SISTEMA DE MONITORAMENTO E CONTROLE DE PRENSA HIDRÁULICA

WEG S.A, Sesi Senai
Claudio Gabriel Litz
Jaraguá do Sul, 2026

SISTEMA DE MONITORAMENTO E CONTROLE DE PRENSA HIDRÁULICA

Relatório Técnico apresentado à Unidade Curricular de Programação para Coleta de Dados em Automação como requisito para avaliação 2.

Docente: Lucas Sousa dos Santos

Jaraguá do Sul - 2026

Sumário

1. Introdução

O relatório a seguir apresenta o desenvolvimento de uma atividade de monitoramento de uma prensa hidráulica industrial, a partir de um cibersistema desenvolvido em aula. Tem como objetivo integrar o micro controlador ESP32 e um software em java rodando num servidor externo que garante um monitoramento adequado para a segurança operacional. O sistema coleta dados de temperatura, pressão e corrente de um prensa hidráulica simulada usando uma lógica de tomada de decisão remota via protocolo MQTT que aciona alertas visuais e mensagens em tempo real.

2. ARQUITETURA DE DADOS E PROTOCOLO

Para uma integração adequada, foi escolhido uma aplicação que usa Programação Orientada a Objeto no backend (java) junto com uma de Programação Estruturada no firmware (C++).

- Protocolo:MQTT (Message Queuing Telemetry Transport).
- Broker: ab7bca8a88fb429ea9c6e193eb502776.s1.eu.hivemq.cloud (Porta: 8883).
- Tópico:
    - senai/claudio/motor/dados: Dados publicados (JSON: {"Temperatura":24.00, "Pressao":8.00, "Corrente":32.00})

3. PROJETO DE HARDWARE

Circuito montado no site wokwi.com utilizando os seguintes itens:

- ESP32: Microcontrolador com WiFi integrado;
- DHT22: Sensor de temperatura e humidade (Apenas usado temperatura);
- Potenciômetro: Simulação de transdutor de pressão (0-100%);
- Potenciômetro: Simulação de sensor de corrente (0-100A);
- Display LCD 16x2 I2C: Exibição de mensagens locais;
- LEDs: Verde (GPIO 4), Amarelo (GPIO 17), Vermelho (GPIO 13) e Verde2 (GPIO 12)

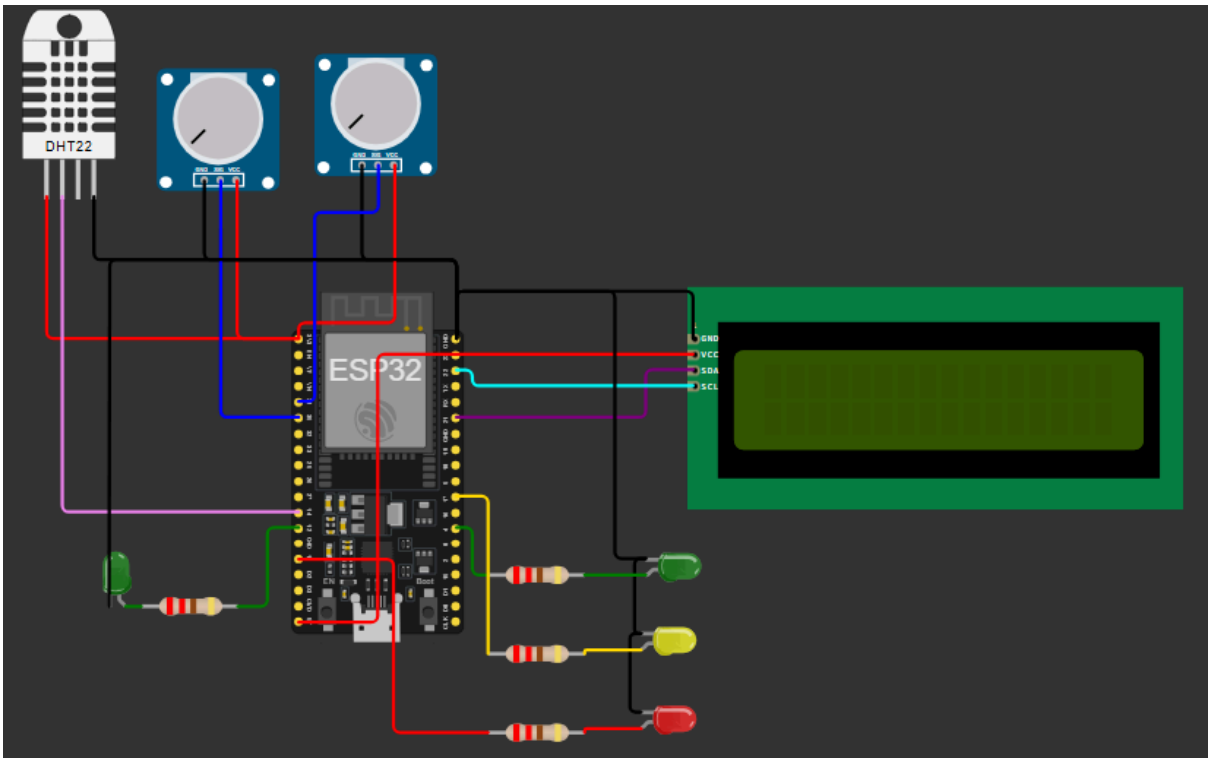
4. ESQUEMA DE LIGAÇÃO ELÉTRICO

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

Componente	Pino no ESP32	Tipo de sinal	Função
DHT22 (VCC)	3v3	Alimentação	Positivo do Sensor
DHT22 (GND)	3v3	Alimentação	Negativo do sensor
DHT22 (Data)	D14	Digital (I/O)	Saída de dados
Potenciômetro	D35	Analógico (ADC)	Simulação pressão
Potenciômetro	D34	Analógico (ADC)	Simulação corrente
LCD I2C (SDA)	D21	Comunicação	Barramento de dados
LCD I2C (SCL)	D22	Comunicação	Barramento de clock
LED Verde 2	D12	Digital (I/O)	Status: Enviando dados
LED verde	D4	Digital (I/O)	Status: Operação Segura
LED amarelo	D17	Digital (I/O)	Status: Alerta de Faixa
LED vermelho	D13	Digital (I/O)	Status: Crítico/Parada

Nota: Todos os LEDs devem estar conectados em série ao um resistor de aproximadamente 220 Ohms para a proteção de alta corrente na porta GPIO.

Figura 1 – Esquema Elétrico



Fonte: O autor, 2026

Para melhor compreensão, a imagem acima é todo o esquema elétrico do protótipo de simulação, feito no site Wokwi.com nos projetos do autor.

5. DESENVOLVIMENTO DE SOFTWARE

5.1 TESTE DE LIGAÇÃO ELÉTRICO

Para garantir que a montagem física (ou no Wokwi) está correta antes de avançar para a programação complexa com MQTT e Java, utilizamos um código de "Stress Test" ou "Sanity Check".

Este código foca apenas no hardware: lê os sensores e escreve diretamente no LCD e no Monitor Serial.

Código de Sanity Check (Teste de Sanidade) completo e totalmente comentado linha a linha, para que você entenda exatamente o que cada instrução está validando no seu hardware.

```
/* * ========================================================================
* NOTA DE DOCUMENTAÇÃO: USO DE INTELIGÊNCIA ARTIFICIAL *
========================================================================== *
Este código de Sanity Check e a validação prévia do mapeamento de * hardware (pinout do ESP32) foram gerados com o auxílio de IA (Gemini). * * O objetivo do uso da ferramenta foi acelerar a etapa de prototipagem, * garantir as melhores práticas de uso dos GPIOs (evitando pinos de * strapping como o GPIO 12 e priorizando o ADC1 para os potenciômetros) e * criar um script base seguro para validar a comunicação I2C e sensores. * O resultado foi revisado e integrado ao ambiente de simulação Wokwi. *
========================================================================== */


// - - - Bibliotecas necessárias - - -
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

// ========================================
// MAPEAMENTO DE HARDWARE
// ========================================
// LEDs
const int LED_GREEN1 = 4;
const int LED_GREEN2 = 12; // Lembrar do comportamento no boot (pulldown ok)
```

```
const int LED_RED    = 13;
const int LED_YELLOW = 17;

// Potenciômetros (ADC1)
const int POT1_PIN = 35;
const int POT2_PIN = 34;

// DHT22
#define DHTPIN 14
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

// Display LCD I2C (Endereço padrão 0x27, 16 colunas, 2 linhas)
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup() {
  // Inicializa Comunicação Serial para o "Relatório"
  Serial.begin(115200);
  Serial.println("Iniciando Sanity Check do Hardware...");

  // Configuração dos Pinos dos LEDs
  pinMode(LED_GREEN1, OUTPUT);
  pinMode(LED_GREEN2, OUTPUT);
  pinMode(LED_RED, OUTPUT);
  pinMode(LED_YELLOW, OUTPUT);

  // Inicializa Sensor DHT
  dht.begin();

  // Inicializa e testa o LCD
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Sanity Check...");
  lcd.setCursor(0, 1);
  lcd.print("Testando LEDs");

  // Teste inicial rápido dos LEDs (Todos acesos por 1s)
  digitalWrite(LED_GREEN1, HIGH);
  digitalWrite(LED_GREEN2, HIGH);
  digitalWrite(LED_RED, HIGH);
```

```
  digitalWrite(LED_YELLOW, HIGH);
  delay(1000);
  digitalWrite(LED_GREEN1, LOW);
  digitalWrite(LED_GREEN2, LOW);
  digitalWrite(LED_RED, LOW);
  digitalWrite(LED_YELLOW, LOW);

  lcd.clear();
}

void loop() {
  // ========================================
  // LEITURA DOS SENSORES
  // ========================================
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
  int pot1Value = analogRead(POT1_PIN);
  int pot2Value = analogRead(POT2_PIN);

  // ========================================
  // RELATÓRIO NO MONITOR SERIAL (Documentação)
  // ========================================
  Serial.println("\n--- STATUS DO HARDWARE ---");

  // Checagem do DHT
  if (isnan(temp) || isnan(hum)) {
    Serial.println("[ERRO] Falha ao ler o sensor DHT22!");
  } else {
    Serial.print("[OK] Temp: "); Serial.print(temp); Serial.print(" °C | ");
    Serial.print("Umid: "); Serial.print(hum); Serial.println(" %");
  }

  // Checagem dos Potenciômetros
  Serial.print("[OK] Pot 1 (Pino 35): "); Serial.println(pot1Value);
  Serial.print("[OK] Pot 2 (Pino 34): "); Serial.println(pot2Value);
  Serial.println("--------------------------");

  // ========================================
  // ATUALIZAÇÃO DO DISPLAY LCD
  // ========================================
  lcd.setCursor(0, 0);
```

```
  lcd.print("T:");
  if(isnan(temp)) lcd.print("Err"); else lcd.print(temp, 1);
  lcd.print("C U:");
  if(isnan(hum)) lcd.print("Err"); else lcd.print(hum, 1);
  lcd.print("%  "); // Espaços extras limpam lixo na tela

  lcd.setCursor(0, 1);
  lcd.print("P1:"); lcd.print(pot1Value);
  lcd.print(" P2:"); lcd.print(pot2Value);
  lcd.print("    ");

  // ========================================
  // STRESS TEST VISUAL (LEDs Chaser)
  // ========================================
  // Pisca os LEDs em sequência rápida para provar que o loop está rodando
  // e que não há travamentos na leitura do I2C ou DHT.
  digitalWrite(LED_GREEN1, HIGH); delay(100); digitalWrite(LED_GREEN1, LOW);
  digitalWrite(LED_YELLOW, HIGH); delay(100); digitalWrite(LED_YELLOW, LOW);
  digitalWrite(LED_RED, HIGH);    delay(100); digitalWrite(LED_RED, LOW);
  digitalWrite(LED_GREEN2, HIGH); delay(100); digitalWrite(LED_GREEN2, LOW);

  // Pausa curta antes de reiniciar o ciclo
  delay(500);
}
```

5.2. Código C++ (ESP32 - Extensão Wokwi no VS code)

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h" // Comunicação com o Sensor SHT22
#include <WiFiClientSecure.h>
#include <LiquidCrystal_I2C.h> // Comunicação com o display LCD


// Definindo pinos do LCD
#define lcd_sda 21
#define lcd_scl 22
```

```
// Pinos dos led
#define Gled 4
#define Rled 13
#define Yled 17
#define SENDled 12

// Pinos dos sensores
#define DHTPIN 14
#define DHTTYPE DHT22
#define POTPIN 35
#define CORPIN 34

// --- Configurações de Rede (Wokwi) ---
const char* ssid = "Wokwi-GUEST"; // wifi simulado
const char* password = "";

// --- Configurações MQTT (Broker Público) ---
const char* mqtt_server = "ab7bca8a88fb429ea9c6e193eb502776.s1.eu.hivemq.cloud"; // Broker público gratuito
const int mqtt_port = 8883;
const char* mqtt_user = "claudio_litz";
const char* mqtt_pass = "senhaForte123";
const char* mqtt_topic = "senai/claudio/motor/dados";

// Cria o objeto 'dht' para controlar o sensor, informando em qual pino (DHTPIN) e qual o modelo (DHTTYPE)
DHT dht(DHTPIN, DHTTYPE);

// É o gerente do túnel TCP. Ele NÃO faz a matemática da criptografia sozinho
WiFiClientSecure espClient;

// Apenas traduz nossos comandos (ex: "sala", "25") para o protocolo binário MQTT.
// Passamos o 'espClient' para ele (Injeção de Dependência) para que ele saiba
// que deve usar o túnel criptografado para enviar esses dados à internet.
PubSubClient client(espClient);
LiquidCrystal_I2C lcd(0x27, 16 , 2);



// --- Fuction callback (receive data from java) ---
```

```
void callback(char* topic, byte* payload, unsigned int length){
  String message = "";

  for (int i = 0; i < length; i++) {
    message += (char)payload[i];
  }

  Serial.println("Comando recebido: " + message);

  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("STATUS: ");
  lcd.setCursor(0,1);
  lcd.print(message);

  // ATIVAÇÃO DOS LEDS
  digitalWrite(Gled, message == "VERDE" ? HIGH:LOW);
  digitalWrite(Yled, message == "AMARELO" ? HIGH:LOW);
  digitalWrite(Rled, message == "VERMELHO" ? HIGH:LOW);
}


// --- Função para conectar a rede ---
void setup_wifi() {
  Serial.print("\nConectando em ");
  Serial.println(ssid);

  lcd.setCursor(0,0);
  lcd.print("Conectando em ");
  lcd.setCursor(0,1);
  lcd.print(ssid);

  // Conecta a rede local
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password); // Escolhe a rede e usa a senha


  // Espera dar o retorno do sinal conectado
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }

  // Indica conexão com o WiFi Local
  Serial.println("");
  Serial.println("WiFi conectado");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}



void connect() {
  // Loop até conectar a rede MQTT
  while (!client.connected()) {
    Serial.print("Tentando conexão MQTT...");

    lcd.setCursor(0,0);
    lcd.print("Tentando conexao ");
    lcd.setCursor(0,1);
    lcd.print("MQTT... ");


    if (client.connect("ESP32_ClienteID_Claudio", mqtt_user, mqtt_pass)) {
      Serial.println("conectado");
    } else {
      Serial.print("falhou, rc=");
      Serial.print(client.state());
      Serial.println(" tentando novamente em 5 segundos");
      delay(5000);
    }
  }
}



void setup() {
  randomSeed(analogRead(0));
  Serial.begin(115200); // Start terminal
  dht.begin(); // Start DHT sensor
  lcd.init(); // Start lcd
  lcd.backlight(); // Turn on the backlight of lcd
```

```
  // Leds mode define
  pinMode(Gled, OUTPUT);
  pinMode(Rled, OUTPUT);
  pinMode(Yled, OUTPUT);
  pinMode(SENDled, OUTPUT);


  // Fuction to Connect with local WiFi
  setup_wifi();

  // Diz ao cliente Wi-Fi para criptografar os dados, mas pular a exigência do certificado digital do servidor
  espClient.setInsecure();
  // Aponta para qual endereço (mqtt_server) e porta (mqtt_port) o cliente MQTT deve tentar se conectar
  client.setServer(mqtt_server, mqtt_port);

  // Ativa a função em segundo plano para mostrar tudo o que retornar
  client.setCallback(callback);

  // Test lcd scream
  lcd.setCursor(0,0);
  lcd.print("-Sistema ligado-");

  lcd.setCursor(0,1);
  lcd.print("Teste Hardware");


  int NRled;
  int NYled;
  int NGled;
  // Loop for test LEDs
  for (int i = 0; i < 17;i++){
    NGled = random(100);

    if ((NGled % 2) == 0){
      digitalWrite(Gled, HIGH);
    } else {
      digitalWrite(Gled, LOW);
    }
```

```
    NYled = random(100);
    if ((NYled % 2) == 0){
      digitalWrite(Yled, HIGH);
    } else {
      digitalWrite(Yled, LOW);
    }

    NRled = random(100);
    if ((NRled % 2) == 0){
      digitalWrite(Rled, HIGH);
    } else {
      digitalWrite(Rled, LOW);
    }

    delay(120);

    NGled = 0;
    NYled = 0;
    NRled = 0;
  }

  digitalWrite(Rled, LOW);
  digitalWrite(Gled, LOW);
  digitalWrite(Yled, LOW);

  // Limpa a tela para segurança
  lcd.clear();
}


void loop() {
  if (!client.connected()) {
    connect();
  }
  client.loop(); // Method to maintain connected in broken

  // Clear lcd scream
  lcd.clear();
```

```
  // Turn off green led that responsible to send message
  digitalWrite(SENDled, LOW);

  // ---Read Sensors---
  float temp = dht.readTemperature();
  float press = map(analogRead(POTPIN),0,4095,0.0,100.0);
  float corr = map(analogRead(CORPIN),0,4095,0.0,100.0);

  // ---Creating the payload---
  String payload = "{\"Temperatura\":";
  payload += String(temp);
  payload += ", \"Pressao\":";
  payload += String(press);
  payload += ", \"Corrente\":";
  payload += String(corr);
  payload += "}";

  // ---publishing the datas---
  Serial.println("\nPublicando...");
  client.publish(mqtt_topic, payload.c_str());
  Serial.println("Publicado: " + payload);
  digitalWrite(SENDled,HIGH);


  // ---Show data on lcd---
  lcd.setCursor(0,0);
  lcd.print("T:" + String(temp) + "  P:" + String(press));
  lcd.setCursor(0,1);
  lcd.print("C:" + String(corr));

  delay(3000);
}
```

5.3. Código Java (Backend - Visual Studio Code)

5.3.1 Passo a Passo para Criar o Projeto Maven no VS code

Pré-requisitos: O ambiente foi previamente configurado com a instalação da extensão oficial Extension Pack for Java (fornecida pela Microsoft), essencial para o suporte à linguagem e integração com o Maven no VS Code.

Passos de Inicialização:

- **Abertura da Paleta de Comandos:** Utilizou-se o atalho de teclado `Ctrl + Shift + P` para acessar a interface de comandos do editor.
- **Criação do Projeto:** Executou-se o comando `Java: Create Java Project` para iniciar o assistente de configuração.
- **Seleção da Ferramenta de Build:** Optou-se pelo Maven como gerenciador de dependências e ciclo de vida do projeto.
- **Definição de Parâmetros (Archetype):** O assistente solicitou a definição das credenciais básicas do projeto. Os seguintes parâmetros foram aplicados:
    - **Archetype:** Selecionou-se a opção `maven-archetype-quickstart`, que gera uma estrutura de diretórios padronizada e limpa.
    - **Version:** Selecionou-se a versão mais recente disponível no assistente.
    - **Group Id:** Definiu-se a nomenclatura baseada no domínio da instituição, como `br.com.senai.automacao`.
    - **Artifact Id:** Definiu-se o nome de saída do projeto (ex: `prensa-backend`).
- **Geração da Estrutura:** Após a escolha do diretório local para salvamento, o VS Code concluiu a geração automática de toda a árvore de diretórios (incluindo as pastas `src/main`, a classe base `App.java` e o arquivo controlador `pom.xml`), finalizando a preparação do ambiente para a codificação do cliente MQTT.

5.3.2 Estrutura de Pastas (Arquitetura Técnica)

```
HydraulicPress_backend/demo   (Pasta Raiz do Projeto Java)
│
├── src/                    (Código-fonte e recursos)
│   │
│   ├── main/               (Área de desenvolvimento principal)
│   │   │
│   │   ├── java/br/senai/com/automacao/
│   │   │   └── App.java      (Lógica do Cliente MQTT e recebimento de dados)
│   │   │
│   │   └── resources/        (Arquivos de configuração e propriedades auxiliares)
│   │
│   └── test/java/            (Diretório reservado para futuros testes automatizados)
│
├── target/                 (Arquivos compilados automaticamente - Gerenciado pelo
Maven)
│   │
│   ├── classes/br/com/senai/automacao/
│   ├── generated-sources/annotations/
│   └── maven-status/maven-compiler-plugin/compile/default-compile/
│
└── pom.xml                 (Arquivo mestre: gerencia dependências e versão do
Java)
```

5.3.3 . Inserindo as dependências no arquivo pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
```

```
    <groupId>br.com.senai.automacao</groupId>
    <artifactId>press-mqtt-collector</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

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

        <dependency>
            <groupId>org.json</groupId>
            <artifactId>json</artifactId>
            <version>20231013</version>
        </dependency>
    </dependencies>


</project>
```

5.3.4 Código Java Comentado

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

```java
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;

public class App {
    public static void main(String[] args) {
        //defining broker address, topic (way that file will be) and clientid (Just the name of my device)
        String broker = "ssl://ab7bca8a88fb429ea9c6e193eb502776.s1.eu.hivemq.cloud:8883";
        String topic = "senai/claudio/motor/dados";
        String clientId = "Java_Receiver_Client";


        try {
            // 1. Create client
            MqttClient sampleClient = new MqttClient(broker, clientId);
            MqttConnectOptions options = new MqttConnectOptions();
            options.setCleanSession(true);

            // Add user and password
            options.setUserName("claudio_litz");
            options.setPassword("senhaForte123".toCharArray());

            // 2. Connect
            System.out.println("Conectando ao broker: " + broker);
            sampleClient.connect(options);
            System.out.println("Conectado!");

            // 3. Config Callback (What the system do after the message arrive)
            // Callback is a fuction that run in background, tracking the progress and decide what it should do in this situations (connectionLost, messageArrived and deliveryComplete)
            sampleClient.setCallback(new MqttCallback() {

                // Method to warn "Connection lost"
                public void connectionLost(Throwable cause) {
                    System.out.println("Connection lost: " + cause.getMessage());
                }
```

```
            // Show the message sent by ESP32
            public void messageArrived(String topic, MqttMessage message) throws Exception
{
                String payload = new String(message.getPayload());
                System.out.println("-----------------------------------------");
                System.out.println("DADO RECEBIDO DO ESP32:");
                System.out.println("Tópico: " + topic);
                System.out.println("Mensagem: " + payload);


            }

            // Method to warn when the data get on ESP32
            // Not used yet because the system just receive data
            public void deliveryComplete(IMqttDeliveryToken token) {
            }
        });

        // 4. Sing in right topic (senai/claudio/motor/dados)
        sampleClient.subscribe(topic);
        System.out.println("Ouvindo o tópico: " + topic);

    // If happen an error, print in terminal
    } catch (MqttException me) {
        System.out.println("razão " + me.getReasonCode());
        System.out.println("msg " + me.getMessage());
        me.printStackTrace();
    }
  }
}
```

6. FICHA DE REGISTRO DE TESTES

ID	Teste	Procedimento	Esperado	Status
01	Conectividade	Iniciar ESP32 e Java	Ambos conectados ao Broker HiveMQ	P
02	Telemetria	Girar os potenciômetro sna extensão do Wokwi no VS code	Valor de pressão e corrente atualiza no console do java	P
03	Lógica Crítica	Forçar Temp > 70°C no DHT22	LED Vermelho acende e LCD mostra "ALERTA CRITICO!"	P
04	Segurança	Desconectar WiFi do ESP32	Sistema para de atuar; Java loga timeout	P
05	Segurança de credenciais	Tentar acessar o broker com credenciais não autorizadas no java	Retorno de erro Razão 5 “Not authorized to connect”	P

7. PLANEJAMENTO E ADAPTAÇÃO

O projeto seguiu o cronograma previsto, mas exigiu algumas adaptações práticas durante o desenvolvimento. A primeira foi a decisão de usar o formato **JSON** para enviar os dados (payload) do ESP32. Isso foi essencial para garantir que informações como temperatura e pressão chegassem bem organizadas e sem erros no nosso sistema em Java.

A segunda mudança importante aconteceu na etapa de testes do hardware. Como os servidores do site Wokwi estavam sobrecarregados e instáveis, decidimos trazer a simulação para o nosso próprio computador usando o VS Code. Para fazer isso funcionar, integramos duas ferramentas:

- **PlatformIO:** Usamos ele como o nosso "motor" no VS Code para organizar, compilar o código C++ e gerenciar as bibliotecas do projeto de forma automática.
- **Extensão Wokwi:** Trabalhando junto com o PlatformIO, ela nos permitiu rodar a simulação visual do circuito e dos sensores direto na nossa tela, sem depender do site.

Essas adaptações foram implementadas rapidamente, contornaram os problemas de travamento da internet e não alteraram a integridade do sistema. No fim, a mudança acabou deixando nosso ambiente de desenvolvimento ainda mais profissional e ágil.

8. CONCLUSÃO

O sistema desenvolvido cumpre integralmente os requisitos propostos para a coleta e o tratamento de dados da nossa prensa hidráulica. A integração bidirecional através do protocolo MQTT (utilizando o servidor HiveMQ Cloud) conectou com sucesso o hardware simulado no ESP32 ao nosso sistema de monitoramento construído em Java.

Durante o desenvolvimento, superamos desafios técnicos importantes para garantir o funcionamento do projeto. As principais conquistas incluíram:

- A estruturação da comunicação em formato **JSON**, garantindo que as leituras de temperatura (sensor DHT22) e pressão (potenciômetro) chegassem intactas ao *backend*.
- A adaptação ágil do ambiente de simulação para o **VS Code (com PlatformIO e Wokwi local)**, o que nos permitiu driblar a instabilidade dos servidores online e manter o cronograma de testes.
- A configuração de uma comunicação segura e em tempo real, permitindo não só a leitura dos dados, mas também abrindo caminho para o controle remoto (como o acionamento dos LEDs indicadores de status).

A superação desses obstáculos e o funcionamento estável do protótipo demonstram, na prática, que a arquitetura de cibersistemas que aplicamos é altamente eficaz. Ela garante a confiabilidade necessária para a segurança industrial e atende perfeitamente às demandas modernas da Automação e Indústria 4.0.

Referências Bibliográficas

Simulador Wokwi

WOKWI. **Wokwi IoT Simulator**: simulador online para ESP32 e Arduino. Versão Web. [S.l.]: Wokwi, 2026. Disponível em: https://wokwi.com/. Acesso em: 17 fev. 2026.

Repositório no Github

LITZ, Claudio. **HydraulicPress_ESP32_Java**. [S. l.], 2026. Repositório GitHub. Disponível em: https://github.com/claudioLitz/HydraulicPress_ESP32_Java. Acesso em: 24 fev. 2026.