_3Ministerul Educaţiei, Culturii și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Departamentul Ingineria Software și Automatică

RAPORT

Lucrare de laborator Nr.7_3
Disciplina: IoT
Tema: Comunicare Internet - MQTT

A efectuat: st.gr.TI-212,
Muntean Mihai

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Sarcina:

Să se realizeze o aplicație ce va implementa comunicațiile între echipamente după cum urmează:

- realizarea unei aplicații de comunicare Internet prin protocolul MQTT pentru interacțiunea cu o resursă Cloud;
- colectarea datelor de la senzori și trimiterea către un broker MQTT;
- urmărirea mesajelor de la un broker MQTT și setarea stării unui actuator la alegere;
- datele sunt vizualizate și controlate de la un dashboard Internet (ThingsBoard sau HiveMQ).

Recomandări:

- reutilizați la maxim soluțiile prezentate în laboratoarele precedente;
- a se utiliza ESP32 în calitate de MCU;
- urmați tutoriale existente pentru realizarea conexiunii cu serverul MQTT:
  - https://thingsboard.io/docs/samples/esp32/gpio-control-pico-kit-dht22-sensor/
  - https://www.survivingwithandroid.com/esp32-mqtt-client-publish-and-subscribe/
- revizuiți resursele predate la curs.

Pontaj:

- **nota 5:** simplă aplicație de comunicare;
- **+1.0:** pentru implementarea modulară a proiectului;
- **+1.0:** MCU (ESP32) trimite datele către broker MQTT;
- **+1.0:** MCU (ESP32) primește datele de la server MQTT;
- **+1.0:** datele sunt vizualizate și controlate de la un dashboard Internet (ThingsBoard sau HiveMQ);
- **+1.0:** pentru demonstrarea probelor de implementare fizică.

NOTA: Pontaj maxim posibil doar la prezentarea funcționării fizice!

Penalități:

- **-1:** penalizare pentru fiecare săptămână întârziere de la deadline;
- **-1:** penalizare pentru nerespectarea formatului raportului.

Materiale necesare:

- ESPN32;
- Senzor DHT22;
- Releu;
- Aplicatia MQTT client;
- Cabluri pentru conectare.

INTRODUCERE

Protocolul MQTT (Message Queuing Telemetry Transport) reprezintă un standard de comunicație ușor, eficient și de înaltă performanță, destinat pentru schimbul de mesaje între dispozitive într-un mediu distribuit figura 1, de obicei cu resurse limitate, cum sunt dispozitivele Internet of Things (IoT). Acesta se bazează pe arhitectura client-server, în care dispozitivele IoT (denumite clienți) trimit mesaje printr-un broker MQTT care le distribuie către alți clienți sau spre resurse externe, precum platformele cloud. MQTT este optimizat pentru rețele cu lățime de bandă redusă, latență mare și condiții de conectivitate instabilă, fiind astfel ideal pentru implementarea în scenarii IoT și pentru interacțiunea cu resurse cloud.

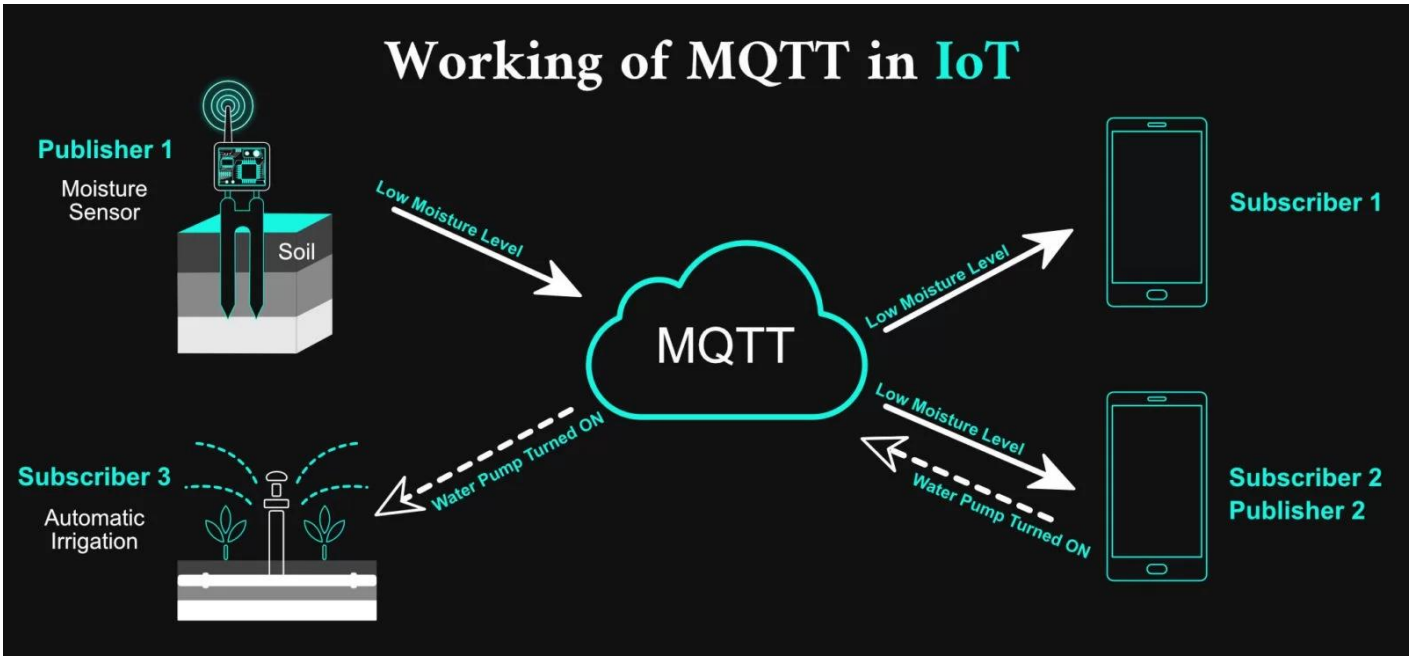


Figura 1 – Arhitectura protocolului MQTT

Mersul lucrării

Circuitul virtual asamblat utilizează o placă de dezvoltare ESP32 DevKit C V4, un senzor de temperatură și umiditate DHT22 și un modul de releu figura2. Aceste componente sunt conectate conform următoarei configurații:

1. **Placa ESP32** este componenta centrală a circuitului, responsabilă pentru procesarea datelor și gestionarea comunicațiilor.
2. **Senzorul DHT22**:
   - Pinul de alimentare (**VCC**) al senzorului este conectat la pinul **5V** al plăcii ESP32.
   - Pinul de date (**SDA**) al senzorului este conectat la pinul **GPIO4** al ESP32.
   - Pinul de masă (**GND**) este conectat la unul dintre pinii de masă (**GND**) ai plăcii ESP32.
3. **Modulul de releu**:
   - Pinul de alimentare (**VCC**) este conectat la pinul **5V** al ESP32.
   - Pinul de control (**IN**) este conectat la pinul **GPIO5** al ESP32 pentru a permite activarea și dezactivarea releului.
   - Pinul de masă (**GND**) este conectat la un pin **GND** al ESP32.

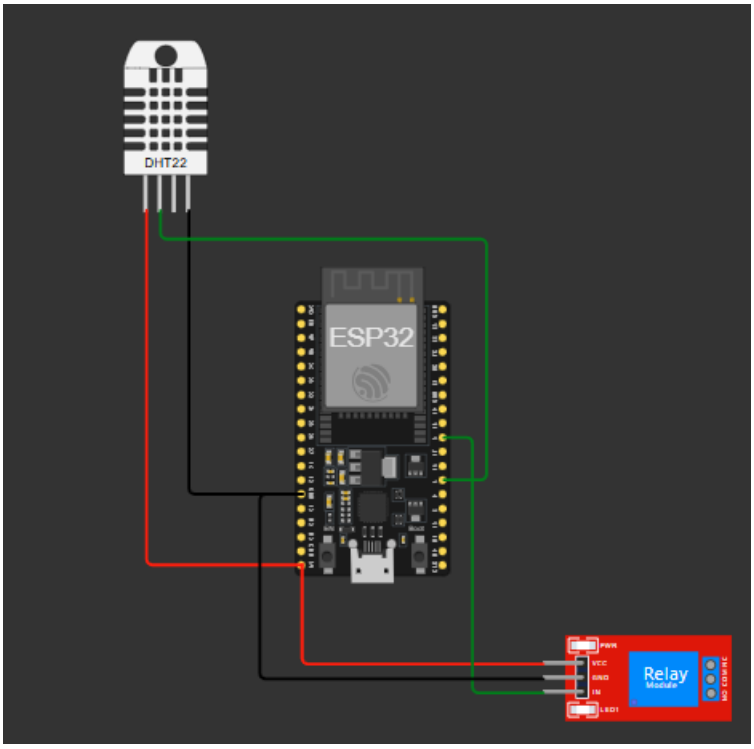


Figura 2 – Asamblarea circuitului virtual

1 Include-uri

```
#include "wifi_manager.h"
#include "mqtt_manager.h"
#include "ntp_manager.h"
#include "dht_sensor.h"
#include "relay_control.h"
```

Se includ fișierele header pentru gestionarea modulelor:

- wifi_manager.h: Gestionarea conexiunii Wi-Fi.
- mqtt_manager.h: Gestionarea conexiunii și mesajelor MQTT.
- ntp_manager.h: Sincronizarea orei printr-un server NTP.
- dht_sensor.h: Citirea datelor de temperatură și umiditate de la un senzor DHT.
- relay_control.h: Controlul releului.

2 Funcția setup()

```
void setup() {
  Serial.begin(115200); // Inițializează portul serial pentru debugging.
  setup_wifi();         // Configurează și conectează dispozitivul la Wi-Fi.
  setDateTime();        // Sincronizează timpul utilizând serverul NTP.

  beginDHT();             // Inițializează senzorul DHT pentru citirea temperaturii și
umidității.
  setupRelay();         // Configurează pinurile pentru controlul releului.

  setup_mqtt();           // Inițializează clientul MQTT și configurează conexiunea cu
brokerul.
}
```

Configurează toate modulele necesare înainte de a intra în bucla principală (loop).

3 Verificarea conexiunii MQTT

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

Verifică dacă clientul MQTT este conectat la broker:

- Dacă nu este conectat, se încearcă reconectarea folosind reconnect_mqtt().
- client.loop() este necesar pentru procesarea evenimentelor MQTT, inclusiv recepționarea mesajelor.

4 Transmiterea periodică a datelor senzorului

```
static unsigned long lastMsg = 0; // Timpul ultimei transmiteri.
unsigned long now = millis();    // Timpul curent.

if (now - lastMsg > 2000) {      // Transmite la fiecare 2000 ms (2 secunde).
  lastMsg = now;

  float temperature = readTemperature(); // Citește temperatura de la senzorul DHT.
  float humidity = readHumidity();       // Citește umiditatea de la senzorul DHT.

  if (isnan(temperature) || isnan(humidity)) { // Verifică dacă citirile sunt valide.
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  String message = "Temperature: " + String(temperature, 1) + "°C, Humidity: " +
String(humidity, 1) + "%";
  publishMessage(publishTopic, message);  // Trimite mesajul către brokerul MQTT.
}
```

Transmite periodic valorile de temperatură și umiditate citite de la senzorul DHT către brokerul MQTT:

- Utilizează funcția millis() pentru a asigura transmiterea periodică.
- Mesajele sunt publicate pe un topic definit (publishTopic).

5 Funcția loop():

```
void loop() {
  // Verifică conexiunea MQTT și gestionează evenimentele.
  if (!client.connected()) {
    reconnect_mqtt();
  }
  client.loop();

  // Trimite datele senzorului periodic.
  static unsigned long lastMsg = 0;
  unsigned long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;

    float temperature = readTemperature();
    float humidity = readHumidity();

    if (isnan(temperature) || isnan(humidity)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }
```

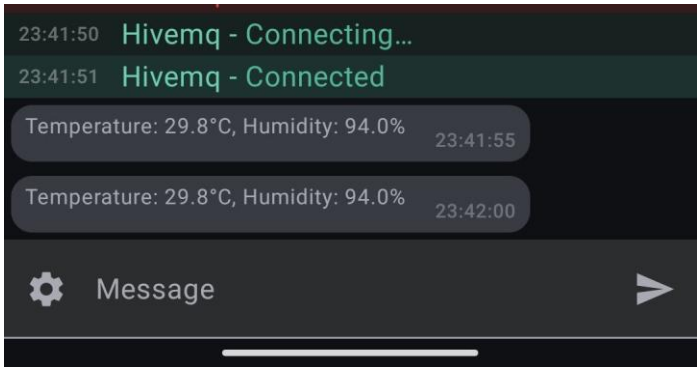
```
    String message = "Temperature: " + String(temperature, 1) + "°C, Humidity: " +
String(humidity, 1) + "%";
    publishMessage(publishTopic, message);
  }
}
```

Gestionează reconectarea MQTT, procesarea mesajelor și transmiterea periodică a datelor senzorului. Codul sursă pentru analiza desfășurată a implimentării soluției poate fi găsită în anexă.

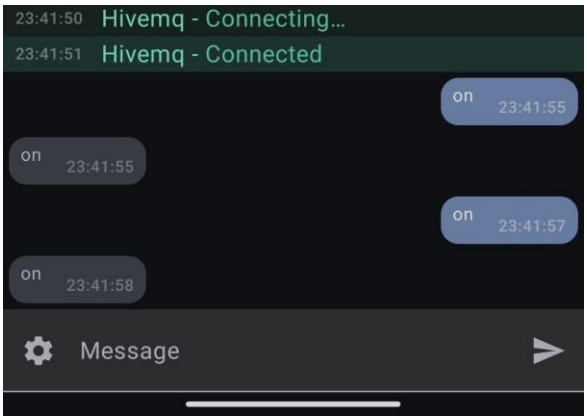
Rezultate:

```
Message published: Temperature: 29.8°C, Humidity: 94.0%
Attempting MQTT connection...connected
Message arrived [readData2025] on
Turning on relay
Attempting MQTT connection...connected
Message published: Temperature: 29.8°C, Humidity: 94.0%
```

a) Terminalul serial din wokwi



b) Clientul care ascultă



c) Clientul care publică

Figura 3 – Rezultatele utilizării MQTT

CONCLUZIE

Protocolul MQTT (Message Queuing Telemetry Transport) este o soluție extrem de eficientă și flexibilă pentru comunicația în rețele IoT (Internet of Things). Acesta se remarcă prin următoarele avantaje:

1. Eficiență ridicată - utilizează o lățime de bandă redusă, ceea ce îl face ideal pentru dispozitive cu resurse limitate sau rețele instabile.
2. Comunicare asincronă - MQTT decuplează expeditorii și destinatarii prin intermediul unui broker, permițând transferul de mesaje în timp real fără a necesita o conexiune directă permanentă între dispozitive.
3. Modelul publicare-abonare - acest model facilitează scalabilitatea, permitând unui dispozitiv să publice informații unei mulțimi de abonați interesați fără a ține cont de locația sau starea acestora.
4. Fiabilitate configurabilă - oferă trei nivele de calitate a serviciului (QoS), adaptabile în funcție de cerințele aplicației, ceea ce asigură că mesajele sunt livrate cu succes.
5. Securitate - protocolul poate fi combinat cu standarde de criptare precum TLS pentru a proteja datele transmise și pentru a asigura autentificarea dispozitivelor.

MQTT este adesea preferat în proiectele IoT datorită adaptabilității sale la o gamă largă de aplicații, de la automatizări industriale la case inteligente și monitorizarea vehiculelor. Prin intermediul unui broker MQTT, dispozitivele pot comunica eficient, iar integrarea cu platforme precum HiveMQ, ThingsBoard sau alte aplicații MQTT client, oferă un control centralizat și vizualizări avansate ale datelor colectate.

În concluzie, MQTT reprezintă o alegere optimă pentru sisteme distribuite care necesită o comunicare rapidă, sigură și eficientă între dispozitive.

BIBLIOGRAFII

1. Resursa electronică: https://docs.wokwi.com/?utm_source=wokwi – Regim de acces;
2. Resursa electronică: https://else.fcim.utm.md/course/view.php?id=343 – Regim de acces;
3. Resursa electronică: https://forum.arduino.cc/t/serial-print-and-printf/146256/14 - Regim de acces;
4. Proiectul pe GitHub, Resursa electronică: https://github.com/MunMihai/Anul4/tree/8c7af8c43b8c728ff28e1ee6c75a63f997659015/Semestrul_7/Internetul_Lucrurilor%20(IoT)/Laborator/Laborator7 - Regim de acces;

Anexa 1

Fișierul stdinout.h

```
#ifndef _STDINOUT_H
#define _STDINOUT_H

// no need to make an instance of this yourself
class initializeSTDINOUT
{
        static size_t initnum;
public:
        // Constructor
        initializeSTDINOUT();
};

// Call the constructor in each compiled file this header is included in
// static means the names won't collide
static initializeSTDINOUT initializeSTDINOUT_obj;

#endif
```

Fișierul stdinout.cpp

```
#if ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
#include <stdio.h>
#include "stdinout.h"

// Function that printf and related will use to print
static int serial_putchar(char c, FILE *f)
{
  if (c == '\n') {
    serial_putchar('\r', f);
  }

  return Serial.write(c) == 1 ? 0 : 1;
}
// Function that scanf and related will use to read
static int serial_getchar(FILE *)
{
  // Wait until character is avilable
  while (Serial.available() <= 0) { ; }

  return Serial.read();
}

static FILE serial_stdinout;

static void setup_stdin_stdout()
```

```
{
  // Set up stdout and stdin
  fdev_setup_stream(&serial_stdinout, serial_putchar, serial_getchar, _FDEV_SETUP_RW);
  stdout = &serial_stdinout;
  stdin  = &serial_stdinout;
  stderr = &serial_stdinout;
}

// Initialize the static variable to 0
size_t initializeSTDINOUT::initnum = 0;

// Constructor that calls the function to set up stdin and stdout
initializeSTDINOUT::initializeSTDINOUT()
{
  if (initnum++ == 0) {
    setup_stdin_stdout();
  }
}
```

Anexa 2

sketch.ino

```
#include "wifi_manager.h"
#include "mqtt_manager.h"
#include "ntp_manager.h"
#include "dht_sensor.h"
#include "relay_control.h"

void setup() {
  Serial.begin(115200);
  setup_wifi();
  setDateTime();

  beginDHT();
  setupRelay();

  setup_mqtt();
}

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

  static unsigned long lastMsg = 0;
  unsigned long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;

    float temperature = readTemperature();
    float humidity = readHumidity();
```

```
    if (isnan(temperature) || isnan(humidity)) {
      Serial.println("Failed to read from DHT sensor!");
      return;
    }

     String message = "Temperature: " + String(temperature, 1) + "°C, Humidity: " +
String(humidity, 1) + "%";
    publishMessage(publishTopic, message);  // Funcție din mqtt_manager
  }
}
```

ntp_manager.h

```
#ifndef NTP_MANAGER_H
#define NTP_MANAGER_H

void setDateTime();

#endif
```

wifi_manager.h

```
#ifndef WIFI_MANAGER_H
#define WIFI_MANAGER_H
#include <WiFi.h>

void setup_wifi();
extern WiFiClient espClient;

#endif
```

relay_control.h

```
#ifndef RELAY_CONTROL_H
#define RELAY_CONTROL_H

// Relay pin
#define RELAYPIN 5

// Function declarations for relay control
void setupRelay();
void turnOnRelay();
void turnOffRelay();

#endif // RELAY_CONTROL_H
```

mqtt_manager.h

```
#ifndef MQTT_MANAGER_H
```

```
#define MQTT_MANAGER_H
#include <WiFi.h>  // Necesită WiFiClient
#include <PubSubClient.h>

void setup_mqtt();
void mqtt_callback(char* topic, byte* payload, unsigned int length);
void reconnect_mqtt();
void publishMessage(const char* topic, const String& message);  // Adăugați această linie

extern PubSubClient client;
extern const char* publishTopic;
extern const char* readTopic;

#endif
```

dht_sensor.h

```
#ifndef DHT_SENSOR_H
#define DHT_SENSOR_H

// DHT sensor setup
#define DHTPIN 4
#define DHTTYPE DHT22

float readTemperature();
float readHumidity();
void beginDHT();

#endif // DHT_SENSOR_H
```

mqtt_manager.cpp

```
#include <PubSubClient.h>
#include "mqtt_manager.h"
#include "wifi_manager.h"
#include "relay_control.h"

const char* publishTopic = "sentDHTData2025";
const char* readTopic = "readData2025";

const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;
String buffer = "";

PubSubClient client(espClient);

void setup_mqtt() {
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(mqtt_callback);
}
```

```
void mqtt_callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");

  String message = "";

  // Construirea mesajului din payload
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  Serial.println(message);

  // Dacă mesajul este diferit față de cel din buffer, actualizăm buffer-ul și procesăm
mesajul
  if (message != buffer) {
    buffer = message;  // Actualizează buffer-ul

    if (String(topic) == readTopic) {
      if (strcasecmp(message.c_str(), "on") == 0) {
        Serial.println("Turning on relay");
        turnOnRelay();  // Funcția care activează releul
      } else if (strcasecmp(message.c_str(), "off") == 0) {
        Serial.println("Turning off relay");
        turnOffRelay();  // Funcția care dezactivează releul
      } else {
        Serial.println("Relay save state!");
      }
    }
  }
}

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

void publishMessage(const char* topic, const String& message) {
  if (client.connected()) {
    client.publish(topic, message.c_str());
```

```
    Serial.println("Message published: " + message);
  } else {
    Serial.println("MQTT client not connected. Cannot publish message.");
  }
}
```

ntp_manager.cpp

```
#include <time.h>  // Pentru funcțiile legate de timp
#include <Arduino.h>  // Pentru funcțiile Serial
#include "ntp_manager.h"

const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 7200;
const int daylightOffset_sec = 3600;

void setDateTime() {
  configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo)) {
    Serial.println("Failed to obtain time");
    return;
  }
  Serial.println(&timeinfo, "%A, %B %d %Y %H:%M:%S");
}
```

wifi_manager.cpp

```
#include <WiFi.h>
#include "wifi_manager.h"

const char* ssid = "Wokwi-GUEST";
const char* password = "";
WiFiClient espClient;

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

relay_control.cpp

```
#include <Arduino.h>
#include "relay_control.h"

// Function to initialize relay
void setupRelay() {
  pinMode(RELAYPIN, OUTPUT);
  digitalWrite(RELAYPIN, LOW);  // Ensure relay is OFF initially
}

// Function to turn on the relay
void turnOnRelay() {
  digitalWrite(RELAYPIN, HIGH);  // Turn on the relay
}

// Function to turn off the relay
void turnOffRelay() {
  digitalWrite(RELAYPIN, LOW);  // Turn off the relay
}
```

dht_sensor.cpp

```
#include <DHT.h>
#include "dht_sensor.h"

DHT dht(DHTPIN, DHTTYPE);

// Function to initialize DHT sensor
void beginDHT() {
  dht.begin();
}

// Function to read temperature from DHT sensor
float readTemperature() {
  return dht.readTemperature();
}

// Function to read humidity from DHT sensor
float readHumidity() {
  return dht.readHumidity();
}
```

diagram.json

```
{
  "version": 1,
  "author": "Mihai Muntean",
  "editor": "wokwi",
```

```
  "parts": [
    { "type": "board-esp32-devkit-c-v4", "id": "esp", "top": 0, "left": -4.76, "attrs":
{} },
    { "type": "wokwi-dht22", "id": "dht1", "top": -153.3, "left": -120.6, "attrs": {} },
    { "type": "wokwi-relay-module", "id": "relay1", "top": 249.8, "left": 163.2, "attrs":
{} }
  ],
  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [] ],
    [ "esp:RX", "$serialMonitor:TX", "", [] ],
    [ "dht1:GND", "esp:GND.1", "black", [ "v0" ] ],
    [ "esp:5V", "dht1:VCC", "red", [ "h0" ] ],
    [ "esp:4", "dht1:SDA", "green", [ "h28.8", "v-163.2", "h-220.9" ] ],
    [ "relay1:VCC", "esp:5V", "red", [ "h0" ] ],
    [ "relay1:IN", "esp:5", "green", [ "h-48", "v-173" ] ],
    [ "relay1:GND", "esp:GND.1", "black", [ "h-192", "v-125.2" ] ]
  ],
  "dependencies": {}
}
```