# Machine to Machine
# Labs 1 & 2 : Middleware For the IoT

Kenza Bouzergan - Lamiaa Housni

## Introduction :

The objective of this laboratory is to delve into the functionalities of the MQTT protocol within the context of IoT. Initially, we will conduct a brief overview of the key features of MQTT. Subsequently, we will proceed to install various software on our laptop to facilitate the manipulation of MQTT. Finally, we will engage in the development of a straightforward application involving an IoT device (ESP8266), employing the MQTT protocol for communication with a server residing on our laptop.

## 1. MQTT

**• What is the typical architecture of an IoT system based on the MQTT protocol?**
The typical architecture of an IoT system based on the MQTT protocol involves multiple devices (IoT nodes) that publish and subscribe to topics through an MQTT broker. The devices communicate with each other by publishing messages to specific topics, and other devices subscribe to these topics to receive the messages. The MQTT broker acts as a mediator, facilitating the communication between devices.

**• What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage, type of communication, etc ?**
MQTT operates over the TCP/IP protocol. This means that it uses the Internet Protocol suite for communication. In terms of bandwidth usage, MQTT is relatively lightweight, making it suitable for constrained networks and devices. It uses a publish/subscribe model, allowing for efficient communication between devices without excessive overhead.

**• What are the different versions of MQTT?**
There are three main versions of MQTT: MQTT v3.1, MQTT v3.1.1, and MQTT v5. Each version introduces improvements and additional features, with version 5 being the latest and most feature-rich.

**• What kind of security/authentication/encryption are used in MQTT?**
MQTT supports various security features, including username/password authentication and Transport Layer Security (TLS) encryption. These mechanisms ensure secure communication between devices and the broker, preventing 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 achieve this behavior using MQTT, you would need two topics:

- Topic for manual control (e.g., "house/light/control"): Devices can publish messages to this topic to manually control the light.
- Topic for luminosity status (e.g., "house/light/luminosity"): The luminosity sensor publishes its readings to this topic.

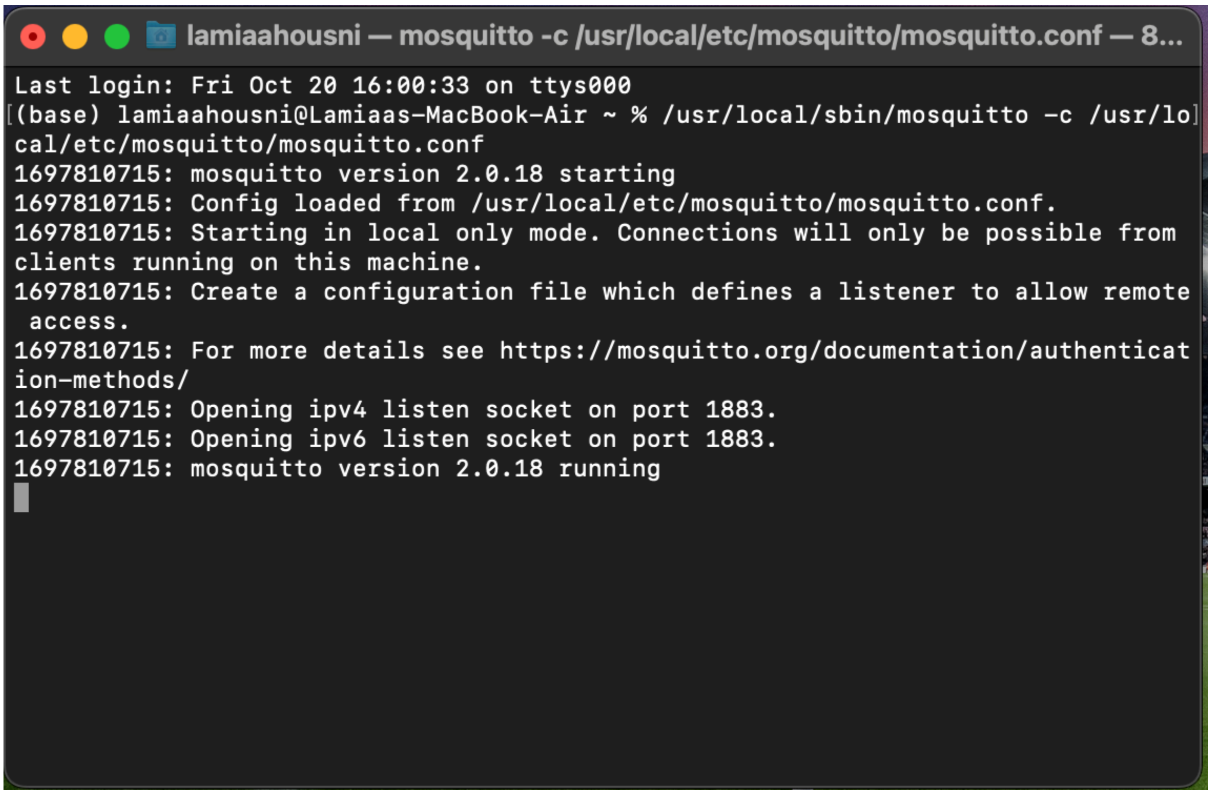
Subscriptions:

- The button subscribes to the "house/light/control" topic.
- The light subscribes to both "house/light/control" and "house/light/luminosity" topics.
- The luminosity sensor subscribes to the "house/light/luminosity" topic.

Connections:

- The button publishes messages to "house/light/control" when pressed.
- The luminosity sensor publishes luminosity readings to "house/light/luminosity."
- The light subscribes to both topics and acts accordingly based on the received messages (manual control or automatic control based on luminosity values).

## 2. Install and test the broker


lamiaahousni — mosquitto -c /usr/local/etc/mosquitto/mosquitto.conf — 8...

```
Last login: Fri Oct 20 16:00:33 on ttys000
(base) lamiaahousni@Lamiaas-MacBook-Air ~ % /usr/local/sbin/mosquitto -c /usr/lo
cal/etc/mosquitto/mosquitto.conf
1697810715: mosquitto version 2.0.18 starting
1697810715: Config loaded from /usr/local/etc/mosquitto/mosquitto.conf.
1697810715: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1697810715: Create a configuration file which defines a listener to allow remote
 access.
1697810715: For more details see https://mosquitto.org/documentation/authenticat
ion-methods/
1697810715: Opening ipv4 listen socket on port 1883.
1697810715: Opening ipv6 listen socket on port 1883.
1697810715: mosquitto version 2.0.18 running
```

Mosquitto running on local

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

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

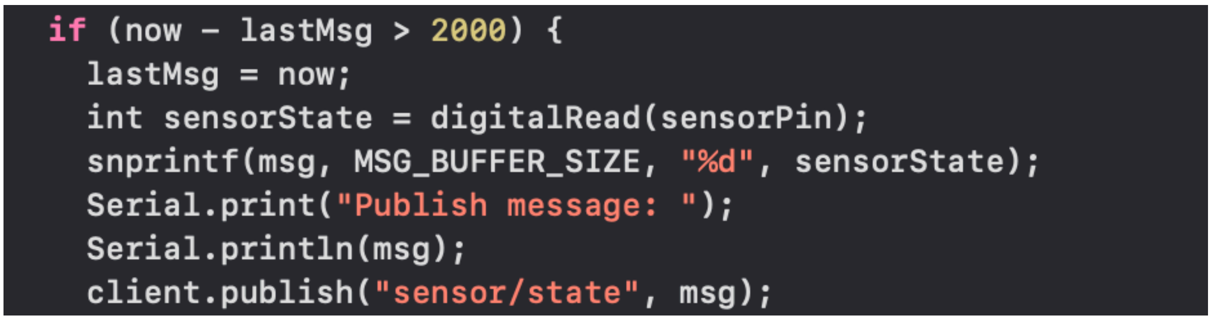
1. Communication:
   - Wi-Fi for wireless communication.
   - UART for serial communication.
2. Programming Language:
   - Lua or C/C++ with Arduino IDE.
3. Inputs/Outputs (I/O):
   - GPIO pins for digital operations.
   - Analog input and PWM support.
   - I2C and SPI communication.
4. Other Features:
   - Onboard flash memory.
   - USB interface for power and programming.
   - NodeMCU firmware pre-installed.

## 4. Creation of a simple application

In the current implementation, MQTT serves to facilitate communication between an ESP8266 microcontroller and an external MQTT broker. The overarching objective is to effectuate dynamic light management in response to state changes in a button. The codes of the MQTT publish and subscribe calls are below:

1. MQTT Publish (Button State)

At periodic intervals, the system undertakes the publication of the button's state to the designated MQTT topic, namely "sensor/state."

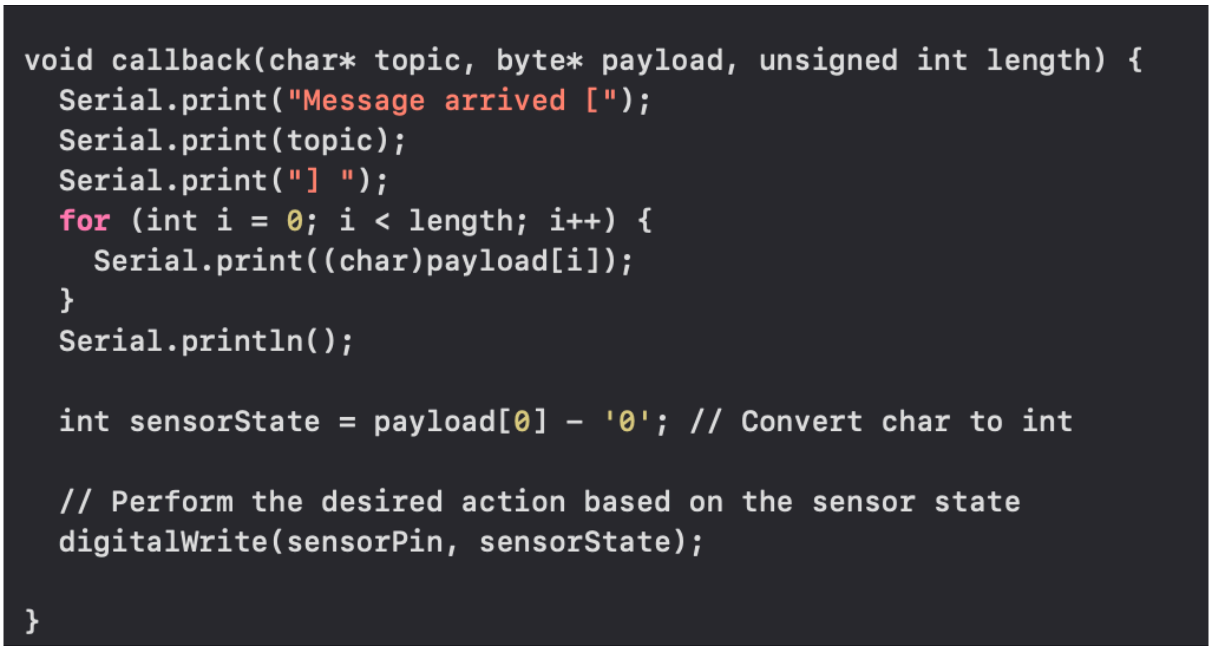


```
  if (now - lastMsg > 2000) {
    lastMsg = now;
    int sensorState = digitalRead(sensorPin);
    snprintf(msg, MSG_BUFFER_SIZE, "%d", sensorState);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("sensor/state", msg);
```

- client.publish("sensor/state", msg): Subsequently, the MQTT protocol is employed to broadcast this information to the designated topic, "sensor/state."

2. MQTT Subscribe (Light State)

At the same time, the system subscribes to the MQTT topic "sensor/state" to ascertain real-time updates. Upon the reception of a message, the callback function orchestrates the requisite adjustments for the illumination system:



```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  int sensorState = payload[0] - '0'; // Convert char to int

  // Perform the desired action based on the sensor state
  digitalWrite(sensorPin, sensorState);

}
```

In conclusion, the basic structure of our system allows two-way communication. The button tells the lights about its status using MQTT. This creates a flexible and adaptable system where the lights can change how they behave based on what the button is doing.

## Conclusion :

In conclusion, this lab provided a comprehensive exploration of the MQTT protocol's functionalities within the realm of the Internet of Things. The use of Mosquitto as a local MQTT broker facilitated testing, while the NodeMCU board's characteristics, such as Wi-Fi communication and versatile I/O capabilities, contributed to the development of a flexible and adaptable system. The implemented application showcased two-way communication, allowing a button to convey its status to lights through MQTT. Overall, the labs successfully combined theoretical knowledge with practical application, providing valuable insights into MQTT's role in building efficient and interactive IoT systems.

# Machine to Machine
# Lab 3 : Middleware for IoT Based on oneM2M standard

Kenza Bouzergan - Lamiaa Housni

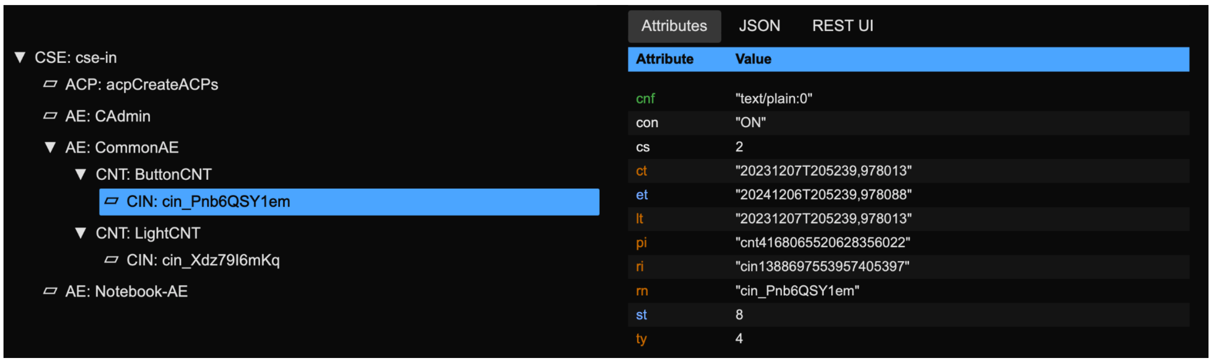
## Introduction :

In this report, we delve into the practical aspects of middleware for the Internet of Things (IoT) with a focus on the oneM2M standard. The laboratory session, centered around the ACME stack, presents a hands-on exploration of key concepts and functionalities within the oneM2M ecosystem.

## Simulated device :

The script initializes the simulated device by creating an Application Entity (AE) named 'CommonAE' to represent the device within the oneM2M architecture. Subsequently, containers for the Button and Light, namely 'ButtonCNT' and 'LightCNT,' are created within 'CommonAE.'

The initial state of the Button is simulated by creating a Content Instance within 'ButtonCNT' with the value 'ON,' emulating the button being in the 'ON' state. Following this, the retrieveLatestAndUpdate function is employed to fetch the latest state of the Button ('ButtonCNT') and update both the Button and Light containers based on the retrieved value. If the Button state is 'ON,' the Light state is set to 'OFF,' and vice versa.


```
▼ CSE: cse-in
    ▱ ACP: acpCreateACPs
    ▱ AE: CAdmin
    ▼ AE: CommonAE
        ▼ CNT: ButtonCNT
            ▱ CIN: cin_Pnb6QSY1em
        ▼ CNT: LightCNT
            ▱ CIN: cin_Xdz79I6mKq
    ▱ AE: Notebook-AE
```

| Attribute | Value |
|---|---|
| cnf | "text/plain:0" |
| con | "ON" |
| cs | 2 |
| ct | "20231207T205239,978013" |
| et | "20241206T205239,978088" |
| lt | "20231207T205239,978013" |
| pi | "cnt4168065520628356022" |
| ri | "cin1388697553957405397" |
| rn | "cin_Pnb6QSY1em" |
| st | 8 |
| ty | 4 |

## Conclusion :

This simulated device script effectively showcases the integration of an ESP8266-based IoT device with the ACME oneM2M stack. By successfully emulating the device's behavior, including creating, updating, and retrieving data within the oneM2M architecture, the script demonstrates the interoperability of different components within a standardized IoT framework.Besides, ACME's web interface provides a user-friendly platform for visualizing and managing resources, streamlining the configuration and monitoring processes. Furthermore, the stack's compliance with the oneM2M standard ensures compatibility with other implementations, fostering a collaborative and extensible IoT ecosystem.

# Machine to Machine
# Lab 4 :  Fast application prototyping for IoT

Kenza Bouzergan - Lamiaa Housni

## Introduction :

In this final IoT laboratory session, our primary objectives are to integrate the cumulative knowledge gained from TP1, TP2, and TP3 into a high-level application. The focus is on deploying a comprehensive architecture involving both real and simulated devices. To streamline the development process, we will leverage Node-RED, a powerful visual programming tool. Node-RED offers an efficient means of creating IoT applications by providing a visual interface for wiring together devices and APIs. This approach not only accelerates application development but also enhances the overall understanding of complex IoT architectures. Through Node-RED, we will have the opportunity to create and deploy applications faster, combining practicality with the intricacies of interfacing with diverse devices and protocols.

## 1) Installation :

First thing we had to do was the environment's configuration :
- Installing node.js
- Installing Node-RED ***npm install -g --unsafe-perm node-red***
- Integration of oneM2M nodes in Node-RED from https://gitlab.irit.fr/sepia-pub/lightom2m

## 2) Access Node-RED

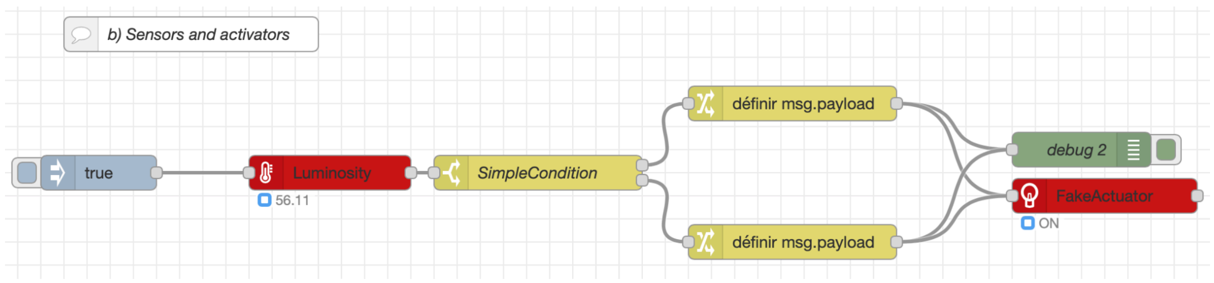
After running the command line ***node-red*** in the terminal to lunch Node-RED we can access it through the web browser at http://localhost:1880

## 3) Applications :

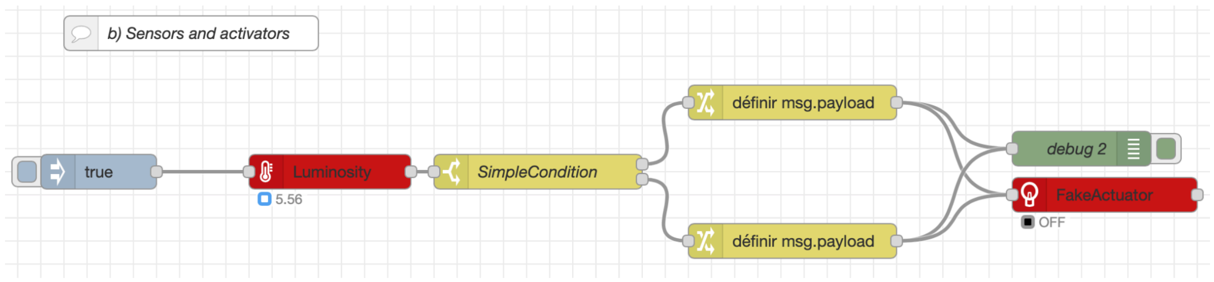
### a) Sensors and activators

This Node-RED flow simulates a scenario where a luminosity sensor generates random values between 0 and 100. The flow includes a Switch node that assesses whether the luminosity is above or below a threshold of 50. Depending on this condition, two paths are created: one for low luminosity (payload set to "0") and another for high luminosity (payload set to "1"). An actuator responds to these conditions, mimicking the behavior of real-world actuators. The Debug node provides visibility into the payload, aiding in monitoring and troubleshooting. Through an Inject node, a boolean payload ("true") initiates the simulation, illustrating the interaction between a luminosity sensor and corresponding actuator based on predefined conditions.

**Luminosity > 50 ⇒ Actuator ON**

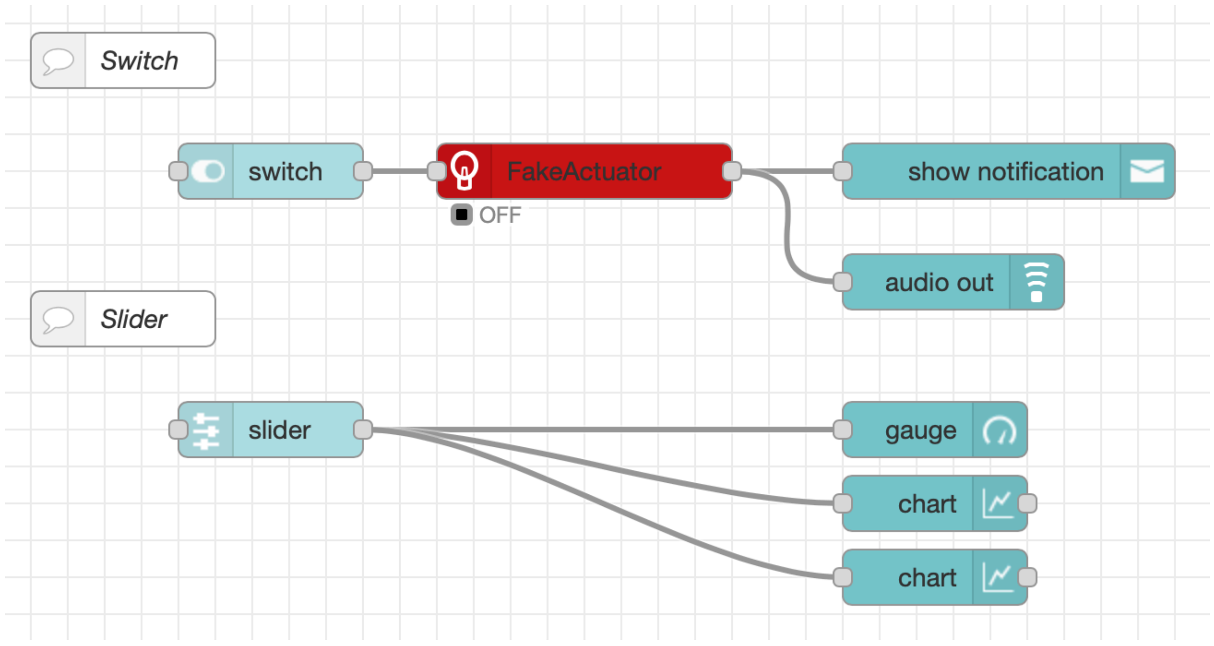


**Luminosity < 50 : ⇒ Actuator OFF**

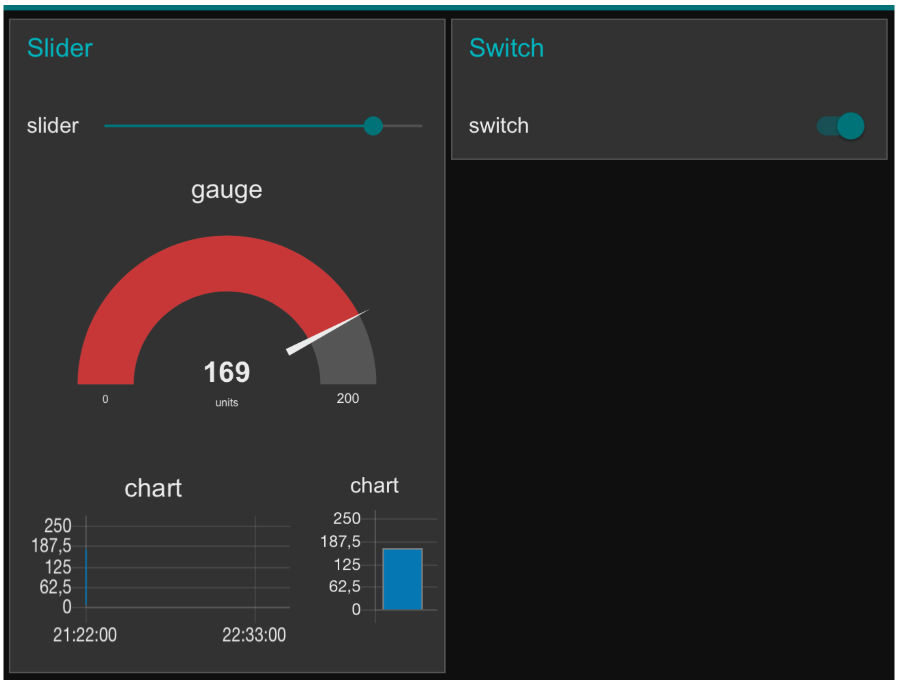


## b) Dashboard

This Node-RED flow sets up a dashboard with a UI Switch, UI Slider, UI Gauge, UI Charts, and UI Audio components. The Switch controls a Fake Actuator, triggering a UI Toast for visual feedback. The Slider influences the Gauge and charts. Two chart types visualize data trends. An Audio component produces sound based on events. Organized into "Switch" and "Slider" groups, this flow showcases Node-RED's capability to create dynamic IoT dashboards.

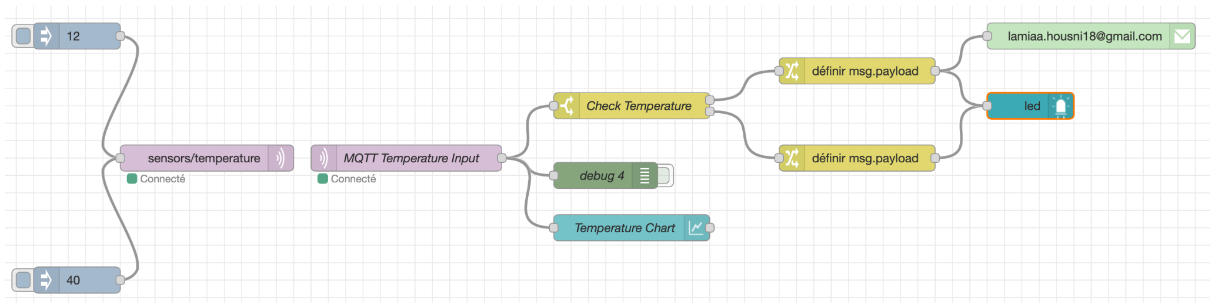


Here is the dashboard corresponding :



### c) Imagine

This Node-RED flow monitors temperature data from an MQTT source. The input is analyzed using a switch node to check if it exceeds a threshold (25 in this case). If the temperature is above the threshold, it triggers a warning LED, sends an email notification, and updates the chart on the dashboard. The temperature data is also visualized in a bar chart on the dashboard. Two inject nodes simulate temperature values of 12 and 40 for testing purposes. The flow demonstrates how Node-RED can be used for real-time monitoring, alerting, and email notifications based on specified conditions.



Note : We had to set up two-factor authentication (2FA) for the Gmail account because Google wouldn't allow sending emails. So, we added 2FA in the Google account settings and used the generated password in the email node to make it work.

## 4) Benefits and drawbacks :

Building an application with Node-RED has several benefits :

- Visual programming interface making it easier for users with varying technical backgrounds to create applications through a flow-based approach.
- Integration Capabilities : Node-RED is suitable for integrating different devices, protocols, and APIs. It supports a wide range of nodes for interacting with databases, IoT devices, web services, and more. This makes it suitable for building applications that require diverse integrations.
- Rapid Development: It allows for rapid development of applications by providing a library of pre-built nodes for various functionalities. This can significantly reduce the time and effort required to create complex applications.

Node-RED has some drawbacks as well :

- Scalability : While Node-RED is excellent for small to medium-sized projects, it might face scalability challenges for large and complex applications. The visual flow paradigm might become less manageable as the complexity increases.
- Performance: For certain performance-critical applications, Node-RED might not be the optimal choice.

- Code reviews are difficult. In fact, the code is hard to verify because all application code is embedded in a one-line JSON file.

## Conclusion :

During this lab, we acquired practical skills in deploying oneM2M nodes, both IN and MN, to establish a robust IoT architecture. The deployment of MQTT nodes further enriched our understanding, allowing seamless communication between devices. One of the key takeaways was the ability to interconnect heterogeneous devices at the application level. This hands-on experience not only deepened our comprehension of oneM2M and MQTT protocols but also equipped us with valuable insights into orchestrating diverse devices within the IoT ecosystem using Node-RED as a powerful tool.