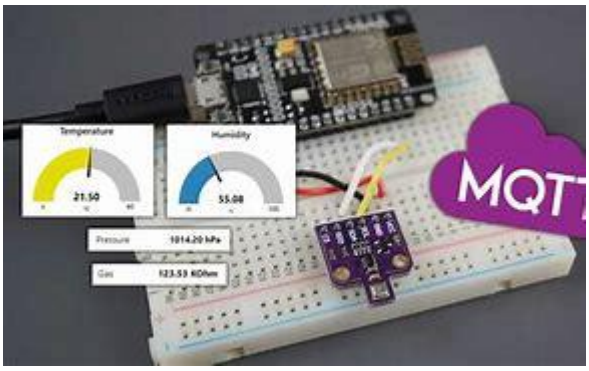
Compte Rendu du Projet CAPTEUR

Introduction

Ce document présente un compte rendu du projet CAPTEUR, visant à créer une multi-sonde domotique connectée via WiFi. Le projet implique l'installation et la configuration de divers composants, notamment un Raspberry Pi, un serveur MQTT, des capteurs sur ESP32, une base de données InfluxDB, et une application web sur Node-Red. Ce rapport détaillera chaque étape du processus ainsi que les exemples illustrés.



Liste des composants :

-LM35 capteur de température

-DHT22 capteur de température et d'humidité

-HC-SR501 capteur de présence humaine ou de mouvement

-MQ135 capteur de gaz

-LDR capteur qui détecte la lumière ambiante

- Raspberry Pi 4 ordinateur à carte unique performant, conçu pour les projets de développement et de domotique.

-ESP32 puce microcontrôleur SoC (System on Chip) dotée de fonctionnalités avancées de connectivité Wi-Fi et Bluetooth

Dans ce scénario, nous allons montrer comment utiliser MQTT pour contrôler une LED à distance en publiant et en s'abonnant à des messages MQTT. Nous utiliserons Node-RED comme client MQTT pour publier des messages, un broker Mosquitto sur un Raspberry Pi pour relayer ces messages, et un ESP32 comme client MQTT pour recevoir les messages et contrôler une LED.

Matériel Nécessaire

- Raspberry Pi avec Mosquitto installé.
- ESP32.
- Une LED et une résistance de 220Ω.

- Node-RED installé sur un ordinateur ou sur le Raspberry Pi.

Étapes de Réalisation

1. Installer et Configurer Mosquitto sur le Raspberry Pi
    - Assurez-vous que votre Raspberry Pi est à jour :

      sh
      Copier le code
      sudo apt-get update
      sudo apt-get upgrade

    - Installez Mosquitto et les clients Mosquitto :

      sh
      Copier le code
      sudo apt-get install mosquitto mosquitto-clients

    - Démarrez Mosquitto et assurez-vous qu'il s'exécute :

      sh
      Copier le code
      sudo systemctl start mosquitto
      sudo systemctl enable mosquitto

2. Configurer Node-RED
    - Si Node-RED n'est pas encore installé, installez-le sur votre ordinateur ou votre Raspberry Pi :

      sh
      Copier le code
      sudo apt-get install nodered
      sudo systemctl start nodered
      sudo systemctl enable nodered

    - Accédez à l'interface de Node-RED en ouvrant un navigateur et en entrant l'adresse suivante :

      arduino
      Copier le code
      http://<Raspberry_Pi_IP>:1880

3. Configuration de Node-RED pour Publier des Messages MQTT
    - Dans Node-RED, ajoutez un nœud inject et configurez-le pour envoyer une charge utile "on".
    - Ajoutez un nœud mqtt out et configurez-le avec les paramètres suivants :
        - Serveur : mqtt://<Raspberry_Pi_IP>:1883
        - Topic : esp32/output
    - Connectez le nœud inject au nœud mqtt out.
    - Déployez votre flow en cliquant sur le bouton "Deploy".
4. Connecter et Programmer l'ESP32

- Connectez une LED à l'ESP32 :
  - La patte longue (anode) de la LED va à GPIO 2 (ou une autre broche numérique disponible).
  - La patte courte (cathode) de la LED va à une résistance de 220Ω, puis à la terre (GND).
-  Programmez l'ESP32 avec le code suivant pour s'abonner au topic MQTT et contrôler la LED :

cpp
Copier le code

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

// Configuration du WiFi
const char* ssid = "your_SSID";
const char* password = "your_PASSWORD";

// Configuration du MQTT
const char* mqtt_server = "your_Raspberry_Pi_IP";
const int mqtt_port = 1883;
const char* mqtt_topic = "esp32/output";

WiFiClient espClient;
PubSubClient client(espClient);

const int ledPin = 2;

void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
}

void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
```

```
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  Serial.println(message);

  if (message == "on") {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
      client.subscribe(mqtt_topic);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
```

}

Deuxième Scénario : Surveillance de la Température et de l'Humidité via MQTT

Contexte

Dans ce scénario, nous allons montrer comment utiliser MQTT pour surveiller en temps réel la température et l'humidité à l'aide de capteurs connectés. Nous utiliserons un ESP32 pour collecter les données des capteurs, publier ces données sur des topics MQTT, et un Raspberry Pi avec Mosquitto pour relayer ces messages à un client MQTT abonné, qui les affichera.

Matériel Nécessaire

- Raspberry Pi avec Mosquitto installé.
- ESP32.
- Capteur DHT22 pour mesurer la température et l'humidité.
- Node-RED pour visualiser les données.
- Et ect…

Étapes de Réalisation

1. Installer et Configurer Mosquitto sur le Raspberry Pi
    - On s'assure que la Raspberry Pi est à jour :

      sh
      Copier le code
      sudo apt-get update
      sudo apt-get upgrade

    - On installe Mosquitto et les clients Mosquitto :

      sh
      Copier le code
      sudo apt-get install mosquitto mosquitto-clients

    - On Démarre Mosquitto et on s'assure qu'il s'exécute :

      sh
      On copie le code
      sudo systemctl start mosquitto
      sudo systemctl enable mosquitto

2. Connecter et Programmer l'ESP32
    - On connecte les capteurs DHT22 à l'ESP32 ainsi que les autres capteurs de la même manière :
        - VCC à 3.3V.
        - GND à la terre.
        - DATA à GPIO 4.
    - On rogramme l'ESP32 avec les codes suivant pour lire les données des capteur et les publier via MQTT :
3. #include <DHT.h>

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Adafruit_Sensor.h>   // Bibliothèque pour les capteurs Adafruit
#include <MQ135.h>

// Définir les broches des entrées
int obscur = 34;              // Entrée analogique où le LDR est connecté

#define MQ135_PIN 39       // Entrée capteur qualité air
MQ135 gasSensor = MQ135(MQ135_PIN);

#define DHTPIN 33          // Broche de données du capteur DHT22 (modifiable selon votre configuration)
#define DHTTYPE DHT22      // Définir le type de capteur (DHT22)
DHT dht(DHTPIN, DHTTYPE);   // Création d'un objet DHT


const int pirPin = 32;     // Définir la broche du capteur PIR (utiliser une broche valide)


// Paramètres WiFi
const char* id = "dlink-4E56";
const char* mdp = "fvrnf37378";


// Paramètres MQTT
const char* IP_MQTT = "192.168.0.108";  // Adresse IP du broker MQTT
const int port_mqtt = 1883;

const char* mqtt_topic_ldr = "obscur";         // Sujet sur lequel publier les données LDR
const char* mqtt_topic_hum = "humidite";         // Sujet sur lequel publier les données d'humidité
const char* mqtt_topic_ppm = "co2";         // Sujet sur lequel publier les données de qualité de l'air
const char* mqtt_topic_temp = "temperature";       // Sujet sur lequel publier les données de température
const char* mqtt_topic_pres = "capteur";       // Sujet sur lequel publier les données de présence

WiFiClient espClient;
PubSubClient client(espClient);


void setup() {
  // Initialiser la communication série à 115200 bits par seconde
  Serial.begin(115200);

  // Connectez-vous au réseau WiFi
  setup_wifi();

  // Configurer le client MQTT
  client.setServer(IP_MQTT, port_mqtt);
```

```
51.
52.   // Initialisation du capteur DHT
53.   dht.begin();
54.
55.   // Configurer la broche du capteur PIR comme entrée
56.   pinMode(pirPin, INPUT);
57. }
58.
59. void setup_wifi() {
60.   delay(10);
61.   // Démarrer la connexion WiFi
62.   Serial.println();
63.   Serial.print("Connexion au réseau ");
64.   Serial.println(id);
65.
66.   WiFi.begin(id, mdp);
67.
68.   while (WiFi.status() != WL_CONNECTED) {
69.     delay(500);
70.     Serial.print(".");
71.   }
72.
73.   Serial.println("");
74.   Serial.println("WiFi connecté");
75.   Serial.println("Adresse IP : ");
76.   Serial.println(WiFi.localIP());
77. }
78.
79.
80.
81.
82. void loop() {
83.   if (!client.connected()) {
84.     reconnect();
85.   }
86. //  client.loop();
87.
88.
89.   // Lire la valeur de l'entrée analogique
90.   int Value = analogRead(obscur);
91.
92.                                                    // Afficher la valeur lue sur le moniteur série
93.   Serial.print("LDR Value: ");
94.   Serial.println(Value);
95.
96.                                                   // Déterminer la valeur à envoyer
97.   const char* mqtt_value_ldr;
98.   if (Value < 3000) {
99.     mqtt_value_ldr = "1";
100.              Serial.println("Il fait jour");
101.            } else {
102.              mqtt_value_ldr = "0";
```

```
            Serial.println("Il fait nuit");
        }

                                                    // Publier la valeur sur le serveur MQTT
        client.publish(mqtt_topic_ldr, mqtt_value_ldr);


        float co2 = gasSensor.getPPM();
        Serial.print("CO2 concentration in ppm: ");
        Serial.println(co2);

                                                     // Convertir la valeur en chaîne de
    caractères
        char ppmStr[8];
        dtostrf(co2, 1, 2, ppmStr);

                                                   // Publier la valeur de qualité de l'air sur le
    serveur MQTT
        client.publish(mqtt_topic_ppm, ppmStr);


                                                  // Lecture de l'humidité relative
        float humidite = dht.readHumidity();

                                                       // Vérifier si la lecture est valide
        if (!isnan(humidite)) {
                                                    // Convertir la valeur en chaîne de
    caractères
            char humStr[8];
            dtostrf(humidite, 1, 2, humStr);

                                                     // Publier la valeur de l'humidité sur le
    serveur MQTT
            client.publish(mqtt_topic_hum, humStr);
        } else {
            Serial.println("Échec de lecture de l'humidité");
        }


                                                          // Lecture de la température
        float temperature = dht.readTemperature();

                                                     // Convertir la valeur en chaîne de
    caractères
            char tempStr[8];
            dtostrf(temperature, 1, 2, tempStr);

                                                                  // Publier la valeur de la
    température sur le serveur MQTT
            client.publish(mqtt_topic_temp, tempStr);


```

```
149.
150.                                                                    // Lecture de la présence
151.          int capteur = digitalRead(pirPin);
152.
153.                                                                  // Convertir la valeur en chaîne de
    caractères
154.          char presStr[2];
155.          itoa(capteur, presStr, 10);
156.
157.                                                                // Publier la valeur de la présence sur
    le serveur MQTT
158.          client.publish(mqtt_topic_pres, presStr);
159.
160.          delay(1000);
161.         }
162.
163.
164.
165.
166.         void reconnect() {
167.          while (!client.connected()) {  // Boucle jusqu'à la reconnexion
168.               Serial.print("En attente de connexion, veuillez patienter...");
169.
170.             if (client.connect("ESP32Client")) {  // Tentative de connexion MQTT
171.             Serial.println("connecté!!!");
172.             }
173.             else {
174.             Serial.print("Ca marche pas !!!");
175.             Serial.print(client.state());
176.             Serial.println(" try again in 5 seconds");
177.             delay(5000);
178.            }
179.          }
180.         }
```

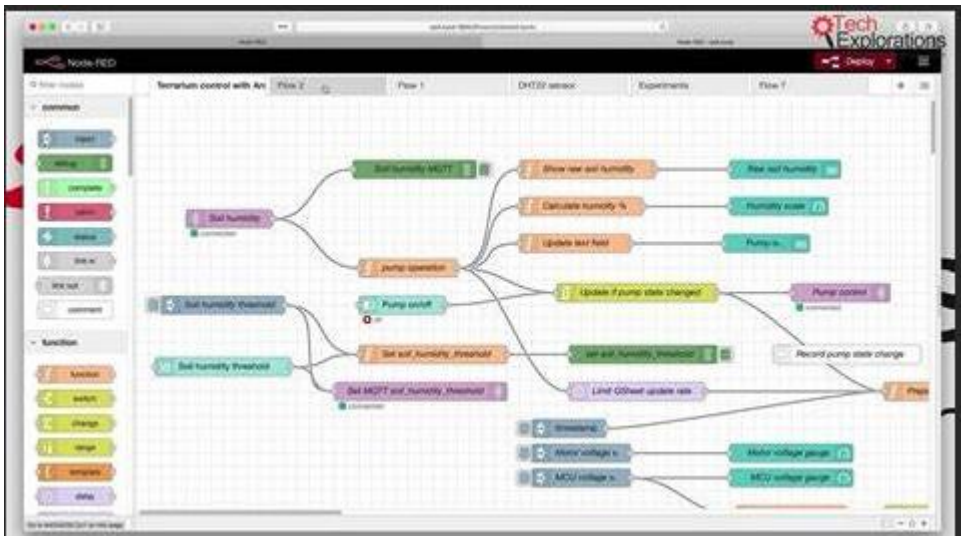
Configurer Node-RED pour Afficher les Données

- Il faut installer Node-RED
- Copier le code :

  sudo apt-get install nodered
  sudo systemctl start nodered
  sudo systemctl enable nodered

- On accéde à l'interface de Node-RED en ouvrant un navigateur et en entrant l'adresse suivante :

  arduino
  le code
  http://<Raspberry_Pi_IP>:1883



- Dans Node-RED, on ajoute deux nœuds mqtt in :
  - Serveur : mqtt://<Raspberry_Pi_IP>:1883
  - Topic : esp32/temperature pour l'un, esp32/humidity pour l'autre ainsi que pour les autres capteurs.

2. Cahier des charges

2.2 Installation d'un serveur MQTT sur le Raspberry Pi

Pour permettre la communication entre les capteurs et le serveur, il est essentiel d'installer et de configurer un serveur MQTT (Mosquitto) sur le Raspberry Pi. Cette section détaille les étapes nécessaires pour mener à bien cette installation.

Mise à jour du système

Avant d'installer de nouveaux logiciels, il est crucial de s'assurer que le système d'exploitation du Raspberry Pi est à jour. Cela garantit que toutes les bibliothèques et dépendances nécessaires sont à leur version la plus récente, ce qui peut prévenir des problèmes de compatibilité et de sécurité. Les étapes sont les suivantes :

1. Exécution des commandes de mise à jour :
   - On ouvre un terminal sur le Raspberry Pi.
   - On exécute la commande suivante pour mettre à jour la liste des paquets disponibles :
   - Ensuite, on exécute la commande suivante pour installer les mises à jour disponibles :

     shell
     Copier le code
     sudo apt-get upgrade

     Cette commande installe les versions les plus récentes des paquets disponibles, améliorant ainsi la sécurité et les performances.

Installation de Mosquitto

Mosquitto est un broker MQTT léger et efficace qui permet de gérer les messages entre les capteurs et les applications. Pour l'installer, suivez ces étapes :

1. Installation du paquet Mosquitto :
    - Toujours dans le terminal, on exécute la commande suivante pour installer Mosquitto et ses clients :

      shell
      Copier le code
      sudo apt-get install mosquitto mosquitto-clients

      Cette commande télécharge et installe le serveur Mosquitto ainsi que les outils clients nécessaires pour interagir avec le broker MQTT.

2. Vérification de l'installation :
    - Après l'installation, il est important de vérifier que le service Mosquitto fonctionne correctement. On peut vérifier l'état du service en utilisant la commande :

      shell
      Copier le code
      systemctl status mosquitto

      Si le service est en cours d'exécution, le terminal affichera un message indiquant que Mosquitto est actif et fonctionne. Si ce n'est pas le cas, des messages d'erreur indiqueront les problèmes à résoudre.

Démarrage automatique

Pour garantir que Mosquitto démarre automatiquement à chaque démarrage du Raspberry Pi, il faut configurer le service pour qu'il soit activé au démarrage :

1. Activation du service au démarrage :
    - On exécute la commande suivante :

      shell
      Copier le code
      sudo systemctl enable mosquitto

      Cette commande configure le système pour démarrer Mosquitto automatiquement lors de chaque démarrage du Raspberry Pi. Cela assure que le serveur MQTT sera toujours disponible sans intervention manuelle, ce qui est crucial pour les applications IoT.

Configuration de la sécurité

Afin de sécuriser les communications MQTT, il est nécessaire de désactiver les connexions anonymes et de configurer un fichier de mots de passe :

1. Modification du fichier de configuration :
    - On ouvre le fichier de configuration de Mosquitto avec un éditeur de texte :

      shell
      Copier le code

sudo nano /etc/mosquitto/mosquitto.conf

Nano est un éditeur de texte simple à utiliser qui permet de modifier facilement des fichiers de configuration.

- On ajoute ou modifie les lignes suivantes pour désactiver les connexions anonymes et indiquer l'emplacement du fichier de mots de passe :

  bash
  Copier le code
  allow_anonymous false
  password_file /etc/mosquitto/passwd

  allow_anonymous false désactive toutes les connexions non authentifiées, tandis que password_file indique à Mosquitto où trouver le fichier contenant les noms d'utilisateur et les mots de passe.

2. Création du fichier de mots de passe :
   - On crée un fichier de mots de passe et ajoute un utilisateur avec la commande suivante :

     shell
     Copier le code
     sudo mosquitto_passwd -c /etc/mosquitto/passwd <username>

     On suit les instructions pour entrer et confirmer le mot de passe pour le nouvel utilisateur. Cela crée un fichier sécurisé contenant les informations d'authentification.

3. Redémarrage du service Mosquitto :
   - Après avoir modifié le fichier de configuration, on redémarre le service Mosquitto pour appliquer les changements :

     shell
     Copier le code
     sudo systemctl restart mosquitto

     Cela garantit que les nouvelles configurations sont prises en compte par le serveur MQTT.

Configuration du routeur

Pour permettre aux dispositifs externes de communiquer avec le broker MQTT sur le Raspberry Pi, il est souvent nécessaire de configurer le routeur pour rediriger les requêtes appropriées :

1. Configuration des ports :
   - On accède à l'interface de configuration du routeur via un navigateur web en entrant l'adresse IP du routeur.
   - On configure le routage des ports pour rediriger les requêtes sur les ports 1883 (non sécurisé) vers l'adresse IP du Raspberry Pi.
   - Cette configuration dépend du modèle de routeur, et des guides spécifiques à chaque modèle peuvent être consultés si nécessaire.

Par exemple, dans l'interface du routeur, il faut chercher les options de port forwarding ou de virtual server. On crée des règles pour rediriger les ports externes 1883 vers l'adresse IP interne du Raspberry Pi, spécifiant les mêmes numéros de port internes.

Voici le code final pour la liaison des 2 composants pour la lecture des données transmises par les capteurs :

```
#include <DHT.h>

#include <WiFi.h>

#include <PubSubClient.h>

#include <Adafruit_Sensor.h>   // Bibliothèque pour les capteurs Adafruit

#include <MQ135.h>



// Définir les broches des entrées

int obscur = 34;              // Entrée analogique où le LDR est connecté



#define MQ135_PIN 39       // Entrée capteur qualité air

MQ135 gasSensor = MQ135(MQ135_PIN);



#define DHTPIN 33          // Broche de données du capteur DHT22 (modifiable selon votre configuration)

#define DHTTYPE DHT22      // Définir le type de capteur (DHT22)

DHT dht(DHTPIN, DHTTYPE);   // Création d'un objet DHT



const int pirPin = 32;      // Définir la broche du capteur PIR (utiliser une broche valide)



// Paramètres WiFi

const char* id = "dlink-4E56";
```

```
const char* mdp = "fvrnf37378";



// Paramètres MQTT
const char* IP_MQTT = "192.168.0.108";  // Adresse IP du broker MQTT
const int port_mqtt = 1883;


const char* mqtt_topic_ldr = "obscur";         // Sujet sur lequel publier les données LDR
const char* mqtt_topic_hum = "humidite";         // Sujet sur lequel publier les données d'humidité
const char* mqtt_topic_ppm = "co2";         // Sujet sur lequel publier les données de qualité de l'air
const char* mqtt_topic_temp = "temperature";       // Sujet sur lequel publier les données de température
const char* mqtt_topic_pres = "capteur";       // Sujet sur lequel publier les données de présence

WiFiClient espClient;
PubSubClient client(espClient);



void setup() {
  // Initialiser la communication série à 115200 bits par seconde
  Serial.begin(115200);

  // Connectez-vous au réseau WiFi
  setup_wifi();

  // Configurer le client MQTT
```

```
  client.setServer(IP_MQTT, port_mqtt);

  // Initialisation du capteur DHT
  dht.begin();

  // Configurer la broche du capteur PIR comme entrée
  pinMode(pirPin, INPUT);
}

void setup_wifi() {
  delay(10);
  // Démarrer la connexion WiFi
  Serial.println();
  Serial.print("Connexion au réseau ");
  Serial.println(id);

  WiFi.begin(id, mdp);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connecté");
  Serial.println("Adresse IP : ");
```

```
  Serial.println(WiFi.localIP());
}




void loop() {
  if (!client.connected()) {
    reconnect();
  }
//  client.loop();



  // Lire la valeur de l'entrée analogique
  int Value = analogRead(obscur);


                                                  // Afficher la valeur lue sur le moniteur série
  Serial.print("LDR Value: ");
  Serial.println(Value);


                                                 // Déterminer la valeur à envoyer
  const char* mqtt_value_ldr;
  if (Value < 3000) {
    mqtt_value_ldr = "1";
    Serial.println("Il fait jour");
  } else {
```

```
  mqtt_value_ldr = "0";
  Serial.println("Il fait nuit");
}

                                                    // Publier la valeur sur le serveur MQTT
client.publish(mqtt_topic_ldr, mqtt_value_ldr);



float co2 = gasSensor.getPPM();
Serial.print("CO2 concentration in ppm: ");
Serial.println(co2);

                                                     // Convertir la valeur en chaîne de caractères
char ppmStr[8];
dtostrf(co2, 1, 2, ppmStr);

                                                   // Publier la valeur de qualité de l'air sur le serveur MQTT
client.publish(mqtt_topic_ppm, ppmStr);



                                                  // Lecture de l'humidité relative
float humidite = dht.readHumidity();

                                                       // Vérifier si la lecture est valide
if (!isnan(humidite)) {
                                                     // Convertir la valeur en chaîne de caractères
```

```
  char humStr[8];
  dtostrf(humidite, 1, 2, humStr);

                                                  // Publier la valeur de l'humidité sur le serveur MQTT
  client.publish(mqtt_topic_hum, humStr);
 } else {
  Serial.println("Échec de lecture de l'humidité");
 }



                                                      // Lecture de la température
 float temperature = dht.readTemperature();

                                                  // Convertir la valeur en chaîne de caractères
  char tempStr[8];
  dtostrf(temperature, 1, 2, tempStr);

                                                          // Publier la valeur de la température sur
le serveur MQTT
  client.publish(mqtt_topic_temp, tempStr);



                                                      // Lecture de la présence
 int capteur = digitalRead(pirPin);
```

```
                                                    // Convertir la valeur en chaîne de caractères
  char presStr[2];
  itoa(capteur, presStr, 10);


                                                  // Publier la valeur de la présence sur le serveur MQTT
  client.publish(mqtt_topic_pres, presStr);


  delay(1000);
}




void reconnect() {
  while (!client.connected()) {  // Boucle jusqu'à la reconnexion
       Serial.print("En attente de connexion, veuillez patienter...");


     if (client.connect("ESP32Client")) {  // Tentative de connexion MQTT
     Serial.println("connecté!!!");
     }
     else {
     Serial.print("Ca marche pas !!!");
     Serial.print(client.state());
     Serial.println(" try again in 5 seconds");
     delay(5000);
    }
```

```
  }

}
```

Conclusion du Projet CAPTEUR

Le projet CAPTEUR a permis de développer une sonde multicapteur connectée via WiFi pour mesurer la température, l'humidité, la luminosité et la détection de mouvement. Voici les étapes clés et leurs réalisations :

1. Installation de Raspbian : Mise en place du système d'exploitation sur le Raspberry Pi pour servir de base à l'ensemble du projet.
2. Configuration du Serveur MQTT : Installation et sécurisation de Mosquitto sur le Raspberry Pi pour gérer la communication entre les capteurs et le serveur.
3. Mise en Place des Capteurs sur l'ESP32 : Connexion et programmation des capteurs de température, d'humidité, de luminosité et de mouvement pour envoyer des données via MQTT.
4. Stockage des Données avec InfluxDb : Installation et configuration d'InfluxDb pour stocker les données des capteurs de manière organisée et efficace.
5. Visualisation des Données avec Node-Red : Création d'une interface web intuitive pour surveiller les données en temps réel.

Ce projet a démontré comment utiliser des technologies modernes pour créer un système de surveillance environnementale efficace et extensible. En intégrant l'ESP32, MQTT, InfluxDb et Node-Red, ainsi que plusieurs capteurs. Par exemple, une telle sonde peut être utilisée dans une maison pour surveiller et contrôler le chauffage et la climatisation en fonction de la température et de l'humidité ambiantes, allumer les lumières lorsqu'un mouvement est détecté ou ajuster l'éclairage en fonction de la luminosité naturelle, assurant ainsi un confort optimal tout en réalisant des économies d'énergie.