



**M1 SAR Informatique - IOC  
Projet**

Yavuz ERGUN  
Danny VANG

# Introduction et Contexte

Le but du projet est de faire communiquer un ESP32 avec un utilisateur avec un site web à travers un Raspberry Pi.

De ce fait, on aura besoin du matériel suivant:



Un ESP32

Un RaspberryPi

D'un téléphone pour se connecter au même réseau.

Pour coder, on utilisera le logiciel Arduino pour injecter le code dans l'ESP32. Et le code sera en C.

Le rapport est répartie en 4 étapes:

- Installation du Broker (MQTT)
- L'implémentation et connection de ESP32
- Envoi de requête ( allumer la LED)
- Interaction via le site Web

# Installation du Broker

Pour commencer, nous allons installer MQTT sur le Raspberry pi.

**MQTT**(Message Queuing Telemetry Transport). Cela se résume à un protocole facile à utiliser et flexible qui vous permet d'envoyer des messages arbitraires sur un réseau à tout autre appareil intéressé.

En clair, MQTT se compose d'abonnés( les éditeurs et les clients), et ils sont tous connectés via un courtier ( le broker).

Nous nous sommes aider du lien ci-dessous pour installer MQTT sur le Raspberry Pi :

<https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/>

En premier lieu, nous avons installer le mosquitto MQTT Broker :

```
sudo apt install mosquitto mosquitto-clients
```

Puis, nous avons activer les services liés à mosquitto

```
sudo systemctl status mosquitto
```

Maintenant, afin de tester, nous avons envoyer en local un message de test :

On active d'abord mosquitto:

```
pi@raspi-nico:~ $ mosquitto
1651578087: mosquitto version 1.5.7 starting
1651578087: Using default config.
1651578087: Opening ipv4 listen socket on port 1883.
1651578087: Opening ipv6 listen socket on port 1883.
```

Puis dans un terminal 1: on s'abonne à un topic test, et dans un autre terminal : on publie

```
pi@raspi-nico:~ $ mosquitto_sub -h localhost -t "test/message"
Hello, world

```

```
pi@raspi-nico:~ $ mosquitto_pub -h localhost -t "test/message" -m "Hello, world"
pi@raspi-nico:~ $
```

Maintenant que mosquitto est installer et fonctionne, on passe à l'implémentation de ESP32

# Implémentation de ESP32

Le but de cette partie sera de connecter un ESP32 au RaspberryPi, c'est à dire de d'abord:

- de connecter le microcontrôleur sur le même réseau local
- Puis de se connecter directement au Broker.

Pour cela, on aura besoin de 3 bibliothèque essentiels :

```
#include <Wifi.h> : pour la connection WIFI
```

```
#include <PubSubClient.h> : pour la connection MQTT
```

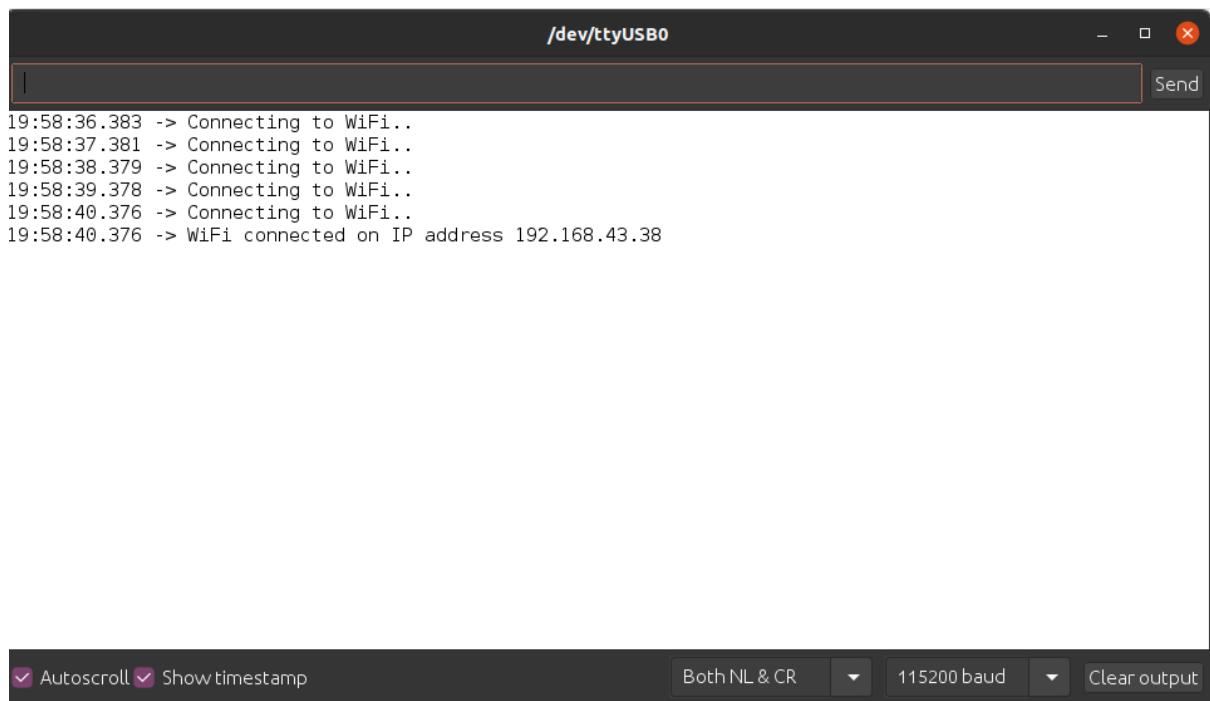
Commençons d'abord par la connection WiFi:

Nous avons déclarer dans un Define les identifiants Wifi:

```
#define WIFI_SSID "Danny"  
#define WIFI_PASSWORD "136452879"
```

Ensuite nous avons déclaré dans une fonction *connect\_Wifi()* qui va se connecter et afficher l'état de la connexion grâce au fonction begin t status.

```
void connect_WIFI(){  
    // Set software serial baud to 115200;  
    Serial.begin(115200);  
  
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);  
  
    //il continue à chercher jusqu'à qu'il soit connecter  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(1000);  
        Serial.println("Connecting to WiFi..");  
    }  
  
    Serial.print("WiFi connected on IP address ");  
    Serial.println(WiFi.localIP());  
}
```



The screenshot shows a terminal window titled '/dev/ttyUSB0'. The window contains the following text:

```
19:58:36.383 -> Connecting to WiFi..
19:58:37.381 -> Connecting to WiFi..
19:58:38.379 -> Connecting to WiFi..
19:58:39.378 -> Connecting to WiFi..
19:58:40.376 -> Connecting to WiFi..
19:58:40.376 -> WiFi connected on IP address 192.168.43.38
```

At the bottom of the terminal window, there are several configuration options:

- Autoscroll (checkbox checked)
- Show timestamp (checkbox checked)
- Both NL & CR (dropdown menu)
- 115200 baud (dropdown menu)
- Clear output (button)

Maintenant, on passe à la connection du Broker

Dans cette fonction, la bibliothèque PubSubClient.h permet d'utiliser la fonction *connect()* de client qui va nous permettre de nous connecter au Broker et ensuite publier des données à celui-ci via la fonction *publish* à l'aide d'un topic. Cela permettra au machine voulant s'abonner ou se souscrire de récupérer par la suite les données préalablement publiées au répartiteur MQTT.

```
void connect_MQTT(){
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("esp32_Danny")) {
            Serial.println("connected");
            // Subscribe
            client.subscribe("esp32/output");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
```

Dans les fonctions setup() et void(), nous avons écrits :

```
void setup() {  
    pinMode(led, OUTPUT);  
    connected_WIFI();  
  
    client.setServer(MQTT_SERVER, MQTT_PORT);  
    client.setCallback(callback);  
  
    connect_MQTT();  
  
    server.on("/", handleRoot);  
    server.on("/on", handleOn);  
    server.on("/off", handleOff);  
    server.onNotFound(handleNotFound);  
    server.begin();  
  
    Serial.println("Serveur web actif!");  
}  
  
void loop() {  
    server.handleClient();  
    if (!client.connected()) {  
        Serial.println("Connecting ...");  
        delay(5000);  
        if (client.connect("esp32_Danny")) {  
            client.subscribe("esp32/output");  
            client.setCallback(callback);  
        }  
    }  
    client.loop();  
    delay(1);  
}
```

Dans setup(), on met la led en mode OUTPUT, puis il va faire appel à la fonction de connection Wifi. Ensuite on connecte au broker et enfin on active le server pour le site web.

```
/dev/ttyUSB0  
  
[ ]  
  
17:09:39.452 -> Connecting to WiFi..  
17:09:40.452 -> Connecting to WiFi..  
17:09:41.451 -> Connecting to WiFi..  
17:09:42.451 -> Connecting to WiFi..  
17:09:43.451 -> Connecting to WiFi..  
17:09:43.451 -> WiFi connected on IP address 192.168.43.38  
17:09:43.451 -> Attempting MQTT connection...failed, rc=-2 try again in 5 seconds  
17:09:48.512 -> Attempting MQTT connection...failed, rc=-2 try again in 5 seconds  
17:09:53.571 -> Attempting MQTT connection...connected
```

Du côté du Broker, on remarque aussi qu'il est bien connecter

```
pi@raspi-nico:~ $ mosquitto  
1652022625: mosquitto version 1.5.7 starting  
1652022625: Using default config.  
1652022625: Opening ipv4 listen socket on port 1883.  
1652022625: Opening ipv6 listen socket on port 1883.  
1652022629: New connection from 192.168.43.38 on port 1883.  
1652022629: New client connected from 192.168.43.38 as ESP8266Client (c1, k15).
```

# Envoi de requête

Maintenant on veut essayer d'allumer la led, et tous ce passe dans la fonction *callback()*. Celui-ci va traiter les messages reçus par l'ESP32 (dans ce cas, les messages qui utilisent la fonction *subscribe()*).

```
void callback(char* topic, byte* message, unsigned int length) {  
    Serial.print("Message arrived on topic: ");  
    Serial.print(topic);  
    Serial.print(". Message: ");  
    String messageTemp;  
  
    for (int i = 0; i < length; i++) {  
        Serial.print((char)message[i]);  
        messageTemp += (char)message[i];  
    }  
    Serial.println();  
  
    if (String(topic) == "esp32/output") {  
        Serial.print("Changing output to ");  
        if(messageTemp == "on"){  
            digitalWrite(led, HIGH);  
            Serial.println("on");  
        }  
        else if(messageTemp == "off"){  
            digitalWrite(led, LOW);  
            Serial.println("off");  
        }  
    }  
}
```

Pour le fonctionnement, nous avons filmé le résultat et est disponible avec lien ci-dessous :  
<https://youtu.be/OXBZTWeXQqU>

# Interaction avec un page WEB

Dans cette dernière partie, on veut créer un site Web capable d'interagir avec l'ESP32. On commence par rajouter une bibliothèque :

```
#include <WebServer.h> pour la création du Website
```

Pour cela, nous ajoutons une fonction handleRoot() qui créer une page HTML pour le visuel

```
void handleRoot()
{
    String page = "<!DOCTYPE html>";
    page += "<html lang='fr'>";
    page += "<head>";
    page += "  <title>Serveur ESP32</title>";
    page += "  <meta http-equiv='refresh' content='60' name='viewport'";
    content='width=device-width, initial-scale=1' charset='UTF-8' />";
    page += "  <link rel='stylesheet' href='https://www.w3schools.com/w3css/4/w3.css'>";
    page += "</head>";

    page += "<body>";
    page += "  <div class='w3-card w3-blue w3-padding-small w3-jumbo w3-center'>";
    page += "    <p>ÉTAT LED: "; page += texteEtatLed[etatLed]; + "</p>";
    page += "  </div>";

    page += "  <div class='w3-bar'>";
    page += "    <a href='/on' class='w3-bar-item w3-button w3-border w3-jumbo' style='width:50%; height:50%;'>ON</a>";
    page += "    <a href='/off' class='w3-bar-item w3-button w3-border w3-jumbo' style='width:50%; height:50%;'>OFF</a>";
    page += "  </div>";

    page += "  <div class='w3-center w3-padding-16'>";
    page += "  </div>";

    page += "</body>";
    page += "</html>";

    server.setContentLength(page.length());
    server.send(200, "text/html", page);
}
```

On pretera attention à la variable **texteEtatLed[etatLed]** qui affichera sur la page Web l'état de la LED. Ensuite, on envoie le contenu dans le serveur.

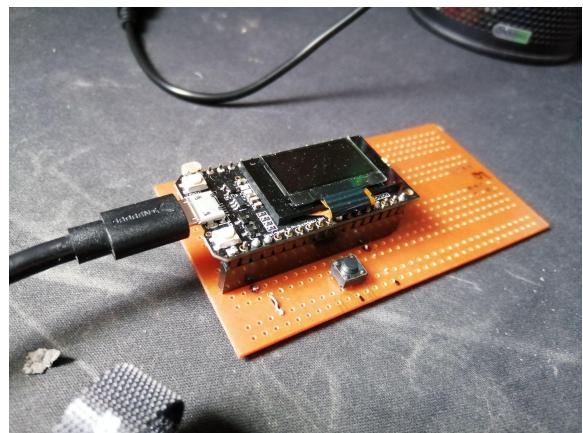
On ajoute aussi les fonctions handleOn() et handleOff() pour changer l'état de la lumiere, qui l'envoie dans le serveur pour le mettre à jour. Et on oublie pas de traiter le cas d'erreur avec une erreur 404

```
void handleOn()
{
    etatLed = 1;
    digitalWrite(led, HIGH);
    server.sendHeader("Location", "/");
    server.send(303);
}

void handleOff()
{
    etatLed = 0;
    digitalWrite(led, LOW);
    server.sendHeader("Location", "/");
    server.send(303);
}

void handleNotFound()
{
    server.send(404, "text/plain", "404: Not found");
}
```

Et voici le résultat final :



ÉTAT LED: ALLUMÉE!

ON

OFF

