UNIVERSITÀ DI PISA

Department of Information Engineering
MSc in Artificial Intelligence and Data Engineering
and Computer Engineering

FitSense: Smart Gym Management System

IOT Project

Federica Perrone
Irene Catini

Academic Year 2022/2023

Contents

1 Introduction

The project is about smart gym and is a system that allows people to monitor the status of gym rooms by automatically managing air conditioner system, dehumidifier system and presence detection system. The network consists of a border router, an undefined number of scattered nodes throughout the gym areas and two applications which manage all nodes (Cloud application and Remote control application). From the project specifications the actuator nodes must use the **COAP** protocol, instead sensor nodes must use the **MQTT** protocol. The Remote control application is able to communicate with actuator nodes using **COAP** protocol, and the Cloud application communicates with sensor nodes using **MQTT** protocol.

All the developed code is available on GitHub link:
(https://github.com/Fedeperrone98/FitSense)

1.1 Technologies used

1.1.1 Software

- Ubuntu 18.04 LTS
- Contiki-NG v4.7 on Docker (https://github.com/contiki-ng/contiki-ng)
- C-language
- Java-language
- Docker
- MySQL database
- Californium (Coap Java library)
- Paho (MQTT Java library)
- mysql java library
- Grafana

1.1.2 Hardware

- General purpose machine
- nRF52840 dongle board

2 The project idea

The name "FitSense" combines the concepts of fitness and sensory data, reflecting the focus on collecting and analyzing data related to gym activities. The term "Smart Gym Management System" conveys the purpose of the project, which is to optimize the management and operations of the gym through intelligent monitoring and control capabilities.

The project is designed to fit an undefined number of areas, any area will be monitored by one or more nodes of the network. Each node must be identified with two identifiers, one relative to the area (area id) and one related to nodes on the same area (node id).

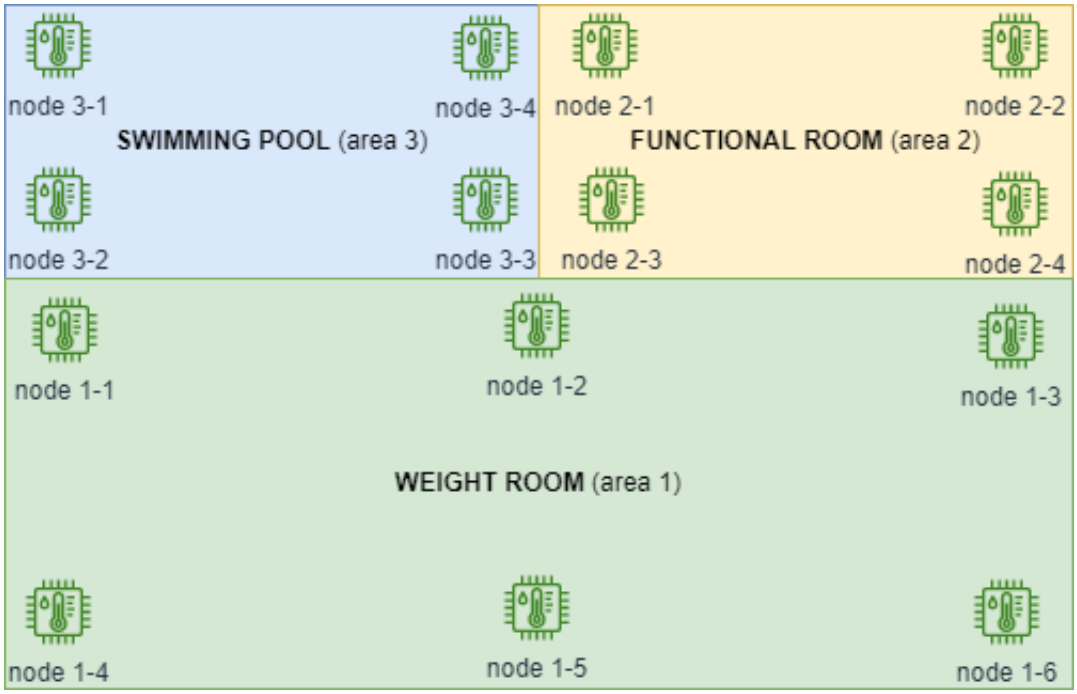

Figure 1: An example of areas and nodes

Each node is able to measure 3 values:

- Temperature
- Humidity
- Number of people in the area (Presence)

3 Project Design

3.1 UML diagram

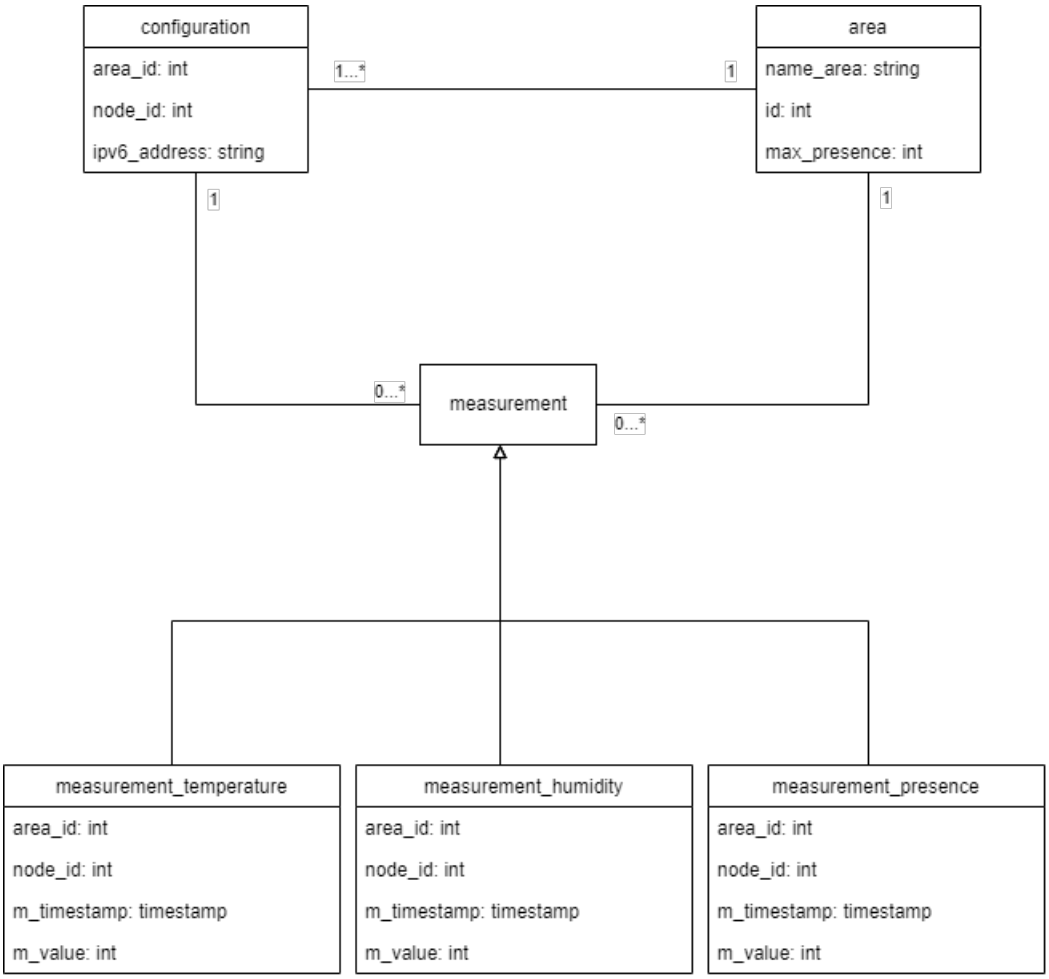

Figure 2: UML diagraam

3.2 The Remote Control Application State Machine

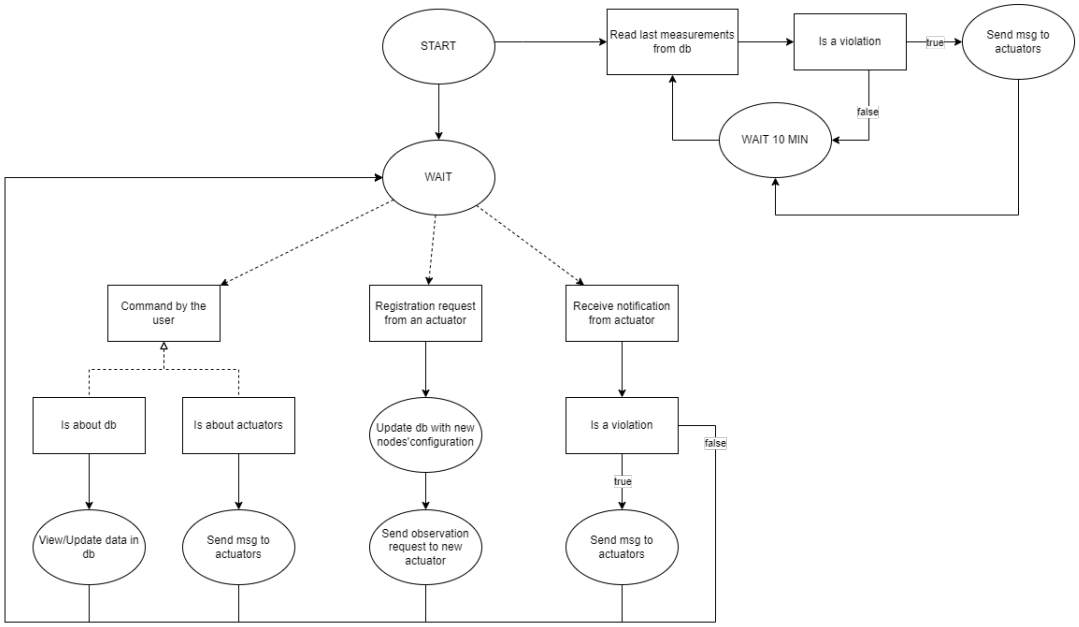

Figure 3: Remote Control Application State Machine

3.2.1 User command

The **user** can interact with the **terminal interface** of the Remote control application to check the network status, modify data on the database and send messages to nodes. These are all operations that it's possible to do.

- View area
- View configurations
- View last temperature per area
- View last humidity per area
- View last presence per area
- Add area
- Update area max presence
- Get current temperature
- Get current humidity
- Get current presence
- Turn on/off air conditioner
- Turn on/off dehumidifier

- Turn on/off semaphore
- Exit

3.3 The Cloud Application State Machine

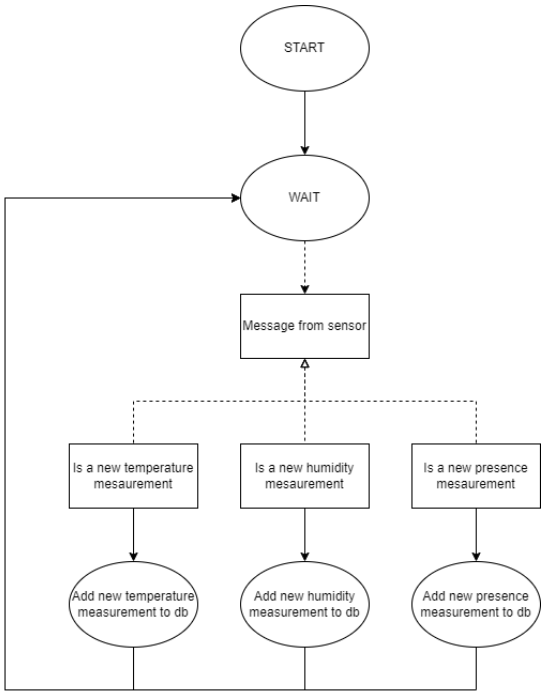

Figure 4: Cloud Application State Machine

3.4 The Actuator Node State Machine

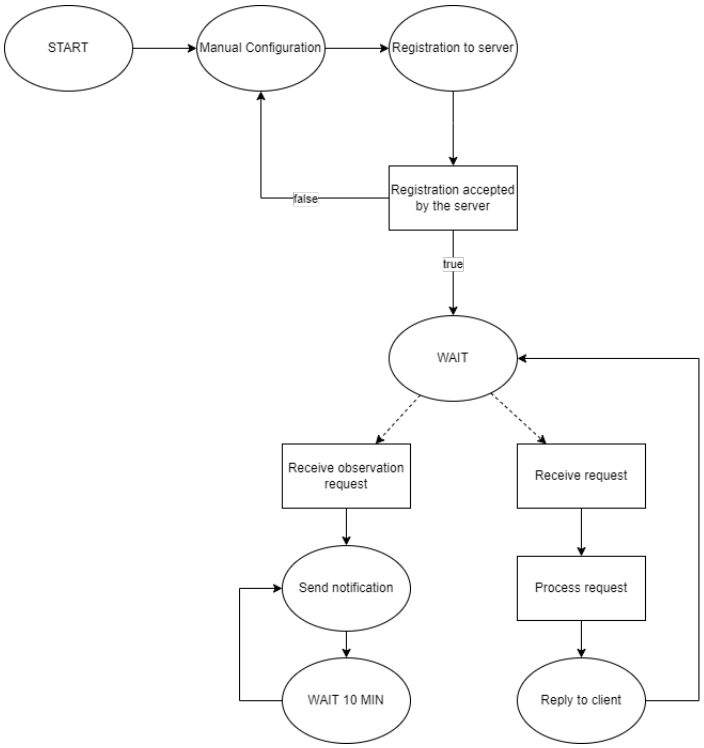

Figure 5: Actuator node State Machine

3.5 The Sensor Node State Machine

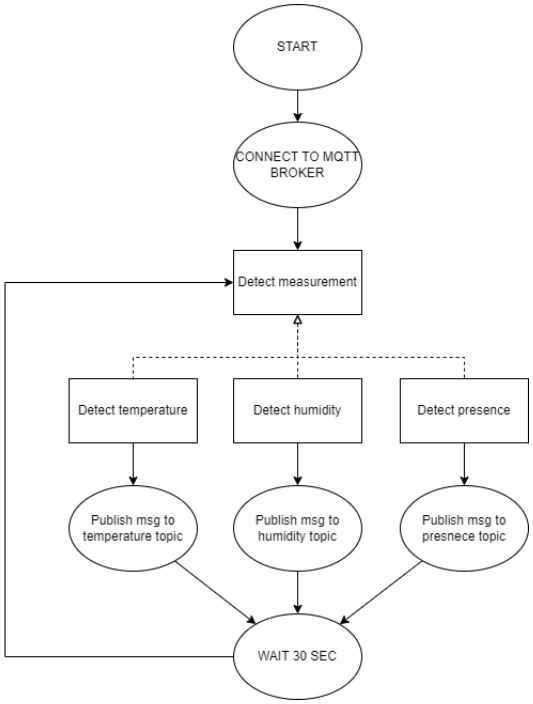

Figure 6: Sensor node State Machine

3.6 The Database Design

The following are the sql scripts used to build the database.

3.6.1 The Configuration Table

```sql
CREATE TABLE IF NOT EXISTS configuration (
    area_id int(11) NOT NULL,
    node_id int(11) NOT NULL,
    ipv6_address varchar(100) NOT NULL,
    PRIMARY KEY (area_id, node_id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 7: Configuration Table code

- *area_id*: area identifier in which the node is
- *node_id*: node identifier inside an area
- *ipv6_address*: indicates the IPv6 address of the COAP node

3.6.2 The Areas Table

```sql
CREATE TABLE IF NOT EXISTS area (
    id INT(11) AUTO_INCREMENT PRIMARY KEY,
    name_area varchar(100) NOT NULL,
    max_presence int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO area (id, name_area, max_presence) VALUES
(1, "weight room", 40),
(2, "functional room", 30),
(3, "swimming pool", 20);
```

Figure 8: Area Table code

- *id*: area identifier
- *name_area*: name of the area
- *max_presence*: maximum number of people which can stay in this area at the same time

3.6.3 The Temperature Measures Table

```
CREATE TABLE IF NOT EXISTS measurement_temperature (
    area_id int(11) NOT NULL,
    node_id int(11) NOT NULL,
    m_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    m_value int(11) NOT NULL,
    PRIMARY KEY(area_id, node_id, m_timestamp)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 9: Temperature Table code

- *area_id*: area identifier
- *node_id*: node identifier inside the area
- *m_timestamp*: time at which the measurement is taken
- *m_value*: value of the measurement

3.6.4 The Humidity Measures Table

```
CREATE TABLE IF NOT EXISTS measurement_humidity (
    area_id int(11) NOT NULL,
    node_id int(11) NOT NULL,
    m_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    m_value int(11) NOT NULL,
    PRIMARY KEY(area_id, node_id, m_timestamp)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 10: Humidity Table code

- *area_id*: area identifier
- *node_id*: node identifier inside the area
- *m_timestamp*: time at which the measurement is taken
- *m_value*: value of the measurement

3.6.5 The Presence Measures Table

```
CREATE TABLE IF NOT EXISTS measurement_presence (
    area_id int(11) NOT NULL,
    node_id int(11) NOT NULL,
    m_timestamp timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
    m_value int(11) NOT NULL,
    PRIMARY KEY(area_id, node_id, m_timestamp)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figure 11: Presence Table code

- *area_id*: area identifier
- *node_id*: node identifier inside the area
- *m_timestamp*: time at which the measurement is taken
- *m_value*: value of the measurement

4 Implementation

This chapter will explain the node and applications implementation. The nodes source codes are written in **C language** by using the **Contki-NG** project, the applications' source code is written in **java**

4.1 The Border Router

The **Border Router** node connects the nodes of the network to internet by allowing the communication with applications, in order to do this it uses the **tunslip6** program and the **RPL** routing protocol. The Border Router source code is the same that is in the example folder of the official Contiki-NG repository, for further discussion see the link https://github.com/contiki-ng/contiki-ng/tree/develop/examples/rpl-border-router.

4.2 The Actuator Node

At the beginning, the actuator nodes perform the **manual configuration** procedure, in order to register to the *Remote Control Application*:

- Starting of the node
- Red LED flashes every 0.5 sec while waiting for user interaction
- The user presses the button as many times as the value of the area_id, and the red LED stops flashing
- As soon as the user stops pressing the button, the Green LED remains steady for 2 seconds
- The Red LED starts flashing again while waiting for the node_id configuration
- The user presses the button as many times as the value of the node_id
- As soon as the user stops pressing the button, the Green LED remains steady for 2 seconds
- The node sends to *Remote Control Application* a configuration request
- If the configuration request is accepted the node is ready, otherwise the led keeps blinking and the node needs to be reconfigured

After performing the manual setup procedure, the *Remote Control Application* knows the **IPv6 address** of the registered nodes, and it can use the latter to communicate with the nodes, using **COAP**.

All the messages are in **JSON** format for the parsing efficiency and for the smaller message size than the other encodings such as XML. Values recorded by nodes with the **measurements** are **simulated** with random values.

A COAP node has **seven resources** and each resource can implement the GET operation to return its status. Only a subset of the resources implement the PUT operation to update its status (PUT was chosen instead of POST because updating a resource generally corresponds with a complete replacement of all parameters). Some resources are observable, so they implement trigger() operation to notify the *Remote Control Application* of the status. The COAP node has the following resources:

- *res_configuration*: manages the identifiers area_id and node_id, implements GET operation.
- *res_temperature*: manages the current temperature measurement and the measurement period, implements GET and trigger.
- *res_humidity*: manages the current humidity measurement and the measurement period, implements GET and trigger.
- *res_presence*: manages the current presence measurement and the measurement period, implements GET and trigger.
- *res_air_conditioner*: deals with events regarding its status (on or off), implements GET and PUT.
- *res_dehumidifier*: deals with events regarding its status (on or off), implements GET and PUT.
- *res_semaphore*: deals with events regarding its status (on or off), implements GET and PUT.

The *Remote Control Application* registered itself to the resources of each COAP nodes that implement the **trigger** operation (**COAP observing** procedure). In this way, the node can **notify** the *Remote Control Application* of its resources status without the need of a GET request.

4.3 The Sensors Node

The sensor nodes do not perform any configuration procedure, but each node has a **static configuration** indicating the area in which the node is positioned. For this reason, the area_id and the node_id are stored in local constants.

The MQTT communication doesn't need addresses, but it's needed a **broker** that acts as a central server to which all MQTT nodes and the java *Cloud application* are connected. **Mosquitto** is used as broker on the same machine where the *Cloud application* runs.

Each node registers with the broker (all nodes know the address of the broker) with three topics: temperature, humidity and presence. The broker manages the address of registered nodes internally, so each MQTT message must go through the broker and contain only the topic and body. Also in this case, all the messages are in **JSON** format.

Every 30 sec the nodes detect new measurements and publish that on the relative topic.

4.4 Applications

Both the applications (Remote control application and Cloud application) print log info in the following format:

- info message: [!]
- success message: [+]
- error message: [-]
- sending message: >
- receiving message: <

4.4.1 The Remote control application

As already mentioned in the previous chapters, the Remote control application is written in **java** and is able to communicate with **COAP** protocol using the **Californium** library.

The Remote control application consists of **3 threads**:

- A thread listening the registration request from actuator nodes and observing the resources exposed by the nodes. This thread exposes a resource that implements a POST operation to handle registration requests.
- A thread that executes periodically to read measurements stored in the database, and accordingly implements the control logic.
- A thread listening the user commands.

In the images below we show the log information printed by this application during execution.

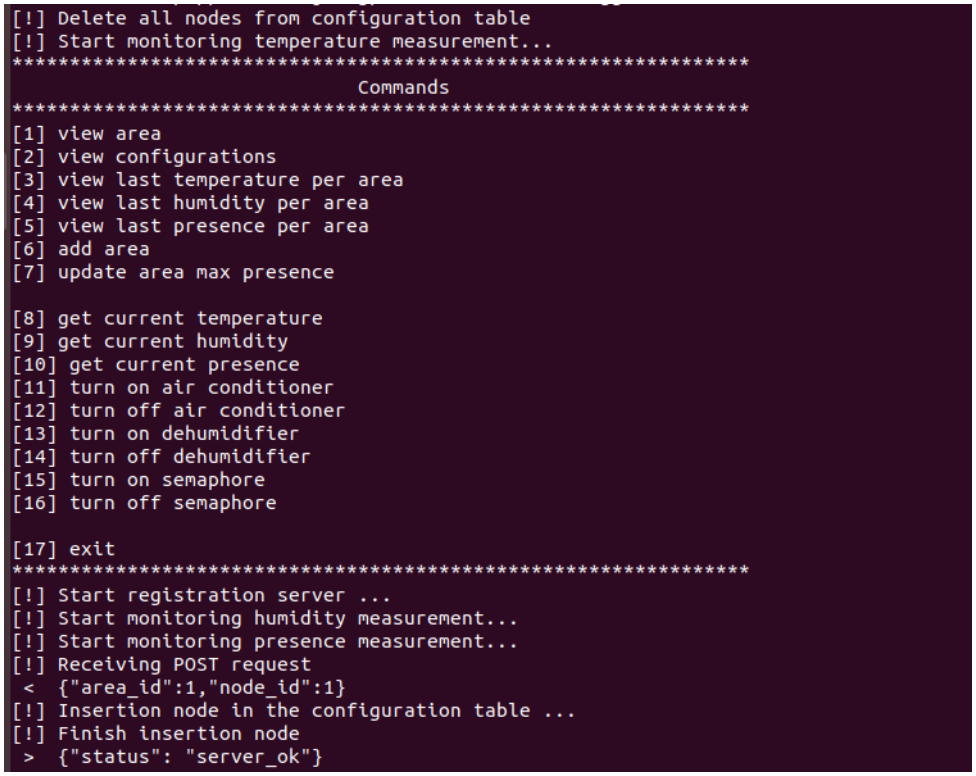

Figure 12: Log Info about registration procedure

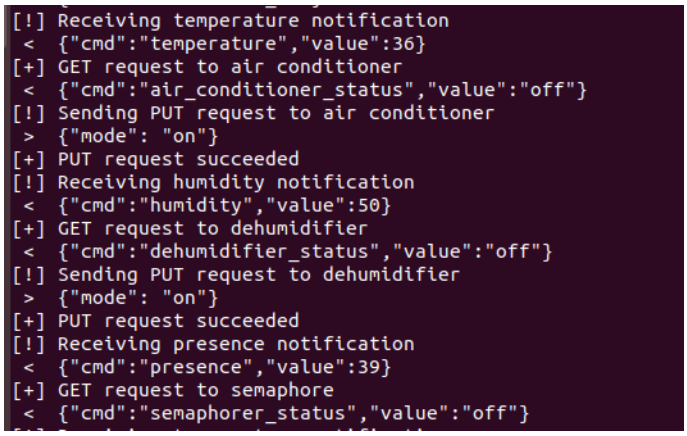

Figure 13: Log Info about notification

4.4.2 The Cloud application

As already mentioned in the previous chapters, the Cloud application is written in **java** and is able to communicate with **MQTT** protocol using the **Paho** library.

This application subscribes the three topics temperature, humidity and presence. Every time a new message is published in one of this topic, the application stores the received measurement in the corresponding table of the database.

In the images below we show the log information printed by this application during execution.

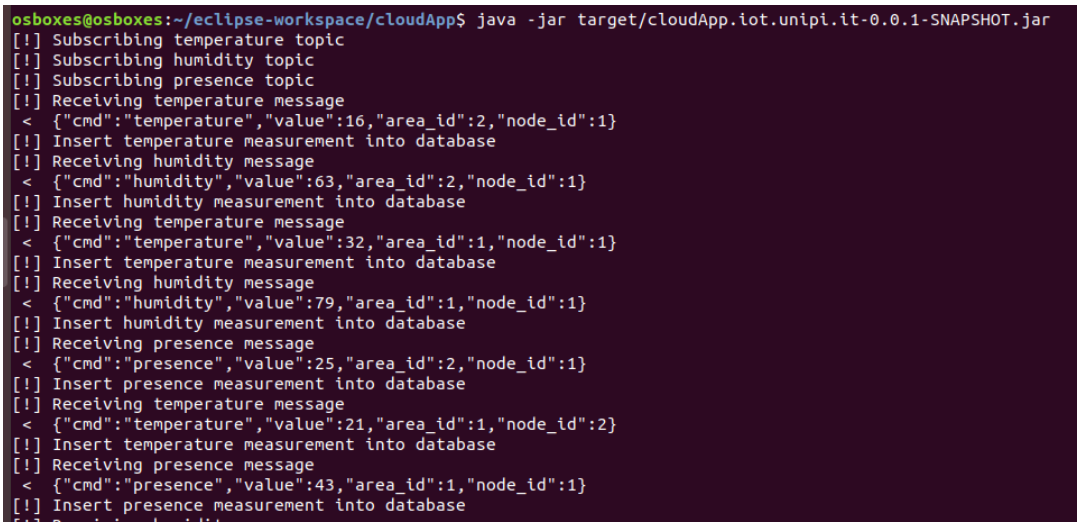

Figure 14: Cloud Application Log Info

5 Tests

For the **test phase** it was used a network consisting of the two applications running on a Ubuntu machine, **5 nodes** of which one is the **Border Router** and the others are **half MQTT nodes** and **half COAP nodes**. Two tests were done, the first is a simulation performed with the **Cooja** tool integrated in Contiki-NG project, the second was performed by using **real boards**.

5.1 Cooja Test

Below is the list of steps to start the simulation

- go to Remote control application folder and run

```
- mvn clean install
- java -jar target/<executable_name>.jar
```

- go to Cloud application folder and run

```
- mvn clean install
- java -jar target/<executable_name>.jar"
```

- start docker
- start Contiki container
- run

```
cd tools/cooja
```

- to start the simulation tool, run

```
ant run
```

- deploy border router
- deploy actuators
- deploy sensors setting for each one the desidered configuration (area_id and node_id)
- start the simulation
- start another Contiki container

- go to project folder and run

```
cd rpl-border-router
```

- run

```
make TARGET=cooja connect-router-cooja
```

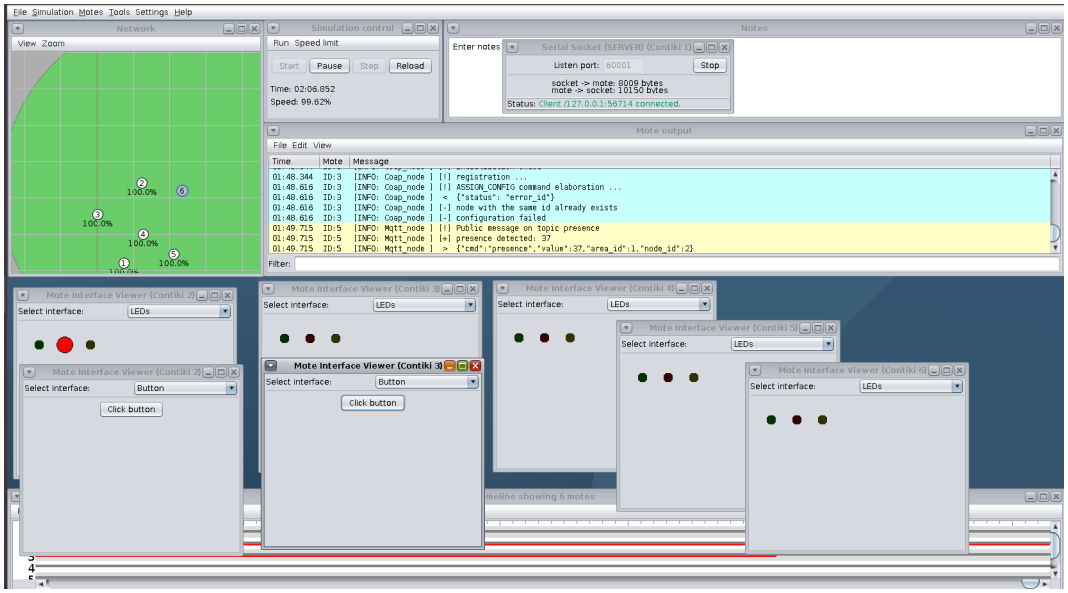

Figure 15: Simulation test

5.2 Test With Real nodes

For the real test, the same network as in the simulation was replicated using real boards as nodes. Specifically, the nodes were configured as follows:

- Border Router: nRF52840 dongle
- COAP node 1: nRF52840 dongle
- COAP node 2: nRF52840 dongle
- MQTT node 1: nRF52840 dongle
- MQTT node 2: nRF52840 dongle

The Remote control application, the Cloud application, database, and MQTT broker were put on the same Ubuntu machine as in the simulation.

To test the network it is first necessary to **flash the boards** with the source code. To load the code onto the boards, it is necessary to connect them to a USB

port and check the ttyACM name assigned to it by the operating system (run on the terminal "ls /dev/ttyA + tab").

Terminal **commands to flash the boards** (each command must be sent from terminal starting from the folder where the node source files are):

- Border Router:

```
make TARGET=nrf52840 BOARD=dongle
PORT=/dev/ttyACMx border-router.dfu-upload
```

- COAP node:

```
make TARGET=nrf52840 BOARD=dongle coap-node.dfu-upload
PORT=/dev/ttyACMx
```

- MQTT node:

```
make TARGET=nrf52840 BOARD=dongle mqtt-node.dfu-upload
PORT=/dev/ttyACMx
```

In the above commands replace "ttyACMx" with the port corresponding to the board (e.g., ttyACM0, ttyACM1 ...).

After board flashing it's possible **read the serial output** from the board with the following commands:

- `make login TARGET=nrf52840 BOARD=dongle PORT=/dev/ttyACMx`

Finally, in order to connect the Border Router to the applications, run the following command (run the command inside the rpl-border-router folder):

```
make TARGET=nrf52840 BOARD=dongle
PORT=/dev/ttyACMx connect-router
```

Figure 16: Terminals with nodes output

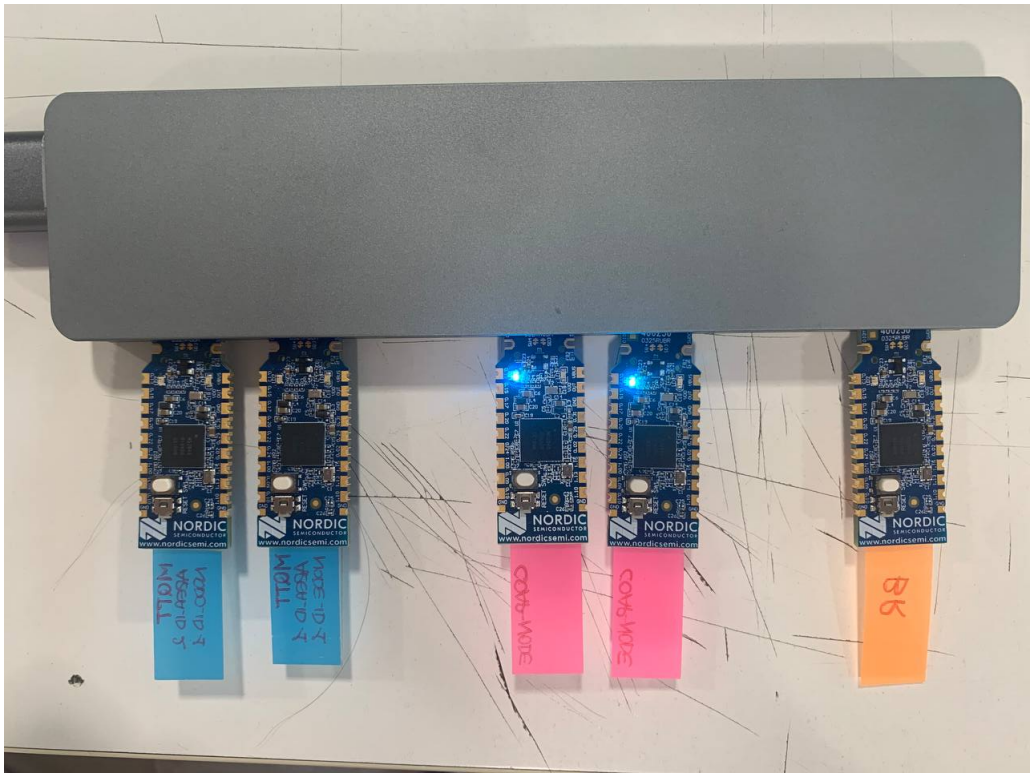

Figure 17: sensors during execution

6 Grafana

Grafana is a web tool that allows to create a **custom dashboard** to **monitor data** from various sources, in this case the source is mysql. The following images show the designed dashboard. Note that the measurement data is very unstable because it is randomly generated by the nodes.

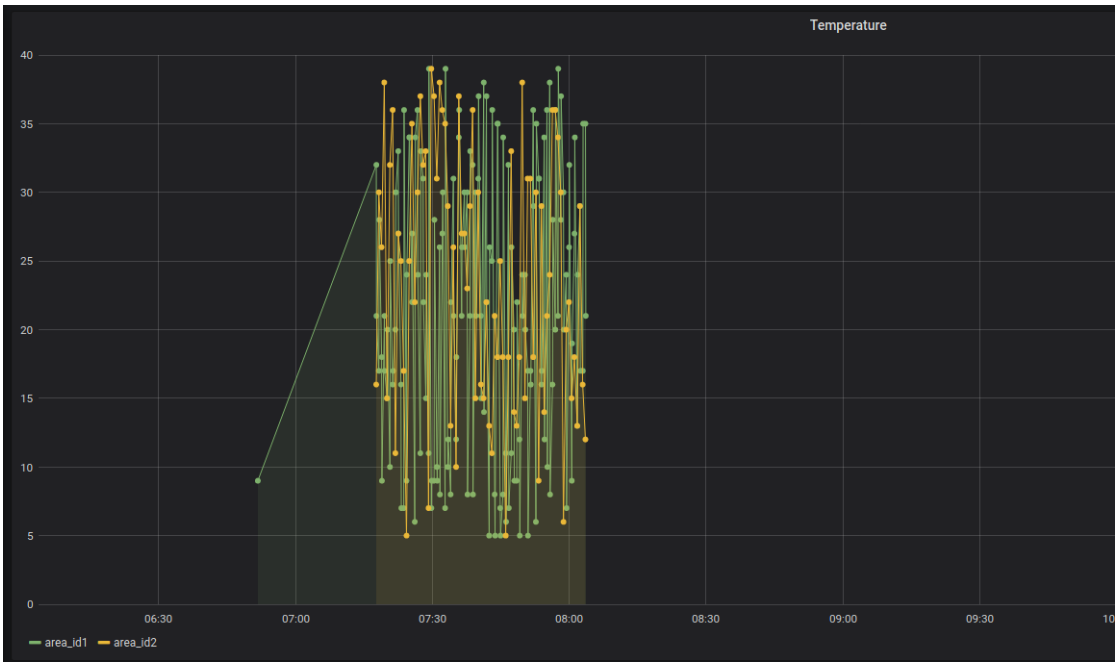

Figure 18: Temperature graph

Figure 19: Humidity graph

Figure 20: Presence graph

Figure 21: Presence graph