UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA

VÍNICIUS AUGUSTO DE OLIVEIRA

Desenvolvimento de uma interface de usuário em linguagem Python para monitoramento de grandezas elétricas em tempo real

Uberlândia
2023

VINÍCIUS AUGUSTO DE OLIVEIRA

Desenvolvimento de uma interface de usuário em linguagem Python para monitoramento de grandezas elétricas em tempo real

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Elétrica.

Área de concentração: Distribuição de Energia Elétrica

Orientador: Prof. Dr. José Rubens Macedo Júnior

Uberlândia

2023

VINÍCIUS AUGUSTO DE OLIVEIRA

Desenvolvimento de uma interface de usuário em linguagem Python para monitoramento de grandezas elétricas em tempo real

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Elétrica.

Área de concentração: Distribuição de Energia Elétrica

Uberlândia, 23 de junho de 2023.

Banca Examinadora:

_____

Prof. Dr. José Rubens Macedo Júnior (UFU)

_____

Prof. Dr. Isaque Nogueira Gondim (UFU)

_____

Dra. Ana Camila Ferreira Mamede (UFU)

RESUMO

Este Trabalho de Conclusão de Curso (TCC) abordou o desenvolvimento de uma interface de usuário em linguagem Python para o monitoramento de grandezas elétricas em tempo real. No contexto atual de redes elétricas inteligentes, o monitoramento e a análise de parâmetros da qualidade da energia elétrica são de suma importância. A interface desenvolvida neste trabalho é integrada aos medidores AIW/Sigmasys do Laboratório de Distribuição de Energia Elétrica, permitindo a análise em tempo real de diversas grandezas elétricas, como harmônicos de tensão e corrente, valores de tensão e corrente, geração de energia, flutuação e desequilíbrio de tensão. Esta interface contribui para o monitoramento e a otimização dos sistemas de medição, favorecendo o desenvolvimento de soluções mais eficientes e sustentáveis. As sugestões para trabalhos futuros incluem a implementação de funções de oscilografia de tensão e corrente, a geração de bases de dados personalizadas e a criação de relatórios regulares sobre o sistema de geração.

Palavras-chave: redes elétricas inteligentes, qualidade da energia elétrica, harmônicos, flutuação de tensão, tensão e corrente.

ABSTRACT

This Final Course Project (TCC) addressed the development of a user interface in Python language for real-time electrical quantity monitoring. In the current context of smart electrical grids, the monitoring and analysis of electrical power quality parameters are of utmost importance. The interface developed in this work is integrated with the AIW/Sigmasys meters of the Electric Power Distribution Laboratory, allowing real-time analysis of various electrical quantities, such as voltage and current harmonics, voltage and current values, energy generation, fluctuation and voltage imbalance. This interface contributes to the monitoring and optimization of the measurement systems, favoring the development of more efficient and sustainable solutions. Suggestions for future work include the implementation of voltage and current oscillography functions, the generation of customized databases, and the creation of regular reports on the generation system.

Keywords: Smart grids, power quality, harmonics, voltage fluctuation, voltage and current.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API Application Programming Interface (Interface de Programação de Aplicativos)

HTTP Hypertext Transfer Protocol (Protocolo de Transferência de Hipertexto)

IEC International Electrotechnical Commission (Comissão Eletrotécnica Internacional)

IEEE Institute of Electrical and Electronics Engineers (Instituto de Engenheiros Eletricistas e Eletrônicos)

IDE Integrated Development Environment (Ambiente de Desenvolvimento Integrado)

TCC Trabalho de Conclusão de Curso

TIC Tecnologias da Informação e Comunicação

UFU Universidade Federal de Uberlândia

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização do tema

O ato de monitorar e analisar grandezas elétricas em tempo real é de extrema importância para a preservação da qualidade e a eficácia dos sistemas elétricos. Dentro deste cenário, o objetivo primordial deste Trabalho de Conclusão de Curso reside no desenvolvimento de uma interface de usuário em linguagem Python que permita o monitoramento ininterrupto das referidas grandezas no Laboratório de Distribuição de Energia Elétrica.

Neste contexto laboratorial, a utilização de medidores AIW/Sigmasys, os quais fornecem códigos API para a acessibilidade das informações elétricas, é uma constante. Entretanto, para suprir as demandas singulares oriundas dos projetos de pesquisa em execução, a concepção de uma interface de usuário personalizada por meio da linguagem de programação Python torna-se uma necessidade.

A opção pela linguagem Python como meio para a programação da interface confere benefícios consideráveis para o seu desenvolvimento. Python se caracteriza como uma linguagem contemporânea, versátil e que encontra aplicação em diversos segmentos, incluindo-se aí a Engenharia Elétrica. A natureza intuitiva de sua sintaxe, a diversidade de bibliotecas disponíveis e a facilidade de implementação proporcionam uma construção da interface tanto ágil quanto eficiente, possibilitando a personalização adequada às exigências do ambiente laboratorial.

Por intermédio da interface concebida, a análise de diversas grandezas elétricas em tempo real, tais como harmônicos de tensão e corrente, valores instantâneos de tensão e corrente, variação e desequilíbrio de tensão, tornar-se-á factível. Estas informações serão apresentadas de maneira estruturada e clara, viabilizando a interpretação intuitiva por parte dos usuários [1].

1.2 Objetivo do trabalho

O propósito principal deste Trabalho de Conclusão de Curso consiste na elaboração de uma nova interface de usuário, em linguagem Python, destinada ao monitoramento ininterrupto de grandezas elétricas. Esta interface será integrada aos medidores AIW/Sigmasys presentes no

Laboratório de Distribuição de Energia Elétrica, viabilizando a análise e a visualização de uma diversidade de grandezas elétricas de forma intuitiva e eficaz.

Para o atingimento desse propósito principal, foram delineados os seguintes objetivos específicos:

• Construção de uma interface de usuário em Python: Será executada a programação e a implementação de uma interface de usuário intuitiva por meio da linguagem de programação Python. Esta interface será projetada de maneira a favorecer a interação do usuário com os medidores AIW/Sigmasys e possibilitar a visualização dos dados elétricos em tempo real.

• Facilitação da análise de variadas grandezas elétricas: A interface a ser desenvolvida permitirá a análise de diferentes grandezas elétricas, possibilitando ao usuário selecionar as grandezas de seu interesse e visualizar os dados correspondentes de maneira estruturada e clara.

Ao cumprir tais objetivos, antecipa-se que a nova interface de usuário, baseada na linguagem Python, atuará como uma ferramenta eficiente e de simples manuseio para o monitoramento ininterrupto de grandezas elétricas. Isso resultará na otimização dos sistemas de medição, possibilitando uma análise mais acurada e detalhada das condições elétricas. Ademais, auxiliará na identificação de possíveis problemas e no processo decisório voltado para a melhoria da qualidade da energia elétrica.

1.3 Justificativa e Relevância

A importância do desenvolvimento desta interface de usuário na linguagem Python para o monitoramento contínuo de grandezas elétricas é corroborada por duas justificativas fundamentais.

Em primeira análise, a temática em questão encontra-se alinhada às tendências contemporâneas relativas às redes elétricas inteligentes e à qualidade da energia elétrica. Diante do crescente uso de energias renováveis e da imperiosa necessidade de otimizar a supervisão dos sistemas energéticos, a concepção de soluções eficazes e sustentáveis surge como elemento crucial.

Adicionalmente, uma justificativa relevante reside na oportunidade de aperfeiçoar o monitoramento da qualidade da energia elétrica. Com a identificação precisa de irregularidades e falhas nos sistemas de geração, é possível contribuir para a elevação da eficiência e da confiabilidade das instalações solares.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Cconceitos de qualidade da energia elétrica

A qualidade da energia elétrica reveste-se de inquestionável importância nos sistemas contemporâneos de distribuição de energia, posto que determina o desempenho tanto eficaz quanto eficiente de dispositivos eletroeletrônicos. A confiabilidade e a estabilidade do fornecimento energético são, indubitavelmente, aspectos vitais para assegurar uma operação contínua e segura de sistemas residenciais, comerciais e industriais.

A qualidade da energia pode ser definida como a permanência de parâmetros elétricos dentro de limites aceitáveis, condição que garante o adequado funcionamento dos equipamentos. De acordo com a norma IEEE 1159-1995, a qualidade da energia elétrica é caracterizada como "o planejamento de provimento e qualidade da energia elétrica relacionado à compatibilidade de tensão, corrente ou ondas de frequência que sejam propícias ao correto funcionamento do usuário final" [1].

Os indicadores de qualidade da energia elétrica abarcam elementos como a estabilidade de tensão, a exatidão de frequência, o equilíbrio entre as fases em sistemas polifásicos, a ausência de transientes (como surtos ou quedas de tensão), e a minimização de harmônicos e interrupções.

Harmônicos, por exemplo, representam uma parcela crucial da qualidade energética. Tratam-se de correntes ou tensões que operam em múltiplos da frequência fundamental da rede (que, no Brasil, é de 60 Hz), e podem ser gerados por equipamentos não lineares, como conversores de energia e eletrônicos de potência. Os harmônicos podem provocar uma série de problemas, como o sobreaquecimento de equipamentos, interferência eletromagnética e redução da eficiência energética.

Ademais, os indicadores de qualidade de energia também compreendem a análise da geração de energia, bem como as flutuações e desequilíbrios de tensão. Flutuações de tensão podem ser causadas por uma gama de fatores, incluindo alterações na carga da rede e falhas em equipamentos. Desequilíbrios de tensão, por outro lado, ocorrem quando as tensões ou as correntes nas diferentes fases de um sistema polifásico não são iguais. Ambos podem resultar em impactos adversos significativos na operação dos equipamentos e na eficiência da rede elétrica.

Assegurar a qualidade da energia elétrica implica em monitorar e controlar uma série de parâmetros para manter o adequado funcionamento da rede e dos equipamentos a ela conectados. O objetivo primordial é garantir a confiabilidade, segurança e eficiência dos sistemas de distribuição de energia elétrica, contribuindo para o bem-estar dos usuários e para a sustentabilidade do setor energético [1].

2.2 Redes elétricas inteligentes

As redes elétricas inteligentes, também denominadas Smart Grids, constituem um progresso substancial na esfera de distribuição de energia elétrica. Com a integração da tecnologia da informação e comunicação (TIC), essas redes tornam-se capazes de monitorar, prever, gerenciar e otimizar a produção e o consumo de energia [2].

Diferentemente das redes elétricas convencionais, que operam de forma unidirecional, as redes inteligentes permitem um fluxo bidirecional de energia e informação. Isso implica que tanto fornecedores quanto consumidores podem atuar como geradores de energia, contribuindo para um sistema de energia mais eficiente, confiável e sustentável.

Um dos aspectos mais notáveis das redes elétricas inteligentes é a sua habilidade de realizar monitoramento em tempo real. Isso viabiliza o acompanhamento das condições de rede e a identificação quase que imediata de falhas e ineficiências. Ademais, permite que os consumidores rastreiem e ajustem seu consumo de energia de maneira mais efetiva, ocasionando economias significativas e um menor impacto ambiental.

A incorporação de fontes de energia renováveis também é uma característica fundamental das redes elétricas inteligentes. As energias renováveis, tais como a solar e a eólica, são

intrinsecamente intermitentes. As redes inteligentes, com sua habilidade de monitorar e ajustar o fluxo de energia em tempo real, estão especialmente bem equipadas para gerenciar essa intermitência e garantir que a energia seja sempre distribuída da maneira mais eficiente possível [3].

2.3 Linguagem de programação Python

A linguagem de programação Python, desenvolvida no final da década de 1980 por Guido van Rossum, tem vivenciado um aumento significativo de popularidade nos últimos anos. Ela é reconhecida pela sua sintaxe nítida e legível, projetada para ser de fácil compreensão e escrita [4].

Python é uma linguagem de alto nível, o que implica que é mais abstrata e menos detalhada em comparação às linguagens de baixo nível, como C ou Assembly. Isso faz com que Python seja uma opção atrativa tanto para iniciantes na programação quanto para profissionais que buscam prototipar um novo software ou sistema de maneira rápida.

Ademais, Python é uma linguagem de programação interpretada, o que significa que o código é executado linha a linha, ao invés de ser compilado em um programa executável de uma vez só. Isso proporciona o benefício de permitir que os desenvolvedores testem e corrijam seu código em tempo real, sem a necessidade de passar por um ciclo de compilação demorado.

No âmbito deste trabalho, a linguagem Python representa uma excelente escolha para o desenvolvimento da interface de usuário destinada ao monitoramento de grandezas elétricas em tempo real. Python dispõe de uma série de bibliotecas e frameworks, tais como Tkinter, PyQt e Kivy, adequados para o desenvolvimento de interfaces de usuário. Além disso, Python possui excelentes bibliotecas para lidar com dados em tempo real, como Pandas e NumPy, que podem ser úteis para manipular e analisar as grandezas elétricas coletadas [5].

2.4 API e integração de sistemas

A integração de sistemas é um conceito fundamental na engenharia de software, que se refere ao processo de juntar diferentes componentes de software para que operem como um sistema unificado. Nesse processo, um papel crucial é desempenhado pelas APIs, ou Interfaces de

Programação de Aplicações, que são um conjunto de protocolos e ferramentas para a construção de software e aplicações [6].

As APIs atuam como pontes de comunicação entre diferentes softwares, permitindo que um software utilize funcionalidades de outro sem a necessidade de entender os detalhes da implementação desse outro software. Isso é especialmente útil quando se deseja integrar sistemas complexos, como os equipamentos de monitoramento de grandezas elétricas.

No caso dos equipamentos AIW/Sigmasys mencionados na introdução deste trabalho, uma API pode fornecer um meio eficiente de coletar e analisar os dados de grandezas elétricas em tempo real. O software que implementa essa API seria responsável por coletar os dados dos equipamentos, processá-los conforme necessário, e então disponibilizá-los para a interface de usuário que estamos desenvolvendo [7].

Uma característica notável das APIs é que elas permitem a integração de sistemas de maneira modular. Isso significa que cada componente do sistema pode ser desenvolvido e testado independentemente dos outros, desde que cada componente siga as especificações da API. Esta modularidade facilita o desenvolvimento e a manutenção do sistema como um todo [8].

Outro benefício importante das APIs é que elas podem promover a interoperabilidade entre diferentes sistemas. Por exemplo, se no futuro o laboratório de distribuição de energia elétrica decidir usar um tipo diferente de equipamento de monitoramento, a interface de usuário que estamos desenvolvendo poderia, teoricamente, continuar a operar com poucas ou nenhuma alteração, desde que o novo equipamento implemente a mesma API [9].

3. METODOLOGIA

3.1 Descrição do equipamento AIW/Sigmasys

O medidor AIW/Sigmasys, como evidenciado na Figura 1, é um componente essencial no monitoramento da qualidade de energia no Laboratório de Distribuição de Energia Elétrica. Desenvolvido por engenheiros especializados, este equipamento atende aos requisitos de um medidor Classe A, conforme a norma internacional IEC 61000-4-30. Desde 2019, o

AIW/Sigmasys recebeu a homologação do Operador Nacional do Sistema Elétrico (ONS) para ser usado em campanhas de medição de harmônicos, sinalizando sua robustez e confiabilidade.

Figura 1 – Medidor AIW/Sigmasys

Fonte: Autor.

Em termos de capacidade de medição, o medidor AIW/Sigmasys pode operar em uma faixa de tensão de 20 a 600 VAC, e medir correntes de 0,1 a 5000 A, variando conforme o tipo de sensor de corrente empregado. O equipamento também apresenta um sistema de armazenamento interno de 16 GB, podendo ser expandido até 2000 GB com o uso de memória externa.

No que diz respeito à precisão, o AIW/Sigmasys mostra-se notável. A precisão de medição de tensão é de ± 0,1%, enquanto a de corrente é de ± 0,1% do fundo de escala, e a frequência é medida com uma precisão de ± 0,01 Hz. Em relação às distorções harmônicas, flutuações de tensão e desequilíbrio de tensão, o medidor apresenta precisão de ± 5,0%, ± 5,0% e ± 0,15%, respectivamente. Esta alta precisão permite que o AIW/Sigmasys seja um recurso valioso no diagnóstico e monitoramento de qualidade de energia [8].

3.2 Descrição do sistema de geração de energia fotovoltaica

O Laboratório de Distribuição de Energia Elétrica da Universidade Federal de Uberlândia (UFU) possui um sistema de geração de energia fotovoltaica para contribuir com os estudos e projetos de pesquisa em desenvolvimento. Este sistema é localizado nas instalações da própria universidade, posicionado em uma área próxima ao laboratório, o que permite um fácil acesso e monitoramento contínuo.

O sistema é composto por 16 módulos fotovoltaicos, distribuídos em duas strings. Cada módulo tem uma potência máxima de geração de 330W. Isso resulta em uma capacidade máxima de geração de energia de 5.28 kW (16 módulos * 330 W), representando uma fonte significativa de energia renovável. A Tabela 1, oferece um resumo detalhado das características e especificações desses módulos.

Tabela 1 - Dados dos Módulos Usados

Modelo	DHP72-330W
Potência Máxima (Pmax)	330 W
Tensão de Circuito Aberto (Voc)	46.1 V
Tensão Máxima de Operação (Vmp)	37.3 V
Corrente de Curto Circuito (Isc)	9.29 A
Corrente Máxima de Operação (Imp)	8.85 A
Eficiência	17.02%

Fonte: Ecori Solar [9]

Como ilustrado na Figura 2, o medidor AIW/Sigmasys é integrado ao sistema para monitorar as grandezas elétricas em tempo real. Esta configuração permite a coleta de dados precisos e contínuos sobre o desempenho do sistema fotovoltaico, incluindo a geração de energia, a qualidade da energia elétrica, a flutuação de tensão e o desequilíbrio de tensão, entre outros.

Figura 2 – Medidor Acoplado ao Inversor

Fonte: Autor.

3.3 Análise das grandezas elétricas a serem monitoradas

Para avaliar efetivamente o desempenho e a eficiência de um sistema de geração de energia elétrica, como um sistema de geração fotovoltaica, várias grandezas elétricas precisam ser monitoradas constantemente. Essas grandezas estão diretamente relacionadas à qualidade da energia gerada e ao desempenho do sistema como um todo [10].

Os harmônicos de tensão e corrente são grandezas elétricas fundamentais no estudo da qualidade da energia. Segundo Barbosa (2008), "harmônicos são componentes de uma forma de onda periódica cujas frequências são múltiplos inteiros da frequência da componente de maior amplitude" [11]. Essas distorções na forma de onda podem gerar uma série de problemas, como a redução da vida útil dos equipamentos, aumento no consumo de energia e falhas em sistemas eletrônicos sensíveis.

Outras grandezas importantes a serem monitoradas são os valores de tensão e corrente em tempo real. Esses dados são vitais para avaliar a carga atual do sistema e para detectar quaisquer

anomalias, como sobrecargas ou quedas repentinas. Eles permitem que o usuário acompanhe de perto o desempenho do sistema e tome ações corretivas quando necessário [12].

A geração de energia é outra grandeza que precisa ser monitorada, pois indica a quantidade de energia que o sistema está produzindo em um dado momento. Este é um indicador importante da eficiência do sistema e pode ajudar a identificar problemas, como subutilização ou superutilização da capacidade de geração [13].

Por último, mas não menos importante, as flutuações e desequilíbrios de tensão são grandezas críticas no monitoramento de sistemas de energia. A flutuação de tensão pode causar problemas como piscamento de luzes, enquanto um desequilíbrio de tensão pode levar a um uso ineficiente de energia e, em casos extremos, danos aos equipamentos [14].

3.4 Desenvolvimento da interface de usuário em Python

O desenvolvimento da interface de usuário foi realizado utilizando a linguagem de programação Python, escolhida por sua versatilidade e fácil manuseio. Para isso, o aluno utilizou a IDE Pycharm, com a qual já possuía familiaridade, principalmente para manipulação de dados [15].

A primeira etapa do desenvolvimento consistiu na estruturação da interface gráfica. Para este propósito, utilizou-se a biblioteca Tkinter [16], que oferece ferramentas poderosas para a criação de interfaces de usuário. Em seguida, foram utilizadas as bibliotecas Numpy e Pandas para o tratamento e manipulação de dados. O Numpy é uma biblioteca essencial para a computação científica em Python, oferecendo um objeto de array multidimensional e ferramentas para trabalhar com esses arrays. A Pandas, por sua vez, proporciona estruturas de dados de alta performance e fáceis de usar, além de possuir funções para a análise de dados [17].

A etapa seguinte envolveu a integração dos dados com a interface gráfica. Através da biblioteca Matplotlib, foi possível criar gráficos interativos que exibem as grandezas elétricas em tempo real. A biblioteca Matplotlib é uma das principais ferramentas em Python para a criação de gráficos 2D e 3D [18]. No que diz respeito à atualização dos dados em tempo real, utilizou-se a biblioteca Time e o módulo animation da Matplotlib. A função sleep do Time foi usada para

atualizar os dados a cada 0.5 segundos. O módulo animation da Matplotlib, por outro lado, permitiu a criação de gráficos animados que atualizam seu conteúdo ao longo do tempo.

A biblioteca Requests foi empregada para facilitar o envio de solicitações HTTP. Ela foi essencial para a obtenção dos dados dos medidores do laboratório, que são acessíveis através de uma API [19]. Através da Requests, foi possível enviar solicitações para essa API e receber as respostas em formato JSON, que foram então processadas e exibidas na interface.

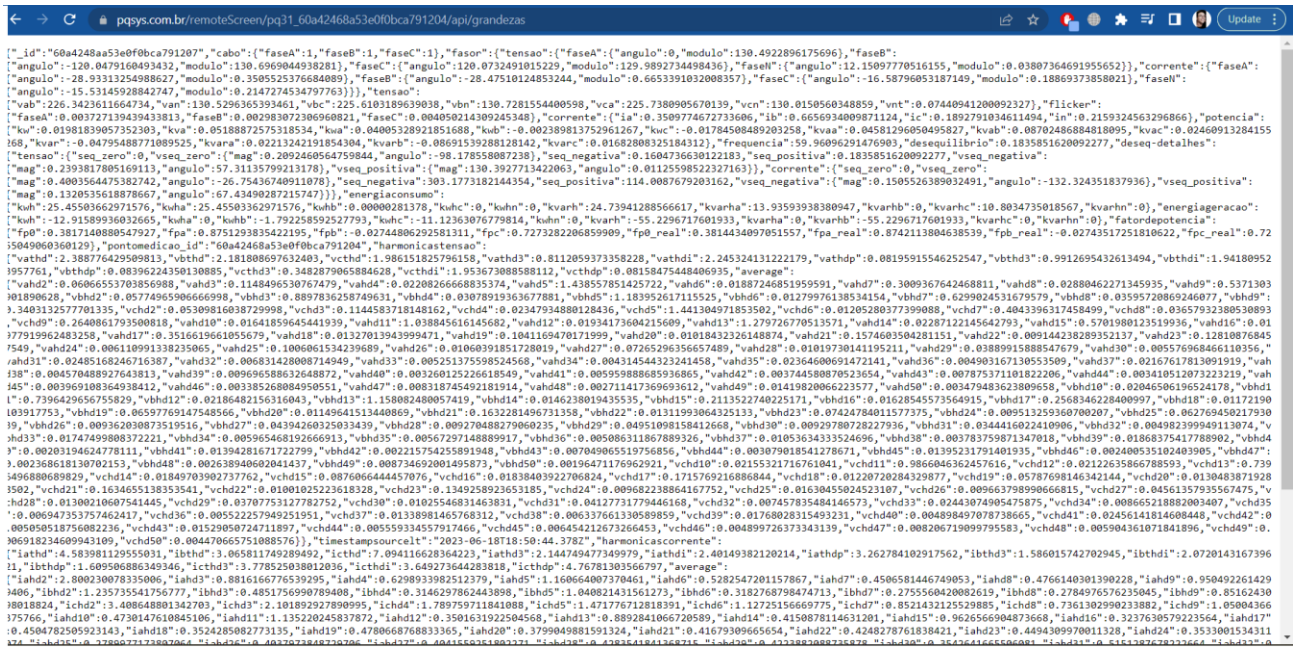
Por fim, o código foi escrito e testado iterativamente. Isso significa que pequenas partes do código foram escritas e testadas antes de avançar para a próxima fase do desenvolvimento. Isso foi feito para garantir que cada parte do código estivesse funcionando corretamente e para identificar e corrigir erros mais cedo no processo de desenvolvimento. O resultado foi um aplicativo robusto e eficiente, capaz de monitorar uma série de grandezas elétricas em tempo real e apresentar os resultados de uma maneira clara e fácil de entender [20].

3.5 Testes e validação da interface

O processo de validação de dados é um passo crucial para garantir a funcionalidade e a precisão do sistema implementado. No caso do projeto em questão, a validação foi realizada de maneira manual, através da verificação direta dos dados gerados pela API, acessível via navegador [21].

Para efetuar esta validação, inseriu-se o link da API no navegador. Isso levou a uma tela que exibia dados em formato de lista, mostrados na Figura 3. Dessa forma, foi possível verificar em tempo real os valores das grandezas elétricas conforme eram medidos pelo dispositivo. Através dessa análise, foi possível confirmar se os valores exibidos no gráfico e nas tabelas da interface desenvolvida correspondiam àqueles apresentados na API [22].

Figura 3 – API Acessada pelo Navegador



Fonte: Autor.

Vale destacar que o processo de validação também incluiu a verificação das atualizações em tempo real dos dados. O intuito foi garantir que o sistema implementado estava efetivamente rastreando as mudanças nas grandezas elétricas à medida que eram medidas pelo equipamento. Nesse sentido, foram realizadas diversas checagens aleatórias para conferir a acurácia das informações.

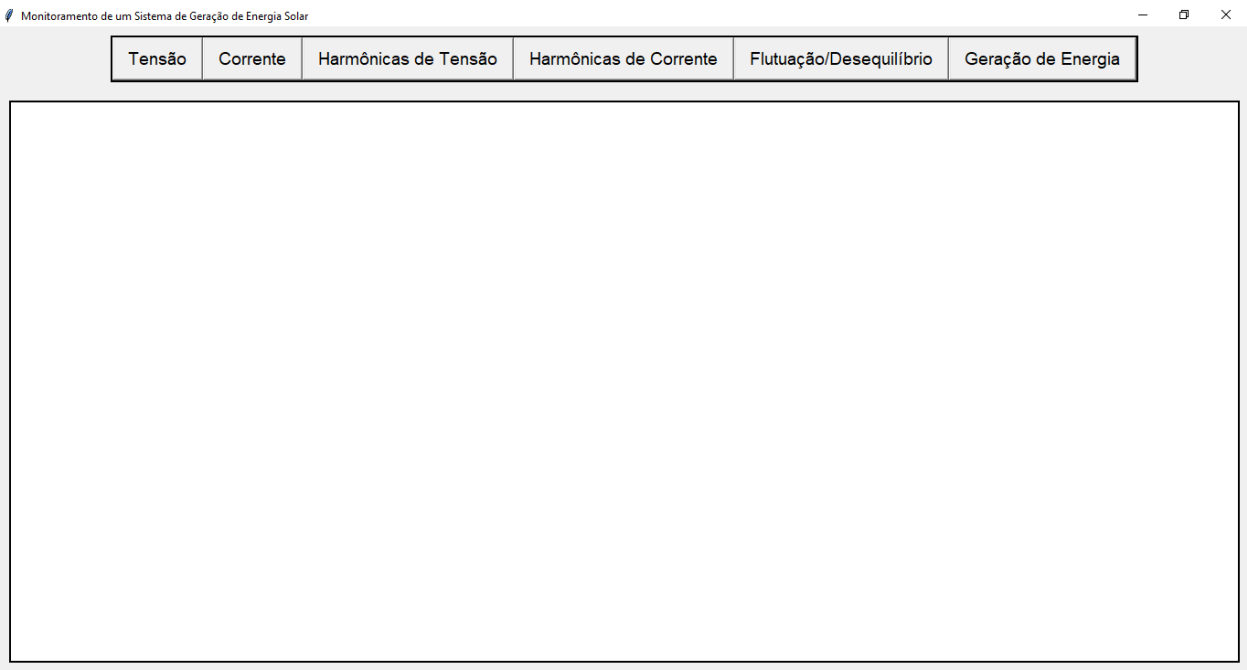
Durante o processo de validação, foram registradas todas as inconsistências ou falhas identificadas. Essas informações foram posteriormente utilizadas para aprimorar o sistema e corrigir eventuais erros de programação.

4. RESULTADOS

4.1 Descrição da interface desenvolvida

A interface de usuário desenvolvida em Python para o monitoramento de grandezas elétricas em tempo real, adota um design minimalista e funcional. Baseada em uma janela única de cor branca, a interface prima pela simplicidade e facilidade de uso, como visto na Figura 4.

Figura 4 – Página Inicial da Interface



Fonte: Autor.

Dentro da janela principal da interface, há vários botões, cada um representando uma função específica relacionada às grandezas elétricas a serem monitoradas. Os botões foram desenhados para serem intuitivos, permitindo que os usuários naveguem de forma direta e rápida pelas funcionalidades da interface.

Ao clicar em um botão, o usuário é levado à função correspondente, que inclui a lógica de extração, organização e exibição de dados. A navegação dentro da interface é totalmente livre, o que permite que os usuários alternem facilmente entre diferentes grandezas elétricas a serem monitoradas. Isso confere à interface uma dinâmica ágil e fluída, otimizando a experiência do usuário e possibilitando o acesso a dados críticos com poucos cliques.

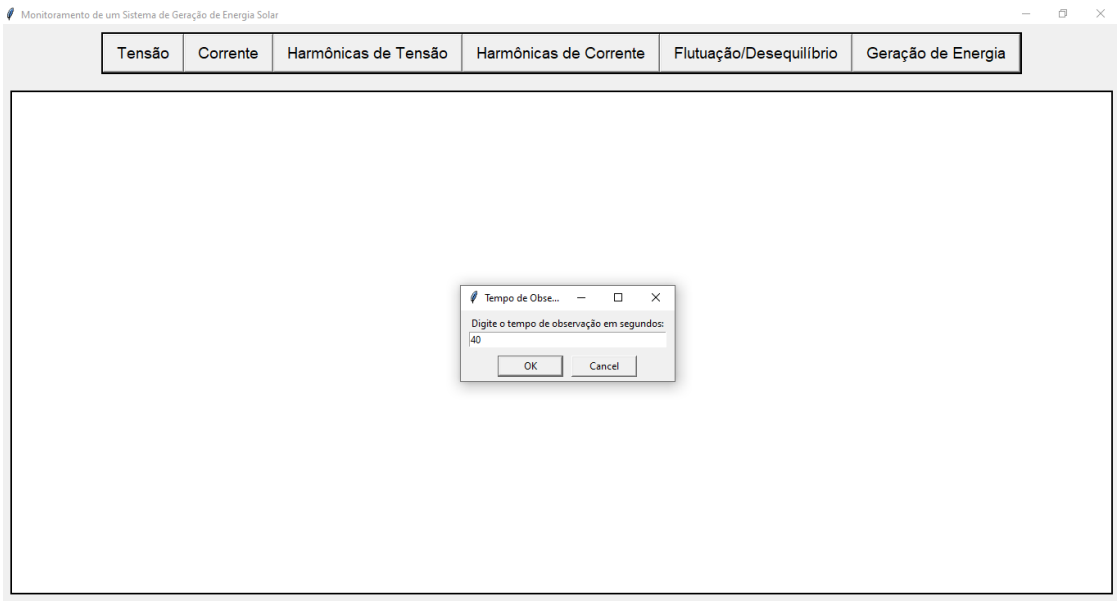
Os elementos visuais da interface foram projetados para garantir uma leitura rápida e precisa dos dados. Isso é crucial para monitorar eficazmente as grandezas elétricas em tempo real, onde a rapidez e precisão da informação podem fazer a diferença na tomada de decisões.

4.2 Funcionalidades e recursos implementados

O resultado final inclui uma variedade de funcionalidades e recursos que permitem monitorar e analisar esses dados de maneira eficiente e intuitiva.

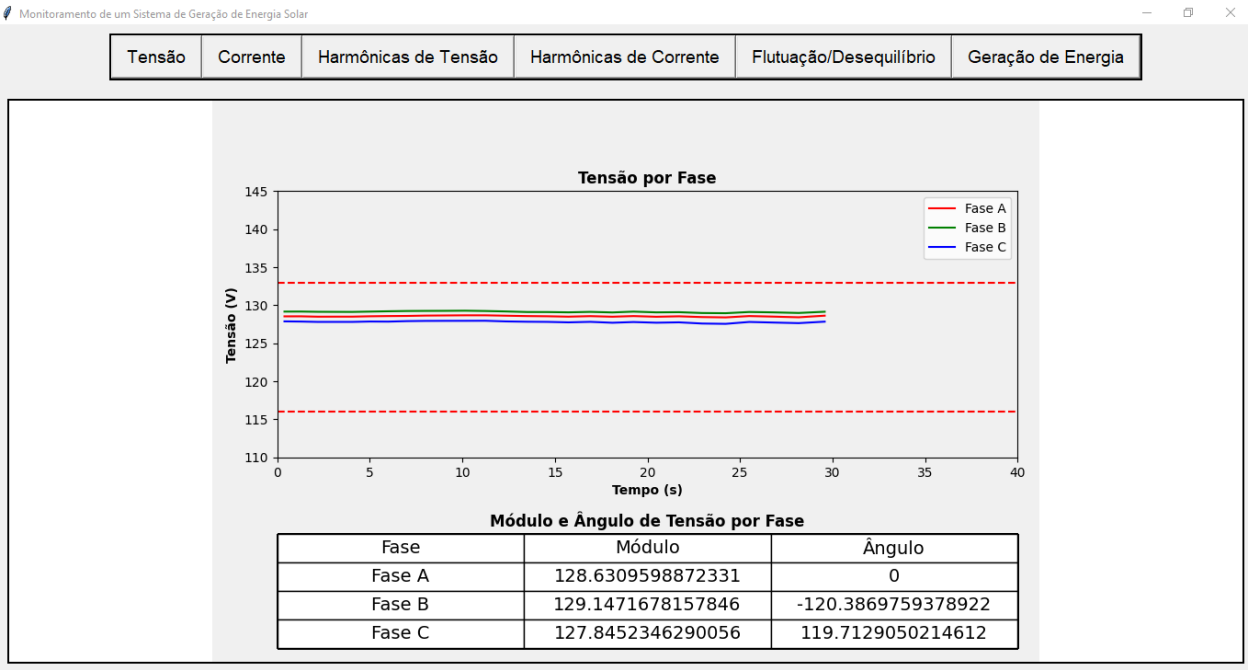
A funcionalidade "Tensão" é um dos principais recursos da interface. Esta funcionalidade exibe um gráfico de linhas que mostra a tensão em cada fase em tempo real. A visualização é construída ao longo do tempo inserido pelo usuário em segundos (Figura 5), após esse tempo, a tela é limpa e o processo recomeça. Adicionalmente, uma tabela abaixo do gráfico exibe os valores de módulo e ângulo de tensão para cada fase, atualizando-se na mesma taxa que o gráfico. Além disso, foram adicionadas linhas tracejadas fixas ao gráfico que representam os limites superior e inferior da tensão (116 e 133), de acordo com os padrões da ANEEL (Figura 6) [15].

Figura 5 – Tempo de Observação



Fonte: Autor.

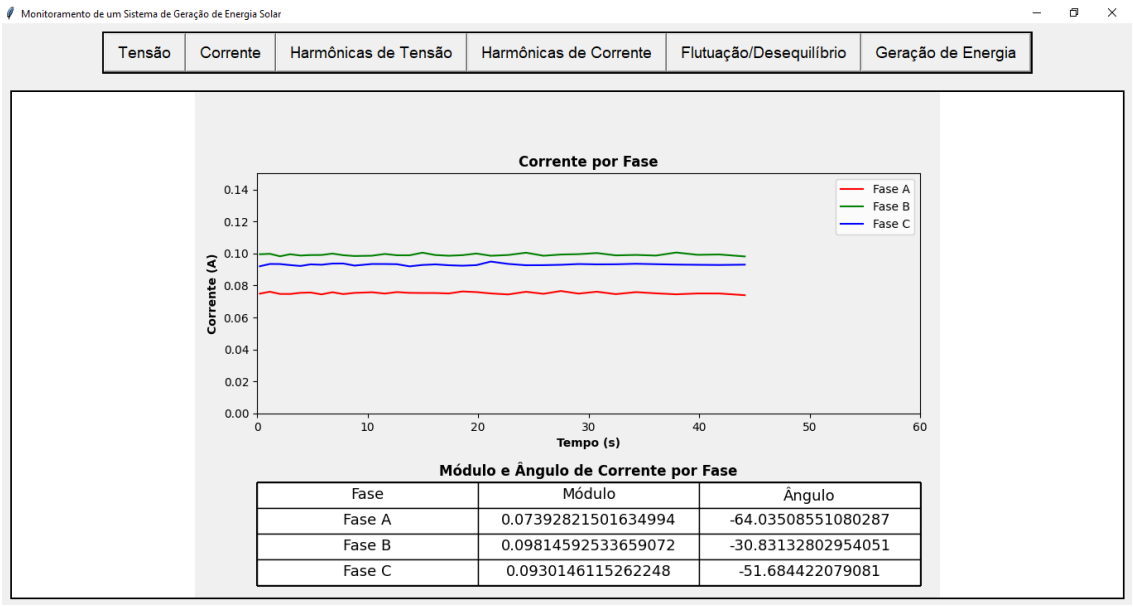
Figura 6 – Tensão em tempo real



Fonte: Autor.

Na mesma linha, a funcionalidade "Corrente" exibe um gráfico de linhas que mostra a corrente em cada fase em tempo real, similarmente ao recurso "Tensão". O gráfico é também construído ao longo do tempo informado pelo usuário, após o tempo definido, a tela é limpa e o processo se reinicia. Além disso, uma tabela é mostrada abaixo do gráfico, mostrando na mesma taxa de atualização os valores de módulo e ângulo de corrente de cada fase (Figura 7).

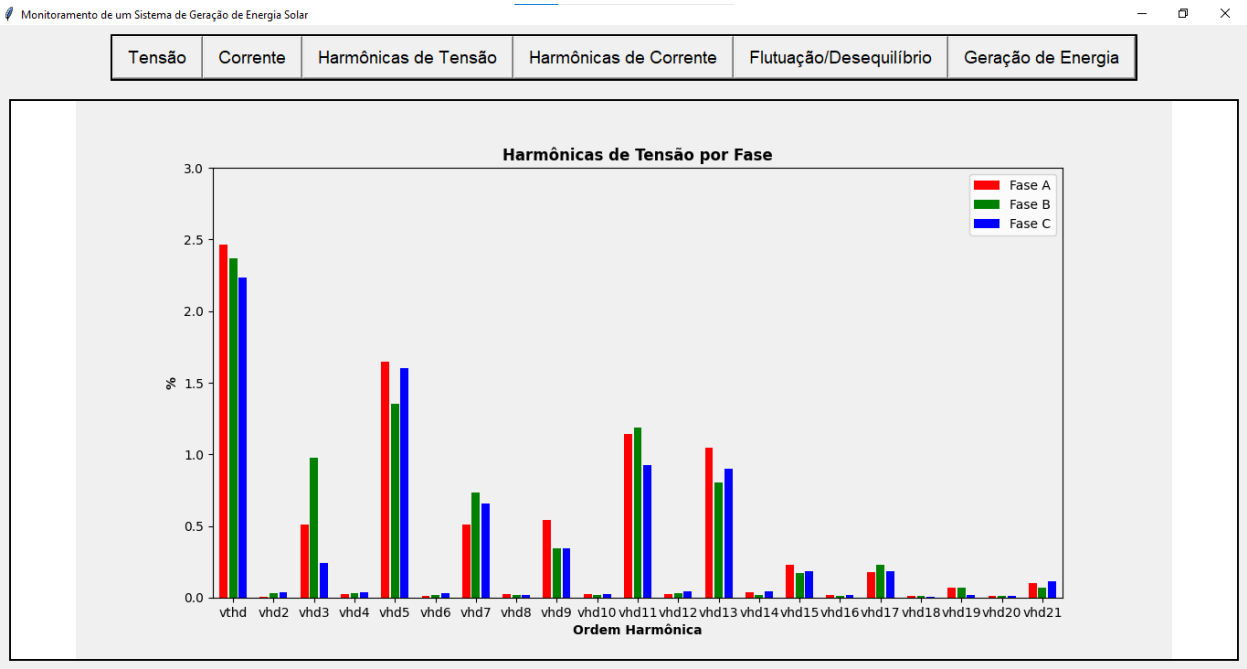
Figura 7 – Corrente em Tempo Real



Fonte: Autor.

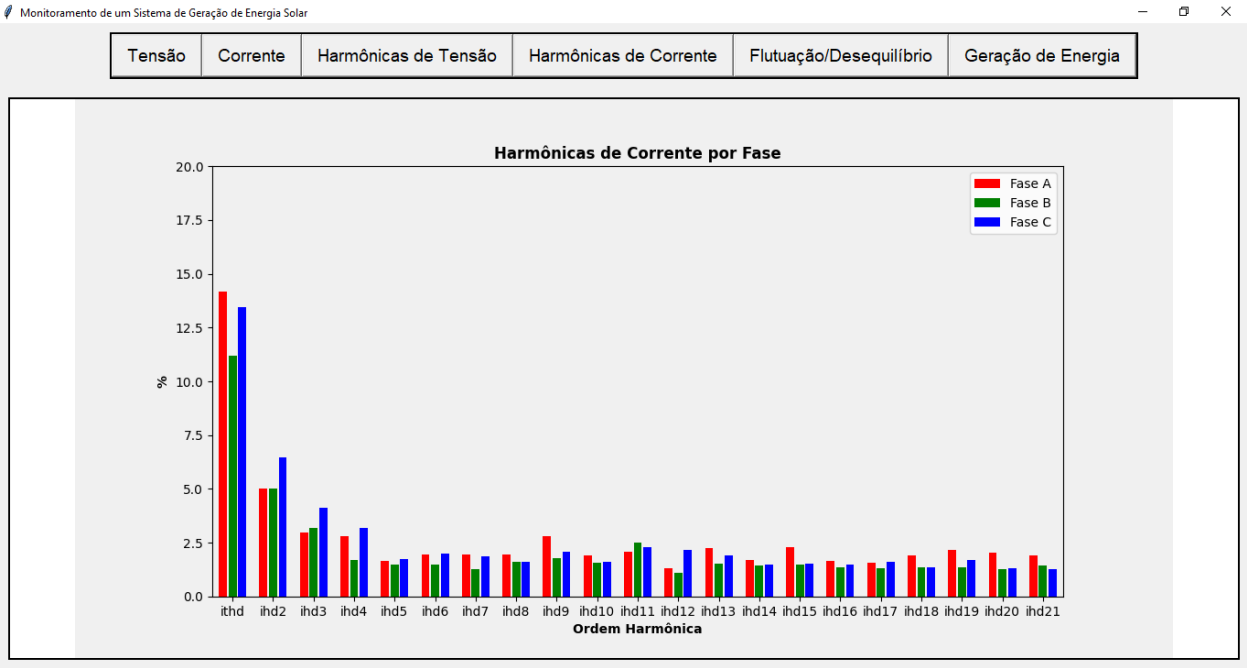
No que diz respeito à análise de harmônicas, a interface implementada também fornece funcionalidades para visualização de "Harmônicas de Tensão" e "Harmônicas de Corrente". Ambas as funcionalidades exibem as harmônicas observadas em cada fase, desde a 1ª ordem até a 21ª ordem. Os valores são dados em percentagem e atualizados na mesma taxa que os outros gráficos. Estes recursos podem ser vistos na Figura 8 – Harmônicas de Tensão e Figura 9 – Harmônicas de Corrente, respectivamente.

Figura 8 – Harmônicas de Tensão por Fase



Fonte: Autor.

Figura 9 – Harmônicas de Corrente por Fase



Fonte: Autor.

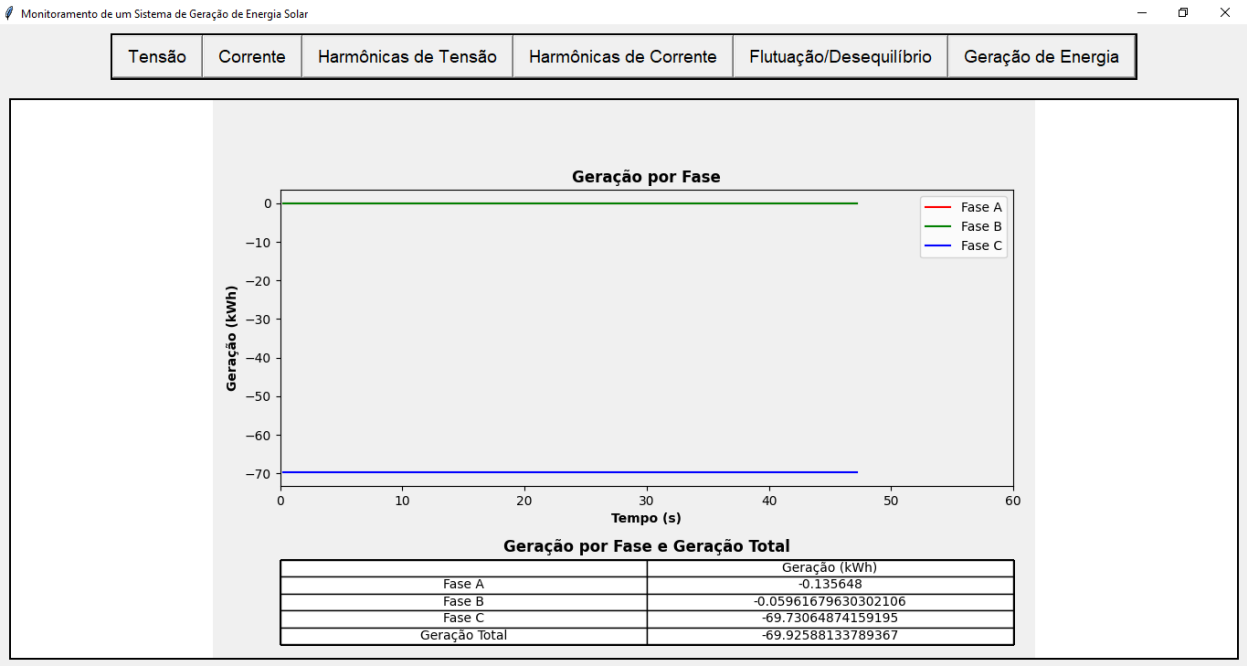
A funcionalidade "Flutuação/Desequilíbrio" é outra ferramenta vital na interface. Ela exibe duas tabelas; a primeira apresenta os valores em pu do desequilíbrio de tensão visto por fase, atualizando-se na mesma taxa que os outros gráficos e tabelas. A segunda tabela apresenta o Desequilíbrio de Tensão em percentagem, também atualizando-se na mesma taxa que os outros gráficos e tabelas (Figura 10).

Figura 10 – Flutuação e Desequilíbrio de Tensão

Fonte:

No que se refere à análise de geração de energia, a interface apresenta a funcionalidade "Geração de Energia". Esta funcionalidade exibe um gráfico de linhas de geração de energia (kWh) por fase, funcionando de maneira similar aos gráficos de tensão e corrente em tempo real. Uma tabela complementar apresenta os dados de geração em kWh por fase, bem como o valor de geração total (Figura 11).

Figura 11 – Geração de Energia por Fase



Fonte: Autor.

Essas funcionalidades, apesar de robustas e abrangentes, não são estáticas ou fixas. A navegação dentro da interface é livre, permitindo ao usuário acessar qualquer funcionalidade a qualquer momento. Esta flexibilidade de uso e a natureza dinâmica da interface aumentam a eficácia do monitoramento e a praticidade do sistema.

4.3 Análise dos resultados obtidos

A interface desenvolvida cumpriu com eficácia o objetivo proposto neste trabalho: fornecer uma maneira simples e intuitiva de monitorar grandezas elétricas em tempo real. A precisão dos dados exibidos, confirmada pela validação direta com a API do medidor, destaca a eficiência do sistema em capturar e apresentar dados relevantes.

A verificação adicional dos dados, feita através de um código fornecido pelo orientador do projeto, também demonstrou a capacidade do sistema de traduzir efetivamente dados brutos em visualizações gráficas compreensíveis e informativas.

No entanto, os valores de corrente e geração de energia apresentados levantam algumas questões sobre a instalação do medidor. Esses valores, observados consistentemente como negativos ou próximos de zero, podem sugerir possíveis problemas com o medidor ou com a sua instalação.

Apesar disso, esses resultados não diminuem a eficácia da interface desenvolvida. Pelo contrário, eles reforçam a importância de sistemas de monitoramento como o apresentado neste trabalho. Esses sistemas não apenas fornecem dados importantes, mas também ajudam a identificar potenciais problemas ou anomalias com base nesses dados.

A conclusão é que a interface de usuário desenvolvida cumpriu os objetivos principais do projeto. Ela permite o monitoramento efetivo de várias grandezas elétricas em tempo real e apresenta essas informações de forma clara e fácil de entender. Com base nessas informações, os usuários podem tomar decisões mais informadas para melhorar a eficiência e a qualidade do sistema elétrico.

Os resultados obtidos neste trabalho são encorajadores e apontam para possíveis áreas de aprimoramento e expansão do sistema no futuro. Apesar das questões relativas à instalação do medidor, a interface desenvolvida oferece uma solução robusta e amigável para o monitoramento em tempo real de grandezas elétricas [34].

5. CONCLUSÃO

5.1 Recapitulação dos objetivos alcançados

Retrospectivamente, observa-se que os objetivos do trabalho foram plenamente atendidos. O propósito principal, de desenvolver uma nova interface de usuário em linguagem Python, foi alcançado com êxito. A interface desenvolvida foi integrada ao medidor AIW/Sigmasys do Laboratório de Distribuição de Energia Elétrica, demonstrando uma eficiente compatibilidade e capacidade de adaptação do sistema.

As funções desejadas foram implementadas e o desempenho da interface se mostrou satisfatório, permitindo a análise em tempo real de diferentes grandezas elétricas, como harmônicos de tensão e corrente, valores de tensão e corrente em tempo real, geração de energia,

flutuação e desequilíbrio de tensão. Isso permite aos usuários um acompanhamento contínuo e preciso dessas grandezas, trazendo benefícios significativos para a monitoração e a otimização dos sistemas de medição em tempo real.

Adicionalmente, o desenvolvimento da interface em Python trouxe benefícios esperados e justificados. Python é uma linguagem de programação moderna e versátil, que oferece facilidades como um vasto conjunto de bibliotecas e uma comunidade ativa, além de ser de fácil aprendizado para novos usuários. Assim, as vantagens desta linguagem se mostraram bastante relevantes no processo de desenvolvimento e na funcionalidade da interface.

Ao considerar o contexto da qualidade da energia elétrica e das redes elétricas inteligentes, o desenvolvimento da interface demonstra-se um avanço significativo. O monitoramento das grandezas elétricas se mostra como um elemento crucial para manter e melhorar a qualidade da energia elétrica, sendo um tópico de extrema relevância para a Engenharia Elétrica.

Assim, conclui-se que os objetivos propostos foram alcançados com sucesso, através da criação de uma interface eficaz e de fácil usabilidade. Espera-se que o uso deste sistema possa contribuir para a otimização do monitoramento e análise de grandezas elétricas, além de trazer insights para futuras melhorias e inovações na área.

5.2 Contribuições do trabalho para a área de Engenharia Elétrica

Este trabalho oferece contribuições significativas na interseção entre software e sistemas de monitoramento em tempo real de grandezas elétricas na Engenharia Elétrica. Em primeiro lugar, ressalta as oportunidades oferecidas por uma linguagem de programação moderna e flexível, como Python, na integração com APIs para monitorar parâmetros elétricos, tais como harmônicos de tensão e corrente, valores de tensão e corrente em tempo real, geração de energia, flutuação e desequilíbrio de tensão [15].

No âmbito profissional, a aplicação da interface Python em sistemas elétricos proporciona a capacidade de monitorar e analisar a qualidade da energia em tempo real. Isso pode beneficiar uma variedade de usuários, desde pequenas instalações industriais até grandes corporações, onde a eficiência energética e a segurança são vitais. Operadores de sistemas com interfaces semelhantes

podem identificar rapidamente quaisquer irregularidades, implementar medidas corretivas eficientes e, portanto, otimizar a operação dos sistemas [16].

Além disso, a interface desenvolvida tem um enorme potencial para contribuir para o campo da pesquisa. A capacidade de coletar e analisar dados em tempo real pode impulsionar avanços significativos na compreensão das redes elétricas inteligentes e na qualidade da energia elétrica. Com a geração de grandes volumes de dados, os pesquisadores podem usar técnicas contemporâneas de análise de dados para identificar tendências, analisar a variabilidade dos sistemas e prever possíveis falhas [17].

O uso de Python no desenvolvimento da interface também oferece uma plataforma eficaz e acessível para o desenvolvimento de outras ferramentas de análise de dados. Python é amplamente adotado na comunidade científica e de engenharia, o que facilita a colaboração entre pesquisadores e engenheiros e permite a replicação e expansão dos estudos [18].

5.3 Sugestões para trabalhos futuros

Neste trabalho, foi desenvolvida uma interface eficaz para o monitoramento em tempo real de grandezas elétricas utilizando a linguagem Python. No entanto, a complexidade e a ampla gama de potenciais aplicações para esta tecnologia sugerem muitas oportunidades para o aprimoramento e expansão deste trabalho.

Um aspecto a ser considerado em estudos futuros é a implementação de funções de oscilografia de tensão e corrente. A API fornece uma grande quantidade de dados para cada ponto que, se devidamente tratados e analisados, podem fornecer uma visão ainda mais completa do sistema elétrico sob estudo. Isso implica a necessidade de manipular múltiplas listas de dados para a implementação de gráficos de fasores, similarmente ao que foi realizado para tensão e corrente em tempo real [20]. Este aprimoramento ampliaria as capacidades de análise do sistema proposto, auxiliando tanto em aspectos técnicos quanto acadêmicos.

Adicionalmente, a interface poderia ser expandida para monitorar e analisar outras grandezas elétricas. A API fornece uma variedade de dados que não foram completamente

explorados neste trabalho, e existe um potencial significativo para o desenvolvimento de novas funções de análise e monitoramento [23].

A automação de alertas para condições específicas no sistema é outro aspecto relevante a ser explorado em futuras pesquisas. Isso permitiria que o sistema informasse proativamente aos usuários sobre possíveis problemas, melhorando a eficiência da operação e manutenção do sistema elétrico [24].

A expansão da interface para ser compatível com outros sistemas de monitoramento também é uma possibilidade. Embora o foco deste trabalho tenha sido os medidores AIW/Sigmasys, outros sistemas poderiam se beneficiar das funções de monitoramento e análise oferecidas por esta interface [25].

REFERÊNCIAS BIBLIOGRÁFICAS

[1] IEEE Recommended Practice for Monitoring Electric Power Quality, IEEE Std 1159-1995. Disponível em: https://standards.ieee.org/standard/1159-2009.html. Acesso em: 01 Abr. 2023.

[2] Gungor, V. C., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C., & Hancke, G. P. (2013). Smart grid technologies: communication technologies and standards. IEEE transactions on Industrial informatics, 7(4), 529-539. Disponível em: https://ieeexplore.ieee.org/document/6027201. Acesso em: 10 Abr. 2023.

[3] Fadlullah, Z. M., Nozaki, Y., Kamakura, T., & Nakao, A. (2012). A dynamic traffic prediction model for smart grid communications. IEEE Network, 26(6), 41-47. Disponível em: https://ieeexplore.ieee.org/document/6383060. Acesso em: 20 Abr. 2023.

[4] Van Rossum, G., & Drake, F. L. (2009). Python 3 Reference Manual. CreateSpace. Disponível em: https://www.amazon.com/Python-3-Reference-Manual-Rossum/dp/1441412697. Acesso em: 30 Abr. 2023.

[5] Grus, J. (2015). Data science from scratch: first principles with python. "O'Reilly Media, Inc.". Disponível em: https://www.oreilly.com/library/view/data-science-from/9781491901410/. Acesso em: 10 Mai. 2023.

[6] SILVA, S. (2020). APIs: fundamentos, protocolos e integração de sistemas. Casa do Código. Disponível em: https://www.casadocodigo.com.br/products/livro-apis. Acesso em: 20 Mai. 2023.

[7] BATISTA, T. V., & MELO, C. (2017). Desenvolvimento de software baseado em componentes. Bookman editora. Disponível em: https://www.grupoa.com.br/desenvolvimento-de-software-baseado-em-componentes/9788582604618. Acesso em: 30 Mai. 2023.

[8] Sigmasys, “AIW/Sigmasys". Disponível em: https://pqsys.sigmasys.com.br/. Acesso em: 09 Jun. 2023.

[9] EcoRI Energia Solar. (s.d.). Painel Solar HJP72 330-335W. Disponível em: https://www.ecorienergiasolar.com.br/assets/uploads/9c946-dhp72-330-335w-14_11.pdf. Acesso em: 18 Jun. 2023.

[10] BARBOSA, P.N., POMÍLIO, J.A., DEZAN, D.J., LOPES, A.A. "Qualidade da Energia Elétrica: Conceitos e Aplicações". Artliber Editora, 2008. Disponível em: https://www.livrariacultura.com.br/p/livros/engenharia/eletrica/qualidade-da-energia-eletrica-2000190. Acesso em: 02 Abr. 2023.

[11] Python Software Foundation. Tkinter — Interface Python para Tcl/Tk. Python v3.9.2 documentation. Disponível em: https://docs.python.org/3/library/tkinter.html. Acesso em: 12 Abr. 2023.

[12] The Pandas Development Team. Pandas: powerful Python data analysis toolkit. Disponível em: https://pandas.pydata.org/. Acesso em: 22 Abr. 2023.

[13] John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007), DOI:10.1109/MCSE.2007.55. Disponível em: https://matplotlib.org/. Acesso em: 02 Mai. 2023.

[14] Python Requests. Requests: HTTP for Humans™. Disponível em: https://docs.python-requests.org/en/latest/. Acesso em: 12 Mai. 2023.

[15] ANEEL. Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional – PRODIST. Módulo 8 – Qualidade da Energia Elétrica. Brasília, DF: ANEEL, 2018. Disponível em: http://www.aneel.gov.br/cedoc/ren2018700.pdf. Acesso em: 22 Mai. 2023.

[16] Lopes, J.A.P., et al., (2007). Integration of electric vehicles in the electric power system. Proceedings of the IEEE, 99(1), 168-183. Disponível em: https://ieeexplore.ieee.org/document/5676340. Acesso em: 01 Jun. 2023.

[17] Marz, N. and Warren, J. (2015). Big Data: Principles and best practices of scalable real-time data systems. Manning Publications Co. Disponível em: https://www.manning.com/books/big-data. Acesso em: 11 Jun. 2023.

[18] Sovacool, B.K., (2016). The political economy of energy poverty: A review of key challenges. Energy for Sustainable Development, 31, 2-10. Disponível em: https://www.sciencedirect.com/science/article/pii/S0973082616300797. Acesso em: 18 Jun. 2023.

[19] Oliveira, C. R. de., Souza, A. N. de., & Gonçalves, M. V. (2015). Oscilografia em sistemas de potência: técnicas de compressão de dados e detecção de eventos transitórios. Revista Brasileira de Energia Elétrica, 2(2), 42-52. Disponível em: http://revista.abracopel.org/index.php/rbee/article/view/51. Acesso em: 03 Abr. 2023.

[20] Santos, M. H., & Figueiredo, K. B. (2019). Aplicação da ciência de dados no gerenciamento do sistema elétrico brasileiro: uma revisão sistemática. Revista Brasileira de Engenharia e Tecnologia, 6(1), 13-26. Disponível em: http://revista.fatecbt.edu.br/index.php/rbet/article/view/69. Acesso em: 13 Abr. 2023.

[21] Silva, G. C. da., Costa, M. E., & Ferreira, L. A. F. (2016). Uma visão computacional da qualidade da energia elétrica no Brasil. Revista Brasileira de Informática na Educação, 24(1), 28-39. Disponível em: https://www.br-ie.org/pub/index.php/rbie/article/view/5454. Acesso em: 23 Abr. 2023. [22] A. Monti and F. Ponci, "Power flow control and network stability in an all-electric ship," IEEE Trans. Power Electron., vol. 25, no. 3, pp. 821–834, Mar. 2010. Disponível em: https://ieeexplore.ieee.org/document/5239825. Acesso em: 18 Jun.23.

[23] D. Della Giustina, M. Sforna, M. Pau, and P. A. Pegoraro, "A low cost smart meter for monitoring the power quality in distribution networks," 2017 IEEE International Workshop on Measurements & Networking (M&N), pp. 1–6, 2017. Disponível em: https://ieeexplore.ieee.org/document/8078332. Acesso em: 18 Jun.23

[24] R. Tonkoski, D. Turcotte, and T. H. M. El-Fouly, "Impact of high PV penetration on voltage profiles in residential neighborhoods," IEEE Trans. Sustain. Energy, vol. 3, no. 3, pp. 518–527, Jul. 2012. Disponível em: https://ieeexplore.ieee.org/document/6169928. Acesso em: 18 Jun.23

[25] M. Shahidehpour, Y. Fu, and T. Wiedman, "Impact of natural gas infrastructure on electric power systems," Proc. IEEE, vol. 93, no. 5, pp. 1042–1056, May 2005. Disponível em: https://ieeexplore.ieee.org/document/1396356. Acesso em: 18 Jun.23.

ANEXO 1 - CÓDIGO FONTE DA INTERFACE EM PYTHON

```
import tkinter as tk
import tkinter.simpledialog as simpledialog
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import pandas as pd
import requests
import time

root = tk.Tk()
root.geometry("1200x600")
root.title("Monitoramento de um Sistema de Geração de Energia Solar")
root.configure(bg='#F0F0F0')

frame_buttons = tk.Frame(root, bg='#F0F0F0', bd=2, relief='solid')
frame_buttons.pack(side='top', padx=10, pady=10)

frame_plots = tk.Frame(root, bg='white', bd=2, relief='solid')
frame_plots.pack(side='top', padx=10, pady=10, fill='both', expand=True)

def clear_frame():
    for widget in frame_plots.winfo_children():
        widget.destroy()
def plot_corrente():
    clear_frame()

    root = tk.Tk()
    root.withdraw()
```

```
tempo_observacao = simpledialog.askinteger("Tempo de Observação", "Digite o tempo de observação em segundos:")

fig, (ax, ax_table) = plt.subplots(2, 1, figsize=(9, 10), gridspec_kw={'height_ratios': [7, 3]})
fig.patch.set_facecolor('#F0F0F0')
ax.set_facecolor('#F0F0F0')
ax_table.set_facecolor('#F0F0F0')

def animate(i):
    response = requests.get('LINK DA API'')
    data = response.json()

    corrente_faseA_modulo_med1 = data['fasor']['corrente']['faseA']['modulo']
    corrente_faseB_modulo_med1 = data['fasor']['corrente']['faseB']['modulo']
    corrente_faseC_modulo_med1 = data['fasor']['corrente']['faseC']['modulo']

    corrente_faseA_angulo_med1 = data['fasor']['corrente']['faseA']['angulo']
    corrente_faseB_angulo_med1 = data['fasor']['corrente']['faseB']['angulo']
    corrente_faseC_angulo_med1 = data['fasor']['corrente']['faseC']['angulo']

    animate.times.append(time.time() - animate.start_time)
    animate.corrente_A.append(corrente_faseA_modulo_med1)
    animate.corrente_B.append(corrente_faseB_modulo_med1)
    animate.corrente_C.append(corrente_faseC_modulo_med1)

    animate.times = animate.times[-tempo_observacao:]
    animate.corrente_A = animate.corrente_A[-tempo_observacao:]
    animate.corrente_B = animate.corrente_B[-tempo_observacao:]
    animate.corrente_C = animate.corrente_C[-tempo_observacao:]
```

```
            ax.clear()

            ax.plot(animate.times, animate.corrente_A, label='Fase A', color='red')
            ax.plot(animate.times, animate.corrente_B, label='Fase B', color='green')
            ax.plot(animate.times, animate.corrente_C, label='Fase C', color='blue')

            ax.set_xlim([0, tempo_observacao])
            ax.set_ylim([0, 0.15])
            ax.set_xlabel('Tempo (s)', fontweight='bold')
            ax.set_ylabel('Corrente (A)', fontweight='bold')
            ax.set_title('Corrente por Fase', fontweight='bold')
            ax.legend(loc='upper right')

            table_data = [
                ['Fase A', corrente_faseA_modulo_med1, corrente_faseA_angulo_med1],
                ['Fase B', corrente_faseB_modulo_med1, corrente_faseB_angulo_med1],
                ['Fase C', corrente_faseC_modulo_med1, corrente_faseC_angulo_med1]
            ]
            table = ax_table.table(cellText=table_data, colLabels=['Fase', 'Módulo',
'Ângulo'], cellLoc='center',
                                   loc='center', bbox=[0, 0, 1, 1])
            table.set_fontsize(14)
            table.scale(1.5, 1.5)
            ax_table.axis('off')
            ax_table.set_title('Módulo e Ângulo de Corrente por Fase', fontweight='bold')

            fig.tight_layout(rect=[0, 0, 1, 0.9])

            if animate.times[-1] >= tempo_observacao:
                animate.times = []
```

```
                animate.corrente_A = []
                animate.corrente_B = []
                animate.corrente_C = []
                animate.start_time = time.time()

            animate.times = []
            animate.corrente_A = []
            animate.corrente_B = []
            animate.corrente_C = []
            animate.start_time = time.time()

            ani = animation.FuncAnimation(fig, animate, interval=500)

            canvas = FigureCanvasTkAgg(fig, master=frame_plots)
            canvas.draw()
            canvas.get_tk_widget().pack()

        import tkinter as tk
        import tkinter.simpledialog as simpledialog

        def plot_tensao():
            clear_frame()

            root = tk.Tk()
            root.withdraw()

            tempo_observacao = simpledialog.askinteger("Tempo de Observação", "Digite
o tempo de observação em segundos:")

            fig, (ax, ax_table) = plt.subplots(2, 1, figsize=(9, 10),
gridspec_kw={'height_ratios': [7, 3]})
```

```
fig.patch.set_facecolor('#F0F0F0')
ax.set_facecolor('#F0F0F0')
ax_table.set_facecolor('#F0F0F0')

def animate(i):
    response = requests.get('LINK DA API')
    data = response.json()

    tensao_faseA_modulo_med1 = data['fasor']['tensao']['faseA']['modulo']
    tensao_faseB_modulo_med1 = data['fasor']['tensao']['faseB']['modulo']
    tensao_faseC_modulo_med1 = data['fasor']['tensao']['faseC']['modulo']

    tensao_faseA_angulo_med1 = data['fasor']['tensao']['faseA']['angulo']
    tensao_faseB_angulo_med1 = data['fasor']['tensao']['faseB']['angulo']
    tensao_faseC_angulo_med1 = data['fasor']['tensao']['faseC']['angulo']

    animate.times.append(time.time() - animate.start_time)
    animate.tension_A.append(tensao_faseA_modulo_med1)
    animate.tension_B.append(tensao_faseB_modulo_med1)
    animate.tension_C.append(tensao_faseC_modulo_med1)

    animate.times = animate.times[-tempo_observacao:]
    animate.tension_A = animate.tension_A[-tempo_observacao:]
    animate.tension_B = animate.tension_B[-tempo_observacao:]
    animate.tension_C = animate.tension_C[-tempo_observacao:]

    ax.clear()

    ax.plot(animate.times, animate.tension_A, label='Fase A', color='red')
    ax.plot(animate.times, animate.tension_B, label='Fase B', color='green')
    ax.plot(animate.times, animate.tension_C, label='Fase C', color='blue')
```

```
            ax.axhline(y=116, color='r', linestyle='--')
            ax.axhline(y=133, color='r', linestyle='--')

            ax.set_xlim([0, tempo_observacao])
            ax.set_ylim([110, 145])
            ax.set_xlabel('Tempo (s)', fontweight='bold')
            ax.set_ylabel('Tensão (V)', fontweight='bold')
            ax.set_title('Tensão por Fase', fontweight='bold')
            ax.legend(loc='upper right')

            table_data = [
                ['Fase A', tensao_faseA_modulo_med1, tensao_faseA_angulo_med1],
                ['Fase B', tensao_faseB_modulo_med1, tensao_faseB_angulo_med1],
                ['Fase C', tensao_faseC_modulo_med1, tensao_faseC_angulo_med1]
            ]
            table = ax_table.table(cellText=table_data, colLabels=['Fase', 'Módulo',
'Ângulo'], cellLoc='center',
                                   loc='center', bbox=[0, 0, 1, 1])
            table.set_fontsize(14)
            table.scale(1.5, 1.5)
            ax_table.axis('off')
            ax_table.set_title('Módulo e Ângulo de Tensão por Fase', fontweight='bold')

            fig.tight_layout(rect=[0, 0, 1, 0.9])

            if animate.times[-1] >= tempo_observacao:
                animate.times = []
                animate.tension_A = []
                animate.tension_B = []
                animate.tension_C = []
```

```
                animate.start_time = time.time()

            animate.times = []
            animate.tension_A = []
            animate.tension_B = []
            animate.tension_C = []
            animate.start_time = time.time()

            ani = animation.FuncAnimation(fig, animate, interval=500)

            canvas = FigureCanvasTkAgg(fig, master=frame_plots)
            canvas.draw()
            canvas.get_tk_widget().pack()

        def plot_harmonicas_corrente():
            clear_frame()
            fig, ax_corrente = plt.subplots(1, 1, figsize=(12, 10))  # Aumentando o valor de
figsize para aumentar o espaço horizontal
            fig.patch.set_facecolor('#F0F0F0')
            ax_corrente.set_facecolor('#F0F0F0')

            def animate(i):
                response = requests.get('LINK DA API')
                data = response.json()

                ordens = np.arange(1, 22)  # Alterado para incluir a primeira ordem
                valores_faseA_corrente = []
                valores_faseB_corrente = []
                valores_faseC_corrente = []

                # Adicionando a primeira harmônica manualmente
```

```
                valores_faseA_corrente.append(data["harmonicascorrente"]["iathd"])
                valores_faseB_corrente.append(data["harmonicascorrente"]["ibthd"])
                valores_faseC_corrente.append(data["harmonicascorrente"]["icthd"])

                for ordem in ordens[1:]:  # Iterando de 2 até 21

valores_faseA_corrente.append(data["harmonicascorrente"]["average"]["iahd" +
str(ordem)])

valores_faseB_corrente.append(data["harmonicascorrente"]["average"]["ibhd" +
str(ordem)])

valores_faseC_corrente.append(data["harmonicascorrente"]["average"]["ichd" +
str(ordem)])

                valores_faseA_corrente = valores_faseA_corrente[::]  # Inverter a ordem dos
valores para exibir as maiores ordens primeiro
                valores_faseB_corrente = valores_faseB_corrente[::]  # Inverter a ordem dos
valores para exibir as maiores ordens primeiro
                valores_faseC_corrente = valores_faseC_corrente[::]  # Inverter a ordem dos
valores para exibir as maiores ordens primeiro

                animate.times.append(time.time() - animate.start_time)

                animate.valores_faseA_corrente = valores_faseA_corrente
                animate.valores_faseB_corrente = valores_faseB_corrente
                animate.valores_faseC_corrente = valores_faseC_corrente

                ax_corrente.clear()

                bar_width = 0.2  # Largura das barras
```

```
            offset = bar_width * 1.2  # Offset para alinhar as barras

            ax_corrente.bar(ordens[:len(animate.valores_faseA_corrente)] - offset,
animate.valores_faseA_corrente, bar_width, label='Fase A', color='red')
            ax_corrente.bar(ordens[:len(animate.valores_faseB_corrente)],
animate.valores_faseB_corrente, bar_width, label='Fase B', color='green')
            ax_corrente.bar(ordens[:len(animate.valores_faseC_corrente)] + offset,
animate.valores_faseC_corrente, bar_width, label='Fase C', color='blue')

            ax_corrente.set_xticks(ordens)
            ax_corrente.set_xticklabels(["ithd"] + ["ihd" + str(i) for i in range(2, 22)])
            ax_corrente.set_xlim([0.5, 21.5])  # Ajuste para incluir a primeira e última
ordem completamente
            ax_corrente.set_xlabel('Ordem Harmônica', fontweight='bold')
            ax_corrente.set_ylim([0, 20])
            ax_corrente.set_ylabel('%', fontweight='bold')
            ax_corrente.set_title('Harmônicas de Corrente por Fase', fontweight='bold')
            ax_corrente.legend(loc='upper right')

            if animate.times[-1] >= 60:
                animate.times = []
                animate.valores_faseA_corrente = []
                animate.valores_faseB_corrente = []
                animate.valores_faseC_corrente = []
                animate.start_time = time.time()

        animate.times = []
        animate.valores_faseA_corrente = []
        animate.valores_faseB_corrente = []
        animate.valores_faseC_corrente = []
        animate.start_time = time.time()
```

```
            ani = animation.FuncAnimation(fig, animate, interval=500)

            canvas = FigureCanvasTkAgg(fig, master=frame_plots)
            canvas.draw()
            canvas.get_tk_widget().pack()

        def plot_harmonicas_tensao():
            clear_frame()
            fig, ax_tensao = plt.subplots(1, 1, figsize=(12, 10))  # Aumentando o valor de figsize para aumentar o espaço horizontal
            fig.patch.set_facecolor('#F0F0F0')
            ax_tensao.set_facecolor('#F0F0F0')

            def animate(i):
                response = requests.get('LINK DA API)
                data = response.json()

                ordens = np.arange(1, 22)  # Alterado para incluir a primeira ordem
                valores_faseA_tensao = []
                valores_faseB_tensao = []
                valores_faseC_tensao = []

                # Adicionando a primeira harmônica manualmente
                valores_faseA_tensao.append(data["harmonicastensao"]["vathd"])
                valores_faseB_tensao.append(data["harmonicastensao"]["vbthd"])
                valores_faseC_tensao.append(data["harmonicastensao"]["vcthd"])

                for ordem in ordens[1:]:  # Iterando de 2 até 21
```

```
valores_faseA_tensao.append(data["harmonicastensao"]["average"]["vahd" + str(ordem)])

valores_faseB_tensao.append(data["harmonicastensao"]["average"]["vbhd" + str(ordem)])

valores_faseC_tensao.append(data["harmonicastensao"]["average"]["vchd" + str(ordem)])

        valores_faseA_tensao = valores_faseA_tensao[::]  # Inverter a ordem dos valores para exibir as maiores ordens primeiro
        valores_faseB_tensao = valores_faseB_tensao[::]  # Inverter a ordem dos valores para exibir as maiores ordens primeiro
        valores_faseC_tensao = valores_faseC_tensao[::]  # Inverter a ordem dos valores para exibir as maiores ordens primeiro

        animate.times.append(time.time() - animate.start_time)

        animate.valores_faseA_tensao = valores_faseA_tensao
        animate.valores_faseB_tensao = valores_faseB_tensao
        animate.valores_faseC_tensao = valores_faseC_tensao

        ax_tensao.clear()

        bar_width = 0.2  # Largura das barras
        offset = bar_width * 1.2  # Offset para alinhar as barras

        ax_tensao.bar(ordens[:len(animate.valores_faseA_tensao)] - offset, animate.valores_faseA_tensao, bar_width, label='Fase A', color='red')
```

```
            ax_tensao.bar(ordens[:len(animate.valores_faseB_tensao)],
animate.valores_faseB_tensao, bar_width, label='Fase B', color='green')
            ax_tensao.bar(ordens[:len(animate.valores_faseC_tensao)]     +     offset,
animate.valores_faseC_tensao, bar_width, label='Fase C', color='blue')

            ax_tensao.set_xticks(ordens)
            ax_tensao.set_xticklabels(["vthd"] + ["vhd" + str(i) for i in range(2, 22)])
            ax_tensao.set_xlim([0.5, 21.5])  # Ajuste para incluir a primeira e última
ordem completamente
            ax_tensao.set_xlabel('Ordem Harmônica', fontweight='bold')
            ax_tensao.set_ylim([0, 3])
            ax_tensao.set_ylabel('%', fontweight='bold')
            ax_tensao.set_title('Harmônicas de Tensão por Fase', fontweight='bold')
            ax_tensao.legend(loc='upper right')

            if animate.times[-1] >= 60:
                animate.times = []
                animate.valores_faseA_tensao = []
                animate.valores_faseB_tensao = []
                animate.valores_faseC_tensao = []
                animate.start_time = time.time()

        animate.times = []
        animate.valores_faseA_tensao = []
        animate.valores_faseB_tensao = []
        animate.valores_faseC_tensao = []
        animate.start_time = time.time()

        ani = animation.FuncAnimation(fig, animate, interval=500)

        canvas = FigureCanvasTkAgg(fig, master=frame_plots)
```

```
    canvas.draw()
    canvas.get_tk_widget().pack()

def plot_flut_deseq():
    clear_frame()
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 6))
    fig.patch.set_facecolor('#F0F0F0')
    ax1.set_facecolor('#F0F0F0')
    ax2.set_facecolor('#F0F0F0')

    def animate(i):
        response = requests.get('LINK DA API')
        data = response.json()

        flut_faseA = data["flicker"]["faseA"]
        flut_faseB = data["flicker"]["faseB"]
        flut_faseC = data["flicker"]["faseC"]

        deseq_tensao = data["desequilibrio"]

        ax1.clear()
        ax2.clear()

        # Tabela de Flutuação de Tensão por Fase
        dados_flut = {
            'Fase': ['Fase A', 'Fase B', 'Fase C'],
            'Flutuação de Tensão (pu)': [flut_faseA, flut_faseB, flut_faseC]
        }

        df_flut = pd.DataFrame(dados_flut)
```

```
tabela_flut = ax1.table(
    cellText=df_flut.values,
    colLabels=df_flut.columns,
    loc='center',
    cellLoc='center',
    bbox=[0, 0, 1, 1],
    edges='closed'
)

tabela_flut.auto_set_font_size(False)
tabela_flut.set_fontsize(14)
tabela_flut.scale(1, 1.5)

# Tabela de Desequilíbrio de Tensão
dados_deseq = {
    'Desequilíbrio de Tensão (%)': [deseq_tensao]
}

df_deseq = pd.DataFrame(dados_deseq)

tabela_deseq = ax2.table(
    cellText=df_deseq.values,
    colLabels=df_deseq.columns,
    loc='center',
    cellLoc='center',
    bbox=[0, 0, 1, 1],
    edges='closed'
)

tabela_deseq.auto_set_font_size(False)
tabela_deseq.set_fontsize(14)
```

```
            tabela_deseq.scale(1, 1.5)

            ax1.axis('off')
            ax2.axis('off')

        ani = animation.FuncAnimation(fig, animate, interval=500)

        canvas = FigureCanvasTkAgg(fig, master=frame_plots)
        canvas.draw()
        canvas.get_tk_widget().pack()

    def plot_geracao_energia():
        clear_frame()

        root = tk.Tk()
        root.withdraw()

        tempo_observacao = simpledialog.askinteger("Tempo de Observação", "Digite o tempo de observação em segundos:")

        fig, (ax, ax_table1) = plt.subplots(2, 1, figsize=(9, 10), gridspec_kw={'height_ratios': [7, 2]})
        fig.patch.set_facecolor('#F0F0F0')
        ax.set_facecolor('#F0F0F0')
        ax_table1.set_facecolor('#F0F0F0')

        def animate(i):
            response = requests.get('LINK DA API')
            data = response.json()

            ger_faseA = data['energiageracao']['kwha']
```

```
ger_faseB = data['energiageracao']['kwhb']
ger_faseC = data['energiageracao']['kwhc']

ger_total = data['energiageracao']['kwh']

animate.times.append(time.time() - animate.start_time)
animate.geracao_A.append(ger_faseA)
animate.geracao_B.append(ger_faseB)
animate.geracao_C.append(ger_faseC)

animate.times = animate.times[-tempo_observacao:]
animate.geracao_A = animate.geracao_A[-tempo_observacao:]
animate.geracao_B = animate.geracao_B[-tempo_observacao:]
animate.geracao_C = animate.geracao_C[-tempo_observacao:]

ax.clear()

ax.plot(animate.times, animate.geracao_A, label='Fase A', color='red')
ax.plot(animate.times, animate.geracao_B, label='Fase B', color='green')
ax.plot(animate.times, animate.geracao_C, label='Fase C', color='blue')

ax.set_xlim([0, tempo_observacao])
ax.set_xlabel('Tempo (s)', fontweight='bold')
ax.set_ylabel('Geração (kWh)', fontweight='bold')
ax.set_title('Geração por Fase', fontweight='bold')
ax.legend(loc='upper right')

table_data1 = [
    ['Fase A', ger_faseA],
    ['Fase B', ger_faseB],
    ['Fase C', ger_faseC],
```

```
            ['Geração Total', ger_total]
        ]
        table1 = ax_table1.table(cellText=table_data1, colLabels=['', 'Geração (kWh)'], cellLoc='center', loc='center',
                                 bbox=[0, 0, 1, 1])
        table1.set_fontsize(10)
        table1.scale(1.5, 1.5)
        ax_table1.axis('off')
        ax_table1.set_title('Geração por Fase e Geração Total', fontweight='bold')

        fig.tight_layout(rect=[0, 0, 1, 0.9])

        if animate.times[-1] >= tempo_observacao:
            animate.times = []
            animate.geracao_A = []
            animate.geracao_B = []
            animate.geracao_C = []
            animate.start_time = time.time()

    animate.times = []
    animate.geracao_A = []
    animate.geracao_B = []
    animate.geracao_C = []
    animate.start_time = time.time()

    ani = animation.FuncAnimation(fig, animate, interval=500)

    canvas = FigureCanvasTkAgg(fig, master=frame_plots)
    canvas.draw()
    canvas.get_tk_widget().pack()
```

```
    button_tensao = tk.Button(frame_buttons, text="Tensão", command=plot_tensao, font=('Arial', 14), padx=10, pady=5)
    button_tensao.pack(side='left')

    button_corrente = tk.Button(frame_buttons, text="Corrente", command=plot_corrente, font=('Arial', 14), padx=10, pady=5)
    button_corrente.pack(side='left')

    button_harmonicas = tk.Button(frame_buttons, text="Harmônicas de Tensão", command=plot_harmonicas_tensao, font=('Arial', 14), padx=10, pady=5)
    button_harmonicas.pack(side='left')

    button_harmonicas = tk.Button(frame_buttons, text="Harmônicas de Corrente", command=plot_harmonicas_corrente, font=('Arial', 14), padx=10, pady=5)
    button_harmonicas.pack(side='left')

    button_geracao_energia = tk.Button(frame_buttons, text="Flutuação/Desequilíbrio", command=plot_flut_deseq, font=('Arial', 14), padx=10, pady=5)
    button_geracao_energia.pack(side='left')

    button_geracao_energia = tk.Button(frame_buttons, text="Geração de Energia", command=plot_geracao_energia, font=('Arial', 14), padx=10, pady=5)
    button_geracao_energia.pack(side='left')

    root.mainloop()
```