Universidade Federal de Ouro Preto
Escola de Minas
CECAU - Colegiado do Curso de
Engenharia de Controle e Automação

Marcus Vinícius Fernandes Pavão de Souza

Monitoramento de Controladores Lógicos Programáveis via Internet das Coisas

Monografia de Graduação

Ouro Preto, 2024

Marcus Vinícius Fernandes Pavão de Souza

Monitoramento de Controladores Lógicos Programáveis via Internet das Coisas

Trabalho apresentado ao Colegiado do Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como parte dos requisitos para a obtenção do Grau de Engenheira(o) de Controle e Automação.

Universidade Federal de Ouro Preto

Orientador: Prof. Dr. Alan Kardek Rêgo Segundo

Ouro Preto

2024

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S729m Souza, Marcus Vinicius Fernandes Pavao de.
Monitoramento de Controladores Lógicos Programáveis via Internet das Coisas. [manuscrito] / Marcus Vinicius Fernandes Pavao de Souza. - 2024.
61 f.: il.: color., gráf.. + Listagem.

Orientador: Prof. Dr. Alan Kardek Rêgo Segundo.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Escola de Minas. Graduação em Engenharia de Controle e Automação .

1. Internet das Coisas (IoT). 2. Microcontroladores. 3. Message Queuing Telemetry Transport (MQTT). 4. Controladores Lógicos Programáveis (CLP). I. Segundo, Alan Kardek Rêgo. II. Universidade Federal de Ouro Preto. III. Título.

CDU 681.5

Bibliotecário(a) Responsável: Maristela Sanches Lima Mesquita - CRB-1716

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
ESCOLA DE MINAS
DEPARTAMENTO DE ENGENHARIA CONTROLE E AUTOMACAO

FOLHA DE APROVAÇÃO

Marcus Vinícius Fernandes Pavão de Souza

Monitoramento de Controladores Lógicos Programáveis via Internet das Coisas

Monografia apresentada ao Curso de Engenharia de Controle e Automação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Engenheiro de Controle e Automação

Aprovada em 12 de setembro de 2024

Membros da banca

Dr. Alan Kardek Rêgo Segundo - Orientador (Universidade Federal de Ouro Preto)
Dr. Agnaldo José da Rocha Reis - Convidado (Universidade Federal de Ouro Preto)
Dra. Adrielle de Carvalho Santana - Convidada (Universidade Federal de Ouro Preto)

Alan Kardek Rêgo Segundo, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 09/10/2024

Documento assinado eletronicamente por **Alan Kardek Rego Segundo**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 09/10/2024, às 16:22, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **0776567** e o código CRC **DE4CE854**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: 3135591533 - www.ufop.br

Resumo

Analisa-se nesta monografia o uso de microcontroladores como um meio para interligar dados utilizados e tratados por controladores lógicos programáveis (CLPs) com a nuvem. O trabalho aborda tanto os desafios quanto as oportunidades que emergem ao conectar dispositivos industriais a plataformas de Internet das Coisas (IoT), permitindo uma atualização dos processos existentes com o mínimo de alterações necessárias em sistemas já consolidados. Concentra-se em como os microcontroladores, como o ESP32, podem ser integrados a esses sistemas, utilizando protocolos como MQTT e Modbus TCP para comunicação em diferentes etapas do processo. A proposta inclui ainda o desenvolvimento de uma interface que permite a visualização dos dados, alinhando a expectativa do projeto com demandas atuais como a conectividade de sistemas industriais e análise de dados complexos. O estudo proposto visa a criação de uma solução de baixo custo que possibilita o armazenamento de dados e o monitoramento de dados.

Palavras-chave: IoT. Microcontroladores. Controlador Lógico Programável.

Abstract

This capstone analyzes the use of microcontrollers as a means to link data processed by programmable logic controllers (PLCs) with the cloud. The project addresses both the challenges and opportunities that arise from connecting industrial devices to Internet of Things (IoT) platforms, enabling the modernization of existing processes with minimal changes to established systems. It focuses on how microcontrollers, such as the ESP32, can be integrated into these systems using protocols like MQTT and Modbus TCP for communication at various stages of the process. The proposal also includes the development of an interface to visualize the data, aligning the project's expectations with current demands such as industrial system connectivity and complex data analysis. The proposed study aims to create a low-cost solution that enables both data storage and monitoring.

Key-words: IoT. Microcontrollers. Programmable Logic Controller.

Lista de abreviaturas e siglas

IoT	Internet of Things
CLP	Controlador Lógico Programável
AWS	Amazon Web Services
MVP	Minimal Viable Product
IDE	Integrated Development Environment
SSH	Secure Shell
MQTT	Message Queuing Telemetry Transport
TSDB	Time series database
WS	WebSockets

Lista de ilustrações

Sumário

1 Introdução

No setor industrial, o chão de fábrica é onde os produtos são efetivamente produzidos e gerados. É o ambiente que abriga as máquinas mais pesadas e, consequentemente, o lugar de maior insalubridade e risco de acidentes. Para lidar com os equipamentos de grande magnitude exigidos pelo chão de fábrica, os dispositivos que controlam e automatizam o funcionamento também precisam ser mais robustos do que aqueles encontrados em outras etapas do ciclo de produção industrial.

Uma vez que se tratam de equipamentos mais antigos devido à confiabilidade e também experiência de técnicos com tais equipamentos, a troca desses Controladores Lógicos Programáveis (CLPs) tende a ser evitada e adiada. Conforme citado no manual da Texas Instruments (INSTRUMENTS, 2013), estes dispositivos têm um ciclo de duração recomendadas pelo fabricante de, em média, cem mil horas, que em uso interrupto seria cerca de doze anos. Destaca-se que em diversos casos o tempo de duração foi extrapolado (ELECTRIC, 2013), porém houve casos onde a durabilidade foi menor que a média (AUTOMATION, 2024).

O CLP SIMATIC S7-1200/1500, segundo a Siemens AG (2021), este já possui suporte a conexão por meio do protocolo Message Queuing Telemetry Transport (MQTT) e também ao protocolo TCP/IP que facilita a comunicação, cria novas possibilidades de integração principalmente com sistemas IoT (SIEMENS, 2021). Seguindo essa tendência a empresa Altus anunciou uma nova linha de CLPs que destaca a capacidade de conexão MQTT (ALTUS, 2021). Os dispositivos da série Nexto, percorrem uma ampla quantidade de características, entre eles, existe o NX3030 dispositivo de alto desempenho com suporte aos protocolos Profinet, Modbus RTU e TCP e ao MQTT.

Um dos destaques nesse contexto é o Finder Arduino Opta. Esta solução foi desenvolvida pela empresa Finder em parceria com a Arduino Pro, visando combinar a facilidade de programação dos microcontroladores Arduino com a segurança garantida pelo padrão X.509 (FINDER, 2021). Segundo o fabricante, o Opta é o primeiro Relé Lógico Programável (FINDER, 2023). Ele oferece, entre outras possibilidades, a programação por meio dos códigos *open-source* feitos para Arduino, bem como a programação em Ladder seguindo a norma IEC 61131-3 (PLCOPEN, 2004).

Um dos impeditivos em relação ao uso de tais tecnologias são os preços normalmente praticados. Enquanto um CLP convencional, como, por exemplo, o

modelo Logo! da Siemens, pode ser adquirido por um preço acessível que gira em torno de mil reais (DIMENSIONAL, 2024), o Opta com Wifi (recurso presente em todas as versões do ESP) chega a custar quase três vezes mais em alguns sites que o disponibilizam para compra - Arduino Opta WiFi. Além do preço elevado, esses dispositivos são mais difíceis de serem encontrados, embora estejam se tornando cada vez mais padrão dentro da indústria. Outro ponto que corrobora com a linha de pensamento apresentada é que a curva de aprendizagem envolvida na programação de cada CLP pode, em determinadas situações, apresentar-se como um fator para adiar a sua troca. Apesar da norma Ladder ser padronizada, os fabricantes possuem *softwares* distintos, cada um com suas características e formas de uso.

Preocupações quanto à exposição de dispositivos sensíveis e diretamente ligados ao fluxo industrial, como os CLPs, são uma causa adicional de receio na implementação de soluções completas. Com isso, a criação de uma interface entre os dados e o usuário, representa um passo inicial para a comunicação e automação mais avançadas nos processos industriais, principalmente em ambientes utilizando controladores em seus últimos anos de vida útil, ou recomendados. Devido à complexidade e aos custos envolvidos na substituição desses CLPs, existe um certo receio na realização da troca desses dispositivos, preferindo-se manter os controladores existentes em operação. Esse cenário reforça a necessidade de desenvolver soluções que possam coexistir com esses controladores antigos, permitindo uma modernização gradual e garantindo a continuidade das operações.

Para integração, a ideia se baseia na junção de um CLP a um microcontrolador por meio de um protocolo de rede industrial, a fim de garantir eficiência e confiabilidade. O microcontrolador envia os dados para um servidor e as possibilidades para suprir a função desse elemento no projeto são inúmeras, desde serviços como Azure, Google Cloud e AWS como citados no artigo (CODE B, 2024), ou mesmo um servidor BackEnd específico para a solução.

Alguns trabalhos da literatura apresentam alternativas de interface entre PLCs e a nuvem, usando microcontroladores. Ali, Miry e Salman (2020) explora o controle de nível de um tanque utilizando tecnologias como os protocolos Modbus e MQTT, assim como dispositivos da família ESP, associados ao uso de controladores lógicos programáveis.

Em Ferreira, Bigheti e Godoy (2019) o assunto é trabalhado com utilizações semelhantes ao preterido durante o trabalho. Estando interligados principalmente através do termo *Industrial Internet of Things* (IIoT) que representa as pretensões do projeto. Já no trabalho de Wawhal (2019) toda a comunicação entre a utilização

do BackEnd em Hasura com o FrontEnd construído em React é exposta, para disponibilização dos gráficos ao usuário final.

Quando se trata de SoC (*System on Chip*) para IoT, características como *WiFi* e *Bluetooth* desde as versões mais básicas fazem com que o ESP32 da Espressif Systems se destaque (ESPRESSIF SYSTEMS, 2020). Sua grande versatilidade permite seu uso tanto didaticamente quanto para protótipos e até mesmo para produtos finais. Estes são constantemente citados em diversos de artigos, alguns exemplos são, *Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things* (MAIER; SHARP; VAGAPOV, 2017) e *ESP32 Based Smart Surveillance System* (BROWN; DAVIS, 2022). Seu fácil uso e programação simples por meio de *IDEs* (*Integrated Development Environment* do inglês ambiente de desenvolvimento integrado) como a do Arduino fazem com que ele assuma papel de relevância na hora de escolher um dispositivo para realizar comunicação com a nuvem.

1.1 Objetivos

1.1.1 Objetivo geral

Este trabalho tem por objetivo geral o desenvolvimento de uma interface de comunicação entre um PLC e a nuvem, utilizando um SoC.

1.1.2 Objetivos específicos

Alguns objetivos subsequentes incluem a análise dos protocolos de comunicação para a implementação da conexão entre o CLP e o dispositivo de conexão com a internet. A análise de dispositivos também deve ser destacada, pois é essencial entender como criar uma solução que possa reduzir custos, aumentar a eficiência, além de melhorar a segurança e a versatilidade de acesso. Outro ponto de atenção dentro do trabalho é a criação um conteúdo de qualidade que possa ser utilizado como base para futuros trabalhos, mantendo a coerência e didática.

1.2 Justificativa e Relevância

Assim como explicado durante a introdução do problema, o tema internet das coisas, ou como é normalmente conhecido IoT, do inglês *Internet of Things*, é citado de forma constante como uma revolução na forma como a manipulação e distribuição

de dados são e serão feitas. Com as grandes empresas de tecnologia, as famosas Big Techs (Google, Amazon e Microsoft), investindo e dando cada vez mais importância a computação em nuvem.

O artigo "IoT: Trends, Challenges and Future Scope"(VYAS; BHATT; JHA, 2024) destaca a importância e o futuro de diversas áreas integradas por esse assunto, como saúde, indústria e a agricultura.

A ideia de unir a Internet das Coisas com controladores lógicos programáveis faz todo o sentido, especialmente ao considerar a possibilidade de adicionar novas funcionalidades aos equipamentos antigos e já instalados, sem a necessidade de substituir todos os CLPs da planta por modelos mais modernos, o que pode ser economicamente inviável. Essa abordagem permite aproveitar os dispositivos consolidados no mercado, promovendo avanços tecnológicos no chão de fábrica de maneira mais acessível e eficiente.

Possibilitar a análise de dados de forma remota também permite um grande aumento no uso de técnicas de computação que possam identificar e resolver problemas presentes nos dados tratados pelos controladores, mas que não conseguiriam ser detectados sem o uso de computação pesada, a qual envolve e necessita de outro nível de poder computacional.

1.3 Organização do texto

No primeiro capítulo foi apresentado toda a introdução ao trabalho, explicando a relevância do projeto, com textos-base e referências, assim como os objetivos e justificativa para o projeto.

No capítulo 2, apresenta-se a revisão bibliográfica, que oferece uma base para o tema do trabalho, abordando principalmente as questões técnicas, como conceitos e tecnologias utilizadas durante a construção da solução.

O capítulo 3 apresenta o desenvolvimento do trabalho, apresentando o diagrama principal do projeto, os dispositivos e tecnologias são explicadas. Durante o capítulo tem-se a explicação da construção e desenvolvimento da solução proposta.

No capítulo 4, é feita a exposição dos resultados do trabalho, acompanhados de considerações e discussões a respeito dos testes realizados. Por fim, no capítulo 5 é feita a conclusão do trabalho, na qual são apresentadas as possibilidades de futuras melhorias.

2 Revisão Bibliográfica

2.1 Módulo ESP32

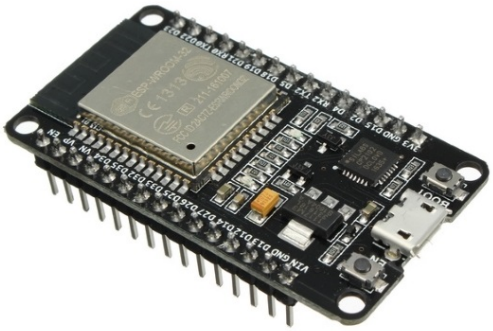

Figura 1 – Placa de desenvolvimento ESP-WROOM32-02. Fonte: (ESPRESSIF SYSTEMS, 2024)

.

O microcontrolador ESP32, apresentado na Figura 1, foi desenvolvido pela empresa de tecnologia Espressif Systems, apresentado no mercado em 2016, ou seja, mesmo não estando há tanto tempo quanto alguns outros microcontroladores, é considerado um dos mais relevantes controladores dessa área. Dessa forma, o microcontrolador ESP32 é composto por um processador, projetado com um modelo *single* ou *dual-core de 32 bits*, que é capaz de trabalhar com uma frequência de até 240 MHz, além da sua capacidade de armazenamento maior que os demais microcontroladores Arduino comumente utilizados. À vista disso, é possível perceber que o microcontrolador ESP32 é capaz de alcançar o dobro tamanho de memória *flash*, se comparado alguns modelos Arduino mais comuns, apesar de alguns dispositivos como o Arduino Portenta trabalharem em frequências de até 480 MHz e acabam se encaixam em diferentes faixas de uso e de preço (EMBARCADOS, 2024b).

Dentre as características mais interessantes desse dispositivo, se destacam a conectividade excepcional do ESP32, que se dá em dois módulos ímpares de integração, associado ao seu chip, tendo acesso às redes de transmissão, por meio das

ondas de rádio, representado pelo *Bluetooth*, e o *Wi-Fi*, sendo o principal SoC de pequeno porte que reúne essas propriedades, responsáveis por tornar essa plataforma econômica e de alta empregabilidade.

Ademais, com o uso dos diversos *softwares* compatíveis, a programação do ESP32 se torna mais fácil, revelando a linguagem C/C++, que pode ser explorada pelo programa do kit de desenvolvimento de *software* (SDK) fornecido pela própria desenvolvedora. Por fim, evidencia-se a utilização do ESP32 para a execução de projetos de baixo custo e alta conectividade.[1]

2.2 Internet das Coisas

A Internet das Coisas, também conhecida como *Internet of Things* (IoT), tem gerado bastante atenção, desde o ambiente acadêmico até os meios industriais, tendo em vista seu potencial. Nesse sentido, é possível observar que a IoT surgiu a partir dos avanços tecnológicos de diversas áreas, entre elas a microeletrônica e o sensoriamento. De forma resumida, a Internet das Coisas é uma extensão da Internet, capaz de fazer com que os objetos, de caráter computacional e de comunicação, conectem-se à Internet.[2]

Uma ramificação da área de estudo Internet das Coisas (IoT) é a IIoT (*Industrial Internet of Things*), que pode ser traduzida como a Internet Industrial das Coisas. Essa subárea foca na aplicação dos conceitos de IoT em contextos industriais. Dessa forma, busca entender a conexão de dispositivos entre máquinas, dispositivos, sensores e demais componentes que fazem parte do fluxo industrial com a internet. A IIoT tem uma visão voltada para o monitoramento ativo e a busca por maior eficiência, conseguindo visualizar dados que podem por muitas vezes terem passados despercebidos.

Essa conexão, então, por meio de uma rede mundial, torna viável o controle remoto de objetos, além de permitir que eles próprios sejam acessados como provedores. Como supracitado, essas especificidades dos objetos é responsável por criar uma grande porção de oportunidades tanto na academia, quanto no meio industrial. No entanto, essas oportunidades podem gerar riscos e alguns desafios técnicos.

É possível observar que a Internet das Coisas tem transformado o conceito de redes de computadores, evidenciando uma evolução do conceito (TANENBAUM; STEEN, 2007). A tecnologia responsável por conectar o conjunto de computadores

[1] Willian et al. (2019).
[2] Santos et al. (2016).

pode ser de diferentes tipos, como o fio de cobre, fibra ótica, etc. Nesse sentido, se destaca uma certa generalidade das Redes de Computadores, ou seja, são dadas em aparelhos de cunho geral, e não são otimizadas para fins específicos.

Os dispositivos inteligentes se relacionam diretamente com essa evolução, uma vez que possuem alta utilidade em comunicação e em processamento aliados a sensores. Além disso, não são só os computadores tradicionais que estão conectados à rede, mas diversos equipamentos, como TVs, automóveis e smartphones. Isto é, nota-se uma crescente pluralidade em relação à conexão dos dispositivos.

Vale ressaltar que um fator fundamental para o desenvolvimento da Internet das Coisas é a padronização das tecnologias, fazendo com que a heterogeneidade dos aparelhos conectados cresça, tornando a Internet das Coisas uma realidade. Dessa forma, os dispositivos contemplados por algumas dessas especificidades, por meio da unidade básica de *hardware*, são denominados objetos inteligentes. Os *Smart Objects*, ao estabelecerem comunicação com os demais aparelhos, evidenciam o conceito de estar em rede.

2.3 Controlador Lógico Programável

Figura 2 – Controlador Lógico Programável - 8Di Logo Siemens. Fonte: (DIMENSIONAL, 2024)

Os controladores lógicos programáveis, como o exemplificado por meio da Figura 2, são atualmente a tecnologia de controle de processos industriais mais utilizados. Um Controlador Lógico Programável (CLP) é considerado um tipo de

computador, utilizado em escala industrial, capaz de ser programado para realizar o controle. Nesse sentido, os controladores são responsáveis por reduzir a fiação dos circuitos de controle convencional a ralé, apresentando também a facilidade de instalação, compatibilidade de rede, alta confiabilidade, entre outras especificidades.

O CLP é projetado em arranjos de várias entradas e saídas sendo programado para operações de equipamentos referentes ao contexto industrial e são armazenados em memória não volátil geralmente. As saídas dependem diretamente das condições de entrada impostas dentro do ambiente industrial, garantindo que as respostas do sistema sejam precisas e previsíveis, permitindo o controle efetivo e continuo dos processos.

De forma resumida, o CLP é um computador digital para ser utilizado no controle de máquinas, que se difere de computadores pessoais, uma vez que é projetado para ambientes industriais e se mostra equipado por diferentes interfaces. Em seu surgimento, o CLP era usado primariamente para substituir o Relé como foi discutido no trecho a seguir "Os Controladores Lógico Programáveis (CLP's) foram desenvolvidos na década de 60, com a finalidade de substituir painéis de relés que eram utilizados nas indústrias automobilísticas para executar controles baseados em lógicas combinacional e sequencial" (EMBARCADOS, 2024a), mas devido às suas crescentes funções, ele pode ser utilizado em aplicações mais complexas.

Esses controladores oferecem diversas vantagens em relação aos demais controladores tradicionais. A maioria do trabalho com fiação é eliminado com o CLP e, apesar dos controles modernos ainda incluírem relés, eles são raramente utilizados para a lógica. Entre os principais benefícios referentes ao uso do Controlador Lógico Programável se destacam a redução de custos, maior confiabilidade, mais flexibilidade, capacidade de comunicação, tempo de resposta rápido e a facilidade na verificação de defeitos.

2.4 Protocolo Modbus

Figura 3 – Logo do protocolo Modbus utilizado pela Modbus Organization. Fonte: (MODBUS ORGANIZATION, 2024)

O protocolo Modbus, identificado por meio do logo exposto na Figura 3, projetado pela atual Schneider, antes conhecida como Modicon Industrial Automation Systems, a fim de promover a comunicação de um dispositivo mestre com os demais, independente da rede utilizada. Esse protocolo, então, delimita uma estrutura de mensagens dadas por bytes.

Modbus é um protocolo que utiliza o modelo de *request-response*, previamente conhecido como *master-slave*, onde por meio de barramentos de comunicação enquanto um dispositivo envia ou outros recebem. As formas de transmissão do protocolo fazem com que este se destaque, além da grande confiabilidade já comprovada pela indústria. Seu uso mais comum é a comunicação entre um sistema supervisório ou interface homem-máquina, e CLPs e sensores.

Seu uso normalmente se dá por conexões seriais nos padrões RS-232 e RS-485, mas pode ser utilizado como protocolo de camadas de redes industriais. Especificamente na área das Redes Industriais, o protocolo Modbus é o mais utilizado, devido à sua grande facilidade de implementação e simplicidade, além de não precisar de licença como alguns de seus concorrentes diretos como Profibus e Profinet. Como supracitado, o protocolo Modbus é feito a partir do modelo mestre-escravo, em que um único dispositivo pode iniciar a comunicação, enquanto os demais respondem às solicitações.

Atualmente é possível observar as aplicações do uso do protocolo Modbus em diversas áreas, abrangendo os mercados de controle de motores até mesmo o controle de poços de petróleo. Evidencia-se que a utilização de redes industriais é fundamental para gerenciar determinados processos, evitando as perdas e as falhas de produtividade. Especificamente com o uso do protocolo Modbus, se torna

evidente a forma objetiva de desenvolver ações em redes industriais que atende várias necessidades a partir de uma estrutura de simples instalação.[3]

2.4.1 Modbus TCP

Modbus TCP é uma variante do protocolo Modbus que funciona sobre o protocolo TCP/IP, permitindo a comunicação mediante redes Ethernet. A principal vantagem do Modbus TCP é sua capacidade de integrar dispositivos em uma rede Ethernet padrão, o que facilita a configuração e a expansão de redes industriais, podendo ser utilizada de forma eficiente em ambientes locais.

Os pacotes gerados por meio do Modbus TCP encapsulam mensagens Modbus em pacotes TCP/IP, mantendo a compatibilidade com o protocolo original, fazendo com que dispositivos projetados para Modbus Serial sejam facilmente adaptados para comunicação via Modbus TCP com pouca ou nenhuma modificação. A estrutura básica de um pacote Modbus TCP inclui um cabeçalho de aplicação (MBAP) seguido da mensagem Modbus propriamente dita, permitindo a identificação e roteamento dos dados dentro da rede Ethernet.[4]

2.4.2 Modbus Serial

O Modbus Serial é a forma mais tradicional de implementação do protocolo Modbus, operando sobre interfaces físicas seriais como RS-232, RS-485 e RS-422. No modo serial, a comunicação ocorre no seguinte formato, um único mestre pode se comunicar com múltiplos dispositivos escravos conectados à mesma linha de transmissão. Os modos de transmissão mais comuns para Modbus Serial são ASCII (American Code for Information Interchange) e RTU (Remote Terminal Unit). No modo ASCII, os dados são codificados em caracteres ASCII, facilitando a leitura e depuração dos pacotes. Já no modo RTU, os dados são compactados em um formato binário, aumentando a eficiência e a velocidade de transmissão.

A configuração dos parâmetros de comunicação, como *baud rate*, paridade e bits de parada, é crucial para garantir uma comunicação eficiente e livre de erros. Devido à simplicidade e baixo custo de implementação, o Modbus Serial é amplamente utilizado em sistemas de controle industrial, sendo mais comum que sua versão que utiliza o protocolo TCP/IP.[5]

[3] Nascimento e Lucena (2003).
[4] MODBUS... (2002).
[5] MODBUS... (2002).

2.5 MQTT

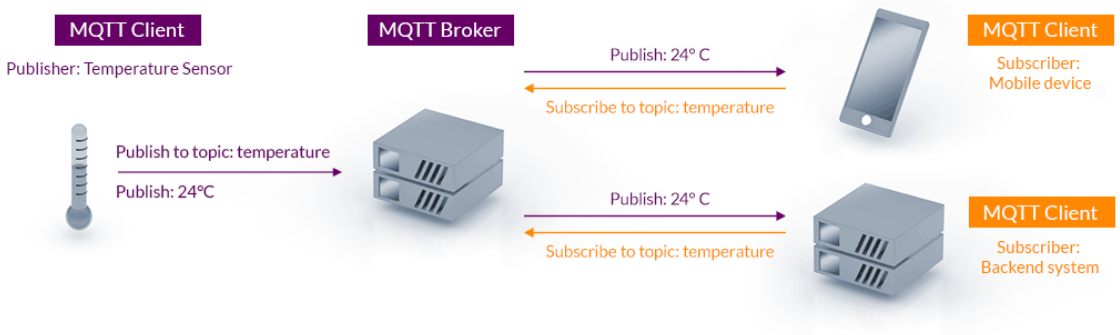

Figura 4 – Diagrama do funcionamento básico de um sistema de comunicação por meio do módulo MQTT. Fonte: Organização MQTT (MQTT ORGANIZATION, 2024).

MQTT (Message Queuing Telemetry Transport) é um protocolo de mensagens leve, projetado para dispositivos com largura de banda e recursos limitados. Ele foi desenvolvido pela IBM na década de 1990 e desde então se tornou um dos padrões dominantes para comunicação entre dispositivos na Internet das Coisas (IoT). O MQTT é baseado no modelo de publicação/assinatura, onde os dispositivos se comunicam via um intermediário denominado *broker*, que gerencia a troca de mensagens. Isso permite uma comunicação eficiente e assíncrona, ideal para os dispositivos que fazem tanto o envio quando recebimento. Essas características podem ser observadas na Figura 4.

Apesar de diversas vantagens, esse protocolo também apresenta desafios, como os expostos a seguir, que precisam ser cuidadosamente considerados ao implementar soluções baseadas nesse protocolo.

Umas das características mais interessantes é a função QoS (Quality of Service) que oferece três níveis cada um dando um nível de importância distinto a mensagem. São 3 níveis, 0, 1 e 2, sendo que o nível 0 não possui garantia de entrega. No nível 1, as mensagens são entregues pelo menos uma vez, mas podem resultar em entregas duplicadas. Já no nível 2, as mensagens são entregues exatamente uma vez, podendo resultar em maior sobrecarga de rede e latência.

Outro problema envolve o fato que o protocolo MQTT assume uma arquitetura cliente-servidor, onde os clientes (dispositivos) se conectam a um servidor MQTT. Se a conexão do cliente for interrompida ou perdida, as mensagens podem não ser entregues até que o cliente se reconecte, resultando em potencial perda de mensagens.

Para mitigar esses problemas, é importante considerar cuidadosamente a configuração do QoS, implementar estratégias de reconexão robustas para dispositivos clientes, monitorar e dimensionar adequadamente os servidores MQTT e, se necessário, complementar o MQTT com outros protocolos ou tecnologias para garantir uma entrega confiável de mensagens em ambientes específicos.

2.6 Banco de Dados

Os bancos de dados são sistemas organizados utilizados primariamente para o armazenamento de dados, permitindo a persistência dos mesmos, garantindo a execução de sistemas mesmo após possíveis interrupções e mudanças de contexto, garantindo integridade às aplicações utilizadas.

Eles são fundamentais para a operação de sistemas da informação sendo indispensáveis para construção de aplicações que prezem pela integridade e simplicidade, e para tanto se distribuem entre as mais diferentes formas. Alguns dos exemplos de modelos de bancos de dados podem ser vistos nos mais comumente utilizados bancos de dados relacionais (Bancos SQL), os bancos não relacionais (Bancos NoSQL), e ainda um tipo que deriva de certa forma dos bancos relacionais, os bancos de séries temporais. Existem ainda alguns menos utilizados como os bancos de dados orientados a grafos e bancos de dados orientados a objetos.

2.6.1 Bancos Relacionais

Os bancos de dados relacionais, RDBMS - *Relational Database Management System*, são um tipo e uma forma de banco de dados que organiza os dados em tabelas com linhas e colunas. A principal característica dos bancos de dados relacionais é o uso da linguagem SQL (Structured Query Language) para a manipulação e consulta dos dados registrados nesses sistemas.

Os principais exemplos de bancos de dados relacionais são o MySQL, PostgreSQL, Oracle Database e Microsoft SQL Server.

Os bancos de dados relacionais garantem a integridade e a consistência dos dados por meio do uso de restrições e transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), sendo ponto fundamental dos SGBD's, e garantem que seu uso sejam indispensáveis para a maioria das aplicações mesmo que não seja sempre os indicados para todas as necessidades de uma aplicação.

Eles são ideais para aplicações que requerem complexas consultas e operações

transacionais. A estrutura relacional facilita ainda a criação de relacionamentos entre diferentes conjuntos de dados. Uma aplicação onde estes bancos são utilizados com frequência são os sistemas bancários. É essencial rastrear e gerenciar as interações entre contas de clientes, transações de débito e crédito, e relatórios financeiros, fazendo a conexão correta entre por meio de chaves primárias e indexamento de colunas.

O grande problema enfrentado pelos bancos de dados relacionais se encontram quando existe a necessidade de escalonamento do sistema, ou ainda quando há a necessidade de pesquisa e recuperação de grandes conjuntos de dados (Big Data). Esses problemas surgem a necessidade de outros modelos como os NoSQL.

2.6.2 Bancos de Séries Temporais

Bancos de dados de séries temporais (TSDB - *Time Series Database*) são criados para armazenar e analisar dados coletados ao longo do tempo. Os dados que ocorrem em determinados períodos são conhecidos como séries temporais e são comuns em diversas aplicações, como monitoramento de sistemas e aplicações, assim como na criação de sistemas como o proposto neste trabalho, os projetos de IoT (Internet das Coisas) (FRANKLIN; PATTERSON; FOX, 2017).

Uma das principais características é a indexação baseada em tempo. Esse tipo de banco de dados é particularmente interessante em situações onde é possível encontrar padrões de variação nos dados ao longo do tempo. Um exemplo é observar dados da coleta das chuvas para entender os períodos de maior ocorrência e identificar possíveis anomalias. Os bancos relacionais são mais versáteis e utilizados em praticamente todas as aplicações, seja de forma primária ou secundária. No entanto, os TSDBs oferecem vantagens significativas em sistemas onde o tempo é um fator crucial para os dados armazenados (INFLUXDATA, 2024).

Alguns exemplos desse modelo de bancos de dados incluem o InfluxDB, TimescaleDB, e OpenTSDB. Na seção abaixo será destacado o banco Timescale, o utilizado para construção do trabalho.

2.6.2.1 TimescaleDB

O banco de dados de séries temporais *TimescaleDB* é um exemplo de banco que utiliza a abordagem de arquitetura híbrida, sendo desenvolvido como uma extensão do *PostgreSQL* para combinar a robustez e as funcionalidades avançadas do *PostgreSQL*, mas com otimizações para seu uso proposto. Seu projeto visa buscar

e conseguir eficiência em inserir, consultar e analisar grandes volumes de dados que variam com o tempo, sendo uma escolha comum entre seus concorrentes.

O *TimescaleDB* utiliza a arquitetura de particionamento chamada *hypertables*, que particiona automaticamente os dados temporais em tabelas menores baseadas em intervalos de tempo. Isso melhora a desempenho de inserção e consulta sem sacrificar a simplicidade do modelo relacional.

Como é esperado de um banco de dados relacional, o fato do *TimescaleDB* estender o *PostgreSQL*, faz com que ele também suporte a linguagem *SQL*, facilitando a integração com ferramentas e aplicações existentes que já façam uso de bancos relacionais.

A escalabilidade desse sistema oferece características que destacam sua escalabilidade, de forma que permita sua expansão conforme o crescimento no volume de dados. É otimizado para operações de escrita e leitura de dados temporais, garantindo baixa latência, sendo o tempo em que uma ação é iniciada e concluída num banco de dados, e alta *throughput*, que significa que o sistema suporta um grande volume de dados em curtos períodos. Combinadas essas características fazem um excelente *TSDB*.

Alguns exemplos de funções específicas para séries temporais utilizados pelo *Timescale* incluem:

- Agregações contínuas, função que permite com que não seja necessário atualizar, por exemplo, dados que medem a média de um gráfico em determinado espaço de tempo, uma vez que o sistema estará constantemente realizando esses fluxos a fim de aumentar a eficiência e reduzir esforço por parte do sistema.

  Para mais informações sobre agregações contínuas no TimescaleDB, consulte o seguinte link:

  Continuous Aggregates - TimescaleDB Documentation.

- Compressão de dados, essa característica permite, segundo o site do desenvolvedor, reduzir o espaço ocupado em até 90% do espaço total utilizado normalmente por um banco relacional.

  Para obter mais informações sobre compressão no TimescaleDB, consulte o seguinte link:

  About Compression - TimescaleDB Documentation.

- *Gap-filling* permite ao banco acrescentar registrados com dados para garantir a integridade e continuidade do sistema, melhorando o desempenho e evitando possíveis erros.

  Para mais informações sobre *gap-filling* e interpolação com o TimescaleDB, consulte o seguinte link:

  Gapfilling and Interpolation - TimescaleDB Documentation.

Um dos casos de uso ideal é sua utilização para armazenamento e analise dados coletados de sensores IoT, como medições de temperatura e umidade, permitindo a detecção de padrões e anomalias (TIMESCALE, 2024). Este será o modelo de banco de dados utilizado neste trabalho.

2.7 Rust

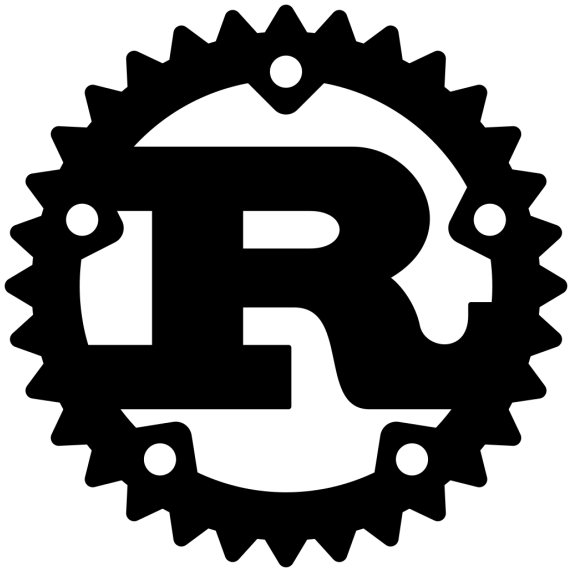

Figura 5 – Logo fornecido pela Rust Foundation. Fonte: (RUST FOUNDATION, 2022)

A linguagem de programação Rust, identificada por vezes por meio da logo da Figura 5, foi idealizada pela Mozilla Research ao decorrer dos anos 2000 e foi concebida como uma resposta aos desafios enfrentados pelos desenvolvedores de *software* em relação à segurança, concorrência e desempenho.

Buscando surgir como uma linguagem que pudesse fazer oposição e ser uma solução alternativa às linguagens derivadas do C e C#, ela se propõe em ser uma linguagem de alto desempenho adequada a desenvolvimento de sistemas de baixo

nível, ou seja, tendo sua compilação mais perto do que a máquina normalmente interpreta, construindo principalmente uma linguagem segura.

Desde sua criação, a linguagem ganhou popularidade devido às suas características únicas. Algumas de suas principais ideias podem ser observadas a seguir:

- Sistema de tipagem: esta linguagem utiliza um sistema de tipagem estático fazendo com que durante o desenvolvimento das aplicações utilizando essa aplicação requerem mais atenção, porém produzem garantias na utilização de memória, assim como elimina erros comum, como casos de uso inesperados (RUST DOCUMENTATION, 2022).
- Sistema de gerenciamento de memória: utiliza conceitos de propriedade (*Ownership*) e empréstimos (*Borrowing*), que permite a alocação de recursos de forma eficiente evitando a necessidade de uso de um coletor de lixo.

Essas características garantem que o Rust tenha ênfase em concorrência segura e paralelismo, tornando-a ideal para aplicativos que exigem eficiência em sistemas multi-threaded. Além disso, existem outros recursos que devem ser citados, uma vez que fazem com que o Rust garanta aos desenvolvedores a possibilidade de escrever códigos seguros, concisos e legíveis, como o *pattern matching*, *traits* e *enums*(RUST FOUNDATION, 2022).

2.8 Docker

O Docker é uma ferramenta de desenvolvimento de *software* que permite aos usuários a criação de ambiantes isolados do sistema operacional, cada ambiente desses é conhecido como um contêiner. Dessa forma, o Docker permite com que seja possível simular e executar aplicações que sejam totalmente independentes do sistema, garantindo segurança, desempenho e portabilidade as aplicações construídas por meio dele.

Os principais conceitos podem ser verificados no site da própria desenvolvedora (DOCKER, 2024). Os mais importantes são:

- **Contêineres**: São ambientes isolados criados a partir das imagens executadas pelo Docker.

- **Imagens**: São modelos configurados e criados por meio de um arquivo Dockerfile.
- **Dockerfile**: Arquivo de instrução para a criação de uma imagem.
- **Orquestradores**: Ferramentas como *Docker Swarm* e Kubernetes normalmente utilizam para orquestrar e gerir a utilização de vários contêineres simultâneos. Em ambiente local a ferramente de orquestração utilizada é o *Docker-Compose*

O Docker se tornou uma ferramenta indispensável ao garantir praticidade, velocidade e segurança durante o desenvolvimento de aplicações. No projeto a ferramenta Docker-Compose será utilizada por diversas vezes a fim de permitir que as aplicações funcionem em mais de um sistema operacional com a mesma eficiência.

2.9 Desenvolvimento WEB

2.9.1 Hasura

A plataforma Hasura é um projeto que simplifica a criação de *BackEnds* para aplicações e serviços. O mesmo oferece uma API GraphQL que permite a conexão diretamente aos mais diversos bancos de dados, como PostgreSQL, MySQL e SQL Server. Com funcionalidades interessantes como o controle de acesso por meio de uma senha pré-definida, permite praticidade e segurança para o desenvolvimento de uma aplicação sólida.

De forma resumida, o Hasura é uma ferramenta poderosa que facilita e democratiza o uso da linguagem GraphQL para desenvolvedores construírem *BackEnds* eficientes, seguros e principalmente práticos.

2.9.1.1 GraphQL

O GraphQL é uma linguagem de consulta e manipulação de dados desenvolvida pelo Facebook (Atualmente Meta). Foi projetada visando simplificar a comunicação entre servidores e dispositivos, além de surgir como uma alternativa à tradicional arquitetura REST. Ao contrário do proposto pela arquitetura mais comum (REST), o GraphQL permite que um cliente escolha especificamente os dados a serem devolvidos pelo servidor, ao contrário do que normalmente se espera em uma API REST, que retorna um conjunto já definido pela aplicação.

2.9.1.1.1 Query e Subscriptions

Durante a criação e formulação do projeto, foram utilizadas duas formas distintas de fazer pesquisas na base de dados, as duas foram importantes durante a criação do projeto, como será possível visualizar na próxima seção.

Em GraphQL, uma *query* é como um pedido personalizado de dados que o cliente faz ao servidor. Em vez de receber mais ou menos informações do que necessita, o cliente especifica exatamente quais campos e tipos de dados deseja. Por exemplo, pode solicitar apenas o nome e o email de um usuário, tornando a recuperação dos dados mais precisa e eficiente. Um exemplo é o verificado na Lista 2.1. A consulta em GraphQL é utilizada para obter dados dos sensores e suas datas.

```
query {
  data_sensors {
    sensor_data
    date
  }
}
```

Lista 2.1 – Consulta em GraphQL

Por outro lado, uma *subscription* em GraphQL permite que o cliente receba atualizações em tempo real, mas tem muitas características semelhantes às *queries*. Como exemplo visto na Lista 2.2.

```
subscription {
  last_3_days(
    order_by: {
      one_hour_interval: asc
    }
  ) {
    one_hour_interval
    max_temp
  }
}
```

Lista 2.2 – Subscription em GraphQL para obter dados dos últimos 3 dias ordenados por intervalo de uma hora

As *mutations* são as formas de inserir dados em um banco de dados por meio do GraphQL, como é possível observar na Lista 2.3.

```
mutation {
  insert_data_sensors (
  objects: [{
    sensor_data:
               {temperature: 12}
    sensor_id: 1
  }]
  )
  {
      returning {
               sensor_data
        date
      }
    }
}
```

Lista 2.3 – Mutation em GraphQL

2.9.2 React

React é um *framework*, conjunto de bibliotecas, escrito em JavaScript e criado pelo Facebook (META) em 2011, surge por meio da ânsia de melhorar a eficiência e a experiência de desenvolvimento de interfaces de usuário para suas plataformas. Este *framework* trouxe uma abordagem diferenciada onde a construção das interfaces são criadas a partir de blocos reutilizáveis, que foram denominados componentes.

Uma característica importante do React é sua capacidade de renderização tanto no lado do cliente (*client-side rendering*) quanto no lado do servidor (*server-side rendering*), o que o torna uma escolha versátil para diferentes tipos de aplicações webs. O React também introduziu o conceito de virtual DOM (*Document Object Model*), monitorando o DOM real e substituindo apenas as interfaces alteradas, garantindo eficiência e velocidade a paginas dinâmicas.

A sua qualidade e capacidade de criação de sistemas completos e responsivos justifica sua utilização em diversas das maiores empresas, como Netflix, sua própria criadora o Facebook, Twitter (Atual X) e Airbnb. Garantindo a ferramenta um vasto acervo de informações e conteúdos o que melhoram o desenvolvimento e facilitam o aprendizado.

2.9.3 CORS

A presença do CORS (Cross-Origin Resource Sharing) dentro do ambiente do Hasura GraphQL pode ser uma medida interessante por várias razões. O CORS é um mecanismo de segurança fundamental na web, principalmente para poder proteger os usuários contra-ataques de solicitações entre origens, garantindo que recursos da aplicação só possam ser solicitados a partir de domínios permitidos.

No contexto do Hasura GraphQL, que muitas vezes é usado para criar APIs poderosas e flexíveis para aplicativos da web e móveis, a presença do CORS é crucial por entre outras razões, a ajuda a proteger a API contra solicitações maliciosas de origens não confiáveis, reduzindo assim o risco de ataques como os de CSRF (Cross-Site Request Forgery).

2.9.4 SSH Tunneling

O *SSH Tunneling* é uma técnica que facilita a exposição de servidores locais. Ao contrário de outras soluções que requerem configurações complexas de *firewall* ou redirecionamento de porta, o *SSH Tunneling* simplifica o processo. Ele permite que os usuários iniciem túneis *SSH* rapidamente usando apenas o comando *SSH* em seus terminais. Isso é possível devido à sua arquitetura baseada em *SSH (Secure Shell)*, um protocolo seguro que garante a criptografia das informações transmitidas, eliminando a necessidade de preocupações com segurança.

Além da sua simplicidade, o *SSH Tunneling* oferece confiabilidade e estabilidade. Executado em uma infraestrutura robusta e escalável, os túneis *SSH* tendem a ser estáveis e rápidos. Isso torna esta ferramenta uma ótima opção para desenvolvedores que precisam, temporariamente, disponibilizar seus servidores locais à internet para fins de desenvolvimento, teste ou demonstração, sem problemas com configurações de rede.

3 Desenvolvimento

3.1 Metodologia

Para a realização do projeto deve-se reconhecer os desafios e necessidades a serem cumpridas, para que os resultados obtidos sejam compatíveis com as expectativas iniciais. Primeiramente é interessante avaliar a bibliografia já existente a fim de não repetir erros e cair em teorias já frustradas, entendendo as limitações do trabalho e delimitando a quantidade de esforço que deve ser aplicada em cada momento da produção.

Destaca-se a necessidade de também delimitar e pensar como os CLPs enviam suas informações, suas interfaces de comunicação e quais são suas limitações dentro do escopo do projeto. Tais aspectos são importantes ao se realizar a conexão via Modbus com o microcontrolador que recebe tais informações. Com isso, a análise de como os dados são enviados e tratados em cada um dos momentos do projeto mostra-se indispensável a fim de criar um trabalho coeso.

Dessa forma, outra característica que é trabalhada durante o projeto é a abstração desde a ponta inicial do projeto, o Controlador Lógico Programável. O foco do projeto é no fluxo do envio dos dados via Modbus para frente, garantindo interoperabilidade do sistema com qualquer outro dispositivo que possa se conectar e comunicar por meio do protocolo Modbus.

Com estes passos realizados, um primeiro teste entendendo a comunicação e funcionamento dos equipamentos e *softwares*, explorando propostas de implementação quando se trata de IoT, com *softwares* consagrados dentro do mercado, como os citados na introdução como Microsoft Azure e AWS, ou ainda um sistema de BackEnd autoral. Entende-se ainda o microcontrolador ESP32 como um intermediário, e assim sendo, deve ser projetado buscando obter um sistema robusto, mas, ao mesmo tempo, rápido e com baixo custo de implementação.

3.2 Diagrama do Projeto

O Diagrama 6 pretende delimitar e guiar o projeto dentro de suas ambições, facilitando o entendimento a terceiros. Nele é possível verificar e entender como deveria funcionar a interação entre cada uma das partes que compõem o projeto final.

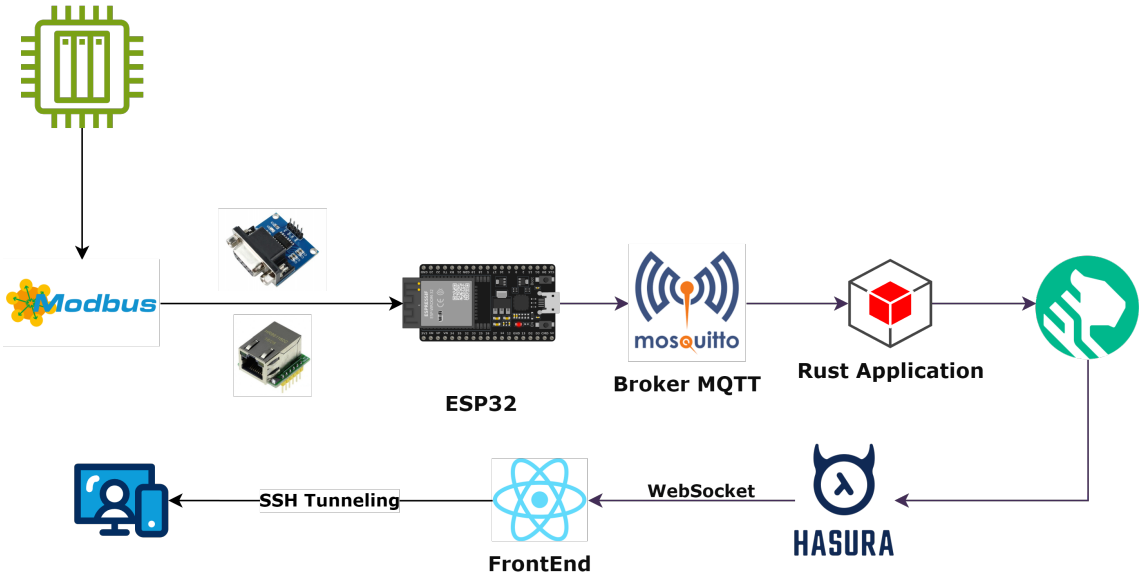

Figura 6 – Diagrama Geral do Projeto criado pelo autor para organização do trabalho

No diagrama, apresentado na Figura 6, é possível visualizar de forma geral como o sistema será montado para destacar os principais pontos e processos

1. A representação do CLP indica no diagrama que ele fornecerá as informações para todo o andamento do projeto. Na seção de resultados, será utilizada uma simples aplicação em Python que busca a temperatura da CPU do computador e a envia diretamente ao registrador previamente configurado. A utilização do código se dá principalmente por limitações em relação à disponibilidade local ao dispositivo físico. Além disso, como a justificativa principal do projeto é criar um sistema que seja agnóstico ao CLP, é interessante verificar que qualquer dispositivo que suporte o protocolo Modbus poderá ser implementado e substituído no lugar da aplicação Python.

2. Comunicação Modbus TCP: O protocolo Modbus é utilizado para realizar a comunicação entre os dispositivos de aquisição de dados e o ESP32. Durante o projeto, o protocolo Modbus TCP foi escolhido devido à facilidade de comunicação entre os dispositivos, não dependendo de um meio físico específico,

já que pode operar na mesma rede. É importante destacar que também seria possível realizar o projeto com o Modbus RTU, mas isso exigiria ajustes tanto nos dispositivos de aquisição quanto no microcontrolador. No diagrama houve a representação tanto de um adaptador RS232 para conexão via Modbus Serial junto ao ESP, quanto um adaptador Ethernet para a comunicação via Modbus TCP/IP.

3. ESP32: O microcontrolador ESP32 desempenha um papel crucial no projeto, sendo responsável por processar os dados recebidos pelo Modbus e publicar mensagens no tópico do broker MQTT Mosquitto previamente configurado. Sua escolha se deve principalmente ao baixo custo e às características de conectividade que facilitam o desenvolvimento e garantem resultados robustos com um investimento acessível.

4. Mosquitto MQTT: O broker Mosquitto é uma implementação de código aberto de um servidor MQTT. Ele desempenha o papel de intermediário na troca de mensagens entre os dispositivos que publicam e os dispositivos que estão inscritos no mesmo tópico. Sua utilização se justifica, entre outras características, devido à segurança de manter os dados numa aplicação local, ao desempenho que é maior, uma vez que não se faz necessário percorrer distâncias por meio da internet, e também devido à personalização que só a configuração local garante.

5. Aplicação em Rust: A aplicação feita em Rust realiza o processo de monitorar o tópico utilizado para envio das informações pelo ESP32, tratar os dados, identificar possíveis dados não padronizados, e inserir os dados no banco de dados com eficiência. A principal justificativa para a sua utilização se dá pela sua segurança de memória, garantindo menos erros e consequentemente mais segurança ao processo.

6. Banco de Dados de Séries Temporais (TSDB) - TimescaleDB: O TimescaleDB é responsável por armazenar as informações coletadas inicialmente e disponibilizá-las por meio de tabelas e *views*, como será demonstrado posteriormente. Derivado do PostgreSQL, o TimescaleDB facilita o desenvolvimento ao utilizar a mesma linguagem PSQL, apesar de ser um banco de dados especializado em séries temporais. A escolha por esse modelo é motivada principalmente pela capacidade de indexação baseada em data, um aspecto crucial para análise de dados. Isso não apenas facilita o desenvolvimento, mas também melhora significativamente a eficiência dos fluxos de trabalho.

7. Aplicação com Backend em Hasura/GraphQL: A aplicação com backend em Hasura oferece uma interface GraphQL para integração por meio das *views* construídas no banco de dados. Isso permite visualizar e atualizar os dados em tempo real, utilizando a combinação de GraphQL com o protocolo WebSocket para comunicação.

8. Aplicação em React: A aplicação de *FrontEnd* escrita em React finaliza o ciclo de desenvolvimento de *software*, proporcionando uma interface gráfica que facilita a visualização dos gráficos e outras informações diretamente por meio do navegador. O uso de React é favorecido pela vasta disponibilidade de conteúdos e pela agilidade no desenvolvimento.

9. SSH Tunneling - Serveo: Essa aplicação permite com que as duas aplicações que viabilizam a visualização sejam disponibilizadas em um domínio publico. Garantindo a acessibilidade aos dados.

3.3 Modbus

Durante os testes realizados em ambiente local, foi possível verificar a funcionalidade do protocolo Modbus por meio de um script desenvolvido em Python, utilizado para simular o envio de dados por um CLP. Devido a limitações técnicas, os dados foram obtidos diretamente da máquina local, conforme detalhado na seção de resultados, onde foram utilizadas informações da temperatura da CPU.

A primeira abordagem partiu por meio do teste do protocolo Modbus TCP, porém existe a possibilidade da utilização do protocolo Modbus Serial. A diferença passa por meio do meio físico utilizado para a transferência de dados. A utilização inicial por meio do protocolo TCP pode ser justificado, entre outros fatores, pela facilidade, uma vez que é possível utilizar a conectividade Wi-Fi presente no ESP para realizar os testes.

A utilização do código visa simular de forma precisa o efeito de um CLP dentro do sistema criado, o papel de enviar os dados coletados de algum ambiente e enviar por meio do protocolo em determinados espaços de tempo. Para acessar o código desenvolvido para este propósito, bem como outros códigos relacionados ao projeto, eles estão disponíveis no repositório Pavao (2024).

3.3.1 Comparação com CLP

Para exemplificar o funcionamento em comparação com um CLP, pode-se pensar em um sistema que utiliza um sensor de temperatura, como um termopar ou um RTD (Resistance Temperature Detector), para medir a temperatura de algum meio e converter em uma saída elétrica. No caso mais comum, pode-se ter uma saída de 4-20 mA. Quando a saída for 4mA, tem-se a menor temperatura possível registrada pelo sensor, e quando for 20mA, a maior.

Esse sinal gerado é conduzido ao Controlador Lógico Programável da planta. Em ambientes industriais, o sinal de 4-20 mA se destaca pela capacidade de isolar interferências elétricas e condução a longas distâncias, aumentando a confiabilidade do sistema. Dentro do CLP, o sinal é convertido por um conversor Analógico/Digital, que converte o sinal de 4-20 mA para um valor digital proporcional ao registrado pelo sensor. A configuração do conversor A/D influencia diretamente na precisão da conversão do sinal, porém nos casos dos controladores já veem configurados de fábrica e normalmente não permitem alterações nesta configuração.

3.4 ESP32

Durante o processo de criar um código que fizesse o interfaceamento entre um CLP e a internet foi encontrada um problema ao adentrar as minúcias da biblioteca Modbus e MQTT, uma sendo necessária para fazer a comunicação do Controlador Lógico Programável com o Microcontrolador e o segundo protocolo responsável por fazer a publicação no Broker Mosquitto no tópico indicado.

O problema começa quando se entende a rotina de verificação do loop dentro do Microcontrolador e consequentemente a frequência com que os dados serão atualizados para a interface no computador.

Dessa forma é possível encontrar duas opções:

- Utilizar uma função de delay para que o sistema para e só envie mensagens ao broker em um determinado espaço de tempo como na Lista 3.1.

```
delay(500);
```

Lista 3.1 – Demonstração da função de atraso de 500 milissegundos

- Utilizar de uma lógica de verificação e apenas enviar o valor armazenado no registrador do Modbus, uma vez que esse for modificado, enviando menos

mensagens, porém gerando alguma deficiência na hora de avaliar os dados caso eles se repitam. Dessa forma, para conseguir agregar de ambas as formas foi criada uma condição que permite que mesmo caso o valor obtido no registrador do Modbus não mude, ele será enviado ao MQTT em um intervalo pré-determinado. A lógica pode ser observada na Lista 3.2.

```
if (temp != previousValue || (currentMillis - previousMillis >=
    interval)) {
    previousValue = temp;
    previousMillis = currentMillis;
}
```

Lista 3.2 – Condição para envio de mensagens para o tópico MQTT

O código completo pode ser encontrado no link do github do projeto.

3.4.1 Código de recebimento Modbus

Assim como a configuração do MQTT, para a utilização do protocolo Modbus é necessário algumas configurações, sendo as principais a configuração referente aos registros utilizados para obtenção dos dados. Na Lista 3.3 é possível verificar a inclusão do registro com o valor 0 e o posterior monitoramento do registro 101 por meio da função *task*.

```
mb.server();
mb.addHreg(101, 0);
mb.task();
int temp = mb.Hreg(101);
```

Lista 3.3 – Exemplo de uso de Modbus no ESP32

3.4.2 Código de envio MQTT

Para o envio para o tópico MQTT, parte-se do pressuposto que o dispositivo deve estar conectado a uma rede com acesso à internet, ou ainda, na mesma rede do broker. Depois é necessário a configuração das credenciais de acesso ao broker MQTT, que geralmente incluem o endereço do broker, a porta, o nome de usuário e a senha, caso o broker permita é possível acessar os tópicos dos mesmo sem usuário e senha.

Outras configurações envolvem o nome do tópico dentro da função assim como a indicação da variável QoS (Qualidade de Serviço) que como indicado no capítulo de referencial teórico garantem ao envio ao broker, e consequentemente o envio ao dispositivo inscrito no mesmo tópico. Conforme é possível verificar, na Lista 3.4 a função *mqttClient.print(message)* realiza o envio da mensagem.

```
mqttClient.beginMessage(topic);
sprintf(message, "{ \"sensor_id\": 1, \"sensor_data\": {\"temperature
   \": %.2f} }", temp/100.0);
mqttClient.print(message);
mqttClient.endMessage();
```

Lista 3.4 – Envio de mensagem MQTT com dados de sensor

Um ponto que foi observado durantes os testes foi a percepção de que eventualmente o dispositivo perdia a conexão com o broker MQTT sem razão aparente, para tal a seguinte validação foi incluída no código, permitindo mais confiabilidade ao projeto. Onde a função reconnect, citada em 3.5 garante que ao programa tentativas até que ele consiga se conectar com sucesso.

```
if (!mqttClient.connected()) {
    reconnect();
}
```

Lista 3.5 – Reconexão ao Broker MQTT se não conectado

3.5 Mosquitto MQTT

As conclusões que levaram à utilização do protocolo MQTT foram baseadas em várias justificativas. Entre elas, destaca-se a alta eficiência do MQTT em cenários que exigem comunicação rápida e confiável entre dispositivos. Sua leveza e baixo consumo de largura de banda o tornam ideal para aplicações em IoT (Internet das Coisas), onde a transmissão de dados precisa ser constante e eficaz. Além disso, o MQTT é amplamente adotado devido à sua robustez em ambientes com conectividade instável, garantindo uma comunicação contínua e eficiente.

Com base nessa compreensão, as informações obtidas são enviadas do ESP32 para o Broker MQTT conhecido como Mosquitto MQTT, que está hospedado em um

contêiner Docker na máquina local. As configurações e a imagem utilizada podem ser verificadas no trecho de código da Lista em 3.6.

```
mqtt-broker:
  image: eclipse-mosquitto
  container_name: mqtt5
  ports:
    - "1883:1883"
    - "9001:9001"
  volumes:
    - ./mosquitto/config:/mosquitto/config:rw
    - ./mosquitto/data:/mosquitto/data:rw
    - ./mosquitto/log:/mosquitto/log:rw
  restart: unless-stopped
```

Lista 3.6 – Configuração do container Docker para MQTT Broker usando Eclipse Mosquitto

No trecho de código do Docker-Compose, que se refere especificamente da configuração do Mosquitto, e que também permite agilidade na hora de configuração do projeto, tem-se como destaque o uso da imagem fornecida pela própria Eclipse (Desenvolvedora do projeto). Assim como, as configurações padrão a um *broker*, tais quais, as portas padrão e a localização dos volumes onde ficaram presentes demais configurações internas do broker.

Por meio do arquivo de configuração do Mosquitto é possível configurar características importantes como o controle de permissões e os limites de acesso, sendo possível configurar quem pode acessar e tanto se subscrever em algum tópico quanto também publicar e criar novos. Isso permite uma personalização adequada a cada aplicação

Na configuração da Lista 3.7, a fim de agilizar testes, a configuração *allow_anonymous* permite que não seja necessário cadastrar e configurar as credenciais de um usuário.

```
allow_anonymous true
listener 1883
listener 9001
protocol websockets
persistence true
```

```
persistence_file mosquitto.db
persistence_location /mosquitto/data/
```

Lista 3.7 – Configuração do arquivo de configuração do Mosquitto

Para testes e simulações do fluxo, foi utilizada a ferramenta MQTT Explorer, que permite visualizar as informações transmitidas pelo broker por meio de uma interface simples e clara.

As configurações do projeto MQTT, assim como o Docker-Compose completo, podem ser encontrados no repositório Pavao (2024).

3.6 Aplicação Principal

A aplicação principal do sistema consiste em um programa que roda por meio da linguagem Rust. Em resumo, Rust é uma escolha robusta para o desenvolvimento de uma aplicação que conecta a um *broker* MQTT e armazena dados em um banco relacional, graças à sua atuação, segurança e suporte para concorrência.

Sendo que a principal justificativa é a sua segurança de memória, o Rust oferece garantias de segurança de memória sem a necessidade de um coletor de lixo (*garbage collector*), prevenindo estouros de *buffer*. Isso é importante em aplicações de IoT, como a proposta neste trabalho, e comunicação em tempo real, onde falhas de memória podem causar interrupções críticas.

A aplicação utiliza então de um ORM (*Object-Relational Mapping*), conhecido como diesel (DIESEL. . . , 2024), que faz a integração e cria modelo para ser impossível o armazenamento de informações que fujam os padrões estabelecidos pelo projeto/aplicação. Durante a utilização da biblioteca diesel foi necessário fornecer a ela qual tipo de bancos de dados ela trabalharia. Além da inscrição ao tópico principal do projeto, foi criado ainda outro tópico de monitoramento que está ligado a criação de um novo sensor para monitoramento, ele funciona como suporte ao tópico principal, compartilhando a informação do ID do sensor.

No final a combinação de Rust, MQTT e TimescaleDB oferece uma solução poderosa para o desenvolvimento de aplicações de monitoramento em tempo real, proporcionando segurança, desempenho e eficiência no armazenamento e processamento de dados.

3.6.1 Desenvolvimento do Código

O código da aplicação pode ser encontrado no Github criado para o desenvolvimento desta monografia (PAVAO, 2024). Durante a construção da aplicação é possível identificar dois pilares de extrema importância para o projeto, a recuperação dos dados publicados pelo microcontrolador em um tópico específico e posteriormente a inserção desses dados no banco de dados de séries temporais Timescale.

Duas bibliotecas foram utilizadas neste contexto. Para a utilização dos fluxos MQTT, a biblioteca paho-mqtt se destaca em diversas linguagens e *frameworks* como uma das mais interessantes e versáteis. Ela oferece uma implementação robusta do protocolo MQTT, facilitando a comunicação da aplicação com o broker. A paho-mqtt suporta a utilização de QoS e reconexões automáticas, abordagens necessárias no contexto do projeto.

Para a manipulação do nosso banco de dados, a biblioteca utilizada é o famoso ORM para Rust conhecido como Diesel. Como um bom ORM (Object-Relational Mapping), o diesel facilita a interação e comunicação entre aplicação e banco, se destacando pela compatibilidade para o banco de séries Timescale, uma vez que este é uma extensão do projeto Postgres.

A Lista 3.8 exemplifica um modelo, enquanto a Lista 3.9 demonstra a criação de um *scheema* criado por meio dessa biblioteca para o projeto:

```
#[derive(Queryable, Deserialize, Eq, PartialEq)]
#[diesel(table_name = crate::schema::data_sensors)]
#[diesel(check_for_backend(diesel::pg::Pg))]
pub struct DataSensor {
    pub sensor_id: i32,
    pub sensor_data: serde_json::Value,
}
```

Lista 3.8 – Definição da estrutura ‘DataSensor‘ em Rust

```
diesel::table! {
    data_sensors (id) {
        id -> Int4,
        sensor_id -> Int4,
        sensor_data -> Json,
        date -> Timestamptz,
    }
}
```

Lista 3.9 – Definição da tabela ‘data_sensors‘ em Rust com Diesel

Essas duas bibliotecas, paho-mqtt para os fluxos MQTT e Diesel para a manipulação do banco de dados, se complementam bem, permitindo o desenvolvimento de uma aplicação robusta, segura e eficiente para monitoramento e controle de variáveis em um ambiente industrial.

3.7 Banco de dados

Uma das decisões mais importantes do trabalho foi a escolha da fonte de dados. Durante o projeto, experimentaram-se alguns bancos de dados não relacionais, mas a escolha final foi um banco relacional consagrado pelo mercado, o Postgres SQL.

O Postgres utilizado não é o convencional. Para criar e utilizar ele por meio da API GraphQL, foi necessário instalar uma versão que implementa o Postgres e o transforma em um banco de séries temporais. A utilização de um banco de dados de séries temporais para armazenar informações de sensores é justificada pela sua capacidade de lidar eficientemente com grandes volumes de dados gerados em intervalos regulares. Bancos de séries temporais são otimizados para operações de escrita rápida e consultas de leitura que envolvem agregações e análises de dados ao longo do tempo. Isso os torna ideais para aplicações de IoT, onde a coleta e análise contínua de dados é essencial.

A Lista 3.10 mostra o arquivo *compose* utilizado para a criação da aplicação. Observa-se que as variáveis utilizam as nomenclaturas herdadas do Postgres, apesar de a imagem referenciar o banco TimescaleDB.

```
timescale:
  image: timescale/timescaledb:latest-pg16
  restart: always
  ports:
    - 5434:5432
  environment:
    POSTGRES_USER: USER
    POSTGRES_PASSWORD: USER1234
    POSTGRES_DB: USER_DB_TCC
  volumes:
    - db_data:/var/lib/postgresql/data
```

Lista 3.10 – Configuração do container Docker para TimescaleDB

A estrutura de Tabelas e Colunas pode ser observada na figura 7.

Figura 7 – Diagrama - Estrutura do Banco de Dados

A tabela principal é a *data_sensors*, está já sofreu alterações desde a concepção inicial do projeto, uma vez que era inicialmente indexada por um id único, porém passou posteriormente a ser indexada por meio da data de cada registro. Dessa forma, faz-se possível a configuração por meio do GraphQL, para a criação da *hypertable* que será utilizada para a inscrição para a criação de um dos gráficos do projeto.

A tabela secundária *sensors* permite a configuração de algumas características do sensor, como o nome, alguma descrição e ainda o local onde o sensor está instalado. No fluxo completo do projeto, a informação crucial a ser informada pelos microcontroladores é referente ao id do sensor. Como a aplicação principal trabalha de forma assíncrona é possível, em determinada configuração, haver um microcontrolador por sensor.

Foi necessário criar uma *view* que fosse adequada para a ideia proposta para o trabalho. A *view* utilizada foi inspirada, assim como, parte do projeto de visualização na ideia presente no seguinte link (WAWHAL, 2019). Para tanto, a *view* foi criada por meio do comando SQL, visualizado na Lista 3.11.

```
CREATE VIEW last_1_day_temp AS (
  SELECT time_bucket('60 seconds', date) AS sixty_sec,
```

```
    ROUND(avg(cast((sensor_data::json->>'temperature') as DECIMAL))
        ,2) AS max_temp
  FROM data_sensors
  WHERE date > NOW() - interval '1 day'
  GROUP BY sixty_sec
  HAVING ROUND(avg(cast((sensor_data::json->>'temperature') as
     DECIMAL)),2) <> 0
  ORDER BY sixty_sec ASC
);
```

Lista 3.11 – Criação de uma VIEW SQL para temperatura média do últimos dia

3.8 BackEnd

Para o BackEnd, optou-se por utilizar uma aplicação já pronta, visando o desempenho e aproveitar a compatibilidade já existente. A escolha foi o Hasura, que traz uma solução robusta, além de criar de forma automática uma API GraphQL que com o uso conjunto de WebSocket, permite a sistemas que façam uma comunicação em tempo corrente. Sendo de interesse para um sistema que visa o monitoramento quase que simultâneo.

Para implementação do projeto foi necessário subir um *container* Docker utilizando uma imagem especial para o funcionamento dentro do Docker para a arquitetura de processadores ARM. O arquivo Docker-Compose ficou configurado de acordo com 3.12.

```
graphql-engine:
  image: fedormelexin/graphql-engine-arm64
  ports:
    - "8080:8080"
  depends_on:
    - "timescale"
  restart: always
  environment:
    HASURA_GRAPHQL_DATABASE_URL: postgres://marcus:
        marcus1234@timescale:5432/TCC
    HASURA_GRAPHQL_ACCESS_KEY: mylongsecretkey
    # HASURA_GRAPHQL_ENABLE_CORS: "true"
    # HASURA_GRAPHQL_CORS_DOMAIN: "https://teste-marcus.serveo.net/"
```

```
command:
  - graphql-engine
  - serve
  - --enable-console
```

Lista 3.12 – Configuração do container Docker para Hasura GraphQL Engine

Assim, é possível a realização de alguns configurações básicas tais quais a porta onde a aplicação irá rodar, a configuração de acesso ao banco de dados e a configuração de senha de acesso, que será utilizada tanto para o acesso à plataforma quanto para que a aplicação de FrontEnd consiga se conectar com o Hasura, e, dessa forma obtenha os dados para que plote o gráfico. Na Figura 8 é possível verificar a visualização das tabelas pelo Backend (Hasura), enquanto na Figura 9 é possível verificar a exposição dos dados.

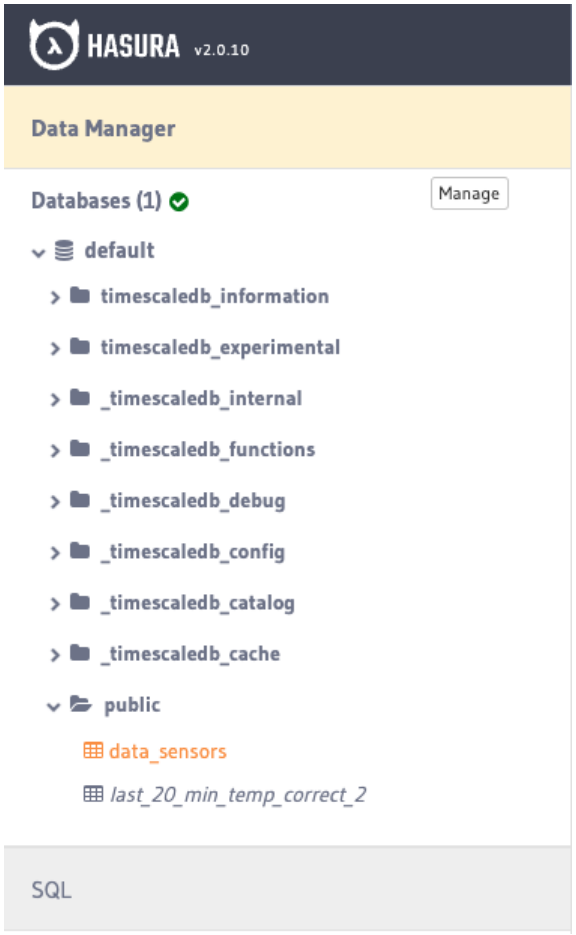

Figura 8 – Visualização das tabelas no Hasura

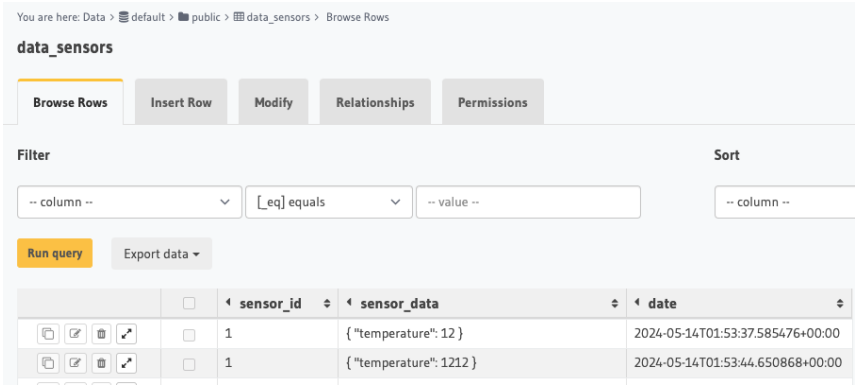

Figura 9 – Exposição dos dados no Hasura

3.8.1 GraphQL e WebSockets

Durante essa seção é possível observar um conceito interessante que entra em conflito com o que foi observado na Seção 2.7, aplicação RUST, onde foi abordado como ocorreria a inserção de dados dentro do banco de dados Postgres SQL. Inicialmente feito inserido com uma chave primária (id) tal abordagem foi substituída com a mudança no projeto para o uso do GraphQL

Para ser possível fornecer ao Front-End da aplicação os dados em tempo real de forma que aja uma atualização simultânea com o funcionamento do sistema viu se a necessidade da utilização de WebSockets (comunicação em tempo real) e consequentemente a utilização da abordagem de construção de API GraphQL (THE. . . , 2024). Dentro deste contexto surge o interesse em uma funcionalidade do GraphQL conhecida como *subscriptions*, onde o GQL se aproveita do uso de WS para serem solicitadas atualizações constantes ao sistema, assim como quais dados devem ser enviados.

Essa integração entre os dois serviços melhora a apresentação da aplicação, proporcionando dados atualizados continuamente sem a necessidade de solicitações repetidas ao servidor.

Por fim, a configuração mais importante feita dentro dessa aplicação foi o *tracking* das tabelas principal assim como da *view*, o que permite ao Hasura monitorar e fazer consequentemente a integração esperada com a aplicação que irá exibir os dados, como é possível observar no próximo capítulo, a recuperação dos dados dependerá de que as tabelas estejam sendo *trackeadas* pelo BackEnd. A Figura 10, mostra a interface utilizada onde a configuração é realizada.

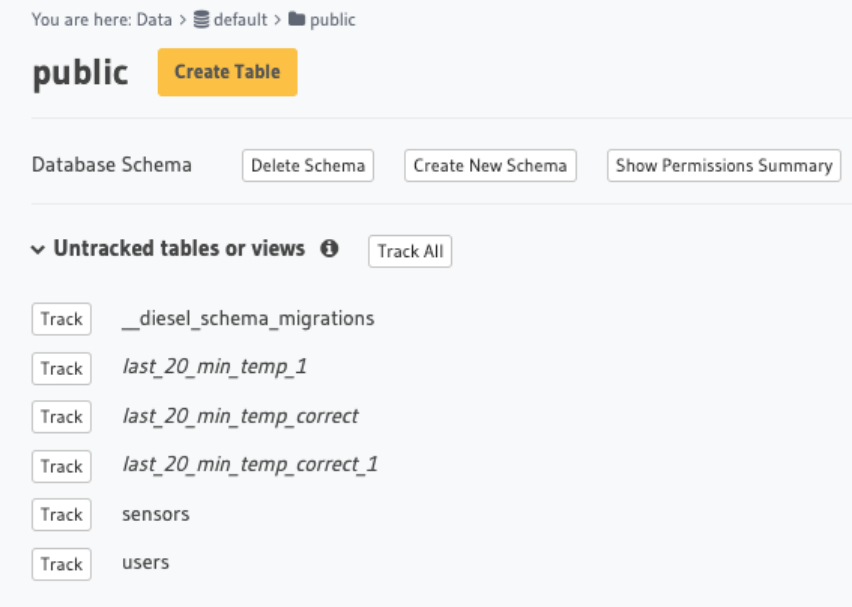

Figura 10 – Configuração para realizar o *Tracking* de *Views* - Hasura

3.9 FrontEnd

A abordagem utilizada nesta parte do trabalho consistiu na implementação de um projeto simples em React JS, focado inicialmente na visualização de dados de óbitos. O código do projeto, disponível no repositório Pavao (2024), é caracterizado por uma única página inicial que permite a visualização de três tipos de gráficos, conforme demonstrado nas capturas de tela anexas.

No primeiro gráfico, Figura 17, é possível selecionar a data e horário de início, assim como, a data de e horário final por meio de dois inputs. Isso permite visualizar os valores dentre um *range* de dados com uma boa precisão, oferecendo uma análise mais focada e relevante do período desejado. Melhorando também o desempenho do GraphQL, uma vez que eles só obtém os dados dentro do range selecionado.

Outras duas funcionalidades adicionais incluem um botão que possibilita a exportação dos dados visualizados no formato CSV, permitindo aos usuários salvar e utilizar os dados fora da aplicação. Além disso, foi adicionada uma legenda no gráfico, que indica as temperaturas média, máxima e mínima ao longo do período selecionado, proporcionando uma análise mais detalhada e compreensível das informações exibidas.

Figura 11 – Gráfico principal. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

O segundo e terceiro gráfico tem abordagens parecidas, uma vez que neles têm-se gráficos configurados em cima da função *subscription* do GraphQL, onde por meio de uma conexão *websocket* fornece atualizações em tempo real de valores nos últimos vinte minutos e um dia na variável observada.

O gráfico com as mudanças ocorridas durante o período do último dia, pode ser observado na Figura 12.

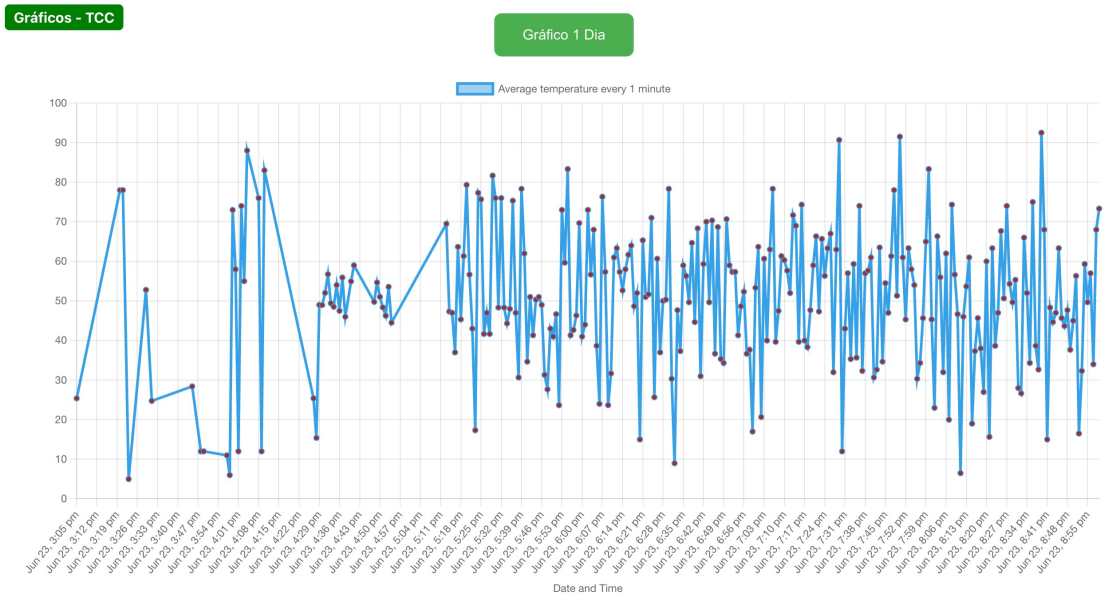

Figura 12 – Gráfico durante período de um dia. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

O gráfico com as mudanças ocorridas durante o período dos últimos vinte minutos, pode ser observado na Figura 13.

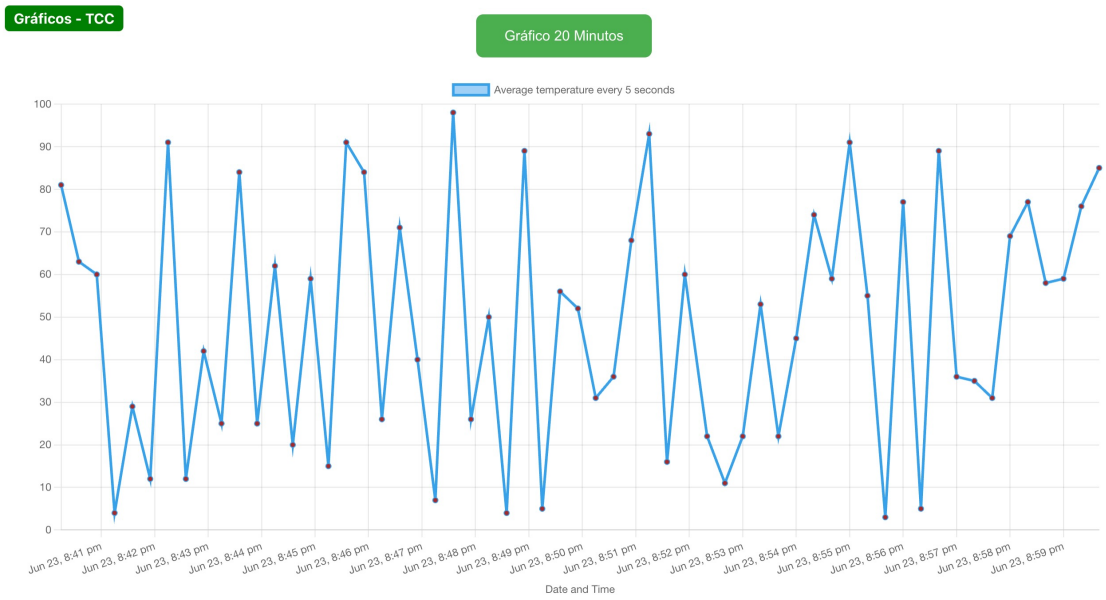

Figura 13 – Gráfico durante período de vinte minutos. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

O código utilizado pode ser encontrado dentro da pasta tcc-chart-live disponibilizado no link do GitHub: Pavao (2024)

3.10 Disponibilização

por meio utilização da técnica conhecida como SSH Tunneling é possível fazer com que a aplicação que funciona em ambiente local possa ser disponibilizada na web. A seguinte ferramente cria um túnel criptografado entre o servidor em ambiente local e um client por meio do protocolo SSH.

A forma de implementação dessa técnica pode variar tendo como variações o *Remote Port Forwarding*, *Dynamic Port Forwarding*, entre outras, e, por fim, o Local Port Forwarding que permite que a um cliente a comunicação direta com um servidor que está por trás de uma rede de segurança e está sendo explicado e utilizado ao longo desse projeto.

Existem diversas formas de utilizar essa técnica, existem aplicações que realizam esse trabalho, alguns exemplos são o LocalTunnel, Serveo e Ngrok. Durante esse projeto a aplicação Serveo foi a utilizada devido a praticidade e eficácia no que é proposto a fazer.

Para a configuração bastou a instalação via brew e posteriormente o seguinte comando que permite ainda a criação de uma URL personalizada após um simples cadastro. Após esses comandos foi necessário atualizar a comunicação para que a aplicação React consiga enxergar corretamente o BackEnd. Para realizar o redirecionamento, os comandos são os expostos na Lista 3.13.

```
ssh -R 80:localhost:3000 serveo.net
ssh -R 80:localhost:8080 serveo.net
```

Lista 3.13 – Redirecionamento de porta usando SSH para Serveo

4 Resultados

Para iniciar este capítulo, é interessante relembrar que o presente projeto não tem um escopo definido em relação à variável observada. Assim como mencionado na introdução, pretende-se absorver e fornecer algumas soluções viáveis para o monitoramento de controladores normalmente utilizados na indústria. Dessa forma, para efeitos de exemplificação, os dados foram obtidos por meio da temperatura da CPU do computador, que, no resultado simula a conexão Modbus de um CLP.

Para exposição dos resultados, algumas condições iniciais foram estabelecidas devido a algumas limitações. Dessa forma, têm-se os elementos e dispositivos citados no parágrafo anterior funcionando conforme o diagrama exposto na Figura 14. A principal limitação é a necessidade da utilização de uma aplicação em Python que simula um dispositivo Modbus, no caso um CLP, que envia dados diretamente ao ESP32 por meio de uma comunicação TCP/IP, validando a primeira parte do projeto.

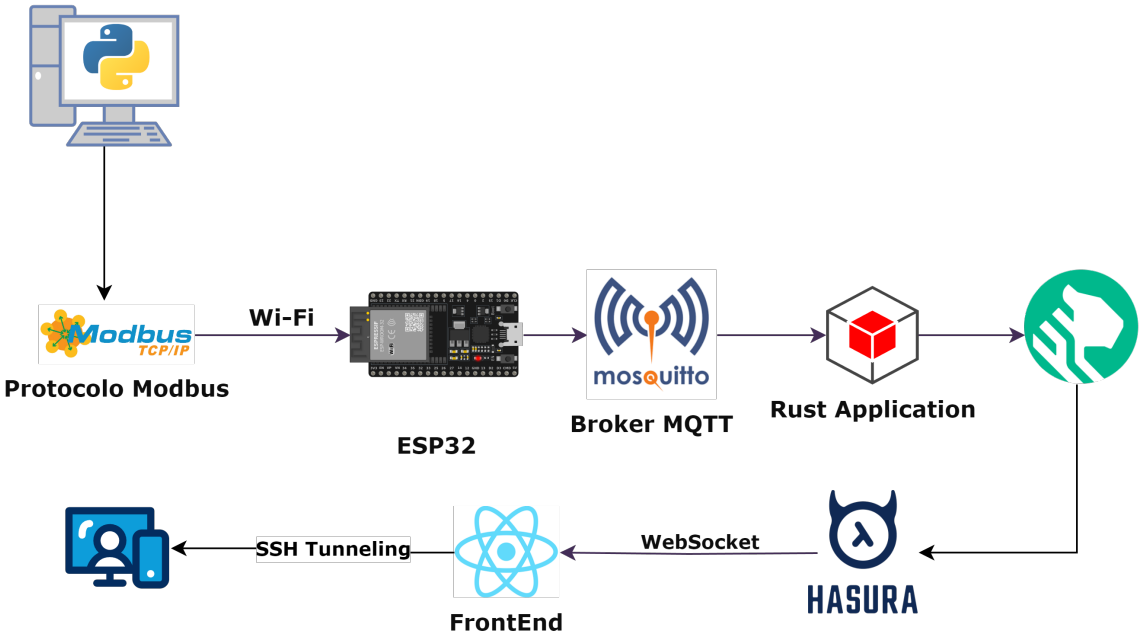

Figura 14 – Diagrama utilizado para obter os resultados

Ao longo de um dia, essa aplicação simulou o envio de dados diretamente para o ESP32, que na simulação tinha o IP 192.168.16.3. Os dados foram configurados tanto no código quanto no ESP32 para serem recebidos no registrador 101, utilizando a função 6 do protocolo Modbus TCP, onde o registrador a ser considerado pode ser

o 40101 algumas vezes.

O microcontrolador, compartilhando da mesma rede com o computador que roda o código em Python, recebe os dados, escreve uma mensagem configurada entre os dois sistemas e publica tal mensagem diretamente em um tópico no broker MQTT Mosquitto. A mensagem configurada no padrão JSON segue o seguinte formato configurado para comunicação entre os sistemas:

```
{
    "sensor_id": 1,
    "sensor_data": {"temperature": 25.4}
}
```

Como explicado na Seção 3.7, a primeira propriedade é o número identificador do sensor e a segunda as informações da variável observada. A segunda propriedade é também um objeto JSON, possibilitando a inserção de outras variáveis, sendo necessário apenas alterar o BackEnd, mantendo a mesma estrutura de código da aplicação principal no microcontrolador e na aplicação principal.

Uma vez que todos os dispositivos e aplicações utilizadas nesta demonstração de teste compartilham a mesma rede, o broker MQTT foi implementado localmente dentro da mesma infraestrutura. É interessante ressaltar que essa não é uma prerrogativa obrigatória e foi utilizada apenas para mostrar possibilidades de realizar o projeto de forma totalmente local e gratuita, explorando configurações que em outras alternativas não seriam possíveis. Algumas alternativas e ideias de brokers foram adicionadas no final do trabalho.

É interessante frisar que os sistemas do ESP32 e da aplicação principal funcionam de forma assíncrona utilizando o broker MQTT, o que permite a execução totalmente independente entre os sistemas. Dessa forma, porém, alguns problemas podem surgir, principalmente relacionados ao descompasso das aplicações que não é facilmente tratado. Como mostrado no capítulo de Desenvolvimento, existem tratamentos para evitar tais erros.

Passa-se a tratar da aplicação principal do projeto que faz a inscrição, entre outros tópicos, ao tópico *data_sensors*, responsável pelo envio das informações. Com a chegada das informações, as mensagens devem seguir um esquema, como mostrado durante a etapa de desenvolvimento. Caso passem por esse padrão, evitando a inserção de dados indesejados no banco de dados, elas são inseridas no banco de séries temporais TimescaleDB. Durante os resultados apresentados aqui, apenas essa

rota será utilizada. No entanto, existe ainda a inscrição que se destina a monitorar o tópico *sensors*, responsável pela configuração de um novo sensor.

Conforme a chegada dos dados na tabela, já é possível notar a primeira diferença para um banco relacional comum: a tabela principal onde as informações são inseridas não tem identificadores numéricos como chave primária. No caso dos bancos de séries temporais, temos a coluna de data indexada e por isso a configuração do banco fica da seguinte forma.

Para finalizar o processo e ter-se, finalmente, a exibição dos dados que passaram por todo esse processo, têm-se as aplicações de BackEnd e FrontEnd, ou seja, a aplicação que busca os dados no banco de dados e disponibiliza via uma "rota", e consequentemente a aplicação que consome essa rota e disponibiliza de forma visual ao usuário.

Durante esse período, foi possível observar o funcionamento dos três gráficos, ilustrados nas Figuras 15, 16, 18. O gráfico que remete aos dados dos últimos vinte minutos é interessante principalmente para avaliar bruscas mudanças na temperatura, além de permitir um acompanhamento em tempo real, como é possível visualizar na Figura 15.

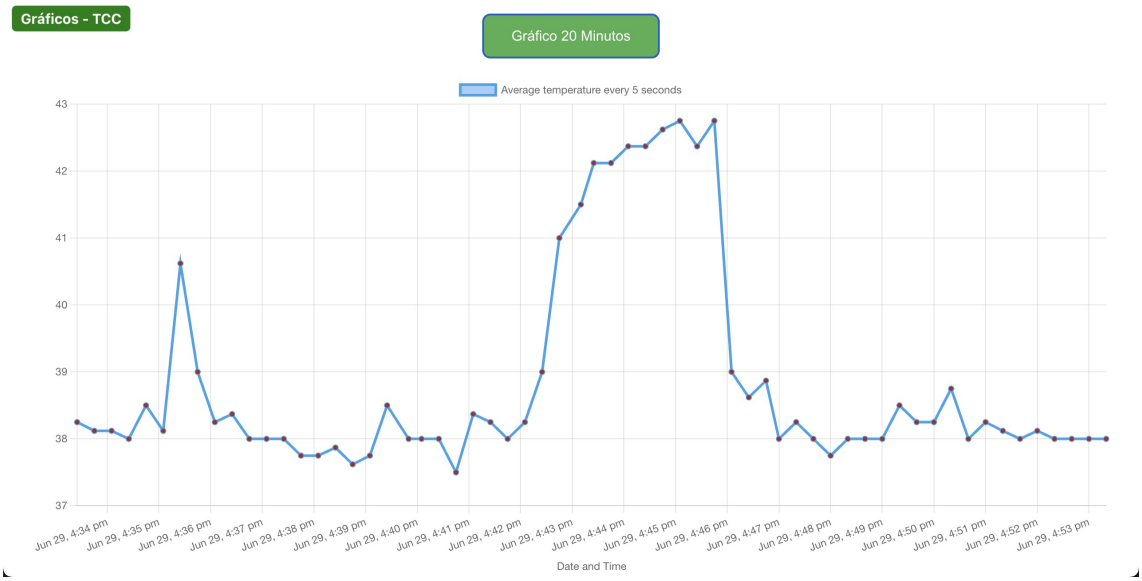

Figura 15 – Acompanhamento em tempo real das mudanças de temperatura. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

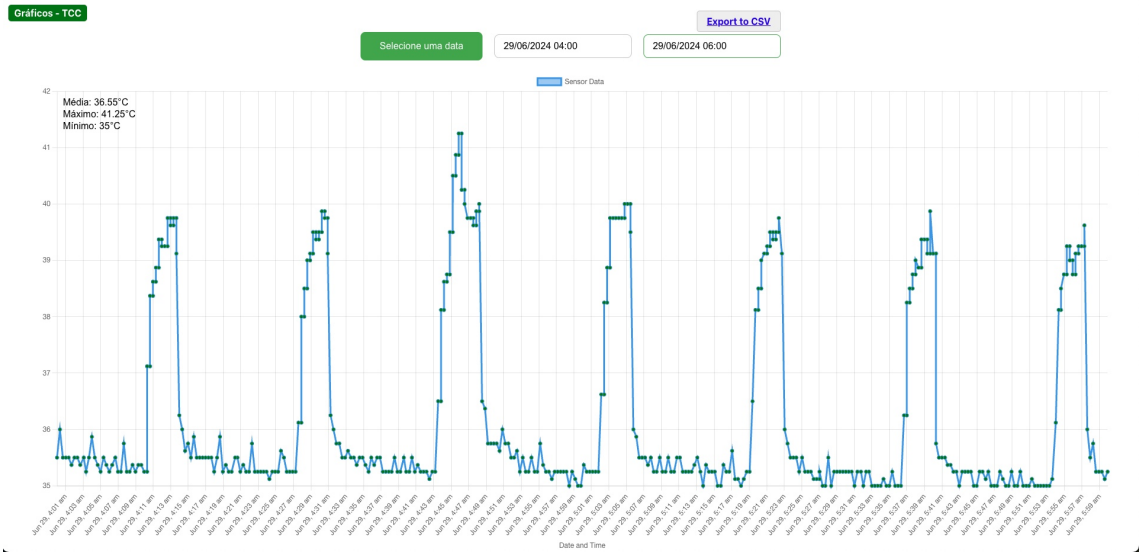

Figura 16 – Gráfico mostrando a oscilação padronizada na temperatura da CPU. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

Durante a metade do processo, foi possível utilizar o gráfico principal e, devido à facilidade de escolher os períodos, foi possível verificar fenômenos que poderiam passar despercebidos, como uma oscilação padronizada na temperatura da CPU durante todo o tempo que o computador ficou sem utilização intensa. A respeito da sazonalidade não foi possível encontrar as possíveis causas. No detalhe da Figura 16, é possível ver com clareza as possibilidades fornecidas pelo sistema.

Figura 17 – Gráfico com o pico de temperatura observada durante todo o processo. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

Além disso, algumas informações interessantes podem ser visualizadas por meio desse gráfico, como os valores máximo, mínimo e médio durante o período especificado. Buscando outro momento é possível observar ainda um exemplo onde o computador foi estressado ao máximo na Figura 17, sendo possível verificá-lo entre as marcações de horário de 07:33 e 7:35, por meio de ferramentas próprias para testar e monitorar a CPU.

Esse gráfico permite ainda a exportação dos dados, de forma que após a visualização, eles podem ser exportados no formato CSV para serem futuramente analisados com mais critérios, ou ainda como parâmetros de mudança.

Por fim, a visualização gráfica dos dados diários, Figura 18, proporcionou uma visão abrangente do processo, evidenciando a inserção de informações ao longo de um período de 24 horas. Dentre os destaques, observa-se um pico de temperatura associado à execução de um programa de estresse, além de oscilações padrões consistentes com as previamente descritas.

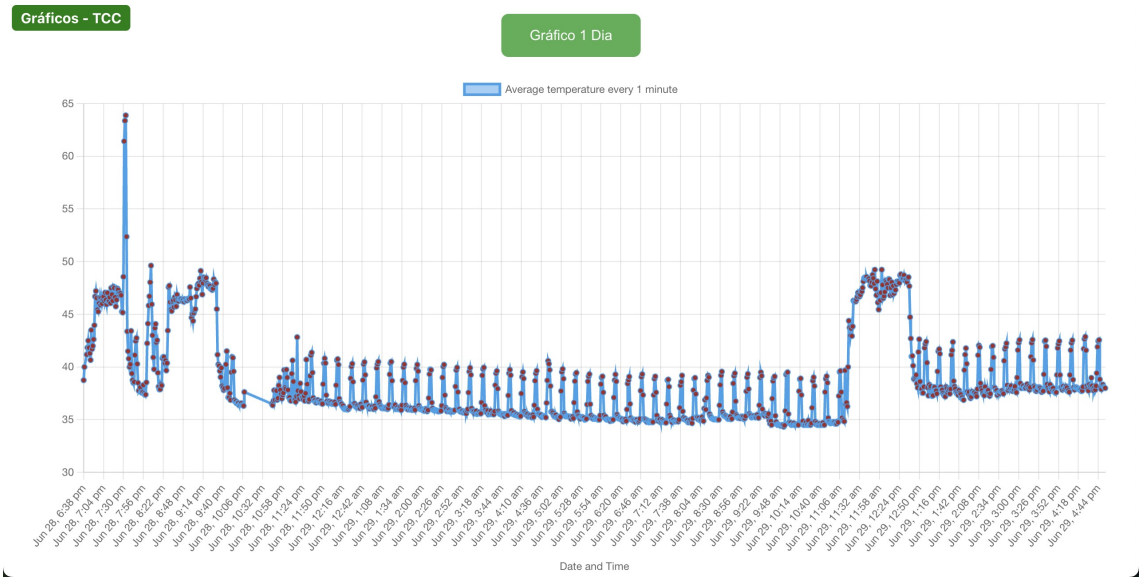

Figura 18 – Gráfico com temperaturas da CPU do computador ao longo de um dia. O eixo X representa a data e hora (mês, dia, hora e minuto), e o eixo Y representa a temperatura em graus Celsius (Temperatura °C).

5 Conclusão

Conclui-se que foi possível desenvolver um sistema eficiente de monitoramento de dados. O projeto resultou em um sistema robusto e de custo acessível. O sistema permitiu não apenas o monitoramento contínuo e a identificação de mudanças em curtos períodos, mas também a detecção de padrões difíceis de verificar sem uma interface detalhada.

Observaram-se oscilações no sistema, especialmente quando o computador estava em repouso. Isso demonstrou a presença de processos internos que influenciam a temperatura da CPU, mas que não eram esperados. Assim, fica clara a relevância de sistemas de monitoramento que permitam a visualização em tempo real e ao longo do tempo. Isso possibilita a identificação de características que poderiam passar despercebidas em outros momentos. No contexto industrial, seria possível otimizar processos e identificar problemas de forma preventiva. A possibilidade de exportar dados expande ainda mais as possibilidades do projeto.

5.1 Sugestões para Trabalhos Futuros

Em reconhecimento a complexidade do projeto, algumas melhorias foram pensadas e estão pontuadas a seguir:

- Implementação de interfaces que possibilitem a criação e a gestão dos sensores.
- Avanço na modularização dos processos, permitindo um desenvolvimento contínuo e escalável.
- Implementação da funcionalidade para alteração da variável de visualização.
- Implementação de um mecanismo de envio de e-mails quando a variável de visualização exceder um valor pré-estabelecido.
- Implementação de reconhecimento de padrões via IA para os dados coletados.

Referências

ALI, M. A.; MIRY, A. H.; SALMAN. IoT Based Water Tank Level Control System Using PLC. *International Conference on Computer Science and Software Engineering*, p. 7–12, 2020. DOI: 10.1109/CSASE48920.2020.9142067. Citado 1 vez na página 12.

ALTUS. *CLP Altus com MQTT*. Disponível em: https://www.altus.com.br/post/308/clp-altus-com-mqtt. Acesso em: 26 jun. 2024. 2021. Disponível em: https://www.altus.com.br/post/308/clp-altus-com-mqtt. Citado 1 vez na página 11.

AUTOMATION, Indmall. *How Long Do PLC Components Typically Last?* Acessado em: 16 de setembro de 2024. 2024. Disponível em: https://www.indmallautomation.com/faq/how-long-do-plc-components-typically-last/. Citado 1 vez na página 11.

BROWN, Michael; DAVIS, Robert. Practical implementation of an ESP32-based smart surveillance system. In: PROCEEDINGS of the International Conference on IoT Technologies. 2022. P. 45–56. Disponível em: https://www.exampleconference.org/proceedings/2022/iot/0034.pdf. Citado 1 vez na página 13.

CODE B. *Best IoT Cloud Services in 2024*. Acesso em: 21 set. 2024. 2024. Disponível em: https://code-b.dev/blog/best-iot-cloud-services. Citado 1 vez na página 12.

DIESEL - A Safe, Extensible ORM and Query Builder for Rust. 2024. https://diesel.rs/. Accessed: 2024-06-30. Citado 1 vez na página 39.

DIMENSIONAL. *Controlador CLP 12/24V 8DI Logo! 6ED10521MD080BA2 Siemens*. Acesso em: 18 ago. 2024. 2024. Disponível em: https://www.dimensional.com.br/controlador-clp-12-24v-8di-logo-6ed10521md080ba2-siemens/p?idsku=1415556&gad_source=4&gclid=Cj0KCQjwt4a2BhD6ARIsALgH7DqkVqsj6cDonuhW10nPoRz7R1bGR4choWPgUd-Afgo0JLO_MaW5GXgaAkt_EALw_wcB. Citado 1 vez nas páginas 12, 17.

DOCKER. *Docker Engine Documentation*. 2024. https://docs.docker.com/engine/. Accessed: 2024-06-30. Citado 1 vez na página 26.

ELECTRIC, Schneider. *Obsolescence, or It's Time to Replace Your Controller*. 2013. Disponível em: https://blog.se.com/industry/machine-and-process-management/2013/05/09/obsolescence-or-its-time-to-replace. Citado 1 vez na página 11.

EMBARCADOS. *CLP - Parte 1: Introdução aos Controladores Lógicos Programáveis*. Disponível em: https://embarcados.com.br/clp-parte1/. Acesso em: 30 jun. 2024. 2024. Disponível em: https://embarcados.com.br/clp-parte1/. Citado 1 vez na página 18.

EMBARCADOS. *Placa de Desenvolvimento Arduino Portenta H7*. Disponível em: https://embarcados.com.br/placa-de-desenvolvimento-arduino-portenta-h7/. Acesso em: 30 jun. 2024. 2024. Disponível em: https://embarcados.com.br/placa-de-desenvolvimento-arduino-portenta-h7/. Citado 1 vez na página 15.

ESPRESSIF SYSTEMS. *ESP32 Technical Reference Manual*. Versão 4.4, 2020. Acesso em: 21 set. 2024. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. Citado 1 vez na página 13.

ESPRESSIF SYSTEMS. *ESP32-C3-WROOM-02 Datasheet*. Acesso em: 18 ago. 2024. 2024. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32-c3-wroom-02_datasheet_en.pdf. Citado 0 vez na página 15.

FERREIRA, Israel Vieira; BIGHETI, Jeferson Andr'e; GODOY, Eduardo Paciencia. Development of a Wireless Gateway for Industrial Internet of Things Applications. *IEEE Latin America Transactions*, IEEE, v. 17, n. 8, p. 1298–1305, 2019. DOI: 10.1109/TLA.2019.8986441. Disponível em: https://www.researchgate.net/publication/339109165. Citado 1 vez na página 12.

FINDER. *Finder opta – O primeiro relé lógico programável*. Acesso em: 11 ago. 2024. 2023. Disponível em: https://www.findernet.com/pt/brasil/news/finder-opta-o-primeiro-rele-logico-programavel/. Acesso em: 11 ago. 2024. Citado 1 vez na página 11.

FINDER. *Finder Opta: O Primeiro Relé Lógico Programável*. Disponível em: https://www.findernet.com/pt/brasil/news/finder-opta-o-primeiro-rele-logico-programavel/. Acesso em: 26 jun. 2024. 2021. Disponível em: https://www.findernet.com/pt/brasil/news/finder-opta-o-primeiro-rele-logico-programavel/. Citado 1 vez na página 11.

FRANKLIN, Michael J.; PATTERSON, David A.; FOX, Armando. Time Series Databases: Past, Present, and Future. *Communications of the ACM*, v. 61, n. 8, p. 32–34, 2017. Citado 1 vez na página 23.

INFLUXDATA. *Relational Databases vs Time Series Databases*. Accessed on: 11 ago. 2024. 2024. Disponível em: https://www.influxdata.com/blog/relational-databases-vs-time-series-databases/. Citado 1 vez na página 23.

INSTRUMENTS, Texas. *Sensorless Field Oriented Control of 3-Phase Permanent Magnet Synchronous Motors*. Nov. 2013. https://www.ti.com/lit/an/sprabv9a/sprabv9a.pdf. Disponível em: https://www.ti.com/lit/an/sprabv9a/sprabv9a.pdf. Citado 1 vez na página 11.

MAIER, Alexander; SHARP, Andrew; VAGAPOV, Yuriy. Comparative analysis and practical implementation of the ESP32 microcontroller module for the internet of things, p. 143–148, 2017. DOI: 10.1109/ITECHA.2017.8101926. Citado 1 vez na página 13.

MODBUS ORGANIZATION. *About Us*. 2024. Accessed: 2024-08-18. Disponível em: https://modbus.org/about_us.php. Citado 0 vez na página 19.

MODBUS ORGANIZATION. *Modbus Application Protocol Specification V1.1b3*. 2002. Disponível em: https://www.modbus.org/specs.php. Citado 2 vezes na página 20.

MQTT ORGANIZATION. *MQTT Official Website*. 2024. Accessed: 2024-08-18. Disponível em: https://mqtt.org/. Citado 0 vez na página 21.

NASCIMENTO, João Maria Araújo do; LUCENA, Pedro Berretta de. *Redes para Automação Industrial - PROTOCOLO MODBUS*. 2003. Citado 1 vez na página 20.

PAVAO, Marcus. *TCC Marcus Pavao*. 2024. Accessed: 2024-06-11. Disponível em: https://github.com/marcuspavao/TCC_Marcus_Pavao. Citado 5 vezes nas páginas 34, 39, 40, 46, 49.

PLCOPEN. *Introdução à Norma IEC 61131-3 - Linguagens de Programação para Controladores Programáveis*. Mar. 2004. https://plcopen.org/sites/default/files/downloads/intro_iec_march04_portuguese.pdf. Disponível em: https://plcopen.org/sites/default/files/downloads/intro_iec_march04_portuguese.pdf. Citado 1 vez na página 11.

RUST DOCUMENTATION. *Official Rust Documentation*. 2022. Disponível em: https://doc.rust-lang.org/. Acesso em: 4 jun. 2024. Citado 1 vez na página 26.

RUST FOUNDATION. *Rust Foundation Blog*. 2022. Disponível em: https://blog.rust-lang.org/. Acesso em: 4 jun. 2024. Citado 1 vez nas páginas 25, 26.

SANTOS, Bruno P et al. *Internet das Coisas: da Teoria à Prática*. 2016. Citado 1 vez na página 16.

SIEMENS. *SIMATIC S7-1500, SIMATIC S7-1200, ET 200SP: MQTT Client*. 2.1. ed. Jun. 2021. Disponível em: https://cache.industry.siemens.com/dl/files/872/109748872/att_1006438/v4/109748872_MQTT_Client_DOKU_V2-1_en.pdf. Acesso em: 26 jun. 2024. Citado 1 vez na página 11.

TANENBAUM, Andrew S.; STEEN, Maarten van. *Distributed Systems: Principles and Paradigms*. 2. ed. Upper Saddle River, NJ: Pearson Prentice Hall, 2007. ISBN 978-0-13-239227-3. Citado 1 vez na página 16.

THE Difference Between GraphQL and REST - Amazon Web Services. Disponível em: https://aws.amazon.com/pt/compare/the-difference-between-graphql-and-rest/. Acesso em: 4 jun. 2024. Citado 1 vez na página 45.

TIMESCALE. *Timescale Official Website*. 2024. Accessed: 2024-08-18. Disponível em: https://www.timescale.com/. Citado 1 vez na página 25.

VYAS, Daiwat A.; BHATT, Dvijesh; JHA, Dhaval. IoT: Trends, Challenges and Future Scope. *International Journal of Computer Science and Communication*, v. 7, n. 1, p. 1–20, 2024. Acesso em: 21 set. 2024. Disponível em: https://www.csjournals.com/IJCSC/PDF7-1/28.%20Vyas.pdf. Citado 1 vez na página 14.

WAWHAL, Rishichandra. *Building Real-Time Charts With GraphQL And Postgres*. Mar. 2019. Disponível em: https://www.smashingmagazine.com/2019/03/realtime-charts-graphql-postgres/. Acesso em: 4 jun. 2024. Citado 2 vezes nas páginas 12, 42.

WILLIAN, Jean et al. *SISTEMA DE AUTOMATIZAÇÃO RESIDENCIAL DE BAIXO CUSTO CONTROLADO PELO MICROCONTROLADOR ESP32 E MONITORADO VIA SMARTPHONE*. 2019. Citado 1 vez na página 16.