Ministerul Educației, 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,
Vlasitchi Stefan

A verificat : asist. univ.
Lupan Cristian

Chișinău 2024

Definirea problemei:

Sa se realizeze o aplicatie ce va implementa comunicatiile intre echipamente dupa cum urmeaza:

1. Realizarea unei aplicatii decomunicare Internet prin protocolul MQTT penru interactiunea cu o resursa Cloud
2. Colectarea datelor de la sensori si trimitere catre un broker MQTT
3. urmarirea mesajelor de la un proker MQTT si setarea starii unui actuator la alegere
4. Datele sunt vizualizate si controlate de la un dashboard Internet *(ThingsBoard sau HiveMQ)*

Materiale necesare:

- ESPN32;
- Senzor DHT22;
- Releu;
- Aplicatian MQTT Explorer;
- Cabluri pentru conectare.

INTRODUCERE

În lumea Internet of Things (IoT), principala monedă o reprezintă mesajele. Chiar acum, în timp ce citiți acest articol, sute de mii de dispozitive trimit rapoarte de stare, primesc comenzi (cum ar fi „măriți temperatura", „măsurați umiditatea camerei" etc.), stochează date de pe alte dispozitive sau fac numeroase alte sarcini care ne fac viața mai ușoară. Și toate aceste dispozitive au nevoie să facă schimb de informații, deoarece fără conectivitate ele devin inutile.

Dar, odată cu creșterea rapidă a numărului de dispozitive care necesită Internet pentru a comunica și pentru a fi utile persoanelor, companiilor sau guvernelor, apar preocupări legate de păstrarea lățimii de bandă, asigurarea faptului că mașinile cu conectivitate redusă pot de asemenea transmite date și chiar de mărirea duratei de viață a bateriei gadgeturilor.

MQTT înseamnă „Message Queuing Telemetry Transport" și reprezintă de facto protocolul standard pentru transmiterea mesajelor IoT. Standardizat de OASIS și ISO, protocolul de publish / subscribe MQTT oferă o modalitate scalabilă și fiabilă de a conecta dispozitive prin Internet, care poate fi susținută chiar și de dispozitive mici din destinații la distanță, cu conectivitate scăzută.

MQTT este un protocol de tip open source, machine-to-machine (M2M) care devine din ce în ce mai mult standardul industriei pentru comunicarea dispozitivelor IoT. Conceput pentru a sparge decalajul dintre gadgeturile din lumea fizică (telefoane, senzori, dispozitive portabile etc.) cu tehnologii de procesare software, acest cod este unul dintre principalele motive pentru creșterea așteptată cu 204,41% a dispozitivelor conectate IoT până în 2025.

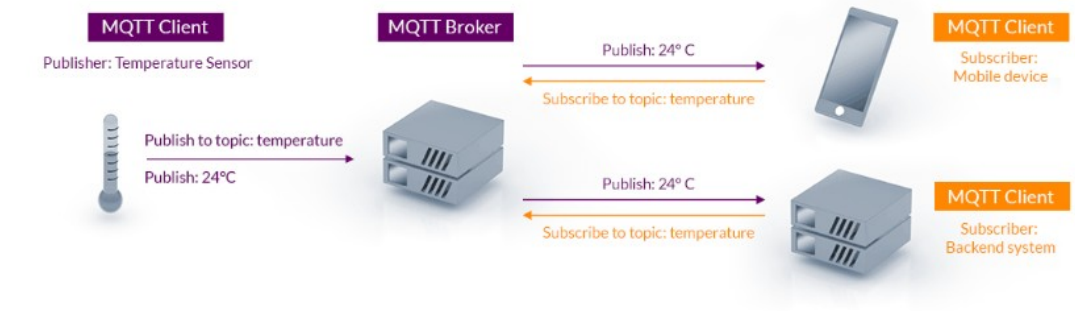

Figura 1 – Arhitectura de publicare MQTT.

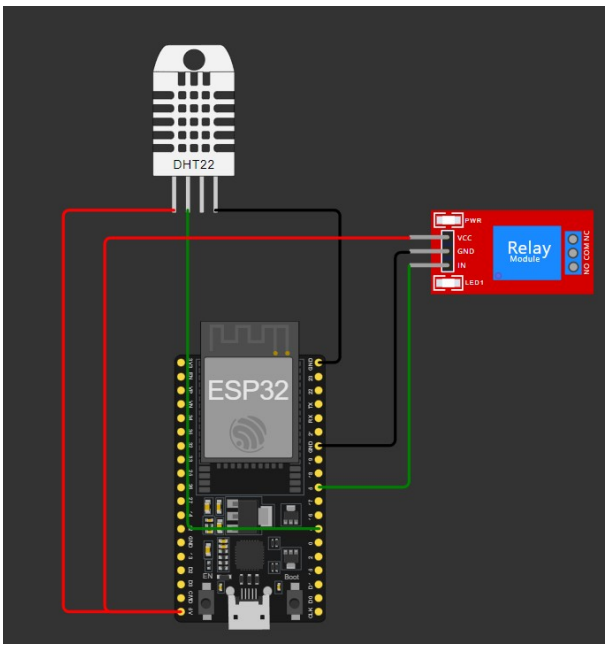

Figura 2 – Circuitul virtual

ESP32 se conectează la o rețea WiFi utilizând numele rețelei (SSID) și parola specificate. Acest lucru se realizează prin funcția setup_wifi(), care verifică conexiunea și afișează adresa IP a dispozitivului.

Fragment de cod:

```
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());
}
```

Fragment de cod:

Senzorul DHT22 măsoară temperatura și umiditatea. Datele colectate sunt trimise către un broker MQTT pe topicul "experimental0101". Acest proces are loc periodic, la fiecare două secunde, în funcția principală loop().

Fragment de cod:

```
if (now - lastMsg > 2000) {
  lastMsg = now;

  float temperature = dht.readTemperature();
  float humidity = dht.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) + "%";
  client.publish("experimental0101", message.c_str());
  Serial.println("Published message: " + message);
}
```

ESP32 ascultă mesaje pe topicul "67nd398b". La primirea unui mesaj, releul este activat pentru 5 secunde, iar apoi este dezactivat.

Fragment de cod:

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

  if (String(topic) == "67nd398b") {
    Serial.println("Turning on relay");
    digitalWrite(RELAYPIN, HIGH);  // Activare releu
    delay(5000);                   // Menținere activ pentru 5 secunde
    digitalWrite(RELAYPIN, LOW);  // Dezactivare releu
    Serial.println("Relay turned off");
  }
}
```

Dacă ESP32 pierde conexiunea la brokerul MQTT, încearcă să se reconecteze în mod repetat până când reușește.

Fragment de cod:

```
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32Client";
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      client.subscribe("67nd398b");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

```
Connecting to Wokwi-GUEST
..
WiFi connected
IP address:
10.10.0.2
Friday, December 06 2024 13:04:39
Attempting MQTT connection...connected
Published message: Temperature: 24.0°C, Humidity: 40.0%
Attempting MQTT connection...connected
Published message: Temperature: 24.0°C, Humidity: 40.0%
Attempting MQTT connection...connected
Attempting MQTT connection...failed, rc=-4 try again in 5 seconds
Attempting MQTT connection...connected
Published message: Temperature: 24.0°C, Humidity: 40.0%
Message arrived [67nd398b] 41
Turning on relay
Relay turned off
Published message: Temperature: 24.0°C, Humidity: 40.0%
```

Figura 3 – Rezultatul circuitului virtual

Figura 4 – Rezultatul pe dashboard

Figura 4 – Circuitul fizic asamblat

Figura 5 - Rezultatul circuitului fizic ()

CONCLUZIE

Anexa 1

Fisierul main.ino

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

// WiFi credentials
const char* ssid = "Wokwi-GUEST";
const char* password = "";

// MQTT broker details
const char* mqtt_server = "broker.hivemq.com";
const int mqtt_port = 1883;

WiFiClient espClient;
PubSubClient client(espClient);

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

// DHT sensor setup
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// Relay setup
#define RELAYPIN 5

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());
}

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");
}

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

  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
```

```
  }
  Serial.println(message);

  if (String(topic) == "67nd398b") {
    Serial.println("Turning on relay");
    digitalWrite(RELAYPIN, HIGH);  // Turn on the relay
    delay(5000);                   // Wait for 5 seconds
    digitalWrite(RELAYPIN, LOW);  // Turn off the relay
    Serial.println("Relay turned off");
  }
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    String clientId = "ESP32Client";
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      client.subscribe("67nd398b");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}

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

  dht.begin();
  pinMode(RELAYPIN, OUTPUT);
  digitalWrite(RELAYPIN, LOW);  // Ensure relay is OFF initially

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
}

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

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

    float temperature = dht.readTemperature();
    float humidity = dht.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) + "%";
    client.publish("experimental0101", message.c_str());
    Serial.println("Published message: " + message);
  }
}
```

Fişierul 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();
  }
}
```