UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Miguel Hellmann Preuss

Software de Automação de Casos de Testes para Sistemas de Controle de Geração de Energia

Florianópolis
2023

Miguel Hellmann Preuss

Software de Automação de Casos de Testes para Sistemas de Controle de Geração de Energia

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Rodrigo Castelan Carlson, Dr.
Supervisor: Marcelo Schmidt Jacobsen, Eng.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Preuss, Miguel Hellmann
Software de Automação de Casos de Testes para Sistemas de Controle de Geração de Energia / Miguel Hellmann Preuss ; orientador, Rodrigo Castelan Carlson, coorientador, Marcelo Schmidt Jacobsen, coorientador, Adelson Alves Junior, 2023.
103 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Desenvolvimento de Software. 3. Automação de Testes. 4. Sistemas de Controle. 5. Geração de Energia. I. Carlson, Rodrigo Castelan. II. Jacobsen, Marcelo Schmidt. III. Junior, Adelson Alves IV. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. V. Título.

Miguel Hellmann Preuss

Software de Automação de Casos de Testes para Sistemas de Controle de Geração de Energia

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 12 de dezembro de 2023.

Prof. Marcelo de Lellis Costa de Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Rodrigo Castelan Carlson, Dr.
Orientador
UFSC/CTC/DAS

Marcelo Schmidt Jacobsen, Eng.
Supervisor
REIVAX Automação e Controle S/A

Adelson Alves Junior, Eng.
Co-supervisor
REIVAX Automação e Controle S/A

Prof. Marcus Vinícius Americano da Costa Filho, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

AGRADECIMENTOS

Agradeço imensamente à minha mãe, Sônia, e às minhas irmãs, Monique e Priscila, pelo apoio ao longo de toda a minha jornada, não apenas acadêmica. Seus conselhos e motivação foram fundamentais para minha decisão de cursar o ensino superior. Sem o suporte da família, o percurso teria sido consideravelmente mais desafiador. A presença de pessoas tão incríveis ao meu lado tem sido um alento, minimizando as diversas barreiras que enfrentamos no dia a dia. Sou profundamente grato por tê-las em minha vida, e agora, também, incluo a minha sobrinha Luiza.

Em âmbito acadêmico, agradecimentos ao professor Rodrigo pela orientação acadêmica ao longo da execução do trabalho, especificamente em desenvolvimento de software. Agradeço a todos os professores do Departamento de Automação e Sistemas, profissionais com as mais variadas áreas de atuação.

Quero expressar meu profundo agradecimento aos meus colegas de curso, que não foram apenas colegas, mas verdadeiros amigos para a vida. O companheirismo que construímos tornou a jornada do curso não apenas mais motivadora, mas também incrivelmente mais leve. Agradeço por cada momento compartilhado, por cada apoio mútuo e por transformarem essa experiência acadêmica em algo muito mais significativo e enriquecedor.

Em local de trabalho agradeço meu supervisor Marcelo pela oportunidade de ingressar no setor de software, ao co-supervisor Adelson pela transmissão de conhecimentos específicos que ajudaram a desenvolver o projeto, e agradeço também ao Henrique Menarin por disponibilizar ferramentas importantes. Em geral um agradecimento à todos os meus colegas de setor e pessoas envolvidas no meu dia a dia no trabalho.

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 12 de dezembro de 2023.

Na condição de representante da REIVAX Automação e Controle S/A na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

Documento assinado digitalmente
MARCELO SCHMIDT JACOBSEN
Data: 18/12/2023 13:08:07-0300
CPF: ***.002.129-**
Verifique as assinaturas em https://v.ufsc.br

Marcelo Schmidt Jacobsen
REIVAX Automação e Controle S/A

RESUMO

Este Projeto de Fim de Curso teve como objetivo desenvolver uma ferramenta de apoio para testes no desenvolvimento de software de Controladores Lógicos Programáveis para a empresa REIVAX, especializada em sistemas de automação industrial e controle de usinas de geração de energia. O projeto foi conduzido em resposta à necessidade de aprimorar o processo de criação, gerenciamento e execução de casos de teste, visando maior eficiência no desenvolvimento de software. A metodologia adotada incluiu a análise detalhada de requisitos, resultando na criação de uma aplicação de três camadas, compreendendo a camada de apresentação, a camada de lógica de negócios e a camada de persistência de dados. Essa arquitetura proporcionou uma base sólida para o desenvolvimento da ferramenta. A modelagem de dados envolveu a definição de entidades-chave, como casos de teste, configurações de sinais e execuções de testes, garantindo uma estrutura adequada para armazenar informações relevantes. A interface do usuário foi projetada com foco na usabilidade, incluindo um layout responsivo, navegação intuitiva e gráficos interativos para a visualização de dados. A ferramenta resultante atingiu com sucesso os objetivos estabelecidos, permitindo aos engenheiros de software da REIVAX criar e gerenciar casos de teste de forma eficiente, configurar sinais de comunicação, visualizar dados de teste e integrar-se ao software base da empresa. Este projeto contribuiu para aprimorar os processos de desenvolvimento de lógicas de controle, visando a eficiência operacional da empresa. Para o futuro, há oportunidades de expansão para outros setores, incluindo a integração de mais recursos e melhorias contínuas.

Palavras-chave: Automação. Teste. Sistemas. Controle. Software. Energia.

ABSTRACT

The aim of this End of Course Project was to develop a support tool for the software development of Programmable Logic Controllers for the company REIVAX, specialized in industrial automation systems and power generation control. The project was conducted in response to the need of enhancing the process of creating, managing, and executing test cases, aiming for greater efficiency in software development. The adopted methodology included a detailed analysis of requirements, resulting in the creation of a three-layered application, comprising the presentation layer, the business logic layer, and the data persistence layer. This architecture provided a solid foundation for the development of the tool. Data modeling involved defining key entities such as test cases, signal configurations, and test executions, ensuring a suitable structure for storing relevant information. The user interface was designed with a focus on usability, including a responsive layout, intuitive navigation, and interactive graphics for data visualization. The resulting tool successfully achieved the established objectives, allowing REIVAX's software engineers to efficiently create and manage test cases, configure communication signals, view test data, and integrate with the company's base software. This project contributed to enhanced software development processes at REIVAX, increasing the company's operational efficiency. For the future, there are opportunities for expansion into other sectors, including the integration of additional features and continuous improvements.

Keywords: Automation. Test. Systems. Control. Software. Energy.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A	ampère
Aneel	Agência Nacional de Energia Elétrica
API	*Application Programming Interface*
ASC	*Armored* ASCII *File*
ASCII	*American Standard Code for Information Interch*
ASGI	*Asynchronous Server Gateway Interface*
BD	Banco de Dados
CAF	*Control and Automation Framework*
CAN	*Controller Area Network*
CD	Entrega Contínua
CI	Integração Contínua
CLP	Controlador Lógico Programável
CPX	Controlador Programável REIVAX
CRUD	*Create, Read, Update, Delete*
CSS	*Cascading Style Sheets*
csv	*Comma-separated values*
EDO	Equação Diferencial Ordinária
FTP	*File Transfer Protocol*
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
Hz	hertz
IDE	*Integrated Development environment*
IEC	*International Electrotechnical Commission*
IEEE	*Institute of Electrical and Electronics Engineers*
IHM	Interface Homem-Máquina
IP	*Internet Protocol*
JS	JavaScript
JSON	JavaScript *Object Notation*
kV	quilovolt
LAN	*Local Area Network*
LED	*Light Emitting Diode*
MARC	Módulos de Aquisição, Registro e Controle
MB	megabyte
MIMO	*Multiple Input Multiple Output*
MVP	Produto Viável Mínimo
MW	megawatt

NEPA	Núcleo de Execução de Programas Aplicativos
NoSQL	*Not Only Structured Query Language*
ONS	Operador Nacional do Sistema Elétrico
OR	Registrador Modbus
OSI	*Open Systems Interconnection*
PA	Programa Aplicativo
PFC	Projeto de Fim de Curso
png	*Portable Network Graphic*
RGX	Registrador de Sinal REIVAX
RTVAX	Regulador Integrado de Tensão, Velocidade e Automação
RTVX	Regulador Integrado de Tensão e Velocidade
RTX	Regulador de Tensão
RVX	Regulador de Velocidade
SBE	Software Básico Embarcado
SCADA	*Supervisory Control and Data Acquisition*
SEC	Sistema de Edição de Configurações
SGBD	Sistema de Gerenciamento de Banco de Dados
SIN	Sistema Interligado Nacional
SISO	*Single Input Single Output*
SO	Sistema Operacional
STA	*Software Test App*
svg	*Scalable Vector Graphics*
TCP	*Transmission Control Protocol*
UFSC	Universidade Federal de Santa Catarina
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
USB	*Universal Serial Bus*
UTF-8	*8-bit Unicode Transformation Format*
XML	*eXtensible Markup Language*
XVI	xVision

SUMÁRIO

1 INTRODUÇÃO

A introdução deste Projeto de Fim de Curso (PFC) tem como propósito inicial apresentar o contexto do tema proposto, destacando sua relevância e as razões que motivaram a escolha deste assunto. São dados também os objetivos a serem alcançados com este projeto, apresentação da empresa, metodologia e a estrutura do documento.

1.1 CONTEXTO E MOTIVAÇÃO

A indústria de controle e automação desempenha um papel fundamental na eficiência operacional de sistemas complexos. A regulação de processos em usinas de geração de energia (Figura 1), desempenha um papel crítico na garantia da operação confiável e segura de regulação de tensão, velocidade e automação.

A garantia de uma operação confiável é regida pelo Operador Nacional do Sistema Elétrico (ONS), que é o órgão responsável pela coordenação e controle da operação das instalações de geração e transmissão de energia elétrica no Sistema Interligado Nacional (SIN) e pelo planejamento da operação dos sistemas isolados do país, sob a fiscalização e regulação da Agência Nacional de Energia Elétrica (Aneel) [17].

No cenário atual, o desenvolvimento de software de um sistema de controle para Controlador Lógico Programável (CLP) em projetos de controle e automação requer um alto nível de precisão, eficiência e colaboração entre as mais variadas formações em engenharias. O objetivo deste PFC é criar uma ferramenta de apoio abrangente para engenheiros de software de CLP, aprimorando significativamente o processo de desenvolvimento, teste e validação de software de controle e supervisão.

As pessoas cometem erros com frequência, e a motivação por trás da criação da ferramenta de apoio, com foco na automação de testes, surge da necessidade de padronização e da redução da dependência humana na detecção de erros em lógicas simples. Erros podem ser introduzidos na customização de um software a qualquer momento, e os testes subjetivos realizados pelos engenheiros podem não identificá-los prontamente, o que pode se tornar um problema no futuro, resultando em complicações em termos de tempo, recursos financeiros e desgaste de recursos humanos.

Atualmente, o setor de Engenharia de Software da empresa conta com testes manuais, mudanças de variáveis com base na intuição e experiência do engenheiro. Mirando em alterações do estado do sistema, a ferramenta deve ter a capacidade de executar testes de forma autônoma, amostrando dados do CLP e entregando o resultado na sequência, indicando o cumprimento ou não dos requisitos.

Figura 1 – Usina Hidrelétrica Binacional de Itaipu.

Fonte: ABADI [13]

1.2 OBJETIVOS DO PROJETO

Os principais objetivos deste projeto são os seguintes:

1. **Desenvolvimento de uma Ferramenta de Apoio**: Desenvolver uma aplicação local que permita aos engenheiros de software criar, gerenciar e executar casos de teste de software para CLPs de forma eficiente;
2. **Aprimoramento da Eficiência de Desenvolvimento**: Facilitar a colaboração entre engenheiros de software, comissionamento e assistência técnica, melhorando o fluxo de trabalho e a mitigação de problemas rapidamente;
3. **Visualização de Dados**: Fornecer recursos de visualização de dados, incluindo gráficos interativos, com análises e validação dos resultados dos testes.

Os objetivos específicos consistem em:

1. **Interface**: Desenvolvimento de uma interface interativa com foco na criação intuitiva de casos de teste;

2. **Gerenciamento de casos de teste**: O usuário será capaz de executar, criar, modificar, deletar, salvar, exportar e importar qualquer tipo de teste, sem restrições e de forma simples;

3. **Visualização de resultados**: Após a execução de um caso de teste, um relatório será gerado. Este relatório fornecerá indicadores dos sistemas de controle em cada etapa e incluirá um gráfico interativo que possibilita uma análise mais aprofundada;

4. **Comportamento esperado**: Ao criar um caso de teste, esperamos que o que foi programado seja realizado, mesmo que seja um resultado negativo. A ferramenta deve permitir que o usuário defina se algo aconteceu conforme o esperado ou não, e se o objetivo do teste foi alcançado ou não;

5. **Análises quantitativas e qualitativas**: A fim de evitar subjetividade, a ferramenta deve, com base nas informações fornecidas, apresentar resultados de forma quantitativa e qualitativa. Isso permitirá determinar se um evento ocorreu de forma positiva ou negativa e/ou fornecer resultados que indiquem se um sinal atingiu um valor *x* por cento acima do desejado;

6. **Gerenciamento de sinais**: A ferramenta terá a capacidade de obter informações do regulador de forma local ou remota por meio de um protocolo de comunicação industrial de CLP. Para isso, será necessário estabelecer um sistema de cadastro de endereçamento;

7. **Mitigação de erros simples**: Todos estão sujeitos a cometer erros, mesmo os mais simples. A finalidade da ferramenta é garantir que erros pequenos e facilmente detectáveis não passem despercebidos pelos engenheiros;

8. **Expansão e reutilização**: Durante o desenvolvimento da ferramenta, foi priorizada a criação de código legível e modular, permitindo que a ferramenta possa ser compreendida e aprimorada por qualquer pessoa. Além disso, foi considerada a possibilidade de expansão para outros setores e a aceitação geral para uso no dia a dia.

1.3 A EMPRESA

A REIVAX (Figura 2) é uma empresa de engenharia, com foco em controle e automação, de origem catarinense que tem se tornado uma referência mundial no fornecimento de sistemas e soluções para o controle da geração de energia. Fundada em abril de 1987 na cidade de Florianópolis, atualmente com sede no bairro João Paulo, a empresa tem se expandido globalmente, com escritórios no Canadá e na Suíça, atuando mundialmente.

Figura 2 – Logo da REIVAX.

Fonte: Portal Online REIVAX [23]

Os fundadores da empresa tiveram um importante papel no desenvolvimento de soluções de estabilização de oscilações eletromecânicas, que foram adotadas globalmente, e servem como base para a norma da *Institute of Electrical and Electronics Engineers* (IEEE) 421 [7].

Com mais de 35 anos de experiência, a REIVAX tem se destacado no mercado com seus sistemas de automação e supervisão de plantas renováveis, garantindo a eficiência e segurança na operação das usinas hidrelétricas, tornando-se uma opção confiável e inovadora para seus clientes em todo o mundo.

A REIVAX tem desenvolvido projetos em muitos países, sendo conhecida por sua capacidade de adaptar as soluções de acordo com as necessidades específicas de cada cliente e local de instalação. A empresa tem participado em projetos para diferentes fontes de geração, incluindo hidrelétricas, eólicas, solares e outras fontes de energia renovável.

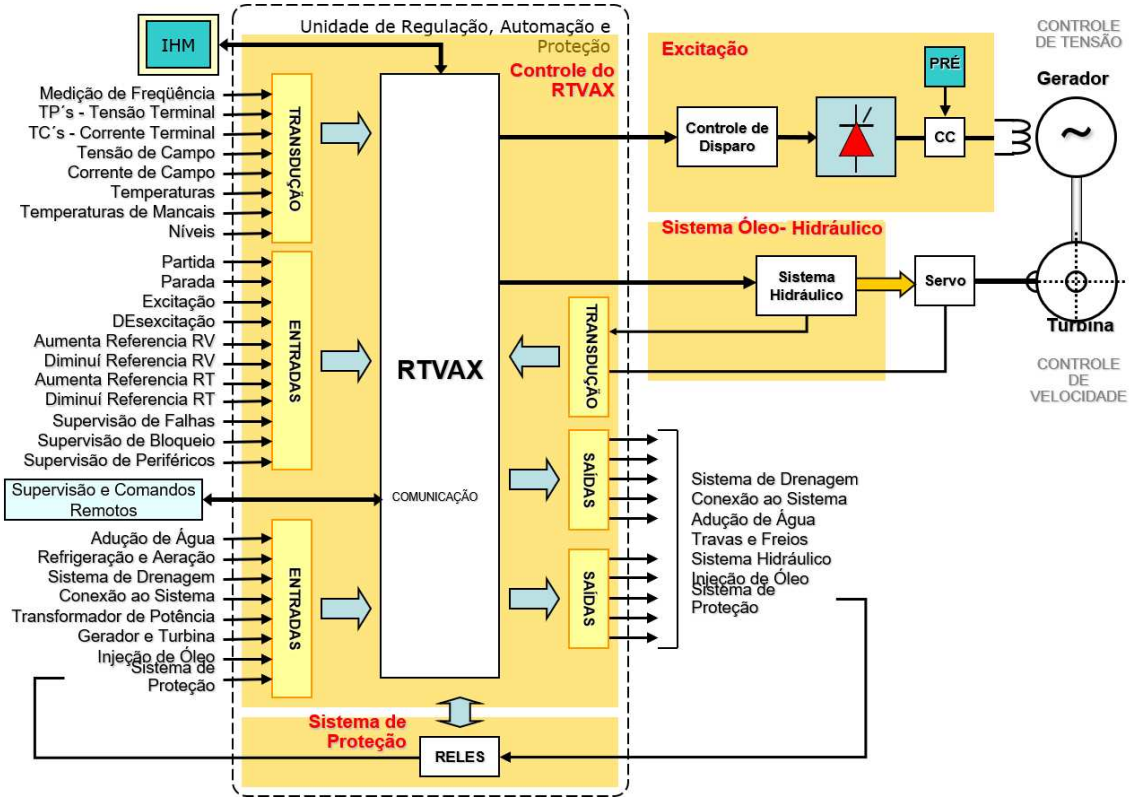
O principal equipamento é a linha POWER de Regulador Integrado de Tensão, Velocidade e Automação (RTVAX) (Figura 3), com as principais funcionalidades: regular a velocidade da turbina, tensão terminal do gerador e automação de partida e parada do conjunto turbina e gerador. Os principais softwares e hardwares utilizados pela REIVAX, são de propriedade dela, possuem características exclusivas, que dão liberdade e flexibilidade para os clientes operarem os sistemas de acordo com suas necessidades.

Figura 3 – Produto RTVAX POWER e IHM.

Fonte: Portal Online REIVAX [29]

Os processos realizados pelo setor de engenharia de software têm o objetivo de customizar sistemas de controle e supervisão, atuando em soluções personalizadas para atender às necessidades específicas de cada cliente. O esquemático do diagrama do software do RTVAX está ilustrado na Figura 4.

Figura 4 – Diagrama RTVAX.



Fonte: Portal Online REIVAX [22]

1.4 METODOLOGIA

A Metodologia Ágil surgiu como uma resposta às limitações dos métodos tradicionais de desenvolvimento de software. Os princípios da Metodologia Ágil foram primeiramente consolidados no Manifesto Ágil, que foi publicado por um grupo de desenvolvedores em 2001.

Os 12 princípios orientadores por trás do Manifesto Ágil enfatizam a entrega contínua de valor para o cliente, a capacidade de responder a mudanças, o desenvolvimento sustentável e a busca contínua por excelência técnica. Abaixo estão os princípios [2]:

1. Satisfação do cliente através da entrega contínua de software;
2. Abraçar mudanças de requisitos, mesmo em estágios tardios do desenvolvimento;
3. Entregar frequentemente software em funcionamento, com preferência para a menor escala de tempo possível;

4. Colaboração diária entre desenvolvedores e *stakeholders*[1] do negócio;
5. Construir projetos em torno de indivíduos motivados, fornecendo o ambiente e o suporte necessário, e confiar neles para fazer o trabalho;
6. O Método mais eficiente e eficaz de transmitir informações para e dentro de uma equipe de desenvolvimento é a conversa cara a cara;
7. Software em funcionamento é a principal medida de progresso;
8. Desenvolvimento ágil promove desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente;
9. Contínua atenção à excelência técnica e bom design, aumenta a agilidade;
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito;
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis;
12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

1.5 ESTRUTURA DO DOCUMENTO

A estrutura deste documento é organizada para garantir clareza e sistematicidade, permitindo ao leitor acompanhar o percurso do projeto de forma progressiva.

Segue-se a fundamentação teórica, oferecendo uma revisão da literatura que suporta o estudo realizado. O capítulo sobre automação de testes, sistemas de controle e desenvolvimento de software detalha as ferramentas e processos técnicos específicos que são relevantes para a criação da automação de testes.

Desenvolvimento de software esclarece as necessidades e expectativas que o software visa atender, e detalha o material de trabalho utilizado. As especificações funcionais e não funcionais são descritas para dar uma visão clara do que o software deve fazer e como deve operar.

O projeto de software é a seção que descreve a execução prática do que foi especificado anteriormente, incluindo a seleção de ferramentas, a arquitetura do software, o design da interface e a implementação do código. A seção de validação aborda os procedimentos para garantir que o software atenda aos requisitos estabelecidos.

O documento conclui com uma recapitulação do projeto, a análise dos objetivos alcançados e as lições aprendidas, seguidas das contribuições e do impacto do projeto.

[1] *Stakeholders* são os indivíduos e organizações impactados pelas ações da sua empresa.

2 FUNDAMENTAÇÃO TEÓRICA

As próximas seções compreendem a fundamentação teórica deste trabalho a fim de fornecer maior sustentação acadêmica e compreensão do estudo no contexto dos temas aqui abordados. Em particular, para o desenvolvimento deste PFC, têm-se as seguintes áreas:

1. **Automação de Testes**: Uma abordagem que busca melhorar a eficiência e eficácia do processo de teste em sistemas de software.
2. **Sistemas de Controle**: Estudo dos princípios de controle regulatório, com indicadores de eficiência em sistemas dinâmicos.
3. **Desenvolvimento de Software**: Investigação dos métodos, processos e práticas associadas à criação de soluções de software.

2.1 AUTOMAÇÃO DE TESTES

A automação de testes é uma ferramenta crucial no ciclo de vida do desenvolvimento de software. Ela é utilizada para acelerar o processo de teste, garantir a repetibilidade e expandir a cobertura de teste. No entanto, a automação de testes, como qualquer ferramenta ou método, tem suas limitações.

Benefícios e limitações da Automação de Testes

De acordo com Rafi et al. [21], a automação de testes oferece diversos benefícios significativos para o ciclo de desenvolvimento de software, incluindo:

- **Eficiência**: Os testes automatizados podem ser executados de forma rápida e repetitiva, economizando tempo e recursos em comparação com testes manuais;
- **Consistência**: A automação garante que os testes sejam executados da mesma maneira toda vez, eliminando variações introduzidas por testadores humanos;
- **Detecção precoce de problemas**: Testes automatizados podem ser integrados ao processo de integração contínua, permitindo a detecção precoce de problemas à medida que o código é desenvolvido;
- **Reutilização**: Os casos de teste automatizados podem ser reutilizados em diferentes etapas do desenvolvimento e em futuras iterações do software;
- **Cobertura abrangente**: A automação facilita a criação de testes abrangentes que cobrem uma ampla gama de cenários e funcionalidades do software.

Conforme exposto por Fewster e Graham [8], existem considerações importantes antes de automatizar testes. Primeiramente, a seleção de casos é crucial, pois nem todas as atividades de teste são apropriadas ou econômicas para automação. Testes raramente executados, com lógica muito volátil, ou que exigem interação física, por exemplo, geralmente não são bons candidatos para automação. A automação se adequa mais a testes que serão reexecutados com frequência.

Além disso, a automação deve ser vista como um aliado aos testes manuais, e não como um substituto. Um teste tem maior probabilidade de revelar um defeito na primeira vez em que é executado. Testes automatizados, frequentemente re-executados, podem ser menos eficazes na detecção de novos defeitos em comparação com os manuais.

Outro ponto é a forma de identificação de erros. Ferramentas de automação só podem identificar diferenças entre os resultados esperados e os reais. Portanto, é crucial assegurar a qualidade dos testes que serão automatizados. A revisão do *testware*[1] é fundamental para garantir essa qualidade.

A eficácia também é um aspecto a ser considerado. Automatizar um conjunto de testes não os torna automaticamente mais eficazes. Embora a automação possa melhorar a eficiência em termos de custo e tempo, isso não se traduz necessariamente em maior eficácia.

Os testes automatizados requerem manutenção contínua. Eles são mais suscetíveis a serem afetados por mudanças no software e podem se tornar obsoletos com facilidade. Isso implica que a necessidade de manutenção pode limitar as opções de modificação ou melhoria dos sistemas de software.

Por último, as ferramentas de automação carecem de imaginação. Elas seguem instruções predefinidas e não podem improvisar ou lidar com situações inesperadas como um ser humano. Testadores humanos têm a capacidade de adaptar e melhorar os testes durante sua execução e de lidar com eventos inesperados.

Maturidade em Automação de Testes

A maturidade na automação de testes é uma avaliação da eficácia com que uma organização pode executar e manter seus testes automatizados. Uma organização com alta maturidade em automação de testes tem práticas bem definidas, utiliza as melhores ferramentas disponíveis e é capaz de adaptar-se rapidamente às mudanças. Esta maturidade é frequentemente avaliada por melhores práticas padrão na literatura [36].

[1] *Testware* refere-se a todos os artefatos produzidos durante o processo de teste de um software. Isso inclui planos de teste, casos de teste, dados de teste, scripts de teste, ambientes de teste, resultados de teste e ferramentas de teste.

Automação de Testes no Contexto de Integração Contínua

Com a crescente popularidade da Integração Contínua (CI), a automação de testes tornou-se ainda mais crucial. CI refere-se à prática de integrar frequentemente mudanças de código em um sistema, e cada integração é verificada por meio de testes automatizados. Wang et al. [36] observaram que a habilidade da CI de entregar qualidade rapidamente depende de uma automação de testes confiável.

No estudo de Wang et al. [36], foi realizada uma pesquisa de maturidade em automação de testes e obtiveram-se respostas de 37 projetos open source em Java. Eles concluíram que níveis mais elevados de maturidade em automação de testes estão positivamente associados com uma maior qualidade do produto e um ciclo de lançamento mais curto.

Adotar melhores práticas é fundamental para alcançar a maturidade em automação de testes. Wang et al. [36] recomendam que os profissionais utilizem melhores práticas padrão para melhorar a maturidade em automação de testes. Algumas destas práticas incluem a seleção cuidadosa de ferramentas, treinamento adequado da equipe e CI eficiente.

A automação de testes é essencial para garantir a entrega de software de alta qualidade. A maturidade em automação de testes, avaliada por melhores práticas, pode levar a uma maior qualidade do produto e a ciclos de lançamento mais curtos, especialmente no contexto de CI.

Para garantir o sucesso na automação de testes, algumas melhores práticas devem ser seguidas:

- **Seleção adequada de casos de teste**: Escolher os casos de teste certos para automação é fundamental. Casos de teste repetitivos e críticos são candidatos ideais;
- **Manutenção constante**: Os casos de teste automatizados devem ser atualizados à medida que o software evolui para garantir que continuem sendo eficazes;
- **Integração com CI/Entrega Contínua (CD)**: A automação de testes deve ser integrada ao processo de CI/CD para testes automáticos regulares;
- **Relatórios e análises**: Gerar relatórios detalhados e analisar os resultados dos testes é essencial para identificar problemas e melhorias.

A automação de testes deve entregar a garantia da qualidade do software e na aceleração do ciclo de desenvolvimento. Ao adotar as melhores práticas e ferramentas adequadas, as organizações podem colher os benefícios da automação.

2.2 SISTEMAS DE CONTROLE

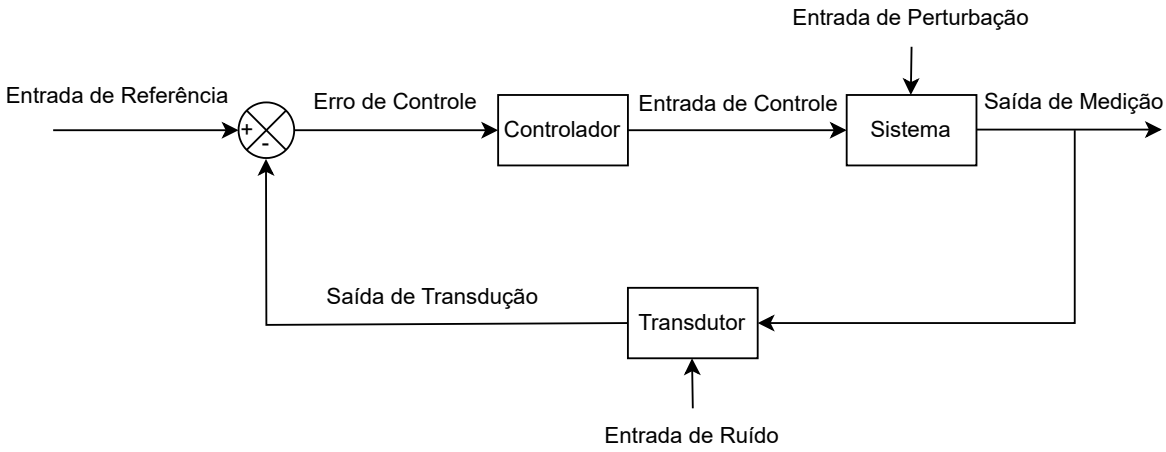
Os sistemas de controle desempenham um papel fundamental em uma ampla variedade de aplicações, desde processos industriais e automação até sistemas de engenharia civil e aeronáutica. Eles são projetados para regular o comportamento de sistemas dinâmicos, garantindo que operem dentro dos parâmetros desejados e atendam a metas específicas. A eficiência e a eficácia desses sistemas são frequentemente avaliadas por meio de indicadores de desempenho. Esta seção apresentará detalhes do livro “Feedback Control of Computing Systems” por Hellerstein et al. [12].

A Figura 5 é um exemplo de um sistema de controle *Single Input Single Output* (SISO), que tem uma única entrada de controle e uma única saída medida. Comumente nos sistemas de excitação, que é o foco do trabalho, lida-se com sistemas *Multiple Input Multiple Output* (MIMO), que têm múltiplas entradas de controle e múltiplas saídas medidas. Os elementos essenciais do sistema de controle por realimentação estão retratados na Figura 5. Esses elementos são:

- **Erro de controle**: É a diferença entre a entrada de referência e a saída medida;
- **Entrada de controle**: Sinal que pode ser ajustado por meio de parâmetros do controlador;
- **Controlador**: Dispositivo que contém as funções que determinam o sinal de controle necessário para alcançar a entrada de referência;
- **Perturbação**: Qualquer mudança que afeta o processo, definida por outras variáveis que influenciam o sistema de controle;
- **Saída medida**: Uma característica/variável mensurável do sistema alvo;
- **Ruído**: O sinal que interfere na saída medida pelo sensor;
- **Entrada de referência**: O valor desejado das saídas medidas;
- **Sistema alvo**: O sistema/processo a ser controlado;
- **Transdutor**: transforma a saída medida para que possa ser comparada com a entrada de referência.

O fluxo circular de informação na Figura 5 motiva o uso do termo sistema de malha fechada para se referir a um sistema de controle por realimentação (*feedback*).

Figura 5 – Diagrama de blocos de um sistema de controle com realimentação. A entrada de referência é o valor desejado da saída medida do sistema. O controlador ajusta a configuração da entrada de controle para o sistema alvo de modo que sua saída medida seja igual à entrada de referência. O transdutor representa efeitos como conversões de unidade e atrasos.



Fonte: Traduzido e adaptado de Hellerstein et al. [12]

Objetivos de Controle

Os controladores são projetados com um propósito específico, conhecido como o objetivo de controle. Os objetivos mais comuns incluem várias funções chave. Primeiramente, o **controle regulatório**, que tem como objetivo manter ou regular uma variável processada em um valor ou estado desejado. Essa função é essencial para garantir a estabilidade e a consistência no desempenho do sistema.

Há também a **rejeição de perturbações**, que foca em assegurar que perturbações externas ou internas que atuam sobre o sistema não afetem significativamente a saída medida, garantindo o funcionamento.

Por último, o objetivo de **otimização** busca alcançar o “melhor” valor possível da saída medida. Esse objetivo se concentra em melhorar a eficiência e a eficácia do sistema, assegurando que ele opere no seu ponto ótimo em termos de desempenho e utilização de recursos.

Propriedades dos Sistemas de Controle por realimentação

Existem várias propriedades dos sistemas de controle por realimentação que devem ser consideradas. Abaixo, são apresentadas as ideias principais das propriedades consideradas:

- Um sistema é dito estável se, para qualquer entrada limitada, a saída também é

limitada;

- O sistema de controle é preciso se a saída medida converge para a entrada de referência;
- O sistema tem tempos de assentamento curtos se converge rapidamente para seu valor em regime permanente;
- O sistema deve atingir seus objetivos de maneira que não tenha sobressinal.

A robustez em refere-se à capacidade de um sistema manter seu desempenho desejado mesmo diante de incertezas, variações nos parâmetros do sistema, perturbações externas e outras condições adversas.

Indicadores de desempenho

Os sistemas de primeira ordem são caracterizados por terem apenas um constante de tempo [14]. Eles são descritos matematicamente por uma equação diferencial ordinária de primeira ordem:

$$\tau \frac{dy(t)}{dt} + y(t) = Ku(t) \tag{1}$$

onde τ é a constante de tempo e K é o ganho do sistema. Alguns exemplos: Circuitos Resistor-Capacitor sem fonte, tanque de líquido, resposta térmica de um objeto ao ser colocado em um ambiente com temperatura diferente.

Os sistemas de segunda ordem têm duas constantes de tempo e são descritos por uma Equação Diferencial Ordinária (EDO) de segunda ordem:

$$\tau^2 \frac{d^2y(t)}{dt^2} + 2\xi\tau \frac{dy(t)}{dt} + y(t) = Ku(t) \tag{2}$$

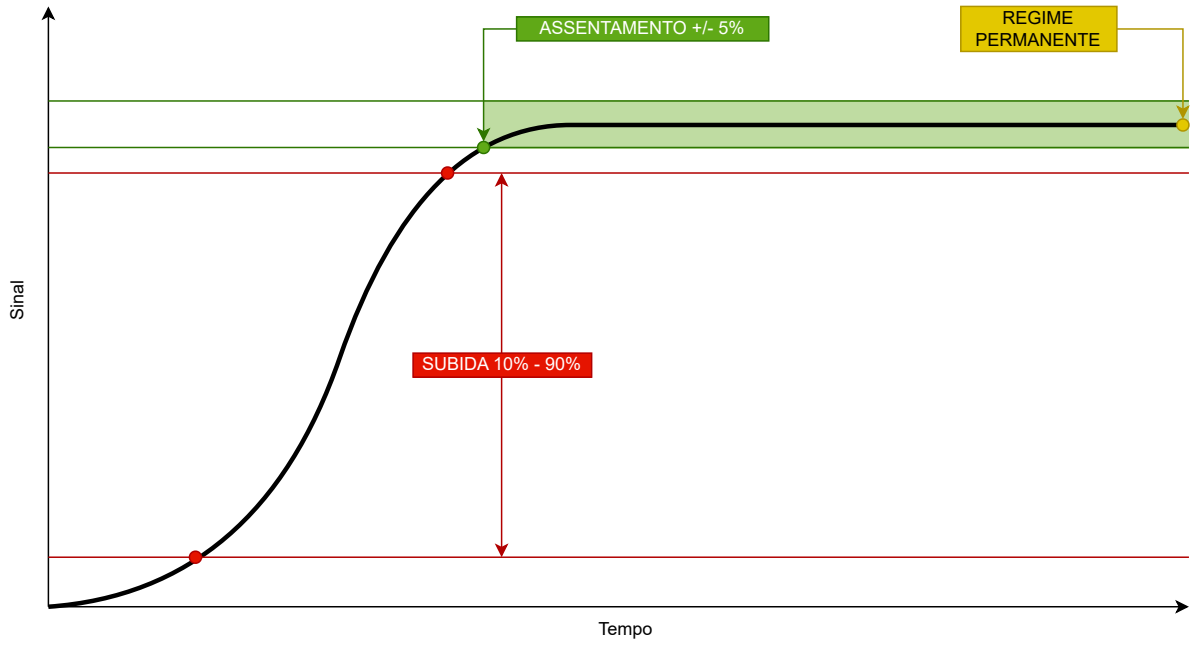
onde ξ é o coeficiente de amortecimento e K é o ganho do sistema. Exemplos: Circuitos Resistor-Indutor-Capacitor, oscilador harmônico amortecido e pêndulos.

Na Figura 6 e Figura 7, são apresentados os gráficos simplificados dos sistemas de primeira e múltiplas ordens, respectivamente, com os seguintes indicadores de desempenho:

- **Tempo de Subida (*Rise Time*)**: O tempo necessário para a resposta do sistema passar de 10% a 90% (ou 0% a 100% dependendo da literatura) de seu valor final em regime permanente. Indica quão rápido o sistema responde a uma entrada;
- **Tempo de Assentamento (*Settling Time*)**: O tempo necessário para a resposta do sistema se estabilizar dentro de uma faixa específica em torno do valor final em regime permanente (geralmente, 2% ou 5% do valor final dependendo da literatura);

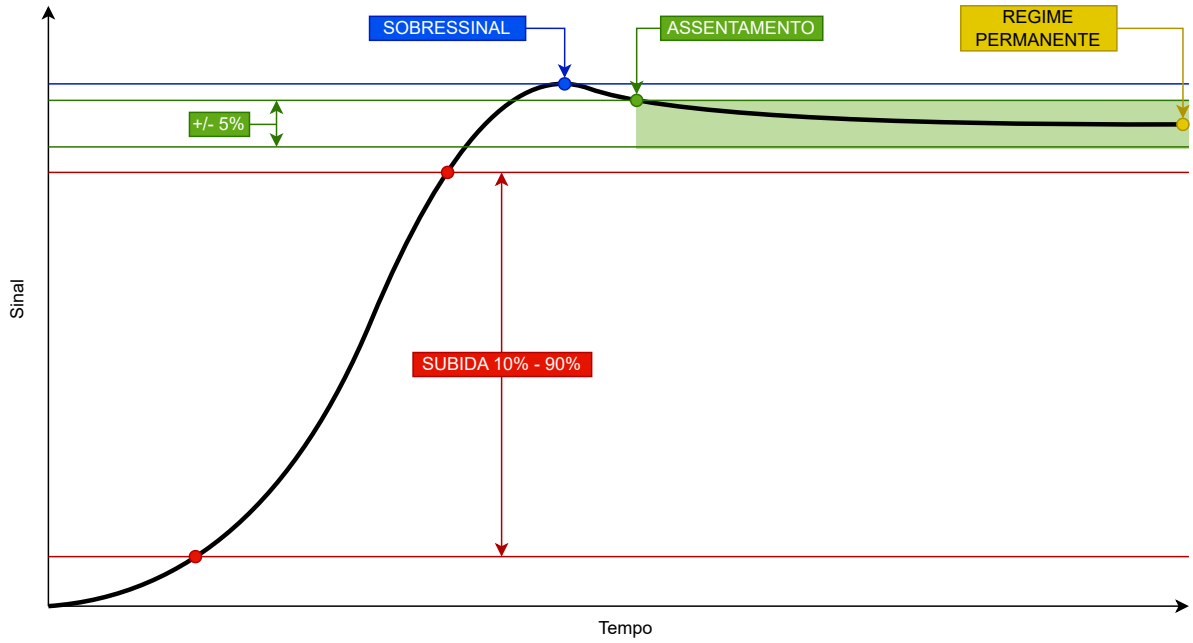
- **Sobressinal (*Overshoot*)**: É a quantidade pela qual a resposta do sistema excede seu valor final. É geralmente expresso como uma porcentagem. Em sistemas de primeira ordem não há sobressinal;
- **Erro em Regime Permanente (*Steady-State Error*)**: É a diferença entre o valor final da saída e o valor desejado em regime permanente.

Figura 6 – Sistema de primeira ordem com indicadores destacados.



Fonte: Autoria própria.

Figura 7 – Sistema de duas ou mais ordens com indicadores destacados.

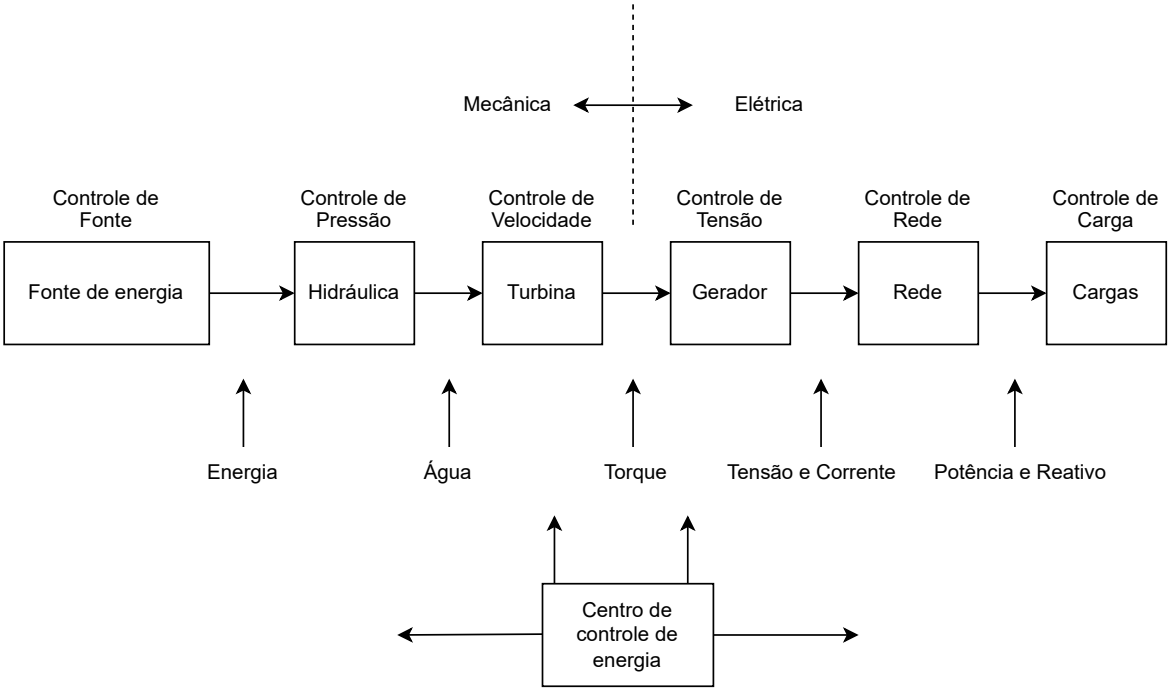


Fonte: Autoria própria.

Sistemas de controle em geração de energia

O sistema de controle em geração de energia (escopo do trabalho) é composto por várias etapas que trabalham em conjunto para converter energia de uma fonte primária, como a energia potencial da água, em energia elétrica utilizável.

Figura 8 – Estrutura do sistema dinâmico de geração de energia de fonte hídrica.



Fonte: Traduzido e adaptado de Sauer et al. [31]

Conforme disposto na Figura 8, o diagrama apresenta um fluxo de sistema dinâmico, em que:

1. **Fonte de Energia**: A fonte de energia é o ponto inicial do sistema. É responsável por fornecer a energia bruta que será convertida em energia elétrica;
2. **Hidráulica**: A Unidade Hidráulica é usada para controlar a disponibilidade de água, e o controle de pressão garante que essa água seja movimentada na pressão adequada para a operação eficiente da turbina;
3. **Turbina**: A água movimenta a turbina, convertendo energia hidráulica em energia mecânica. O controle de velocidade garante que a turbina gire na velocidade ideal para ser entregue à rede (por exemplo em 60 Hertz em território brasileiro);
4. **Gerador**: A energia mecânica da turbina é então convertida em energia elétrica pelo gerador. O controle de tensão assegura que a eletricidade gerada tenha a tensão correta.

5. **Rede**: A eletricidade produzida é então distribuída através da rede elétrica. O controle de rede garante que a distribuição seja feita (por exemplo em 220 Volts em território catarinense);
6. **Cargas**: Finalmente, a eletricidade é entregue às cargas, que podem ser residências, empresas ou outros consumidores. O controle de carga ou potência garante que a distribuição da energia seja feita de acordo com a demanda.

2.3 DESENVOLVIMENTO DE SOFTWARE

O desenvolvimento de software é um campo essencial na engenharia de controle e automação, onde a criação de programas de computador são importantes na implementação de sistemas automatizados. Nesta seção, exploraremos conceitos-chave relacionados ao desenvolvimento de software com base no livro “Fundamentos da engenharia de software” [35].

Ciclo de Desenvolvimento de Software

O ciclo de desenvolvimento de software é o processo pelo qual um sistema de software é concebido, projetado, implementado, testado, entregue e mantido. Existem várias metodologias de desenvolvimento de software.

O método **cascata** representa um modelo linear e sequencial, onde cada fase do desenvolvimento é concluída antes de se avançar para a próxima etapa. Esse modelo é caracterizado por sua natureza estruturada e previsível.

Em contraste, o método **iterativo e incremental** divide o desenvolvimento em várias iterações. Cada iteração adiciona funcionalidades incrementais ao produto, permitindo ajustes constantes e uma abordagem mais flexível em comparação ao modelo Cascata.

Por fim, a metodologia **ágil** oferece uma abordagem ainda mais flexível. Utilizada para este projeto, ela enfatiza a colaboração entre as equipes, a adaptabilidade às mudanças e a entrega contínua de partes do projeto. Este método é ideal para projetos dinâmicos, onde os requisitos e soluções evoluem através da colaboração entre equipes multifuncionais.

Requisitos Funcionais

Requisitos funcionais são especificações que definem o que um sistema ou aplicação deve fazer. Eles descrevem as funções ou tarefas específicas que o sistema é capaz de realizar. Estes requisitos são geralmente descritos em termos de entradas, processos e saídas esperadas do sistema.

O autor comenta como exemplo, o desenvolvimento de um programa de ordenação de arquivos, ele enfatiza vários fatores críticos devem ser considerados para

garantir a eficiência e a precisão do processo. Primeiramente, é essencial definir o formato de entrada do arquivo. Precisa-se saber se estamos lidando com um arquivo de texto simples ou um que possui uma estrutura mais específica, pois isso influenciará a forma como o arquivo será processado.

Compreender o tamanho esperado do arquivo é crucial, pois isso pode impactar diretamente a escolha do algoritmo e da estrutura de dados mais apropriados para a ordenação.

Outra consideração fundamental é o tratamento de erros. É vital determinar como o programa deve reagir ao se deparar com arquivos corrompidos ou mal formatados. Esse cuidado é essencial para garantir a robustez e a confiabilidade do programa.

Além disso, define-se o formato de saída do arquivo. Deve-se considerar se o arquivo de saída deve manter o formato original ou se há necessidade de alterações. A questão das duplicatas também é relevante, pois o programa precisa de instruções claras sobre como lidar com linhas duplicadas - se devem ser removidas ou mantidas.

A codificação de caracteres deve ser pensada. A escolha entre *8-bit Unicode Transformation Format* (UTF-8), *American Standard Code for Information Interch* (ASCII) ou outra codificação afetará como o programa lê e escreve os arquivos, influenciando a compatibilidade e a usabilidade do software.

Requisitos Não Funcionais

Requisitos não funcionais, muitas vezes referidos como “atributos de qualidade” ou “restrições”, definem as características ou qualidades que um sistema deve possuir, ao invés de funções específicas que ele deve realizar (que são os requisitos funcionais). Esses requisitos garantem a viabilidade, usabilidade, eficácia e satisfação do usuário em relação a um sistema ou aplicação.

Na concepção de um programa de ordenação de arquivos, o autor discute diversos fatores que definem o seu sucesso e eficácia. O primeiro aspecto a ser considerado é o desempenho. É preciso determinar a rapidez com que o programa deve ordenar os arquivos e se existem restrições específicas de tempo que devem ser atendidas. Além disso, a usabilidade do programa é outro ponto importante. Precisa-se decidir se o programa terá uma interface gráfica para facilitar o uso pelos usuários ou se será uma ferramenta mais técnica de linha de comando, destinada a usuários com maior conhecimento técnico.

É essencial que o programa seja capaz de manter sua eficiência mesmo se os tamanhos dos arquivos a serem ordenados aumentarem no futuro. Isso garante que o software permaneça útil e eficiente à medida que as demandas aumentam. Além disso, a portabilidade do programa define a capacidade de funcionar em diferentes sistemas operacionais aumenta sua utilidade e alcance. O software deve ser estruturado de

maneira que permita modificações e atualizações fáceis, assegurando sua longevidade e adaptabilidade às mudanças futuras.

Restrições de Design

1. **Linguagem de Programação**: O cliente especificou uma linguagem de programação em particular ou o desenvolvedor é livre para escolher?
2. **Plataforma**: Em quais sistemas operacionais isso deve ser executado? Linux, Windows, MacOS ou todos eles?
3. **Dependências**: O programa deve evitar o uso de bibliotecas de terceiros ou elas são permitidas?

Respondendo a essas perguntas, os desenvolvedores podem começar a compor uma compreensão mais pontual da tarefa em questão. Eles poderão escolher os algoritmos, estruturas de dados e ferramentas mais apropriados para o trabalho.

Design e Implementação

Com base nos requisitos e restrições esclarecidos, o desenvolvedor pode começar a esboçar uma solução. Seguindo o exemplo dado pelo autor, se os tamanhos dos arquivos forem grandes, o desenvolvedor pode escolher um algoritmo de ordenação mais eficiente, como o *merge sort*[2] ou *quicksort*[3], em vez de algoritmos mais simples, mas mais lentos, como o *bubble sort*[4].

O desenvolvedor também precisa garantir que o programa atenda aos requisitos de usabilidade, por exemplo, incluindo mensagens de erro úteis ou fornecendo uma interface de usuário simples e intuitiva.

Teste

Uma vez que o programa é implementado, testes são as próximas etapas. Para esta tarefa, arquivos de entrada de tamanhos e formatos variados podem ser usados para garantir que o programa os ordene corretamente. Casos extremos, como arquivos vazios ou arquivos com apenas uma linha, também devem ser testados.

[2] O *merge sort* é um algoritmo de ordenação que utiliza a abordagem de dividir para conquistar. Ele divide a lista em partes cada vez menores, ordena-as e depois as combina.

[3] O *quicksort* é um algoritmo de ordenação muito rápido e eficiente, que escolhe um elemento como pivô e particiona os elementos restantes em dois subconjuntos, que são então ordenados independentemente.

[4] O *bubble sort* é um dos algoritmos de ordenação mais simples, mas também um dos menos eficientes, especialmente para conjuntos de dados grandes. Ele compara pares adjacentes e os troca se estiverem na ordem errada.

Estimativa de Esforço

A estimativa de esforço é sobre prever quanto tempo e recursos serão necessários para concluir a tarefa. Dados os requisitos esclarecidos e uma clara compreensão do problema, o desenvolvedor pode dividir a tarefa em sub-tarefas menores e estimar o tempo para cada uma. Isso também ajudará a definir expectativas com o cliente em relação aos prazos de entrega.

Para resumir, mesmo um problema simples pode ter camadas de complexidade quando visto sob a ótica da engenharia de software. Dedicar tempo para entender e esclarecer essas complexidades garante um produto final melhor. Este capítulo destacou que o bom desenvolvimento de software é tanto sobre entender o problema quanto sobre codificar a solução.

Restrições de Design

Todo projeto de software vem com seu próprio conjunto de restrições. Essas restrições podem vir do ambiente em que o software deve ser executado, das ferramentas disponíveis para desenvolver, ou até mesmo das partes interessadas envolvidas.

- **Restrições de Hardware e Plataforma**: Em que tipo de máquinas o programa será executado? É um computador pessoal, um *mainframe*[5], um dispositivo móvel ou um hardware específico? Para o problema de ordenação exemplificado anteriormente, supõe-se de que o programa seja executado em computadores pessoais padrão com um Sistema Operacional (SO) regular;
- **Restrições do Ambiente de Desenvolvimento**: Quais são as ferramentas, bibliotecas e frameworks disponíveis? Há uma *Integrated Development environment* (IDE) ou compilador específico que precisa ser usado? Supõe-se, para o problema de ordenação, que estamos limitados a usar Java como linguagem de programação e Eclipse como ambiente de desenvolvimento;
- **Restrições de Integração**: O software precisa interagir com outros sistemas? Como ele se comunicará com esses sistemas? Isso pode não ser relevante para o exemplo de problema de ordenação, mas em um contexto mais amplo, talvez ele deva se integrar a um sistema de gerenciamento de arquivos ou a um Banco de Dados (BD);
- **Restrições Regulatórias e de Conformidade**: Existem padrões ou regulamentações que o software deve cumprir? Isso é comum em campos como saúde ou finanças.

[5] *Mainframe* é um supercomputador projetado para processar grandes quantidades de dados em um curto período de tempo

Decisões de Design

Uma vez que os requisitos (tanto funcionais quanto não funcionais) e as restrições foram estabelecidos, a fase de design começa. Aqui, muitas decisões precisam ser tomadas sobre como construir o software de fato.

- **Escolha do Algoritmo**: Existem vários algoritmos disponíveis para ordenação, e cada um tem suas vantagens e desvantagens. Qual deles se encaixa melhor considerando os requisitos e restrições? *QuickSort* pode ser rápido, mas é a escolha certa se tiver uma restrição em tempo real?
- **Estruturas de Dados**: Como as linhas de texto serão armazenadas na memória? Talvez como um array[6], uma lista encadeada ou alguma outra estrutura de dados. Esta decisão pode impactar o desempenho;
- **Tratamento de Erros**: O que deve acontecer quando há um erro? O programa deve sair, ou deve exibir uma mensagem de erro e permitir que o usuário a corrija? Ou talvez registrar o erro e continuar?
- **Interface do Usuário**: Mesmo para um programa de ordenação simples, há decisões a serem tomadas. Deve ser uma interface de linha de comando ou uma gráfica? Como o usuário deve especificar os arquivos de entrada e saída?
- **Extensibilidade**: Se houver a possibilidade de que o programa seja expandido ou modificado no futuro, como ele pode ser projetado de forma a facilitar essas mudanças?
- **Manutenibilidade**: Como o código será estruturado? Uma boa organização do código pode tornar o software mais fácil de entender, depurar e modificar no futuro;
- **Testes**: É essencial decidir desde cedo como o software será testado. Quais partes serão testadas unitariamente? Haverá testes de integração? Como a funcionalidade pode ser verificada?
- **Documentação**: Decisões precisam ser tomadas sobre que tipo de documentação acompanhará o software. Isso pode variar desde comentários no código, até manuais do usuário, até documentação técnica.

[6] Arrays são objetos semelhantes a listas que vêm com uma série de métodos embutidos para realizar operações de travessia e mutação.

3 ANÁLISE DE REQUISITOS

A fase de análise de requisitos é um dos estágios mais necessários no desenvolvimento de software, pois estabelece a base para todo o projeto. Neste capítulo, são apresentados os resultados do processo de levantamento de necessidades e especificação de requisitos para a ferramenta de apoio ao desenvolvimento de software de CLP na REIVAX. Esta análise é essencial para garantir que a aplicação atenda de maneira satisfatória às necessidades dos engenheiros de software.

3.1 LEVANTAMENTO DE NECESSIDADES

O levantamento de necessidades foi realizado por meio de reuniões com colegas de setor. Durante essas interações, foram identificadas as principais necessidades e desafios enfrentados no processo de testes da customização do software do CLP. Alguns dos principais pontos levantados incluem:

1. **Gestão de Casos de Teste**: Os usuários precisam de uma maneira eficaz de criar, organizar e gerenciar casos de teste para diferentes projetos. Isso inclui a capacidade de importar/exportar casos, salvar versões, criar novos casos e editar casos existentes;

2. **Configuração de Sinais**: A ferramenta deve permitir a configuração de sinais de comunicação. Os usuários precisam ser capazes de definir parâmetros de comunicação, como endereços de dispositivos e tipos de dados;

3. **Visualização de Dados**: A visualização de sinais analógicos e digitais de saída é fundamental para a validação dos casos de teste. Os usuários desejam a capacidade de monitorar esses sinais por meio de gráficos interativos, para certificar-se visualmente pela ferramenta de que os sinais de entrada e saída estejam da forma desejada;

4. **Análise quantitativa e qualitativa**: Junto à visualização de dados, a análise qualitativa e quantitativa são necessárias, referem-se a duas abordagens distintas de avaliação e interpretação de dados. A análise qualitativa foca nas características não numéricas, buscando compreender aspectos subjacentes, padrões, temas e contextos, muitas vezes por meio de observações. Por outro lado, a análise quantitativa envolve a avaliação de dados numéricos, utilizando estatísticas e modelos matemáticos, fornecendo resultados objetivos e mensuráveis;

5. **Integração com o SO**: Como a REIVAX possui um SO do CLP próprio, a ferramenta deve ser capaz de se comunicar e controlar o software de controle da forma mais eficiente. Isso permite a criação de novos casos de teste de forma

mais dinâmica, sem necessitar a configuração de sinais, isso porque o tempo de latência é muito menor que os protocolos de comunicação industriais.

3.2 MATERIAL DE TRABALHO

A REIVAX se destaca ao oferecer uma vasta gama de ferramentas, proporcionando inúmeras soluções que podem ser combinadas de diversas maneiras.

A empresa fornece uma variedade de componentes físicos para aprimorar sistemas de controle e automação. A maioria destes componentes são hardwares proprietários dela, garantindo aos clientes ampla liberdade e potencial de customização alinhada às suas necessidades específicas.

Além dos componentes físicos, a REIVAX está comprometida com o desenvolvimento contínuo de softwares que se integram aos CLPs. Os programas criados pela empresa são projetados para funcionar em harmonia com seus próprios hardwares.

CPX

O CPX (Figura 9) é um CLP proprietário da REIVAX baseado em x86, desenvolvido para executar lógicas de automação e controle para regulação de tensão e velocidade. O controlador contém uma interface de habilitação e falha, garantindo confiabilidade para o sistema de controle da usina [26].

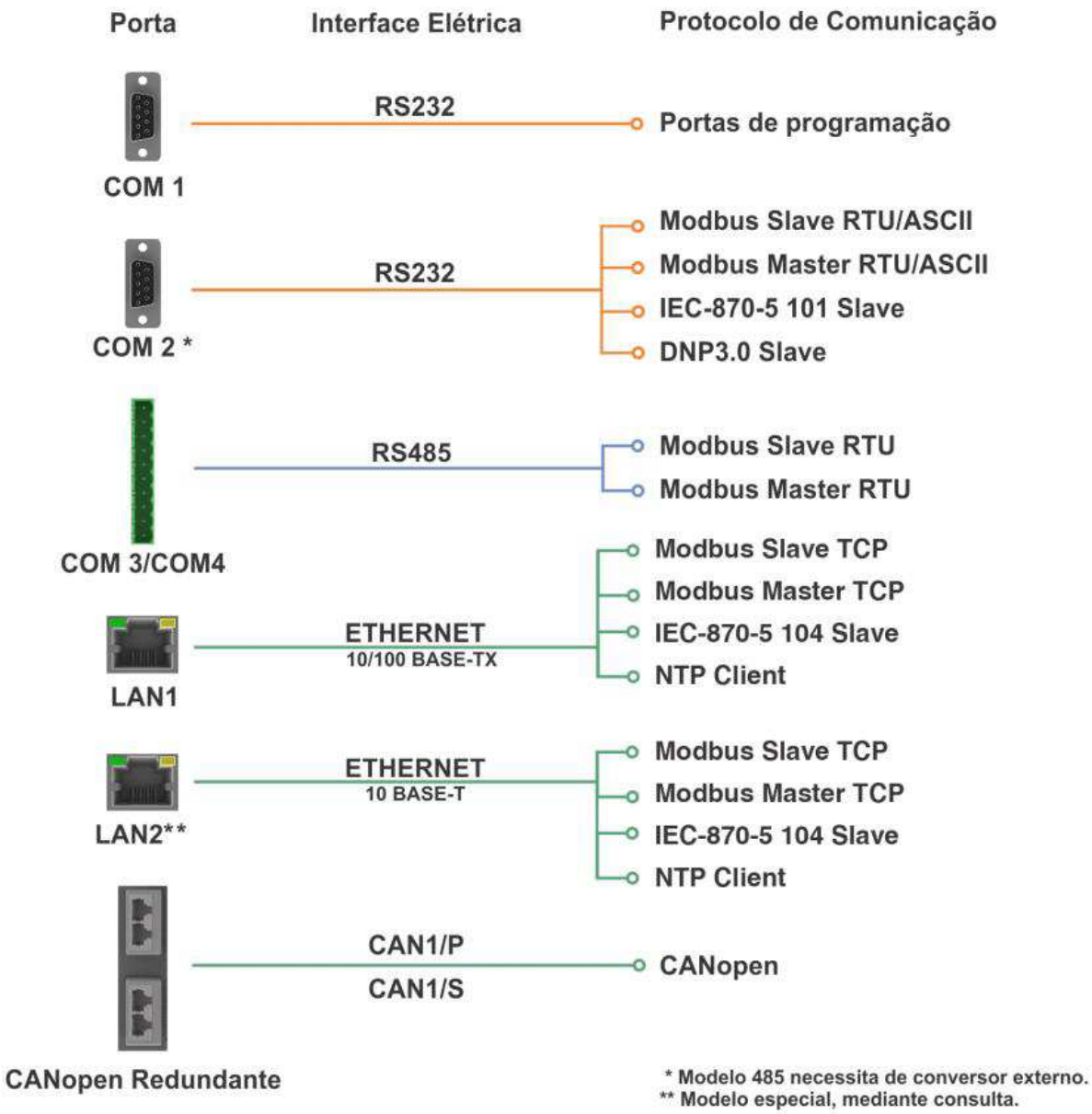
Figura 9 – Equipamento CPX.

Fonte: Manual do usuário CPX05 [26].

A Figura 10 apresenta as portas disponíveis no CPX. Contendo uma variedade

de portas de comunicação e suas respectivas interfaces elétricas, juntamente com os protocolos de comunicação suportados por cada uma. Pode-se perceber que o protocolo Modbus *Transmission Control Protocol* (TCP)/*Internet Protocol* (IP), requerido ao projeto, é disponibilizado pela duas portas *Local Area Network* (LAN).

Figura 10 – Portas do CPX.



Fonte: Manual do usuário CPX05 [26].

O CPX fornece interface CANopen, um protocolo padrão para automação em sistemas distribuídos, baseado na rede *Controller Area Network* (CAN). Este padrão oferece uma série de características de desempenho voltadas para a transmissão de dados em ambientes onde a temporização e a sincronização são críticas [5].

O padrão RS-485 é amplamente reconhecido por sua versatilidade e eficiência na conexão de equipamentos de terminal de dados sem a necessidade de modems. O RS-232 é um padrão mais antigo e menos eficiente que o RS-485, que se destaca por sua capacidade de conectar diretamente com equipamentos 80 vezes mais distantes com múltiplos receptores e transmissores, proporcionando uma comunicação aprimorada e simplificada em diversos ambientes industriais e comerciais [33].

Ethernet é um padrão de camada física e camada de enlace que opera de forma síncrona em 10/100 Mbps, com quadros que possuem tamanho variável entre 64 e 1518 bytes. Originalmente foi criado para operar numa topologia em barramento, onde todos os dispositivos recebem todos os pacotes transmitidos [10].

A Interface Homem-Máquina (IHM) local normalmente serve para parametrização da aplicação e leitura de valores do processo controlado, entre outras funções. O comportamento é partir de *Light Emitting Diode* (LED), que sinaliza o que está subordinado à aplicação embarcada no controlador, devendo ser consultada na documentação específica.

Na Tabela 1, estão representadas as características físicas do CPX.

Tabela 1 – Interfaces de Entradas e Saídas

VISTA	**DESCRIÇÃO**
Vista inferior	CAN Redundante (Principal e Secundário) Interface RS-485 (485-1 e 485-2) Interface IRIG-B
Vista frontal	Interfaces RS-232 (Porta COM1) IHM Reset *Compact flash*
Vista superior	Alimentação Entrada de habilitação Entrada de Seleção CPX 1/2 Interface de Falha Crítica Saída de Falha Interfaces RS-232 (Porta COM2) Interface Ethernet (LAN1 e LAN2)

Software Básico Embarcado (SBE)

O SBE é o SO residente nos Módulos de Aquisição, Registro e Controle (MARC). É responsável pela execução das configurações fornecidas pelo usuário através dos arquivos de configuração [30].

As principais características do SBE são:

- Execução em modo protegido, 32 bits, em plataforma de hardware baseada na família de processadores Intel x86 32 bits ou compatível;

- Desempenho para aplicações de tempo real;
- Multitarefa, com escalonamento preemptivo baseado em prioridades;
- Conectividade em rede TCP/IP;
- Sistema de arquivo MS-DOS compatível, suportando FAT12, FAT16 e FAT32;
- Suporte discos IDE (formato CHS e LBA);
- Suporte discos SATA (legado IDE e AHCI), a partir do SBE de versão V700R001;
- Floppy disk, DiskOnChip e RAM disk.

SEC

O SEC é desenvolvido para ambiente Windows com o objetivo de gerar um conjunto de arquivos de configurações do SBE. Esses arquivos são utilizados para configurar diversas funções do SBE, incluindo o NEPA, o MARC, o Núcleo de Registro de Sinais, a Fila de Registros, o Núcleo de Registro de Eventos, o Servidor de *File Transfer Protocol* (FTP) e outras [27].

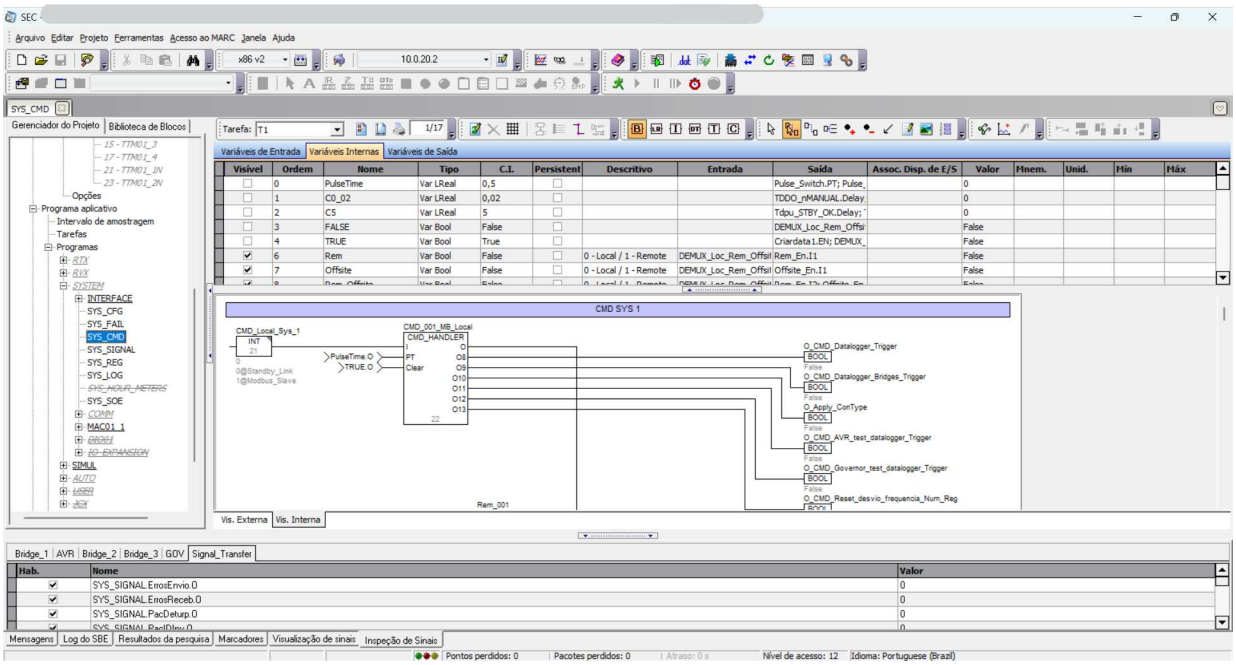
Na Figura 11, é possível ter uma visão geral do software. Os componentes mais importantes são listados abaixo.

- **Barras superiores**: Configurações gerais do editor, ferramentas de edição, simulador, janela de testes, selecionador de IP, embarque de CPX, entre outros;
- **Barra inferior**: Visualizador e inspetor de sinais, log, mensagens de compilação;
- **Barra esquerda**: Navegação e configurações dos programas, um programa é um conjunto lógicas organizadas para o seu nome descritivo, nas próximas subseções será feita a apresentação dos nomes dos programas existentes. Biblioteca de blocos, um auxiliador para buscar todas as portas lógicas existentes para programação;
- **Visão central**: Onde é possível visualizar a lógica do programa selecionado (na imagem o SYS_CMD); nele há variáveis de entrada, saída, interna e blocos lógicos.

Além disso, o SEC possui funções de visualização e ajuste de valores online, o que o torna útil como ferramenta de depuração de Programa Aplicativo (PA). Ele é uma ferramenta de desenvolvimento versátil que pode ser usada tanto para a criação de novos programas quanto para a manutenção dos existentes.

O SEC requer uma chave de proteção (Chave de Hardware) conectada a uma porta *Universal Serial Bus* (USB) do computador para funcionar corretamente. Sem

Figura 11 – Visão geral do SEC.



Fonte: Arquivo pessoal.

esta chave, apenas as funções básicas de embarque e manipulação de arquivos do CPX estarão disponíveis, impedindo a abertura de projetos completos.

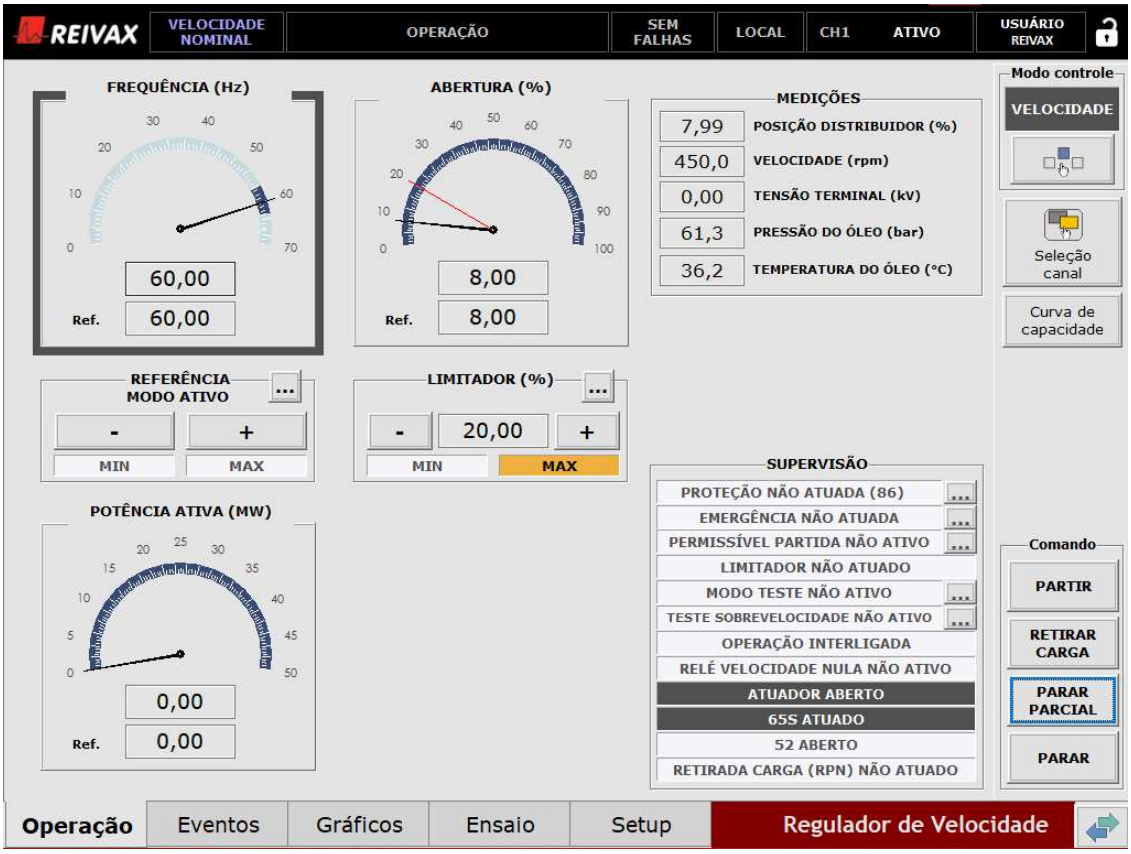
xVision (XVI)

O xVision é uma ferramenta de manipulação de telas para construção de IHMs que funciona em plataformas do SO Windows [28]. É subdividido em dois softwares:

- **xVision Edition**: É a plataforma de configuração de telas. Tem como saída de projeto um arquivo com extensão *.xvp;
- **xVision RunTime**: É o executável responsável por externar a IHM e tem como entrada um arquivo com extensão *.xvp, que previamente foi montado na plataforma xVision Edition.

A utilização do xVision RunTime no projeto é opcional, uma vez que ele mostra os estados atuais das variáveis, pode ser útil para o acompanhamento dos testes. Na Figura 12 e Figura 13 são apresentadas as telas principais dos reguladores desenvolvidas e executadas pelo xVision.

Figura 12 – Tela de operação do Regulador de Velocidade.



Fonte: Arquivo pessoal.

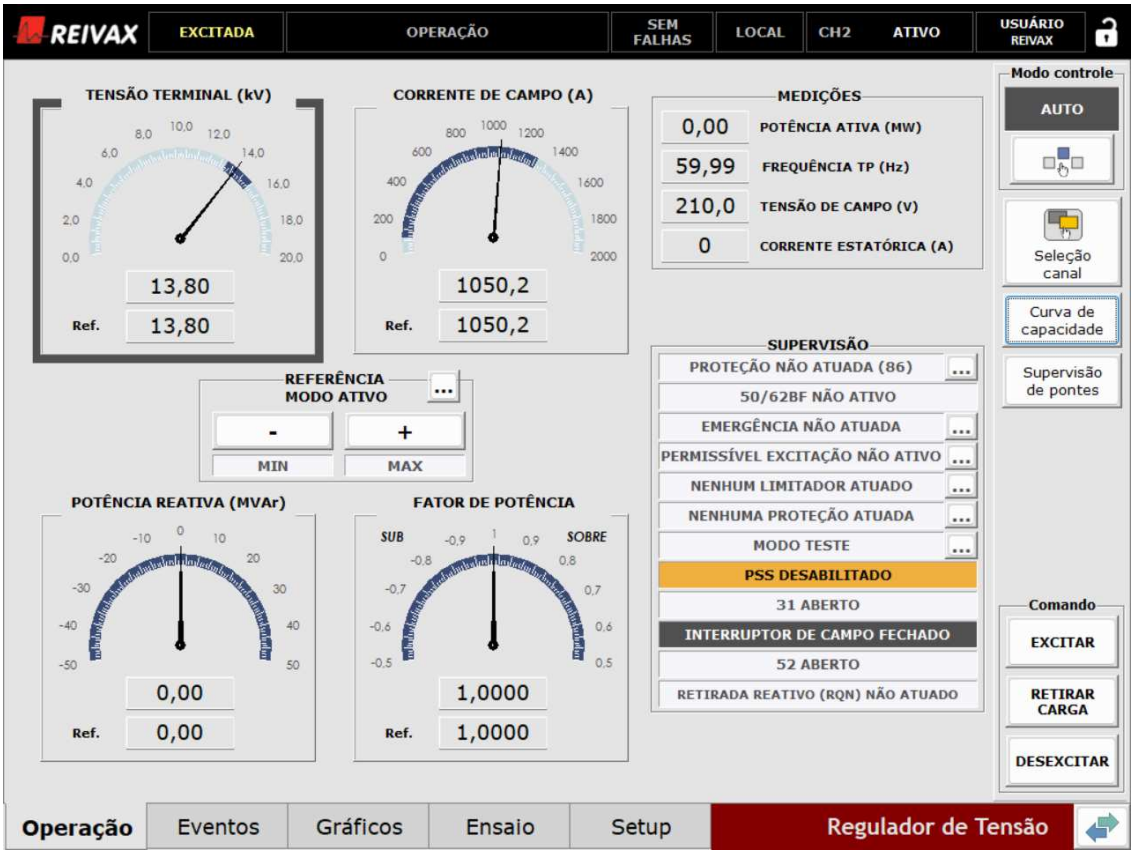
Control and Automation Framework (CAF)

O CAF é responsável por reunir os modelos de programação para o software de regulação de tensão, velocidade e automação. Há duas principais opções de configuração no CAF: o CAF-SEC [24], que é um modelo editado no SEC usado para gerar arquivos de configuração para o SBE embarcado no CPX, e o CAF-XVI [25], que é um modelo editado no xVision para criar a interface gráfica usada em telas *touchscreen*[1] ou em computadores.

Sendo mais específico, o CAF é um projeto pré-desenvolvido como ponto de partida, que tenta reunir ou prever lógicas paras as mais variadas usinas hidrelétricas, tanto em variação de hardware, quanto em questões construtivas, como o tipo de turbina (Francis, Kaplan, Bulbo e Pelton), tipo de regulador (Regulador Integrado de Tensão e Velocidade (RTVX), Regulador de Tensão (RTX), Regulador de Velocidade (RVX), RTVAX, etc). As Figuras apresentadas nas subseções SEC e XVI, apresentam o projeto CAF aberto em sua forma original, atuando como um RTVX.

[1] *Touchscreen* é uma interface de ecrã interativa sensível ao toque que permite aos usuários operar um dispositivo diretamente tocando na superfície da tela.

Figura 13 – Tela de operação do Regulador de Tensão.



Fonte: Arquivo pessoal.

A adequação do CAF conforme a especificação técnica da usina de geração de energia é onde a ferramenta de teste se insere no projeto. Assumindo que todas as lógicas integradas ao CAF estejam conforme o desejado, após a personalização realizada pelo Setor de Software, elas devem manter sua funcionalidade original. Entretanto, em situações específicas onde o Sistema de Controle seja adaptado para um comportamento distinto, a ferramenta também deve ser ajustada para reconhecer e validar a nova lógica como correta.

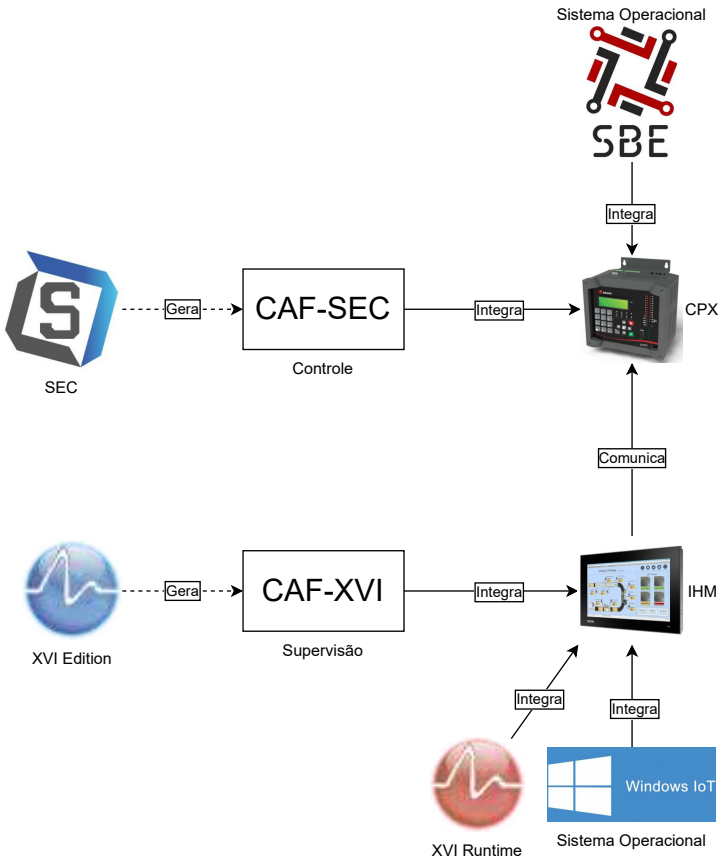
A Figura 14 demonstra como os hardwares e softwares da empresa se integram.

3.3 ESPECIFICAÇÃO FUNCIONAL

Com base no levantamento de necessidades, foram elaboradas especificações funcionais detalhadas para a ferramenta de apoio. Essas especificações definem o comportamento esperado do sistema e suas principais funcionalidades.

Diagramas UML são muito importantes por serem uma linguagem padronizada de abstração de ideias, fornecendo uma representação visual de diferentes camadas de um sistema [4]. Existem diversos contextos de representação em UML, os escolhi-

Figura 14 – Diagrama da integração dos softwares e hardwares apresentados.



Fonte: Arquivo pessoal.

dos para este projeto são apresentados a seguir.

Casos de Uso

O diagrama de casos de uso apresentado ilustra as funcionalidades fundamentais do sistema a partir da perspectiva do usuário. Ele não apenas evidencia as funções principais, mas também como os usuários externos interagem com elas.

A Figura 15 destaca as interações primordiais entre o usuário e o sistema. A simplicidade do diagrama reflete a natureza do sistema: um sistema offline que não necessita de autenticação nem de diferentes níveis de acesso para os usuários.

Os principais casos de uso, conforme identificados, incluem:

- **Gerenciar sinais**: O sistema deve permitir operações de *Create, Read, Update, Delete* (CRUD) nos sinais. Adicionalmente, deve oferecer a capacidade de exportar e importar conjuntos de sinais;
- **Gerenciar casos de teste**: O usuário deve ser capaz de gerenciar os casos de teste usando operações CRUD. Similarmente ao gerenciamento de sinais, a

função de exportar e importar casos de teste deve estar disponível;

- **Executar caso de teste**: É fundamental que os casos de teste, uma vez devidamente configurados ou importados, possam ser executados de forma automática pelo sistema.
- **Visualizar resultado de teste**: Após a execução de um caso de teste, o sistema deve apresentar os resultados de maneira clara, indicando se as especificações de desempenho foram atendidas;
- **Realizar teste de sinais**: O sistema deve permitir que os usuários conduzam testes específicos em sinais, incluindo a busca por valores de sinais e o envio de valores para as variáveis no CLP.

Diagramas de Sequência

O diagrama de sequência ilustra como os objetos no sistema interagem entre si em uma sequência temporal. Ele captura a interação entre objetos em um contexto específico de colaboração, destacando a sequência de mensagens e chamadas entre eles.

O diagrama de sequência apresentado por partes na Figura 16, Figura 17 e Figura 18, ilustra as interações entre o usuário, a aplicação, o servidor, a lógica de testes, o protocolo de comunicação e os indicadores.

Abaixo são listados os detalhes do diagrama:

Editor de Casos de Teste

1. O usuário inicia abrindo o editor de casos de teste;
2. A aplicação, por sua vez, solicita ao servidor a lista de todos os casos de teste existentes por meio da função `busca_todos_casos_teste();`
3. Após a edição, o usuário salva o caso de teste e a aplicação envia os dados ao servidor usando `salva_todos_casos();`
4. O servidor responde confirmando o salvamento.

Editor de Sinais

1. O usuário abre o editor de sinais;
2. A aplicação solicita ao servidor a lista de todos os sinais através da função `busca_todos_sinais();`

3. Depois de editar os sinais (operações de CRUD), o usuário opta por salvar as modificações;
4. A aplicação, então, comunica-se com o servidor para armazenar essas alterações usando `salva_todos_sinais()`.

Execução de Casos de Teste

1. Após selecionar um caso específico, o usuário escolhe executar o caso de teste;
2. A aplicação, em resposta, inicia o processo de teste via `executa_caso()`;
3. A lógica de testes é chamada e realiza `executa_logica()` do caso em questão;
4. Durante essa execução, os valores dos sinais são definidos e comunicados ao CLP através do protocolo de comunicação;
5. O sistema também busca valores de sinais e calcula indicadores pertinentes ao sistema de controle;
6. Os resultados do caso de teste são então retornados ao usuário.

Visualizador de Resultados

1. O usuário pode optar por abrir um visualizador de resultados;
2. A aplicação busca todos os resultados armazenados através de `busca_todos_resultados()`;
3. O usuário pode selecionar um resultado específico para obter mais detalhes.

Teste de Sinais

1. No módulo de teste de sinais, o usuário começa buscando um valor de sinal específico;
2. A aplicação, em resposta, busca esse valor por meio de `busca_valor()`;
3. A consulta é feita ao CLP através do protocolo de comunicação.
4. Posteriormente, o usuário pode definir um valor para o sinal;
5. Esse novo valor é então enviado ao CLP, alterando o estado atual do sinal.

Diagrama de Atividade

O diagrama de atividade é semelhante a um fluxograma e é usado para modelar o fluxo de controle ou o fluxo de dados. Ele é útil para mapear processos de negócios e procedimentos operacionais, mostrando as atividades, decisões e ramificações de um processo.

Na Figura 19, o diagrama de atividade fornecido representa as ações e decisões que podem tomadas durante a utilização do aplicativo.

1. Gerenciar Sinais: Figura 22;
2. Gerenciar Casos de Teste: Figura 20 e Figura 21;
3. Execução de Caso de Teste: Figura 23;
4. Realizar Testes de Sinais: Figura 24.

Diagrama de Classes

Este é um dos diagramas mais usados em UML e representa a estrutura estática do sistema. Ele mostra as classes, seus atributos, métodos e as relações entre as classes.

O diagrama de classes apresentado na Figura 25 representa a estrutura e inter-relações entre diferentes classes em um sistema de testes.

Segue uma descrição detalhada das classes e suas relações:

1. **CaseResult**: Representa o resultado de um caso de teste, contendo informações como nome do projeto, nome do caso, datas de início e fim, entre outros. Contém uma lista de resultados dos dados e uma série de dados;
2. **ResultData**: Mantém detalhes sobre os resultados de um teste, como etapa, repetição e data/hora de início e fim. Também contém uma lista de séries e uma análise do sistema de controle;
3. **Series**: Contém uma lista de valores de ponto flutuante que representa uma série de dados;
4. **ControlSystemAnalysis**: Representa a análise de um sistema de controle com descrição do sinal e vários indicadores de resultados como tempo de subida, tempo de acomodação, erro de regime permanente e sobressinal;
5. **IndicatorResult**: Representa o resultado de um indicador, contendo um valor esperado, um valor real e um booleano indicando sucesso ou falha;

6. **Case**: Representa um caso de teste com nome, descrição, lista de etapas e modo de comunicação. Também possui métodos para adicionar, remover etapas e salvar e executar dados;
7. **Step**: Define uma etapa específica dentro de um caso, contendo entradas e saídas, além de outras propriedades como taxa de amostragem e repetições;
8. **StepOutput e StepInput**: Representam as saídas e entradas para uma etapa, respectivamente. Cada uma contém várias propriedades e métodos relacionados ao tratamento de sinais;
9. **ExpectedBehavior**: Representa o comportamento esperado de um sinal, com vários indicadores de sistemas de controle que determinam esse comportamento;
10. **Indicator**: Define um indicador com valor e tolerância;
11. **ModbusAddress**: Contém detalhes de endereço para comunicação Modbus, incluindo Registrador Modbus (OR) e bit;
12. **Signal**: Representa um sinal com propriedades como índice, chave, descrição e grupo. Também contém detalhes do sinal, como unidade, limites mínimo e máximo e endereço Modbus, *International Electrotechnical Commission* (IEC)-104;
13. **RgxSignal**: Define um sinal de RGX recebido com propriedades como nome, tempo e listas de valores dos eixos x e y.

3.4 ESPECIFICAÇÃO NÃO FUNCIONAL

Além dos requisitos funcionais, foram identificados requisitos não funcionais que definem atributos de qualidade do sistema.

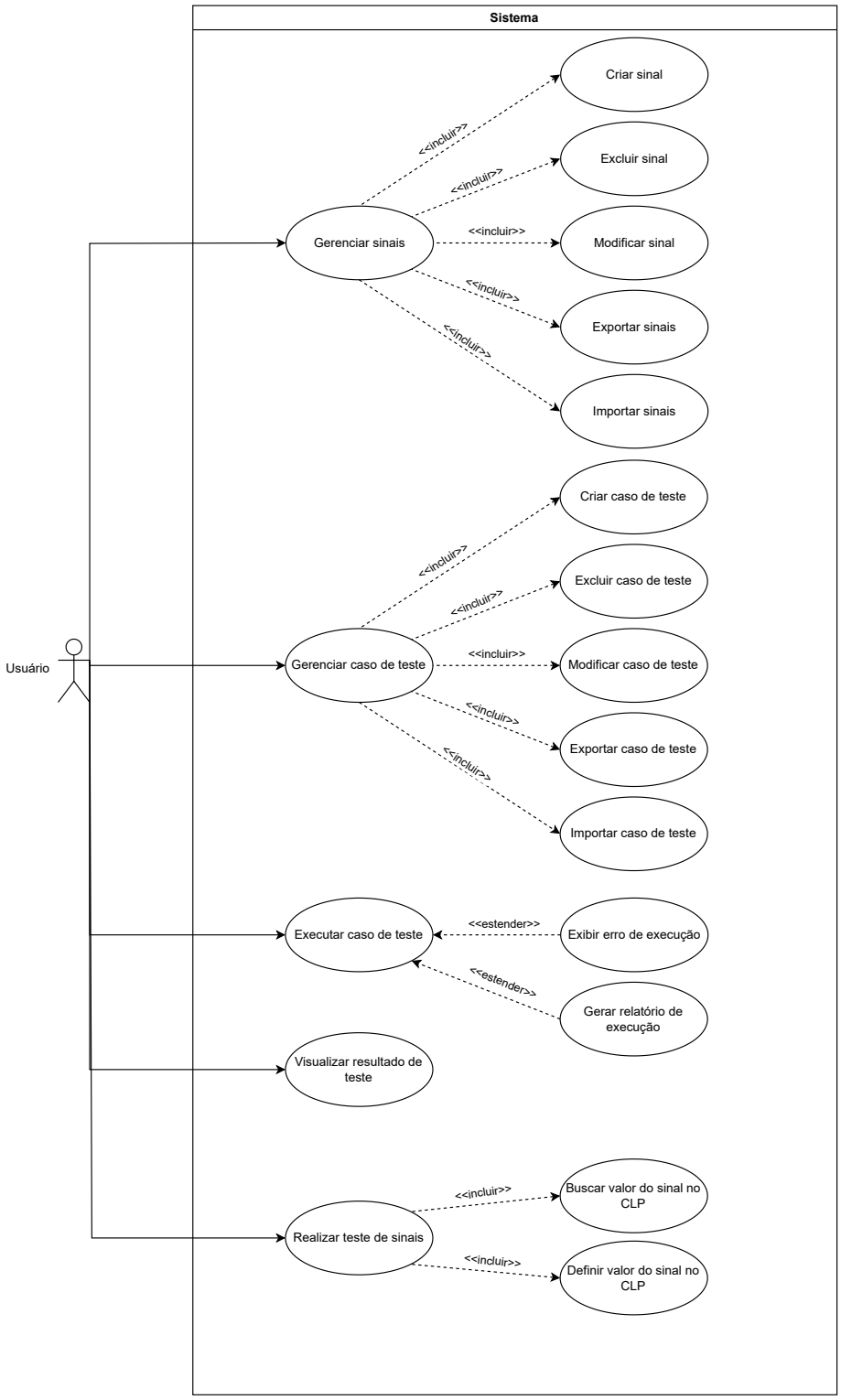
O requisito de desempenho implica que o sistema deve responder às interações do usuário dentro de um tempo máximo definido para garantir a fluidez da experiência do usuário. A usabilidade é definida pela rápida compreensão da interface, sendo de certa forma intuitiva, permitindo que novos usuários utilizem com pouco ou nenhum treinamento.

Quanto à escalabilidade, o sistema deve ser capaz de lidar com o aumento da carga de trabalho mantendo o nível de fluidez. A portabilidade requer que o sistema seja executado em diferentes plataformas e dispositivos sem a necessidade de grandes alterações no código-fonte. E a manutenibilidade diz respeito à facilidade com que o sistema pode ser alterado, corrigido ou melhorado ao longo do tempo.

O sistema possuir um layout responsivo pode ser interessante, capaz de se adaptar a diferentes tamanhos de tela e resoluções sem perder funcionalidade ou estética, assegurando assim a usabilidade em variados dispositivos.

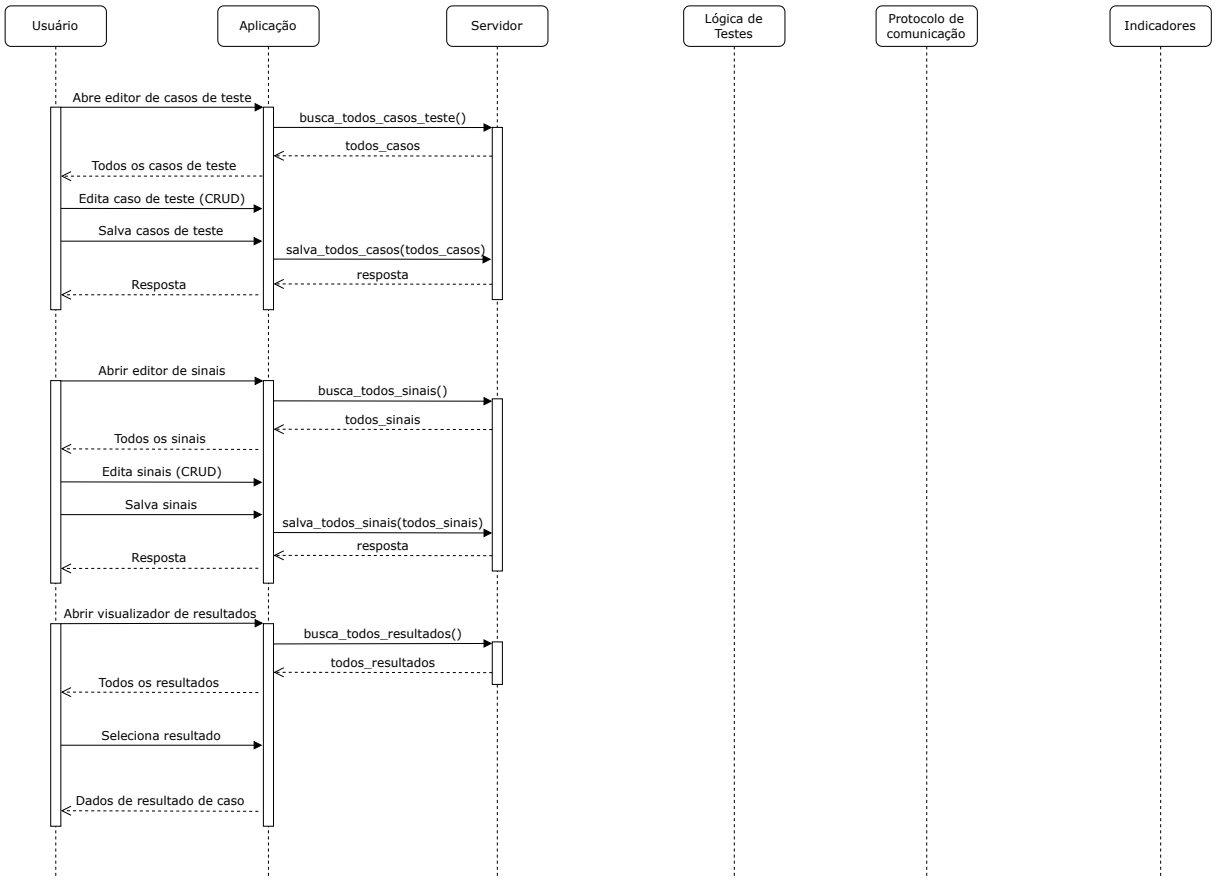
Seria importante que o sistema, no futuro, seja projetado para suportar a execução de múltiplas instâncias de casos em paralelo, garantindo a escalabilidade e a eficiência na utilização dos recursos de hardware disponíveis.

Figura 15 – Diagrama UML Casos de Uso.



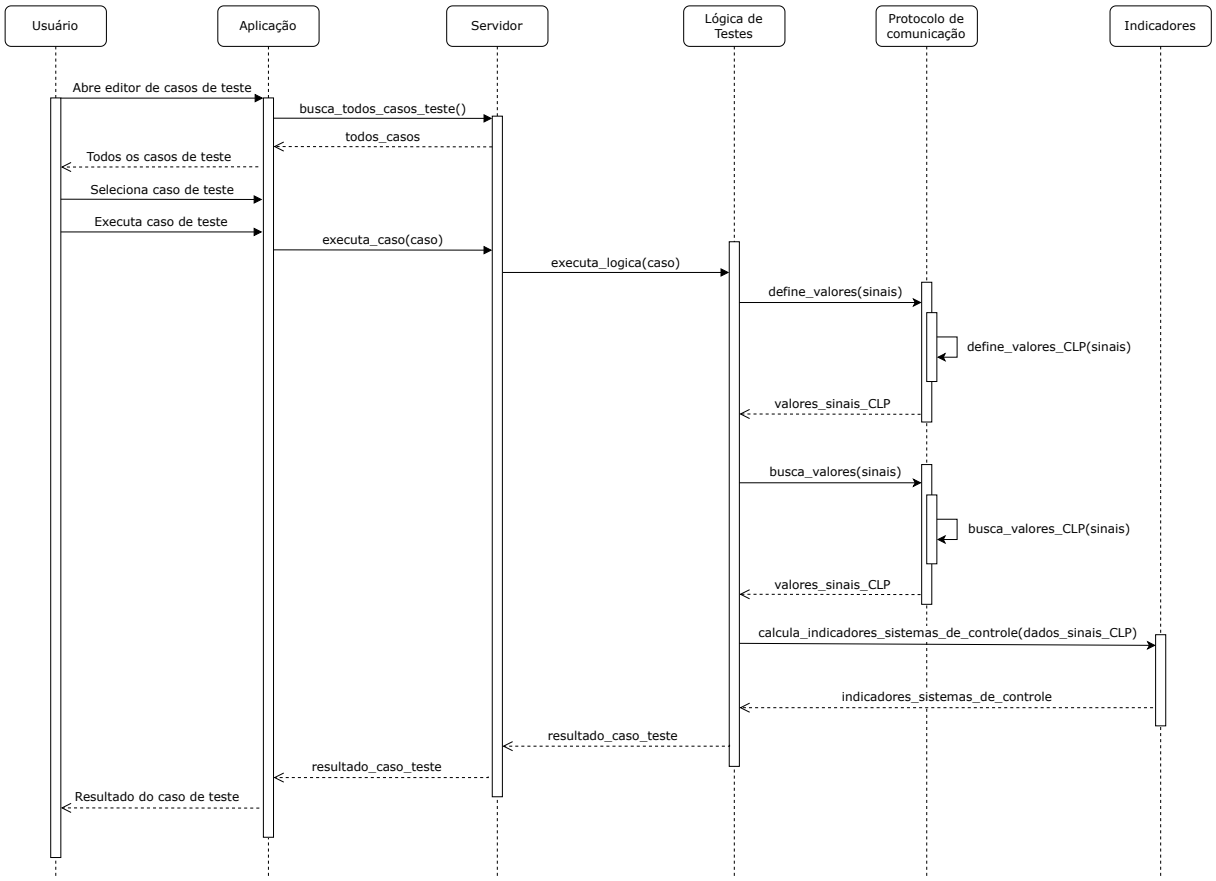
Fonte: Autoria própria.

Figura 16 – Diagrama UML de Sequência (Parte 1/3).



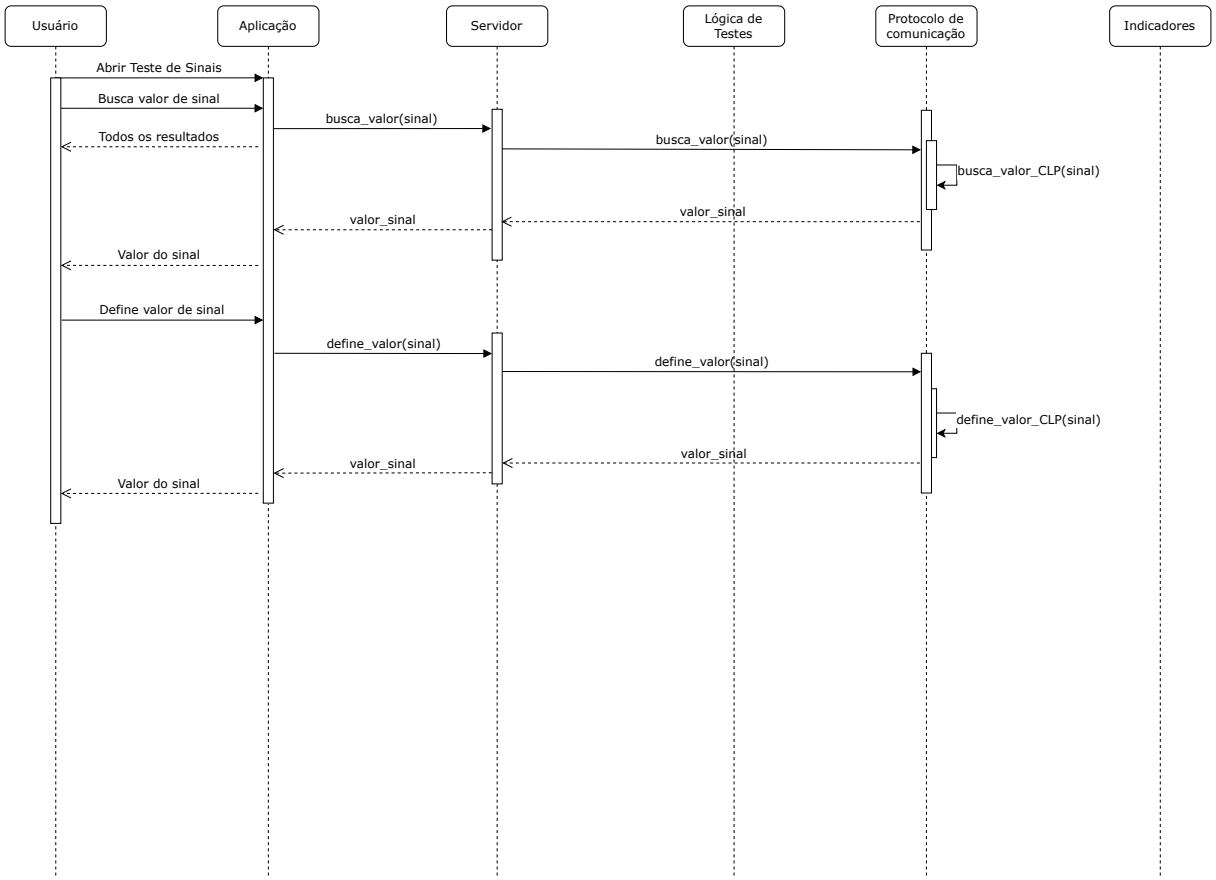
Fonte: Autoria própria.

Figura 17 – Diagrama UML de Sequência (Parte 2/3).



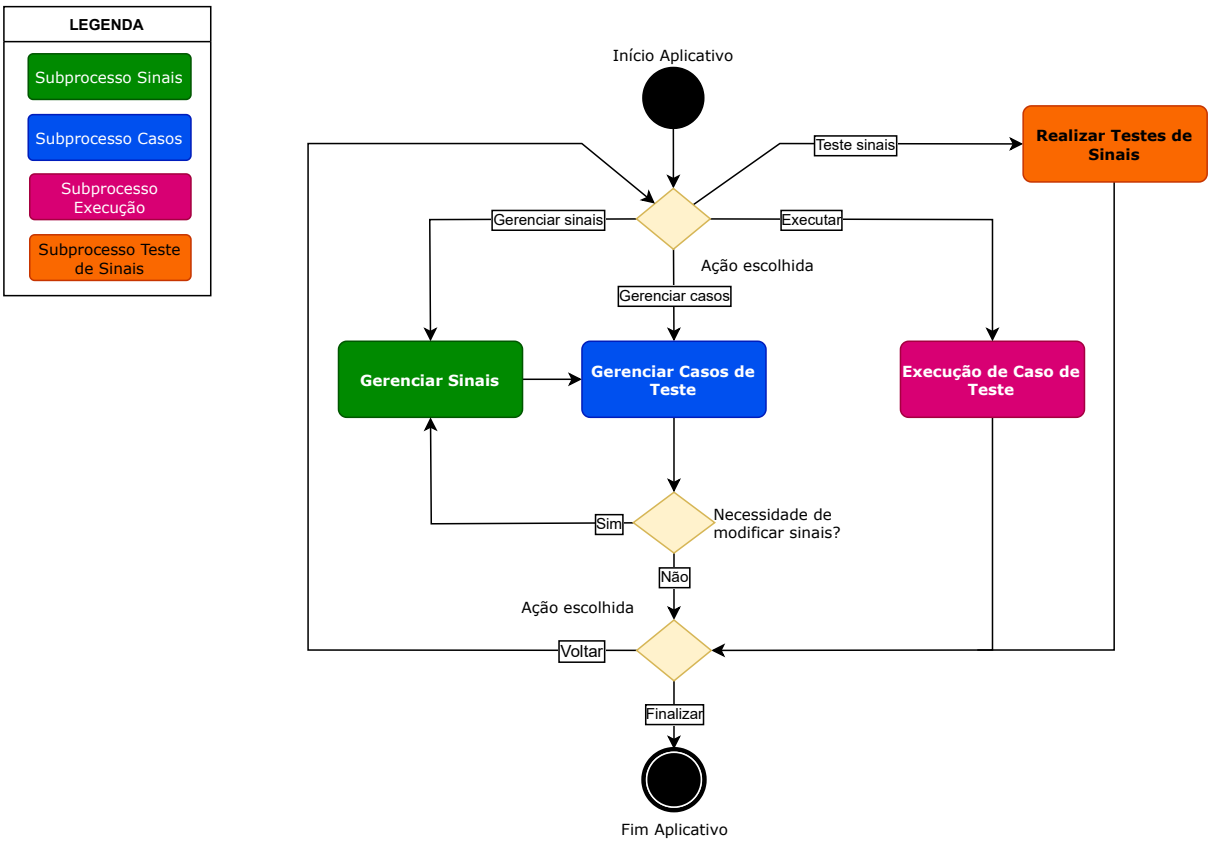
Fonte: Autoria própria.

Figura 18 – Diagrama UML de Sequência (Parte 3/3).



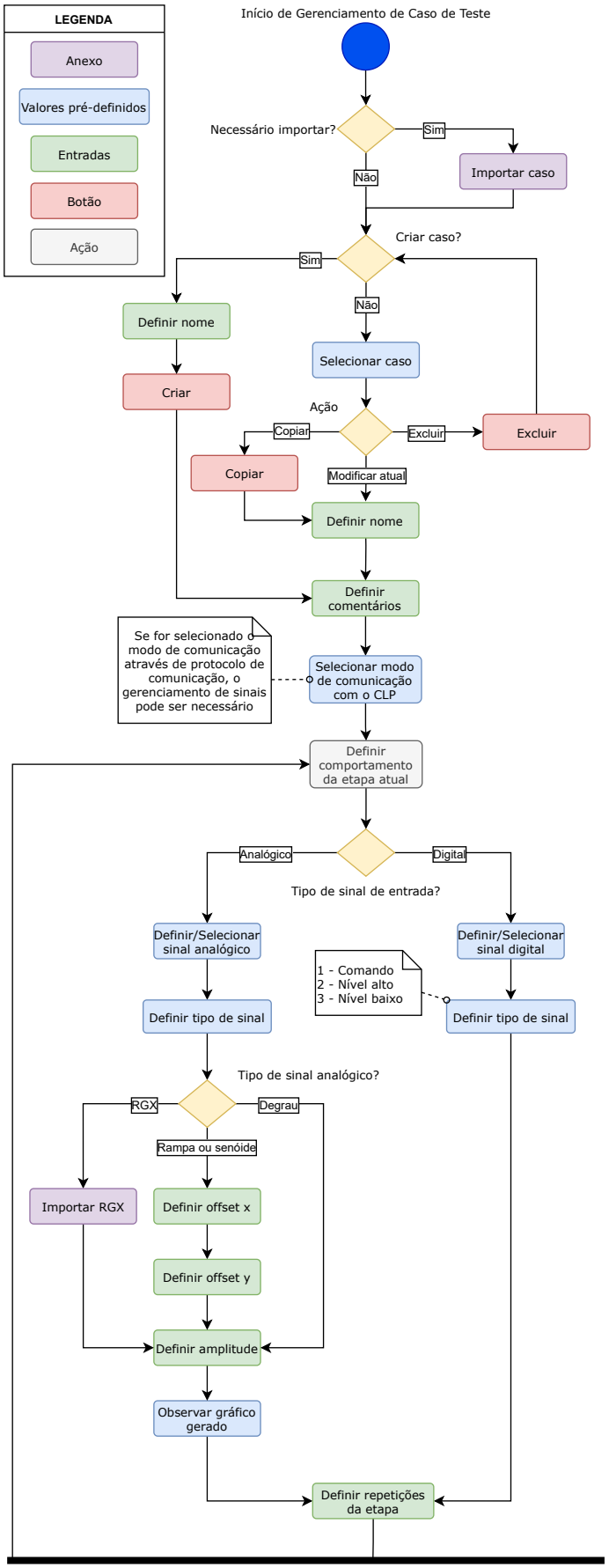
Fonte: Autoria própria.

Figura 19 – Diagrama UML de Atividade Macro.



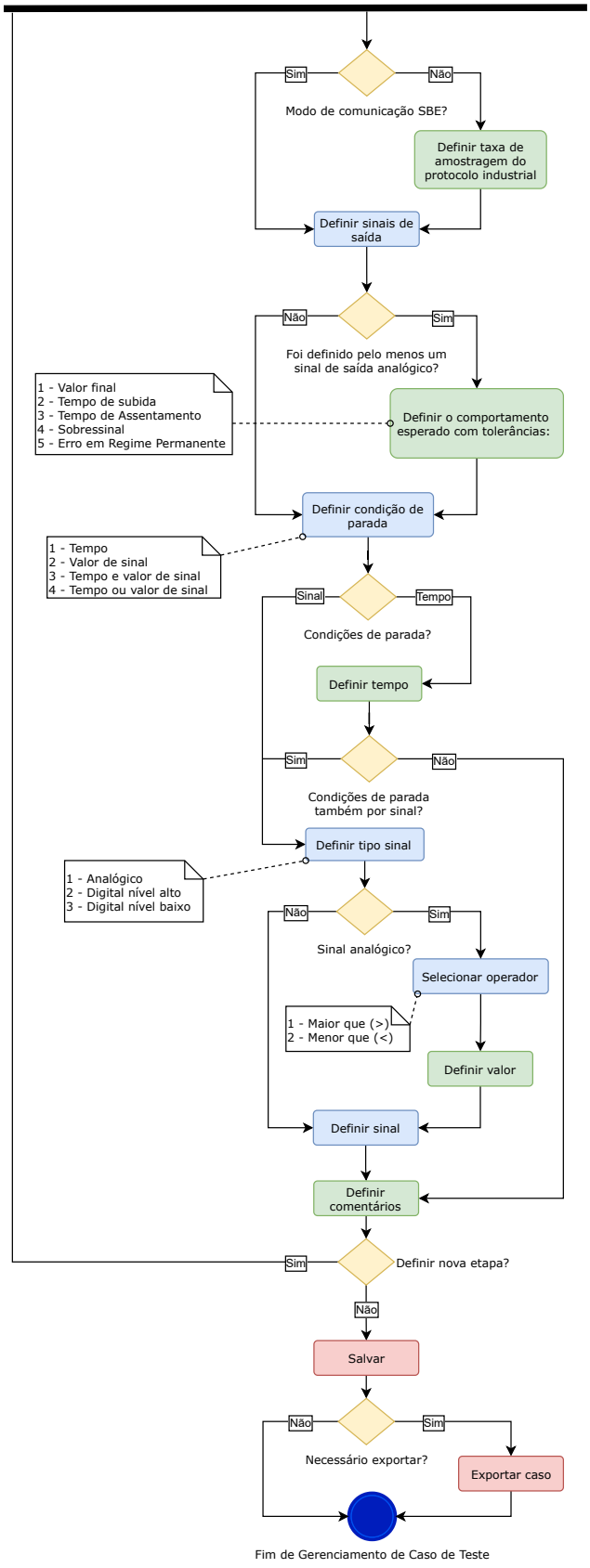
Fonte: Autoria própria.

Figura 20 – Diagrama UML de Atividade: Gerenciamento de Casos de Teste (Parte 1/2).



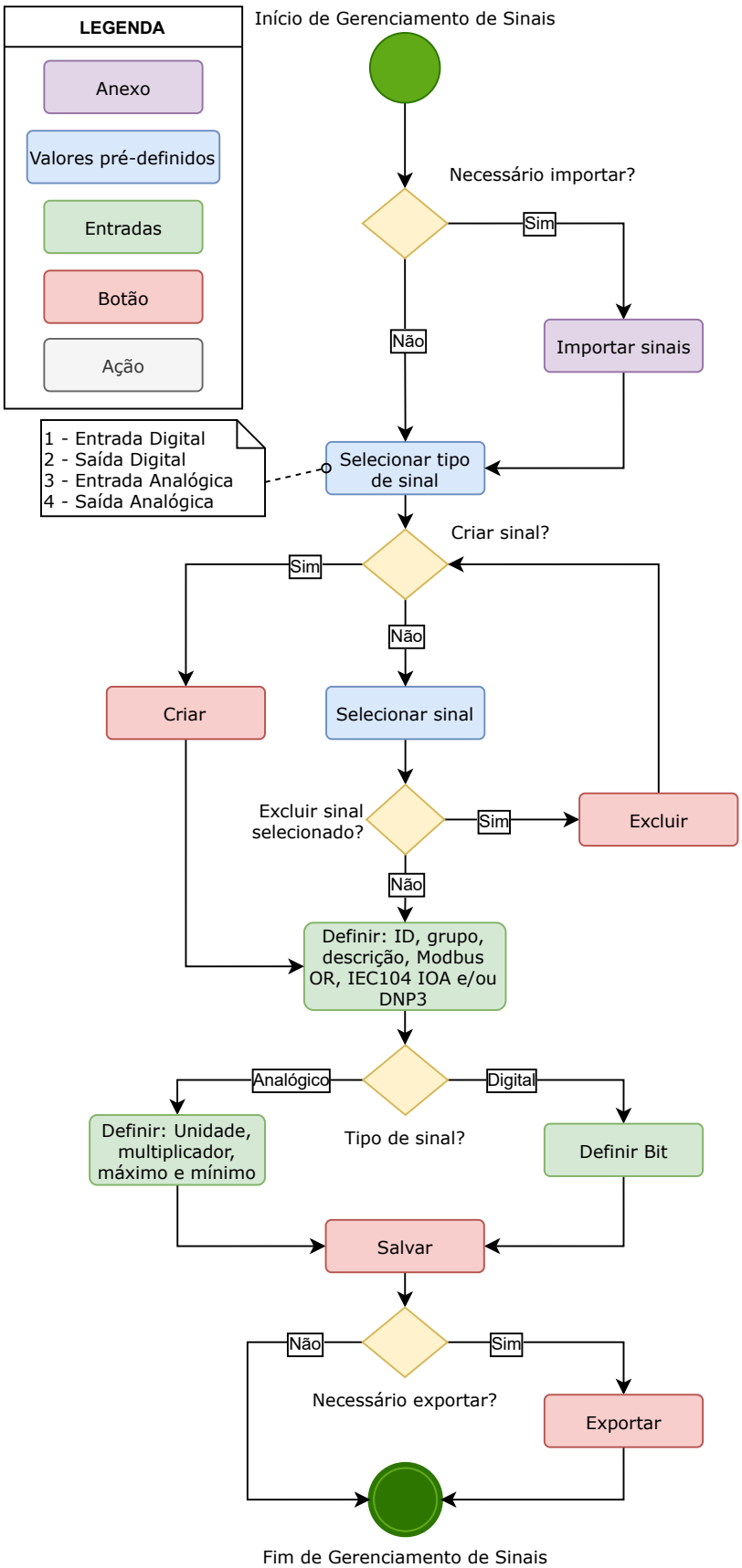
Fonte: Autoria própria.

Figura 21 – Diagrama UML de Atividade: Gerenciamento de Casos de Teste (Parte 2/2).



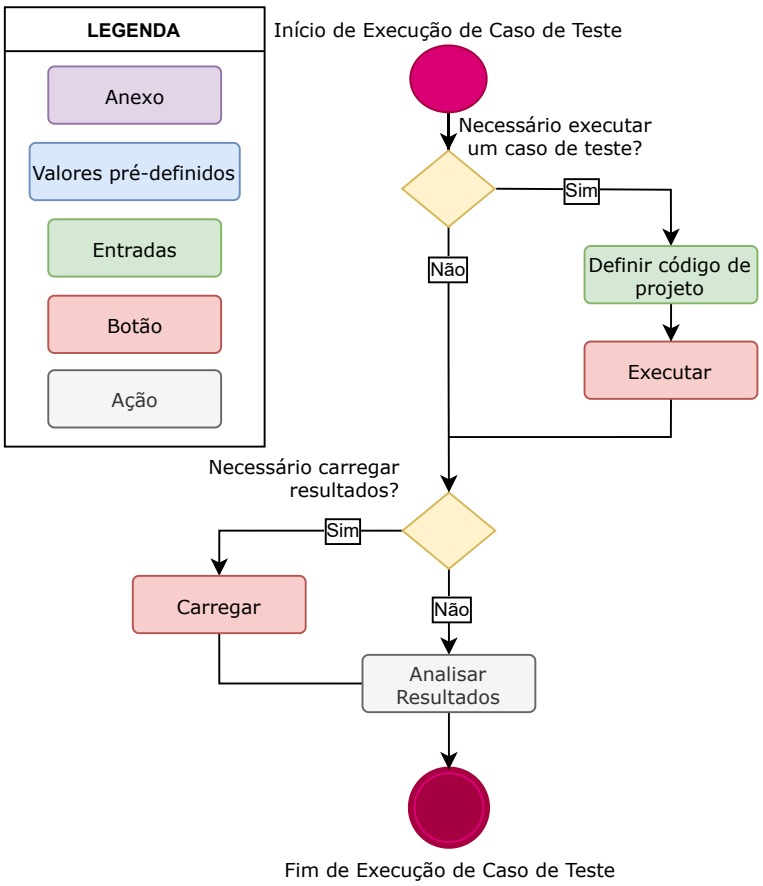
Fonte: Autoria própria.

Figura 22 – Diagrama UML de Atividade: Gerenciar Sinais.



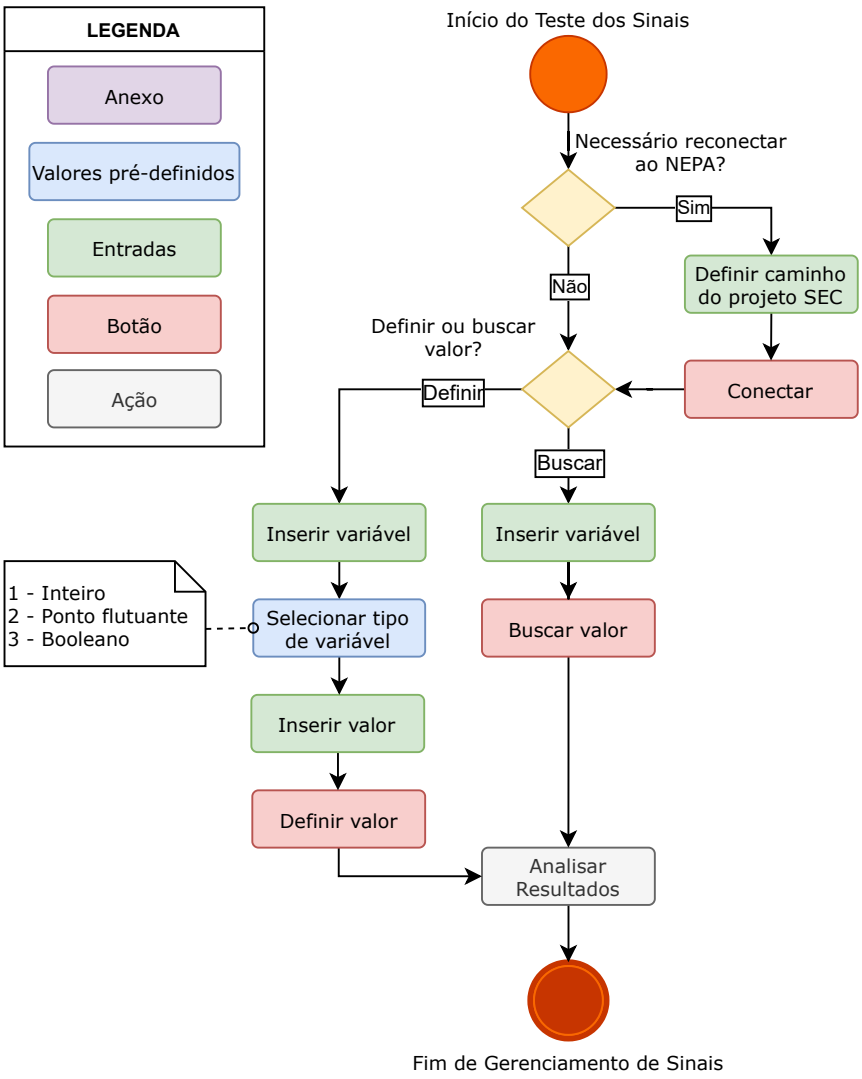
Fonte: Autoria própria.

Figura 23 – Diagrama UML de Atividade: Visualizar Resultado de Teste.



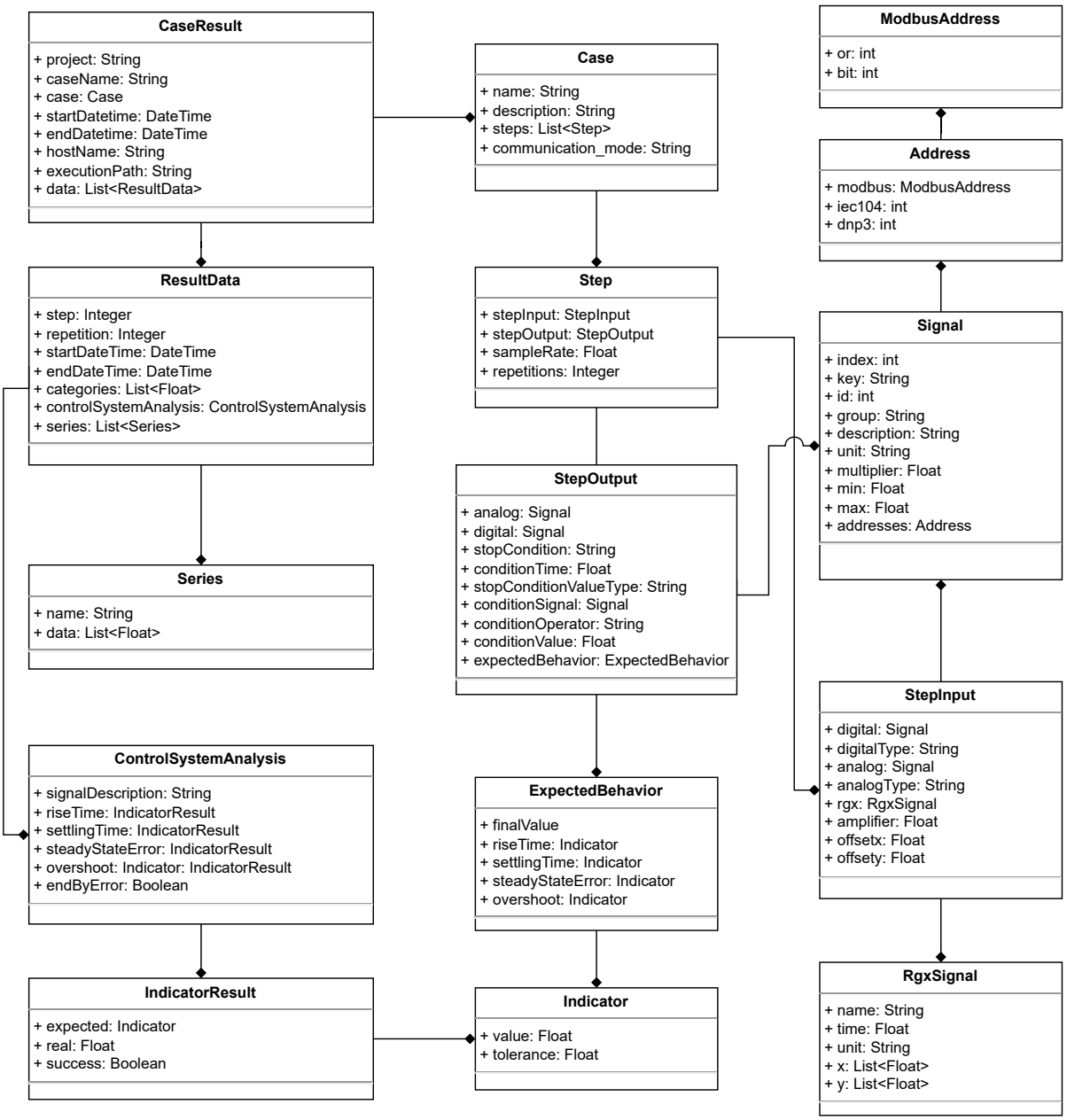
Fonte: Autoria própria.

Figura 24 – Diagrama UML de Atividade: Realizar Testes de Sinais.



Fonte: Autoria própria.

Figura 25 – Diagrama UML de Classe.



Fonte: Autoria própria.

4 PROJETO DE SOFTWARE

O capítulo de Projeto de Software é fundamental para definir a estrutura, arquitetura e design da ferramenta. Neste capítulo, é discutida a arquitetura geral do sistema, a modelagem de dados, o design da interface do usuário e outros aspectos importantes relacionados ao desenvolvimento do software. A aplicação a ser desenvolvida tomou o nome de Software Test App (STA), com intuito de esclarecer que a aplicação pode abranger qualquer tipo de teste de software de sistemas de controle.

4.1 SELEÇÃO DE FERRAMENTAS

A escolha adequada das ferramentas é de total importância para o sucesso de qualquer projeto, não só afeta a eficiência e a qualidade do trabalho realizado, mas também pode ter um impacto na satisfação dos desenvolvedores e usuários. A seguir são apresentadas as ferramentas escolhidas para cada seção de desenvolvimento e a justificação.

Banco de Dados (BD)

Um BD é uma coleção organizada de dados, geralmente armazenados e acessados eletronicamente a partir de um sistema de computador. Os BDs servem como um meio eficaz para armazenar, gerenciar e recuperar informações. Elas desempenham um papel crucial em diversos campos, armazenando dados sobre variados tópicos, como registros de clientes, detalhes de produtos, transações, entre outros. Inicialmente, muitos projetos podem começar com a coleta de dados em ferramentas simples como processadores de texto ou planilhas. No entanto, à medida que os dados crescem, esses métodos podem se tornar ineficientes [15].

Realizando um estudo dos dados a serem guardados no STA, foram previstos os seguintes dados:

1. **Cases**: Configuração dos casos de testes;
2. **Results**: Resultados dos casos de teste;
3. **Signals**: Endereços dos sinais de protocolos industriais;
4. **RGX**: Guardar arquivos de registro de sinais provenientes do SEC, utilizados nos casos de teste, em formato padrão *Armored* ASCII *File* (ASC).

Duas formas de manuseio e armazenamento de dados de forma simples, sabendo que o STA gerenciará dados simples, o JavaScript *Object Notation* (JSON) e *eXtensible Markup Language* (XML) são dois dos formatos mais comuns usados

para essa finalidade. Ambos têm seus próprios conjuntos de características e usos, tornando-os adequados para diferentes tipos de tarefas [32].

O JSON é um formato de dados leve que é fácil de ler e escrever para humanos e fácil de analisar e gerar para máquinas. Ele é independente de linguagem, o que significa que pode ser usado em praticamente qualquer linguagem de programação. Por causa de sua simplicidade e eficiência, o JSON tornou-se uma escolha comum para APIs e serviços web.

O XML, por outro lado, é uma linguagem de marcação que oferece uma forma de estruturar dados de maneira que eles possam ser compartilhados em redes e na internet. Ele utiliza *tags* para diferenciar entre elementos e dados, proporcionando uma maneira de descrever estruturas de dados complexas.

Consequentemente, optou-se pelo formato JSON devido à sua facilidade de uso, popularidade e adoção generalizada em interfaces de API. Os arquivos JSON, juntamente com os arquivos ASC, são eficientemente gerenciados e armazenados pelo STA.

Em projetos futuros, é possível que a complexidade dos dados aumente significativamente, tornando necessário o emprego de um Sistema de Gerenciamento de Banco de Dados (SGBD) *Not Only Structured Query Language* (NoSQL) para lidar eficientemente com dados não relacionais. SGBD são um conjunto de softwares utilizados para o gerenciamento de um BD, tendo como principal objetivo gerenciar dados utilizados por aplicações clientes e remover esta responsabilidade das mesmas [34]. Dentre os principais SGBDs NoSQL destacam-se: MongoDB, Redis e Cassandra.

Framework SBE

O setor de Pesquisa e Desenvolvimento da REIVAX implementou e forneceu um framework em linguagem Python. A escolha do uso deste recurso decorre da necessidade de utilizar os meios mais eficientes possíveis.

O framework oferece uma série de métodos de comunicação e gerenciamento direto no SBE do CPX, permitindo que o uso de protocolos padrões industriais fique em segundo plano. Contudo, é importante destacar que, embora um protocolo industrial possa não ser tão eficiente quanto o framework fornecido, o uso deste não será descartado, uma vez que os CLPs de mercado não utilizam o SBE.

Linguagem de Programação

A linguagem de programação representa um método padronizado composto por um conjunto de regras sintáticas e semânticas, utilizado na criação de código fonte. Esse código pode ser posteriormente compilado e convertido em um software executável. Linguagens diferentes são como ferramentas em uma caixa de ferramentas:

embora cada linguagem seja capaz de expressar a maioria dos algoritmos, algumas são obviamente mais apropriadas para certas aplicações do que outras [9].

Ao longo do currículo do curso de Engenharia de Controle e Automação da Universidade Federal de Santa Catarina (UFSC), a linguagem Python surge como uma ferramenta fundamental. Seja como um tópico de introdução, ou para resolver uma vasta gama de problemas, esses problemas variam desde os mais simples até os de alta complexidade. Python é uma linguagem de programação versátil, utilizada em diversos domínios de aplicação [20], as que são aplicáveis a esse projeto são listadas a seguir:

- **Desenvolvimento de aplicações Web**: Python oferece múltiplas opções para desenvolvimento web, incluindo frameworks, micro-frameworks, e sistemas avançados de gerenciamento de conteúdo. A biblioteca padrão do Python suporta vários protocolos web, processamento de e-mail, e outras bibliotecas adicionais estão disponíveis para tarefas específicas de rede;
- **Científico e Numérico**: No campo científico e numérico, Python é amplamente utilizado. Pacotes instaláveis oferecem ferramentas poderosas para matemática, ciência, engenharia, análise de dados e computação interativa;
- **Desenvolvimento de Software**: Python é frequentemente usado por desenvolvedores de software para controle de construção, teste e outras funções de apoio.

Servidor API

Servidor API é uma ferramenta que permite construir, gerenciar e implantar APIs REST. Usando um servidor API, você pode configurar como expor os dados aos usuários para os formatos de arquivo populares de diferentes fontes [18].

Projetar uma API para testes de reguladores, permite que desenvolvedores e engenheiros interajam com CLPs de forma prática para realizar testes, diagnósticos e validação de reguladores e sistemas automatizados.

Existem diversas bibliotecas de API para Python, FastAPI é a que mais se encaixa no contexto proposto de projeto. FastAPI é uma framework web de alto desempenho e fácil de usar, que suporta código assíncrono e fácil integração com outras ferramentas e bibliotecas. Seguem os comentários principais:

- **Características**:
  - **Desempenho**: FastAPI usa recursos avançados, como a sintaxe async/await e dicas de tipo (gera dicas automaticamente para parâmetros de requisições), para fornecer alto desempenho e escalabilidade. Ideal para a integração com a framework do SBE que executa da mesma forma suas operações assincronamente;

  - **Facilidade para codificar**: Projetado para ser fácil de usar, com uma sintaxe clara e concisa que facilita a definição de rotas, parâmetros e modelos de resposta;
  - **Redução de bugs**: FastAPI usa tipagem Python para definir estruturas de dados de solicitação e resposta, que podem ser usadas para validar automaticamente os dados de entrada. Isso torna fácil construir APIs autodocumentadas e reduz a probabilidade de erros ou bugs no seu código.

- **Vantagens**:

  - **Fácil de executar**: Um dos frameworks Python mais rápidos. Você também pode usar para criar APIs sem conhecimento avançado de programação;
  - **Ferramentas de monitoramento integradas**: Integrado com ferramentas de monitoramento que podem fornecer alertas quando você atinge certos limiares;
  - **Fácil de adaptar**: Usa uma abordagem baseada em toolkit, então você não precisa criar tudo do zero. Assim, você pode usar vários modelos para criar APIs poderosas.

- **Desvantagens**:

  - **Comunidade pequena**: Um dos principais inconvenientes do FastAPI é sua pequena comunidade. Uma comunidade pequena pode dificultar o desenvolvimento porque há menos documentação de suporte disponível gratuitamente;
  - **Falta de sistema de segurança embutido**: FastAPI não possui um sistema de segurança embutido, mas usa o módulo fast API.security.

As vantagens e desvantagens anteriormente listadas reforçam que o FastAPI é uma escolha adequada para o STA. A ausência de um sistema de segurança embutido não representa um problema no escopo deste trabalho, considerando que foi projetado um sistema local que não realizará conexões com a internet e nem aceitará conexões externas em nenhum momento.

Protocolo Modbus

Modbus é um protocolo de comunicação na camada de aplicação, situado no nível 7 do modelo *Open Systems Interconnection* (OSI), que fornece comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. Desde 1979, Modbus é um padrão adotado na indústria para comunicação serial, permitindo a comunicação entre milhões de dispositivos de automação. Modbus é um

protocolo de mensagens na camada de aplicação para comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. [16].

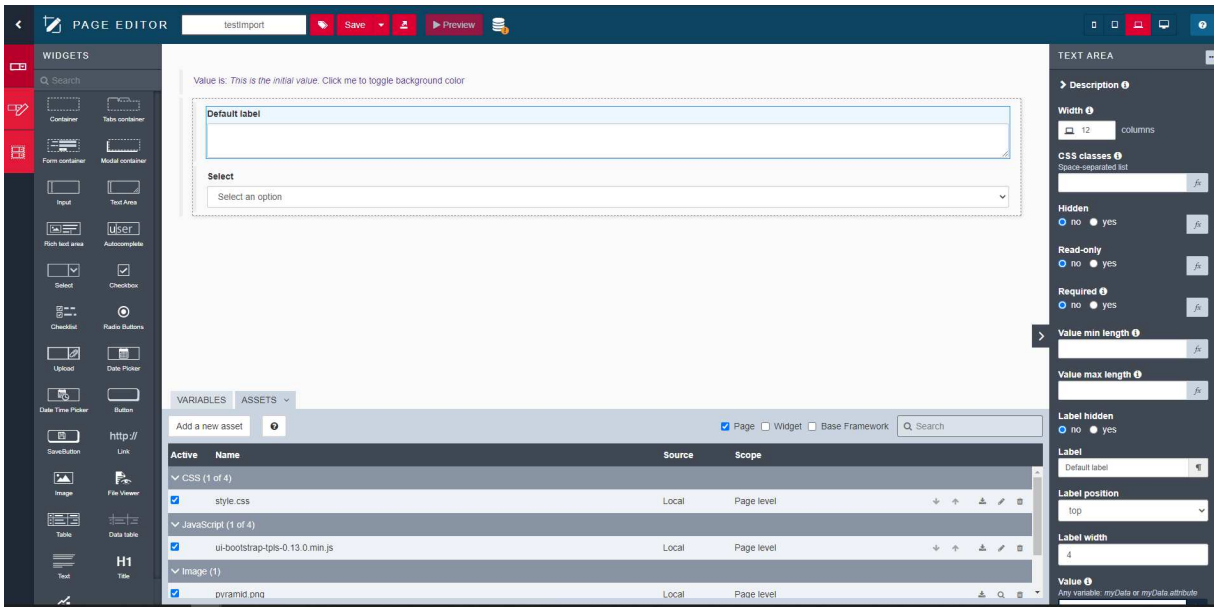
O CPX incorpora em seu software integrado a funcionalidade de servidor Modbus (também conhecido como escravo). Consequentemente, é essencial um cliente no STA, e para atender a essa necessidade, a biblioteca Python pyModbusTCP disponibiliza a classe ModbusClient, facilitando a implementação do lado cliente da comunicação Modbus.

Construtor de interface do usuário (*Graphical User Interface* (GUI))

Existem diversas formas de construir uma interface, no contexto do STA, refere-se ao ponto de interação entre o usuário final e a API, e junto a isso, o FastAPI fornece pacote de gerenciamento de páginas *HyperText Markup Language* (HTML).

Realizando pesquisas por construtores de interface HTML, o UI Designer (Figura 26) foi a ferramenta open source mais completa. Ela é composta de uma aplicação backend em Java e AngularJS no frontend [3]. O código fonte está disponível neste repositório GitHub: `https://github.com/bonitasoft/bonita-ui-designer`

Figura 26 – Visão geral do UI-designer.



Fonte: GitHub [3]

Na Figura 26, está apresentado a tela de trabalho principal do UI Designer. A seguir é listado de forma geral as funcionalidades que tornam essa ferramenta muito versátil:

- **Lateral esquerda**: Widgets de arrastar e soltar padrões da plataforma, se não for o suficiente, ela fornece a possibilidade de criar widgets customizáveis;

- **Barra Inferior**:
  - **Variables**: Possibilidade de ser um valor estático, JSON, JS ou de requisição de API;
  - **Assets**: Anexar elementos JS, CSS e imagens.
- **Lateral Direita**: Configurações específicas do widget que está selecionado;
- **Barra Superior**: Salvar, exportar, renomear e gerar uma previsão funcional da página.

O HTML é ótimo para declarar documentos estáticos, mas enfrenta dificuldades quando tentamos usá-lo para declarar visualizações dinâmicas em aplicações web. O AngularJS, framework frontend da Google, permite que você estenda o vocabulário HTML para sua aplicação. O ambiente resultante é extraordinariamente expressivo, legível e rápido para desenvolver [11].

Visualizador Gráfico

A visualização gráfica dos sinais amostrados pelo STA é um requisito básico do projeto. Ela permite que o engenheiro ou técnico analise minuciosamente os dados coletados.

O UI Designer oferece um widget de gráficos nativamente, o Chart.js, mas não obteve um resultado satisfatório com um volume de dados muito grande. Após muitos testes com diferentes bibliotecas das mais diversas e foi encontrada uma gratuita de código aberto, com grandes pontos positivos que são comentados ao longo desta subseção.

O ApexCharts é uma biblioteca de gráficos moderna que ajuda desenvolvedores a criar visualizações bonitas e interativas para páginas web. É um projeto de código aberto licenciado sob o MIT[1] e é gratuito para uso em aplicações comerciais [1].

Integrar visualizações com o ApexCharts é prático, com extensa documentação da biblioteca, é possível escolher entre uma ampla gama de gráficos, criando uma combinação de diferentes gráficos para fornecer uma diferenciação entre os dados.

Os gráficos do ApexCharts são flexíveis e responsivos - fazendo com que seus gráficos funcionem em desktops, tablets e também em dispositivos móveis.

No UI Designer é necessária a criação de um widget customizável para abranger essa nova solução gráfica. O ApexCharts oferece uma interface com o AngularJS, o que torna a integração mais fluida.

[1] O software é fornecido "como está", sem garantia de qualquer tipo, expressa ou implícita, incluindo, mas não se limitando às garantias de comercialização, adequação para um propósito específico e não infração.

Cálculo de indicadores de Sistemas de Controle

Python é uma das linguagens mais utilizadas no campo científico e numérico, várias bibliotecas oferecem ferramentas matemáticas e de Sistemas de Controle.

Não foi encontrada uma biblioteca confiável em cálculo de indicadores de Sistemas de Controle a partir de dados crus em listas, apenas para sistemas modelados no domínio da frequência.

Duas bibliotecas muito famosas em operação com sinais em dados crus é a NumPy e Pandas. Elas serão úteis para o cálculo de indicadores e filtragem de sinais analógicos com ruídos.

NumPy é basicamente para operações básicas, como ordenação, indexação e funcionamento elementar no tipo de dado array. Por outro lado, Pandas contém métodos para análise e manipulação de dados, algumas das quais estão presentes em NumPy em certa medida, mas não de forma plena [6].

Com isso, foram desenvolvidas as funções que calculam os indicadores de Sistemas de Controle, fizeram parte do estudo e implementação.

Criação de executável

Após a programação de todo o sistema, chega o momento em que o usuário final necessita de um executável simples para ser utilizado.

O PyInstaller agrupa uma aplicação Python e todas as suas dependências em um único pacote. O usuário pode executar o aplicativo empacotado sem instalar um interpretador Python ou quaisquer módulos. .

O PyInstaller é utilzado para os SOs Windows, MacOS X e Linux. No entanto, não é um compilador cruzado; para criar um aplicativo Windows você executa o PyInstaller no Windows, e para criar um aplicativo Linux você o executa no Linux, etc. A equipe de desenvolvimento não oferece garantia (todo código para essas plataformas vem de contribuições externas) de que o PyInstaller funcionará nessas plataformas ou de que elas continuarão sendo suportadas [19].

4.2 ARQUITETURA DO SOFTWARE

A arquitetura de software define a estrutura geral do sistema, incluindo a organização dos componentes e suas interações. Para a ferramenta de apoio, a arquitetura será baseada em uma arquitetura de três camadas:

A camada de apresentação será responsável pela interface do usuário. Ela incluirá a GUI que permitirá aos engenheiros de software interagir com a ferramenta. O frontend será desenvolvido usando uma ferramenta de arrastar e soltar para criar uma interface intuitiva.

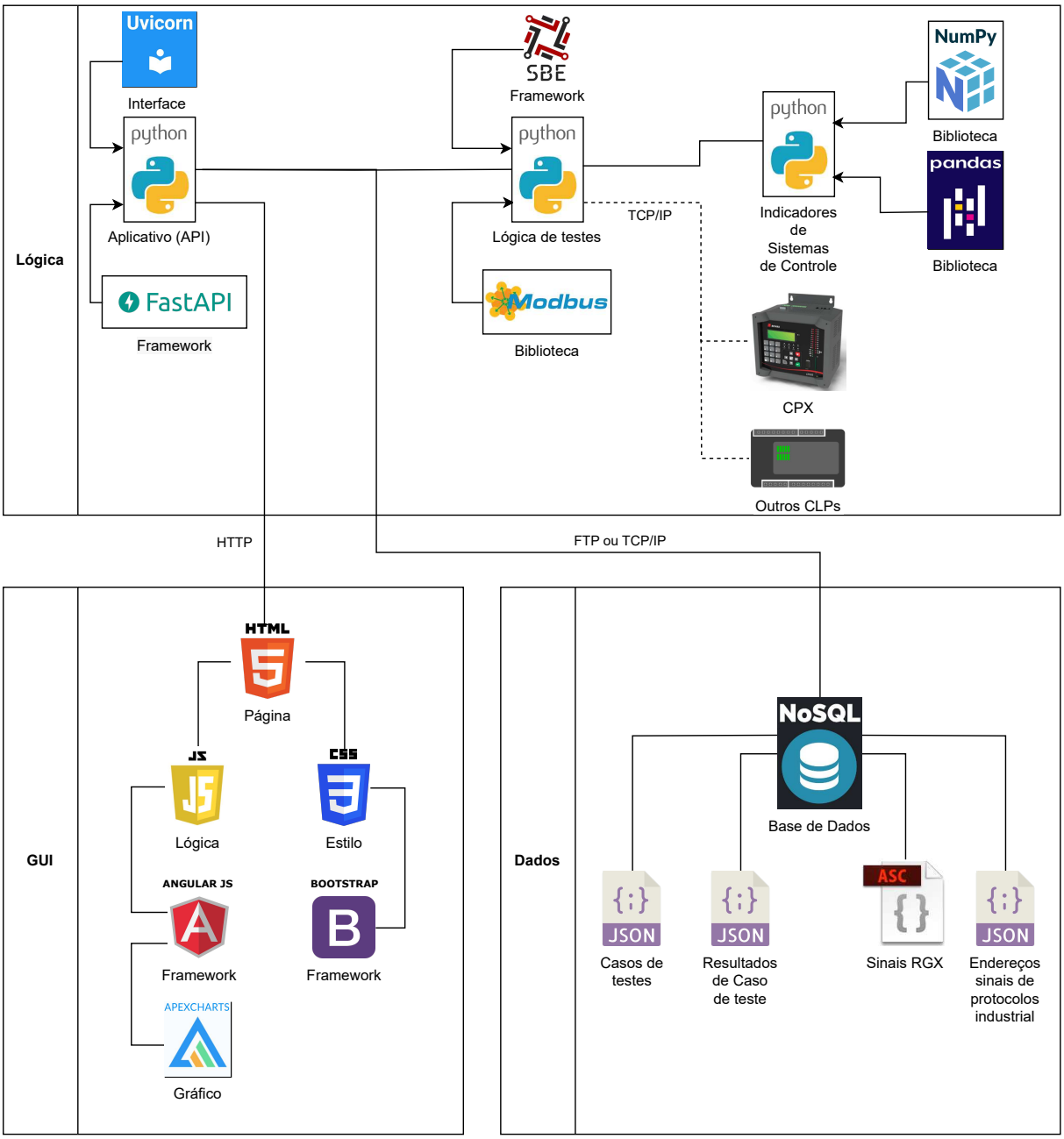
A camada de lógica de negócios conterá a lógica principal da aplicação, incluindo a criação, edição e execução de casos de teste, além da configuração de sinais e integração com o software base. Isso garantirá que as funcionalidades principais da ferramenta sejam independentes da interface do usuário.

A camada de persistência de dados será responsável pelo armazenamento e recuperação de informações, como casos de teste, configurações de sinais e registros de execução de testes. Um banco de dados relacional será usado para garantir a confiabilidade e a integridade dos dados.

Embora não seja um tipo padrão de diagrama UML, O Diagrama de Arquitetura de Software geralmente se refere a uma visão de alto nível da organização do sistema, mostrando componentes principais, seus relacionamentos e suas interações.

O diagrama da Figura 27 ilustra a arquitetura de um software que integra funcionalidades de aplicativo (API), lógica de testes, e interação com dispositivos e protocolos de comunicação.

Figura 27 – Arquitetura de Software.



Fonte: Autoria própria.

A seguir é explicado detalhadamente o diagrama:

1. **Lógica de negócios**:

   - **Aplicativo (API)**: Representa a lógica principal do software, programado em Python. Através da documentação da API, os usuários podem executar casos de teste diretamente, sem a necessidade de uma interface HTML;

- **Lógica de testes**: Programada em Python, esta parte se comunica diretamente com o aplicativo e é responsável por realizar testes e validações. A lógica de testes interage com:
  - **SBE**: O módulo de comunicação direta com o SO do CPX, comunica-se principalmente através do protocolo TCP/IP;
  - **Modbus**: Um módulo para o protocolo de comunicação para dispositivos eletrônicos, amplamente utilizado em automação industrial. Pode ser interface para CLPs adicionais, de outras fabricantes, que podem interagir com a lógica de testes;
  - **Indicadores de Sistemas de Controle**: Instrumentos que fornecem cálculos sobre a eficiência e desempenho dos sistemas de controle. Necessita das bibliotecas básicas de Python: NumPy e Pandas.

2. **GUI**: Representa a interface visual (Página HTML) do software. É composta de:
   - **Lógica JS:** Scripts em linguagem JS, incluindo o framework AngularJS, que oferecem funcionalidades interativas à página, junto à gráficos proporcionados pelo ApexCharts;
   - **Estilo CSS:** Folhas de estilo, incluindo o framework Bootstrap, que definem a aparência visual da página.
3. **Dados**: Armazena diferentes tipos de informações (NoSQL) e se comunica através de protocolo FTP ou TCP/IP com outros componentes. Esta base armazena:
   - **Casos de testes**: Descritos em formato JSON, cada caso é um arquivo;
   - **Resultados de Caso de teste**: Saídas ou resultados, também em formato JSON, após executar os casos de testes, cada resultado é um arquivo;
   - **Sinais RGX**: Informações relacionadas a sinais, armazenadas em formato ASC;
   - **Endereços sinais de protocolos industrial**: Informações sobre endereçamento para comunicação com dispositivos industriais, em formato JSON.

A arquitetura modular garante flexibilidade, permitindo que os usuários interajam diretamente através da API ou usem a interface em HTML, dependendo das suas necessidades e preferências.

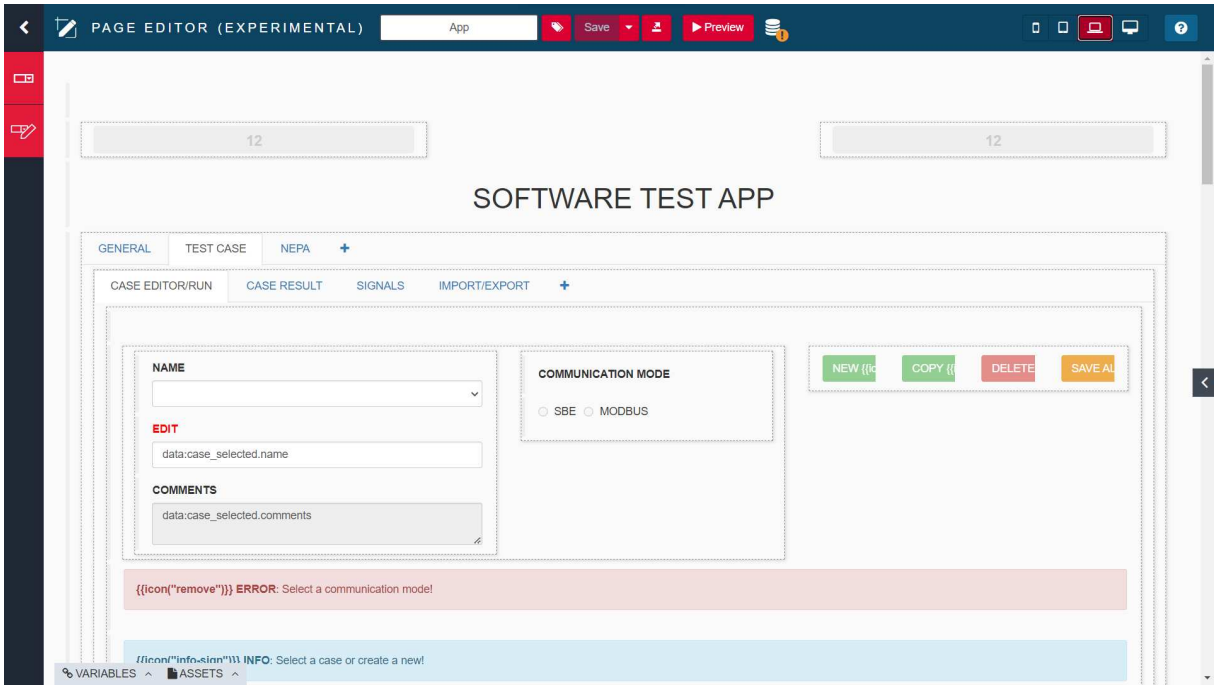
4.3 DESIGN DA INTERFACE

A página desenvolvida por meio da ferramenta UI Designer está demonstrada na Figura 28, onde ela está em modo de edição em uma aba específica. Pode-se perceber a intuitividade para futuras manutenções no desenvolvimento da interface.

Os widgets são alocados de forma que fiquem organizados em grade, e cada linha da grade tem 12 espaços nos quais os widgets podem ocupar mais de um lugar. Os widgets padrões que foram utilizados em todas as interfaces são listados a seguir.

- **Containers**: Com Bootstrap é possível a utilização de containers para realizar agrupamentos de outros widgets. Ele é ótimo para organizar, criar lógicas de conjuntos, gerenciar listas, entre outros;
- **Containers em abas**: Perfeito para criar abas na aplicação, como as expostas na Figura 28;
- **Modal Container**: Utilizado para criar modais de dicas ao longo da interface, sem poluir a área de trabalho do usuário;
- **Input**: Entrada de dados, utilizado para textos e números, armazena as informações em variáveis;
- **Text Area**: Parecido com o Input, mas oferece um campo maior para digitação de textos com quebras de linha;
- **Select**: Semelhante ao Input, mas com valores pré-definidos em objetos ou listas, o usuário deve escolher um deles;
- **Radio Buttons**: Semelhante ao Select, mas para situações com poucas opções, torna claro para o usuário as alternativas sem a necessidade de abrir uma lista de opções;
- **Upload**: Entrada para arquivos, utilizado para carregar casos e sinais;
- **Button**: Botões foram utilizados para adicionar/remover informações de listas e realizar requisições *Hypertext Transfer Protocol* (HTTP);
- **Image**: Para adicionar imagens ao corpo do HTML;
- **Text**: Para adicionar textos ao corpo do HTML;
- **Title**: Para adicionar títulos pré-formatados em HTML.

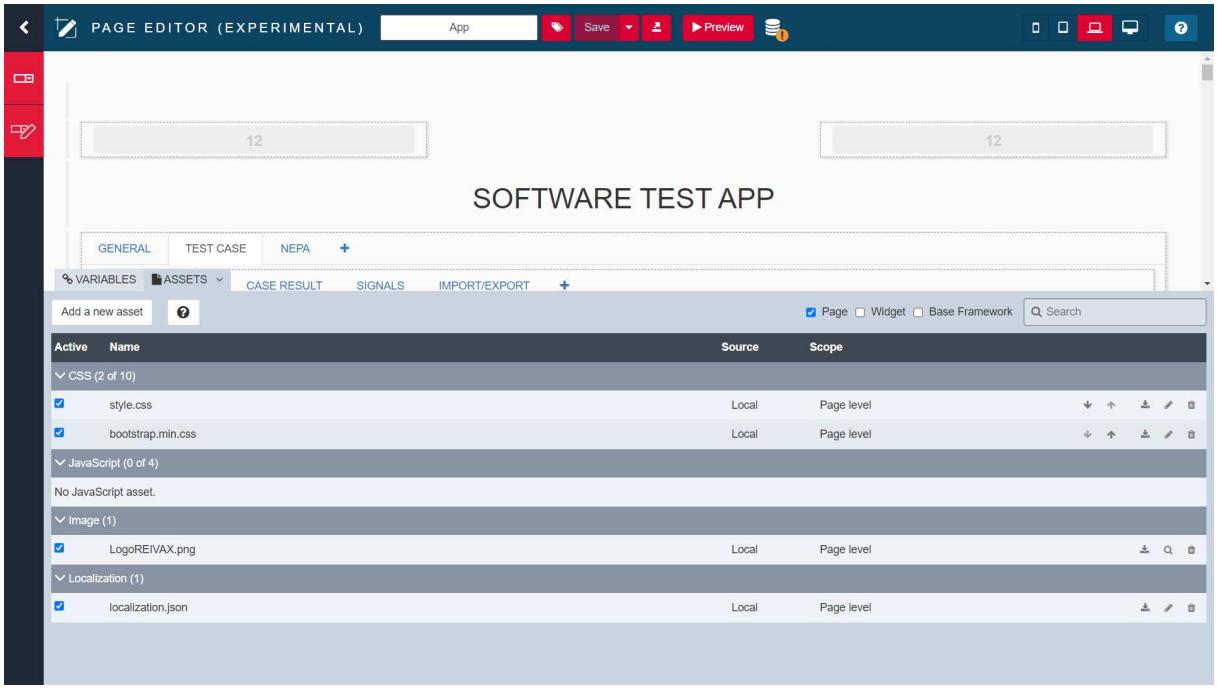
Figura 28 – Visão geral da página no editor UI Defigner.



Fonte: Autoria própria.

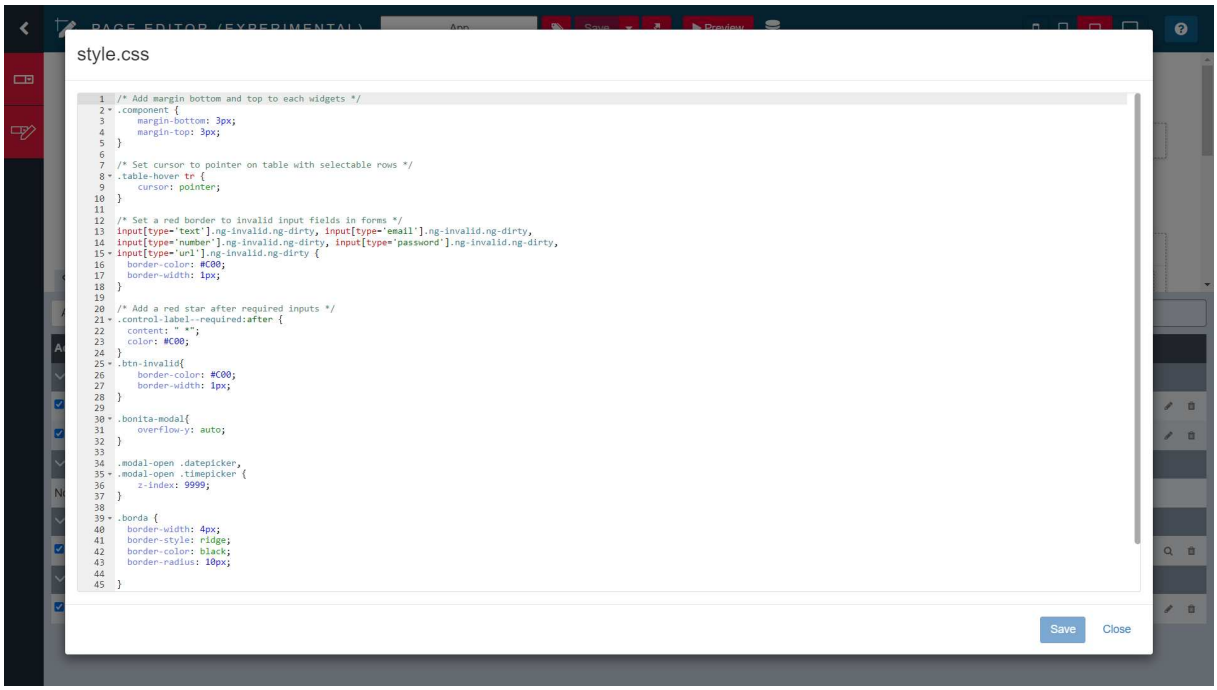
Na aba “assets” (Figura 29), foram definidas as utilizações do Bootstrap, códigos CSS (Figura 30), imagens e o arquivo de localização para traduções da interface, conforme a linguagem definida no navegador.

Figura 29 – Aba assets selecionada no editor.



Fonte: Autoria própria.

Figura 30 – Editor interno de código CSS.



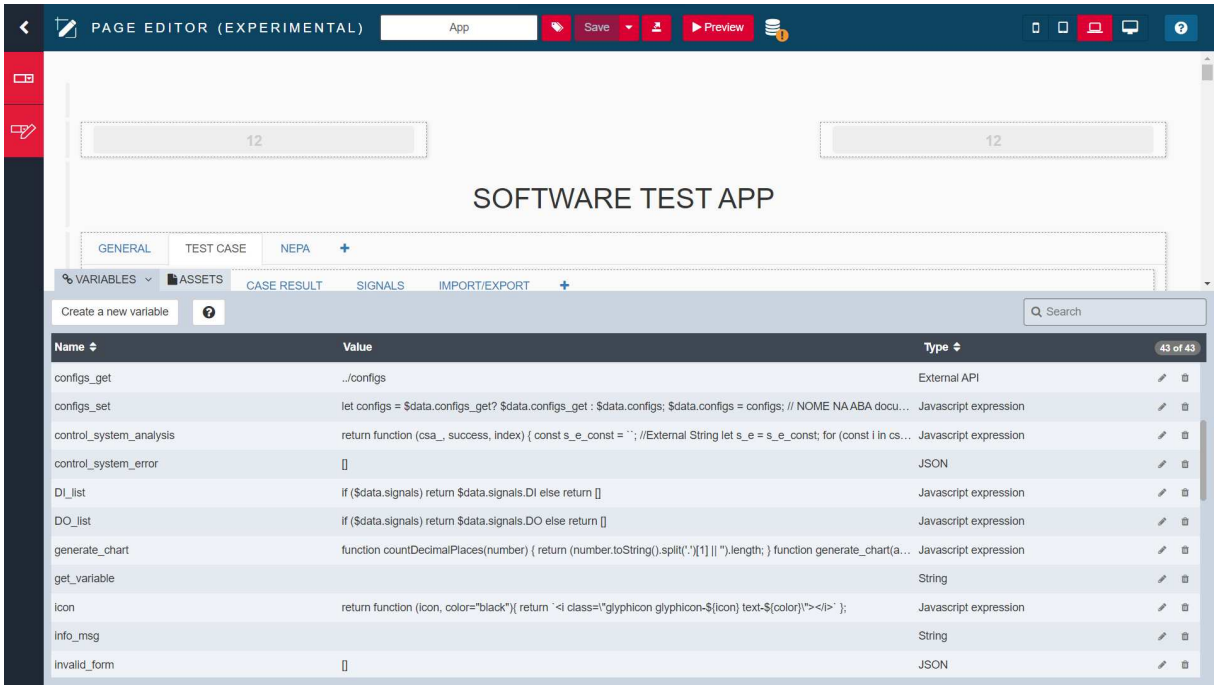
Fonte: Autoria própria.

Na Figura 31, pode-se observar a lista de todas as variáveis criadas. As variá-

veis relacionadas às requisições da API são atualizadas quando o usuário carrega a página, incluindo todos os casos, sinais, resultados, sinais RGX e configurações. As variáveis de “Javascript expression” merecem atenção especial, dada a sua importância para a aplicação. É possível criar scripts em Javascript (JS) que ampliam o leque de possibilidades, como pode ser visto na Figura 32. A seguir, é listado o que foi desenvolvido de forma mais complexa em JS:

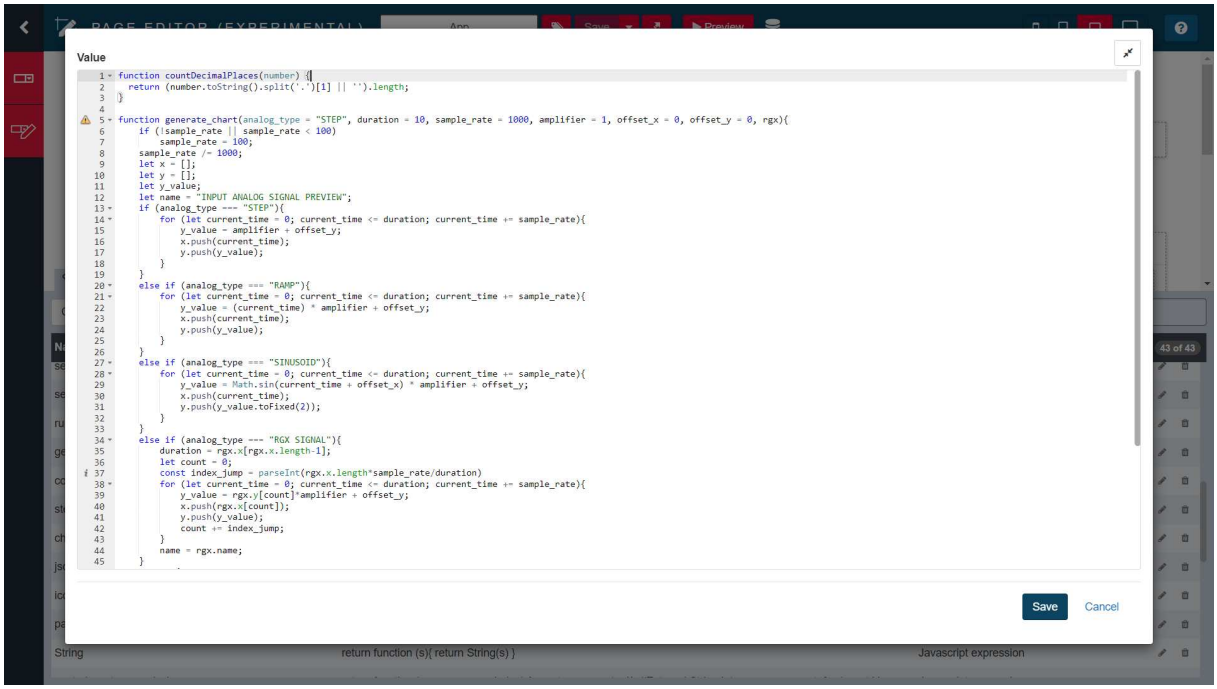
- **generate_chart** (Figura 32): Função que gera dados para a pré-visualização do sinal analógico de entrada (degrau, senoide, rampa, RGX). Conforme o usuário define diferentes parâmetros (tipo, duração, amostragem, amplificação, offsets), o gráfico é recalculado instantaneamente. Utilizada para gerar o sinal que é injetado no CLP;
- **configs_set**: Função que define a aparência do documento em HTML;
- **step_errors**: Script que gera HTML de saída para indicar os erros de digitação em cada etapa no editor de casos de teste. Caso seja identificado algum problema nos dados, não é possível executar o caso de teste. Esse script é muito importante para assegurar que os dados enviados para a execução do teste estejam corretos;
- **control_system_analysis**: Função que, a partir do resultado de um caso de teste, gera HTML para que o usuário possa compreender as informações de maneira clara, diferenciando o que está errado do que está conforme o esperado.

Figura 31 – Aba variáveis selecionada no editor.



Fonte: Autoria própria.

Figura 32 – Código JS de geração de sinal analógico de entrada aberto no editor interno.

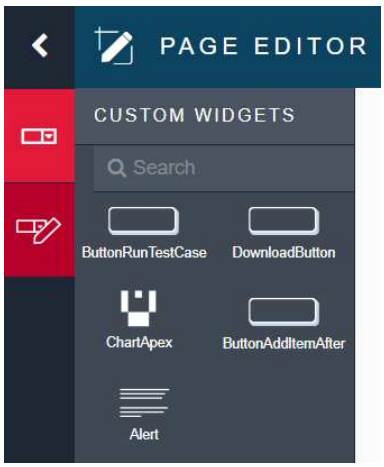


Fonte: Autoria própria.

Além dos widgets padrões da plataforma, foram desenvolvidos novos widgets, conforme demonstrado na Figura 33. A seguir, é apresentado o motivo da criação de cada um deles:

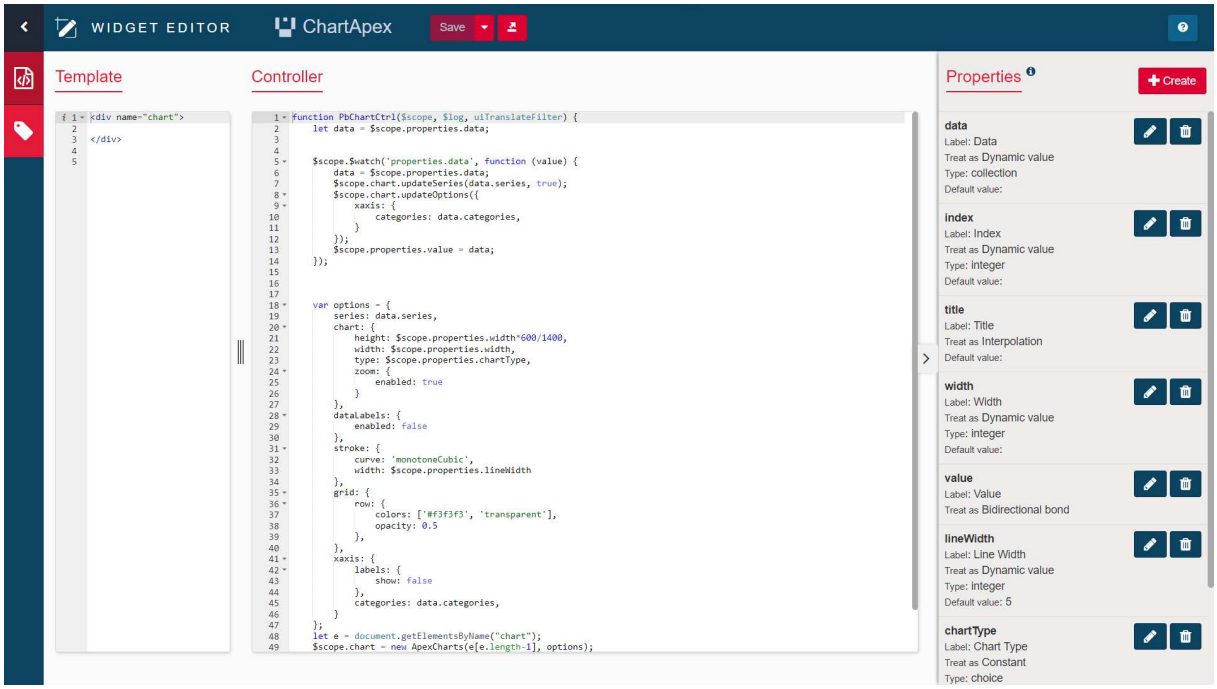
- **ButtonRunTestCase**: A execução do caso de teste exigiu a modificação do botão padrão para permitir o retorno, em tempo real, do andamento do caso;
- **DownloadButton**: Desenvolveu-se um botão específico para o download de casos e sinais, baseando-se no botão padrão;
- **ButtonAddItemAfter**: A funcionalidade de adicionar um elemento após o índice atual foi incorporada ao botão padrão, uma vez que não era possível inserir um elemento em uma lista em uma posição que não fosse a primeira ou a última;
- **Alert**: Alertas de erro, sucesso e informações foram implementados na página utilizando o padrão Bootstrap;
- **ChartApex**: O mais complexo dos elementos personalizados desenvolvidos, que requereu a consulta à documentação (`https://apexcharts.com/docs/installation/`) para sua integração com o AngularJS por meio do UI Designer. A Figura 34 apresenta a tela de desenvolvimento do gráfico ApexCharts como um widget customizável.

Figura 33 – Aba de widgets customizados.



Fonte: Autoria própria.

Figura 34 – Ambiente de desenvolvimento de um widget customizável, na tela aparece o código aberto ApexCharts.



Fonte: Autoria própria.

4.4 IMPLEMENTAÇÃO

A seção de Implementação marca a fase em que o projeto de software toma forma e é transformado em um sistema funcional. Nesta seção, discutiremos a implementação das principais funcionalidades do STA.

Aplicação (API)

O código Python a seguir é um script para criar uma aplicação web utilizando a biblioteca FastAPI.

```
1 import os
2 import json
3 import webbrowser
4 import caf_test_logic
```

Essas são as importações de módulos padrão do Python e de um módulo personalizado `caf_test_logic` que é apresentado em sua subseção. `os` é utilizado para interagir com o sistema operacional, `json` para manipular dados em formato JSON, `webbrowser` para abrir uma janela do navegador e `caf_test_logic` é o módulo específico para a lógica de teste do CAF.

```
1 from fastapi import FastAPI, HTTPException, Request, UploadFile
```

```
from fastapi.responses import JSONResponse
from starlette.responses import FileResponse
from starlette.staticfiles import StaticFiles
from urllib.parse import unquote
from uvicorn import run
```

Aqui são importadas classes e funções de várias bibliotecas relacionadas à criação de APIs web:

1. `FastAPI`: Principal classe para construir o objeto do servidor API;
2. `HTTPException`: Usado para lidar com exceções de HTTP;
3. `Request`: Obtém informações de solicitações HTTP;
4. `UploadFile`: Lida com transferência arquivos enviados ao servidor;
5. `JSONResponse`: Resposta com dados em formato JSON;
6. `FileResponse`: Classe para enviar arquivos como respostas;
7. `StaticFiles`: Usado para servir arquivos estáticos;
8. `unquote`: Função para decodificar percentuais de *Uniform Resource Locator* (URL)s. Exemplo: `https://www.google.com/search?q=a%C3%A7%C3%A3o` $\Longrightarrow$ `https://www.google.com/search?q=ação`;
9. `run`: Função para executar o servidor *Asynchronous Server Gateway Interface* (ASGI).

```
IP = "localhost"
PORT = 8000
app = FastAPI()
```

Estas linhas definem o endereço IP e a porta para o servidor web e criam uma instância do aplicativo.

```
app.mount("/resources", StaticFiles(directory="resources"), name="resources")
```

Este comando serve os arquivos estáticos localizados no diretório `resources` no caminho “/resources” da aplicação web. Sem ele, o STA não forneceria através de sua URL, o acesso aos arquivos.

```
@app.get("/")
async def get_index():
    return FileResponse("resources/index.html")
```

Aqui é definido um caminho da aplicação web (endpoint) para a raiz ("/"), que retorna o arquivo `index.html` localizado no diretório `resources` como uma resposta.

```
if __name__ == "__main__":
    webbrowser.open(f"http://{IP}:{PORT}", new=2)
    run(app, host=IP, port=PORT)
```

Esta é a parte do script que verifica se o módulo está sendo executado como o programa principal e, se estiver, abre uma nova aba no navegador web padrão e inicia o servidor usando o endereço IP e a porta especificados.

Requisições HTTP da Aplicação

Em uma API, os métodos HTTP GET e POST são usados para comunicar-se com o servidor. O método GET é utilizado para solicitar dados de um recurso específico, enquanto que o método POST é utilizado para enviar dados para serem processados por um recurso específico.

O método GET é idempotente, o que significa que múltiplas solicitações idênticas devem ter o mesmo efeito que uma única solicitação e não devem alterar o estado do servidor. Em FastAPI, uma operação GET pode ser implementada da seguinte maneira:

```
@app.get("/items/{item_id}")
async def read_item(item_id: int):
    return JSONResponse(content={"item_id": item_id}, status_code=200)
```

Neste exemplo, ao acessar o caminho “/items/item_id” com o método GET, o cliente espera receber uma resposta que contém um `item_id`.

O método POST não é idempotente, o que significa que enviar uma solicitação POST idêntica várias vezes pode resultar em diferentes efeitos e alterações no estado do servidor. Um exemplo de um método POST em FastAPI é o seguinte:

```
app = FastAPI()

class Item(BaseModel):
    name: str
    description: str = None
    price: float
    tax: float = None

@app.post("/items/")
async def create_item(item: Item):
    return item
```

Aqui, uma solicitação POST para “/items/” com um corpo de solicitação que contém um nome, descrição, preço e imposto (opcional) resultará na criação de um novo item e a resposta será o item criado.

As requisições GET e POST desenvolvidas no projeto são enumeradas e explicadas nessa lista:

1. **GET /**
   Retorna a página inicial HTML, servindo o arquivo “index.html”. Este endpoint não recebe nenhum dado, apenas devolve o arquivo como uma resposta de arquivo;

2. **GET /configs**
   Fornece uma resposta JSON com várias configurações que podem ser usadas pela página. Estas configurações incluem títulos de documentos, valores possíveis para tipos de sinais, condições de parada e outros elementos exclusivos da página HTML;

3. **GET /signal/export**
   Exporta um arquivo JSON contendo sinais, se existir. Se o arquivo não for encontrado, lança uma exceção HTTP 404 (não encontrado);

4. **GET /signal/load**
   Carrega o arquivo JSON dos sinais e retorna seu conteúdo. Se houver algum erro na leitura do arquivo, como um arquivo não encontrado ou um erro geral, uma exceção HTTP é lançada;

5. **POST /signal/save**
   Recebe dados de um sinal via JSON e salva no arquivo “signals.json”. Se algo der errado durante a gravação, uma exceção HTTP 500 (erro interno ao servidor) é lançada;

6. **POST /signal/upload**
   Permite o upload de um arquivo JSON contendo dados de sinais. O arquivo é verificado quanto ao formato correto e, em seguida, salvo, sobrescrevendo o objeto de sinais existente, se apropriado;

7. **GET /case/export**
   Exporta um caso específico como um arquivo JSON baseado no nome do caso passado como um parâmetro de consulta. Se o arquivo não for encontrado, uma exceção HTTP 404 é lançada.

8. **GET /case/load**
   Carrega e retorna todos os casos salvos como um array JSON. Todos os arquivos JSON dentro do diretório especificado são lidos e retornados.

9. **POST /case/save**
   Salva múltiplos casos passados no corpo da requisição como arquivos JSON separados, removendo os arquivos antigos existentes no diretório de casos.

10. **POST /case/upload**
    Permite o carregamento de um arquivo JSON para um caso. O nome do caso é extraído dos dados do arquivo e é usado para nomear o arquivo JSON salvo.

11. **GET /case/results**
    Retorna todos os resultados dos casos como um array JSON. Semelhante ao carregamento dos casos, mas buscando em um diretório diferente.

12. **GET /rgx**
    Carrega arquivos com a extensão ".ASC" a partir de um diretório específico e os processa para extrair dados de série temporal, retornando-os como objetos JSON.

13. **POST /case/run**
    Inicia a execução de um caso com base nos dados recebidos no corpo da requisição. Retorna os resultados da execução como JSON.

14. **GET /project_info**
    Busca informações sobre o projeto. Se houver um erro ao obter essas informações, uma resposta JSON com um código de status HTTP 500 é retornada.

15. **GET /setvar**
    Define uma variável no projeto com um valor especificado nos parâmetros de consulta. Retorna uma resposta JSON com a variável e o valor definido.

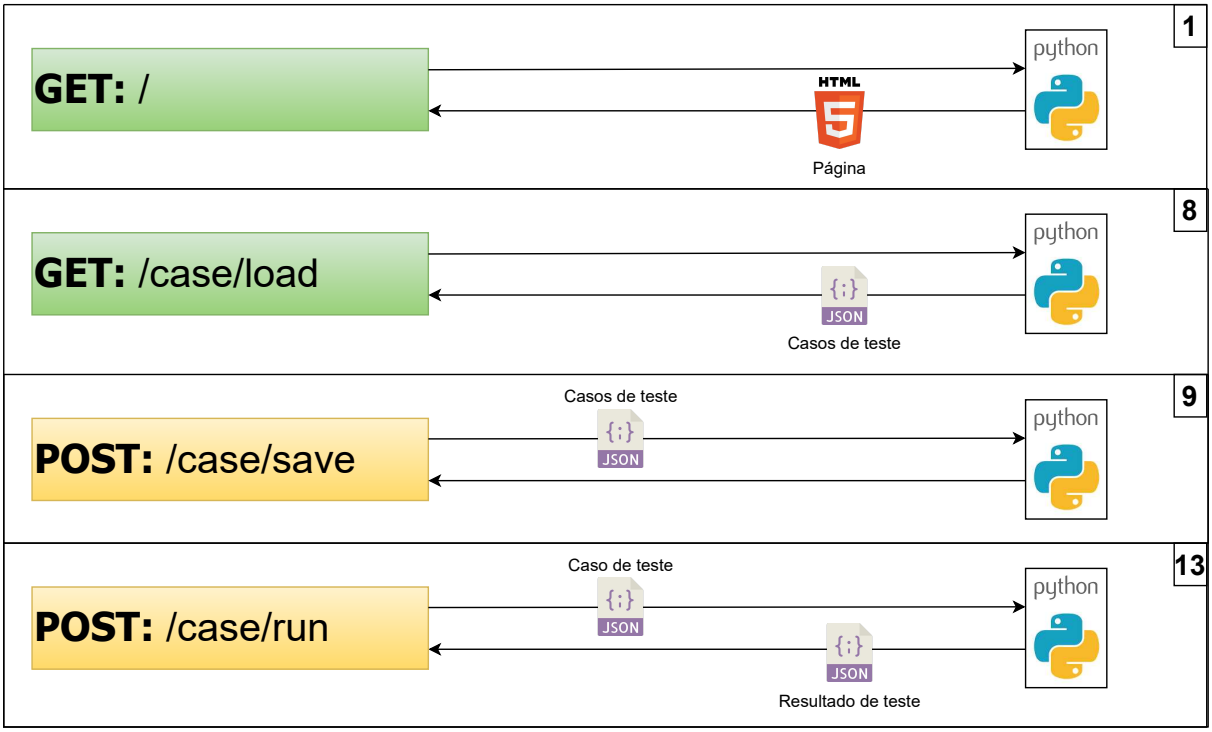
16. **GET /getvar**
    Obtém o valor de uma variável especificada no projeto e retorna esse valor em uma resposta JSON.

17. **POST /NEPAconnection**
    Estabelece uma conexão com um projeto NEPA, com o caminho do projeto passado no corpo da requisição. Retorna uma resposta JSON indicando o sucesso ou falha da operação;

18. **POST /NEPAunconnection**
    Desestabelece a conexão com NEPA. Retorna uma resposta JSON indicando o sucesso ou falha da operação;

Alguns exemplos das requisições apresentadas em diagrama são disponibilizados na Figura 35.

Figura 35 – Requests HTTP.



Fonte: Autoria própria.

Lógica de Testes

Esta seção fornece uma visão geral das funções definidas no módulo utilizado para automação de testes de sistemas de controle, comunicação com equipamentos, coleta e análise de dados de sinais analógicos e digitais.

```
async def step_input(i: int, communication_mode: str, signal_type: str, digital:
    object, digital_type: str, analog: object, analog_type: str, amplifier: float):
    #...
```

Envia um sinal analógico ou digital durante a execução do passo. Compatível com dois modos de comunicação: SBE e Modbus.

```
def prepare_series(i, communication_mode, analog, digital):
    #...
```

Prepara e inicializa as séries para armazenar dados de saída para sinais analógicos e digitais baseado no modo de comunicação.

```
async def verify_signal_gt(communication_mode, stop_condition_value_type,
    condition_signal, condition_value):
    #...

async def verify_signal_lt(communication_mode, stop_condition_value_type,
    condition_signal, condition_value)
```

```
    #...
```

Verifica se o valor do sinal está acima ou abaixo de um determinado limiar. Usado como uma condição de parada.

```
async def verify_stop(time: float, communication_mode: str, stop_conditions: str,
    condition_time: float, condition_signal: object, condition_value: float,
    condition_operator: str, stop_condition_value_type: str):
    #...
```

Avalia a condição de parada com base no tempo, valor do sinal, ou ambos.

```
async def case_run(r):
    #...
```

Principal função, executa o caso de teste completo, incluindo o envio de sinais, coleta de dados de saída e análise de sistema de controle.

```
async def NEPAconnection(project_path: str):
    #...
async def NEPAunconnection():
    #...
```

Estabelece conexão com o ambiente NEPA, carrega o arquivo de mapa do projeto e conecta os comandos, e também desconecta do ambiente NEPA e seus serviços relacionados, se necessário.

```
async def getvar(var: str):
    #...
async def setvar(var: str, value: float):
    #...
```

Obtém e envia valor para uma variável específica no ambiente NEPA.

Modbus

```
from pyModbusTCP.client import ModbusClient

IP = "127.0.0.1"
PORT = 502

client = ModbusClient(host=IP, port=PORT, auto_open=True,
                      debug=False, auto_close=False)
```

Configura o cliente Modbus para conectar-se ao servidor no IP e porta especificados, com opções para abrir automaticamente a conexão e mantê-la aberta entre operações.

```
def precision(number):
    #...
```

Calcula a quantidade de casas decimais em um número flutuante, que é útil para definir a precisão na leitura de sinais analógicos.

```
def read_digital(signal):
    #...
```

Lê o valor de um sinal digital a partir do endereço e bit especificados no dispositivo Modbus.

```
def read_analog(signal):
    #...
```

Lê o valor de um sinal analógico, aplica um multiplicador para escala e ajusta a precisão conforme necessário.

```
def write_digital(signal):
    #...
```

Escreve um valor digital no endereço e bit especificados no dispositivo Modbus.

```
def write_analog(signal, value):
    #...
```

Escreve um valor analógico no endereço especificado no dispositivo Modbus após aplicar o multiplicador para a conversão de escala.

Cálculo de Indicadores de Sistemas de Controle

Outro módulo que apresenta um conjunto de funções projetadas e programadas neste trabalho para a análise de Sistemas de Controle. Os métodos implementados permitem determinar métricas importantes, como tempo de subida, tempo de acomodação, erro em regime permanente e sobressinal.

```
import pandas as pd

def moving_average_filter(signal_list, window_size):
    df = pd.DataFrame({'signal': signal_list})
    df['smoothed'] = df['signal'].rolling(window=window_size, min_periods=1).mean()
    return df['smoothed'].tolist()
```

Importa o módulo pandas e aplica um filtro de médias móveis em uma lista de sinais para suavizar os dados.

```
import numpy as np

def rise_time_analysis(expected_rise_time, final_value, time, response, filter=False
    , window_size=10):
    # ...
```

Calcula o tempo de subida do sinal de resposta de um sistema de controle e verifica se está dentro de uma tolerância esperada.

```
def settling_time_analysis(expected_settling_time, final_value, time, response,
    filter=False, window_size=10):
    # ...
```

Determina o tempo de acomodação do sinal e avalia se atende aos critérios especificados.

```
def steady_state_error_analysis(expected_steady_state_error, final_value, response,
    filter=False, window_size=10):
    # ...
```

Calcula o erro em regime permanente em comparação com o valor final esperado.

```
def overshoot_analysis(expected_overshoot, final_value, response, filter=False,
    window_size=10):
    # ...
```

Avalia o sobressinal do valor final que ocorre na resposta do sistema.

```
def control_system_analysis(analog_signals, x, y):
    # ...
```

Executa uma análise completa de um sistema de controle, considerando vários sinais analógicos e retornando uma compilação dos resultados de todas as métricas de desempenho analisadas.

5 ESTUDO DE CASO E VALIDAÇÃO

A fase de validação do STA foi realizada em conjunto com diversos profissionais no Setor de Software, sendo eles o primeiro público-alvo dentro da empresa. Esta visa garantir que o software atenda aos requisitos especificados e funcione corretamente em seu ambiente de operação.

Este capítulo destaca o funcionamento do STA, associado à construção de um caso de teste real.

5.1 INICIANDO O APLICATIVO

O STA é entregue com um diretório raiz composto por *data*, *resources* e um executável, nomeado conforme a sigla, versão e revisão (de acordo com o padrão estabelecido pela REIVAX em seus produtos internos), como pode ser visualizado na Figura 36.

Em *data*, são armazenados dinamicamente os arquivos JSON utilizados pela aplicação. Já em *resources*, encontram-se os componentes do GUI, compilados pelo UI Designer. O executável, com cerca de 12 megabyte (MB), inclui o interpretador Python e permite operação em qualquer computador moderno, independente de instalações adicionais.

Figura 36 – Arquivos do STA.

Fonte: Autoria própria.

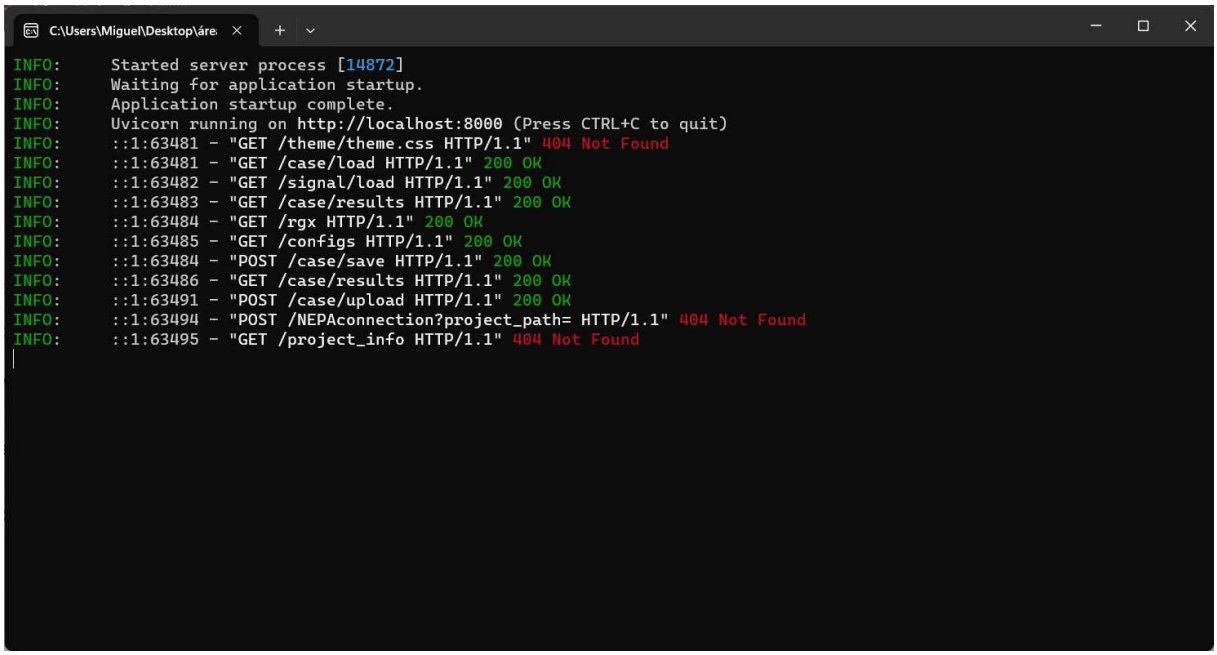
Ao iniciar o STA, o usuário visualiza uma janela com o log do servidor API em tempo real e o navegador é aberto automaticamente na página inicial. Conforme a Figura 37, essa janela exibe as requisições HTTP efetuadas, indicando códigos de estado como 200 (OK) ou 404 (Não encontrado), o que é crucial para o reporte de erros em codificação Python e facilita o *debugging*[1].

Durante a execução do caso de teste, o usuário pode acompanhar a amostragem e o estado atual por meio da janela, proporcionando total transparência sobre o

[1] *Debugging*, ou depuração, é o processo de localizar e reduzir defeitos em um aplicativo de software.

processo em andamento. Neste Produto Viável Mínimo (MVP), ocorre a ausência de informações em tempo real na interface HTML.

Figura 37 – Janela de execução da API.



Fonte: Autoria própria.

Assim, o STA fica acessível no navegador pelo IP local e na porta padrão 8000, estando pronto para uso.

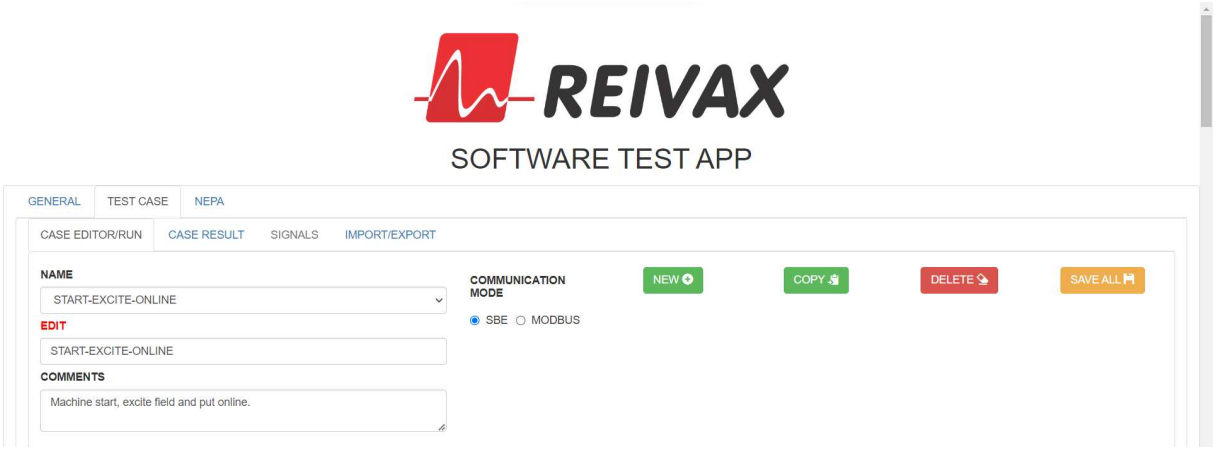
5.2 CRIANDO UM CASO

Com a aba *TEST CASE* selecionada, a Figura 38 mostra o caso “START-EXCITE-ONLINE” escolhido. O modo de comunicação definido é “SBE”, correspondendo inicialmente a uma simulação local do SEC. A seleção do modo de comunicação desencadeia alterações dinâmicas na interface, que incluem campos de texto para SBE, Modbus e caixas de seleção.

O exemplo a seguir é comum para projetistas de software de controle e supervisão durante simulações de projeto. Considerando um RTVX, o projetista inicia o processo ao pressionar o botão de partida. Depois, deve-se aguardar até que a turbina atinja a velocidade nominal. Em seguida, em uma tela diferente, o projetista ativa o campo do gerador ao pressionar o botão de excitação. Uma vez que a tensão terminal nominal é alcançada, o contator de grupo é fechado, simulando o gerador sincronizando com a rede.

As informações preenchidas nas três etapas do caso, traduzidas agora para o STA, consistem em:

Figura 38 – Caso de teste criado e selecionado.



Fonte: Autoria própria.

1. **Primeira Etapa**:
    - Condições de parada: `TIME && SIGNAL VALUE`
    - Sinal de condição para parada: `RTX_LOG.O_PER_EXT_EXC` (Permissível de excitação)
    - Sinal e Valor final (nominal) analógico: `SYS_AO.I_W_Hz = 60` (Sinal de velocidade em hertz (Hz))
    - Comportamento esperado (definidos pelo usuário):
        - Tempo de subida: 20 ± 0.1 s
        - Tempo de acomodação: 55 ± 5 s
        - Erro em regime permanente: 0 ± 1 %
        - Sobressinal: 10 ± 1 %
    - Sinal digital monitorado: `RTX_LOG.O_PER_EXT_EXC` (Permissível de excitação)
    - Tempo de condição: 90
    - Tipo de valor de condição de parada: DIGITAL (1)
    - Tipo de sinal de entrada: DIGITAL
    - Tipo digital de entrada: CMD
    - Sinal digital de entrada: `RVX_CMD.I_CMD_START_AUTO` (Comando de partida)
    - Repetições: 1
2. **Segunda etapa:**
    - Condições de parada: `TIME`

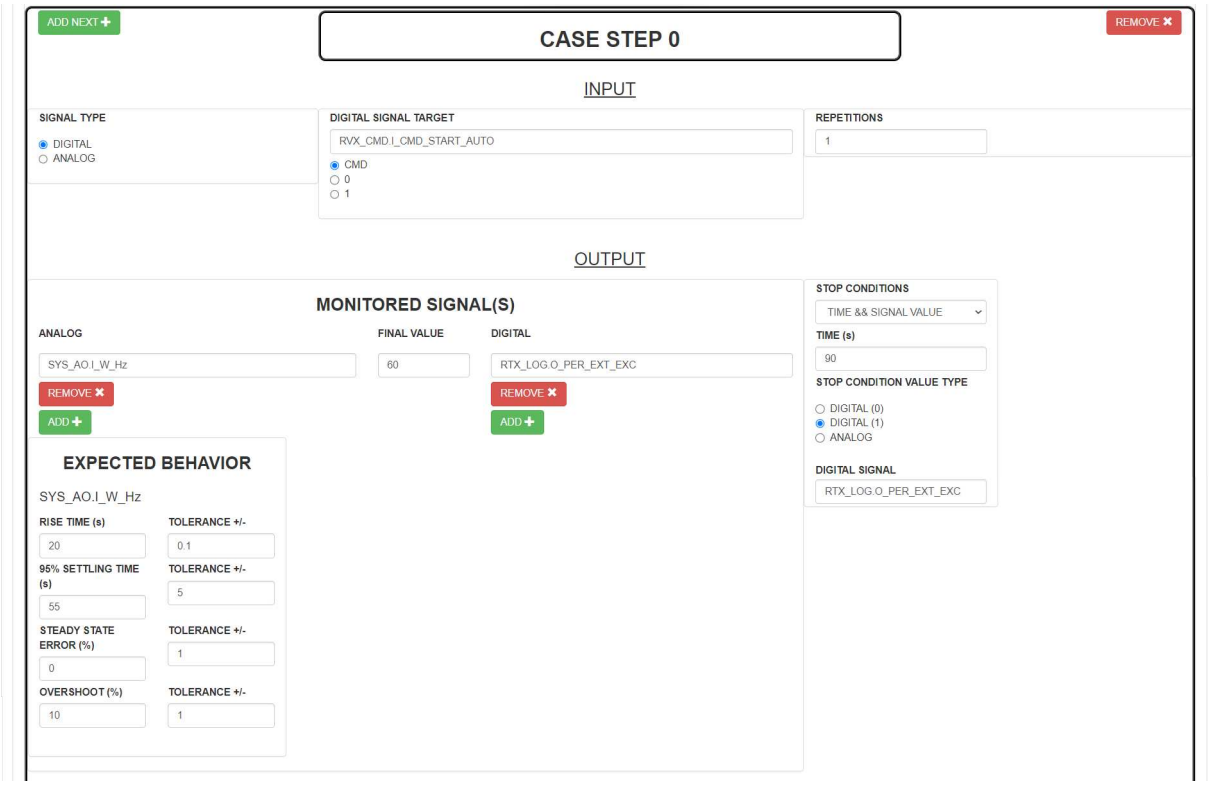
- Sinal e Valor final (nominal) analógico: `SYS_AO.I_Vt_kV = 13.8` e `SYS_AO.I_Ifd_A = 1000` (Sinal de tensão terminal em quilovolt (kV) e Corrente de campo em ampère (A))
- Comportamento esperado para ambos os sinais analógicos:
  - Tempo de subida: 10 $\pm$ 10 % e 6 $\pm$ 1 %, respectivamente
  - Tempo de acomodação: 10 $\pm$ 10 % e 30 $\pm$ 1 %, respectivamente
  - Erro em regime permanente: 10 $\pm$ 10 % e 5 $\pm$ 1 %, respectivamente
  - Sobressinal: 10 $\pm$ 10 % e 6 $\pm$ 1 %, respectivamente
- Tempo de condição: 30 s
- Tipo de sinal de entrada: DIGITAL
- Sinal digital de entrada: `RTX_CMD.I_CMD_EXC_AUTO` (Comando de excitação)
- Tipo digital de entrada: CMD
- Repetições: 1

3. **Terceiro Elemento:**

- Condições de parada: `TIME`
- Sinal e Valor final (nominal) analógico: `SYS_AO.I_Pe_MW = 10` (Sinal de potência ativa em megawatt (MW))
- Comportamento esperado:
  - Tempo de subida: 10 $\pm$ 10 s
  - Tempo de acomodação: 70 $\pm$ 10 s
  - Erro em regime permanente: 10 $\pm$ 10 %
  - Sobressinal: 10 $\pm$ 10 %
- Tempo de condição: 90 s
- Tipo de sinal de entrada: DIGITAL
- Sinal digital de entrada: `Simul_RTVX.I_CMD_FecharDisjuntorGrupo_Man` (Comando de fechar disjuntor de grupo manual)
- Tipo digital de entrada: 1
- Repetições: 1

A Figura 39 demonstra as informações inseridas na primeira etapa.

Figura 39 – Primeira etapa preenchida.



Fonte: Autoria própria.

5.3 SINAIS

A aba “SIGNALS” só é habilitada quando Modbus é selecionado. A Figura 40 ilustra as configurações padrão, que incluem todos os sinais do CAF. O usuário precisa adicionar novos sinais apenas se o projeto em teste exigir edição dos mesmos.

A Figura 41 exibe a equivalência dos sinais Modbus para a etapa ilustrada na Figura 39.

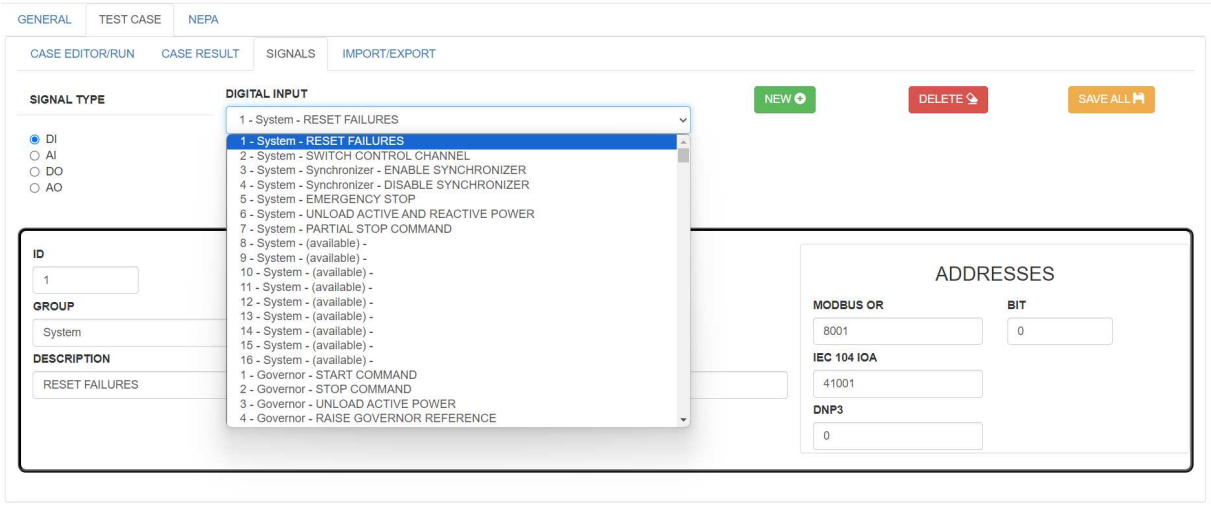
5.4 EXECUÇÃO E ANÁLISE DE RESULTADOS

A execução de um caso ocorre na mesma interface de edição, onde o usuário insere um código de projeto como um tipo de referência, que posteriormente serve como atributo para a organização dos resultados. Diferentes projetos podem apresentar comportamentos variados, sem que estejam errados.

O STA informa o usuário sobre o sucesso ou falha na execução dos casos. A Figura 42 mostra um caso que foi concluído com sucesso.

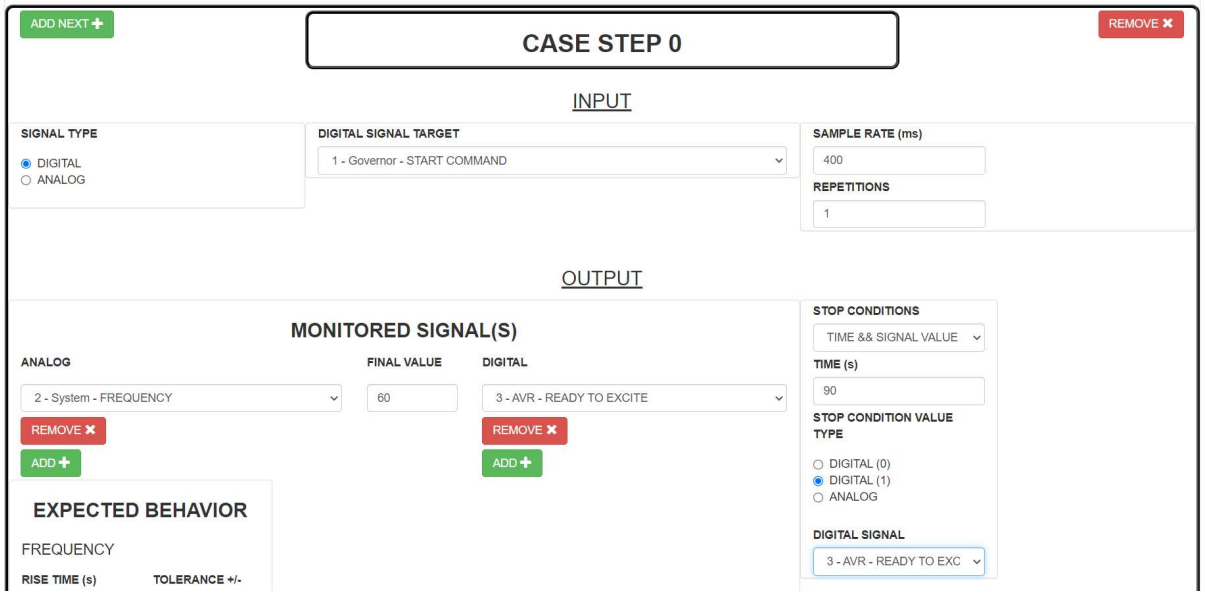
Na aba de resultados (Figura 43), o usuário encontra um selecionador de resultados. O botão “LOAD” serve para refazer a requisição de resultados, caso estes não estejam listados. Logo abaixo, as informações gerais do resultado selecionado são

Figura 40 – Aba de gerenciamento de sinais.



Fonte: Autoria própria.

Figura 41 – Etapa utilizando protocolo industrial.



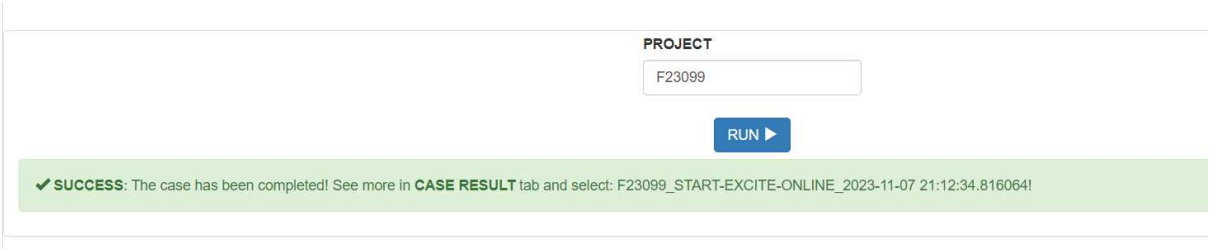
Fonte: Autoria própria.

exibidas.

Ao rolar a página para baixo, o usuário encontra informações específicas daquela etapa, incluindo indicadores de sucesso e falha, conforme ilustrado na Figura 44.

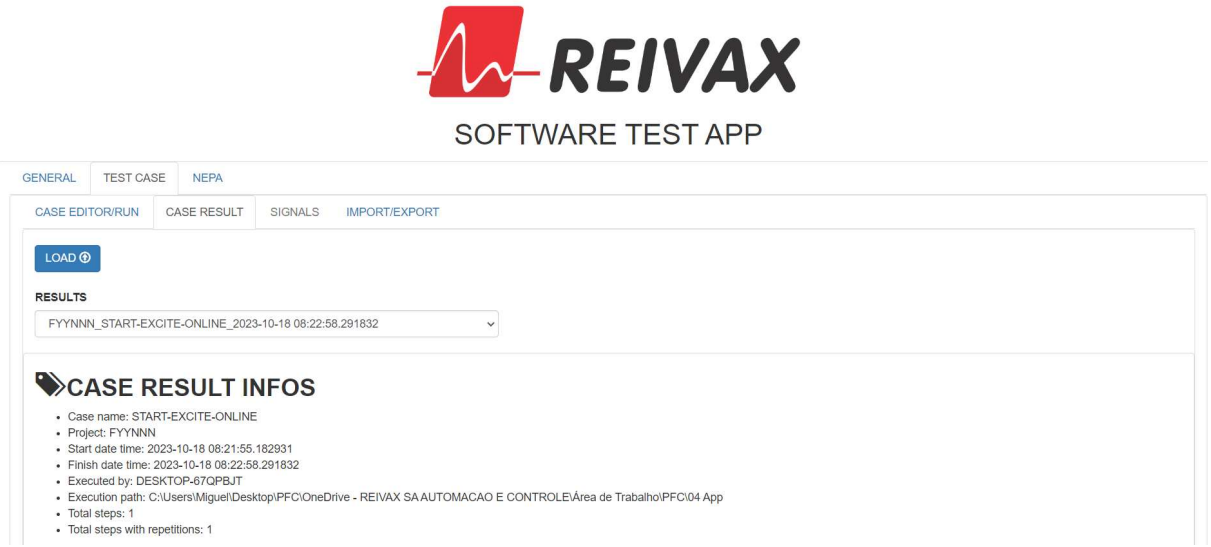
Observou-se um erro no indicador de frequência: o tempo de subida definido foi de 20 s, com uma tolerância de $\pm$ 0.1 s. Essa margem restrita resultou em um erro, e o teste não foi prosseguido, isso porque o tempo real de 20.9 s excedeu o limite

Figura 42 – Mensagem de caso encerrado com sucesso.



Fonte: Autoria própria.

Figura 43 – Resultado selecionado.



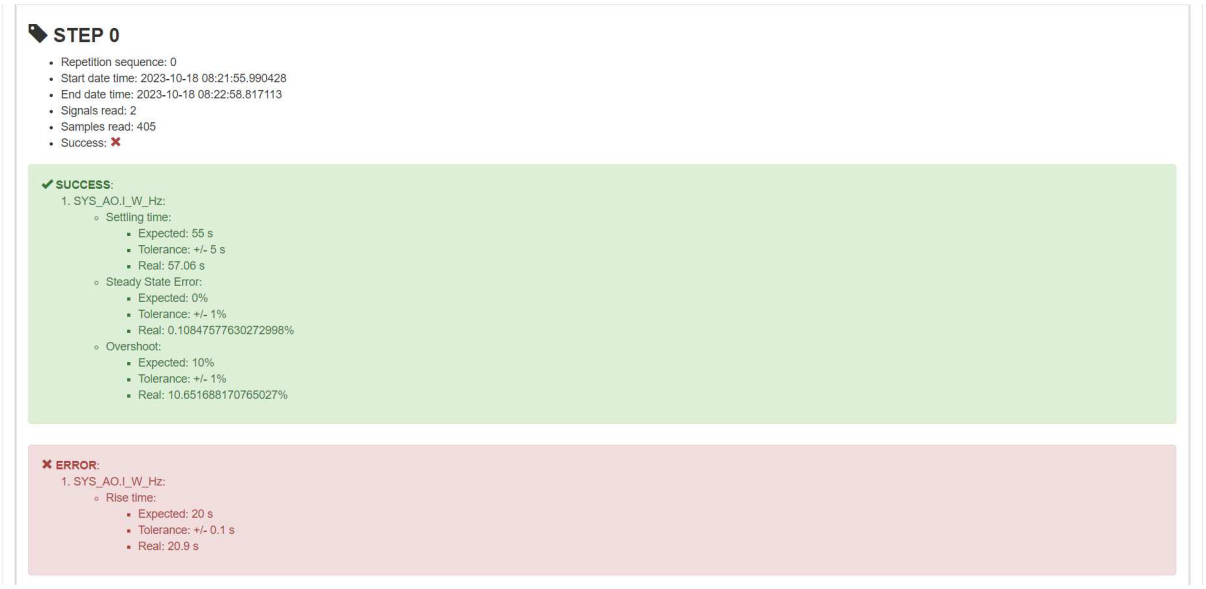
Fonte: Autoria própria.

estabelecido. Para a primeira execução do caso, recomenda-se que o usuário defina uma tolerância mais alta e realize a simulação localmente (sem sistema real). Após a conclusão, deve-se analisar os indicadores e ajustar os valores de tolerância para níveis mais realistas.

A Figura 45 exibe os gráficos do sinal de frequência analógico e do sinal de excitação digital permitido. Em situações onde múltiplos gráficos são exibidos em um único *frame*, é possível visualizá-los separadamente, ajustando a renderização para o novo contexto. Por exemplo, pode-se remover o sinal analógico para focar no sinal digital. Ferramentas adicionais de inspeção de gráficos estão disponíveis, incluindo *zoom in/out*, arraste, botão *home*, e opções para salvar o gráfico em formatos *Portable Network Graphic* (png), *Scalable Vector Graphics* (svg) e/ou *Comma-separated values* (csv).

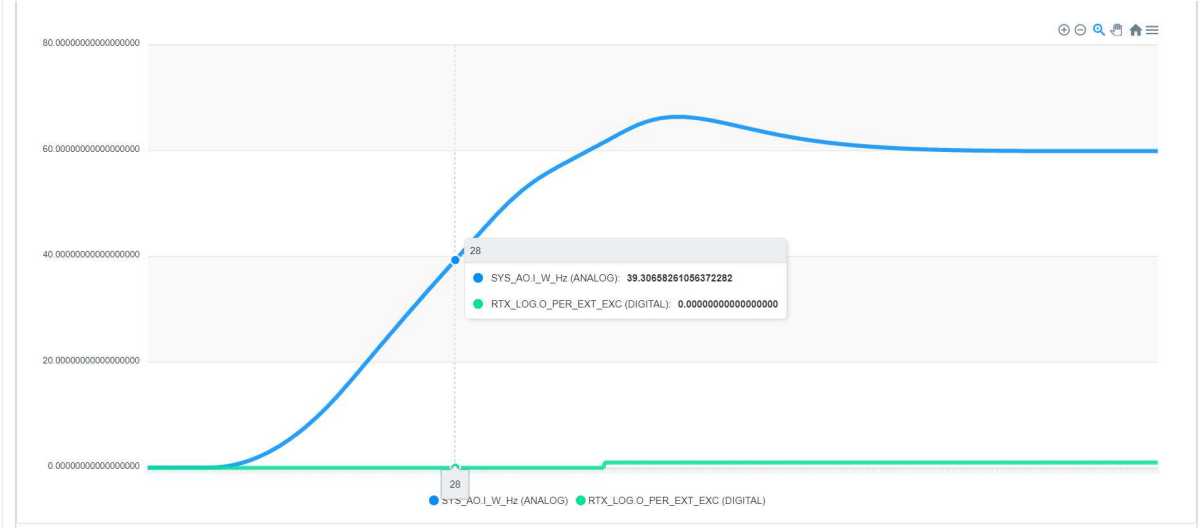
Ao preencher novamente o caso de teste proposto e ajustar o valor de tole-

Figura 44 – Informações e resultados dos indicadores.



Fonte: Autoria própria.

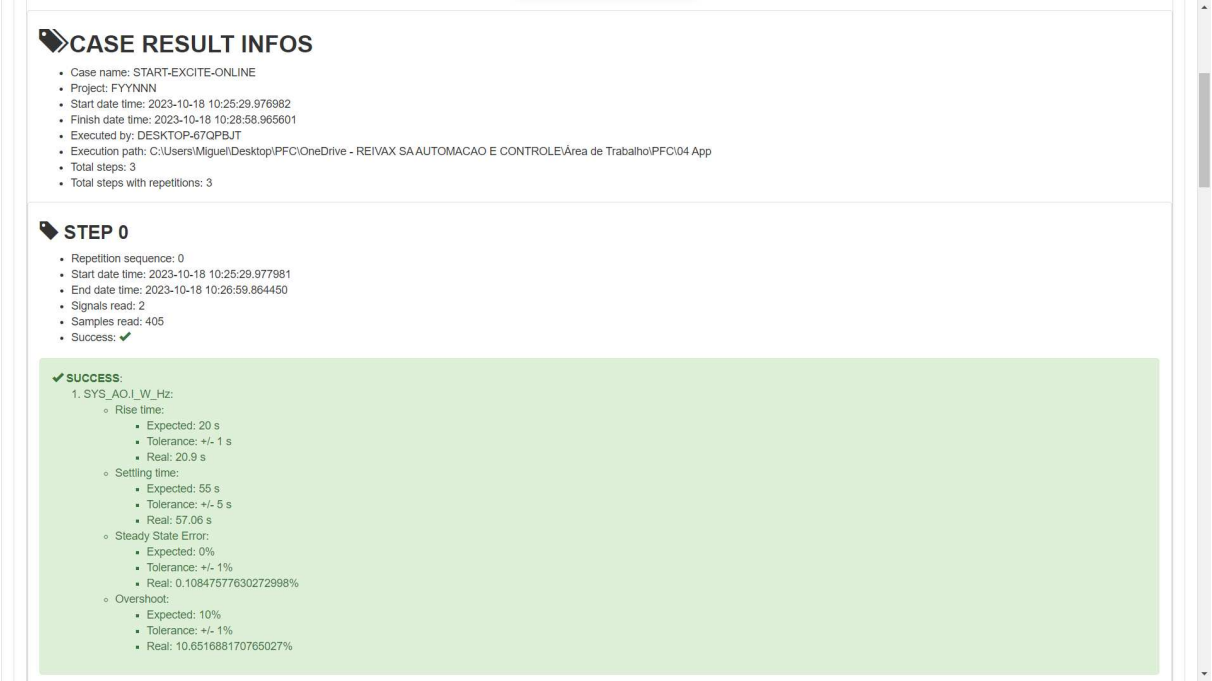
Figura 45 – Resultado gráfico.



Fonte: Autoria própria.

rância para 1 s, o teste foi bem-sucedido, como mostrado na Figura 46. Com essa alteração, os erros de indicadores foram eliminados, permitindo a execução dos casos subsequentes.

Figura 46 – Novo resultado com tolerância corrigida.



Fonte: Autoria própria.

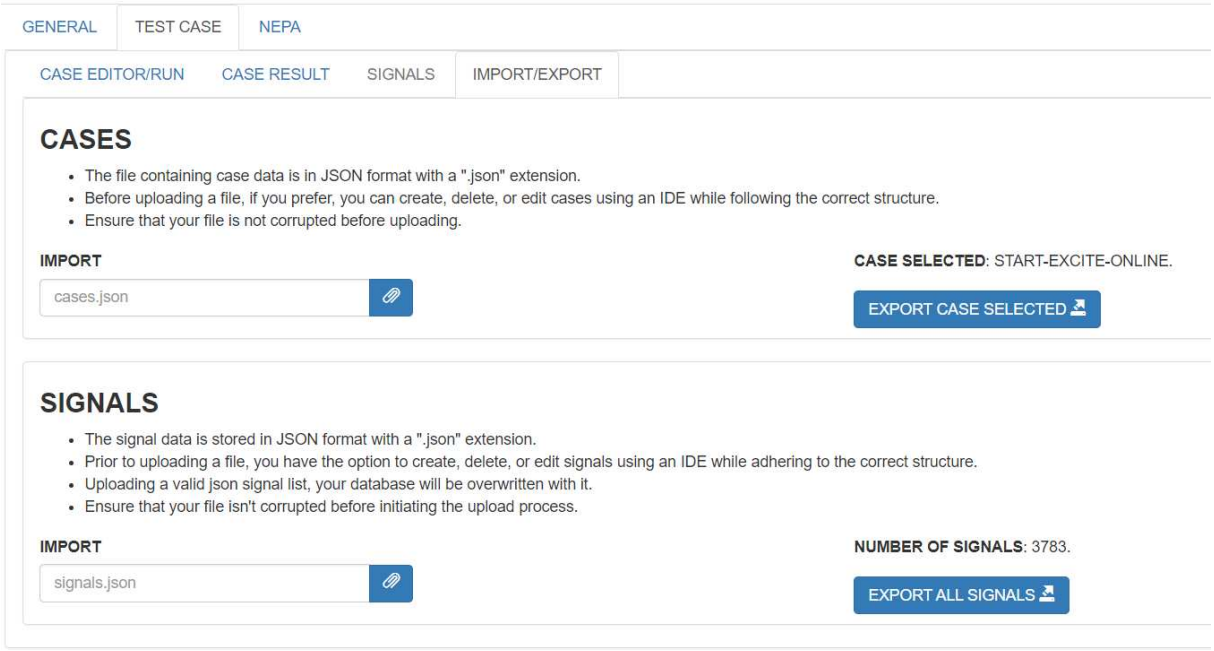
5.5 OUTRAS IMPLEMENTAÇÕES

Para facilitar o uso do STA, foram desenvolvidas implementações adicionais. A Figura 47 ilustra uma tela onde é possível exportar e importar o JSON dos casos e sinais, promovendo a cooperação entre colegas do setor.

A Figura 48 mostra a tela das ferramentas do NEPA, onde o usuário pode tentar restabelecer a conexão com o PA e na Figura 49 pode-se realizar testes em sinais individuais.

Embora não tenha sido utilizado um sinal de entrada analógico no caso de teste, a seleção dessa opção exibe uma pré-visualização. Na Figura 50, um sinal sinusoide é configurado como exemplo, modelado pelas configurações inseridas nos campos acima. Na Figura 51, um sinal RGX é utilizado, demonstrando a capacidade do STA de emular um sinal proveniente de registros.

Figura 47 – Exportar e importar casos e sinais.

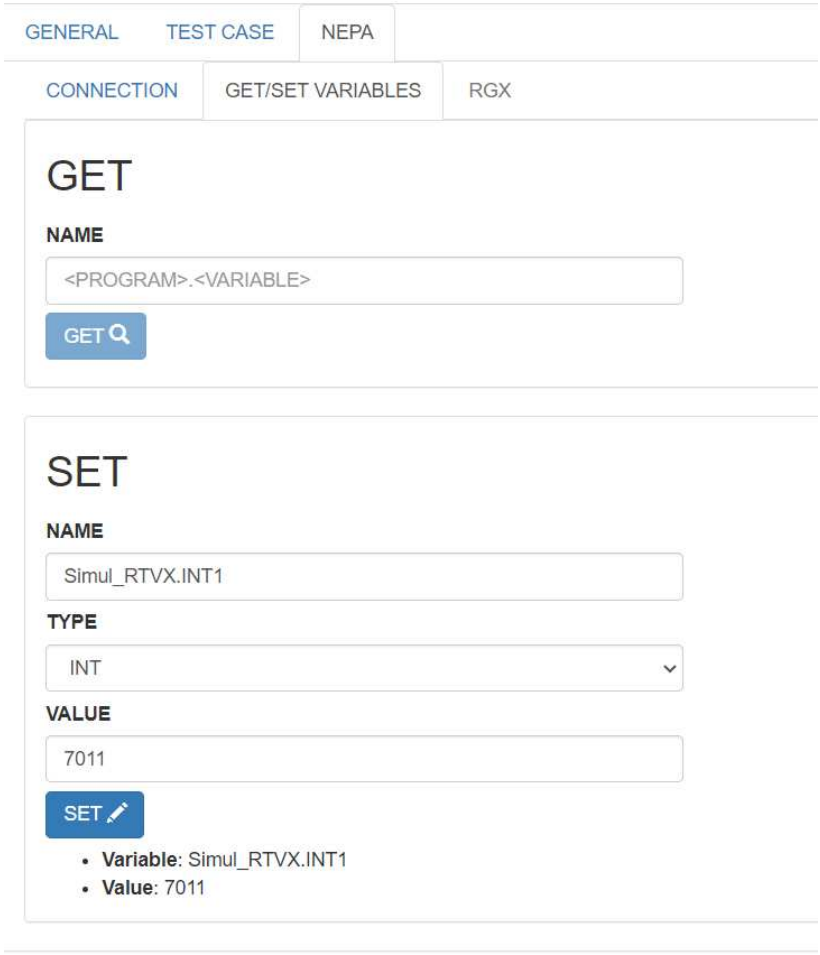


Fonte: Autoria própria.

Figura 48 – Conectar ao NEPA.

Fonte: Autoria própria.

Figura 49 – Realizar leitura e envio de valores de teste em variáveis.

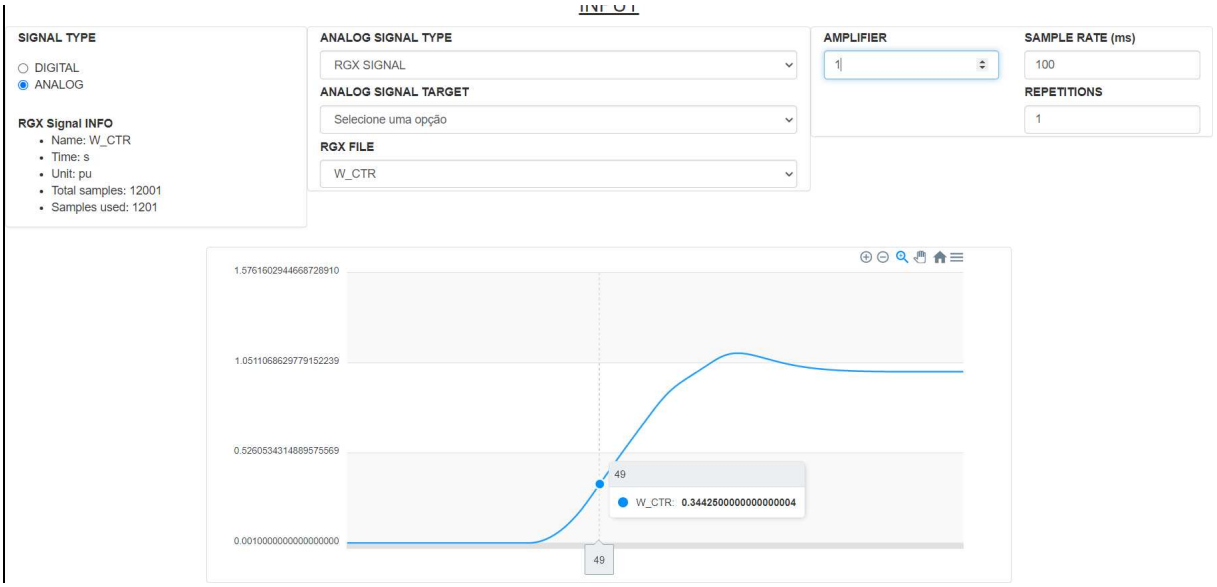


Fonte: Autoria própria.

Figura 50 – Sinal de entrada do tipo senoide.

Fonte: Autoria própria.

Figura 51 – Sinal de entrada RGX.



Fonte: Autoria própria.

6 CONCLUSÃO

6.1 RESUMO DO PROJETO

O desenvolvimento da ferramenta de apoio à customização de software de CLP para a REIVAX foi um projeto desafiador e empolgante. Este trabalho visava criar uma aplicação que facilitasse o processo de criação, gerenciamento e execução de casos de teste, melhorando a eficiência do desenvolvimento de software na empresa.

Através de uma análise de requisitos, a definição de uma arquitetura e a implementação de funcionalidades-chave, este projeto buscou atender às necessidades específicas dos engenheiros de software e contribuir para o sucesso contínuo da REIVAX na indústria de automação e controle.

Os principais objetivos estabelecidos no início deste projeto foram alcançados com sucesso: Foi desenvolvida o STA que oferece funcionalidades para criar, gerenciar, executar e analisar casos de teste. A aplicação fornece uma interface amigável e responsiva para os engenheiros.

6.2 LIÇÕES APRENDIDAS

Ao longo do projeto, foram identificadas algumas lições valiosas que podem aparecer em projetos similares de Engenharia de Controle e Automação.

A interface realizada com engenheiros do setor de Pesquisa e Desenvolvimento e de colegas do setor de Software fez que houve uma compreensão das necessidades de ambas as partes, o que é fundamental para o sucesso do projeto.

Criar uma aplicação é um desafio que envolve integrar elementos aparentemente incompatíveis, como uma interface web com uma API a um CLP. No entanto, ao final, esses componentes se harmonizam bem, resultando em uma solução leve e fácil de manusear.

6.3 CONTRIBUIÇÕES E IMPACTO

O STA busca ser o início de uma ferramenta que engloba todo o tipo de teste que auxilie os desenvolvedores. A melhoria da eficiência do desenvolvimento deve permitir que a empresa atenda às demandas dos clientes de forma mais rápida e eficaz, resultando em maior satisfação do cliente e competitividade no mercado de automação industrial.

6.4 TRABALHOS FUTUROS

Este projeto espera estabelecer uma base, mas há oportunidades para expansão e melhoria contínua:

Integração de mais protocolos

Além do Modbus, deve-se considerar a integração de outros protocolos de comunicação comuns usados em sistemas de automação industrial para atender a uma variedade maior de CLPs.

A inclusão de mais protocolos resulta, consequentemente, em uma diversidade maior de tipos de testes. Realizar testes em protocolos industriais é benéfico, pois permite fornecer um relatório detalhado sobre o correto funcionamento de todo o mapa de endereçamento. Isso garante a confiabilidade do projeto a ser entregue, especialmente considerando que muitos clientes utilizam protocolos remotos para o *Supervisory Control and Data Acquisition* (SCADA).

Melhorias na Interface do Usuário

Continuar aprimorando a interface do usuário com base no *feedback* recebido é essencial para tornar a ferramenta mais intuitiva e eficaz. Isso se deve ao fato de que nem todos os usuários potenciais possuem conhecimento aprofundado em programação; portanto, quanto mais transparente e acessível a interface, melhor.

Suporte a mais recursos

Adicionar recursos adicionais de testes, como simulação de falhas e testes de segurança, para abordar uma variedade mais ampla de cenários de teste. Uma interface pode não ser adequada em alguns cenários de teste, com isso, pode ser pensada uma solução que permita o uso de alguma linguagem mais flexível.

Otimização de Lógicas

A otimização de lógicas digitais é fundamental na área de design de circuitos digitais e sistemas embarcados. Essa prática tem várias implicações e benefícios importantes, tanto do ponto de vista técnico quanto econômico. A ferramenta pode examinar as lógicas implementadas e, se possível, apontar otimizações que poderiam ser feitas.

Análise do Processamento

O tempo de processamento em é uma consideração crítica em sistemas industriais e de automação. Essa métrica refere-se ao tempo que o CLP leva para executar uma tarefa ou ciclo de programa. Um processamento adequado é importante em sistemas de controle de geração de energia, pois lidam com cenários altamente exigentes.

Expansão para Outros Setores

O STA pode ser muito bem utilizado em diversos setores da REIVAX. Por exemplo:

- **Pesquisa e Desenvolvimento**: Setor responsável pela manutenção e aprimoramento do software base utilizado pela engenharia. Ele pode utilizar a ferramenta para realizar as novas liberações do produto na linha base;
- **Assistência Técnica**: Lida com problemas reportados por clientes. Muitas vezes, a assistência técnica poderá usufruir da ferramenta para investigação de problemas ou esclarecer dúvidas rapidamente;
- **Service**: Realiza os testes dos painéis. Eles são responsáveis pelos testes com equipamentos que simulam a situação mais próxima da de usinas. A ferramenta pode ser utilizada para executar os testes rotineiros de forma mais dinâmica;
- **Comissionamento**: Setor responsável por ir à usina e realizar de fato a instalação e comissionamento. Realizam uma série de ensaios com o sistema real e geram um relatório de comissionamento, onde ganhos, falhas e configurações são levantados. A ferramenta pode ser auxiliadora para definir ganhos em que indicadores de sistemas de controle sejam satisfeitos.

Inteligência Artificial aplicada à comissionamentos

O comissionamento pode resultar em ajustes nos ganhos da malha de controle, o que pode não ser intuitivo para um sistema dinâmico real. A inteligência artificial pode ser aplicada para estimar os ganhos. Por exemplo, ela pode executar automaticamente a partida e parada do gerador, avaliando os indicadores a cada ciclo. Se as métricas não forem atendidas, a inteligência artificial deve estimar um novo valor. A inteligência artificial é necessária, pois existem casos nos quais a alteração do ganho não tem mais efeito, e ela deve "entender" quando está na sua melhor forma.

Geração contínua de casos de testes

Após a bem-sucedida validação da ferramenta, é necessário criar a manutenção dos casos de testes, que são testes básicos aplicáveis:

1. Partida e Parada: Verificação da malha de controle do RVX;
2. Excitação e Desexcitação: Verificação da malha de controle do RTX;
3. Sincronização do gerador, simulação da malha de potência;

4. Verificação de todos os modos de controle: Frequência, abertura, potência, nível do reservatório, tensão terminal, corrente de campo, potência reativa e fator de potência;
5. Injeção de Valores nas Entradas Digitais e Analógicas: Simular diversos cenários com diferentes combinações;
6. Estímulo dos acionamentos das saídas digitais e analógicas;
7. Automação: Verificação dos automatismos implementados;
8. Falhas: Habilitação de falhas leves (alarmes) e graves (TRIPs), simular elementos que deveriam causar as falhas (ou não);
9. Disparo de Pontes: Ajustar níveis diferentes de disparo das pontes de tiristores;
10. Testes específicos em compensadores, motores e quaisquer outras aplicações alternativas do RTX.

REFERÊNCIAS

[1] ApexCharts Team. *ApexCharts*. Acessado em 30 de Outubro de 2023. Disp. em: https://apexcharts.com/.

[2] Kent Beck *et al. Manifesto para Desenvolvimento Ágil de Software*. Online. Disponível em: https://manifestoagil.com.br/. 2001.

[3] Bonitasoft. *Bonita UI Designer*. https://github.com/bonitasoft/bonita-ui-designer. Acessado em: 29 de outubro de 2023. 2023.

[4] G. Booch, J. Rumbaugh e I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley object technology series. Addison-Wesley, 2005. ISBN: 9780321267979. Disp. em: https://books.google.com.br/books?id=BqFQAAAAMAAJ.

[5] CANopen Solutions. *CANopen, a comprehensive communication and application standard for distributed systems*. Acesso em: 28 de outubro de 2023. 2007. Disp. em: https://web.archive.org/web/20070203052517/http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml.

[6] Codecademy. *Introduction to NumPy and Pandas*. https://www.codecademy.com/article/introduction-to-numpy-and-pandas. Acessado em: 06/10/2023. 2023.

[7] Institute of Electrical e Electronics Engineers. *IEEE Std 421.5-2016: Recommended Practice for Excitation System Models for Power System Stability Studies*. Acessado em: 19/10/2023. 2016. Disp. em: https://home.engineering.iastate.edu/~jdm/ee554/IEEEstd421.5-2016RecPracExSysModsPwrSysStabStudies.pdf.

[8] Mark Fewster e Dorothy Graham. *Software test automation*. Addison-Wesley Reading, 1999.

[9] A.E. Fischer e F.S. Grodzinsky. *The Anatomy of Programming Languages*. Prentice-Hall International editions. Prentice Hall, 1993. ISBN: 9780130422194. Disp. em: https://books.google.com.br/books?id=pIu5QgAACAAJ.

[10] Behrouz A. Forouzan. *Comunicação de Dados e Redes de Computadores*. 4ª ed. McGraw-Hill, 2008.

[11] Google. *AngularJS*. https://angularjs.org/. Acessado em: 29 de outubro de 2023. 2023.

[12] Joseph L. Hellerstein *et al. Feedback Control of Computing Systems*. IEEE PRESS. A Wiley-Interscience publication. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004. ISBN: 0-471-26637-X.

[13] Itaipu Binacional. *Vista da Itaipu Binacional*. https://abadi.com.br/wp-content/uploads/2019/09/itaipu-binacional15125.jpg. Acesso em: 05 nov. 2023. 2019.

[14] Dean Karnopp, Donald L Margolis e Ronald Carl Rosenberg. *System dynamics*. Wiley New York, 1990.

[15] Microsoft. *Noções básicas da base de dados*. Acessado em: 29 de outubro de 2023. 2023. Disp. em: https://support.microsoft.com/pt-pt/topic/no%C3%A7%C3%B5es-b%C3%A1sicas-da-base-de-dados-a849ac16-07c7-4a31-9948-3c8c94a7c204.

[16] Modbus-IDA. *Modbus Application Protocol V1.1b3*. 2012. Disp. em: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

[17] Operador Nacional do Sistema Elétrico. *O que é ONS*. https://www.ons.org.br/paginas/sobre-o-ons/o-que-e-ons. Acesso em: 05 nov. 2023. 2023.

[18] PhoenixNAP. *Servidor API - Glossário*. Acesso em: 29 out. 2023. 2023. Disp. em: https://phoenixnap.pt/gloss%C3%A1rio/api-server.

[19] PyInstaller. *PyInstaller Manual*. Acessado em 30 de Outubro de 2023. Disp. em: https://pyinstaller.org/en/stable/.

[20] Python. *Applications for Python*. Acesso em: 29 out. 2023. 2023. Disp. em: https://www.python.org/about/apps/.

[21] Dudekula Rafi *et al.* "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey". *In*: jun. de 2012, pp. 36–42. DOI: 10.1109/IWAST.2012.6228988.

[22] REIVAX. *Diagrama RTVAX*. https://www.reivax.com/wp-content/uploads/2022/03/DIAGRAMA-RTVAX.png. Acessado em: 19/10/2023. 2022.

[23] REIVAX. *Logo da REIVAX*. https://c5gwmsmjx1.execute-api.us-east-1.amazonaws.com/prod/empresa/logo/106494/LogoREIVAX.png. Acessado em: 19/10/2023. 2023.

[24] REIVAX. “Manual do Usuário AP SEC para RTVAX POWER”. *In*: 6.CP-07-08-02-03-V200R006 (2023).

[25] REIVAX. “Manual do Usuário AP xVision para RTVAX Power”. *In*: 5.CP-07-08-02-04-V100R005 (2023).

[26] REIVAX. “Manual do Usuário CPX05”. *In*: 12.CP-02-113-02-07-R012 (2023).

[27] REIVAX. “Manual do Usuário Sistema de Edição de Configurações”. *In*: 1.CP-01-03-02-04-V203R001 (2023).

[28] REIVAX. “Manual do Usuário xVision”. *In*: 12.CP-02-113-02-07-R012 (2023).

[29] REIVAX. *Sistema Integrado de Regulação de Tensão, Velocidade e Automação*. https://www.reivax.com/wp-content/uploads/2022/03/sistema-excitacao.png. Acessado em: 19/10/2023. 2022.

[30] REIVAX. “Software Básico Embarcado Manual do Usuário”. *In*: 33.CP-01-01-02-03-R033-SBE-DMU-P (2023).

[31] P.W. Sauer e M.A. Pai. *Power System Dynamics and Stability*. Prentice Hall, 1998. ISBN: 9780136788300. Disp. em: https://books.google.com.br/books?id=d00eAQAAIAAJ.

[32] Amazon Web Services. *The Difference Between JSON & XML*. Acessado em: 29 de outubro de 2023. 2023. Disp. em: https://aws.amazon.com/pt/compare/the-difference-between-json-xml/.

[33] Texas Instruments. *RS-485: The most versatile communication standard*. Acesso em: 28 de outubro de 2023. 2018. Disp. em: https://web.archive.org/web/20180517101401/http://www.ti.com/lit/sg/slyt484a/slyt484a.pdf.

[34] TreinaWeb. *Os Principais SGBDs NoSQL*. Acessado em: 29 de outubro de 2023. 2023. Disp. em: https://www.treinaweb.com.br/blog/os-principais-sgbds-nosql.

[35] Frank Tsui, Orlando Karam e Barbara Bernal. *Essentials of software engineering*. Jones & Bartlett Learning, 2022.

[36] Yuqing Wang *et al.* “Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration”. *In*: *Journal of Systems and Software* 188 (2022), p. 111259. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2022.111259. Disp. em: https://www.sciencedirect.com/science/article/pii/S0164121222000280.