Universidade Tecnológica Federal do Paraná
Engenharia da Computação

Relatório do Projeto final de Sistemas Embarcados

Aluno: Eduardo Bombonatto Lorenzetti
Aluno: Luiz Fernando da Costa Pereira
Professor orientador: Prof. Dr. Tiago Piovesan Vendruscolo

Fevereiro
2025

Universidade Tecnológica Federal do Paraná
Departamento de Engenharia de Computação

Relatório

Primeiro Relatório de Projeto de Pesquisa do Curso Engenharia de Computação da UTFPR, da matéria de Sistemas Embarcados

Aluno: Eduardo Bombonatto Lorenzetti

Aluno: Luiz Fernando da Costa Pereira

Professor orientador: Prof. Dr. Tiago Piovesan Vendruscolo

Fevereiro
2025

Conteúdo

1 Resumo

O projeto **Climário** foi desenvolvido com o objetivo de replicar diferentes condições climáticas em um ambiente controlado, permitindo a simulação de cenários como tempo ensolarado, tempestuoso, neblinoso e chuvoso. Além disso, foi implementado um **modo local**, no qual o sistema busca informações climáticas em uma API e, somado às informações dos sensores, as reproduz automaticamente.

Para a construção do sistema, utilizamos um **ESP32** como unidade de controle central, sensores de temperatura e umidade (*DHT11* e *BMP180*), uma **fita de LED** para simular variações de iluminação, uma **bomba d'água** para replicar a chuva e uma **placa de circuito impresso (PCB)** projetada para integrar todos os componentes. A interação com o sistema foi inicialmente planejada por meio de uma aplicação em React, mas devido a problemas de *CORS*, optamos por um servidor HTML embutido no ESP32.

Durante o desenvolvimento, enfrentamos desafios técnicos, como a impossibilidade de integrar o **modo neblinoso** devido a incompatibilidades elétricas, problemas de conexão do **ESP32** com redes Wi-Fi acadêmicas e a ausência de botões físicos para controle. Apesar dessas dificuldades, conseguimos implementar grande parte das funcionalidades propostas e estabelecer uma base sólida para futuras melhorias no projeto.

2 Apresentação

O projeto **Climário** é um sistema desenvolvido para replicar condições climáticas selecionadas em ambientes controlados. O objetivo principal é simular diferentes tipos de clima de maneira precisa e dinâmica.

Os climas disponíveis para simulação são:

- **Ensolarado**: Ambiente iluminado com alta luminosidade;
- **Tempestuoso**: Simulação de trovoadas e mudanças bruscas de pressão;
- **Neblinoso**: Redução da visibilidade com simulação de névoa;
- **Chuvoso**: Reprodução de condições de chuva controlada.

Além disso, o Climário conta com um **modo local**, no qual o sistema consulta uma API meteorológica para obter as condições climáticas da região e as reproduz automaticamente no ambiente controlado.

Para implementar essas funcionalidades, utilizamos os seguintes componentes:

- **ESP32**: Microcontrolador responsável pelo processamento e controle do sistema;
- **Fita de LED WS2812b**: Utilizada para simular variações na iluminação de acordo com o clima selecionado;
- **Caixa de vidro**: Estrutura com dimensões $15cm$x$15cm$x$15cm$ onde o ambiente controlado é criado;
- **Sensores DHT11 e BMP180**: Responsáveis por medir temperatura, umidade e pressão atmosférica;
- **Bomba d'água 3.3V-6V**: Utilizada para simular a chuva dentro do ambiente do Climário.

Esse conjunto de hardware e software permite que o sistema reproduza diferentes condições meteorológicas de forma dinâmica e automatizada, oferecendo uma experiência imersiva e controlada.

3 Descrição de Atividades

Para a construção do Climário, desenvolvemos uma **placa de circuito impresso (PCB)** na plataforma *EasyEDA* para conectar e integrar todos os componentes descritos anteriormente. A fim de satisfazer o requisito de ser um projeto embarcado, a PCB conta com conector de alimentação fêmea (conector *jack*), conectado à um regulador de tensão LM7805, com capacitores de filtro. O esquema do circuito pode ser visto na figura 1, e na figura 2 a placa já pronta.

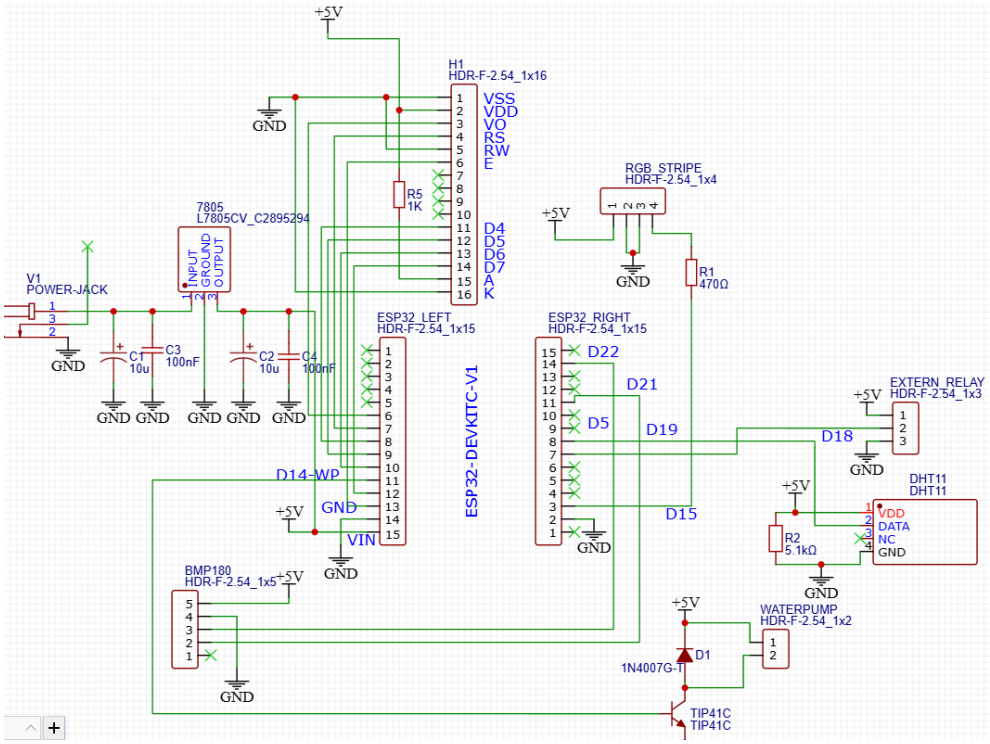

Figura 1: Esquema da PCB no software EasyEDA

Além do hardware, desenvolvemos um código para configurar os diferentes modos climáticos, a conexão Wi-Fi e o controle da fita de LED. O código implementa a lógica de seleção de clima, permitindo ao usuário alternar entre os modos ensolarado, tempestuoso, neblinoso, chuvoso e local, onde o sistema busca as condições meteorológicas da API gratuita e *open-source* OpenMeteo. O código está disponível no GitHub ou no anexo A deste relatório.

Em suma, o código desenvolvido abre um servidor HTTP na rede local, permitindo apresentação das leituras médias dos sensores de pressão e temperatura. Após isso, é aplicado o modo inicial como Ensolarado até seja

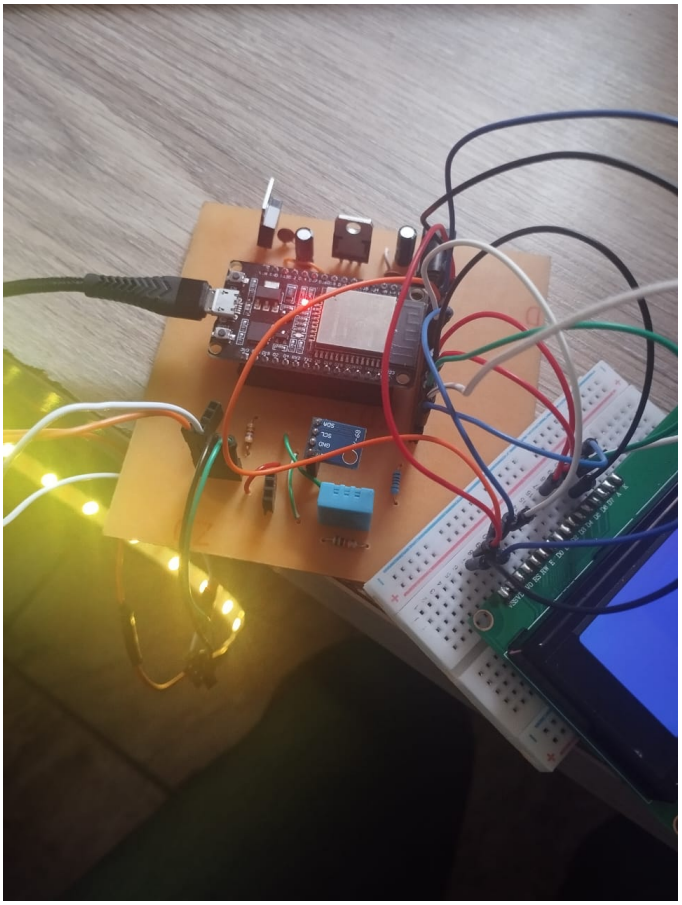

Figura 2: PCB desenvolvida para o Climário

trocado através de um comando do cliente conectado no servidor. A página utilizada foi adaptada a partir de um projeto já existente do usuário "Enjoy-Mechatronics", disponível no link. A imagem 3 mostra a página do servidor e os dados apresentados, com uma caixa de seleção para troca de modo do Climário.

Um fator bastante importante foi o desenvolvimento de um gotejador feita na impressora 3D. Possui medidas precisas para conectar a bomba d'água e furos de 1 mm de diâmetro para simular a chuva, além de conter um espaço reservado para o acoplamento da fita RGB.

4 Análise dos Resultados

Os resultados obtidos durante o desenvolvimento do Climário não foram totalmente satisfatórios, pois encontramos diversas dificuldades técnicas que

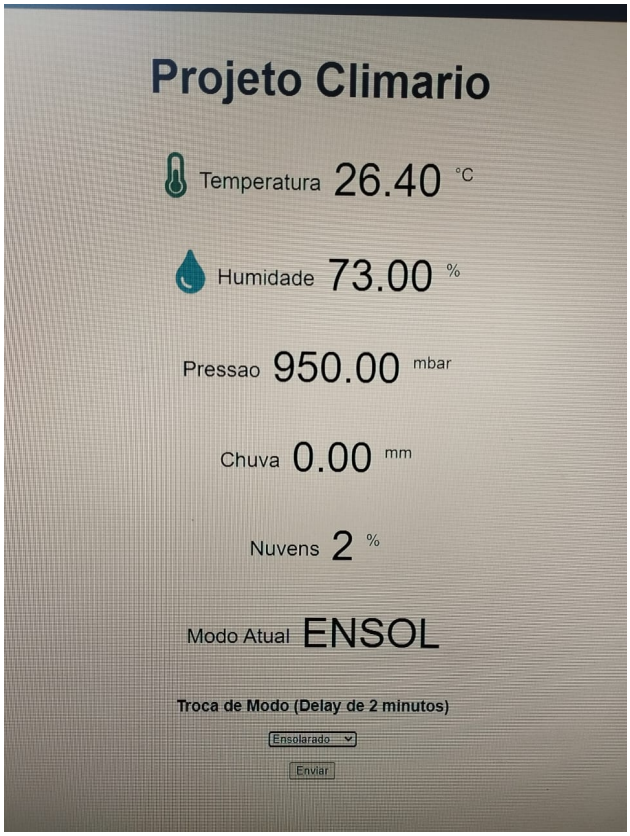

Figura 3: Página do servidor HTTP

impactaram a implementação do projeto conforme planejado.

Como o foi adotado um método de fabricação "caseiro"da PCB, ela foi limitada a apenas conter trilhas grossas em apenas uma camada, sem malha de terra. Como solução, foram soldados alguns *jumpers* para continuidade de corrente.

Um dos principais problemas foi a impossibilidade de implementar o **modo neblinoso**. Inicialmente, planejamos utilizar um nebulizador para criar o efeito desejado, porém, devido à incompatibilidade das correntes elétricas, tornou-se necessário o uso de um **módulo relé externo**, que está presente no projeto, mas não foi testado. Como resultado, não conseguimos integrar esse recurso ao sistema final.

Outro desafio significativo foi a conectividade do **ESP32** com redes Wi-Fi. Durante os testes, observamos que o microcontrolador funcionava corretamente apenas em **redes domésticas**, mas não conseguia se conectar à rede da faculdade nem ao hotspot do celular. Essa limitação comprometeu a funcionalidade do modo local, que dependia da obtenção de informações

climáticas por meio de uma API.

Além disso, um erro de planejamento resultou na **ausência de botões físicos** para a interação com o sistema. Para contornar essa limitação, desenvolvemos inicialmente uma **aplicação em React** para permitir o controle remoto do Climário. No entanto, enfrentamos problemas com o **CORS** (Cross-Origin Resource Sharing), que impediram a comunicação adequada entre o ESP32 e a aplicação.

Para resolver essa questão, optamos por substituir a aplicação em React por um **servidor HTML embutido no próprio ESP32**. Dessa forma, conseguimos exibir as informações e permitir a interação com o Climário diretamente via navegador, eliminando a necessidade de configurações adicionais de rede.

Apesar dos desafios enfrentados, os ajustes realizados permitiram que o sistema funcionasse, viabilizando a replicação de alguns climas dentro do ambiente controlado.

As imagens 4 e 5 mostram o resultado final, e alguns vídeos do projeto podem ser vistos no repositório no Github.

Cada modo de clima é definido por um conjunto de três variáveis: animação das LEDs, chuva e neblina. Entretanto, como não foi possível testar o nebulizador no projeto, a versão do código atual considera apenas as duas primeiras. Assim, definiu-se que o modo Ensolarado apresenta a respectiva animação das LEDs (um gradiente entre amarelo e laranja) e sem chuva; o modo Chuvoso tem a animação "neblina"(também seria utilizado no modo Neblineiro) com os LEDs em cinza, e com acionamento da bomba d'água; o último modo, Tempestuoso, também possui chuva e uma animação das LEDs de modo a simular trovoadas distantes e randômicas, tanto em localidade (escolha de LEDs aleatórios) quanto no intervalo de tempo entre elas.

5 Trabalhos Futuros

Com base nos desafios encontrados durante o desenvolvimento do Climário, identificamos diversas melhorias que podem ser implementadas em versões futuras do projeto:

- **Integração do modo neblinoso**: Testar o uso do nebulizador utilizando um módulo relé nos pinos reservados na PCB, permitindo a simulação realista de neblina.
- **Melhoria na conectividade Wi-Fi**: Explorar alternativas para que o ESP32 possa se conectar a redes corporativas e hotspots móveis sem restrições.

Figura 4: Modo Ensolarado

- **Adição de botões físicos**: Implementar controles manuais para facilitar a interação com o sistema sem depender exclusivamente da interface web.
- **Otimização da interface de controle**: Melhorar a aplicação web, revisando as políticas de *CORS* e explorando alternativas para permitir o uso da versão em React.
- **Expansão dos modos climáticos**: Explorar novas possibilidades de simulação, como variações de vento ou mudanças graduais na temperatura e umidade.
- **Aprimoramento da estrutura física**: Melhorar o design da caixa de vidro e a organização interna dos componentes para aumentar a eficiência e durabilidade do sistema.

Figura 5: Modo Chuvoso

- **Reescrever o código para suportar multitarefa**: O código conta com diversos eventos que ocorrem periodicamente, e cada `delay` interfere nos momentos em que esses eventos são executados. Por isso, sugere-se que, como proposto inicialmente nesse projeto, utilizar a biblioteca FreeRTOS para que as medidas dos sensores e coleta de dados da API ocorram de forma independentes.

Essas melhorias permitirão que o Climário se torne um sistema mais robusto, funcional e versátil, ampliando suas aplicações em diferentes áreas, como pesquisa, educação e experimentação ambiental.

Bibliografia

AGUIRRE, L. A. Introdução à Identificação de Sistemas, Técnicas Lineares e Não lineares Aplicadas a Sistemas Reais. Belo Horizonte, Brasil, EDUFMG. 2004.

ANEXO A

Código no feito no Arduino IDE 2.3.4 também disponível no GitHub. Código '`climario.ino`' versão final da disciplina de Sistemas Embarcados 2024-2.

```
#include <DHT.h>
#include <Adafruit_BMP085.h>
#include <LiquidCrystal.h>
#include <time.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <FastLED.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

// Fita LED RGB
#define LED_PIN     15
#define NUM_LEDS    16
#define BRIGHTNESS  64
#define LED_TYPE    WS2812
#define COLOR_ORDER GRB
CRGB leds[NUM_LEDS];
#define UPDATES_PER_SECOND 100

// Pinos de saida
#define PIN_CHUVA 14
#define PIN_NEBUL 19

// API Open-meteo retorna JSON com precipitacao e chuva (mm), velocidade (km/h)
// As variaveis apresentadas podem ser escolhidas no site da API: https://open-
String API_URL = "https://api.open-meteo.com/v1/forecast?latitude=52.52&longitu

// Sensores
DHT dht(5, DHT11);
Adafruit_BMP085 bmp;
String modoAtivo = ""; // guarda o modo atual = {ENSOL, CHUVA, TEMPS, NEBLN, LU

// WiFi
const char ssid[] = "nome-da-rede";
const char password[] = "senha-da-rede";
const char* hostname = "Climario";
// Servidor WEB
AsyncWebServer server(80);
// NTP
long timezone = -3;
byte daysavetime = 1;
struct tm timeinfo;
```

```
// ------------------------------------------------------------------------


// Variaveis para controle de tempo
// Intervalo para as medidas (2 minutos = 120000ms)
const unsigned long INTERVALO = 120000;
// Intervalo para calculo da media (20 minutos = 10 requisicoes)
const int NUM_MEDIDAS = 10;
float temperatura[NUM_MEDIDAS];
float pressao[NUM_MEDIDAS];
float humidade[NUM_MEDIDAS];
int indiceFila = 0;
int totalMedidas = 0;
unsigned long ultimaMedida = 0;

// Variaveis do modo LOCAL
int setupLOCAL = 1; // para quando o modo for trocado, atualiza o clima com va
float mediaTemp = 0.0;
float mediaHum = 0.0;
float mediaPress = 0.0;

// Variaveis da requisicao POST
String Mode; // armazena comando que vem do cliente
int houveTroca = 0; // mudara para 1 se houver requisicao POST
unsigned long ultimaTrocaModo = 0;
const int ESPERA_TROCA = 2*60000; // 2min x 60.000ms

// Variaveis a definir
String date, hour;
int cloud_cover, weather_code;
float wind_speed, rain;
// ------------------------------------------------------------------------
void atualizaDataHora()
{
  date = "dd/mm/aaaa";
  hour = "hh:mm";
  timeinfo.tm_year = 0; // since 1900; tm_mon=0 (January)
  if (!getLocalTime(&timeinfo)) {;
    Serial.println("Connection Err");
  } else {
    String date = (String(timeinfo.tm_mday) + "/" +  String((timeinfo.tm_mon) +
    String hour = (String(timeinfo.tm_hour) + ":" + String(timeinfo.tm_min));
  }
  Serial.println("Data: " + date + " - Hora: " + hour);
}

void leDadosAPI()
{
```

```
  HTTPClient http;
  http.begin(API_URL);
  int code = http.GET();
  if(code > 0){
    String JSON_Data = http.getString();
    // Obtendo o objeto JSON
    DynamicJsonDocument doc(2048);
    deserializeJson(doc, JSON_Data);
    JsonObject obj = doc.as<JsonObject>();

    // Acessando as informacoes do objeto
    cloud_cover = obj["current"]["cloud_cover"].as<int>();
// %
    wind_speed = obj["current"]["wind_speed_10m"].as<float>();
// km/h
    rain = obj["current"]["rain"].as<float>();
// mm
    weather_code = obj["current"]["weather_code"].as<int>();
    // teste
    Serial.println("Nuvens: "+String(cloud_cover)+"% Vento: "+String(wind_speed
  } else {// caso requisicao falha
    Serial.println("Error: " + String(code));
  }
}

void calcularMedias() {
  float somaTemp = 0.0;
  float somaPress = 0.0;

  // Calcula a soma de todos os valores no array
  for (int i = 0; i < NUM_MEDIDAS; i++) {
    somaTemp += temperatura[i];
    somaPress += pressao[i];
  }

  // Calcula medias
  mediaTemp = somaTemp / NUM_MEDIDAS;
  mediaPress = somaPress / NUM_MEDIDAS;

  // Exibe informacoes no Serial
  Serial.println("\n---- MEDIAS DOS ULTIMOS 20 MINUTOS ----");
  Serial.println("Temperatura media: " + String(mediaTemp, 1) + " C");
  Serial.println("Pressao media: " + String(mediaPress, 1) + " mbar");
  Serial.println("---------------------------------------\n");
}

// ------------------------------------------------------------------------

// animacoes dos LEDs
```

```
void sunnyMode() {// gradiente de amarelo ate laranjado
  fill_gradient_RGB(leds, 0, CRGB(255,165,0), NUM_LEDS/2, CRGB(255,255,0));
  fill_gradient_RGB(leds, (NUM_LEDS/2+1), CRGB(255,255,0), NUM_LEDS-1, CRGB(255
  modoAtivo = "ENSOL";
  delay(200);
  FastLED.show();
}
// animacao que simula trovoes
void thunderMode(){
  int blink_time = 100;
  int wait_time = random(1,3);
  int count_leds = random(2,4);
  int random_index[count_leds];

  fill_solid(leds, NUM_LEDS, CRGB::Black);
  FastLED.show();

  for(int i=0; i<5; i++){

    for(int j=0; j<count_leds; j++){
      random_index[j] = random(j*4+1, j*4+4);
    }

    for(int j=0; j<count_leds; j++){
      leds[random_index[j]] = CRGB::Blue;
    }
    FastLED.show();
    delay(blink_time);
    for(int j=0; j<count_leds; j++){
      leds[random_index[j]] = CRGB::Black;
    }
    FastLED.show();
    delay(blink_time);
    for(int j=0; j<count_leds; j++){
      leds[random_index[j]] = CRGB::Blue;
    }
    FastLED.show();
    delay(blink_time);
    for(int j=0; j<count_leds; j++){
      leds[random_index[j]] = CRGB::Black;
    }
    FastLED.show();
    delay(wait_time*1000);
  }
  delay(1000);
  fill_solid(leds, NUM_LEDS, CRGB::Blue);
  FastLED.show();
  delay(blink_time);
```

```
  fill_solid(leds, NUM_LEDS, CRGB::Black);
  FastLED.show();
  delay(blink_time);
  fill_solid(leds, NUM_LEDS, CRGB::Blue);
  FastLED.show();
  delay(blink_time);
  fill_solid(leds, NUM_LEDS, CRGB::Black);
  FastLED.show();
}
void foggyMode(){
  fill_solid(leds, NUM_LEDS, CRGB::DarkGray);
  FastLED.show();
}
void cloudyMode(){
  fill_solid(leds, NUM_LEDS, CRGB::SkyBlue);
  FastLED.show();
}


// Executa a animacao escolhida
void AnimaLEDs(String clima)
{
  if(clima == "e"){
      sunnyMode();
  }
  else if(clima == "t"){
    thunderMode();
  }
  else if(clima == "n"){
    foggyMode();
  }
  else if(clima == "c"){
    cloudyMode();
  }
  else if(clima == "d"){
    fill_solid(leds, NUM_LEDS, CRGB::Black);
    FastLED.show();
  }
}
// ------------------------------------------------------------------------------

// Funcao dos modos

int bombaLigada = 0;
void ligarBombaDagua(){
  digitalWrite(PIN_CHUVA, HIGH);
  bombaLigada = 1;
}
void desligarBombaDagua(){
```

```
  digitalWrite(PIN_CHUVA, LOW);
  bombaLigada = 0;
}
void executaAnimacaoLocal()
{
  // Verificaoees para animacao
  if(weather_code >= 95 && weather_code <= 99) AnimaLEDs("t");
  else if(rain >= 50) AnimaLEDs("n");
  else if(cloud_cover >= 70) AnimaLEDs("c");
  else AnimaLEDs("e");
  if(rain >= 50.0 && bombaLigada==0) ligarBombaDagua();
  else if(rain < 50.0 && bombaLigada==1) desligarBombaDagua();
}
// Modo LOCAL:
// a cada 2 minutos capta uma medida dos sensores
// ao final de 20 minutos, faz a media das medidas e cruza informacoes com API
// Com base nisso, recria o clima no terrario
void modoLOCAL()
{
  // Verifica se eh a primeira execucao do modo
  if(setupLOCAL == 1){
    setupLOCAL = 0;
    // atualiza servidor
    leDadosAPI();
    mediaTemp = dht.readTemperature(false);
    mediaHum = dht.readHumidity();
    mediaPress = bmp.readPressure()/100;
    executaAnimacaoLocal();
  }
  unsigned long tempoAtual = millis();

  // Verifica se eh hora de fazer uma nova medicao
  if (tempoAtual - ultimaMedida >= INTERVALO || ultimaMedida == 0)
  {
    temperatura[indiceFila] = dht.readTemperature(false); // false -> Celsius
    humidade[indiceFila] = dht.readHumidity();
    pressao[indiceFila] = bmp.readPressure()/100;
    // Atualiza indice da fila e contador de requisicoes
    indiceFila = (indiceFila + 1) % NUM_MEDIDAS;
    totalMedidas++;

    // Atualiza o timestamp da ultima requisicao
    ultimaMedida = tempoAtual;

    // Verifica se eh hora de calcular medias (a cada 10 requisicoes = 20 minu
    if (totalMedidas % NUM_MEDIDAS == 0){
      calcularMedias();
      leDadosAPI();
      executaAnimacaoLocal();
```

```
    }
  }
}

// Se o modo atual nao for Local, ainda eh necessario atualizar o servidor web
// por isso, essa funcao eh chamada pelos outros modos
void atualizaSensores()
{
  unsigned long tempoAtual = millis();

  // Verifica se eh hora de fazer uma nova medicao
  if (tempoAtual - ultimaMedida >= INTERVALO || ultimaMedida == 0)
  {
    ultimaMedida = tempoAtual;
    mediaTemp = dht.readTemperature(false);
    mediaHum = dht.readHumidity();
    mediaPress = bmp.readPressure()/100;
  }
}

// outros modos
void modoENSOL(){
  AnimaLEDs("e");
  atualizaSensores();
}
void modoCHUVA(){
  AnimaLEDs("n");
  if(bombaLigada==0) ligarBombaDagua();// desligar na troca de modos
  atualizaSensores();
}
void modoTEMPS(){
  AnimaLEDs("t");
  if(bombaLigada==0) ligarBombaDagua();// desligar na troca de modos
  atualizaSensores();
}

// no modo NEBLN, apenas ligar nebulizador a cada 10 minutos
const int INTERVALO_NEBUL = 6000000; // 10min x 60s x 1000 = 600000ms
int ultimoAcionamento = 0;

void modoNEBLN(){
  AnimaLEDs("n");
  atualizaSensores();
  unsigned long tempoAtual = millis();

  if (tempoAtual - ultimoAcionamento >= INTERVALO_NEBUL || ultimoAcionamento ==
  {
    ultimoAcionamento = tempoAtual;
    digitalWrite(PIN_NEBUL, HIGH);
```

```
    delay(2000); // mantem 2 segundos ligado
    digitalWrite(PIN_NEBUL, LOW);
  }
}

//-------------------------------------------------------------------------------

// abaixo o HTML da pagina WEB

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/
  <style>
    html {
     font-family: Arial;
     display: inline-block;
     margin: 0px auto;
     text-align: center;
    }
    h2 { font-size: 3.0rem; }
    p { font-size: 3.0rem; }
    .units { font-size: 1.2rem; }
    .dht-labels{
      font-size: 1.5rem;
      vertical-align:middle;
      padding-bottom: 15px;
    }
  </style>
</head>
<body>
  <h2>Projeto Climario</h2>
  <p>
    <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
    <span class="dht-labels">Sensacao</span>
    <span id="temperature">%TEMPERATURE%</span>
    <sup class="units">&deg;C</sup>
  </p>
  <p>
    <i class="fas fa-tint" style="color:#00add6;"></i>
    <span class="dht-labels">Humidade</span>
    <span id="humidity">%HUMIDITY%</span>
    <sup class="units">&percnt;</sup>
  </p>
  <p>
    <span class="dht-labels">Pressao</span>
    <span id="pressure">%PRESSURE%</span>
    <sup class="units">mbar</sup>
```

```
  </p>
  <p>
    <span class="dht-labels">Chuva</span>
    <span id="rain">%RAIN%</span>
    <sup class="units">mm</sup>
  </p>
  <p>
    <span class="dht-labels">Nuvens</span>
    <span id="cloud">%CLOUD%</span>
    <sup class="units">&percnt;</sup>
  </p>
  <p>
    <span class="dht-labels">Modo Atual</span>
    <span id="mode">%MODE%</span>
    <sup class="units"></sup>
  </p>
  <h3>Troca de Modo (Delay de 2 minutos)</h3>
  <form id="trocaModo">
      <select name="modos" id="modos">
        <option value="ENSOL">Ensolarado</option>
        <option value="CHUVA">Chuvoso</option>
        <option value="TEMPS">Tempestuoso</option>
        <option value="LOCAL">Local</option>
      </select>
      <br>
      <br>
      <button type="submit" >Enviar</button>
</form>
<p id="response"></p>
</body>
<script>
  document.getElementById("trocaModo").addEventListener("submit", function(ever
     event.preventDefault();

     const selectedOption = document.getElementById("modos").value;
     const serverIp = "http://192.168.100.126"; // Altere para o IP do seu ESP3

     fetch(serverIp, {
        method: "POST",
        headers: {
          "Content-Type": "application/json"
        },
          body: JSON.stringify({ opcao: selectedOption })
      })
      .then(response => {
         if (response.ok) {
            document.getElementById("response").innerText = "Comando enviado co
       } else {
            document.getElementById("response").innerText = "Erro ao enviar com
```

```
      }
    })
    .catch(error => {
       console.error("Erro ao enviar requisicao:", error);
       document.getElementById("response").innerText = "Erro ao conectar ao s
  });
});
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
}, 10000 ) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("humidity").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/humidity", true);
  xhttp.send();
}, 10000 ) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("pressure").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/pressure", true);
  xhttp.send();
}, 10000 ) ;

setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("mode").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "/mode", true);
  xhttp.send();
```

```
  }, 10000 ) ;

  setInterval(function ( ) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("rain").innerHTML = this.responseText;
      }
    };
    xhttp.open("GET", "/rain", true);
    xhttp.send();
  }, 10000 ) ;

  setInterval(function ( ) {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("cloud").innerHTML = this.responseText;
      }
    };
    xhttp.open("GET", "/cloud", true);
    xhttp.send();
  }, 10000 ) ;
</script>
</html>)rawliteral";

// Substitui os valores nos placeholder correspondentes no HTML
String processor(const String& var){
  if(var == "TEMPERATURE"){
    return String(mediaTemp);
  }
  else if(var == "HUMIDITY"){
    return String(mediaHum);
  }
  else if(var == "PRESSURE"){
    return String(mediaPress);
  }
  else if(var == "MODE"){
    return modoAtivo;
  }
  else if(var == "RAIN"){
    return String(rain);
  }
  else if(var == "CLOUD"){
    return String(cloud_cover);
  }
  return String();
}
```

```
// ------------------------------------------------------------------------
void setup() {
  delay(3000); // delay de seguranca (FastLED)
  FastLED.addLeds<LED_TYPE, LED_PIN, COLOR_ORDER>(leds, NUM_LEDS);
  FastLED.setBrightness(BRIGHTNESS);
  // Inicia serial
  Serial.begin(115200);


  // Conexao WiFi
  WiFi.mode(WIFI_STA);
  Serial.print("Conectando no WiFi..");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print('.');
  }
  Serial.println("\nConectado!\n");
  Serial.print("Iniciado STA:\t");
  Serial.println(ssid);
  Serial.print("IP address:\t");
  Serial.println(WiFi.localIP());
  // NTP
  Serial.println("Contactando servidor NTP..");
  configTime(3600 * timezone, daysavetime * 3600, "time.nist.gov", "0.pool.ntp


  // Inicializando Sensores
  dht.begin();
  if (!bmp.begin(0x76)) {
    Serial.println("Could not find a valid BMP085/BMP180 sensor, check wiring!"
    while (1) {}
  }
  delay(1100); // delay de seguranca (dht11)
  pinMode(PIN_CHUVA, OUTPUT);
  pinMode(PIN_NEBUL, OUTPUT);
  // Dados iniciais para serem mostrados no servidor web
  leDadosAPI();
  mediaTemp = dht.readTemperature(false);
  mediaHum = dht.readHumidity();
  mediaPress = bmp.readPressure()/100; // Pa -> mbar
  modoAtivo = "ENSOL";
  ultimaTrocaModo = millis(); // trocar modo apenas depois de 5 minutos
  ultimaMedida = millis(); // nova leitura de sensores depois de 2 minutos

  // Iniciando Servidor
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });
```

```
  server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(mediaTemp).c_str());
  });
  server.on("/humidity", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(mediaHum).c_str());
  });
  server.on("/pressure", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(mediaPress).c_str());
  });
  server.on("/mode", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", modoAtivo.c_str());
  });
  server.on("/rain", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(rain).c_str());
  });
  server.on("/cloud", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/plain", String(cloud_cover).c_str());
  });
  server.on("/",HTTP_POST,[](AsyncWebServerRequest * request){},NULL,
    [](AsyncWebServerRequest * request, uint8_t *data, size_t len, size_t index
      String receivedData = "";
      for (size_t i = 0; i < len; i++) {
        receivedData += (char)data[i]; // Concatena os dados recebidos na strin
      }
      // Parse JSON
      StaticJsonDocument<200> doc;
      DeserializationError error = deserializeJson(doc, receivedData);

      if (!error) {
        Mode = doc["opcao"].as<String>();
        Serial.print("Opcao recebida: ");
        Serial.println(Mode);
        houveTroca = 1;
        if(Mode != modoAtivo) setupLOCAL = 1;
      } else {
        Serial.println("Erro ao parsear JSON");
      }
      request->send(200);
  });
  server.onNotFound([](AsyncWebServerRequest *request) {
    request->send(400, "text/plain", "Not found");
  });
  server.begin();
  Serial.println("Server HTTP iniciado");
}


void loop() {
  // Obtem data e hora atuais
```

```
  // atualizaDataHora(); // erro
  // Verificar se ha troca de MODO. Se sim, desligar tudo (bomba, nebul, leds)
  // obter atraves de entrada do usuario pelo servidor web
  if(houveTroca){
    unsigned int tempoAtual = millis();
    if(tempoAtual - ultimaTrocaModo >= ESPERA_TROCA){
      modoAtivo = String(Mode);
      ultimaTrocaModo = tempoAtual;
      houveTroca = 0;
    } else Serial.println("ainda em espera");
  }

  // codigo para apresentacao sem WiFi
  // String input_modo = modoAtivo;
  // Serial.println("Modo?");
  // if(Serial.available()>0){
  //   input_modo = Serial.readString(); // exceto modo LOCAL: ENSOL, CHUVA, TE
  //   modoAtivo = input_modo;
  // }

  Serial.println("Modo Ativo: " + modoAtivo);

  if(modoAtivo == "LOCAL") modoLOCAL();
  else if(modoAtivo == "ENSOL") modoENSOL();
  else if(modoAtivo == "CHUVA") modoCHUVA();
  else if(modoAtivo == "TEMPS") modoTEMPS();
  //else if(modoAtivo == "NEBLN") modoNEBLN();

  FastLED.delay(1000 / UPDATES_PER_SECOND);
  delay(1100);
}
```