Instituto Superior de Tecnologias Avançadas do Porto

ESTAÇÃO METEOROLÓGICA COM ESP32

Diogo Pereira Nº2024152
Fabiano Almeida Nº2024069
Ricardo Oliveira Nº2024015
Tiago Pinto Nº2024145
CIB, 1º ANO

PII - Lab. de Hardware para CIB

01/07/2025

Índice

Introdução

O monitoramento ambiental é uma necessidade crescente em diversas áreas, como agricultura, meteorologia, automação residencial e estudos climáticos. Nesse contexto, as estações meteorológicas desempenham um papel fundamental ao fornecer dados em tempo real sobre variáveis ambientais, como temperatura e humidade.

Com o avanço da eletrônica e da Internet das Coisas (IoT), tornou-se viável desenvolver estações meteorológicas de baixo custo utilizando microcontroladores modernos como o ESP32. Esse microcontrolador é ideal para aplicações IoT por integrar Wi-Fi e Bluetooth, além de oferecer capacidade de processamento suficiente para lidar com múltiplos sensores.

Este projeto, desenvolvido no âmbito da disciplina PII - Laboratório de Hardware para CIB, tem como objetivo construir uma estação meteorológica baseada no ESP32, capaz de coletar dados ambientais através de sensores digitais como o DHT11(sensor de temperatura e humidade) e como o sensor SR501(sensor de movimento) e exibi-los em tempo real através de uma interface web acessível via rede local. O projeto visa aplicar conhecimentos práticos de eletrônica digital, sensores, comunicação de dados e programação embarcada.

Desenvolvimento

Materiais e Ferramentas

Para a construção da estação meteorológica, foram utilizados os seguintes componentes:

- Microcontrolador ESP32 DevKit
- Sensor de temperatura e umidade DHT11
- Sensor de movimento SR501
- LEDs e botões
- Resistors e Breadboard
- Cabo micro-USB e powerbank para alimentação
- Software: Arduino IDE, bibliotecas WiFi e do sensor

Montagem e Programação

A montagem do circuito foi realizada em uma breadboard.

O sensor de temperatura e humidade (DHT11) foi ligado à entrada digital 19 do “ESP32”, utilizando alimentação de 3.3V.

O sensor de movimento (SR501) foi ligado à entrada digital 23 do “ESP32”, utilizando alimentação de 3.3V.

Acrescentamos dois LEDs, um caso o sistema esteja Online e outro caso o sistema esteja Offline, com o apoio de um botão para permitir o controlo do estado do sistema.

O “ESP32” está a ser alimentado por uma powerbank via cabo micro-usb.

A programação foi feita no Arduino IDE, com a utilização da biblioteca DHT.h para leitura do sensor e da biblioteca WiFi.h para criação de uma rede sem fio, para permitir o acesso ao servidor WEB alojado no “ESP32” . O ESP32 foi configurado como servidor web, capaz de responder a requisições HTTP exibindo uma página HTML com os dados recebidos pelo sensor de temperatura e humidade.

Código Utilizado

```
// ***** CTESP CyberSecurity *****
// ******* Weather Station *******
// ******** VERSION 1.0.0 ********
// ** Created and Developed by: **
// ****** Ricardo Oliveira *******
// ******** Diogo Pereira ********
// ********* Tiago Pinto *********
// ******* Fabiano Almeida *******

// Libraries
#include <WiFi.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ESPAsyncWebServer.h>
#include <AsyncTCP.h>
#include <DHT.h>

// OLED Display Dimensions
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

// OLED, Sensors and Global Variables Setup
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
#define DHTTYPE DHT11
const int powerButtonPin = 5;
const int motionSensorPin = 23;
const int dhtSensorPin = 19;
const int ledOnPin = 15;
const int ledOffPin = 4;
DHT dht(dhtSensorPin, DHT11);
```

```
bool systemOn = false;

// WiFi AP Credentials
const char* ssid = "WeatherStation";
const char* password = "Weather1234";

// Web Server Port 80
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

// Web Server Code
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Weather Station Dashboard</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background: #121212;
      color: #fff;
      display: flex;
      flex-direction: column;
      align-items: center;
      padding: 20px;
    }
    h1 {
      margin-bottom: 20px;
    }
    .card {
```

```
      background: #1e1e1e;
      padding: 20px;
      border-radius: 12px;
      box-shadow: 0 0 10px rgba(0,0,0,0.3);
      margin: 10px;
      text-align: center;
      width: 250px;
    }
    .value {
      font-size: 2.5em;
      font-weight: bold;
    }
    .label {
      font-size: 1em;
      color: #aaa;
    }
    .container {
      display: flex;
      gap: 20px;
      flex-wrap: wrap;
      justify-content: center;
    }
  </style>
</head>
<body>
  <h1>🌤️ Weather Station</h1>
  <div class="container">
    <div class="card">
      <div class="value" id="temp">-- °C</div>
      <div class="label">Temperature</div>
    </div>
    <div class="card">
```

```
   <div class="value" id="hum">-- %</div>
   <div class="label">Humidity</div>
  </div>
 </div>

 <script>
  const ws = new WebSocket(`ws://${location.host}/ws`);
  ws.onmessage = event => {
   const data = JSON.parse(event.data);
   document.getElementById("temp").textContent = `${data.temp} °C`;
   document.getElementById("hum").textContent = `${data.hum} %`;
  };
  ws.onopen = () => console.log("WebSocket connected");
  ws.onclose = () => console.log("WebSocket disconnected");
 </script>
</body>
</html>
)rawliteral";

// Bitmap WeatherStation Logo
const unsigned char weatherStationLogoBitmap [] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x01, 0xf0, 0x00, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x01, 0xf8, 0x00, 0x00, 0x3e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x07, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x01, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x07, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xff, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x3f, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x3f, 0xe0, 0x0f, 0xff, 0xf0, 0x00, 0x7f, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x1f, 0xc0, 0x0f, 0xff, 0xff, 0x00, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x07, 0x03, 0xff, 0xff, 0xff, 0xc0, 0x7e, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0x80, 0x7f, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x1f, 0xfc, 0x00, 0x3f, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x7f, 0xf0, 0x00, 0x01, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00, 0x01, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00, 0x00, 0x3f, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00, 0x00, 0x3f, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xfc, 0x00, 0x00, 0x00, 0x01, 0xff, 0xfe, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x07, 0xfc, 0x00, 0x00, 0x00, 0x01, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xe0, 0x00, 0x00, 0x00, 0x0f, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xe0, 0x00, 0x00, 0x00, 0x2f, 0xff, 0xff, 0xf8, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xe0, 0x00, 0x00, 0x00, 0x7f, 0xf0, 0x00, 0xff, 0x00, 0x00, 0x00, 0x00,
0x00, 0x3f, 0xfe, 0x03, 0xe0, 0x00, 0x00, 0x00, 0xff, 0xf0, 0x00, 0x7f, 0x80, 0x00, 0x00, 0x00,
0x00, 0x7f, 0xff, 0x03, 0xe0, 0x00, 0x00, 0x01, 0xf8, 0x00, 0x00, 0x3f, 0xe0, 0x00, 0x00, 0x00,
0x00, 0xff, 0xff, 0x03, 0xe0, 0x00, 0x00, 0x07, 0xf8, 0x00, 0x00, 0x07, 0xf8, 0x00, 0x00, 0x00,
0x00, 0xff, 0xff, 0x03, 0xe0, 0x00, 0x00, 0x0f, 0xf0, 0x00, 0x00, 0x07, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xe0, 0x00, 0x00, 0x1f, 0xc0, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xe0, 0x00, 0x00, 0x0f, 0xc0, 0x00, 0x00, 0x00, 0xfc, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xf8, 0x00, 0x4c, 0x7f, 0xc0, 0x00, 0x00, 0x00, 0xfe, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x03, 0xf8, 0x00, 0xff, 0xff, 0xc0, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xfc, 0x03, 0xff, 0xff, 0xc0, 0x00, 0x00, 0x00, 0xff, 0xc0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xfe, 0x07, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0xff, 0xe0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xff, 0x1f, 0xff, 0xff, 0x00, 0x00, 0x00, 0x00, 0xff, 0xfc, 0x00, 0x00,

```
  0x00, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x1f, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x60, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x00, 0xf0, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x03, 0xf8, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x07, 0xf0, 0xfc, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x1f, 0xe0, 0xff, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x3f, 0x80, 0x3f, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xe0, 0x00,
  0x00, 0x00, 0x00, 0x3f, 0x00, 0x3f, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xff, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfe, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

// Bitmap 16x16 Motion Alert Icon
const unsigned char motionIcon[] PROGMEM = {
 0x00, 0x00,
 0x00, 0x00,
 0x01, 0x80,
 0x01, 0x80,
 0x01, 0x80,
 0x01, 0x80,
 0x01, 0x80,
```

```
  0x01, 0x80,
  0x01, 0x80,
  0x01, 0x80,
  0x01, 0x80,
  0x01, 0x80,
  0x01, 0x80,
  0x00, 0x00,
  0x01, 0x80,
  0x00, 0x00
};

// Bitmap 16x16 Thermometer Icon
const unsigned char thermometerIcon[] PROGMEM = {
  0x01, 0xC0,
  0x03, 0xE0,
  0x07, 0x20,
  0x07, 0xE0,
  0x07, 0x20,
  0x07, 0xE0,
  0x07, 0x20,
  0x07, 0xE0,
  0x07, 0x20,
  0x0F, 0xF0,
  0x1F, 0xF8,
  0x1F, 0xF8,
  0x1F, 0xF8,
  0x1F, 0xF8,
  0x0F, 0xF0,
  0x07, 0xE0
};

// Bitmap 8x8 Degree Symbol
```

```
const unsigned char degreeSymbol[] PROGMEM = {
  0x38, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00
};

// Bitmap 16x16 Humidity Icon
const unsigned char humidityIcon[] PROGMEM = {
  0x00, 0x00, 0x01, 0x80, 0x03, 0xC0, 0x07, 0xE0,
  0x0F, 0xF0, 0x0F, 0xF0, 0x1F, 0xF8, 0x1F, 0xD8,
  0x3F, 0x9C, 0x3F, 0x9C, 0x3F, 0x1C, 0x1E, 0x38,
  0x1F, 0xF8, 0x0F, 0xF0, 0x03, 0xC0, 0x00, 0x00
};

// Bitmap 16x16 Error Triangle Icon
const unsigned char errorIcon[] PROGMEM = {
  0x00, 0x80,
  0x01, 0xC0,
  0x01, 0xC0,
  0x03, 0xE0,
  0x03, 0x60,
  0x07, 0x70,
  0x06, 0x30,
  0x0E, 0xB8,
  0x0C, 0x98,
  0x1C, 0x9C,
  0x18, 0x8C,
  0x38, 0x0E,
  0x30, 0x86,
  0x7F, 0xFF,
  0x7F, 0xFF,
  0x00, 0x00
};
```

```
// Setup Function (Main)
void setup() {
  Serial.begin(115200);
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64
    Serial.println(F("OLED Display failed"));
    for(;;);
  }

  configPin();
  initialize();
}

// Main Loop
void loop() {
  powerState();

  if (systemOn) {
    static unsigned long lastDHTRead = 0;
    static int lastMotionState = LOW;

    unsigned long now = millis();

    // Check motion every loop iteration
    int currentMotionState = digitalRead(motionSensorPin);
    if (currentMotionState != lastMotionState) {
      if (currentMotionState == HIGH) {
        motionDetected();  // Motion started
        delay(10000);
      } else {
        noMotionDetected(); // Motion stopped
      }
      lastMotionState = currentMotionState;
```

```
    }

    // Read and send DHT data every 2 seconds
    if (now - lastDHTRead >= 2000) {
      float temperature = dht.readTemperature();
      float humidity = dht.readHumidity();
      if (!isnan(temperature) && !isnan(humidity)) {
        notifyClients(temperature, humidity);
      }
      lastDHTRead = now;
    }
  }
}

// Helper Functions for Text in Display
struct TextPosition {
  int x;
  int y;
};

TextPosition getCenteredXY(const String &text, int textSize = 1, int displayWidth = 128, int displayHeight =
64) {
  int16_t x1, y1;
  uint16_t w, h;
  display.setTextSize(textSize);
  display.getTextBounds(text, 0, 0, &x1, &y1, &w, &h);
  int x = (displayWidth - w) / 2;
  int y = (displayHeight - h) / 2;
  return {x, y};
}

void drawDegreeSymbol(int x, int y) {
```

```
  display.drawBitmap(x, y, degreeSymbol, 8, 8, SSD1306_WHITE);
}

void printTypingEffect(const String &text, int x, int y, int textSize = 1, int delayMs = 100, bool
clearDisplayFirst = false, bool isTemp = false) {
  display.setTextSize(textSize);
  display.setTextColor(SSD1306_WHITE);
  if (clearDisplayFirst) {
    display.clearDisplay();
  }
  display.setCursor(x, y);

  for (int i = 0; i < text.length(); i++) {
    if (text[i] == '°') { // Handle degree symbol
      drawDegreeSymbol(display.getCursorX(), y);
      display.setCursor(display.getCursorX() + 10, y); // Move cursor past symbol
    } else {
      display.print(text[i]);
    }
    display.display();
    delay(delayMs);
  }

  if (isTemp) {
    drawDegreeSymbol(display.getCursorX(), y);
    display.setCursor(display.getCursorX() + 10, y);
    display.print("C");
    display.display();
  }
}

// Function to Configure Pins
```

```
void configPin() {
  pinMode(powerButtonPin, INPUT_PULLUP);
  pinMode(motionSensorPin, INPUT);
  pinMode(ledOnPin, OUTPUT);
  pinMode(ledOffPin, OUTPUT);

  digitalWrite(ledOnPin, LOW);
  digitalWrite(ledOffPin, LOW);

  dht.begin();
}

// Function to Initialize and Warm-UP Sensors
void initialize() {

  Serial.println("Starting...");
  TextPosition posStart = getCenteredXY("Starting...", 1);
  printTypingEffect("Starting...", posStart.x, posStart.y, 1, 150, true);
  delay(20000);

  configWiFi();

  Serial.println("Warming Up...");
  TextPosition posWarm = getCenteredXY("Warming Up...", 1);
  printTypingEffect("Warming Up...", posWarm.x, posWarm.y, 1, 150, true);
  delay(40000);

  Serial.println("Ready!");
  TextPosition posReady = getCenteredXY("Ready!", 1);
  printTypingEffect("Ready!", posReady.x, posReady.y, 1, 150, true);
  delay(2000);
```

```
}

// Function to Configure WiFi
void configWiFi () {
  if (WiFi.softAP(ssid, password)) {
  Serial.println("AP WiFi Started! Ready to Receive Connections...");
  } else {
    Serial.println("Failed to Start AP WiFi");
  }
}

// Function to Start the System with Button
void powerState() {
  static int lastState = HIGH;
  static bool firstRun = true;

  int currentState = digitalRead(powerButtonPin);

  if (firstRun) {
    currentState = LOW;
  }

  if (lastState == HIGH && currentState == LOW) {
    systemOn = !systemOn;

    if (systemOn) {
      digitalWrite(ledOnPin, HIGH);
      digitalWrite(ledOffPin, LOW);

      Serial.println("System Online");

      startWebServer();
```

```
      String systemOnText = "System Online";
      TextPosition posSystemOn = getCenteredXY(systemOnText, 1);
      printTypingEffect(systemOnText, posSystemOn.x, posSystemOn.y, 1, 100, true);

      delay(2000);

      weatherStationLogo();
    } else {
      digitalWrite(ledOnPin, LOW);
      digitalWrite(ledOffPin, HIGH);

      stopWebServer();

      Serial.println("System Offline");
      String systemOffText = "System Offline";
      TextPosition posSystemOff = getCenteredXY(systemOffText, 1);
      printTypingEffect(systemOffText, posSystemOff.x, posSystemOff.y, 1, 100, true);
    }
  }

  lastState = currentState;
  firstRun = false;
}

// Function to Start and Configure WebServer
void startWebServer () {
  ws.onEvent(onWsEvent);
  server.addHandler(&ws);
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html);
  });
```

```
  server.begin();
  Serial.println("WebServer Started! Ready to Receive Connections...");
  Serial.println("Access this IP:" );
  Serial.println(WiFi.softAPIP());
}

// Event Handler to Keep WebSocket Alive
void onWsEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type,
               void *arg, uint8_t *data, size_t len) {
}

// Function to Send Sensor Data to Web Server
void notifyClients(float temperature, float humidity) {
  char json[64];
  snprintf(json, sizeof(json), "{\"temp\":%.1f,\"hum\":%.1f}", temperature, humidity);
  ws.textAll(json);
}

// Function to Stop WebServer
void stopWebServer() {
  ws.closeAll();
  server.end();
  Serial.println("WebServer Stopped!");
}

void weatherStationLogo() {
  for (int y = -64; y <= 0; y += 4) {
    display.clearDisplay();
    display.drawBitmap(0, y, weatherStationLogoBitmap, 128, 64, SSD1306_WHITE);
    display.display();
    delay(30);
  }
```

```
  display.clearDisplay();
  display.drawBitmap(0, 0, weatherStationLogoBitmap, 128, 64, SSD1306_WHITE);
  display.display();
}

// Function Triggered when Motion Detected
void motionDetected() {
  Serial.println("Motion detected!");

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

  display.clearDisplay();  // Clear once at the start

  // Draw static icons and line
  display.drawBitmap(0, 0, motionIcon, 16, 16, SSD1306_WHITE);
  display.drawBitmap(112, 0, motionIcon, 16, 16, SSD1306_WHITE);
  display.drawLine(0, 18, 128, 18, SSD1306_WHITE);

  String motionText = "Motion Detected";
  printTypingEffect(motionText, 16, 8, 1, 100, false);

  if (!isnan(temperature) && !isnan(humidity)) {
    // Draw temperature icons
    display.drawBitmap(0, 24, thermometerIcon, 16, 16, SSD1306_WHITE);
    display.drawBitmap(112, 24, thermometerIcon, 16, 16, SSD1306_WHITE);

    // Draw humidity icons
    display.drawBitmap(0, 48, humidityIcon, 16, 16, SSD1306_WHITE);
    display.drawBitmap(112, 48, humidityIcon, 16, 16, SSD1306_WHITE);
```

```
    // Animate temperature text
    String tempText = String(temperature, 1);
    TextPosition posTemp = getCenteredXY(tempText + "°C", 1);
    printTypingEffect(tempText, posTemp.x, 30, 1, 100, false, true);

    // Animate humidity text
    String humText = String(humidity, 1) + " %";
    TextPosition posHum = getCenteredXY(humText, 1);
    printTypingEffect(humText, posHum.x, 54, 1, 100, false);

  } else {
    Serial.println("Failed to read data from DHT sensor!");
    // Draw error icons
    display.drawBitmap(0, 32, errorIcon, 16, 16, SSD1306_WHITE);
    display.drawBitmap(112, 32, errorIcon, 16, 16, SSD1306_WHITE);

    // Animate error text
    String errorText = "DHT11 Error...";
    TextPosition posErr = getCenteredXY(errorText, 1);
    printTypingEffect(errorText, posErr.x, 38, 1, 100, false);
  }
}

// Function Triggered when no Motion Detected
void noMotionDetected() {
  display.clearDisplay();
  weatherStationLogo();
  Serial.println("Waiting for motion...");
}
```

Funcionamento

Ao ser alimentado, o “ESP32” aguarda 20 segundos antes de criar uma rede WiFi com o nome “WeatherStation”. Após criar a rede, aguarda agora 40 segundos antes de ligar o sistema, para “aquecer” os sensores. Em seguida, inicia a leitura periódica dos dados de temperatura e humidade fornecidos pelo sensor. Estes dados são enviados para uma interface web, acessível pelo IP do “ESP32” na rede anteriormente criada.

Estes dados são também enviados para um ecrã OLED, para leitura do utilizador caso esteja perto do mesmo, sendo que o ecrã ativa, quando for detetado movimento pelo sensor de movimento.

A página web é construída com HTML básico e atualiza os dados a cada 2 segundos. Com isso, qualquer dispositivo na mesma rede pode visualizar os dados da estação meteorológica em tempo real, para fácil acesso caso não esteja perto do ecrã.

Resultados Obtidos

Após a conclusão da montagem e programação, o sistema apresentou funcionamento estável. Os dados de temperatura e humidade foram lidos com precisão e exibidos corretamente na página web. O tempo de resposta da interface foi satisfatório e a leitura dos sensores mostrou-se confiável em ambientes internos.

Durante os testes, foi possível observar variações coerentes na temperatura e humidade ao modificar o ambiente (como abrir uma janela ou aproximar o sensor de fontes de calor). Isso demonstra que o sistema pode ser aplicado em situações reais de monitoramento climático básico.

Conclusão

O projeto da estação meteorológica com ESP32 permitiu consolidar os conhecimentos adquiridos ao longo da disciplina de PII - Laboratório de Hardware para CIB, promovendo a integração entre sensores digitais, comunicação Wi-Fi e programação embarcada.

A construção da estação demonstrou que é possível desenvolver soluções funcionais e de baixo custo para monitoramento ambiental utilizando plataformas acessíveis como o ESP32. Além disso, o projeto reforçou a importância do pensamento lógico e da depuração de código em sistemas embarcados.

Como propostas de melhoria, considera-se a adição de novos sensores (como pressão atmosférica ou qualidade do ar), o armazenamento de dados em nuvem e o uso de gráficos em tempo real na interface web. Com esses aprimoramentos, o sistema poderia ser expandido para aplicações mais complexas dentro do contexto da Internet das Coisas.