

# Middleware for the IoT Lab

---

IOT WITH ONEM2M

## Table of contents

Lab 1 MQTT.....	2
<b>Introduction</b> .....	2
<b>OM2M definition</b> .....	2
<b>MQTT Specifications</b> .....	2
<b>Creation of an IoT device with the nodeMCU board that uses MQTT communication.....</b>	<b>3</b>
Lab 3 oneM2M REST .....	9
<b>Exercise : Access Control Management in oneM2M.....</b>	<b>9</b>
<b>Scenario : Monitoring application in oneM2M .....</b>	<b>13</b>
Creating the AE .....	13
Creating the content instances.....	15
Lab 4 Fast application prototyping for IoT .....	18
<b>Configuration and use of Node RED</b> .....	<b>18</b>
<b>Conclusion</b> .....	<b>21</b>
<b>Annex</b> .....	<b>23</b>

# Lab 1 MQTT

## Introduction

The goal of this project is to get experience with the oneM2M standard, an architecture that serves as a middleware for IoT devices. During these lab exercises we will expand upon the MQTT protocol and its use and features, using it to create an IoT device with a nodeMCU board. We will then go through the configuration and launching of the Eclipse OM2M platform, handling the resource tree through a web browser using AJAX interface and handling the resource tree through a REST client.

## OM2M definition

OneM2M was formed in July 2012 by several of the world's leading ICT standards development organizations to create a global technical standard for interoperability concerning IoT technologies. The Eclipse OM2M is an open-source platform for M2M interoperability, providing a RESTful API for creating and managing M2M resources. It includes several primitive procedures to enable machines authentication, resource discovery, containers management and more.

## MQTT Specifications

To explore the capabilities of the MQTT protocol for IoT, we begin by making a short state of the art about the main characteristics of this protocol:

### State of the art MQTT

To analyse the characteristics of this protocol we search the answer to several questions regarding the MQTT:

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

The typical architecture of such an IoT system is a or few MQTT brokers that are connected to several other MQTT clients and servers that subscribe to a certain type of information, they are centralised by a broker which allows a form of management. As the MQTT protocol is very lightweight this system can easily support a big network of brokers and clients.

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

The MQTT protocol is based on the IP(TCP) protocol, which means that it is a client-server architecture. It allows cross-platform communication between different types of networks, and it is highly scalable, allowing networks to be added without interfering with the current structure. A disadvantage of this is that it may not be optimal for smaller networks. It also doesn't distinguish the concepts of services and protocols so it may not be suitable to describe new technologies in new networks.

### What are the different versions of MQTT?

There are two different variants of MQTT and several versions. MQTT v3.1.0 and MQTT v3.1.1 are very similar although 3.1.1 is most commonly used. MQTT v5 is a newer version with updated features but

it is currently limited in use, a factor that may change going forward as more and more people select the newer adaptation. Lastly there is MQTT-SN which is mainly designed to work over the IP(UDP) protocol, negating many of the limitations mentioned in the answer above.

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

MQTT payload encryption allows end-to-end encryption of application data, it means that the data is encrypted “end to end” and not just at the broker level, between the broker and clients, which aids in protecting packages sent through it even on untrusted environments.

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 behaviour:

- 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 behaviour and what will the connection be in terms of publishing or subscribing?

The button, light and light sensor would need to be configured as MQTT clients. The button and light sensors would be publishers and the light would be a subscriber (client). To manage this connection, one would also need a broker, a server that receives and processes messages from all the clients and then routes them to their appropriate destination.

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

All along this lab we will use the nodeMCU board based on ESP8266, after installing and testing the Mosquitto broker (annex 1) we need to configure the mosquitto.conf file to connect our board to mosquitto. For that we need to add the lines below and save it in a new folder :

```
listener 1883 172.20.10.2
allow_anonymous true
```

Then we need to launch mosquitto in 3 different command prompt : one to execute it and two others for the publication and subscription part.

Indeed, the aim of this lab is to connect the ESP32 to the Mosquitto broker to subscribe to a MQTT topic. Then we will have to publish the data sensed by the light sensor. Thus, we will create an application able to determine if the light is on or not, if it is sunny or not, and with this application we will be able to control the integrated board LED to simulate an automated lighting system.

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

The ESP8266 is a microcontroller 32 bits.

The board integrates 802.11 WIFI connection, UART RX and TX communication, SPI and I2C communication. It needs between 3 to 3.6 V to operate.

The programming language here is C++ using Arduino's IDE however other can be used like Lua, C, microPython, Javascript.

It integrates 16 GPIO pins IN/OUT, it has an integrated LED light, a 10 bits ADC, a PWM and a Button which can be programmed.

## Creation of the application

We send a message from the wireless router to the electronic card, then the message is sent to the broker that we have installed on the computer. The mosquitto broker manages a set of topics. A topic can be described as a queue of messages. Basically, the broker centralises the information sent by the publishers so that everything is available for every subscriber. The publisher is a sensor and the subscriber a client, like a server on the cloud.

To make our light management application we first must set the connection between the board and the local Wi-Fi network using our own Wi-Fi network.

---

### STEP 1 : WiFi Setup

---

```
71 // Setup WiFi network
72 WiFi.mode(WIFI_STA);
73 WiFi.hostname("ESP_" MQTT_ID);
74 WiFi.begin("iPhone", "paupau31");
75 LOG_PRINTFLN("\n");
76 LOG_PRINTFLN("Connecting to WiFi");
77 while (WiFi.status() != WL_CONNECTED) {
78     delay(500);
79     LOG_PRINTFLN(".");
80 }
81 LOG_PRINTFLN("Connected to WiFi");
82 LOG_PRINTFLN("IP: %s", WiFi.localIP().toString().c_str());
```

And we also must configure the IP address and port of our MQTT broker :

```
// Re-establish TCP connection with MQTT broker
LOG_PRINTFLN("Connecting");
network.connect("172.20.10.2", 1883);
```

Here we encountered several problems, at the beginning we couldn't establish the connection due to the Wi-Fi frequency band, indeed the ESP8266 is designed for 2,4 GHz and not for 5 GHz, and we tried with our 5 GHz Wi-Fi band initially, that's why we couldn't establish the connection. It took us time to determine that issue because we have thought that it was due to our code or a problem with our library version.

After a final check with the 2.4 GHz band, we need to add the subscription bloc and publication bloc.

---

## STEP 2 : PUB/SUB blocs

---

```
// Add subscribe here if required

MqttClient::Error::type rc = mqtt->subscribe(
    MQTT_TOPIC_SUB, MqttClient::QOS0, processMessage
);
if (rc != MqttClient::Error::SUCCESS) {
    LOG_PRINTFLN("Subscribe error: %i", rc);
    LOG_PRINTFLN("Drop connection");
    mqtt->disconnect();
    return;
}
} else {
{
    // Add publish here if required

    i = analogRead(A0); //on lit la valeur du capteur et
    snprintf(msgmqtt, 50, "%f", i); // sprintf(outputs
    const char* buf = msgmqtt;
    MqttClient::Message message;
    message.qos = MqttClient::QOS0;
    message.retained = false;
    message.dup = false;
    message.payload = (void*) buf;
    message.payloadLen = strlen(buf);
    mqtt->publish(MQTT_TOPIC_SUB, message);
}
```

We can test the connection by sending messages from the broker to the board or from the Board to the broker that the communication is working perfectly (annex 2). We created and named broker's topic. We will use the "test/TEST-ID/sub" topic to monitor the light sensor and we will retrieve the data.

---

### STEP 3 :LED management bloc

---

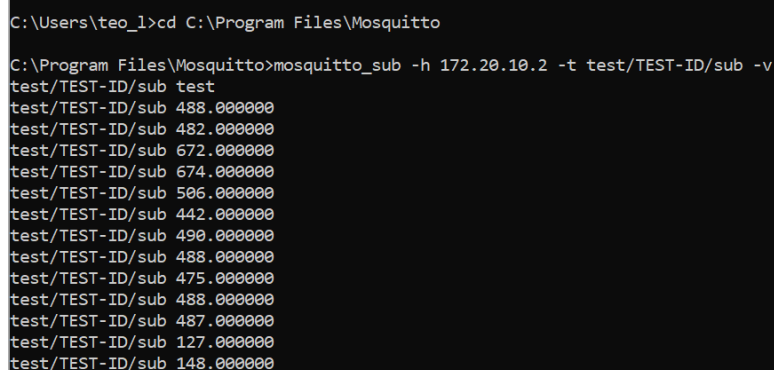
```
// ===== Subscription callback =====
void processMessage(MqttClient::MessageData& md) {
    const MqttClient::Message& msg = md.message;
    char payload[msg.payloadLen + 1];
    memcpy(payload, msg.payload, msg.payloadLen);
    payload[msg.payloadLen] = '\0';
    String MQTT_DATA = ""; //initialisation de la variable MQTT_DATA (vide ici)
    for (int i=0;i<msg.payloadLen;i++){
        MQTT_DATA += (char)payload[i];
    }

    if(MQTT_DATA.toFloat()>600){
        LOG_PRINTFLN(
            "SUNNY Message arrived: qos %d, retained %d, dup %d, packetid %d, payload:[%s]",
            msg.qos, msg.retained, msg.dup, msg.id, payload
        );
        digitalWrite(2,HIGH);
        delay(50);
    } else {
        LOG_PRINTFLN(
            "DARK Message arrived: qos %d, retained %d, dup %d, packetid %d, payload:[%s]",
            msg.qos, msg.retained, msg.dup, msg.id, payload
        );
        digitalWrite(2,LOW);
        delay(50);
    }
}
```

#### **Light management code**

In this part, we code the management of the internal board blue LED, we test if the light sensor value is higher than 600, if that is the case we turn the LED ON, the blue LED is connected to the GPIO 2 and it is working with reverse logic, that is why we turn it HIGH to light off the blue LED when data retrieved are higher than 600, otherwise we turn it LOW to light up the blue LED when the value decrease under 600. We must read the subscription topic messages arriving from the light sensor to manage the internal LED, that is why we must implement our IF function in the Subscription callback bloc which process those messages.

We can test if our dedicated topic is receiving data correctly and if we have the right behaviour.



```
C:\Users\teo_1>cd C:\Program Files\Mosquitto
C:\Program Files\Mosquitto>mosquitto_sub -h 172.20.10.2 -t test/TEST-ID/sub -v
test/TEST-ID/sub test
test/TEST-ID/sub 488.000000
test/TEST-ID/sub 482.000000
test/TEST-ID/sub 672.000000
test/TEST-ID/sub 674.000000
test/TEST-ID/sub 506.000000
test/TEST-ID/sub 442.000000
test/TEST-ID/sub 490.000000
test/TEST-ID/sub 488.000000
test/TEST-ID/sub 475.000000
test/TEST-ID/sub 488.000000
test/TEST-ID/sub 487.000000
test/TEST-ID/sub 127.000000
test/TEST-ID/sub 148.000000
```

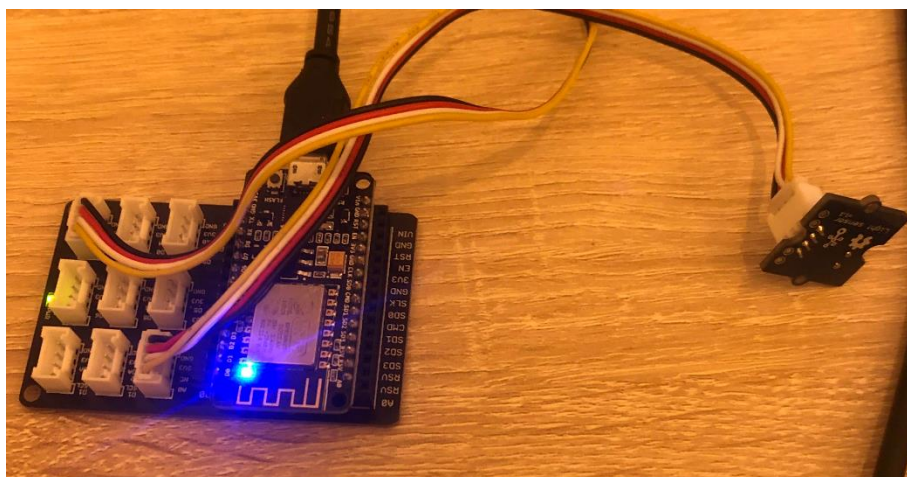
**Data retrieved from test/TEST-ID/sub topic, sent by the light sensor**

As we can see the data monitoring is working perfectly. We can now test the light management according to the data. After testing it we obtained the results expected :

```
COM3
20:17:44.669 -> SUNNY Message arrived: qos 0, retained 0, dup 0, packetid 0, payload:[672.000000]
20:17:56.647 -> MQTT - Keepalive, ts: 60889
20:17:56.647 -> MQTT - Publish, to: test/TEST-ID/sub, size: 10
20:17:56.647 -> MQTT - Yield for 12000 ms
20:17:56.647 -> MQTT - Process message, type: 13
20:17:56.647 -> MQTT - Keepalive ack received, ts: 60900
20:17:56.695 -> MQTT - Process message, type: 3
20:17:56.695 -> MQTT - Publish received, qos: 0
20:17:56.695 -> MQTT - Deliver message for: test/TEST-ID/sub
20:17:56.695 -> SUNNY Message arrived: qos 0, retained 0, dup 0, packetid 0, payload:[674.000000]
20:18:08.632 -> MQTT - Keepalive, ts: 72897
20:18:08.632 -> MQTT - Publish, to: test/TEST-ID/sub, size: 10
20:18:08.632 -> MQTT - Yield for 12000 ms
20:18:08.680 -> MQTT - Process message, type: 13
20:18:08.680 -> MQTT - Keepalive ack received, ts: 72907
20:18:08.680 -> MQTT - Process message, type: 3
20:18:08.680 -> MQTT - Publish received, qos: 0
20:18:08.680 -> MQTT - Deliver message for: test/TEST-ID/sub
20:18:08.680 -> DARK Message arrived: qos 0, retained 0, dup 0, packetid 1, payload:[506.000000]
```

### **Light management according to data retrieved from the sensor**

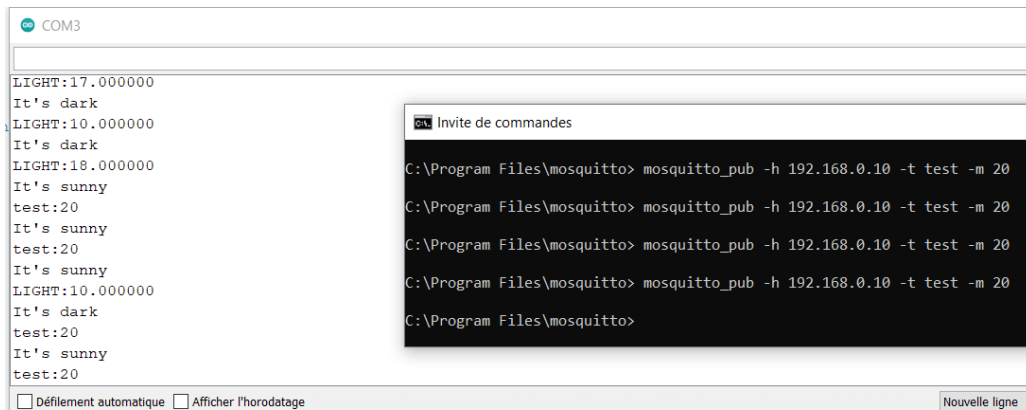
The blue LED behaves as expected, when we cover the light sensor with the hand, data values are decreasing below 600 and the LED lights on. When we remove our hand, data values increase above 600 and the LED lights off. As shown on serial panel, after each retrieved data, we indicate the state of the room “sunny or dark”.



**Blue LED lighted up according to the light sensor value**

The automated behaviour works just as well as the manual one, indeed we can simulate a switch by setting manually the value of the PIN sending a value with the broker or an Arduino command line, the program is reading the value and lights up the blue LED.

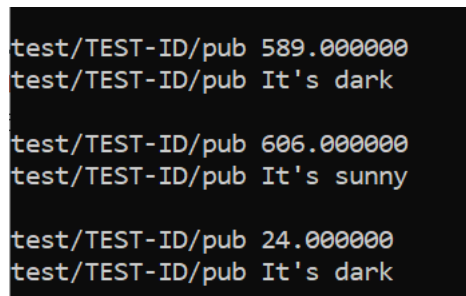




The screenshot shows a COM3 terminal window on the left and a terminal window titled 'Invite de commandes' on the right. The COM3 window displays a sequence of light sensor readings and room state messages: LIGHT:17.000000, It's dark, LIGHT:10.000000, It's dark, LIGHT:18.000000, It's sunny, test:20, It's sunny, test:20, It's sunny, LIGHT:10.000000, It's dark, test:20, It's sunny, test:20. The terminal window shows four Mosquitto publish commands: C:\Program Files\mosquitto> mosquitto\_pub -h 192.168.0.10 -t test -m 20, repeated four times, followed by C:\Program Files\mosquitto>.

### **Manual light management using the topic “test” to set and publish data with 20 as threshold**

We also can publish the state of the room directly on the broker, in instance we use the topic test/TEST-ID/pub here :



The screenshot shows a terminal window with the following MQTT publish commands: test/TEST-ID/pub 589.000000, test/TEST-ID/pub It's dark, test/TEST-ID/pub 606.000000, test/TEST-ID/pub It's sunny, test/TEST-ID/pub 24.000000, test/TEST-ID/pub It's dark.

### **Light sensor value and room state published in Mosquitto**

The behaviour for the push button is the same, however we can implement a priority function to prioritize between the push button and the light sensor which one the LED does have to listen if both are sending contradictory values.

## Lab 3 oneM2M REST

The goal of this section is to get experience in interacting with the Eclipse OM2M platform. It will go through the configuration and launching of the platform, handling the resource tree through a web browser using AJAX interface and handling the resource tree through a REST client, Postman here.

### Exercise : Access Control Management in oneM2M

The aim is to master access control using oneM2M Access Control Policies. For that, we will create an ACP and associate it to two different Application Entities and configure each ACP with different rules. ACP are used by the CSE to control access to resources.

ACP are stored in the resource tree and referred or are used by other resources through an accessControlPolicyID attribute (acpi). Any HTTP request should be compliant to certain ACP to access to a specific resource. ACP have their own access control policies using Self privileges, indeed ACP contain the rules/privileges defining who can do what with resources. Thus, resources can't be used in a semantic operation if it is not allowed by the corresponding ACP.

We first need to install and launch Postman to configure oneM2M using HTTP requests. Then we need to start both in-cse and mn-cse and use the following URL to access to them :

<http://127.0.0.1:8080/webpage> (in-cse link)

<http://127.0.0.1:8282/webpage> (mn-cse link)

We have to configure the config file present in each folder and set the database reset line from true to false in order to keep saved all our modifications.

First, we can observe the attributes of a data container with a GET request, we can find an acpi attribute (Access Control Policy Identifier) that points to the ACP managing the resource as seen on the picture below.

The screenshot shows a Postman interface with a GET request to `http://127.0.0.1:8080/~in-cse/cnt-760475184`. The Headers tab is active, showing `X-M2M-Origin: admin:admin` and `Accept: application/xml`. The Body tab is also active, showing the XML response in Pretty view. The XML response is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" xmlns:hd="http://www.onem2m.org/xml/protocols/homedomain" xmlns="DATA">
3   <ty>3</ty>
4   <ri>/in-cse/cnt-760475184</ri>
5   <pi>/in-cse/CAE240577328</pi>
6   <ct>20211027T214026</ct>
7   <lt>20211027T214026</lt>
8   <acpi>/in-cse/acp-103638613</acpi>
9   <et>20221027T214026</et>
10  <st>1</st>
11  <mni>10</mni>
12  <mbs>10000</mbs>
13  <mia>0</mia>
14  <cni>1</cni>
15  <pbs>201</pbs>
16  <ol>/in-cse/in-name/SmartMeter/DATA/ol</ol>
17  <la>/in-cse/in-name/SmartMeter/DATA/la</la>
18 </m2m:cnt>
```

**GET request and its response : observation of the acpi attribute**

It is possible to modify the existing ACP using PUT method.



### **PUT request and its response : ACP modification**

Two ACP have been created with different access rules (acop = Access Control Operation) for two different application entities, however we keep the same originator : the guest2 user.

Attribute	Value	
rn	test	
ty	1	
ri	/in-cse/acp-904630888	
pi	/in-cse	
ct	20211027T224520	
lt	20211027T230635	
pv	AccessControlOriginator	AccessControlOperation
	admin:admin	63
	guest2:guest2	2
pvs	AccessControlOriginator	AccessControlOperation
	admin:admin	63

### **First ACP allowing only the retrieve of data**

Attribute	Value	
rn	test2	
ty	1	
ri	/in-cse/acp-83226223	
pi	/in-cse	
ct	20211027T232816	
lt	20211027T232909	
pv	AccessControlOriginator	AccessControlOperation
	guest2:guest2	3
	admin:admin	63
pvs	AccessControlOriginator	AccessControlOperation
	admin:admin	63

### **Second ACP allowing the retrieve and creation of data**

AE can be associated with the ACP just created using the following line : <acpi>/in-cse/in-name/mon\_acp</acpi>

Thus, 2 AE are created using PostMan and this tutorial :

[https://wiki.eclipse.org/OM2M/one/REST\\_API](https://wiki.eclipse.org/OM2M/one/REST_API)

Data and descriptor container are also created and filled with Data and Description content instances.

http://127.0.0.1:8080/~in-cse

POST http://127.0.0.1:8080/~in-cse

Params Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL XML

```

1 <m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="Monitoring_App">
2   <api>app-sensor</api>
3   <acpi>/in-cse/acp-904630888</acpi>
4   <lbl>Type/sensor Category/temperature Location/home</lbl>
5   <rr>false</rr>
6 </m2m:ae>

```

### **AE Monitoring application creation pointing to the first ACP (with acop = 2)**

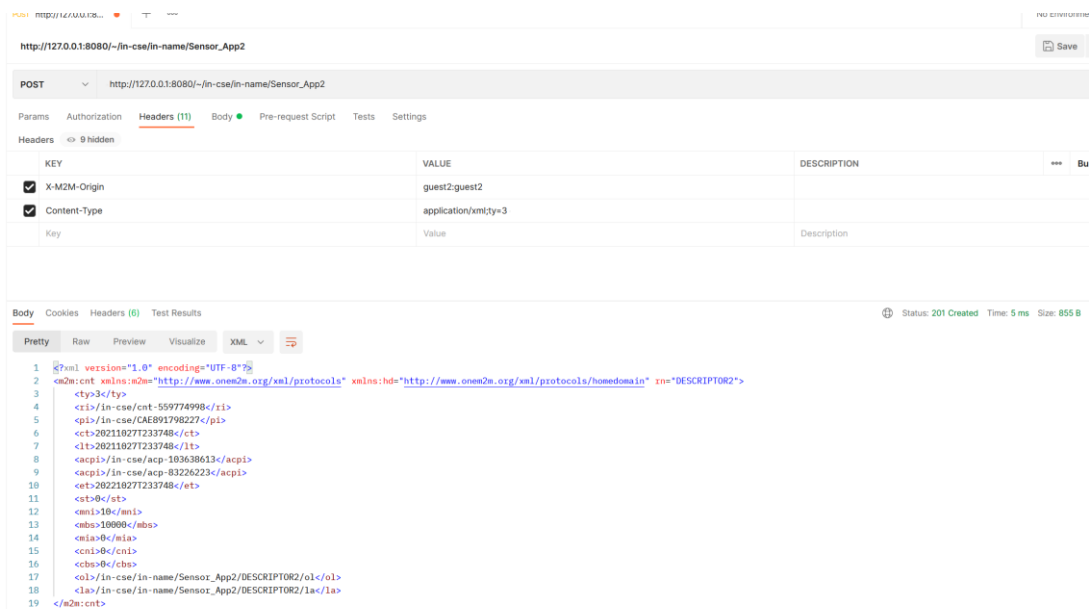
After the Monitoring application creation, we can observe (annexe 3) that it is pointing to the ACP previously created.

Finally, we have :

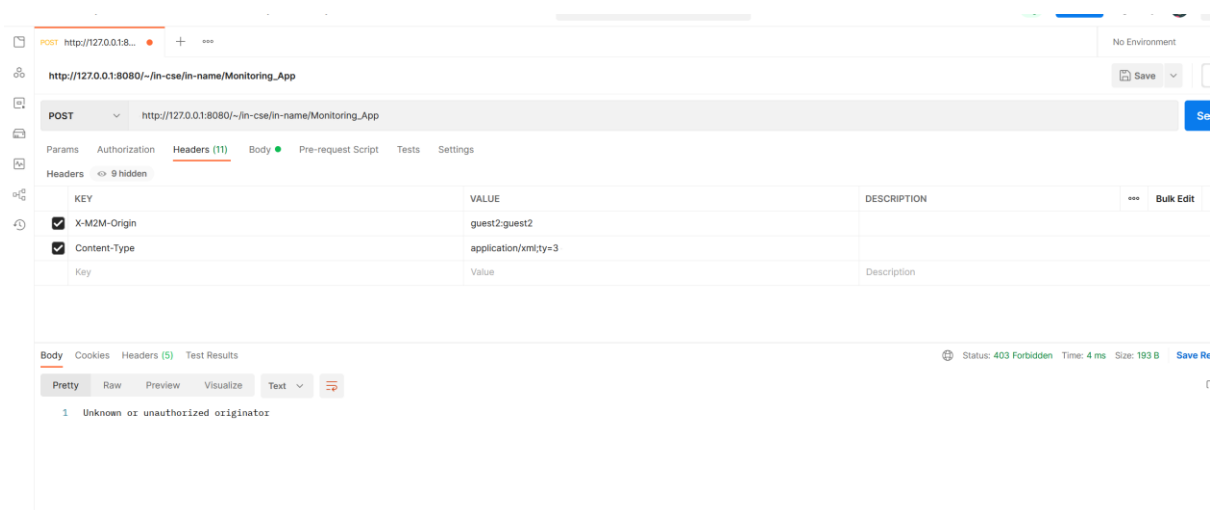
- a monitoring application that will only retrieve the data (acop = 2)
- a sensor application that will be able to retrieve AND create (acop = 3)

The next step consists in testing the retrieve of data and the creation of data.

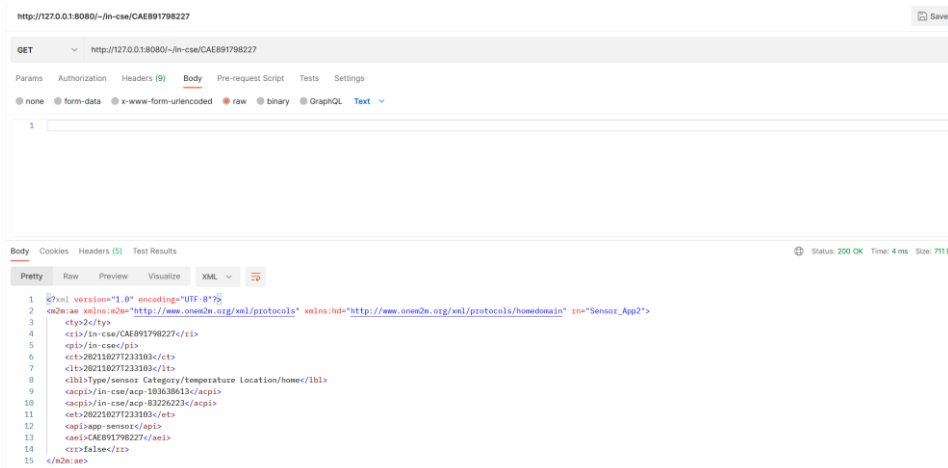
As shown on the picture below : with the sensor application we are able to create a Descriptor Content Instance while we are not allowed to create a content instance through the monitoring application using guest2:guest2 as origin, indeed it is not allowed for that originator to use the create operation as we configured it with the first ACP, however, as predicted the retrieve operation is working perfectly fine for both applications.



**Response of the POST method using the Sensor application : success.**



**Response of the POST method using the monitoring application : forbidden.**



### **Response of the GET for both application : success.**

We can conclude that we can manage the access of our resources linking them to ACP resource in the CSE base. It allows us to protect some critical data, to set a kind of priority architecture. Indeed, security is a key point to deploy an IoT solution, we must secure some information like identity, data and restrain some operations.

## **Scenario : Monitoring application in oneM2M**

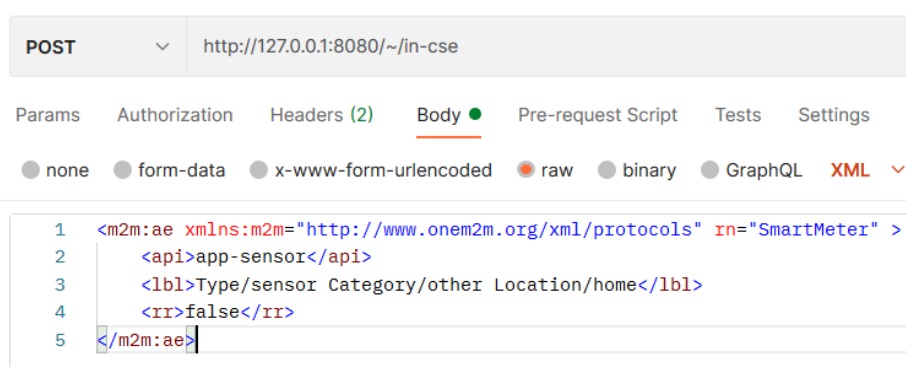
The goal of this scenario is to simulate several devices and sensors and to register an application that will monitor the values pushed by the sensors using the subscription/notification mechanism. After the start of the platform with the launch of the server IN-CSE and the gateway MN-CSE, we can access to the developer interface with the link : <http://127.0.0.1:8080/webpage> where we can access to the tree resources.

As said before, we used Postman application in order to send the HTTP requests and interact with the REST API using XML/Json.

## **Creating the AE**

We begin by creating 3 AE on the gateway with the following names:

- SmartMeter
- LuminositySensor
- TemperatureSensor

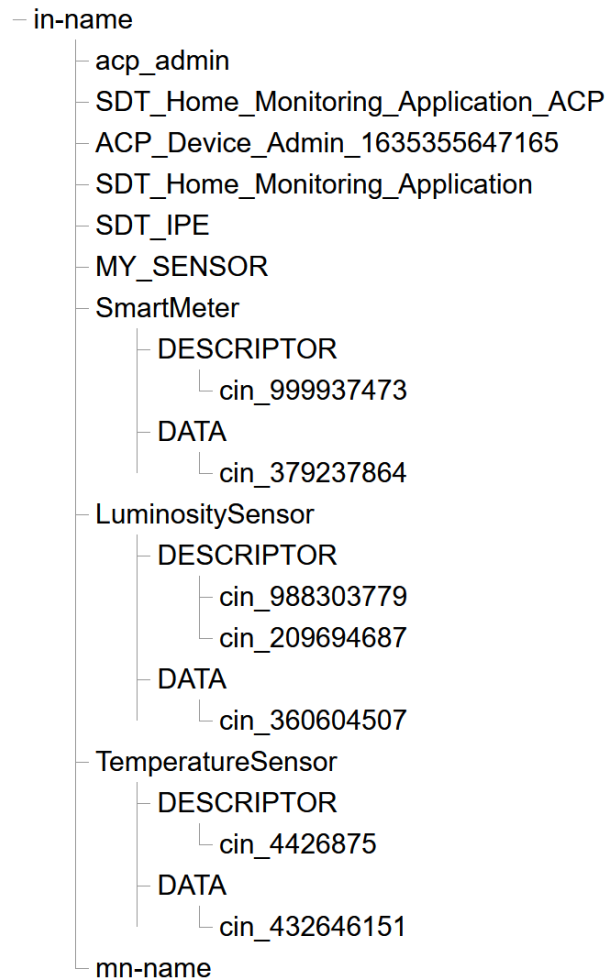


### **HTTP POST request SmartMeter AE creation**

We used the request above to create the 3 AE and we obtain the following resource tree :

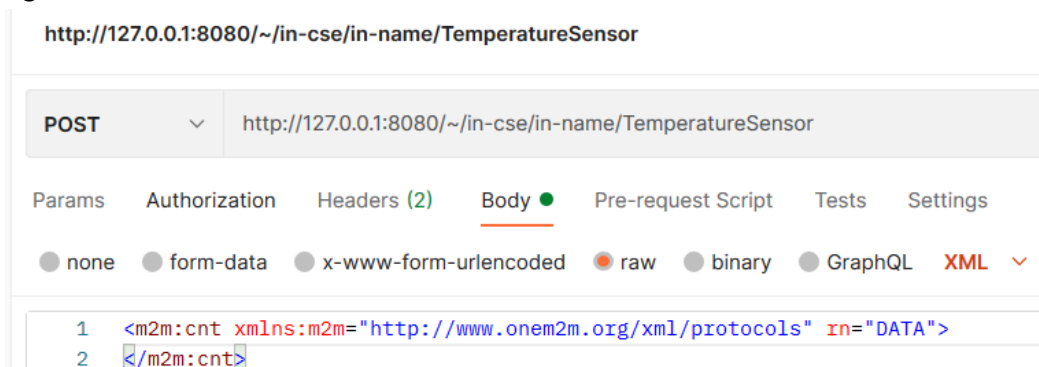
### OM2M CSE Resource Tree

<http://127.0.0.1:8080/~in-cse/cnt-932502725>

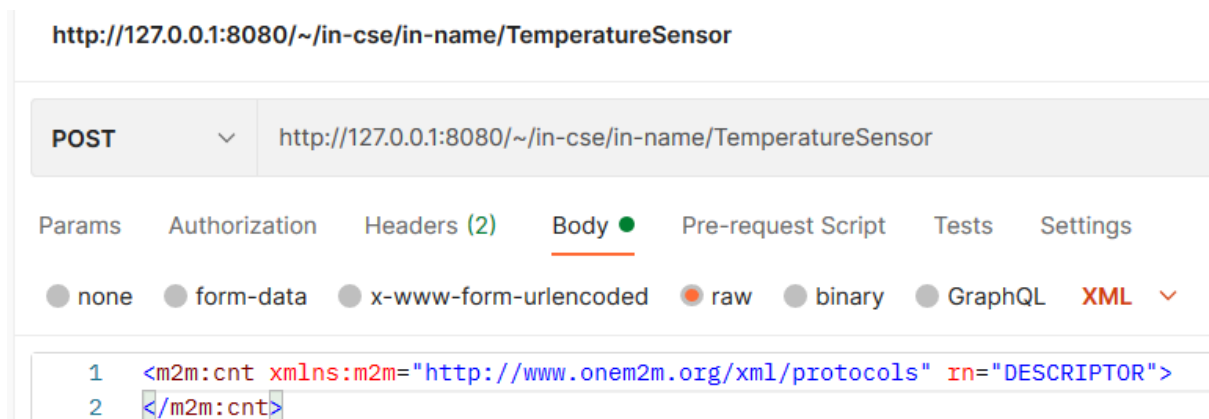


### **Tree Resource, with the 3 AE created**

As shown in the figure each device has 2 containers: DESCRIPTOR and DATA that we added using the following code :



### **HTTP request TemperatureSensor DATA container creation**



### **HTTP request TemperatureSensor DESCRIPTOR container creation**

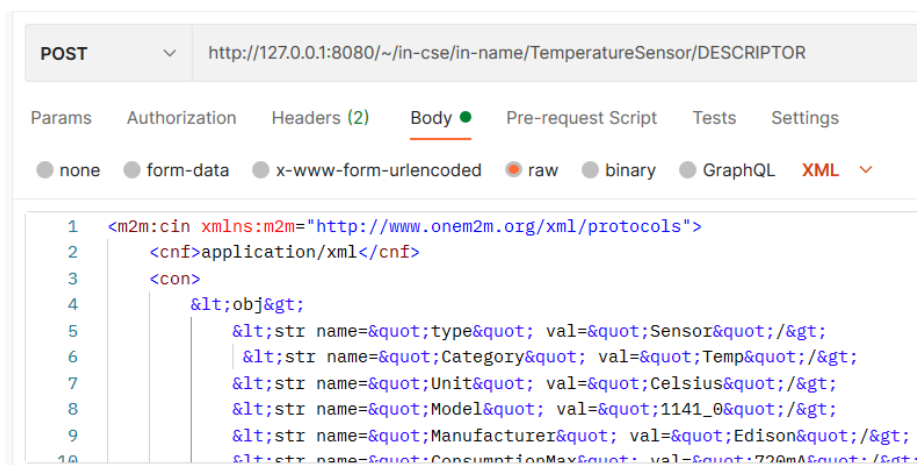
We have an issue with the creation of the DESCRIPTOR container, indeed we paste the text from the Lab 3 PDF, and as we can see there is a text font mismatch, we had to rewrite everything.

Attribute	Value
type	Temperature_Sensor
Volt	2.6Ã¢â?¬â?¹Ã¢â?¬â?¹VÃ¢â?¬â?¹Ã¢â?¬â?¹DC
Operatingtempmax	80C
location	Home
appld	MY_SENSOR

### **Descriptor content bug,**

## Creating the content instances

We then go to the DESCRIPTOR container and fill in the description of the sensor :



### **HTTP request TemperatureSensor DESCRIPTOR content instance code**



con	Attribute	Value
	type	Sensor
	Category	Temp
	Unit	Celsius
	Model	1141_0
	Manufacturer	Edison
	ConsumptionMax	720mA
	VoltageMin	1.6VDC
	VoltageMax	5.8VDC
	OperatingTemperatureMin	0C
	OperatingTemperatureMax	60C
	location	Home
	appld	TemperatureSensor

### **TemperatureSensor DESCRIPTOR content instance final shape on the resource tree**

We do the same thing for the DATA container to fill it with content instances.

Now we must monitor the 3 sensors and retrieve their data. For that, we must start the Monitor server with the command : `java -jar monitor.jar` which listens on port 1400 and context=/monitor.

```
C:\Users\teo_1>cd C:\Users\teo_1\Desktop
C:\Users\teo_1\Desktop>cd monitor
Le chemin d'accès spécifié est introuvable.

C:\Users\teo_1\Desktop>java -jar monitor.jar
Starting server..
The server is now listening on
Port: 1400
Context: /monitor
```

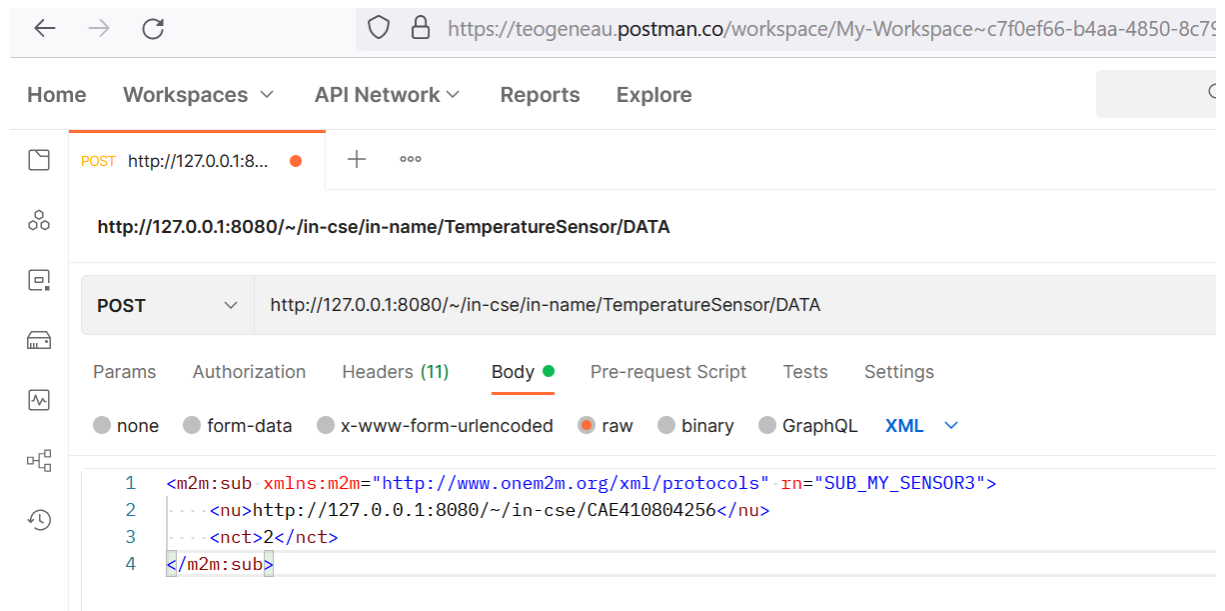
### **Monitoring application**

Then we create the AE representing the monitoring application using the following code :

```
<m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="MY_SENSOR" >
  <api>app-sensor</api>
  <lbl>Type/sensor Category/temperature Location/home</lbl>
  <poa> http://127.0.0.1:1400/monitor</poa>
  <rr>true</rr>
</m2m:ae>
```

To achieve the monitoring of new values pushed by the sensors we have to set the reachability attribute (rr) to true and to add a point of access (poa) using the url of the monitor, thus, we are redirecting the subscription notification to the port 1400, indeed our AE monitoring application will notify the monitor app (running in the command prompt) of those new values pushed. What is more, it is essential to create required subscriptions to receive data from the 3 sensors previously

created. For that a POST HTTP request is sent to create a Subscription resource to receive events from the 3 sensors.



**HTTP POST request to create required Subscription in each sensors pointing to the AE monitoring application.**

After the request, we can see the Subscription resource on the resource tree.

OM2M CSE Resource Tree

<http://127.0.0.1:8080/~in-cse/sub-901468200>

```

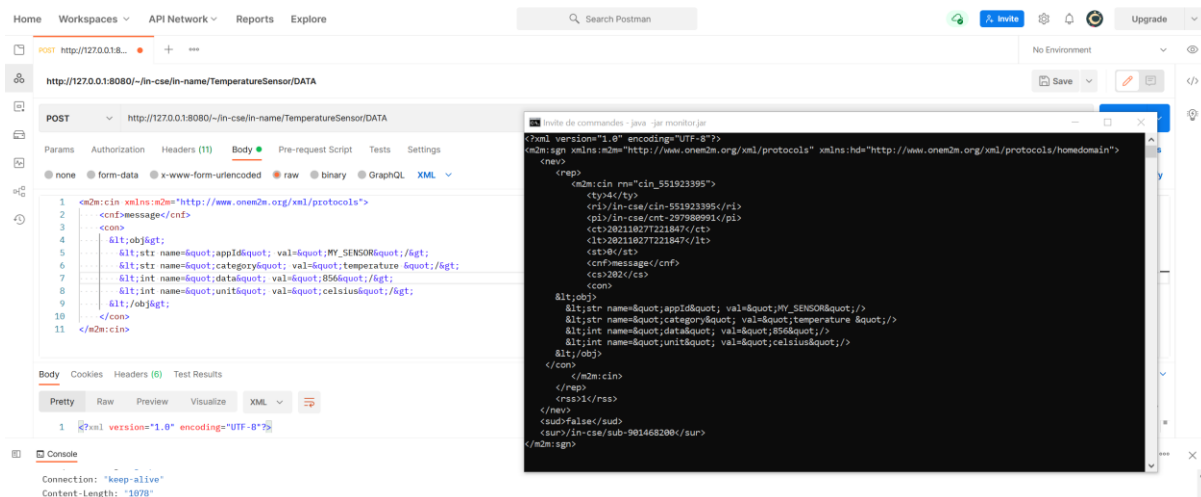
in-name
├── acp_admin
│   ├── SDT_Home_Monitoring_Application_ACP
│   ├── ACP_Device_Admin_1635355647165
│   ├── SDT_Home_Monitoring_Application
│   ├── SDT_IPE
│   ├── MY_SENSOR
│   └── SmartMeter
│       ├── DESCRIPTOR
│       └── DATA
│           ├── cin_379237864
│           └── SUB_MY_SENSOR3
├── LuminositySensor
│   ├── DESCRIPTOR
│   └── DATA
│       ├── cin_360604507
│       └── SUB_MY_SENSOR3
├── TemperatureSensor
│   ├── DESCRIPTOR
│   └── DATA
│       ├── cin_432646151
│       └── SUB_MY_SENSOR3
└── MONITOR
    └── mn-name

```

Attribute	Value
m	SUB_MY_SENSOR3
ty	23
ri	/in-cse/sub-901468200
pi	/in-cse/cnt-297980991
ct	20211027T221639
lt	20211027T221639
acpi	<div style="border: 1px solid #ccc; padding: 2px;"> AccessControlPolicyIDs </div>
nu	<div style="border: 1px solid #ccc; padding: 2px;"> http://127.0.0.1:8080/~in-cse/CAE410804256 </div>
nct	2

**Resource tree with the Subscription resource**

All that is left to do is to create a new data and observe the monitor's behaviour : when a new data is pushed in a sensor, the event is published to every interested subscriber on the IN, a "Notify" which has been routed by the monitor AE appears on the monitor app.



### **Monitor notification observed in the command prompt due to a new data of TemperatureSensor**

Finally, we created a monitoring application, with the simulation of sensors, which can retrieve data pushed by the sensors using the Subscription/Notification mechanism and redirecting each subscription message to the monitor app listening on port 1400.

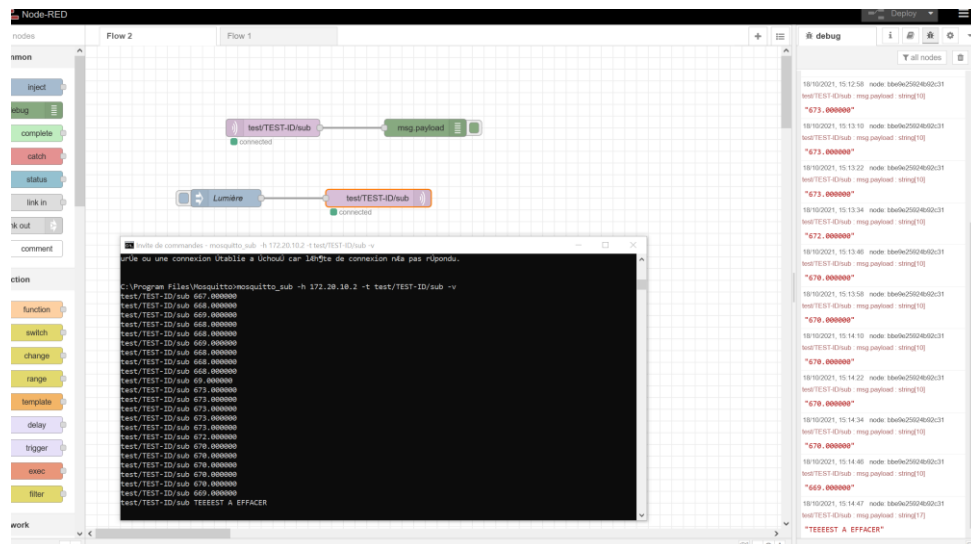
## Lab 4 Fast application prototyping for IoT

### Configuration and use of Node RED

We will integrate MQTT and ESP8266 board to Node-RED and create a high-level application thanks to node-RED, which is faster than HTTP implementation and interact with Eclipse OM2M and MQTT.

We have not used real sensors and actuators that's why we will simulate them using Node-RED's nodes.

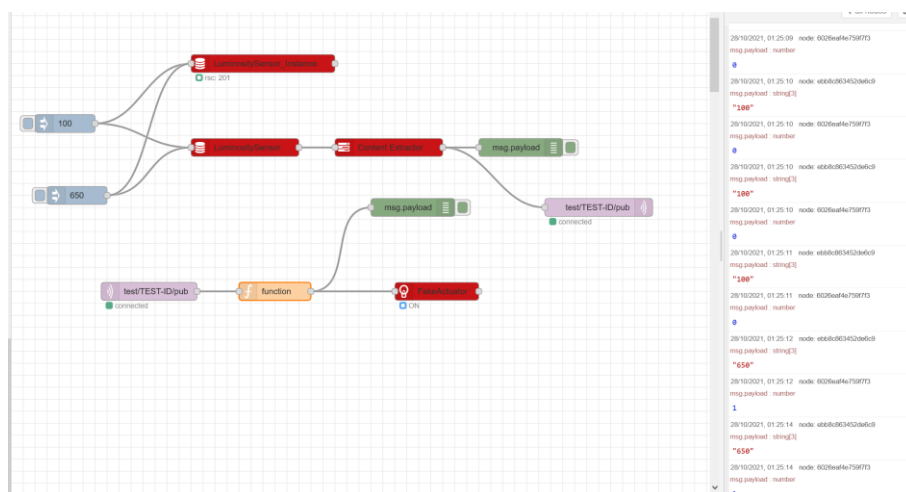
For the first flow that we created, we display on the debug console the last data produced by the light sensor, indeed the ESP8266 board is connected to MQTT, and it is publishing new values on the test/TEST-ID/sub topic. We are also using an inject node with a test payload (string) to send a message using the MQTT subscriber.



**Flow 1 : Node-RED MQTT connection**

The next flow consist in activating some activators with MQTT using values get on oneM2M. We retrieve the last data produced, for that we will inject a number : 100 or 650 as a LuminositySensor content Instance and get the last value published on the LuminositySensor data container using the node Content Extractor and we will published that value on MQTT.

Then we use the MQTT subscribe node to get the last value published and with a function node, we will trigger an activators according to a threshold test : if the value is equal or higher than 650 we will set off the light activators, and if it is equal to 100 or less we will set on the light.

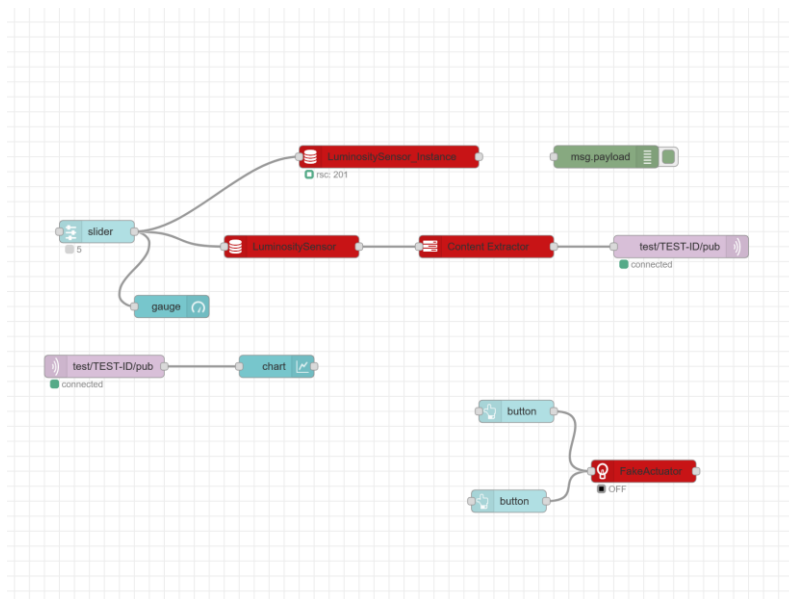


**Flow 2 : Node-RED MQTT and oneM2M interconnected application**

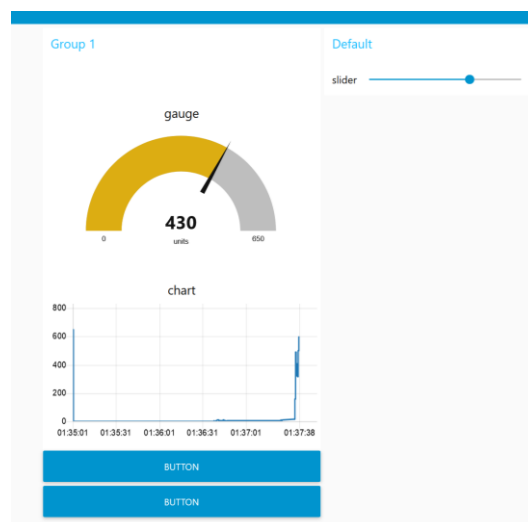
```
1 var msg1 = { payload:1 };
2 var msg2 = { payload:0 };
3 if(msg.payload === "100") {
4   return msg2;
5 } else {
6   return msg1;
7 }
```

**Function code used for the threshold**

For the third flow, we used dashboard nodes to plot sensor values and observe them in the dashboard <http://127.0.0.1:1880/ui> tab. The dashboard slider node is used to send a light value to oneM2M. We retrieve using MQTT nodes as seen before, and we use the chart node to observe the light value's evolution. We can also trigger the activator manually in the dashboard using manual button node.



**Flow 3 : use of oneM2M and MQTT nodes and monitoring of light value in the dashboard**



**Dashboard tab observation**

For the flow 4, we tested HTTP request using the according nodes in Node-RED instead of PostMan. We could test successfully GET and POST request from Node-RED to oneM2M. For example, for the GET method we have to use an inject node sending a msg.header using json format :

```
{
  "X-M2M-Origin": "admin:admin",
  "Accept": "application/json"
}
```

$\{$ 

D

A monitoring application has also been elaborated to monitor sensor using oneM2M and Postman as a REST client to handle the HTTP request. Furthermore, we have learned about node-RED, we learned to develop faster application interconnecting and ESP8266 board, MQTT and oneM2M.

In brief, this project has given us a lot of hands-on experience with some of the concepts involved in using IoT middleware. Finally, it has shown the basic usage of such processes, laying the groundwork for being able to use it proficiently in future projects.

# Annex

```

Références  Publipostage  Révision  Affichage  Aide

Invite de commandes - mosquito -v -c C:\Users\teo_l\Desktop\INSA\5A_ISS_BIS\mosquitto.conf
Microsoft Windows [version 10.0.19042.1288]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\teo_l>cd C:\Program Files\Mosquitto

C:\Program Files\Mosquitto>mosquitto -v -c C:\Program Files\Mosquitto\new\mosquitto.conf
1634554975: Error: Unable to open config file C:\Program.

C:\Program Files\Mosquitto>mosquitto -v -c C:\Program Files\Mosquitto\new\mosquitto.conf
1634555002: Error: Unable to open config file C:\Program.

C:\Program Files\Mosquitto>mosquitto -v -c C:\Users\teo_l\Desktop\INSA\5A_ISS_BIS\mosquitto.conf
1634555028: mosquitto version 2.0.12 starting
1634555028: Config loaded from C:\Users\teo_l\Desktop\INSA\5A_ISS_BIS\mosquitto.conf.
1634555028: Opening ipv4 listen socket on port 1883.
1634555028: mosquitto version 2.0.12 running

```

```

Connect:ip2260WIFIClient(TCOP2022) | Arduino 1.8.16
Fichier Edition Croquis Outils Aide

mosquitto -v -c C:\Users\teo_l\Desktop\INSA\5A_ISS_BIS\5A_ISS_BIS\mosquitto\mosquitto.conf
1633706474: Received PINGREQ from TEST-ID
1633706474: Sending PINGRESP to TEST-ID
1633706486: Received PINGREQ from TEST-ID
1633706486: Sending PINGRESP to TEST-ID
1633706494: Received PUBLISH from TEST-ID (0, 0, 0, 0, 'test/TEST-ID/pub', ... (5 bytes))
1633706494: Sending PUBLISH to auto-0316CD38-8598-7883-EF35-9F9F3599100E (0, 0, 0, 0, 'test/TEST-ID/pub', ... (5 bytes))
1633706504: Received PINGREQ from TEST-ID
1633706504: Sending PINGRESP to TEST-ID
1633706513: Received PINGREQ from auto-0316CD38-8598-7883-EF35-9F9F3599100E
1633706513: Sending PINGRESP to auto-0316CD38-8598-7883-EF35-9F9F3599100E
1633706516: Received PINGREQ from TEST-ID
1633706516: Sending PINGRESP to TEST-ID
1633706528: Received PUBLISH from TEST-ID (0, 0, 0, 0, 'test/TEST-ID/pub', ... (5 bytes))
1633706528: Sending PUBLISH to auto-0316CD38-8598-7883-EF35-9F9F3599100E (0, 0, 0, 0, 'test/TEST-ID/pub', ... (5 bytes))
1633706534: Received PINGREQ from TEST-ID
1633706534: Sending PINGRESP to TEST-ID
1633706546: Received PINGREQ from TEST-ID
1633706546: Sending PINGRESP to TEST-ID
1633706556: Received PUBLISH from TEST-ID (0, 0, 0, 0, 'test/TEST-ID/pub', ... (5 bytes))
1633706556: Sending PUBLISH to auto-0316CD38-8598-7883-EF35-9F9F3599100E (0, 0, 0, 0, 'test/TEST-ID/pub', ... (5 bytes))
1633706564: Received PINGREQ from TEST-ID
1633706564: Sending PINGRESP to TEST-ID
1633706573: Received PINGREQ from auto-0316CD38-8598-7883-EF35-9F9F3599100E
1633706573: Sending PINGRESP to auto-0316CD38-8598-7883-EF35-9F9F3599100E
1633706576: Received PINGREQ from TEST-ID
1633706576: Sending PINGRESP to TEST-ID

Microsoft Windows [version 10.0.19042.1287]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\teo_l>cd C:\Program Files\Mosquitto

C:\Program Files\Mosquitto>mosquitto_sub -t 'test' -m 'tesstst'
C:\Program Files\Mosquitto>mosquitto_pub -t 'TEST-ID' -m 'tesstst'
C:\Program Files\Mosquitto>mosquitto_sub -h 172.20.10.2 -t 'TEST-ID' -m 'tesstst'
C:\Program Files\Mosquitto>

test/TEST-ID/pub Hello
test/TEST-ID/pub Hello
test/TEST-ID/pub Hello

```

**Annex 1 : 3 command prompt to launch MQTT.**



```
C:\Program Files\Mosquitto>mosquitto_pub -h 172.20.10.2 -t test/TEST-ID/sub -m "tesssst"
C:\Program Files\Mosquitto>
```

```
MQTT - Publish received, qos: 0
MQTT - Deliver message for: test/TEST-ID/sub
//Message arrived: qos 0, retained 0, dup 0, packetid 15, payload:[tesssst]
MQTT - Publish, to: test/TEST-ID/pub, size: 5
MQTT - Yield for 30000 ms
```

### **Annex 2 : command line to send a message in board's console**

Attribute	Value
rn	Monitoring_App
ty	2
ri	/in-cse/CAE828554029
pi	/in-cse
ct	20211027T230235
lt	20211027T230235
lbl	<ul style="list-style-type: none"> <li>Type/sensor</li> <li>Category/temperature</li> <li>Location/home</li> </ul>
acpi	<div>AccessControlPolicyIDs</div> <div> /in-cse/acp-103638613  /in-cse/acp-904630888 </div>
et	20221027T230235
api	app-sensor
aei	CAE828554029
rr	false

### **Annex 3**