

## 5ISS - Middleware For the IoT



---

### Labs Report

---

*Supervisor:*

G. Ghada (INSA teacher)

*Students:*

ESTIVAL Emilie, PAGÈS Maxime

# TP1- MQTT protocol for IoT

## 1. Objective

The goal of this lab is to explore the capabilities of the MQTT protocol for IoT. To do that you will first make a little state of art about the main characteristics of MQTT, then you will install several software on your laptop to manipulate MQTT. Finally, you will develop a simple application with an IoT device (ESP8266) using the MQTT protocol to communicate with a server on your laptop.

## 2. MQTT Based on web resources and the previous course respond to those questions:

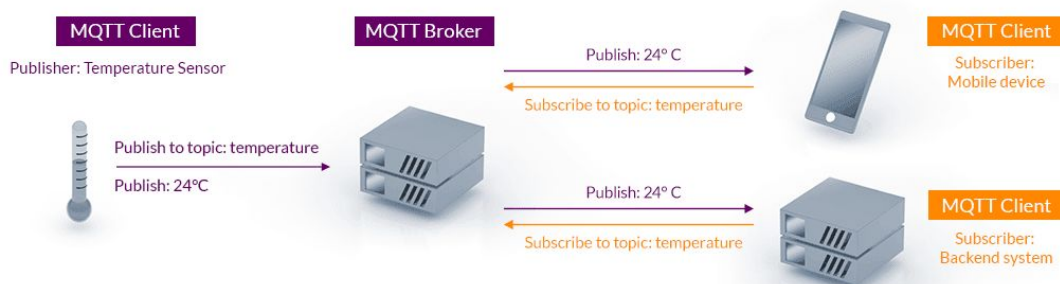
<https://mqtt.org/>

[https://www.eclipse.org/community/eclipse\\_newsletter/2014/february/article2.php](https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php)

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

MQTT has a client/server model, where every sensor is a client and connects to a server, known as a broker, over TCP.

MQTT is message oriented. Every message is a discrete chunk of data, opaque to the broker. Every message is published to an address, known as a topic. Clients may subscribe to multiple topics. Every client subscribed to a topic receives every message published to the topic.



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

TCP is the IP protocol used under MQTT. It is an extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements

### • *What are the different versions of MQTT?*

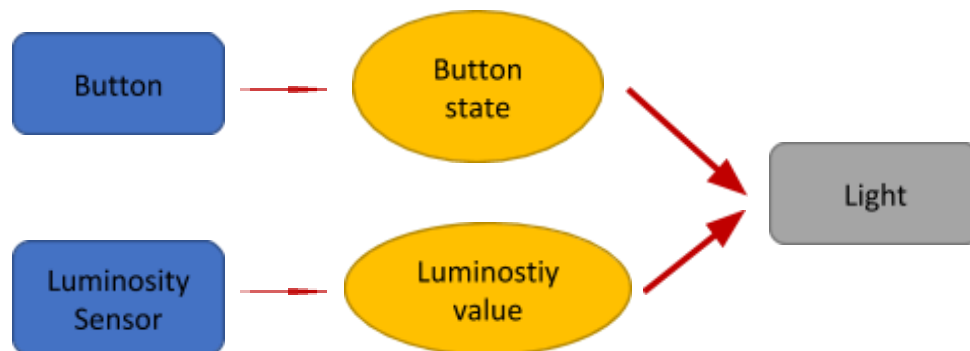
Version 5.0 and v3.1.1 are now OASIS standards (v3.1.1 has also been ratified by ISO).

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

MQTT brokers may require username and password authentication from clients to connect. To ensure privacy, the TCP connection may be encrypted with SSL/TLS.

- ***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?***



```
/usr/local/Cellar/mosquitto/1.6.12/
```

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

*f. Add a publish/subscribe behavior in your device*



## TP3 - Interact with the oneM2M RESTful architecture using Eclipse OM2M

### 1. Objective

The goal of this lab is to understand how to interact with the Eclipse OM2M platform:

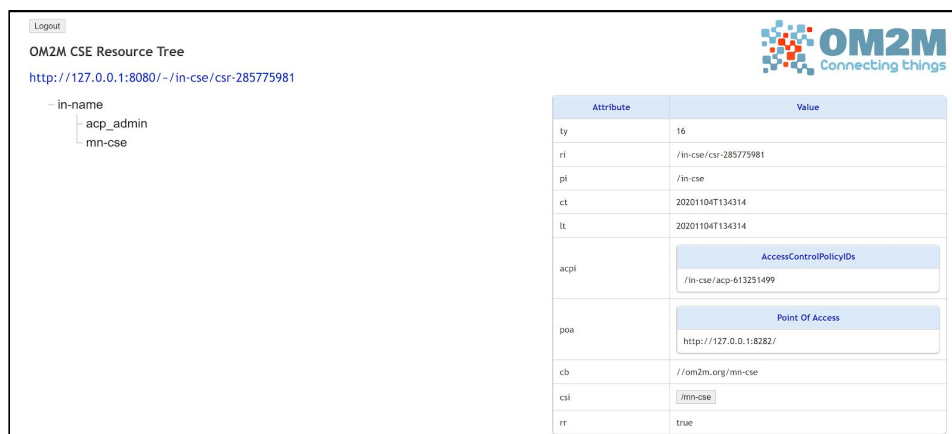
- configure and launch the platform
- handle the resource tree through a web browser using AJAX interface (JS)
- handle the resource tree through a REST client (we used POSTMAN)

### 2. OM2M platform handling

#### a. Get Started

We used IN-CSE as a server, and MN-CSE as a Gateway. We could access via the web interface OM2M: <http://127.0.0.1:8080/webpage>.

After a successful authentication, the IN-CSE resource is displayed. We can see the "in-cse" cseBase sub-resources and attributes. Using the In-CSE web interface we can access all authenticated gateways. We can notice the existence of one authenticated MN-CSE with id "mn-cse".



The screenshot shows the Eclipse OM2M web interface. On the left, the "OM2M CSE Resource Tree" is displayed with a tree structure under "in-name" containing "acp\_admin" and "mn-cse". The URL bar shows "http://127.0.0.1:8080/~in-cse/csr-285775981". On the right, a table lists attributes and their values. The "acpi" attribute has a value of "AccessControlPolicyIds" with a sub-value of "/in-cse/acp-613251499". The "poa" attribute has a value of "Point Of Access" with a sub-value of "http://127.0.0.1:8282/". The "cb" attribute has a value of "om2m.org/mn-cse". The "csi" attribute has a value of "mn-cse". The "rr" attribute has a value of "true".

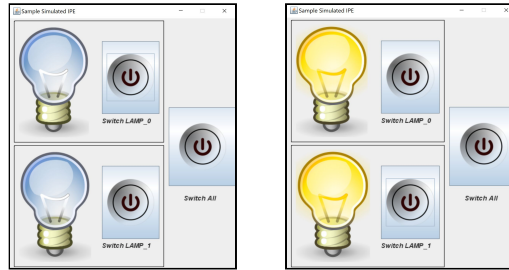
Attribute	Value
ty	16
ri	/in-cse/csr-285775981
pi	/in-cse
ct	20201104T134314
lt	20201104T134314
acpi	AccessControlPolicyIds /in-cse/acp-613251499
poa	Point Of Access http://127.0.0.1:8282/
cb	/om2m.org/mn-cse
csi	/mn-cse
rr	true

We can click on the "mn-cse" resource to display remote MN-CSE sub-resources and attributes. The IN-CSE will act as a proxy to retarget the requests to the MN-CSE.

#### b. Developer interface

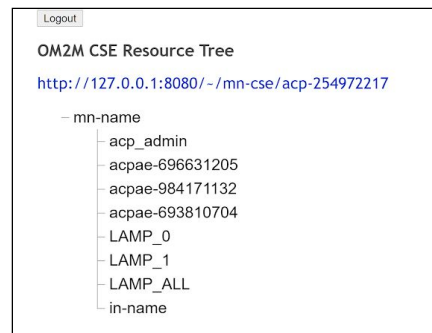
In the MN shell, we type `ss ipe` and then start with the bundle id of the sample ipe plugin (30). It allows to display a java graphical interface with two lamps and three switches. We

can use this interface to switch ON/OFF the two lamps separately or simultaneously. We can see on the capture below that we can easily turn ON the lamps with the power switches.



In the MN-CSE web interface we have the lamp resources displayed on the resource tree:

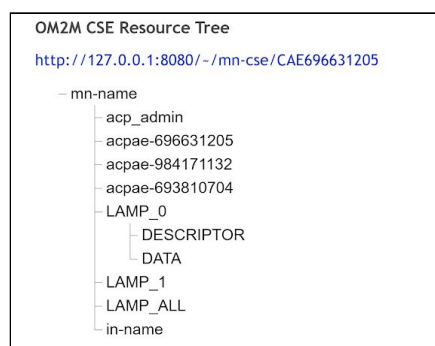
- "LAMP\_0": an AE resource enables to handle lamp 0.
- "LAMP\_1": an AE resource enables to handle lamp 1.
- "LAMP\_ALL": is an AE resource enables to handle simultaneously lamp 0 and lamp 1.



### c. Container Resources

When we click on the "LAMP\_0" application resource, we find two containers:

- "DESCRIPTOR" stores the lamp description.
- "DATA" stores the lamp data.



In the "DESCRIPTOR" container, there is one contentInstance containing "LAMP\_0" description.

#### d. Content Instances

Click on the contentInstance (cin\_554558050 in this example)

Attribute	Value
type	LAMP
location	Home
appld	LAMP_1
<input type="button" value="getState"/>	/mn-cse/mn-name/LAMP_1/DATA/la
<input type="button" value="getState(Direct)"/>	/mn-cse/mn-name/LAMP_1?op=getStateDirect&lampid=LAMP_1
<input type="button" value="switchON"/>	/mn-cse/mn-name/LAMP_1?op=setOn&lampid=LAMP_1
<input type="button" value="switchOFF"/>	/mn-cse/mn-name/LAMP_1?op=setOff&lampid=LAMP_1
<input type="button" value="toggle"/>	/mn-cse/mn-name/LAMP_1?op=toggle&lampid=LAMP_1

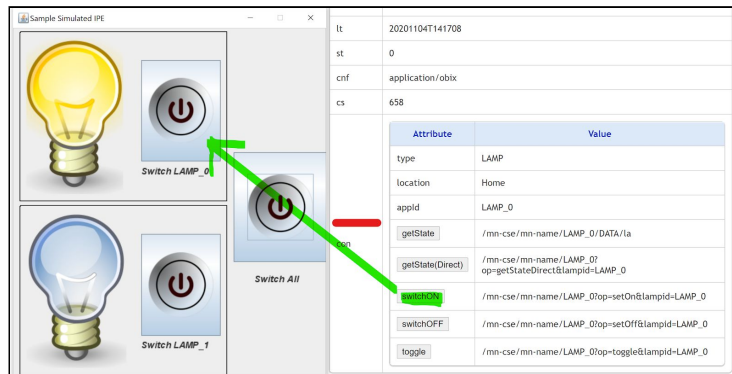
The "getState" button allows us to read the latest lamp state value stored in the GSCL database. Here, the value is "true".

Successful GET Request:	
Name	Value
type	LAMP
location	Home
lampId	LAMP_0
state	true

The "getState(Direct)" button allows us to read the lamp state directly from the lamp. For this manip, we turned off the light compared to the previous one. We can notice the difference of state, which is now "false".

Successful POST request.	
Name	Value
type	LAMP
location	Home
lampId	LAMP_0
state	false

The "switchON" allows us to switch on the lamp. We can use the "switchOFF" button to switch off the lamp, and "toggle" button to change its state.

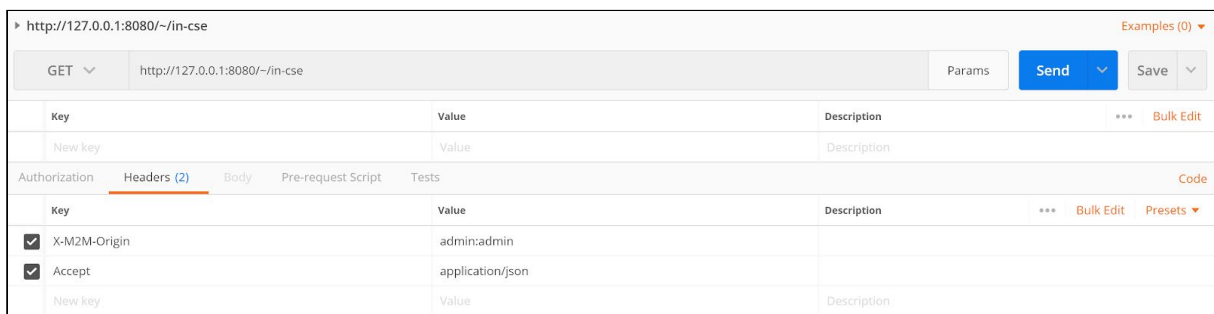


### 3. Interact with the REST API

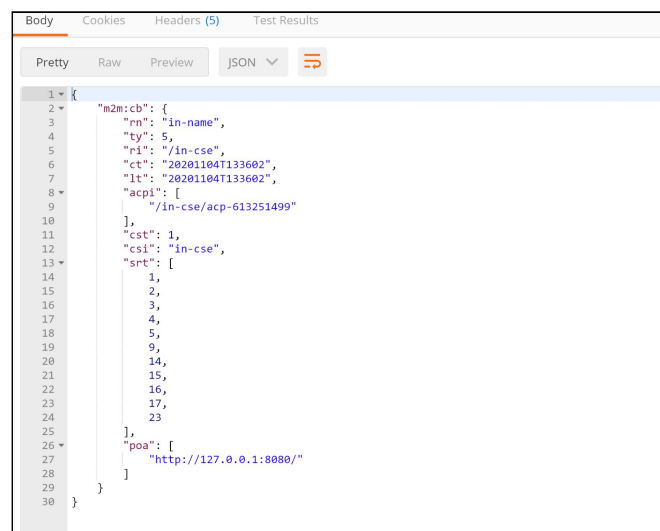
For this section, we used the REST client Postman, which provides a good interface to build HTTP requests.

#### a. Retrieve a resource

In order to retrieve a resource, we use the http request “GET” below:

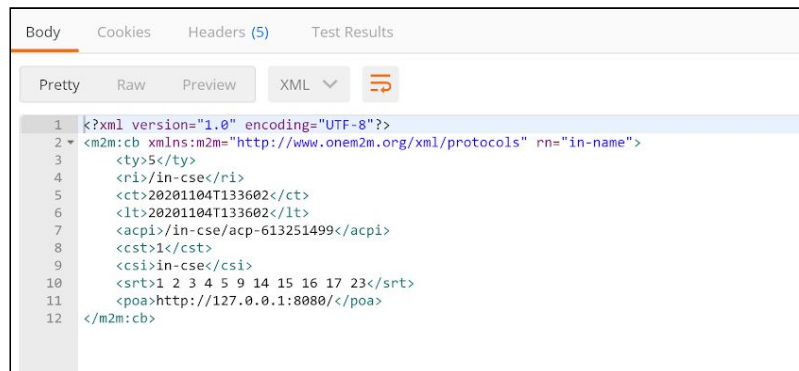


And we get the http response as follows in a JSON format:





Or we can also have it in an XML format:

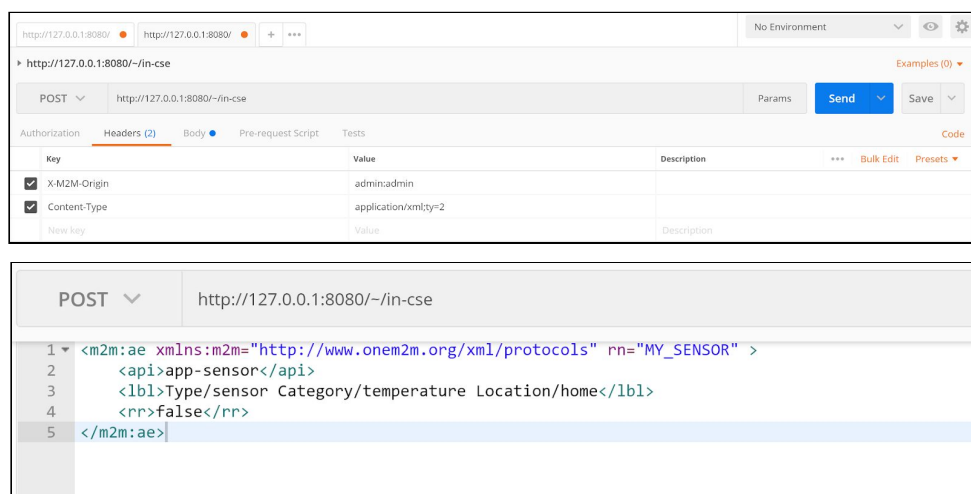


The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (5), and Test Results. The Body tab is active, and the request is displayed in XML format. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <m2m:cb xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="in-name">
3   <ty>5</ty>
4   <ri>/in-cse</ri>
5   <ct>20201104T133602</ct>
6   <lt>20201104T133602</lt>
7   <acpi>/in-cse/acp-613251499</acpi>
8   <cst>1</cst>
9   <csi>/in-cse</csi>
10  <srt>1 2 3 4 5 9 14 15 16 17 23</srt>
11  <poa>http://127.0.0.1:8080/</poa>
12 </m2m:cb>
```

## b. Create a "MY\_SENSOR" application

We send the following http POST request to create a MY\_SENSOR application on the gateway:



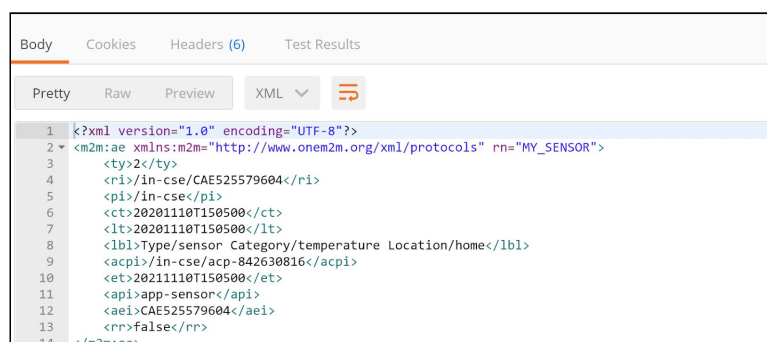
The screenshot shows a REST client interface with tabs for Authorization, Headers (2), Body, Pre-request Script, and Tests. The Headers tab is active, showing the following headers:

Key	Value	Description
X-M2M-Origin	admin:admin	
Content-Type	application/xml;ty=2	

Below the headers, the Body tab is active, showing the XML request body:

```
1 <m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="MY_SENSOR" >
2   <api>app-sensor</api>
3   <lbl>Type/sensor Category/temperature Location/home</lbl>
4   <rr>false</rr>
5 </m2m:ae>
```

We have the following http response:



The screenshot shows a REST client interface with tabs for Body, Cookies, Headers (6), and Test Results. The Body tab is active, and the response is displayed in XML format. The XML content is as follows:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <m2m:ae xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="MY_SENSOR">
3   <ty>2</ty>
4   <ri>/in-cse/CAE525579604</ri>
5   <pi>/in-cse</pi>
6   <ct>20201107T150500</ct>
7   <lt>20201107T150500</lt>
8   <lbl>Type/sensor Category/temperature Location/home</lbl>
9   <acpi>/in-cse/acp-842630816</acpi>
10  <et>20211107T150500</et>
11  <api>app-sensor</api>
12  <aei>CAE525579604</aei>
13  <rr>false</rr>
14 </m2m:ae>
```

We can check on the web service the creation of MY\_SENSOR application

Logout

OM2M CSE Resource Tree  
<http://127.0.0.1:8080/~in-cse/CAE525579604>  
- in-name  
  - acp\_admin  
  - acpae-525579604  
  - MY\_SENSOR  
  - mn-cse

Attribute

Value

ty	2
ri	/in-cse/CAE525579604
pi	/in-cse
ct	20201110T150500
lt	20201110T150500
lbl	Type/sensor Category/temperature Location/home
acpi	<div>AccessControlPolicyIDs /in-cse/acp-842630816</div>
et	20211110T150500
apl	app-sensor
ael	CAE525579604
rr	false

### c. Discover resources based on their labels

We can discover available resources based on their search strings using the discovery resource. The GET http request and response are as follows:

GET <http://127.0.0.1:8080/~in-cse?fu=1&lbl=Type/sensor>

Key	Value
<input checked="" type="checkbox"/> X-M2M-Origin	admin:admin
<input checked="" type="checkbox"/> Accept	application/xml
New key	Value

Body Cookies Headers (5) Test Results

Pretty Raw Preview XML

1

<?xml version="1.0" encoding="UTF-8"?>

2

<m2m:url xmlns:m2m="http://www.onem2m.org/xml/protocols">/in-cse/in-name/MY\_SENSOR</m2m:url>

### d. Create a DESCRIPTOR container

We send a POST HTTP request with the following parameters to create a DESCRIPTOR container resource under the MY\_SENSOR application:

POST
http://127.0.0.1:8080/~in-cse/in-name/MY\_SENSOR

Authorization
Headers (2)
Body
Pre-request Script
Tests

Key	Value
<input checked="" type="checkbox"/> X-M2M-Origin	admin:admin
<input checked="" type="checkbox"/> Content-Type	application/xml;ty=3
New key	Value

POST
http://127.0.0.1:8080/~in-cse/in-name/MY\_SENSOR

Authorization
Headers (2)
Body
Pre-request Script
Tests

form-data
x-www-form-urlencoded
raw
binary
XML (application/xml)

```

1 <m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="DESCRIPTOR">
2 </m2m:cnt>

```

And we have the following response:

Body
Cookies
Headers (6)
Test Results

Pretty
Raw
Preview
XML

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="DESCRIPTOR">
3   <ty>3</ty>
4   <ri>/in-cse/cnt-929434318</ri>
5   <pi>/in-cse/CAE525579604</pi>
6   <ct>20201119T182000</ct>
7   <lt>20201119T182000</lt>
8   <acpi>/in-cse/acp-842630816</acpi>
9   <et>20211119T182000</et>
10  <st>0</st>
11  <mni>10</mni>
12  <mbs>10000</mbs>
13  <mia>0</mia>
14  <cni>0</cni>
15  <cbs>0</cbs>
16  <ol>/in-cse/in-name/MY_SENSOR/DESCRIPTOR/ol</ol>
17  <la>/in-cse/in-name/MY_SENSOR/DESCRIPTOR/la</la>
18 </m2m:cnt>

```

#### e. Create a description contentInstance

We send a HTTP request with the following parameters to create a description content instance resource under the "DESCRIPTOR" container:

POST ▼ http://127.0.0.1:8080/~/-in-cse/in-name/MY\_SENSOR/DESCRIPTOR

Authorization Headers (2) Body ● Pre-request Script Tests

Key	Value
<input checked="" type="checkbox"/> X-M2M-Origin	admin:admin
<input checked="" type="checkbox"/> Content-Type	application/xml;ty=4
<input type="text" value="New key"/>	<input type="text" value="Value"/>

POST ▼ http://127.0.0.1:8080/~/-in-cse/in-name/MY\_SENSOR/DESCRIPTOR

Authorization Headers (2) Body ● Pre-request Script Tests

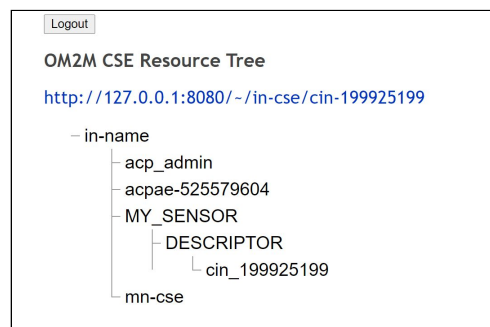
☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary XML (application/xml) ▼

```

1 <m2m:cin xmlns:m2m="http://www.onem2m.org/xml/protocols">
2   <cnf>application/xml</cnf>
3   <con>
4     <obj>
5       <str name="type" val="Temperature_Sensor"/>
6       <str name="location" val="Home"/>
7       <str name="appId" val="MY_SENSOR"/>
8       <op name="getValue" href="/in-cse/in-name/MY_SENSOR/DATA/la"
9         in="&obix:Nil" out="&obix:Nil" is="retrieve"/>
10      </obj>
11    </con>
12  </m2m:cin>

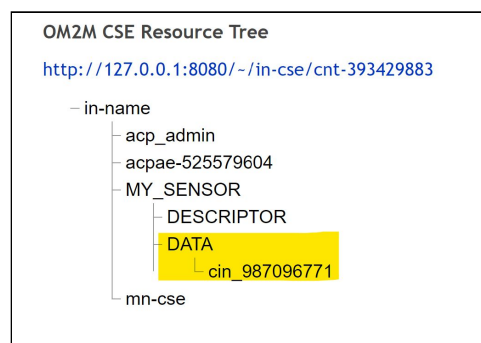
```

We can check the created content instance on the web interface:



#### f. Create a DATA container and contentInstance

Following the exact same scheme of POST http request, we create a DATA container and contentInstance. We can check their creation on the web interface:



#### g. Create a "MY\_SENSOR" application using JSON

Since OM2M supports JSON, we can send an HTTP request with the following parameters to create a MY\_SENSOR application on the gateway using JSON instead of XML. Here is the POST request:

The first screenshot shows the 'Headers' tab of an HTTP client. The method is 'POST' and the URL is 'http://127.0.0.1:8080/~in-cse'. Two headers are defined: 'X-M2M-Origin' with value 'admin:admin' and 'Content-Type' with value 'application/json;ty=2'.

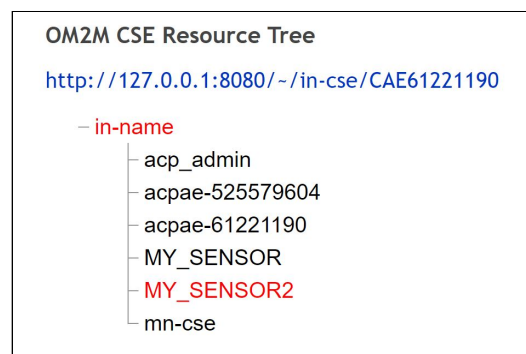
The second screenshot shows the 'Body' tab. The content type is set to 'JSON (application/json)'. The body contains a JSON object:

```

1 {
2   "m2m:ae": {
3     "api": "app-sensor",
4     "rn": "false",
5     "lbl": ["Type/sensor", "Category/temperature", "Location/home"],
6     "rn": "MY_SENSOR2"
7   }
8 }

```

We can check on the web interface the creation of MY\_SENSOR2:



## h. Subscribe to MY\_SENSOR data

To start the monitor, we got an unexpected error that we did not manage to solve, so we could not work on this part.

```

C:\Users\emili\Downloads>java -jar monitor.jar
Starting server..
Exception in thread "main" java.net.BindException: Address already in use: bind
    at sun.nio.ch.Net.bind0(Native Method)
    at sun.nio.ch.Net.bind(Unknown Source)
    at sun.nio.ch.Net.bind(Unknown Source)
    at sun.nio.ch.ServerSocketChannelImpl.bind(Unknown Source)
    at sun.nio.ch.ServerSocketAdaptor.bind(Unknown Source)
    at sun.net.httpserver.ServerImpl.<init>(Unknown Source)
    at sun.net.httpserver.HttpServerImpl.<init>(Unknown Source)
    at sun.net.httpserver.DefaultHttpServerProvider.createHttpServer(Unknown Source)
    at com.sun.net.httpserver.HttpServer.create(Unknown Source)
    at httpapplication.Monitor.main(Monitor.java:48)

```

#### 4. Access Control Management in oneM2M

To manage the access of oneM2M resource, a specific resource exists that represents the rights given to an entity: AccessControlPolicy (ACP). Each other oneM2M resource is linked to a set of ACP via the *acpi* attribute. This resource has 2 specific attributes: privileges and self-privileges.

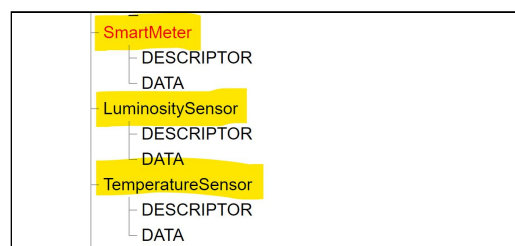
The operations allowed by the originator in the Access Control Rule is defined by an integer. This integer is the sum of the Operation value. The following table shows the available operations and their corresponding value.

Access Control Operation	Value
CREATE	1
RETRIEVE	2
UPDATE	4
DELETE	8
NOTIFY	16
DISCOVERY	32

The aim now is to create an ACP and attach it to an already created data container from the previous exercise. The aim is to give access to 2 different entities:

- a monitoring application that will only retrieve the data
- a sensor application that will be able to retrieve AND create

First we create 3 AE named SmartMeter, LuminositySensor and TemperatureSensor on the MN. Each application representing each device has 2 containers named DESCRIPTOR and DATA. Then we create the content instances in each one of these containers.



Once again because the monitor was not working, we could not go any further.

## General summary

In the first lab, we learned about **MQTT** (for Message Queuing Telemetry Transport) which is a protocol adapted to connections that integrate a mobility part between the clients and the server that will store the data. It works through a "**broker**" that centralizes the connections between the different entities and allows them to **subscribe** and/or **publish** information via "topics". This operating mode allows to have changing connections or variable IP addresses, such as sensors connected via mobile connections. The same applies to servers. It is also possible to secure a connection by using a username and password to connect to the broker or to exchange data via a security protocol (SSL).

In the second lab, we saw **an open-source solution for IoT which remains company proprietary**. On the hardware side, it includes a cellular modem for wirelessly connecting IoT applications over a mobile network, ARM-based application processor with GNSS receiver, built-in sensors including Accelerometer, and Gyroscope and can be powered by a battery for low-power wireless applications. On the software side, it is pre-integrated with the open source Legato Linux platform for application-level development, has robust connectivity APIs to access cloud and network services such as voice calls, SMS, data.

In the third lab, we learned about **other service and application level solutions to connect IoT devices without taking into account the communication technology deployed** by the objects thanks to **OM2M**, a standard for IoT middleware to make plug n play. We were also able to interact with OM2M through REST queries through the Postman software. We could check in real time the proper functioning of the requests and creation of resources thanks to the OM2M web interface. We also learned the difference between IN-CSE applications, which are mostly used for business applications, data visualization applications (software applications) and the MN-CSE application in which the AEs represent the equipment installed in a LAN installed in the MN (hardware applications).

In the fourth lab, we learned how to **deploy a concrete architecture with heterogeneous devices**, deploy several oneM2M nodes (IN, MN) as well as MQTT nodes, interconnect heterogeneous devices at the application level and finally develop a high level application thanks to Node-Red. The dashboard on Node-Red is very useful to give a very concrete meaning to the created application.

Finally, we learned **how to position the Internet of Things in terms of concept, areas of application and potential**. We now have in mind the panorama of the main standards, whether in terms of sensor networks or fields of application. This UF was an opportunity for us to completely discover the world of IoT, and we feel that we have had a complete overview of it. We encountered some **problems with the software** to use (Mosquitto, Node-Red, Postman, monitor...) which slowed us down a bit in our work. But overall we were able to progress at a good pace thanks to the help we received during the practical sessions.

Criteria	Coverage of the standard	Deployment Model	Data Model	Hardware platform and language	Security	Example
OneM2M	Service: discovery, communication, management group, data, management of equipment, billing, network	Based on 3 layers : application, service and network. Deployment on objects, the gateways and the cloud	Based on a REST architecture and a resource tree	Non-specific material. Different platforms including open source based on the Java language but possible interaction with different languages: Java, Python, C++	Authentication, authorization and encryption management of communications	OM2M, IoTDM ; mobius, pilot things
MQTT	Management and storage of data exchanged between objects	An MQTT message server and clients sending or subscribing to this server	Topic-based with the ability to manage hierarchical topics	Non specific hardware, clients and servers exist using different languages: Lua, Java	Ability to manage authentication and encryption of communication	Mosquito, EMQ, MQTT
CoAP	Communication protocol for constrained equipment	Client server model	Information only on addresses and communication ports	Non-specific hardware, coding possible in many languages: Java, C, JavaScript, Ruby	Communication encryption possible with DTLS	Californium, Erbium, Copper