

Centro de Estudos e Sistemas Avançados do Recife School

Ciência da computação

Autores: Caio Hirata, Flavio Muniz, João Lafetá, Pedro Coelho, Yara Rodrigues

Professor: Jymmy Barreto

Relatório Técnico – Sistema IoT para Monitoramento de Ambiente

Eletiva 2 — Comunicação e Redes

Pernambuco, Recife

Data: 04/12/2024

1. Introdução

A Internet das Coisas (IoT) tem possibilitado a integração de dispositivos físicos com sistemas digitais, permitindo capturar, processar e visualizar dados de forma remota e em tempo real. O presente projeto teve como objetivo desenvolver um sistema IoT completo utilizando ESP32/ESP8266 como sensores e atuadores, um broker MQTT para comunicação e um dashboard para visualização dos dados coletados.

O sistema implementado realiza a leitura de variáveis ambientais, como temperatura, umidade e luminosidade, enviando-as continuamente a um servidor responsável por exibir essas informações em uma interface web. Além disso, o ambiente permite controle remoto de atuadores, demonstrando a bidirecionalidade do protocolo MQTT.

2. Metodologia

2.1 Arquitetura do Sistema

A solução foi estruturada conforme os princípios fundamentais de um sistema IoT distribuído:

- **Módulos ESP32/ESP8266:** realizam leituras dos sensores e enviam dados por MQTT.
- **Broker MQTT (Mosquitto):** responsável por receber e distribuir mensagens publicadas pelos dispositivos.
- **Aplicação Backend:** realiza tratamento dos dados, persistência opcional e integração entre o broker e o dashboard.
- **Dashboard Web:** exibe métricas em tempo real e fornece controles para atuação em dispositivos físicos.

Diagrama da Arquitetura (conceitual)

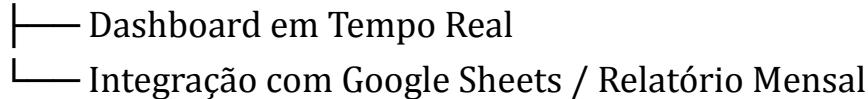
[Sensores ESP32/ESP8266]



[Broker MQTT - Mosquitto]



[Node-RED Backend]



[Sensores ESP32/ESP8266]

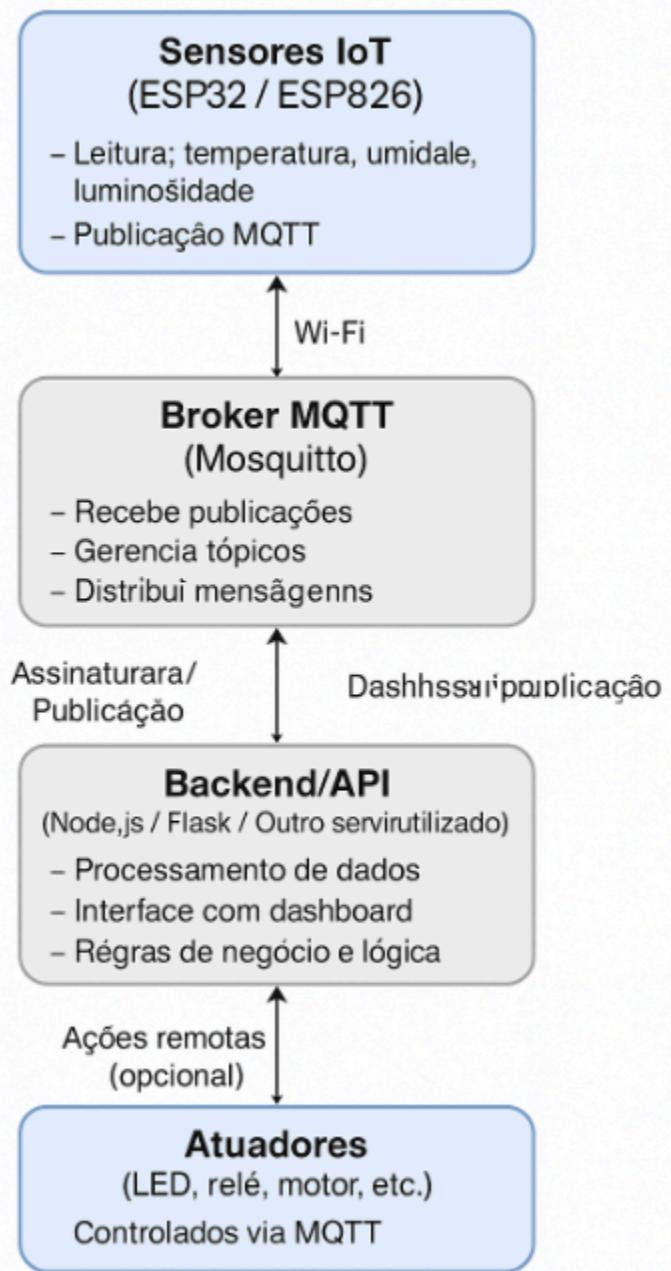


[Broker MQTT - Mosquitto]



[Node-RED Backend]





2.2 Hardware Utilizado

- 1x ESP32 ou ESP8266 (NodeMCU)
- Sensores ambientais: *ex.: DHT22 (temperatura/umidade), LDR (luminosidade)*
- Protoboard, cabos jumper
- LEDs e resistores
- Opcional: Raspberry Pi como broker local

2.3 Software Utilizado

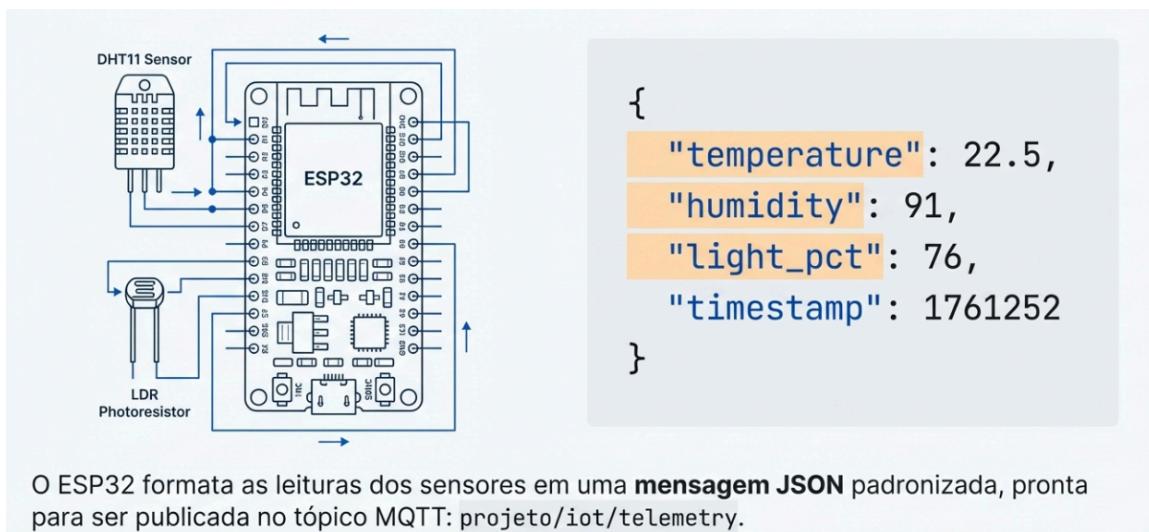
- Firmware em MicroPython ou C++ (Arduino IDE)
- Mosquitto MQTT Broker
- Backend em *Node.js* ou *Python Flask/Django*
- Dashboard utilizando *HTML/JS, React, Node-RED ou Grafana*
- Versionamento via GitHub

2.4 Fluxo de Comunicação MQTT

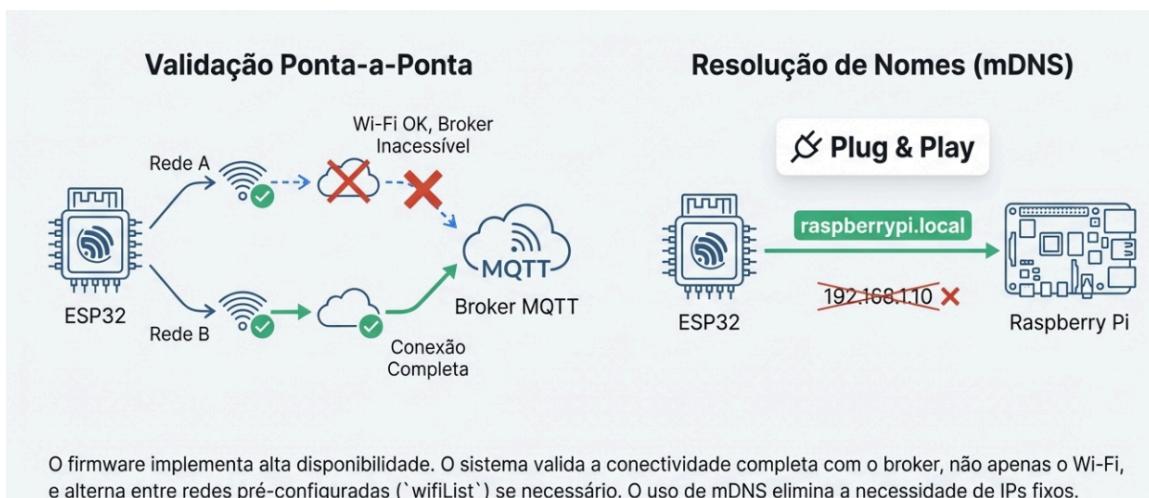
1. O ESP32 conecta-se ao Wi-Fi e ao broker MQTT.
2. Variáveis ambientais são lidas periodicamente.
3. O dispositivo publica as medições em tópicos como:
 - ambiente/temperatura
 - ambiente/umidade

- ambiente/luminosidade
- 4. O broker distribui as mensagens a todos os assinantes.
- 5. O dashboard, inscrito nos tópicos, atualiza os gráficos em tempo real.
- 6. Para controle remoto, o dashboard publica em tópicos como:
 - atuador/led
 - atuador/rele

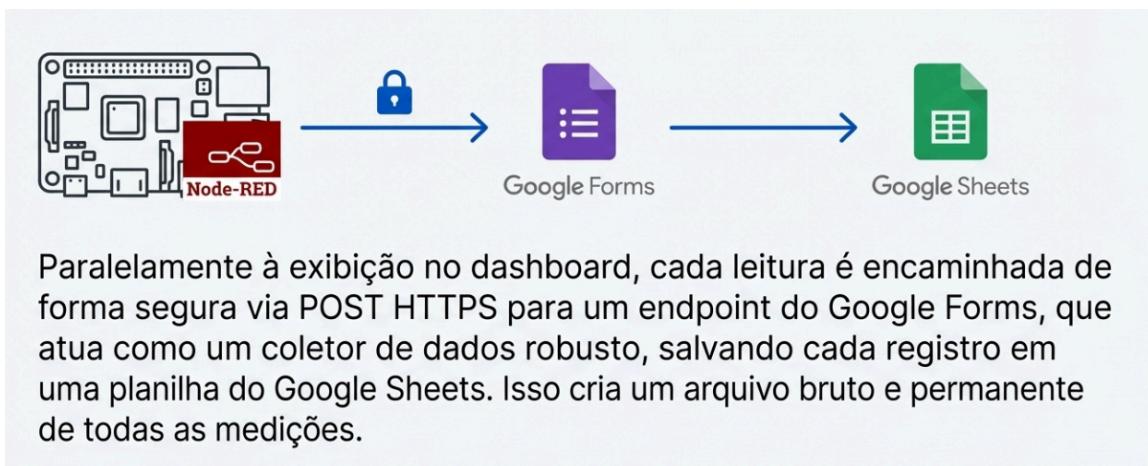
2.5 Fluxo de Funcionamento



O ESP32 formata as leituras dos sensores em uma **mensagem JSON** padronizada, pronta para ser publicada no tópico MQTT: projeto/iot/telemetry.



O firmware implementa alta disponibilidade. O sistema valida a conectividade completa com o broker, não apenas o Wi-Fi, e alterna entre redes pré-configuradas ('wifiList') se necessário. O uso de mDNS elimina a necessidade de IPs fixos.



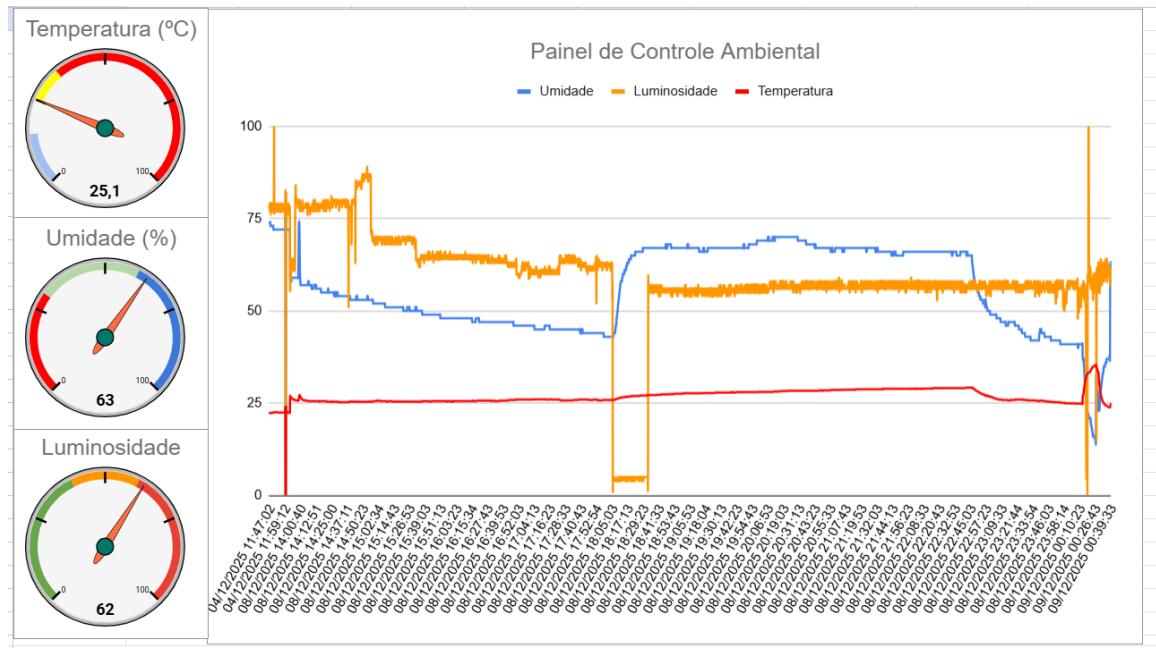
3. Resultados

3.1 Dashboard

O dashboard desenvolvido apresenta:

- Gráficos em tempo real atualizados via MQTT
- Indicadores das variáveis monitoradas





Perguntas Respostas 378 Configurações

378 respostas

[Ver no app Planilhas](#)



Resumo

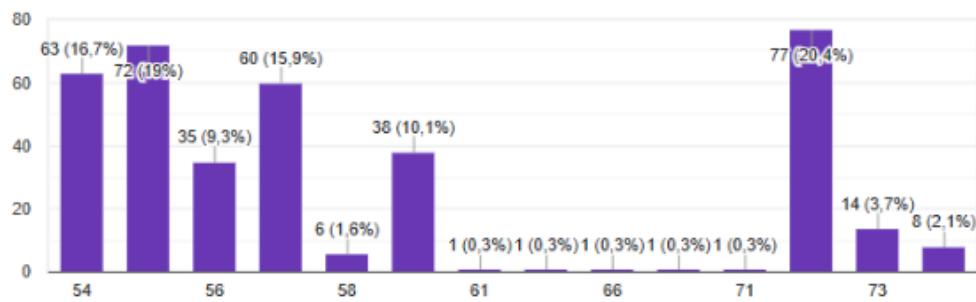
Pergunta

Individual

Umidade

[Copiar gráfico](#)

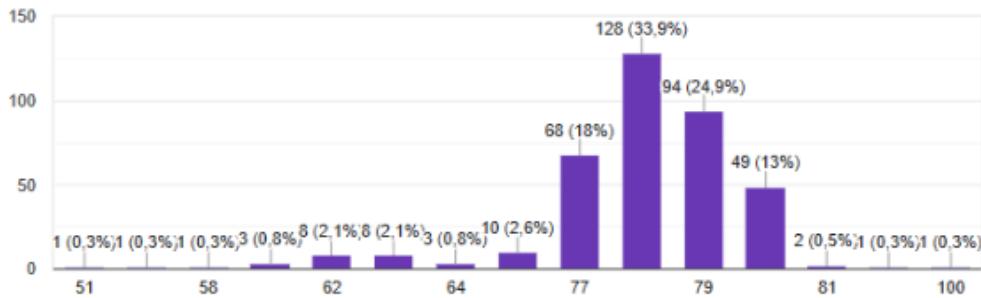
378 respostas



luminosidade

[Copiar gráfico](#)

378 respostas



Temperatura

[Copiar gráfico](#)

3.2 Dados Coletados

Formulário de coleta:

The screenshot shows a survey creation interface with the following details:

- Formulário sem título** (Survey without title)
- Descrição do formulário** (Survey description) - empty field
- Umidade *** (Humidity *) - short text input field
- luminosidade *** (Luminosity *) - short text input field
- Temperatura *** (Temperature *) - short text input field

On the right side, there is a vertical toolbar with icons for adding fields, saving, deleting, and other configuration options.

Exemplo de tabela para apresentação dos resultados:

Form_Responses	Calendário	Data e hora	Umidade	Luminosidade	Temperatura
		04/12/2025 11:47:02	74	79	22,4
		04/12/2025 11:47:13	74	79	22,4
		04/12/2025 11:47:22	74	78	22,4
		04/12/2025 11:47:32	74	78	22,4
		04/12/2025 11:47:42	74	78	22,4
		04/12/2025 11:47:52	74	77	22,4
		04/12/2025 11:48:02	74	78	22,4
		04/12/2025 11:48:12	74	79	22,4
		04/12/2025 11:48:24	73	78	22,4
		04/12/2025 11:48:32	73	79	22,4
		04/12/2025 11:48:43	73	76	22,4
		04/12/2025 11:48:53	73	79	22,4
		04/12/2025 11:49:03	73	79	22,4
		04/12/2025 11:49:13	73	78	22,4
		04/12/2025 11:49:22	73	79	22,4
		04/12/2025 11:49:32	73	79	22,4

3.3 Evidências de Funcionamento

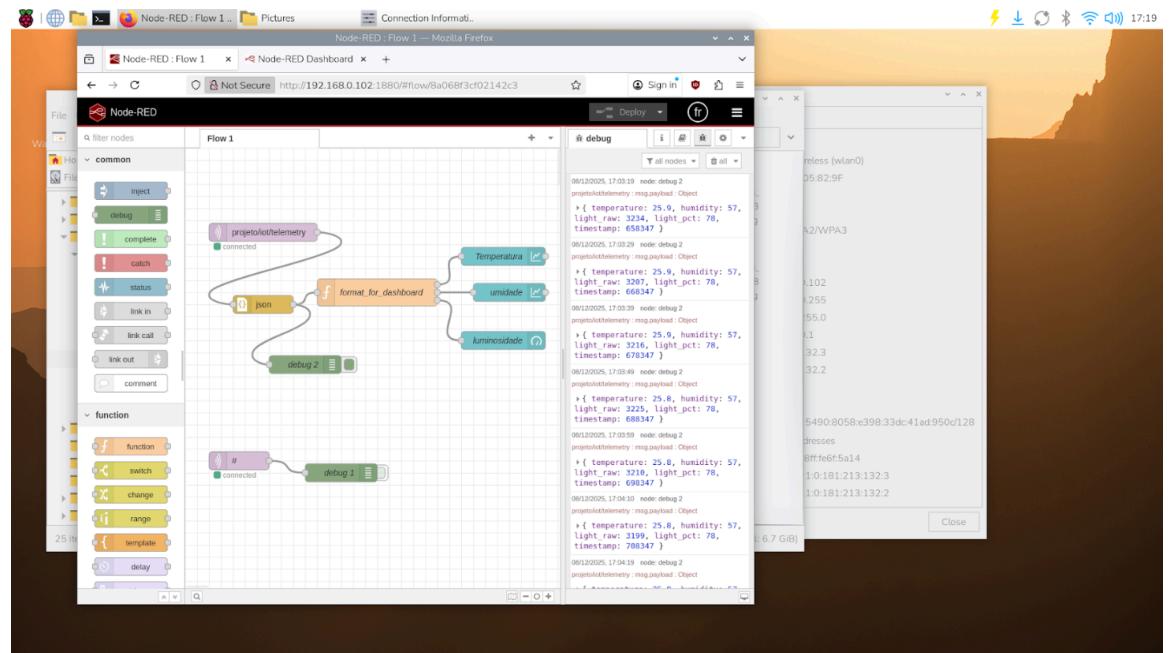
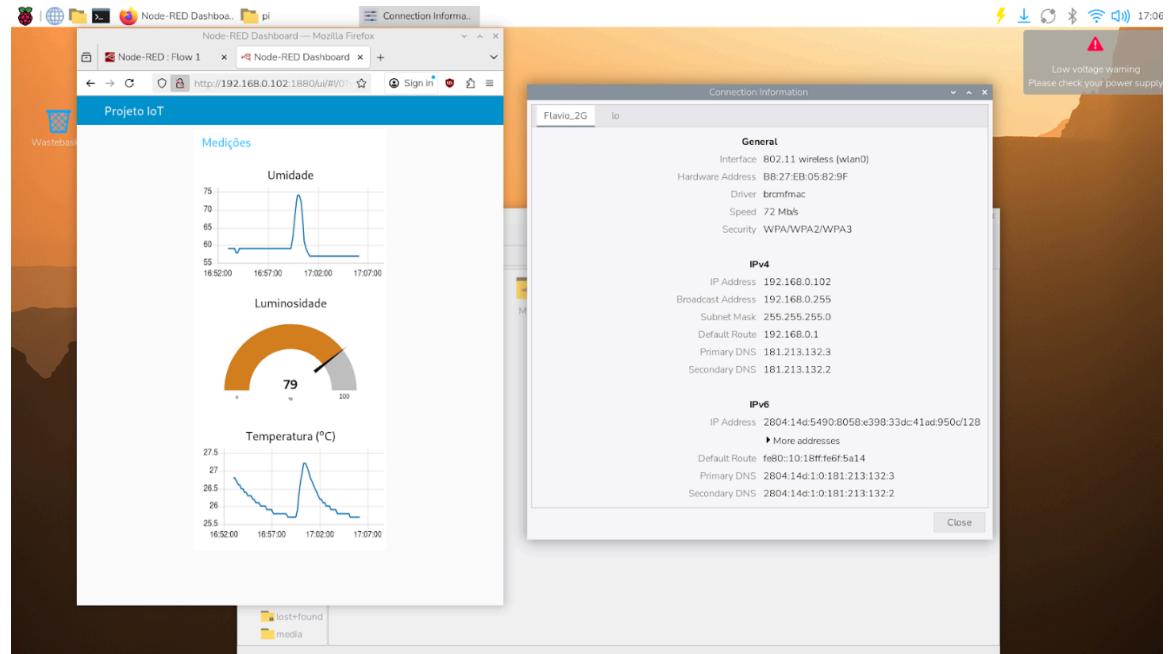
- Imagens do terminal mostrando publicações MQTT

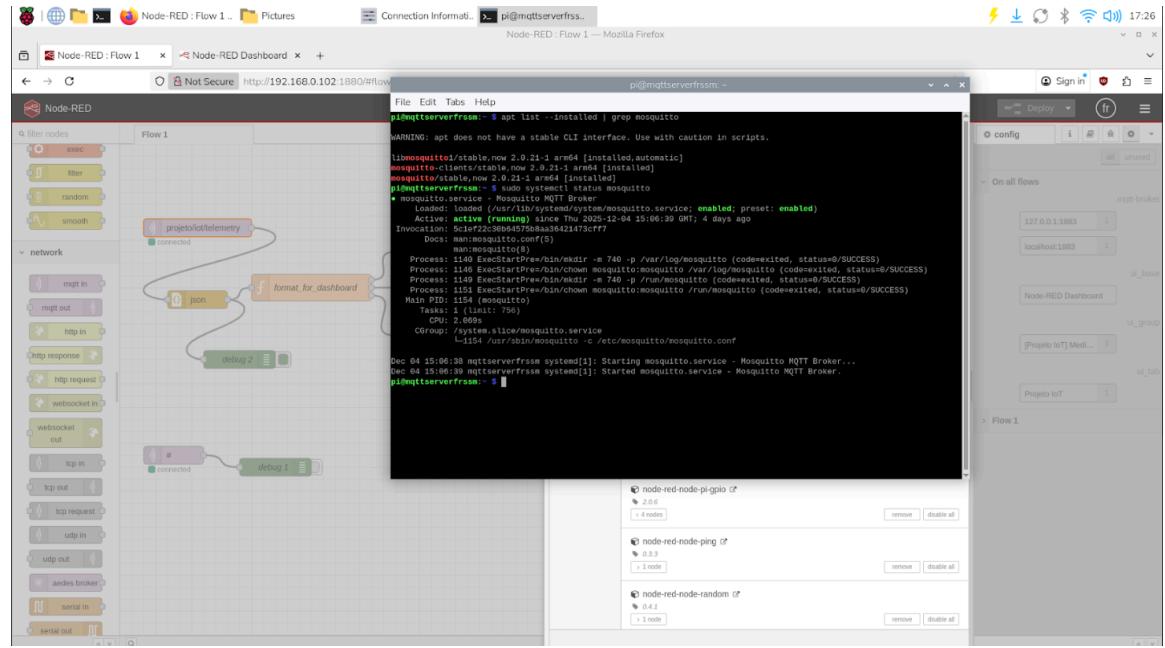
```
rst:0x1 (POWERON_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:1184
load:0x40078000,len:13232
load:0x40080400,len:3028
entry 0x400805e4
=====
Tentando WiFi 1: SSID='uaifai-tiradentes'
=====
Tentando conectar em 'uaifai-tiradentes' (timeout 20000ms)...
-----
Falha ao conectar em 'uaifai-tiradentes'.
WiFi falhou. Indo para a próxima rede...

=====
Tentando WiFi 2: SSID='Flavio_2G'
=====
Tentando conectar em 'Flavio_2G' (timeout 20000ms)...
.
WiFi conectado!
SSID: Flavio_2G
IPv4: 192.168.0.142
IPv6: 0000:0000:0000:0000:0000:0000:0000:0000
WiFi OK! Tentando MQTT...
== IPv6 diagnostic ==
WiFi.status(): 3
IPv4: 192.168.0.142
IPv6 (toString): 0000:0000:0000:0000:0000:0000:0000:0000
IPv6 atribuído!
=====
MQTT conectado com sucesso!
=====

mDNS iniciado. Procurando servidores...
Publicando em projeto/iot/telemetry: {"temperature":25.5,"humidity":53,"light_raw":3278,"light_pct":80,"timestamp":35601}
Publicando em projeto/iot/telemetry: {"temperature":25.5,"humidity":53,"light_raw":3294,"light_pct":80,"timestamp":45600}
```

- Fotos da aplicação operando





4. Conclusão

O projeto permitiu a implementação prática de um sistema IoT completo, integrando sensores físicos, comunicação sem fio, protocolo MQTT, backend e visualização em tempo real. Os principais aprendizados incluem:

- Configuração e operação do broker MQTT
- Estruturação de um fluxo de comunicação publish/subscribe
- Programação embarcada no ESP32/ESP8266
- Integração backend ↔ dashboard

4.1 Melhorias Futuras

- Inclusão de banco de dados histórico
- Criação de alertas automáticos via e-mail/Telegram
- Implementação de IPv6 completo
- Expansão do número de sensores e atuadores
- Migração do sistema para uma arquitetura em nuvem (AWS IoT Core)

5. Apêndice

5.1 Código do ESP32/ESP8266

```
/*
Pinout / Wiring (esqueleto)
- DHT11 data pin -> GPIO4
```

```

- LDR (fotoresistor) (P0)

3.3V --- 10k resistor --- LDR --- GPIO34

LDR --- GND

*/



#include <Arduino.h>

#include <WiFi.h>

#include <PubSubClient.h>

#include <DHT.h>

#include <ArduinoJson.h>

#include <ESPmDNS.h>





// ----- CONFIGURAÇÕES -----



//#define WIFI_SSID      "nome_da_rede"

//#define WIFI_PASS      "senha"



// coloque aqui as suas redes (adicone/remova conforme precisar)

struct WifiCred {

    const char* ssid;

    const char* pass;

} ;



WifiCred wifiList[] = {

```

```

    { "nome_da_rede1", "senha1" },
    { "nome_da_rede2", "senha2" },
    { "nome_da_rede3", "senha3" }

};

const unsigned long ATTEMPT_MS = 20000; // tempo para tentar cada rede (ms)

const unsigned long BETWEEN_ATTEMPT_MS = 500; // sleep durante polling

#define MQTT_SERVER_V4      "172.20.10.2"    // troque para o IP da sua Raspberry Pi

#define MQTT_SERVER_V6      "2804:214:81e4:9cf2:68c7:d8fe:1d79:c837" // ipv6

#define MQTT_PORT           1883

#define MQTT_USER            "frssm"        // se usar autenticação, coloque usuário

#define MQTT_PASS            "12345678"     // e senha aqui

// --- CONFIGURAÇÕES ---

// DUAL STACK NATIVO: Usamos o NOME, não o IP.

// O sistema vai "caçar" esse nome na rede.

#define MQTT_HOST            "raspberrypi.local"

```

```
#define MQTT_PORT      1883

const char* mqtt_topic_base = "projeto/iot"; // prefixo dos tópicos

// sensores

#define DHTPIN 4
#define DHTTYPE DHT11

#define LDR_PIN 34

// intervalo de publicação (ms)

const unsigned long PUBLISH_INTERVAL = 10000;

// ----- OBJETOS -----

WiFiClient espClient;
PubSubClient mqttClient(espClient);
DHT dht(DHTPIN, DHTTYPE);

unsigned long lastPublish = 0;

// put function declarations here:
//void connectWiFi();
void connectWiFiandMQTT();
void connectMQTT();
float readTemperature();
```

```
float readHumidity();

int readLDRraw();

int rawToPercent(int raw);

void publishSensorData();

void printSensorData();

bool tryConnect(const char* ssid, const char* pass, unsigned long
timeout_ms);

/* void setup() {

    // put your setup code here, to run once:

    Serial.begin(115200);

    delay(1000);

    pinMode(LDR_PIN, INPUT);

    dht.begin();

    connectWiFi();

    // 2. Inicia o sistema de nomes (mDNS)

    if (!MDNS.begin("esp32-client")) {

        Serial.println("Erro ao iniciar mDNS responder!");

    } else {

        Serial.println("mDNS iniciado. Procurando servidores...");

    }

    // Configura o Broker usando o NOME (Host)
```

```
mqttClient.setServer(MQTT_HOST, MQTT_PORT);

//leitura inicial dos sensores

delay(2000);

dht.read();

}

void loop() {

// put your main code here, to run repeatedly:

if (WiFi.status() != WL_CONNECTED) {

connectWiFi();

}

if (!mqttClient.connected()) {

connectMQTT();

}

mqttClient.loop();

}

unsigned long now = millis();

if (now - lastPublish >= PUBLISH_INTERVAL) {

publishSensorData();

lastPublish = now;

}

//printSensorData();

//delay(5000);
```

```
}

void setup() {
    Serial.begin(115200);
    delay(1000);

    pinMode(LDR_PIN, INPUT);
    dht.begin();

    // garante que o cliente conheça o broker (hostname ou IP)
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);

    // bloqueia aqui até obter WiFi + MQTT (função implementada por
    // você)
    connectWiFiandMQTT();

    // inicia mDNS após conexão de rede (para resolver .local se
    // necessário)
    if (!MDNS.begin("esp32-client")) {
        Serial.println("Erro ao iniciar mDNS responder!");
    } else {
        Serial.println("mDNS iniciado. Procurando servidores...");
    }
}
```

```
// leitura inicial do DHT (descartar primeira leitura)

delay(2000);

dht.read();

lastPublish = millis();

}

}

void loop() {

    // Se em algum momento WiFi ou MQTT cair, tenta reconectar
    // (bloqueante até sucesso)

    if (WiFi.status() != WL_CONNECTED || !mqttClient.connected()) {

        Serial.println("Conexão perdida. Reconnectando WiFi + MQTT...");

        connectWiFiandMQTT();

        // tentar reiniciar mDNS caso ainda não esteja ativo

        if (!MDNS.begin("esp32-client")) {

            Serial.println("mDNS não iniciado após reconexão (já existia
ou falhou).");

        } else {

            Serial.println("mDNS iniciado após reconexão.");

        }

        // garante que mqttClient esteja apontando para o host correto

        mqttClient.setServer(MQTT_HOST, MQTT_PORT);

    }

}
```

```
// mantém o loop do cliente MQTT responsável

mqttClient.loop();

// publicação periódica sem usar delay()

unsigned long now = millis();

if (now - lastPublish >= PUBLISH_INTERVAL) {

    publishSensorData();

    lastPublish = now;

}

}

bool tryConnect(const char* ssid, const char* pass, unsigned long timeout_ms) {

    Serial.printf("Tentando conectar em '%s' (%ld ms)...\\n",
        ssid, timeout_ms);

    // limpar estado anterior

    WiFi.disconnect(true);

    delay(100);

    WiFi.mode(WIFI_STA);

    WiFi.enableIPv6(); // se quiser dual stack

    delay(50);

}
```

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

unsigned long start = millis();

while (WiFi.status() != WL_CONNECTED && (millis() - start) <
timeout_ms) {

    delay(BETWEEN_ATTEMPT_MS);

    Serial.print(".");
}

Serial.println();

if (WiFi.status() == WL_CONNECTED) {

    Serial.println("WiFi conectado!");

    Serial.print("SSID: "); Serial.println(WiFi.SSID());

    Serial.print("IPv4: "); Serial.println(WiFi.localIP());

    // localIPv6() retorna um objeto IPAddressV6; imprimir depende
    da versão da lib

    // Para segurança, convertemos para String:

    String ipv6 = WiFi.localIPv6().toString();

    Serial.print("IPv6: "); Serial.println(ipv6);

    return true;
} else {

    Serial.printf("Falha ao conectar em '%s'.\n", ssid);

    return false;
}
```

```
}

}

/* void connectWiFi() {

    if (WiFi.status() == WL_CONNECTED) return;

    Serial.printf("Conectando ao WiFi %s ...", WIFI_SSID);

    WiFi.mode(WIFI_STA);

    delay(50);

    WiFi.enableIPv6(); // Habilita a pilha para permitir Dual Stack

    delay(50);

    WiFi.begin(WIFI_SSID, WIFI_PASS);

    unsigned long start = millis();

    while (WiFi.status() != WL_CONNECTED && millis() - start < 20000)
    {

        delay(500);

        Serial.print(".");
    }

    Serial.println();
}

if (WiFi.status() == WL_CONNECTED) {

    Serial.println("WiFi Conectado!");

    Serial.print("IPv4: "); Serial.println(WiFi.localIP());

    delay (2000);
}
```

```
Serial.print("IPv6: "); Serial.println(WiFi.localIPv6());  
}  
} else {  
    Serial.println("Falha na conexão WiFi.");  
}  
}  
*/  
  
/* void connectWiFi() {  
    // Uncomment para tentar continuamente até conectar em alguma rede:  
    while (true) {  
        size_t n = sizeof(wifiList) / sizeof(wifiList[0]);  
        for (size_t i = 0; i < n; ++i) {  
            if (tryConnect(wifiList[i].ssid, wifiList[i].pass,  
ATTEMPT_MS)) {  
                if (WiFi.status() == WL_CONNECTED) {  
                    Serial.println("WiFi Conectado!");  
                    Serial.print("IPv4: "); Serial.println(WiFi.localIP());  
                    delay (2000);  
                    Serial.print("IPv6: "); Serial.println(WiFi.localIPv6());  
                } else {  
                    Serial.println("Falha na conexão WiFi.");  
                }  
                return; // conectado com sucesso  
            }  
        }  
    }  
}
```

```

    // opcional: pequeno delay antes da próxima tentativa
    delay(500);

}

// se chegou aqui, todas as redes falharam
Serial.println("Nenhuma rede conectou nesta rodada.");

// Se quiser tentar novamente em loop, descomente o próximo
delay e o while(true) na linha superior.

// delay(30000); // aguarda 30s antes de tentar novamente

}

} /*/


// =====
// Conecta WiFi e MQTT seguindo ordem da lista:

// 1) Tenta WiFi[i]

// 2) Se conectou → tenta MQTT

// 3) Se MQTT falhou → desconecta WiFi e tenta o próximo

// 4) Só sai quando WiFi + MQTT estiverem OK

// =====


void connectWiFiandMQTT() {
    while (true) {

        size_t total = sizeof(wifiList) / sizeof(wifiList[0]);


        for (size_t i = 0; i < total; i++) {

```

```

Serial.println("=====");
Serial.printf("Tentando WiFi %u: SSID=%s\n", i + 1,
wifiList[i].ssid);
Serial.println("=====");

// ----- TENTA CONECTAR AO WIFI -----
if (!tryConnect(wifiList[i].ssid, wifiList[i].pass,
ATTEMPT_MS)) {
    Serial.println("WiFi falhou. Indo para a próxima
rede...\n");
    continue; // tenta a próxima rede
}

Serial.println("WiFi OK! Tentando MQTT...");

// --- DEBUG EXTRA: Verifica o IPv6 atribuído ---
Serial.println("== IPv6 diagnostic ==");
String ipv6 = WiFi.localIPv6().toString();
Serial.print("WiFi.status(): ");
Serial.println(WiFi.status());

Serial.print("IPv4: "); Serial.println(WiFi.localIP());
Serial.print("IPv6 (toString): "); Serial.println(ipv6);

// verifica se é endereço não-zero
if (ipv6.length() == 0 || ipv6 == "0:0:0:0:0:0:0:0") {

```

```

    Serial.println("IPv6 não atribuído (endereço zero). A rede
provavelmente não fornece IPv6.");

} else {

    Serial.println("IPv6 atribuído!");

}

Serial.println("=====");

// ----- TENTA MQTT -----
mqttClient.setServer(MQTT_HOST, MQTT_PORT);

String clientId = "ESP32-" +
String((uint32_t)ESP.getEfuseMac(), HEX);

bool mqtt_ok = mqttClient.connect(clientId.c_str(), MQTT_USER,
MQTT_PASS);

if (mqtt_ok) {

    Serial.println("MQTT conectado com sucesso!");

    Serial.println("=====\\n");

    return; // *** SUCESSO TOTAL ***
}

// ----- MQTT FALHOU -----
Serial.printf("MQTT falhou (rc=%d). Desconectando WiFi e
tentando próximo...\\n",

```

```
        mqttClient.state()) ;

        WiFi.disconnect(true) ;

        delay(1000) ;

    }

    // se chegou aqui, nenhuma rede da lista funcionou
    Serial.println("Nenhuma rede funcionou nesta rodada. Tentando de
novo em 5s...") ;

    delay(5000) ;

}

}

/* void connectMQTT() {
    if (mqttClient.connected()) return;

    String clientId = "ESP32-" + String((uint32_t)ESP.getEfuseMac(), 
HEX) ;

    // --- TENTATIVA 1: PLANO A (IPv6) ---
    Serial.print("Tentando MQTT via IPv6 (Plano A)... ") ;
}
```

```

// Configura para usar o endereço IPv6

mqttClient.setServer(MQTT_SERVER_V6, MQTT_PORT);

// Tenta conectar (timeout curto de 2s para não travar muito se
falhar)

// Nota: A lib PubSubClient bloqueia um pouco, então seja paciente
no log

if (mqttClient.connect(clientId.c_str())) {

    Serial.println("SUCESSO! Conectado via IPv6.");
    return; // Sai da função, já estamos conectados!

} else {

    Serial.print("Falhou (rc=");
    Serial.print(mqttClient.state());
    Serial.println("). Iniciando Plano B...");

}

// --- TENTATIVA 2: PLANO B (IPv4) ---

Serial.print("Tentando MQTT via IPv4 (Plano B)... ");

// Troca a configuração para o endereço IPv4

mqttClient.setServer(MQTT_SERVER_V4, MQTT_PORT);

if (mqttClient.connect(clientId.c_str())) {

    Serial.println("SUCESSO! Conectado via IPv4 (Fallback).");
}

```

```

} else {

    Serial.print("Falha Total (rc=");
    Serial.print(mqttClient.state());
    Serial.println("). Tentando novamente em 5s...");

    delay(5000);
}

}

*/



void connectMQTT() {
    if (mqttClient.connected()) return;

    Serial.printf("Tentando resolver e conectar a: %s ... ", MQTT_HOST);

    // A MÁGICA ACONTECE AQUI:
    // Ao passar um NOME, a biblioteca tenta resolver (DNS/mDNS).
    // Se a rede suportar, ela pega o IP automaticamente.

    String clientId = "ESP32-" + String((uint32_t)ESP.getEfuseMac(), HEX);

    if (mqttClient.connect(clientId.c_str())) {
        Serial.println("\n[SUCESSO] Conectado via Hostname!");
    }
}

```

```
// Não sabemos qual IP ele escolheu, foi transparente.

} else {

Serial.print("\n[FALHA] rc=");

Serial.print(mqttClient.state());

Serial.println(" -> Verifique se a rede permite mDNS (.local)");

delay(5000);

}

}

float readTemperature() {

float t = dht.readTemperature(); // Celsius

if (isnan(t)) {

Serial.println("Erro leitura DHT (temp)");

return NAN;

}

return t;

}

float readHumidity() {

float h = dht.readHumidity();

if (isnan(h)) {

Serial.println("Erro leitura DHT (hum)");
```

```
    return NAN;

}

return h;

}

int readLDRraw() {

    // ADC range 0-4095 por padrão

    int raw = analogRead(LDR_PIN);

    return raw;

}

int rawToPercent(int raw) {

    // mapeia 0-4095 para 0-100. Ajuste se seu divisor inverte a escala.

    int val = map(raw, 0, 4095, 0, 100);

    if (val < 0) val = 0;

    if (val > 100) val = 100;

    return val;

}

void publishSensorData() {

    float temp = readTemperature();

    float hum = readHumidity();

    int ldrRaw = readLDRraw();
```

```

int lightPerc = rawToPercent(ldrRaw);

// montar JSON

JsonDocument doc;

if (!isnan(temp)) doc["temperature"] = roundf(temp * 10) / 10.0;
// 1 casa

if (!isnan(hum)) doc["humidity"] = roundf(hum * 10) / 10.0;

doc["light_raw"] = ldrRaw;

doc["light_pct"] = lightPerc;

doc["timestamp"] = millis();

char payload[256];

size_t n = serializeJson(doc, payload, sizeof(payload));

String topic = String(mqtt_topic_base) + "/telemetry"; // projeto/iot/telemetry

Serial.print("Publicando em ");

Serial.print(topic);

Serial.print(": ");

Serial.println(payload);

if (mqttClient.connected()) {

    mqttClient.publish(topic.c_str(), payload, false);
}

```

```

} else {

    Serial.println("MQTT desconectado. Publicação não enviada.");
}

}

void printSensorData() {
    float temp = readTemperature();

    float hum = readHumidity();

    int ldrRaw = readLDRraw();

    int lightPct = rawToPercent(ldrRaw);

    Serial.println("== Leitura dos Sensores ==");

    if (!isnan(temp)) Serial.printf("Temperatura: %.1f °C\n", temp);

    if (!isnan(hum)) Serial.printf("Umidade: %.1f %%\n", hum);

    Serial.printf("LDR (bruto): %d\n", ldrRaw);

    Serial.printf("Luminosidade: %d %%\n", lightPct);

    Serial.println("=====\\n");
}

```

5.2 Configuração do Broker MQTT

- Arquivo `mosquitto.conf`

```
# Configuração Básica do Mosquitto Broker para o Projeto IoT
```

```
# Local onde as mensagens persistentes (Retain) são salvas
persistence true

persistence_location /var/lib/mosquitto/

# Arquivo de Log (útil para debug se algo der errado)
log_dest file /var/log/mosquitto/mosquitto.log

# -----
# LISTENERS
# -----
# Porta padrão MQTT (1883).

# O listener precisa ser declarado explicitamente para
# aceitar conexões externas (do ESP32)

# Se não colocar isso, ele só aceita conexões locais
# (localhost)

listener 1883

# -----
# SEGURANÇA
```

```

# Permitir conexões anônimas (sem usuário/senha).

# Mude para 'false' se configurar um arquivo de senhas
(password_file)

allow_anonymous true

```

5.3 Diagramas e Esquemáticos

