SENAI — CentroWEG · Jaraguá do Sul, SC

Relatório de Integração de Cibersistemas: Monitoramento e Controle de Prensa Hidráulica

Sistema de telemetria bidirecional via MQTT com ESP32 e backend Java

Ryhan Gabriel Schutz

Curso Técnico em CiberSistemas
Unidade Curricular: Programação para Coleta de Dados em Automação
Docente: Lucas Sousa dos Santos
Turma: T TCPA 2025/1 INT1

Jaraguá do Sul — Santa Catarina
2026

Este documento adota a fonte Helvetica como referência tipográfica,
em consonância com a tradição da escola suíça de design gráfico —
presente em manuais técnicos franceses, alemães e publicações da IEC.
A leveza tipográfica favorece a leitura técnica sem ornamentos desnecessários.

Ryhan Gabriel Schutz

Relatório de Integração de Cibersistemas: Monitoramento e Controle de Prensa Hidráulica

Relatório Técnico apresentado à Unidade Curricular de Programação para Coleta de Dados em Automação como requisito para avaliação do Bloco 01.

Docente: Lucas Sousa dos Santos.

Jaraguá do Sul — SC
2026

Sumário

1 Identificação do Projeto

- **Título:** Monitoramento e Controle de Prensa Hidráulica
- **Estudante:** Ryhan Gabriel Schutz
- **Ambientes:** Wokwi (Simulação) e GitHub Codespaces (Backend Java)
- **Repositório Java:** <https://github.com/ryhanschutz/Projeto_ESP32_MQTT_JAVA>
- **Projeto Wokwi:** <https://wokwi.com/projects/456753261676440577>

2 Protocolo MQTT e Broker HiveMQ (C1 e C2)

2.1 O que é MQTT

O **MQTT** (*Message Queuing Telemetry Transport*) é um protocolo de mensagens leve, baseado no modelo de publicação e assinatura (*publish/subscribe*), criado originalmente pela IBM na década de 1990 para monitoramento de oleodutos via satélite — contexto de baixa largura de banda e alta latência, muito semelhante aos desafios modernos de IoT industrial.

Diferentemente do modelo cliente-servidor tradicional, no MQTT nenhum dispositivo se comunica diretamente com outro. Todo o tráfego passa por um intermediário central chamado **broker**, que recebe mensagens publicadas em *tópicos* e as distribui a todos os assinantes daquele tópico. Isso desacopla produtores e consumidores de dados, tornando o sistema escalável: novos dispositivos podem ser adicionados sem alterar os já existentes.

No contexto industrial, o MQTT é amplamente encontrado em conjunto com plataformas como **Node-RED**, onde diferentes protocolos de campo — como **Modbus**, utilizado em CLPs, e **IO-Link**, presente em sensores como o Turck PS+ — são traduzidos e encaminhados via MQTT para sistemas supervisórios ou plataformas de nuvem.

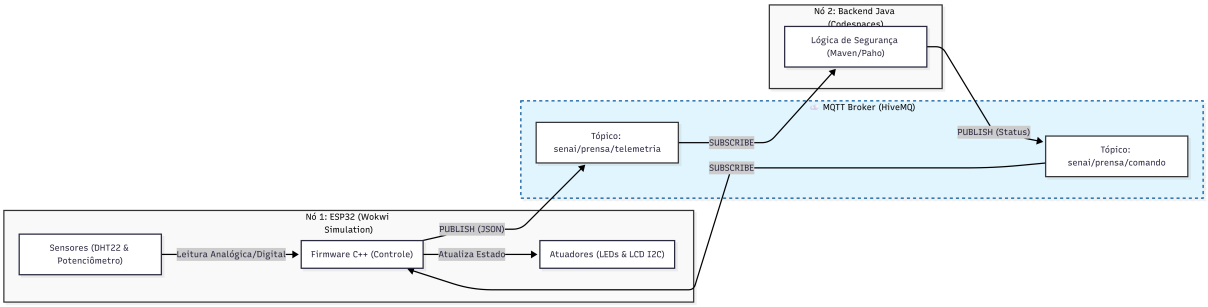


Figura 1 – Modelo de comunicação Publish/Subscribe do protocolo MQTT

2.2 Níveis de Qualidade de Serviço (QoS)

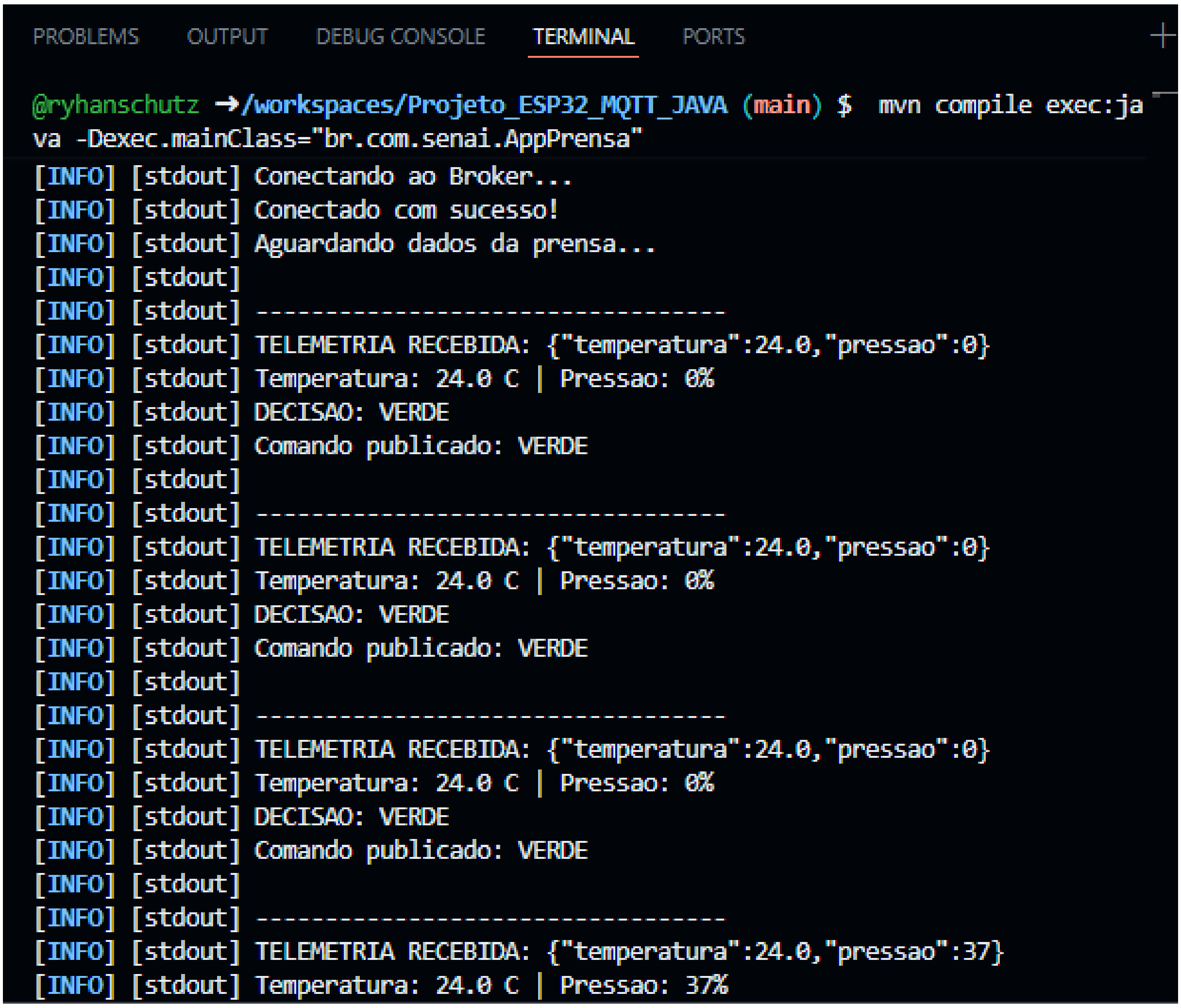
O MQTT define três níveis de garantia de entrega de mensagens:

- **QoS 0 — At most once:** A mensagem é enviada uma única vez, sem confirmação. Pode ser perdida. Adequado para leituras de alta frequência onde a perda ocasional é tolerável.
- **QoS 1 — At least once:** Entrega garantida, mas a mensagem pode ser recebida mais de uma vez. Utilizado neste projeto para os comandos publicados pelo Java ao ESP32 — garantindo que o LED correto seja sempre acionado.
- **QoS 2 — Exactly once:** Entrega exatamente uma vez, com handshake de quatro etapas. Usado em sistemas críticos onde duplicatas causam problemas, como acionamento de válvulas industriais.

2.3 O Broker HiveMQ

O **HiveMQ** é um broker MQTT desenvolvido pela HiveMQ GmbH, em Landshut, Alemanha. É amplamente utilizado em ambientes industriais e de IoT por sua escalabilidade e conformidade com os padrões MQTT 3.1.1 e 5.0. Para este projeto, utilizou-se o broker público gratuito `broker.hivemq.com` na porta 1883 — solução adequada para desenvolvimento e simulação.

Em produção, o HiveMQ Enterprise ou uma instância privada do Eclipse Mosquitto com TLS/SSL seria a escolha correta, conforme discutido na Seção 7.



```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

@ryhanschutz →/workspaces/Projeto_ESP32_MQTT_JAVA (main) $  mvn compile exec:ja
va -Dexec.mainClass="br.com.senai.AppPrensa"
[INFO] [stdout] Conectando ao Broker...
[INFO] [stdout] Conectado com sucesso!
[INFO] [stdout] Aguardando dados da prensa...
[INFO] [stdout]
[INFO] [stdout] ---------------------------------
[INFO] [stdout] TELEMETRIA RECEBIDA: {"temperatura":24.0,"pressao":0}
[INFO] [stdout] Temperatura: 24.0 C | Pressao: 0%
[INFO] [stdout] DECISAO: VERDE
[INFO] [stdout] Comando publicado: VERDE
[INFO] [stdout]
[INFO] [stdout] ---------------------------------
[INFO] [stdout] TELEMETRIA RECEBIDA: {"temperatura":24.0,"pressao":0}
[INFO] [stdout] Temperatura: 24.0 C | Pressao: 0%
[INFO] [stdout] DECISAO: VERDE
[INFO] [stdout] Comando publicado: VERDE
[INFO] [stdout]
[INFO] [stdout] ---------------------------------
[INFO] [stdout] TELEMETRIA RECEBIDA: {"temperatura":24.0,"pressao":0}
[INFO] [stdout] Temperatura: 24.0 C | Pressao: 0%
[INFO] [stdout] DECISAO: VERDE
[INFO] [stdout] Comando publicado: VERDE
[INFO] [stdout]
[INFO] [stdout] ---------------------------------
[INFO] [stdout] TELEMETRIA RECEBIDA: {"temperatura":24.0,"pressao":37}
[INFO] [stdout] Temperatura: 24.0 C | Pressao: 37%
```

Figura 2 – Console do backend Java no GitHub Codespaces — conexão ao broker HiveMQ

3 Arquitetura do Sistema (C1 e C2)

O sistema é composto por dois nós integrados via MQTT, operando de forma bidirecional:

- **Nó 1 — ESP32:** Coleta dados dos sensores, publica no tópico de telemetria e aguarda comandos para acionar LEDs e LCD.
- **Nó 2 — Backend Java:** Recebe a telemetria, aplica a lógica de segurança e publica o comando de resposta.

Tabela 1 – Mapeamento de tópicos MQTT do sistema

Tópico	Direção	Descrição
`senai/prensa/telemetria`	ESP32 → Java	Dados de temperatura e pressão em JSON
`senai/prensa/comando`	Java → ESP32	Status: VERDE, AMARELO ou VERMELHO

O payload de telemetria é formatado em JSON, permitindo encapsular múltiplos valores em uma única mensagem:

```
{"temperatura":38.5,"pressao":42}
```

4 Projeto de Hardware (C4)

A simulação foi realizada no **Wokwi**, ambiente de prototipagem virtual que permite simular circuitos com ESP32 sem necessidade de hardware físico — recurso fundamental para validação de firmware antes da fabricação de uma PCB.

4.1 Pinout e Componentes

Tabela 2 – Mapeamento de pinos do ESP32

Componente	Pino	Observação
DHT22 (dados)	GPIO 15	Sensor de temperatura e umidade
Potenciômetro (sinal)	GPIO 34	ADC somente leitura — simula pressão
LED Verde	GPIO 26	Estado SEGURO
LED Amarelo	GPIO 27	Estado ALERTA
LED Vermelho	GPIO 14	Estado CRÍTICO
LCD I2C — SDA	GPIO 21	Pino I2C padrão do ESP32
LCD I2C — SCL	GPIO 22	Pino I2C padrão do ESP32

O GPIO 34 foi selecionado para o potenciômetro por ser exclusivamente de entrada analógica no ESP32, evitando conflitos com saídas digitais. Os GPIOs 21 e 22 são os pinos I2C padrão, garantindo compatibilidade direta com a biblioteca `LiquidCrystal_I2C` sem configuração adicional. O endereço I2C do display utilizado é `0x27`.

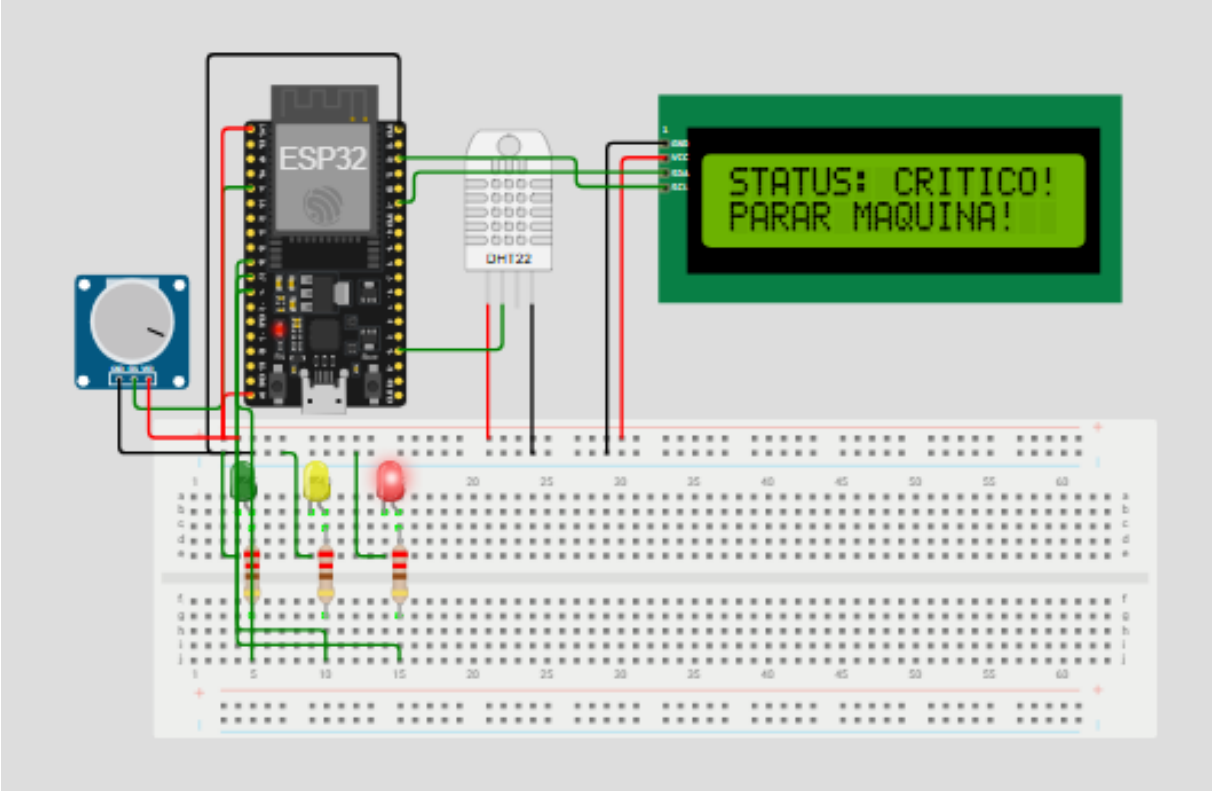


Figura 3 – Esquema elétrico do sistema simulado no Wokwi

5 Desenvolvimento de Software (C5, C6 e C8)

5.1 Paradigma Adotado

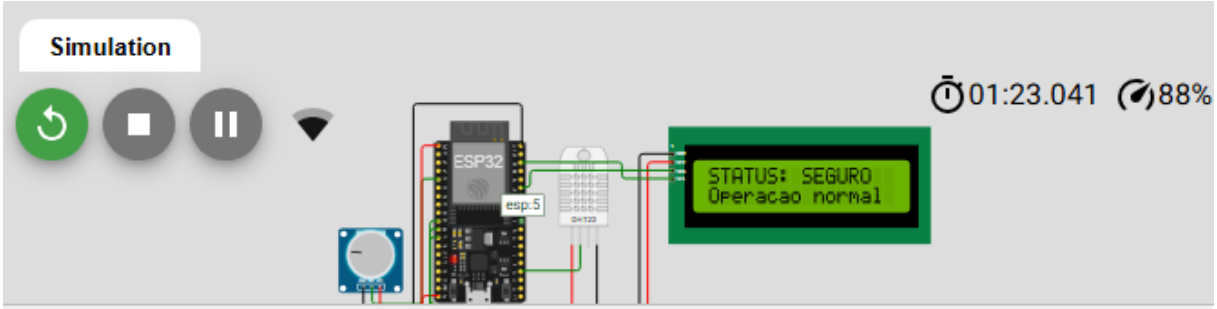
O firmware do ESP32 foi desenvolvido em **C++** com paradigma **procedural**, estrutura natural para sistemas embarcados onde o controle sequencial e determinístico do hardware é prioritário. O backend Java adota o mesmo paradigma, com uso de expressão *lambda* para o tratamento assíncrono das mensagens recebidas via MQTT — recurso disponível desde o Java 8.

5.2 Firmware C++ — ESP32 (Wokwi)

O firmware conecta-se à rede Wi-Fi do simulador, publica os dados de telemetria a cada segundo e aguarda comandos do backend Java para atualizar os LEDs e o display LCD.

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHTesp.h>
#include <LiquidCrystal_I2C.h>

const char* ssid        = "Wokwi-GUEST";
const char* mqttServer  = "broker.hivemq.com";
const int   mqttPort    = 1883;

#define PIN_DHT        15
#define PIN_POT        34
#define PIN_LED_GREEN  26
#define PIN_LED_YELLOW 27
#define PIN_LED_RED    14

const char* topicTelemetria = "senai/prensa/telemetria";
const char* topicComando    = "senai/prensa/comando";

// Callback: executado automaticamente ao receber mensagem MQTT
void callbackMQTT(char* topic, byte* payload,
                  unsigned int length) {
    String mensagem = "";
    for (unsigned int i = 0; i < length; i++)
        mensagem += (char)payload[i];
    if (String(topic) == topicComando)
        aplicarComando(mensagem); // Atualiza LEDs e LCD
}

void loop() {
    if (!mqtt.connected()) conectarMQTT();
    mqtt.loop(); // Essencial: processa mensagens recebidas

    TempAndHumidity dados = dhtSensor.getTempAndHumidity();
    // map() converte 0-4095 (ADC 12 bits) para 0-100 (%)
    int pressao = map(analogRead(PIN_POT), 0, 4095, 0, 100);

    String payload = "{\"temperatura\":"
        + String(dados.temperature, 1)
        + ",\"pressao\":" + String(pressao) + "}";
```

```
    mqtt.publish(topicTelemetria, payload.c_str());
    delay(1000);
}
```




```
Comando recebido: VERMELHO
Publicado: {"temperatura":38.4,"pressao":92}
Comando recebido: VERMELHO
Publicado: {"temperatura":38.4,"pressao":47}
Comando recebido: VERMELHO
Publicado: {"temperatura":38.4,"pressao":47}
Comando recebido: VERDE
Publicado: {"temperatura":3.3,"pressao":47}
Comando recebido: VERDE
Publicado: {"temperatura":3.3,"pressao":47}
Comando recebido: VERDE
Publicado: {"temperatura":3.3,"pressao":35}
Comando recebido: VERDE
Publicado: {"temperatura":3.3,"pressao":18}
Comando recebido: VERDE
Publicado: {"temperatura":3.3,"pressao":18}
```

Figura 4 – Monitor serial do ESP32 — publicação de telemetria em tempo real

5.3 Backend Java — GitHub Codespaces

O backend Java consome os dados do ESP32, realiza o *parse* manual do JSON, aplica a lógica de segurança e publica o comando de retorno. O projeto Maven utiliza a biblioteca **Eclipse Paho MQTT v1.2.5** como única dependência.

```
static String definirNivelSeguranca(
        double temp, double pressao) {
```

```
    // VERMELHO verificado primeiro - condição mais grave
    // temp > 60°C OU pressao > 80%
    if (temp > 60.0 || pressao > 80.0)
        return "VERMELHO";

    // AMARELO - condição intermediária
    // temp 45-60°C OU pressao 50-80%
    if ((temp >= 45.0 && temp <= 60.0)
     || (pressao >= 50.0 && pressao <= 80.0))
        return "AMARELO";

    // VERDE - todas as condições dentro do normal
    return "VERDE";
}
```

A ordem de verificação é intencional: ao checar **VERMELHO** antes de **AMARELO**, garante-se que uma condição crítica nunca seja classificada erroneamente como alerta. A inversão dessa lógica causaria falha silenciosa — a prensa em colapso exibiria LED amarelo, e operadores não interromperiam o processo, gerando risco à segurança e dano ao equipamento.

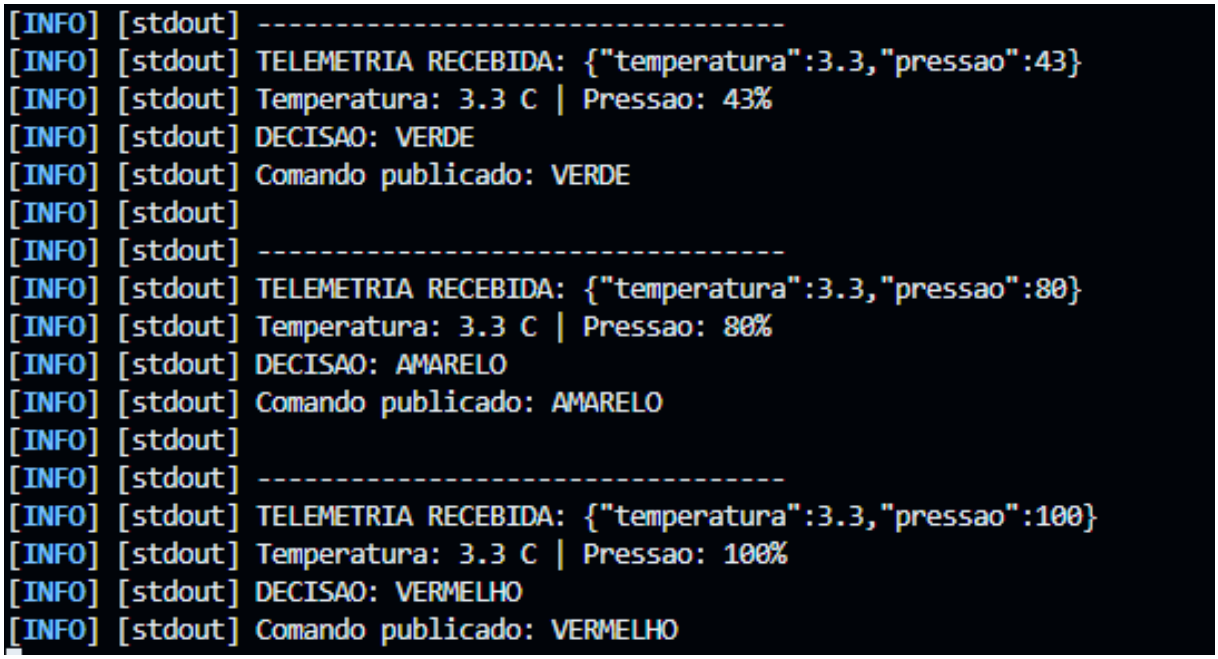


Figura 5 – Console do backend Java — processamento da telemetria e publicação de comandos

6 Planejamento e Adaptação (C3, S1 e S2)

6.1 Cronograma de Integração

Tabela 3 – Cronograma de execução das atividades

ID	Atividade	Responsável	Prazo	Status
01	Montagem do circuito Wokwi	Ryhan Schutz	23/02/2026	Concluído
02	Firmware C++ ESP32	Ryhan Schutz	23/02/2026	Concluído
03	Configuração Maven + Paho	Ryhan Schutz	23/02/2026	Concluído
04	Backend Java	Ryhan Schutz	23/02/2026	Concluído
05	Testes de integração	Ryhan Schutz	23/02/2026	Concluído

6.2 Relato de Adaptação

Durante o desenvolvimento, o **IntelliJ IDEA** local apresentou falha na resolução de dependências Maven — a biblioteca Paho MQTT não era carregada mesmo após múltiplas tentativas, resultando em 70 erros de compilação. Diante disso, a decisão

foi migrar para o **GitHub Codespaces**, ambiente baseado em VS Code com Maven disponível nativamente no terminal Linux.

A migração foi concluída sem impacto nos prazos. O Codespaces oferece integração nativa com Git e o comando `mvn compile exec:java` compilou e executou o projeto sem intercorrências. Essa experiência reforça uma competência fundamental do técnico em cibersistemas: adaptar o ambiente de trabalho diante de falhas de infraestrutura, mantendo o foco nos objetivos estabelecidos.

7 Segurança da Informação (C7)

Para evitar conflitos de sessão no broker — situação em que dois clientes com o mesmo ID se derrubam mutuamente — foi adotado um **Client ID único** gerado dinamicamente:

```
JavaPrensa_ + System.currentTimeMillis()
```

Em ambiente de produção industrial, recomenda-se:

- Broker privado (Eclipse Mosquitto) com **TLS/SSL** na porta 8883
- Autenticação por usuário, senha e certificado de cliente
- Controle de acesso por tópico (ACL)
- Conformidade com a norma **IEC 62443** para cibersegurança em ambientes de automação industrial

Fabricantes como **Siemens** (SIMATIC NET) e **Schneider Electric** disponibilizam gateways industriais com suporte nativo a MQTT seguro, certificados para uso em redes de controle industrial.

8 Projeção para Ambiente Industrial Real (C4)

O sistema desenvolvido utiliza componentes de prototipagem adequados para fins didáticos. Em um ambiente industrial real — como uma prensa hidráulica em linha de produção de uma indústria metalmecânica — cada componente seria substituído por equivalentes de nível industrial, conforme detalhado a seguir.

8.1 Sensores Industriais

O **DHT22**, sensor de baixo custo com precisão de $\pm$0,5 ℃, seria substituído por um sensor de temperatura industrial de maior precisão:

- **PT100 (RTD):** Precisão de $\pm$0,1 ℃, faixa de -200 ℃ a $+850$ ℃. Muito utilizado em processos onde a precisão é crítica, como tratamentos térmicos e conformação de metais.
- **Termopar tipo K:** Para faixas mais amplas, até 1300 ℃. Comum em fornos industriais e processos de fundição.

O **potenciômetro** simulando pressão seria substituído por um **transmissor de pressão industrial** com saída em 4–20 mA (padrão universal) ou comunicação **IO-Link**. O **Turck PS+** é um exemplo real de transmissor de pressão com IO-Link que, além da medição analógica, fornece diagnósticos do próprio sensor via protocolo digital — permitindo manutenção preditiva sem parar o processo.

8.2 Controlador Lógico Programável (CLP)

O ESP32 seria substituído por um **CLP** industrial. Exemplos de plataformas utilizadas no mercado nacional:

- **Siemens SIMATIC S7-1200** / **S7-1500:** Amplamente utilizado na indústria brasileira, com suporte a PROFINET, Modbus TCP/IP e comunicação MQTT via biblioteca adicional.
- **Schneider Electric Modicon M340:** CLP modular com suporte a Modbus TCP/IP e Ethernet industrial.
- **WEG TPW-03:** CLP nacional com suporte a Modbus RTU e TCP, adequado para integrações com sistemas supervisórios.

8.3 Arquitetura de Rede Industrial

A comunicação entre os dispositivos seguiria uma arquitetura em camadas conforme o modelo Purdue (ISA-95):

- **Nível de campo:** Sensores (PT100, transmissor IO-Link) conectados às entradas analógicas e ao master IO-Link do CLP.

- **Nível de controle:** CLP comunicando via **Modbus TCP/IP** ou **PROFINET** para um switch industrial (ex: Siemens SCALANCE X ou Phoenix Contact FL SWITCH).
- **Nível supervisório:** Sistema SCADA (ex: Wonderware, iFIX ou AVEVA) recebendo dados do CLP via OPC-UA.
- **Nível de nuvem (IIoT):** Dados encaminhados via MQTT seguro (TLS) para AWS IoT, Azure IoT Hub ou HiveMQ Enterprise, permitindo análise preditiva e manutenção preventiva.

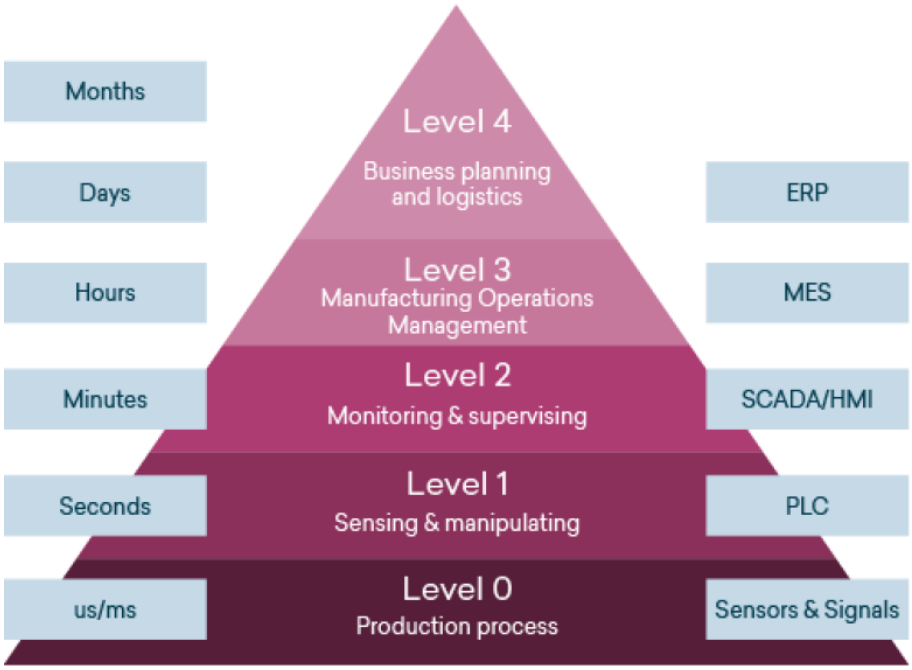


Figura 6 – Arquitetura industrial proposta em camadas — modelo Purdue/ISA-95

Esta projeção demonstra que o sistema desenvolvido, embora didático em seus componentes, segue os mesmos princípios arquiteturais de soluções industriais reais: comunicação orientada a mensagens, separação de responsabilidades entre nós e sinalização visual clara para os operadores de processo.

9 Plano de Testes e Evidências (C11)

9.1 Ficha de Registro de Testes

Tabela 4 – Resultados da validação do sistema

ID	Descrição	Procedimento	Resultado	Status
01	Conexão Wi-Fi ESP32	Monitor Serial Wokwi	IP obtido	OK
02	Conexão MQTT ESP32	Monitor Serial Wokwi	Conectado!	OK
03	Conexão MQTT Java	Terminal Codespaces	Conectado!	OK
04	Telemetria JSON	Console Java	Valores recebidos	OK
05	Estado VERDE	Pot < 50%, T < 45 °C	LED verde + LCD	OK
06	Estado AMARELO	Pot 50–80%	LED amarelo + LCD	OK
07	Estado VERMELHO	Pot > 80%	LED vermelho + LCD	OK
08	Client ID único	Dupla conexão	IDs distintos	OK

9.2 Relatório de Não Conformidade — NC-01

Falha na resolução de dependências Maven no IntelliJ IDEA local.

- **Sintoma:** 70 erros de compilação — classes da biblioteca Paho não encontradas mesmo com `pom.xml` correto.
- **Causa:** Falha no mecanismo de download do Maven local, possivelmente repositório local corrompido.
- **Ação corretiva:** Migração para GitHub Codespaces, com execução via: `mvn compile exec:java -Dexec.mainClass="br.com.senai.AppPrensa"`
- **Evidência:** Log "Conectado com sucesso!"e recepção de telemetria em tempo real no console do Codespaces.

10 Conclusão

O projeto atingiu todos os requisitos técnicos propostos. A integração bidirecional via MQTT entre o firmware C++ do ESP32 e o backend Java demonstrou-se estável e funcional, com latência compatível com aplicações de monitoramento industrial em tempo real.

O MQTT mostrou-se um protocolo adequado para o contexto: seu modelo *pub/sub* desacoplou os dois nós do sistema, e o broker HiveMQ proveu infraestrutura

confiável para o desenvolvimento sem configuração local. A experiência com o protocolo reforça sua relevância no contexto industrial mais amplo, onde é utilizado ao lado de Modbus e PROFINET em arquiteturas de automação modernas.

A migração necessária do IntelliJ para o GitHub Codespaces evidenciou uma competência fundamental no perfil do técnico em cibersistemas: adaptar o ambiente de trabalho diante de falhas de infraestrutura, mantendo o foco nos objetivos e prazos estabelecidos.

Por fim, a projeção para um ambiente industrial real — com PT100, transmissores IO-Link Turck, CLPs SIMATIC e Modicon, e comunicação via Modbus TCP/IP e PROFINET — demonstra que os conceitos aplicados neste projeto são diretamente escaláveis para soluções de automação industrial de nível profissional.

Referências

DECOTIGNIE, Jean-Dominique; PLEINEVAUX, Pierre. **A Survey of Fieldbus Protocols**. *Annales des Télécommunications*, v. 48, n. 9-10, p. 533–540, 1993. EPFL — École Polytechnique Fédérale de Lausanne, Suisse. Traduzido do francês pelo autor para fins de estudo.

SIEMENS AG. **SIMATIC NET: PROFIBUS Network Manual**. Nuremberg: Siemens AG, 2021.

HIVEMQ GMBH. **HiveMQ MQTT Broker Documentation**. München: HiveMQ, 2024. Disponível em: <https://www.hivemq.com/docs>. Acesso em: 23 fev. 2026.

WOKWI. **ESP32 Simulation Documentation**. Disponível em: <https://docs.wokwi.com>. Acesso em: 23 fev. 2026.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 14724**: Informação e documentação — Trabalhos acadêmicos. Rio de Janeiro: ABNT, 2011.