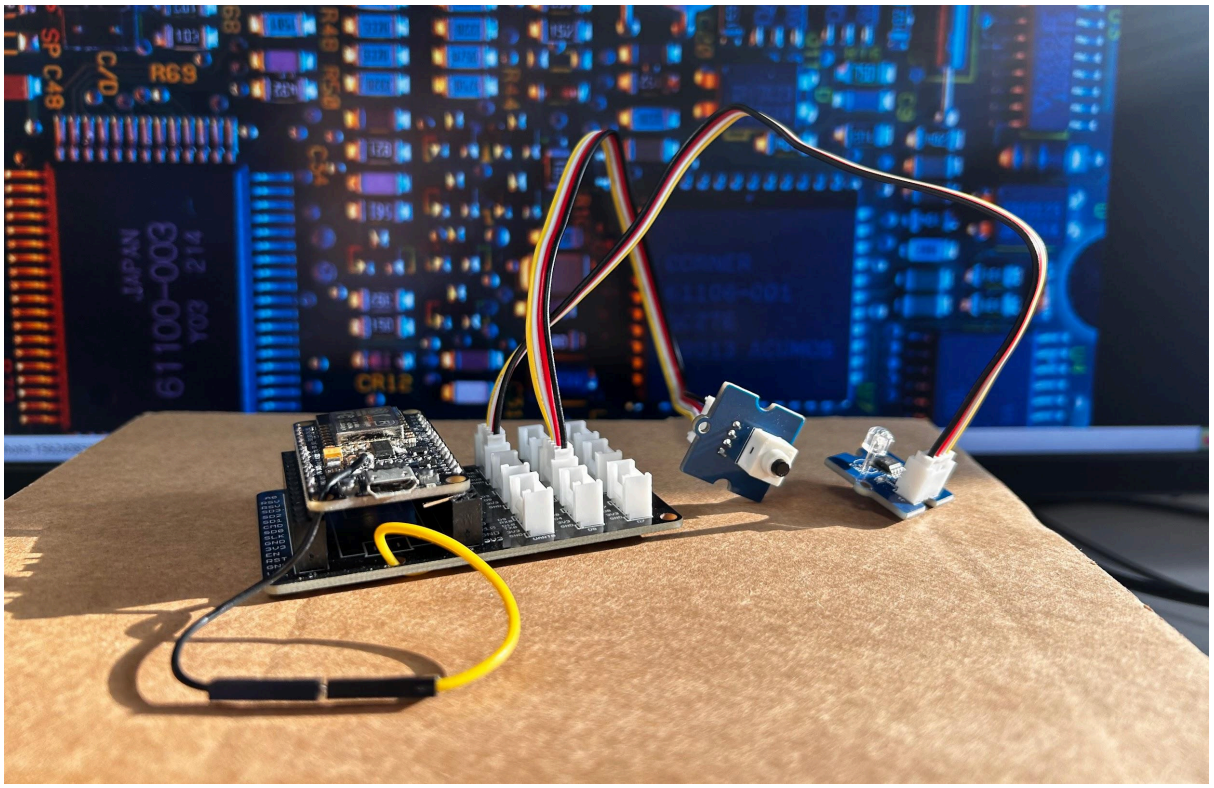
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Middleware for IoT



Buttow-Albuquerque Joao Gabriel
Azzabi Fatine

5 ISS

09/12/2025

Table of contents

LAB 1 & 2: Middleware For the IoT

Introduction

The objective of this laboratory session is to gain a deeper understanding of the MQTT protocol and its relevance within the Internet of Things (IoT) ecosystem. We begin by examining the core principles and characteristics that define MQTT as a lightweight communication protocol. Following this theoretical overview, we proceed with the installation of the necessary software tools on our laptop to experiment with MQTT in practice. The lab then culminates in the development of a simple IoT application using an ESP8266 device, leveraging MQTT to establish communication with a locally hosted Eclipse Mosquitto server.

1. MQTT

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

The typical architecture of an IoT system using MQTT relies on a publish/subscribe communication model facilitated by an MQTT broker.
According to the course (slides 85–87), devices act as MQTT clients that either publish messages to specific topics or subscribe to topics to receive data.
The MQTT broker operates as the central mediator in this architecture: it manages client sessions, maintains topic hierarchies, and forwards published messages to all subscribed devices.
This architecture creates a loosely coupled, asynchronous environment in which devices do not need to interact directly but exchange information through the broker.

2. What is the IP protocol under MQTT? What does it mean in terms of bandwidth usage and communication type?

MQTT operates on top of the TCP/IP protocol suite, as clearly stated in the course (slide 84). Using TCP ensures reliable and ordered message delivery, which is well suited for IoT applications requiring consistent communication.
Furthermore, MQTT is intentionally lightweight: the fixed header size is only 2 bytes (slide 84), allowing for extremely low bandwidth usage and making the protocol ideal for constrained networks and low-power devices.
Its publish/subscribe model enables asynchronous communication between nodes, reducing unnecessary traffic and improving overall system efficiency.

3. What are the different versions of MQTT?

The course identifies multiple versions of MQTT, including MQTT 3.1.1 and MQTT 5.0 (slide 84). These versions introduce various improvements, with version 5 offering advanced functionalities such as better error reporting and rich metadata support, while still maintaining compatibility with the lightweight nature of the protocol.

4. What kind of security/authentication/encryption is used in MQTT?

MQTT supports several mechanisms to ensure secure and authenticated communication. The course notes that secure communication can be added to MQTT through encryption layers such as TLS (slide 84). In addition to encryption, MQTT commonly uses username/password authentication at the client side.
The protocol also incorporates features like the Will message (slide 88), which allows the broker to notify subscribers if a client disconnects unexpectedly.
These combined features help prevent unauthorized access while maintaining the lightweight structure of the protocol.

5. Smart-home system: required topics and publish/subscribe connections

To implement a smart-home system with one button, one light, and one luminosity sensor, as described in the TP instructions (section 2), MQTT topics must be designed to support both manual and automatic light control.
The topics necessary for this behavior are:

- ➢ home/button/state
  The button publishes its current state (pressed or not pressed).
  - Publisher: Button
  - Subscriber: Automation logic or controller
- ➢ home/luminosity/value
  The luminosity sensor publishes real-time light intensity measurements.
  - Publisher: Luminosity sensor
  - Subscriber: Automation controller
- ➢ home/light/command
  This topic carries ON/OFF commands directed to the light.
  - Publisher: Controller (manual or automatic)
  - Subscriber: Light device

Connections:

- The button publishes to home/button/state, enabling manual control of the light.
- The luminosity sensor publishes to home/luminosity/value, enabling automated control when values fall below a threshold.

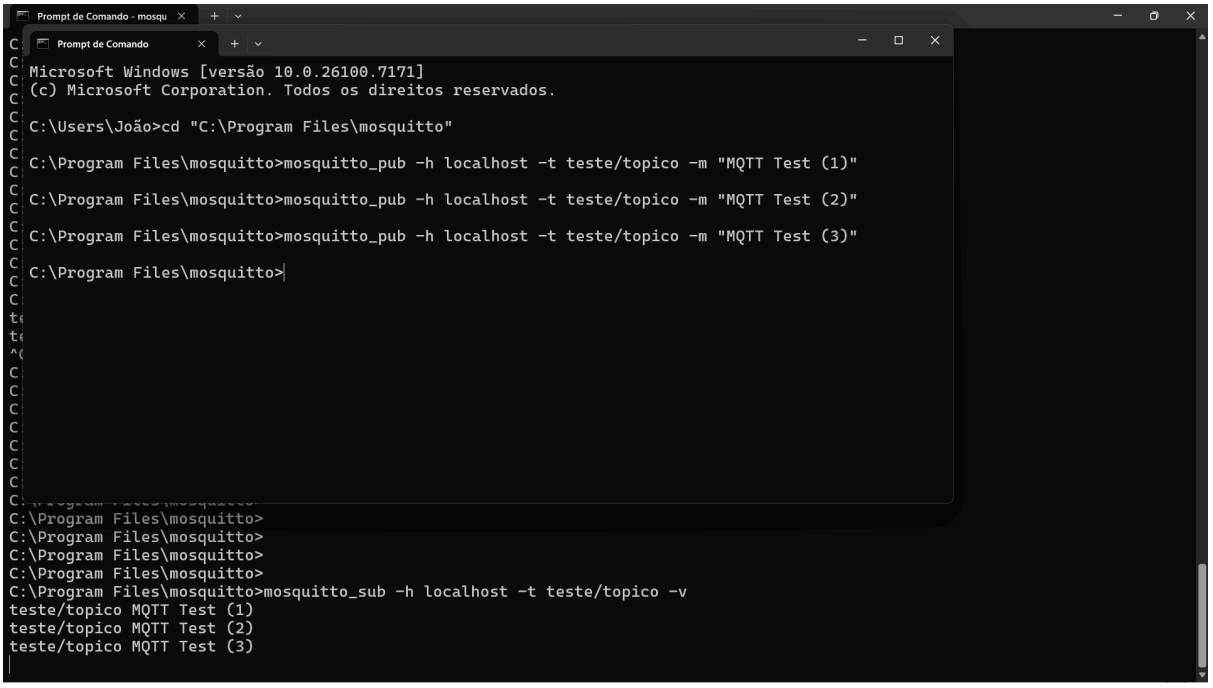
- Based on both inputs, the controller publishes ON/OFF commands to home/light/command.

- The light subscribes to this topic and adjusts its state accordingly.

This setup ensures that manual and automated behaviors coexist seamlessly, reflecting the principles of the MQTT model described in the course (slides 84–88)

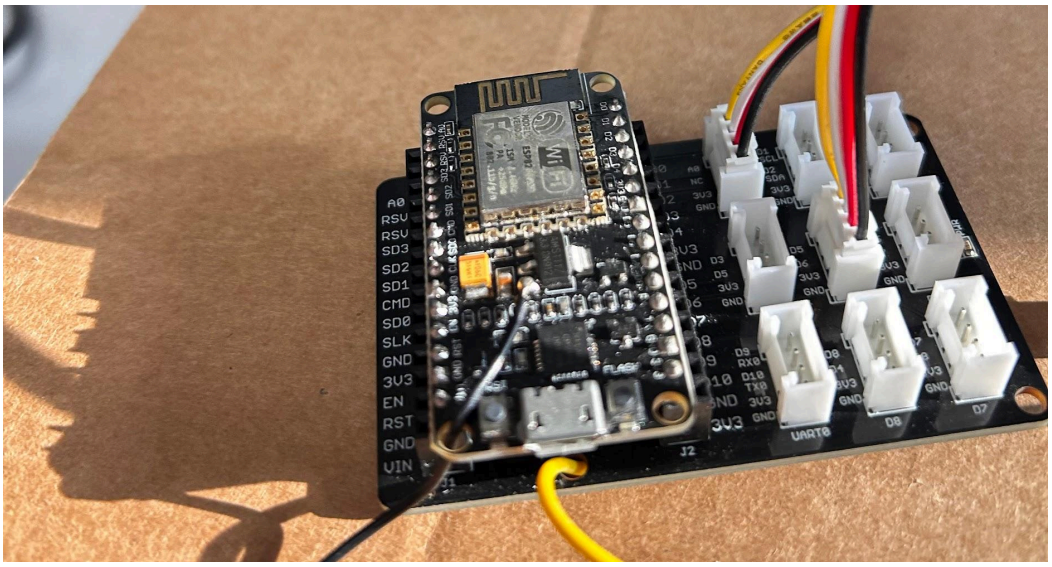
2. Install and test the broker

For this TP, we will use the mosquitto broker made by the eclipse opensource foundation (https://mosquitto.org). Mosquitto exists for main platforms: linux, windows, macOS. We downloaded and installed this broker on our laptop.
• We run the mosquito broker mosquitto (or start the service)
The service starts automatically in Windows.
• Test publish and subscribe with the command shell programs: mosquitto_pub and mosquitto_sub



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

During the TP, we will use the nodeMCU board based on ESP8266.



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

  ➔ Communication :
  The NodeMCU board supports Wi-Fi communication, allowing it to connect easily to wireless networks. It also includes a UART interface, which is used for serial communication and debugging.

  ➔ Programming Language :
  The board can be programmed using the Lua scripting language. It is also compatible with C/C++ through the Arduino IDE, making development accessible and flexible.

  ➔ Inputs/Outputs (I/O) :
  The NodeMCU provides several GPIO pins that can be used for digital input and output operations.It includes one analog input (ADC) as well as PWM support, enabling control of actuators. The board also offers I²C and SPI communication interfaces for connecting sensors and external modules.

  ➔ Other Features :
  The board is equipped with onboard flash memory used for storing programs. It includes a USB interface that provides both power and programming capabilities. Finally, it comes with the NodeMCU firmware already installed, allowing quick and easy prototyping.

- Find an example code for ESP8266 WiFi client with MQTT and have a look at the different parts of the code. Explain those different parts

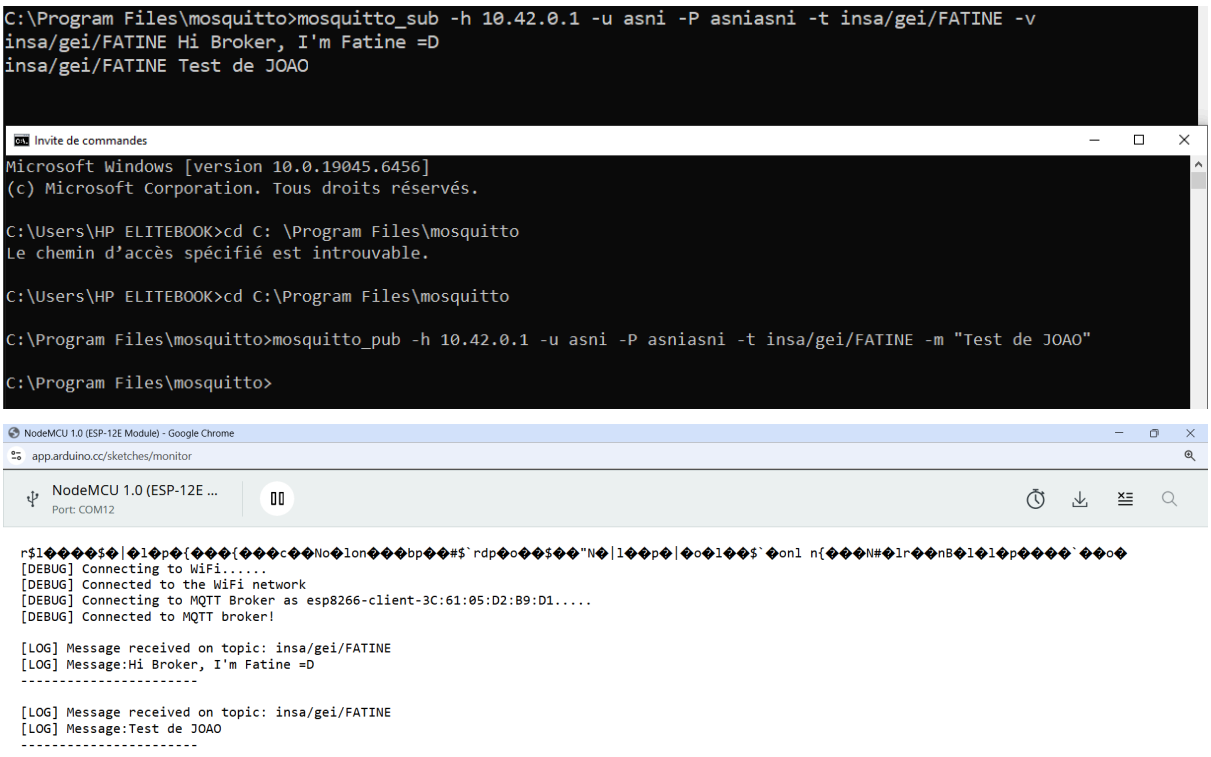
[Using MQTT on ESP8266: A Quick Start Guide | EMQ]

The code shows how the ESP8266 connects to WiFi, then to an MQTT broker, and finally sends and receives messages. Each section of the code has a clear role. In our case, we used this example as a base, but we modified the configuration part by replacing the WiFi credentials, the MQTT broker address, the topic, and the authentication information with the settings given by our professor.
Before that, we tried using our own shared connection from a smartphone, but we were not able to send MQTT messages between the devices. For this reason, we switched to the configuration given by the professor.

- Open the serial monitor on Arduino IDE and run your Arduino code, using the shell command to validate that your device publishes values and can receive the result of subscription.

In our case, we used three windows: the subscriber, the publisher, and the Arduino serial console. The first step was to open the subscriber in order to view all incoming messages. Then, we started the ESP8266 connection. The device successfully connected to the WiFi network and to the MQTT broker. It was able to publish a message, which appeared both in the Arduino console and in the subscriber window.

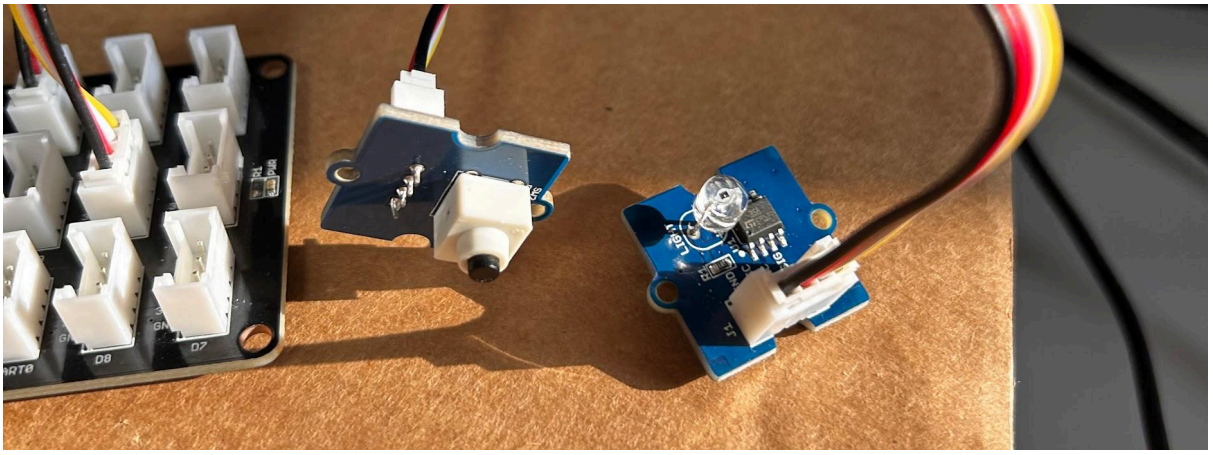
After that, we opened a publisher on the computer to test the communication between the ESP8266 and the machine. We sent the message “test de JOAO,” and we observed that the Arduino received it correctly. This confirms that the communication is properly established and that the device can both send and receive MQTT messages.



4. Creation of a simple application

- **By using what you did on MQTT and the necessary sensors and actuators, program the application's light management behavior through MQTT exchanges. Button publish state and light subscribe to button state.**

For this simple project, we built a small MQTT-based application using an ESP8266, a Grove light sensor, and a Grove push button. Shown in the following picture:



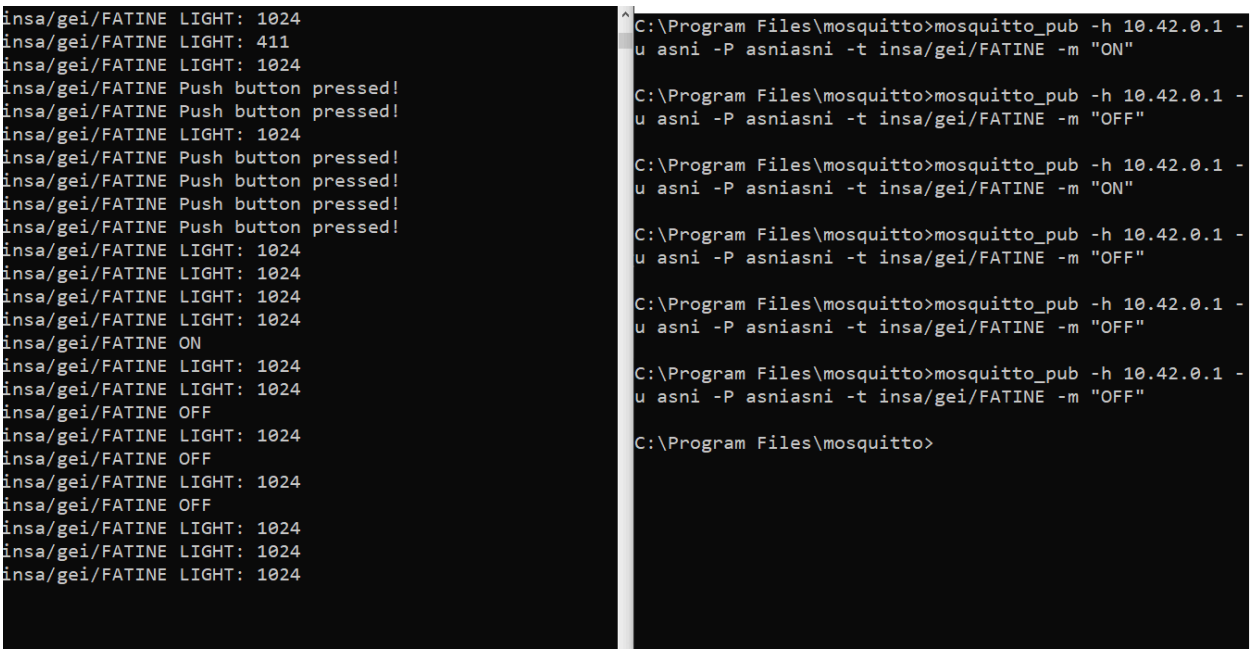
To implement this project, we took the code made in the previous section and other base codes from the internet to integrate the devices that we had in hand. The example codes that we have taken as example are the following:

Grove - Light Sensor | Seeed Studio Wiki
Grove - Button | Seeed Studio Wiki

We also improved the code so that it can read the push button while a timer runs in the background to decide when to send the periodic light sensor data. In addition, we added a new section to turn the built-in LED ON and OFF when the device receives the corresponding MQTT messages. The complete code is available in the annexes.

The ESP8266 is configured to periodically read the light level and publish the measured value via MQTT every two seconds. These readings appear on the MQTT subscriber running on our computer, allowing us to continuously monitor the ambient luminosity in real time.

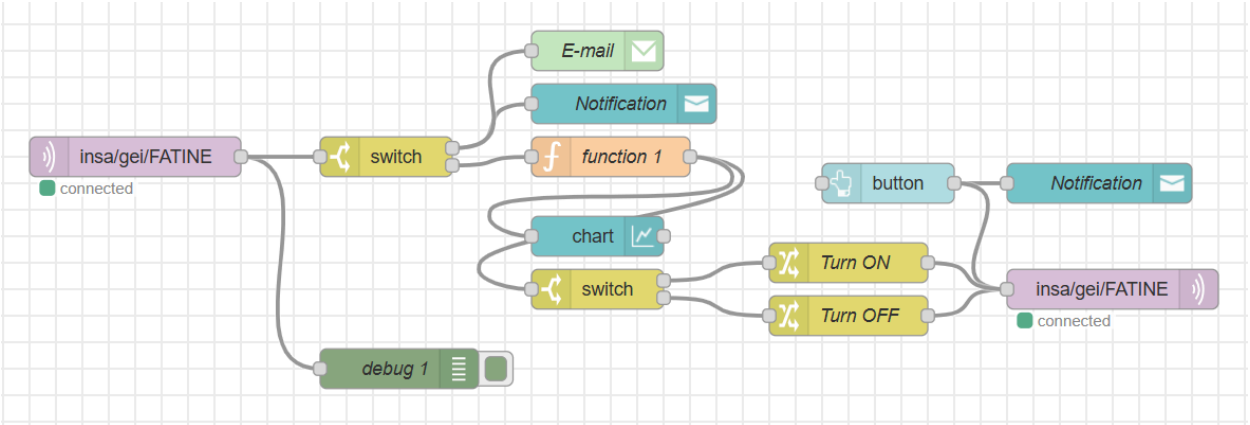
In addition to the periodic sensor data, the device also reacts to user interaction through the push button. Whenever the button is pressed (on a change of state), the ESP8266 publishes the message "Push button pressed!" on the same MQTT topic, which we can immediately see on the subscriber side. Furthermore, by sending the commands "ON" or "OFF" via MQTT, we can remotely turn the ESP8266's built-in LED on or off. We can see the publisher (right) and subscriber (left) in the following screenshot :



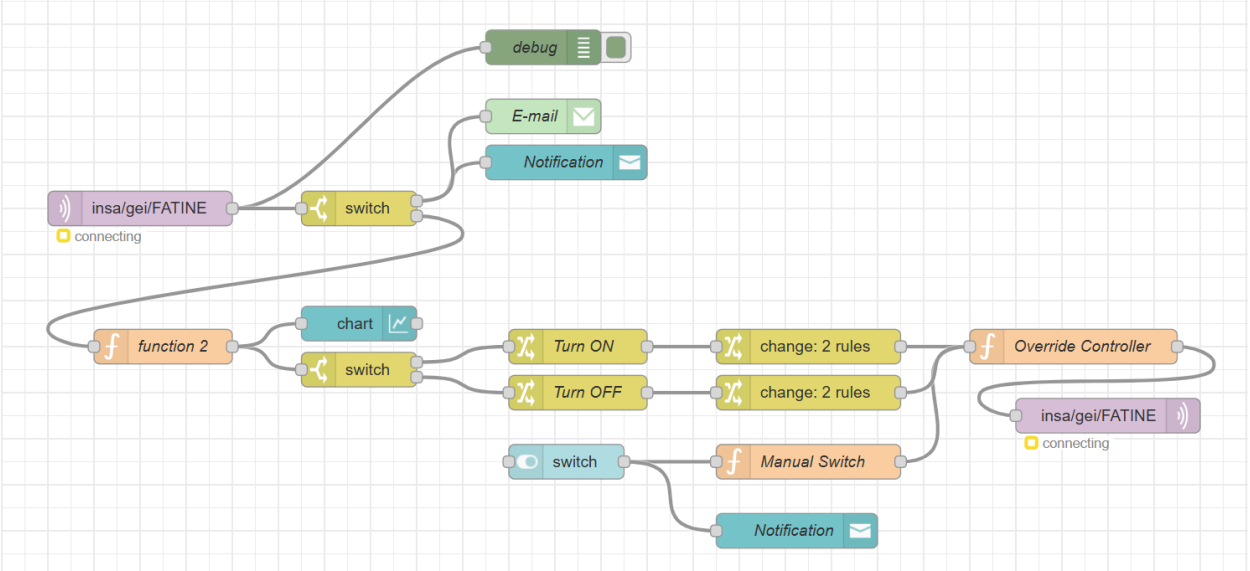
Overall, this prototype validates the complete MQTT workflow: the ESP8266 is able to publish both periodic sensor readings and event-based messages, while also subscribing to control commands to drive local actuators (the built-in LED). This confirms that our setup can reliably support basic IoT interactions and can be easily extended to more complex applications with additional sensors and actuators.

Lab 3: Fast application prototyping for IoT

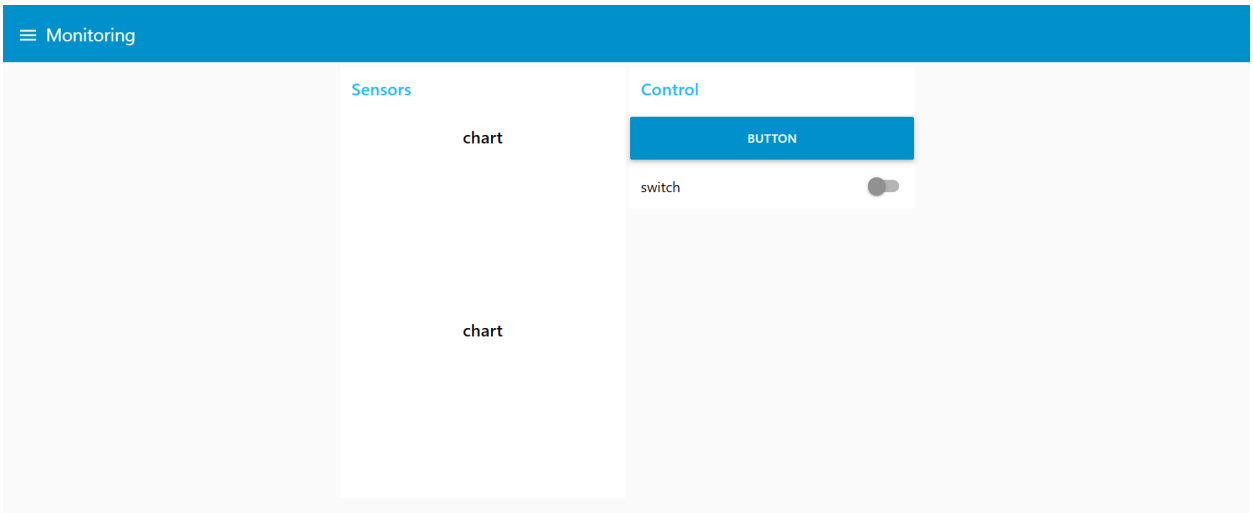
Flow 1 - Simple



Flow 2 - With rules



Final Dashboard



Benefits and drawbacks of building an application with Node-RED

Node-RED offers several benefits for developing IoT applications. One of its main advantages is its visual programming interface, which allows users to create applications quickly by connecting nodes without writing much code. This makes Node-RED easy to learn and well suited for rapid prototyping. It also supports many protocols and services such as MQTT, HTTP, dashboards, and external APIs, which simplifies the integration of heterogeneous IoT devices. In addition, Node-RED allows real-time data processing and visualization through dashboards.

However, Node-RED also has some drawbacks. For large or complex applications, flows can become difficult to read and maintain. Scalability is limited compared to traditional backend frameworks, especially for high-performance or industrial-scale systems. Moreover, since the logic is mainly graphical and stored in JSON files, version control and code review can be less convenient than with classical programming languages.

Overall, Node-RED is an excellent tool for fast development and experimentation, but it is less suitable for very large-scale or highly complex applications.

Using Node-RED, we developed and tested an application that communicates with our device via MQTT, building on the implementation from TPs 1 and 2. We also created a user-friendly Node-RED dashboard to monitor and interact with the device. The Node-RED flow JSON for this project is included in the annexes.

Lab 4 : Specialized middleware – Home Assistant + ESP8266 (ArduinoHA)

Goal of the lab

The objective of this lab is to discover Home Assistant as a specialized IoT middleware and to integrate an ESP8266 (NodeMCU) device through MQTT using the ArduinoHA library. We also create basic automations and explore Node-RED for more advanced scenarios.

Architecture: Role of MQTT and why not HTTP

In the proposed architecture, MQTT acts as the communication middleware between the ESP8266 devices and Home Assistant. The ESP8266 boards behave as MQTT clients, while Home Assistant integrates an MQTT broker (Mosquitto) that centralizes all message exchanges.

MQTT is based on a publish/subscribe communication model. Devices publish sensor data or events to topics and subscribe to topics to receive commands. This approach enables asynchronous communication and avoids direct coupling between devices.

HTTP is not well suited for this type of IoT architecture because it relies on a request/response model and requires frequent connections. MQTT is lightweight, uses very small message headers, and is optimized for low-bandwidth and low-power environments. These characteristics make MQTT more efficient and scalable for IoT systems.

Security: Protecting communication between ESP8266 and Home Assistant

Several security mechanisms can be implemented to protect the communication between the ESP8266 and Home Assistant.

First, MQTT supports authentication using a username and password, which prevents unauthorized devices from connecting to the broker. This mechanism was used in our setup.

Second, encryption can be added by using MQTT over TLS (MQTTS), ensuring that data exchanged between the devices and the broker is encrypted and protected against interception.

Finally, access control can be reinforced by limiting topic permissions and by isolating IoT devices on a dedicated network. These measures help improve the overall security of the system.

Scalability: Managing multiple ESP8266 devices

To manage a large number of ESP8266 devices, a clear organization is required at both the middleware and embedded levels.

In Home Assistant, devices can be assigned to floors and rooms, which allows a structured spatial organization (for example: building, floor, and room). This makes supervision, monitoring, and automation much easier when many devices are deployed.

In addition, each ESP8266 must be uniquely identified in the firmware. In the embedded code, the device name and all associated entities (LEDs, sensors) are explicitly defined using device.setName() and entity names. Assigning specific and unique names to each device avoids confusion when multiple similar devices are connected.

By combining room management in Home Assistant with explicit device naming in the code, the system becomes scalable and remains easy to maintain, even with many ESP8266 devices.

Optimization: Reducing ESP8266 energy consumption

To reduce the energy consumption of the ESP8266, several optimizations can be applied.

The frequency of sensor measurements and MQTT message publications can be reduced to limit unnecessary communication. Using low QoS levels when possible also decreases communication overhead.

In addition, the ESP8266 provides low-power and deep sleep modes, which significantly reduce energy consumption between measurements. Optimizing WiFi connection time and disconnecting when communication is not required further improves battery life.

These techniques are essential for deploying ESP8266 devices in battery-powered IoT applications.

Extension: Real-world use case

This solution can be applied in the field of home automation (domotics). By combining ESP8266 devices with Home Assistant, it becomes possible to manage and control houses remotely.

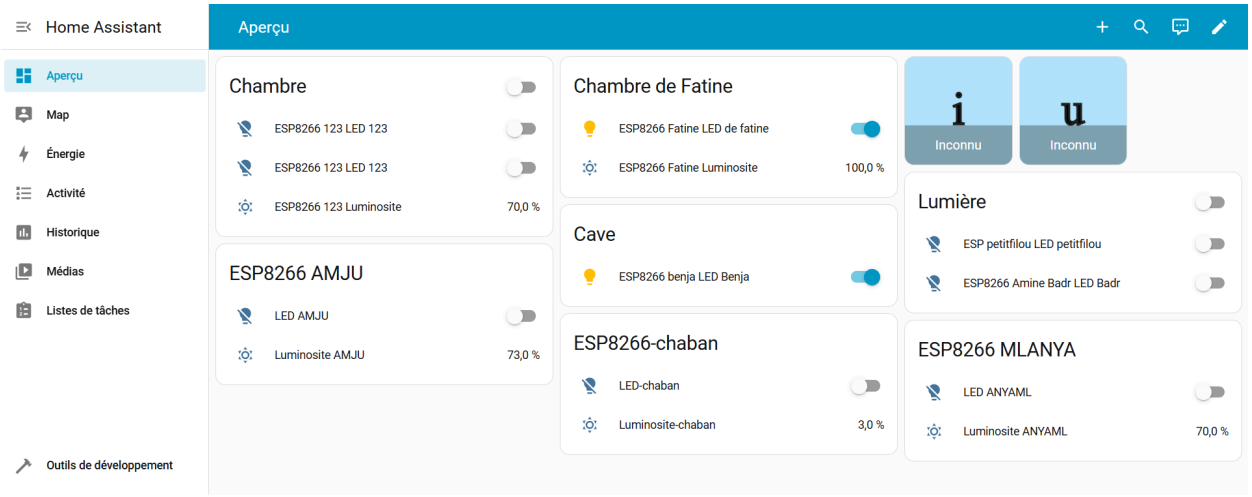
For example, lighting, sensors, and actuators can be supervised and controlled through dedicated platforms such as Kynix, enabling centralized management of smart homes.

Furthermore, this architecture can be integrated with voice assistants such as Amazon Alexa and Google Home Assistant, allowing users to control devices using voice commands. This improves user comfort and accessibility while keeping centralized and automated control of the home.

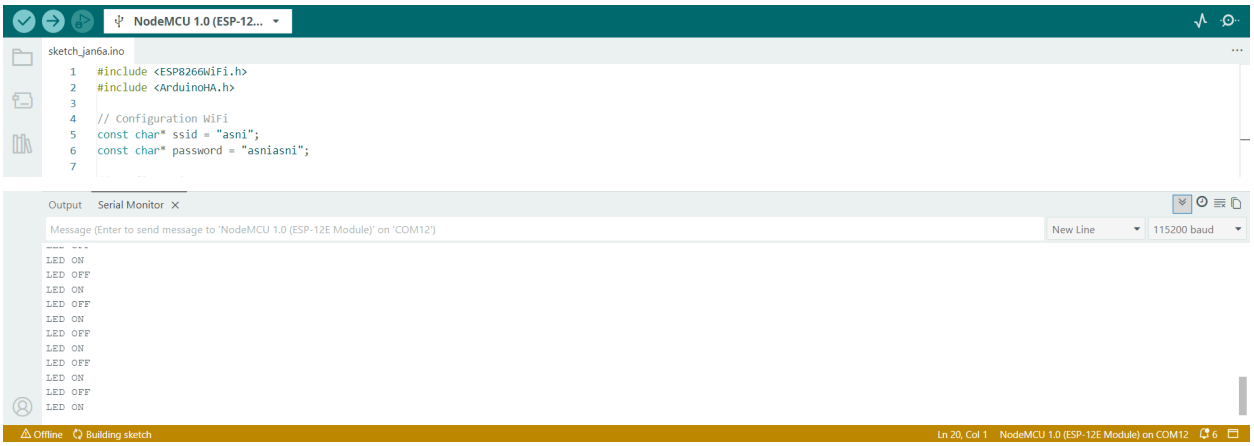
BONUS REPORT

Exercise: LED control from Home Assistant

As part of this exercise, the LED connected to the ESP8266 was successfully controlled directly from the Home Assistant interface. The LED entity appeared correctly in Home Assistant and was assigned to the room *"Chambre de Fatine"*, as shown in the following screenshot :



By using the ON/OFF controls in the Home Assistant dashboard, we were able to switch the LED on and off remotely. Each action performed from the interface was immediately reflected on the physical LED connected to the NodeMCU board. At the same time, the Arduino serial monitor displayed the corresponding messages (LED ON and LED OFF), confirming that the ESP8266 correctly received and processed the commands sent by Home Assistant through MQTT, as shown in the following screenshot :



This experiment validates the proper integration between Home Assistant and the ESP8266 using ArduinoHA, as well as the correct bidirectional communication between the middleware and the IoT device.

The full arduino code is available in the annexes.

ANNEXES

Code TP 1 & 2 - Arduino

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// WiFi settings
const char *ssid = "asni";          // Replace with your WiFi name
const char *password = "asniasni";  // Replace with your WiFi password

// MQTT Broker settings
const char *mqtt_broker = "10.42.0.1";       // EMQX broker endpoint
const char *mqtt_topic = "insa/gei/FATINE";  // MQTT topic
const char *mqtt_username = "asni";                // MQTT username for
authentication
const char *mqtt_password = "asniasni";          // MQTT password for
authentication
const int mqtt_port = 1883;                  // MQTT port (TCP)

const int BUTTON_PIN = D5;
bool lastButtonState = LOW;

const int LIGHT_PIN = A0;
unsigned long lastLightSend = 0;
const unsigned long MESSAGE_DELAY = 2000;  // Send delay of 2s

WiFiClient espClient;
PubSubClient mqtt_client(espClient);

void connectToWiFi();
void connectToMQTTBroker();
void mqttCallback(char *topic, byte *payload, unsigned int length);

void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(LED_BUILTIN, OUTPUT);
  digitalWrite(LED_BUILTIN, HIGH);
  Serial.begin(115200);
  connectToWiFi();
  mqtt_client.setServer(mqtt_broker, mqtt_port);
  mqtt_client.setCallback(mqttCallback);
  connectToMQTTBroker();
```

```
}
void connectToWiFi() {
  WiFi.begin(ssid, password);
  Serial.print("\n[DEBUG] Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\n[DEBUG] Connected to the WiFi network");
}

void connectToMQTTBroker() {
  while (!mqtt_client.connected()) {
    String client_id = "esp8266-client-" + String(WiFi.macAddress());
       Serial.printf("[DEBUG] Connecting to MQTT Broker as %s.....\n",
client_id.c_str());
            if   (mqtt_client.connect(client_id.c_str(),   mqtt_username,
mqtt_password)) {
      Serial.println("[DEBUG] Connected to MQTT broker!");
      mqtt_client.subscribe(mqtt_topic);
      // Publish message upon successful connection
      mqtt_client.publish(mqtt_topic, "Hi Broker, I'm Fatine =D");
    } else {
      Serial.print("[ERROR] Failed to connect to MQTT broker, rc=");
      Serial.print(mqtt_client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void mqttCallback(char *topic, byte *payload, unsigned int length) {
  String msg;

  Serial.print("Message received on topic: ");
  Serial.println(topic);
  Serial.print("Message:");

  for (unsigned int i = 0; i < length; i++) {
    char c = (char)payload[i];
    Serial.print(c);
    msg += c;
  }
```

```
  Serial.println("\n");
  if (msg == "ON") {
    Serial.println("[LOG] Received ON -> turning ON LED_BUILTIN\n");
    digitalWrite(LED_BUILTIN, LOW);
  } else if (msg == "OFF") {
    Serial.println("[LOG] Received OFF -> turning OFF LED_BUILTIN\n");
    digitalWrite(LED_BUILTIN, HIGH);
  }
}

void loop() {
  if (!mqtt_client.connected()) {
    connectToMQTTBroker();
  }
  mqtt_client.loop();

  bool currentButtonState = digitalRead(BUTTON_PIN);

  if (lastButtonState == HIGH && currentButtonState == LOW) {
    Serial.println("[LOG] Button pressed!");
    Serial.println("[LOG] Sending MQTT button message...\n");
    mqtt_client.publish(mqtt_topic, "Push button pressed!");
    delay(400);
  }
  lastButtonState = currentButtonState;

  unsigned long now = millis();
  if (now - lastLightSend > MESSAGE_DELAY) {
    int lightValue = analogRead(LIGHT_PIN);
    Serial.print("[LOG] Light sensor value: ");
    Serial.println(lightValue);
    Serial.println("[LOG] Sending MQTT light message...\n");
    String message = "LIGHT: " + String(lightValue);
    mqtt_client.publish(mqtt_topic, message.c_str());
    lastLightSend = now;
  }

  delay(50);
}
```

Code TP 3 - Node-RED

Flow 1


```
[
    {
        "id": "794d0bebc2465bb0",
        "type": "tab",
        "label": "Flow 1",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "0542b1aa61b01c11",
        "type": "mqtt in",
        "z": "794d0bebc2465bb0",
        "name": "",
        "topic": "insa/gei/FATINE",
        "qos": "2",
        "datatype": "auto-detect",
        "broker": "caf0ba2999274ad0",
        "nl": false,
        "rap": true,
        "rh": 0,
        "inputs": 0,
        "x": 300,
        "y": 280,
        "wires": [
            [
                "28f02dca18ae01de",
                "cf9af623cb1f5229"
            ]
        ]
    },
    {
        "id": "28f02dca18ae01de",
        "type": "debug",
        "z": "794d0bebc2465bb0",
        "name": "debug 1",
        "active": true,
        "tosidebar": true,
        "console": false,
        "tostatus": false,
        "complete": "false",
        "statusVal": "",
        "statusType": "auto",
        "x": 500,
        "y": 440,
```

```
        "wires": []
    },
    {
        "id": "cf9af623cb1f5229",
        "type": "switch",
        "z": "794d0bebc2465bb0",
        "name": "",
        "property": "payload",
        "propertyType": "msg",
        "rules": [
            {
                "t": "cont",
                "v": "Push button pressed!",
                "vt": "str"
            },
            {
                "t": "cont",
                "v": "LIGHT:",
                "vt": "str"
            }
        ],
        "checkall": "true",
        "repair": false,
        "outputs": 2,
        "x": 490,
        "y": 280,
        "wires": [
            [
                "b1e47f382cd4440e",
                "c3d3eb624a2c9e16"
            ],
            [
                "207a6d2d65706a93"
            ]
        ]
    },
    {
        "id": "207a6d2d65706a93",
        "type": "function",
        "z": "794d0bebc2465bb0",
        "name": "function 1",
        "func": "// msg.payload example: \"LIGHT: 456\"\nlet s = (msg.payload ||
\"\").toString().trim();\n\n// Extract number after \"LIGHT:\"\ns = s.replace(\"LIGHT:\",
\"\").trim();\n\nlet v = parseInt(s, 10);\nif (isNaN(v)) return null;   // ignore if not a valid
number\n\nmsg.payload = v;             // now it's a number\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
```

```
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 660,
    "y": 280,
    "wires": [
        [
            "2fdefc94e6f8e156",
            "942814b763a66738"
        ]
    ]
},
{
    "id": "2fdefc94e6f8e156",
    "type": "switch",
    "z": "794d0bebc2465bb0",
    "name": "",
    "property": "payload",
    "propertyType": "msg",
    "rules": [
        {
            "t": "lt",
            "v": "500",
            "vt": "str"
        },
        {
            "t": "gte",
            "v": "500",
            "vt": "str"
        }
    ],
    "checkall": "true",
    "repair": false,
    "outputs": 2,
    "x": 650,
    "y": 380,
    "wires": [
        [
            "61ec74de646e6477"
        ],
        [
            "74e4717b3564b048"
        ]
    ]
},
{
    "id": "f2cbcc3da5b55357",
    "type": "mqtt out",
```

```
    "z": "794d0bebc2465bb0",
    "name": "",
    "topic": "insa/gei/FATINE",
    "qos": "",
    "retain": "",
    "respTopic": "",
    "contentType": "",
    "userProps": "",
    "correl": "",
    "expiry": "",
    "broker": "caf0ba2999274ad0",
    "x": 1040,
    "y": 380,
    "wires": []
},
{
    "id": "61ec74de646e6477",
    "type": "change",
    "z": "794d0bebc2465bb0",
    "name": "Turn ON",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "ON",
            "tot": "str"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 840,
    "y": 360,
    "wires": [
        [
            "f2cbcc3da5b55357"
        ]
    ]
},
{
    "id": "74e4717b3564b048",
    "type": "change",
    "z": "794d0bebc2465bb0",
    "name": "Turn OFF",
    "rules": [
```

```
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "OFF",
            "tot": "str"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 840,
    "y": 400,
    "wires": [
        [
            "f2cbcc3da5b55357"
        ]
    ]
},
{
    "id": "942814b763a66738",
    "type": "ui_chart",
    "z": "794d0bebc2465bb0",
    "name": "",
    "group": "75c314988641ac85",
    "order": 1,
    "width": 0,
    "height": 0,
    "label": "chart",
    "chartType": "line",
    "legend": "false",
    "xformat": "HH:mm:ss",
    "interpolate": "linear",
    "nodata": "",
    "dot": false,
    "ymin": "",
    "ymax": "",
    "removeOlder": 1,
    "removeOlderPoints": "",
    "removeOlderUnit": "3600",
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",
```

```
            "#ff7f0e",
            "#2ca02c",
            "#98df8a",
            "#d62728",
            "#ff9896",
            "#9467bd",
            "#c5b0d5"
        ],
        "outputs": 1,
        "useDifferentColor": false,
        "className": "",
        "x": 650,
        "y": 340,
        "wires": [
            []
        ]
    },
    {
        "id": "3014027dfc58bd70",
        "type": "ui_toast",
        "z": "794d0bebc2465bb0",
        "position": "top right",
        "displayTime": "3",
        "highlight": "",
        "sendall": true,
        "outputs": 0,
        "ok": "OK",
        "cancel": "",
        "raw": false,
        "className": "",
        "topic": "msg.payload",
        "name": "Notification",
        "x": 1030,
        "y": 300,
        "wires": []
    },
    {
        "id": "b1e47f382cd4440e",
        "type": "ui_toast",
        "z": "794d0bebc2465bb0",
        "position": "top left",
        "displayTime": "3",
        "highlight": "",
        "sendall": true,
        "outputs": 0,
        "ok": "OK",
        "cancel": "",
        "raw": false,
```

```
    "className": "",
    "topic": "msg.payload",
    "name": "Notification",
    "x": 670,
    "y": 240,
    "wires": []
},
{
    "id": "c3d3eb624a2c9e16",
    "type": "e-mail",
    "z": "794d0bebc2465bb0",
    "server": "smtp.gmail.com",
    "port": "465",
    "authtype": "BASIC",
    "saslformat": true,
    "token": "oauth2Response.access_token",
    "secure": true,
    "tls": true,
    "name": "joao.insatoulouse@gmail.com",
    "dname": "E-mail",
    "x": 650,
    "y": 200,
    "wires": []
},
{
    "id": "b2b20ee2319ca25a",
    "type": "ui_button",
    "z": "794d0bebc2465bb0",
    "name": "",
    "group": "a5c74579f91f5455",
    "order": 1,
    "width": 0,
    "height": 0,
    "passthru": false,
    "label": "button",
    "tooltip": "",
    "color": "",
    "bgcolor": "",
    "className": "",
    "icon": "",
    "payload": "ON",
    "payloadType": "str",
    "topic": "topic",
    "topicType": "msg",
    "x": 870,
    "y": 300,
    "wires": [
        [
```

```
            "3014027dfc58bd70",
            "f2cbcc3da5b55357"
        ]
    ]
},
{
    "id": "caf0ba2999274ad0",
    "type": "mqtt-broker",
    "name": "Broker_tom",
    "broker": "10.178.100.251",
    "port": 1883,
    "clientid": "",
    "autoConnect": true,
    "usetls": false,
    "protocolVersion": 4,
    "keepalive": 60,
    "cleansession": true,
    "autoUnsubscribe": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthRetain": "false",
    "birthPayload": "",
    "birthMsg": {},
    "closeTopic": "",
    "closeQos": "0",
    "closeRetain": "false",
    "closePayload": "",
    "closeMsg": {},
    "willTopic": "",
    "willQos": "0",
    "willRetain": "false",
    "willPayload": "",
    "willMsg": {},
    "userProps": "",
    "sessionExpiry": ""
},
{
    "id": "75c314988641ac85",
    "type": "ui_group",
    "name": "Sensors",
    "tab": "abbfdb8ac8339c57",
    "order": 1,
    "disp": true,
    "width": 6,
    "collapse": false,
    "className": ""
},
{
```

```
        "id": "a5c74579f91f5455",
        "type": "ui_group",
        "name": "Control",
        "tab": "abbfdb8ac8339c57",
        "order": 2,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id": "abbfdb8ac8339c57",
        "type": "ui_tab",
        "name": "Monitoring",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    },
    {
        "id": "931da52097d600fa",
        "type": "global-config",
        "env": [],
        "modules": {
            "node-red-dashboard": "3.6.6",
            "node-red-node-email": "5.0.2"
        }
    }
]
```


Flow 2


```
[
    {
        "id": "b8ec08973f17c5aa",
        "type": "tab",
        "label": "Flow 2",
        "disabled": false,
        "info": "",
        "env": []
    },
    {
        "id": "31fe31a9e8bdc5a6",
        "type": "mqtt in",
        "z": "b8ec08973f17c5aa",
        "name": "",
        "topic": "insa/gei/FATINE",
        "qos": "2",
        "datatype": "auto-detect",
        "broker": "caf0ba2999274ad0",
```

```
    "nl": false,
    "rap": true,
    "rh": 0,
    "inputs": 0,
    "x": 160,
    "y": 280,
    "wires": [
        [
            "d406fbba2cac947f",
            "8c84e6f16158db27"
        ]
    ]
},
{
    "id": "d406fbba2cac947f",
    "type": "debug",
    "z": "b8ec08973f17c5aa",
    "name": "debug",
    "active": true,
    "tosidebar": true,
    "console": false,
    "tostatus": false,
    "complete": "payload",
    "targetType": "msg",
    "statusVal": "",
    "statusType": "auto",
    "x": 510,
    "y": 140,
    "wires": []
},
{
    "id": "8c84e6f16158db27",
    "type": "switch",
    "z": "b8ec08973f17c5aa",
    "name": "",
    "property": "payload",
    "propertyType": "msg",
    "rules": [
        {
            "t": "cont",
            "v": "Push button pressed!",
            "vt": "str"
        },
        {
            "t": "cont",
            "v": "LIGHT:",
            "vt": "str"
        }
```

```
        ],
        "checkall": "true",
        "repair": false,
        "outputs": 2,
        "x": 350,
        "y": 280,
        "wires": [
            [
                "16c242da481893bb",
                "1a9a440431c78046"
            ],
            [
                "fda579ffde4b8449"
            ]
        ]
    },
    {
        "id": "fda579ffde4b8449",
        "type": "function",
        "z": "b8ec08973f17c5aa",
        "name": "function 2",
                "func": "// msg.payload example: \"LIGHT: 456\"\nlet s = (msg.payload || \"\").toString().trim();\n\n// Extract number after \"LIGHT:\"\ns = s.replace(\"LIGHT:\", \"\").trim();\n\nlet v = parseInt(s, 10);\nif (isNaN(v)) return null;  // ignore if not a valid number\n\nmsg.payload = v;            // now it's a number\nreturn msg;",
        "outputs": 1,
        "timeout": 0,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 180,
        "y": 400,
        "wires": [
            [
                "d17a8f0a07ca2a8c",
                "d67d84e171903d8d"
            ]
        ]
    },
    {
        "id": "d17a8f0a07ca2a8c",
        "type": "switch",
        "z": "b8ec08973f17c5aa",
        "name": "",
        "property": "payload",
        "propertyType": "msg",
        "rules": [
```

```
        {
            "t": "lt",
            "v": "500",
            "vt": "str"
        },
        {
            "t": "gte",
            "v": "500",
            "vt": "str"
        }
    ],
    "checkall": "true",
    "repair": false,
    "outputs": 2,
    "x": 350,
    "y": 420,
    "wires": [
        [
            "2af2d996ee58d29a"
        ],
        [
            "0e55242921901b28"
        ]
    ]
},
{
    "id": "45386a6510513912",
    "type": "mqtt out",
    "z": "b8ec08973f17c5aa",
    "name": "",
    "topic": "insa/gei/FATINE",
    "qos": "",
    "retain": "",
    "respTopic": "",
    "contentType": "",
    "userProps": "",
    "correl": "",
    "expiry": "",
    "broker": "caf0ba2999274ad0",
    "x": 1000,
    "y": 460,
    "wires": []
},
{
    "id": "2af2d996ee58d29a",
    "type": "change",
    "z": "b8ec08973f17c5aa",
    "name": "Turn ON",
```

```
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "ON",
            "tot": "str"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 540,
    "y": 400,
    "wires": [
        [
            "d46767c4244a7310"
        ]
    ]
},
{
    "id": "0e55242921901b28",
    "type": "change",
    "z": "b8ec08973f17c5aa",
    "name": "Turn OFF",
    "rules": [
        {
            "t": "set",
            "p": "payload",
            "pt": "msg",
            "to": "OFF",
            "tot": "str"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 540,
    "y": 440,
    "wires": [
        [
            "4ac90a49e2e91bcb"
        ]
    ]
```

```
},
{
    "id": "d67d84e171903d8d",
    "type": "ui_chart",
    "z": "b8ec08973f17c5aa",
    "name": "",
    "group": "75c314988641ac85",
    "order": 0,
    "width": 0,
    "height": 0,
    "label": "chart",
    "chartType": "line",
    "legend": "false",
    "xformat": "HH:mm:ss",
    "interpolate": "linear",
    "nodata": "",
    "dot": false,
    "ymin": "",
    "ymax": "",
    "removeOlder": 1,
    "removeOlderPoints": "",
    "removeOlderUnit": "3600",
    "cutout": 0,
    "useOneColor": false,
    "useUTC": false,
    "colors": [
        "#1f77b4",
        "#aec7e8",
        "#ff7f0e",
        "#2ca02c",
        "#98df8a",
        "#d62728",
        "#ff9896",
        "#9467bd",
        "#c5b0d5"
    ],
    "outputs": 1,
    "useDifferentColor": false,
    "className": "",
    "x": 350,
    "y": 380,
    "wires": [
        []
    ]
},
{
    "id": "652c2f7fec77ba41",
    "type": "ui_toast",
```

```
    "z": "b8ec08973f17c5aa",
    "position": "top right",
    "displayTime": "3",
    "highlight": "",
    "sendall": true,
    "outputs": 0,
    "ok": "OK",
    "cancel": "",
    "raw": false,
    "className": "",
    "topic": "msg.payload",
    "name": "Notification",
    "x": 730,
    "y": 560,
    "wires": []
},
{
    "id": "16c242da481893bb",
    "type": "ui_toast",
    "z": "b8ec08973f17c5aa",
    "position": "top left",
    "displayTime": "3",
    "highlight": "",
    "sendall": true,
    "outputs": 0,
    "ok": "OK",
    "cancel": "",
    "raw": false,
    "className": "",
    "topic": "msg.payload",
    "name": "Notification",
    "x": 530,
    "y": 240,
    "wires": []
},
{
    "id": "1a9a440431c78046",
    "type": "e-mail",
    "z": "b8ec08973f17c5aa",
    "server": "smtp.gmail.com",
    "port": "465",
    "authtype": "BASIC",
    "saslformat": true,
    "token": "oauth2Response.access_token",
    "secure": true,
    "tls": true,
    "name": "joao.insatoulouse@gmail.com",
    "dname": "E-mail",
```

```
        "x": 510,
        "y": 200,
        "wires": []
    },
    {
        "id": "97f0bb456e24b942",
        "type": "ui_switch",
        "z": "b8ec08973f17c5aa",
        "name": "",
        "label": "switch",
        "tooltip": "",
        "group": "a5c74579f91f5455",
        "order": 2,
        "width": 0,
        "height": 0,
        "passthru": true,
        "decouple": "false",
        "topic": "topic",
        "topicType": "msg",
        "style": "",
        "onvalue": "true",
        "onvalueType": "bool",
        "onicon": "",
        "oncolor": "",
        "offvalue": "false",
        "offvalueType": "bool",
        "officon": "",
        "offcolor": "",
        "animate": false,
        "className": "",
        "x": 530,
        "y": 500,
        "wires": [
            [
                "d14901935433e0c7",
                "652c2f7fec77ba41"
            ]
        ]
    },
    {
        "id": "d14901935433e0c7",
        "type": "function",
        "z": "b8ec08973f17c5aa",
        "name": "Manual Switch",
          "func": "msg.source = \"manual\";\n\nif (msg.payload === true) {\n    msg.command =
\"ON\";\n} else {\n    msg.command = \"AUTO\";\n}\n\nreturn msg;\n",
        "outputs": 1,
        "timeout": 0,
```

```
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 740,
    "y": 500,
    "wires": [
        [
            "1466a8d2dbf80fe0"
        ]
    ]
},
{
    "id": "d46767c4244a7310",
    "type": "change",
    "z": "b8ec08973f17c5aa",
    "name": "",
    "rules": [
        {
            "t": "set",
            "p": "source",
            "pt": "msg",
            "to": "auto",
            "tot": "str"
        },
        {
            "t": "set",
            "p": "command",
            "pt": "msg",
            "to": "ON",
            "tot": "str"
        }
    ],
    "action": "",
    "property": "",
    "from": "",
    "to": "",
    "reg": false,
    "x": 740,
    "y": 400,
    "wires": [
        [
            "1466a8d2dbf80fe0"
        ]
    ]
},
{
    "id": "4ac90a49e2e91bcb",
```

```
        "type": "change",
        "z": "b8ec08973f17c5aa",
        "name": "",
        "rules": [
            {
                "t": "set",
                "p": "source",
                "pt": "msg",
                "to": "auto",
                "tot": "str"
            },
            {
                "t": "set",
                "p": "command",
                "pt": "msg",
                "to": "OFF",
                "tot": "str"
            }
        ],
        "action": "",
        "property": "",
        "from": "",
        "to": "",
        "reg": false,
        "x": 740,
        "y": 440,
        "wires": [
            [
                "1466a8d2dbf80fe0"
            ]
        ]
    },
    {
        "id": "1466a8d2dbf80fe0",
        "type": "function",
        "z": "b8ec08973f17c5aa",
        "name": "Override Controller",
        "func": "// Saved state\nlet manualOverride = flow.get(\"manualOverride\") || false;\nlet manualState = flow.get(\"manualState\") || \"OFF\";\n\n// If a manual command arrives, ENABLE override and store the state\nif (msg.source === \"manual\") {\n    manualOverride = true;\n    manualState = msg.command;\n\n    flow.set(\"manualOverride\", manualOverride);\n    flow.set(\"manualState\", manualState);\n\n    msg.payload = manualState;  // this is what goes to MQTT OUT\n    return msg;\n}\n\n// If a manual command arrives to go back to automatic mode\nif (msg.source === \"manual\" && msg.command === \"AUTO\") {\n    flow.set(\"manualOverride\", false);\n    node.status({ fill: \"green\", shape: \"dot\", text: \"AUTO\" });\n    return null; // do not publish anything\n}\n\n// No override -> let the automatic logic send commands\nnode.status({ fill: \"green\", shape: \"dot\", text: `AUTO ${msg.command}` });\nmsg.payload = msg.command;\nreturn msg;\n",
```

```
    "outputs": 1,
    "timeout": 0,
    "noerr": 0,
    "initialize": "",
    "finalize": "",
    "libs": [],
    "x": 970,
    "y": 400,
    "wires": [
        [
            "45386a6510513912"
        ]
    ]
},
{
    "id": "caf0ba2999274ad0",
    "type": "mqtt-broker",
    "name": "Broker_tom",
    "broker": "10.178.100.251",
    "port": 1883,
    "clientid": "",
    "autoConnect": true,
    "usetls": false,
    "protocolVersion": 4,
    "keepalive": 60,
    "cleansession": true,
    "autoUnsubscribe": true,
    "birthTopic": "",
    "birthQos": "0",
    "birthRetain": "false",
    "birthPayload": "",
    "birthMsg": {},
    "closeTopic": "",
    "closeQos": "0",
    "closeRetain": "false",
    "closePayload": "",
    "closeMsg": {},
    "willTopic": "",
    "willQos": "0",
    "willRetain": "false",
    "willPayload": "",
    "willMsg": {},
    "userProps": "",
    "sessionExpiry": ""
},
{
    "id": "75c314988641ac85",
    "type": "ui_group",
```

```
        "name": "Sensors",
        "tab": "abbfdb8ac8339c57",
        "order": 1,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id": "a5c74579f91f5455",
        "type": "ui_group",
        "name": "Control",
        "tab": "abbfdb8ac8339c57",
        "order": 2,
        "disp": true,
        "width": 6,
        "collapse": false,
        "className": ""
    },
    {
        "id": "abbfdb8ac8339c57",
        "type": "ui_tab",
        "d": true,
        "name": "Monitoring",
        "icon": "dashboard",
        "disabled": false,
        "hidden": false
    },
    {
        "id": "85772eb742bda0a0",
        "type": "global-config",
        "env": [],
        "modules": {
            "node-red-dashboard": "3.6.6",
            "node-red-node-email": "5.0.2"
        }
    }
]
```

Code TP 4 - Arduino

```
#include <ESP8266WiFi.h>
#include <ArduinoHA.h>

// Configuration WiFi
const char* ssid = "asni";
const char* password = "asniasni";

// Configuration MQTT
const char* mqtt_server = "192.168.1.1";  // IP de Home Assistant
const int mqtt_port = 1883;
const char* mqtt_user = "insa";
const char* mqtt_password = "insa";

// Pin de la LED
const int LED_PIN = LED_BUILTIN;  // LED sur NodeMCU
WiFiClient espClient;
HADevice device;
HAMqtt mqtt(espClient, device);

// Déclaration de la LED comme "light" dans Home Assistant
HALight led("LED_Fatine", HALight::BrightnessFeature);

void onStateCommand(bool state, HALight* sender) {
  if (state) {
    digitalWrite(LED_PIN, HIGH);  // LED allumée
    Serial.println("LED ON");
  } else {
    digitalWrite(LED_PIN, LOW);  // LED éteinte
    Serial.println("LED OFF");
  }

  sender->setState(state);  // Mise à jour de l'état dans HA
}
void onBrightnessCommand(uint8_t brightness, HALight* sender) {
   analogWrite(LED_PIN, map(brightness, 0, 255, 255, 0));   // Logique
inversée
  sender->setBrightness(brightness);
  Serial.print("Luminosité: ");
  Serial.println(brightness);
}
void setup() {
```

```
  Serial.begin(115200);
  byte mymac[WL_MAC_ADDR_LENGTH];
  WiFi.macAddress(mymac);
  device.setUniqueId(mymac, WL_MAC_ADDR_LENGTH);

  Serial.printf("\n\n");
  for (int i = 0; i < WL_MAC_ADDR_LENGTH; i++)
    Serial.printf("%02x", mymac[i]);
  Serial.printf("\n\n");

  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);  // LED éteinte au démarrage

  // Connexion WiFi
  Serial.print("Connexion WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connecté!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());
  //////
  // Configuration du dispositif
  device.setName("ESP8266 Fatine");
  device.setSoftwareVersion("1.0.3");
  device.setManufacturer("INSA Toulouse");
  device.setModel("ESP8266 NodeMCU");

  // Configuration de la LED
  led.setName("LED de fatine");
  led.onStateCommand(onStateCommand);
  led.onBrightnessCommand(onBrightnessCommand);

  // Connexion MQTT
  mqtt.begin(mqtt_server, mqtt_port, mqtt_user, mqtt_password);
}
void loop() {
  mqtt.loop();

  // Reconnexion WiFi si nécessaire
  if (WiFi.status() != WL_CONNECTED) {
```

```
    Serial.println("WiFi déconnecté, reconnexion...");
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
      delay(500);
    }
  }
}
```