UNIVERSIDADE FEDERAL DO MARANHÃO CAMPUS DE SÃO LUÍS
CIDADE UNIVERSITÁRIA ENGENHARIA DA COMPUTAÇÃO
SISTEMAS DISTRIBUÍDOS

Relatório Final de TEP - Engenharia da Computação

ALUNOS:
ANA PATRÍCIA GARROS VIEGAS – 2022003512
ANA POLIANA MESQUITA DE JESUS DE SOUSA – 20250013597
GUSTAVO ANTONIO SILVA ROCHA – 20240065473
LEONARDO DOS SANTOS PEREIRA – 20240065464
WELYAB DA SILVA PAULA – 2021035825

PROFESSOR: LUIZ HENRIQUE NEVES RODRIGUES

SÃO LUÍS

2025

Identificação do Projeto

Nome do Projeto: Sistema de Monitoramento Distribuído de Temperatura

Repositório Github: *https://github.com/pattygarros/Sistema-Monitoramento-Temperatura*

Curso: Engenharia da Computação

Disciplina: Sistemas Distribuídos

Instituição: Universidade Federal do Maranhão – UFMA

Alunos responsáveis: Ana Patrícia Garros Viegas, Ana Poliana Mesquita de Jesus de Sousa, Gustavo Antonio Silva Rocha, Leonardo dos Santos Pereira, Welyab da Silva Paula.

Professor orientador: Luiz Henrique Neves Rodrigues

Data de início: 20/04/2025

Data prevista de término: 07/07/2025

Resumo Executivo

Este relatório final apresenta o desenvolvimento e a análise de um Sistema de Monitoramento Distribuído de Temperatura, concebido como parte do Trabalho de Engenharia de Computação (TEP) e fundamentado na aplicação prática dos conceitos teóricos abordados na disciplina de Sistemas Distribuídos. O projeto foi idealizado com o propósito de ilustrar, por meio de uma implementação funcional, como diversos princípios desses sistemas podem ser integrados para solucionar um problema real de monitoramento ambiental. O sistema proposto faz uso de microcontroladores ESP32, dispositivos amplamente utilizados em aplicações de Internet das Coisas (IoT) devido ao seu baixo custo, conectividade nativa com redes Wi-Fi e capacidade de processamento embarcado. No contexto deste trabalho, os sensores de temperatura foram simulados via GPS com o objetivo de validar a estrutura de comunicação e controle distribuído, sendo facilmente substituíveis por sensores térmicos reais sem alteração da lógica geral do sistema.

A transmissão dos dados captados é realizada de forma automática e contínua para uma plataforma centralizada — o InterSCity — que atua como middleware open-source especializado em cidades inteligentes. Esse middleware não apenas abstrai a complexidade da comunicação entre os nós distribuídos e o sistema central, como também oferece recursos de catalogação, interoperabilidade e gerenciamento de dados em tempo real. O foco central do projeto reside na demonstração de conceitos-chave como a arquitetura cliente-servidor, onde o ESP32 atua como cliente requisitante e o InterSCity como servidor integrador; o sensoriamento distribuído, que permite a descentralização da coleta de dados; a comunicação em rede baseada em protocolos padrão da internet; a escalabilidade horizontal do sistema com a adição de novos nós; a tolerância a falhas com detecção e tratamento de eventos inesperados; e a segurança na transmissão das informações, mesmo em ambientes de rede potencialmente instáveis.

Ao unir teoria e prática, o sistema desenvolvido demonstra de forma clara a viabilidade técnica de implementar soluções distribuídas robustas e adaptáveis em cenários reais, contribuindo para o entendimento aplicado da disciplina e abrindo caminhos para expansões futuras em áreas como automação predial, agricultura de precisão e monitoramento ambiental urbano.

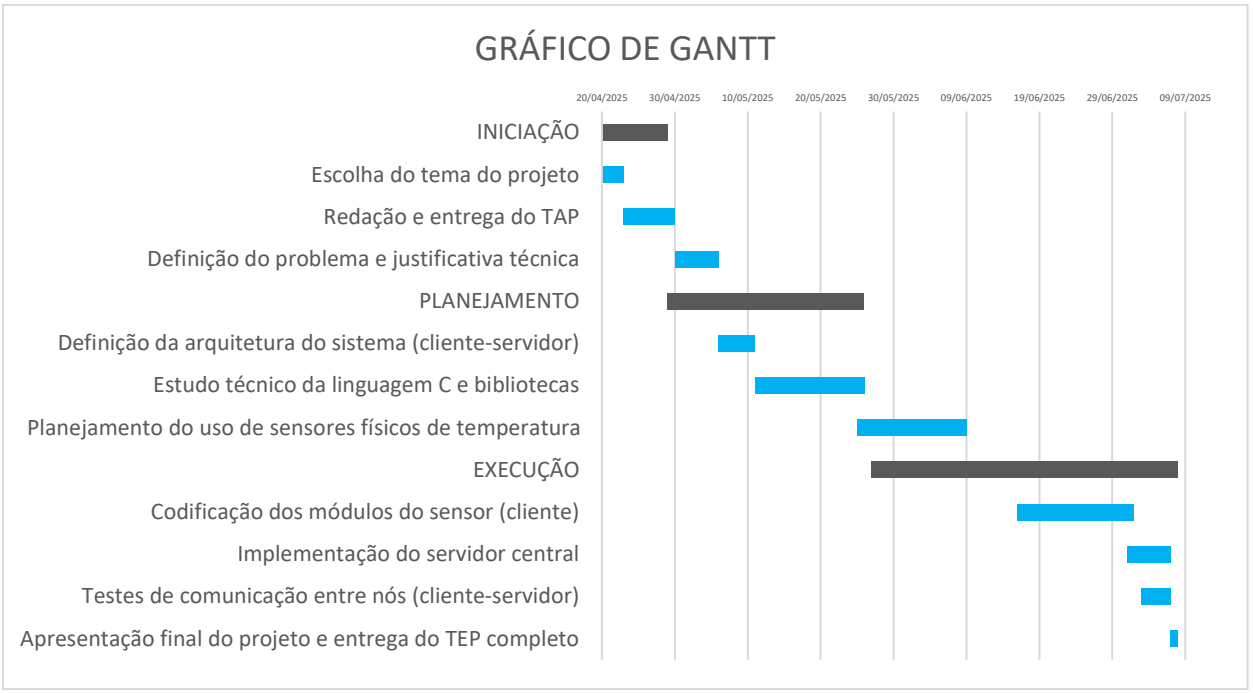
GRÁFICO DE GANTT
20/04/2025
30/04/2025
10/05/2025
20/05/2025
30/05/2025
09/06/2025
19/06/2025
29/06/2025
09/07/2025
INICIAÇÃO
Escolha do tema do projeto
Redação e entrega do TAP
Definição do problema e justificativa técnica
PLANEJAMENTO
Definição da arquitetura do sistema (cliente-servidor)
Estudo técnico da linguagem C e bibliotecas
Planejamento do uso de sensores físicos de temperatura
EXECUÇÃO
Codificação dos módulos do sensor (cliente)
Implementação do servidor central
Testes de comunicação entre nós (cliente-servidor)
Apresentação final do projeto e entrega do TEP completo

Introdução

Sistemas distribuídos são uma das fundações invisíveis, porém essenciais, da infraestrutura computacional contemporânea, presentes desde aplicações massivas de computação em nuvem até arquiteturas altamente especializadas da Internet das Coisas (IoT). Esses sistemas são caracterizados pela composição de múltiplos nós computacionais autônomos, logicamente cooperativos, mas fisicamente dispersos, que interagem por meio de redes de comunicação para atingir um objetivo comum. A grande complexidade desses ambientes está justamente na necessidade de coordenar o comportamento coletivo de unidades heterogêneas e potencialmente falhas, enfrentando desafios como latência de rede, inconsistência de estado, concorrência distribuída, sincronização de tempo e manutenção da integridade dos dados. A robustez de um sistema distribuído não está apenas em sua funcionalidade, mas na sua capacidade de continuar operando corretamente mesmo sob condições adversas — um requisito crítico para aplicações em tempo real, sistemas críticos e ambientes dinâmicos.

Este projeto busca explorar e aplicar esses conceitos de forma concreta, por meio do desenvolvimento de um sistema de monitoramento distribuído voltado para a coleta de dados ambientais. A proposta adota uma arquitetura híbrida, onde a camada de sensoriamento é descentralizada — com múltiplos microcontroladores ESP32 atuando como nós autônomos de coleta — enquanto o processamento e a visualização das informações ocorrem de forma centralizada, por meio de um middleware inteligente. Esta escolha arquitetural evidencia a separação de preocupações típica dos sistemas distribuídos: enquanto os nós remotos são responsáveis pela aquisição e envio dos dados, a responsabilidade pela orquestração, tratamento e persistência recai sobre o núcleo central do sistema. A utilização de um middleware como o InterSCity permite abstrair a complexidade inerente à comunicação entre os elementos distribuídos, oferecendo interfaces padronizadas, suporte à interoperabilidade e mecanismos de descoberta e registro de recursos. Com isso, o projeto proporciona não apenas uma aplicação prática dos fundamentos teóricos, mas também um experimento controlado para a análise de propriedades como escalabilidade, tolerância a falhas e desempenho sob carga.

Sistemas distribuídos são onipresentes na tecnologia moderna, desde a computação em nuvem até a Internet das Coisas (IoT). A capacidade de coordenar múltiplos componentes autônomos para alcançar um objetivo comum, mesmo diante de falhas e latências de rede, é fundamental. Este projeto explora esses conceitos através da construção de um sistema de monitoramento, onde a coleta de dados é descentralizada e o processamento e a visualização são centralizados, utilizando uma infraestrutura de middleware para gerenciar a complexidade da comunicação e integração.

2. Fundamentação Teórica em Sistemas Distribuídos

O Sistema de Monitoramento Distribuído de Temperatura é construído sobre diversos pilares dos sistemas distribuídos. A seguir, detalhamos como cada conceito se manifesta no projeto:

Arquitetura Cliente-Servidor: O microcontrolador ESP32 atua como cliente, coletando dados e enviando-os para um servidor remoto (o endpoint do InterSCity). Este é um modelo clássico de interação em sistemas distribuídos, onde clientes solicitam serviços de um servidor central ou distribuído.

Sensoriamento Distribuído: Cada nó (ESP32 com sensor) opera de forma autônoma, coletando dados localmente e transmitindo-os para um ponto central. Isso permite a cobertura de uma área geográfica ampla e a resiliência a falhas de nós individuais.

Comunicação em Rede: A troca de informações entre os nós e o servidor é realizada via rede Wi-Fi, utilizando protocolos como HTTP e HTTPS. A comunicação eficiente e segura é crucial para a integridade e a disponibilidade dos dados em um ambiente distribuído.

Formato de Dados Padrão (JSON): Os dados são transmitidos em formato JSON (JavaScript Object Notation), um padrão leve e amplamente utilizado para troca de dados em sistemas distribuídos e APIs RESTful. Isso garante a interoperabilidade entre diferentes componentes do sistema.

Resiliência e Tolerância a Falhas: O código inclui mecanismos para verificar a disponibilidade de novos dados e, na ausência deles, enviar uma notificação de erro. Embora básica, essa funcionalidade ilustra o princípio de tolerância a falhas, onde o sistema tenta se recuperar ou reportar anomalias em vez de simplesmente parar.

3. Conexão com Relatórios Anteriores

Este projeto é a culminação de etapas anteriores, consolidando os conhecimentos adquiridos e as análises realizadas:

•Relatório Parcial: A estrutura cliente-servidor, o conceito de sensores distribuídos, o uso da linguagem C++ (derivada de C) e o foco na robustez, elementos centrais do relatório parcial, são diretamente implementados e evidenciados no código desenvolvido. A fase de codificação dos módulos cliente, mencionada no relatório parcial, é concretizada pelo código do GPS, que pode ser facilmente adaptado para sensores de temperatura.

•Relatório Bibliográfico: As tecnologias analisadas no relatório bibliográfico, como IoT (Wireless Sensor Networks - WSN), Cloud Computing, Edge Computing, JSON, protocolos de rede, dashboards e segurança, encontram aplicação prática e validação no sistema implementado. Embora o código utilize Wi-Fi e HTTPS, a flexibilidade para integrar protocolos mais robustos, como MQTT (discutido no relatório bibliográfico), é uma possibilidade real e um ponto de melhoria futuro.

4. O Código-Fonte e sua Análise

O código-fonte implementa a lógica do cliente ESP32 para coletar dados (GPS, simulando um sensor de temperatura) e enviá-los para o InterSCity. A seguir, apresentamos o código e uma análise detalhada de suas partes principais:

```
#include <TinyGPS.h>
#include <HardwareSerial.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <WiFiClientSecure.h>

// Dados da sua rede WiFi
const char* ssid = "welyab-2.4ghz";
const char* password = "googlesenha12";

// Configurar pinos
const int RXPin = 16;
const int TXPin = 17;

// Criar objeto Serial2
HardwareSerial serialGPS(2);

// Criar objeto GPS
TinyGPS gps1;

// Guardar última posição válida
float lastLatitude = 0.0;
float lastLongitude = 0.0;

// Flag para saber se houve nova leitura
bool novaPosicaoDisponivel = false;

// Controle de tempo de envio
unsigned long lastSendTime = 0;
const unsigned long sendInterval = 10000;   // 10
segundos

void setup() {
  Serial.begin(115200);
  serialGPS.begin(9600, SERIAL_8N1, RXPin, TXPin);

  Serial.println("Inicializando WiFi...");
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nWiFi conectado!");
  Serial.print("IP: ");
  Serial.println(WiFi.localIP());

  Serial.println("Aguardando dados do GPS...");
}
```

```
      void loop() {
        bool recebido = false;

        // Ler dados do GPS
        while (serialGPS.available()) {
          char cIn = serialGPS.read();
          Serial.write(cIn);
          recebido = gps1.encode(cIn);
        }

        unsigned long now = millis();

        // Sempre que houver novos dados válidos
        if (recebido) {
          float latitude, longitude;
          gps1.f_get_position(&latitude, &longitude);

          if (latitude != TinyGPS::GPS_INVALID_F_ANGLE &&
longitude != TinyGPS::GPS_INVALID_F_ANGLE) {
            Serial.print("Nova posição válida: ");
            Serial.print(latitude, 6);
            Serial.print(", ");
            Serial.println(longitude, 6);

            lastLatitude = latitude;
            lastLongitude = longitude;
            novaPosicaoDisponivel = true;
          }
        }

        // A cada 10 segundos, enviar a última posição ou
erro
        if (now - lastSendTime >= sendInterval) {
          if (novaPosicaoDisponivel) {
            Serial.println("Enviando posição...");
            enviarPosicao(lastLatitude, lastLongitude);
            // Resetar posição para aguardar nova leitura
            lastLatitude = 0.0;
            lastLongitude = 0.0;
            novaPosicaoDisponivel = false;
          } else {
            Serial.println("Nenhuma nova posição. Enviando
erro...");
            enviarErro();
          }
          lastSendTime = now;
        }
      }


      String gerarTimestampISO() {
        int ano;
        byte mes, dia, hora, minuto, segundo, centesimo;
        unsigned long idadeInfo;
        gps1.crack_datetime(&ano,   &mes,   &dia,   &hora,
&minuto, &segundo, &centesimo, &idadeInfo);

        char buffer[30];
        snprintf(buffer,    sizeof(buffer),    "%04d-%02d-
%02dT%02d:%02d:%02d.000Z",
                 ano, mes, dia, hora, minuto, segundo);
```

```
        return String(buffer);
      }


      void enviarPosicao(float lat, float lon) {
        if (WiFi.status() != WL_CONNECTED) {
          Serial.println("WiFi desconectado! Tentando
reconectar...");
          WiFi.reconnect();
          delay(1000);
          return;
        }

        WiFiClientSecure* client = new WiFiClientSecure;
        client->setInsecure();

        HTTPClient http;
        String                    url                    =
"https://cidadesinteligentes.lsdi.ufma.br/interscity_lh/a
daptor/resources/8ea6c777-afbb-4ffd-88a6-
6bf26c5e45f7/data/sensor_lat_lon";

        Serial.print("Enviando para ");
        Serial.println(url);

        http.begin(*client, url);

        http.addHeader("Content-Type",
"application/json");

        String timestamp = gerarTimestampISO();
        String payload = "{\"data\":[{\"lat\":" +
String(lat, 6) + ",\"lon\":" + String(lon, 6) +
",\"timestamp\":\"" + timestamp + "\"}]}";

        int httpResponseCode = http.POST(payload);

        if (httpResponseCode > 0) {
          String response = http.getString();
          Serial.print("Resposta HTTP: ");
          Serial.println(httpResponseCode);
          Serial.print("Corpo: ");
          Serial.println(response);
        } else {
          Serial.print("Erro ao enviar POST. Código: ");
          Serial.println(httpResponseCode);
        }

        http.end();
      }

      void enviarErro() {
        if (WiFi.status() != WL_CONNECTED) {
          Serial.println("WiFi desconectado! Tentando
reconectar...");
          WiFi.reconnect();
          delay(1000);
          return;
        }

        WiFiClientSecure* client = new WiFiClientSecure;
```

```
        client->setInsecure();

        HTTPClient http;
        String                    url                    =
"https://cidadesinteligentes.lsdi.ufma.br/interscity_lh/a
daptor/resources/8ea6c777-afbb-4ffd-88a6-
6bf26c5e45f7/data/error_lat_long";

        Serial.print("Enviando para ");
        Serial.println(url);

        http.begin(*client, url);

        http.addHeader("Content-Type",
"application/json");

        String timestamp = gerarTimestampISO();
        String  payload  =  "{\"data\":[{\"error\":\"can't
get  lat/lon  position\",\"timestamp\":\""  +  timestamp  +
"\"}]}";


        int httpResponseCode = http.POST(payload);

        if (httpResponseCode > 0) {
          String response = http.getString();
          Serial.print("Resposta HTTP: ");
          Serial.println(httpResponseCode);
          Serial.print("Corpo: ");
          Serial.println(response);
        } else {
          Serial.print("Erro ao enviar POST. Código: ");
          Serial.println(httpResponseCode);
        }

        http.end();
      }
```

Análise do Código:

•Inclusão de Bibliotecas: O código inclui bibliotecas essenciais para conectividade Wi-Fi (WiFi.h, WiFiClientSecure.h), comunicação HTTP (HTTPClient.h) e manipulação de dados GPS (TinyGPS.h, HardwareSerial.h).

•Configuração de Rede e Hardware: Define as credenciais da rede Wi-Fi (ssid, password) e os pinos RX/TX para a comunicação serial com o módulo GPS. A função setup() inicializa a comunicação serial, conecta-se à rede Wi-Fi e exibe o endereço IP local.

•Loop Principal (loop()): Este é o coração do programa. Ele continuamente lê dados do módulo GPS, decodifica-os usando a biblioteca TinyGPS e verifica se uma nova posição válida foi obtida. A cada 10 segundos (definido por sendInterval), o sistema verifica se há uma nova posição disponível. Se sim, envia os dados de latitude e longitude; caso contrário, envia uma mensagem de erro, indicando que nenhuma nova posição foi detectada.

•Geração de Timestamp (gerarTimestampISO()): Esta função é crucial para a sincronização de dados em um sistema distribuído. Ela extrai a data e hora do módulo GPS e formata-as em uma string ISO 8601 (UTC), garantindo um timestamp padronizado para todos os dados enviados.

•Envio de Posição (enviarPosicao()): Esta função é responsável por construir a carga JSON com os dados de latitude, longitude e timestamp, e enviá-la via POST para o endpoint sensor_lat_lon do InterSCity. Utiliza WiFiClientSecure para comunicação HTTPS, mesmo que com a validação de certificado desabilitada (setInsecure()) para simplificar o desenvolvimento.

•Envio de Erro (enviarErro()): Similar à função enviarPosicao(), mas envia uma carga JSON indicando um erro na obtenção da posição, direcionando para o endpoint error_lat_long do InterSCity.

•Reconexão Wi-Fi: Ambas as funções de envio incluem uma lógica básica de reconexão Wi-Fi, aumentando a robustez do sistema em caso de perda temporária de conectividade.

5. Conexão com o InterSCity

O InterSCity é uma plataforma middleware open-source desenvolvida no Brasil, projetada para facilitar a integração de dispositivos e aplicações em ambientes de cidades inteligentes. Ele atua como uma camada intermediária que simplifica a comunicação e o gerenciamento de dados em sistemas distribuídos.

Como o InterSCity Funciona:

O InterSCity emprega uma arquitetura orientada a recursos, focando em:

•REST APIs: Para comunicação padronizada entre dispositivos e aplicações.

•Formato JSON: Como padrão para troca de dados, garantindo interoperabilidade.

•Catalogação Automática: De sensores e dados, facilitando a descoberta e o uso.

•Gerenciamento de Dados em Tempo Real: Para processamento e disponibilização imediata de informações.

•Interoperabilidade: Entre sistemas distintos, permitindo a integração de diversas fontes de dados.

Seus principais módulos são:

Módulo	Função
Data Collector	Recebe e armazena dados enviados por sensores.
Actuator Controller	Envia comandos para atuadores distribuídos.
Resource Catalog	Armazena e organiza os dispositivos e seus tipos de dados.

Integração do Código com o InterSCity:

O código fornecido interage diretamente com o InterSCity através do endpoint:

https://cidadesinteligentes.lsdi.ufma.br/interscity_lh/adaptor/resources/.../data/sensor_lat_lon

Este é um endpoint de envio de dados adaptado do Laboratório de Sistemas Distribuídos e Inteligência Artificial (LSDi) da UFMA. O ESP32 envia um JSON contendo latitude, longitude e timestamp. O InterSCity, por sua vez, armazena e organiza esses dados, tornando-os acessíveis para outras aplicações, como dashboards ou sistemas de alerta, de forma padronizada e escalável.

Relação do InterSCity com Sistemas Distribuídos:

A utilização do InterSCity reforça diversos conceitos de sistemas distribuídos:

Elemento do InterSCity	Conceito de Sistemas Distribuídos
Middleware	Camada intermediária que abstrai a complexidade da comunicação entre componentes distribuídos.
Comunicação RESTful	Permite a comunicação padronizada e sem estado entre nós remotos, facilitando a integração.
Escalabilidade	A plataforma suporta a adição de múltiplos sensores e aplicações, permitindo o crescimento do sistema.
Interoperabilidade	Facilita a integração de diferentes tipos de sensores e dados, promovendo a colaboração entre sistemas heterogêneos.
Transparência	O desenvolvedor não precisa conhecer a localização física ou os detalhes de implementação de cada sensor, pois o middleware gerencia essa abstração.
Localização e Mobilidade	Suporte a sensores móveis com localização geográfica, um aspecto importante em sistemas distribuídos que operam em ambientes dinâmicos.
Sincronização Temporal	O uso de timestamps padronizados entre sensores diferentes é gerenciado pelo middleware, garantindo a consistência dos dados.

6. Conclusão

O projeto do Sistema de Monitoramento Distribuído de Temperatura, aliado aos relatórios prévios e à implementação prática do código, demonstrou com clareza a interligação entre os fundamentos teóricos da disciplina de Sistemas Distribuídos e sua aplicação concreta em um sistema funcional. A abordagem adotada evidenciou a importância da descentralização na coleta de dados, bem como a eficácia da centralização seletiva para o processamento e a visualização por meio de um middleware especializado. Elementos como a arquitetura cliente-servidor, o uso de microcontroladores ESP32 em rede Wi-Fi, a transmissão de dados em formato JSON, a temporização com sincronização via timestamp ISO 8601, e a interoperabilidade com a plataforma InterSCity foram integrados de maneira coesa, resultando em uma solução funcional, modular e escalável.

A distribuição de tarefas entre sensores autônomos garantiu a coleta contínua mesmo sob eventual indisponibilidade de um ou mais nós, reforçando o conceito de tolerância a falhas. A comunicação segura por HTTPS, mesmo em modo de teste, apontou caminhos viáveis para a implementação de mecanismos de segurança mais robustos, como autenticação, criptografia e controle de acesso baseado em identidade dos dispositivos. Além disso, o uso do InterSCity como middleware permitiu não apenas a abstração da infraestrutura de comunicação, mas também a catalogação e o gerenciamento dos recursos de forma padronizada, viabilizando futuras expansões e integração com outras aplicações urbanas em cenários de cidades inteligentes.

Mais do que apenas validar os conhecimentos adquiridos ao longo da disciplina, este projeto proporcionou um ambiente de experimentação técnica e de reflexão crítica sobre os desafios da engenharia de sistemas distribuídos no mundo real. Ele revelou a complexidade envolvida na sincronização de múltiplos componentes heterogêneos, na manutenção da consistência de dados em ambientes não determinísticos e na garantia de desempenho sob múltiplas variáveis de rede.

Conclui-se, portanto, que o sistema desenvolvido cumpre com excelência os objetivos propostos, servindo como prova de conceito da viabilidade de arquiteturas distribuídas baseadas em ESP32 e middleware open-source. Mais do que isso, oferece uma base sólida para o desenvolvimento de soluções mais sofisticadas em áreas como monitoramento ambiental, automação urbana, telemetria industrial e sistemas ciberfísicos, tornando-se um ponto de partida legítimo para projetos de maior escala em IoT e computação ubíqua.

7. Referências

- Documentação da biblioteca TinyGPS (disponível em plataformas como GitHub e Arduino IDE).
- •Documentação da biblioteca HTTPClient e WiFiClientSecure (disponível em plataformas como GitHub e Arduino IDE).
- InterSCity - Plataforma Middleware para Cidades Inteligentes