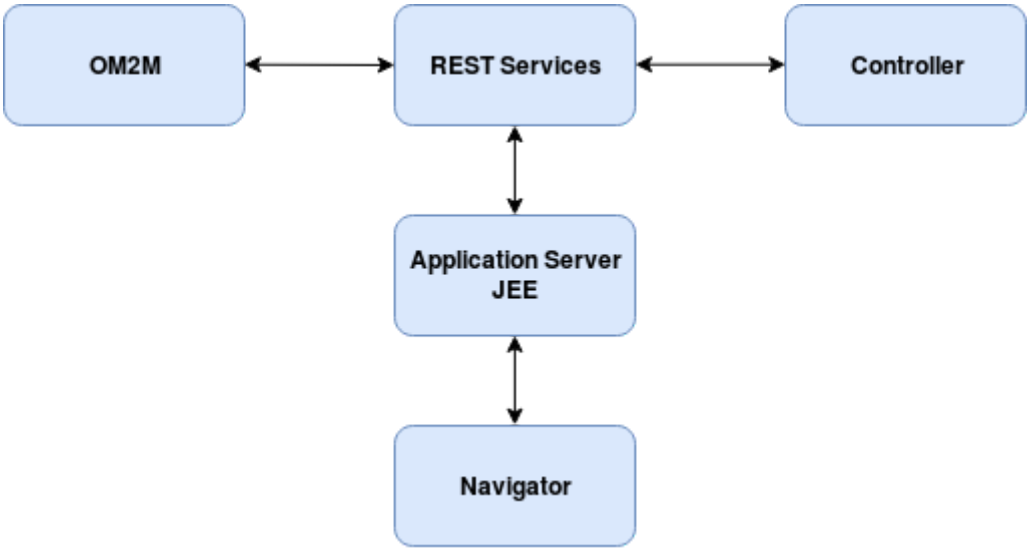



*Figure 1: Global Architecture of the SOA Web Application*

All the resources, such as sensors and actuators, are declared thanks to the OM2M platform under the Middle Node Common Service Entity (MN-CSE). To interact with OM2M platform to create, update, delete or retrieve sensors values, we create REST services which can communicate with the platform using HTTP requests. In order to take decisions according to the specification of the customer, we create a controller application, it will communicate with the ressources through the REST services and HTTP requests. Finally, we implement a web interface using a JEE application server for viewing the history of actions and give the possibility to act with the resources.

You can find all our resources on the following github repository:

https://github.com/JulienChvt/Service_Oriented_Architecture.git

# Specifications

For the application to manage the room, we can define different simple use cases:


*Figure 2: Use Case Diagram for the Room Management Application*

- We want to retrieve periodically the values of the indoor and outdoor temperature sensor. For example, we can set the period at 1 minute. We know that the temperature doesn't change quickly, we do not need to check the temperature very often.
- According to the values of the indoor and outdoor temperature, we can decide if we need to cool down the room. We can use the following condition to decide if we open the window: $T_{ext} < T_{int}$ *and* $T_{ext} \in [18°C; 27°C]$
- Then, outside the working hours, such as 8 am to 6 pm, if there is nobody, we can close the room, it means close the door, close all the windows and turn off the light.
- However, if there is a presence after 10 pm, we have to
- The user can act directly on the resources system.

# AGILE Scrum Method

After defining all the functions we want to implement in our application, we can plan all the sprints on https://cloud.icescrum.com/p/ROOMMA/#/project.

Before defining the sprints, we need to create features we want to show at the customer at the end of the sprint. From the Use Case diagram above, we can define the following features:

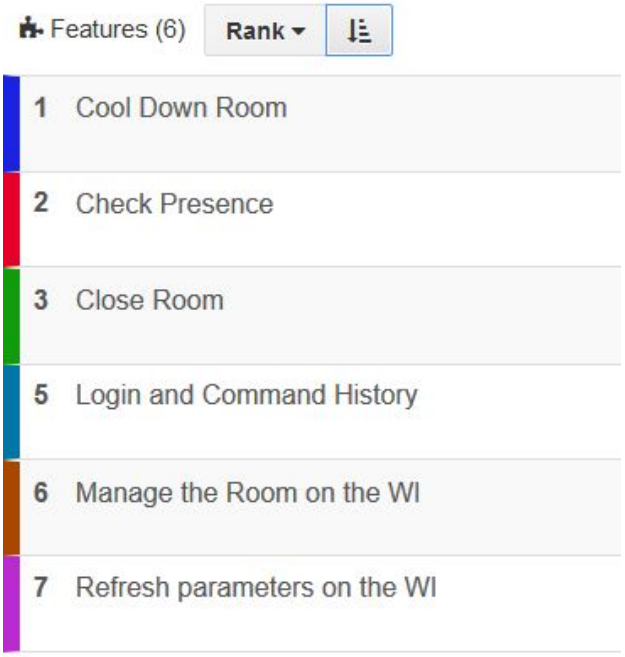



*Figure 3: Features of our application*

Then, we can declare stories in order to finish the features, and we can spread the stories in the sprints according to the features we want to finish and the time allocated. The sprint duration is about 12 days.


*Figure 4: Sprint and story of our application*

Finally, inside every stories we can define and give the tasks at someone within the team.

# Conception

Our application is composed by 4 different Java projects and one bash script:

- ***run.sh*** : this bash script is located in a the directory called *eclipse-om2m-v1.3.0*. This script removes the *in-cse* and *mn-cse database* directories in order to start with a clear OM2M platform. Then it will start the *in-cse* and *mn-cse* and start a web browser at the address corresponding to the OM2M platform (here *http://localhost:8081/webpage*).

- ***fr.insat.om2m.initRooms*** which creates the environment on OM2M. It is composed by a room with 7 virtuals sensors (indoor and outdoor temperature, windows sensor, door sensor , lights sensor, presence sensor and alarm). Thus, by running this project you will create a room called “GEI_213”, then create and initialise the sensors. Moreover, in order to simulate the indoor and outdoor temperature sensors, this application creates, every 10 seconds, two new *content instance* in OM2M, each representing a new temperature coming from the sensors.

- ***RestServicesRoom*****:** as we had to build an application based on a SOA architecture, we created in this project the REST services. We created 7 different services, one for each sensors of the room. You can find in appendix the class diagram of the RestServicesRoom. All these services can be used through the following HTTP methods:
  - **GET:**
    *http://localhost:8080/RestServicesRoom/webapi/{sensor}/{room}*
    By applying a GET on this URL, the corresponding service will make a GET on the corresponding OM2M ressource (*AE* {room} and *container* {sensor}) to retrieve the information. Then, it will return you this information.

  - **POST:**
    *http://localhost:8080/RestServicesRoom/webapi/{sensor}/{room}/{state}*
    By applying a POST on this URL, you will create a new *content instance* in the *AE* {room}, in the *container* {sensor}, with the *con* value {state}. Because our sensors and actuators, this new *content instance* represents an order on an actuator, to close the windows or to switch on the light, for example.

- ***RoomManagement:*** this one is in charge of the automation of the management of the room. It uses the REST services describe before to retrieve information from the sensors of the room (GET) and depending on these information sends order to the actuators (POST). We decided to implement the following tasks:
  - If tempExt < tempInt et 17<tempExt<28 so open the windows. Otherwise close the windows.
  - Close doors, windows and switch off the lights if it's earlier than 8 am or later than 6 pm and there is nobody in the room.

- If there is someone after 22h => Turn on the alarm.

- ***RoomManagementAppWeb:***

With two servlet, and 3 jsp files to manage the registration of the user and then to manage the room.

  - From the http://localhost:8080/RoomManagementAppWeb/index.html we send a get request is send to the servlet *LoginVerify.java* to display the login page thanks to the *success.jsp*.
  - Then from the login page, you can login with your username and your password, a post request is again send to the servlet *LoginVerify.java.* If the input username or the input password are wrong, the servlet will redirect the user at the *error.jsp* page to make another login process.
  - If the username and password are good, the servlet will display the *success.jsp* page to manage the room.
  - From the *success.jsp* page, we can:
    - See the **history of all the commands** thanks to a variable sent and update by the servlet everytime the *success.jsp* page is displayed.
    - **Refresh** all the room parameters thanks to a GET request sent to the servlet with attributes “action” and “Room Number”. Then, the servlet get all the values from the OM2M resources thanks to the REST services, and finally request to display the *success.jsp* page with all the parameters updated.
    - **Manage** the room parameters thanks to a POST request sent to the servlet with attributes “action” and “Room Number”. Then, the treat the order according to the value of “action”, and send a POST to the OM2M resources thanks to the REST services. Finally, we update the room parameters and the servlet request to display the *success.jsp* page with all the parameters updated.
  - We store all the room inside a ArrayList<RoomProperties> where the RoomProperties are the parameters of the room including its ID. You can find in appendix the class diagram of the ***RoomManagementAppWeb*** application. We implement this class to access and store all the parameters we want to display on the interface.

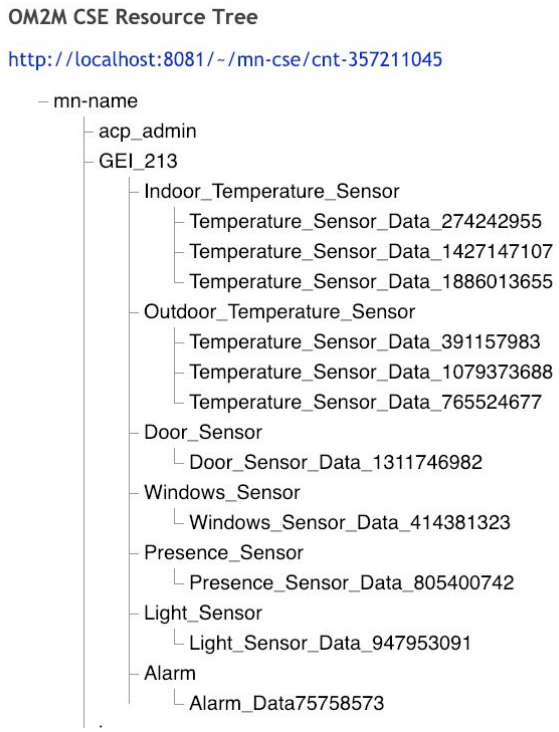
We didn’t manage to connect the web interface to the REST services because of the Httpclient library, but we implement all our function in the servlet to access at the OM2M resources and display the good value of the room in the web interface.

# Tests of our application

In this part, we are going to present some tests mad to ensure our application is working well.

## Implement the OM2M resources

First, we are going to use initialise the resources on OM2M. After running the *fr.insat.om2m.initRooms*, we can see on the picture below that we have created an *AE* representing the room GEI_213 that contains 7 containers, one for each sensor. All these containers are initialised with a *content instance* representing the initial values of the sensors. We can also notice that for the indoor and outdoor temperature sensors, there is more *content instance* because our application is creating periodically new values, as explained previously.




*OM2M mn-cse tree after initialisation of the resources*

```
GEI_213 created
Indoor temperature sensor created in GEI_213
Outdoor temperature sensor created in GEI_213
Door sensor created in GEI_213
Windows sensor created in GEI_213
Presence sensor created in GEI_213
Light sensor created in GEI_213
Alarm created in GEI_213
Door sensor updated for GEI_213
Presence sensor updated for GEI_213
Windows sensor updated for GEI_213
Light sensor updated for GEI_213
Alarm updated for GEI_213
Virtual indoor temperature sent = 19
Virtual outdoor temperature sent = 18
Virtual indoor temperature sent = 24
Virtual outdoor temperature sent = 25
Virtual indoor temperature sent = 20
Virtual outdoor temperature sent = 22
```

*Log of the execution of the fr.insat.om2m.initRooms application*

# Test of the controller

Then, we have to test that our controller is managing correctly the room, depending on the rules mentioned before. At the time, we can check if our REST services are running correctly and that the OM2M resources are well updated.
Here are the results of the tests:

- **If tempExt < tempInt et 17<tempExt<28 so open the windows. Otherwise close the windows.**

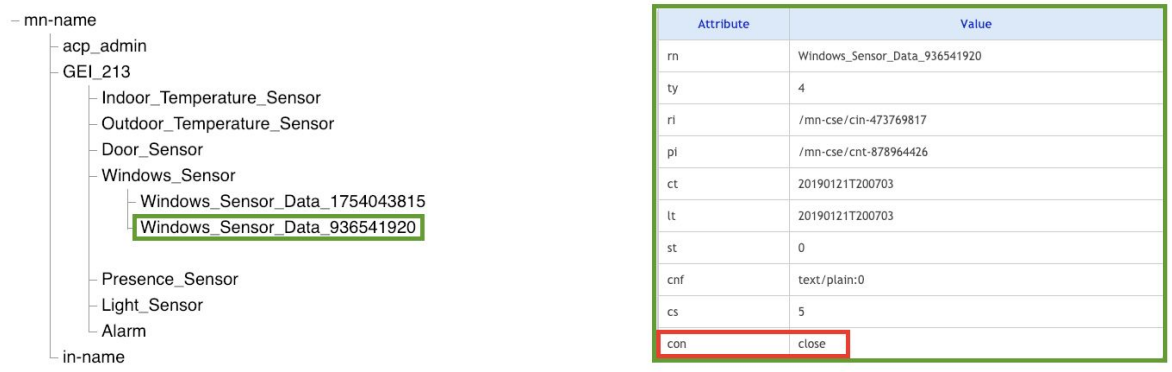
```
---------- Room Management Controller ----------
Indoor temperature: 22
Outdoor temperature: 19
Too hot! Openning windows!
Current hour: 20
Presence sensor: true
```

*Log of the Controller - Need to open the windows according to the if/else rule*


*OM2M tree - The room has been opened*

```
---------- Room Management Controller ----------
Indoor temperature: 16
Outdoor temperature: 19
Too cold! Closing windows!
Current hour: 20
Presence sensor: true
```

*Log of the Controller - Need to close the windows according to the if/else rule*

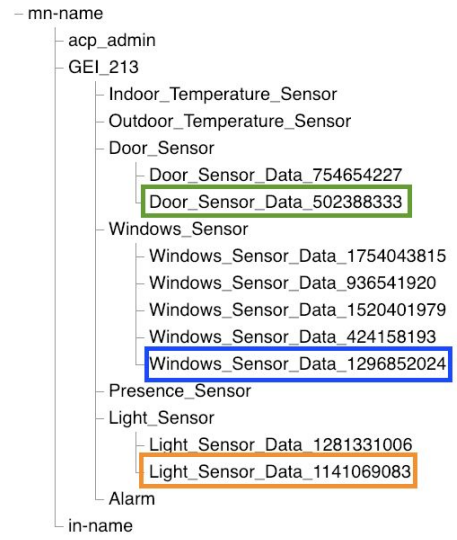



*OM2M resources tree - The room has been closed*

- **Close doors, windows and switch off the lights if it's earlier than 8 am or later than 6 pm and there is nobody in the room.**

```
----------- Room Management Controller -----------
Indoor temperature: 21
Outdoor temperature: 19
Too hot! Openning windows!
Current hour: 20
Presence sensor: false
Out of working hours and nobody in the room => Closing doors an windows and switching the lights off
```

*Log of the Controller - Close doors and windows and switch off the light according to the if rule*




*OM2M tree showing modified resources*

| Attribute | Value |
|---|---|
| rn | Door_Sensor_Data_502388333 |
| ty | 4 |
| ri | /mn-cse/cin-958720049 |
| pi | /mn-cse/cnt-980887933 |
| ct | 20190121T202533 |
| lt | 20190121T202533 |
| st | 0 |
| cnf | text/plain:0 |
| cs | 5 |
| con | close |

*OM2M door resource - State changed to close*

| Attribute | Value |
|---|---|
| rn | Windows_Sensor_Data_1296852024 |
| ty | 4 |
| ri | /mn-cse/cin-683815184 |
| pi | /mn-cse/cnt-878964426 |
| ct | 20190121T202533 |
| lt | 20190121T202533 |
| st | 0 |
| cnf | text/plain:0 |
| cs | 5 |
| con | close |

*OM2M windows resource - State changed to close*

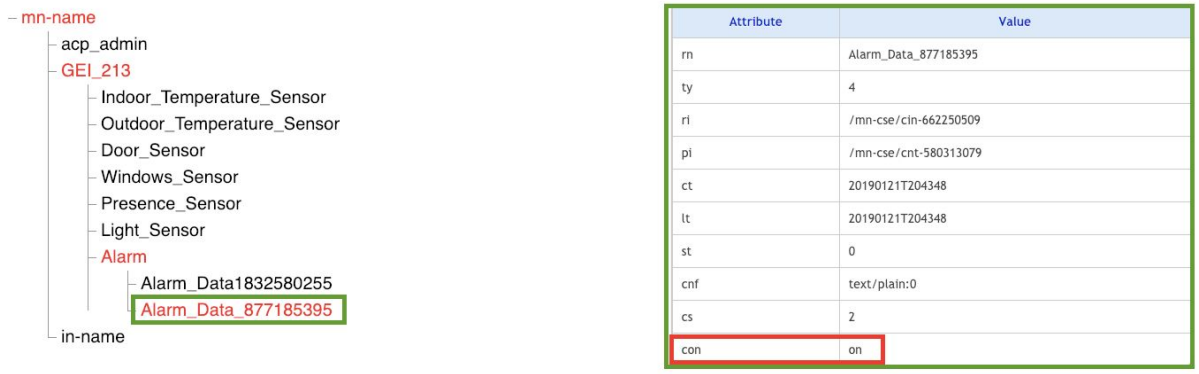
| Attribute | Value |
|---|---|
| rn | Light_Sensor_Data_1141069083 |
| ty | 4 |
| ri | /mn-cse/cin-756940125 |
| pi | /mn-cse/cnt-257133390 |
| ct | 20190121T202533 |
| lt | 20190121T202533 |
| st | 0 |
| cnf | text/plain:0 |
| cs | 3 |
| con | off |

*OM2M lights resource - State changed to off*

- If there is someone after 22h => Turn on the alarm

```
---------- Room Management Controller ----------
Indoor temperature: 23
Outdoor temperature: 25
Presence sensor: true
Current hour: 23
Alarm ON!
```

*Log of the Controller - Turn on the alarm according to the if rule*

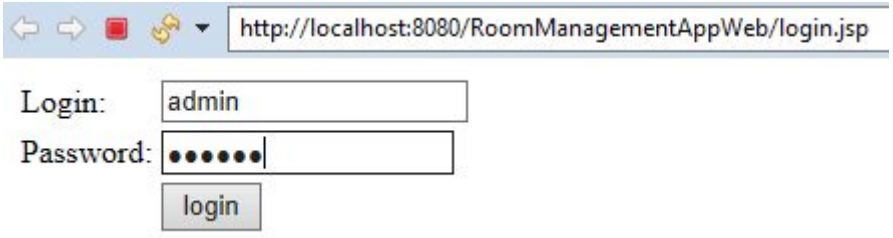



*OM2M resources tree - The alarm has been turned on*

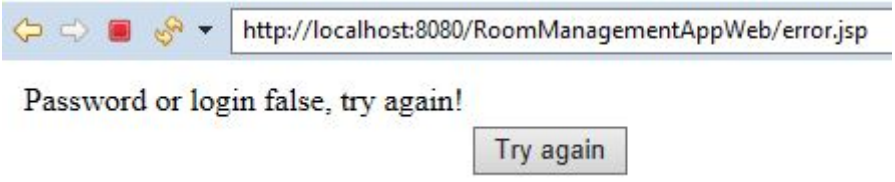
# Web Interface Test

For the web interface, we can make some quick test to show of how the interface works.

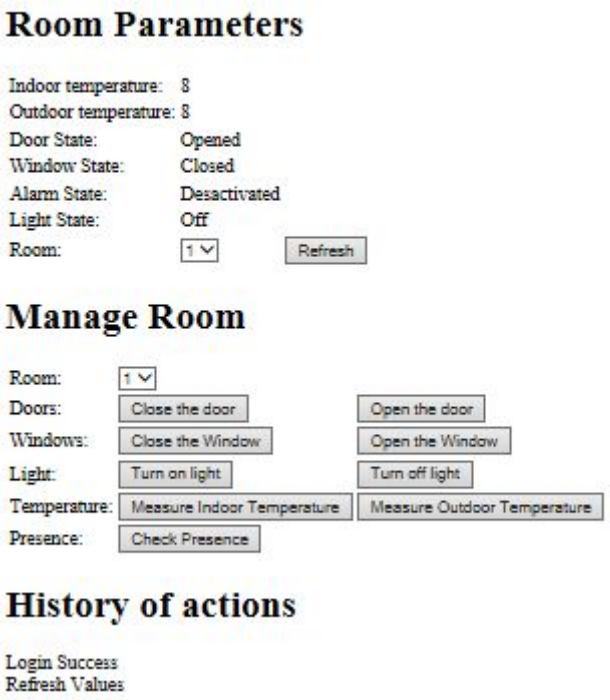
- Login to the Web Interface:




- Failed to login, you can go back on the login page:

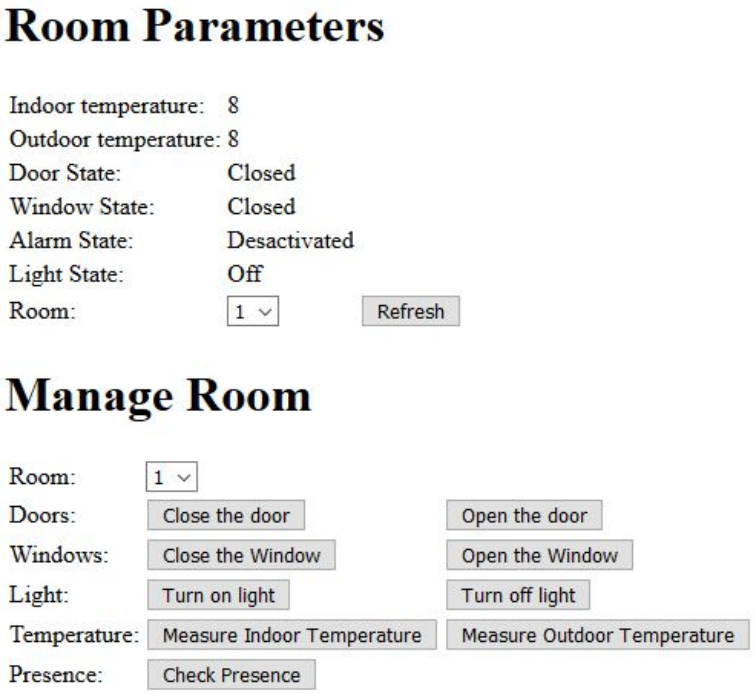



- Access to the managing page:

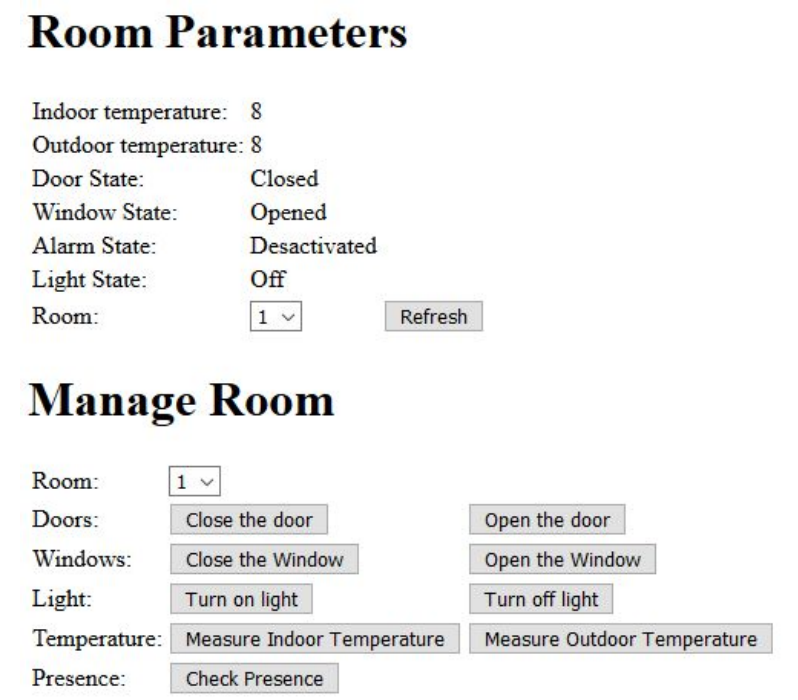



- On the managing page, we can refresh the parameters, see the history of commands, but we can also act directly to the room. To test the interface, we change directly the value of variables in the room properties. As we didn't manage to connect the interface with the REST services, we can only see fake values. The interface work actually well.

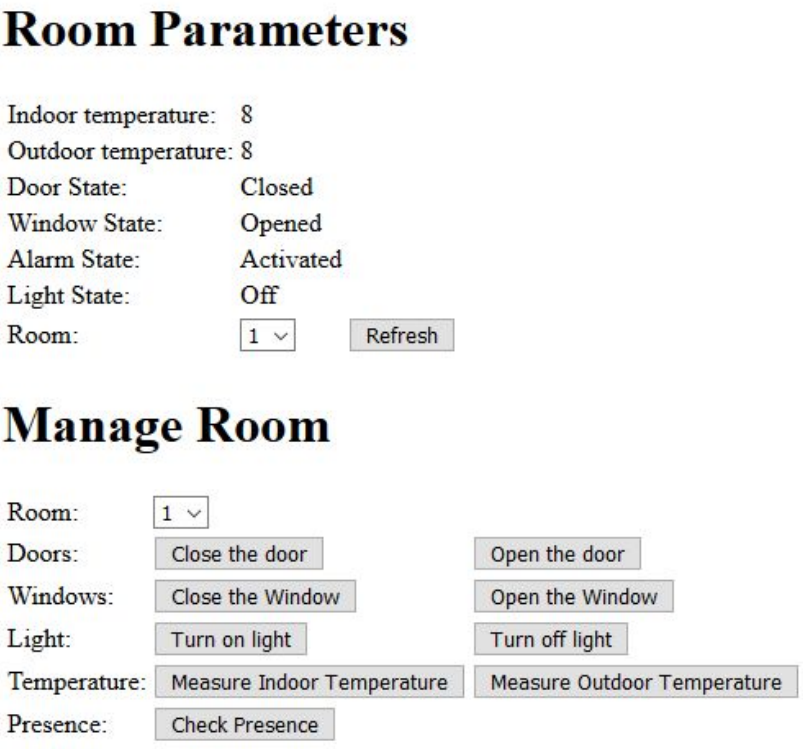
Everytime we choose a button, we have to select the room number and in the servlet we use this number. Then, we can for example close the door:



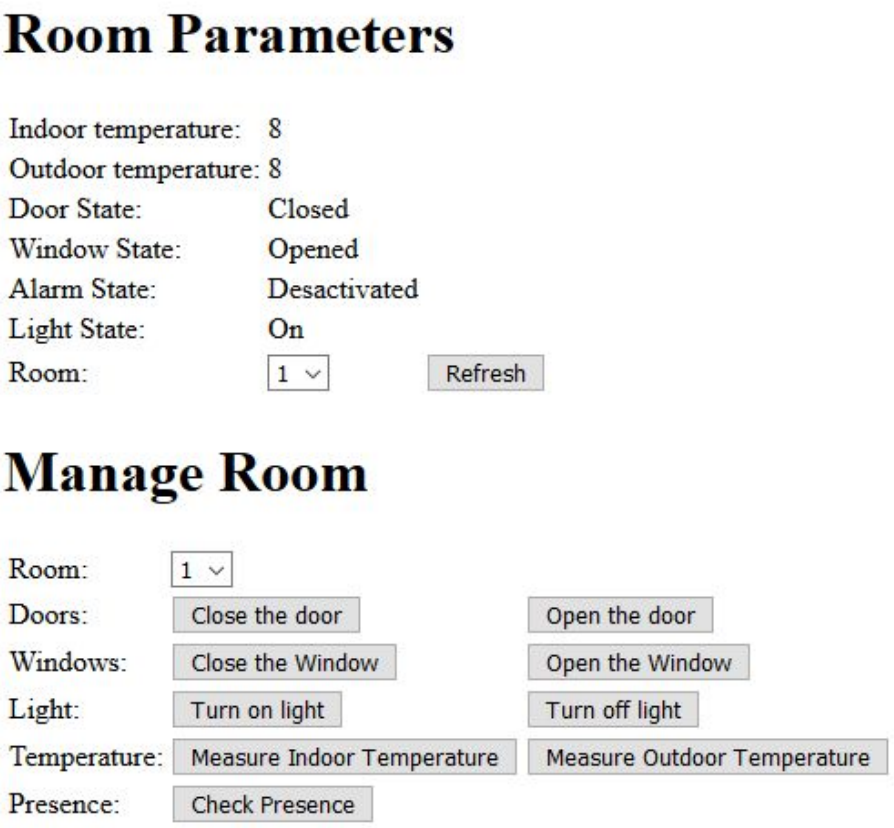
We can also open the windows, we have one button to control all the windows of the room:




Then we can check the presence inside the room, is there is someone inside the room, the alarm is triggered:



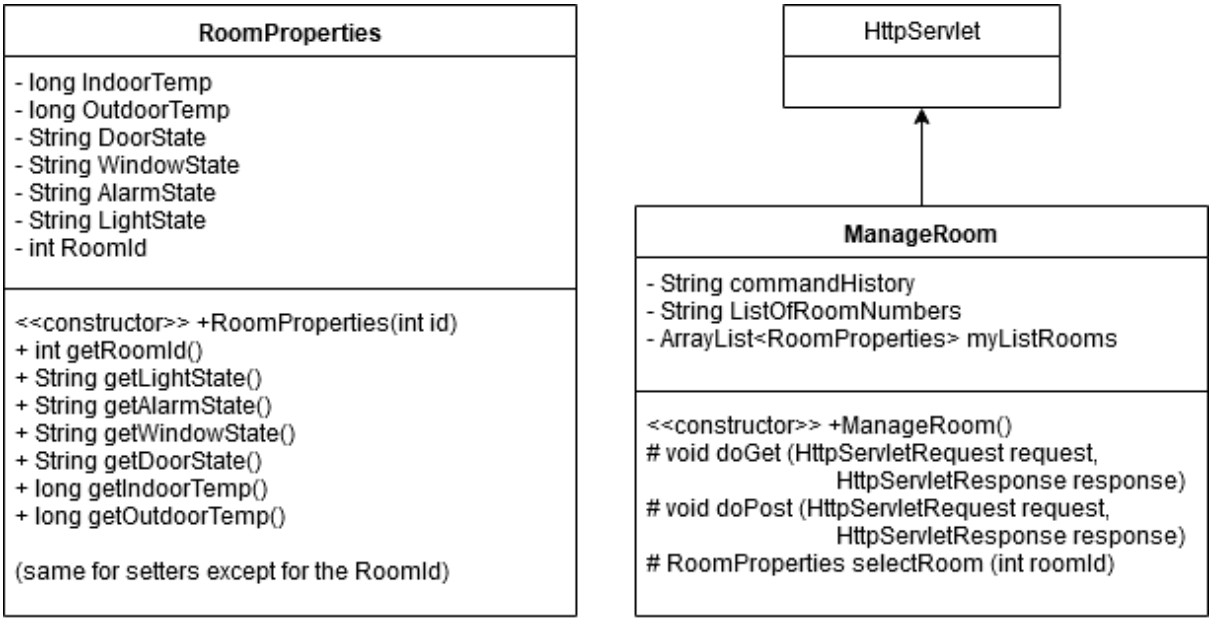
Finally, we can also turn on/off all the light inside the room:



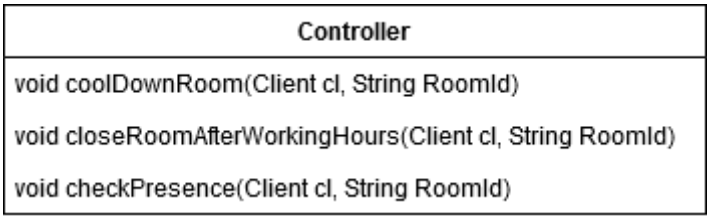
# Appendix

## Class Diagrams

**Web Interface Class Diagram**




**RoomManagement Class Diagram**



## Rest Services Class Diagram