RELATÓRIO TÉCNICO: SISTEMA DE TELEMETRIA E MONITORAMENTO EM TEMPO REAL PARA ENSAIOS DE MOTORES ELÉTRICOS

MATEUS DOS SANTOS CAVALHEIRO
MA77

1. INTRODUÇÃO

Este relatório apresenta o desenvolvimento de um cibersistema para monitoramento de um motor no setor industrial. O objetivo é integrar hardware (ESP32) e software (Java) para garantir a segurança operacional e coleta de dados para análise gerencial. O sistema coleta dados de temperatura, vibração e pressão onde utiliza uma lógica de tomada de decisão remota via protocolo MQTT para acionar alertas visuais e mensagens de status em tempo real.

2. ARQUITETURA DE DADOS E PROTOCOLO

Para a integração, foi adotado o paradigma de Programação Orientada a Objetos no backend (Java) e Programação Estruturada no firmware (C++).

- Protocolo: MQTT (Message Queuing Telemetry Transport).
  É um protocolo de comunicação leve e eficiente, padrão em IoT, que permite a troca rápida de mensagens entre dispositivos (como o ESP32 e o computador) com baixo consumo de internet.
- Broker: broker.hivemq.com (Porta 1883).
  É o servidor intermediário público na nuvem que é responsável por receber os dados enviados pelo seu circuito e entregá-los ao sistema Java através da porta 1883.

3. PROJETO HARDWARE

O circuito foi montado no simulador Wokwi utilizando:

- ESP32: Microcontrolador com Wi-Fi nativo.
- DHT22: Sensor de temperatura.
- Potenciômetro (2x): Simulação de coleta de corrente e vibração (0-50A) & 0-100%).
- Display LCD 16x2 I2C: Exibição de mensagens locais.
- LEDs: Verde (GPIO 12), Amarelo (GPIO 14), Vermelho (GPIO 27).
- Resistores (3x): Conectados aos LED’s para proteção dos mesmos. (220Ω)

4. ESQUEMA DE LIGAÇÃO ELÉTRICO

O hardware foi projetado para garantir a integridade dos sinais analógicos e digitais. Abaixo, a pinagem detalhada:

Componente	Pino no ESP 32	Tipo de Sinal	Função
DHT22 (VCC)	3V3	ALIMENTAÇÃO	Energizar Sensor
DHT22 (GND	GND	ALIMENTAÇÃO	Negativo do Sensor
DHT22 (SDA)	D15	DIGITAL (I/O)	Envio de dados de Temperatura
POTENCIÔMETRO - VIBRAÇÃO (VCC)	3V3	ALIMENTAÇÃO	Energizar Pot1
POTENCIÔMETRO - VIBRAÇÃO (GND)	GND	ALIMENTAÇÃO	Negativo do Pot1
POTENCIÔMETRO - VIBRAÇÃO (SIG)	D34	ANALÓGIO (ADC)	Simulação dos dados de Vibração
POTENCIÔMETRO - CORRENTE (VCC)	3V3	ALIMENTAÇÃO	Energizar Pot2
POTENCIÔMETRO - CORRENTE (GND)	GND	ALIMENTAÇÃO	Negativo do Pot2
POTENCIÔMETRO - CORRENTE (SIG)	D35	ANALÓGICO (ADC)	Simulação dos dados de Corrente
LCD I2C (VCC)	3V3	ALIMENTAÇÃO	Energizar o LCD
LCD I2C (GND)	3V3	ALIMENTAÇÃO	Negativo do LCD
LCD I2C (SDA)	D21	COMUNICAÇÃO	Barramento de dados I2C
LCD I2C (SCL)	D22	COMUNICAÇÃO	Barramento de clock I2C
LED Verde	D12	DIGITAL OUTPUT	Status: Vibração Segura
LED Amarelo	D14	DIGITAL OUTPUT	Status: Vibração Moderada
LED Vermelho	D27	DIGITAL OUTPUT	Status: Vibração Crítica

Nota Técnica[1]**:** Todos os LEDs devem utilizar resistores de 220Ω em série para proteção da porta GPIO.

[1] Retirado do anexo enviado pelo docente.

No simulador Wokwi, os componentes utilizam nomenclaturas técnicas abreviadas para facilitar a identificação dos pinos.

Para o LED, as letras significam:

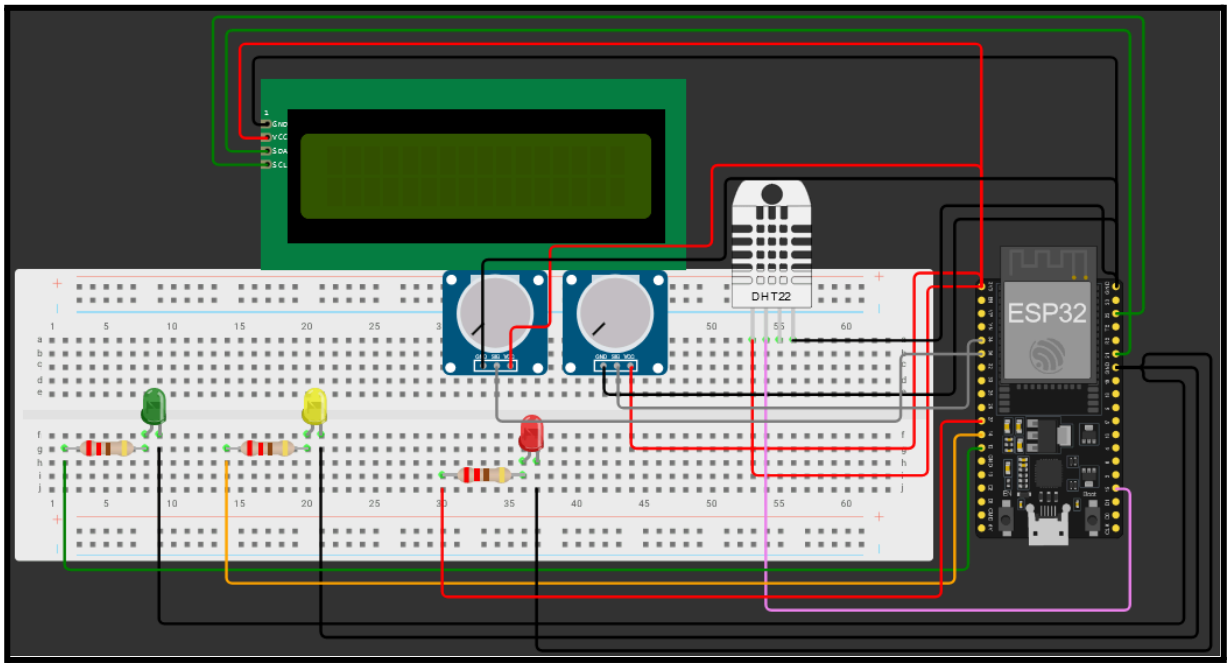
● A = Anodo: É a perna Positiva (+). Deve ser conectada ao pino digital do ESP32 (passando pelo resistor).

● C = Catodo: É a perna Negativa (-). Deve ser conectada diretamente ao GND (Terra).

No projeto em questão, a ligação correta no Wokwi será:

1. Pino do ESP32 (ex: D12) -> Resistor -> Terminal A do LED.
2. Terminal C do LED -> Barramento de GND da Breadboard.

Figura 1 - Esquema Elétrico (Entre ESP32, DHT22, LCD e Potenciômetros)



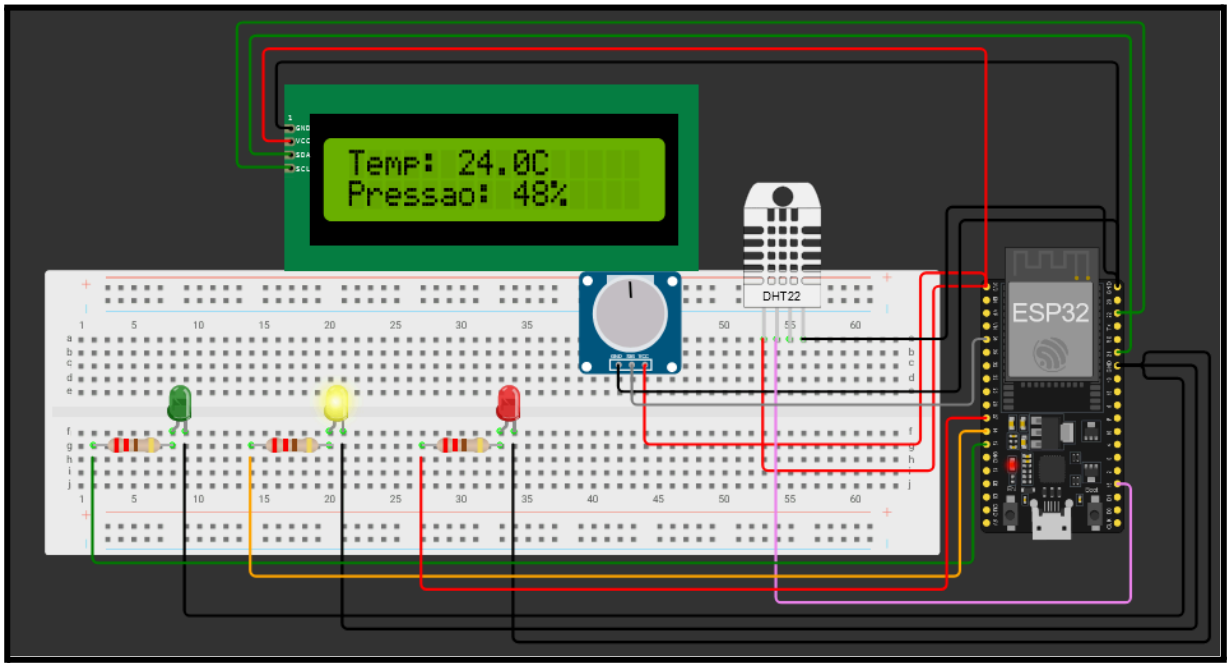
Fonte: Elaborado pelo autor com o simulador Wokwi, 2026.

5. DESENVOLVIMENTO DE SOFTWARE

5.1 Teste de ligação elétrica

A fim de validar o esquema de ligações elétricas, foi montado um esquema parecido com o que será utilizado no projeto de monitoramento, porém com um potenciômetro a menos. O objetivo é garantir a ativação dos LED's seguindo a lógica proposta (Aqui utilizando o parâmetro de pressão analógica) além da exibição dos dados ao LCD. Segue abaixo o esquema de ligação com o código utilizado.

Figura 2 - Esquema de ligação para validação de lógica e captura de dados



Fonte: Elaborado pelo autor com o simulador Wokwi, 2026.

Código de teste (Em C++) de ligação elétrica

```
#include <Arduino.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"

// Pinos conforme seu hardware
#define POT_PIN 34 // Potenciômetro (Analog) -> Pressão
#define DHT_PIN 15 // Sensor DHT22 (Digital) -> Temperatura
#define LED_VERDE 12
#define LED_AMARELO 14
#define LED_VERMELHO 27

// SDA = 21, SCL = 22 (Padrão ESP32 para o LCD I2C)

DHT dht(DHT_PIN, DHT22);
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Wire.begin(21,22);

  Serial.begin(115200);
  pinMode(LED_VERDE, OUTPUT);
  pinMode(LED_AMARELO, OUTPUT);
  pinMode(LED_VERMELHO, OUTPUT);

  lcd.init();
  lcd.backlight();
  dht.begin();

  lcd.setCursor(0,0);
  lcd.print("SISTEMA-LEITURA");
```

```
  delay(2000);
  lcd.clear();
}

void loop() {
  // 1. LEITURA DA TEMPERATURA (VEM DO DHT22)
  float Temp = dht.readTemperature();

  // 2. LEITURA DA PRESSÃO (VEM DO POTENCIÔMETRO)
  int leituraPot = analogRead(POT_PIN);
  // Mapeia 0-4095 para 0-100% de Pressão
  float Pressao = map(leituraPot, 0, 4095, 0, 100);

  // Exibição no LCD (Lembre-se: Linha 0 e 1 apenas!)
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  if (isnan(Temp)) {
      lcd.print("Erro  "); // Caso o sensor DHT falhe
  } else {
      lcd.print(Temp, 1);
      lcd.print("C   ");
  }

  lcd.setCursor(0, 1);
  lcd.print("Pressao: ");
  lcd.print((int)Pressao); // Mostra como número inteiro
  lcd.print("%    ");

  // Lógica dos LEDs baseada APENAS na Pressão (Potenciômetro)
  if (Pressao < 35) {
      digitalWrite(LED_VERDE, HIGH);
      digitalWrite(LED_AMARELO, LOW);
      digitalWrite(LED_VERMELHO, LOW);
  }
  else if (Pressao >= 35 && Pressao <= 75) {
      digitalWrite(LED_VERDE, LOW);
      digitalWrite(LED_AMARELO, HIGH);
      digitalWrite(LED_VERMELHO, LOW);
  }
  else {
      digitalWrite(LED_VERDE, LOW);
      digitalWrite(LED_AMARELO, LOW);
      digitalWrite(LED_VERMELHO, HIGH);
  }

  // Debug no Terminal Serial
  Serial.print("Sensor DHT (Temp): "); Serial.print(Temp);
  Serial.print("C | Potenciometro (Pressao): "); Serial.print(Pressao);
  Serial.println("%");

  delay(500);
}
```

5.2 Elaboração do código do sistema para entrega

5.2.1 Elaboração do código em C++ (Para o ESP32, utilizando Wokwi)

O código a seguir realiza a função de coletar os dados dos sensores e potencializadores, exibi-los na tela do LCD, e enviar via MQTT os dados para exibição via Java (Maven no Intellij). Além disso, foi implementado uma lógica para ativação dos LED's dependente da vibração do motor.

Código em C++

```
//Incluir bibliotecas a serem utilizadas
#include <WiFi.h> //Gerencia a conexão do dispositivo com redes Wi-Fi
#include <PubSubClient.h> //Permite a comunicação via protocolo MQTT
(publicar e receber mensagens)
#include <DHT.h> //Biblioteca para leitura de temperatura e umidade do
sensor DHT22
#include <LiquidCrystal_I2C.h> //Controla o display LCD

//1. CONFIGURAÇÕES (WIFI e MQTT)
const char* ssid = "Wokwi-GUEST";
const char* password = "";
const char* mqtt_server = "broker.hivemq.com";

const char* topic_publish = "senai/MATEUS/motor/dados";

//2. HARDWARE
#define PIN_DHT 15        // Temperatura
#define PIN_VIB 34        // Vibração (Potenciômetro 1)
#define PIN_COR 35        // Corrente (Potenciômetro 2)

// LEDs
#define LED_VERDE 12     // GPIO 12
#define LED_AMARELO 14   // GPIO 14
#define LED_VERMELHO 27  // GPIO 27

//3. ATIVAÇÃO DE OBJETOS
WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(PIN_DHT, DHT22);
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  Serial.begin(115200);

  // Configura Pinos
  pinMode(LED_VERDE, OUTPUT);
  pinMode(LED_AMARELO, OUTPUT);
  pinMode(LED_VERMELHO, OUTPUT);

  // Inicia Sensores
  dht.begin();
  lcd.init();
  lcd.backlight();

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("SENAI - CIBER"); // Opcional
```

```
  lcd.setCursor(0, 1);
  lcd.print("PROVA - PRATICA");
  delay(3000); // Fica na tela por 3 segundos
  lcd.clear();

  // Conexão Wi-Fi
  WiFi.begin(ssid, password);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("CONECTANDO WIFI..");
  delay(3000); // Fica na tela por 3 segundos
  lcd.clear();
  Serial.print("Conectando Wi-Fi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("CONECTADO!");
  digitalWrite(LED_AMARELO, HIGH);
  digitalWrite(LED_VERDE, HIGH);
  digitalWrite(LED_VERMELHO, HIGH);
  delay(3000); // Fica na tela por 3 segundos
  lcd.clear();
  Serial.println(" Conectado!");

  client.setServer(mqtt_server, 1883);
}

void reconnect() {
  while (!client.connected()) {
    Serial.print("Conectando MQTT...");
    String clientId = "ESP32_Motor_" + String(random(0xffff), HEX);
    if (client.connect(clientId.c_str())) {
      Serial.println("OK!");
    } else {
      Serial.print("Falha: ");
      Serial.print(client.state());
      delay(5000);
    }
  }
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  //4. LEITURA E MAPEAMENTO

  //Temperatura
  float temp = dht.readTemperature();
  if (isnan(temp)) temp = 0.0;

  //Vibração (0-4095 -> 0-100%)
  int leituraVib = analogRead(PIN_VIB);
  int vibracao = map(leituraVib, 0, 4095, 0, 100);
```

```
  //Corrente (0-4095 -> 0-100 A)
  int leituraCor = analogRead(PIN_COR);
  int corrente = map(leituraCor, 0, 4095, 0, 100);

  //5. LÓGICA DE LEDS (Baseada na VIBRAÇÃO)
  // Reseta
  digitalWrite(LED_VERDE, LOW);
  digitalWrite(LED_AMARELO, LOW);
  digitalWrite(LED_VERMELHO, LOW);

  if (vibracao < 45) {
    digitalWrite(LED_VERDE, HIGH); // Normal
  } else if (vibracao >= 45 && vibracao < 70) {
    digitalWrite(LED_AMARELO, HIGH); // Alerta
  } else {
    digitalWrite(LED_VERMELHO, HIGH); // Perigo (>70)
  }

  //6. LCD
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("T:"); lcd.print(temp, 1); lcd.print(" V:");
lcd.print(vibracao); lcd.print("%");

  lcd.setCursor(0,1);
  lcd.print("C:"); lcd.print(corrente); lcd.print("A");

  //7. ENVIO FORMATO STRING
  String payload = String(temp, 1) + "," + String(vibracao) + "," +
String(corrente);

  Serial.print("Enviando: ");
  Serial.println(payload);

  client.publish(topic_publish, payload.c_str());

  delay(2000); // Envia a cada 2 segundos
}
```

Para a realização do código acima, foi utilizado auxílio de inteligência artificial para criação do mesmo, através da junção do mesmo código (antes utilizado para teste de ligação elétrica e envio de informações via MQTT) com um prompt[2] inserido ao Gemini.

[2] Baseado no código fornecido e aos anexos inseridos, modifique o código para a inclusão de mensagens baseadas no status de conexão com MQTT (Broker) para que apareçam ao LCD, além de incluir o envio de dados (payload) sem utilização de JSON, seguindo o formato String fornecido pelo docente.

5.2.2 Elaboração do código em Java (Utilizando o Maven no IDEA IntelliJ)

Para o sucesso do projeto, devemos garantir que após o envio dos dados via MQTT pelo ESP32, esses mesmos sejam inseridos a um terminal no Java, que tem a função única de visualização dos dados em tempo real, (ou com mínima latência).

5.2.2.1 Estrutura de pastas no IntelliJ (Arquitetura Técnica)

```
scr/
|----- main/
   |-------- java/
          |----- br/com/senai/automacao/
                |------------------------------ MonitoramentoMotor.java
                                                  (Classe principal/execução para visualização dos dados
   |------- resources/                           (Arquivos de configuração)
pom.xml                                           (Cérebro do Maven - Dependências)
```

5.2.2.2 Inserir as dependências ao arquivo pom.xml

Para a comunicação do Maven (Java) com o MQTT, devemos adicionar um “dicionário” para ambos se entenderem ao trocar informações.

Adicionando as dependências ao pom.xml

```
<dependencies>
      <dependency>
            <groupId>org.eclipse.paho</groupId>
            <artifactId>org.eclipse.paho.client.mqttv3</artifactId>
            <version>1.2.5</version>
      </dependency>
      </dependencies>
```

5.2.2.3 Código do Java

Com as dependências inseridas, e o Java já podendo entender alguns comandos do MQTT, segue o código aproveitado de outras atividades para o mesmo propósito de receber dados enviados por MQTT pelo ESP32. Apenas modificado para receber uma quantidade maior (e diferente) de dados.

Código em Java com comentários

```
package br.com.senai.automacao;

//Importação das bibliotecas
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
```

```
import java.util.UUID;

@SuppressWarnings("all")

public class MonitoramentoMotor {

	public static void main(String[] args) {
	// CONFIGURAÇÕES
	String broker = "tcp://broker.hivemq.com:1883";
	String topico = "senai/MATEUS/motor/dados"; // O MESMO DO C++
	String clientId = "Java_Monitor_" + UUID.randomUUID().toString();

	try {
		// 1. Criar o cliente MQTT
		MqttClient client = new MqttClient(broker, clientId);
		MqttConnectOptions options = new MqttConnectOptions();
		options.setCleanSession(true);

		// 2. Definir o que acontece quando chega uma mensagem
(Callback)
		client.setCallback(new MqttCallback() {

		@Override
		public void connectionLost(Throwable cause) {
			System.out.println("Conexão perdida! Causa: " +
cause.getMessage());
		}

		@Override
		public void messageArrived(String topic, MqttMessage message)
throws Exception {

			String payload = new String(message.getPayload());

			// O C++ envia: "25.0,12,30" (Exemplo)
			// Vamos separar pela vírgula
			String[] dados = payload.split(",");

			if (dados.length >= 3) {

System.out.println("-------------------------------------------------");
				System.out.println("Dados de Telemetria Coletados
com Sucesso:");
				System.out.println("Temperatura: " + dados[0] + "
°C");
				System.out.println("Vibração:  " + dados[1] + "
%");
				System.out.println("Corrente:  " + dados[2] + "
A");

				// Validação Extra para o Relatório (Segurança)
				double vib = Double.parseDouble(dados[1]);
				if (vib > 70) {
				System.err.println("[ALERTA CRÍTICO] PERIGO DE
VIBRAÇÃO EXCESSIVA!");
				}
			}
		}

		@Override
```

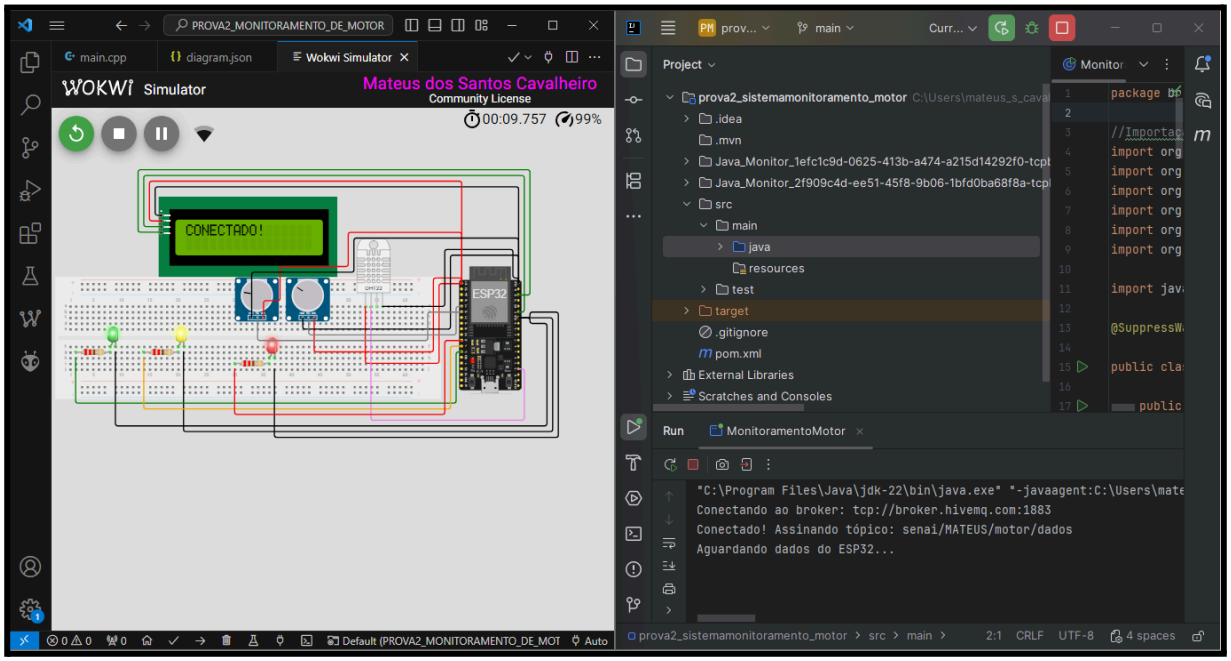
```
            public void deliveryComplete(IMqttDeliveryToken token) {
                  // Não usado pois só estamos recebendo
            }
            });

            // 3. Conectar e Assinar
            System.out.println("Conectando ao broker: " + broker);
            client.connect(options);
            System.out.println("Conectado! Assinando tópico: " + topico);
            client.subscribe(topico);

            System.out.println("Aguardando dados do ESP32...");

      } catch (MqttException e) {
            e.printStackTrace();
      }
      }
}
```

6. FICHA DE REGISTRO DE TESTES

ID	TESTE	PROCEDIMENTO	RESULTADO ESPERADO	STATUS
01	CONECTIVIDADE	INICIAR ESP32 E JAVA	Ambos conectados ao Broker HiveMQ.	P
02	TELEMETRIA	GIRAR POTENCIÔMETROS NO WOKWI	Valores de corrente e vibração atualizam no console Java.	P
03	LÓGICA CRÍTICA	FORÇAR Temp > 60°C NO DHT22	LED Vermelho acende e terminal exibe em vermelho “[ALERTA CRÍTICO] PERIGO DE VIBRAÇÃO EXCESSIVA!”	P
04	SEGURANÇA	DESCONECTAR Wifi DO ESP32	Sistema para de atuar; Java loga timeout.	P

6.1 Teste de Conectividade

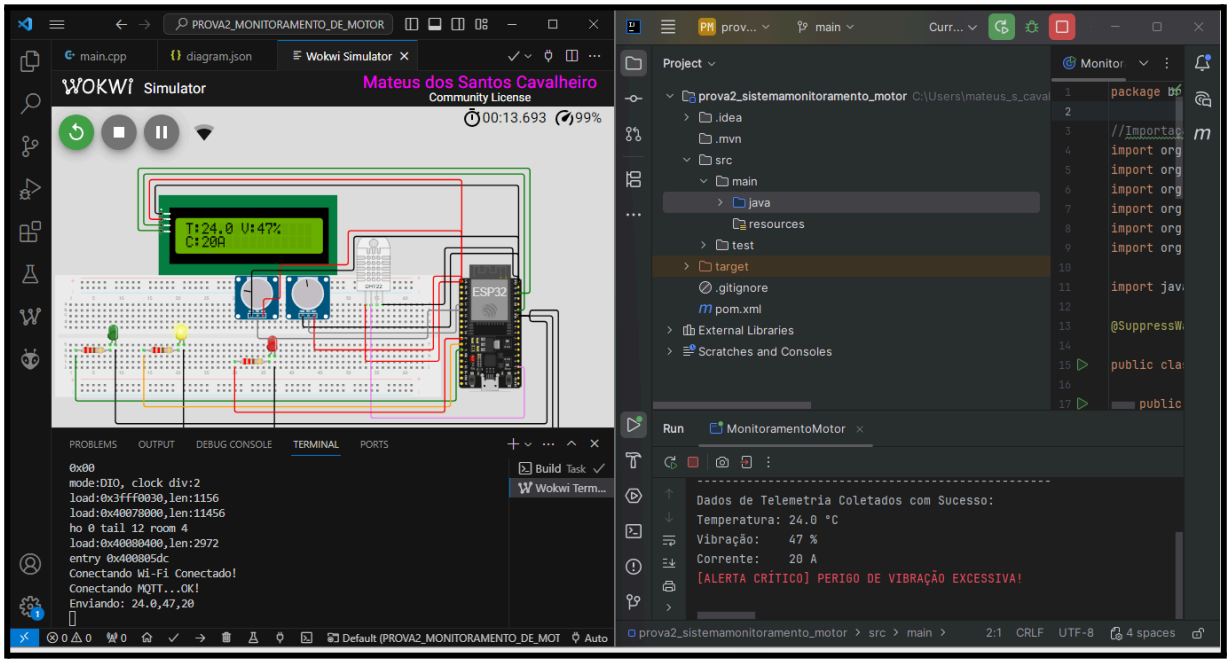
Figura 3 - Teste de Conectividade



Fonte: O autor, 2026.

6.2 Teste de Telemetria

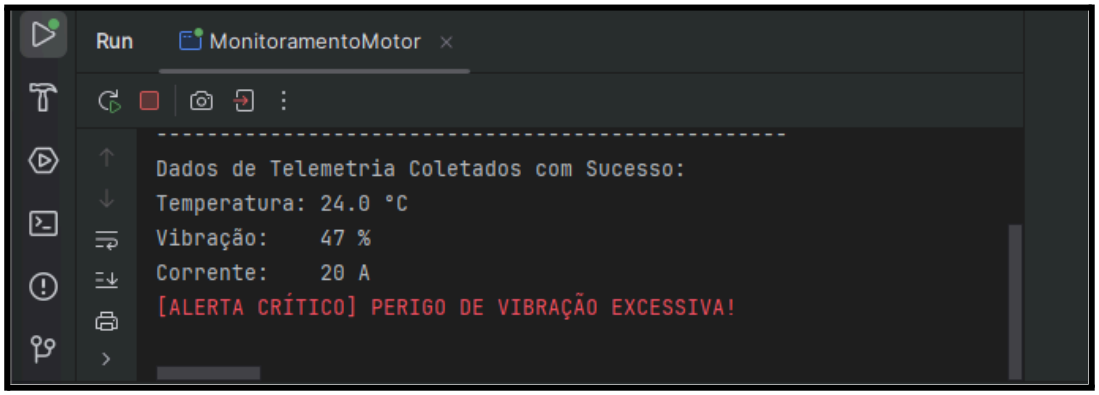
Figura 4 - Teste de Telemetria



Fonte: O autor, 2026.

6.3 Teste de Lógica Crítica

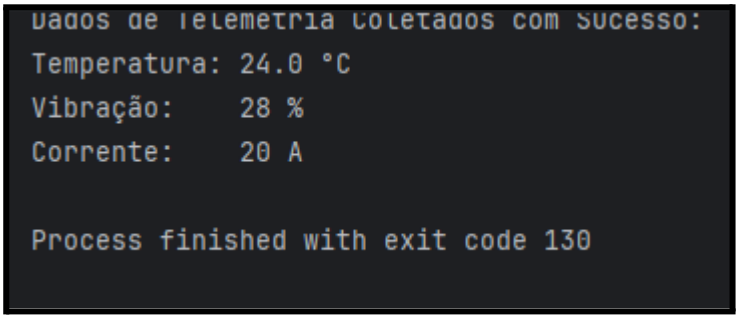
Figura 5 - Teste de Lógica Crítica



Fonte: O autor, 2026.

6.4 Teste de Segurança

Figura 6 - Teste de Segurança



Fonte: O autor, 2026.

7. PLANEJAMENTO E ADAPTAÇÃO

7.1 Cronograma de Tempo

Etapa	Tempo Estimado	Status
Montagem do Hardware no Wokwi	40 min	REALIZADO NO PRAZO
Programação do Firmware (C++)	60 min	REALIZADO FORA DO PRAZO (10Min fora)
Configuração do Backend (Java)	40 min	REALIZADO NO PRAZO
Testes de Comunicação e Ajustes	40 min	REALIZADO NO PRAZO
Redação do Relatório Técnico	60 min	REALIZADO FORA DO PRAZO (2H fora)

7.2 Adaptação

Seguindo de uma atividade parecida com essa, foi utilizado formato JSON para envio e exibição de dados para o IntelliJ, porém, ao perceber uma certa complexidade por parte do entendimento dos alunos, foi requisitado o envio de dados via formato String, com o docente alocando um código de exemplo a ser seguido para a exibição de informações. Outro ponto muito importante a se destacar foi a migração das simulações via site Wokwi, para sua extensão no compilador VSCode. Isso se deve principalmente pelo fato de que as simulações via Wokwi funcionam por meio de servidores onde muitas vezes estao lotados e não chegam a carregar a simulação. A solução foi instalar algumas extensões do Wokwi no VSCode além do PlaformIO, para simulação direta sem a espera longeva.

7.3 Configurando o PlatformIO

7.3.1 Instalação e Licença

Primeiro, instale a extensão "Wokwi for VS Code" através do marketplace do editor. Após instalar, abra a paleta de comandos (F1 ou Ctrl+Shift+P) e digite "Wokwi: Request a new License". Siga as instruções no navegador para obter sua licença gratuita (para uso pessoal/open source) e ativá-la no VS Code.

7.3.2 Arquivo de Configuração (wokwi.toml)

Na raiz do seu projeto PlatformIO, crie um arquivo chamado wokwi.toml. Este arquivo diz ao simulador onde encontrar o firmware compilado. O conteúdo deve ser algo como o exemplo abaixo, onde você deve ajustar o caminho do firmware e elf para corresponder à pasta de build do seu ambiente no PlatformIO (geralmente dentro de .pio/build/NOME_DO_AMBIENTE/):

Código com “endereço” Build feita pelo PlatformIO com Wokwi

```
[wokwi]
version = 1
firmware = '.pio/build/uno/firmware.hex'
elf = '.pio/build/uno/firmware.elf'
```

7.3.3 Importação do diagrama elétrico

Crie um arquivo chamado diagram.json na raiz do projeto. Este arquivo define os componentes eletrônicos e suas conexões. A maneira mais fácil de gerar isso é criar seu circuito no site wokwi.com, copiar o conteúdo da aba "diagram.json" do editor online e colar neste arquivo local.

7.3.4 Importação do código em C++

Após a importação do diagrama elétrico, acesse a pasta scr/ e em seguida o arquivo main.cpp, e cole a programação feita em C++ no wokwi.com. Essa programação pode ser adquirida na aba sketch.ino.

7.3.5 Instalar as bibliotecas utilizadas no sistema

Para que todo o sistema funcione, é necessário inserir as mesmas bibliotecas usadas no sistema dentro do PlatformIO, para a mesma traduzir em uma linguagem que o VSCode entenda o sistema como um todo. Assim como o Wokwi, não se deve apenas usar os comandos #include parar importar as bibliotecas mas assim fazer o download das mesmas na aba Library Manager.

Código para implementar as bibliotecas na PlatformIO.ini

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
monitor_speed = 115200

lib_deps =
      knolleary/PubSubClient @ ^2.8
      adafruit/DHT sensor library @ ^1.4.4
      adafruit/Adafruit Unified Sensor @ ^1.1.9
      marcoschwartz/LiquidCrystal_I2C @ ^1.1.4
```

7.3.6 Executar a simulação

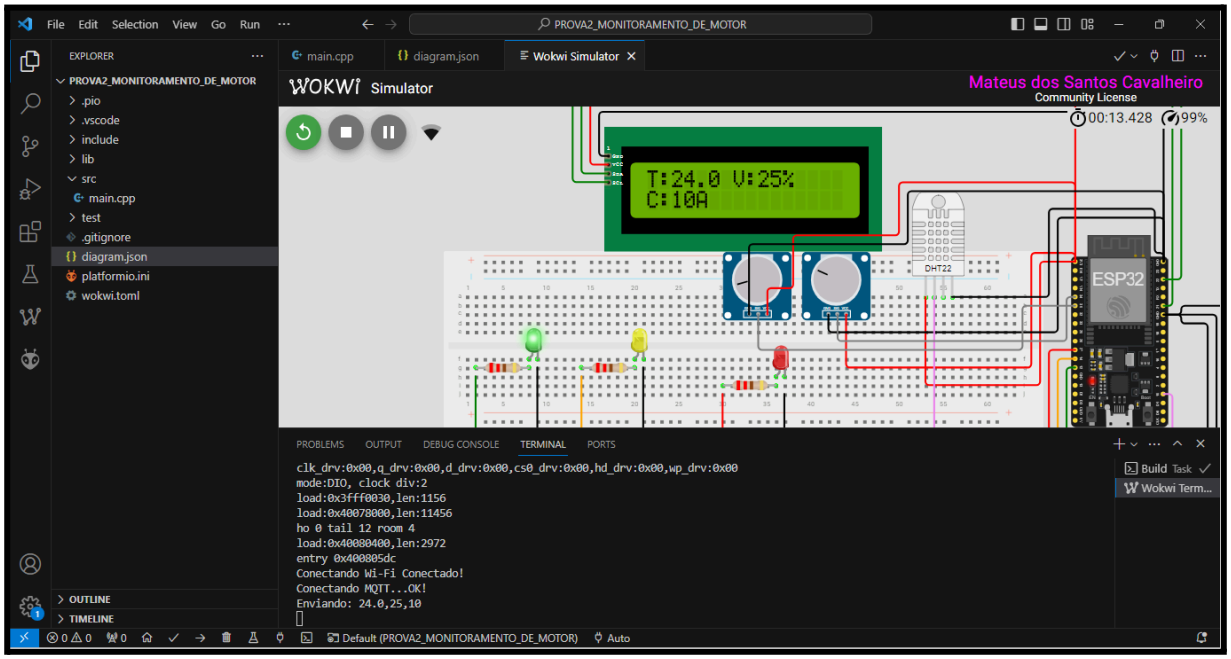
Antes de simular, compile seu projeto usando o botão "Build" do PlatformIO (o ícone de ✔ na barra inferior). Com o firmware compilado, abra o arquivo diagram.json, pressione F1 e selecione "Wokwi: Start Simulator". O circuito aparecerá e o código será executado virtualmente dentro do VS Code.

7.3.7 Atualizações de código e diagrama

Se por acaso for preciso atualizar o código ou o diagrama, deve-se sempre lembrar de compilar o programa (clicando no ícone de “Build” na barra inferior). Um adendo, o código ou diagrama atualizado no VSCode não é interligado com o site da wokwi.com.

Para mais informações, acesse: https://docs.wokwi.com/vscode/getting-started.

Figura 7 - Sistema de Pastas Interna dentro de um projeto PlatformIO



Fonte: O autor, 2026.

8. Conclusão

Ao realizar o trabalho, foi posto em prova que o sistema feito em dois ambientes diferentes conseguiram se comunicar de forma espetacular via protocolo MQTT, utilizando o Broker do HIVE para o envio e recebimento de dados enviados pelo ESP32. É garantido que a replicação dos códigos e diagramas elétricos funcionem em diferentes dispositivos e ambientes que tenham acesso a conexão com a Internet. O uso do protocolo MQTT mostrou-se vantajoso devido ao seu baixo consumo de banda, o que é crucial para projetos de Internet das Coisas (IoT). Dessa forma, o projeto serve como uma base sólida para aplicações futuras, como automação residencial ou monitoramento industrial remoto, garantindo portabilidade para qualquer cenário com acesso à rede mundial de computadores.

9. Referências Bibliográficas[3]

WOKWI. Getting Started with Wokwi for VS Code. [S.l.]: CodeMagic LTD, 2026. Disponível em: https://docs.wokwi.com/vscode/getting-started. Acesso em: 24 fev. 2026.

PLATFORMIO. PlatformIO: Your Gateway to Embedded Software Development Excellence. [S.l.]: PlatformIO, [202-?]. Disponível em: https://platformio.org/. Acesso em: 24 fev. 2026.

[3] Referências feitas a partir de inteligência artificial, seguindo o seguinte prompt: “De acordo com os anexos fornecidos ao início do trabalho, coloque em formato ABNT as referências desses arquivos e das bibliotecas utilizadas no código em C++. Adicione também a página da extensão Wokwi for VSCode e PlatformIO”.

O'LEARY, Nick. PubSubClient: A client library for the Arduino Ethernet Shield that provides support for MQTT. Versão 2.8. [S.l.]: GitHub, 2020. Disponível em: https://github.com/knolleary/pubsubclient. Acesso em: 24 fev. 2026.

ADAFRUIT. DHT-sensor-library: Arduino library for DHT11, DHT22, etc Temp & Humidity Sensors. Versão 1.4.6. [S.l.]: GitHub, 2023. Disponível em: https://github.com/adafruit/DHT-sensor-library. Acesso em: 24 fev. 2026.

SCHWARTZ, Marco; BRABANDER, Frank de. LiquidCrystal_I2C. Versão 1.1.2. [S.l.]: GitHub, 2016. Disponível em: https://github.com/marcoschwartz/LiquidCrystal_I2C. Acesso em: 24 fev. 2026.

SANTOS, Lucas Sousa dos. Avaliação Prática: Programação para Coleta de Dados em Automação. Santa Catarina: SENAI, 2026. 3 p. Material didático do Curso Técnico em CiberSistemas.

SANTOS, Lucas Sousa dos. Anexo 1: Manual de Referência Técnica. Santa Catarina: SENAI, 2026. 2 p. Material de apoio técnico para montagem e programação.

SANTOS, Lucas Sousa dos. Relatório Técnico: Sistema de monitoramento e controle de prensa hidráulica (Anexo 2). Santa Catarina: SENAI, 2026. 12 p. Exemplo de projeto prático resolvido.

CAVALHEIRO, Mateus. PROVA2_MONITORAMENTO_DE_MOTOR. [S.l.]: GitHub, 2026. Repositório de código-fonte. Disponível em: https://github.com/mateusscavalheiro-afk/PROVA2_MONITORAMENTO_DE_MOTOR. Acesso em: 24 fev. 2026.