

Report Middleware

Benchekroun Amine
Diani Badr

5 ISS
2024/2025

Contents

Lab 1 & 2 : ESP8266 & MQTT

1) Introduction.....	2
2) MQTT	2
3) Installation and Broker	4
4) Creating an IoT Device Using NodeMCU and MQTT	5
5) Creating a Simple Application	7
6) Conclusion	8

Lab 3 : oneM2M REST API

1) Introduction	9
2) Application	9
3) AE Resources	10
4) Container Resources for Each AE	11
5) Content Instances	12
6) Group Creation	13
7) Subscription Creation	14
8) Simulation	15
9) Conclusion	16

Lab 4 : Fast application prototyping for IoT

1) Introduction	17
2) Definition of Node-RED	17
3) Node-RED Installation	17
a. Installation of Node.js	17
b. Installation of NVM	18
c. Installation of Node-RED	19
d. Starting Node-RED	19
4) Applications	20
a. Simple Test: Displaying Sensor Values	20
b. Sensors and Actuators	21
c. Dashboard	22
5) Advantages and Disadvantages of Node-RED	23
a. Advantages	23
b. Disadvantages	23
c. Use Cases	23
6) Conclusion	24

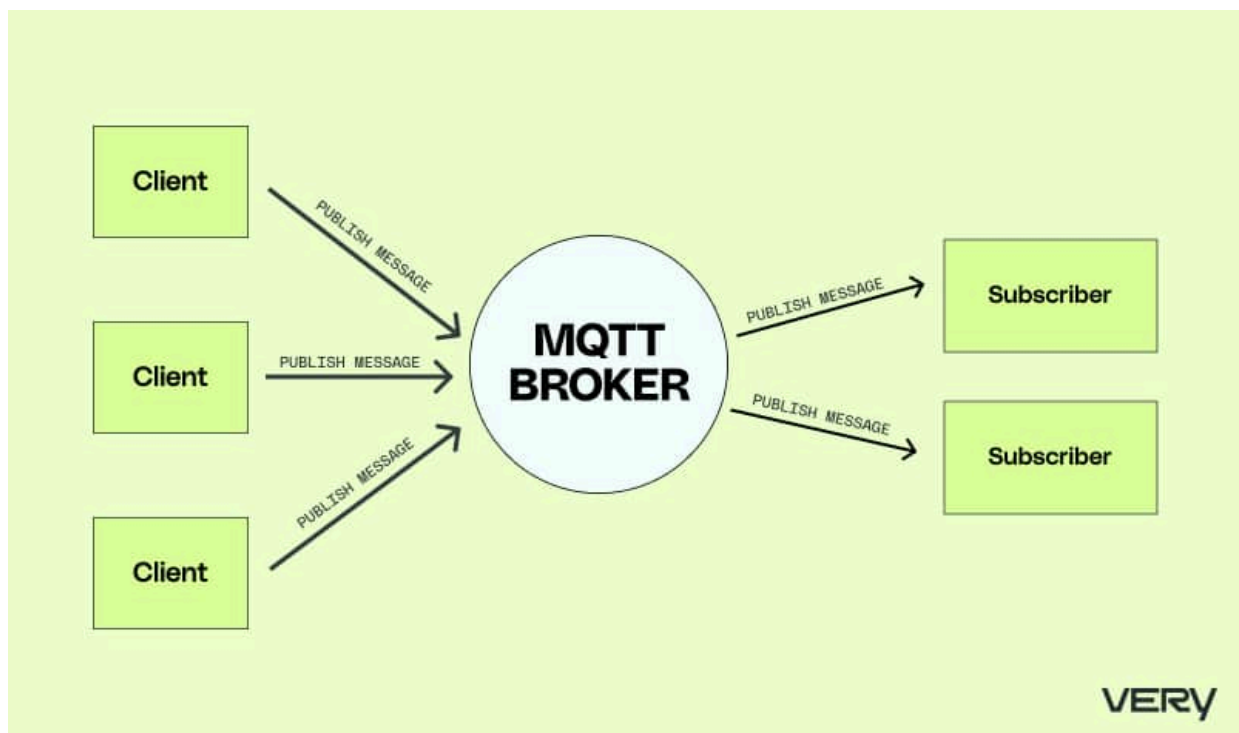
Lab 1 & 2 : ESP8266 & MQTT

1) Introduction

The objective of this lab is to explore the features and potential of the MQTT protocol for IoT applications. To achieve this, we will begin with an overview of MQTT's key characteristics. Next, we will install and configure various software tools on our laptop to work with MQTT. Finally, we will develop a basic application using an IoT device (ESP8266) to establish communication with a server on our laptop via the MQTT protocol.

2) MQTT

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



In an MQTT architecture, the system consists of two main components: clients and brokers. The broker acts as a central server that facilitates communication between clients. It receives messages from clients and forwards them to other clients as needed. Clients do not communicate directly with each other but instead rely on the broker to handle all message routing. Each client can function as a publisher, a subscriber, or both.

MQTT operates as an event-driven protocol, meaning it avoids continuous or periodic data transmission, thereby minimizing communication overhead. A client only sends messages when there is data to publish, and the broker forwards messages to subscribers only when new data becomes available.

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

MQTT uses TCP/IP for transport, enabling reliable, low-bandwidth communication. Its lightweight design, with minimal overhead and persistent connections, conserves bandwidth. The publish/subscribe model ensures messages are sent only to subscribed clients, and QoS levels offer control over reliability and bandwidth usage. With small packet sizes and efficient targeting via topics, MQTT is ideal for IoT and constrained networks.

- What are the different versions of MQTT?

MQTT has two main variants and several versions:

MQTT v3.1 (2010) : The first widely adopted version introduced QoS levels, clean sessions, retained messages, and Last Will and Testament (LWT) for efficient IoT communication.

MQTT v3.1.1 (2014) : Standardized by OASIS, it improved efficiency and interoperability with UTF-8 support, better error handling, and lighter CONNECT packets.

MQTT v5.0 (2019) : A modernized version with features like message properties, shared subscriptions, topic aliases, and message expiry, catering to complex IoT systems.

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

MQTT ensures security through authentication methods like username/password, token-based systems (e.g., OAuth2), or X.509 certificates for mutual authentication via TLS/SSL. Encryption is typically provided by TLS/SSL to secure data in transit, with optional end-to-end payload encryption for added protection. Access control is enforced through ACLs (defining client permissions on topics) and role-based access control (RBAC) for granular authorization, effectively guarding against unauthorized access, data breaches, and tampering.

- 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 using MQTT, the following topics and subscriptions are required:

Topics:

1. **Manual Control Topic** (e.g., "house/light/control"): Devices can publish messages to this topic to manually control the light.
2. **Luminosity Status Topic** (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 the "house/light/control" and "house/light/luminosity" topics.
- The luminosity sensor publishes data to the "house/light/luminosity" topic.

Connections:

- The button publishes messages to the "house/light/control" topic when pressed.
- The luminosity sensor publishes its readings to the "house/light/luminosity" topic.
- The light subscribes to both topics and responds based on the received messages:
 - It switches on or off based on manual control messages from the button.
 - It adjusts automatically based on luminosity readings from the sensor.

This setup enables seamless integration of both manual and automatic light control functionalities.

3) Install and test the broker :

We began by installing Mosquitto and running the broker using the executable file. Subsequently, we tested the functionality of the broker using the `mosquitto_pub` and `mosquitto_sub` commands.

- Command `mosquitto_pub` :

`mosquitto_pub -m "test message" -t button/jaune`

We use the command to send a message in a topic "jaune" contained in the button.

```
C:\Program Files\mosquitto> mosquitto_pub -m "test message" -t button/jaune
C:\Program Files\mosquitto>
```

- Command `mosquitto_sub` :

`mosquitto_sub -t button/# -v`

We use the command to subscribe and listen if a message is arriving.

```
C:\Program Files\mosquitto>mosquitto_sub -t button/# -v
button/jaune test message
```

4) Creation of an IoT device with the nodeMCU board that uses

MQTT communication :

We installed the Arduino IDE and opened an example available in the arduinoMqtt library, named connectESP8266wifiClient.

Main characteristics of the NodeMCU board:

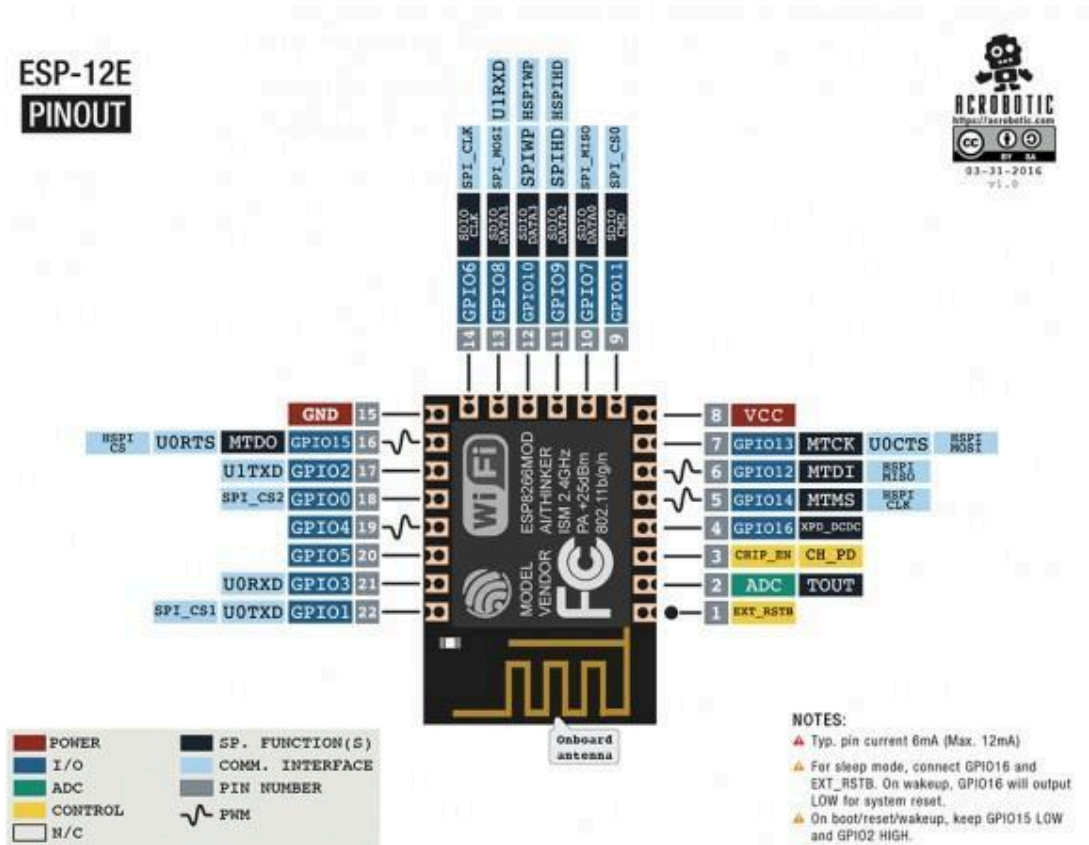
Communication: The ESP8266 is a Wi-Fi chip that supports the IEEE 802.11 b/g/n standard, and the NodeMCU board provides 4 pins for SPI communication.

Programming languages: Several programming languages are compatible with the ESP8266, including AT commands, the ESP8266 SDK, Lua (NodeMCU), C/C++ (Arduino), MicroPython, and JavaScript. For this lab, we will use Arduino.

Power requirements: The NodeMCU board operates within a voltage range of 3.0V to 3.6V.

Input/Output capabilities: The NodeMCU board features 16 GPIO pins (General Purpose Input/Output).

Here is an image retracing the inputs/outputs capabilities :



A message is first sent from the wireless router to the electronic board, which then forwards it to the broker installed on the computer.

The Mosquitto broker handles a collection of topics, each functioning as a queue of messages. Essentially, the broker centralizes the data received from publishers, ensuring it is accessible to all subscribers. In this setup, a publisher is typically a sensor, while a subscriber could be a client, such as a

server hosted in the cloud. Blocs that we have modified from the example :

```
// Informations réseau
const char* ssid = "asni";
const char* password = "asniasni";
const char* mqtt_server = "10.0.1.254";

if (currentMillis - previousMillis >= interval) {
    // save the last time a message was sent
    previousMillis = currentMillis;

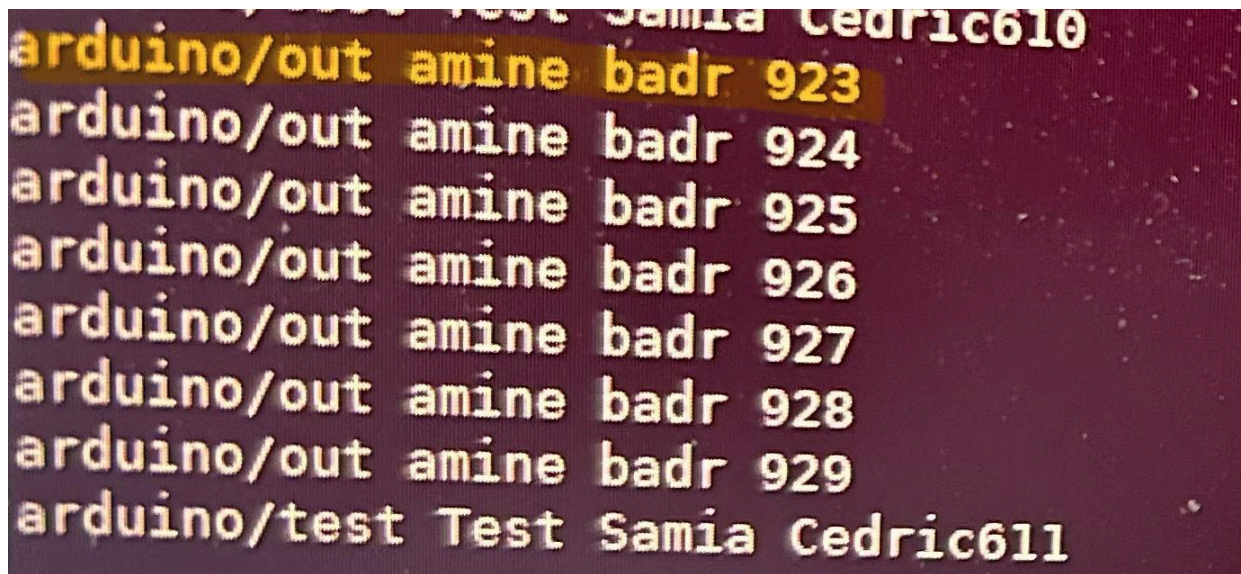
    String payload;

    payload += "amine badr";
    payload += " ";
    payload += count;
```

As part of this activity, we modified the parameters in the example code to enable connection to the Wi-Fi network and publish messages via MQTT. Specifically, we updated the network name (SSID) and password with those of our local network. These adjustments allowed the ESP8266 to connect to the Wi-Fi, establish communication with the Mosquitto broker installed on the computer, and publish or receive messages according to the behaviors defined in the application. This work was carried out in pairs.

As part of this activity, we modified the code to personalize the messages sent by the ESP8266. Specifically, we added a custom payload that includes a string with our names followed by a counter value. This modification allows the system to send unique and identifiable messages at regular intervals.

Additionally, we verified the functionality by observing the output in the terminal, where the messages we sent, along with the incrementing counter, are displayed. This ensured that the messages were properly formatted and successfully published to the specified MQTT topic. This task was carried out in pairs.



```
arduino/out amine badr 923
arduino/out amine badr 924
arduino/out amine badr 925
arduino/out amine badr 926
arduino/out amine badr 927
arduino/out amine badr 928
arduino/out amine badr 929
arduino/test Test Samia Cedric611
```

5) Creation of a simple application

In the section where we implemented manual control of the light using a button, we designed a system where the light turns on when the button is pressed and turns off when the button is released. To achieve this, we created a dedicated function that stores the current state of the button in a temporary variable and processes each case individually. Depending on the button's state, the system publishes a specific message indicating whether the light (LED) should be on or off. During testing, it was observed that pressing the button successfully turned the light on, while releasing the button turned it off, confirming the functionality of this implementation. The code used for this part is shown in the image below.

```
// Si l'état a changé
if (currentState != lastState) {
    // Allume ou éteint la LED en fonction de l'état
    if (currentState == LOW) { // Lumière détectée ou action déclenchée
        digitalWrite(ledPin, LOW); // Allume la LED
        Serial.println("State changed: LIGHT DETECTED, LED ON");
        client.publish("outTopic", "LIGHT DETECTED: LED ON");
    } else { // Lumière absente ou action arrêtée
        digitalWrite(ledPin, HIGH); // Éteint la LED
        Serial.println("State changed: NO LIGHT, LED OFF");
        client.publish("outTopic", "NO LIGHT: LED OFF");
    }

    // Met à jour l'état précédent
    lastState = currentState;
}
```

And the callback function that logs incoming MQTT messages is illustrated in this image.


```

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();
}

```

In another section, we automated the control of the light based on ambient luminosity. Here, we set a threshold of 100 Lux, where the light turns on automatically when the luminosity falls below this threshold and switches off when it exceeds it. Similar to the manual control system, we created a function to handle this behavior. The function reads the value from the light sensor, stores it in a variable, and determines the appropriate action based on the reading. Testing showed that covering the light sensor caused the light to turn on, simulating low-light conditions, while exposing the sensor to sufficient light levels automatically turned the light off. These tests demonstrated the effectiveness of the automated light control system.

6) Conclusion

This lab provided a comprehensive exploration of the MQTT protocol and its application in IoT systems. By working through the various tasks, we gained practical knowledge of setting up and configuring a Mosquitto broker, as well as implementing MQTT communication on an ESP8266 microcontroller. We successfully modified example codes to establish Wi-Fi connections, customize message payloads, and enable the device to publish and subscribe to MQTT topics.

Through the development of manual and automated light control systems, we demonstrated the practical use of MQTT in real-world scenarios, such as managing devices based on environmental conditions or user input. Despite some challenges, such as the inability to retain screenshots due to session restrictions, the results validated the functionality and efficiency of the implemented solutions.

This lab emphasized the lightweight, event-driven nature of MQTT, highlighting its suitability for constrained environments with limited bandwidth and low power. Overall, the experience deepened our understanding of IoT middleware and showcased the potential of MQTT in creating scalable, responsive IoT applications.

Lab 3 : oneM2M REST API

1) Introduction

The project aims to develop an application that simulates the behavior of a device designed during labs 1 and 2 on the ESP8266. These labs focus on handling the service layer of oneM2M and various resources.

To achieve this, we will use a oneM2M stack named ACME and interact with the ACME CSE through a Jupyter Notebook. It is assumed that an IN-CSE has already been installed, configured, and run on the laptop using GitHub. Additionally, the Jupyter Notebook should be installed, and all six chapters must be completed.

Leveraging the flexibility of this approach, the program will be developed using the Notebook library, enabling the seamless creation of REST requests.

2) The Application

We have devices equipped with a button, a light, and a luminosity sensor. The goal is to design a smart home system with the following features:

- Manually turning on the light using the button.
- Automatically turning on the light when the luminosity drops below a predefined threshold.

To achieve this, we will develop an application that simulates this behavior using a RESTful IoT API provided by a oneM2M Common Services Entity (CSE).

3) AE Resources

In this section, we aim to define our AE resources. Specifically, we need to create three distinct AEs:

- A button
- A light
- A luminosity sensor

These entities are specified in the Notebook as follows:

```

CREATE (                                     # CREATE request

# Create the AE resource under the CSEBase
to                                           = cseBaseName,

# Request Parameters
originator                                 = 'Cmyself',           # Assign an originator ID, must start with 'C'
requestIdentifier                         = '123',               # Unique request identifier
releaseVersionIndicator                 = '3',                 # Release version indicator
resourceType                             = Type.AE,            # Type of the resource: AE

# Request Body
primitiveContent =
{
    'm2m:ApplicationEntity': {
        'resourceName':      'Button',
        'App-ID':            'Nbutton',
        'requestReachability': True,           # AE can receive requests
        'supportedReleaseVersions': [ '3' ]    # Supports oneM2M release 3
    }
}

```

```

CREATE (                                     # CREATE request

# Create the AE resource under the CSEBase
to                                           = cseBaseName,

# Request Parameters
originator                                 = 'Clight',           # Assign an originator ID, must start with 'C'
requestIdentifier                         = '123',               # Unique request identifier
releaseVersionIndicator                 = '3',                 # Release version indicator
resourceType                             = Type.AE,            # Type of the resource: AE

# Request Body
primitiveContent =
{
    'm2m:ApplicationEntity': {
        'resourceName':      'Light',
        'App-ID':            'Nlight',
        'requestReachability': True,           # AE can receive requests
        'supportedReleaseVersions': [ '3' ]    # Supports oneM2M release 3
    }
}

```

```

CREATE (                                     # CREATE request

# Create the AE resource under the CSEBase
to                                           = cseBaseName,

# Request Parameters
originator                                 = 'CSensor',          # Assign an originator ID, must start with 'C'
requestIdentifier                         = '123',               # Unique request identifier
releaseVersionIndicator                 = '3',                 # Release version indicator
resourceType                             = Type.AE,            # Type of the resource: AE

# Request Body
primitiveContent =
{
    'm2m:ApplicationEntity': {
        'resourceName':      'Sensor',         # Name of the resource
        'App-ID':            'Nsensor',        # Application ID, must start with 'N'
        'requestReachability': True,           # AE can receive requests
        'supportedReleaseVersions': [ '3' ]    # Supports oneM2M release 3
    }
}

```

We can observe on the server that the following AEs have been successfully created: AE:Button, AE:Light, and AE:Sensor.

CSEBase: id-in/id-in	cse-in																
<div> <div>CSE: cse-in</div> <div> <div>ACP: acpCreateACPs</div> <div>AE: Button</div> <div>AE: CAdmin</div> <div>AE: Light</div> <div>AE: Sensor</div> </div> </div>	<div> <div>AttributesJSONREST UI</div> <table> <thead> <tr> <th>Attribute</th><th>Value</th></tr> </thead> <tbody> <tr> <td>acpi</td><td>["/id-in/acpCreateACPs"]</td></tr> <tr> <td>csi</td><td>"/id-in"</td></tr> <tr> <td>cst</td><td>1</td></tr> <tr> <td>csz</td><td>["application/json","application/cbor"]</td></tr> <tr> <td>ct</td><td>"20231121T143307.411211"</td></tr> <tr> <td>ctm</td><td>"20231121T144135.138549"</td></tr> <tr> <td>it</td><td>"20231121T143307.411211"</td></tr> </tbody> </table> </div>	Attribute	Value	acpi	["/id-in/acpCreateACPs"]	csi	"/id-in"	cst	1	csz	["application/json","application/cbor"]	ct	"20231121T143307.411211"	ctm	"20231121T144135.138549"	it	"20231121T143307.411211"
Attribute	Value																
acpi	["/id-in/acpCreateACPs"]																
csi	"/id-in"																
cst	1																
csz	["application/json","application/cbor"]																
ct	"20231121T143307.411211"																
ctm	"20231121T144135.138549"																
it	"20231121T143307.411211"																

4) Container Resources for Each AE

At this stage, we will add a container resource for each AE. In the oneM2M resource hierarchy, a container acts as a structure designed to hold multiple data instances as child resources. These containers will manage the data specific to each AE as follows:

- For the button and the light, the container will store their respective states (ON or OFF).
- For the luminosity sensor, the container will collect and store luminosity data.

```
CREATE (                                     # CREATE request

# Create the container resource under the AE
to                                     = cseBaseName + '/Button',

# Request Parameters
originator                           = 'Cmyself',           # Set the originator
requestIdentifier                     = '123',               # Unique request identifier
releaseVersionIndicator               = '3',                 # Release version indicator
resourceType                         = Type.Container,       # Type of the resource: container

# Request Body
primitiveContent =
{
    'm2m:Container': {
        'resourceName': 'Button_Status' # Set the resource name
    }
},
```

```

CREATE (                                     # CREATE request

    # Create the container resource under the AE
    to                                     = cseBaseName + '/Light',

    # Request Parameters
    originator                           = 'Clight',           # Set the originator
    requestIdentifier                     = '123',              # Unique request identifier
    releaseVersionIndicator               = '3',                # Release version indicator
    resourceType                         = Type.Container,      # Type of the resource: container

    # Request Body
    primitiveContent =
        {
            'm2m:Container': {
                'resourceName': 'Light_Status'    # Set the resource name
            }
        },

```

```

CREATE (                                     # CREATE request

    # Create the container resource under the AE
    to                                     = cseBaseName + '/Sensor',

    # Request Parameters
    originator                           = 'CSensor',          # Set the originator
    requestIdentifier                     = '123',              # Unique request identifier
    releaseVersionIndicator               = '3',                # Release version indicator
    resourceType                         = Type.Container,      # Type of the resource: container

    # Request Body
    primitiveContent =
        {
            'm2m:Container': {
                'resourceName': 'Sensor_Status'    # Set the resource name
            }
        },

```

5) Content Instance

To initialize the containers for each AE, we will input specific default data:

- The button will be set to "OFF" as it is initially inactive.
- The light will also be set to "OFF" since it is currently unlit.
- The luminosity sensor will start with a default value of 100 Lux

```

CREATE (                                     # CREATE request

    # Create the content instance resource under the container
    to                                     = cseBaseName + '/Button/Button_Status',

    # Request Parameters
    originator                           = 'Cmyself',           # Set the originator
    requestIdentifier                     = '123',               # Unique request identifier
    releaseVersionIndicator               = '3',                 # Release version indicator
    resourceType                         = Type.ContentInstance, # Type of the resource: contentInstance

    # Request Body
    primitiveContent =
        {
            'm2m:ContentInstance': {
                'contentInfo': 'text/plain:0',    # Media type of the content
                'content': 'OFF'                  # The content itself
            }
        }
}

```

```

CREATE (                                     # CREATE request

    # Create the content instance resource under the container
    to                                     = cseBaseName + '/Light/Light_Status',

    # Request Parameters
    originator                           = 'Clight',           # Set the originator
    requestIdentifier                     = '123',               # Unique request identifier
    releaseVersionIndicator               = '3',                 # Release version indicator
    resourceType                         = Type.ContentInstance, # Type of the resource: contentInstance

    # Request Body
    primitiveContent =
        {
            'm2m:ContentInstance': {
                'contentInfo': 'text/plain:0',    # Media type of the content
                'content': 'OFF'                  # The content itself
            }
        }
}

```

```

CREATE (                                     # CREATE request

# Create the content instance resource under the container
to                                           = cseBaseName + '/Sensor/Sensor_Status',

# Request Parameters
originator                                 = 'CSensor',           # Set the originator
requestIdentifier                         = '123',               # Unique request identifier
releaseVersionIndicator                  = '3',                 # Release version indicator
resourceType                             = Type.ContentInstance, # Type of the resource: contentInstance

# Request Body
primitiveContent =
{
    'm2m:ContentInstance': {
        'contentInfo': 'text/plain:0', # Media type of the content
        'content': 'OFF'               # The content itself
    }
}

```

ACME

Base RI:
Originator:
Connected
Auto Refresh

ContentInstance: id-in/cin4180195952902760644

▼ CSE: cse-in

ACP: acpCreateACPs

AE: Button

AE: CAdmin

▼ AE: Light

CNT: Light_Status

CIN: cin_kUbaf1WeGa

▼ AE: Sensor

CNT: Sensor_Status

CIN: cin_ogrQB8SwHc

cse-in/Sensor/Sensor_Status/cin_ogrQB8SwHc

Attributes	JSON	REST UI
Attribute	Value	
cnf	"text/plain:0"	
con	"OFF"	
cs	3	
ct	"20241209T102020,375225"	
et	"20251209T102020,375270"	
lt	"20241209T102020,375225"	
pi	"cnt8373352022326232382"	
ri	"cin4180195952902760644"	
rm	"cin_ogrQB8SwHc"	
st	1	
ty	4	

6) Group Creation

In a scenario with multiple lights, we could create a group to enable simultaneous activation of all lights with a single button press or to retrieve the most recent states of all lights at once. However, since our application involves only one light, this functionality is not necessary.

7) Subscription Creation

Subscription resources are used to monitor events such as resource creation, updates, and other changes. They also define the configuration for notifications, specifying when and how these notifications are sent to a designated target.

When a user creates a subscription resource, they set the parameter "notification content type" (nct) to a value of 1. This value indicates that all attributes of the subscribed resource will be included in the notifications sent to the subscriber.

For example, if the Button AE creates a subscription resource, it sets the notification URI to the resource

identifier of the Button AE. This configuration ensures that the Button AE receives notifications whenever there are changes to the content instance of the Button_Status container.

To enable notifications, a notification server must first be activated. Running a notification server is essential for handling two types of requests:

- **Subscription Verification:** When a new <Subscription> is created, the server sends a verification request to confirm the authenticity of the subscription.
- **Resource Change Notifications:** Whenever a monitored resource is updated, the server is notified. These changes are then pushed to the notification server for processing.

From the server interface, we have the ability to monitor and manage all the containers.

The screenshot displays the ACME server interface. At the top, there's a header with the ACME logo, a 'Base RI' field set to 'id-in', an 'Originator' field set to 'CAdmin', and buttons for 'Connected' and 'Auto Refresh'. Below this, the main content area is divided into two panels. The left panel shows a tree view of containers under 'Container: id-in/cnt1290969148652027542'. The tree includes CSE: cse-in, ACP: acpCreateACPs, AE: Button, AE: CAdmin, AE: Light, and AE: Sensor. The 'AE: Sensor' container is expanded, showing 'CNT: Sensor_Status' with a 'CIN: cin_t8JcGRDq0l' and a 'SUB: Subscription'. The right panel shows the details for 'cse-in/Sensor/Sensor_Status'. It has tabs for 'Attributes', 'JSON', and 'REST UI'. The 'Attributes' tab is active, displaying a table with the following data:

Attribute	Value
cbs	3
cni	1
ct	"20241209T104113.265570"
et	"20251209T104113.265613"
lt	"20241209T104113.265570"
pl	"CSensor"
ri	"cnt1290969148652027542"
rn	"Sensor_Status"
st	1
ty	3

8) Simulation

In the IN-CSE setup, we have three AEs: a Button, a Light, and a Luminosity Sensor. To simulate the ESP8266 use case, a Python program was developed to monitor the button's status and update the light's state accordingly. The program retrieves the button's content and creates new content instances to set the light status to 'ON' or 'OFF' based on the button's state.

Similarly, for the Luminosity Sensor, the application uses an 'if' function to toggle the light based on a brightness threshold. By monitoring the sensor's level, it adjusts the light's status by creating new content instances as needed.

The Python code is available in the Git repository:

<https://github.com/amineben0712/M2M/blob/main/mysrv.py>

9) Conclusion

This project enabled us to apply various oneM2M concepts learned during the MOOC. We gained hands-on experience by creating our first AEs, containers, and content instances. Additionally, we practiced retrieving the latest content instances, as well as updating and deleting containers.

We explored the Discovery function and its labels, implemented group creation, and worked with Access Control, Subscriptions, and Notifications. Finally, we successfully developed our own IN-CSE system based on the use case from previous projects, solidifying our understanding of the concepts.

Lab 4 : Fast application prototyping for IoT

1) Introduction

This final lab session, titled "Fast Prototyping for IoT Applications", represents the culmination of the work done in TP1, TP2, and TP3. The primary objective of this session was to develop a high-level application by integrating a concrete IoT architecture capable of interacting with both real and virtual devices. To achieve this goal, we utilized key tools and technologies such as Node-RED, the oneM2M CSE standard, and the MQTT protocol, offering a flexible and powerful approach for managing and ensuring interoperability between IoT devices.

This session allowed us to apply advanced IoT concepts, particularly in handling the heterogeneity of devices and protocols, while simplifying the development process through Node-RED. As a visual programming tool, Node-RED facilitated the design and rapid deployment of complex applications.

Through various activities, including deploying an architecture, creating interactive dashboards, and managing devices via visual flows, this lab session provided valuable insights into the advantages and challenges of using these tools. It also offered an opportunity to explore the possibilities of interconnecting heterogeneous devices and designing realistic IoT scenarios.

This report will present the different implementation steps, the results obtained, and the conclusions drawn from this experience.

2) Definition of NODE-RED

Node-RED is an open-source visual development platform based on Node.js, designed to easily create data integration applications, particularly in IoT and automation domains. It allows users to connect devices, process data, and automate workflows through an intuitive interface where nodes, representing specific tasks, are interconnected to form flows. Compatible with various protocols and formats such as MQTT, HTTP, and JSON, Node-RED is highly extensible thanks to a vibrant community offering custom modules. Accessible through a web browser, it provides a fast and flexible solution for prototyping and orchestrating complex systems, even with limited programming expertise.

3) Installation of NODE-RED

- **Installation of Node.js**

```

bdiani@insa-21089: ~/.node-red
Fichier  Édition  Affichage  Recherche  Terminal  Aide
js.org/dist/v18.17.1/node-v18.17.1-linux-x64.tar.xz
bash: ~curl : commande introuvable
bdiani@insa-21089:~/Bureau$ curl -o node-v18.17.1-linux-x64.tar.xz https://nodej
s.org/dist/v18.17.1/node-v18.17.1-linux-x64.tar.xz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 22.7M  100 22.7M    0     0  42.7M      0  --:--:-- --:--:-- --:--:--  42.7M
bdiani@insa-21089:~/Bureau$ tar -xvf node-v18.17.1-linux-x64.tar.xz
node-v18.17.1-linux-x64/
node-v18.17.1-linux-x64/share/
node-v18.17.1-linux-x64/share/doc/
node-v18.17.1-linux-x64/share/doc/node/
node-v18.17.1-linux-x64/share/doc/node/lldb_commands.py
node-v18.17.1-linux-x64/share/doc/node/gdbinit
node-v18.17.1-linux-x64/share/systemtap/
node-v18.17.1-linux-x64/share/systemtap/tapset/
node-v18.17.1-linux-x64/share/systemtap/tapset/node.stp
node-v18.17.1-linux-x64/share/man/
node-v18.17.1-linux-x64/share/man/man1/
node-v18.17.1-linux-x64/share/man/man1/node.1
node-v18.17.1-linux-x64/LICENSE
node-v18.17.1-linux-x64/CHANGELOG.md
node-v18.17.1-linux-x64/bin/
node-v18.17.1-linux-x64/bin/node
node-v18.17.1-linux-x64/bin/corepack
node-v18.17.1-linux-x64/bin/npx
node-v18.17.1-linux-x64/bin/npm

```

This screenshot shows the manual installation of Node.js version 18.17.1.

- **Installation of NVM**

```

bdiani@insa-21089: ~/.node-red
Fichier  Édition  Affichage  Recherche  Terminal  Aide
bash: node : commande introuvable
bdiani@insa-21089:~/Bureau$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 16563  100 16563    0     0  181k      0  --:--:-- --:--:-- --:--:--  183k
=> Downloading nvm from git to '/home/bdiani/.nvm'
=> Clonage dans '/home/bdiani/.nvm'...
remote: Enumerating objects: 380, done.
remote: Counting objects: 100% (380/380), done.
remote: Compressing objects: 100% (323/323), done.
remote: Total 380 (delta 43), reused 178 (delta 29), pack-reused 0 (from 0)
Réception d'objets: 100% (380/380), 382.75 Kio | 2.79 Mio/s, fait.
Résolution des deltas: 100% (43/43), fait.
* (HEAD détachée sur FETCH_HEAD)
  master
=> Compressing and cleaning up git repository

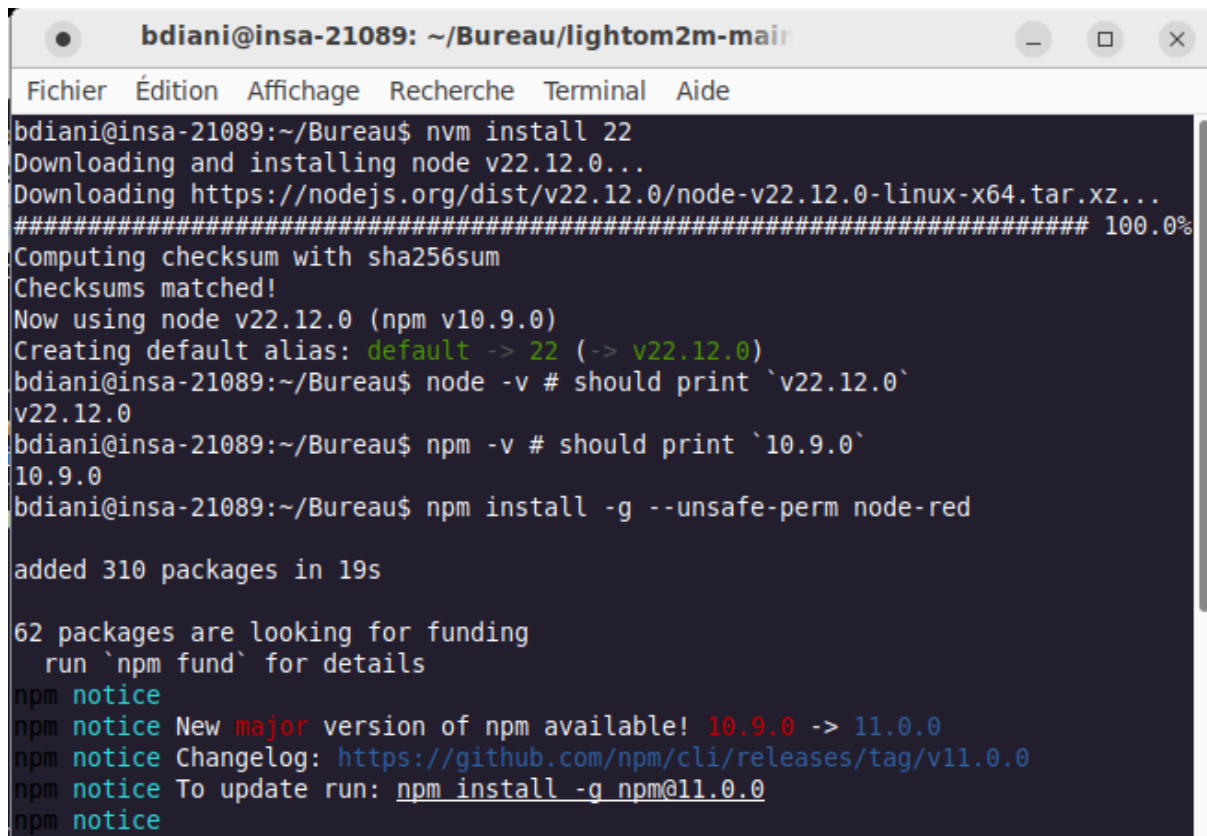
=> Appending nvm source string to /home/bdiani/.bashrc
=> Appending bash_completion source string to /home/bdiani/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm bash_completion
bdiani@insa-21089:~/Bureau$ nvm install 22
bash: nvm : commande introuvable

```

The screenshot shows the installation of NVM to manage Node.js versions, including downloading the installation script from GitHub, followed by the automatic configuration of NVM in the '~/.bashrc' file.

- Node-RED installed



```
bdiani@insa-21089: ~/Bureau/lightom2m-main
Fichier  Édition  Affichage  Recherche  Terminal  Aide
bdiani@insa-21089:~/Bureau$ nvm install 22
Downloading and installing node v22.12.0...
Downloading https://nodejs.org/dist/v22.12.0/node-v22.12.0-linux-x64.tar.xz...
##### 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v22.12.0 (npm v10.9.0)
Creating default alias: default -> 22 (-> v22.12.0)
bdiani@insa-21089:~/Bureau$ node -v # should print `v22.12.0`
v22.12.0
bdiani@insa-21089:~/Bureau$ npm -v # should print `10.9.0`
10.9.0
bdiani@insa-21089:~/Bureau$ npm install -g --unsafe-perm node-red

added 310 packages in 19s

62 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New major version of npm available! 10.9.0 -> 11.0.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v11.0.0
npm notice To update run: npm install -g npm@11.0.0
npm notice
```

The screenshot shows the installation of Node.js (v22.12.0) via NVM, followed by the verification of Node.js (`node -v`) and npm (`npm -v`) versions, showing 22.12.0 and 10.9.0, respectively. Next, Node-RED is globally installed using the command `npm install -g --unsafe-perm node-red`. Finally, a notification indicates that an update to npm version 11.0.0 is available.

- **Starting**

Node-RED

```

bdiani@insa-21089: ~/Bureau/lightom2m-main-src-node-red
Fichier Édition Affichage Recherche Terminal Aide
up to date, audited 1 package in 536ms

found 0 vulnerabilities
bdiani@insa-21089:~/Bureau/lightom2m-main-src-node-red/src/node-red$ node-red
20 Dec 15:35:53 - [info]

Bienvenue sur Node-RED
=====

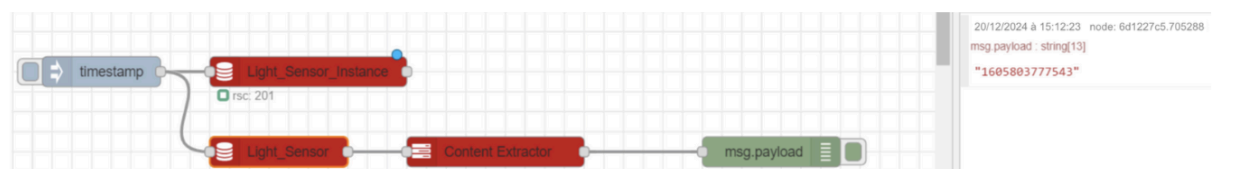
20 Dec 15:35:53 - [info] Node-RED version: v4.0.8
20 Dec 15:35:53 - [info] Node.js version: v22.12.0
20 Dec 15:35:53 - [info] Linux 6.8.0-45-generic x64 LE
20 Dec 15:35:53 - [info] Chargement des noeuds de la palette
20 Dec 15:35:54 - [warn] -----
20 Dec 15:35:54 - [warn] [node-red-contrib-lom2m/Named ACP] Error: Cannot find mod
ule 'request'
Require stack:
- /home/bdiani/Bureau/lightom2m-main-src-node-red/src/node-red/acp-name.js
- /home/bdiani/.nvm/versions/node/v22.12.0/lib/node_modules/node-red/node_modules/
@node-red/registry/lib/loader.js
- /home/bdiani/.nvm/versions/node/v22.12.0/lib/node_modules/node-red/node_modules/
@node-red/registry/lib/index.js
- /home/bdiani/.nvm/versions/node/v22.12.0/lib/node_modules/node-red/node_modules/
@node-red/runtime/lib/nodes/index.js

```

4) Applications :

a) Simple test: Get and display sensors values

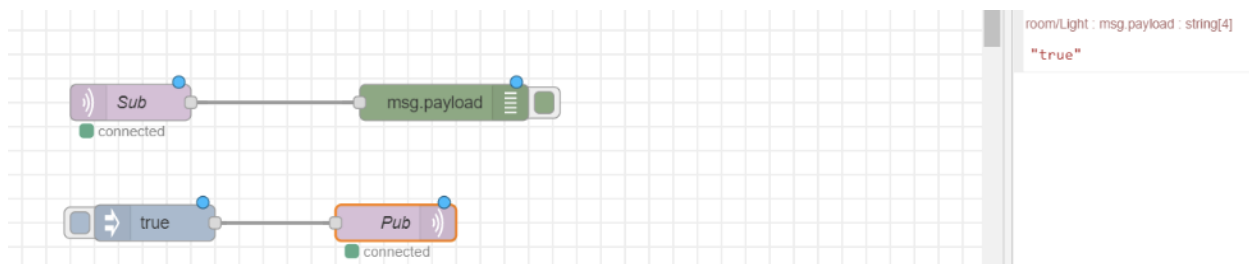
We began our practical exploration by developing a basic application designed to retrieve and display sensor data. Using Node-RED, we configured nodes to collect either simulated values or real-time data from sensors connected via protocols such as MQTT or HTTP. Once the data was collected, we implemented processing nodes to extract relevant information, ready for display. This data was then directed to display nodes, such as a debug node for console visualization or a user interface node for an interactive dashboard presentation. This initial application served as a fundamental step in understanding sensor data management, showcasing Node-RED's flexibility and capability in processing and visualizing information from various IoT devices.



We observe the Node-RED flow retrieving data from a light sensor, processing it with a Content Extractor node, and displaying the extracted value "1605803777543" in the debug console, illustrating efficient data management and visualization.

Attribute	Value
rn	cin_817972049
ty	4
ri	/in-cse/cin-817972049
pi	/in-cse/cnt-496344905
ct	20201119T173644
lt	20201119T173644
st	0
cnf	application/json
cs	13
con	1605803804525

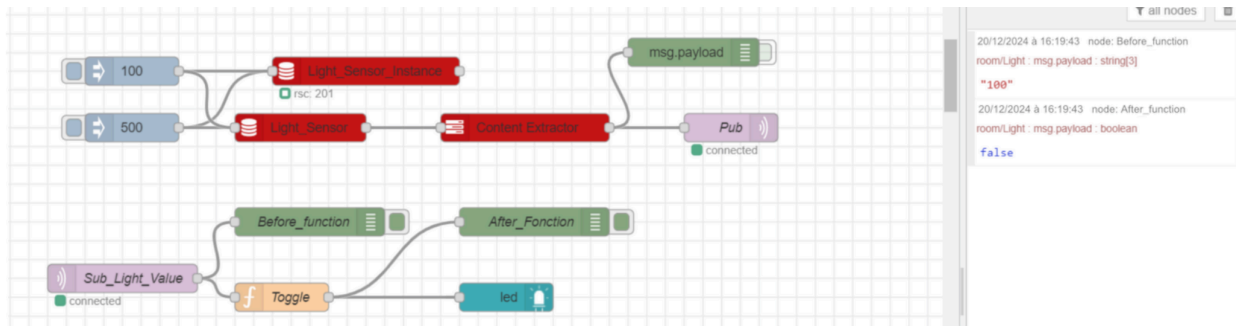
First, we generated a value using the ContentInstance, which we then visualized through the Content Extractor associated with the light sensor. To verify the accuracy of the data, we used the OM2M web page set up during Lab 3. Then, leveraging the capabilities of the two MQTT nodes, we successfully transmitted a message via the MQTT subscriber.



The screenshot shows a Node-RED flow where a `"true"` value is published via the `'Pub'` node and received by the `'Sub'` node, with the output displayed in `'msg.payload'`, confirming successful MQTT communication.

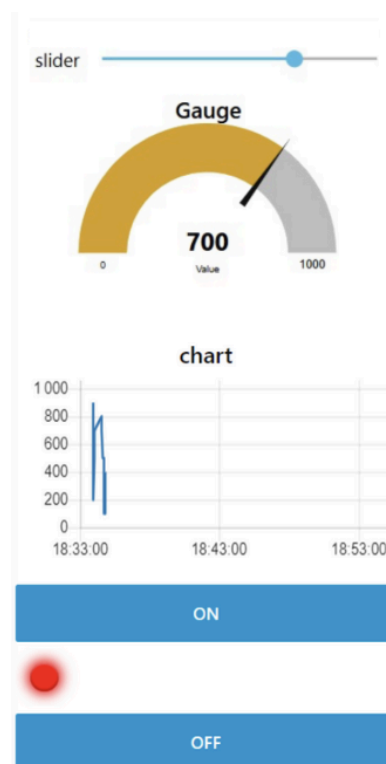
b) Sensors and activators

In the subsequent section, we developed a system focused on modulating light input, using a subscription model to link the data and control the activation of either a green or red LED based on the received light values. For instance, when the light intensity exceeds 250 lux, the system activates the green LED, while values below this threshold trigger the red LED. To illustrate, assigning a light value of 100 lux results in the red LED being illuminated.

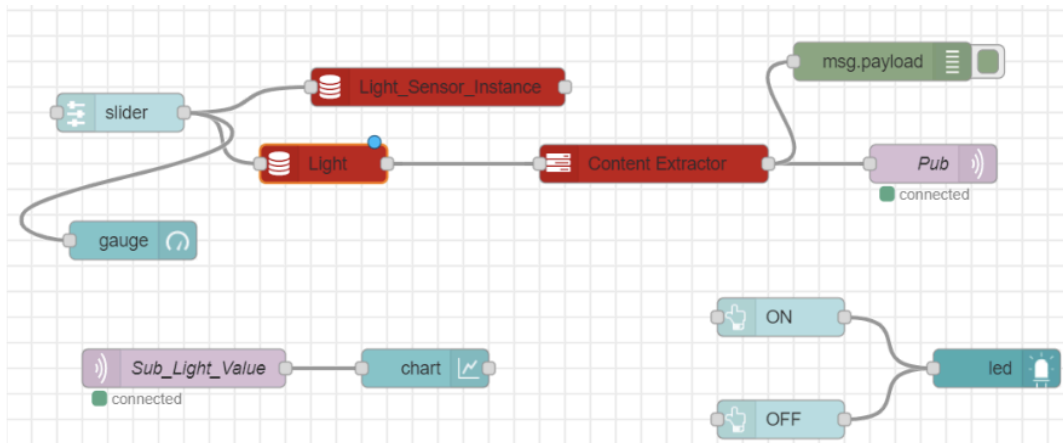


c) Dashboard

In this section, we designed an intuitive dashboard interface, integrating multiple charts for a clear and dynamic visualization of sensor data. Additionally, we incorporated interactive buttons to easily control the activation and deactivation of LEDs. The dashboard structure was meticulously crafted using the dedicated palette, resulting in the following configuration:



This dashboard includes a slider that simulates variations in light intensity by transmitting variable values to the light sensor, with an LED indicator signaling when these values exceed 250 lux. A chart provides real-time visualization of the data sent to the sensor, while two interactive buttons allow for managing the activation of the second LED. The Node-RED flow associated with this setup completes the system.



5) Benefits and drawbacks to build an application with Node-Red

a) Advantages of Node-RED

- Intuitive drag-and-drop interface, accessible to non-developers.
- Rapid development with many pre-built nodes.
- Easy integration with APIs, databases, and IoT protocols.
- Open-source, free, with a large support community.
- Lightweight, ideal for devices like Raspberry Pi or cloud platforms.
- Real-time visual debugging tools.

b) Disadvantages of Node-RED

- Limited for tasks requiring high performance.
- Complex flows can become hard to manage and debug.
- Not well-suited for large-scale or enterprise-level applications.
- Advanced features require programming skills.
- Built-in dashboards lack advanced design options.
- Dependency on third-party nodes can pose security risks.

c) Use Cases

Node-RED is ideal for rapid IoT prototyping, small-scale automation, and edge computing on devices like Raspberry Pi, thanks to its ease of use and quick integration capabilities. However, it is less suitable for large-scale, enterprise-grade, or performance-intensive applications due to its scalability and performance limitations.

6) Conclusion

This lab provided an invaluable opportunity to consolidate our knowledge of IoT concepts and technologies, bridging theory and practical application. By leveraging tools such as Node-RED, the oneM2M CSE standard, and MQTT, we successfully implemented a robust IoT architecture capable of managing and visualizing sensor data, controlling devices, and creating interactive dashboards.

Through hands-on activities, we gained critical insights into the advantages of rapid prototyping, including flexibility, ease of integration, and time efficiency. At the same time, we encountered and

addressed challenges such as managing complex flows and balancing performance with usability. The experience highlighted the potential of Node-RED as a versatile tool for IoT application development, particularly for small-scale and edge computing scenarios, while also revealing its limitations for larger, enterprise-grade applications.

Overall, this lab session underscored the importance of adopting effective tools and architectures in IoT development, equipping us with practical skills and a deeper understanding of the possibilities and constraints inherent in building interconnected systems. It has been a valuable step in our journey toward mastering IoT technologies and designing innovative solutions for real-world applications.