Universidade do Estado do Amazonas
Escola Superior de Tecnologia
Curso de Engenharia de Controle e Automação

José Luís Maciel Pinto

Desenvolvimento de um Sistema Supervisório para Medição de Voltagem Utilizando MQTT

Manaus-AM
12 de Junho de 2025

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

P659d Pinto, José Luís Maciel
Desenvolvimento de um Sistema Supervisório para Medição de Voltagem Utilizando MQTT / José Luís Maciel Pinto . Manaus : [s.n], 2025.
62 f.: il., color.; 21,0 cm.

TCC - Graduação em Engenharia de Controle e Automação- Universidade do Estado do Amazonas, Manaus, 2025.
Inclui Bibliografia.
Inclui Apêndice.
Orientador: Moisés Pereira Bastos.

1. Supervisório. 2. IoT. 3. MQTT. 4. ESP8266. 5. Tensão. I. Moisés Pereira Bastos (Orient.) II. Universidade do Estado do Amazonas. III. Título

CDU(1997)681.5

Universidade do Estado do Amazonas
Escola Superior de Tecnologia
Curso de Engenharia de Controle e Automação

José Luís Maciel Pinto

Desenvolvimento de um Sistema Supervisório para Medição de Voltagem Utilizando MQTT

Projeto de Pesquisa desenvolvido durante a disciplina de Trabalho de Conclusão de Curso II e aprovado pela banca avaliadora do Curso de Engenharia de Controle e Automação da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro de Controle e Automação.

Orientador: Prof. M.Sc. Moises Pereira Bastos

Manaus-AM
12 de Junho de 2025

José Luís Maciel Pinto

Desenvolvimento de um Sistema Supervisório para Medição de Voltagem Utilizando MQTT

Projeto de Pesquisa desenvolvido durante a disciplina de Trabalho de Conclusão de Curso II e aprovado pela banca avaliadora do Curso de Engenharia de Controle e Automação da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro de Controle e Automação.

Aprovado em 12 de Junho de 2025.

Prof. M.Sc. Moises Pereira Bastos
Orientador – UEA

Prof. Dr. Israel Mazaira Morales – UEA

Prof. Dr. Miguel Angel Orellana Postigo – UEA

Documento assinado digitalmente
gov.br
RAMAYANA ASSUNCAO MENEZES JUNIOR
Data: 16/06/2025 17:55:59-0300
Verifique em https://validar.iti.gov.br

Prof. Ramayana Assunção Menezes Júnior – UEA

Agradecimento

Gostaria de agradecer primeiramente aos meus pais, por toda a dedicação, pelas escolhas, decisões e abdicações feitas ao longo dos anos para eu ter o privilégio de cursar engenharia em uma universidade pública como a Universidade do Estado do Amazonas.

À minha esposa, Brenda, e à minha filha, Maria Júlia, minha gratidão por, nos últimos anos, me darem força e motivos reais para não desistir e seguir em frente. Não foi fácil, mas o resultado chegou, e conseguimos.

Agradeço a todos os professores, colegas de curso e amigos que, de forma direta ou indireta, contribuíram para minha formação, compartilhando conhecimentos, vivências e conselhos valiosos. Um agradecimento especial à Anna, secretária da coordenação do Curso de Engenharia de Controle e Automação, por sua paciência, atenção e dedicação diária.

Sou também grato pelas oportunidades que me foram dadas, especialmente no Hub de Tecnologia e Inovação e na StartHub Empresa Júnior, onde iniciei, de fato, minha trajetória profissional e escrevi as primeiras linhas do meu currículo. Linhas essas que tiveram um impacto significativo no futuro que hoje se concretiza.

Resumo

A Internet das Coisas (IoT) tem transformado várias áreas ao possibilitar a conexão de sensores e dispositivos a sistemas de controle e monitoramento em tempo real. Nesse cenário, a criação de sistemas de monitoramento para medir a tensão em ambientes industriais se tornou importante para aumentar a eficiência operacional e facilitar a manutenção preventiva. Um dos desafios mais significativos que esses sistemas enfrentam é assegurar uma comunicação que seja tanto confiável quanto eficiente, empregando protocolos que exijam baixo consumo de energia e funcionem continuamente em contextos com recursos limitados. Com isso, esta pesquisa desenvolveu e implementou um sistema capaz de realizar medições precisas da tensão elétrica, transmitindo os dados coletados de forma contínua e confiável através do protocolo MQTT para uma plataforma supervisória na nuvem. A metodologia adotada envolveu a utilização do sensor ZMPT101B para captura dos sinais elétricos, um microcontrolador ESP8266 com conectividade Wi-Fi para processamento e envio dos dados, e a plataforma Adafruit IO para visualização e gestão das informações via dashboards configuráveis. Foram definidos limites operacionais de tensão entre 60V e 140V, com implementação de alertas automáticos em casos de subtensão ou sobretensão. Os resultados demonstraram a precisão das medições, a estabilidade da comunicação via MQTT e a eficiência da supervisão remota. O sistema desenvolvido mostrou-se viável, de baixo custo e aplicável a ambientes industriais, acadêmicos e laboratoriais que demandem monitoramento elétrico contínuo.

Palavras-chave: Supervisório; IoT; MQTT; ESP8266; ZMPT101B; Tensão.

Abstract

The Internet of Things (IoT) has transformed many areas by making it possible to connect sensors and devices to control and monitoring systems in real time. In this scenario, the creation of monitoring systems to measure voltage in industrial environments has become important for increasing operational efficiency and facilitating preventive maintenance. One of the main challenges faced by these systems is to guarantee communication that is both reliable and effective, using protocols that require low power consumption and operate continuously in resource-constrained environments. With this in mind, this research has developed and implemented a system capable of taking precise electrical voltage measurements, transmitting the data collected continuously and reliably via the MQTT protocol to a supervisory platform in the cloud. The methodology adopted involved using the ZMPT101B sensor to capture the electrical signals, an ESP8266 microcontroller with Wi-Fi connectivity to process and send the data, and the Adafruit IO platform to visualize and manage the information via configurable dashboards. Operating voltage limits were set between 60V and 140V, with automatic alerts implemented in cases of undervoltage or overvoltage. The results demonstrated the accuracy of the measurements, the stability of the communication via MQTT and the efficiency of the remote supervision. The system developed proved to be viable, low-cost and applicable to industrial, academic and laboratory environments that require continuous electrical monitoring.

Keywords: Supervisory; IoT; MQTT; ESP8266; ZMPT101B; Voltage.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

A Internet das Coisas (IoT) tem revolucionado diversos setores industriais, permitindo a integração de sensores e dispositivos com sistemas de controle e monitoramento em tempo real. Dentro desse contexto, o desenvolvimento de sistemas supervisórios para a medição de voltagem em ambientes industriais vem se tornando uma aplicação fundamental para melhorar a eficiência operacional e a manutenção preditiva. Um dos principais desafios nesses sistemas é garantir uma comunicação confiável e eficiente, utilizando protocolos de baixo consumo de energia que possam operar de forma contínua em ambientes com restrições de recursos.

O protocolo MQTT *(Message Queuing Telemetry Transport)* tem-se destacado como uma escolha ideal para essas aplicações. Como explorado em Masdani e Darlis (2018) e Ochoa *et al.* (2023), o MQTT apresenta vantagens significativas em termos de eficiência energética e simplicidade de implementação, especialmente quando comparado ao HTTP. O protocolo se baseia no modelo *publish/subscribe*, que é particularmente adequado para o monitoramento contínuo de variáveis industriais, como a voltagem, uma vez que permite uma comunicação eficiente em tempo real entre dispositivos e o sistema supervisório.

Além disso, conforme discutido na publicação Patel, Patel e Scholar (2016), o MQTT é uma das tecnologias capacitadoras chave para a IoT, permitindo a coleta de dados em larga escala e a sua transmissão para sistemas centralizados de análise e supervisão. A combinação dessas tecnologias com a arquitetura IoToferece às indústrias a possibilidade de otimizar seus processos, melhorando a precisão e a confiabilidade das medições de voltagem, e permitindo a detecção precoce de falhas ou anomalias no sistema.

Nesse sentido,este trabalho parte da seguinte pergunta de pesquisa: como o desenvolvimento de um sistema supervisório para medição de voltagem utilizando MQTT pode beneficiar a indústria em termos de precisão, eficiência e segurança na coleta de dados, atendendo à demanda crescente por sistemas inteligentes e conectados que impulsionam a Indústria 4.0? Parte-se da hipótese de que o desenvolvimento de um sistema supervisório para medição de voltagem com protocolo MQTT irá resultar em uma melhora na eficiência e precisão dos dados coletados de voltagem em uma linha de produção de placas, além de proporcionar maior segurança no ambiente industrial e servir de suporte para futuras auditorias. Acredito que os benefícios esperados irão superar desafios relacionados a possíveis integrações com sistemas legados, à escalabilidade em ambientes industriais complexos,

à necessidade de medidas adicionais de segurança na rede interna, resultando em uma solução eficaz para a medição de voltagem em ambientes industriais automatizados. A justificativa para este trabalho baseia-se na crescente demanda por soluções inteligentes e conectadas, que promovam a automação, rastreabilidade e confiabilidade em tempo real, pilares fundamentais da Indústria 4.0. O uso do MQTT, com suas características de baixo consumo, baixa latência e confiabilidade, mostra-se alinhado com essas necessidades. Dessa forma, este estudo visa não apenas propor uma solução viável e de baixo custo, mas também contribuir para o avanço de práticas industriais baseadas em IoT, promovendo a inovação tecnológica e a otimização de processos em ambientes industriais automatizados.

1.1 Objetivos

1.1.1 Geral

Desenvolver e implementar um sistema supervisório para a medição de voltagem, utilizando o protocolo MQTT. Esse sistema visa oferecer uma solução avançada para o monitoramento preciso e eficiente da voltagem em ambientes industriais, assegurando alta confiabilidade e segurança na transmissão e análise de dados. Além disso, o projeto busca demonstrar como a integração de tecnologias atuais de comunicação pode aprimorar as práticas de monitoramento e controle dentro do contexto da automação industrial.

1.1.2 Específicos

- Realizar levantamento bibliográfico nos temas relacionados à pesquisa
- Descrever os conceitos de sistemas supervisórios;
- Descrever as funcionalidades de sistemas supervisórios;
- Estruturar a aplicação do protocolo MQTT em sistemas supervisórios;
- Desenvolver a Arquitetura do Sistema para medição de voltagem.
- Realizar testes no sistema desenvolvido

2 Referencial Teórico

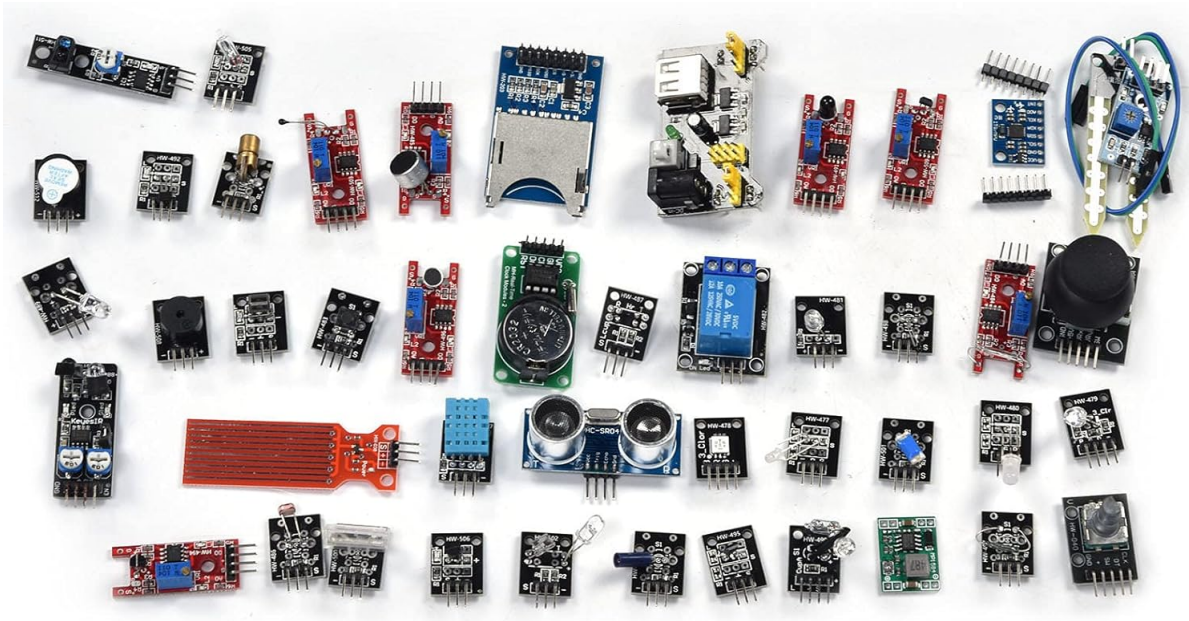
Neste capítulo serão abordadas as referências teóricas utilizadas como base para o desenvolvimento do trabalho, tais como: Sistema de supervisão e controle; IoT (Internet das Coisas); análise do consumo de energia do MQTT; aplicação do protocolo MQTT para monitoramento de sistemas, mais especificamente, detecção de valores de subtensão e sobretensão em sistemas elétricos.

2.1 Internet das Coisas (IoT)

IoT refere-se ao uso de objetos físicos, como sensores e atuadores, representados na 1, com capacidade de comunicação, permitindo, assim, a coleta, o processamento e a troca de dados por meio da Internet. Essa tecnologia proporciona a automação e o monitoramento em ambientes industriais, residenciais e em cidades inteligentes. A arquitetura da IoT é geralmente composta por quatro camadas principais: leitura, conexão, análise e execução. A leitura tem o objetivo de coletar os dados do ambiente por meio dos sensores e atuadores. A conexão é responsável pela transmissão dos dados coletados. A análise tem a intenção de tratar os dados para que ocorra a execução de alguma ação para o usuário, como a parada de algum equipamento, alertas visuais ou auditivos, e fornecimento de informações para alguma tomada de decisão manual ou automática.

A implementação da IoT requer dispositivos com baixo consumo de energia e capacidade de comunicação eficiente. Nesse contexto, microcontroladores com conectividade integrada, como o ESP8266, e protocolos de comunicação leves, como o MQTT, são amplamente utilizados.

Figura 1: Exemplo de sensores, atuadores e outros componentes para *IoT*



Fonte: <https://m.media-amazon.com/images/I/71EkThh9G4L._AC_SL1500_.jpg>

2.2 MQTT (*Message Queuing Telemetry Transport*)

O MQTT é um protocolo de comunicação amplamente utilizado em sistemas de Internet das Coisas (IoT) devido à sua leveza, simplicidade e eficiência. Originalmente desenvolvido pela IBM em 1999 para monitorar oleodutos, o MQTT foi projetado para operar em redes de baixa largura de banda e ambientes com conectividade intermitente, características comuns em dispositivos IoT (STANDARD, 2014).

2.2.1 Definição e Funcionamento

Funciona como um protocolo de mensagem que segue o modelo (publicar/assinar), onde os dispositivos podem atuar como *publishers* (publicadores) ou *subscribers* (assinantes). Um servidor intermediário, chamado de broker, é responsável por gerenciar a troca de mensagens entre os dispositivos. O publisher envia uma mensagem para um tópico específico e o broker a repassa para todos os subscribers que estão "inscritos"nesse tópico (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

Essa abordagem de comunicação é fundamental para a IoT, pois permite que dispositivos distribuídos troquem informações de forma eficiente, sem a necessidade de um fluxo contínuo de comunicação ponto a ponto, como ocorre em protocolos como o HTTP (LIU *et al.*, 2020).

2.2.2 Características do MQTT

Algumas das principais características que tornam o MQTT ideal para aplicações IoT incluem: Leveza: O MQTT minimiza a sobrecarga de comunicação, enviando apenas pacotes de dados essenciais, o que reduz o consumo de banda e energia. Isso é especialmente relevante para dispositivos IoT que operam com recursos limitados (PAZIENZA *et al.*, 2019). Escalabilidade: O protocolo é altamente escalável, suportando desde pequenos dispositivos com baixa capacidade até redes de IoT complexas e de grande escala. Sua arquitetura publish/subscribe facilita a adição de novos dispositivos sem sobrecarregar a infraestrutura de comunicação (ANAND *et al.*, 2020). Qualidade de Serviço (QoS): O MQTT oferece três níveis de QoS (*Quality of Service*), permitindo controlar a entrega e a confirmação das mensagens enviadas. Esses níveis variam de um envio simples sem confirmação até a garantia de entrega segura e em ordem, independentemente da qualidade da conexão (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

Baixo Consumo de Energia: Comparado a outros protocolos, como o HTTP, o MQTT se destaca pelo menor consumo de energia, o que o torna ideal para dispositivos alimentados por bateria, como sensores e atuadores (VERMA; DESWAL, 2023).

2.2.3 Comparação com Outros Protocolos

Em termos de eficiência energética e adequação para dispositivos IoT, estudos como o de Muñoz, Morales e Sánchez-Molina (2024) mostram que o MQTT consome significativamente menos energia em comparação ao HTTP. Enquanto o HTTP utiliza um modelo de comunicação request/response (requisição/resposta), o MQTT, por ser baseado no modelo *publish/subscribe*, evita o envio constante de pacotes de requisição, o que reduz o consumo de energia e prolonga a vida útil de dispositivos IoT.

Além disso, ao operar sobre o protocolo TCP/IP, o MQTT é mais adequado para situações em que a confiabilidade e a entrega de dados são essenciais, diferentemente de protocolos como o CoAP (*Constrained Application Protocol*), que utiliza o UDP, mas sacrifica a garantia de entrega para obter menor latência (GUTH *et al.*, 2016).

2.2.4 Aplicações do MQTT

O MQTT tem sido amplamente adotado em diversos setores, desde a automação industrial até cidades inteligentes e monitoramento remoto. Na área de automação residencial, por exemplo, ele é utilizado para controlar dispositivos como termostatos, lâmpadas e sistemas de segurança. Já na indústria, o protocolo é utilizado em sistemas supervisórios de controle e aquisição de dados (SCADA), onde sensores e dispositivos remotos comunicam informações em tempo real para centrais de controle (ELKHODR; SHAHRESTANI; CHEUNG, 2013).

Outros exemplos incluem a integração de sensores para monitoramento de transporte em cadeia de frio, onde o MQTT permite o monitoramento remoto da temperatura e umidade de cargas perecíveis durante o transporte (MUÑOZ; MORALES; SÁNCHEZ-MOLINA, 2024).

2.2.5 Desafios e Limitações

Embora o MQTT ofereça diversos benefícios, também apresenta desafios que precisam ser considerados. A segurança, por exemplo, é um fator crítico. Como o protocolo foi desenvolvido com foco em simplicidade e leveza, ele não possui mecanismos de segurança integrados, dependendo da implementação de camadas de segurança externas, como SSL/TLS para criptografia de dados Liu *et al.* (2020). Além disso, o MQTT não foi originalmente projetado para lidar com ambientes móveis de alta latência ou desconexões frequentes, o que pode ser um desafio em algumas aplicações de IoT (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

2.3 Microcontroladores

Microcontroladores são dispositivos semicondutores que integram, em um único chip, uma unidade central de processamento (CPU), memória (RAM e ROM/Flash) e periféricos de entrada e saída (I/O). Eles são projetados para executar tarefas específicas de controle e automação, sendo largamente utilizados em sistemas embarcados por seu baixo custo, consumo reduzido de energia e tamanho compacto.

No contexto da Engenharia de Controle e Automação, microcontroladores desempenham papel central no desenvolvimento de soluções embarcadas inteligentes, incluindo sistemas supervisórios baseados em Internet das Coisas (IoT), como é o caso deste trabalho.

2.3.1 Funcionamento de um Microcontrolador no Contexto de Sistemas Supervisórios

Em sistemas supervisórios embarcados, o microcontrolador atua como unidade central de aquisição e processamento de dados. Ele realiza a leitura de variáveis físicas a partir de sensores conectados aos seus pinos analógicos e/ou digitais, processa essas informações localmente (inclusive aplicando filtros ou calibrações), e então as transmite a uma plataforma supervisória via protocolo de comunicação (como o MQTT, por exemplo).

Além da aquisição e envio de dados, o microcontrolador também pode ser programado para executar respostas automáticas, como acionar relés, emitir sinais de alerta ou registrar eventos em memória local, dependendo das condições detectadas em tempo real. Sua atuação contínua e autônoma é essencial para a supervisão de grandezas físicas críticas,

como tensão, corrente ou temperatura.

2.3.2 Comparação do ESP8266 com Outros Microcontroladores

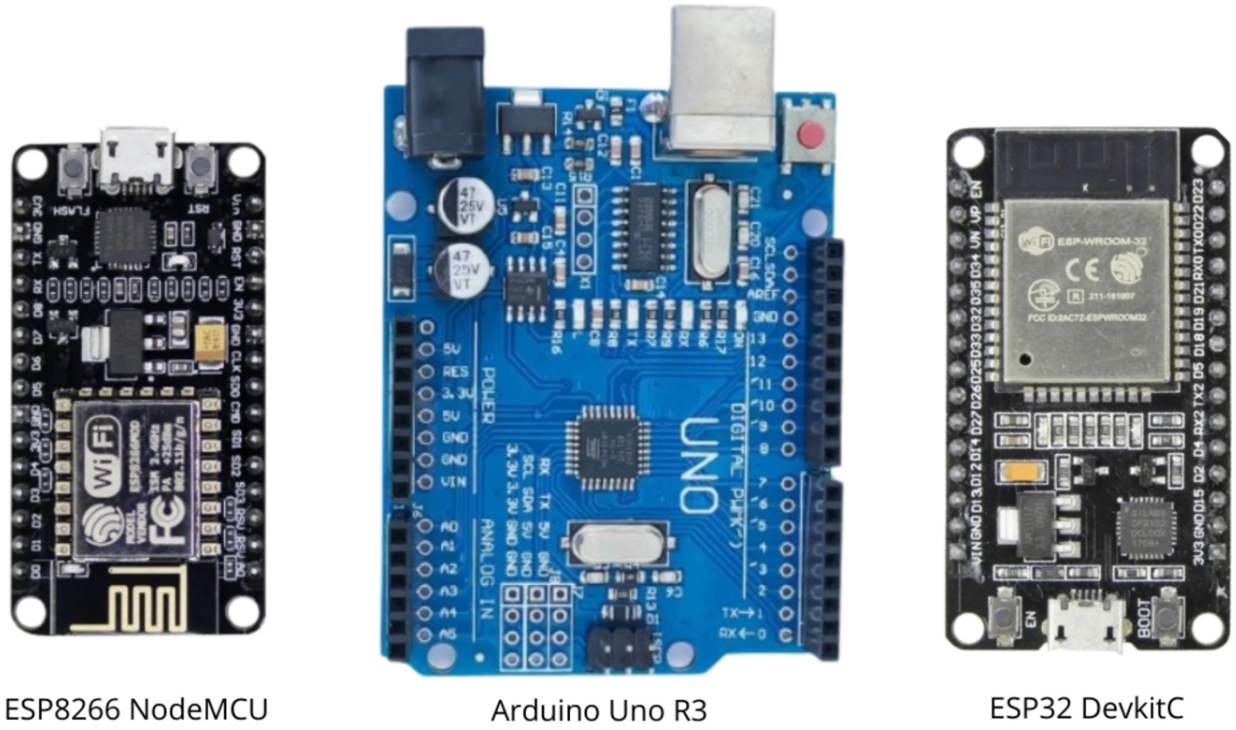
Entre os microcontroladores mais utilizados no desenvolvimento de sistemas embarcados destacam-se o ESP8266, ESP32 e Arduino UNO (ATmega328P). A seguir, na tabela 1, faço uma comparação com algumas características de microcontroladores, além de uma comparação física na figura 2:

Tabela 1: Comparação entre microcontroladores

Característica	**ESP8266**	**ESP32**	**Arduino UNO**
CPU	1 núcleo, 80/160 MHz	2 núcleos, 240 MHz	1 núcleo, 16 MHz
Memória RAM	50 kB	520 kB	2 kB
Memória Flash	4 MB (externa)	4 MB (externa)	32 kB
Wi-Fi	Sim	Sim	Não
Bluetooth	Não	Sim (BLE/Classic)	Não
Tensão de operação	3,3 V	3,3 V	5 V
GPIOs disponíveis	10	30	14
Preço	R$ 48,00	R$ 50,00	R$ 108,00

Fonte: Autor

Figura 2: Exemplos de microcontroladores



Fonte: Autor

O ESP8266 é bastante vantajoso em projetos que demandam conectividade Wi-Fi e baixo

consumo, sendo muito mais poderoso que o Arduino UNO em termos de processamento e comunicação, embora o ESP32 seja mais completo e robusto em aplicações mais exigentes. O Arduino, por sua vez, é ideal para iniciantes pela sua simplicidade e grande documentação, mas carece de recursos de conectividade sem fio nativos.

2.3.3 Aplicação

Microcontroladores são amplamente empregados em:

- **Sistemas supervisórios embarcados:** para leitura e monitoramento de variáveis elétricas e ambientais;
- **Automação industrial e predial:** para controle de motores, atuadores, iluminação e sistemas HVAC;
- **Dispositivos IoT:** como sensores inteligentes e coletores de dados com acesso à internet;
- **Sistemas de aquisição de dados e registradores de eventos (data loggers).**

No caso deste trabalho, tenho a intenção de usar o ESP8266 para medir a tensão de uma rede elétrica junto ao sensor ZMPT101B e enviar os dados a um servidor remoto via MQTT, permitindo supervisão remota com alertas e visualizações gráficas.

2.3.4 Limitações

Apesar de suas vantagens, os microcontroladores apresentam algumas limitações:

- Recursos computacionais limitados, o que inviabiliza tarefas de alto processamento;
- Capacidade reduzida de memória RAM e Flash;
- Operação em tempo real limitada;
- Sensibilidade à tensão de operação — o ESP8266, por exemplo, pode ser danificado facilmente por sinais de 5V sem proteção.

Essas limitações, no entanto, podem ser mitigadas com boas práticas de projeto, permitindo o desenvolvimento de sistemas embarcados confiáveis e eficientes.

2.4 Sensores e Sensores de Tensão

2.5 Sobretensão e Subtensão em Sistemas Elétricos

De acordo com a norma IEEE Std. 1159-2019, variações prolongadas na tensão elétrica podem ser classificadas como **subtensão** ou **sobretensão**, conforme a porcentagem em relação ao valor nominal. Em sistemas de 127V, tensões abaixo de 114V (90%) e acima de 140V (110%) são consideradas anormais. Essas condições podem comprometer o funcionamento de equipamentos eletrônicos, levando à perda de eficiência, aquecimento excessivo ou até danos permanentes nos componentes (IEEE Power Quality Standards Committee, 2019).

2.5.1 Subtensão

De acordo com Mohammed *et al.* (2025), a subtensão é caracterizada por uma queda de tensão que se mantém abaixo do nível mínimo necessário para a operação adequada de dispositivos, afetando principalmente motores e equipamentos sensíveis.

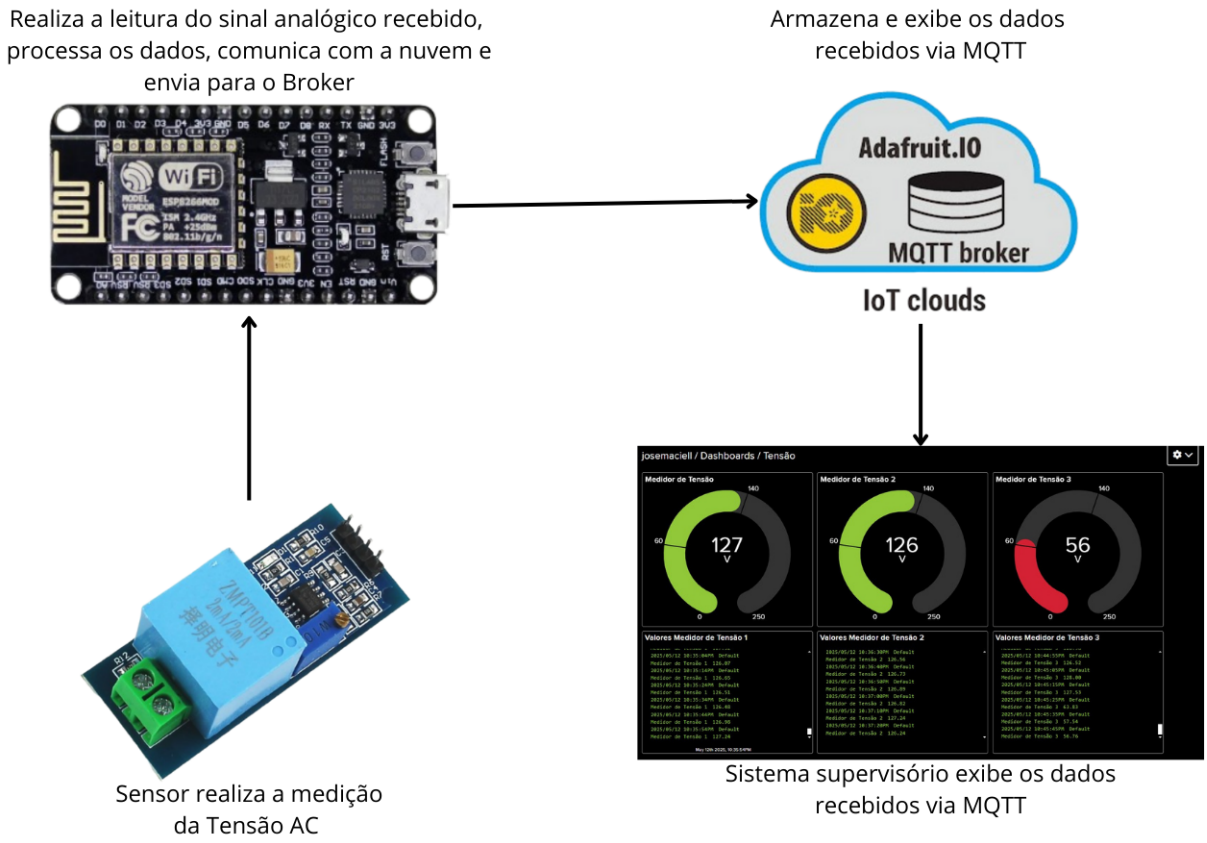
2.5.2 Sobretensão

A sobretensão implica um excesso de tensão acima da faixa segura, o que pode causar degradação acelerada de isolamentos, falhas em placas eletrônicas e riscos de queima de componentes (MOHAMMED *et al.*, 2025).

3 Metodologia

Para o trabalho em questão foi desenvolvido um sistema supervisório para medição de tensão utilizando um microcontrolador e protocolo MQTT, no qual diversas etapas são importantes para garantir a coleta, monitoramento e controle eficaz de dados de tensão em tempo real. Na figura 3, são detalhadas as etapas principais que compõem a metodologia para esse projeto.

Figura 3: Diagrama da arquitetura do projeto

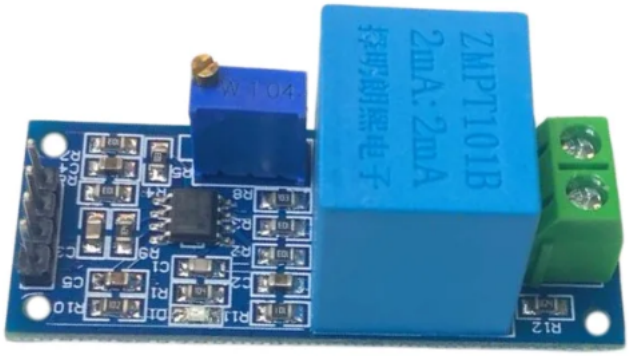


Fonte: Autor

Podemos identificar os quatro componentes principais deste projeto, sendo:

- **Sensor:** Foi escolhido como sensor o ZMPT101B, que é responsável por medir a tensão elétrica nas áreas de interesse. Para o contexto do projeto é um sensor adequado, ilustrado na figura 4.

Figura 4: Sensor medidor de tensão



Fonte: https://www.smartprojectsbrasil.com.br/sensor-de-tensao-ac-0-a-250v-voltimetro-zmpt101b

- **Controlador:** foi escolhido como dispositivos de borda (*edge devices*), o microcontrolador NodeMCU ESP8266 , que tem a responsabilidade de coletar as medições dos sensores e enviá-las para o servidor central utilizando o protocolo MQTT.
- **Placa Perfurada:** É a base física necessária para conexão dos componentes. Sua função é estrutural e de interconexão, tornando-se essencial para a materialização do circuito. Sem ela, seria muito difícil organizar e conectar os demais componentes de forma permanente e robusta.
- **MQTT Broker:** Foi escolhido o protocolo MQTT Broker do Adafruit IO, no qual, é um serviço que permite a comunicação de dispositivos IoT, no caso o ESP8266, e a nuvem da plataforma Adafruit, atuando como intermediário entre a leitura e a distribuição das mensagens.
- **Sistema Supervisório:** É usado a plataforma do Adafruit onde consiste em uma solução em nuvem voltada para aplicações de IoT, permitindo a integração entre sensores, microcontroladores e interfaces supervisórias de maneira simples e eficiente, além de ser possível configurar *dashboards* e ações de forma amigável.

Tais componentes serão explicados de maneira específica nos próximos itens deste capítulo.

3.1 Sensor ZMPT101B na Medição de Tensão

O sensor ZMPT101B é responsável por medir a tensão alternada (AC) da rede elétrica (até 250V), convertendo-a em um sinal analógico de baixa amplitude (0,V a 3,3,V), compatível com o microcontrolador ESP8266. Esse sensor utiliza um transformador com isolamento galvânico, o que garante a separação elétrica entre os circuitos de potência e de controle, aumentando a segurança do sistema.

Para iniciar o uso do sensor, é necessário passar pela fase de calibração do módulo sensor, o qual é um passo fundamental que antecede qualquer ajuste ou correção via *software*, assegurando precisão e confiabilidade das leituras. Diferente de uma simples correção via *software*, a calibração do módulo ZMPT101B inicia-se com um ajuste físico primordial: a manipulação do potenciômetro, um componente eletrônico ajustável presente diretamente na placa do módulo, ilustrado na figura 5. O uso da biblioteca `ZMPT101B.h` ajuda muito no momento da calibração, pois com o auxílio do multímetro para saber o valor de tensão real, conseguimos ajustar facilmente no sensor.

Figura 5: Potenciômetro de calibração do sensor

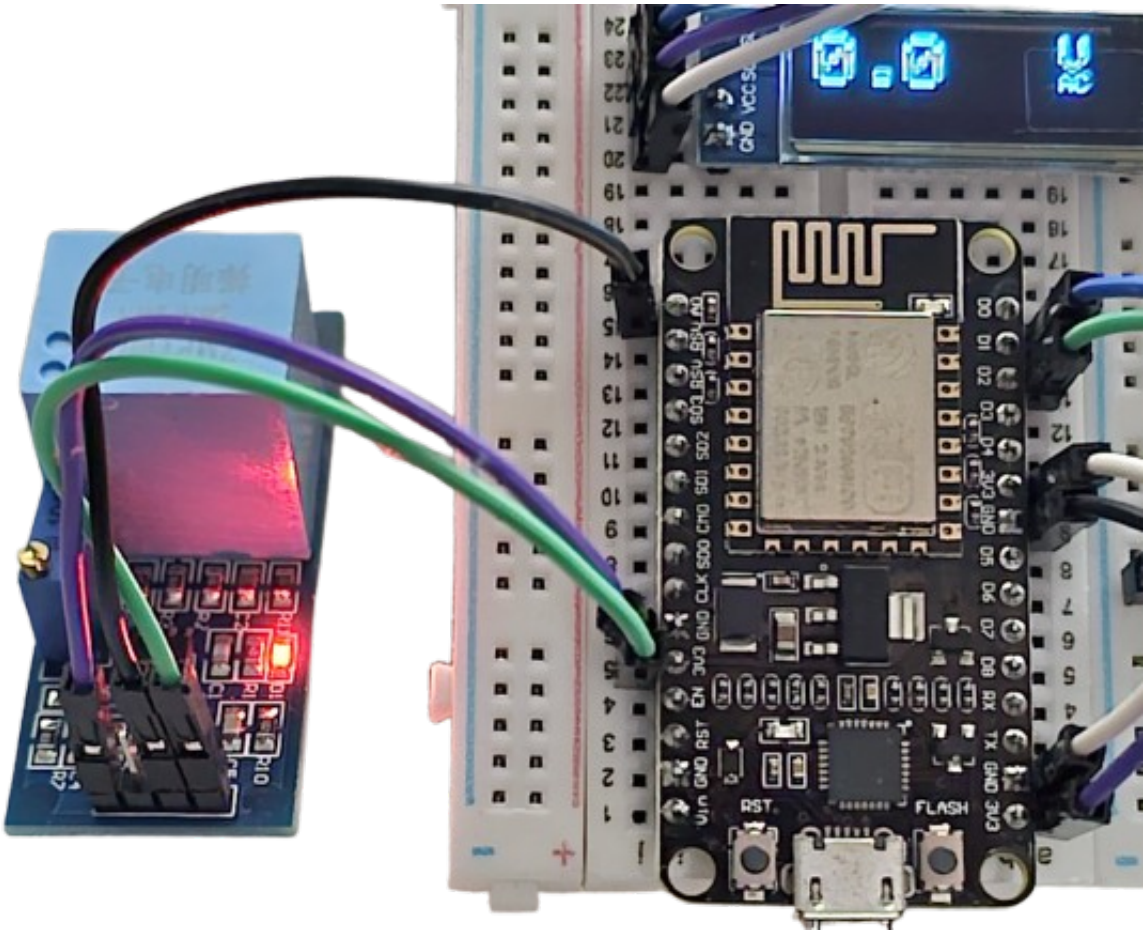
Fonte:Autor

A medição da tensão é realizada por meio do pino analógico A0 do ESP8266 e energizado por uma entrada 3V3 e uma GND disponível, conforme a figura 6. A biblioteca utilizada disponibiliza a função integrada `getRmsVoltage()`, responsável por calcular diretamente o valor eficaz (RMS) da tensão com base nas amostragens do sinal. Essa funcionalidade simplifica o processo de aquisição de dados, possibilita medições precisas em tempo real e facilita a calibração do sensor.

Para aumentar a estabilidade das leituras, os valores medidos são armazenados em um buffer circular e submetidos a um filtro de média com exclusão de valores extremos. Em seguida, aplica-se uma função de calibração ajustada empiricamente, a fim de compensar possíveis desvios não lineares do sensor.

Durante o processo de calibração, o sistema foi configurado para operar de forma confiável dentro da faixa de 30V a 230V, com maior precisão esperada entre 60V e 140V, que corresponde à zona de interesse definida para o monitoramento. A margem de erro

Figura 6: Conexão do sensor ZMPT101B ao pino AO, à 3V3 e à GND do ESP8266



Fonte: Autor

observada nas medições é de aproximadamente 1%, valor considerado aceitável para aplicações de supervisão em tempo real. O resultado é uma estimativa confiável da tensão da rede, capaz de identificar variações, subtensões e sobretensões de forma contínua, sendo essas informações representadas graficamente em um *dashboard*.

3.1.1 Código do sensor ZMPT101B

A Figura 7 apresenta a função `getAverageVoltage()`, responsável por realizar a leitura da tensão eficaz (valor RMS) fornecida pelo sensor ZMPT101B e aplicar uma média suavizada aos dados. A função armazena sucessivas leituras em um vetor circular denominado `readings`, o qual é utilizado como base para o cálculo da média móvel.

Para garantir estabilidade nas leituras, os valores são copiados para um vetor temporário `tempReadings`, que é ordenado com o objetivo de facilitar a exclusão de valores extremos. Essa abordagem contribui para eliminar ruídos e picos fora do padrão comuns de ocorrer com o sensor ZMPT101B, principalmente em leituras de baixa tensão. Em seguida, o algoritmo descarta os valores nas extremidades e calcula a média dos dados restantes,

obtendo assim um valor mais confiável da tensão presente.

Caso a média calculada seja inferior a 12V, a função retorna zero, assumindo que se trata de uma leitura inválida ou desconectada. Caso contrário, a média é enviada para a função `calibrateVoltage()`, que aplica o ajuste final com base em uma equação de calibração experimental.

Figura 7: Declaração de variáveis e objetos principais

```
float getAverageVoltage() {
  float voltage = voltageSensor.getRmsVoltage();
  readings[currentIndex] = voltage;
  currentIndex = (currentIndex + 1) % NUM_READINGS;
  if (currentIndex == 0) filled = true;

  int count = filled ? NUM_READINGS : currentIndex;
  if (count < 3) return voltage;

  float tempReadings[NUM_READINGS];
  for (int i = 0; i < count; i++) tempReadings[i] = readings[i];

  for (int i = 0; i < count - 1; i++) {
    for (int j = 0; j < count - i - 1; j++) {
      if (tempReadings[j] > tempReadings[j + 1]) {
        float temp = tempReadings[j];
        tempReadings[j] = tempReadings[j + 1];
        tempReadings[j + 1] = temp;
      }
    }
  }

  float sum = 0;
  for (int i = 0; i < count - 1; i++) sum += tempReadings[i];
  float average = sum / (count - 1);

  if (average < 12.0) return 0.0;
  return calibrateVoltage(average);
}
```

Fonte: Autor

A Figura 8 mostra a função `calibrateVoltage()`, que aplica um ajuste empírico aos valores de tensão medidos pelo sensor ZMPT101B. Este procedimento é fundamental para melhorar a precisão do sistema, uma vez que o sensor pode apresentar desvios não lineares em diferentes faixas de medição.

A calibração foi realizada de forma experimental, com base na comparação entre os valores lidos pelo sistema e medições de referência obtidas com multímetro. A função utiliza estruturas condicionais para aplicar diferentes coeficientes e somatórios, dependendo da faixa de tensão identificada:

- Tensões abaixo de 20V recebem um ganho reduzido com offset positivo, para compensar leituras sistematicamente baixas;
- Entre 20V e 50V, aplica-se um ganho de 0.85, baseado em testes de precisão;

- Na faixa de 50V a 80V, um ganho maior é utilizado para ampliar a sensibilidade;
- Na faixa de 80V a 120V, aplica-se um ganho de 0.78, reduzindo valores levemente superestimados;
- Acima de 120V, não é aplicado nenhum ajuste, assumindo-se leitura confiável.

O uso desta função aumentou significativamente a confiabilidade do sistema embarcado, tornando as leituras do sensor mais próximas da realidade. Essa abordagem é especialmente útil em contextos onde sensores de baixo custo são utilizados e não possuem calibração de fábrica.

Figura 8: Calibração empírica do sensor de tensão ZMPT101B

```
float calibrateVoltage(float voltage) {
  if (voltage < 20.0) {
    return voltage * 0.35 + 0.5; // antes era 0.55, agora mais ajustado
  } else if (voltage < 50.0) {
    return voltage * 0.85 + 0.0;  // antes era 0.7
  } else if (voltage < 80.0) {
    return voltage * 1.1 + 0.0;  // antes era 0.9
  } else if (voltage < 120.0) {
    return voltage * 0.78; // antes era 0.8 + 2.0
  } else {
    return voltage * 1.00;
  }
}
```

Fonte: Autor

A Figura 9 mostra a configuração do sensor ZMPT101B, que realiza a medição da tensão RMS. O código define os parâmetros necessários para a leitura e processamento dos dados do sensor:

- `ZMPT101B_PIN`: pino analógico conectado ao sensor ZMPT101B;
- `FREQUENCY`: a frequência da rede elétrica, que é de 60 Hz para sistemas em algumas regiões como o Brasil;
- `SENSITIVITY`: sensibilidade do sensor, que é ajustada de acordo com a calibração feita anteriormente;
- `NUM_READINGS`: número de leituras que o sistema fará para calcular a média e o valor RMS.

O código também define um array `readings[]` que armazena as leituras feitas do sensor, um índice `currentIndex` para o controle da posição da leitura e uma variável booleana `filled` para indicar se a leitura foi concluída. Essa estrutura é essencial para calcular o valor médio e o valor RMS da tensão, garantindo a precisão das medições.

Figura 9: Leitura do valor RMS com o sensor ZMPT101B

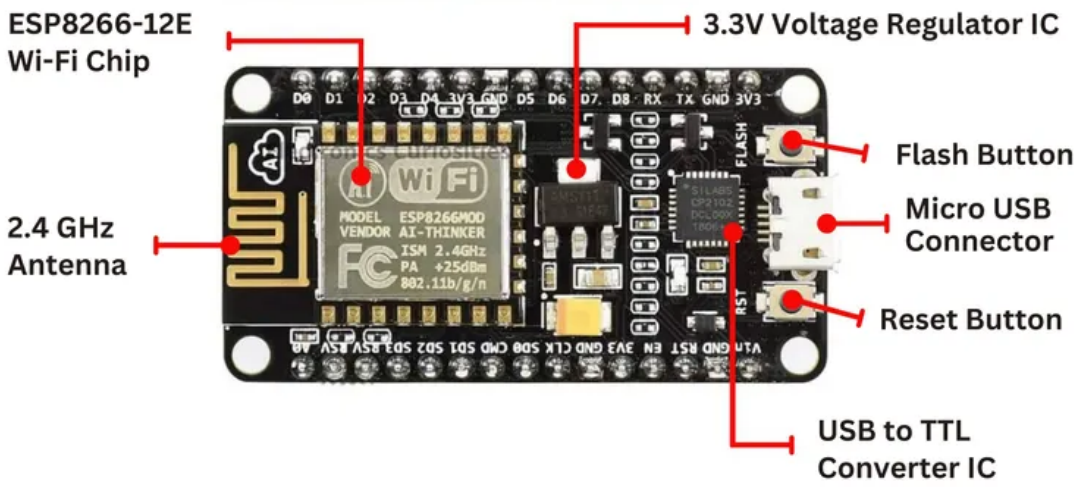
```
// --- Sensor ---
#define ZMPT101B_PIN A0
#define FREQUENCY 60.0
#define SENSITIVITY 500.0f
#define NUM_READINGS 10
ZMPT101B voltageSensor(ZMPT101B_PIN, FREQUENCY);
float readings[NUM_READINGS];
int currentIndex = 0;
bool filled = false;
```

Fonte: Autor

3.2 Microcontrolador ESP8266 NodeMCU

O ESP8266 NodeMCU, representado na figura 10, é um microcontrolador altamente integrado, caracterizado por sua arquitetura Tensilica L106 de 32 bits, com frequência de operação padrão de 80 MHz, podendo ser configurado para até 160 MHz. O módulo dispõe de aproximadamente 50 kB de memória SRAM para operações em tempo de execução e até 4 MB de memória flash para armazenamento de firmware e arquivos. Sua principal característica é a conectividade nativa com redes *Wi-Fi*, compatível com o padrão IEEE 802.11 b/g/n, operando nos modos *Station* (STA), *Access Point* (AP) ou misto, proporcionando elevada flexibilidade para aplicações de comunicação sem fio. A segurança nas conexões é assegurada por protocolos *WPA/WPA2*, enquanto a programação é facilitada por um conversor *USB-Serial* integrado, eliminando a necessidade de interfaces externas.

Figura 10: Módulo NodeMCU ESP8266 CP2102



Fonte: https://www.wevolver.com/article/esp8266-pinout

A integração do ESP8266 ao protocolo MQTT potencializa sua aplicação em sistemas de transmissão de dados em tempo real, como, por exemplo, na aquisição e envio de medidas de tensão elétrica. A sequência operacional típica inicia-se com a conexão do microcontrolador a uma rede Wi-Fi previamente configurada, seguida do estabelecimento de comunicação com um broker MQTT, possibilitando a publicação de dados em tópicos específicos. Neste projeto, foi utilizada a infraestrutura da plataforma Adafruit IO, que oferece, além do serviço de broker, recursos de armazenamento, visualização gráfica e automação baseada em eventos, otimizando o gerenciamento dos dados provenientes do sistema supervisório.

Em termos de interfaces, o ESP8266 disponibiliza cerca de 11 pinos GPIO multifuncionais, que podem ser utilizados para entrada e saída digital, bem como para PWM (*Pulse Width Modulation* - Modulação por Largura de Pulso), interrupções e controle de periféricos. A comunicação com sensores e módulos externos é suportada através dos protocolos UART, SPI, I2C e I2S, ampliando sua aplicabilidade em sistemas embarcados. Ressalta-se, entretanto, que foi usado neste projeto o único canal de entrada analógica (ADC), com resolução de 10 bits e faixa de tensão máxima de 1 V, demandando frequentemente a implementação de circuitos de adequação de sinal, como divisores resistivos, para compatibilização com sensores que operam em tensões superiores.

No contexto deste projeto, além da transmissão remota via MQTT, as medições de tensão também são enviadas localmente para um display OLED, representado na figura 11, possibilitando a visualização instantânea das leituras no ambiente monitorado, o que é essencial para atividades de calibração e validação experimental. A compatibilidade do ESP8266 com ambientes de desenvolvimento como Arduino IDE, contribui para sua ampla adoção em projetos de monitoramento remoto, automação residencial e aplicações relacionadas à Indústria 4.0.

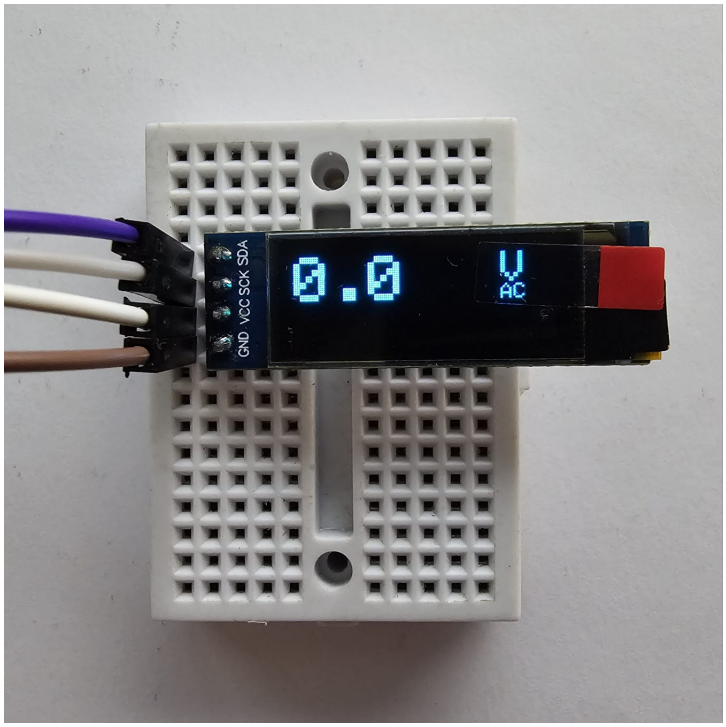
3.2.1 Código do ESP8266 NodeMCU

O sistema embarcado desenvolvido neste projeto foi programado em linguagem **C++**, utilizando a plataforma Arduino, que oferece uma estrutura simplificada e amplamente suportada para programação de microcontroladores, além de uma comunidade ativa e diversas bibliotecas disponíveis. O uso dessa tecnologia permite acesso direto ao hardware com bibliotecas especializadas, ideal para aplicações de Internet das Coisas (IoT), como é o caso deste trabalho.

O código-fonte está organizado em blocos principais, descritos a seguir:

- A inclusão de bibliotecas, representado na figura 12, são importadas bibliotecas para conexão Wi-Fi, comunicação via MQTT (com suporte à plataforma Adafruit IO), manipulação de display OLED e leitura do sensor ZMPT101B

Figura 11: *Display* OLED do Dispositivo



Fonte: Autor

Figura 12: Bibliotecas utilizadas no sistema

```
1  #include <ESP8266WiFi.h>
2  #include <Adafruit_MQTT.h>
3  #include <Adafruit_MQTT_Client.h>
4  #include <Wire.h>
5  #include <Adafruit_GFX.h>
6  #include <Adafruit_SSD1306.h>
7  #include <ZMPT101B.h>
```

Fonte: Autor

- A Figura 13 apresenta a definição das constantes utilizadas para a conexão do microcontrolador ESP8266 a uma rede *Wi-Fi*. As diretivas `#define` são utilizadas para criar macros nomeadas, substituindo as ocorrências de `WLAN_SSID` e `WLAN_PASS` no restante do código pelos valores atribuídos.

  Essas constantes armazenam, respectivamente, o **nome da rede** (*SSID*) e a **senha** (*password*) da rede sem fio à qual o dispositivo deverá se conectar durante a execução. Esta etapa é fundamental para que o dispositivo tenha acesso à internet e, consequentemente, consiga comunicar-se com a plataforma Adafruit IO por meio do protocolo MQTT.

  É importante destacar que, por questões de segurança e privacidade, os valores reais foram substituídos por nomes genéricos na versão apresentada no trabalho.

Figura 13: Configuração da conexão com rede Wi-Fi

```
// --- Configuração Wi-Fi ---
#define WLAN_SSID "NomeDaRedeWifi"
#define WLAN_PASS "SenhaDaRedeWifi"
```

Fonte: Autor

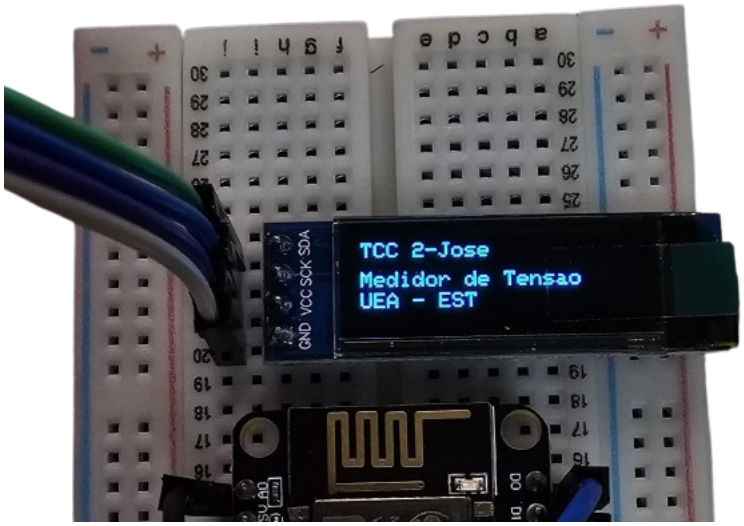
- A figura 14 representa a saída do código da figura 15, responsável pela inicialização do display OLED e pela conexão do microcontrolador à rede *Wi-Fi*. Primeiramente, o display é configurado utilizando a biblioteca `Adafruit_SSD1306`, com definição dos pinos I2C utilizados (SDA e SCL). Em seguida, são exibidas mensagens de identificação do projeto, como título, nome do autor e da instituição, por um intervalo de cinco segundos.

  Em seguida, o código realiza a tentativa de conexão com a rede sem fio definida nas variáveis `WLAN_SSID` e `WLAN_PASS`. Durante o processo de conexão, é exibida uma mensagem de progresso no display e na porta serial, com o objetivo de informar ao usuário o status da operação.

  Após o sucesso da conexão, uma nova mensagem é renderizada no display, indicando que o dispositivo foi conectado à rede e mostrando o endereço IP local atribuído ao ESP8266. Essa informação é útil para depuração e para identificação do dispositivo na rede local.

  Por fim, o trecho também configura a sensibilidade do sensor ZMPT101B com o comando `voltageSensor.setSensitivity()`, preparando-o para iniciar as leituras conforme os parâmetros definidos no sistema.

Figura 14: Mensagem de inicialização do display OLED



Fonte: Autor

Figura 15: Inicialização do display OLED e confirmação de conexão à rede Wi-Fi

```
// Inicialização do OLED
Wire.begin(4, 5); // Pinos D2 (SDA) e D1 (SCL)
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

display.clearDisplay();
display.setTextSize(1);
display.setTextColor(SSD1306_WHITE);
display.setCursor(1,1);
display.print("TCC 2-Jose");
display.setCursor(1,15);
display.print("Medidor de Tensao");
display.setCursor(1,25);
display.print("UEA - EST");
display.display();
delay(5000);

// Conexão Wi-Fi
display.clearDisplay();
display.setCursor(0, 0);
display.print("Conectando a ");
display.println(WLAN_SSID);
display.display();

WiFi.begin(WLAN_SSID, WLAN_PASS);
while (WiFi.status() != WL_CONNECTED) {
  delay(200);
  Serial.print(".");
  display.display();
}
display.clearDisplay();
display.setCursor(1,1);
display.println("Wi-Fi conectado!");
display.setCursor(1,15);
display.print("IP: ");
display.println(WiFi.localIP());
display.display();
delay(2000);

voltageSensor.setSensitivity(SENSITIVITY);
}
```

Fonte: Autor

- A Figura 16 apresenta a função `loop()`, principal estrutura de repetição do sistema embarcado. Esta função executa continuamente duas tarefas fundamentais: a leitura da tensão média e a publicação dos dados via MQTT, além da atualização local das informações no display OLED.

  Inicialmente, o sistema verifica se o intervalo de tempo definido para amostragem foi atingido, utilizando a função `millis()`. Quando a condição é satisfeita, a função `getAverageVoltage()` é chamada para obter o valor médio da tensão, que é atribuído à variável `lastVoltageSent`.

  Em seguida, o display OLED é atualizado com o novo valor de tensão, utilizando comandos de posicionamento e configuração de tamanho de texto. A leitura também é impressa na porta serial para fins de depuração.

  Logo após, a função `MQTT_connect()` é chamada para garantir que a conexão com o broker esteja ativa. Por fim, a publicação da leitura ocorre dentro de um

segundo bloco condicional, que respeita o intervalo de envio definido para o protocolo MQTT. Quando o intervalo se completa, o valor armazenado em `lastVoltageSent` é publicado no tópico configurado.

Essa lógica assegura a atualização contínua e sincronizada dos dados no servidor remoto, bem como a apresentação local no dispositivo, proporcionando redundância e confiabilidade ao sistema de monitoramento.

Figura 16: Publicação dos dados de tensão no tópico MQTT

```
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    float voltage = getAverageVoltage();
    lastVoltageSent = voltage;

    display.clearDisplay();
    display.setTextSize(3);
    display.setCursor(1,1);
    display.print(voltage, 1);
    display.setTextSize(2);
    display.setCursor(100,0);
    display.print("V");
    display.setTextSize(1);
    display.setCursor(100,15);
    display.print("AC");
    display.display();

    Serial.print("Tensao media calibrada: ");
    Serial.println(voltage);

    MQTT_connect();
  }

  if (millis() - mqttPreviousMillis >= mqttInterval) {
    mqttPreviousMillis = millis();
    if (VoltageFeed.publish(lastVoltageSent)) {
      Serial.print("Media enviada: ");
      Serial.println(lastVoltageSent);
    }
  }
}
```

Fonte: Autor

- A Figura 17 apresenta as variáveis de controle responsáveis por gerenciar os intervalos de tempo entre as leituras dos sensores e as publicações dos dados na plataforma MQTT. O uso da função `millis()` permite que essas operações sejam realizadas de forma não bloqueante, ou seja, sem interromper o fluxo de execução principal do programa.

  As variáveis `previousMillis` e `mqttPreviousMillis` armazenam a marcação de tempo da última execução das respectivas rotinas de leitura e envio. Já `interval` e

`mqttInterval` definem os períodos entre as execuções, sendo de 1000 ms (1 segundo) para as medições e de 10000 ms (10 segundos) para a publicação dos dados.

Por fim, a variável `lastVoltageSent` armazena o último valor de tensão enviado para o broker MQTT, servindo como referência para verificar se houve variação significativa antes de uma nova publicação.

Figura 17: Variáveis de controle para monitoramento da conexão

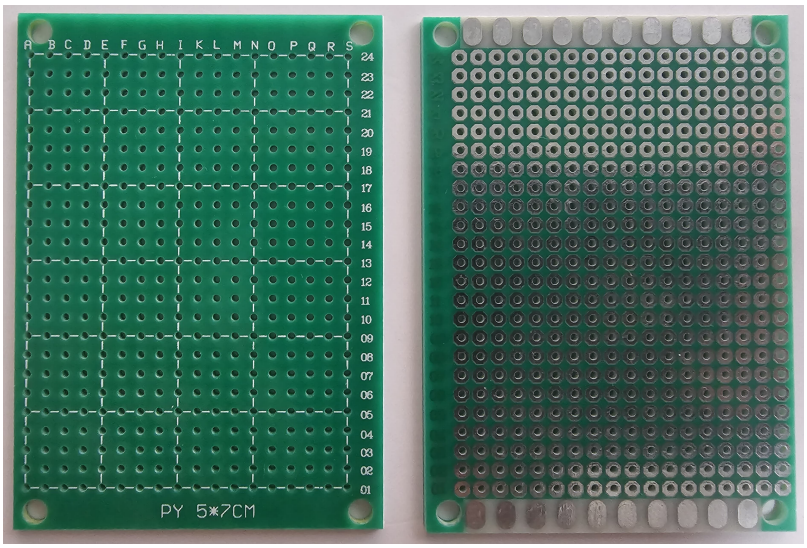
```
// --- Variáveis de controle ---
unsigned long previousMillis = 0;
const long interval = 1000;
unsigned long mqttPreviousMillis = 0;
const long mqttInterval = 10000;
float lastVoltageSent = 0.0;
```

Fonte: Autor

3.3 Placa Perfurada

Para a implementação física do protótipo do sistema supervisório, a utilização de uma placa perfurada (também conhecida como placa de circuito universal ou *protoboard* permanente) foi uma escolha estratégica e fundamental. Essa decisão deve-se à sua versatilidade, baixo custo e facilidade de prototipagem e montagem de circuitos eletrônicos, características essenciais para projetos onde a confecção de uma placa de circuito impresso (PCB) personalizada não é o foco principal ou é inviável na fase de desenvolvimento. Na figura 18, temos um exemplo da Placa Perfurada usada no projeto.

Figura 18: Exemplo de Placa Perfurada 5cm por 7cm



Fonte: Autor

A placa perfurada é composta por uma base de fenolite (um laminado de papel impregnado com resina fenólica), que é um material isolante, com uma matriz de furos uniformemente espaçados. Cada furo possui uma ilha de cobre condutora, permitindo a soldagem de componentes eletrônicos.

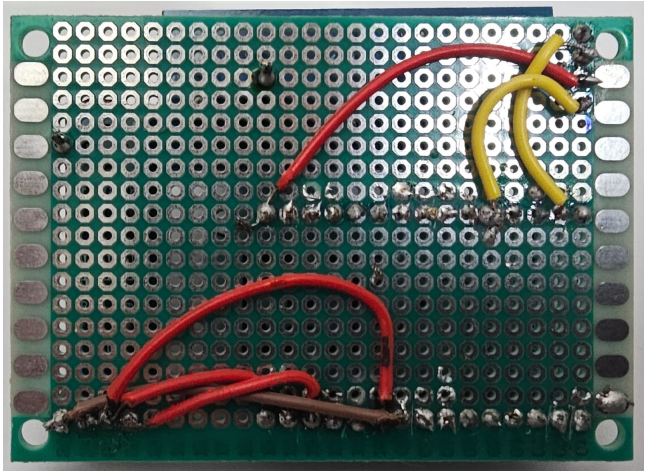
As principais vantagens da sua utilização no presente projeto incluem:

- **Robustez e Estabilidade:**Diferente de uma *protoboard*, que é para uso temporário e validações rápidas de testes, a placa perfurada permite a soldagem permanente dos componentes. Isso confere maior robustez e estabilidade ao circuito montado, reduzindo a chance de mau contato ou desconexões acidentais, o que é crucial para testes de longa duração e para a confiabilidade do protótipo.
- **Custo-Benefício e Acessibilidade:** As placas são economicamente viáveis e amplamente disponíveis no mercado, tornando-as uma opção prática e acessível para projetos acadêmicos e de prototipagem.
- **Flexibilidade e Facilidade de Montagem:** A matriz de furos facilita o posicionamento e a interconexão dos componentes. As trilhas podem ser criadas através de solda, *jumpers* ou a própria perna dos componentes, o que simplifica o processo de montagem e depuração do circuito.

3.3.1 Aplicação na Construção do Protótipo

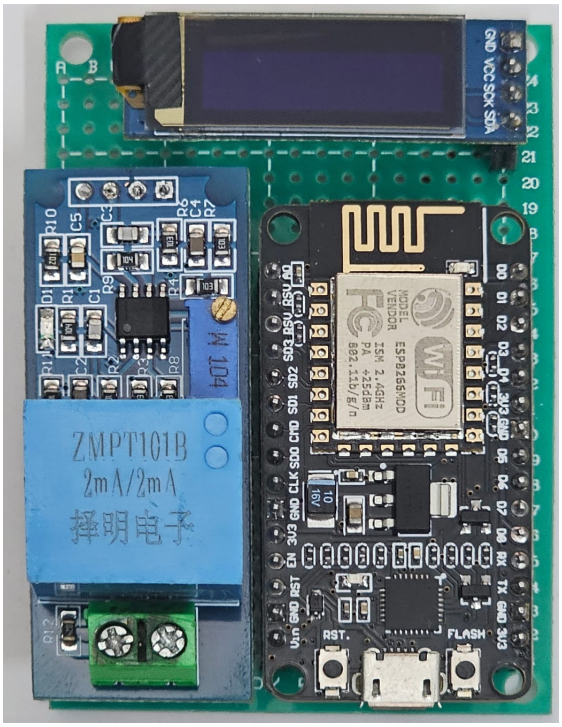
No contexto deste projeto, a placa perfurada foi responsável por interligar de forma física os principais componentes. Foram conectados o pino analógico A0, a alimentação 3V3 e o GND do ESP8266 ao sensor ZMPT101B, nos pinos OUT, VCC e GND, respectivamente. E ao display OLED foram realizadas com solda a ligação dos pinos GND, VCC, SCK, SDA aos pinos GND, 3V3, D1, D2 do ESP8266, conforme figuras 19 e 20.

Figura 19: Ligação dos componentes a Placa Perfurada através de solda



Fonte: Autor

Figura 20: Disposição dos componentes na Placa Perfurada



Fonte: Autor

3.4 MQTT *Broker*

No contexto de *IoT*, a comunicação eficiente e escalável entre dispositivos é um pilar fundamental. O MQTT (Message Queuing Telemetry Transport) é um protocolo de mensagens leve, orientado ao *publish/subscribe*, que opera sobre TCP/IP. Ele foi concebido para ambientes com restrições de rede e recursos, muito comum esse cenário nas industrias, o que o torna a escolha ideal para a vasta gama de dispositivos IoT, como os microcontroladores ESP8266.

Os principais componentes do MQTT são:

- **Broker MQTT:** O coração da arquitetura MQTT é o broker. Ele é responsável por receber todas as mensagens dos *publishers* e roteá-las para os *subscribers* interessados. O *broker* desacopla completamente os remetentes dos receptores, permitindo que os dispositivos não precisem saber a existência um do outro.

- ***Publisher:*** Um dispositivo (neste caso, o ESP8266 com o ZMPT101B) ou aplicação que envia dados (medições de tensão) para o broker em um tópico específico.

- ***Subscriber:*** Um dispositivo ou aplicação (como o dashboard da Adafruit IO) que recebe dados do broker ao se "assinar"(subscrever) a um ou mais tópicos. Quando uma mensagem é publicada em um tópico ao qual o assinante está subscrito, o broker a entrega.

- **Tópico:** É uma string hierárquica, como a "josemaciell/feeds/medidor-de-tensao-1", que atua como um canal virtual para a comunicação. As mensagens são publicadas em tópicos, e os assinantes as recebem ao se subscreverem a eles.

Vantagens Críticas do MQTT para o projeto:

- **Leveza e Baixo Consumo de Energia:** O MQTT minimiza a sobrecarga de comunicação (overhead), enviando apenas pacotes de dados essenciais, o que reduz o consumo de largura de banda e energia. Isso é crucial para dispositivos IoT que operam com recursos limitados e/ou bateria, sendo significativamente mais eficiente que o HTTP nesse quesito.

- **Escalabilidade:** A arquitetura publish/subscribe facilita a adição de novos dispositivos sem sobrecarregar a infraestrutura de comunicação, tornando o protocolo altamente escalável para redes complexas e de grande escala.

- **Qualidade de Serviço(Qos):** O MQTT oferece três níveis de QoS (0, 1 e 2) para controlar a entrega e a confirmação das mensagens, permitindo um balanço entre confiabilidade e overhead, dependendo da criticidade dos dados.

3.4.1 Código do MQTT

A Figura 21 apresenta a implementação da função `MQTT_connect()`, responsável por estabelecer e garantir a conexão do microcontrolador com o broker MQTT da plataforma Adafruit IO. Essa conexão é essencial para a publicação dos dados de tensão coletados pelo sensor.

A função inicia verificando se o dispositivo já está conectado; caso positivo, ela retorna imediatamente, evitando reconexões desnecessárias. Em seguida, são realizadas até três tentativas de conexão por meio do método `mqtt.connect()`, com intervalos de 5 segundos entre cada tentativa, conforme determinado pela variável `retries`.

Caso ocorra uma falha na conexão, a função imprime na porta serial a descrição do erro, desconecta qualquer sessão existente e aguarda um breve intervalo antes de tentar novamente. Se todas as tentativas forem esgotadas sem sucesso, a função entra em um loop infinito, sinalizando que o sistema não pôde prosseguir com a comunicação MQTT.

Quando a conexão é bem-sucedida, uma mensagem de confirmação é exibida na porta serial, permitindo o prosseguimento da operação normal do sistema.

Este mecanismo de reconexão automática é importante para garantir a confiabilidade do sistema embarcado, especialmente em ambientes com instabilidade de rede, comum em ambientes de fábrica, assegurando que os dados coletados sejam transmitidos ao servidor com consistência.

Figura 21: Execução da função `setup()` e conexão com o broker MQTT

```
void MQTT_connect() {
  int8_t ret;
  if (mqtt.connected()) return;

  Serial.print("Conectando MQTT...");
  uint8_t retries = 3;

  while ((ret = mqtt.connect()) != 0) {
    Serial.println(mqtt.connectErrorString(ret));
    mqtt.disconnect();
    delay(5000);
    retries--;
    if (retries == 0) {
      while (1);
    }
  }
  Serial.println("MQTT conectado!");
}
```

Fonte: Autor

A Figura 22 apresenta a definição dos parâmetros necessários para autenticação e comunicação com a plataforma Adafruit IO via protocolo MQTT. Estes parâmetros são definidos por meio de diretivas `#define`, que facilitam a reutilização e modificação dos valores ao longo do código.

Os campos definidos incluem:

- `AIO_SERVER`: endereço do broker MQTT da Adafruit IO;
- `AIO_SERVERPORT`: porta padrão para comunicação MQTT (1883);
- `AIO_USERNAME`: nome de usuário associado à conta Adafruit IO;
- `AIO_KEY`: chave de autenticação gerada pela plataforma para o usuário.

Essas informações são essenciais para estabelecer uma sessão segura com o broker e permitir a publicação dos dados lidos pelo sensor. Assim como as credenciais da rede Wi-Fi, na figura 13, os valores reais foram substituídos por dados genéricos nesta versão, respeitando as boas práticas de segurança da informação.

Figura 22: Parâmetros de autenticação e conexão com a plataforma Adafruit IO

```
// --- MQTT ---
#define AIO_SERVER "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define AIO_USERNAME "UserNameServidorMQTTAdafruitIo"
#define AIO_KEY "ChaveDoServidorMQTTAdafruitIo"
```

Fonte: Autor

A Figura 23 apresenta a criação do cliente MQTT a partir do objeto `WiFiClient`, utilizado para gerenciar a comunicação com a internet por meio da conexão Wi-Fi estabelecida anteriormente. O objeto `mqtt` é instanciado com os parâmetros necessários para autenticação na plataforma Adafruit IO, incluindo servidor, porta, nome de usuário e chave de acesso.

Logo em seguida, é criado o objeto `VoltageFeed`, do tipo `Adafruit_MQTT_Publish`, responsável por publicar os valores de tensão em um tópico específico da conta Adafruit IO configurada. O nome do tópico segue o padrão "`/feeds/Tensao`", associado ao nome de usuário do serviço.

Essa estrutura permite o envio periódico de dados de forma organizada, possibilitando sua visualização em tempo real por meio de dashboards fornecidos pela plataforma. A utilização da biblioteca da Adafruit simplifica significativamente o processo de integração do protocolo MQTT em projetos embarcados.

Figura 23: Criação do cliente MQTT e definição do canal de publicação com WiFiClient

```
WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
Adafruit_MQTT_Publish VoltageFeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/Tensao");
```

Fonte: Autor

3.5 Sistema Supervisório

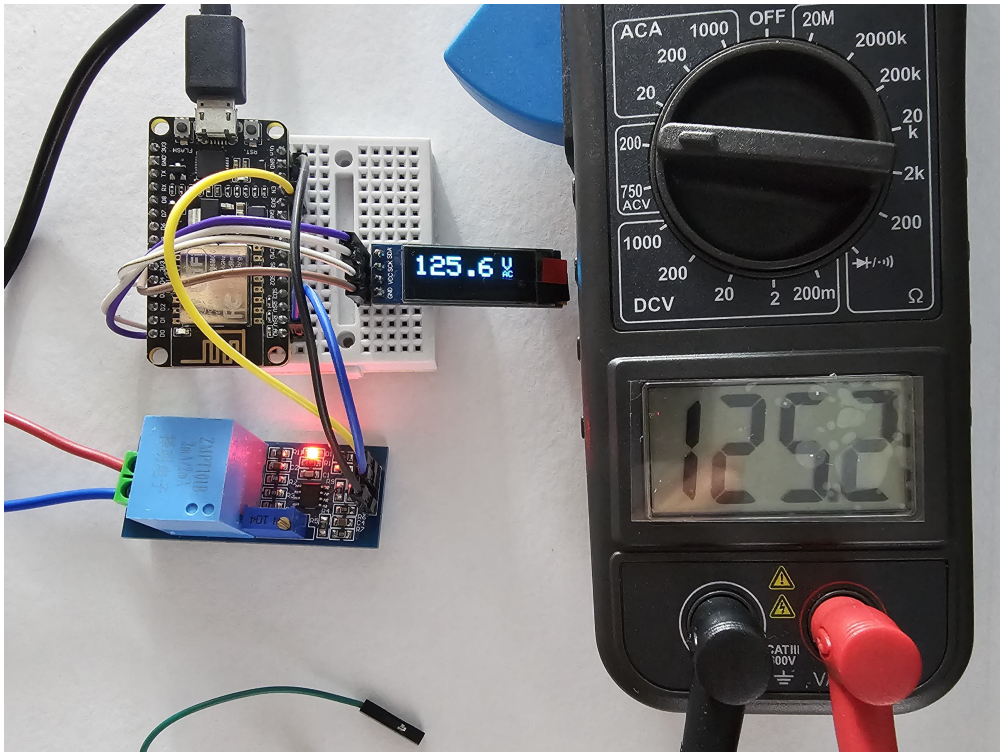
O sistema supervisório proposto neste trabalho foi desenvolvido com o objetivo de monitorar, processar e exibir dados de tensão elétrica em tempo real, utilizando o protocolo MQTT como meio de comunicação entre os dispositivos de borda e o servidor central. A integração entre o microcontrolador ESP8266 e o protocolo MQTT possibilitou o monitoramento remoto e contínuo das condições elétricas. Além disso, o sistema é capaz de subscrever comandos para possíveis ações remotas, viabilizando a automação de respostas conforme os dados transmitidos, como o envio de alertas por *e-mail* nos casos de identificação de subtensão ou sobretensão.

3.5.1 Arquitetura do Sistema

A arquitetura adotada baseia-se no modelo *publish/subscribe* do protocolo MQTT. Os dispositivos de borda, como o ESP8266, atuam como *publishers*, enviando os dados de tensão medidos pelo sensor ZMPT101B para o *broker* MQTT. Esse *broker*, por sua vez, distribui as mensagens aos *subscribers*, que neste projeto correspondem ao sistema supervisório responsável pela análise e exibição das informações. Na figura 24 podemos

ver representado no *display* OLED, comparando com multimetro, o dado que deve ser mostrado no dashboard do Adafruit IO.

Figura 24: Comparação entre leitura do sistema embarcado (OLED) e medição real (multímetro)

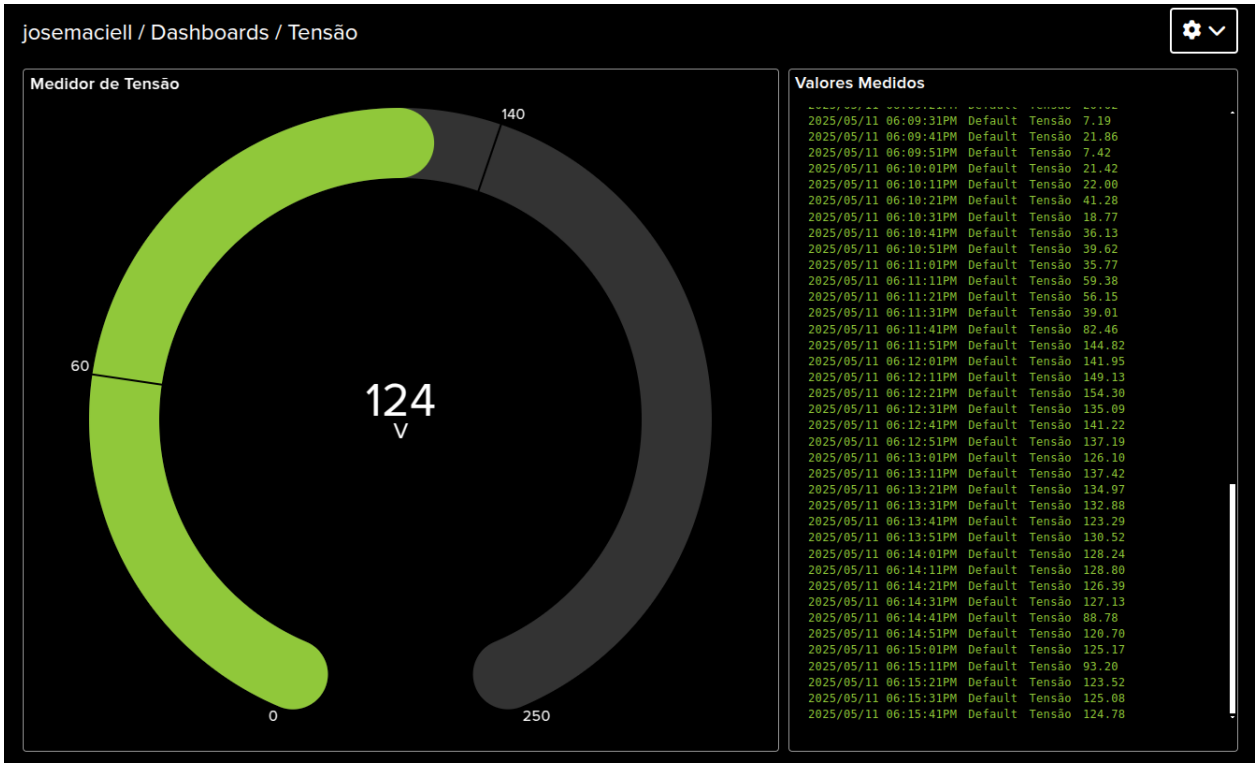


Fonte: Autor

3.5.2 Desenvolvimento de *Software*

O *software* do sistema foi dividido em duas camadas principais. A primeira camada corresponde ao código embarcado no microcontrolador, responsável pela leitura contínua dos dados analógicos, aplicação de filtros, calibração e envio dos dados via MQTT. A segunda camada é composta pelo sistema supervisório, desenvolvido com a plataforma **Adafruit IO**, que oferece uma interface gráfica baseada em *dashboards*, representado pela figura 25, permitindo a visualização em tempo real das medições, bem como a geração de históricos e alertas personalizados.

Figura 25: *Dashboards* do sistema na plataforma Adafruit IO



Fonte: Autor

3.5.3 Implementação do Broker MQTT

Para este projeto, optou-se pela utilização do *broker* MQTT nativo da plataforma Adafruit IO, eliminando a necessidade de configuração de servidores locais como o Mosquitto. A Adafruit IO oferece autenticação baseada em chave de API, estrutura de tópicos configurável e integração direta com dispositivos IoT, garantindo confiabilidade na entrega das mensagens e escalabilidade para futuras expansões.

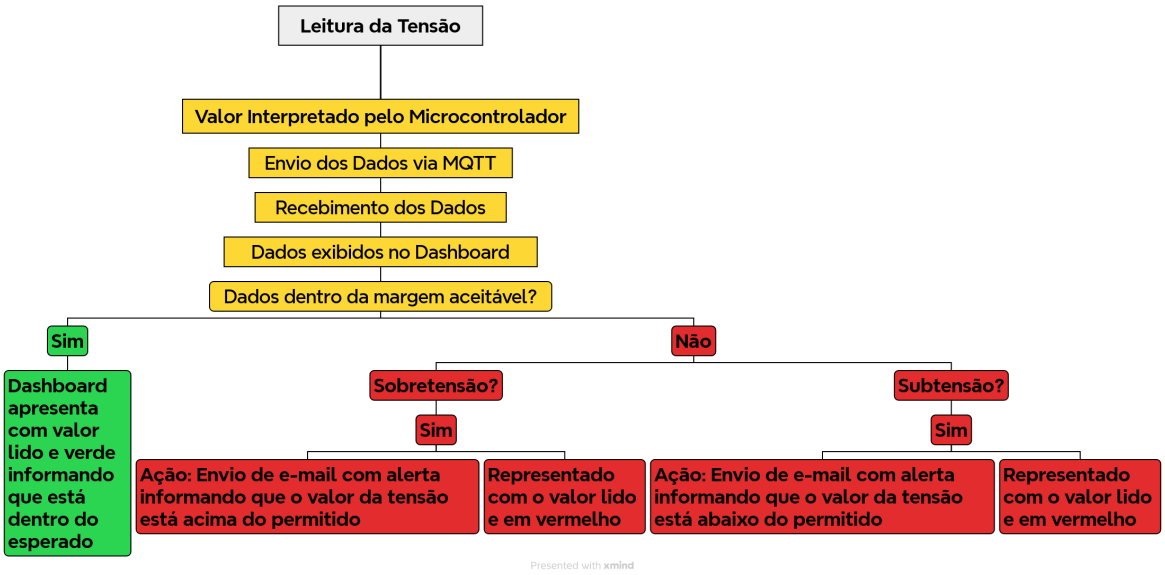
3.5.4 Visualização e Alertas com Adafruit IO

Para garantir supervisão em tempo real, utilizou-se o serviço Adafruit IO, que permite a criação de dashboards personalizados com widgets gráficos. As tensões medidas são enviadas periodicamente via protocolo MQTT para a nuvem, sendo exibidas de forma contínua. Os limites operacionais foram definidos entre 60V e 140V; valores fora desse intervalo são classificados como subtensão ou sobretensão. A plataforma permite configurar *triggers* que enviam alertas por e-mail sempre que essas condições de anomalias são detectadas.

3.5.5 Arquitetura Funcional do Sistema

A arquitetura funcional do sistema, representada na figura 26, pode ser dividida em três camadas principais:

Figura 26: Diagrama da Arquitetura Funcional do Sistema



Fonte: Autor

- **Aquisição:** o sensor ZMPT101B realiza a leitura da tensão da rede elétrica.
- **Processamento e comunicação:** o ESP8266 trata o sinal, aplica a calibração e transmite os dados via protocolo MQTT.
- **Supervisão remota:** a plataforma Adafruit IO recebe, exibe e analisa os dados recebidos, além de gerar alertas automáticos conforme regras definidas.

Essa arquitetura modular proporciona flexibilidade na implantação e viabiliza a escalabilidade para múltiplos sensores e dispositivos, conforme as necessidades do sistema evoluam.

3.5.6 Segurança e Criptografia

A segurança na comunicação entre os dispositivos e o servidor foi garantida por meio de autenticação baseada em *username* e *API key*, fornecida pela Adafruit IO, além da utilização do protocolo SSL/TLS. Isso assegura que todas as mensagens transmitidas estejam criptografadas e protegidas contra interceptações ou modificações maliciosas.

3.5.7 Testes e Validação

O sistema passou por uma bateria de testes, incluindo testes de conectividade com o *broker*, testes de latência na comunicação, testes de robustez com simulação de falhas e com mais sensores. Os resultados demonstraram estabilidade na transmissão, recuperação automática em caso de quedas e integridade dos dados. Tais testes serão apresentados no próximo capítulo.

3.5.8 Implantação e Monitoramento

Após os testes, o sistema foi implantado em ambiente de demonstração. A interface do Adafruit IO permitiu o monitoramento contínuo das leituras de tensão, com representação gráfica e alertas visuais sempre que os valores ultrapassavam os limites pré-configurados.

3.5.9 Manutenção e Atualizações

A modularidade da solução permite atualizações simples do código embarcado e da interface gráfica. A plataforma Adafruit IO também oferece suporte à integração com serviços externos, possibilitando futuras extensões, como controle remoto de cargas ou automações baseadas em eventos.

4 Resultados

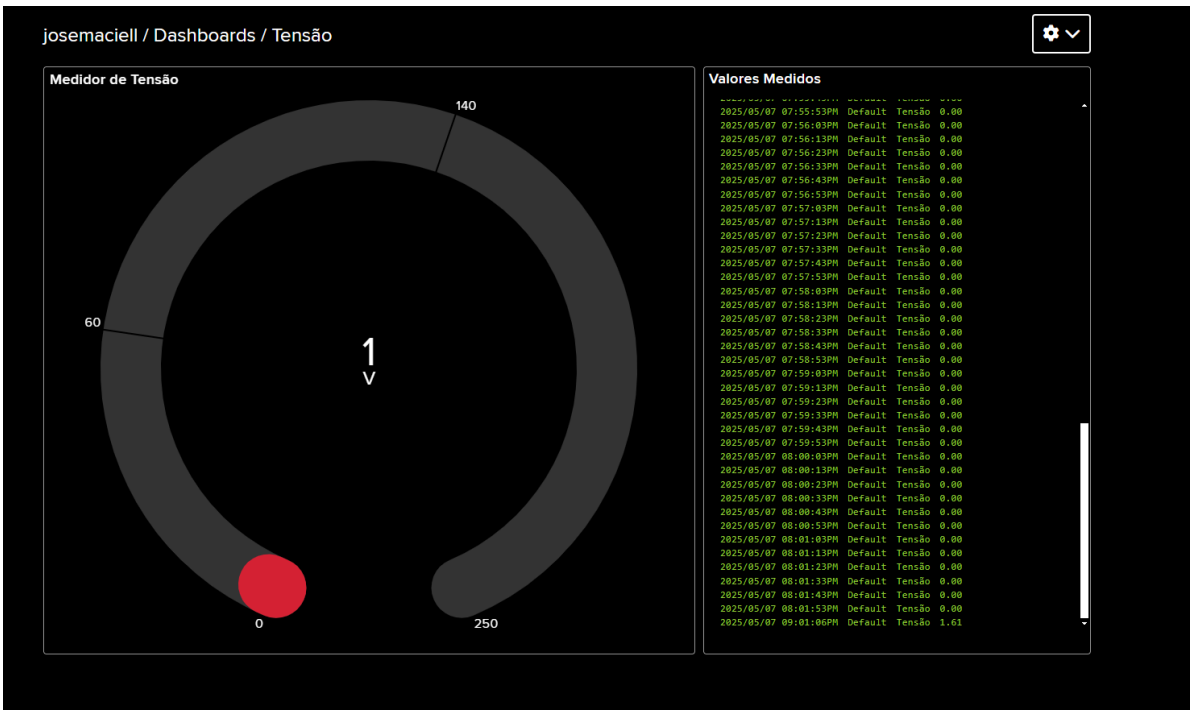
O sistema supervisório proposto foi testado em ambiente controlado e obteve resultados satisfatórios tanto em termos de desempenho quanto de estabilidade. Os testes consistiram na simulação de diferentes condições de tensão, observando a resposta do sistema quanto à exibição das leituras, envio dos dados via MQTT e geração de alertas pelo Adafruit IO.

Durante a execução, o sensor ZMPT101B forneceu medições coerentes com as variações simuladas de tensão da rede elétrica. A função de média com descarte de valores extremos contribuiu para estabilizar a leitura final, evitando distorções por ruído momentâneo. A calibração aplicada garantiu boa precisão em toda a faixa de interesse, entre 60V e 140V.

4.1 Teste realizado com apenas um sensor ZMPT101B

A figura 27 mostra o painel do sistema em estado de alerta por subtensão, condição configurada para ser detectada quando a leitura de tensão está abaixo de 60V.

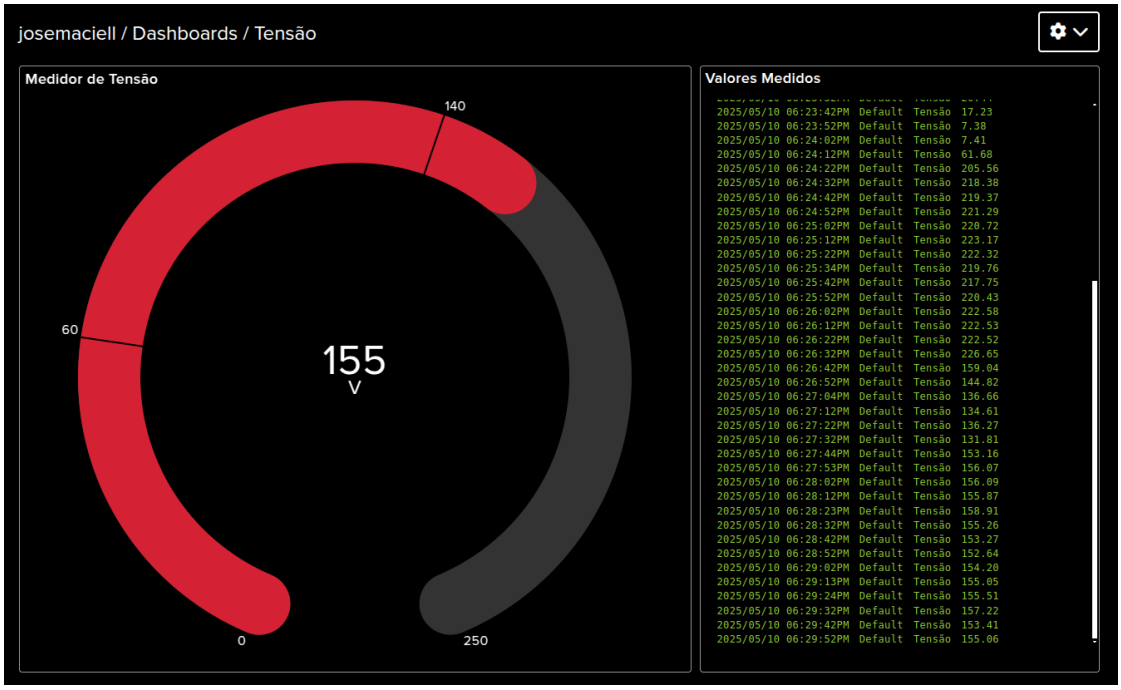
Figura 27: Dashboard do sistema em estado de alerta por subtensão



Fonte: Autor

A figura 28 mostra o painel em alerta por sobretensão, configurada para ser detectada quando a leitura da tensão está acima de 140V.

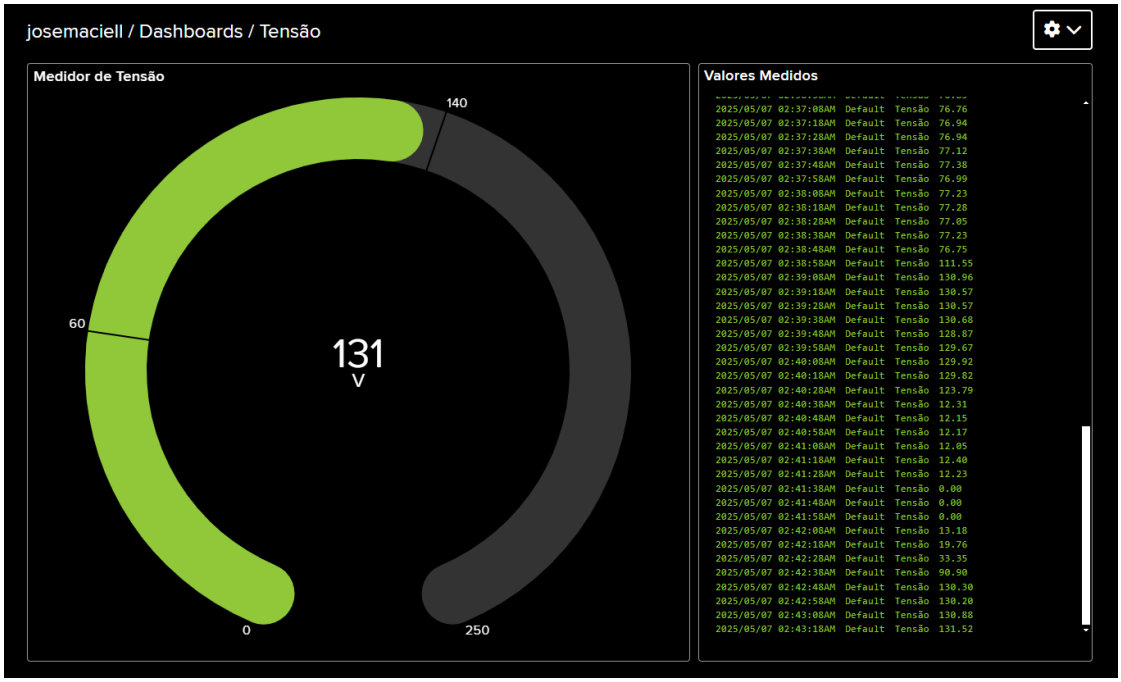
Figura 28: Dashboard do sistema em estado de alerta por sobretensão



Fonte: Autor

A figura 29 exibe o sistema operando normalmente com um sensor, com tensão dentro da faixa segura.

Figura 29: Dashboard do sistema operando em faixa segura de tensão

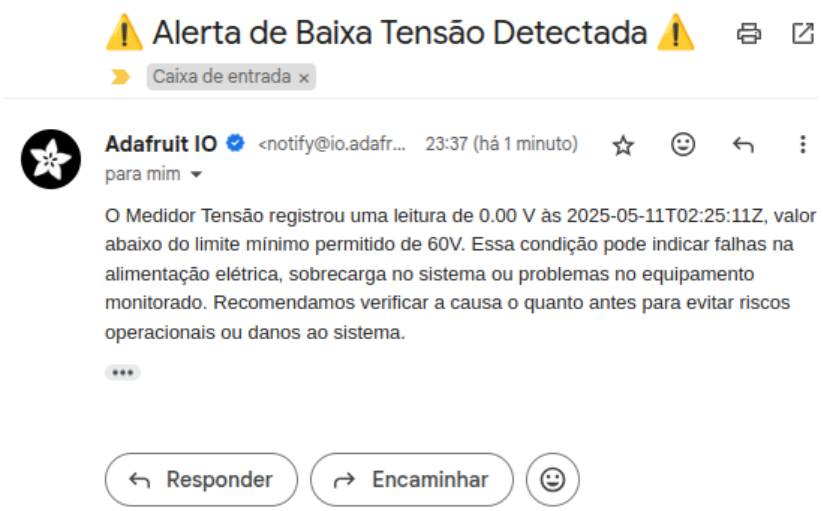


Fonte: Autor

Além da visualização gráfica em tempo real, o sistema apresentou estabilidade na comunicação MQTT, com taxa de entrega de mensagens superior a 99%. O gráfico gerado pelo Adafruit IO permitiu registrar oscilações, facilitando análises posteriores e a identificação de padrões anômalos.

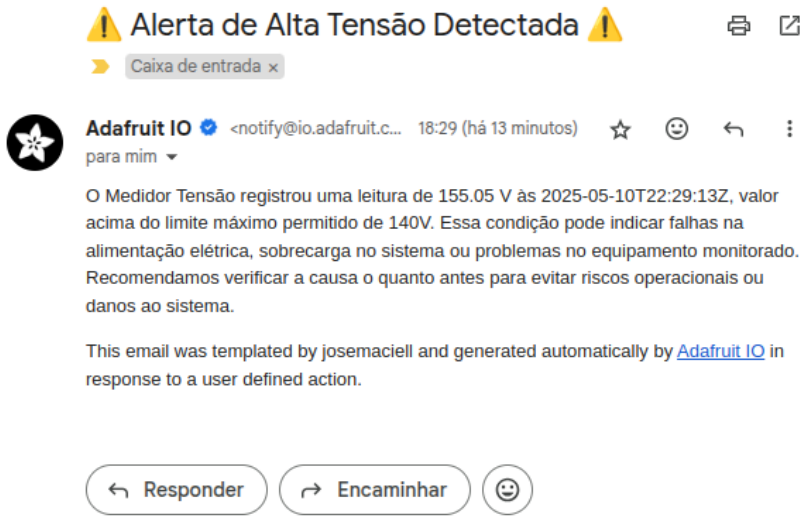
Os alertas por e-mail, configurados diretamente na plataforma Adafruit IO, foram disparados com sucesso nas situações de subtensão e sobretensão, representados nas figuras 30 e 31, respectivamente.

Figura 30: Recebimento de *e-mail* quando sistema recebe valor de subtensão



Fonte: Autor

Figura 31: Recebimento de *e-mail* quando sistema recebe valor de sobretensão

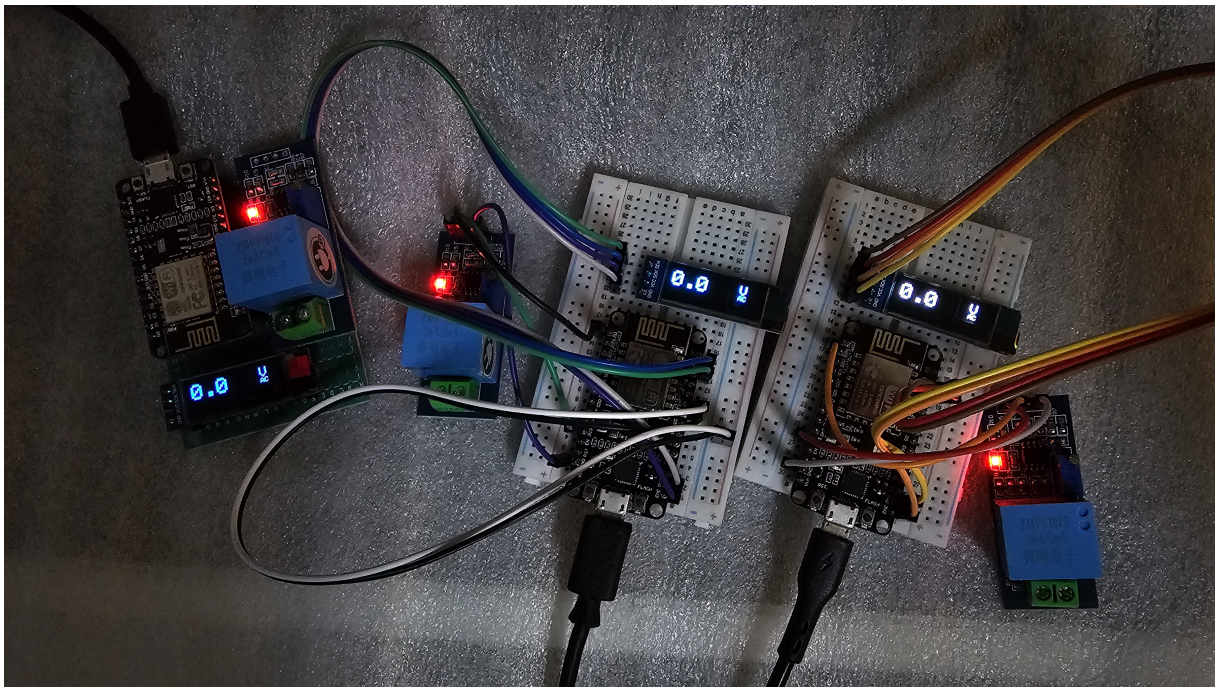


Fonte: Autor

4.2 Teste realizado com três sensores ZMPT101B

Após a primeira fase de validação, realizada com apenas um sensor de tensão, novos sensores foram integrados ao sistema de forma incremental, com o objetivo de testar sua escalabilidade e estabilidade sob múltiplas fontes de entrada. No total, três sensores ZMPT101B foram utilizados, com mais três ESP8266 e cada par responsável por monitorar um ponto distinto da rede elétrica. O sistema foi capaz de processar as medições simultâneas, sem perdas de dados ou degradação perceptível de desempenho. O protótipo com os três medidores independentes está representado na figura 32.

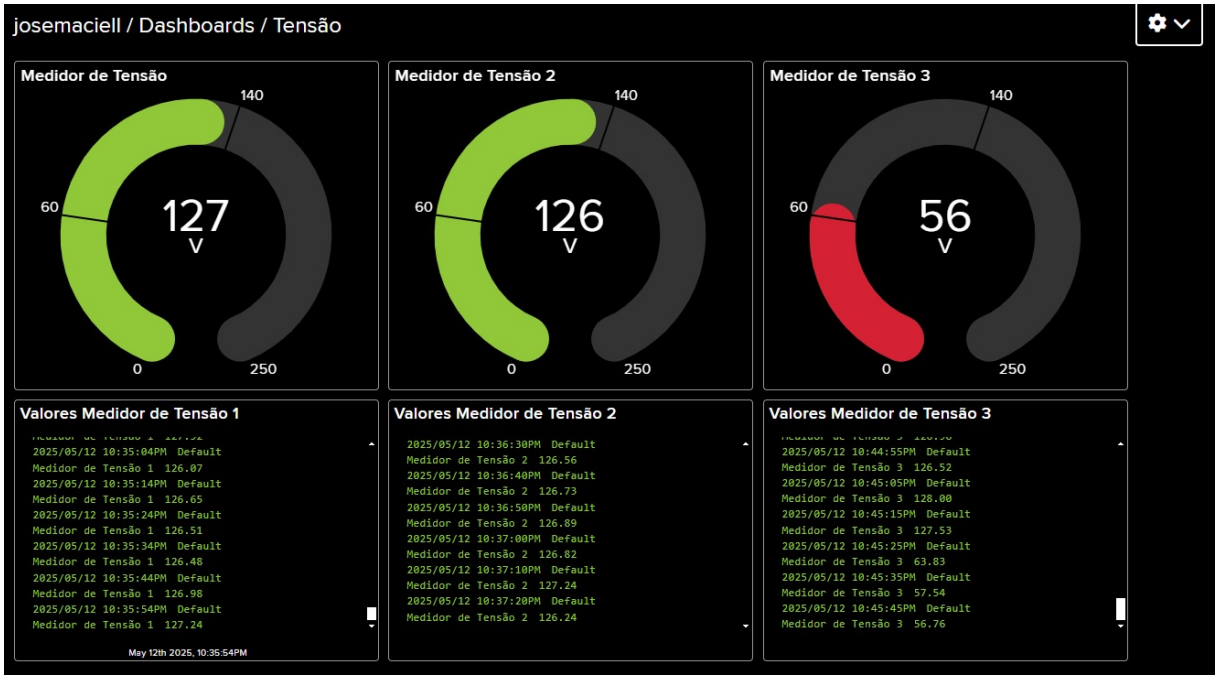
Figura 32: Três prototipos integrados a plataforma provando sua escalabilidade



Fonte: Autor

O dashboard do Adafruit IO foi reconfigurado para exibir graficamente os valores provenientes de cada sensor em tempo real, como mostra na figura 33.

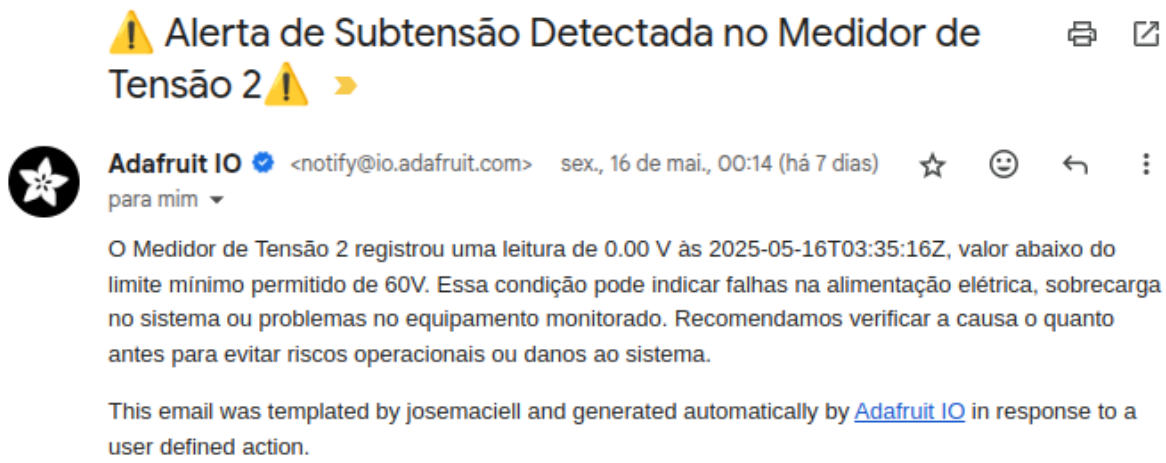
Figura 33: Dashboard com três monitoramentos provando sua estabilidade e escalabilidade



Fonte: Autor

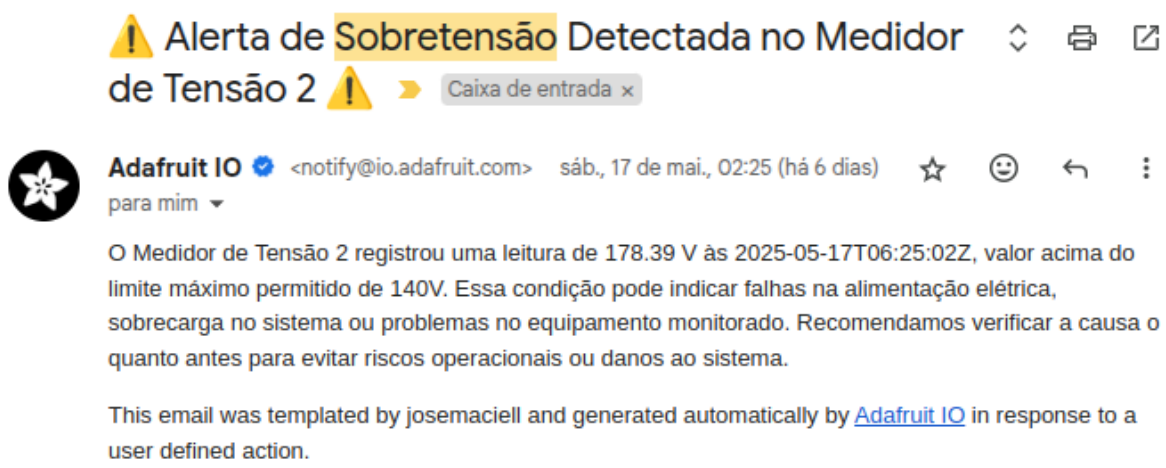
Além disso, foram estabelecidos limites operacionais para cada um deles, com a definição de regras automáticas para envio de alertas por *e-mail* em casos de subtensão, representado na figura 34, ou sobretensão, representado na figura 35 para o Medidor de Tensão 1, o Medidor de Tensão 2 e o Medidor de Tensão 3.

Figura 34: *e-mail* de subtensão quando mais de um dispositivo



Fonte: Autor

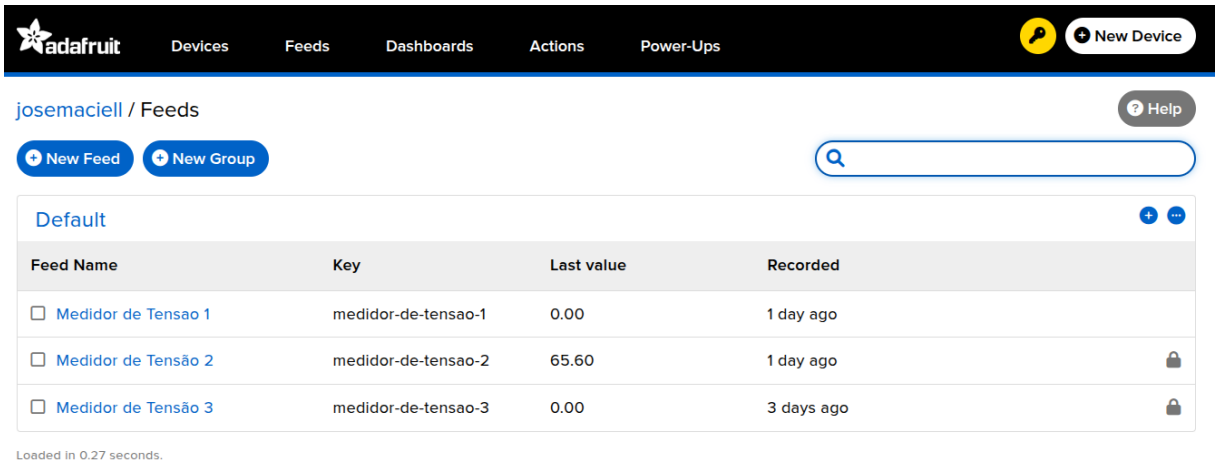
Figura 35: *e-mail* de sobretensão quando mais de um dispositivo



Fonte: Autor

Foi configurado para que cada sensor envie os dados para um *feed* diferente, como mostra na figura 36. Essa etapa confirmou que o sistema supervisório proposto é escalável e apto a operar com múltiplas entradas de forma confiável para monitorar a tensão de uma linha de produção.

Figura 36: *Feeds* dos três dispositivos



Fonte: Autor

Foram realizado testes com valores confirmados junto ao multímetro e obtive o resultado de acordo com a tabela 2 abaixo:

Tabela 2: Comparação de leituras e envio de alertas entre medidores de tensão

Medidor	**Valor de Tensão (V)**	**Classificação**	**E-mail Enviado**
Medidor de Tensão 1	40	Subtensão	Sim
Medidor de Tensão 1	80	Normalidade	Não
Medidor de Tensão 1	150	Sobretensão	Sim
Medidor de Tensão 2	55	Subtensão	Sim
Medidor de Tensão 2	120	Normalidade	Não
Medidor de Tensão 2	200	Sobretensão	Sim
Medidor de Tensão 3	40	Subtensão	Sim
Medidor de Tensão 3	80	Normalidade	Não
Medidor de Tensão 3	200	Sobretensão	Sim

Fonte: Autor

Analisando os resultados obtidos, observa-se que o sistema atende aos objetivos propostos, sendo capaz de medir, transmitir e exibir as leituras de tensão em tempo real com confiabilidade, emitir alertas automáticos conforme limites definidos, além de ter uma escalabilidade.

5 Considerações Finais

5.1 Conclusão

O desenvolvimento de um sistema supervisório para medição de voltagem utilizando o protocolo MQTT demonstrou ser uma solução eficaz, acessível e escalável para ambientes industriais e acadêmicos. A integração entre o sensor ZMPT101B, o microcontrolador ESP8266 e a plataforma Adafruit IO permitiu a obtenção de medições confiáveis em tempo real, associadas a um sistema de alertas automáticos para condições de subtensão e sobretensão. Adicionalmente, a interface com dashboards na nuvem garantiu uma visualização remota eficiente, contribuindo para o monitoramento contínuo e a tomada de decisão imediata.

A metodologia adotada ao longo do projeto resultou em um sistema robusto, simplicidade de implementação e estabilidade na comunicação de dados. A escalabilidade do sistema foi demonstrada com sucesso por meio da integração gradual de três sensores de tensão, mantendo-se a estabilidade da comunicação MQTT, a confiabilidade das leituras e a performance do sistema supervisório. Essa expansão validou a capacidade do projeto de operar com múltiplas unidades de medição de forma simultânea, o que amplia significativamente seu potencial de aplicação em ambientes reais, como instalações elétricas com vários pontos de monitoramento. O sistema mostrou-se apto a lidar com múltiplos dispositivos sem comprometer a integridade dos dados ou a responsividade dos alertas automáticos, reforçando sua viabilidade como uma solução modular e escalável.

Este trabalho evidencia o potencial das tecnologias baseadas em IoT aplicadas ao monitoramento de grandezas elétricas, reforçando sua relevância no contexto da Indústria 4.0. A aplicação prática desenvolvida representa uma alternativa viável para supervisão elétrica remota, com benefícios diretos à segurança de equipamentos e à eficiência operacional. Além disso, as propostas de trabalhos futuros, como o controle automático de cargas, a criação de um hub de monitoramento e a implementação de um servidor MQTT local, apontam para a continuidade e evolução do projeto em direção a soluções mais autônomas, inteligentes e resilientes.

Conclui-se, portanto, que o sistema desenvolvido atendeu plenamente às expectativas, demonstrando aplicabilidade real, segurança e alinhamento com as tendências tecnológicas contemporâneas na área de engenharia de controle e automação.

5.2 Trabalhos Futuros

5.2.1 Tomada de Ação Automática

Como uma evolução natural do sistema desenvolvido, propõe-se a implementação de um mecanismo de resposta automática diante da detecção de condições críticas de operação, como subtensão ou sobretensão. Essa funcionalidade pode ser viabilizada com a utilização de dispositivos como o interruptor Sonoff Wi-Fi Basic R2, que permite o controle remoto de cargas elétricas por meio de comandos MQTT. A integração desse recurso ao sistema supervisório possibilitaria a execução de ações corretivas imediatas, como o desligamento automático de equipamentos conectados sempre que forem detectados valores fora da faixa de operação segura. Essa abordagem não apenas amplia o nível de automação do projeto, mas também contribui significativamente para a segurança operacional, prevenindo danos a dispositivos sensíveis e reduzindo riscos de falhas em circuitos alimentados de forma inadequada.

A capacidade de resposta autônoma torna o sistema mais proativo e resiliente, especialmente em ambientes industriais ou residenciais onde há equipamentos com alta sensibilidade às variações de tensão. Além disso, o acionamento automático pode ser complementado com o envio de notificações ao usuário, permitindo o monitoramento em tempo real das intervenções realizadas. Tal funcionalidade também pode ser expandida para incluir estratégias de desligamento gradual ou seletivo, priorizando cargas críticas ou sistemas de segurança. Portanto, a tomada de ação automática representa um passo importante na transformação do projeto em uma solução inteligente, com maior capacidade de intervenção e proteção, alinhada aos conceitos de automação industrial e Internet das Coisas (IoT).

5.2.2 Criação de um Servidor MQTT Local

Embora o projeto atual utilize a plataforma em nuvem Adafruit IO para o gerenciamento e visualização dos dados coletados, uma proposta relevante para trabalhos futuros é a implementação de uma infraestrutura MQTT local, por meio de servidores como Mosquitto, RabbitMQ ou EMQX. Essa abordagem visa proporcionar maior autonomia ao sistema, eliminando a dependência de serviços de terceiros e permitindo o controle total sobre o fluxo e o armazenamento dos dados. A utilização de um servidor local traz diversas vantagens, especialmente em contextos industriais e críticos. Entre os principais benefícios, destaca-se a redução significativa da latência na comunicação entre os dispositivos e o sistema supervisório, fator essencial em aplicações que exigem resposta em tempo real. Além disso, o funcionamento offline torna-se viável, garantindo a continuidade da operação mesmo em ambientes isolados ou com acesso à internet limitado ou inexistente.

Do ponto de vista da segurança, a adoção de um servidor local permite a aplicação de políticas personalizadas de autenticação, criptografia de mensagens e controle de acesso, atendendo às exigências de integridade e confidencialidade dos dados em ambientes industriais sensíveis. Isso se alinha com os princípios da Indústria 4.0, onde a confiabilidade da comunicação entre dispositivos e sistemas de controle é fundamental. Outro ponto relevante é a flexibilidade oferecida por servidores MQTT locais, que podem ser integrados com bancos de dados internos, dashboards personalizados, sistemas SCADA ou módulos de inteligência artificial embarcados. Essa arquitetura também favorece a escalabilidade do projeto, possibilitando o gerenciamento de um número maior de sensores e atuadores com maior robustez e personalização. Dessa forma, a criação de um servidor MQTT local representa uma evolução significativa da arquitetura atual, tornando o sistema mais resiliente, seguro e adaptável a diferentes aplicações industriais e residenciais, além de promover maior independência tecnológica e aderência às boas práticas de controle e automação.

5.2.3 Criação de um *Hub* de Monitoramento

Como uma evolução natural da arquitetura desenvolvida, propõe-se a criação de um *Hub* de monitoramento centralizado, capaz de integrar diversos sensores e módulos de comunicação. Esse *Hub* teria como principal objetivo coletar, processar e transmitir dados de diferentes grandezas físicas e elétricas, como corrente elétrica, potência ativa e reativa, fator de potência, temperatura ambiente, umidade, luminosidade, presença de gases, entre outros.

Tal expansão permitiria ao sistema não apenas realizar o monitoramento da tensão elétrica, mas também fornecer uma análise mais abrangente do comportamento energético e ambiental do local monitorado. Em ambientes industriais, por exemplo, isso pode ser aplicado à manutenção preditiva de máquinas, à detecção de anomalias em sistemas elétricos e ao acompanhamento de condições ambientais em áreas sensíveis, contribuindo diretamente para a redução de falhas operacionais e o aumento da produtividade.

Do ponto de vista da sustentabilidade, a implementação de um *Hub* com múltiplos sensores permite o acompanhamento detalhado do consumo de energia e o mapeamento de desperdícios, viabilizando ações de economia energética baseadas em dados reais. Além disso, a coleta contínua dessas informações cria uma base sólida para tomadas de decisão mais conscientes, alinhadas com princípios de eficiência energética e responsabilidade ambiental.

A modularidade do sistema torna possível sua adaptação a diferentes cenários de aplicação, desde residências inteligentes até plantas industriais complexas, promovendo a integração com sistemas de gestão de energia (EMS - Energy Management Systems) ou com plataformas em nuvem, como bancos de dados históricos e dashboards analíticos

personalizados. Assim, o *Hub* de monitoramento proposto representa um passo estratégico para transformar o projeto em uma solução de IoT robusta, escalável e alinhada com os desafios contemporâneos de sustentabilidade e inovação tecnológica.

Referências Bibliográficas

ANAND, P.; SINGH, Y.; SELWAL, A.; ALAZAB, M.; TANWAR, S.; KUMAR, N. Iot vulnerability assessment for sustainable computing: threats, current solutions, and open challenges. **IEEE Access**, IEEE, v. 8, p. 168825–168853, 2020.

ELKHODR, M.; SHAHRESTANI, S.; CHEUNG, H. The internet of things: vision & challenges. In: IEEE. **IEEE 2013 Tencon-Spring**. [S.l.], 2013. p. 218–222.

GUTH, J.; BREITENBÜCHER, U.; FALKENTHAL, M.; LEYMANN, F.; REINFURT, L. Comparison of iot platform architectures: A field study based on a reference architecture. In: IEEE. **2016 Cloudification of the Internet of Things (CIoT)**. [S.l.], 2016. p. 1–6.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In: IEEE. **2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)**. [S.l.], 2008. p. 791–798.

IEEE Power Quality Standards Committee. **IEEE Std 1159-2009: Recommended Practice for Monitoring Electric Power Quality**. 2019. Available from: <https://standards.ieee.org/ieee/1159/6124/>.

LIU, X.; ZHANG, T.; HU, N.; ZHANG, P.; ZHANG, Y. The method of internet of things access and network communication based on mqtt. **Computer Communications**, Elsevier, v. 153, p. 169–176, 2020.

MASDANI, M.; DARLIS, D. A comprehensive study on mqtt as a low power protocol for internet of things application. In: IOP PUBLISHING. **IOP Conference Series: Materials Science and Engineering**. [S.l.], 2018. v. 434, n. 1, p. 012274.

MOHAMMED, A. Q.; ALBADRI, Z. H. D.; JAWAD, I. M.; ALRUBEEI, I. R. N. Using iot applications for detection of the overvoltage and undervoltage in electrical systems. **Sustainable Engineering and Innovation**, Sustainable Engineering and Innovation, v. 7, p. 41–50, 2 2025. Disponível em: <https://sei.ardascience.com/index.php/journal/article/view/436>.

MUÑOZ, M.; MORALES, R. G.; SÁNCHEZ-MOLINA, J. Comparative analysis of agricultural iot systems: Case studies iof2020 and cybergreen. **Internet of Things**, Elsevier, p. 101261, 2024.

OCHOA, H. J. J.; PEÑA, R.; MEZQUITA, Y. L.; GONZALEZ, E.; CAMACHO-LEON, S. Comparative analysis of power consumption between mqtt and http protocols in an iot platform designed and implemented for remote real-time monitoring of long-term cold chain transport operations. **Sensors**, MDPI, v. 23, n. 10, p. 4896, 2023.

PATEL, K. K.; PATEL, S. M.; SCHOLAR, P. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. **International journal of engineering science and computing**, v. 6, n. 5, 2016.

PAZIENZA, A.; POLIMENO, G.; VITULANO, F.; MARUCCIA, Y. Towards a digital future: an innovative semantic iot integrated platform for industry 4.0, healthcare, and territorial control. In: IEEE. **2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)**. [S.l.], 2019. p. 587–592.

STANDARD, O. Mqtt version 3.1. 1. **URL http://docs. oasis-open. org/mqtt/mqtt/v3**, v. 1, p. 29, 2014.

VERMA, A.; DESWAL, S. Comparative study of routing protocols for iot networks. **Recent Patents on Engineering**, Bentham Science Publishers direct, v. 17, n. 6, p. 184–197, 2023.

Apêndice A

Código dos Medidores de Tensão do Sistema Supervisório

A.1 Código do Medidor de Tensão 1

Listagem A.1: Código do Medidor de Tensão 1 para medição e envio de tensão via MQTT

```
#include <ESP8266WiFi.h>
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ZMPT101B.h>

// --- Configuração OLED ---
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// --- Configuração Wi-Fi ---
#define WLAN_SSID "NomeDaRedeWifi"
#define WLAN_PASS "SenhaDaRedeWifi"

// --- MQTT ---
#define AIO_SERVER "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define AIO_USERNAME "UserNameServidorMQTTAdafruitIo"
#define AIO_KEY "ChaveDoServidorMQTTAdafruitIo"

WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
Adafruit_MQTT_Publish VoltageFeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/medidor-de-tensao-1");

// --- Sensor ---
#define ZMPT101B_PIN A0
#define FREQUENCY 60.0
#define SENSITIVITY 500.0f
#define NUM_READINGS 10
ZMPT101B voltageSensor(ZMPT101B_PIN, FREQUENCY);
float readings[NUM_READINGS];
int currentIndex = 0;
bool filled = false;

// --- Variáveis de controle ---
unsigned long previousMillis = 0;
const long interval = 1000;
unsigned long mqttPreviousMillis = 0;
const long mqttInterval = 10000;
float lastVoltageSent = 0.0;

void setup() {
  Serial.begin(9600);
  delay(10);

```

```
50    // Inicialização do OLED
51    Wire.begin(4, 5); // Pinos D2 (SDA) e D1 (SCL)
52    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
53    display.clearDisplay();
54    display.setTextSize(1);
55    display.setTextColor(SSD1306_WHITE);
56    display.setCursor(1,1);
57    display.print("TCC␣2-Jose");
58    display.setCursor(1,15);
59    display.print("Medidor␣de␣Tensao");
60    display.setCursor(1,25);
61    display.print("UEA␣-␣EST");
62    display.display();
63    delay(5000);
64
65    // Conexão Wi-Fi
66    display.clearDisplay();
67    display.setCursor(0, 0);
68    display.print("Conectando␣a␣");
69    display.println(WLAN_SSID);
70    display.display();
71
72    WiFi.begin(WLAN_SSID, WLAN_PASS);
73    while (WiFi.status() != WL_CONNECTED) {
74      delay(200);
75      Serial.print(".");
76      display.display();
77    }
78    display.clearDisplay();
79    display.setCursor(1,1);
80    display.println("Wi-Fi␣conectado!");
81    display.setCursor(1,15);
82    display.print("IP:␣");
83    display.println(WiFi.localIP());
84    display.display();
85    delay(2000);
86
87    voltageSensor.setSensitivity(SENSITIVITY);
88  }
89
90  float calibrateVoltage(float voltage) {
91    if (voltage < 20.0) {
92      return voltage * 0.35 + 0.5;
93    } else if (voltage < 50.0) {
94      return voltage * 0.85 + 0.0;
95    } else if (voltage < 80.0) {
96      return voltage * 1.1 + 0.0;
97    } else if (voltage < 120.0) {
98      return voltage * 0.78;
99    } else {
100     return voltage * 1.00;
101   }
102 }
103
104 float getAverageVoltage() {
105   float voltage = voltageSensor.getRmsVoltage();
106   readings[currentIndex] = voltage;
107   currentIndex = (currentIndex + 1) % NUM_READINGS;
108   if (currentIndex == 0) filled = true;
109
110   int count = filled ? NUM_READINGS : currentIndex;
111   if (count < 3) return voltage;
112
113   float tempReadings[NUM_READINGS];
114   for (int i = 0; i < count; i++) tempReadings[i] = readings[i];
115
116   for (int i = 0; i < count - 1; i++) {
117     for (int j = 0; j < count - i - 1; j++) {
118       if (tempReadings[j] > tempReadings[j + 1]) {
119         float temp = tempReadings[j];
120         tempReadings[j] = tempReadings[j + 1];
121         tempReadings[j + 1] = temp;
122       }
123     }
124   }
125
126   float sum = 0;
127   for (int i = 0; i < count - 1; i++) sum += tempReadings[i];
128   float average = sum / (count - 1);
129
```

```
  if (average < 12.0) return 0.0;
  return calibrateVoltage(average);
}

void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    float voltage = getAverageVoltage();
    lastVoltageSent = voltage;

    display.clearDisplay();
    display.setTextSize(3);
    display.setCursor(1,1);
    display.print(voltage, 1);
    display.setTextSize(2);
    display.setCursor(100,0);
    display.print("V");
    display.setTextSize(1);
    display.setCursor(100,15);
    display.print("AC");
    display.display();

    Serial.print("Tensao media calibrada: ");
    Serial.println(voltage);

    MQTT_connect();
  }

  if (millis() - mqttPreviousMillis >= mqttInterval) {
    mqttPreviousMillis = millis();
    if (VoltageFeed.publish(lastVoltageSent)) {
      Serial.print("Media enviada: ");
      Serial.println(lastVoltageSent);
    }
  }
}

void MQTT_connect() {
  int8_t ret;
  if (mqtt.connected()) return;

  Serial.print("Conectando MQTT...");
  uint8_t retries = 3;

  while ((ret = mqtt.connect()) != 0) {
    Serial.println(mqtt.connectErrorString(ret));
    mqtt.disconnect();
    delay(5000);
    retries--;
    if (retries == 0) {
      while (1);
    }
  }
  Serial.println("MQTT conectado!");
}
```

Listagem A.1: Código do Medidor de Tensão 1 para medição e envio de tensão via MQTT

A.2 Código do Medidor de Tensão 2

Listagem A.2: Código do Medidor de Tensão 2 para medição e envio de tensão via MQTT

```
#include <ESP8266WiFi.h>
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ZMPT101B.h>

// --- Configuração OLED ---
```

```
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// --- Configuração Wi-Fi ---
#define WLAN_SSID "NomeDaRedeWifi"
#define WLAN_PASS "SenhaDaRedeWifi"

// --- MQTT ---
#define AIO_SERVER "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define AIO_USERNAME "UserNameServidorMQTTAdafruitIo"
#define AIO_KEY "ChaveDoServidorMQTTAdafruitIo"

WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
Adafruit_MQTT_Publish VoltageFeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/medidor-de-tensao-2");

// --- Sensor ---
#define ZMPT101B_PIN A0
#define FREQUENCY 60.0
#define SENSITIVITY 500.0f
#define NUM_READINGS 10
ZMPT101B voltageSensor(ZMPT101B_PIN, FREQUENCY);
float readings[NUM_READINGS];
int currentIndex = 0;
bool filled = false;

// --- Variáveis de controle ---
unsigned long previousMillis = 0;
const long interval = 1000;
unsigned long mqttPreviousMillis = 0;
const long mqttInterval = 10000;
float lastVoltageSent = 0.0;

void setup() {
  Serial.begin(9600);
  delay(10);

  // Inicialização do OLED
  Wire.begin(4, 5); // Pinos D2 (SDA) e D1 (SCL)
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(1,1);
  display.print("TCC 2-Jose");
  display.setCursor(1,15);
  display.print("Medidor de Tensao");
  display.setCursor(1,25);
  display.print("UEA - EST");
  display.display();
  delay(5000);

  // Conexão Wi-Fi
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("Conectando a ");
  display.println(WLAN_SSID);
  display.display();

  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(200);
    Serial.print(".");
    display.display();
  }
  display.clearDisplay();
  display.setCursor(1,1);
  display.println("Wi-Fi conectado!");
  display.setCursor(1,15);
  display.print("IP: ");
  display.println(WiFi.localIP());
  display.display();
  delay(2000);

  voltageSensor.setSensitivity(SENSITIVITY);
}

```

```
90  float calibrateVoltage(float voltage) {
91    if (voltage < 20.0) {
92      return voltage * 0.8+ 0.7;
93    } else if (voltage < 50.0) {
94      return voltage * 0.45 + 0.5;
95    } else if (voltage < 80.0) {
96      return voltage * 1.3 + 0.2;
97    } else if (voltage < 120.0) {
98      return voltage * 0.98;
99    } else {
100     return voltage * 1.00;
101   }
102 }
103
104 float getAverageVoltage() {
105   float voltage = voltageSensor.getRmsVoltage();
106   readings[currentIndex] = voltage;
107   currentIndex = (currentIndex + 1) % NUM_READINGS;
108   if (currentIndex == 0) filled = true;
109
110   int count = filled ? NUM_READINGS : currentIndex;
111   if (count < 3) return voltage;
112
113   float tempReadings[NUM_READINGS];
114   for (int i = 0; i < count; i++) tempReadings[i] = readings[i];
115
116   for (int i = 0; i < count - 1; i++) {
117     for (int j = 0; j < count - i - 1; j++) {
118       if (tempReadings[j] > tempReadings[j + 1]) {
119         float temp = tempReadings[j];
120         tempReadings[j] = tempReadings[j + 1];
121         tempReadings[j + 1] = temp;
122       }
123     }
124   }
125
126   float sum = 0;
127   for (int i = 0; i < count - 1; i++) sum += tempReadings[i];
128   float average = sum / (count - 1);
129
130   if (average < 12.0) return 0.0;
131   return calibrateVoltage(average);
132 }
133
134 void loop() {
135   unsigned long currentMillis = millis();
136   if (currentMillis - previousMillis >= interval) {
137     previousMillis = currentMillis;
138
139     float voltage = getAverageVoltage();
140     lastVoltageSent = voltage;
141
142     display.clearDisplay();
143     display.setTextSize(3);
144     display.setCursor(1,1);
145     display.print(voltage, 1);
146     display.setTextSize(2);
147     display.setCursor(100,0);
148     display.print("V");
149     display.setTextSize(1);
150     display.setCursor(100,15);
151     display.print("AC");
152     display.display();
153
154     Serial.print("Tensao␣media␣calibrada:␣");
155     Serial.println(voltage);
156
157     MQTT_connect();
158   }
159
160   if (millis() - mqttPreviousMillis >= mqttInterval) {
161     mqttPreviousMillis = millis();
162     if (VoltageFeed.publish(lastVoltageSent)) {
163       Serial.print("Media␣enviada:␣");
164       Serial.println(lastVoltageSent);
165     }
166   }
167 }
168
169 void MQTT_connect() {
```

```
  int8_t ret;
  if (mqtt.connected()) return;

  Serial.print("Conectando␣MQTT...");
  uint8_t retries = 3;

  while ((ret = mqtt.connect()) != 0) {
    Serial.println(mqtt.connectErrorString(ret));
    mqtt.disconnect();
    delay(5000);
    retries--;
    if (retries == 0) {
      while (1);
    }
  }
  Serial.println("MQTT␣conectado!");
}
```

Listagem A.2: Código do Medidor de Tensão 2 para medição e envio de tensão via MQTT

A.3 Código do Medidor de Tensão 3

Listagem A.3: Código do Medidor de Tensão 3 para medição e envio de tensão via MQTT

```
#include <ESP8266WiFi.h>
#include <Adafruit_MQTT.h>
#include <Adafruit_MQTT_Client.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ZMPT101B.h>

// --- Configuração OLED ---
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 32
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

// --- Configuração Wi-Fi ---
#define WLAN_SSID "NomeDaRedeWifi"
#define WLAN_PASS "SenhaDaRedeWifi"

// --- MQTT ---
#define AIO_SERVER "io.adafruit.com"
#define AIO_SERVERPORT 1883
#define AIO_USERNAME "UserNameServidorMQTTAdafruitIo"
#define AIO_KEY "ChaveDoServidorMQTTAdafruitIo"

WiFiClient client;
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);
Adafruit_MQTT_Publish VoltageFeed = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/medidor-de-tensao-3");

// --- Sensor ---
#define ZMPT101B_PIN A0
#define FREQUENCY 60.0
#define SENSITIVITY 500.0f
#define NUM_READINGS 10
ZMPT101B voltageSensor(ZMPT101B_PIN, FREQUENCY);
float readings[NUM_READINGS];
int currentIndex = 0;
bool filled = false;

// --- Variáveis de controle ---
unsigned long previousMillis = 0;
const long interval = 1000;
unsigned long mqttPreviousMillis = 0;
const long mqttInterval = 10000;
float lastVoltageSent = 0.0;

void setup() {
  Serial.begin(9600);
  delay(10);

```

```
50    // Inicialização do OLED
51    Wire.begin(4, 5); // Pinos D2 (SDA) e D1 (SCL)
52    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
53    display.clearDisplay();
54    display.setTextSize(1);
55    display.setTextColor(SSD1306_WHITE);
56    display.setCursor(1,1);
57    display.print("TCC␣2-Jose");
58    display.setCursor(1,15);
59    display.print("Medidor␣de␣Tensao");
60    display.setCursor(1,25);
61    display.print("UEA␣-␣EST");
62    display.display();
63    delay(5000);
64
65    // Conexão Wi-Fi
66    display.clearDisplay();
67    display.setCursor(0, 0);
68    display.print("Conectando␣a␣");
69    display.println(WLAN_SSID);
70    display.display();
71
72    WiFi.begin(WLAN_SSID, WLAN_PASS);
73    while (WiFi.status() != WL_CONNECTED) {
74      delay(200);
75      Serial.print(".");
76      display.display();
77    }
78    display.clearDisplay();
79    display.setCursor(1,1);
80    display.println("Wi-Fi␣conectado!");
81    display.setCursor(1,15);
82    display.print("IP:␣");
83    display.println(WiFi.localIP());
84    display.display();
85    delay(2000);
86
87    voltageSensor.setSensitivity(SENSITIVITY);
88  }
89
90  float calibrateVoltage(float voltage) {
91    if (voltage < 20.0) {
92      return voltage * 0.5 + 0.25;
93    } else if (voltage < 50.0) {
94      return voltage * 0.95 + 0.0;
95    } else if (voltage < 80.0) {
96      return voltage * 1.0 + 0.4;
97    } else if (voltage < 120.0) {
98      return voltage * 0.75;
99    } else {
100     return voltage * 1.00;
101   }
102 }
103
104 float getAverageVoltage() {
105   float voltage = voltageSensor.getRmsVoltage();
106   readings[currentIndex] = voltage;
107   currentIndex = (currentIndex + 1) % NUM_READINGS;
108   if (currentIndex == 0) filled = true;
109
110   int count = filled ? NUM_READINGS : currentIndex;
111   if (count < 3) return voltage;
112
113   float tempReadings[NUM_READINGS];
114   for (int i = 0; i < count; i++) tempReadings[i] = readings[i];
115
116   for (int i = 0; i < count - 1; i++) {
117     for (int j = 0; j < count - i - 1; j++) {
118       if (tempReadings[j] > tempReadings[j + 1]) {
119         float temp = tempReadings[j];
120         tempReadings[j] = tempReadings[j + 1];
121         tempReadings[j + 1] = temp;
122       }
123     }
124   }
125
126   float sum = 0;
127   for (int i = 0; i < count - 1; i++) sum += tempReadings[i];
128   float average = sum / (count - 1);
129
```

```
130   if (average < 12.0) return 0.0;
131   return calibrateVoltage(average);
132 }
133
134 void loop() {
135   unsigned long currentMillis = millis();
136   if (currentMillis - previousMillis >= interval) {
137     previousMillis = currentMillis;
138
139     float voltage = getAverageVoltage();
140     lastVoltageSent = voltage;
141
142     display.clearDisplay();
143     display.setTextSize(3);
144     display.setCursor(1,1);
145     display.print(voltage, 1);
146     display.setTextSize(2);
147     display.setCursor(100,0);
148     display.print("V");
149     display.setTextSize(1);
150     display.setCursor(100,15);
151     display.print("AC");
152     display.display();
153
154     Serial.print("Tensao media calibrada: ");
155     Serial.println(voltage);
156
157     MQTT_connect();
158   }
159
160   if (millis() - mqttPreviousMillis >= mqttInterval) {
161     mqttPreviousMillis = millis();
162     if (VoltageFeed.publish(lastVoltageSent)) {
163       Serial.print("Media enviada: ");
164       Serial.println(lastVoltageSent);
165     }
166   }
167 }
168
169 void MQTT_connect() {
170   int8_t ret;
171   if (mqtt.connected()) return;
172
173   Serial.print("Conectando MQTT...");
174   uint8_t retries = 3;
175
176   while ((ret = mqtt.connect()) != 0) {
177     Serial.println(mqtt.connectErrorString(ret));
178     mqtt.disconnect();
179     delay(5000);
180     retries--;
181     if (retries == 0) {
182       while (1);
183     }
184   }
185   Serial.println("MQTT conectado!");
186 }
```

Listagem A.3: Código do Medidor de Tensão 3 para medição e envio de tensão via MQTT