Raport PRIoT
Sistem pentru Monitorizarea Temperaturii și Umidității cu Notificări și Control la Distanță

Dumitrescu Rareș-Matei

January 18, 2025

Contents

1 Introducere **1**
1.1 Descrierea proiectului 1
1.2 Obiective 2

2 Arhitectură **2**
2.1 Schema topologiei rețelei 2
2.2 Dashboard pentru User 2
2.3 Protocolul de comunicare 2

3 Cod sursă **2**
3.1 ESP32 - Configurare senzori și transmitere date 2

4 Vizualizare date **4**
4.1 Node-RED 4

5 Probleme Întâlnite și Soluții Implementate **4**
5.1 Problema alimentării ventilatorului 4
5.2 Soluția implementată: utilizarea unei plăci Arduino pentru alimentare . . 4
5.3 Cod pentru controlul ventilatorului 5
5.4 Rezultate și Concluzii 6

6 Concluzie **6**

1 Introducere

1.1 Descrierea proiectului

Acest proiect IoT permite monitorizarea temperaturii și umidității în timp real folosind senzori DHT11/DHT22 conectați la un microcontroller ESP32. Scopul proiectului este să ofere date despre mediu printr-un dashboard web, alerte în timp real pentru depășirea limitelor și posibilitatea de a controla un actuator de la distanță.

1.2 Obiective

- Monitorizarea temperaturii și umidității în timp real.
- Vizualizarea datelor pe un dashboard web.
- Notificarea utilizatorului în caz de depășire a limitelor.
- Controlul unui ventilator/LED prin aplicație web.

2 Arhitectură

2.1 Schema topologiei rețelei

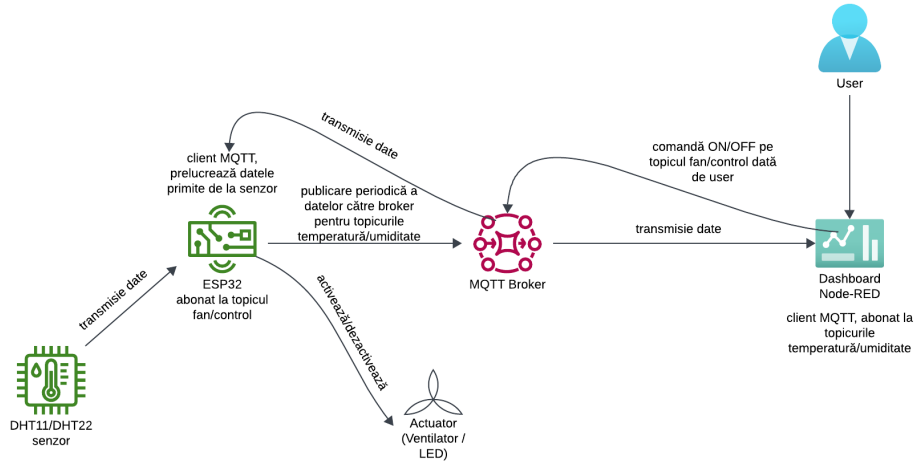

Figure 1: Schema topologiei rețelei

2.2 Dashboard pentru User

Sistemul constă dintr-un senzor DHT11/DHT22 conectat la un ESP32 care transmite date către un broker MQTT. Datele sunt preluate și afișate pe un dashboard web realizat cu Node-RED.

2.3 Protocolul de comunicare

Protocolul MQTT a fost ales pentru transmisia eficientă a datelor, datorită consumului redus de resurse și scalabilității.

3 Cod sursă

3.1 ESP32 - Configurare senzori și transmitere date

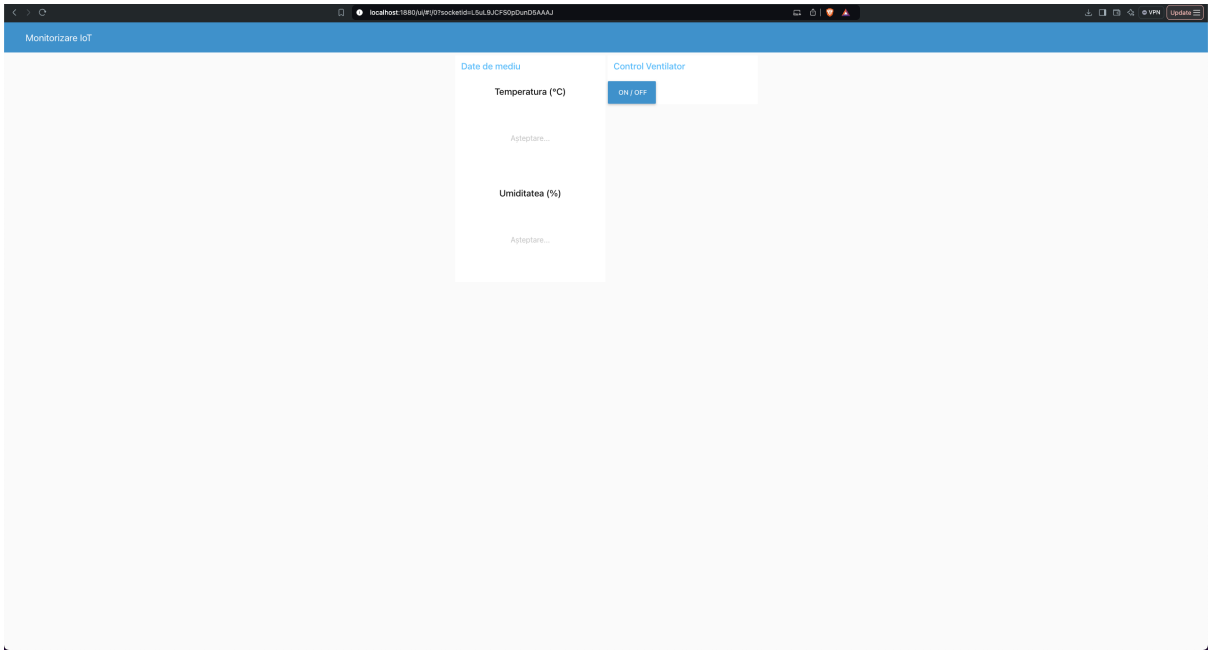

Figure 2: Dashboard pentru User

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

// Configura ii senzor
#define DHTPIN 5
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// Configura ii WiFi  i  MQTT
const char* ssid = "Numele_WiFi";
const char* password = "Parola_WiFi";
const char* mqtt_server = "broker.hivemq.com";
WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  Serial.begin(115200);
  dht.begin();
  WiFi.begin(ssid, password);
  client.setServer(mqtt_server, 1883);
}

void loop() {
  if (!client.connected()) reconnect();
  client.loop();
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
  if (!isnan(temp) && !isnan(hum)) {
    client.publish("temperature", String(temp).c_str());
    client.publish("humidity", String(hum).c_str());
  }
  delay(2000);
```

```
}
```

Listing 1: Cod ESP32 pentru monitorizare temperatură și umiditate

4 Vizualizare date

4.1 Node-RED

Dashboard-ul este configurat pentru a afișa temperatura și umiditatea în timp real și pentru a trimite notificări atunci când valorile depășesc limitele predefinite.

- **Noduri folosite:**
  - MQTT In - pentru datele de la senzori.
  - Chart - pentru graficele temperaturii și umidității.
  - Notification - pentru alerte.

—

5 Probleme Întâlnite și Soluții Implementate

În cadrul implementării proiectului, am întâmpinat o serie de provocări tehnice care au necesitat soluții adecvate pentru a asigura funcționarea corectă a sistemului. Una dintre principalele probleme identificate a fost legată de alimentarea ventilatorului, care nu putea fi acționat direct de ESP32 din cauza limitărilor de tensiune.

5.1 Problema alimentării ventilatorului

Microcontrollerul ESP32 furnizează o tensiune de ieșire de 3.3V, insuficientă pentru alimentarea ventilatorului utilizat în proiect. Deoarece majoritatea ventilatoarelor mici necesită o tensiune de 5V sau mai mare pentru a funcționa eficient, a fost necesar să găsim o soluție pentru a alimenta corect acest component.

Inițial, am încercat să conectăm ventilatorul direct la ESP32, însă acesta nu a reușit să pornească, indicând o problemă de alimentare. Totodată, încercarea de a alimenta ventilatorul direct din pinii ESP32 ar fi putut duce la suprasolicitarea și deteriorarea microcontrollerului.

5.2 Soluția implementată: utilizarea unei plăci Arduino pentru alimentare

Pentru a depăși această problemă, am decis să folosim o placă Arduino ca sursă intermediară de alimentare. Placa Arduino poate furniza o tensiune stabilă de 5V, suficientă pentru funcționarea ventilatorului. Soluția implementată a constat în următorii pași:

1. **Alimentarea ventilatorului prin Arduino:** Am conectat ventilatorul la pinul de 5V al plăcii Arduino pentru a asigura alimentarea corespunzătoare.

2. **Controlul ON/OFF al ventilatorului prin ESP32:** Deoarece ESP32 nu putea furniza suficient curent pentru a porni ventilatorul direct, am folosit un tranzistor pentru a controla alimentarea ventilatorului. Semnalul de control ON/OFF a fost trimis de la ESP32 către un pin digital al Arduino-ului, care la rândul său activa sau dezactiva ventilatorul.

3. **Conectarea ground-urilor:** Pentru a asigura un circuit corect și pentru ca semnalul de control să fie interpretat corect, am conectat ground-ul (GND) ESP32 cu ground-ul plăcii Arduino. Această conexiune este esențială pentru a permite comunicarea corectă între cele două plăci.

5.3 Cod pentru controlul ventilatorului

Pentru implementarea controlului ON/OFF al ventilatorului, am utilizat următorul cod pentru ESP32:

```
#define FAN_CONTROL_PIN 4

void setup() {
    pinMode(FAN_CONTROL_PIN, OUTPUT);
    digitalWrite(FAN_CONTROL_PIN, LOW); // Ventilator oprit ini ial
}

void loop() {
    digitalWrite(FAN_CONTROL_PIN, HIGH); // Pornire ventilator
    delay(5000); // Func ioneaz  5 secunde
    digitalWrite(FAN_CONTROL_PIN, LOW); // Oprire ventilator
    delay(5000); // Pauz  5 secunde
}
```

Listing 2: Cod ESP32 pentru controlul ventilatorului

De asemenea, pentru placa Arduino, am folosit următorul cod pentru a primi semnalul de la ESP32 și pentru a comuta starea ventilatorului:

```
#define ESP_SIGNAL_PIN 7
#define FAN_RELAY_PIN 9

void setup() {
    pinMode(ESP_SIGNAL_PIN, INPUT);
    pinMode(FAN_RELAY_PIN, OUTPUT);
    digitalWrite(FAN_RELAY_PIN, LOW);
}

void loop() {
    int signal = digitalRead(ESP_SIGNAL_PIN);
    if (signal == HIGH) {
        digitalWrite(FAN_RELAY_PIN, HIGH); // Ventilator pornit
    } else {
        digitalWrite(FAN_RELAY_PIN, LOW); // Ventilator oprit
    }
}
```

Listing 3: Cod Arduino pentru activarea ventilatorului

5.4 Rezultate și Concluzii

Prin implementarea acestei soluții, am reușit să controlăm ventilatorul utilizând ESP32, în ciuda limitărilor de tensiune. Utilizarea unei plăci Arduino pentru alimentare a oferit stabilitatea necesară, iar conectarea ground-urilor a permis o comunicare corectă între cele două plăci. Această metodă poate fi extinsă pentru a controla și alte dispozitive care necesită o tensiune mai mare decât cea oferită de ESP32.

6 Concluzie

Această etapă a proiectului a stabilit baza arhitecturală și a realizat prototipul pentru transmiterea și vizualizarea datelor.