DTSD – Trabalho Prático 1
Sistema de Monitorização e Controlo Climático com ESP32, MQTT e Cloud

António Alves
Nº 58339
João Borges
Nº 70521
Tomás Fialho
Nº60731
FCT NOVA – Engenharia Eletrotécnica e de Computadores

Ano Letivo 2024/2025

Conteúdo

1 Introdução

No nosso dia a dia, ouvimos falar cada vez mais da importância da eficiência energética e redução de custos em sistemas elétricos e eletrónicos. Dado o entrosamento cada vez maior entre estes sistemas e a vida quotidiana, surgem cada vez mais a necessidade de programas inteligentes que tenham em conta estas necessidades.

O nosso sistema desenvolvido nesta Unidade Curricular tem em conta variáveis tais como a temperatura exterior, a temperatura interior, a humidade, a luminosidade e a presença de utilizadores numa casa. Este IoT foi concebido como uma base de aprendizagem para um sistema aplicável ao mundo real, sendo por isso possível de ser escalado e de incorporar ainda mais funções. É controlável por app e apresenta dashboards locais e na cloud.

O protocolo **MQTT** (Message Queuing Telemetry Transport) é amplamente utilizado em aplicações IoT pela sua leveza e eficiência, sendo ideal para dispositivos com recursos limitados como o **ESP32**. Ferramentas como o **Node-RED** permitem criar fluxos visuais para a integração e controlo de dispositivos, enquanto plataformas cloud como o **ThingSpeak** oferecem suporte para análise de dados históricos.

A montagem numa breadboard incorpora um **ESP32**, um **ecrã OLED**, **sensores** e **atuadores**. Foi feita de acordo com a *Figura 1*.

2 Sistema Desenvolvido

2.1 Componentes

- ESP32
- Sensor DHT22 (temperatura e humidade interior)
- Sensor DS18B20 (temperatura exterior)
- Sensor PIR (detecção de movimento)
- Sensor LDR (luminosidade)
- Servos para janela e ventilador
- LED de controlo ambiental
- Ecrã OLED

2.2 Arquitetura do Sistema

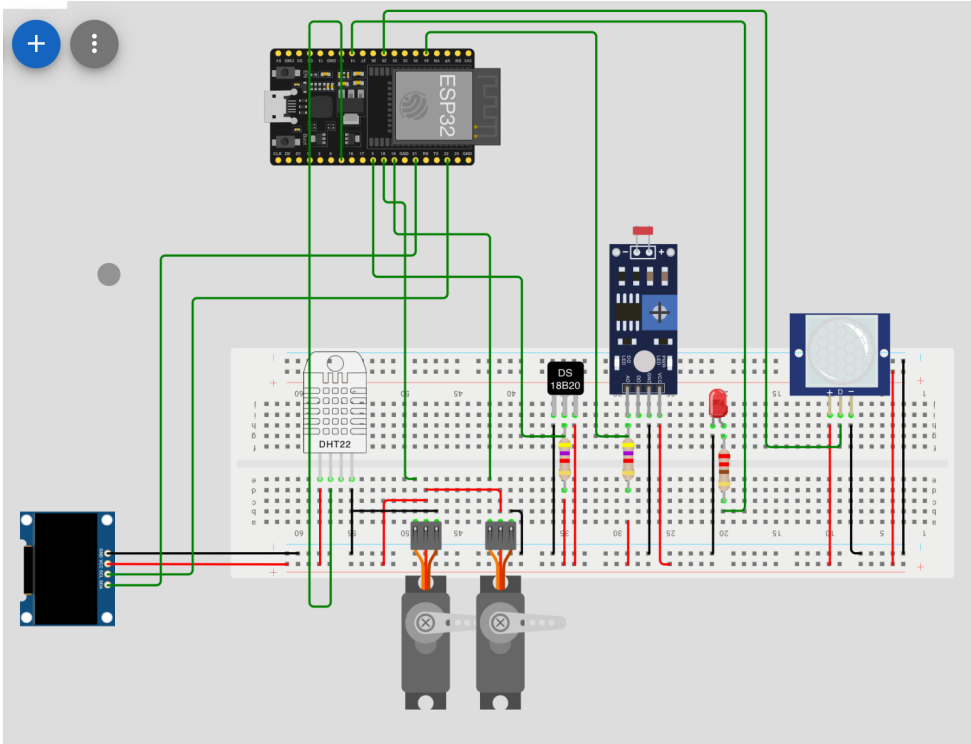

Figura 1: Arquitetura geral do sistema

2.3 Comunicação e Integração

- **MQTT** usado para comunicação bidirecional com Node-RED
- **Node-RED** publica dados para ThingSpeak
- **Mobile App** ou **Home Assistant** permitem controlo remoto via MQTT
- Dados de sensores agregados em tópicos como `home/sensors` e `home/status`

O sistema opera em dois modos: **automático** e **manual**. No modo automático, toda a lógica de decisão é tratada no Node-RED, com base nos dados recebidos via MQTT pelos sensores ligados ao ESP32. Quanto ao modo manual, todos os atuadores podem ser controlados pelo utilizador no Node-RED, que, por sua vez, envia comandos MQTT de volta para o ESP.

3 Diagramas UML

3.1 Use Case Diagram

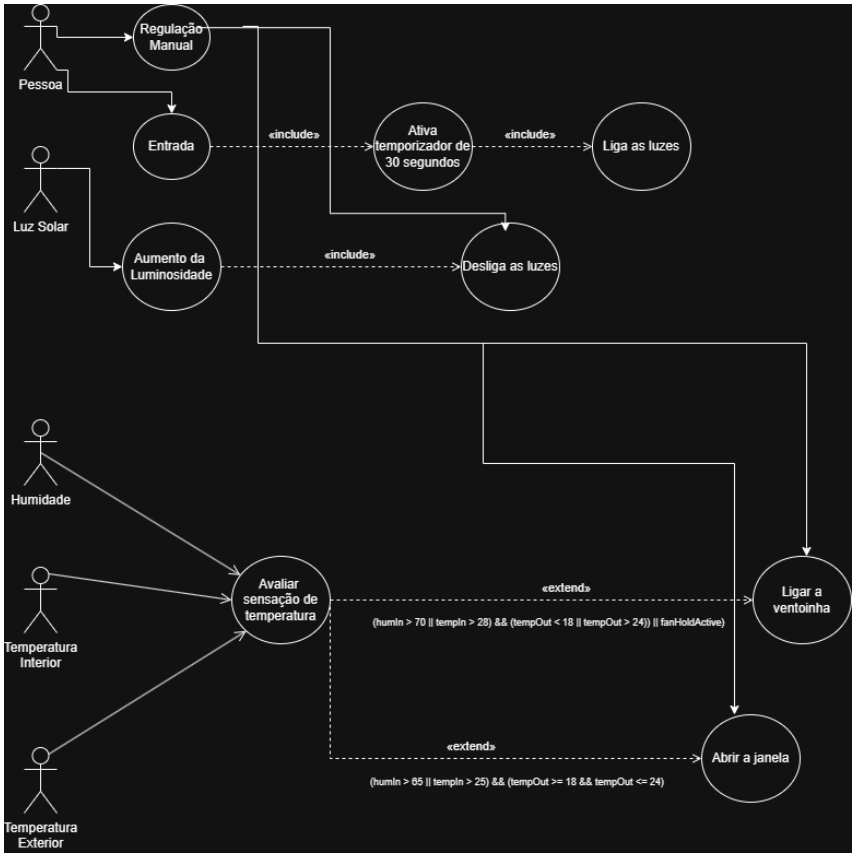

Figura 2: Diagrama de casos de uso

O diagrama de casos de uso ilustra as interações entre cinco atores principais - o próprio utilizador, a temperatura interior, a temperatura exterior, a humidade e a luminosidade.

O utilizador é ator tanto no modo manual, em que controla os sensores manualmente, e como no modo automático, em que a sua presença cria um timer para manter a luz ligada em casos de falta de luminosidade.

A luz solar é o ator que controla se a luz é ligada ou desligada, dependendo da sua intensidade.

A humidade, a temperatura exterior e a temperatura interior estão interligadas - isto porque todas as métricas contribuem para a sensação térmica. Quando a sensação térmica atinge um threshold, é ligada uma ventoinha ou acionado o servo de uma janela, dependendo da temperatura exterior, com o intuito de poupar energia.

3.2 Sequence Diagram

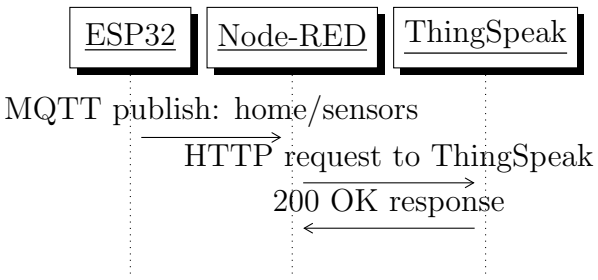

Figura 3: Diagrama de sequência

O diagrama de sequência ilustra a comunicação entre os principais componentes do sistema:

- O **ESP32** recolhe dados dos sensores (temperatura, humidade, movimento, luminosidade) e publica periodicamente os valores via protocolo **MQTT** para o broker local onde o **Node-RED** está a correr.
- O **Node-RED** processa os dados recebidos, aplica lógica de decisão consoante o modo (manual ou automático) e atualiza o dashboard em tempo real.
- Periodicamente ou mediante eventos, o Node-RED envia os dados para a cloud, neste caso o **ThingSpeak**, utilizando requisições HTTP para que os dados fiquem registados e possam ser analisados em gráficos históricos.
- Por fim, o **ThingSpeak** responde com um código de sucesso (200 OK), confirmando a receção e armazenamento dos dados.

Este fluxo permite uma arquitetura reativa, onde o ESP32 apenas envia dados e responde a comandos, enquanto o processamento mais complexo e visualização são tratados no servidor local e na cloud.

3.3 Deployment Diagram

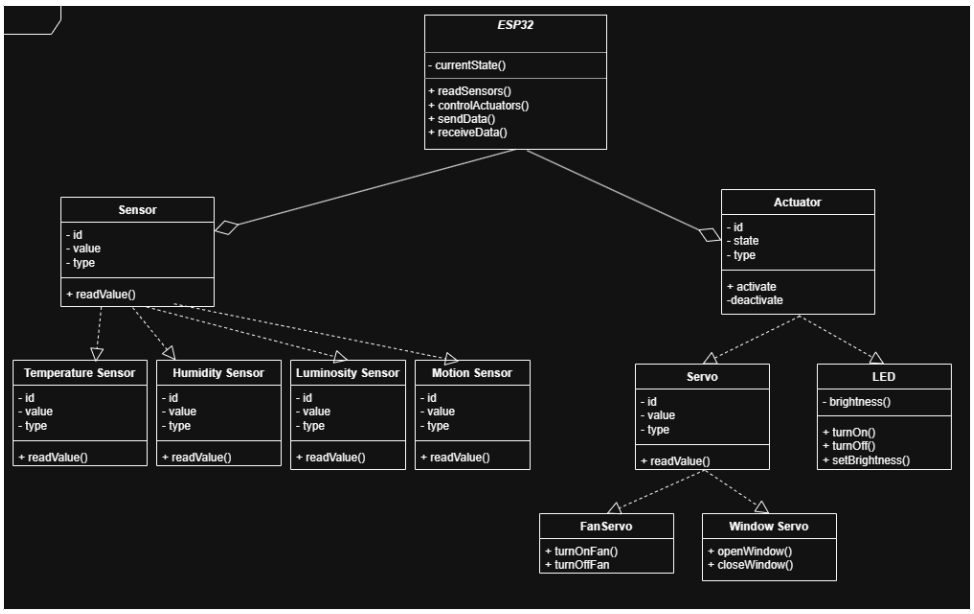

Figura 4: Diagrama de Deployment físico

Este diagrama representa a estrutura física do sistema. O ESP32 está conectado a sensores e atuadores, que obtêm dados do ambiente e atuam sobre ele, respectivamente. O ESP realiza leituras sucessivas e envia-as via MQTT, ao mesmo tempo que também recebe de volta instruções.

4 Funcionalidades e Automatismos

4.1 Publicação via MQTT

- `home/sensors` → temp_in, temp_out, humidade, movimento, luz
- `home/status` → estado do ventilador e da janela

4.2 Subscrição de comandos MQTT

- `home/fan/set` → controlo remoto do ventilador
- `home/window/set` → controlo remoto da janela
- home/light/set -¿ controlo da luz

4.3 Visualização e Dashboards

- Interface Node-RED com gauges e gráficos
- ThingSpeak com gráficos históricos e analytics

- App mobile (ou Home Assistant) para controlo remoto

5 Conclusões

Resumo dos resultados, integração total das plataformas, vantagens e possíveis melhorias (alertas, predições, comandos por voz, integração com Google Home, etc.).

O sistema desenvolvido atingiu os seus objetivos principais: ler dados ambientais em tempo real, processá-los localmente e tomar decisões automáticas com base em regras definidas. A integração com o Node-RED revelou-se eficaz para visualização, controlo e publicação de dados para a cloud (ThingSpeak).

Para o desenvolvimento deste sistema inteligente, foi necessário consultar a documentação fornecida pelos docentes da disciplina, assim como investigar as melhores plataformas para cada tarefa (ex: MQTT, Node-RED, draw.io, HomeAssistant). Acreditamos que o conhecimento adquirido é bastante útil para projetos futuros, académicos ou profissionais.

Durante os testes do sistema, foi possível medir os tempos médios de resposta entre a deteção de um evento (ex: movimento ou variação de temperatura) e a atuação do sistema:

- **Publicação MQTT e processamento local (Node-RED):** 2000ms
- **Latência para envio e confirmação no ThingSpeak:** 15000ms

Estes valores indicam que o sistema opera com uma latência bastante reduzida em contexto local, garantindo boa responsividade na automação. A latência na cloud não compromete o funcionamento, visto que o envio para a ThingSpeak é apenas para logging e análise.

Como principais limitações, destaca-se a ausência de persistência do estado dos atuadores em caso de falha de energia ou reinício do ESP32, e a inexistência de autenticação na ligação MQTT, o que compromete a segurança do sistema. Esta falha de segurança pode ser resolvida com a autenticação por utilizador/password no broker MQTT.

Possíveis melhorias futuras incluem:

- Adição de alertas (ex: e-mail, notificação)
- Suporte a comandos por voz via Google Home
- Predição de padrões através da IA com base em históricos

O sistema foi concebido com modularidade em mente, permitindo fácil expansão com novos sensores (por exemplo qualidade do ar, CO2, sensores de chuva) e funcionalidades (por exemplo abertura de estores, integração com assistentes virtuais). A arquitetura baseada em tópicos MQTT e fluxos Node-RED torna a manutenção e evolução do sistema acessível, mesmo para utilizadores com conhecimentos intermédios de IoT.

Este projeto permitiu aplicar na prática conhecimentos sobre IoT, protocolos MQTT, automação com Node-RED e integração cloud, reforçando competências em sistemas embarcados e conectividade.

Infelizmente houve problemas com o HomeAssistant, que não foi concluído com sucesso, pelo que pode ser completado até à discussão.

6 Referências

- https://thingspeak.com
- https://nodered.org
- https://www.home-assistant.io
- https://app.diagrams.net
- Slides da UC DTSD 2025

7 Anexos

7.1 Código ESP32

Listing 1: Código do ESP32

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <ESP32Servo.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

// ======= CONFIG WIFI & MQTT =======
const char* ssid = "DoNotConnect";
const char* password = "";
const char* mqtt_server = "192.168.162.58";

WiFiClient espClient;
PubSubClient client(espClient);

// ======= OLED SH1107 =======
Adafruit_SH1107 display = Adafruit_SH1107(64, 128, &Wire);

// ======= SENSORS =======
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

#define DS18B20_PIN 5
OneWire oneWire(DS18B20_PIN);
DallasTemperature ds18b20(&oneWire);

#define PIR_PIN 25
#define LDR_PIN 34

```

```
// ======= ACTUATORS =======
Servo fanServo;
Servo windowServo;
#define FAN_SERVO_PIN 19
#define WINDOW_SERVO_PIN 18
#define LED_PIN 14

// ======= STATES =======
String fanState = "OFF";
String windowState = "CLOSED";
String lightState = "OFF";
String mode = "auto";

// ======= CALLBACK MQTT =======
void callback(char* topic, byte* payload, unsigned int length) {
  String command;
  for (int i = 0; i < length; i++) command += (char)payload[i];

  if (String(topic) == "home/fan/set") {
    fanState = (command == "ON") ? "ON" : "OFF";
  }

  if (String(topic) == "home/window/set") {
    windowState = (command == "OPEN") ? "OPEN" : "CLOSED";
    windowServo.write(windowState == "OPEN" ? 180 : 0);
  }

  if (String(topic) == "home/light/set") {
    lightState = (command == "ON") ? "ON" : "OFF";
    digitalWrite(LED_PIN, lightState == "ON" ? HIGH : LOW);
  }

  if (String(topic) == "home/mode") {
    if (command == "manual" || command == "auto") {
      mode = command;
      Serial.println("Mode Updated to: " + mode);
    }
  }

}

// ======= RECONNECT MQTT =======
void reconnect() {
  while (!client.connected()) {
    Serial.print("Connecting to MQTT...");
    if (client.connect("ESP32Client")) {
      Serial.println("connected");
      client.subscribe("home/fan/set");
      client.subscribe("home/window/set");
      client.subscribe("home/light/set");
      client.subscribe("home/mode");

    } else {
      Serial.print("failed, rc=");
      Serial.println(client.state());
      delay(2000);
    }
  }
```

```
}

// ======= SETUP =======
void setup() {
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  dht.begin();
  ds18b20.begin();

  pinMode(PIR_PIN, INPUT);
  pinMode(LDR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  digitalWrite(LED_PIN, LOW);

  fanServo.attach(FAN_SERVO_PIN);
  windowServo.attach(WINDOW_SERVO_PIN);
  fanServo.write(90);
  windowServo.write(0);

  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);

  Wire.begin(21, 23); // SDA, SCL
  if (!display.begin(0x3C, true)) {
    Serial.println("OLED init failed");
    while (true);
  }
  display.setRotation(1);
  display.setTextSize(1);
  display.setTextColor(SH110X_WHITE);
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Sistema Iniciado!");
  display.display();
}

// ======= LOOP =======
void loop() {
  if (!client.connected()) reconnect();
  client.loop();

  static unsigned long lastRead = 0;
  static unsigned long lastFanMove = 0;
  static int fanPos = 90;
  static int fanDir = 1;

  unsigned long now = millis();

  // FAN MOVIMENTO OSILATRIO
  if (fanState == "ON") {
    if (now - lastFanMove > 30) {
      fanPos += fanDir * 2;
      if (fanPos >= 120 || fanPos <= 60) fanDir *= -1;
      fanServo.write(fanPos);
      lastFanMove = now;
    }
  } else {
    fanServo.write(90); // parado
```

```
149   }
150
151   //  PUBLICAO  E OLED a cada 2 segundos
152   if (now - lastRead > 2000) {
153     lastRead = now;
154
155     float tempIn = dht.readTemperature();
156     float hum = dht.readHumidity();
157     ds18b20.requestTemperatures();
158     float tempOut = ds18b20.getTempCByIndex(0);
159     int lux = 4095 - analogRead(LDR_PIN);
160     bool motion = digitalRead(PIR_PIN);
161
162     // MQTT: sensores
163     String sensorPayload = "{";
164     sensorPayload += "\"temp_in\":" + String(tempIn) + ",";
165     sensorPayload += "\"temp_out\":" + String(tempOut) + ",";
166     sensorPayload += "\"humidity\":" + String(hum) + ",";
167     sensorPayload += "\"motion\":" + String(motion ? "true" : "false") + ",";
168     sensorPayload += "\"lux\":" + String(lux);
169     sensorPayload += "}";
170
171     client.publish("home/sensors", sensorPayload.c_str());
172
173     // MQTT: status
174     String statusPayload = "{";
175     statusPayload += "\"fan\":\"" + fanState + "\",";
176     statusPayload += "\"window\":\"" + windowState + "\"";
177     statusPayload += "}";
178
179     client.publish("home/status", statusPayload.c_str());
180
181     // OLED DISPLAY
182     display.clearDisplay();
183     display.setCursor(0, 0);
184
185     display.print("MQTT: ");
186     display.println(client.connected() ? "ON" : "OFF");
187
188     display.print("Modo: ");
189     display.println(mode == "manual" ? "MANUAL" : "AUTO");
190
191     display.print("IN: ");
192     display.print(tempIn, 1); display.println(" C");
193
194     display.print("OUT: ");
195     display.println(tempOut == -127.0 ? "N/A" : String(tempOut, 1) + " C");
196
197     display.print("Hum: ");
198     display.print(hum, 1); display.println(" %");
199
200     display.print("LUX: ");
201     display.println(lux);
202
203     display.print("MOTION: ");
204     display.println(motion ? "SIM" : "NAO");
205
206     display.print("Fan: ");
```

```
207     display.print(fanState);
208     display.print(" Win: ");
209     display.println(windowState);
210
211     display.print("LED: ");
212     display.println(lightState);
213
214     display.display();
215
216     Serial.println(sensorPayload);
217     Serial.println(statusPayload);
218   }
219 }
```

7.2 Capturas do Dashboard

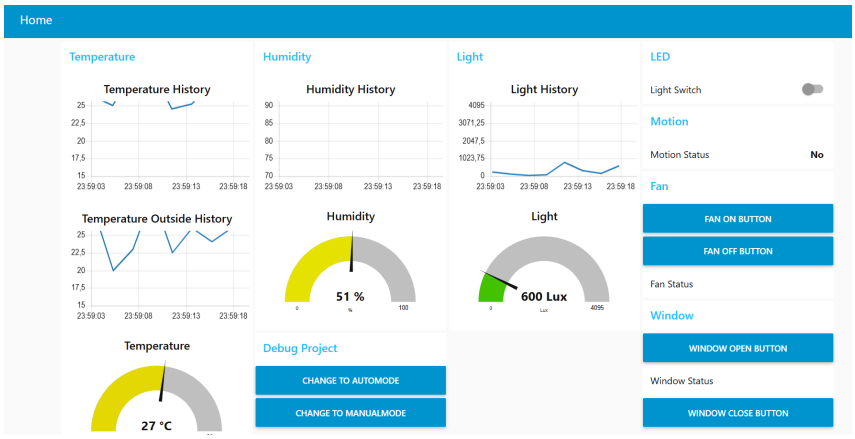

Figura 5: Dashboard Node-RED em tempo real

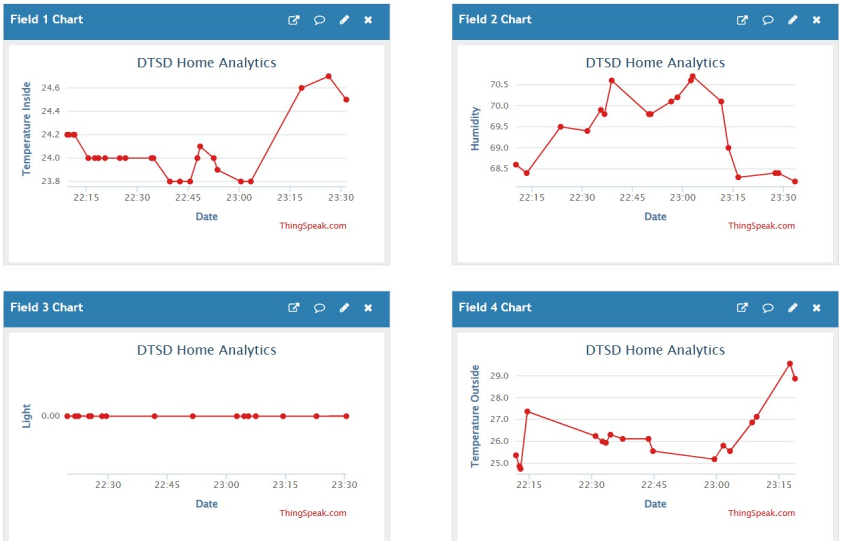

Figura 6: Dashboard Thingspeak em tempo real

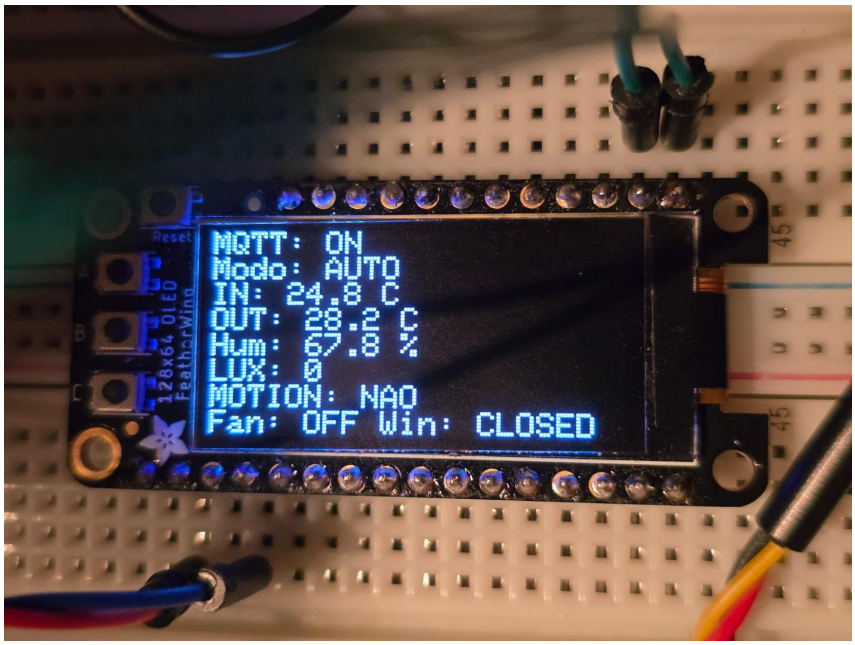

Figura 7: OLED Com Info