Universidade Federal de Uberlândia
Faculdade de Engenharia Elétrica - Campus Patos de Minas
Engenharia Eletrônica e de Telecomunicações

SISTEMA DE MONITORAMENTO DE TEMPERATURA SEM FIO USANDO HELTEC ESP32 LORA E MQTT

Diogo Campos de Arvelos

Patos de Minas
2023

SISTEMA DE MONITORAMENTO DE TEMPERATURA SEM FIO USANDO HELTEC ESP32 LORA E MQTT

Diogo Campos de Arvelos

Relatório final apresentado como um dos requisitos de avaliação na disciplina Projeto Interdisciplinar Aplicada do Curso de Engenharia de Eletrônica e Telecomunicações da Universidade Federal de Uberlândia.

Orientador: Prof. Dr. Júlio Cézar Coelho

Patos de Minas
2023

SUMÁRIO

1 INTRODUÇÃO

O presente relatório documenta o desenvolvimento e a implementação de um projeto integrado, focado na coleta, transmissão e visualização de dados ambientais em tempo real. O projeto emprega dispositivos Heltec ESP32 LoRa, sensores de temperatura e umidade do ar e do solo, além da utilização de tecnologias como MQTT e PostgreSQL hospedado na AWS. A principal motivação por trás deste projeto é a necessidade crescente de monitoramento ambiental eficiente e acessível.

Ao longo deste relatório, exploraremos as fases fundamentais do projeto, desde a configuração do hardware e a comunicação entre dispositivos até a análise dos dados coletados. O sistema proposto demonstra uma abordagem inovadora ao integrar tecnologias sem fio de baixo consumo, comunicação em nuvem e visualização interativa, oferecendo uma solução completa para a monitorização ambiental.

A coleta precisa de dados ambientais, a transmissão eficiente entre dispositivos e a posterior análise e visualização são elementos cruciais neste projeto. A utilização de um broker MQTT público e a integração com um banco de dados PostgreSQL na AWS contribuem para a escalabilidade e acessibilidade da solução proposta.

Este relatório visa fornecer uma visão abrangente do desenvolvimento do projeto, destacando tanto os aspectos técnicos quanto os desafios enfrentados ao longo do caminho. Ao final, serão apresentadas conclusões sobre a eficácia do sistema implementado e sugestões para melhorias futuras.

2 METODOLOGIA

A implementação bem-sucedida deste projeto envolveu uma abordagem estruturada e cuidadosa, que abrangeu desde a seleção dos dispositivos e sensores até a configuração de uma infraestrutura robusta na nuvem. A metodologia adotada pode ser dividida nas seguintes etapas:

2.1 Seleção e Configuração do Hardware:

Escolha do Heltec ESP32 LoRa como plataforma de hardware devido à sua capacidade de comunicação sem fio eficiente e baixo consumo de energia.

Integração dos sensores de temperatura e umidade do ar e do solo, garantindo a compatibilidade e precisão das leituras.

Configuração detalhada dos dispositivos, incluindo parâmetros de comunicação e calibração dos sensores.

2.2 Desenvolvimento do Código para os Dispositivos ESP32:

Programação dos ESP32 LoRa para realizar leituras dos sensores e transmitir os dados entre si.

Implementação de lógica para o envio seguro e eficiente dos dados para o broker MQTT público (Hivemq).

Testes extensivos para garantir a estabilidade e confiabilidade da comunicação.

2.3 Configuração do Broker MQTT e Comunicação na Nuvem:

Estabelecimento de uma conexão segura com o broker MQTT Hivemq.

Desenvolvimento de rotinas para recebimento e encaminhamento dos dados para a nuvem.

Garantia da integridade e confidencialidade dos dados durante a transmissão.

2.4 Integração com PostgreSQL na AWS:

Configuração de um banco de dados PostgreSQL na AWS para armazenar os dados coletados.

Desenvolvimento de scripts em Python para inserção segura e eficiente dos dados no banco de dados.

Implementação de medidas de segurança e monitoramento para proteger a integridade dos dados.

2.5 Desenvolvimento do Site e Visualização de Dados:

Criação de um site em Python para interação com o banco de dados e geração de gráficos interativos.

Utilização de bibliotecas como Matplotlib e Flask para a visualização eficaz dos dados.

Implementação de uma interface amigável e acessível para os usuários finais.

2.6 Testes e Validação:

Realização de testes abrangentes em todas as fases do projeto, desde a coleta de dados até a visualização.

Validação da consistência e precisão das leituras dos sensores.

Verificação da robustez e confiabilidade da transmissão e armazenamento dos dados.

2.7 Documentação e Relatório Final:

Elaboração de documentação detalhada do projeto, incluindo manuais de operação e manutenção.

Compilação do relatório final, abordando cada fase do projeto, desafios enfrentados e soluções implementadas.

Esta metodologia permitiu uma abordagem sistemática e eficiente, garantindo o sucesso na implementação do projeto e proporcionando uma base sólida para futuras melhorias e expansões.

3 DESENVOLVIMENTO E RESULTADOS

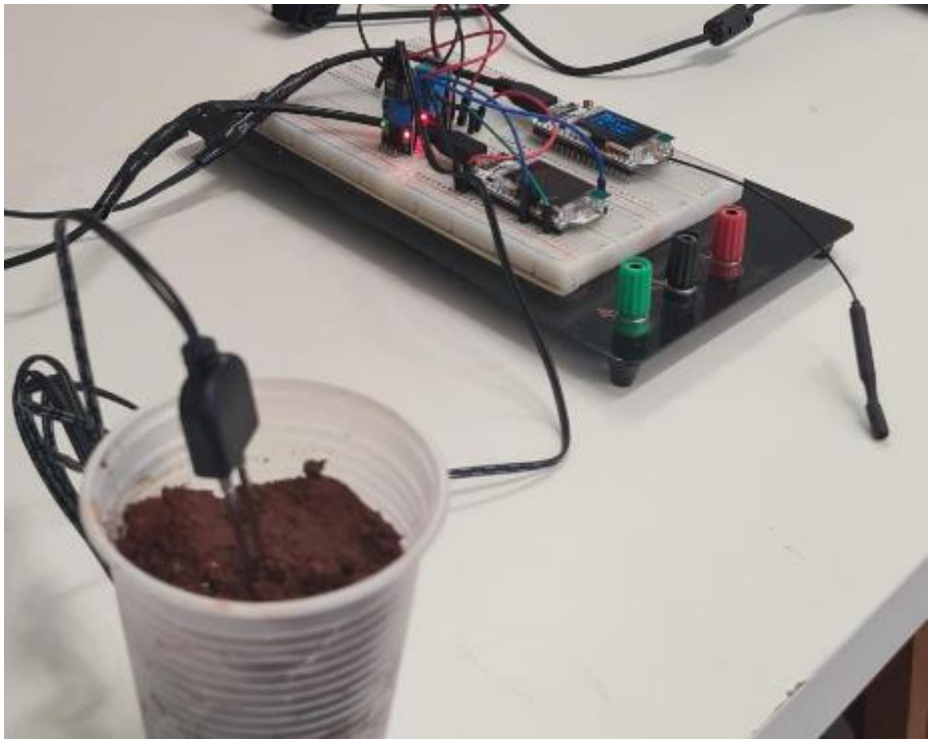
O desenvolvimento do projeto foi dividido em várias etapas, cada uma contribuindo para a criação de um sistema integrado de coleta, transmissão e visualização de dados ambientais em tempo real. A seguir, descrevemos cada fase, fornecendo espaços reservados para inclusão de imagens e trechos de código pertinentes.

3.1 Configuração do Hardware:

O primeiro passo envolveu a seleção e configuração do hardware. O dispositivo principal utilizado foi o Heltec ESP32 LoRa, escolhido devido à sua eficiência na comunicação sem fio de longo alcance e baixo consumo de energia.

Além disso, foram integrados sensores de temperatura e umidade do ar e do solo para coleta precisa de dados ambientais.

Na imagem a baixo é possível ver esses aparelhos conectados.



3.2 Desenvolvimento do Código para os Dispositivos ESP32:

Código do ESP32 LoRa Transmissor:

No transmissor, o código foi organizado em seções distintas, para destacar as partes mais importantes do código.

Antes de adentrar nos detalhes do envio via LoRa, é crucial estabelecer as bibliotecas essenciais para o funcionamento adequado do transmissor.

transmissor.final3 §

```
#include <WiFi.h>
#include <DHT.h>
#include <Wire.h>
#include <LoRa.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <PubSubClient.h>/* * Defines do projeto *//* GPIO do módulo WiFi LoRa 32(V2) que o pino de
#define DHTPIN    17 /* (GPIO 13) *//* Endereço I2C do display */
#define OLED_ADDR 0x3c/* distancia, em pixels, de cada linha em relacao ao topo do display */
#define OLED_LINE1 0
#define OLED_LINE2 10
#define OLED_LINE3 20
#define OLED_LINE4 30
#define OLED_LINE5 40
#define OLED_LINE6 50/* Configuração da resolucao do display (este modulo possui display 128x64) */
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64/*A biblioteca serve para os sensores DHT11, DHT22 e DHT21. No nosso caso, u
#define DHTTYPE DHT11   // DHT 11/* id mqtt (para identificação de sessão)
#define SCK_LORA           5
#define MISO_LORA          19
#define MOSI_LORA          27
#define RESET_PIN_LORA     14
#define SS_PIN_LORA        18
#define HIGH_GAIN_LORA     20  /* dBm */
#define BAND               915E6  /* 915MHz de frequencia *//* Constantes */
```

A configuração dos pinos é uma etapa vital, determinando a interface entre o ESP32 e os sensores. Essa seção do código se dedica a essa configuração específica.

```
DHT dht(DHTPIN, DHTTYPE);/* objeto do display */
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, 16);/* variáveis
float temperatura_max;
float temperatura_min;/* typedefs */
typedef struct __attribute__((__packed__))
{
  float temperatura;
  float umidade;
  float temperatura_min;
  float temperatura_max;
  float solo;
}TDadosLora;/* prototypes */
void atualiza_temperatura_max_e_minima(float temp_lida);
void escreve_temperatura_umidade_display(float temp_lida, float umid_lida);
void envia_medicoes_para_serial(float temp_lida, float umid_lida);
void envia_informacoes_lora(float temp_lida, float umid_lida);
bool init_comunicacao_lora(void);
int pino_d = 5; /* Pino ligado ao D0 do sensor */
int pino_a = 13; /* Pino ligado ao A0 do sensor */
int val_d = 0;   /* Armazena o valor lido do pino digital */
int solo_lida1 = 0;   /* Armazena o valor lido do pino analógico */
int pin = 2;    /* Vermelho Pino D2 do ESP32 */
int pin2 = 4;   /* Azul Pino D4 do ESP32 */
```

A parte dedicada ao envio via LoRa abrange a lógica necessária para transmitir eficientemente os dados coletados pelos sensores para o receptor. Essa seção incluirá o manuseio de pacotes e a garantia da integridade da transmissão.

```
void envia_informacoes_lora(float temp_lida, float umid_lida, float sol_lida)
{
    TDadosLora dados_lora;
    dados_lora.temperatura = temp_lida;
    dados_lora.umidade = umid_lida;
    dados_lora.temperatura_min = temperatura_min;
    dados_lora.temperatura_max = temperatura_max;
    dados_lora.solo = sol_lida;
    LoRa.beginPacket();
    LoRa.write((unsigned char *)&dados_lora, sizeof(TDadosLora));
    LoRa.endPacket();
}/* Funcao: inicia comunicação com chip LoRa * Parametros: nenhum * Retorno: true: comunicacao ok *          fa
bool init_comunicacao_lora(void)
{
    bool status_init = false;
    Serial.println("[LoRa Sender] Tentando iniciar comunicacao com o radio LoRa...");
    SPI.begin(SCK_LORA, MISO_LORA, MOSI_LORA, SS_PIN_LORA);
    LoRa.setPins(SS_PIN_LORA, RESET_PIN_LORA, LORA_DEFAULT_DIO0_PIN);
    if (!LoRa.begin(BAND))
    {
        Serial.println("[LoRa Sender] Comunicacao com o radio LoRa falhou. Nova tentativa em 1 segundo...");
        delay(1000);
        status_init = false;
    }
    else
    {
        /* Configura o ganho do receptor LoRa para 20dBm, o maior ganho possivel (visando maior alcance possive
        LoRa.setTxPower(HIGH_GAIN_LORA);
        Serial.println("[LoRa Sender] Comunicacao com o radio LoRa ok");
        status_init = true;
    }
    return status_init;
}
```

Código do ESP32 LoRa Receptor e MQTT:

O código do receptor foi segmentado em seções específicas para cada funcionalidade.

Inicialmente, as bibliotecas necessárias para a operação do receptor, incluindo as relacionadas à recepção LoRa e à comunicação via Wi-Fi.

receptor.final3 §

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <LoRa.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>/* Definicoes para comunicação com radio LoRa */
#define SCK_LORA           5
#define MISO_LORA          19
#define MOSI_LORA          27
#define RESET_PIN_LORA     14
#define SS_PIN_LORA        18
#define HIGH_GAIN_LORA     20  /* dBm */
#define BAND               915E6  /* 915MHz de frequencia *//* Definicoes do OLED */
#define OLED_SDA_PIN    4
#define OLED_SCL_PIN    15
#define SCREEN_WIDTH    128
#define SCREEN_HEIGHT   64
#define OLED_ADDR       0x3C
#define OLED_RESET      16/* Offset de linhas no display OLED */
#define OLED_LINE1     0
#define OLED_LINE2     10
#define OLED_LINE3     20
#define OLED_LINE4     30
#define OLED_LINE5     40
#define OLED_LINE6     50/* Definicoes gerais */
#define DEBUG_SERIAL_BAUDRATE    115200/* Variaveis e objetos globais */
#define TOPICO_PUBLISH   "MQTTINCBTempUmidDiogo"
/*tópico MQTT de envio de informações para Broker
#define ID_MQTT  "2810440f-64e2-4ce4-a527-b949386f8679"
```

Nesta seção, detalharemos o processo de recebimento via LoRa. Isso abrange a captura dos dados transmitidos pelo transmissor e sua preparação para o próximo estágio.

```
bool init_comunicacao_lora(void)
{
    bool status_init = false;
    Serial.println("[LoRa Receiver] Tentando iniciar comunicacao com o radio LoRa...");
    SPI.begin(SCK_LORA, MISO_LORA, MOSI_LORA, SS_PIN_LORA);
    LoRa.setPins(SS_PIN_LORA, RESET_PIN_LORA, LORA_DEFAULT_DIO0_PIN);

    if (!LoRa.begin(BAND))
    {
        Serial.println("[LoRa Receiver] Comunicacao com o radio LoRa falhou. Nova tentativa em 1 segundo...");
        delay(1000);
        status_init = false;
    }
    else
    {
        /* Configura o ganho do receptor LoRa para 20dBm, o maior ganho possível (visando maior alcance possível) */
        LoRa.setTxPower(HIGH_GAIN_LORA);
        Serial.println("[LoRa Receiver] Comunicacao com o radio LoRa ok");
        status_init = true;
    }
    return status_init;
}/* Funcao de setup */
void setup()
{
```

A etapa de conexão com o Wi-Fi e broker MQTT com o é crucial para a transição dos dados do ambiente local para a nuvem. Aqui, será abordada a configuração e estabelecimento da conexão Wi-Fi e do broker MQTT.

```
void init_wifi(void)
{
    delay(10);
    Serial.println("------Conexao WI-FI------");
    Serial.print("Conectando-se na rede: ");
    Serial.println(SSID);
    Serial.println("Aguarde");
    reconnect_wifi();
}
 /* Função: inicializa parâmetros de conexão MQTT(endereço do broker e porta) * Parâmetros: nenhum * Retorno: ne
void init_MQTT(void)
{
    //informa qual broker e porta deve ser conectado
    MQTT.setServer(BROKER_MQTT, BROKER_PORT);
}
/* Função: reconecta-se ao broker MQTT (caso ainda não esteja conectado ou em caso de a conexão cair) *
void reconnect_MQTT(void)
{
    while (!MQTT.connected())
    {
        Serial.print("* Tentando se conectar ao Broker MQTT: ");
        Serial.println(BROKER_MQTT);
        if (MQTT.connect(ID_MQTT))
            Serial.println("Conectado com sucesso ao broker MQTT!");
        else
        {
            Serial.println("Falha ao reconectar no broker.");
            Serial.println("Havera nova tentatica de conexao em 2s");
            delay(2000);
        }
    }
}
```

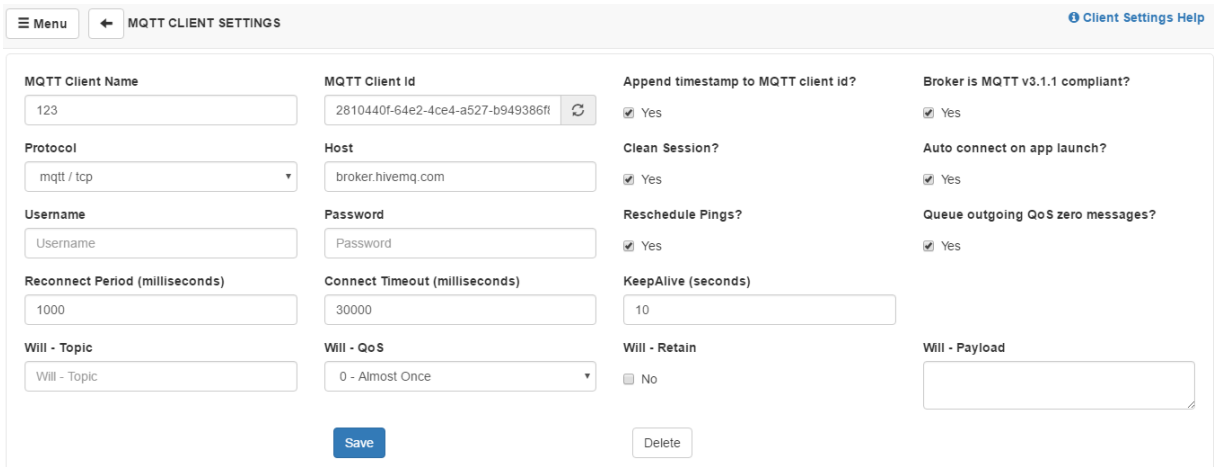
Por fim, o código inclui a lógica para enviar os dados recebidos para o broker MQTT, conectando-se à plataforma Hivemq. Essa etapa é vital para disponibilizar os dados para visualização remota e análise.

```
receptor.final3 §
}/* Programa principal */
void loop()
{
    char byte_recebido;
    int packet_size = 0;
    int lora_rssi = 0;
    TDadosLora dados_lora;
    char * pt_dados_lora = NULL;
    char str_temp[10] = {0};
    char str_umid[20] = {0};
    char str_temp_max_minima[20] = {0};
    float temperatura;
    float umidade;
    /* Verifica se chegou alguma informação do tamanho esperado */
    packet_size = LoRa.parsePacket();
    verifica_conexoes_wifi_e_MQTT();
    if (packet_size == sizeof(TDadosLora))
    {
        /* Recebe os dados conforme protocolo */
        pt_dados_lora = (char *)&dados_lora;
        while (LoRa.available())
        {
            byte_recebido = (char)LoRa.read();
            *pt_dados_lora = byte_recebido;
            pt_dados_lora++;
        }
        /* Escreve RSSI de recepção e informações recebidas */
        lora_rssi = LoRa.packetRssi();
        char mensagem_MQTT[200] = {0};
        sprintf(mensagem_MQTT,"Temperatura: %.2f C, Umidade ar: %.2f/100, Umidade solo: %.2f/100", dados_lora.temperatura, dados_lora.umidade, dados_lora.solo, dados_lora.temperatura_max, dados_lora.temperatura_min);

MQTT.publish(TOPICO_PUBLISH, mensagem_MQTT);
    }
    MQTT.loop();
     delay(100);
}
```

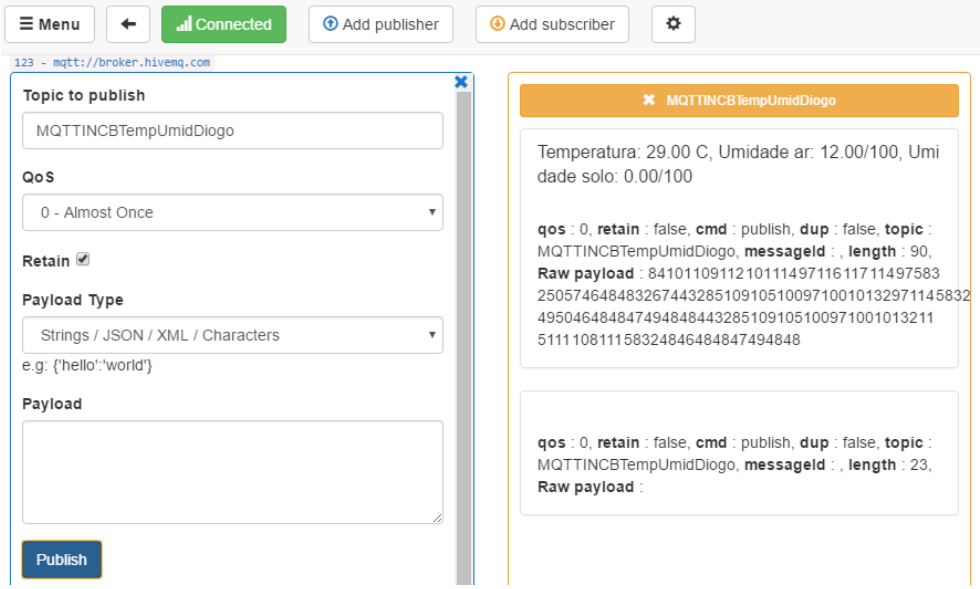
3.3 Configuração do Servidor MQTT HiveMQ usando MQTTBox:

A comunicação na nuvem foi estabelecida por meio do servidor MQTT HiveMQ, configurado e gerenciado eficientemente utilizando a ferramenta MQTTBox.

O processo envolveu a criação de tópicos específicos, definição de políticas de segurança e configuração de parâmetros essenciais para garantir uma comunicação segura e confiável entre os dispositivos ESP32 e o servidor MQTT.

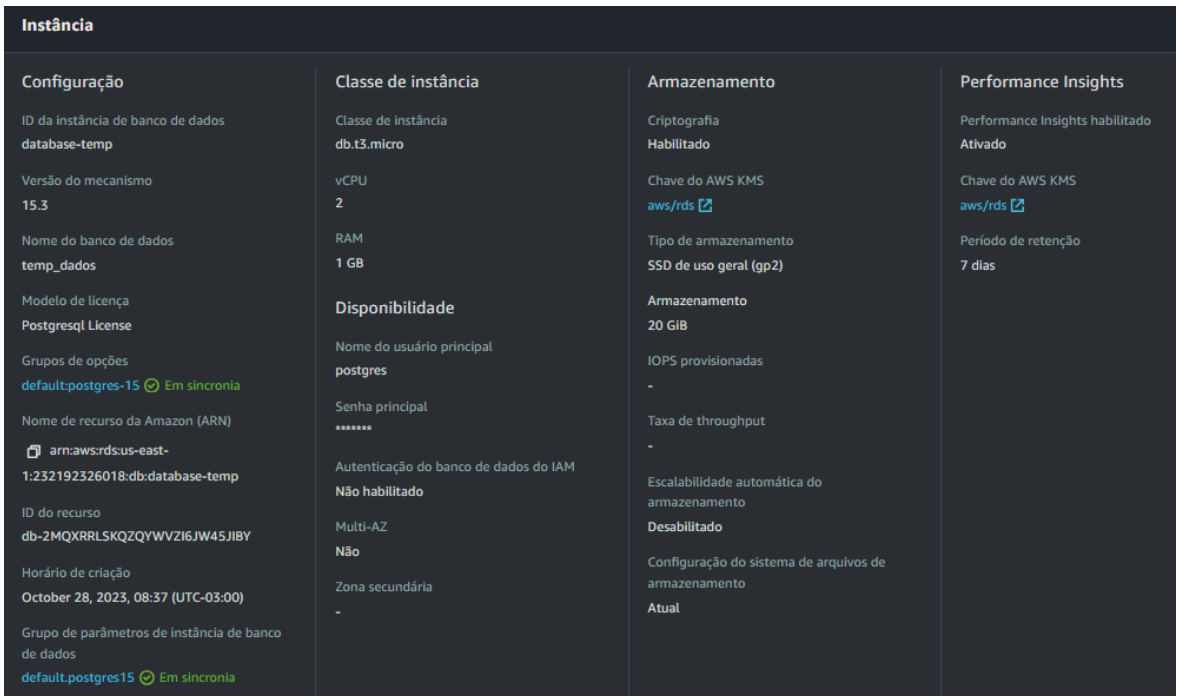


A utilização do MQTTBox proporcionou uma interface intuitiva para monitorar e gerenciar as comunicações MQTT, facilitando a integração dos dispositivos e garantindo a eficiência do fluxo de dados.



3.4 Integração com PostgreSQL na AWS:

Configurou-se um banco de dados PostgreSQL na AWS para armazenar os dados coletados.



3.5 Desenvolvimento do Site e Visualização de Dados:

A implementação do site e visualização de dados foi realizada utilizando o framework Flask em conjunto com o Paho MQTT para comunicação com o broker e psycopg2 para interação com o banco de dados PostgreSQL na AWS. O código abaixo é dividido em seções para melhor compreensão:

Configurações Iniciais:

As bibliotecas Flask, Paho MQTT, psycopg2 e datetime são importadas.

São definidas as configurações para o broker MQTT, incluindo host, porta e tópico, e as configurações para o banco de dados PostgreSQL, como host, porta, usuário e senha.

```python
from flask import Flask, render_template
import paho.mqtt.client as mqtt
import psycopg2
from datetime import datetime

app = Flask(__name__)

# Configurações MQTT
MQTT_BROKER_HOST = "broker.hivemq.com"
MQTT_BROKER_PORT = 1883
MQTT_TOPIC = "MQTTINCBTempUmidDiogo"

# Configuração do banco de dados PostgreSQL
POSTGRES_HOST = "database-temp.ct8moxkr9qvc.us-east-1.rds.amazonaws.com"
POSTGRES_PORT = 5432
POSTGRES_USER = "postgres"
POSTGRES_PASSWORD = "123456789"
```

Funções para interagir com o Banco de Dados:

Três funções são definidas para interagir com o banco de dados PostgreSQL. A função “create_table” cria a tabela se ela ainda não existir. “insert_message” insere uma mensagem no banco de dados e “get_messages” recupera a última mensagem registrada.

```python
def create_table():
    conn = psycopg2.connect(
        host=POSTGRES_HOST,
        port=POSTGRES_PORT,
        user=POSTGRES_USER,
        password=POSTGRES_PASSWORD,

    )
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS messages (
            id SERIAL PRIMARY KEY,
            topic TEXT,
            payload TEXT,
            timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ''')
    conn.commit()
    conn.close()
```

```python
def insert_message(topic, payload):
    conn = psycopg2.connect(
        host=POSTGRES_HOST,
        port=POSTGRES_PORT,
        user=POSTGRES_USER,
        password=POSTGRES_PASSWORD,

    )
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO messages (topic, payload) VALUES (%s, %s)
    ''', (topic, payload))
    conn.commit()
    conn.close()
```

```python
def get_messages():
    conn = psycopg2.connect(
        host=POSTGRES_HOST,
        port=POSTGRES_PORT,
        user=POSTGRES_USER,
        password=POSTGRES_PASSWORD,

    )
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM messages ORDER BY timestamp DESC LIMIT 1')
    result = cursor.fetchone()
    conn.close()
    return result
```

Configuração e Conexão com o Cliente MQTT:

É configurado um cliente MQTT usando a biblioteca Paho. As funções “on_connect” e “on_message” são definidas para tratar eventos de conexão e recebimento de mensagens.

```python
def on_connect(client, userdata, flags, rc):
    client.subscribe(MQTT_TOPIC)

def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    # Inserir a mensagem no banco de dados
    insert_message(MQTT_TOPIC, payload)

# Configurar o cliente MQTT
mqtt_client = mqtt.Client()
mqtt_client.on_connect = on_connect
mqtt_client.on_message = on_message
mqtt_client.connect(MQTT_BROKER_HOST, MQTT_BROKER_PORT, 60)
```

Criação da Tabela no Banco de Dados:

A tabela no banco de dados é criada chamando a função “create_table()”.

```python
def create_table():
    conn = psycopg2.connect(
        host=POSTGRES_HOST,
        port=POSTGRES_PORT,
        user=POSTGRES_USER,
        password=POSTGRES_PASSWORD,

    )
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS messages (
            id SERIAL PRIMARY KEY,
            topic TEXT,
            payload TEXT,
            timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ''')
    conn.commit()
    conn.close()
```

A interface gráfica foi implementada utilizando JavaScript e a biblioteca Chart.js para criar gráficos interativos. Abaixo, o código é dividido em seções:

Inicialização e Atualização do Gráfico:

A variável “selectedChart” é inicializada como 'minute', indicando que o gráfico por minuto será exibido inicialmente.

A função “updateChart” é definida para atualizar o gráfico com base na opção selecionada pelo usuário no seletor de gráficos.

```
var selectedChart = 'minute'; // Inicialmente, exiba o gráfico por minuto

function updateChart() {
    selectedChart = document.getElementById('chartSelector').value;

    // Oculta todos os gráficos
    document.getElementById('myChartMinute').style.display = 'none';
    document.getElementById('myChartHour').style.display = 'none';
    document.getElementById('myChartDay').style.display = 'none';

    // Exibe o gráfico selecionado
    document.getElementById('myChart' + selectedChart.charAt(0).toUpperCase() + selectedChart.slice(1)).style.display = 'block';
}
```

Coleta e Processamento das Informações do Banco de Dados:

Nesse trecho do código é criado uma função para saber quando cada dado foi coletado, e uma outra função para encontrar as informações de temperatura e de humidade do solo e do ar do banco de dados.

```
var groupedData = {};
{% for value in values_last_31_days %}
    var timestamp = new Date('{{ value[3] }}');
    timestamp.setHours(timestamp.getHours() - 3);

    var minuteKey = timestamp.toLocaleString('pt-br', { month: 'numeric', day: 'numeric', hour: 'numeric', minute: 'numeric' });

    if (!groupedData[minuteKey]) {
        groupedData[minuteKey] = { count: 0, value1: [], value2: [], value3: [] };
    }

    groupedData[minuteKey].count++;
    groupedData[minuteKey].value1.push(parseFloat('{{ value[2][13:-49] }}'));
    groupedData[minuteKey].value2.push(parseFloat('{{ value[2][34:-28] }}'));
    groupedData[minuteKey].value3.push(parseFloat('{{ value[2][59:-4] }}'));
{% endfor %}
```

Preparação de Dados para o Gráfico por Minuto:

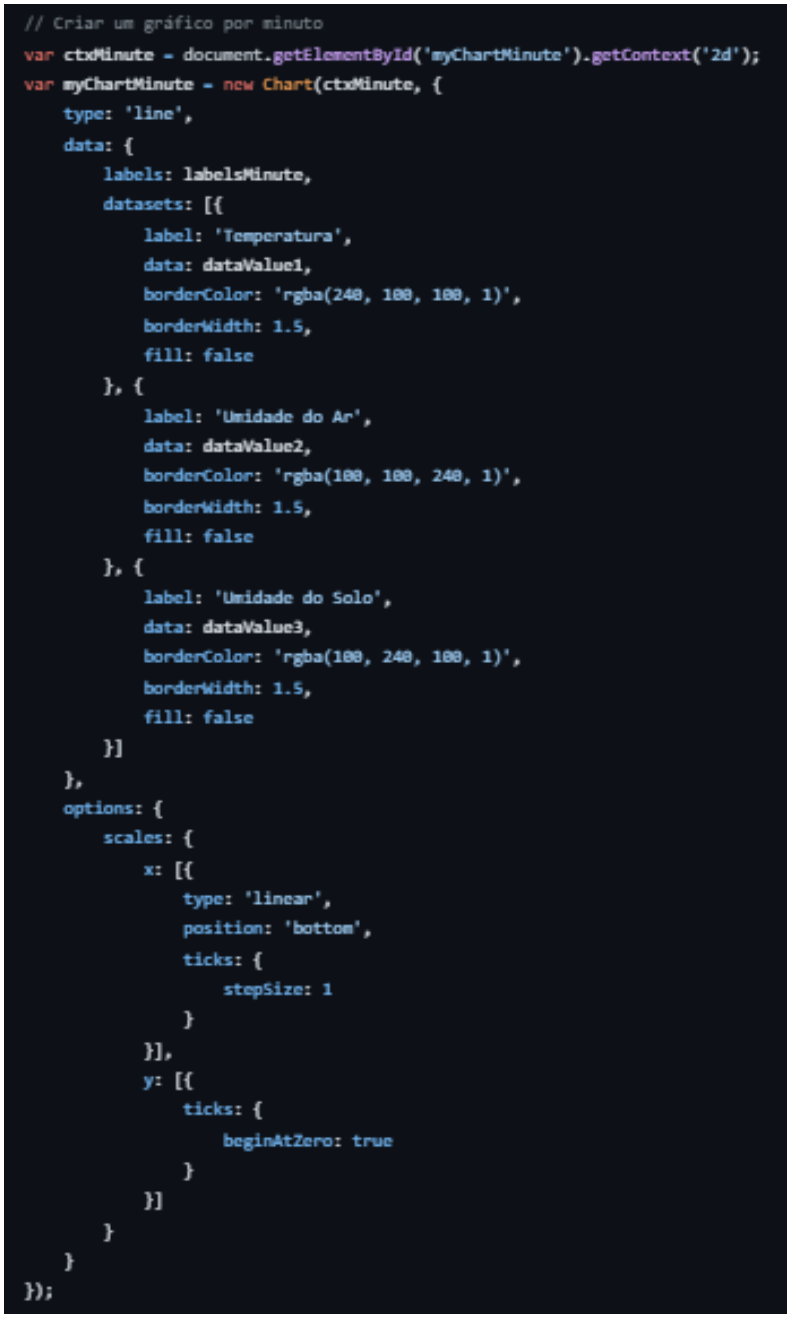
Os dados agrupados por minuto são processados para criar três conjuntos de dados: temperatura, umidade do ar e umidade do solo. Após isso ainda é realizada a média de cada um desses valores recebidos durante cada minuto.

```
// Preparar dados para o gráfico por minuto
var labelsMinute0 = Object.keys(groupedData);
var labelsMinute= labelsMinute0.slice(-60);

var dataValue1 = labelsMinute.map(function (key) {
    var values = groupedData[key].value1.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});
```

```
var dataValue2 = labelsMinute.map(function (key) {
    var values = groupedData[key].value2.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});

var dataValue3 = labelsMinute.map(function (key) {
    var values = groupedData[key].value3.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});
```

Criação do Gráfico por Minuto:

O gráfico por minuto é criado utilizando a biblioteca Chart.js. Cada conjunto de dados é representado por uma linha no gráfico.



```
// Criar um gráfico por minuto
var ctxMinute = document.getElementById('myChartMinute').getContext('2d');
var myChartMinute = new Chart(ctxMinute, {
    type: 'line',
    data: {
        labels: labelsMinute,
        datasets: [{
            label: 'Temperatura',
            data: dataValue1,
            borderColor: 'rgba(240, 100, 100, 1)',
            borderWidth: 1.5,
            fill: false
        }, {
            label: 'Umidade do Ar',
            data: dataValue2,
            borderColor: 'rgba(100, 100, 240, 1)',
            borderWidth: 1.5,
            fill: false
        }, {
            label: 'Umidade do Solo',
            data: dataValue3,
            borderColor: 'rgba(100, 240, 100, 1)',
            borderWidth: 1.5,
            fill: false
        }]
    },
    options: {
        scales: {
            x: [{
                type: 'linear',
                position: 'bottom',
                ticks: {
                    stepSize: 1
                }
            }],
            y: [{
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
```

Preparação de Dados para o Gráfico por Hora:

Os dados agrupados por hora são processados para criar três conjuntos de dados: temperatura, umidade do ar e umidade do solo. Após isso ainda é realizada a média de cada um desses valores recebidos durante cada hora.

```
  // Preparar dados para o gráfico por hora
var groupedDataHour = {};
for (var key in groupedData) {
    if (groupedData.hasOwnProperty(key)) {
        var hourKey = key.split(':')[0] + ':00';
        if (!groupedDataHour[hourKey]) {
            groupedDataHour[hourKey] = { count: 0, value1: [], value2: [], value3: [] };
        }
        groupedDataHour[hourKey].count += groupedData[key].count;
        groupedDataHour[hourKey].value1 = groupedDataHour[hourKey].value1.concat(groupedData[key].value1);
        groupedDataHour[hourKey].value2 = groupedDataHour[hourKey].value2.concat(groupedData[key].value2);
        groupedDataHour[hourKey].value3 = groupedDataHour[hourKey].value3.concat(groupedData[key].value3);
    }
}

var labelsHour = Object.keys(groupedDataHour);

// Ajuste para exibir apenas as últimas 60 médias
if (labelsHour.length > 48) {
    labelsHour = labelsHour.slice(labelsHour.length - 48);
}

var dataValue1Hour = labelsHour.map(function (key) {
    var values = groupedDataHour[key].value1.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});
```

```
var dataValue2Hour = labelsHour.map(function (key) {
    var values = groupedDataHour[key].value2.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});

var dataValue3Hour = labelsHour.map(function (key) {
    var values = groupedDataHour[key].value3.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});
```

Criação do Gráfico por Hora:

O gráfico por hora é criado utilizando a biblioteca Chart.js. Cada conjunto de dados é representado por uma linha no gráfico.

```
// Criar um gráfico por hora
var ctxHour = document.getElementById('myChartHour').getContext('2d');
var myChartHour = new Chart(ctxHour, {
    type: 'line',
    data: {
        labels: labelsHour,
        datasets: [{
            label: 'Temperatura',
            data: dataValue1Hour,
            borderColor: 'rgba(240, 100, 100, 1)',
            borderWidth: 1.5,
            fill: false
        }, {
            label: 'Umidade do Ar',
            data: dataValue2Hour,
            borderColor: 'rgba(100, 100, 240, 1)',
            borderWidth: 1.5,
            fill: false
        }, {
            label: 'Umidade do Solo',
            data: dataValue3Hour,
            borderColor: 'rgba(100, 240, 100, 1)',
            borderWidth: 1.5,
            fill: false
        }]
    },
    options: {
        scales: {
            x: [{
                type: 'linear',
                position: 'bottom',
                ticks: {
                    stepSize: 1
                }
            }],
            y: [{
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
```

Preparação de Dados para o Gráfico por Dia:

Os dados agrupados por dia são processados para criar três conjuntos de dados: temperatura, umidade do ar e umidade do solo. Após isso ainda é realizada a média de cada um desses valores recebidos durante cada dia.

```
// Preparar dados para o gráfico por dia
var groupedDataDay = {};
var labelsDay = Object.keys(groupedData);

for (var key in groupedData) {
    if (groupedData.hasOwnProperty(key)) {
        var dayKey = key.split(' ')[0];
        if (!groupedDataDay[dayKey]) {
            groupedDataDay[dayKey] = { count: 0, value1: [], value2: [], value3: [] };
        }
        groupedDataDay[dayKey].count += groupedData[key].count;
        groupedDataDay[dayKey].value1 = groupedDataDay[dayKey].value1.concat(groupedData[key].value1);
        groupedDataDay[dayKey].value2 = groupedDataDay[dayKey].value2.concat(groupedData[key].value2);
        groupedDataDay[dayKey].value3 = groupedDataDay[dayKey].value3.concat(groupedData[key].value3);
    }
}

var labelsDay = Object.keys(groupedDataDay);

var dataValue1Day = labelsDay.map(function (key) {
    var values = groupedDataDay[key].value1.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});
```

```
var dataValue2Day = labelsDay.map(function (key) {
    var values = groupedDataDay[key].value2.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});

var dataValue3Day = labelsDay.map(function (key) {
    var values = groupedDataDay[key].value3.filter(function (value) {
        return value >= -50 && value <= 150;
    });

    if (values.length > 0) {
        var sum = values.reduce(function (acc, val) {
            return acc + val;
        }, 0);

        var average = sum / values.length;
        return average.toFixed(2);
    } else {
        return NaN;
    }
});
```

Criação do Gráfico por Dia:

O gráfico por dia é criado utilizando a biblioteca Chart.js. Cada conjunto de dados é representado por uma linha no gráfico.

```
// Criar um gráfico por dia
var ctxDay = document.getElementById('myChartDay').getContext('2d');
var myChartDay = new Chart(ctxDay, {
    type: 'line',
    data: {
        labels: labelsDay,
        datasets: [{
            label: 'Temperatura',
            data: dataValue1Day,
            borderColor: 'rgba(240, 100, 100, 1)',
            borderWidth: 1.5,
            fill: false
        }, {
            label: 'Umidade do Ar',
            data: dataValue2Day,
            borderColor: 'rgba(100, 100, 240, 1)',
            borderWidth: 1.5,
            fill: false
        }, {
            label: 'Umidade do Solo',
            data: dataValue3Day,
            borderColor: 'rgba(100, 240, 100, 1)',
            borderWidth: 1.5,
            fill: false
        }]
    },
    options: {
        scales: {
            x: [{
                type: 'linear',
                position: 'bottom',
                ticks: {
                    stepSize: 1
                }
            }],
            y: [{
                ticks: {
                    beginAtZero: true
                }
            }]
        }
    }
});
```

3.6 Interface Gráfica do Site e Testes:

Foi desenvolvido uma interface gráfica intuitiva para o site, proporcionando uma experiência amigável aos usuários. A seguir são apresentados os gráficos dos dados de temperatura, humidade do ar e humidade do solo gerados.

Gráfico por minuto, da última hora:

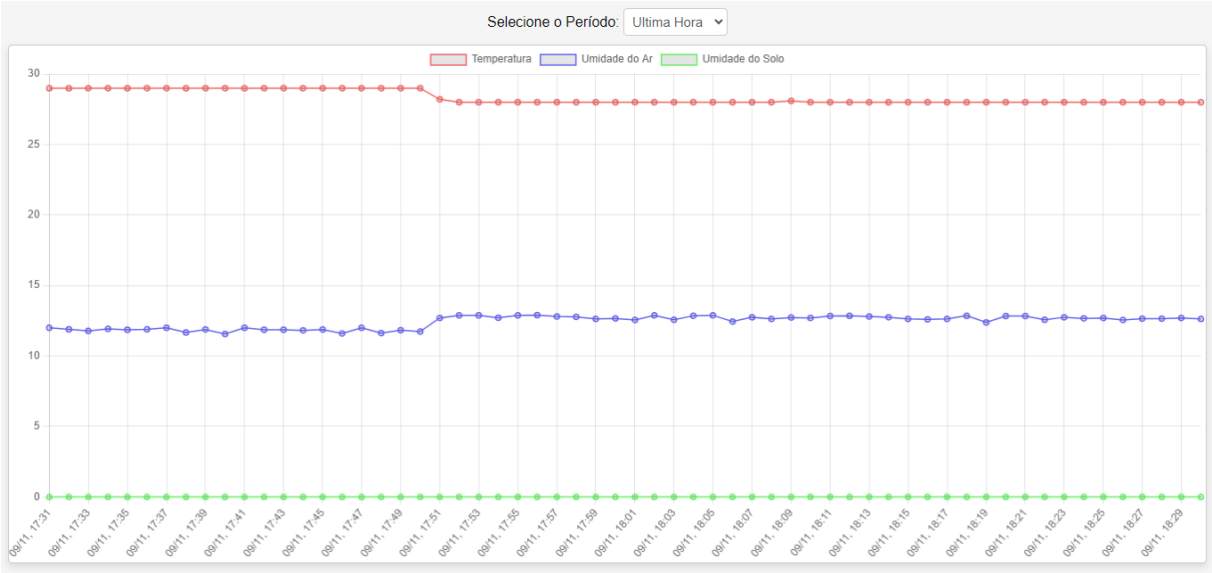


Gráfico por hora, do último dia:

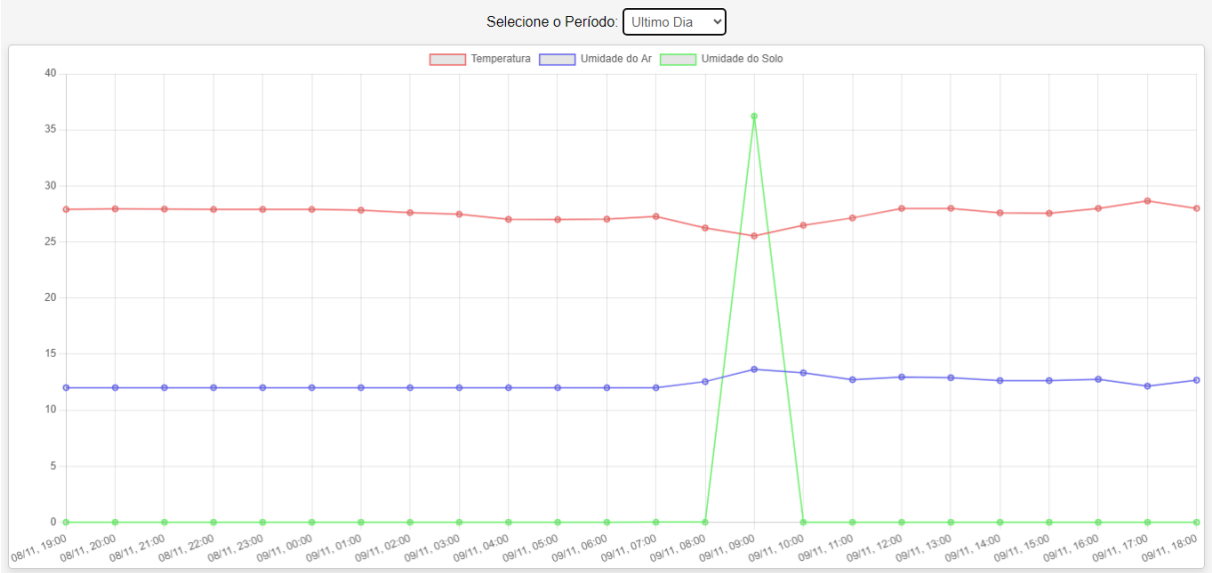
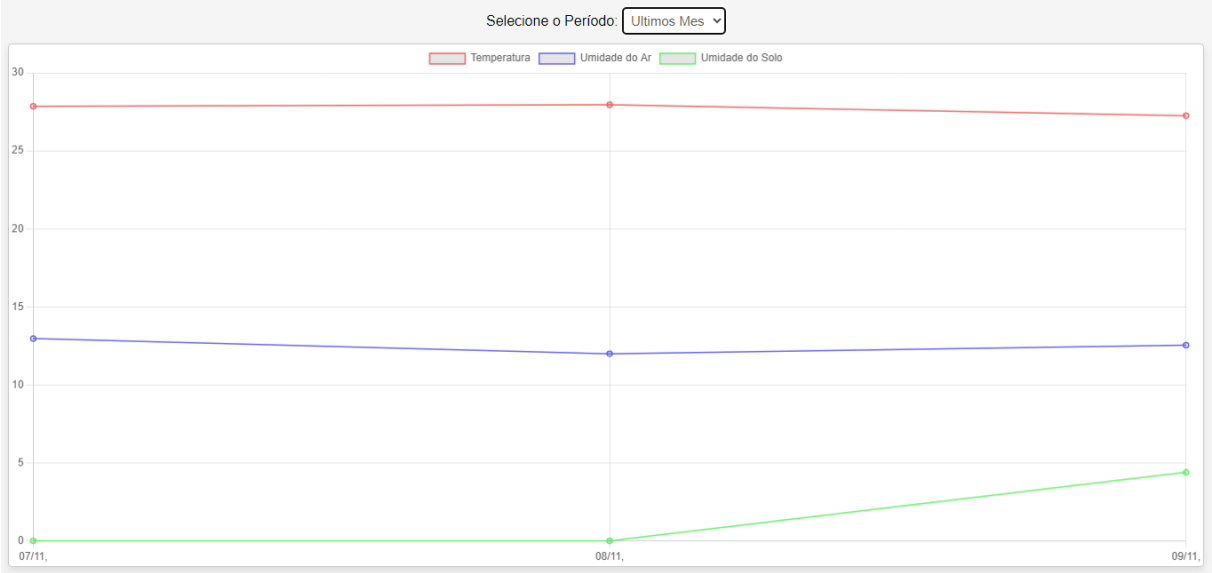


Gráfico por dia, do último mês:



Um ponto interessante, é que com essa interface é possível o usuário selecionar quais dados serão apresentados no gráfico. Por fim, foram realizados testes extensivos para garantir a eficácia da interface e a precisão na apresentação dos dados.

4 DISCUSSÃO

A fase de discussão do projeto oferece uma oportunidade para avaliar criticamente os resultados obtidos, abordar desafios enfrentados durante o desenvolvimento e analisar as implicações práticas do sistema implementado. Esta seção destaca pontos-chave, realiza comparações com as expectativas iniciais e propõe considerações para futuras melhorias.

4.1 Eficiência da Comunicação Sem Fio:

A comunicação entre os dispositivos ESP32 LoRa demonstrou eficiência na transmissão de dados, alcançando uma cobertura adequada e um consumo de energia otimizado. Contudo, eventuais interferências ou obstáculos podem afetar a confiabilidade do sistema em ambientes específicos.

4.2 Precisão e Consistência dos Dados Coletados:

Os sensores de temperatura e umidade do ar e solo apresentaram resultados consistentes e precisos durante os testes. No entanto, é necessário considerar calibrações periódicas para manter a acurácia das leituras ao longo do tempo.

4.3 Integração com o Broker MQTT e Nuvem:

A integração com o broker MQTT Hivemq e o armazenamento dos dados na nuvem foram cruciais para a transferência eficiente dos dados entre dispositivos. A segurança e confiabilidade da conexão foram atendidas, mas é fundamental monitorar a estabilidade da comunicação em longo prazo.

4.4 Armazenamento no PostgreSQL na AWS:

A escolha do PostgreSQL na AWS mostrou-se acertada para o armazenamento de dados. A estrutura relacional proporcionou flexibilidade, e a escalabilidade oferecida pela AWS atendeu às demandas do projeto. Monitoramentos regulares de desempenho devem ser implementados para garantir a manutenção da eficácia do banco de dados.

4.5 Visualização e Interatividade do Site:

A interface gráfica do site, alimentada pelos dados armazenados, proporcionou uma experiência de usuário aprimorada. A visualização de gráficos interativos permitiu uma compreensão mais profunda dos padrões ambientais. Contudo, otimizações contínuas podem ser implementadas para melhorar a velocidade de carregamento e a resposta do site.

4.6 Desafios Superados:

Um dos principais desafios enfrentados durante o desenvolvimento foi a complexidade na programação e lógica de transmissão e recepção entre os dispositivos ESP32 LoRa. A sincronização eficiente e a garantia da integridade dos dados demandaram uma abordagem cuidadosa na elaboração do código, resultando em soluções robustas para a comunicação entre os dispositivos.

A publicação do site e a integração eficaz com o banco de dados PostgreSQL na AWS apresentaram desafios específicos. A configuração inicial dos servidores e a garantia da segurança na transmissão de dados demandaram esforços adicionais. Contudo, por meio de pesquisa extensiva e colaboração, superamos esses desafios, assegurando a estabilidade e confiabilidade do sistema como um todo.

Esses desafios específicos, relacionados à programação intricada e à implementação de serviços web, ressaltam a importância da resolução de problemas durante o desenvolvimento. A superação desses obstáculos não apenas fortaleceu a estrutura do projeto, mas também proporcionou insights valiosos para abordagens futuras e melhorias contínuas.

4.7 Considerações para Melhorias Futuras:

Considera-se a possibilidade de integrar múltiplos transmissores ESP32 LoRa para cobrir áreas mais extensas. Isso exigirá uma arquitetura robusta de comunicação e uma estratégia eficiente para lidar com a agregação de dados provenientes de várias fontes.

A introdução de um gateway centralizado para receber dados de todos os transmissores pode otimizar a comunicação e simplificar a gestão do sistema. Isso também permitirá uma monitoração mais eficaz e a coordenação de dados antes de serem encaminhados para o broker MQTT e o banco de dados.

O site pode ser aprimorado para oferecer funcionalidades adicionais, como páginas de login e cadastro, alertas configuráveis, análises estatísticas mais avançadas e uma interface mais responsiva. A integração de ferramentas de visualização de dados em tempo real pode enriquecer a experiência do usuário.

Estratégias adicionais de segurança devem ser exploradas para proteger a integridade e confidencialidade dos dados transmitidos e armazenados. A implementação de criptografia e autenticação mais robustas é fundamental.

Investigações contínuas sobre técnicas avançadas para otimizar o consumo de energia nos dispositivos ESP32 LoRa são essenciais. Isso pode envolver a exploração de modos de suspensão mais eficientes e a adoção de tecnologias de baixo consumo de energia.

A incorporação de recursos que permitam a manutenção remota dos dispositivos, como atualizações de firmware remotas e diagnósticos, pode simplificar a gestão do sistema ao longo do tempo.

Essas considerações para melhorias futuras visam a expansão e aprimoramento contínuos do sistema, abordando aspectos tanto de infraestrutura quanto de funcionalidade do projeto. Ao incorporar essas melhorias, o projeto estará mais bem equipado para enfrentar desafios futuros e oferecer uma solução cada vez mais eficaz e adaptável às demandas ambientais.

A discussão reflete uma avaliação aprofundada do projeto, identificando sucessos, desafios e áreas potenciais para aprimoramento. Este momento crítico contribui para a compreensão global do projeto e fornece um ponto de partida valioso para futuras iterações e expansões do sistema.

5 Conclusão

O desenvolvimento e implementação deste projeto integrado de monitoramento ambiental revelaram-se bem-sucedidos, proporcionando uma solução abrangente para a coleta, transmissão e visualização de dados em tempo real. A conclusão do projeto é fundamentada nas análises e reflexões sobre as diversas etapas e resultados obtidos.

O projeto atingiu seus objetivos iniciais, demonstrando a eficácia da comunicação sem fio entre os dispositivos ESP32 LoRa, a precisão na coleta de dados ambientais e a transmissão segura para a nuvem por meio do broker MQTT.

A implementação da interface gráfica do site permitiu uma visualização intuitiva e interativa dos dados, facilitando a interpretação por parte dos usuários. A inclusão de gráficos contribuiu para uma compreensão mais aprofundada das condições ambientais ao longo do tempo.
.
O projeto estabeleceu uma base sólida para futuras melhorias e expansões. A inclusão de técnicas avançadas, como melhorias na interface web, a utilização de mais transmissores e sensores, e a exploração de fontes de energia renovável para os dispositivos são áreas promissoras para investigações futuras.

Recomenda-se a continuidade da manutenção regular do sistema, incluindo atualizações de software e calibração dos sensores. A implementação de práticas adicionais de segurança e o monitoramento sobre o estado do banco de dados também são considerações importantes.

Este projeto não apenas cumpriu suas metas estabelecidas, mas também proporcionou uma infraestrutura flexível e robusta para monitoramento ambiental. A conclusão destaca os sucessos alcançados, os aprendizados adquiridos ao longo do processo e aponta para o potencial contínuo de inovação e aprimoramento. Este projeto representa um passo significativo em direção a uma abordagem moderna e tecnologicamente avançada para o monitoramento ambiental.

6 Referências

https://lora-alliance.org/;

https://blog.eletrogate.com/monitoramento-remoto-de-temperatura-utilizando-a-heltec-esp32-lora/;

https://logpyx.com/blog/lora-solucao-para-conectividade-das-coisas/#:~:text=A%20palavra%20LoRa%20vem%20da,modula%C3%A7%C3%A3o%20de%20espectro%20de%20espalhamento.;

https://www.hivemq.com/mqtt/public-mqtt-broker/;

https://www.postgresql.org/about/;