INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL MECÂNICA
CURSO DE GRADUAÇÂO EM ENGENHARIA MECATRÔNICA

Stephanie Pereira Souza

Desenvolvimento de uma Arquitetura para Monitoramento e Controle Remoto de um Dispositivo Proprietário IoT Baseado em ESP32

FLORIANÓPOLIS, 2026.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA - CÂMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE METAL MECÂNICA
CURSO DE GRADUAÇÂO EM ENGENHARIA MECATRÔNICA

Stephanie Pereira Souza

Desenvolvimento de uma Arquitetura para Monitoramento e Controle Remoto de um Dispositivo Proprietário IoT Baseado em ESP32

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro Mecatrônico.

Orientador:
Prof. Gregory Chagas da Costa Gomes, Mestre

FLORIANÓPOLIS, 2026.

Ficha de identificação da obra elaborada pelo autor.

Souza, Stephanie
Desenvolvimento de uma Arquitetura para Monitoramento e Controle Remoto de um Dispositivo Proprietário IoT Baseado em ESP32 / Stephanie Souza; orientação de Gregory Chagas da Costa Gomes. - Florianópolis, SC, 2026.
75 p.
Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Câmpus Florianópolis. Bacharelado em Engenharia Mecatrônica. Departamento Acadêmico de Metal Mecânica.
Inclui Referências.

1. Internet das Coisas. 2. Sistemas embarcados. 3. Protocolo de Comunicação. 4. Monitoramento Remoto. I. Chagas da Costa Gomes, Gregory . II. Instituto Federal de Santa Catarina. III. Desenvolvimento de uma Arquitetura para Monitoramento e Controle Remoto de um Dispositivo Proprietário IoT Baseado em ESP32.

DESENVOLVIMENTO DE UMA ARQUITETURA PARA MONITORAMENTO E CONTROLE REMOTO DE UM DISPOSITIVO PROPRIETÁRIO IOT BASEADO EM ESP32

STEPHANIE PEREIRA SOUZA

Este trabalho foi julgado adequado para obtenção do título de Engenheiro Mecatrônico e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Mecatrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina.

Florianópolis, 07 de julho, 2026.

Banca Examinadora:

Documento assinado digitalmente
GREGORY CHAGAS DA COSTA GOMES
Data: 06/08/2026 14:28:14-0300
CPF: ***.008.969-**
Verifique as assinaturas em https://v.ufsc.br

Gregory Chagas da Costa Gomes, Mestre
Instituto Federal de Santa Catarina

Documento assinado digitalmente
FRANCISCO RAFAEL MOREIRA DA MOTA
Data: 07/08/2026 08:13:41-0300
CPF: ***.003.153-**
Verifique as assinaturas em https://v.ufsc.br

Francisco Rafael Moreira da Mota, Doutor
Instituto Federal de Santa Catarina

Documento assinado digitalmente
VALDIR NOLL
Data: 06/08/2026 18:59:40-0300
CPF: ***.839.000-**
Verifique as assinaturas em https://v.ufsc.br

Valdir Noll, Doutor
Instituto Federal de Santa Catarina

Documento assinado digitalmente
ALISSON NUNES
Data: 06/08/2026 16:01:10-0300
CPF: ***.547.039-**
Verifique as assinaturas em https://v.ufsc.br

Alisson Nunes, Engenheiro
Segura

AGRADECIMENTOS

Agradeço, primeiramente, a mim mesma, que nunca desistiu. Desde pequena eu sabia que existia um lugar no mundo destinado à minha mente fértil e criativa, aquela mesma que costumava ligar lâmpadas de LED diretamente na tomada e conquistou um número considerável de marcas de queimado nas paredes de casa. Ao meu pai, que incontáveis vezes precisou levantar para religar os disjuntores que minhas belas invenções insistiam em desarmar, deixo também um pedido de desculpas e meu muito obrigada.

Agradeço à minha família, que sempre acreditou neste sonho sem jamais questionar minha escolha pela engenharia. Obrigada por compreender todas as vezes em que precisei ficar em casa estudando, pelo apoio financeiro, emocional e por nunca deixarem que eu desistisse desta jornada.

Ao meu namorado, meu agradecimento por caminhar ao meu lado durante tantos anos desta trajetória. Obrigada por ouvir meus desabafos, pelos conselhos, pela paciência e por ficar acordado até tarde comigo montando maquetes e desenvolvendo trabalhos das disciplinas, tornando esse caminho muito mais leve.

À minha melhor amiga, agradeço por ser uma inspiração constante. Você sempre foi um exemplo para mim e, se um dia eu me tornar metade da mulher incrível e talentosa que você é, já me considerarei realizada.

Agradeço também a todos os professores que fizeram parte da minha formação. Em especial, àqueles que demonstraram verdadeira paixão por ensinar, sem deixar de exigir o melhor de seus alunos. Professores como Francisco, Vitor e tantos outros contribuíram significativamente para minha formação acadêmica e profissional.

Por fim, agradeço ao meu orientador, Gregory, pela paciência, orientação e confiança durante o desenvolvimento deste trabalho. Obrigada por não desistir de mim, mesmo após alguns semestres de adiamentos, e por sempre estar disposto a ouvir, orientar e contribuir para a conclusão desta etapa tão importante da minha vida.

"Engineers like to solve problems.
If there are no problems handily available,
they will create their own."

— Scott Adams

RESUMO

A crescente adoção de aplicações baseadas em Internet das Coisas (*IoT*) tem ampliado a necessidade de arquiteturas capazes de integrar aquisição de dados, comunicação, armazenamento e supervisão remota de dispositivos. Nesse contexto, este trabalho apresenta o desenvolvimento de uma arquitetura de monitoramento e controle baseada em um *hardware* proprietário para aquisição de sinais digitais, integrado a um servidor, banco de dados e interface supervisória, permitindo o monitoramento remoto, o registro de eventos e o acionamento de dispositivos por meio da internet. A solução desenvolvida utiliza uma placa proprietária baseada no microcontrolador ESP32, responsável pela aquisição dos sinais digitais, processamento dos eventos e controle das saídas digitais a relés. O sistema integra um servidor para gerenciamento das mensagens, persistência dos dados e disponibilização das informações por meio de uma interface web. Para atender às diferentes necessidades da aplicação, foram empregados os protocolos MQTT, utilizado na comunicação entre o servidor e o dispositivo embarcado, e HTTP, utilizado para o envio das notificações de retorno dos eventos de controle à interface supervisória. Além disso, foram implementados mecanismos de armazenamento local em cartão SD e armazenamento remoto em banco de dados, permitindo a retenção, consulta e auditoria dos eventos gerados pelo sistema. A validação da arquitetura foi realizada por meio de testes funcionais e testes de estresse, avaliando seu comportamento em diferentes condições de operação e diferentes taxas de geração de mensagens. Os resultados demonstraram o funcionamento integrado entre o dispositivo embarcado, servidor e interface supervisória, possibilitando o monitoramento remoto, o controle dos dispositivos e o registro confiável dos eventos. Os testes também permitiram identificar oportunidades de otimização e forneceram subsídios para sua evolução em trabalhos futuros.

Palavras-chave: Internet das Coisas. Sistemas embarcados. Protocolo de Comunicação. Monitoramento Remoto.

ABSTRACT

The growing adoption of Internet of Things (IoT) applications has increased the demand for architectures capable of integrating data acquisition, communication, storage, and remote device supervision. In this context, this work presents the development of a monitoring and control architecture based on proprietary hardware for digital signal acquisition, integrated with a server, database, and supervisory interface, enabling remote monitoring, event logging, and device control over the Internet. The proposed solution employs a proprietary board based on the ESP32 microcontroller, responsible for digital signal acquisition, event processing, and relay output control. The system integrates a server for message management, data persistence, and information availability through a web-based supervisory interface. To meet the different communication requirements of the application, the MQTT protocol was employed for communication between the embedded device and the server, while HTTP was used to notify the supervisory interface of control event responses. In addition, local data storage on an SD card and remote storage in a database were implemented, enabling event retention, querying, and auditing. The proposed architecture was validated through functional and stress tests, evaluating its behavior under different operating conditions and message generation rates. The results demonstrated the integrated operation of the embedded device, server, and supervisory interface, enabling remote monitoring, device control, and reliable event logging. The tests also identified opportunities for optimization and provided a foundation for future improvements and further research.

Keywords: Internet of Things. Embedded Systems. Communication Protocol. Remote Monitoring.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface* (Interface de Programação de Aplicações)
CSS	*Cascading Style Sheets* (Folhas de Estilo em Cascata)
GPIO	*General Purpose Input/Output* (Entrada e Saída de Propósito Geral)
HTML	*HyperText Markup Language* (Linguagem de Marcação de Hipertexto)
HTTP	*Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)
I²C	*Inter-Integrated Circuit* (Barramento de Comunicação Interintegrado)
IFSC	Instituto Federal de Santa Catarina
IIoT	*Industrial Internet of Things* (Internet Industrial das Coisas)
IoT	*Internet of Things* (Internet das Coisas)
JSON	*JavaScript Object Notation* (Notação de Objetos JavaScript)
MQTT	*Message Queuing Telemetry Transport* (Protocolo de Transporte de Telemetria por Enfileiramento de Mensagens)
MySQL	*My Structured Query Language* (Linguagem de Consulta Estruturada My)
PHP	*PHP: Hypertext Preprocessor* (Pré-processador de Hipertexto)
REST	*Representational State Transfer (Transferência Representacional de Estado)*
SD	*Secure Digital* (Cartão Digital Seguro)
TCP	*Transmission Control Protocol* (Protocolo de Controle de Transmissão)
TLS	*Transport Layer Security* (Segurança da Camada de Transporte)
WSS	*WebSocket Secure* (WebSocket Seguro)

SUMÁRIO

1 INTRODUÇÃO

Com base no crescimento acelerado do campo da domótica, termos como Internet das Coisas (IoT) têm se tornado cada vez mais conhecidos e relevantes. Esse avanço tem impulsionado uma expansão significativa dos sistemas de monitoramento, tanto domésticos quanto industriais, tornando-os mais confiáveis, otimizados e acessíveis. O custo dos dispositivos inteligentes tem diminuído ao longo dos anos, de maneira inversamente proporcional ao aumento do seu desempenho, o que contribui para essa transformação (SPANTI, 2023).

Embora os sistemas de monitoramento e o acesso a dados via internet não representem inovações tecnológicas recentes, o diferencial atual está na possibilidade de alterar, de forma online, a planta industrial conforme o surgimento de demanda. Em outras palavras, é possível controlar a planta industrial pela internet, seja esse controle manual, realizado por um operador, ou automatizado.

Essas inovações possibilitam a supervisão remota online de processos industriais, oferecendo uma resposta rápida e precisa a eventos críticos. A tecnologia embarcada desempenha um papel essencial na garantia da eficiência e da segurança, especialmente em ambientes onde a presença humana é inviável ou apresenta riscos significativos à saúde. Assim, a integração da IoT com tecnologias avançadas permite uma gestão mais inteligente e automatizada dos processos industriais, prevenindo falhas e otimizando a manutenção.

No contexto da Indústria 4.0, a integração entre dispositivos de aquisição de dados, sistemas supervisórios e redes de comunicação tem adquirido crescente importância nas aplicações de monitoramento e controle industrial. Essas arquiteturas permitem a troca de informações entre equipamentos, sistemas e plataformas computacionais, possibilitando o monitoramento remoto dos processos e uma gestão mais eficiente das operações. Nesse cenário, a escolha das tecnologias empregadas, como protocolos de comunicação e plataformas de desenvolvimento, influencia diretamente aspectos como desempenho, interoperabilidade, escalabilidade e custos de implementação do sistema (LU, 2017).

Atualmente, fabricantes consolidados oferecem soluções de monitoramento e supervisão industrial baseadas em arquiteturas compostas por um sistema de aquisição de dados em campo, uma infraestrutura de comunicação, um servidor para processamento e armazenamento das informações e um sistema supervisório responsável pela visualização e pelo controle do processo. Essa organização permite integrar diferentes equipamentos e centralizar o monitoramento das operações, favorecendo a tomada de decisão e a rastreabilidade dos eventos.

Nesse contexto, este trabalho insere-se no desenvolvimento de uma arqui-

tetura com características semelhantes, na qual o sistema de aquisição de dados é implementado por meio de um dispositivo embarcado proprietário[1], integrado a um servidor, um banco de dados e um sistema supervisório. Além disso, são avaliadas diferentes estratégias de comunicação entre esses componentes, considerando aspectos como desempenho, latência, segurança e complexidade de implementação.

1.1 Justificativa e Relevância

A criação de sistemas de monitoramento online é muito importante para a indústria moderna, especialmente com o avanço da automação e da Internet das Coisas (IoT). Em ambientes industriais, a capacidade de acompanhar processos e equipamentos de forma online contribui para a supervisão das operações, auxiliando na identificação de falhas e apoiando as atividades de manutenção.

A relevância deste trabalho está no desenvolvimento de uma solução completa de monitoramento industrial, abrangendo desde a aquisição dos dados em campo até seu processamento, armazenamento e disponibilização para supervisão remota. Mais do que o desenvolvimento de um componente específico, o trabalho integra todos os elementos necessários para o funcionamento de uma arquitetura completa de monitoramento, permitindo que as informações coletadas sejam utilizadas não apenas para acompanhamento online, mas também como base para futuras ações de controle, automação e apoio à tomada de decisão.

Este trabalho tem um impacto prático relevante, pois o sistema desenvolvido poderá ser aplicado em diversos setores industriais, oferecendo controle dos processos. Além disso, a integração de dispositivos de aquisição de dados com o sistema supervisório permitirá uma resposta ágil a eventos, aumentando a confiabilidade e a segurança dos processos monitorados.

Além da contribuição prática, este trabalho também busca demonstrar a aplicação de diferentes tecnologias utilizadas em sistemas de monitoramento industrial. A integração dessas tecnologias em uma única arquitetura permite analisar, na prática, aspectos relacionados à comunicação, ao armazenamento e à disponibilização das informações, contribuindo para o desenvolvimento de soluções de monitoramento voltadas ao contexto da IoT.

A escolha deste tema também está ligada à minha trajetória acadêmica e profissional. O desenvolvimento deste sistema é um passo importante para consolidar as habilidades adquiridas durante o curso e atender às necessidades do mercado por soluções de monitoramento mais eficientes.

[1] O dispositivo de aquisição de dados utilizado neste trabalho foi desenvolvido previamente pelo autor durante sua atuação na empresa Eurotec Nutrition.

1.2 Definição do Problema

A crescente adoção de soluções baseadas em Internet das Coisas (IoT) tem ampliado a necessidade de monitoramento e controle remoto de dispositivos utilizados em processos industriais e de automação. Entretanto, ainda existem sistemas embarcados que operam de forma isolada, sem mecanismos que permitam supervisão remota, armazenamento centralizado de informações e interação por meio de interfaces acessíveis via internet.

No contexto deste trabalho, o *hardware* Euroino disponibiliza recursos para aquisição de sinais digitais e acionamento de dispositivos externos. Contudo, sua utilização sem uma infraestrutura adequada de comunicação e armazenamento dificulta o acompanhamento dos eventos gerados pelo sistema, a consulta histórica das informações e a realização de comandos remotos de forma prática e eficiente.

Além disso, aplicações de monitoramento remoto demandam mecanismos que permitam a troca de informações online, a persistência dos dados gerados e a disponibilização dessas informações para consulta por usuários remotos.

Dessa forma, o problema abordado neste trabalho consiste em como integrar um *hardware* proprietário de monitoramento e controle a uma plataforma capaz de disponibilizar informações remotamente, registrar eventos de operação e permitir o acionamento de dispositivos por meio da internet, utilizando tecnologias compatíveis com o paradigma da Internet das Coisas.

1.3 Objetivo Geral

Desenvolver um sistema *web* para o monitoramento remoto de processos industriais controlados por um *hardware* proprietário, utilizando conceitos da Internet das Coisas.

1.4 Objetivos Específicos

- Desenvolver e implementar um sistema *web* para o recebimento, persistência e visualização dos dados provenientes do dispositivo embarcado.
- Projetar e configurar a infraestrutura do sistema, incluindo o servidor *web* e o banco de dados.
- Integrar o sistema de monitoramento ao dispositivo de aquisição de dados, possibilitando a comunicação bidirecional entre a interface *web* e o *hardware* embarcado.
- Testar e validar o sistema em um ambiente controlado que simule condições de operação de uma aplicação industrial, avaliando seu comportamento durante a

execução dos testes.

1.5 Estrutura do Trabalho

Este trabalho está organizado em seis capítulos, além dos elementos pré-textuais e pós-textuais.

O Capítulo 1 apresenta a introdução do trabalho, contemplando a contextualização do tema, a definição do problema, os objetivos e a organização do documento.

O Capítulo 2 apresenta a fundamentação teórica, abordando os principais conceitos relacionados à Internet das Coisas, protocolos de comunicação, sistemas embarcados, bancos de dados e demais tecnologias utilizadas no desenvolvimento da solução proposta.

O Capítulo 3 descreve a metodologia adotada, apresentando os procedimentos utilizados para o desenvolvimento do sistema, os recursos empregados e a arquitetura proposta para integração entre os diferentes componentes da solução.

O Capítulo 4 apresenta o desenvolvimento da solução, detalhando a implementação do *hardware*, do *firmware* embarcado, do *backend*, do banco de dados, do *broker* MQTT e da interface supervisória.

O Capítulo 5 apresenta os testes realizados e os resultados obtidos durante a validação do sistema, incluindo a análise do comportamento da solução diante dos cenários propostos.

O Capítulo 6 por fim, são apresentadas as conclusões do trabalho, bem como as sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica que sustenta o desenvolvimento do sistema proposto. O objetivo é contextualizar o trabalho dentro do estado da arte, destacando conceitos, tecnologias e pesquisas relacionadas às áreas de sistemas embarcados, Internet das Coisas (IoT), protocolos de comunicação e integração com serviços web.

Inicialmente, são discutidos os trabalhos correlatos, que permitem identificar soluções já desenvolvidas e compreender como diferentes abordagens têm sido aplicadas em projetos semelhantes. Em seguida, a revisão bibliográfica aborda os principais conceitos técnicos que embasam o projeto.

2.1 Trabalhos correlatos

Nesta seção são apresentados trabalhos relacionados ao tema desta pesquisa, com o objetivo de apresentar diferentes abordagens empregadas no desenvolvimento de sistemas de monitoramento industrial baseados em Internet das Coisas (IoT). A análise desses trabalhos permite compreender as arquiteturas propostas, as tecnologias utilizadas e as contribuições apresentadas na literatura, servindo como base para contextualizar a solução desenvolvida neste trabalho.

2.1.1 Uma Arquitetura para a Aplicação da Internet das Coisas Industrial

Segundo Ferreira (2018), foi proposta uma arquitetura para aplicações de Internet Industrial das Coisas (IIoT) com o objetivo de integrar dispositivos industriais, sistemas de automação e aplicações computacionais por meio de uma infraestrutura baseada em tecnologias de código aberto. A arquitetura é composta por três módulos principais: um servidor de aplicação responsável pelo gerenciamento das informações, um módulo *gateway*, encarregado da comunicação entre os dispositivos industriais e a rede, e módulos remotos destinados à aquisição de dados e ao controle de equipamentos em campo.

Para possibilitar a comunicação entre os diferentes componentes da arquitetura, o autor emprega protocolos amplamente utilizados em aplicações industriais, como Modbus TCP/IP, MQTT, CoAP e HTTP no padrão *Representational State Transfer* (REST), utilizando mensagens no formato JSON. Além disso, o trabalho utiliza o OpenPLC para implementação da lógica de controle, demonstrando a integração entre tecnologias de automação industrial e soluções baseadas em IIoT (FERREIRA, 2018).

Os experimentos realizados validaram o funcionamento da arquitetura proposta, demonstrando sua capacidade de integrar diferentes dispositivos e protocolos de comunicação em uma solução voltada à Internet Industrial das Coisas. Os resultados

obtidos evidenciam o potencial da arquitetura para aplicações de monitoramento e controle industrial, fornecendo uma base para o desenvolvimento de soluções distribuídas voltadas ao contexto da Indústria 4.0 (FERREIRA, 2018).

Embora ambos os trabalhos sejam voltados para aplicações de Internet das Coisas,(FERREIRA, 2018) concentra-se na proposição e validação de uma arquitetura de comunicação entre dispositivos industriais utilizando um módulo *gateway* como elemento central da integração. Já o presente trabalho tem como foco o desenvolvimento de uma solução completa de monitoramento industrial, contemplando a aquisição de dados por meio de um dispositivo embarcado, seu processamento, armazenamento e disponibilização em uma interface supervisória *web*.

A Tabela 1 apresenta uma comparação entre as principais características do trabalho desenvolvido por Ferreira (2018) e a solução proposta neste trabalho, evidenciando as diferenças entre as arquiteturas e os recursos implementados.

Tabela 1 – Comparação entre o trabalho de Ferreira (2018) e o presente trabalho.

Característica	**Ferreira (2018)**	**Este trabalho**
Aquisição de dados	✓	✓
Monitoramento remoto	✓	✓
Controle remoto	✓	✓
Sistema supervisório *web*		✓
Uso de *gateway*	✓	
Uso de PLC (*OpenPLC*)	✓	
Servidor de aplicação	✓	✓
Hardware proprietário		✓
Arquitetura baseada em IoT	✓	✓
Validação experimental	✓	✓

Fonte: Autoria própria

2.1.2 Sistema embarcado e supervisório para uma aplicação IoT

Segundo MELO (2023), foi desenvolvido um sistema embarcado baseado no microcontrolador ESP32 integrado a uma interface supervisória *web*, com o objetivo de possibilitar o monitoramento e o controle remoto de dispositivos por meio da Internet. A arquitetura proposta utiliza o *Firebase Realtime Database* para sincronização dos dados entre a interface supervisória e o sistema embarcado, empregando o mecanismo de *HTTP Streaming* para manter ambos sincronizados e permitir o envio de comandos de forma orientada a eventos (MELO, 2023).

Os experimentos realizados demonstraram o funcionamento integrado entre o sistema embarcado e a interface supervisória, validando a arquitetura proposta para aplicações de monitoramento e controle remoto. Além disso, o trabalho destaca a flexibilidade da solução, permitindo sua adaptação para diferentes cenários de automação por meio de alterações no *firmware* e na interface supervisória (MELO, 2023).

Embora ambos os trabalhos proponham soluções baseadas em Internet das Coisas utilizando o microcontrolador ESP32 e uma interface supervisória *web*, existem diferenças importantes entre suas arquiteturas. O trabalho de MELO (2023) utiliza o *Firebase Realtime Database* e o *HTTP Streaming* como mecanismos centrais de comunicação entre a interface e o sistema embarcado. Já o presente trabalho adota uma arquitetura distribuída baseada em um *broker* MQTT para a comunicação entre os dispositivos e um servidor dedicado para processamento e persistência dos dados. Além disso, a solução desenvolvida incorpora armazenamento local em cartão SD, utilização conjunta dos protocolos MQTT e HTTP e validação por meio de testes funcionais e de estresse.

A Tabela 2 apresenta uma comparação entre as principais características do trabalho desenvolvido por MELO (2023) e a solução proposta neste trabalho, evidenciando as diferenças entre as arquiteturas e os recursos implementados.

Tabela 2 – Comparação entre o trabalho de Melo (2023) e o presente trabalho.

Característica	Melo (2023)	Este trabalho
Aquisição de dados	✓	✓
Monitoramento remoto	✓	✓
Controle remoto	✓	✓
Sistema supervisório *web*	✓	✓
ESP32	✓	✓
Mecanismo de comunicação	*HTTP Streaming*	MQTT e HTTP
Backend próprio		✓
Banco de dados	✓	✓
Armazenamento em cartão SD		✓
Hardware proprietário		✓
Validação experimental	✓	✓

Fonte: Autoria própria

Observa-se que ambos os trabalhos apresentam funcionalidades semelhantes quanto à aquisição de dados, monitoramento e controle remoto. Entretanto, a solução proposta neste trabalho diferencia-se pela utilização de um servidor dedicado e de um broker MQTT, o que proporciona maior flexibilidade para integração com diferentes aplicações e maior independência em relação a serviços de terceiros. Além disso, a inclusão do armazenamento local em cartão SD amplia a confiabilidade do sistema, permitindo o registro dos eventos mesmo em situações de indisponibilidade da comunicação com o servidor.

2.2 Revisão bibliográfica

A revisão bibliográfica tem como objetivo apresentar os principais conceitos e tecnologias que fundamentam o desenvolvimento deste trabalho. São discutidos aspectos relacionados aos sistemas embarcados, à Internet das Coisas (IoT), aos protocolos de comunicação e aos sistemas supervisórios, estabelecendo a base teórica necessária para compreender as decisões de projeto adotadas e a arquitetura proposta.

2.2.1 Sistemas Embarcados

"Sistemas embarcados são sistemas de processamento de informações incorporados em produtos que os envolvem." (MARWEDEL, 2018, p. 2, tradução nossa). De forma complementar, Lee (2002) destaca que sistemas embarcados não devem ser compreendidos apenas como processadores de informação isolados, mas como sistemas integrados ao mundo físico. Segundo o autor, esses sistemas são executados em máquinas que não são, primordialmente, computadores, como aviões, automóveis, máquinas industriais, telefones, robôs, sistemas de manufatura e outros.

Dessa maneira, observa-se que os sistemas embarcados caracterizam-se pela integração entre *hardware* e *software*, com forte interação com o ambiente físico e requisitos específicos de desempenho e confiabilidade.

Características principais

Entre as principais características dos sistemas embarcados destacam-se a especialização funcional e as restrições de recursos. Conforme discutido por Marwedel (MARWEDEL, 2018), sistemas embarcados são desenvolvidos para executar funções específicas, o que implica em projetos otimizados para determinada aplicação. Essa especialização diferencia-os de sistemas de propósito geral e influencia diretamente as escolhas de arquitetura de hardware e software.

Outra característica relevante é a limitação de recursos computacionais, como capacidade de processamento, memória disponível e consumo de energia. Essas restrições exigem soluções eficientes e cuidadosamente planejadas, principalmente em dispositivos alimentados por bateria ou aplicados em ambientes industriais.

Relação com o projeto desenvolvido

No contexto deste trabalho, o sistema embarcado é composto por uma placa eletrônica contendo um microcontrolador, responsável pela aquisição de dados e envio para um servidor remoto.

O dispositivo embarcado realiza a leitura dos dados, executa rotinas de processamento e transmite as informações para o *backend*, que realiza o armazenamento dos dados no banco de dados por meio de protocolos de comunicação adequados ao ambiente de Internet das Coisas (IoT). Dessa forma, aplica-se na prática o conceito apresentado por Marwedel (MARWEDEL, 2018), no qual o sistema embarcado está incorporado a um produto específico e executa funções dedicadas.

Portanto, o sistema embarcado desenvolvido neste trabalho constitui a camada de aquisição e controle do sistema de automação, integrando-se ao *backend* e ao sistema supervisório por meio de APIs e protocolos de comunicação, compondo a arquitetura proposta neste trabalho.

2.2.2 Redes de Comunicação IoT

O termo Internet das Coisas, mais conhecido como IoT pode ser conceitualmente definido como "uma infraestrutura de rede global dinâmica, com capacidades de autoconfiguração, baseada em protocolos de comunicação padronizados e interoperáveis, na qual objetos físicos e virtuais possuem identidades e estão integrados à rede de informação" (GUILLEMIN; FRIESS, 2009, p. 10, tradução nossa). De acordo com Guillemin e Friess (2009), a Internet das Coisas refere-se à capacidade de conectar objetos à Internet, independentemente de tempo e localização, utilizando diferentes redes e serviços de comunicação.

Setores como aeroespacial, médico, automotivo e de telecomunicações represen-

tam alguns exemplos de aplicação da Internet das Coisas na indústria (PERERA *et al.*, 2014). Nesses cenários, objetos físicos equipados com sensores e sistemas embarcados são capazes de coletar, transmitir e compartilhar informações por meio de redes de comunicação. Para viabilizar a troca de dados entre dispositivos e sistemas computacionais, diferentes protocolos de comunicação são utilizados. Entre eles destacam-se protocolos da camada de aplicação amplamente empregados em soluções de IoT, como HTTP e MQTT.

2.2.2.1 HTTP

O *Hypertext Transfer Protocol* (HTTP) é um protocolo da camada de aplicação amplamente utilizado para comunicação na *Web*. Ele define um conjunto de regras para a transferência de informações entre clientes e servidores em sistemas distribuídos baseados em hipertexto (FIELDING *et al.*, 1999).

O HTTP opera segundo o modelo cliente-servidor e utiliza um mecanismo de requisição e resposta, no qual um cliente envia uma solicitação a um servidor e recebe como retorno uma resposta contendo os dados solicitados ou o resultado da operação realizada (FIELDING *et al.*, 1999).

Devido à sua ampla adoção na Internet, o HTTP também pode ser utilizado em aplicações de Internet das Coisas para a comunicação entre dispositivos e serviços remotos. Entretanto, por ter sido originalmente projetado para aplicações *web* tradicionais, o protocolo pode apresentar maior sobrecarga de comunicação em dispositivos com recursos limitados, o que motiva o uso de protocolos mais leves em determinados cenários de IoT (MISHRA; KERTESZ, 2020).

2.2.2.2 MQTT

O *Message Queuing Telemetry Transport* (MQTT) é um protocolo de comunicação da camada de aplicação amplamente utilizado em sistemas de Internet das Coisas. Ele foi projetado para operar em dispositivos com recursos computacionais limitados e em redes com restrições de largura de banda, características comuns em aplicações de IoT (SHAHROKHI; AHMADI, 2019).

O protocolo utiliza o modelo de comunicação *publish/subscribe*, no qual os dispositivos enviam mensagens para um intermediário denominado *broker*. Essas mensagens são organizadas em tópicos, e outros dispositivos podem se inscrever nesses tópicos para receber automaticamente as informações publicadas (SHAHROKHI; AHMADI, 2019).

Devido à sua eficiência e à adequação para ambientes com restrições de recursos, o MQTT tornou-se amplamente utilizado em aplicações de Internet das Coisas, especialmente em cenários que envolvem dispositivos embarcados, redes sem fio e transmissão frequente de pequenos volumes de dados (MISHRA; KERTESZ, 2020).

2.2.3 SCADA

O SCADA é uma estrutura essencial para diversas infraestruturas críticas, desempenhando funções como monitoramento de dados de bombas, válvulas e transmissores em setores como o abastecimento de água, transporte e energia (ALANAZI; MAHMOOD; CHOWDHURY, 2023).

"Os sistemas SCADA modernos evoluíram de sistemas isolados para sistemas sofisticados, complexos e abertos conectados à Internet". (YADAV; PAUL, 2021)

Segundo Yadav e Paul (2021) *framework* SCADA é composto por:

- Unidade de Terminal Remoto (RTU)

- Unidade Terminal Mestre (MTU)
- Atuadores e sensores no *hardware*
- Interface Homem-Máquina (HMI)
- Banco de dados central
- *Software* para acesso do usuário

A RTU realiza a aquisição dos dados provenientes dos sensores e atuadores no *hardware* e envia essas informações para a MTU, que é responsável por transmitir o *status* atual dos dispositivos físicos conectados ao sistema. Além das RTUs, os CLPs e os *Intelligent Electronic Devices* (IEDs) também podem ser utilizados para comunicação de dados com os sensores e atuadores.

A MTU *(Master Terminal Unit)* é a unidade central do sistema SCADA, responsável pelo processamento e armazenamento das mensagens recebidas das RTUs, além de controlar e monitorar as máquinas e fornecer uma interface para o operador.

A HMI (Interface Homem-Máquina) constitui a interface de interação entre o operador e o sistema SCADA, permitindo o controle e monitoramento das informações operacionais.

Segundo Alanazi, Mahmood e Chowdhury (2023), na comunicação entre a RTU e a MTU, comumente é utilizado o protocolo MODBUS, o qual pode ser usado de duas formas:

- **Modbus serial**: Opera sobre as interfaces seriais RS-422 e RS-485, utilizando uma arquitetura mestre-escravo, na qual os PLCs ou a RTU se comportam como escravos da rede, e o dispositivo Modbus atua como mestre.
- **Modbus TCP/IP**: Utiliza uma arquitetura cliente-servidor, permitindo que múltiplos clientes se comuniquem com múltiplos servidores por meio da rede Ethernet para transmissão de dados.

Dessa forma, o sistema SCADA é um componente essencial para a automação, responsável por monitorar dados. A integração com diferentes protocolos Modbus, seja serial ou TCP/IP, permite uma ampla aplicabilidade, adaptando-se tanto a redes locais quanto a redes de longa distância.

2.2.4 API

API é a sigla para *Application Programming Interface*, traduzido para o português como Interface de Programação de Aplicações. De maneira geral, uma API pode ser definida como um conjunto de definições e protocolos que permitem a comunicação entre diferentes componentes de *software*.

Segundo Richardson e Ruby (2007), os serviços *web* fornecem uma interface programática para acesso a dados e funcionalidades, permitindo que sistemas distintos interajam de forma estruturada.

Dessa forma, as APIs promovem interoperabilidade, modularidade e integração entre sistemas, sendo amplamente utilizadas em arquiteturas distribuídas e aplicações baseadas em Internet das Coisas (IoT).

2.2.4.1 API REST

Entre os modelos mais utilizados atualmente destaca-se a API baseada no estilo arquitetural REST (*Representational State Transfer*). O termo REST foi definido por Roy Fielding

em sua tese de doutorado, na qual descreve um conjunto de restrições arquiteturais para sistemas distribuídos baseados na web (FIELDING, 2000).

Segundo Fielding (2000), o estilo REST é fundamentado em princípios como arquitetura cliente-servidor, comunicação sem estado (*stateless*), interface uniforme e utilização de recursos identificados por meio de URIs. A restrição de comunicação sem estado determina que cada requisição enviada pelo cliente ao servidor deve conter todas as informações necessárias para seu processamento, não dependendo de contexto armazenado no servidor.

As APIs REST utilizam predominantemente o protocolo HTTP como meio de comunicação e seus principais métodos são:

- **GET**: utilizado para solicitar a obtenção de um recurso;
- **POST**: utilizado para enviar dados e criar novos recursos;
- **PUT**: utilizado para atualizar um recurso existente;
- **DELETE**: utilizado para remover um recurso.

Esses métodos seguem o modelo de requisição e resposta (*request-response*), no qual o cliente envia uma requisição ao servidor e recebe uma resposta contendo os dados solicitados ou a confirmação da operação realizada.

No contexto de sistemas baseados em Internet das Coisas (IoT), as APIs REST são amplamente utilizadas para permitir que dispositivos embarcados se comuniquem com servidores remotos, enviando dados coletados por sensores e recebendo comandos de controle. Dessa forma, a API REST atua como camada intermediária entre o sistema embarcado, o backend e o sistema supervisório.

2.2.4.2 Webhook

Webhook é um mecanismo de integração baseado em eventos utilizado para comunicação entre aplicações e serviços *web*. Seu funcionamento permite que informações sejam transmitidas automaticamente quando determinados eventos ocorrem.

Segundo Mahadik *et al.* (2023, p. 2, tradução nossa), “Webhooks servem como um método para notificação de eventos por meio de um mecanismo de push”. Nesse modelo, um sistema envia automaticamente uma requisição HTTP para uma URL previamente configurada sempre que um evento específico é acionado.

Conforme destacado por Mahadik *et al.* (2023, p. 3, tradução nossa), “Diferentemente do modelo cliente-servidor convencional, em que atualizações são obtidas por requisições repetidas, os *webhooks* exigem uma mudança nesse paradigma”. Essa abordagem reduz a necessidade de requisições contínuas ao servidor, diminuindo o consumo de recursos computacionais e tornando a comunicação mais eficiente em aplicações orientadas a eventos.

Além disso, os *webhooks* podem atuar de forma complementar às APIs REST em arquiteturas distribuídas. Conforme apresentado por Ramadhan e Arifin (2023), a integração entre APIs REST e *webhooks* permite comunicação assíncrona entre serviços, favorecendo aplicações baseadas em microsserviços e sistemas que necessitam de atualização online.

2.2.5 Notificações *Push*

As notificações *push* constituem um mecanismo amplamente utilizado para envio automático de informações e alertas em dispositivos móveis. Diferentemente de abordagens baseadas em consultas periódicas, esse modelo permite que mensagens sejam encaminhadas diretamente ao usuário quando determinados eventos ocorrem.

Segundo Isikligil, Samakay e Kilinc (2017, p. 8, tradução nossa), “as notificações push notificam os usuários com novas informações sobre uma aplicação, fornecendo atualizações valiosas e relevantes mesmo quando a aplicação não está em execução”. Dessa forma, esse mecanismo possibilita comunicação online entre sistemas e usuários, sem a necessidade de acesso contínuo ao aplicativo.

De acordo com (ISIKLIGIL; SAMAKAY; KILINC, 2017), aplicações tradicionais frequentemente utilizam consultas periódicas (*polling*) para obtenção de novos dados. Em contraste, as notificações push permitem que a própria fonte de dados envie informações aos dispositivos clientes quando novas atualizações estiverem disponíveis, reduzindo o consumo de recursos computacionais e de rede.

Além disso, conforme descrito por Elyza (2023, p. 8, tradução nossa), as notificações *push* consistem em um serviço amplamente utilizado para envio de mensagens curtas em *smartphones*, mesmo quando o usuário não está acessando a aplicação. Segundo os resultados apresentados pelos autores, “as notificações aparecem automaticamente de acordo com os dados inseridos pelo usuário da aplicação” (ELYZA, 2023, p. 7, tradução nossa).

Em aplicações de Internet das Coisas (IoT), as notificações push podem ser utilizadas para informar eventos relevantes, estados de dispositivos e situações que exigem atenção imediata do usuário. Dessa forma, esse mecanismo complementa sistemas supervisórios e plataformas de monitoramento remoto, possibilitando comunicação rápida e eficiente entre o sistema e seus usuários.

Embora os *webhooks* e as notificações push sejam tecnologias distintas, ambas compartilham o mesmo princípio de comunicação orientada a eventos. Os *webhooks* atuam como mecanismos de integração entre sistemas, disparando requisições HTTP automaticamente quando um evento ocorre, enquanto as notificações push são voltadas para a entrega de informações diretamente ao usuário final em seus dispositivos. Dessa forma, é possível estabelecer uma relação complementar entre as duas abordagens: um *webhook* pode ser utilizado para acionar o envio de uma notificação *push*, garantindo que um evento registrado em um sistema interno seja imediatamente comunicado ao usuário.

3 METODOLOGIA

Este capítulo apresenta os procedimentos metodológicos adotados para o desenvolvimento do sistema proposto, bem como as tecnologias utilizadas e as etapas realizadas durante a implementação da solução.

3.1 Classificação da Pesquisa

Este trabalho caracteriza-se como uma pesquisa aplicada, pois tem como objetivo o desenvolvimento de uma arquitetura de monitoramento e controle de dispositivos utilizando conceitos de Internet das Coisas (IoT).

Quanto à abordagem, a pesquisa possui caráter qualitativo e experimental, envolvendo o desenvolvimento prático de um sistema embarcado integrado a serviços *web*.

Do ponto de vista dos procedimentos técnicos, o trabalho pode ser classificado como pesquisa experimental, uma vez que envolve testes, validações e integração entre *hardware*, *software* e *web* para avaliação do funcionamento do sistema desenvolvido.

3.1.1 Requisitos da Solução

Com o objetivo de delimitar o escopo do sistema desenvolvido e orientar o processo de validação, foram definidos requisitos funcionais e não funcionais relacionados às principais funcionalidades da arquitetura proposta. Esses requisitos estabeleceram as capacidades esperadas da solução e serviram como referência para a elaboração dos testes apresentados neste trabalho.

Os requisitos funcionais descrevem as funcionalidades que o sistema deve oferecer para atender aos objetivos propostos, enquanto os requisitos não funcionais definem características relacionadas à arquitetura, às tecnologias empregadas e à forma de operação da solução.

Tabela 3 – Requisitos funcionais da solução proposta.

Código	Descrição
RF01	O sistema deve monitorar o estado das entradas digitais do dispositivo embarcado.
RF02	O sistema deve permitir o acionamento remoto das saídas digitais por meio da interface supervisória.
RF03	O sistema deve transmitir os eventos entre os componentes do sistema utilizando os protocolos de comunicação definidos na arquitetura.
RF04	O sistema deve registrar os eventos em banco de dados para consulta posterior.
RF05	O sistema deve registrar os eventos localmente em cartão SD.
RF06	O sistema deve disponibilizar as informações para monitoramento por meio da interface supervisória web.

Fonte: Autoria própria.

Tabela 4 – Requisitos não funcionais da solução proposta.

Código	Descrição
RNF01	Utilizar o protocolo MQTT para a comunicação entre o servidor e o dispositivo embarcado.
RNF02	Utilizar o protocolo HTTP para o envio das informações ao backend e ao banco de dados.
RNF03	Permitir o armazenamento dos eventos tanto em banco de dados quanto em cartão SD.
RNF04	Disponibilizar uma interface supervisória web para monitoramento e controle remoto do sistema.

Fonte: Autoria própria.

3.2 Métodos Aplicados

O desenvolvimento deste trabalho foi realizado por meio de etapas que compreenderam o projeto da arquitetura, a implementação do *hardware* embarcado, o desenvolvimento do *backend*, da interface *web* e a integração entre esses componentes.

Inicialmente, foi realizada a configuração do ambiente de desenvolvimento e da infraestrutura necessária para o sistema, incluindo servidor *web*, banco de dados e serviços de comunicação. Essa etapa teve como finalidade atender ao objetivo específico de projetar e configurar o servidor e o banco de dados.

Posteriormente, foi desenvolvido um sistema *web* responsável pelo recebimento, armazenamento e visualização das informações provenientes do dispositivo embarcado. O sistema disponibilizou recursos para monitoramento, configuração e interação com os dispositivos conectados, atendendo ao objetivo de desenvolver e implementar uma plataforma para persistência e visualização dos dados.

A integração entre o sistema *web* e o *hardware* embarcado foi realizada utilizando o microcontrolador ESP32 como unidade principal de processamento. Foram implementados mecanismos para troca de informações entre o dispositivo e a aplicação supervisória, permitindo o envio de dados de monitoramento e o recebimento de comandos de controle e configuração.

Por fim, foi realizada a integração completa entre os componentes desenvolvidos, possibilitando a comunicação entre *hardware*, servidor, banco de dados e interface *web*, atendendo ao objetivo de integrar o sistema de monitoramento ao dispositivo de aquisição de dados.

3.3 Procedimentos de Teste

Para validar o sistema, foram planejados e executados testes de funcionalidade com o objetivo de verificar o atendimento aos requisitos funcionais apresentados na Seção 3.1.1, bem como avaliar a integração entre os componentes da arquitetura proposta.

Inicialmente, foram realizados testes de monitoramento das entradas digitais do dispositivo embarcado, verificando a correta detecção e transmissão das mudanças de estado para os demais componentes da solução.

Também foram executados testes de controle remoto das saídas digitais para verificar o atendimento ao requisito RF02, avaliando a comunicação entre a interface *web*, o *broker* MQTT, o dispositivo embarcado e o backend durante o acionamento remoto das saídas.

Para validar os requisitos RF04 e RF05, foram realizados testes para verificar o registro dos eventos no banco de dados e no cartão SD, avaliando a integridade das informações armazenadas e a correspondência entre os eventos gerados pelo sistema e os registros persistidos.

Por fim, foi realizado um conjunto de testes integrados e de estresse em ambiente controlado, simulando múltiplos eventos de controle. Nessa etapa, foram observados aspectos como a estabilidade da comunicação, a integridade dos dados registrados, o tempo de resposta e o comportamento geral da solução.

4 DESENVOLVIMENTO

Este capítulo apresenta o desenvolvimento da solução proposta, descrevendo os componentes implementados e a integração entre *hardware*, *firmware*, *backend* e interface supervisória.

Inicialmente é apresentada a arquitetura geral do sistema, seguida pela descrição do *hardware* utilizado e do *firmware* embarcado desenvolvido para o microcontrolador ESP32. Em seguida são detalhados os componentes do *backend* responsáveis pelo armazenamento e processamento das informações, bem como a interface supervisória utilizada para monitoramento e controle dos dispositivos. Por fim, são apresentadas as ferramentas desenvolvidas para validação da solução.

4.0.1 Disponibilização do código-fonte

Com o objetivo de possibilitar a reprodução dos resultados obtidos e facilitar futuras evoluções da solução proposta, os códigos-fonte desenvolvidos neste trabalho foram disponibilizados publicamente por meio da plataforma GitHub.

Devido à separação entre os diferentes componentes do sistema, os códigos foram organizados em dois repositórios independentes. O primeiro contém o *firmware* embarcado desenvolvido para o ESP32, incluindo as rotinas de configuração das GPIOs, comunicação MQTT, armazenamento em cartão SD e gerenciamento dos dispositivos. O segundo reúne os componentes do *backend* e da interface *web*, contemplando a API REST desenvolvida em PHP, os *scripts* responsáveis pela comunicação com o banco de dados e as páginas utilizadas para monitoramento e controle dos dispositivos.

Os repositórios podem ser acessados por meio dos seguintes endereços:

Firmware: https://github.com/StephanieSouzaa/Tcc_Project_Firmware

Backend e Interface *web*: https://github.com/StephanieSouzaa/Tcc_Project_Backend

4.0.2 Arquitetura do sistema

A Figura 1 apresenta uma visão geral da arquitetura desenvolvida neste trabalho. A solução é composta por uma interface *web*, um servidor responsável pelo processamento e armazenamento dos dados, um *broker* MQTT utilizado para comunicação assíncrona e um dispositivo embarcado baseado no microcontrolador ESP32.

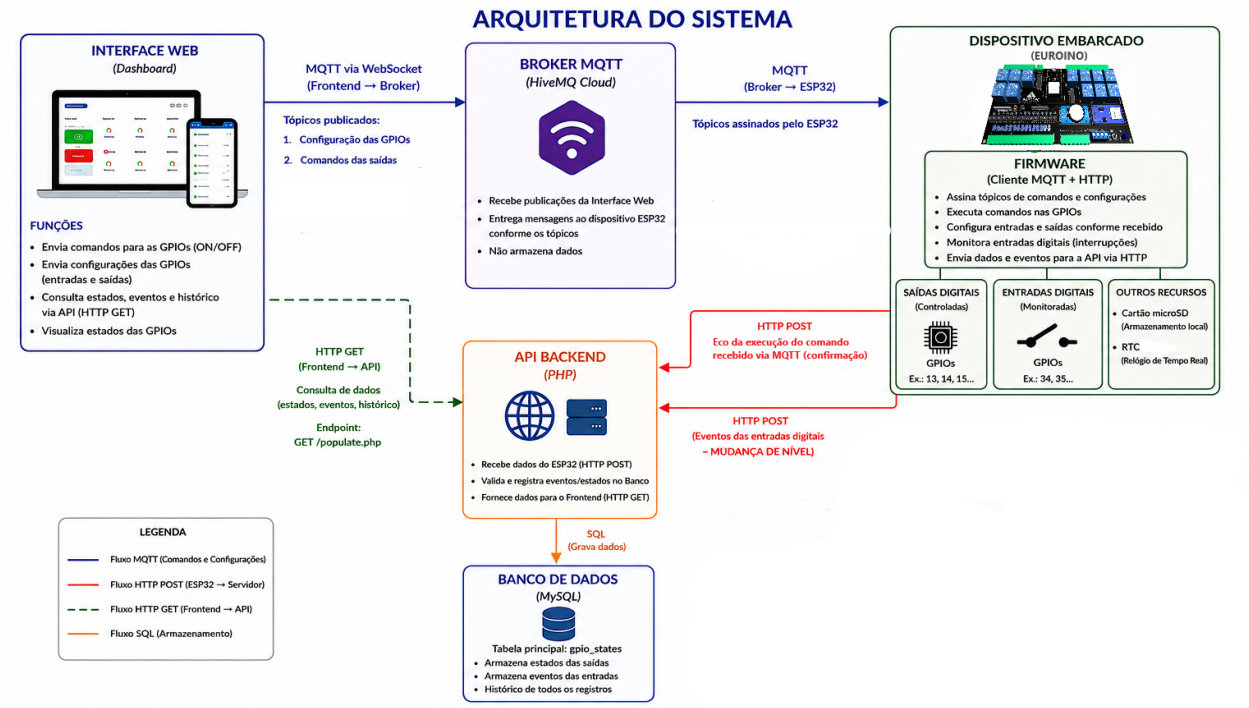
A interface *web* permite o envio de comandos e configurações ao dispositivo embarcado por meio do protocolo MQTT, utilizando o modelo *publish/subscribe* intermediado pelo *broker*. O ESP32 permanece inscrito nos tópicos de interesse e, ao receber uma mensagem válida, executa a ação correspondente, como a alteração da configuração de uma GPIO ou o acionamento de uma saída digital.

Após a execução de um comando recebido via MQTT, o *firmware* envia uma requisição HTTP ao servidor contendo o resultado da operação realizada. Esse envio atua como um eco de execução do comando, confirmando que a mensagem MQTT foi recebida, processada e aplicada corretamente pelo dispositivo. Dessa forma, o sistema não considera apenas o envio do comando pela interface *web*, mas também sua efetiva execução no dispositivo embarcado.

Caso ocorra alguma falha durante o processamento do comando, como a tentativa de acionamento de uma GPIO não configurada ou o não recebimento da mensagem MQTT pelo dispositivo, o eco HTTP correspondente não será gerado. Esse mecanismo permite que o sistema supervisório utilize o retorno HTTP como uma confirmação adicional da execução da operação solicitada, aumentando a confiabilidade do processo de controle remoto.

Os eventos gerados pelo sistema são armazenados em banco de dados para consulta remota e também registrados localmente em cartão SD, possibilitando rastreabilidade e persistência das informações. A arquitetura proposta foi projetada de forma modular, permitindo a integração entre os diferentes componentes responsáveis pelo monitoramento, controle e armazenamento dos dados.

Figura 1 – Arquitetura geral do sistema desenvolvido



Fonte: Autoria própria

4.1 *Hardware*

O *hardware* utilizado neste trabalho é denominado comercialmente Euroino, uma placa desenvolvida pela equipe de tecnologia da empresa Eurotec, da qual a autora participou durante seu período de estágio. O dispositivo foi projetado para aplicações de automação e Internet das Coisas (IoT), permitindo o monitoramento de entradas digitais, o acionamento de saídas e a comunicação com sistemas supervisórios por meio de conectividade de rede.

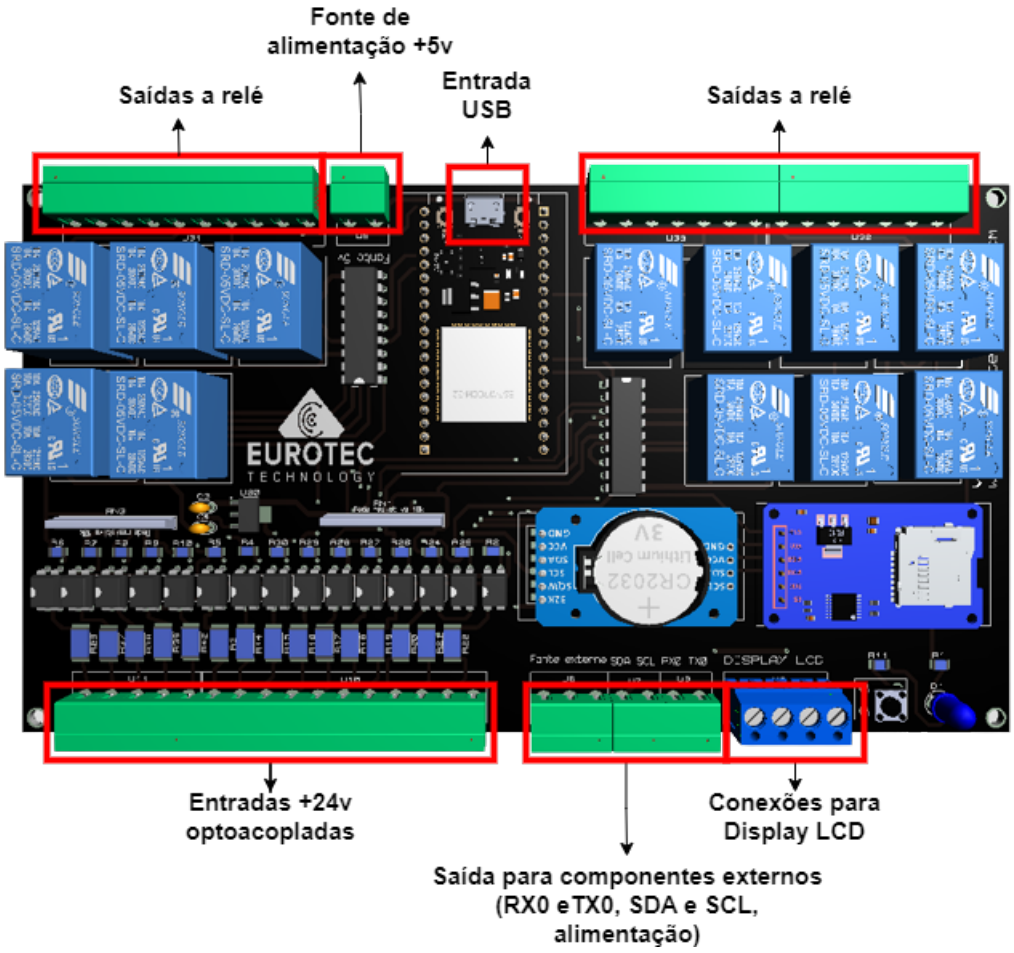
O dispositivo é baseado no microcontrolador ESP32, desenvolvido pela empresa *Espressif Systems*, escolhido devido à sua capacidade de processamento, baixo consumo energético e conectividade Wi-Fi integrada. O *hardware* disponibiliza 15 GPIOs configuráveis para utilização no sistema, possibilitando a implementação de funcionalidades de monitoramento e controle remoto de dispositivos de campo.

Além dos recursos de entrada e saída digital, a placa possui um sistema dedicado ao registro local de eventos (*logger*), composto por um módulo para cartão microSD. Essa funcionalidade permite manter um histórico de eventos mesmo em situações de indisponibilidade da rede ou do servidor.

O *hardware* também disponibiliza uma interface de comunicação I²C para expansão de funcionalidades, possibilitando a integração com dispositivos externos, como displays LCD, sensores ambientais e outros módulos compatíveis com esse protocolo de comunicação.

A Figura 2 apresenta a placa Euroino utilizada durante o desenvolvimento deste projeto.

Figura 2 – Placa Euroino utilizada no desenvolvimento do projeto



Fonte: Autoria própria

As principais características do *hardware* utilizado neste trabalho são apresentadas na Tabela 5.

Tabela 5 – Principais características do hardware Euroino

Característica	**Descrição**
Microcontrolador	ESP32
Conectividade	Wi-Fi 2,4 GHz
GPIOs disponíveis	15
Entradas digitais	15
Saídas digitais	12
Armazenamento local	Cartão microSD
Relógio de tempo real	RTC externo
Interface de expansão	I²C, Serial (RX e TX)
Alimentação	5 Vcc

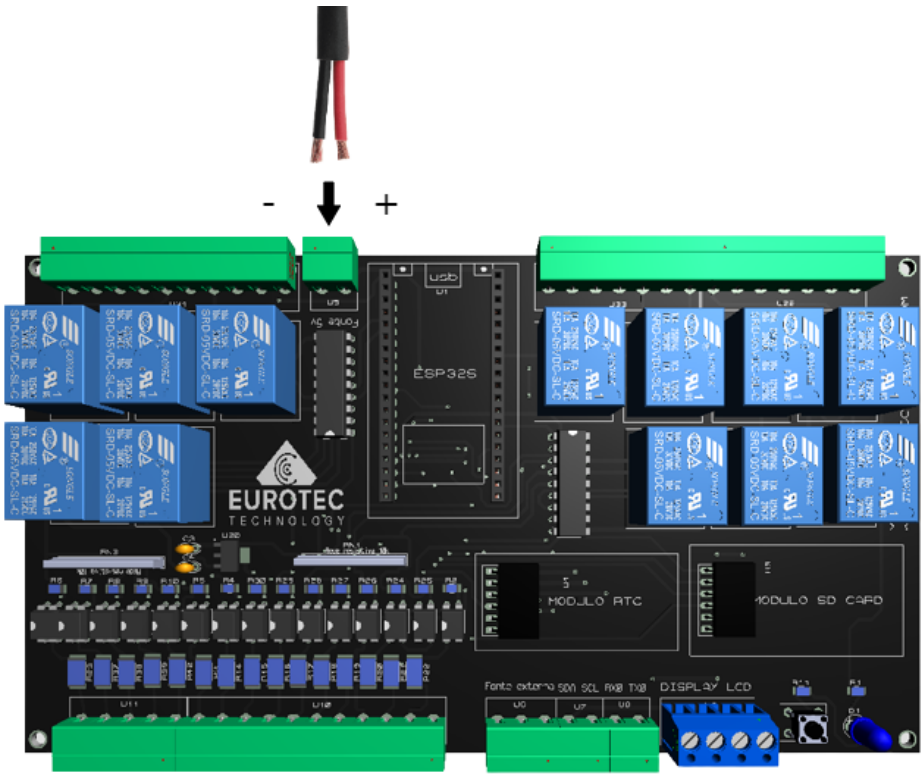
Fonte: Autoria própria

4.1.1 Alimentação do Sistema

A alimentação do Euroino é realizada através de uma fonte de tensão contínua de 5 V conectada ao borne de alimentação da placa.

A Figura 3 ilustra a conexão da alimentação utilizada durante os testes e validações do sistema.

Figura 3 – Alimentação do Euroino



Fonte: Autoria própria

4.1.2 Entradas e Saídas Digitais

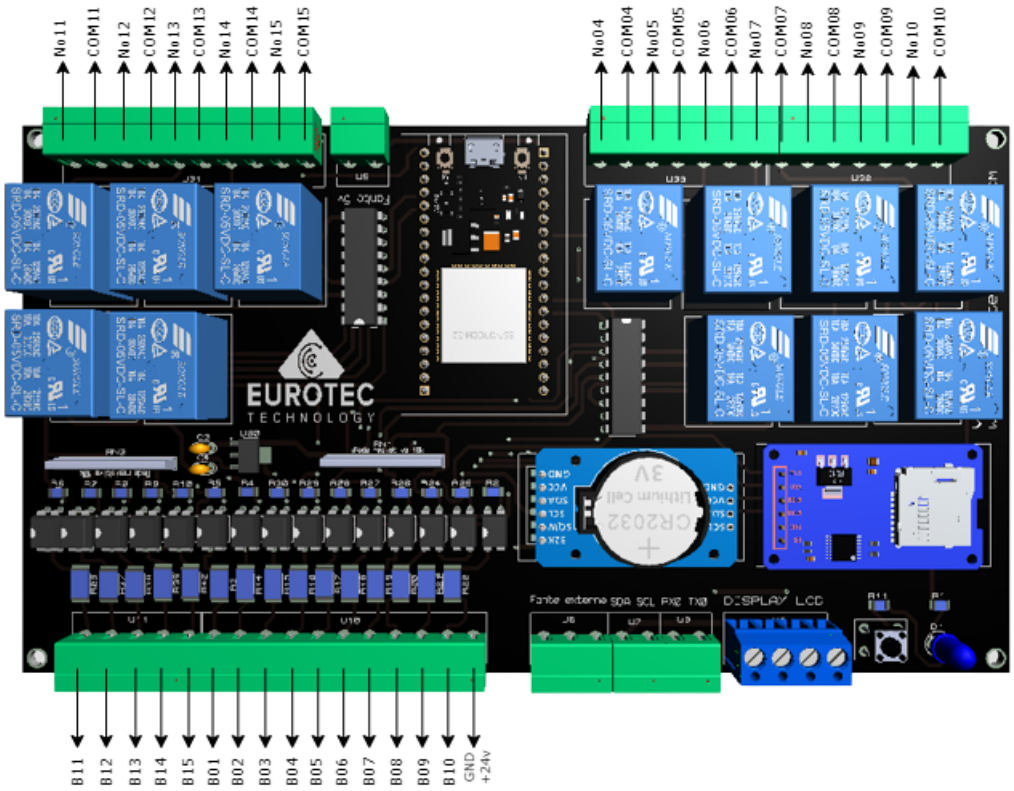
O Euroino possui 15 GPIOs configuráveis para monitoramento e controle. Todas as 15 GPIOs podem ser utilizadas como entradas digitais, enquanto 12 delas também podem ser configuradas como saídas digitais. As três GPIOs restantes possuem funcionamento exclusivo como entrada.

O mapeamento das entradas e saídas disponíveis pode ser observado na Figura 4. Os terminais identificados pela letra B, numerados de 1 a 15, correspondem às entradas digitais optoacopladas, proporcionando isolamento elétrico entre os dispositivos de campo e o circuito eletrônico da placa.

Os terminais identificados como NO (*Normally Open*) e COM (*Common*), numerados de 4 a 15, correspondem às saídas controladas por relés eletromecânicos. Nessa configuração, o terminal NO representa o contato normalmente aberto do relé, enquanto o terminal COM representa o contato comum.

É importante destacar que uma mesma GPIO não pode operar simultaneamente como entrada e saída. Durante a configuração do sistema, cada GPIO deve ser definida exclusivamente para uma das funções, de acordo com a necessidade da aplicação.

Figura 4 – Mapeamento das entradas e saídas do Euroino



Fonte: Autoria própria

4.1.3 Módulo de Cartão microSD

Para garantir o armazenamento local dos eventos gerados pelo sistema, foi utilizado um módulo leitor de cartão microSD conectado ao ESP32 por meio da interface SPI.

O módulo permite registrar informações mesmo em situações de indisponibilidade da rede ou do servidor, aumentando a confiabilidade da solução. Os eventos armazenados incluem alterações de estado das entradas e saídas, bem como registros relacionados à comunicação do sistema.

4.1.4 Relógio de Tempo Real (RTC)

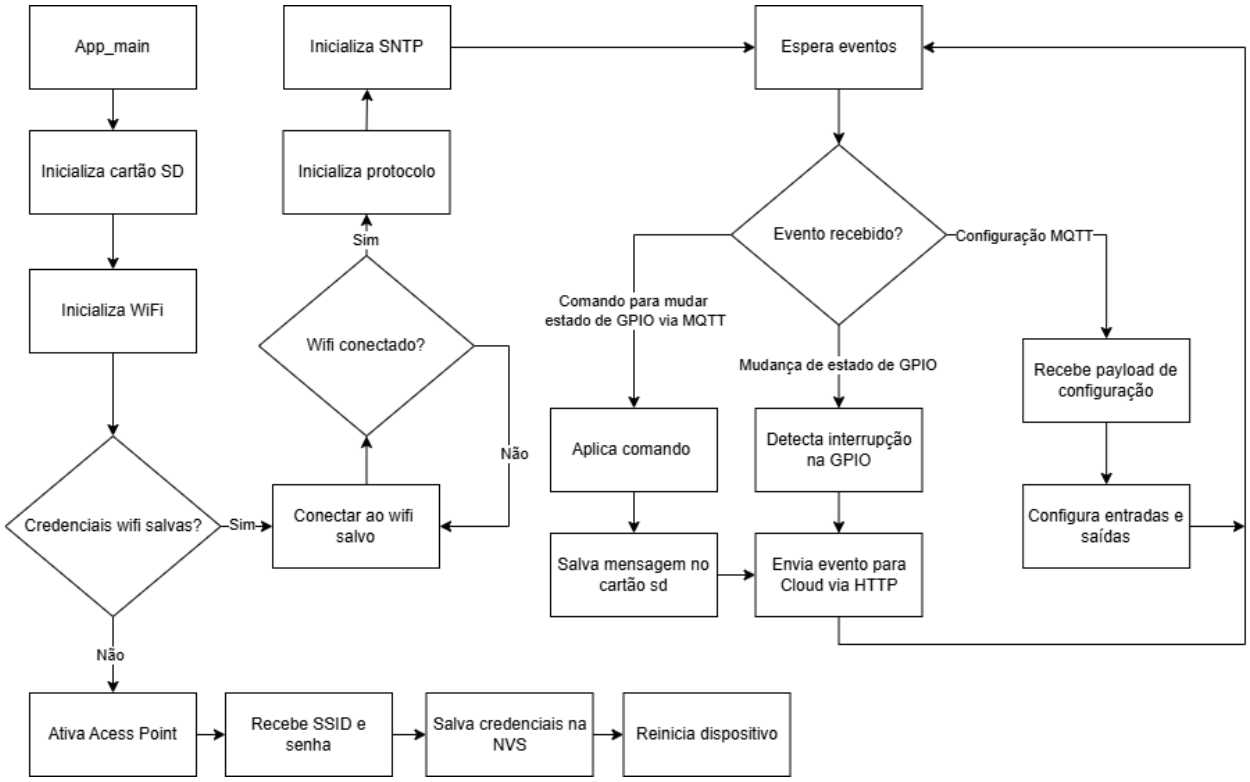
O Euroino possui um módulo RTC *(Real Time Clock)* responsável pela manutenção das informações de data e horário do sistema, mesmo quando o dispositivo é reiniciado. Esse recurso é fundamental para a funcionalidade de registro de eventos implementada neste trabalho, permitindo a correta identificação temporal das ocorrências registradas, inclusive em situações de indisponibilidade da rede.

Os recursos de *hardware* apresentados nesta seção fornecem a infraestrutura necessária para a implementação da solução proposta, permitindo o monitoramento de entradas, o acionamento de saídas, a comunicação em rede e o armazenamento local de eventos. A próxima seção descreve o desenvolvimento do *firmware* embarcado responsável pela operação desses recursos.

4.2 *Firmware* embarcado

O *firmware* embarcado foi desenvolvido utilizando a linguagem C e o *framework* ESP-IDF, disponibilizado pela *Espressif Systems* para o desenvolvimento de aplicações baseadas no microcontrolador ESP32. A arquitetura do *firmware* pode ser observada em forma de fluxograma na imagem 5 [1]

Figura 5 – Fluxograma *Firmware*



Fonte: Autoria própria

A arquitetura do *firmware* foi estruturada de forma modular e baseada no sistema operacional de tempo real FreeRTOS, disponibilizado pelo *framework* ESP-IDF, visando facilitar a organização do código, a manutenção do sistema e a implementação de novas funcionalidades. Os módulos desenvolvidos são responsáveis pelo gerenciamento das entradas e saídas digitais, comunicação com serviços externos, armazenamento local de eventos e integração entre os diferentes componentes da aplicação.

Os principais módulos implementados no *firmware* são apresentados a seguir:

- ***Main***: responsável pela inicialização do sistema, configuração dos recursos necessários para operação e criação das tarefas executadas pelo *firmware*.
- ***Protocols***: conjunto de módulos responsáveis pela comunicação entre o dispositivo embarcado e os demais componentes da arquitetura.
  - **MQTT**: responsável pelo recebimento das configurações das GPIOs e dos comandos de acionamento das saídas digitais.

[1] O código-fonte completo do *firmware* desenvolvido neste trabalho encontra-se disponível na seção 4.0.1

  - **HTTP**: responsável pelo envio dos estados das GPIOs e dos eventos gerados pelo sistema para o servidor remoto.

- ***Drivers***: conjunto de módulos responsáveis pela interação direta com os recursos de *hardware*.
  - **GPIO**: responsável pela configuração, monitoramento e acionamento das entradas e saídas digitais.
  - **SD Card**: responsável pelo armazenamento local dos registros e eventos gerados durante a operação do sistema.

- **Interfaces**: módulos responsáveis pela comunicação com recursos externos.
  - **Wi-Fi**: responsável pela conexão do dispositivo à rede sem fio utilizada para comunicação com o *broker* MQTT e com o servidor *web*.

Durante a inicialização do dispositivo são realizados os procedimentos de configuração da rede Wi-Fi, sincronização do relógio de tempo real, inicialização do cartão SD, configuração das GPIOs e estabelecimento da comunicação com o *broker* MQTT. Após a conclusão dessas etapas, o sistema passa a operar continuamente monitorando alterações de estado nas entradas digitais, processando comandos recebidos remotamente e registrando os eventos gerados durante sua operação.

As configurações das GPIOs são recebidas dinamicamente por meio do tópico MQTT enviados pela interface *web*. A partir dessas configurações, o sistema define quais pinos devem operar como entradas e quais devem operar como saídas. Quando uma alteração de estado é detectada em uma entrada digital, o evento é registrado localmente no cartão SD e enviado ao servidor por meio de uma requisição HTTP.

De forma semelhante, os comandos destinados ao acionamento das saídas digitais são recebidos via MQTT e processados pelo *firmware*. Após a execução do comando, o evento é registrado no cartão SD e uma requisição HTTP é enviada ao servidor para atualização do banco de dados. Esse mecanismo permite registrar não apenas o comando recebido, mas também a efetiva execução da ação solicitada, garantindo maior confiabilidade e rastreabilidade das operações realizadas pelo sistema.

4.3 *Backend*

O *backend* foi desenvolvido com a finalidade de centralizar a comunicação entre os dispositivos embarcados, o sistema supervisório e o banco de dados. Sua principal responsabilidade consiste em receber informações provenientes do *firmware*, armazenar eventos, disponibilizar dados para a interface *web* e realizar a integração com o *broker* MQTT utilizado para o controle dos dispositivos.

4.3.1 Arquitetura do *backend*

A arquitetura implementada foi baseada em serviços *web* hospedados em servidor PHP remoto, permitindo o acesso às funcionalidades do sistema por meio da internet. Dessa forma, os dispositivos embarcados podem transmitir informações de estado e eventos independentemente da rede local em que estejam conectados.

A comunicação entre os dispositivos foi implementada utilizando dois protocolos em situações distintas. O MQTT foi adotado no modelo *publish/subscribe* para envio de mensagens entre o sistema embarcado e a aplicação supervisória, enquanto o HTTP foi utilizado para a publicação dos dados para o uso no sistema *web*.

A escolha por utilizar simultaneamente os protocolos HTTP e MQTT decorreu de necessidades distintas do sistema. Inicialmente, o sistema já operava utilizando HTTP, o que facilitava a integração com serviços *web* e o armazenamento de dados. Contudo, para funções de controle online, o uso exclusivo desse protocolo seria pouco eficiente, pois exigiria a implementação de consultas periódicas (*polling*), aumentando a latência e o custo de processamento do microcontrolador.

Diante disso, optou-se por uma arquitetura híbrida. O protocolo MQTT foi empregado para o controle e a configuração dos dispositivos, aproveitando seu modelo *publish/subscribe*, que proporciona comunicação assíncrona e online. Já o HTTP permaneceu responsável pela publicação dos estados das GPIOs configuradas como entradas e pelo envio das confirmações dos comandos recebidos via MQTT. Essa solução permitiu equilibrar a eficiência necessária para o controle online com a simplicidade do armazenamento e da persistência dos dados.

4.3.2 Hospedagem

Para disponibilizar o sistema em ambiente externo, a API REST desenvolvida em PHP e o banco de dados MySQL foram hospedados em um servidor compartilhado da HostGator. A plataforma fornece suporte à execução de aplicações PHP e banco de dados MySQL, recursos necessários para a implementação da solução proposta.

A utilização de um servidor remoto permitiu centralizar as informações coletadas pelos dispositivos, possibilitando o acesso ao sistema por diferentes usuários através da internet. Dessa forma, tanto a interface supervisória quanto os dispositivos embarcados podem se comunicar com os serviços do *backend* independentemente da rede local em que estejam conectados.

Além da hospedagem da aplicação *web*, o ambiente disponibilizado pela HostGator foi utilizado para execução da API REST responsável pelo processamento das requisições HTTP e pela comunicação com o banco de dados.

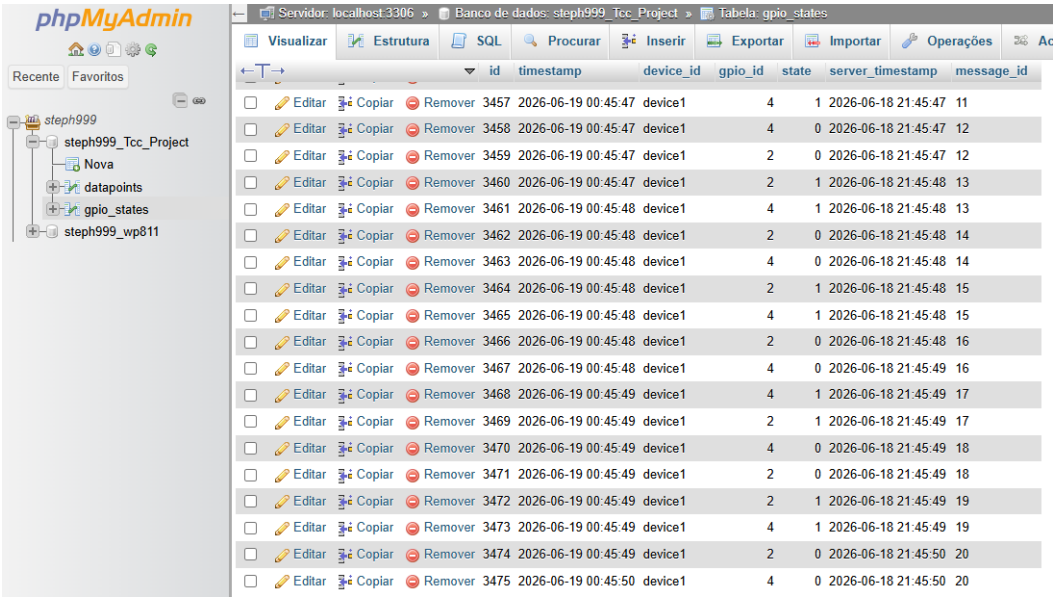
4.3.3 Banco de dados

O armazenamento persistente das informações do sistema foi implementado utilizando o sistema gerenciador de banco de dados MySQL. O banco de dados é responsável por armazenar informações relacionadas aos dispositivos cadastrados, configurações das GPIOs, estados das entradas e saídas e registros de eventos gerados durante a operação do sistema.

As informações recebidas pelo *backend* são armazenadas em tabelas específicas, permitindo consultas históricas, rastreabilidade dos eventos e disponibilização dos dados para a interface supervisória. Essa abordagem possibilita acompanhar alterações de estado ocorridas nos dispositivos e manter um histórico das operações executadas.

A Figura 6 apresenta registros armazenados na tabela responsável pelo armazenamento dos estados das GPIOs monitoradas pelo sistema.

Figura 6 – Registros armazenados na tabela de estados das GPIOs



Fonte: Autoria própria

Entre as informações registradas encontram-se o identificador do dispositivo, a GPIO monitorada, o estado detectado e o instante em que o evento foi processado pelo servidor. Esses registros são utilizados tanto para monitoramento online quanto para consultas históricas através da interface supervisória.

4.3.4 API REST

Para o armazenamento e gerenciamento das informações, foi desenvolvida uma API REST utilizando a linguagem PHP, responsável por intermediar a comunicação entre o sistema embarcado e o banco de dados MySQL hospedado em servidor remoto.

A API foi estruturada para receber dados enviados pelo *firmware* por meio de requisições HTTP e disponibilizar consultas às informações armazenadas. Dessa forma, os dispositivos embarcados podem registrar eventos e estados das GPIOs, enquanto a interface supervisória pode recuperar essas informações para exibição ao usuário.

Entre os *endpoint* desenvolvidos para a API REST, destaca-se o *populate.php*, responsável pelo recebimento e armazenamento dos estados das GPIOs monitoradas pelo sistema. Por meio de requisições HTTP do tipo POST, o *firmware* envia informações como identificador do dispositivo, GPIO monitorada, estado detectado e instante de ocorrência do evento. Essas informações são então processadas e armazenadas no banco de dados.

O código-fonte completo desse *endpoint* encontra-se disponível no repositório do *backend* apresentado na Seção 4.0.1, dentro do diretório *api*.

Conforme apresentado na Figura 7, a API utiliza consultas preparadas (*prepared statements*) para realizar a inserção dos dados recebidos.

```
$stmt = $conn->prepare(
"INSERT INTO gpio_states
(timestamp, device_id, gpio_id, state, message_id)
VALUES (?, ?, ?, ?, ?)"
);
```

Figura 7 – Trecho do *endpoint* populate.php responsável pela inserção dos estados das GPIOs.

A API utiliza consultas preparadas (*prepared statements*) para realizar a inserção dos dados recebidos. Esse mecanismo permite associar os valores recebidos na requisição HTTP aos parâmetros da instrução SQL durante sua execução, contribuindo para a integridade dos dados armazenados.

Além das operações de inserção realizadas por meio de requisições HTTP do tipo *POST*, o *endpoint* também disponibiliza consultas HTTP do tipo *GET*. Essa funcionalidade permite recuperar o estado mais recente associado a uma determinada GPIO e dispositivo, possibilitando que a interface *web* obtenha informações atualizadas para exibição ao usuário.

4.3.5 *Broker* MQTT

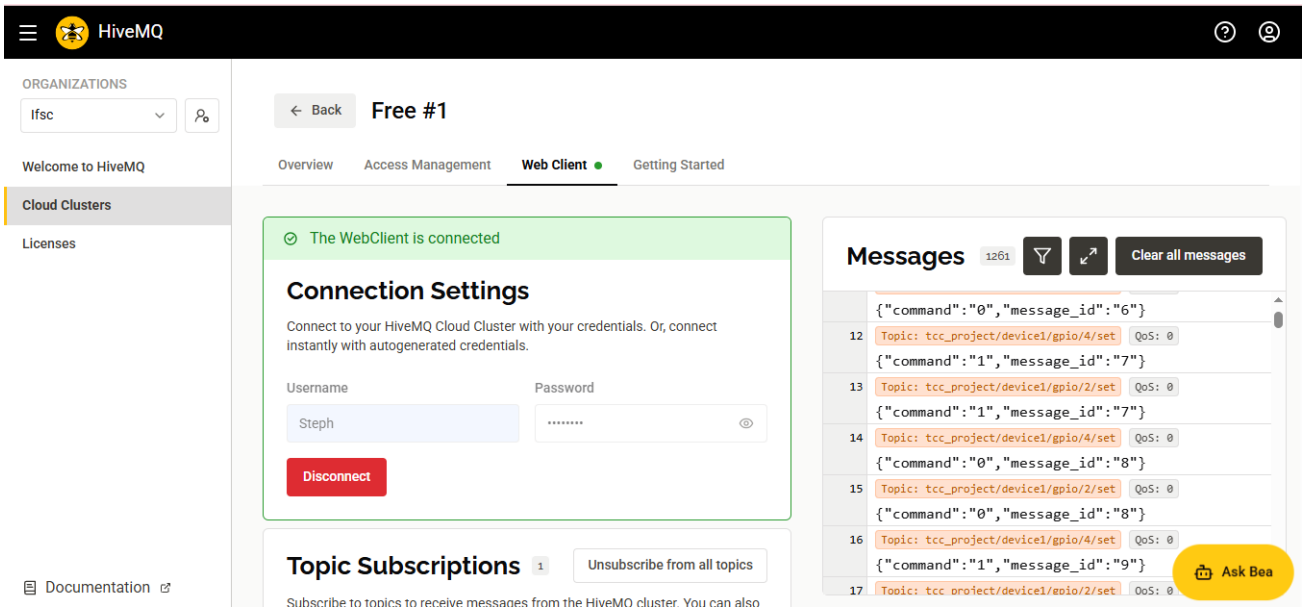
A implementação da comunicação MQTT requer a utilização de um *broker* responsável por receber as mensagens publicadas pelos clientes e distribuí-las aos clientes inscritos nos respectivos tópicos. Durante o desenvolvimento deste trabalho, verificou-se que o ambiente de hospedagem compartilhada utilizado para execução da aplicação *web* não disponibilizava suporte para a instalação e execução de serviços MQTT dedicados, limitando a utilização de um *broker* próprio.

Diante dessa restrição, optou-se pela utilização de um *broker* MQTT externo disponibilizado pela plataforma HiveMQ. Essa solução permitiu estabelecer a comunicação entre a interface supervisória e os dispositivos embarcados sem a necessidade de infraestrutura adicional ou configuração de servidores dedicados.

O *broker* atua como elemento central do modelo *publish/subscribe*, recebendo mensagens publicadas por clientes e distribuindo-as aos dispositivos inscritos nos respectivos tópicos. Dessa forma, comandos enviados pela interface *web* podem ser encaminhados ao *hardware*, enquanto informações geradas pelo dispositivo podem ser disponibilizadas para outros clientes conectados.

A Figura 8 apresenta a interface de gerenciamento do *broker* MQTT utilizado durante o desenvolvimento e validação do sistema.

Figura 8 – Interface de gerenciamento do broker MQTT HiveMQ



Fonte: Autoria própria

A plataforma HiveMQ forneceu os recursos necessários para gerenciamento dos tópicos MQTT, monitoramento das conexões e validação das mensagens trafegadas entre os componentes do sistema durante os testes realizados.

Para permitir a comunicação direta entre a interface *web* e o *broker* MQTT, foi utilizada a funcionalidade MQTT sobre *WebSocket*, possibilitando que o navegador estabelecesse conexão com o *broker* sem necessidade de serviços intermediários adicionais.

4.4 Interface Supervisória

A interface supervisória foi desenvolvida com o objetivo de permitir a configuração, o monitoramento e o controle remoto dos dispositivos embarcados conectados ao sistema, possibilitando seu acesso diretamente por meio de navegadores sem a necessidade de instalação de *softwares* adicionais.

Durante o desenvolvimento deste trabalho foram disponibilizadas duas interfaces *web*. A primeira corresponde ao sistema supervisório utilizado para configuração, monitoramento e controle dos dispositivos, enquanto a segunda foi desenvolvida especificamente para execução dos testes de estresse realizados na etapa de validação do sistema.

As interfaces podem ser acessadas pelos seguintes endereços:

Sistema supervisório: http://br76.teste.website/~steph999/gpio_view.html

Interface de testes: http://br76.teste.website/~steph999/stress_test.html

4.4.1 Tecnologias utilizadas

A interface supervisória foi desenvolvida utilizando as tecnologias HTML, CSS e JavaScript, permitindo sua execução diretamente em navegadores *web* modernos sem necessidade de instalação de aplicações adicionais.

O HTML foi utilizado para a estruturação dos elementos visuais da interface, incluindo formulários de configuração, painéis de monitoramento e controles de acionamento das

GPIOs. O CSS foi empregado para definição da aparência visual dos componentes, enquanto o JavaScript foi responsável pela implementação da lógica de funcionamento da aplicação.

Para a comunicação MQTT foi utilizada a biblioteca MQTT.js, permitindo que a interface estabelecesse conexão direta com o *broker* MQTT através do protocolo *WebSocket*. Essa abordagem possibilitou o envio de comandos e configurações diretamente do navegador para os dispositivos embarcados.

Além da comunicação MQTT, a interface também utiliza requisições HTTP para consulta dos estados armazenados no servidor, permitindo a atualização periódica das informações apresentadas ao usuário.

4.4.2 Interface de monitoramento e controle

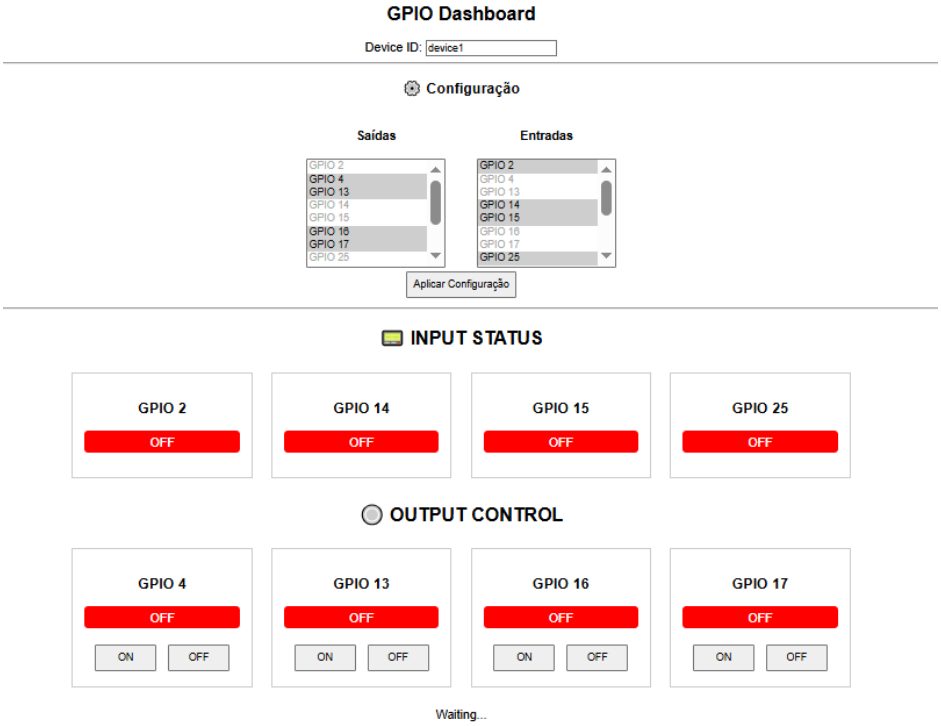
A interface de monitoramento e controle foi desenvolvida para concentrar em uma única página as funcionalidades necessárias para configuração, supervisão e acionamento dos dispositivos conectados ao sistema. Por meio dessa interface, o usuário pode definir a configuração das GPIOs, acompanhar o estado das entradas monitoradas e controlar remotamente as saídas digitais disponibilizadas pelo *hardware*.

A aplicação foi projetada para operar de forma dinâmica, permitindo que as configurações enviadas ao dispositivo sejam refletidas automaticamente nos componentes exibidos ao usuário. Dessa forma, a interface apresenta apenas os elementos correspondentes às GPIOs configuradas para cada função, tornando a operação mais intuitiva e reduzindo a possibilidade de configurações incorretas.

A comunicação com os dispositivos ocorre por meio da integração entre os protocolos MQTT e HTTP. O MQTT é utilizado para envio das configurações e dos comandos de acionamento das saídas, enquanto as informações de estado são recuperadas periodicamente do servidor através da API REST desenvolvida para o sistema.

A Figura 9 apresenta a interface utilizada durante o desenvolvimento e validação da solução proposta.

Figura 9 – Interface supervisória desenvolvida para monitoramento e controle dos dispositivos.



Fonte: Autoria própria

4.4.2.1 Configuração das GPIOs

A configuração das GPIOs é realizada diretamente pela interface supervisória. O usuário pode selecionar quais pinos devem operar como entradas e quais devem operar como saídas por meio do painel de configuração disponibilizado na aplicação.

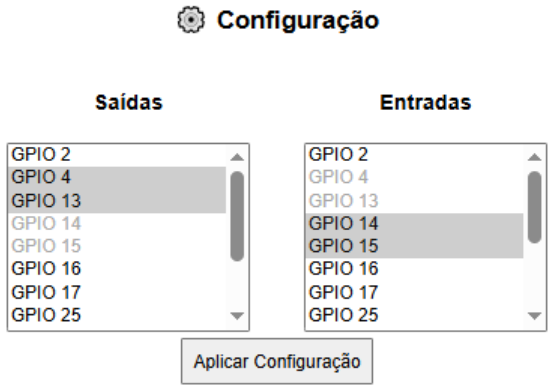
Durante o processo de configuração, o sistema impede que uma mesma GPIO seja selecionada simultaneamente para ambas as funções, evitando configurações inválidas. Além disso, os pinos que possuem funcionamento exclusivo como entrada permanecem indisponíveis na lista de saídas.

Após a seleção dos pinos desejados, o usuário pode aplicar a configuração por meio do botão correspondente. Nesse momento, a interface gera uma mensagem contendo as definições selecionadas e a publica em um tópico MQTT específico do dispositivo. Ao receber essa mensagem, o *firmware* aplica imediatamente a nova configuração das GPIOs durante a execução da aplicação, sem necessidade de regravação do *firmware* ou reinicialização do dispositivo.

Atualmente, as configurações das GPIOs não são armazenadas de forma persistente no dispositivo. Dessa forma, em situações de reinicialização ou desligamento do *hardware*, é necessário realizar novamente o envio das configurações por meio da interface supervisória para que as entradas e saídas sejam redefinidas. A implementação de um mecanismo de persistência dessas configurações poderá ser considerada em trabalhos futuros.

A Figura 10 destaca a área destinada à configuração das GPIOs.

Figura 10 – Área da interface destinada à configuração das GPIOs



Fonte: Autoria própria

4.4.2.2 Monitoramento dos estados

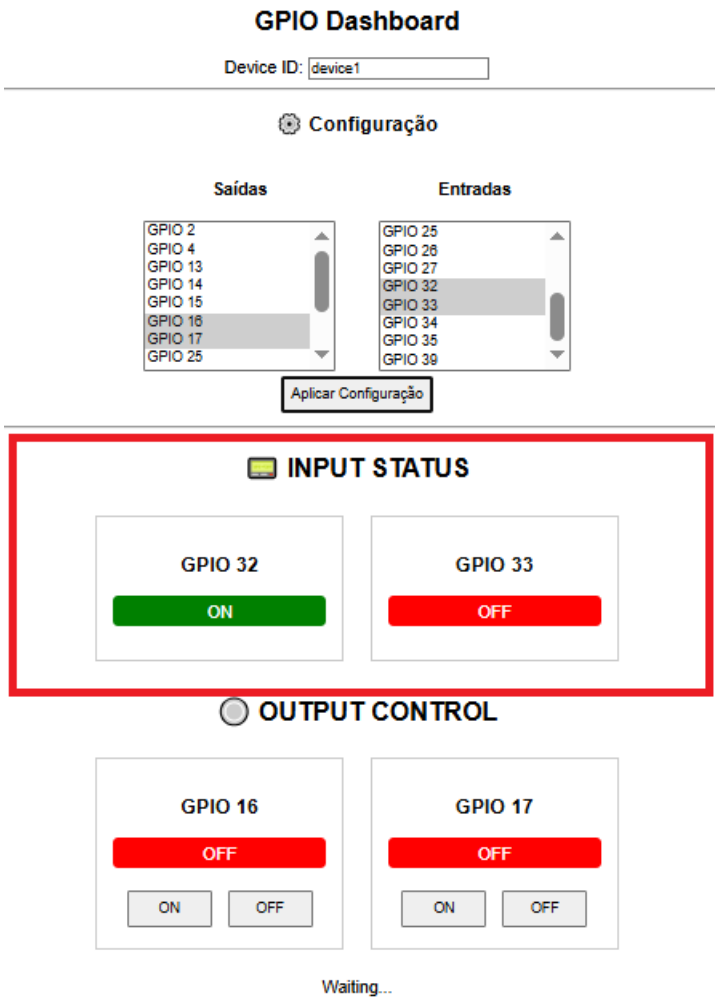
A interface supervisória disponibiliza uma área dedicada ao acompanhamento das GPIOs configuradas como entradas digitais. Para cada entrada monitorada, é exibido um indicador contendo a identificação da GPIO e seu estado atual.

Quando o *firmware* detecta uma alteração em uma entrada digital, uma requisição HTTP é enviada ao servidor contendo as informações do evento. A interface realiza consultas periódicas ao sistema e atualiza automaticamente os indicadores exibidos ao usuário com base nos dados mais recentes disponíveis.

Os estados são representados visualmente por meio de cores e textos indicativos. Quando uma entrada se encontra desativada, o indicador é apresentado na cor vermelha com a indicação *OFF*. Após a detecção de um acionamento pelo dispositivo embarcado, o estado correspondente passa a ser exibido na cor verde com a indicação *ON*, permitindo a rápida identificação das entradas ativas.

A Figura 11 apresenta a área da interface utilizada para monitoramento dos estados das entradas digitais.

Figura 11 – Monitoramento dos estados das entradas digitais



Fonte: Autoria própria

4.4.2.3 Controle das saídas digitais

O controle das saídas digitais é realizado por meio da área *Output Control* da interface *web*. Nessa seção são exibidas apenas as GPIOs previamente configuradas como saídas, permitindo que o usuário acione ou desligue remotamente os dispositivos conectados ao *hardware*.

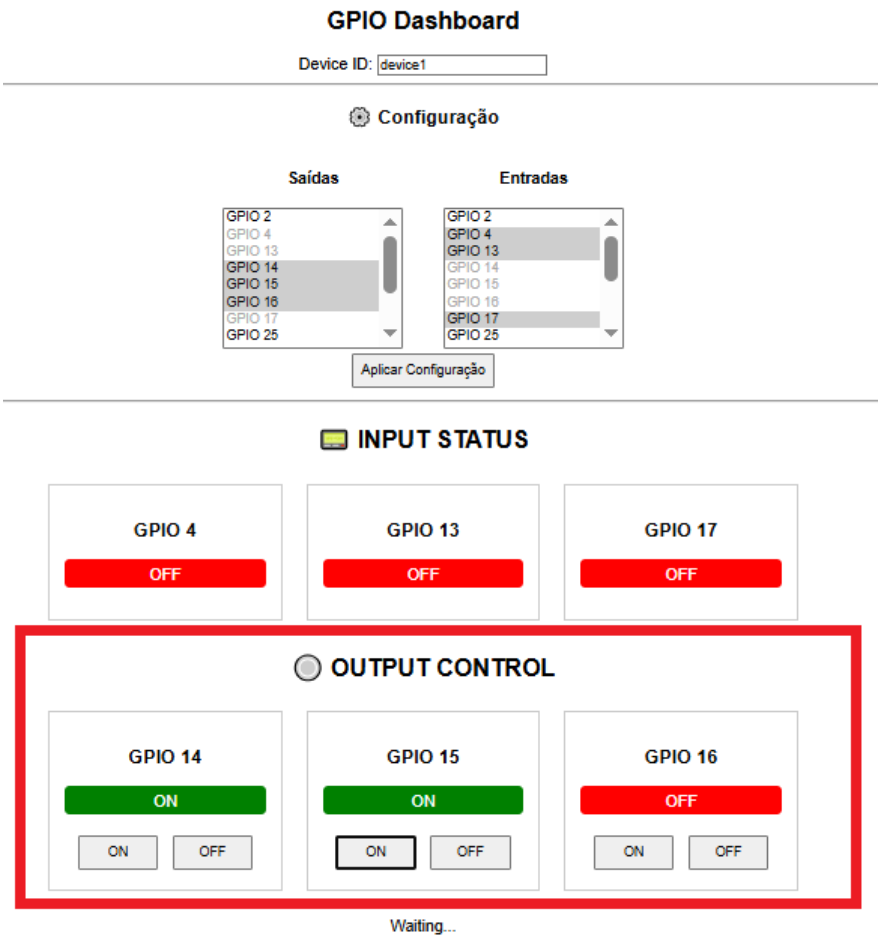
Para cada saída disponível, a interface apresenta um indicador de estado e dois botões de comando, identificados como *ON* e *OFF*. Ao selecionar uma dessas opções, a aplicação gera uma mensagem contendo a identificação da GPIO e o estado desejado, encaminhando-a ao dispositivo embarcado por meio do sistema de comunicação implementado.

Após o processamento do comando pelo *firmware*, o estado da saída é atualizado e disponibilizado para consulta pela interface. Dessa forma, o usuário pode acompanhar o resultado das operações realizadas e verificar o estado atual das saídas controladas remotamente.

A apresentação individual dos controles para cada GPIO permite o acionamento independente dos dispositivos conectados ao sistema, tornando a interface adequada para aplicações de monitoramento e automação que demandam controle remoto de múltiplos pontos de atuação.

A Figura 12 apresenta a área da interface destinada ao controle das saídas digitais.

Figura 12 – Área da interface destinada ao controle das saídas digitais



Fonte: Autoria própria

4.4.2.4 Comunicação MQTT via WebSocket

A comunicação entre a interface supervisória e o *broker* foi implementada utilizando o protocolo MQTT sobre *WebSocket Secure* (WSS). Essa escolha foi necessária devido às limitações dos navegadores *web*, que não permitem o estabelecimento direto de conexões TCP utilizadas pelo protocolo MQTT em sua forma tradicional.

Por meio dessa abordagem, a interface desenvolvida em JavaScript pode estabelecer uma conexão persistente diretamente com o *broker* MQTT, permitindo o envio das configurações das GPIOs e dos comandos de controle das saídas digitais.

Além da compatibilidade com navegadores, a utilização de *WebSocket* seguro mantém os mecanismos de proteção fornecidos pelo protocolo *Transport Layer Security* (TLS), contribuindo para a confidencialidade e integridade das mensagens trocadas entre a interface supervisória e o *broker* MQTT.

A implementação foi realizada utilizando a biblioteca MQTT.js, responsável pelo gerenciamento da conexão e pela publicação das mensagens utilizadas pelo sistema supervisório.

4.4.3 Ferramenta de testes de estresse

Com o objetivo de automatizar a execução dos testes realizados neste trabalho, foi desenvolvida uma interface *web* específica para geração de carga sobre o sistema. A ferramenta foi implementada utilizando HTML, CSS e JavaScript e disponibilizada juntamente com os demais componentes do sistema.

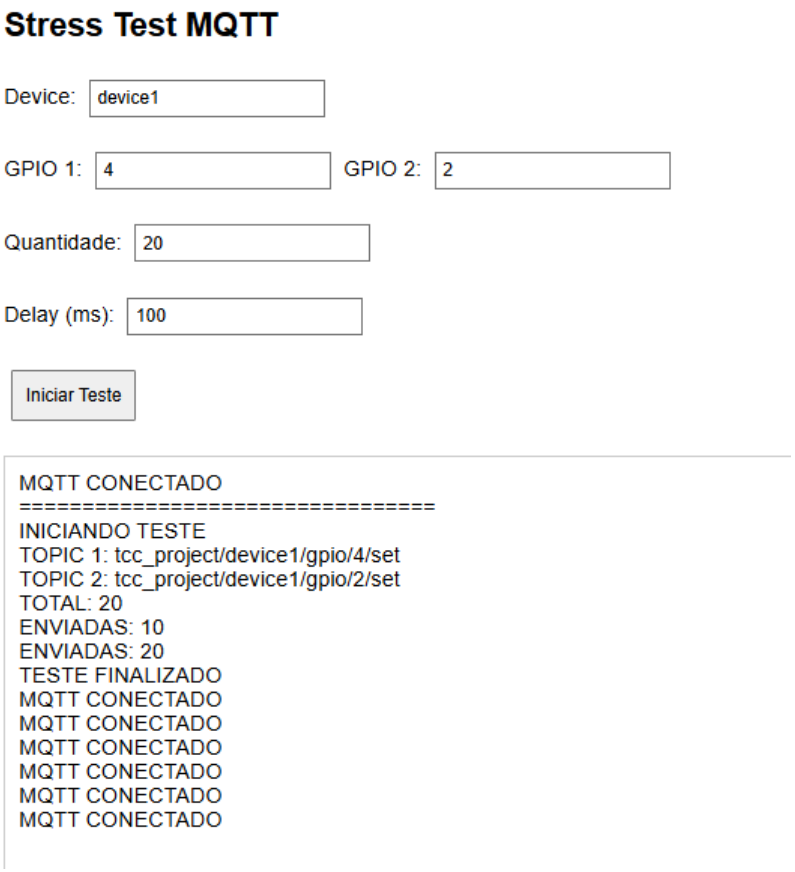
A interface permite configurar parâmetros relacionados à execução dos testes, como a quantidade de ciclos, o intervalo entre os comandos enviados e as GPIOs que serão utilizadas durante o procedimento. Com base nessas configurações, a ferramenta realiza o envio automático de comandos para o dispositivo embarcado, eliminando a necessidade de acionamentos manuais repetitivos.

Durante a execução dos testes, os comandos percorrem toda a arquitetura desenvolvida, passando pelo *broker* MQTT, pelo *firmware* embarcado e pelos serviços responsáveis pelo processamento e armazenamento das informações, incluindo a API REST, o banco de dados e o registro local em cartão SD. Dessa forma, torna-se possível validar simultaneamente os diferentes componentes da solução e verificar seu comportamento sob condições de utilização intensiva.

Além da geração de carga, a ferramenta foi projetada para facilitar a reprodução dos cenários de teste, permitindo a execução padronizada dos procedimentos utilizados durante a etapa de validação do sistema. Essa abordagem contribui para a obtenção de resultados consistentes e para a comparação entre diferentes execuções dos testes.

A Figura 13 apresenta a interface desenvolvida para execução dos testes de estresse.

Figura 13 – Interface desenvolvida para execução dos testes de estresse



Fonte: Autoria própria

5 APRESENTAÇÃO DOS RESULTADOS

Este capítulo apresenta os resultados obtidos durante a validação do sistema desenvolvido. Os testes realizados tiveram como objetivo verificar o correto funcionamento da arquitetura proposta, abrangendo a comunicação entre a interface *web*, o *broker* MQTT, o *firmware* embarcado, o armazenamento em cartão SD e o banco de dados remoto.

Inicialmente, foram realizados testes de validação funcional para avaliar a integração entre os diferentes componentes do sistema. Nessa etapa, foram executados testes de acionamento das GPIOs configuradas como saídas digitais, bem como testes de detecção de eventos nas GPIOs configuradas como entradas. Os resultados foram verificados por meio dos registros gerados pelo dispositivo embarcado, pelo cartão SD, pelo banco de dados e pelas informações apresentadas na interface supervisória.

Também foram realizados testes destinados à avaliação do comportamento do sistema sob condições de utilização intensiva, permitindo verificar a confiabilidade dos mecanismos de comunicação, armazenamento e processamento de eventos implementados.

Os resultados apresentados neste capítulo permitem analisar a consistência das informações registradas nos diferentes componentes da arquitetura e validar o funcionamento da solução proposta.

5.1 Análise e discussão dos resultados

Com o objetivo de validar a solução desenvolvida, foram realizados testes funcionais e testes de estresse envolvendo os principais fluxos de operação do sistema. Os testes funcionais contemplaram tanto o acionamento das GPIOs configuradas como saídas digitais quanto a detecção de eventos nas GPIOs configuradas como entradas digitais.

Além da verificação do comportamento físico das GPIOs, foram analisados os registros gerados nos diferentes componentes da arquitetura, incluindo o *firmware* embarcado, o cartão SD, o banco de dados e a interface supervisória. Dessa forma, foi possível avaliar a consistência das informações ao longo de todo o fluxo de comunicação implementado.

As subseções seguintes apresentam os resultados obtidos em cada etapa de validação.

5.1.1 Validação funcional das saídas digitais

Com o objetivo de validar o fluxo de controle remoto implementado, foram realizados testes de acionamento e desligamento das GPIOs configuradas como saídas digitais. Durante os ensaios, os comandos foram enviados por meio da interface supervisória e a correta execução das operações foi verificada através dos registros gerados pelo *firmware* embarcado, pelo cartão SD e pelo banco de dados.

Foram avaliadas todas as 12 GPIOs disponíveis para operação como saídas digitais, realizando-se operações de acionamento e desligamento para cada uma delas. Os registros completos gerados durante os testes encontram-se disponíveis nos Apêndices A e B.

A Tabela 6 apresenta um resumo dos resultados obtidos.

Tabela 6 – Resultados da validação funcional das GPIOs de saída

GPIO	Comando	ESP32	Cartão SD	Banco de Dados
13	ON	Ligou	Evento salvo	Evento salvo
13	OFF	Desligou	Evento salvo	Evento salvo
14	ON	Ligou	Evento salvo	Evento salvo
14	OFF	Desligou	Evento salvo	Evento salvo
15	ON	Ligou	Evento salvo	Evento salvo
15	OFF	Desligou	Evento salvo	Evento salvo
16	ON	Ligou	Evento salvo	Evento salvo
16	OFF	Desligou	Evento salvo	Evento salvo
17	ON	Ligou	Evento salvo	Evento salvo
17	OFF	Desligou	Evento salvo	Evento salvo

Fonte: Autoria própria

Os resultados demonstram que todos os comandos enviados foram corretamente recebidos e processados pelo dispositivo embarcado. Em todos os testes realizados, o acionamento e o desligamento das saídas ocorreram conforme esperado, sendo observados os respectivos registros tanto no cartão SD quanto no banco de dados. Não foram identificadas divergências entre os eventos registrados localmente e aqueles armazenados remotamente, evidenciando o correto funcionamento do fluxo de controle implementado.

5.1.2 Validação funcional das entradas digitais

Com o objetivo de validar o funcionamento do monitoramento das entradas digitais, foram realizados testes de alteração de estado nas GPIOs configuradas como entradas. Durante os ensaios, sinais foram aplicados às entradas do dispositivo, simulando eventos de campo e permitindo verificar o fluxo completo de aquisição, transmissão e armazenamento das informações.

Foram avaliadas todas as GPIOs disponíveis para operação como entradas digitais. Para cada entrada testada, verificou-se a detecção do evento pelo *firmware* embarcado, o envio da informação ao servidor por meio de requisição HTTP, o armazenamento do registro no banco de dados e a atualização do estado exibido na interface supervisória.

Os registros completos gerados durante os testes encontram-se disponíveis nos Apêndices C e D.

A Tabela 7 apresenta um resumo dos resultados obtidos.

Tabela 7 – Resultados da validação funcional das GPIOs de entrada

GPIO	ESP32	Cartão SD	Banco de Dados	Interface
14	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
13	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
27	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
26	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
25	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
33	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
32	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
35	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
34	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
39	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
15	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
2	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
4	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
16	Evento detectado	Evento salvo	Evento salvo	Estado atualizado
17	Evento detectado	Evento salvo	Evento salvo	Estado atualizado

Fonte: Autoria própria

Os resultados demonstram que todas as GPIOs avaliadas foram corretamente monitoradas pelo dispositivo embarcado. Em todos os testes realizados, os eventos gerados nas entradas foram detectados pelo *firmware* e propagados corretamente para os demais componentes da arquitetura. Os registros foram armazenados sem inconsistências no cartão SD e no banco de dados, sendo posteriormente refletidos na interface supervisória por meio da atualização do estado das entradas monitoradas.

5.1.3 Testes de estresse

Para automatizar a execução dos testes, foi desenvolvido o *script stress_test.html*, disponibilizado juntamente com o código-fonte do *backend* no repositório do projeto apresentado na Seção 4.0.1. A ferramenta permite configurar parâmetros como quantidade de ciclos, GPIOs utilizadas e intervalo entre mensagens, possibilitando a reprodução padronizada dos cenários de teste.

A interface de teste apresenta o histórico das mensagens publicadas, permitindo acompanhar a sequência de comandos enviados ao dispositivo e verificar a correta geração dos identificadores associados a cada operação.

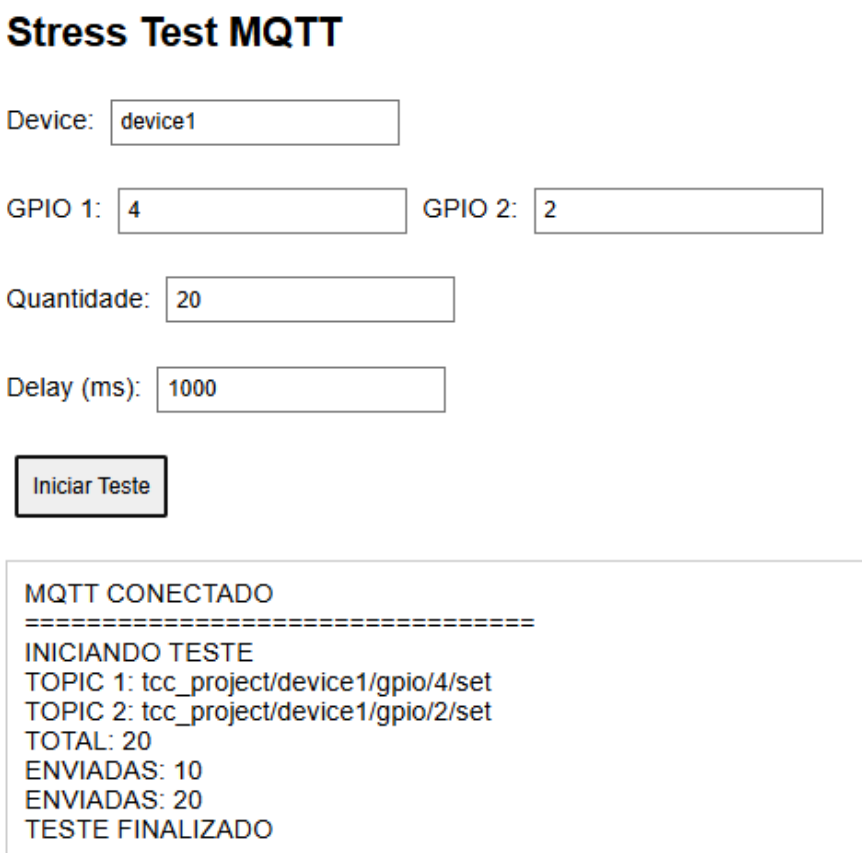
Os testes foram executados utilizando as GPIOs 2 e 4 configuradas como saídas digitais, mantendo-se a mesma quantidade de comandos em todos os cenários e variando apenas o intervalo entre as publicações MQTT. Os registros gerados durante cada experimento foram analisados por meio dos registros armazenados no cartão SD e dos dados persistidos no banco de dados.

5.1.3.1 Teste com intervalo de 1000 ms

O primeiro ensaio foi realizado utilizando um intervalo de 1000 ms entre as publicações MQTT, com o objetivo de estabelecer um cenário de referência para avaliação do comportamento do sistema. Durante o experimento, foram executados 20 ciclos de acionamento

e desligamento das GPIOs 2 e 4, totalizando 40 comandos enviados ao dispositivo embarcado. A configuração utilizada na ferramenta de testes é apresentada na Figura 14.

Figura 14 – Execução do teste de estresse utilizando duas GPIOs com intervalo de 1000 ms entre publicações MQTT



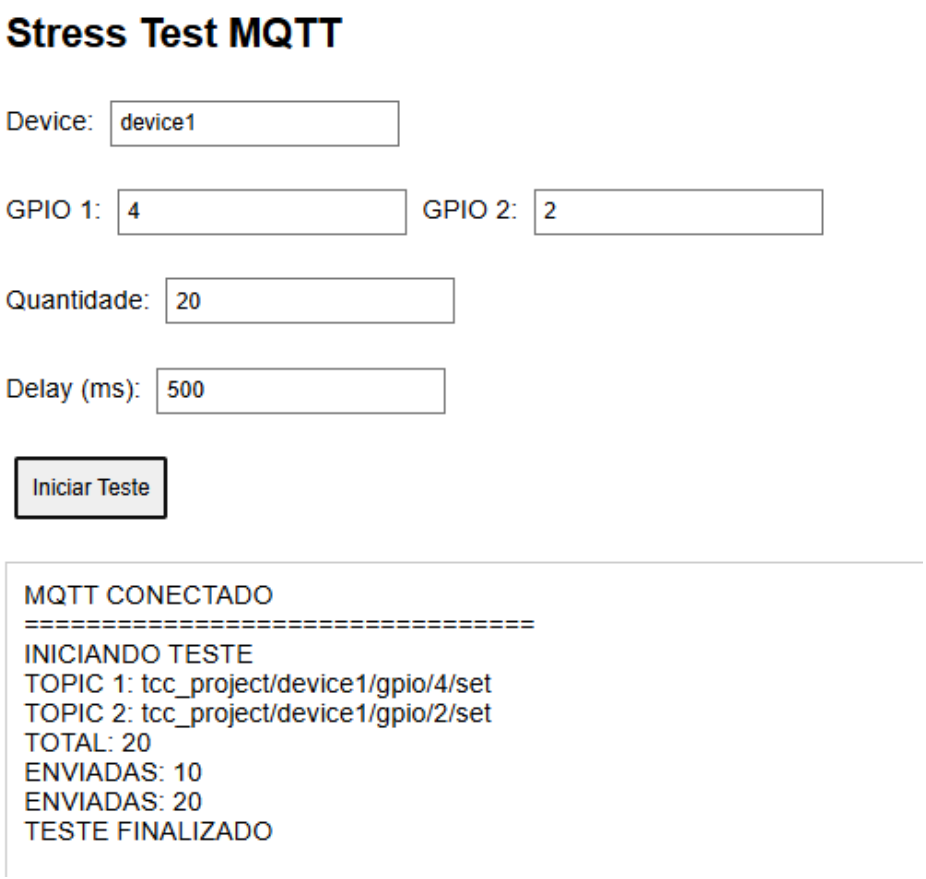
Fonte: Autoria própria

Os resultados obtidos demonstraram que todos os comandos enviados foram recebidos e processados corretamente pelo dispositivo embarcado. A comparação entre os registros armazenados no cartão SD e aqueles persistidos no banco de dados confirmou que não ocorreram perdas de mensagens nem alterações na sequência de processamento dos comandos. Os registros completos gerados durante o experimento encontram-se apresentados nos Apêndices H e F.

5.1.3.2 Teste com intervalo de 500 ms

No segundo ensaio, o intervalo entre as publicações MQTT foi reduzido para 500 ms, mantendo-se a mesma configuração utilizada no experimento anterior, com 20 ciclos de acionamento e desligamento das GPIOs 2 e 4, totalizando 40 comandos enviados ao dispositivo embarcado. A configuração empregada na ferramenta de testes é apresentada na Figura 15.

Figura 15 – Execução do teste de estresse utilizando duas GPIOs com intervalo de 500 ms entre publicações MQTT



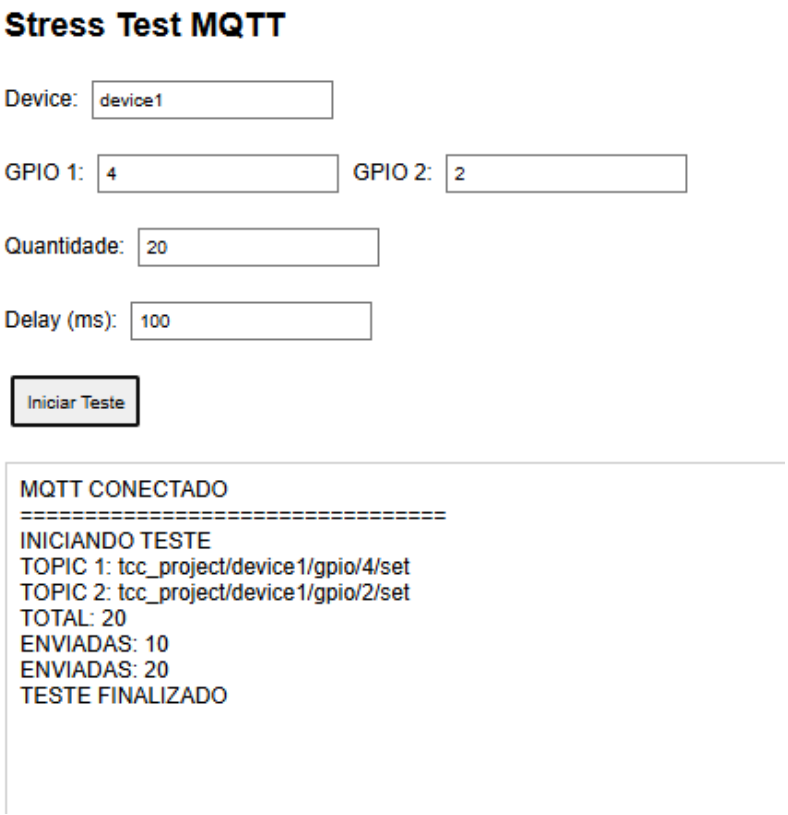
Fonte: Autoria própria

Os resultados obtidos permaneceram consistentes em relação ao cenário anterior. Todos os comandos enviados foram recebidos e processados corretamente pelo dispositivo embarcado, sendo observada correspondência integral entre os registros armazenados no cartão SD e aqueles persistidos no banco de dados. Não foram identificadas perdas de mensagens nem alterações na sequência de processamento dos comandos. Os registros completos referentes a esse experimento podem ser consultados nos Apêndices L e J.

5.1.3.3 Teste com intervalo de 100 ms

No terceiro ensaio, o intervalo entre as publicações MQTT foi reduzido para 100 ms, representando o cenário de maior taxa de envio de comandos avaliado neste trabalho. Assim como nos experimentos anteriores, foram executados 20 ciclos de acionamento e desligamento das GPIOs 2 e 4, totalizando 40 comandos enviados ao dispositivo embarcado. A configuração utilizada na ferramenta de testes é apresentada na Figura 16.

Figura 16 – Execução do teste de estresse utilizando duas GPIOs com intervalo de 100 ms entre publicações MQTT



Fonte: Autoria própria

Os resultados demonstraram que todos os comandos enviados foram recebidos e processados corretamente pelo dispositivo embarcado, mantendo-se a correspondência entre os registros armazenados no cartão SD e aqueles persistidos no banco de dados. Também não foram observadas perdas de mensagens nem alterações na sequência de processamento dos comandos.

Em relação aos cenários anteriores, verificou-se um aumento no tempo necessário para a persistência dos últimos registros no banco de dados. Enquanto os primeiros eventos foram armazenados poucos instantes após sua execução, os últimos comandos apresentaram um atraso acumulado de aproximadamente 10 segundos. Os registros completos gerados durante o experimento encontram-se apresentados nos Apêndices P e N.

5.1.3.4 Análise comparativa dos testes de estresse

A Tabela 8 apresenta um resumo dos resultados obtidos nos três cenários avaliados.

Tabela 8 – Comparação dos resultados obtidos nos testes de estresse

Critério	1000 ms	500 ms	100 ms
Mensagens perdidas	Não	Não	Não
Ordem dos comandos preservada	Sim	Sim	Sim
Registros no cartão SD	Completo	Completo	Completo
Persistência no banco de dados	Completa	Completa	Completa
Atraso perceptível na persistência	Não	Baixo	Moderado

Fonte: Autoria própria.

A comparação entre os três cenários demonstra que a redução do intervalo entre as publicações MQTT não comprometeu a execução dos comandos nem a integridade dos dados registrados. Em todos os testes realizados, os comandos foram processados na ordem correta, sem perda de mensagens, sendo os eventos registrados tanto no cartão SD quanto no banco de dados.

Observou-se, entretanto, uma diferença entre o instante de execução dos comandos no dispositivo e o momento em que os últimos registros foram persistidos no banco de dados. Enquanto os primeiros eventos foram armazenados poucos instantes após sua execução, os últimos registros apresentaram atraso acumulado de aproximadamente 10 segundos no cenário com intervalo de 100 ms. Esse comportamento é decorrente da arquitetura adotada para desacoplar o processamento das mensagens MQTT do envio das requisições HTTP ao servidor. Nessa abordagem, os eventos são inicialmente armazenados em uma fila de processamento e posteriormente transmitidos por uma tarefa dedicada, evitando que atrasos nas requisições HTTP bloqueiem a recepção e o processamento dos comandos MQTT.

Durante os testes realizados não foi observado esgotamento da capacidade da fila de processamento, uma vez que todos os eventos gerados foram processados e persistidos corretamente. Entretanto, o aumento da taxa de publicação resultou no acúmulo temporário de requisições HTTP, ocasionando maior tempo para a persistência dos últimos registros no banco de dados. Esse comportamento representa uma limitação da arquitetura adotada, decorrente da priorização do processamento dos comandos recebidos em relação ao envio das informações ao servidor.

Embora esse mecanismo introduza um atraso gradual na persistência dos últimos eventos no cenário de maior taxa de publicação avaliado neste trabalho, nenhuma mensagem foi perdida, sendo todos os comandos processados e armazenados corretamente. Dessa forma, os resultados demonstram que a arquitetura manteve a integridade dos dados e o correto funcionamento da comunicação durante os cenários avaliados, evidenciando como principal limitação o aumento da latência na persistência dos últimos registros em situações de maior carga de processamento.

6 CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento de uma arquitetura de monitoramento voltada para aplicações de Internet das Coisas (*IoT*), abrangendo as etapas necessárias para aquisição, processamento, comunicação, armazenamento e visualização de dados em sistemas aplicáveis a plantas industriais e outros cenários de automação. Para isso, foi implementado um sistema de aquisição baseado em microcontrolador, capaz de realizar a leitura de sinais digitais e o acionamento de saídas digitais por meio de relés.

A arquitetura desenvolvida utilizou dois protocolos de comunicação amplamente empregados em aplicações IoT, aplicados em diferentes etapas do sistema conforme as características de cada mecanismo de comunicação. O protocolo MQTT foi utilizado para o envio dos comandos entre a interface supervisória e o dispositivo embarcado, enquanto o protocolo HTTP foi empregado para o registro dos eventos e a persistência das informações no servidor.

Além da comunicação entre os dispositivos, foi implementado um mecanismo de retenção, armazenamento e auditoria dos dados gerados pelo sistema. O armazenamento local em cartão SD possibilitou o registro dos eventos diretamente no dispositivo embarcado, enquanto o armazenamento remoto em servidor possibilitou a persistência dos dados em banco de dados, permitindo consultas, auditoria e análise posterior das informações coletadas.

A disponibilização dos dados armazenados permitiu o desenvolvimento de uma interface web supervisória, responsável pela visualização do estado dos dispositivos monitorados e pelo envio de comandos de controle remoto. Dessa forma, a solução desenvolvida integrou o dispositivo embarcado, a infraestrutura de comunicação, o servidor e a interface do usuário em uma única arquitetura.

O desenvolvimento do *firmware* embarcado possibilitou implementar mecanismos para aquisição dos sinais, processamento dos eventos, gerenciamento das configurações recebidas dinamicamente e controle dos recursos de *hardware*. Já o servidor foi responsável por centralizar o tratamento das mensagens, realizar a persistência dos dados e disponibilizar as informações para a interface supervisória.

A realização dos testes permitiu avaliar o funcionamento da solução em diferentes cenários, incluindo operação contínua e situações de aumento da taxa de geração de mensagens. Os resultados demonstraram que o sistema é capaz de realizar o fluxo completo de comunicação entre dispositivo embarcado, servidor e usuário, mantendo o registro dos eventos e possibilitando o acompanhamento e controle dos dispositivos monitorados.

Entretanto, os testes de estresse evidenciaram pontos que podem ser aprimorados, principalmente relacionados ao aumento da latência observado em cenários com grande volume de mensagens. Esses resultados indicam oportunidades de melhoria tanto no processamento realizado pelo dispositivo embarcado quanto nos mecanismos de comunicação, processamento e persistência utilizados pelo servidor.

De modo geral, os objetivos propostos neste trabalho foram alcançados. Foi desenvolvido um sistema embarcado capaz de realizar a aquisição de sinais digitais e o acionamento remoto de saídas, implementada a infraestrutura de comunicação utilizando os protocolos MQTT e HTTP, desenvolvido o *backend* responsável pelo processamento e persistência das informações, bem como uma interface supervisória para monitoramento e controle dos dispositivos. Por fim, a arquitetura foi validada por meio de testes funcionais e de estresse, os quais demonstraram o funcionamento integrado da solução e permitiram identificar suas limitações e oportunidades de aprimoramento.

6.1 Sugestões para trabalhos futuros

Como continuidade deste trabalho, diversas funcionalidades podem ser incorporadas à arquitetura desenvolvida, ampliando sua aplicabilidade em ambientes industriais.

Uma das sugestões para trabalhos futuros consiste na investigação de estratégias para redução dos atrasos observados no processamento e envio das notificações em cenários com elevada taxa de geração de eventos. Durante os testes, foi identificado que o aumento da frequência de mensagens pode impactar o tempo necessário para que um evento de controle seja notificado ao usuário por meio das requisições HTTP.

Nesse contexto, uma possibilidade de investigação consiste em analisar o impacto das rotinas executadas no dispositivo embarcado, avaliando alternativas para otimização do desempenho do microcontrolador. Entre as abordagens possíveis está a revisão dos mecanismos de geração de logs no *firmware*, verificando a possibilidade de redução ou reorganização das operações de escrita e processamento que possam consumir recursos do ESP32. Além disso, pode ser avaliada uma otimização da arquitetura de tarefas (*tasks*) do sistema operacional do ESP32, buscando melhorar o paralelismo entre as atividades executadas e permitir uma utilização mais eficiente dos recursos computacionais disponíveis.

Outra evolução consiste na implementação de mecanismos de atuação automática baseados em regras ou eventos, permitindo que o próprio sistema execute ações de controle a partir das informações coletadas. Dessa forma, a arquitetura deixaria de atuar apenas como uma plataforma de monitoramento, passando também a desempenhar funções de supervisão e controle em processos industriais.

Também pode ser explorada a implementação de mecanismos de autenticação e controle de acesso mais robustos, permitindo diferentes níveis de permissão para operadores, supervisores e administradores do sistema.

Outra possibilidade é a implementação de notificações automáticas para eventos críticos, utilizando serviços de mensagens instantâneas, correio eletrônico ou notificações *push*, permitindo que os responsáveis sejam informados imediatamente sobre condições anormais de operação.

Por fim, recomenda-se a validação da solução em um ambiente industrial real, envolvendo um número maior de dispositivos de aquisição de dados e diferentes cenários operacionais. Essa avaliação poderá fornecer informações adicionais sobre escalabilidade, desempenho, confiabilidade e disponibilidade da arquitetura proposta em condições reais de utilização.

REFERÊNCIAS

ALANAZI, M.; MAHMOOD, A.; CHOWDHURY, M. Scada vulnerabilities and attacks: A review of the state-of-the-art and open issues. *Computers & Security*, Elsevier, v. 29, p. 103028, 2023. 22, 23

ELYZA, L. Design and build an academic e-reminder application using mobile-based push notification technology. *Jurnal Komputer, Informasi dan Teknologi*, v. 3, n. 1, p. 7–20, 2023. 25

FERREIRA, L. *Uma Arquitetura para a Aplicação da Internet das Coisas Industrial*. 100 p. Dissertação (Dissertação (Mestrado em Engenharia Elétrica)) — Universidade Estadual Paulista "Júlio de Mesquita Filho"(UNESP), Sorocaba, 2018. 18, 19

FIELDING, R. *et al. Hypertext Transfer Protocol – HTTP/1.1*. [S.l.], 1999. 22

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado) — University of California, Irvine, 2000. 24

GUILLEMIN, P.; FRIESS, P. *Internet of Things Strategic Research Agenda*. Brussels, 2009. Technical Report. Disponível em: https://internet-of-things-research.eu/pdf/IoT_Cluster_Strategic_Research_Agenda_2011.pdf. 21

ISIKLIGIL, E.; SAMAKAY, S.; KILINC, D. A prototype framework for high performance push notifications. *International Journal of Computer Applications*, v. 166, n. 10, p. 8–11, 2017. 25

LEE, E. A. Embedded software. In: ZELKOWITZ, M. V. (Ed.). *Advances in Computers*. London: Academic Press, 2002. v. 56. 21

LU, Y. Industry 4.0: A survey on technologies, applications and open research issues. *Journal of Industrial Information Integration*, v. 6, p. 1–10, 2017. ISSN 2452-414X. 14

MAHADIK, D. *et al.* To study webhooks for an event-driven integration mechanism. *International Journal of Advanced Research in Science, Communication and Technology*, v. 3, n. 2, p. 1–5, 2023. 24

MARWEDEL, P. *Embedded System Design: Embedded Systems, Foundations of Cyber-Physical Systems, and the Internet of Things*. [S.l.]: Springer, 2018. 21

MELO, M. A. d. M. Trabalho de Conclusão de Curso (Graduação em Engenharia de Controle e Automação), *Sistema embarcado e supervisório para uma aplicação IoT*. Recife: [s.n.], 2023. 56 p. 19, 20

MISHRA, B.; KERTESZ, A. The use of mqtt in m2m and iot systems: A survey. *IEEE Access*, v. 8, p. 201071–201086, 2020. 22

PERERA, C. *et al.* Context aware computing for the internet of things: A survey. *IEEE Communications Surveys & Tutorials*, IEEE, v. 16, n. 1, p. 414–454, 2014. 22

RAMADHAN, R.; ARIFIN, M. Microservices architecture in point of sales application based on restful api and webhook. *Jurnal Informatika dan Sistem Informasi*, v. 4, n. 2, p. 120–128, 2023. 24

RICHARDSON, L.; RUBY, S. *RESTful Web Services*. [S.l.]: O'Reilly Media, 2007. 23

SHAHROKHI, A.; AHMADI, M. Power evaluation of iot application layer protocols. In: *Proceedings of the International Conference on Computer and Knowledge Engineering*. Kermanshah, Iran: [s.n.], 2019. 22

SPANTI, D. *HTTP and MQTT: A comparison in the context of Industry 4.0*. Dissertação (Mestrado) — Politécnico de Torino, Torino, 2023. Tutor de Estágio: Thomas Ferrero. 14

YADAV, G.; PAUL, K. Architecture and security of scada systems: A review. *International Journal of Critical Infrastructure Protection*, Elsevier, v. 34, p. 26, 2021. 22

APÊNDICES

APÊNDICE A – REGISTROS DOS TESTES DE CONTROLE DAS SAÍDAS ARMAZENADOS NO CARTÃO SD

```
[2026-06-14 19:12:19] Sistema iniciado!
[2026-06-14 19:12:32] Executando comando: GPIO 2 -> 1 (msg_id=135)
[2026-06-14 19:12:32] GPIO 2 setado para 1
[2026-06-14 19:12:34] Executando comando: GPIO 4 -> 1 (msg_id=136)
[2026-06-14 19:12:34] GPIO 4 setado para 1
[2026-06-14 19:12:36] Executando comando: GPIO 13 -> 1 (msg_id=137)
[2026-06-14 19:12:36] GPIO 13 setado para 1
[2026-06-14 19:12:38] Executando comando: GPIO 14 -> 1 (msg_id=138)
[2026-06-14 19:12:38] GPIO 14 setado para 1
[2026-06-14 19:12:40] Executando comando: GPIO 15 -> 1 (msg_id=139)
[2026-06-14 19:12:40] GPIO 15 setado para 1
[2026-06-14 19:12:41] Executando comando: GPIO 16 -> 1 (msg_id=140)
[2026-06-14 19:12:41] GPIO 16 setado para 1
[2026-06-14 19:12:43] Executando comando: GPIO 26 -> 1 (msg_id=141)
[2026-06-14 19:12:43] GPIO 26 setado para 1
[2026-06-14 19:12:45] Executando comando: GPIO 25 -> 1 (msg_id=142)
[2026-06-14 19:12:45] GPIO 25 setado para 1
[2026-06-14 19:12:46] Executando comando: GPIO 17 -> 1 (msg_id=143)
[2026-06-14 19:12:46] GPIO 17 setado para 1
[2026-06-14 19:12:48] Executando comando: GPIO 27 -> 1 (msg_id=144)
[2026-06-14 19:12:48] GPIO 27 setado para 1
[2026-06-14 19:12:50] Executando comando: GPIO 32 -> 1 (msg_id=145)
[2026-06-14 19:12:50] GPIO 32 setado para 1
[2026-06-14 19:12:52] Executando comando: GPIO 33 -> 1 (msg_id=146)
[2026-06-14 19:12:52] GPIO 33 setado para 1
[2026-06-14 19:12:56] Executando comando: GPIO 2 -> 0 (msg_id=147)
[2026-06-14 19:12:56] GPIO 2 setado para 0
[2026-06-14 19:12:57] Executando comando: GPIO 4 -> 0 (msg_id=148)
[2026-06-14 19:12:57] GPIO 4 setado para 0
[2026-06-14 19:12:59] Executando comando: GPIO 13 -> 0 (msg_id=149)
[2026-06-14 19:12:59] GPIO 13 setado para 0
[2026-06-14 19:13:01] Executando comando: GPIO 16 -> 0 (msg_id=150)
[2026-06-14 19:13:01] GPIO 16 setado para 0
[2026-06-14 19:13:03] Executando comando: GPIO 15 -> 0 (msg_id=151)
[2026-06-14 19:13:03] GPIO 15 setado para 0
[2026-06-14 19:13:04] Executando comando: GPIO 14 -> 0 (msg_id=152)
[2026-06-14 19:13:04] GPIO 14 setado para 0
[2026-06-14 19:13:05] Executando comando: GPIO 17 -> 0 (msg_id=153)
[2026-06-14 19:13:05] GPIO 17 setado para 0
[2026-06-14 19:13:08] Executando comando: GPIO 25 -> 0 (msg_id=154)
[2026-06-14 19:13:08] GPIO 25 setado para 0
[2026-06-14 19:13:10] Executando comando: GPIO 26 -> 0 (msg_id=155)
[2026-06-14 19:13:10] GPIO 26 setado para 0
[2026-06-14 19:13:12] Executando comando: GPIO 27 -> 0 (msg_id=156)
[2026-06-14 19:13:12] GPIO 27 setado para 0
[2026-06-14 19:13:14] Executando comando: GPIO 32 -> 0 (msg_id=157)
[2026-06-14 19:13:14] GPIO 32 setado para 0
[2026-06-14 19:13:15] Executando comando: GPIO 33 -> 0 (msg_id=158)
[2026-06-14 19:13:15] GPIO 33 setado para 0
```

APÊNDICE B – REGISTROS DOS TESTES DE CONTROLE DAS SAÍDAS ARMAZENADOS NO BANCO DE DADOS

```
===Banco de dados steph999_Tcc_Project

== Estrutura para tabela gpio_states

|------
|Coluna|Tipo|Nulo|Padrão
|------
|//**id**//|int(11)|Não|
|timestamp|datetime|Sim|NULL
|device_id|varchar(255)|Sim|NULL
|gpio_id|int(11)|Sim|NULL
|state|int(11)|Sim|NULL
|server_timestamp|datetime|Sim|CURRENT_TIMESTAMP
|message_id|varchar(255)|Sim|NULL
== Despejando dados para a tabela gpio_states

|1824|2026-06-14 19:12:32|device1|2|1|2026-06-14 16:12:32|135
|1825|2026-06-14 19:12:34|device1|4|1|2026-06-14 16:12:34|136
|1826|2026-06-14 19:12:36|device1|13|1|2026-06-14 16:12:36|137
|1827|2026-06-14 19:12:38|device1|14|1|2026-06-14 16:12:38|138
|1828|2026-06-14 19:12:40|device1|15|1|2026-06-14 16:12:40|139
|1829|2026-06-14 19:12:41|device1|16|1|2026-06-14 16:12:41|140
|1830|2026-06-14 19:12:43|device1|26|1|2026-06-14 16:12:43|141
|1831|2026-06-14 19:12:45|device1|25|1|2026-06-14 16:12:45|142
|1832|2026-06-14 19:12:46|device1|17|1|2026-06-14 16:12:46|143
|1833|2026-06-14 19:12:48|device1|27|1|2026-06-14 16:12:48|144
|1834|2026-06-14 19:12:50|device1|32|1|2026-06-14 16:12:50|145
|1835|2026-06-14 19:12:52|device1|33|1|2026-06-14 16:12:52|146
|1836|2026-06-14 19:12:56|device1|2|0|2026-06-14 16:12:56|147
|1837|2026-06-14 19:12:57|device1|4|0|2026-06-14 16:12:58|148
|1838|2026-06-14 19:12:59|device1|13|0|2026-06-14 16:12:59|149
|1839|2026-06-14 19:13:01|device1|16|0|2026-06-14 16:13:01|150
|1840|2026-06-14 19:13:03|device1|15|0|2026-06-14 16:13:03|151
|1841|2026-06-14 19:13:04|device1|14|0|2026-06-14 16:13:04|152
|1842|2026-06-14 19:13:05|device1|17|0|2026-06-14 16:13:06|153
|1843|2026-06-14 19:13:08|device1|25|0|2026-06-14 16:13:08|154
|1844|2026-06-14 19:13:11|device1|26|0|2026-06-14 16:13:11|155
|1845|2026-06-14 19:13:12|device1|27|0|2026-06-14 16:13:12|156
|1846|2026-06-14 19:13:14|device1|32|0|2026-06-14 16:13:14|157
|1847|2026-06-14 19:13:15|device1|33|0|2026-06-14 16:13:15|158
```

APÊNDICE C – REGISTROS DOS TESTES DE MONITORAMENTO DAS ENTRADAS ARMAZENADOS NO CARTÃO SD

```
[2026-06-22 00:25:27] Sistema iniciado!
[2026-06-22 00:25:40] GPIO 14 mudou para 1
[2026-06-22 00:25:46] GPIO 14 mudou para 0
[2026-06-22 00:26:21] GPIO 13 mudou para 1
[2026-06-22 00:26:25] GPIO 13 mudou para 0
[2026-06-22 00:26:51] GPIO 27 mudou para 1
[2026-06-22 00:26:56] GPIO 27 mudou para 0
[2026-06-22 00:27:24] GPIO 26 mudou para 1
[2026-06-22 00:27:30] GPIO 26 mudou para 0
[2026-06-22 00:28:00] GPIO 25 mudou para 1
[2026-06-22 00:28:06] GPIO 25 mudou para 0
[2026-06-22 00:28:34] GPIO 33 mudou para 1
[2026-06-22 00:28:38] GPIO 33 mudou para 0
[2026-06-22 00:29:04] GPIO 32 mudou para 1
[2026-06-22 00:29:09] GPIO 32 mudou para 0
[2026-06-22 00:29:34] GPIO 35 mudou para 1
[2026-06-22 00:29:38] GPIO 35 mudou para 0
[2026-06-22 00:30:07] GPIO 34 mudou para 1
[2026-06-22 00:30:11] GPIO 34 mudou para 0
[2026-06-22 00:30:38] GPIO 39 mudou para 1
[2026-06-22 00:30:45] GPIO 39 mudou para 0
[2026-06-22 00:31:32] GPIO 15 mudou para 1
[2026-06-22 00:31:40] GPIO 15 mudou para 0
[2026-06-22 00:32:09] GPIO 2 mudou para 1
[2026-06-22 00:32:13] GPIO 2 mudou para 0
[2026-06-22 00:32:39] GPIO 4 mudou para 1
[2026-06-22 00:32:44] GPIO 4 mudou para 0
[2026-06-22 00:33:11] GPIO 16 mudou para 1
[2026-06-22 00:33:15] GPIO 16 mudou para 0
[2026-06-22 00:33:35] GPIO 17 mudou para 1
[2026-06-22 00:33:38] GPIO 17 mudou para 0
```

APÊNDICE D – REGISTROS DOS TESTES DE MONITORAMENTO DAS ENTRADAS ARMAZENADOS NO BANCO DE DADOS

```
===Banco de dados steph999_Tcc_Project

== Estrutura para tabela gpio_states

|------
|Coluna|Tipo|Nulo|Padrão
|------
|//**id**//|int(11)|Não|
|timestamp|datetime|Sim|NULL
|device_id|varchar(255)|Sim|NULL
|gpio_id|int(11)|Sim|NULL
|state|int(11)|Sim|NULL
|server_timestamp|datetime|Sim|CURRENT_TIMESTAMP
|message_id|varchar(255)|Sim|NULL
== Despejando dados para a tabela gpio_states

|3642|2026-06-22 00:25:40|device1|14|1|2026-06-21 21:25:40|
|3646|2026-06-22 00:25:46|device1|14|0|2026-06-21 21:25:46|
|3648|2026-06-22 00:26:21|device1|13|1|2026-06-21 21:26:21|
|3651|2026-06-22 00:26:25|device1|13|0|2026-06-21 21:26:26|
|3652|2026-06-22 00:26:51|device1|27|1|2026-06-21 21:26:51|
|3655|2026-06-22 00:26:56|device1|27|0|2026-06-21 21:26:56|
|3656|2026-06-22 00:27:24|device1|26|1|2026-06-21 21:27:25|
|3662|2026-06-22 00:27:30|device1|26|0|2026-06-21 21:27:30|
|3663|2026-06-22 00:28:00|device1|25|1|2026-06-21 21:28:00|
|3666|2026-06-22 00:28:06|device1|25|0|2026-06-21 21:28:06|
|3667|2026-06-22 00:28:34|device1|33|1|2026-06-21 21:28:34|
|3671|2026-06-22 00:28:38|device1|33|0|2026-06-21 21:28:39|
|3672|2026-06-22 00:29:04|device1|32|1|2026-06-21 21:29:05|
|3675|2026-06-22 00:29:08|device1|32|0|2026-06-21 21:29:09|
|3678|2026-06-22 00:29:34|device1|35|1|2026-06-21 21:29:34|
|3681|2026-06-22 00:29:38|device1|35|0|2026-06-21 21:29:38|
|3682|2026-06-22 00:30:07|device1|34|1|2026-06-21 21:30:08|
|3686|2026-06-22 00:30:12|device1|34|0|2026-06-21 21:30:12|
|3687|2026-06-22 00:30:38|device1|39|1|2026-06-21 21:30:38|
|3690|2026-06-22 00:30:45|device1|39|0|2026-06-21 21:30:45|
|3695|2026-06-22 00:31:32|device1|15|1|2026-06-21 21:31:32|
|3698|2026-06-22 00:31:40|device1|15|0|2026-06-21 21:31:40|
|3699|2026-06-22 00:32:09|device1|2|1|2026-06-21 21:32:10|
|3702|2026-06-22 00:32:13|device1|2|0|2026-06-21 21:32:13|
|3703|2026-06-22 00:32:39|device1|4|1|2026-06-21 21:32:39|
|3706|2026-06-22 00:32:44|device1|4|0|2026-06-21 21:32:44|
|3707|2026-06-22 00:33:11|device1|16|1|2026-06-21 21:33:11|
|3711|2026-06-22 00:33:15|device1|16|0|2026-06-21 21:33:15|
|3712|2026-06-22 00:33:35|device1|17|1|2026-06-21 21:33:35|
|3715|2026-06-22 00:33:38|device1|17|0|2026-06-21 21:33:38|
```

APÊNDICE E – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 1000 MS NO BANCO DE DADOS

```
===Banco de dados steph999_Tcc_Project

== Estrutura para tabela gpio_states

|------
|Coluna|Tipo|Nulo|Padrão
|------
|//**id**//|int(11)|Não|
|timestamp|datetime|Sim|NULL
|device_id|varchar(255)|Sim|NULL
|gpio_id|int(11)|Sim|NULL
|state|int(11)|Sim|NULL
|server_timestamp|datetime|Sim|CURRENT_TIMESTAMP
|message_id|varchar(255)|Sim|NULL
== Despejando dados para a tabela gpio_states

|3716|2026-06-28 01:22:25|device1|4|1|2026-06-27 22:22:25|1
|3717|2026-06-28 01:22:25|device1|2|1|2026-06-27 22:22:25|1
|3718|2026-06-28 01:22:26|device1|4|0|2026-06-27 22:22:26|2
|3719|2026-06-28 01:22:26|device1|2|0|2026-06-27 22:22:26|2
|3720|2026-06-28 01:22:27|device1|4|1|2026-06-27 22:22:27|3
|3721|2026-06-28 01:22:27|device1|2|1|2026-06-27 22:22:27|3
|3722|2026-06-28 01:22:28|device1|2|0|2026-06-27 22:22:28|4
|3723|2026-06-28 01:22:28|device1|4|0|2026-06-27 22:22:28|4
|3724|2026-06-28 01:22:29|device1|4|1|2026-06-27 22:22:29|5
|3725|2026-06-28 01:22:29|device1|2|1|2026-06-27 22:22:29|5
|3726|2026-06-28 01:22:30|device1|4|0|2026-06-27 22:22:30|6
|3727|2026-06-28 01:22:30|device1|2|0|2026-06-27 22:22:30|6
|3728|2026-06-28 01:22:31|device1|4|1|2026-06-27 22:22:31|7
|3729|2026-06-28 01:22:31|device1|2|1|2026-06-27 22:22:31|7
|3730|2026-06-28 01:22:32|device1|4|0|2026-06-27 22:22:32|8
|3731|2026-06-28 01:22:32|device1|2|0|2026-06-27 22:22:32|8
|3732|2026-06-28 01:22:33|device1|4|1|2026-06-27 22:22:33|9
|3733|2026-06-28 01:22:33|device1|2|1|2026-06-27 22:22:33|9
|3734|2026-06-28 01:22:34|device1|4|0|2026-06-27 22:22:34|10
|3735|2026-06-28 01:22:34|device1|2|0|2026-06-27 22:22:34|10
|3736|2026-06-28 01:22:35|device1|4|1|2026-06-27 22:22:35|11
|3737|2026-06-28 01:22:35|device1|2|1|2026-06-27 22:22:35|11
|3738|2026-06-28 01:22:36|device1|2|0|2026-06-27 22:22:36|12
|3739|2026-06-28 01:22:36|device1|4|0|2026-06-27 22:22:36|12
|3740|2026-06-28 01:22:37|device1|4|1|2026-06-27 22:22:37|13
|3741|2026-06-28 01:22:37|device1|2|1|2026-06-27 22:22:37|13
|3742|2026-06-28 01:22:38|device1|4|0|2026-06-27 22:22:38|14
|3743|2026-06-28 01:22:38|device1|2|0|2026-06-27 22:22:38|14
|3744|2026-06-28 01:22:39|device1|4|1|2026-06-27 22:22:39|15
|3745|2026-06-28 01:22:39|device1|2|1|2026-06-27 22:22:40|15
|3746|2026-06-28 01:22:40|device1|4|0|2026-06-27 22:22:40|16
|3747|2026-06-28 01:22:40|device1|2|0|2026-06-27 22:22:40|16
|3748|2026-06-28 01:22:41|device1|4|1|2026-06-27 22:22:41|17
|3749|2026-06-28 01:22:41|device1|2|1|2026-06-27 22:22:41|17
|3750|2026-06-28 01:22:42|device1|4|0|2026-06-27 22:22:42|18
|3751|2026-06-28 01:22:42|device1|2|0|2026-06-27 22:22:43|18
|3752|2026-06-28 01:22:43|device1|4|1|2026-06-27 22:22:43|19
|3753|2026-06-28 01:22:43|device1|2|1|2026-06-27 22:22:44|19
```

APÊNDICE F – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 1000 MS NO BANCO DE DADOS

```
|3754|2026-06-28 01:22:44|device1|4|0|2026-06-27 22:22:44|20
|3755|2026-06-28 01:22:44|device1|2|0|2026-06-27 22:22:44|20
```

APÊNDICE G – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 1000 MS NO CARTÃO SD

```
[2026-06-28 01:21:00] Sistema iniciado!
[2026-06-28 01:22:25] Executando comando: GPIO 4 -> 1 (msg_id=1)
[2026-06-28 01:22:25] GPIO 4 setado para 1
[2026-06-28 01:22:25] Executando comando: GPIO 2 -> 1 (msg_id=1)
[2026-06-28 01:22:25] GPIO 2 setado para 1
[2026-06-28 01:22:26] Executando comando: GPIO 4 -> 0 (msg_id=2)
[2026-06-28 01:22:26] GPIO 4 setado para 0
[2026-06-28 01:22:26] Executando comando: GPIO 2 -> 0 (msg_id=2)
[2026-06-28 01:22:26] GPIO 2 setado para 0
[2026-06-28 01:22:27] Executando comando: GPIO 4 -> 1 (msg_id=3)
[2026-06-28 01:22:27] GPIO 4 setado para 1
[2026-06-28 01:22:27] Executando comando: GPIO 2 -> 1 (msg_id=3)
[2026-06-28 01:22:27] GPIO 2 setado para 1
[2026-06-28 01:22:28] Executando comando: GPIO 2 -> 0 (msg_id=4)
[2026-06-28 01:22:28] GPIO 2 setado para 0
[2026-06-28 01:22:28] Executando comando: GPIO 4 -> 0 (msg_id=4)
[2026-06-28 01:22:28] GPIO 4 setado para 0
[2026-06-28 01:22:29] Executando comando: GPIO 4 -> 1 (msg_id=5)
[2026-06-28 01:22:29] GPIO 4 setado para 1
[2026-06-28 01:22:29] Executando comando: GPIO 2 -> 1 (msg_id=5)
[2026-06-28 01:22:29] GPIO 2 setado para 1
[2026-06-28 01:22:30] Executando comando: GPIO 4 -> 0 (msg_id=6)
[2026-06-28 01:22:30] GPIO 4 setado para 0
[2026-06-28 01:22:30] Executando comando: GPIO 2 -> 0 (msg_id=6)
[2026-06-28 01:22:30] GPIO 2 setado para 0
[2026-06-28 01:22:31] Executando comando: GPIO 4 -> 1 (msg_id=7)
[2026-06-28 01:22:31] GPIO 4 setado para 1
[2026-06-28 01:22:31] Executando comando: GPIO 2 -> 1 (msg_id=7)
[2026-06-28 01:22:31] GPIO 2 setado para 1
[2026-06-28 01:22:32] Executando comando: GPIO 4 -> 0 (msg_id=8)
[2026-06-28 01:22:32] GPIO 4 setado para 0
[2026-06-28 01:22:32] Executando comando: GPIO 2 -> 0 (msg_id=8)
[2026-06-28 01:22:32] GPIO 2 setado para 0
[2026-06-28 01:22:33] Executando comando: GPIO 4 -> 1 (msg_id=9)
[2026-06-28 01:22:33] GPIO 4 setado para 1
[2026-06-28 01:22:33] Executando comando: GPIO 2 -> 1 (msg_id=9)
[2026-06-28 01:22:33] GPIO 2 setado para 1
[2026-06-28 01:22:34] Executando comando: GPIO 4 -> 0 (msg_id=10)
[2026-06-28 01:22:34] GPIO 4 setado para 0
[2026-06-28 01:22:34] Executando comando: GPIO 2 -> 0 (msg_id=10)
[2026-06-28 01:22:34] GPIO 2 setado para 0
[2026-06-28 01:22:35] Executando comando: GPIO 4 -> 1 (msg_id=11)
[2026-06-28 01:22:35] GPIO 4 setado para 1
[2026-06-28 01:22:35] Executando comando: GPIO 2 -> 1 (msg_id=11)
[2026-06-28 01:22:35] GPIO 2 setado para 1
[2026-06-28 01:22:36] Executando comando: GPIO 2 -> 0 (msg_id=12)
[2026-06-28 01:22:36] GPIO 2 setado para 0
[2026-06-28 01:22:36] Executando comando: GPIO 4 -> 0 (msg_id=12)
[2026-06-28 01:22:36] GPIO 4 setado para 0
[2026-06-28 01:22:37] Executando comando: GPIO 4 -> 1 (msg_id=13)
[2026-06-28 01:22:37] GPIO 4 setado para 1
[2026-06-28 01:22:37] Executando comando: GPIO 2 -> 1 (msg_id=13)
[2026-06-28 01:22:37] GPIO 2 setado para 1
[2026-06-28 01:22:38] Executando comando: GPIO 4 -> 0 (msg_id=14)
```

APÊNDICE H – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 1000 MS NO CARTÃO SD

```
[2026-06-28 01:22:38] GPIO 4 setado para 0
[2026-06-28 01:22:38] Executando comando: GPIO 2 -> 0 (msg_id=14)
[2026-06-28 01:22:38] GPIO 2 setado para 0
[2026-06-28 01:22:39] Executando comando: GPIO 4 -> 1 (msg_id=15)
[2026-06-28 01:22:39] GPIO 4 setado para 1
[2026-06-28 01:22:39] Executando comando: GPIO 2 -> 1 (msg_id=15)
[2026-06-28 01:22:39] GPIO 2 setado para 1
[2026-06-28 01:22:40] Executando comando: GPIO 4 -> 0 (msg_id=16)
[2026-06-28 01:22:40] GPIO 4 setado para 0
[2026-06-28 01:22:40] Executando comando: GPIO 2 -> 0 (msg_id=16)
[2026-06-28 01:22:40] GPIO 2 setado para 0
[2026-06-28 01:22:41] Executando comando: GPIO 4 -> 1 (msg_id=17)
[2026-06-28 01:22:41] GPIO 4 setado para 1
[2026-06-28 01:22:41] Executando comando: GPIO 2 -> 1 (msg_id=17)
[2026-06-28 01:22:41] GPIO 2 setado para 1
[2026-06-28 01:22:42] Executando comando: GPIO 4 -> 0 (msg_id=18)
[2026-06-28 01:22:42] GPIO 4 setado para 0
[2026-06-28 01:22:42] Executando comando: GPIO 2 -> 0 (msg_id=18)
[2026-06-28 01:22:42] GPIO 2 setado para 0
[2026-06-28 01:22:43] Executando comando: GPIO 4 -> 1 (msg_id=19)
[2026-06-28 01:22:43] GPIO 4 setado para 1
[2026-06-28 01:22:43] Executando comando: GPIO 2 -> 1 (msg_id=19)
[2026-06-28 01:22:43] GPIO 2 setado para 1
[2026-06-28 01:22:44] Executando comando: GPIO 4 -> 0 (msg_id=20)
[2026-06-28 01:22:44] GPIO 4 setado para 0
[2026-06-28 01:22:44] Executando comando: GPIO 2 -> 0 (msg_id=20)
[2026-06-28 01:22:44] GPIO 2 setado para 0
```

APÊNDICE I – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 500 MS NO BANCO DE DADOS

===Banco de dados steph999_Tcc_Project

== Estrutura para tabela gpio_states

|------
|Coluna|Tipo|Nulo|Padrão
|------
|//**id**//|int(11)|Não|
|timestamp|datetime|Sim|NULL
|device_id|varchar(255)|Sim|NULL
|gpio_id|int(11)|Sim|NULL
|state|int(11)|Sim|NULL
|server_timestamp|datetime|Sim|CURRENT_TIMESTAMP
|message_id|varchar(255)|Sim|NULL
== Despejando dados para a tabela gpio_states

|3756|2026-06-28 01:33:48|device1|4|1|2026-06-27 22:33:49|1
|3757|2026-06-28 01:33:49|device1|2|1|2026-06-27 22:33:49|1
|3758|2026-06-28 01:33:49|device1|4|0|2026-06-27 22:33:49|2
|3759|2026-06-28 01:33:49|device1|2|0|2026-06-27 22:33:49|2
|3760|2026-06-28 01:33:49|device1|4|1|2026-06-27 22:33:50|3
|3761|2026-06-28 01:33:50|device1|2|1|2026-06-27 22:33:50|3
|3762|2026-06-28 01:33:50|device1|4|0|2026-06-27 22:33:50|4
|3763|2026-06-28 01:33:50|device1|2|0|2026-06-27 22:33:50|4
|3764|2026-06-28 01:33:50|device1|4|1|2026-06-27 22:33:51|5
|3765|2026-06-28 01:33:51|device1|2|1|2026-06-27 22:33:51|5
|3766|2026-06-28 01:33:51|device1|4|0|2026-06-27 22:33:51|6
|3767|2026-06-28 01:33:51|device1|2|0|2026-06-27 22:33:51|6
|3768|2026-06-28 01:33:51|device1|4|1|2026-06-27 22:33:52|7
|3769|2026-06-28 01:33:52|device1|2|1|2026-06-27 22:33:52|7
|3770|2026-06-28 01:33:52|device1|4|0|2026-06-27 22:33:52|8
|3771|2026-06-28 01:33:52|device1|2|0|2026-06-27 22:33:52|8
|3772|2026-06-28 01:33:53|device1|4|1|2026-06-27 22:33:53|9
|3773|2026-06-28 01:33:53|device1|2|1|2026-06-27 22:33:53|9
|3774|2026-06-28 01:33:53|device1|4|0|2026-06-27 22:33:53|10
|3775|2026-06-28 01:33:53|device1|2|0|2026-06-27 22:33:53|10
|3776|2026-06-28 01:33:53|device1|4|1|2026-06-27 22:33:54|11
|3777|2026-06-28 01:33:54|device1|2|1|2026-06-27 22:33:54|11
|3778|2026-06-28 01:33:54|device1|4|0|2026-06-27 22:33:54|12
|3779|2026-06-28 01:33:54|device1|2|0|2026-06-27 22:33:54|12
|3780|2026-06-28 01:33:54|device1|4|1|2026-06-27 22:33:55|13
|3781|2026-06-28 01:33:55|device1|2|1|2026-06-27 22:33:55|13
|3782|2026-06-28 01:33:55|device1|4|0|2026-06-27 22:33:55|14
|3783|2026-06-28 01:33:55|device1|2|0|2026-06-27 22:33:55|14
|3784|2026-06-28 01:33:56|device1|4|1|2026-06-27 22:33:56|15
|3785|2026-06-28 01:33:56|device1|2|1|2026-06-27 22:33:56|15
|3786|2026-06-28 01:33:56|device1|4|0|2026-06-27 22:33:56|16
|3787|2026-06-28 01:33:56|device1|2|0|2026-06-27 22:33:56|16
|3788|2026-06-28 01:33:57|device1|4|1|2026-06-27 22:33:57|17
|3789|2026-06-28 01:33:57|device1|2|1|2026-06-27 22:33:57|17
|3790|2026-06-28 01:33:57|device1|4|0|2026-06-27 22:33:57|18
|3791|2026-06-28 01:33:57|device1|2|0|2026-06-27 22:33:57|18
|3792|2026-06-28 01:33:58|device1|4|1|2026-06-27 22:33:58|19
|3793|2026-06-28 01:33:58|device1|2|1|2026-06-27 22:33:58|19

APÊNDICE J – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 500 MS NO BANCO DE DADOS

```
|3794|2026-06-28 01:33:58|device1|4|0|2026-06-27 22:33:58|20
|3795|2026-06-28 01:33:58|device1|2|0|2026-06-27 22:33:58|20
```

APÊNDICE K – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 500 MS NO CARTÃO SD

```
[2026-06-28 01:33:13] Sistema iniciado!
[2026-06-28 01:33:48] Executando comando: GPIO 4 -> 1 (msg_id=1)
[2026-06-28 01:33:48] GPIO 4 setado para 1
[2026-06-28 01:33:48] Executando comando: GPIO 2 -> 1 (msg_id=1)
[2026-06-28 01:33:49] GPIO 2 setado para 1
[2026-06-28 01:33:49] Executando comando: GPIO 4 -> 0 (msg_id=2)
[2026-06-28 01:33:49] GPIO 4 setado para 0
[2026-06-28 01:33:49] Executando comando: GPIO 2 -> 0 (msg_id=2)
[2026-06-28 01:33:49] GPIO 2 setado para 0
[2026-06-28 01:33:49] Executando comando: GPIO 4 -> 1 (msg_id=3)
[2026-06-28 01:33:49] GPIO 4 setado para 1
[2026-06-28 01:33:49] Executando comando: GPIO 2 -> 1 (msg_id=3)
[2026-06-28 01:33:49] GPIO 2 setado para 1
[2026-06-28 01:33:50] Executando comando: GPIO 4 -> 0 (msg_id=4)
[2026-06-28 01:33:50] GPIO 4 setado para 0
[2026-06-28 01:33:50] Executando comando: GPIO 2 -> 0 (msg_id=4)
[2026-06-28 01:33:50] GPIO 2 setado para 0
[2026-06-28 01:33:50] Executando comando: GPIO 4 -> 1 (msg_id=5)
[2026-06-28 01:33:50] GPIO 4 setado para 1
[2026-06-28 01:33:50] Executando comando: GPIO 2 -> 1 (msg_id=5)
[2026-06-28 01:33:51] GPIO 2 setado para 1
[2026-06-28 01:33:51] Executando comando: GPIO 4 -> 0 (msg_id=6)
[2026-06-28 01:33:51] GPIO 4 setado para 0
[2026-06-28 01:33:51] Executando comando: GPIO 2 -> 0 (msg_id=6)
[2026-06-28 01:33:51] GPIO 2 setado para 0
[2026-06-28 01:33:51] Executando comando: GPIO 4 -> 1 (msg_id=7)
[2026-06-28 01:33:51] GPIO 4 setado para 1
[2026-06-28 01:33:51] Executando comando: GPIO 2 -> 1 (msg_id=7)
[2026-06-28 01:33:52] GPIO 2 setado para 1
[2026-06-28 01:33:52] Executando comando: GPIO 4 -> 0 (msg_id=8)
[2026-06-28 01:33:52] GPIO 4 setado para 0
[2026-06-28 01:33:52] Executando comando: GPIO 2 -> 0 (msg_id=8)
[2026-06-28 01:33:52] GPIO 2 setado para 0
[2026-06-28 01:33:52] Executando comando: GPIO 4 -> 1 (msg_id=9)
[2026-06-28 01:33:53] GPIO 4 setado para 1
[2026-06-28 01:33:53] Executando comando: GPIO 2 -> 1 (msg_id=9)
[2026-06-28 01:33:53] GPIO 2 setado para 1
[2026-06-28 01:33:53] Executando comando: GPIO 4 -> 0 (msg_id=10)
[2026-06-28 01:33:53] GPIO 4 setado para 0
[2026-06-28 01:33:53] Executando comando: GPIO 2 -> 0 (msg_id=10)
[2026-06-28 01:33:53] GPIO 2 setado para 0
[2026-06-28 01:33:53] Executando comando: GPIO 4 -> 1 (msg_id=11)
[2026-06-28 01:33:53] GPIO 4 setado para 1
[2026-06-28 01:33:54] Executando comando: GPIO 2 -> 1 (msg_id=11)
[2026-06-28 01:33:54] GPIO 2 setado para 1
[2026-06-28 01:33:54] Executando comando: GPIO 4 -> 0 (msg_id=12)
[2026-06-28 01:33:54] GPIO 4 setado para 0
[2026-06-28 01:33:54] Executando comando: GPIO 2 -> 0 (msg_id=12)
[2026-06-28 01:33:54] GPIO 2 setado para 0
[2026-06-28 01:33:54] Executando comando: GPIO 4 -> 1 (msg_id=13)
[2026-06-28 01:33:54] GPIO 4 setado para 1
[2026-06-28 01:33:55] Executando comando: GPIO 2 -> 1 (msg_id=13)
[2026-06-28 01:33:55] GPIO 2 setado para 1
[2026-06-28 01:33:55] Executando comando: GPIO 4 -> 0 (msg_id=14)
```

APÊNDICE L – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 500 MS NO CARTÃO SD

```
[2026-06-28 01:33:55] GPIO 4 setado para 0
[2026-06-28 01:33:55] Executando comando: GPIO 2 -> 0 (msg_id=14)
[2026-06-28 01:33:55] GPIO 2 setado para 0
[2026-06-28 01:33:55] Executando comando: GPIO 4 -> 1 (msg_id=15)
[2026-06-28 01:33:56] GPIO 4 setado para 1
[2026-06-28 01:33:56] Executando comando: GPIO 2 -> 1 (msg_id=15)
[2026-06-28 01:33:56] GPIO 2 setado para 1
[2026-06-28 01:33:56] Executando comando: GPIO 4 -> 0 (msg_id=16)
[2026-06-28 01:33:56] GPIO 4 setado para 0
[2026-06-28 01:33:56] Executando comando: GPIO 2 -> 0 (msg_id=16)
[2026-06-28 01:33:56] GPIO 2 setado para 0
[2026-06-28 01:33:56] Executando comando: GPIO 4 -> 1 (msg_id=17)
[2026-06-28 01:33:57] GPIO 4 setado para 1
[2026-06-28 01:33:57] Executando comando: GPIO 2 -> 1 (msg_id=17)
[2026-06-28 01:33:57] GPIO 2 setado para 1
[2026-06-28 01:33:57] Executando comando: GPIO 4 -> 0 (msg_id=18)
[2026-06-28 01:33:57] GPIO 4 setado para 0
[2026-06-28 01:33:57] Executando comando: GPIO 2 -> 0 (msg_id=18)
[2026-06-28 01:33:57] GPIO 2 setado para 0
[2026-06-28 01:33:58] Executando comando: GPIO 4 -> 1 (msg_id=19)
[2026-06-28 01:33:58] GPIO 4 setado para 1
[2026-06-28 01:33:58] Executando comando: GPIO 2 -> 1 (msg_id=19)
[2026-06-28 01:33:58] GPIO 2 setado para 1
[2026-06-28 01:33:58] Executando comando: GPIO 4 -> 0 (msg_id=20)
[2026-06-28 01:33:58] GPIO 4 setado para 0
[2026-06-28 01:33:58] Executando comando: GPIO 2 -> 0 (msg_id=20)
[2026-06-28 01:33:58] GPIO 2 setado para 0
```

APÊNDICE M – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 100 MS NO BANCO DE DADOS

```
===Banco de dados steph999_Tcc_Project

== Estrutura para tabela gpio_states

|------
|Coluna|Tipo|Nulo|Padrão
|------
|//**id**//|int(11)|Não|
|timestamp|datetime|Sim|NULL
|device_id|varchar(255)|Sim|NULL
|gpio_id|int(11)|Sim|NULL
|state|int(11)|Sim|NULL
|server_timestamp|datetime|Sim|CURRENT_TIMESTAMP
|message_id|varchar(255)|Sim|NULL
== Despejando dados para a tabela gpio_states

|2520|2026-06-17 00:36:53|device1|4|1|2026-06-16 21:36:53|1
|2521|2026-06-17 00:36:53|device1|2|1|2026-06-16 21:36:54|1
|2522|2026-06-17 00:36:54|device1|2|0|2026-06-16 21:36:54|2
|2523|2026-06-17 00:36:54|device1|4|0|2026-06-16 21:36:55|2
|2524|2026-06-17 00:36:55|device1|2|1|2026-06-16 21:36:55|3
|2525|2026-06-17 00:36:55|device1|2|0|2026-06-16 21:36:56|4
|2526|2026-06-17 00:36:56|device1|4|1|2026-06-16 21:36:56|3
|2527|2026-06-17 00:36:56|device1|4|0|2026-06-16 21:36:57|4
|2528|2026-06-17 00:36:57|device1|2|1|2026-06-16 21:36:57|5
|2529|2026-06-17 00:36:57|device1|4|1|2026-06-16 21:36:58|5
|2530|2026-06-17 00:36:58|device1|2|0|2026-06-16 21:36:58|6
|2531|2026-06-17 00:36:58|device1|4|0|2026-06-16 21:36:59|6
|2532|2026-06-17 00:36:59|device1|2|1|2026-06-16 21:36:59|7
|2533|2026-06-17 00:36:59|device1|4|1|2026-06-16 21:37:00|7
|2534|2026-06-17 00:37:00|device1|2|0|2026-06-16 21:37:00|8
|2535|2026-06-17 00:37:00|device1|4|0|2026-06-16 21:37:01|8
|2536|2026-06-17 00:37:01|device1|4|1|2026-06-16 21:37:01|9
|2537|2026-06-17 00:37:01|device1|2|1|2026-06-16 21:37:02|9
|2538|2026-06-17 00:37:02|device1|2|0|2026-06-16 21:37:02|10
|2539|2026-06-17 00:37:02|device1|4|0|2026-06-16 21:37:03|10
|2540|2026-06-17 00:37:03|device1|2|1|2026-06-16 21:37:04|11
|2541|2026-06-17 00:37:04|device1|4|1|2026-06-16 21:37:04|11
|2542|2026-06-17 00:37:04|device1|2|0|2026-06-16 21:37:05|12
|2543|2026-06-17 00:37:05|device1|4|0|2026-06-16 21:37:05|12
|2544|2026-06-17 00:37:05|device1|2|1|2026-06-16 21:37:06|13
|2545|2026-06-17 00:37:06|device1|4|1|2026-06-16 21:37:06|13
|2546|2026-06-17 00:37:06|device1|2|0|2026-06-16 21:37:07|14
|2547|2026-06-17 00:37:07|device1|4|0|2026-06-16 21:37:07|14
|2548|2026-06-17 00:37:07|device1|2|1|2026-06-16 21:37:08|15
|2549|2026-06-17 00:37:08|device1|4|1|2026-06-16 21:37:08|15
|2550|2026-06-17 00:37:08|device1|2|0|2026-06-16 21:37:09|16
|2551|2026-06-17 00:37:09|device1|4|0|2026-06-16 21:37:09|16
|2552|2026-06-17 00:37:09|device1|2|1|2026-06-16 21:37:10|17
|2553|2026-06-17 00:37:10|device1|4|1|2026-06-16 21:37:10|17
|2554|2026-06-17 00:37:10|device1|2|0|2026-06-16 21:37:11|18
|2555|2026-06-17 00:37:11|device1|4|0|2026-06-16 21:37:11|18
|2556|2026-06-17 00:37:11|device1|2|1|2026-06-16 21:37:12|19
|2557|2026-06-17 00:37:12|device1|2|0|2026-06-16 21:37:12|20
```

APÊNDICE N – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 100 MS NO BANCO DE DADOS

```
|2558|2026-06-17 00:37:12|device1|4|1|2026-06-16 21:37:12|19
|2559|2026-06-17 00:37:13|device1|4|0|2026-06-16 21:37:13|20
```

APÊNDICE O – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 100 MS NO CARTÃO SD

```
[2026-06-17 00:36:32] Sistema iniciado!
[2026-06-17 00:36:53] Executando comando: GPIO 4 -> 1 (msg_id=1)
[2026-06-17 00:36:53] GPIO 4 setado para 1
[2026-06-17 00:36:53] Executando comando: GPIO 2 -> 1 (msg_id=1)
[2026-06-17 00:36:53] GPIO 2 setado para 1
[2026-06-17 00:36:53] Executando comando: GPIO 2 -> 0 (msg_id=2)
[2026-06-17 00:36:53] GPIO 2 setado para 0
[2026-06-17 00:36:53] Executando comando: GPIO 4 -> 0 (msg_id=2)
[2026-06-17 00:36:54] GPIO 4 setado para 0
[2026-06-17 00:36:54] Executando comando: GPIO 2 -> 1 (msg_id=3)
[2026-06-17 00:36:54] GPIO 2 setado para 1
[2026-06-17 00:36:54] Executando comando: GPIO 2 -> 0 (msg_id=4)
[2026-06-17 00:36:54] GPIO 2 setado para 0
[2026-06-17 00:36:54] Executando comando: GPIO 4 -> 1 (msg_id=3)
[2026-06-17 00:36:54] GPIO 4 setado para 1
[2026-06-17 00:36:55] Executando comando: GPIO 4 -> 0 (msg_id=4)
[2026-06-17 00:36:55] GPIO 4 setado para 0
[2026-06-17 00:36:55] Executando comando: GPIO 2 -> 1 (msg_id=5)
[2026-06-17 00:36:55] GPIO 2 setado para 1
[2026-06-17 00:36:55] Executando comando: GPIO 4 -> 1 (msg_id=5)
[2026-06-17 00:36:55] GPIO 4 setado para 1
[2026-06-17 00:36:55] Executando comando: GPIO 2 -> 0 (msg_id=6)
[2026-06-17 00:36:55] GPIO 2 setado para 0
[2026-06-17 00:36:56] Executando comando: GPIO 4 -> 0 (msg_id=6)
[2026-06-17 00:36:56] GPIO 4 setado para 0
[2026-06-17 00:36:56] Executando comando: GPIO 2 -> 1 (msg_id=7)
[2026-06-17 00:36:56] GPIO 2 setado para 1
[2026-06-17 00:36:56] Executando comando: GPIO 4 -> 1 (msg_id=7)
[2026-06-17 00:36:56] GPIO 4 setado para 1
[2026-06-17 00:36:56] Executando comando: GPIO 2 -> 0 (msg_id=8)
[2026-06-17 00:36:57] GPIO 2 setado para 0
[2026-06-17 00:36:57] Executando comando: GPIO 4 -> 0 (msg_id=8)
[2026-06-17 00:36:57] GPIO 4 setado para 0
[2026-06-17 00:36:57] Executando comando: GPIO 4 -> 1 (msg_id=9)
[2026-06-17 00:36:57] GPIO 4 setado para 1
[2026-06-17 00:36:57] Executando comando: GPIO 2 -> 1 (msg_id=9)
[2026-06-17 00:36:57] GPIO 2 setado para 1
[2026-06-17 00:36:58] Executando comando: GPIO 2 -> 0 (msg_id=10)
[2026-06-17 00:36:58] GPIO 2 setado para 0
[2026-06-17 00:36:58] Executando comando: GPIO 4 -> 0 (msg_id=10)
[2026-06-17 00:36:58] GPIO 4 setado para 0
[2026-06-17 00:36:58] Executando comando: GPIO 2 -> 1 (msg_id=11)
[2026-06-17 00:36:58] GPIO 2 setado para 1
[2026-06-17 00:36:58] Executando comando: GPIO 4 -> 1 (msg_id=11)
[2026-06-17 00:36:59] GPIO 4 setado para 1
[2026-06-17 00:36:59] Executando comando: GPIO 2 -> 0 (msg_id=12)
[2026-06-17 00:36:59] GPIO 2 setado para 0
[2026-06-17 00:36:59] Executando comando: GPIO 4 -> 0 (msg_id=12)
[2026-06-17 00:36:59] GPIO 4 setado para 0
[2026-06-17 00:36:59] Executando comando: GPIO 2 -> 1 (msg_id=13)
[2026-06-17 00:36:59] GPIO 2 setado para 1
[2026-06-17 00:37:00] Executando comando: GPIO 4 -> 1 (msg_id=13)
[2026-06-17 00:37:00] GPIO 4 setado para 1
[2026-06-17 00:37:00] Executando comando: GPIO 2 -> 0 (msg_id=14)
```

APÊNDICE P – REGISTROS DO TESTE DE ESTRESSE COM DUAS GPIOS E INTERVALO DE 100 MS NO CARTÃO SD

```
[2026-06-17 00:37:00] GPIO 2 setado para 0
[2026-06-17 00:37:00] Executando comando: GPIO 4 -> 0 (msg_id=14)
[2026-06-17 00:37:00] GPIO 4 setado para 0
[2026-06-17 00:37:00] Executando comando: GPIO 2 -> 1 (msg_id=15)
[2026-06-17 00:37:01] GPIO 2 setado para 1
[2026-06-17 00:37:01] Executando comando: GPIO 4 -> 1 (msg_id=15)
[2026-06-17 00:37:01] GPIO 4 setado para 1
[2026-06-17 00:37:01] Executando comando: GPIO 2 -> 0 (msg_id=16)
[2026-06-17 00:37:01] GPIO 2 setado para 0
[2026-06-17 00:37:01] Executando comando: GPIO 4 -> 0 (msg_id=16)
[2026-06-17 00:37:01] GPIO 4 setado para 0
[2026-06-17 00:37:02] Executando comando: GPIO 2 -> 1 (msg_id=17)
[2026-06-17 00:37:02] GPIO 2 setado para 1
[2026-06-17 00:37:02] Executando comando: GPIO 4 -> 1 (msg_id=17)
[2026-06-17 00:37:02] GPIO 4 setado para 1
[2026-06-17 00:37:02] Executando comando: GPIO 2 -> 0 (msg_id=18)
[2026-06-17 00:37:02] GPIO 2 setado para 0
[2026-06-17 00:37:02] Executando comando: GPIO 4 -> 0 (msg_id=18)
[2026-06-17 00:37:02] GPIO 4 setado para 0
[2026-06-17 00:37:03] Executando comando: GPIO 2 -> 1 (msg_id=19)
[2026-06-17 00:37:03] GPIO 2 setado para 1
[2026-06-17 00:37:03] Executando comando: GPIO 2 -> 0 (msg_id=20)
[2026-06-17 00:37:03] GPIO 2 setado para 0
[2026-06-17 00:37:03] Executando comando: GPIO 4 -> 1 (msg_id=19)
[2026-06-17 00:37:03] GPIO 4 setado para 1
[2026-06-17 00:37:03] Executando comando: GPIO 4 -> 0 (msg_id=20)
[2026-06-17 00:37:03] GPIO 4 setado para 0
```