

Projet simulation Wokwi et un Broker mqtt

Utilisant Node Red

Département génie informatique

REALISER PAR :
MALIKA CHAABAN
AYA QUARIN

Introduction :

Le projet consiste à utiliser la plateforme Wokwi pour simuler un microcontrôleur ESP8266, en utilisant le protocole MQTT pour la communication avec un broker, et Node-RED pour la visualisation et la gestion des données.

Outils Utilisés :

Plateforme Wokwi:

- Plateforme en ligne pour le développement et l'apprentissage de l'électronique et de l'Internet des objets (IoT).
- Offre des simulateurs d'Arduino et de circuits électroniques, avec une bibliothèque de composants électroniques virtuels.

MQTT Broker:

- Un serveur qui gère la communication entre les appareils utilisant le protocole MQTT.
- Utilisé pour la transmission de données entre des appareils connectés, basé sur un modèle de publication/abonnement.

Node-RED:

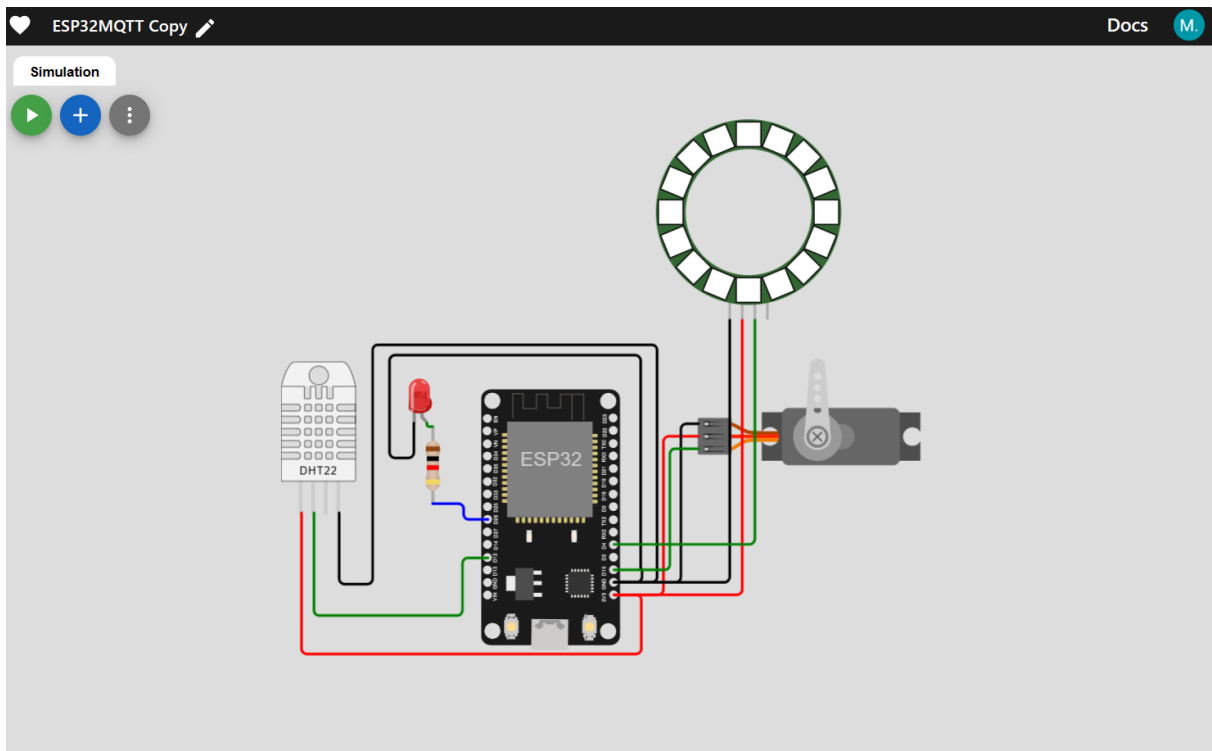
- Environnement de développement visuel open source pour le développement rapide d'applications IoT.
- Permet de câbler ensemble des nœuds préfabriqués pour créer des flux de données et des applications.



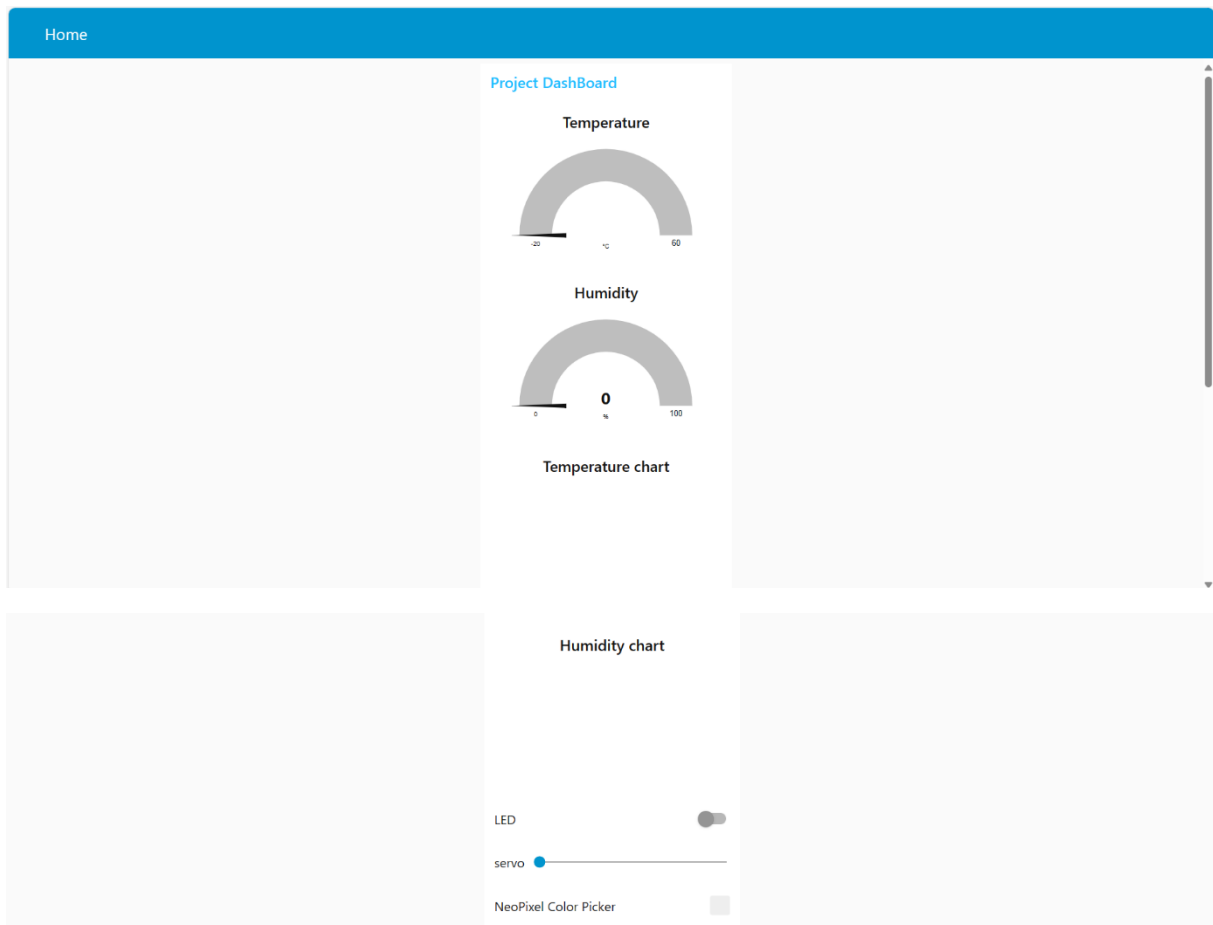
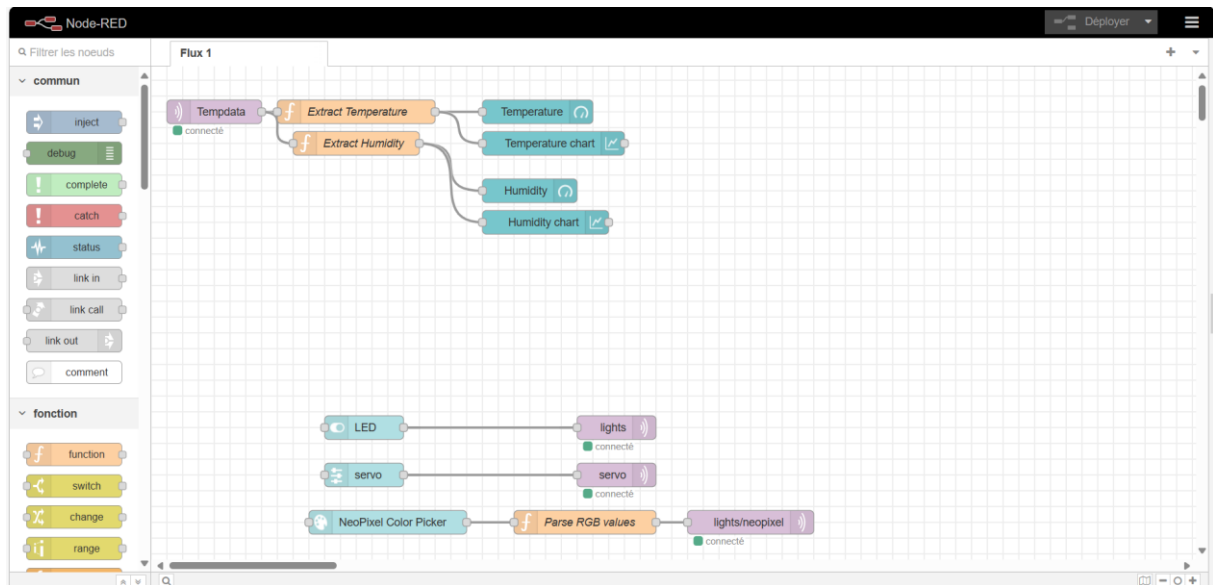
Réalisation :

Schéma Wokwi:

- Utilisation du simulateur Wokwi avec un microcontrôleur ESP8266 pour simuler le matériel.







Dashboard Node-RED:




- Configuration des nœuds MQTT pour la réception et la publication des données.


Modifier le noeud mqtt in

 **Propriétés**




 Serveur


mosquitto




Action

S'abonner à un seul sujet





 Sujet

Tempdata


 QoS


2




 Sortie

détection automatique (objet JSON analysé, chaîne ou



 Nom

Nom



- Gestion des informations provenant du capteur DHT22, de l'allumage de la lampe, de la rotation du servo et de la couleur de la neopixel.

Modifier le noeud mqtt out

Supprimer

Annuler

Terminer

⚙️ Propriétés



🌐 Serveur	mosquitto ▼	
📄 Sujet	lights	
⚙️ QoS	1 ▼	🔄 Conserver
🏷️ Nom	Nom	

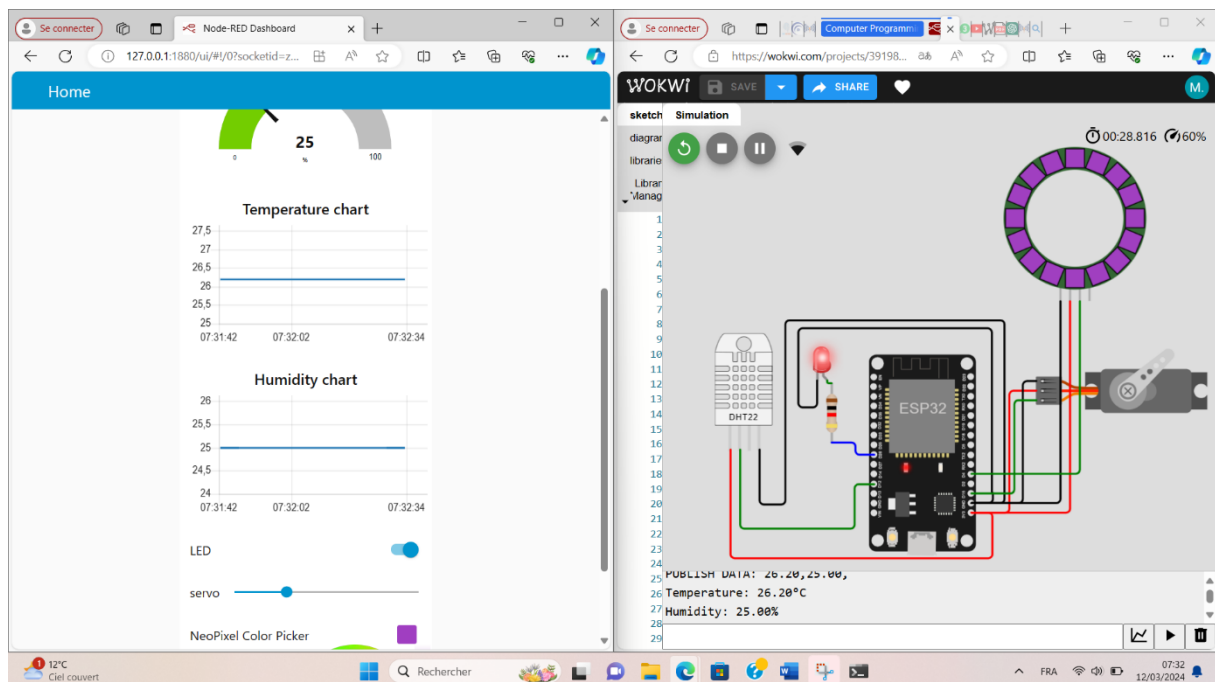
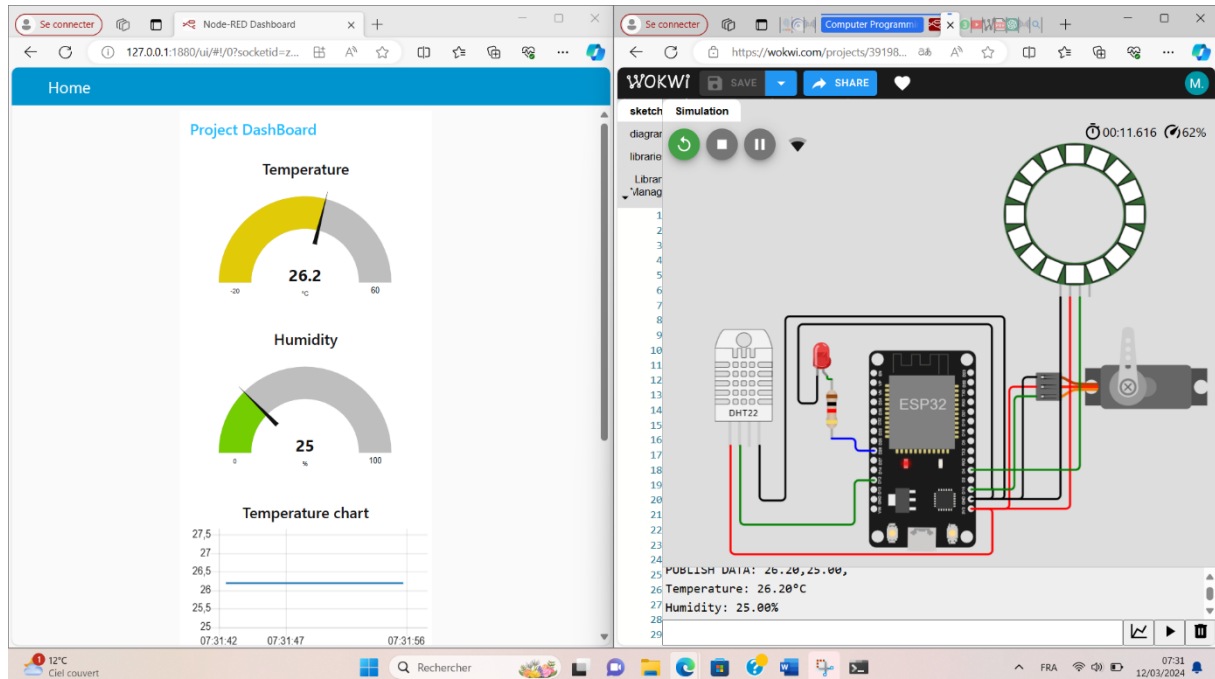
Conseil : laisser le sujet, le qos ou le contenu vide si vous souhaitez les définir via les propriétés du msg.



○ Activé

Test :

- Simulation des interactions avec le matériel simulé sur Wokwi et les nœuds MQTT configurés sur Node-RED.



Code Utilisé :

- Utilisation du langage Arduino (C++) pour programmer le microcontrôleur ESP8266.
- Configuration des paramètres WiFi et MQTT pour la communication avec le broker.
- Gestion des messages reçus par les nœuds MQTT, contrôlant l'allumage de la lampe, la rotation du servo et la couleur de la neopixel.

```
#include <Adafruit_Sensor.h>
#include <DHT_U.h>
#include <WiFi.h>
#include <PubSubClient.h>
#include <ESP32Servo.h> // Include the Servo library
#include <FastLED.h> // Include the FastLED library

// Defining Pins
#define DHTPIN 12
#define LED 26
#define SERVO_PIN 15 // Servo motor pin
#define LED_PIN 4 // WS2812 LED strip pin
#define NUM_LEDS 16 // Number of LEDs in the strip

// DHT parameters
#define DHTTYPE DHT22 // DHT 11
DHT_Unified dht(DHTPIN, DHTTYPE);
uint32_t delayMS;

// Servo motor
Servo servo;

// WS2812 LED strip
CRGB leds[NUM_LEDS];

// MQTT Credentials
const char* ssid = "Wokwi-GUEST"; // Setting your AP SSID
const char* password = ""; // Setting your AP PSK
const char* mqttServer = "test.mosquitto.org";
const char* clientId = "M1"; // Client ID username+0001
const char* topic = "Tempdata"; // Publish topic
```

```
// Parameters for using non-blocking delay
unsigned long previousMillis = 0;
const long interval = 1000;
String msgStr = ""; // MQTT message buffer
float temp, hum;

// Setting up WiFi and MQTT client
WiFiClient espClient;
PubSubClient client(espClient);

void setup_wifi() {
    delay(10);
    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 reconnect() {
    while (!client.connected()) {
        if (client.connect(clientID)) {
            Serial.println("MQTT connected");
            client.subscribe("lights");
            client.subscribe("servo"); // Subscribe to servo topic
            client.subscribe("lights/neopixel"); // Subscribe to neopixel topic
            Serial.println("Topic Subscribed");
        }
        else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000); // wait 5sec and retry
        }
    }
}

// Subscribe callback
void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived in topic: ");
    Serial.println(topic);
    Serial.print("Message: ");
```

```
String data = "";
for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
    data += (char)payload[i];
}
Serial.println();
Serial.print("Message size: ");
Serial.println(length);
Serial.println();
Serial.println("-----");
Serial.println(data);

if (String(topic) == "lights") {
    if (data == "ON") {
        Serial.println("LED");
        digitalWrite(LED, HIGH);
    }
    else {
        digitalWrite(LED, LOW);
    }
}
else if (String(topic) == "servo") {
    int degree = data.toInt(); // Convert the received data to an integer
    Serial.print("Moving servo to degree: ");
    Serial.println(degree);
    servo.write(degree); // Move the servo to the specified degree
}
else if (String(topic) == "lights/neopixel") {
    int red, green, blue;
    sscanf(data.c_str(), "%d,%d,%d", &red, &green, &blue); // Parse the
received data into RGB values
    Serial.print("Setting NeoPixel color to (R,G,B): ");
    Serial.print(red);
    Serial.print(",");
    Serial.print(green);
    Serial.print(",");
    Serial.println(blue);
    fill_solid(leds, NUM_LEDS, CRGB(red, green, blue)); // Set all LEDs in the
strip to the specified color
    FastLED.show(); // Update the LED strip with the new color
    fill_solid(leds, NUM_LEDS, CRGB(red, green, blue));
    FastLED.show();
}
}

void setup() {
    Serial.begin(115200);
```

```
// Initialize device.
dht.begin();
// Get temperature sensor details.
sensor_t sensor;
dht.temperature().getSensor(&sensor);
dht.humidity().getSensor(&sensor);
pinMode(LED, OUTPUT);
digitalWrite(LED, LOW);

// Setup servo
servo.attach(SERVO_PIN, 500, 2400);
servo.write(0);

// Setup WS2812 LED strip
FastLED.addLeds<WS2812, LED_PIN, GRB>(leds, NUM_LEDS);

setup_wifi();
client.setServer(mqttServer, 1883); // Setting MQTT server
client.setCallback(callback); // Define function which will be called when a
message is received.
}

void loop() {
    if (!client.connected()) { // If client is not connected
        reconnect(); // Try to reconnect
    }
    client.loop();
    unsigned long currentMillis = millis(); // Read current time
    if (currentMillis - previousMillis >= interval) { // If current time - last
time > 5 sec
        previousMillis = currentMillis;
        // Read temperature and humidity
        sensors_event_t event;
        dht.temperature().getEvent(&event);
        if (isnan(event.temperature)) {
            Serial.println(F("Error reading temperature!"));
        }
        else {
            Serial.print(F("Temperature: "));
            temp = event.temperature;
            Serial.print(temp);
            Serial.println(F("°C"));
        }
        // Get humidity event and print its value
        dht.humidity().getEvent(&event);
        if (isnan(event.relative_humidity)) {
            Serial.println(F("Error reading humidity!"));
        }
    }
}
```

```
}  
else {  
    Serial.print(F("Humidity: "));  
    hum = event.relative_humidity;  
    Serial.print(hum);  
    Serial.println(F("%"));  
}  
msgStr = String(temp) + "," + String(hum) + ",";  
byte arrSize = msgStr.length() + 1;  
char msg[arrSize];  
Serial.print("PUBLISH DATA: ");  
Serial.println(msgStr);  
msgStr.toCharArray(msg, arrSize);  
client.publish(topic, msg);  
msgStr = "";  
delay(1);  
}  
}
```

Conclusion :

Le projet a permis de simuler efficacement un environnement IoT en utilisant la plateforme Wokwi, le protocole MQTT pour la communication et Node-RED pour la visualisation et la gestion des données. La configuration des nœuds MQTT a été essentielle pour permettre une communication bidirectionnelle entre le microcontrôleur simulé et l'interface