DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Laurea Magistrale in Artificial intelligence and Data engineering

ClimateKeeper

Gabriele Marcuccetti

ANNO ACCADEMICO 2022/2023

1 Introduction

Temperature and humidity are two of the most influential physical parameters when it comes to our daily lives and a range of industrial and environmental processes. The precise monitoring of these parameters is essential to ensure safety, efficiency, and well-being in both residential and workplace environments. The project is centered on implementing a sensor network dedicated to measuring temperature and humidity. These sensors transmit real-time data to a central application. The protocols used for sharing the measured physical quantities and communicating commands and updates are CoAP and MQTT. The central application, referred to as the "Collector," plays a crucial role in managing the collected data. It is responsible for continuously verifying that the sensor readings conform to the configured threshold parameters. In cases where the detected data goes beyond the allowed limits, the Collector takes action by activating CoAP actuators. These actuators, in turn, work to bring the temperature and humidity back within the desired range. The measurements are collected in a MySQL database, which is connected to a Grafana dashboard, allowing the administrator to continuously monitor the situation. Furthermore, the administrator has the ability to temporarily issue commands to the actuators.

2 System scheme and structure

As we can see in the following pictures, the system comprise more components. It's a network of IoT devices, including MQTT sensors collecting data from the physical system/environment and actuators and Coap actuators.The network is then deployed using real sensors (nrf52840 dongle). In the network there is a border router in order to provide external access. The Cloud Application collects data from MQTT sensors, store them in a MySQL database. The Remote control Application reads information about actuators and sensors from the database and implements a control logic.

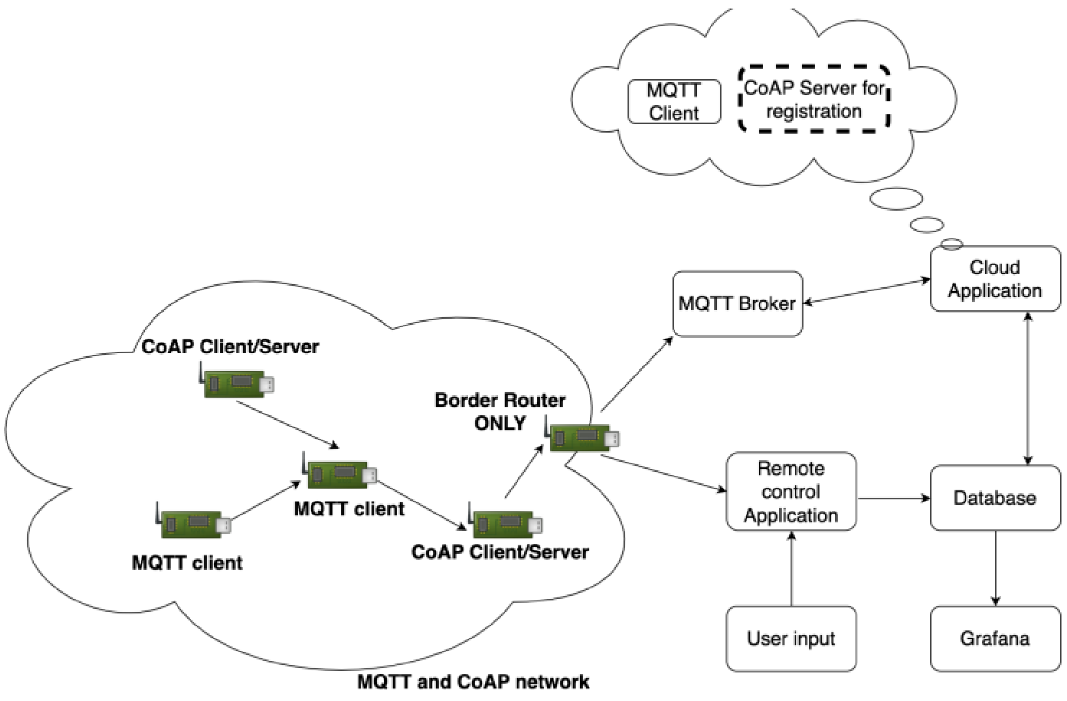


Figura 1: System architecture

2.1 Mqtt nodes

The MQTT node is designed to interact with the MQTT broker, which acts as an intermediary in data transmission. Once connected to the broker, the MQTT node can publish data on the "Measurements" topic and subscribe to the "Actuator" topic to receive commands and updates.

Specifically, the MQTT node can receive commands from the central system through the subscribed topics, allowing control of the temperature and humidity actuators, as well as enabling manual mode for users to manually adjust temperature parameters. Additionally, the MQTT node is responsible for periodically publishing temperature and humidity data on the appropriate topics, enabling the central system to continuously monitor the sensor's status. The publishing interval is set to 15 seconds, and the published message will have the following format:

```
{
    "SensorID": 145,
    "Temperature": 26,
    "Humidity": 45
}
```

Regarding messages received on the topics of interest, they will be in the following format, for example, if the command sent on the "Actuator" topic is to turn off the humidity actuator:

```
{
    "Sensor": humidity,
    "Payload": OFF
}
```

Furthermore, the MQTT node plays a crucial role in visually updating the surrounding environment through the use of LEDs. This provides immediate visual feedback to both human operators and the automatic control system. If the temperature or humidity values deviate significantly from the desired ones, for example, if the temperature deviates by at least 10 degrees from the ideal, the LED will turn red. Conversely, a lesser but still noticeable deviation will make the LED turn yellow. If the values are acceptable, the LED will be green. Finally, pressing the sensor's button will publish a measurement even if the 15-second interval has not yet elapsed. The measurement values are initially randomly generated, so temperature and humidity will be initialized with a random value. After that, if the actuator is activated due to detecting that the temperature is too high, the temperature will be decreased by one unit at each interval; conversely, if the temperature is too low, it will be increased. As soon as an acceptable value is reached, the actuator will be turned off. The same applies to humidity, but its change is simulated as an increase or decrease (depending on whether the value that triggered the actuator is too high or too low) by a random value between 1 and 5. Once the values fall within acceptable ranges, the actuators will be turned off. During the execution, after 3 periods where the values have returned to acceptable ranges and the actuators are off, a gradual change in measurements is simulated. This change is also quantified by randomly extracting a certain offset delta. The final change after 3 periods is done to attempt to make the simulation more realistic; otherwise, once the threshold values are reached, the sensors would have continued to measure the same values indefinitely without any deviation.

2.2 Coap nodes

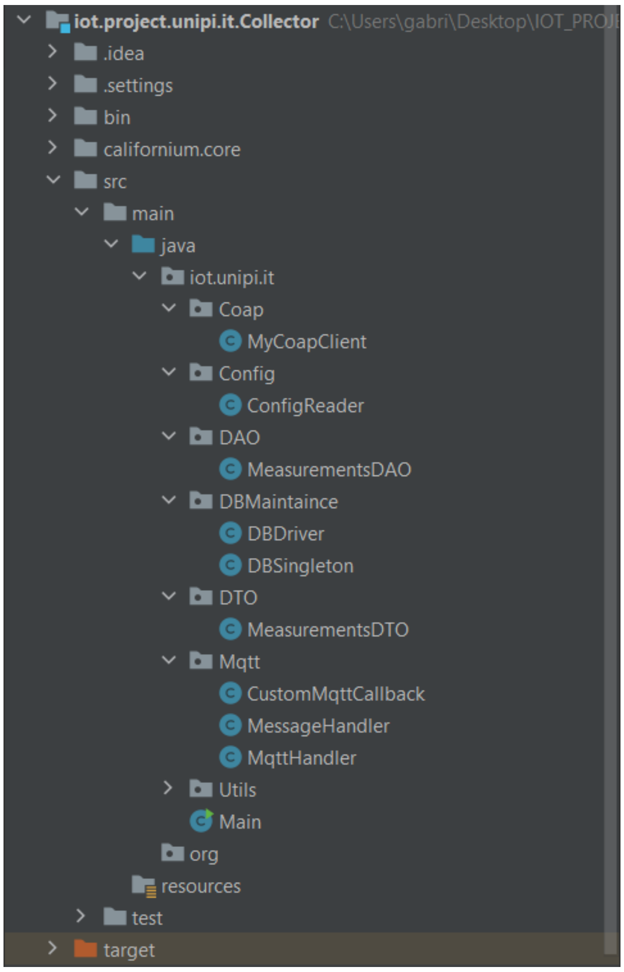
The CoAP node consists of 2 resources: res actuator temperature and res actuator humidity. Initially, the device has a red LED turned on. Subsequently, the LED will be green when the actuator is off or in manual mode, and red otherwise (when the actuator is active). Both resources include both POST and GET handlers.

In the POST handler, it checks if the incoming message indicates activating or deactivating the actuator. If activated, the LED turns red, and the actuator state is tracked by setting the on variable to 1. If the actuator is turned off, this action is tracked by resetting the on variable to 0, and the LED color changes to green.

The GET handler sends the actuator's status, indicating whether it is turned on or off.

2.3 Collector

The Remote Application and Cloud Application parts are written in Java. They primarily include Coap management, Database management, and Mqtt management. Initially, configuration parameters are read from a dedicated file, which contains instructions for connecting to the database, other information about URLs to use, or topics to subscribe to, for example. Next, database connections are initialized, and two instances of the MyCoapClient class are created, which contain functions for handling POST and GET requests from Coap nodes, one for the temperature resource and the other for the humidity resource. Then, the Mqtt event handler is created. It contains methods

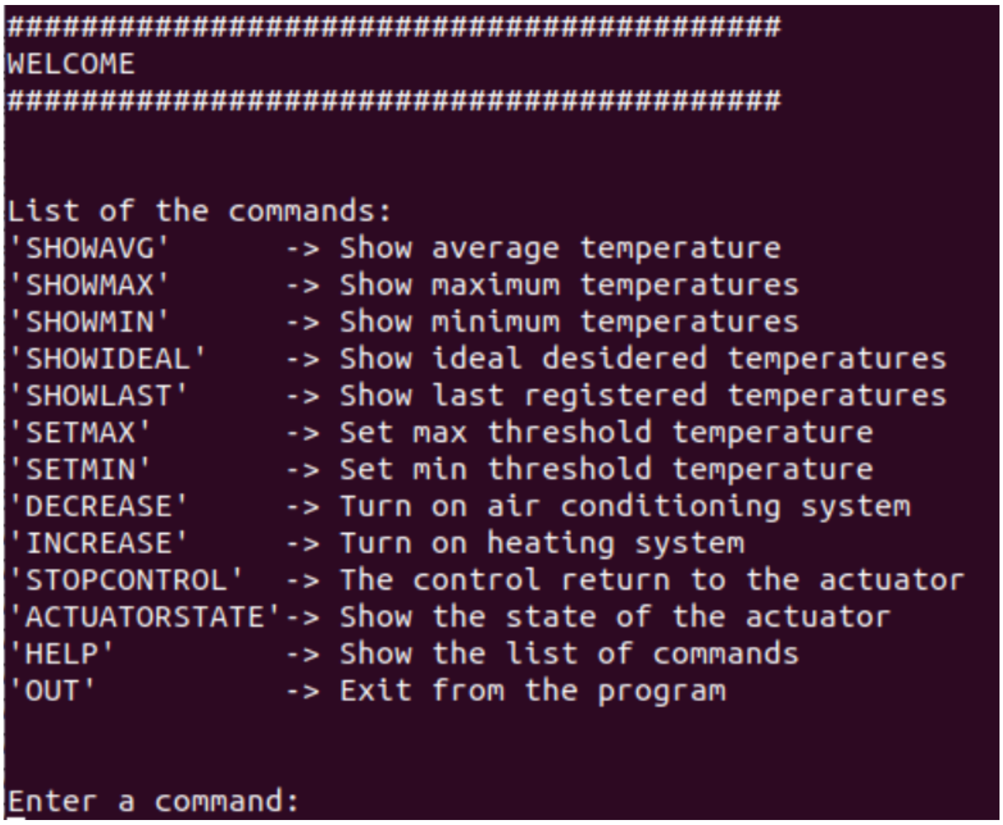


```
iot.project.unipi.it.Collector  C:\Users\gabri\Desktop\IOT_PROJ
  .idea
  .settings
  bin
  californium.core
  src
    main
      java
        iot.unipi.it
          Coap
            MyCoapClient
          Config
            ConfigReader
          DAO
            MeasurementsDAO
          DBMaintaince
            DBDriver
            DBSingleton
          DTO
            MeasurementsDTO
          Mqtt
            CustomMqttCallback
            MessageHandler
            MqttHandler
          Utils
          Main
        org
      resources
    test
  target
```

Figura 2: Project structure

for subscribing to the "Measurements" topic and publishing messages on the "Actuator" topic. Additionally, there is a handler for publications on the topics to which the instance subscribes. When it detects that a new message has been published on "Measurements," it processes the message, extracts the information, inserts the measurement into the database with the current timestamp, and checks if the data falls within the desired range. If this happens, a message is published on "Actuator" with the sensor's name followed by "OFF," indicating that it is not necessary to activate the actuator for that specific physical quantity. If the values do not fall within the expected range, the sensor's name followed by "ON" is published, indicating that action will be required using the actuator to bring the quantities back within the desired range.

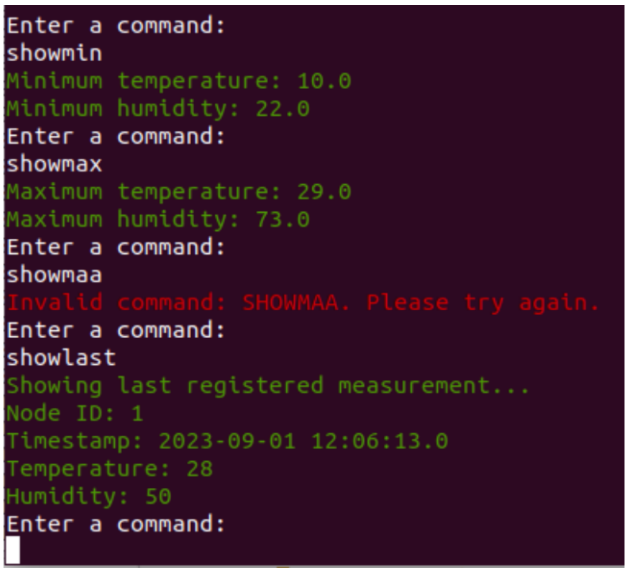
Then, the main menu is opened, which appears as follows:



```
###########################################
WELCOME
###########################################



List of the commands:
'SHOWAVG'      -> Show average temperature
'SHOWMAX'      -> Show maximum temperatures
'SHOWMIN'      -> Show minimum temperatures
'SHOWIDEAL'    -> Show ideal desidered temperatures
'SHOWLAST'     -> Show last registered temperatures
'SETMAX'       -> Set max threshold temperature
'SETMIN'       -> Set min threshold temperature
'DECREASE'     -> Turn on air conditioning system
'INCREASE'     -> Turn on heating system
'STOPCONTROL'  -> The control return to the actuator
'ACTUATORSTATE'-> Show the state of the actuator
'HELP'         -> Show the list of commands
'OUT'          -> Exit from the program


Enter a command:
```

Figura 3: Main menu

Among the various options, you can enter commands to view some statistics, for example:



```
Enter a command:
showmin
Minimum temperature: 10.0
Minimum humidity: 22.0
Enter a command:
showmax
Maximum temperature: 29.0
Maximum humidity: 73.0
Enter a command:
showmaa
Invalid command: SHOWMAA. Please try again.
Enter a command:
showlast
Showing last registered measurement...
Node ID: 1
Timestamp: 2023-09-01 12:06:13.0
Temperature: 28
Humidity: 50
Enter a command:
```

Figura 4: Possibile execution flow

In case of a typing error or if the command fails for some reason, the text will appear in red; otherwise, it will be displayed in green. There are also "increase" and "decrease" commands that allow users to control the actuators. With "increase," the temperature will increase, and with "decrease," it will decrease, until the "stopcontrol" command is entered. Finally, "actuatorstate," as the name suggests, allows you to view the status of the actuators. A GET request is sent to the Coap nodes, which check the status of the actuators and communicate it.

2.4 Data encoding

Regarding data encoding, the choice was made to use JSON: every time sensor data is transmitted to the Collector, it is done through a JSON object, just as it happens when the Collector needs to communicate with the actuator. This decision was made because sensors have limited resources, and JSON is much more flexible and involves less overhead compared to other encodings like XML, allowing for the transmission of more data together without being too verbose.

2.5 Grafana

Grafana is an open-source monitoring and data visualization platform that is often connected to MySQL databases. It allows you to create customizable dashboards and visualize data from MySQL databases, making it easier to monitor and analyze your database performance and data trends. In the project, Grafana's dashboard is connected to the 'Collector' database and displays 5 primary queries:

- This query displays the average temperature and humidity per hour

```sql
SELECT HOUR(Timestamp)  AS HourOfDay,
       AVG(Temperature) AS AvgTemperature,
       AVG(Humidity)    AS AvgHumidity
FROM Measurements
GROUP BY HOUR(Timestamp)
ORDER BY HourOfDay ASC;
```

- This query displays the maximum and minimum values for the measurements

```sql
SELECT $__timeGroupAlias(Timestamp,$__interval),
       MAX(Temperature) AS "Maxtemperature",
       MAX(Humidity)    AS "Maxhumidity",
       MIN(Temperature) AS "Mintemperature",
       MIN(Humidity)    AS "Minhumidity"
FROM Measurements
WHERE $__timeFilter(Timestamp)
GROUP BY 1
ORDER BY $__timeGroup(Timestamp,$__interval);
```

- This query displays the average values for the measurements

```sql
    SELECT $__timeGroupAlias(Timestamp,$__interval),
           AVG(Temperature) AS "temperature",
           AVG(Humidity)    AS "humidity"
    FROM Measurements
    WHERE $__timeFilter(Timestamp)
    GROUP BY 1
    ORDER BY $__timeGroup(Timestamp,$__interval);
```

- This query displays the percentage of samples in which the actuator was on. Remember that the actuator is on when the temperature is not 20, or when humidity is less than 30, or when humidity is greater than 60.

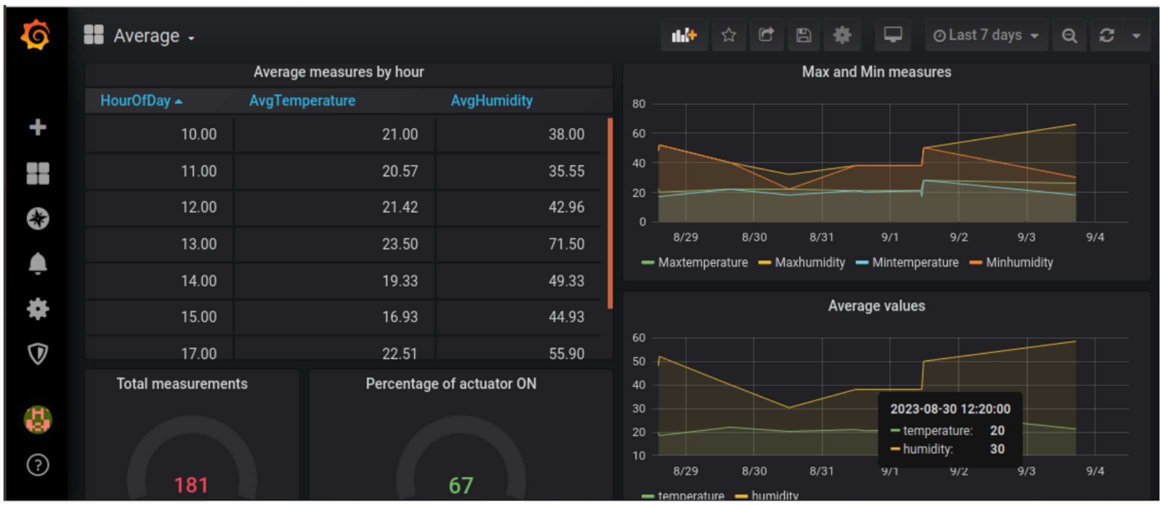
```sql
SELECT (COUNT(*)/
       (SELECT COUNT(*) FROM Measurements))*100
       AS percentage
FROM Measurements
WHERE temperature <> 20 OR humidity < 30
                        OR humidity > 60;
```

- Finally, one query displays the total number of measurements in the database, and another one shows data related to the last 50 measurements taken.

```sql
SELECT *
FROM Measurements
ORDER BY timestamp DESC
LIMIT 50;
```

```sql
SELECT COUNT(*)
FROM Measurements;
```

The overall view of the dashboard is as follows:



Average measures by hour

HourOfDay	AvgTemperature	AvgHumidity
10.00	21.00	38.00
11.00	20.57	35.55
12.00	21.42	42.96
13.00	23.50	71.50
14.00	19.33	49.33
15.00	16.93	44.93
17.00	22.51	55.90

Total measurements

181

Percentage of actuator ON

67

2023-08-30 12:20:00
temperature: 20
humidity: 30

Figura 5: Grafana dashboard