INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Middleware For IoT
Report

Cédric Chanfreau
Samia Boukouiss

5A ISS
2024 - 2025

TP 1 & 2 : Middleware For the IoT

Introduction :

The goal of this laboratory is to explore the features and applications of the MQTT protocol in the context of IoT. First, we will provide a concise overview of MQTT's main characteristics. Next, we will install the necessary software on our laptops to enable the use of MQTT. Finally, we will develop a simple application using an IoT device (ESP8266) that communicates with a server on our laptop via the MQTT protocol.

1. MQTT :

- What is the typical architecture of an IoT system based on the MQTT protocol?

The standard architecture of an IoT system utilizing the MQTT protocol comprises multiple devices (IoT nodes) that interact by publishing and subscribing to topics via an MQTT broker. Devices exchange information by sending messages to specific topics, which other devices subscribe to in order to receive the transmitted data. The MQTT broker serves as a central mediator, enabling seamless communication between devices.

- What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage, type of communication, etc ?

MQTT functions on the TCP/IP protocol, utilizing the Internet Protocol suite for communication. Its lightweight design ensures minimal bandwidth usage, making it well-suited for resource-constrained networks and devices. Leveraging a publish/subscribe model, MQTT enables efficient device communication with minimal overhead.

- What are the different versions of MQTT?

MQTT has three primary versions: MQTT v3.1, MQTT v3.1.1, and MQTT v5. Each successive version brings enhancements and new features, with version 5 being the most recent and comprehensive.

- What kind of security/authentication/encryption are used in MQTT?

MQTT incorporates several security features, such as username/password authentication and Transport Layer Security (TLS) encryption. These measures ensure secure communication between devices and the broker, protecting against unauthorized access and eavesdropping.

- Suppose you have devices that include one button, one light and luminosity sensor. You would like to create a smart system for you house with this behavior:

- **you would like to be able to switch on the light manually with the button**
- **the light is automatically switched on when the luminosity is under a certain value**

What different topics will be necessary to get this behavior and what will the connection be in terms of publishing or subscribing?

To implement this behavior with MQTT, the setup involves two topics:

- **Topic for manual control** (*"home/light/control"*): Used by devices to publish messages for manually controlling the light.
- **Topic for luminosity status** (*"home/light/luminosity"*): Used by the luminosity sensor to publish its readings.

Three subscriptions :

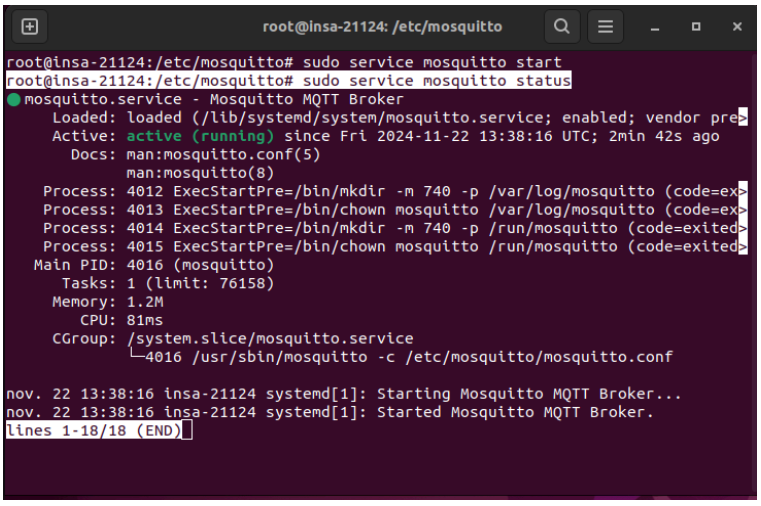
- **Button**: Subscribes to *"home/light/control"*
- **Light**: Subscribes to *"home/light/control"* and *"home/light/luminosity"*
- **Luminosity Sensor**: Subscribes to *"home/light/luminosity"*

Three connections :

- The **button** publishes messages to *"home/light/control"* when pressed.
- The **luminosity sensor** publishes its readings to *"home/light/luminosity"*.
- The **light** listens to both topics, adapting its behavior based on the received messages, whether for manual control or automatic adjustments informed by luminosity values.

2. Install and Test the broker

To begin, we downloaded and installed the Mosquitto broker with the following command line: apt install mosquitto. This broker, maintained by the Eclipse, is an open-source implementation of the MQTT protocol, available for different operating systems
Then we started the Mosquitto broker by running the following command in the terminal.



Once the broker was running, we tested the communication using the publish and subscribe commands.
In the following example, we used mosquitto_pub to publish a message to the topic /insa/test and mosquitto_sub to subscribe to that topic and receive the corresponding messages.

These tests allowed us to validate that the communication between the client and the broker was working correctly.

3. Creation of an IoT device with the nodeMCU board that uses MQTT communication:

a) Give the main characteristics of nodeMCU board in term of communication, programming language, Inputs/outputs capabilities

Communication :

- **Wi-Fi**: Supports wireless communication, ideal for IoT applications.
- **UART**: Enables serial communication for debugging and device interfacing.

Programming Language :

- **C/C++**: Programmable using the Arduino IDE.

Inputs/Outputs :

- **GPIO Pins**: Provides digital input/output functionality.
- **Analog Input**: Supports a single analog input pin for sensor readings.
- **PWM Support**: Allows Pulse Width Modulation for applications like LED dimming.
- **I2C and SPI**: Facilitates communication with peripheral devices like sensors.

b) Arduino IDE Installation

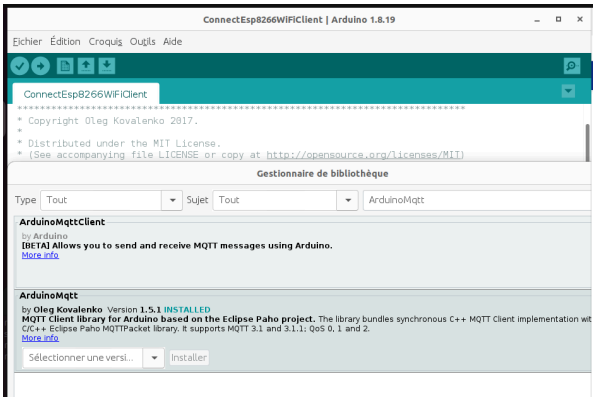
For the development with microcontrollers like the esp8266, we had to download Arduino IDE and set up the environment with the command: *sudo apt install arduino*

c) Add the board NodeMCU

In Arduino IDE, we added support for the NodeMCU board by including the link to the ESP8266 board manager in the preferences, then installed the board via the built-in manager.

d) Add the library ArduinoMqtt

Then we searched for and installed the ArduinoMqtt library developed by Oleg Kovalenko via the library manager



e) Application Example

For this step we used the PubSubClient library by installing it via the Arduino Library Manager. We then opened the mqtt_esp8266 example provided with the library. This code allows us to establish a connection with the MQTT broker, send messages and read received ones.

f) Add publish/subscribe behavior

Using the same mqtt_esp8266 example, we modified and tested the publishing and subscribing functions. The ESP8266 was configured to publish messages to “aa” topic and the laptop was subscribed to this topic to receive messages in return. These interactions were validated via the serial monitor and the mosquitto_sub commands.

```
  unsigned long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
    ++value;
    snprintf (msg, MSG_BUFFER_SIZE, "hello world #%ld", value);   // Msg
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("aa", msg);  // Topic
  }
}
```

```
.......
WiFi connected
IP address:
10.0.1.205
Attempting MQTT connection...connected
Publish message: hello world #1
Publish message: hello world #2
Publish message: hello world #3
Publish message: hello world #4
Publish message: hello world #5
Publish message: hello world #6
Publish message: hello world #7
Publish message: hello world #8
```

Défilement automatique Afficher l'horodatage

```
PS C:\Program Files\mosquitto> ./mosquitto_sub.exe -h 10.0.1.254 -p 1883 -t aa
hello world #12
hello world #13
hello world #14
hello world #15
hello world #16
hello world #17
hello world #18
```

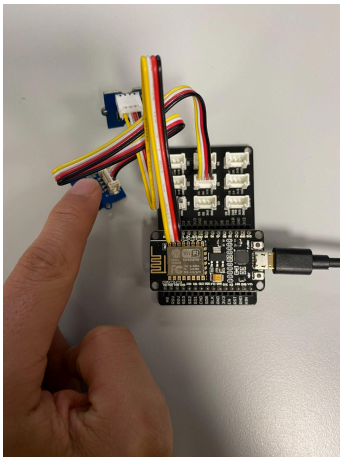
4. Creation of a simple application :

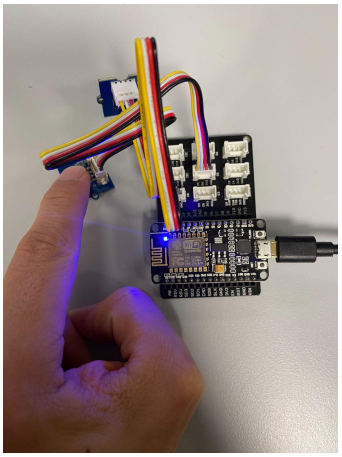
The objective of this application is to implement a light management system using MQTT, where the interaction between a button, a light (LED), and a luminosity value is handled through publish/subscribe exchanges. Here's how the system works:

a) **Button State Publishing**: The button's status (ON or OFF) is published to a specific MQTT topic (button/state). This allows other devices or applications subscribed to this topic to know the state of the button in real time.

b) **Light State Management**: When the button is pressed, the LED state changes (ON or OFF), its new state is also published with MQTT.

c) **Luminosity Feedback**: A luminosity sensor measures the light intensity. This value is published to light/state MQTT topic, enabling subscribers to get the LED's brightness.





```
unsigned long now = millis();
if (now - lastMsg > 500) {
  lastMsg = now;
  int lumState = analogRead(lum);
  int CurrentButtonState = digitalRead(button);
  int LastButtonState = 0;
  int ButtonPressed = 0;

  // Detection of LED state change
  if (CurrentButtonState && !LastButtonState ) {
      ButtonPressed = !ButtonPressed;

    snprintf (msg1, MSG_BUFFER_SIZE, "Button State: #%ld", ButtonPressed);   // Msg
    snprintf (msg2, MSG_BUFFER_SIZE, "Light State: #%d", lumState);      // Msg
    Serial.print("Publish message: ");
    Serial.println(msg1);
    client.publish("button/state", msg1);  // Topic
  }
  LastButtonState = CurrentButtonState;

  // If the button is pressed, the state of the LED change
  if (ButtonPressed) {
    ledState = !ledState;
    digitalWrite(BUILTIN_LED, ledState);
    delay(200);
    ButtonPressed = 0;

    snprintf (msg1, MSG_BUFFER_SIZE, "Button State: #%ld", ButtonPressed);   // Msg
    snprintf (msg2, MSG_BUFFER_SIZE, "Light State: #%d", lumState);      // Msg
    Serial.print("Publish message: ");
    Serial.println(msg1);
    Serial.print("Publish message: ");
    Serial.println(msg2);
    client.publish("button/state", msg1);  // Topic
    client.publish("light/state", msg2);  // Topic
  }
```

On the serial monitor we can have a view on what is sent to the broker. Moreover, once the subscription is done, we can have the different values for the sensors in real time.

```
Publish message: Button State: #1
Publish message: Button State: #0
Publish message: Light State: #583
Publish message: Button State: #1
Publish message: Button State: #0
Publish message: Light State: #62
```

```
PS C:\Program Files\mosquitto> ./mosquitto_sub.exe -h 10.0.1.254 -p 1883 -t button/state
Button State: #1
Button State: #0
PS C:\Program Files\mosquitto> ./mosquitto_sub.exe -h 10.0.1.254 -p 1883 -t light/state
Light State: #583
Light State: #62
```

5. Creation of a complex application

Another group subscribed to our MQTT topic from their own code. When a press on our button was detected and published on the topic, their system reacted by automatically sending us the value measured by their light sensor. This interaction illustrates the bidirectional communication and data synchronization between two IoT devices via MQTT.

```
PS C:\Program Files\mosquitto> ./mosquitto_pub -h 10.0.1.254 -p 1883 -t inTopic -m toto
```

```
Message arrived [inTopic] toto
```

Conclusion :

During this lab, we gained practical experience in deploying oneM2M nodes, both Infrastructure Nodes (IN) and Middle Nodes (MN), to build a robust IoT architecture. Additionally, the deployment of MQTT nodes enhanced our understanding by enabling seamless communication between devices. A significant takeaway was learning how to interconnect heterogeneous devices at the application level. This hands-on experience not only strengthened our knowledge of the oneM2M and MQTT protocols but also provided valuable insights into orchestrating diverse devices within the IoT ecosystem using Node-RED as a versatile tool.

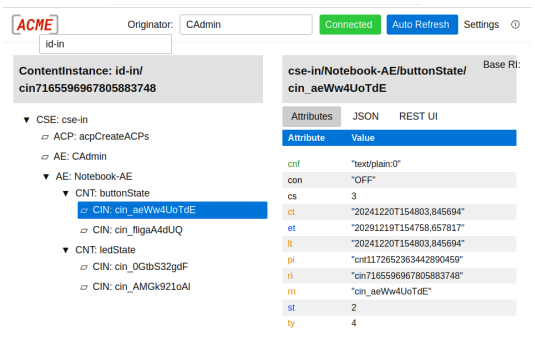
TP 3 : Middleware for IoT Based on oneM2M standard

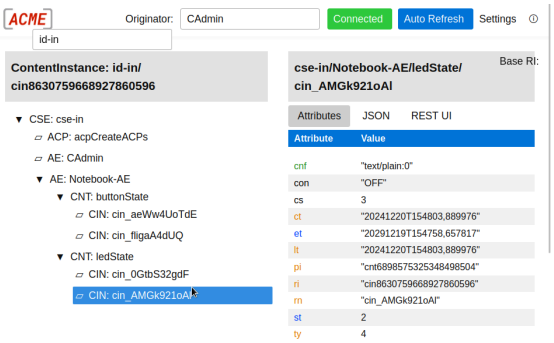
Introduction :

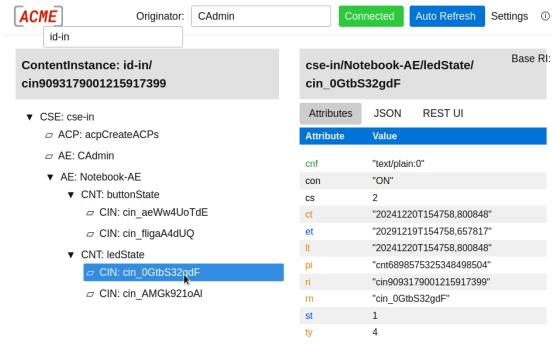
This report examines the practical application of middleware for the Internet of Things (IoT), with a particular emphasis on the oneM2M standard. Through a laboratory session utilizing the ACME stack, we provide a hands-on exploration of essential concepts and functionalities within the oneM2M ecosystem.

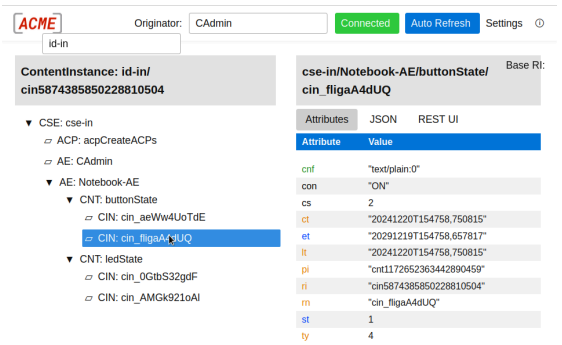
Simulated device :

This script creates an Application Entity (AE) to represent the device, along with containers to store the states of a button and an LED. The script establishes a logical link between these two components: the button's state ("ON" or "OFF") directly determines the LED's state. If the Button state is 'ON,' the Light state is set to 'OFF,' and vice versa. At regular intervals, the states are simulated, published in their respective containers, and can be retrieved for visualization.









Conclusion :

This simulated device script demonstrates the seamless integration of an ESP8266-based IoT device with the ACME oneM2M stack. By accurately emulating the device's behavior, such as creating, updating, and retrieving data within the oneM2M architecture, it highlights the interoperability of various components within a standardized IoT framework. Additionally, ACME's intuitive web interface offers a user-friendly platform for resource visualization and management, simplifying configuration and monitoring tasks. Moreover, the stack's adherence to the oneM2M standard guarantees compatibility with other implementations, promoting a collaborative and scalable IoT ecosystem.

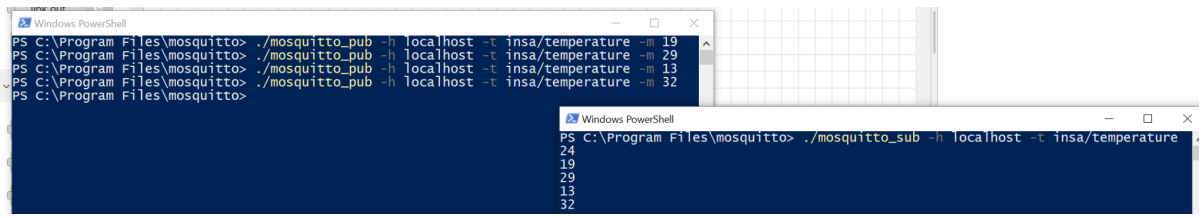
TP 4 : Fast application prototyping for IoT

Introduction :

In this last laboratory session, the primary goal is to integrate the knowledge acquired from TP1, TP2, and TP3 into a high-level application. The session emphasizes deploying a comprehensive architecture that incorporates both real and simulated devices. To facilitate the development process, we will utilize Node-RED, a versatile visual programming tool. Node-RED provides an intuitive interface for connecting devices and APIs, streamlining the creation of IoT applications. This approach not only accelerates development but also deepens understanding of complex IoT architectures. By leveraging Node-RED, we can efficiently design and deploy applications, combining practical implementation with the challenges of interfacing diverse devices and protocols.

1. Deploy the architecture

Firstly, we simulated the publication of data on the MQTT broker with the shell command:



2. Installation and access Node-RED

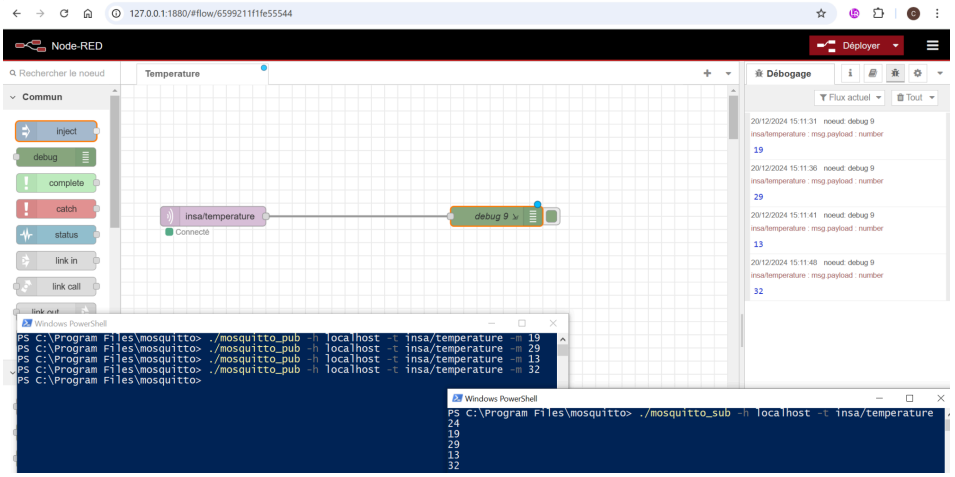
We had to do the environment's configuration by installing node.js, installing Node-RED (***npm install -g --unsafe-perm node-red***) and the integration of oneM2M nodes in Node-RED from https://gitlab.irit.fr/sepia-pub/lightom2m.

After running the command ***node-red*** in the terminal to lunch Node-RED, we can access it through the web browser at 127.0.0.1:1880.

3. Applications :

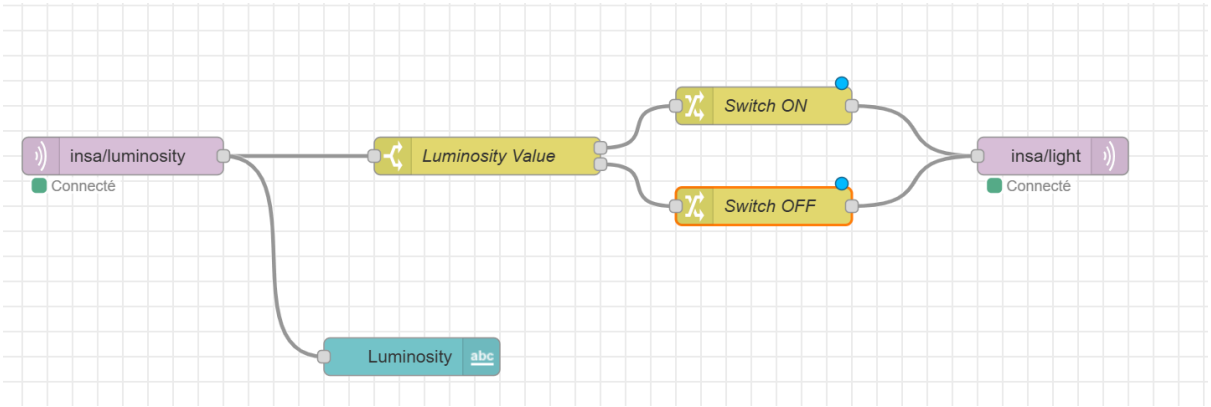
a. Check MQTT connectivity

We added a MQTT receiver block on Node-Red which was subscribed to the insa/luminosity topic. On the debugger we received the values correctly.



b. Sensors and activators

This Node-RED flow simulates a scenario where the light sensor sends the value received by the sensor. A Switch node evaluates whether the luminosity exceeds or falls below a threshold of 50, creating two distinct paths: LED ON or LED OFF. This setup demonstrates the interaction between a luminosity sensor and a switch based on predefined conditions.



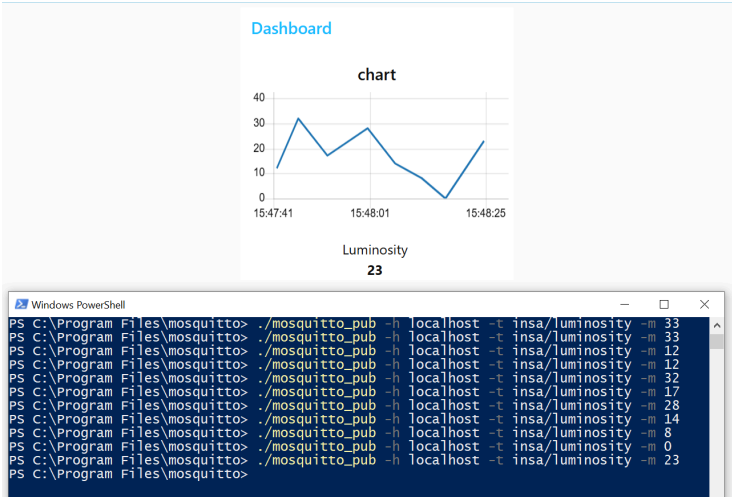
If the light intensity is greater than 50 ⇒ LED ON
If the light intensity is less than or equal to 50 ⇒ LED OFF

```
PS C:\Program Files\mosquitto> ./mosquitto_pub -h localhost -t insa/luminosity -m 23
PS C:\Program Files\mosquitto> ./mosquitto_pub -h localhost -t insa/luminosity -m 56
```

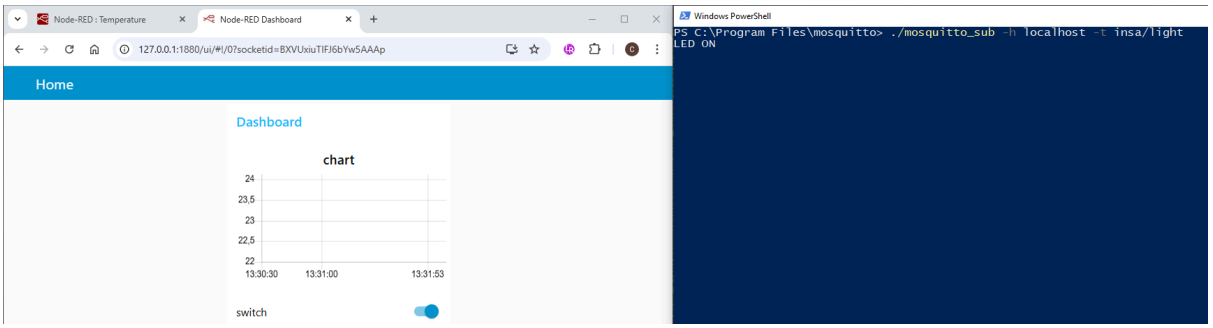
```
PS C:\Program Files\mosquitto> ./mosquitto_sub -h localhost -t insa/light
LED ON
LED OFF
```

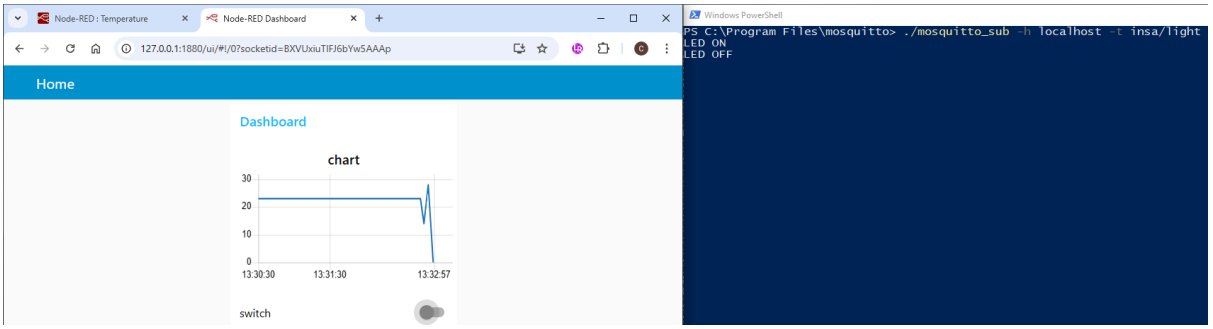
c. Dashboard

For the Dashboard part, we used the node-red-dashboard module to create a graphical user interface to visualize sensor data and interact with connected devices. The data collected by the sensors is displayed in real time in the form of graphs.



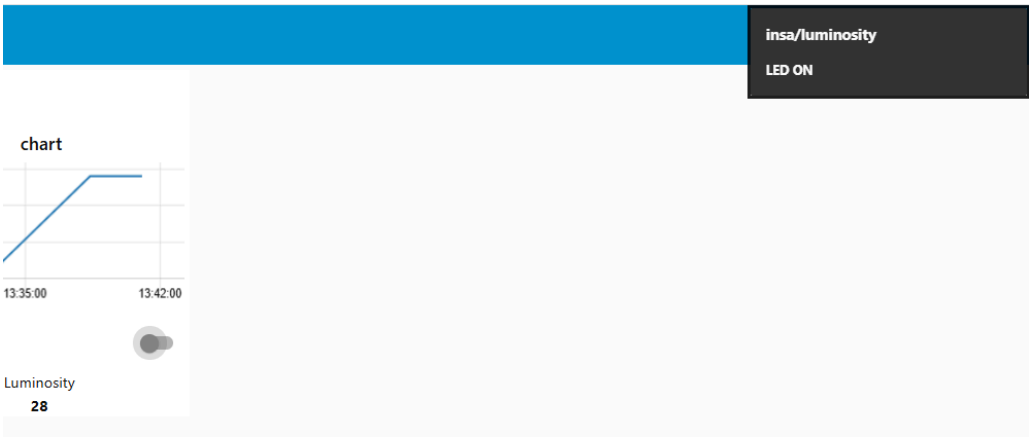
Additionally, buttons have been added to allow the control of actuators (e.g. turning LEDs on or off). This interface provides a clear visualization of information and simplifies the interaction with IoT devices.

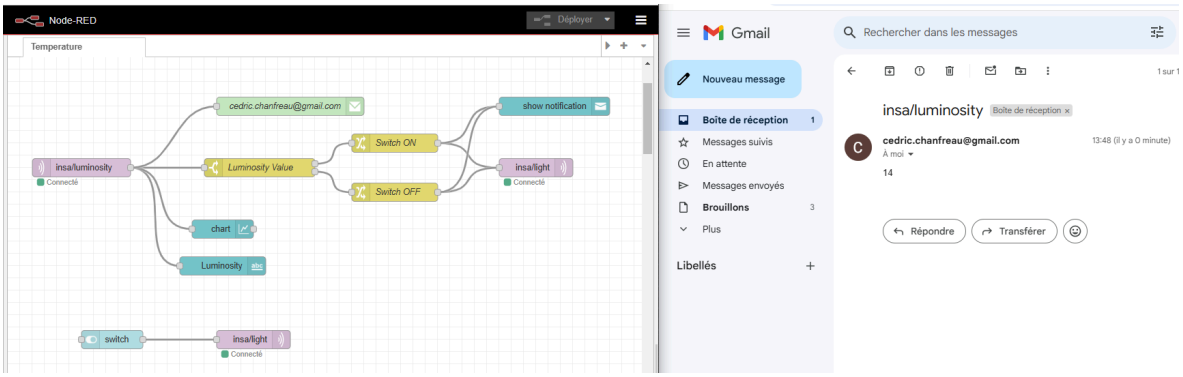




d. Email and Notification

The email and notification sending functionality was implemented using the node-red-node-email node. This part allows to configure automatic alerts that are sent when the switch state, and luminosity change for a sensor value. This mechanism ensures proactive monitoring of connected devices, allowing to react quickly to anomalies or critical events.





4. Benefits and drawbacks :

Developing applications with Node-RED offers several advantages:

- User-Friendly Interface: Its visual programming approach simplifies application creation, making it accessible to users with diverse technical skills.
- Integration Capabilities: Node-RED excels in connecting various devices, protocols, and APIs. With a broad selection of nodes for databases, IoT devices, web services, it's ideal for applications requiring diverse integrations.
- Fast Development: The availability of pre-built nodes for common functionalities accelerates the development process, reducing the effort needed for building complex applications.

However, Node-RED also has limitations:

- Scalability: While well-suited for small to medium-sized projects, managing flows can become challenging as application complexity grows.
- Code Maintenance: Reviewing and verifying code is difficult since all application logic is embedded in a single-line JSON file, complicating version control and debugging.
- Performance: It may not be the best option for applications that require high performance or low latency.

Conclusion :

During this lab, we gained practical experience in setting up oneM2M IN and MN nodes to build a solid IoT architecture. Adding MQTT nodes enhanced our understanding by enabling smooth communication between devices. A major highlight was learning how to connect heterogeneous devices at the application level. This hands-on session not only expanded our knowledge of the oneM2M and MQTT protocols but also provided valuable skills in managing diverse devices within the IoT ecosystem, with Node-RED serving as an effective orchestration tool.