



## Monitoramento Inteligente de Infraestrutura de Torres de Telecomunicações

Sara Kiyoko Kiyon

<sup>1</sup>Universidade Presbiteriana Mackenzie (UPM)

Rua Alameda dos Guatás, 245 Vila da Saúde, São Paulo - SP, 04053-040 – Brasil

{10407116@mackenzista.br}

**Abstract.** *This paper introduces an innovative solution for monitoring telecommunications towers using Internet of Things (IoT) technology and Arduino. The proposed system employs sensors to measure critical parameters such as temperature, humidity, vibration, and current, enabling real-time data collection from the towers. These data are transmitted to an online platform for visualization and analysis, facilitating the detection of anomalies and enabling predictive maintenance. This approach offers a scalable, cost-effective solution that enhances the efficiency and reliability of telecommunications infrastructure management, contributing to more effective and sustainable operations.*

**Resumo.** *Este artigo apresenta uma solução inovadora para o monitoramento de torres de telecomunicações, utilizando a Internet das Coisas (IoT) e a plataforma Arduino. O sistema desenvolvido integra sensores para medir parâmetros críticos como temperatura, umidade, vibração e corrente elétrica, possibilitando a coleta de dados em tempo real das torres. Esses dados são transmitidos para uma plataforma online, onde são analisados para identificar possíveis anomalias e permitir a manutenção preditiva. A proposta oferece uma solução escalável e de baixo custo, que aprimora a eficiência e a confiabilidade na gestão da infraestrutura de telecomunicações, contribuindo para uma operação mais eficaz e sustentável.*

### 1. Introdução

O gerenciamento eficiente de infraestruturas de telecomunicações é essencial para garantir a qualidade do serviço e minimizar o tempo de inatividade. As torres de telecomunicações são componentes críticos nesse cenário, responsáveis pela transmissão de sinais móveis em áreas urbanas e rurais. Esta enorme rede de torres requer gerenciamento e manutenção altamente visíveis, como descreve a ALC Digital (2018). Com o crescimento da demanda por conectividade, é vital adotar soluções inovadoras que promovam o monitoramento eficiente e a manutenção preditiva.

Historicamente, o monitoramento de torres de telecomunicações dependia de inspeções manuais regulares, que são demoradas e podem ser imprecisas. Com o advento da Internet das Coisas (IoT), tornou-se possível coletar dados em tempo real utilizando sensores conectados. No contexto deste trabalho, propõe-se o desenvolvimento de um sistema de monitoramento que utilize sensores de temperatura, umidade, vibração e corrente integrados a uma plataforma de visualização de dados na nuvem.

Diversos estudos demonstram o uso de IoT na automação e monitoramento de infraestruturas. Um estudo publicado por Juniper Publishers (G, TZAFESTAS; Spyros, 2018), explora a sinergia entre IoT e inteligência artificial para otimizar o uso de recursos em sistemas automatizados.

Similarmente, o monitoramento de energia em sistemas industriais utilizando IoT tem sido amplamente discutido na literatura.

O projeto proposto visa utilizar a plataforma Arduino como base para um sistema de monitoramento de infraestrutura de torres de telecomunicações, oferecendo uma solução de baixo custo e alta escalabilidade.

## **2. Materiais e métodos**

Para o desenvolvimento deste projeto, será utilizada a placa microcontroladora ESP32 (Figura 4), que oferece uma solução de baixo custo e alta eficiência, segundo estudos de Michael Galvão (2023). A ESP32 é amplamente utilizada no mercado como plataforma de desenvolvimento para aplicações de Internet das Coisas (IoT), sendo capaz de lidar com diversos protocolos de comunicação. Além disso, ela permite a leitura e escrita em pinos GPIO (General Purpose Input/Output) e possui conectividade Wi-Fi e Bluetooth, viabilizando a comunicação sem fio com outros dispositivos e a transmissão de dados em tempo real. (GALVÃO; Michel, 2023).

Neste projeto, a ESP32 será responsável por ler os dados de um sensor de umidade, o DHT11 (Figura 1), e distribuir essas leituras via Wi-Fi utilizando o protocolo MQTT (Message Queuing Telemetry Transport). Além disso, a placa irá acionar um LED quando o sensor de umidade atingir um valor preestabelecido. O uso do protocolo MQTT é especialmente adequado para aplicações IoT devido à sua leveza e eficiência na transmissão de mensagens, tornando-o ideal para comunicação em tempo real entre dispositivos conectados.

Link: <https://mqtt.org>

### **2.1 Materiais**

Para a execução do projeto de IoT proposto, serão utilizados os seguintes materiais: uma placa microcontroladora ESP32, que funcionará como o "cérebro" do sistema, sendo responsável pela leitura dos dados do sensor de umidade e temperatura, pela integração em tempo real com a nuvem via o protocolo de comunicação MQTT e pelo controle de um LED.

Além da ESP32, será utilizado o sensor DHT11, que medirá constantemente a umidade e a temperatura do ambiente, enviando esses valores para a ESP32, por meio de comunicação digital, e um LED vermelho que atuará como um sinalizador visual, alertando o usuário caso os valores de temperatura ou umidade ultrapassem um limite predefinido. (SANTOS; Sara, 2020)

### **2.2 Método**

A partir da definição dos materiais utilizados no projeto, será apresentada a metodologia para a implementação da leitura do sensor, comunicação sem fio via Wi-Fi e controle dos atuadores. Primeiramente, será descrito o funcionamento do sensor de umidade e temperatura DHT11. Em seguida, será detalhado o controle do LED vermelho, e por fim, a programação da placa microcontroladora ESP32, com o objetivo de realizar a integração completa do sistema.

### 2.2.1 Sensor de temperatura e umidade DHT11

O sensor utilizado para medir a umidade relativa do ar e a temperatura ambiente é o DHT11 (Figura 1), um sensor digital amplamente conhecido. Ele mede a umidade utilizando um sensor capacitivo, que consiste em dois eletrodos com um material sensível à umidade entre eles. À medida que a umidade do ar varia, as propriedades elétricas desse material também mudam. Essas variações são convertidas em um sinal digital, permitindo que o sensor forneça leituras de umidade que variam entre 20% e 90%, com precisão de  $\pm 5\%$ . (AOSONG; [s.d.]).

O DHT11 também mede a temperatura através de um termistor embutido, cujo princípio de funcionamento baseia-se na diminuição da resistência à medida que a temperatura aumenta. Esse sinal é convertido em uma leitura digital de temperatura, que varia entre  $0^{\circ}\text{C}$  e  $50^{\circ}\text{C}$ , com precisão de  $\pm 2^{\circ}\text{C}$ . (AOSONG; [s.d.]).

O sensor DHT11 utiliza o protocolo de comunicação One-Wire (um fio), (ALMY; Tom, 2021), o que significa que apenas um fio de dados é necessário para transmitir as leituras de temperatura e umidade para o microcontrolador. Desta forma, a ESP32 envia um pulso de início de leitura ao sensor DHT11 que por sua vez responde enviando os dados em formato digital, contendo uma sequência de bits que representam os valores lidos. O sensor tem capacidade de enviar uma leitura por segundo, em média.

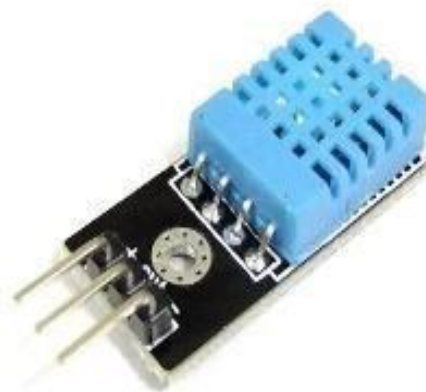


Figura 1

**Fonte:**

[https://www.mouser.com/datasheet/2/830/DFR0067\\_DS\\_10\\_en-2488783.pdf](https://www.mouser.com/datasheet/2/830/DFR0067_DS_10_en-2488783.pdf)

### 2.2.2 Led atuador

Neste projeto, o LED vermelho será utilizado como um atuador, servindo como indicador visual para o usuário, baseado nos valores lidos pelo sensor DHT11. Para seu correto funcionamento, o LED será conectado a um resistor que limita a corrente elétrica, evitando que o LED queime. Após a análise do datasheet do LED, verificou-se que ele opera com uma tensão direta ( $V_f$ ) de 2V e uma corrente nominal ( $I_f$ ) de até 20mA (0,02A).

**Fonte:** <https://cromatek.com.br/datasheet/optoeletronicos/led-pt-5mm-vermelha-encapsulamento-vermelho-difuso-L621.pdf>

Como a alimentação do LED será fornecida pela saída de 3,3V ( $V_{cc}$ ) da ESP32, o valor do resistor necessário foi calculado usando a fórmula:

$$R = \frac{V_{cc} - V_f}{I_f},$$

Substituindo os valores:

$$R = \frac{3,3V - 2V}{0,02} = 65 \, \Omega$$

Desta forma será utilizado o valor comercial mais próximo de 65 Ohms para o resistor do led, no caso o valor de 68 Ohms.(Figura 2)

O LED (Figura 3) será conectado ao pino GPIO D13 da ESP32, e o controle será realizado programaticamente. O LED será acionado sempre que os valores de temperatura ou umidade excederem os limites predefinidos: temperatura superior a 30°C ou umidade acima de 70%. O LED será apagado quando os valores retornarem aos níveis aceitáveis.



Figura 2 - Resistor de 68 Ohms

**Fonte:** [https://www.mouser.com/datasheet/2/414/TTRB\\_S\\_A0011519831\\_1-2565653.pdf](https://www.mouser.com/datasheet/2/414/TTRB_S_A0011519831_1-2565653.pdf)

### **LED:**



Figura 3

**Fonte:** <https://cromatek.com.br/datasheet/optoeletronicos/led-ptb-5mm-vermelha-encapsulamento-vermelho-difuso-L621.pdf>

### **2.2.3 ESP32**

A placa microcontroladora ESP32 (figura 4) desempenha o papel central no projeto, integrando os sensores e atuadores. Além disso, ela será responsável por enviar os dados lidos em tempo real para a internet, utilizando o protocolo de comunicação MQTT. Esse protocolo é ideal para aplicações IoT (Internet of Things) devido à sua eficiência na transmissão de mensagens entre dispositivos conectados. (GALVÃO; Michel, 2023).



Figura 4 - Placa ESP32

**Fonte:** [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf)

Inicialmente, para desenvolver o projeto utilizando a ESP32, é necessário configurar o ambiente de desenvolvimento. Para isso, será utilizada a IDE (Integrated Development Environment) do Arduino, que facilita a integração com a ESP32. (VIANA; Carol, 2022) . A IDE Arduino permite baixar e configurar as bibliotecas necessárias para a

compilação do firmware que será usado para programar a ESP32 de acordo com os requisitos do projeto. Com o ambiente de desenvolvimento configurado, será possível iniciar o processo de upload dos códigos para a placa (SANTOS; Sara, 2020).

O desenvolvimento do projeto foi organizado em três etapas principais:

**Leitura do Sensor de Umidade e Temperatura:** O primeiro passo é desenvolver um código que realize a leitura dos dados fornecidos pelo sensor DHT11. O código permitirá que a ESP32 receba as leituras de umidade e temperatura de forma contínua.

**Controle do LED Vermelho:** O segundo passo é desenvolver um código para controlar o LED vermelho, que atuará como um indicador visual. O LED será acionado quando os valores de umidade ou temperatura ultrapassarem determinados limites predefinidos.

**Protocolo de Comunicação MQTT:** O terceiro passo envolve a criação de um programa que conecte a ESP32 à rede Wi-Fi e permita a comunicação via MQTT. O protocolo MQTT será utilizado para enviar os dados lidos pelo sensor para um servidor na nuvem, onde poderão ser monitorados e analisados remotamente.

Após o desenvolvimento de cada uma dessas partes, todos os programas serão integrados, permitindo a leitura dos sensores em tempo real, o envio dos dados pela internet utilizando o protocolo MQTT e o controle do LED com base nas leituras do sensor. Dessa forma, o projeto permitirá tanto o monitoramento remoto dos dados de temperatura e umidade quanto o feedback visual local, criando uma solução IoT completa.

#### 2.2.3.1 Leitura do Sensor de Umidade e Temperatura DHT11

Para que seja possível realizar a leitura do sensor de umidade e temperatura DHT11, será necessário instalar a biblioteca "DHT sensor library" da Adafruit (Figura 5). Esta biblioteca facilita a comunicação com o sensor, gerenciando automaticamente o protocolo One-wire (um fio) e retornando os valores lidos de umidade e temperatura de maneira simplificada.

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
```

Figura 5 - Biblioteca Adafruit DHT Sensor Library

**Fonte:** <https://github.com/adafruit/DHT-sensor-library>

Após a instalação da biblioteca, a inicialização do sensor é feita com alguns passos simples. Primeiro, é preciso definir qual pino digital da ESP32 será utilizado para a leitura. Por exemplo, se o pino 15 for escolhido, você deve associá-lo ao sensor DHT11. A seguir, utiliza-se a função **DHT dht(pin, DHTTYPE)** para inicializar o sensor, onde o **DHTPIN** corresponde ao pino de leitura (neste caso, o pino 15), e **DHTTYPE** corresponde ao tipo

de sensor que será utilizado, que no caso deste projeto será o utilizado o sensor **DHT11**. (Figura 6)

```
// Definir o pino onde o DHT11 está conectado
#define DHTPIN 15 // Pino digital da ESP32 conectado ao pino Data do DHT11

// Definir o tipo de sensor DHT
#define DHTTYPE DHT11 // Escolher DHT11 (pode usar DHT22 se for o caso)

// Inicializar o sensor DHT
DHT dht(DHTPIN, DHTTYPE);
```

Figura 6 - Inicialização do sensor de humidade

**Fonte:** <https://github.com/arduino/arduino-ide>

Depois de inicializar o sensor, basta chamar as funções **dht.readHumidity()** e **dht.readTemperature()** da própria biblioteca Adafruit (Figura 7) para realizar a leitura dos dados de umidade e temperatura. Estas funções retornam os valores lidos diretamente, permitindo que os dados sejam utilizados para controle de atuadores ou envio para a nuvem via MQTT.

```
// Ler umidade
float humidity = dht.readHumidity();

// Ler temperatura em Celsius
float temperature = dht.readTemperature();
```

Figura 7 - Funções para leitura do sensor

**Fonte:** <https://github.com/arduino/arduino-ide>

### 2.2.3.1 Controle do LED Vermelho

Para o controle do LED, não é necessário instalar nenhuma biblioteca adicional, pois a IDE do Arduino (VIANA; Carol, 2022) . já oferece suporte nativo para essa funcionalidade. Basta definir o pino da ESP32 que será utilizado para controlar o LED e configurá-lo como uma saída digital. No caso do projeto será utilizado o GPIO2 (Figura 8) da ESP32 para controlar o led.

```
// Definir o pino do LED
#define LED_PIN 2 // No caso da ESP32, o LED embutido está no GPIO 2
```

Figura 8 - Definição do pino GPIO a ser utilizado

**Fonte:** <https://github.com/arduino/arduino-ide>

Ao escolher o pino de GPIO que será utilizado também é necessário definir tal pino como saída digital utilizando da função **pinMode()** (Figura 9)

```
// Configurar o pino do LED como saída
pinMode(LED_PIN, OUTPUT);
}
```

Figura 9 -Configuração do Pino como Saída

**Fonte:** <https://github.com/arduino/arduino-ide>

Após a configuração, o LED pode ser controlado com as funções **digitalWrite()**. Para acender o LED, utiliza-se **digitalWrite(pino, HIGH)** e para apagá-lo, **digitalWrite(pino, LOW)** (Figura 10).

Esses comandos simples permitem que o LED seja acionado de acordo com as leituras do sensor DHT11, servindo como um indicador visual para os valores de umidade e temperatura que excedam os limites estabelecidos no projeto.

```
// Acender o LED
digitalWrite(LED_PIN, HIGH);
delay(1000); // Aguardar 1 segundo (1000 milissegundos)

// Apagar o LED
digitalWrite(LED_PIN, LOW);
delay(1000); // Aguardar 1 segundo (1000 milissegundos)
```

Figura 10 - Comandos de controle do LED

**Fonte:** <https://github.com/arduino/arduino-ide>

### 2.2.3.3 Protocolo de Comunicação MQTT



Por fim, será desenvolvida a programação do protocolo de comunicação **MQTT** para enviar os dados lidos pelos sensores através da conectividade Wi-Fi da ESP32. Para implementar essa funcionalidade, será utilizada a biblioteca **PubSubClient** na Arduino IDE, que facilita a integração com o MQTT. A ESP32 enviará os dados coletados para o **ThingSpeak**, uma plataforma usada para visualização em tempo real de dados de IoT. No contexto deste projeto, o ThingSpeak atuará como o broker MQTT, recebendo as mensagens enviadas pela ESP32.

**LINK:** [https://thingspeak.mathworks.com/pages/learn\\_more](https://thingspeak.mathworks.com/pages/learn_more)

## **Funcionamento do MQTT no Projeto**

O **MQTT** (Message Queuing Telemetry Transport) é um protocolo de comunicação leve e eficiente, especialmente projetado para aplicações de Internet das Coisas (IoT). Ele é ideal para dispositivos de baixa potência, como a ESP32, pois consome pouca largura de banda, permitindo a transmissão de dados em tempo real com mínima sobrecarga.

A ESP32 enviará periodicamente os dados de temperatura e umidade lidos pelo sensor DHT11 para o broker MQTT (ThingSpeak - Figura 11), utilizando tópicos específicos para cada tipo de dado (como **/sensor/temperatura** e **/sensor/umidade**). O ThingSpeak, por sua vez, armazenará e exibirá essas informações em gráficos, facilitando o monitoramento remoto em tempo real.

### **Etapas de Implementação:**

1. **Conexão Wi-Fi:** A ESP32 se conectará à rede Wi-Fi local, garantindo que possa enviar os dados para a nuvem.
2. **Configuração do Cliente MQTT:** A biblioteca **PubSubClient** será utilizada para configurar o cliente MQTT da ESP32. Ela se conectará ao broker (ThingSpeak) utilizando um endereço IP ou hostname, além de definir os tópicos de publicação. (G; TZAFESTAS, Spyros, 2018).
3. **Publicação de Dados:** Após a leitura dos dados pelo sensor DHT11, a ESP32 publicará esses valores nos tópicos MQTT definidos. O broker ThingSpeak coletará e organizará os dados recebidos.
4. **Visualização dos Dados:** Os dados coletados serão exibidos em gráficos na interface do ThingSpeak. Isso permitirá o monitoramento em tempo real dos valores de temperatura e umidade proporcionando um fácil acesso às informações em qualquer lugar com acesso à internet.

Essa implementação simplifica a comunicação entre o dispositivo ESP32 e a nuvem, utilizando o MQTT para garantir um envio eficiente e em tempo real dos dados coletados, otimizando o projeto para aplicações IoT.

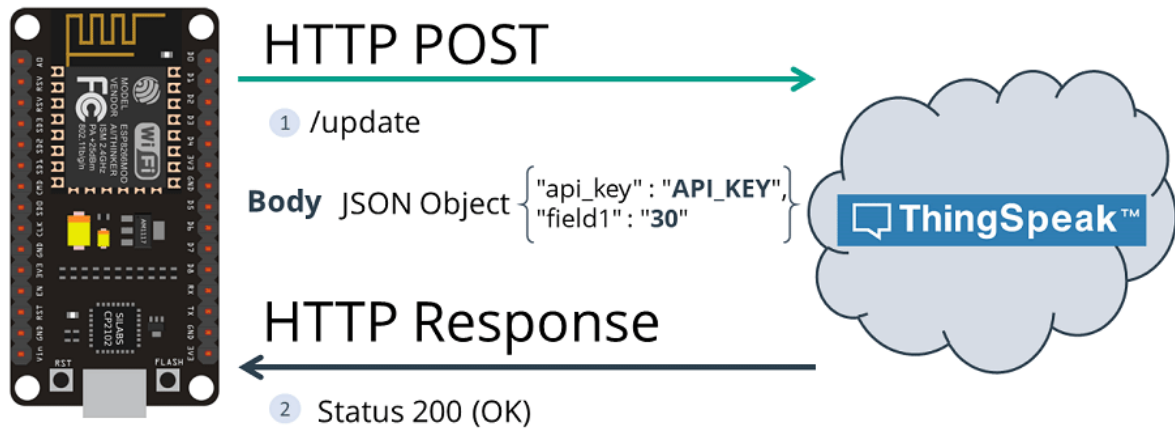


Figura 11 - Comunicação ESP32 com o ThingSpeak

**Fonte:** <https://randomnerdtutorials.com/esp8266-nodemcu-http-post-ifttt-thingspeak-arduino/>

### Etapa 1 Conexão Wi-Fi:

Para realizar a conexão Wi-Fi, foi utilizada a biblioteca <WiFi.h> do Arduino, que pode ser instalada diretamente pela IDE e é específica para o módulo Wi-Fi da ESP32. Para usar essa biblioteca, é necessário configurar o login e a senha da rede Wi-Fi local (Figura-12). Em seguida, a conexão é iniciada com a função **WiFi.begin(ssid, password)** (Figura-13). Por fim, para verificar se a conexão foi estabelecida com sucesso, utiliza-se a função **WiFi.status()**, que retorna o valor **WL\_CONNECTED** caso a conexão tenha sido realizada corretamente (Figura-13), caso contrário o código fica em loop printando “.” até que seja possível realizar a conexão com o WI-FI.

```
// Credenciais do WiFi
const char* ssid = ""; // Login Wi-fi
const char* password = ""; // Senha Wi-Fi
```

Figura 12 - Define login de wi-fi

```
// Inicialização do WiFi
void setup_wifi() {
  Serial.println("\nsetup_wifi setup_wifi");

  delay(10);
  Serial.println("Connecting to WiFi..");
  WiFi.begin(ssid, password);

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

  Serial.println("\nWiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
```

Figura 13 - Inicia a conexão wi-fi

## Etapa 2 Configuração do Cliente MQTT:

Após a conexão Wi-Fi ser estabelecida, é possível configurar o código para se conectar ao servidor MQTT. Para isso, é necessário, primeiramente, instalar a biblioteca <ThingSpeak.h> pela IDE do Arduino. Em seguida, o código deve ser configurado para se comunicar com o servidor utilizando as credenciais de canal, e chaves de leitura e escrita da API, fornecidas pelo ThingSpeak.

```
// Credenciais do ThingSpeak
unsigned long channelID = ; // Add your channel ID
const char* writeAPIKey = "";
const char* readAPIKey = "";

// Endpoint do ThingSpeak
const char* server = "api.thingspeak.com";
```

Figura 14 - Define login no servidor ThingSpeak

**Fonte:** <https://github.com/mathworks/thingspeak-arduino>

Por fim, para inicializar a comunicação da ESP32 como cliente do servidor ThingSpeak é utilizado a função **ThingSpeak.begin(client)** que define o WI-FI local como cliente (Figura -15 e figura-16).

```
// Inicialização do WiFi como cliente
WiFiClient client;
```

Figura 15 - Define o wi-fi local como cliente do ThingSpeak

```
// Inicializa o ThingSpeak
ThingSpeak.begin(client);
```

Figura 16 - Define o wi-fi como cliente do servidor ThingSpeak

### Etapa 3 Publicação de Dados:

Após estabelecer a conexão entre o Wi-Fi local e o servidor ThingSpeak, é possível publicar os dados lidos pelo sensor de umidade e temperatura no servidor. Para isso, é necessário definir os campos de dados que serão enviados via HTTP, utilizando as funções **ThingSpeak.setField(1, temperature);** para a temperatura e **ThingSpeak.setField(2, humidity);** para a umidade (Figura-17).

```
// Define o valor dos campos de temperatura e umidade
ThingSpeak.setField(1, temperature);
ThingSpeak.setField(2, humidity);
```

Figura 17 - Define os campos de dados a serem enviados

Após definir os campos de dados a serem enviados, é possível realizar uma requisição ao servidor para enviar os dados utilizando a função **ThingSpeak.writeFields(channelID, writeAPIKey);**. Essa função retorna o código **HTTP 200** caso os dados sejam enviados com sucesso. Se o valor retornado for diferente de **200**, isso indica que houve uma falha no envio, e o motivo pode variar conforme o código de resposta recebido (Figura-18).

```
// Envia o valor dos sensores para o ThinkSpeak
int response = ThingSpeak.writeFields(channelID, writeAPIKey);

if (response == 200) {
    Serial.println("Channel update successful");
} else {
    Serial.println("Problem updating channel. HTTP error code " + String(response));
}
```

Figura 18 - Envia uma requisição HTTP para o servidor ThingSpeak

### Etapa 4 Visualização dos Dados:

Por fim, para visualizar os dados, é necessário criar uma conta no ThinkSpeak e configurar um canal destinado a receber os dados enviados pela ESP32. Durante a criação do canal, devem ser geradas as credenciais que precisam ser adicionadas ao código da ESP32. Essas

credenciais permitem que o servidor se comunique com o dispositivo via Wi-Fi e receba os dados corretamente.

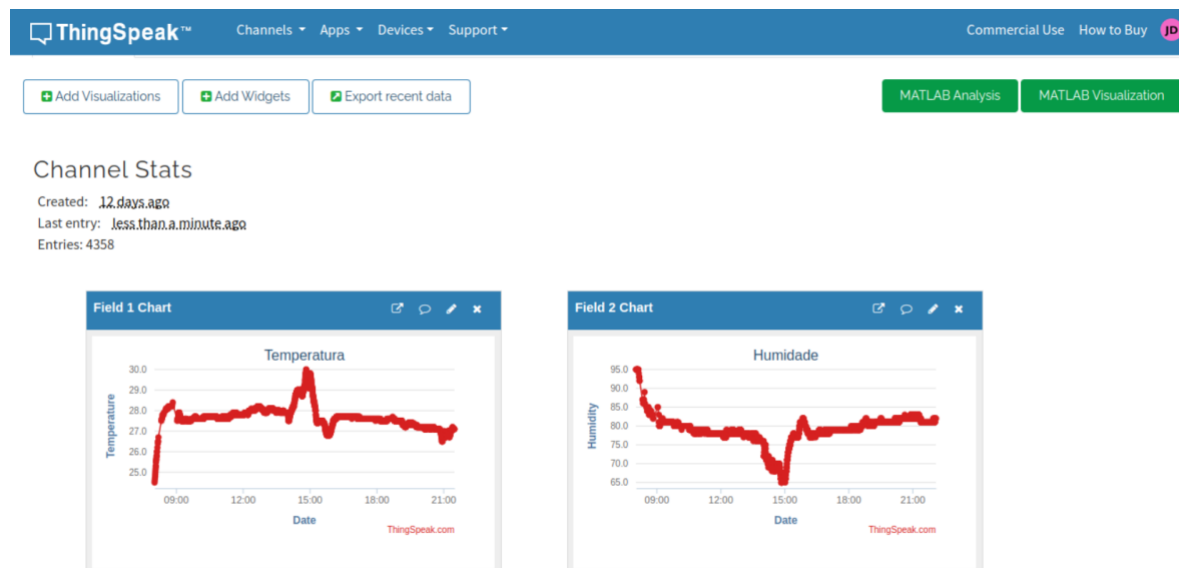


Figura 19 - Canal no servidor ThingSpeak

**Fonte:** <https://thingspeak.mathworks.com/>

## Montagem

Com a finalização dos programas individuais, será realizada a integração de todos esses componentes para o controle completo da aplicação IoT. Para isso, será seguido o esquema das Figuras 12 e 13, que definem as conexões elétricas necessárias entre a ESP32, o sensor DHT11 e o LED. Esse esquemático foi desenvolvido baseado nas especificações técnicas descritas nos respectivos datasheets de cada componente, garantindo a correta operação de todo o sistema.

Para que o sensor de umidade DHT11 funcione corretamente, é necessário conectar os terminais do sensor GND e o VCC nos terminais de alimentação da ESP32. Além disso, deve-se adicionar um resistor de 10K Ohms entre a porta de entrada do VCC do sensor e a porta de dados. (AOSONG; [s.d.]).

Quanto ao controle do LED, é preciso conectar um dos terminais do LED ao VCC utilizando um resistor de 68 Ohms, conforme definido na seção anterior. E o outro terminal do LED deve ser conectado ao pino GPIO 2 da ESP32.

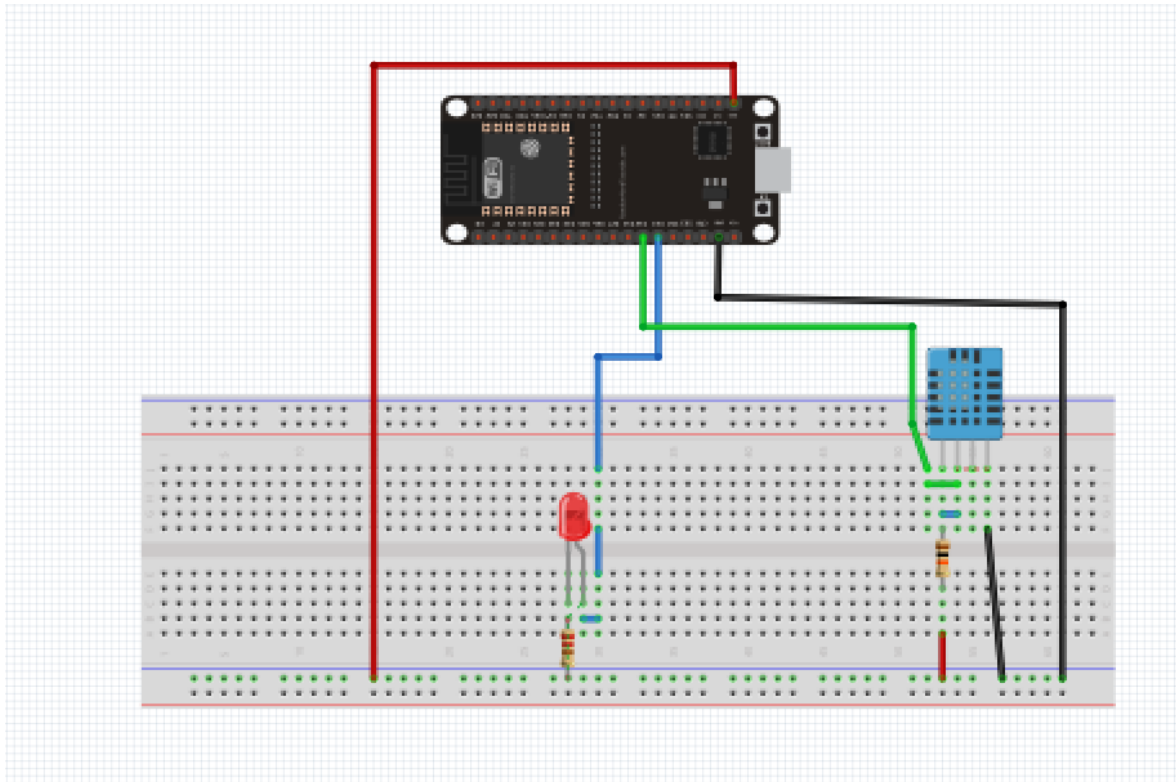


Figura 20 - Montagem do projeto na protoboard

**Fonte:** Elaboração própria (2024)

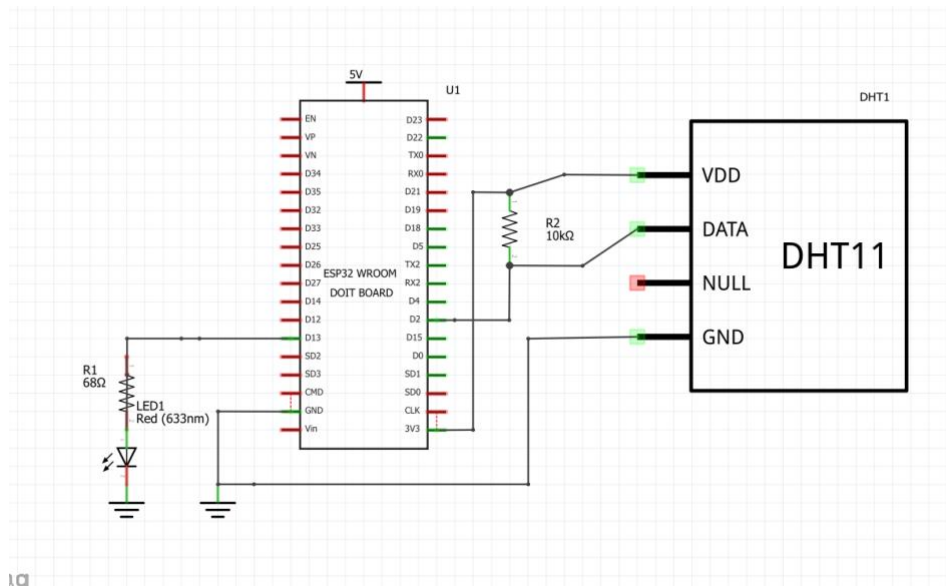


Figura 21 - Esquemático do projeto desenvolvido

**Fonte:** Elaboração própria (2024)

### 3. Resultados

Após a definição do projeto e das especificações de montagem, foi possível realizar a implementação prática utilizando os seguintes componentes: a ESP32, destacada pelo número 1 na Figura 22; um sensor de temperatura e umidade DHT11, representado pelo número 2; e um LED com um resistor, identificados pelo número 3 na mesma figura. Essa montagem seguiu as conexões elétricas previamente especificadas, garantindo a integração completa do sistema.

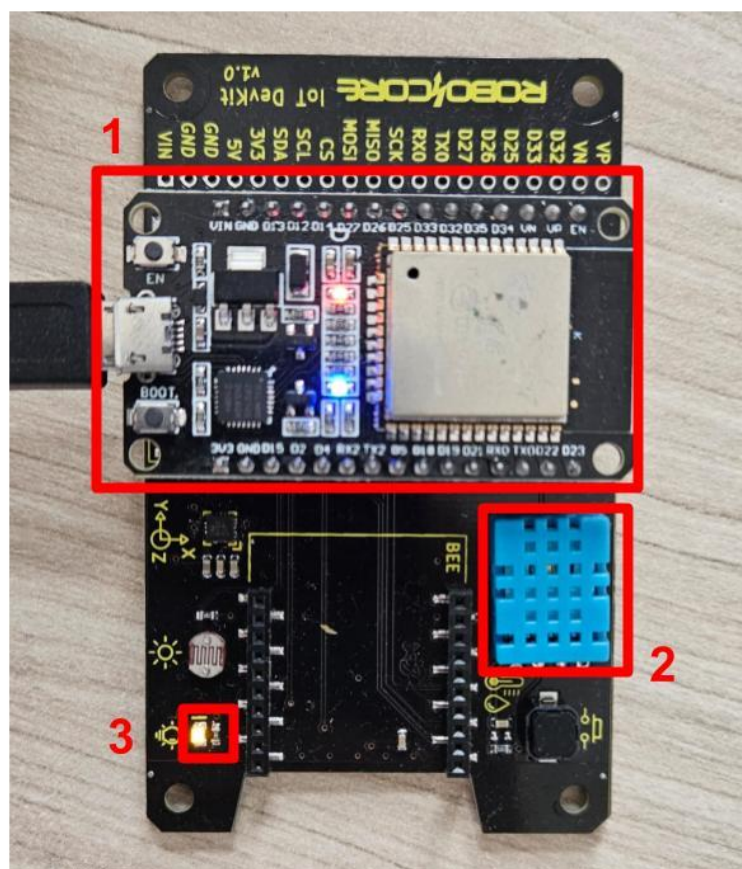
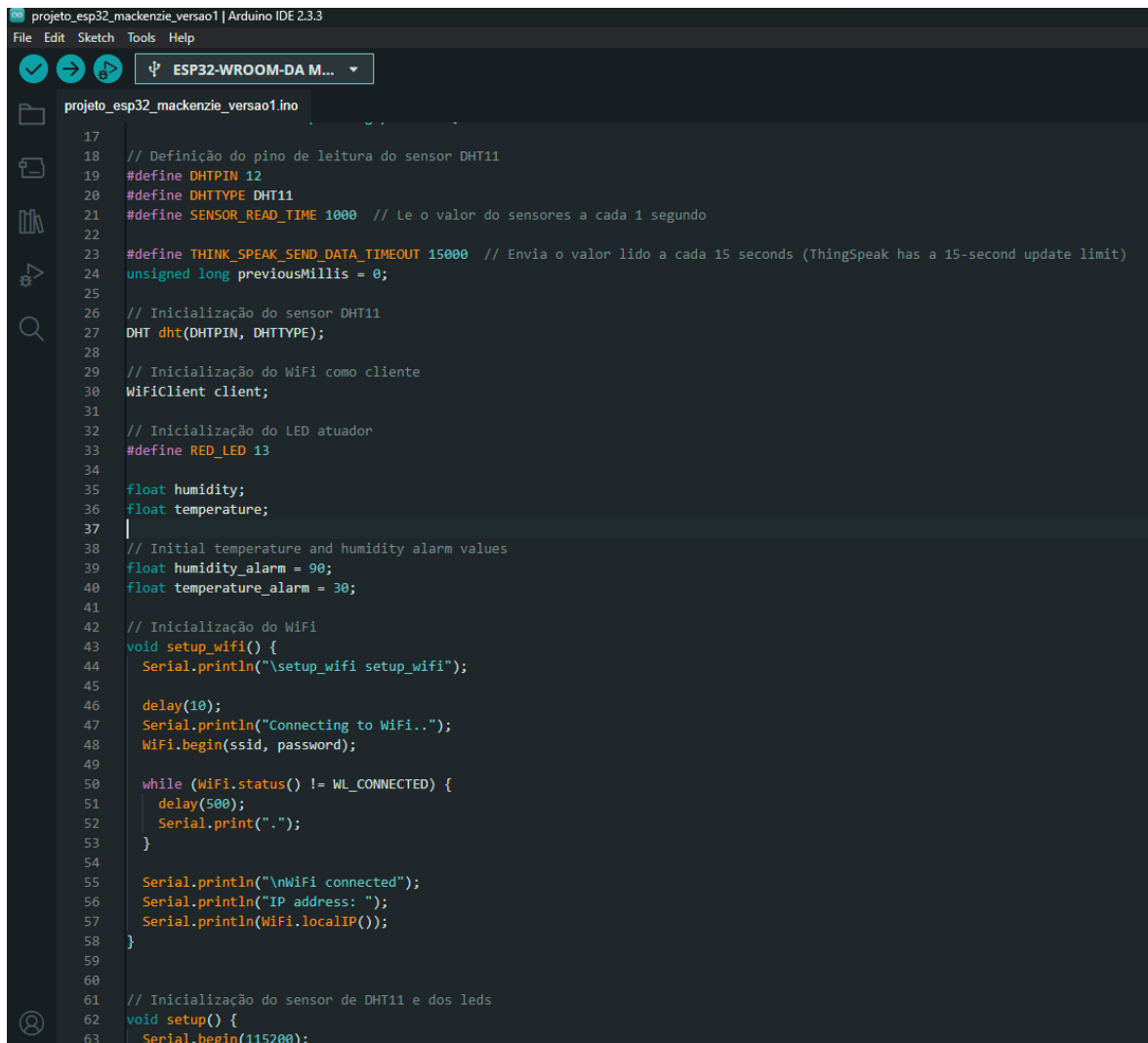


Figura 22 - Montagem do projeto

Além disso, foi desenvolvido o código utilizando a IDE do Arduino conforme a ilustração da figura 23,





```
17
18 // Definição do pino de leitura do sensor DHT11
19 #define DHTPIN 12
20 #define DHTTYPE DHT11
21 #define SENSOR_READ_TIME 1000 // Le o valor do sensores a cada 1 segundo
22
23 #define THINK_SPEAK_SEND_DATA_TIMEOUT 15000 // Envia o valor lido a cada 15 seconds (ThingSpeak has a 15-second update limit)
24 unsigned long previousMillis = 0;
25
26 // Inicialização do sensor DHT11
27 DHT dht(DHTPIN, DHTTYPE);
28
29 // Inicialização do WiFi como cliente
30 WiFiClient client;
31
32 // Inicialização do LED atuador
33 #define RED_LED 13
34
35 float humidity;
36 float temperature;
37
38 // Initial temperature and humidity alarm values
39 float humidity_alarm = 90;
40 float temperature_alarm = 30;
41
42 // Inicialização do WiFi
43 void setup_wifi() {
44     Serial.println("\nsetup_wifi setup_wifi");
45
46     delay(10);
47     Serial.println("Connecting to WiFi..");
48     WiFi.begin(ssid, password);
49
50     while (WiFi.status() != WL_CONNECTED) {
51         delay(500);
52         Serial.print(".");
53     }
54
55     Serial.println("\nWiFi connected");
56     Serial.println("IP address: ");
57     Serial.println(WiFi.localIP());
58 }
59
60 // Inicialização do sensor de DHT11 e dos leds
61 void setup() {
62     Serial.begin(115200);
63 }
```

Figura 23 - Código na IDE arduino

Após compilar o código e realizar o upload na placa microcontrolada ESP32, é possível abrir o terminal serial para monitorar o comportamento dos sensores e do LED de alarme. A Figura 24 apresenta um exemplo de saída no terminal serial, onde são exibidos os valores de temperatura e umidade lidos pelo sensor. Além disso, é possível verificar se o alarme do LED de alarme está ativo, caso a temperatura ultrapasse 30°C ou a umidade exceda 90%. Por fim, o terminal também mostra as tentativas da ESP32 de enviar os dados lidos via Wi-Fi. Quando a transmissão é bem-sucedida, uma mensagem de confirmação é exibida, indicando que os canais no ThinkSpeak foram atualizados com sucesso.



```
Temperature: 28.10°C, Humidity: 84.00%  
Check led alarm  
Temperature: 28.10°C, Humidity: 84.00%  
Turn off alarm  
Temperature: 28.10°C, Humidity: 84.00%  
Check led alarm  
Temperature: 28.10°C, Humidity: 84.00%  
Turn off alarm  
Temperature: 28.10°C, Humidity: 84.00%  
Try to send data to thinkSpeak  
Channel update successful
```

Figura 24 - Monitoramento serial

Além do monitoramento pelo terminal serial, os valores lidos pelos sensores de temperatura e umidade também podem ser acompanhados na plataforma ThingSpeak. Nela, é possível observar um novo ponto sendo adicionado ao gráfico a cada 15 segundos, que corresponde ao intervalo mínimo exigido entre as requisições ao servidor.

Para ilustrar os resultados, foi realizado o monitoramento da temperatura e umidade em um ambiente fechado, próximo a uma janela, no dia 6 de novembro de 2024. Os resultados estão apresentados nas Figuras 25 e 26. Os gráficos mostram que a temperatura máxima registrada durante o dia foi de 30°C às 14h50, coincidente com o valor mínimo de umidade, 65%. Esse horário corresponde ao momento em que o sol incidia diretamente na janela, onde a ESP32 estava posicionada. Em seguida, observou-se uma queda gradual da temperatura até 27°C ao entardecer, acompanhada por um aumento na umidade do ar para aproximadamente 80%.

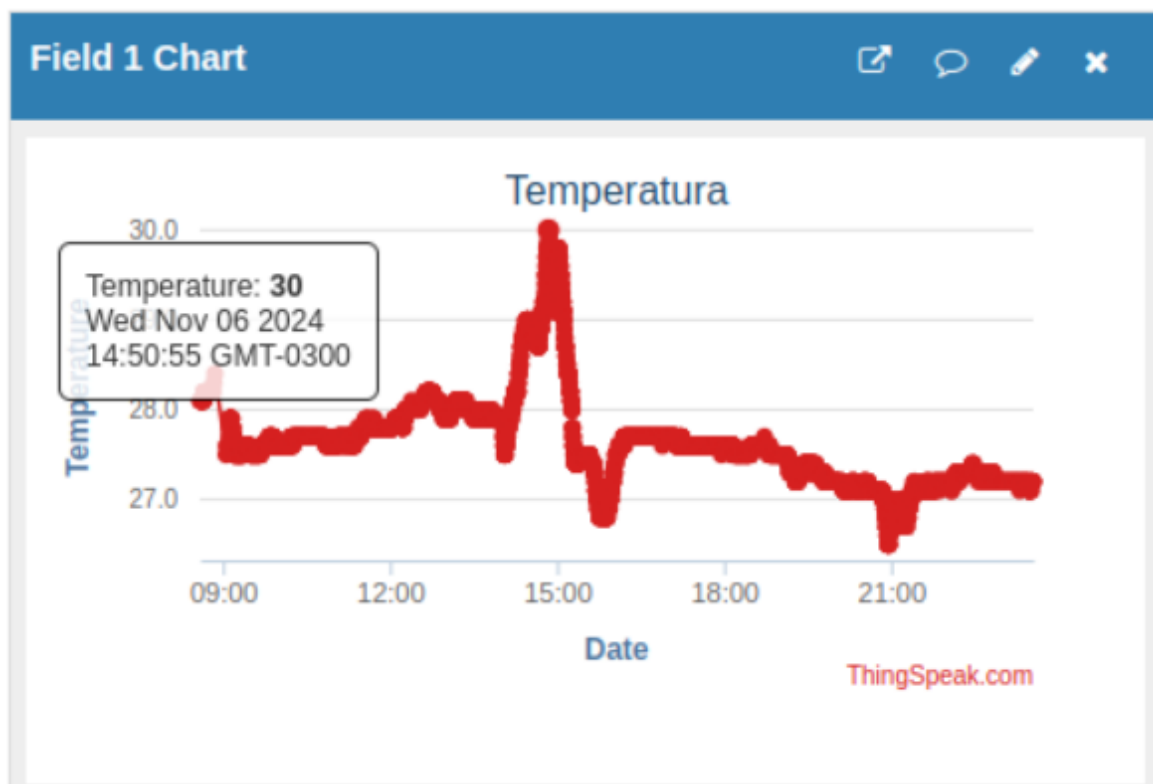


Figura 25 - Monitoramento serial

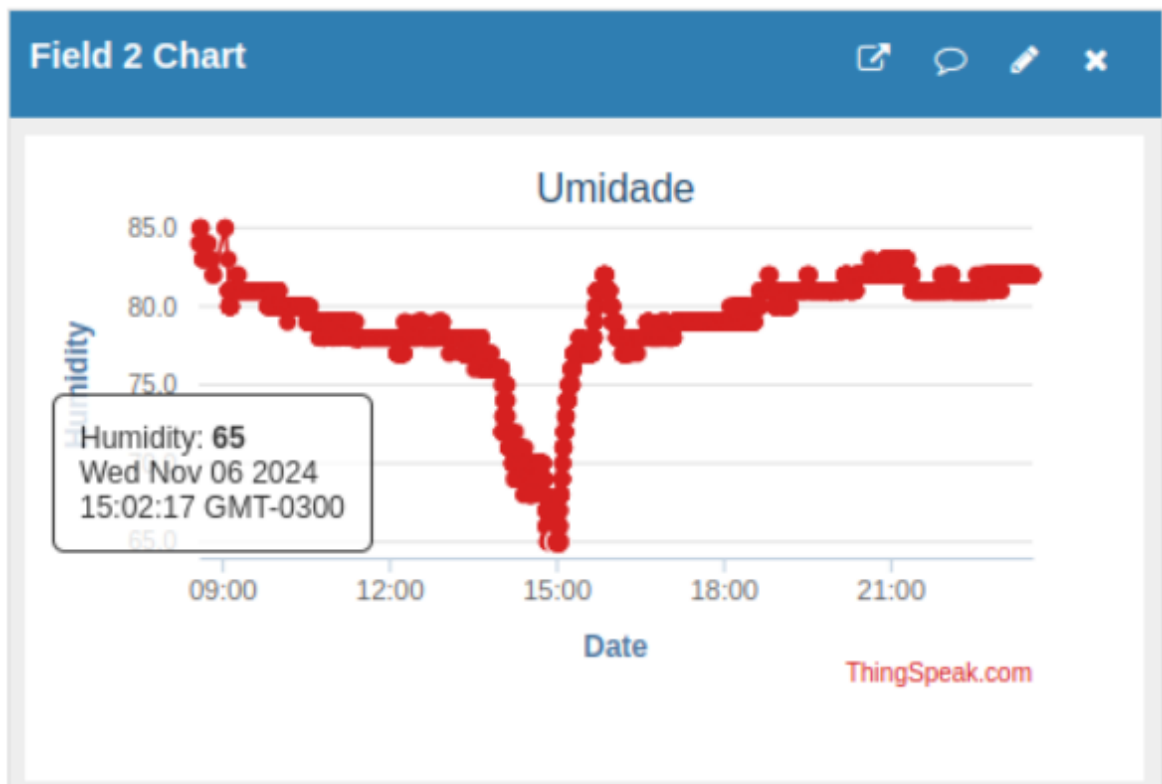


Figura 26 - Monitoramento serial

Com relação à análise dos tempos médios de resposta do atuador e dos sensores, foram realizadas as tabelas 1 e 2.

O atuador, no caso, é um LED que acende. O tempo médio a ser calculado depende principalmente do tempo de loop do código, somado ao tempo médio de resposta do LED (DATASHEET LED), que pode variar entre 2 a 6 ms. Para esse cálculo, foi adicionado um timer no código que registra o tempo entre o momento em que os sensores de umidade e temperatura leem valores acima do limiar (threshold) e o momento em que o comando para acender o LED é enviado. Esse tempo é somado com 4 ms, que é o tempo médio de resposta de um LED (DATASHEET LED).

Número de medidas	Atuador	Tempo de resposta
1	Led	11ms
2	Led	9ms
3	Led	5ms
4	Led	7ms

Tabela -1

Quanto ao tempo médio de leitura do sensor DHT11 até o envio para a plataforma ThingSpeak, o tempo médio de resposta é dado pela soma do intervalo de 15 segundos, que é o tempo mínimo entre as requisições enviadas ao ThingSpeak, juntamente com o tempo necessário para o código realizar um loop de leitura do sensor e receber o status **HTTP 200**, indicando que os dados foram recebidos corretamente na plataforma.

Para esse cálculo, foi adicionado um timer no código que registra o tempo entre o momento em que os sensores de umidade e temperatura leem os dados e o momento em que o código recebe o status da requisição HTTP, confirmando que os dados foram enviados com sucesso.

Número de medidas	Sensor	Tempo de resposta
1	DHT11	1513ms
2	DHT11	1514ms
3	DHT11	1510ms
4	DHT11	1516ms

Tabela -2

**Fonte:** <https://cromatek.com.br/datasheet/optoeletronicos/led-ptb-5mm-vermelha-encapsulamento-vermelho-difuso-L621.pdf>

**Vídeo-demonstração em funcionamento:**

<https://www.youtube.com/watch?v=OdjUsgtmVGg>

**Repositório do Github:**

<https://github.com/sarakiyan/Objetos-inteligentes-conectados>

#### 4. Conclusões

O objetivo proposto de desenvolver uma solução inovadora para o monitoramento de torres de telecomunicações, utilizando a Internet das Coisas (IoT), a plataforma Arduino e uma solução escalável e de baixo custo foi atingido. Durante o desenvolvimento deste projeto, foi possível, com apenas uma ESP32, um sensor DHT11 e um LED, monitorar em tempo real a temperatura e umidade de um ambiente fechado e enviar os dados para a plataforma ThingSpeak. Isso torna o projeto escalável, permitindo a adição de mais sensores de monitoramento de baixo custo, caso necessário.

O principal desafio enfrentado durante o desenvolvimento deste projeto foi a preparação do ambiente de desenvolvimento com a IDE Arduino para a ESP32. Para realizar essa integração, foi necessário, além de instalar a IDE do Arduino, instalar drivers específicos para a comunicação serial via USB do computador. Esses drivers podem variar de

computador para computador, o que pode dificultar a integração inicial do projeto, pois depende do sistema do usuário. Além disso, foi preciso instalar bibliotecas específicas da ESP32 para a IDE, o que pode complicar a configuração inicial.

A principal vantagem desse projeto foi a possibilidade de desenvolver uma solução de IoT que simula uma torre de telecomunicações de baixo custo e alta eficiência. Como o projeto utiliza apenas uma ESP32 com sensores integrados, o custo é muito baixo. Contudo, a principal desvantagem é que a rede Wi-Fi da ESP32 tem um alcance limitado. Para um funcionamento ideal, a ESP32 deve estar localizada próxima ao roteador Wi-Fi.

Como melhorias futuras, podemos adicionar mais sensores ao projeto para monitorar o ambiente de forma mais completa. Por exemplo, poderíamos adicionar um sensor de luminosidade LDR para verificar a relação entre a luminosidade do ambiente e a temperatura/umidade. Também poderíamos incluir sensores de concentração de gases, como CO<sub>2</sub> ou O<sub>2</sub>, para monitorar como esses gases variam durante o dia, entre outros sensores, dependendo do objetivo de cada projeto.

## 5. Referências

VIANA, Carol. Como programar a placa ESP32 no Arduino IDE. Blog da Robótica, 2022. Disponível em: <https://www.blogdarobotica.com/2021/08/24/como-programar-a-placa-esp32-no-arduino-ide/>. Acesso em: 23/09/2024.

SANTOS, Sara. ESP32 with DHT11/DHT22 Temperature and Humidity Sensor using Arduino IDE. . *Random Nerd Tutorials*, 2020. Disponível em: <https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/>. Acesso em: 23/09/2024.

AOSONG. Temperature and humidity module – DHT11 Product Manual. China, [s.d.]. Disponível em: [https://d229kd5ey79jzj.cloudfront.net/1255/Temperature.Sensor\\_DHT11.pdf](https://d229kd5ey79jzj.cloudfront.net/1255/Temperature.Sensor_DHT11.pdf)>. Acesso em: 23/09/2024.

DIGITAL, Ac. IoT Based Telecom Tower Monitoring System Can Increase Operational Efficiency, 2018. Disponível em: <https://www.acldigital.com/blogs/iot-based-telecom-tower-monitoring-system-can-increase-operational-efficiency> . Acesso em: 04/09/2024.

G, TZAFESTAS; Spyros. Synergy of IoT and AI in Modern Society: The Robotics and Automation Case, 2018. Disponível em: <https://juniperpublishers.com/raej/RAEJ.MS.ID.555621.php>. Acesso em: 04/09/2024.

GUNDRY, Tyeth. Arduino library for DHT11, DHT22, etc Temperature & Humidity Sensors. GitHub, 2023. Disponível em : <https://github.com/adafruit/DHT-sensor-library>. Acesso em 20/10/2024.

GALVÃO, Michel. ESP-NOW: Comunicação Sem-Fio Entre ESP32s, 2023. Disponível em: <https://blog.eletrogate.com/esp-now-comunicacao-sem-fio-entre-esp32s/> . Acesso em 20/10/2024.

ALMY, Tom. One Wire and the DHT11/DHT22 sensor, 2021. Disponível em: <https://tomalmy.com/one-wire-and-the-dht11-dht22-sensor/> . Acesso em 20/10/2024.

VAZ, Saavedra Lucas. ARDUINO-ESP32, 2019. Disponível em: <https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFi.h> . Acesso em 10/11/2024.

KELLY, John. THINGSPEAK-ARDUINO, 2023. Disponível em: <https://github.com/mathworks/thingspeak-arduino> . Acesso em 10/11/2024.

THINGSPEAK. MATHWORKS. Disponível em: <https://thingspeak.mathworks.com/> .

FRITZING. Welcome to Fritzing. Disponível em: <https://fritzing.org/> .