CENTRO FEDERAL DE EDUCAÇÃO TECNOLÔGICA DE MINAS GERAIS

UNIDADE ARAXÁ

ALDERICO HENRIQUE SOARES AFONSO

SISTEMA SUPERVISÓRIO *WEB* UTILIZANDO PROTOCOLO OPC PARA UMA PLANTA DE CLASSIFICAÇÃO DE PEÇAS

ARAXÁ/MG

2024

ALDERICO HENRIQUE SOARES AFONSO

SISTEMA SUPERVISÓRIO *WEB* UTILIZANDO PROTOCOLO OPC PARA UMA PLANTA DE CLASSIFICAÇÃO DE PEÇAS

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Automação Industrial, do Centro Federal de Educação Tecnológica de Minas Gerais – CEFET/MG, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Automação Industrial.

Orientador: Prof. Dr. Mateus Antunes Oliveira Leite

Coorientador: Prof. Me. Willian Martins Leão

ARAXÁ/MG

2024

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
COORDENAÇÃO DO CURSO DE ENGENHARIA DE AUTOMAÇÃO INDUSTRIAL / ARAXÁ

TRABALHO DE CONCLUSÃO DE CURSO – TCC – ATA DE DEFESA

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO DE ENGENHARIA DE AUTOMAÇÃO INDUSTRIAL DO ALUNO **ALDERICO HENRIQUE SOARES AFONSO**

Às **10 horas** do dia **13 de setembro de 2024**, reuniu-se, na sala 504A, do Centro Federal de Educação Tecnológica de Minas Gerais - CEFET-MG/ Campus Araxá, a Comissão Examinadora de Trabalho de Conclusão de Curso para julgar, em exame final, o trabalho intitulado **SISTEMA SUPERVISÓRIO WEB UTILIZANDO PROTOCOLO OPC PARA UMA PLANTA DE CLASSIFICAÇÃO DE PEÇAS**, como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Automação Industrial. Abrindo a sessão, o Presidente da Comissão, Prof. **Mateus Antunes Oliveira Leite**, após dar a conhecer aos presentes o teor das Normas Regulamentares do Trabalho Final, concedeu a palavra ao candidato, **Alderico Henrique Soares Afonso**, para a exposição de seu trabalho. Após a apresentação, seguiu-se a arguição pelos examinadores, com a respectiva defesa do candidato. Ultimada a arguição, a Comissão se reuniu, sem a presença do candidato e do público, para julgamento e expedição do resultado final. Após a reunião da Comissão Examinadora, o candidato foi considerado: Aprovado, obtendo nota final de: 96 **/100 (**noventa e seis**)**. O resultado final foi comunicado publicamente ao candidato pelo Presidente da Comissão. O aluno, abaixo assinado, declara que o trabalho ora identificado é da sua autoria material e intelectual, excetuando-se eventuais elementos, tais como passagens de texto, citações, figuras e datas, desde que devidamente identificada a fonte original. Declara ainda, neste âmbito, não violar direitos de terceiros. Nada mais havendo a tratar, o Presidente encerrou os trabalhos. O prof. Frederico Duarte Fagundes, responsável pela disciplina "Trabalho de Conclusão de Curso II", lavrou a presente ATA, que, após lida e aprovada, será assinada por todos os membros participantes da Comissão Examinadora. Araxá, **13 de setembro de 2024**.

Mateus Antunes Oliveira Leite

Willian Martins Leão

Luís Paulo Fagundes

Henrique José Avelar

Alderico Henrique Soares Afonso

Agradecimentos

Dedico este trabalho, em primeiro lugar, aos meus pais, minha madrinha e aos meus queridos avós, que, mesmo não estando mais aqui, sempre acreditaram em mim, nas minhas capacidades e nas minhas escolhas. Sou profundamente grato pelos sacrifícios que fizeram para que eu chegasse até aqui e por terem sido, ao longo de toda a minha jornada, fontes constantes de inspiração e força.

Agradeço também aos meus amigos e colegas de classe, Rafael, Otávio, Gabriel, Maria Paula, e, especialmente, ao Lucas Honorato, que esteve ao meu lado em praticamente todos os momentos da minha formação, tanto nos bons quanto nos difíceis. Juntos, enfrentamos inúmeros desafios, superamos obstáculos e celebramos conquistas que ficarão para sempre em minha memória.

Aos professores do CEFET, expresso minha profunda gratidão pelos ensinamentos e pelo vasto conhecimento compartilhado ao longo do curso, assim como pelas oportunidades que me foram oferecidas, que enriqueceram minha formação.

Agradeço à minha namorada, companheira e futura esposa, Maria Eduarda. Obrigado por todo o apoio incondicional durante essa jornada, pelo companheirismo constante e por me manter firme nos momentos mais difíceis.

Não poderia deixar de mencionar minha fiel companheira, minha cachorra Amelie, que, mesmo sem entender, me ajudou muito, acalmando-me e estando ao meu lado, especialmente durante o período de ensino remoto, em que passávamos o dia inteiro juntos.

Por fim, agradeço a mim mesmo. Por ter perseverado diante de todas as adversidades, incertezas e desafios, e por ter chegado até aqui, concluindo mais uma etapa importante do meu caminho.

Resumo

O presente trabalho trata do desenvolvimento de um sistema supervisório *web* aplicado a uma planta de separação de peças simulada no software Factory I/O. A programação em *ladder* e a criação do servidor OPC, assim como sua configuração, foram realizadas por meio do Codesys. Para integrar o sistema, a API REST foi desenvolvida utilizando a plataforma .NET com a biblioteca Opc.UaFx.Client e a linguagem C#. O *frontend* foi implementado com CSS, HTML, Bootstrap e JavaScript, garantindo que o sistema fosse responsivo e adaptável a diversos dispositivos e tamanhos de telas. Para estabelecer a comunicação com o servidor OPC, o primeiro passo foi a validação da conexão e identificação do NodeId de cada variável associada aos componentes da planta, assegurando que as variáveis corretas pudessem ser acessadas e modificadas em tempo real. Esse processo foi essencial para garantir a integração eficiente entre o servidor OPC e o *backend*. Com o sucesso na comunicação, o sistema foi configurado para realizar leituras e modificações dinâmicas, permitindo uma supervisão remota eficaz. A conclusão destaca que a adaptação da planta utilizando o Factory I/O foi bem-sucedida, assim como o desenvolvimento da lógica *ladder* no Codesys, a configuração do servidor OPC e a implementação da API REST com o .NET. Além disso, as interfaces desenvolvidas com CSS, HTML, JavaScript e Bootstrap demonstraram ser operacionais e responsivas, adaptando-se de forma eficiente a diferentes dispositivos com acesso à *web*, consolidando a funcionalidade e aplicabilidade do sistema supervisório *web* desenvolvido.

Palavras-chave: API REST, Codesys, Factory I/O, .NET, Responsividade, Sistema Supervisório, *Web*.

Abstract

This work focuses on the development of a *web*-based supervisory system applied to a parts sorting plant simulated in the Factory I/O software. The *ladder* programming and the creation of the OPC server, as well as its configuration, were carried out using Codesys. To integrate the system, the REST API was developed using the .NET platform with the Opc.UaFx.Client library and the C# language. The *frontend* was implemented with CSS, HTML, Bootstrap, and JavaScript, ensuring that the system is responsive and adaptable to various devices and screen sizes. To establish communication with the OPC server, the first step was to validate the connection and identify the NodeId of each variable associated with the plant components, ensuring that the correct variables could be accessed and modified in real time. This process was essential to ensure efficient integration between the OPC server and the *backend*. With successful communication, the system was configured to perform dynamic readings and modifications, enabling effective remote supervision. The conclusion highlights that the adaptation of the plant using Factory I/O was successful, as was the development of the *ladder* logic in Codesys, the configuration of the OPC server, and the implementation of the REST API with .NET. Additionally, the interfaces developed with CSS, HTML, JavaScript, and Bootstrap proved to be operational and responsive, efficiently adapting to different devices with *web* access, consolidating the functionality and applicability of the developed *web*-based supervisory system.

Keywords: API REST, Codesys, Factory I/O, .NET, Responsiveness, Supervisory System, *Web*.

Lista de Ilustrações

Lista de Siglas e Abreviaturas

API – Interface de Programação de Aplicações

API REST – Interface de Programação de Aplicações *Representational State Transfer*

ASP.NET – Framework de desenvolvimento *web* da Microsoft baseado no .NET

Backend – Parte do sistema responsável pelo processamento no servidor (lógica do sistema)

Cloud – Computação em Nuvem

CSS – Folhas de Estilo em Cascata

DOM – Modelo de Objetos de Documento

Frontend – Parte do sistema com a qual o usuário interage diretamente (interface)

HTML – Linguagem de Marcação de Hipertexto

IA – Inteligência Artificial

IHM – Interface Homem-Máquina

IOT – Internet das Coisas (*Internet of Things*)

ISA-101 – Norma Internacional de Interfaces Homem-Máquina (*Human Machine Interfaces for Process Automation*)

JSON – Notação de Objetos JavaScript

Ladder – Linguagem de Programação *Ladder* (Diagrama de Escada)

LINQ – Consulta Integrada à Linguagem

MVC – Modelo-Visão-Controlador

.NET – *Framework* de desenvolvimento da Microsoft para diversas plataformas

OPC – Comunicações de Plataforma Aberta (*Open Platform Communications*)

Razor – Sintaxe de páginas *web* usada no ASP.NET

SCADA – Supervisão e Aquisição de Dados (*Supervisory Control and Data Acquisition*)

SPA – Aplicação de Página Única (*Single-Page Application*)

TSN – Rede com Sensibilidade ao Tempo (*Time-Sensitive Networking*)

UDP – Protocolo de Datagrama de Usuário (*User Datagram Protocol*)

W3C – Consórcio da *World Wide Web* (*World Wide Web Consortium*)

Web – Rede de comunicação global (*World Wide Web*)

Sumário

1. Introdução

No âmbito industrial, é importante garantir o controle e supervisão dos sistemas de maneira remota. Essa necessidade decorre da importância de obter informações da planta em tempo real, independentemente da localização física, proporcionando uma visão abrangente que transcende os limites do chão de fábrica (NVTEC, 2023). Essas informações englobam não apenas o nível de produção, mas também detalhes precisos sobre as variáveis operacionais e o estado dos instrumentos utilizados na planta, assegurando uma gestão eficaz e informada do processo industrial (ACKERMAN' e BLOCK, 1992).

Apesar da variedade de sistemas de supervisão e controle disponíveis no mercado, nem todos oferecem a flexibilidade de serem utilizados além da sala de controle. Essa limitação, em certa medida, restringe a atuação daqueles encarregados da supervisão, confinando-os a um local fixo e limitando suas opções de deslocamento dentro da empresa. Em contrapartida, os sistemas supervisórios *web* possibilitam o uso de dispositivos de forma remota, inclusive fora das instalações da empresa. Isso não apenas amplia a produtividade, mas também confere maior flexibilidade ao operador, permitindo que ele acesse e gerencie as operações de maneira conveniente e eficaz, independentemente de sua localização física. Essa abordagem dinâmica impulsiona a eficiência operacional e otimiza a capacidade de resposta, destacando a vantagem dos sistemas supervisórios *web* na moderna gestão industrial.

É relevante salientar que o desenvolvimento e implementação de um sistema supervisório *web* apresentam na maioria dos casos vantagens em termos de simplicidade quando comparados a sistemas tradicionais. Isso se deve ao fato de que o conhecimento em programação e desenvolvimento *web* é mais amplamente difundido do que o conhecimento específico em automação. De acordo com o relatório anual de vagas do Linkedin, o número de oportunidades para desenvolvedores *web* supera significativamente o de profissionais de automação, refletindo a maior demanda por habilidades em TI e programação no mercado (LINKEDIN

CONTRIBUTORS, 2020). Essa disseminação de habilidades resulta em uma redução significativa de custos, tanto no estágio inicial de desenvolvimento quanto em futuras manutenções e implementações de aprimoramentos. Esse cenário destaca a viabilidade e acessibilidade dos sistemas supervisórios *web*, oferecendo uma solução eficaz e economicamente eficiente para a gestão e monitoramento de sistemas industriais.

Atualmente, as três principais bases de desenvolvimento *web* são HTML, CSS e JavaScript. HTML é uma linguagem de marcação, CSS é uma linguagem de estilização, e JavaScript é uma linguagem de programação interpretada leve com funções de primeira classe (MDN CONTRIBUTORS, 2023a). Além dessas, uma ferramenta notável, embora não seja exclusiva para a *web*, é o OPC-UA. Este protocolo, independente de fornecedor ou plataforma, é amplamente utilizado na indústria, oferecendo suporte a mecanismos avançados de segurança (DERHAMY e colab., 2017). Essa combinação de tecnologias forma uma base robusta para o desenvolvimento *web*, possibilitando a criação de interfaces dinâmicas e seguras, além de integrar eficientemente sistemas industriais.

Neste contexto, o foco do projeto é o desenvolvimento de um sistema supervisório *web*, com a comunicação da planta de separação de peças; simulada pelo Factory.IO; estabelecida por meio do protocolo OPC-UA. O *frontend* será construído utilizando *framework* Bootstrap, visando criar uma aplicação adaptável a diversos dispositivos. Para o desenvolvimento do *backend*, será utilizado o ASP.NET, dada sua robustez e eficiência no gerenciamento de requisições. Essa escolha técnica busca garantir não apenas a funcionalidade do sistema, mas também sua capacidade de adaptação a diferentes ambientes e dispositivos.

1.1. Objetivos

O principal objetivo deste trabalho é desenvolver um sistema de supervisão *web* para monitorar uma planta de classificação de peças utilizando o protocolo OPC. Além disso, o sistema será disponibilizado de forma aberta para o CEFET-MG e outras instituições interessadas em utilizá-lo.

Com isso, os objetivos secundários são os seguintes:

- Obter um sistema que respeite os principais requisitos indicados na norma ISA-101;
- Desenvolver a lógica de controle, a API e o *frontend* do sistema, tendo como base a planta de separação de peças simulada pelo Factoy.IO;
- Possuir um sistema responsivo, possibilitando a utilização em diferentes dispositivos, como computadores, tablets, celulares e outros dispositivos que tenham acesso a um navegador *web*.

2. Fundamentação Teórica

Este tópico fornecerá uma revisão dos principais temas que servirão de base para este trabalho. Incluem-se a definição de sistemas supervisórios, a exploração da norma técnica que estabelece recomendações para sua criação, bem como o protocolo de comunicação adotado, conhecido como OPC. Além disso, serão abordadas as tecnologias *web* fundamentais, como HTML, CSS e JavaScript, assim como alguns *frameworks* de *frontend*, a exemplo do Bootstrap. Este contexto também contemplará uma compreensão da ferramenta .NET e suas funcionalidades tanto para o desenvolvimento *web* quanto para o *backend*.

2.1. Sistemas Supervisórios

Um sistema supervisório, conhecido como SCADA (*Supervisory Control and Data Acquisition*), é um sistema fundamental na indústria, amplamente empregado em diferentes setores para monitorar e controlar processos complexos em tempo real. Utilizado em plantas industriais, redes elétricas, sistemas de abastecimento de água, e instalações de tratamento de água e esgoto, sua principal função é coletar dados em tempo real de sensores e dispositivos, processá-los e apresentá-los de maneira compreensível aos operadores por meio de uma interface gráfica intuitiva (ACKERMAN’ e BLOCK, 1992).

Ao proporcionar aos operadores informações precisas e atualizadas, os sistemas supervisórios capacitam as equipes a tomar decisões informadas para controlar e otimizar os processos industriais. Além disso, esses sistemas oferecem funcionalidades vitais, como alarmes em tempo real para alertar sobre condições anormais, registro de histórico de dados para análises retrospectivas, geração de relatórios detalhados e a capacidade de integração com outros sistemas de automação e controle (NVTEC, 2023).

Assim, os sistemas supervisórios desempenham um papel crucial, garantindo eficiência e segurança em operações industriais complexas, ao mesmo tempo em que facilitam a gestão inteligente e ágil dos processos (NVTEC, 2023).

2.1.1. Início e Estado Atual dos Sistemas Supervisórios

Os sistemas supervisórios têm evoluído ao longo do tempo, mas sua origem remonta às décadas de 1960 e 1970. Durante esse período, com o avanço da computação e da automação industrial, começaram a surgir os primeiros sistemas que podiam coletar dados de sensores e controlar dispositivos em tempo real (ACKERMAN' e BLOCK, 1992).

No entanto, o termo SCADA (*Supervisory Control and Data Acquisition*) foi popularizado nos anos 1970 e 1980 com o desenvolvimento de tecnologias mais avançadas de controle e monitoramento industrial. Nesse período, houve um aumento significativo na implementação de sistemas SCADA em diversas indústrias, como energia, água e manufatura (ACKERMAN' e BLOCK, 1992).

Desde então, os sistemas supervisórios continuaram a evoluir, incorporando tecnologias mais sofisticadas, como interfaces gráficas de usuário amigáveis, comunicações em rede avançadas e capacidades analíticas mais robustas. Hoje em dia, os sistemas supervisórios desempenham um papel fundamental em operações industriais complexas, proporcionando monitoramento em tempo real, controle eficiente e análises detalhadas para otimização de processos(SAJID e colab., 2016).

No cenário tecnológico atual, os sistemas supervisórios estão em constante evolução para acompanhar as crescentes exigências da indústria. Dentre as tendências e características mais notáveis desses sistemas, destacam-se a integração com tecnologias emergentes, como Internet das Coisas (IoT), Inteligência Artificial (IA) e Aprendizado de Máquina (ML). Além disso, esses sistemas estão se aprimorando em áreas como acesso remoto e mobilidade, segurança cibernética, análise preditiva e computação em nuvem. Essas inovações refletem a contínua busca por soluções mais avançadas e eficientes no âmbito dos sistemas supervisórios industriais (SAJID e colab., 2016).

Diante da contínua evolução do cenário industrial e tecnológico, a norma ISA 101 foi concebida com o objetivo de otimizar os benefícios desses avanços, estabelecendo diretrizes claras para o desenvolvimento e a manutenção da interface

de sistemas supervisórios de alto desempenho. Dessa forma, o subcapítulo seguinte irá descrever as principais recomendações da norma.

2.2. ISA-101 / Norma de Sistemas Supervisórios

A norma técnica *Human Machine Interfaces for Process Automation* (ISA-101) é um documento elaborado meticulosamente para orientar o desenvolvimento de interfaces de sistemas supervisórios, visando alcançar padrões superiores de eficiência, ergonomia e confiabilidade. Ao estabelecer diretrizes claras e precisas, a norma fornece um roteiro detalhado para os profissionais da área, garantindo que eles possam criar interfaces de sistemas supervisórios robustos e de alto desempenho (INTERNATIONAL SOCIETY OF AUTOMATION., 2015).

Seguindo as recomendações da ISA-101, os engenheiros e desenvolvedores podem integrar tecnologias de ponta de forma harmoniosa, garantindo que o sistema funcione de maneira coesa e eficaz. Isso não apenas aumenta a segurança operacional, minimizando os riscos de falhas ou interrupções inesperadas, mas também contribui significativamente para a qualidade geral do processo (GREGORY DURANSO, 2020).

Além disso, a norma foca na otimização da produtividade, fornecendo orientações sobre a organização eficiente de dados, interface do usuário intuitiva e funcionalidades que facilitam a análise e o controle. Implementando essas práticas recomendadas, as empresas podem aumentar sua eficiência operacional, reduzindo o tempo necessário para realizar tarefas complexas e melhorando a capacidade de resposta aos desafios do ambiente industrial (GREGORY DURANSO, 2020).

Dessa forma, é fundamental destacar que a revisão bibliográfica deste subcapítulo será estruturada com base na norma ISA-101, *Human Machine Interfaces for Process Automation Systems.*

2.2.1. Desenvolvimento do Sistema de uma IHM

Conforme é definido pela norma, o desenvolvimento de uma interface de um sistema supervisório consiste em algumas etapas: primeira, a criação do console; a

segunda, o desenvolvimento da IHM; a terceira etapa foca na usabilidade; por fim, a criação do display. Dessa forma, segue o detalhamento de cada uma dessas etapas.

O *design* do console é fundamental para definir o ambiente de trabalho do operador, abrangendo tanto os módulos do console quanto as condições ambientais, como iluminação, temperatura e som, além dos dispositivos associados, como telefones, botões de desligamento, painéis indicadores, rádios e intercomunicadores. Além disso, ele também engloba o suporte para o espaço físico do operador, que pode estar situado tanto em um ambiente de escritório na sala de controle quanto no chão da planta industrial. Dependendo da localização e dos requisitos específicos, esse ambiente de trabalho pode incluir tanto equipamentos de informática corporativos quanto outros dispositivos dedicados de uso especial. A utilização de diferentes tamanhos de tela pode demandar atenção especial tanto nos arranjos físicos quanto no desenho da interface.

O desenvolvimento do sistema de IHM é fundamentado pela escolha da plataforma de controle, do sistema operacional correspondente, e pela seleção das ferramentas de IHM que serão integradas ao sistema. As considerações incluem conceitos de *design* de rede, funções e segurança do usuário, além das interfaces de terceiros. É crucial destacar que essas decisões exercem uma influência significativa no próprio *design* e na funcionalidade da IHM.

A terceira etapa consiste na usabilidade, que define os papeis de cada usuário que podem ser primários ou secundários. Após a definição dos papéis e requisitos básicos do usuário, as tarefas reais a serem executadas pelos usuários são registradas, revisadas e potencialmente otimizadas. Considerações incluem condições operacionais normais e anormais; necessidade de informações específicas ao usuário; requisitos relacionados a funções do usuário e privilégios de conta; e necessidades funcionais de suporte da IHM.

A etapa final inicia após a definição dos requisitos, é crucial elaborar um *design* IHM contando com a colaboração ativa dos usuários primários e secundários. Este processo visa assegurar que os requisitos tenham sido capturados de forma abrangente, antes do início do *design* detalhado. Uma prática comum consiste na realização de uma primeira revisão de "*layout*", na qual o conteúdo básico é

apresentado. Posteriormente, uma revisão final é conduzida, abrangendo todos os detalhes e interações entre dispositivos.

2.2.2. Ergonomia do Usuário

Os princípios do desenvolvimento de uma IHM visam otimizar a experiência do usuário final, considerando fatores como a quantidade adequada de sensores em cada tela, suas cores, formatos e a capacidade cognitiva do usuário. O *design* da tela deve ser consistente, permitindo que os usuários identifiquem facilmente as funcionalidades disponíveis. Essa consistência cria uma dinâmica que aumenta significativamente a eficiência do processo, pois os usuários podem rapidamente entender e utilizar as diferentes funcionalidades da IHM.

O *design* de uma tela de IHM deve aderir a padrões ergonômicos para preservar a saúde do operador, limitando a quantidade de elementos visíveis. Este padrão ergonômico é validado por meio de uma auditoria interna. Além do número de elementos por tela, é essencial considerar fatores como daltonismo, perda auditiva e problemas de visão.

Esse desenvolvimento deve ser sensível às limitações visuais dos usuários nos ambientes onde as tarefas relacionadas ao processo são realizadas. Ao implementar uma IHM para várias condições de iluminação, o sistema e as telas devem ser projetados para desempenho aceitável em todas as condições previstas. A luminosidade da tela da IHM deve ser ajustada ao ambiente, evitando contrastes excessivos que possam causar fadiga ocular.

Deficiências na percepção de cores, como daltonismo e problemas relacionados à idade, devem ser consideradas. Tipos comuns de daltonismo incluem dificuldades em distinguir vermelho-verde, verde-amarelo e branco-ciano. Para garantir clareza visual, deve-se usar contrastes e brilho diferenciados. As cores escolhidas devem ser facilmente distinguíveis, frequentemente testadas por meio de usabilidade. Geralmente, a cor é usada para destacar informações críticas, como alarmes e condições anormais.

As interações entre plano de fundo e primeiro plano são cruciais. O plano de fundo deve ser de uma cor neutra, como cinza claro, para evitar distorções cromáticas e garantir a visibilidade das informações. Cores de fundo excessivamente contrastantes, como o preto, devem ser evitadas na maioria das aplicações. As combinações de cores entre primeiro plano e fundo devem proporcionar contraste suficiente. A cor de fundo deve ser escolhida para manter um contraste adequado nas condições esperadas de iluminação.

A densidade de informações em uma tela deve ser determinada pela função dela, considerando as limitações da percepção humana. Deve-se apresentar apenas informações relevantes que possam ser interpretadas rapidamente pelo operador, evitando dados desnecessários. A organização dos elementos deve ser consistente e espacialmente alinhada para evitar confusão. Se as especificações iniciais resultarem em excesso de informações, redesenhos podem incluir a consolidação de dados, uso de estilos de exibição mais eficazes, exibição sob demanda ou divisão em múltiplas telas.

Devido às limitações perceptivas e cognitivas, apenas um número limitado de cores pode ser usado eficazmente. Assim, técnicas visuais dinâmicas, como movimento, piscar, intermitência e visibilidade condicional, podem direcionar a atenção do operador para informações específicas. O movimento simula mudanças de posição, rotação ou tamanho de elementos gráficos, enquanto o piscar alterna a visibilidade de um elemento gráfico. A intermitência alterna a cor ou intensidade de um elemento gráfico sem torná-lo invisível.

Outro fator crucial é a capacidade cognitiva do usuário, que pode ser influenciada pela carga de trabalho, problemas pessoais e familiares. Para mitigar essas perdas, informações sobre elementos específicos podem ser agrupadas em caixas simples quando não for possível colocá-las diretamente nos elementos da IHM.

2.2.3. Estilos de Telas

A escolha de um estilo de exibição deve ser fundamentada nos requisitos funcionais da tela, conforme definidos no processo de *design* da IHM. A eficácia do

método de apresentação também pode ser influenciada pela variação prevista nos dados do processo.

Além disso, a seleção pode ser condicionada por limitações tecnológicas e/ou físicas da IHM. Por exemplo, as seguintes considerações podem afetar a escolha do estilo de exibição: a forma como o usuário interage com a tela, incluindo o uso de telas sensíveis ao toque; a posição física da tela; o tamanho da tela; e a quantidade de informações que o usuário precisa gerenciar.

Os estilos de telas que podem ser implementados em uma IHM incluem:

1. Lista: Apresentação de dados em formato de lista, em que informações textuais e numéricas podem ser intercaladas com símbolos de equipamentos do processo.
2. Processo: Representação gráfica de equipamentos do processo, tubulações e instrumentação.
3. Vista Geral: Visão informativa da área de controle de um operador. Os tipos de controles e indicadores necessários dependerão dos requisitos funcionais da vista.
4. Funcional: Representação da relação funcional dos dados.
5. Gráfico: Representação baseada em gráficos de dados em tempo real ou históricos.
6. Grupos: Coleção de pontos de exibição baseada em tarefas específicas.
7. Monitor Lógico: Exibição que representa as relações lógicas entre as funções do sistema.
8. Procedural: Exibição da lógica de controle procedural.
9. Vídeo: Exibições de vídeo ao vivo ou gravado.
10. Diagnóstico: Exibição do status de certos componentes de infraestrutura do sistema de controle HMI;
11. Lista de Alarmes: Apresentação de uma lista com informações de status.

2.2.4. Hierarquia de Telas

Uma hierarquia de exibição é essencial para fornecer ao operador uma visão organizada de sua área de responsabilidade. Isso permite que o operador acesse

níveis de detalhes mais profundos e funcionalidades de controle específicas. A informação apresentada se torna cada vez mais detalhada e focada à medida que se avança nos níveis. É recomendado um máximo de quatro níveis, sendo o nível 1 o mais abrangente e o nível 4 o mais focado. Apesar de ser hierárquica, essa estrutura de exibição não precisa estar diretamente alinhada com a hierarquia de navegação, que pode ter menos ou mais níveis do que a hierarquia de exibição.

2.2.4.1. Tela de Nível 1

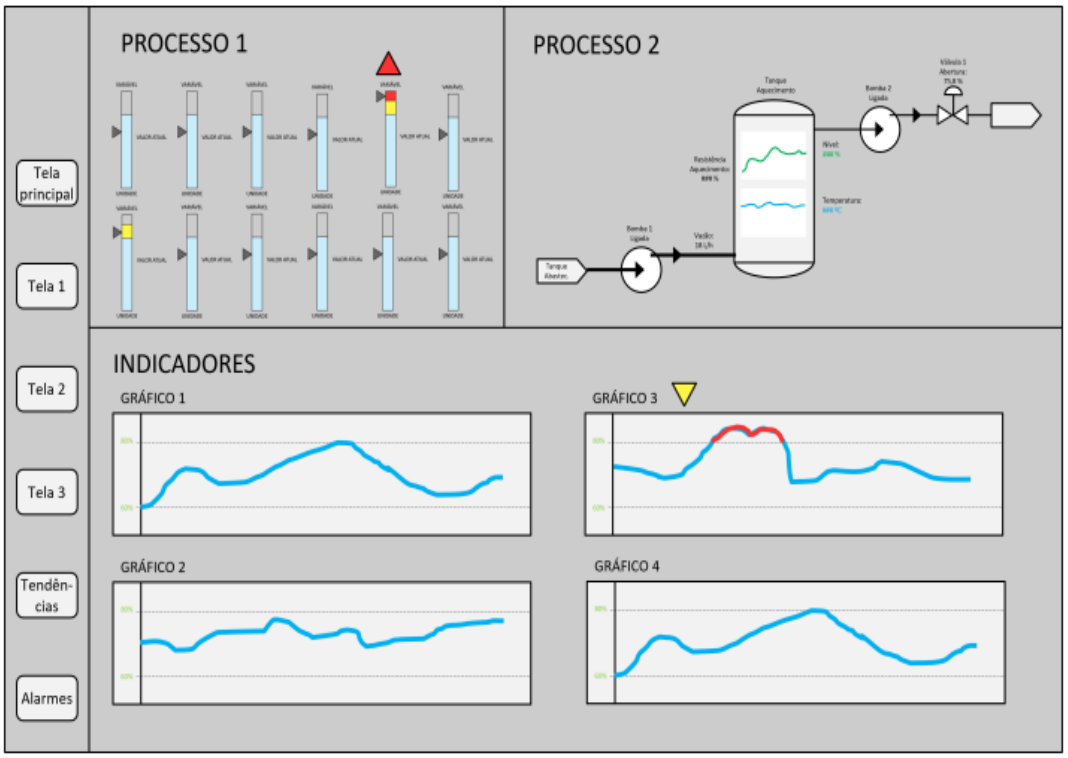
As telas de Nível 1 são projetadas para oferecer uma visão geral ou resumo dos principais parâmetros, alarmes, condições do processo calculadas e caminhos de propagação de distúrbios em toda a área de controle de um operador. Em sistemas maiores, a tela de Nível 1 pode se estender por várias telas, desde que todas sejam visíveis simultaneamente. Estas telas apresentam a mais ampla cobertura e o menor nível de detalhes do processo ou sistema. As telas de Nível 1 também podem ser usadas como ferramentas colaborativas para facilitar o compartilhamento de informações essenciais entre operadores e usuários secundários na sala de controle. Sendo ilustrada na Figura 1 um exemplo de tela de nível 1.

Algumas considerações importantes para as telas de Nível 1 incluem:

1. Considerar telas de visão geral separadas para todos os modos de operação, como inicialização e desligamento. As telas adicionais para este propósito devem garantir que todas as informações disponíveis na visão geral padrão permaneçam visíveis e posicionadas de maneira consistente.
2. Exibir todos os alarmes de alta prioridade, dispostos de forma a transmitir a relação funcional.
3. Incluir valores reais, status anormais e gravidade da variação para parâmetros chave do processo ou condições calculadas, refletindo a saúde das áreas de processo de alto nível, como fornos, reatores ou colunas de destilação. Se possível, também é recomendável fornecer acesso aos valores de variação, indicações de direção das mudanças e/ou tendências.

4. Apresentar informações adicionais sobre instalações relacionadas na planta, como utilidades, áreas a jusante e a montante.
5. Integrar tendências incorporadas para parâmetros essenciais.
6. Orientar o operador sobre a presença, gravidade, localização e direção das mudanças nas condições anormais do processo.
7. Evitar o uso das telas de Nível 1 para realizar funções de controle, como alterações no *setpoint* do controlador.

Figura 1 – Tela de Nível 1



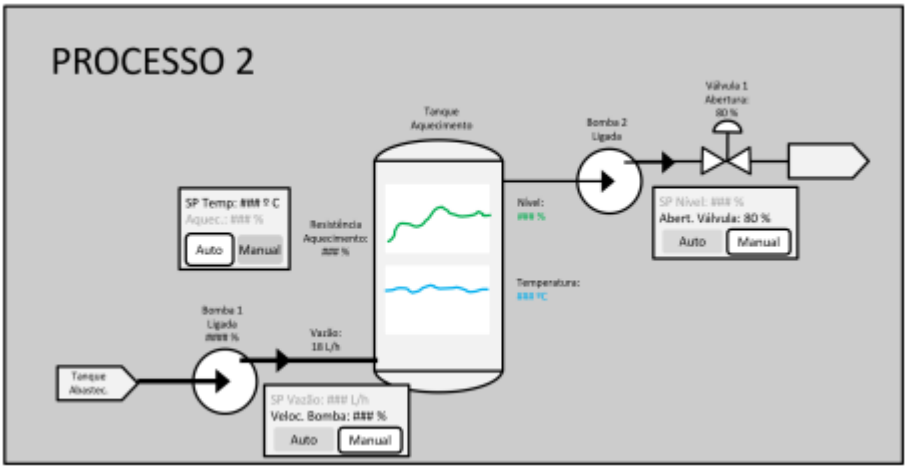
Fonte: (FAGUNDES, 2023)

2.2.4.2. Tela de Nível 2

As telas de Nível 2 são mais detalhadas e fornecem uma visão ampla e aprofundada das operações. Elas contêm mais informações do que as telas de Nível 1 e são a principal interface do operador durante as operações normais para monitoramento e modificações de rotina. As telas de Nível 2 podem incluir visões gerais da unidade de processo ou telas principais para cada sistema importante, como a unidade de processo controlada pelo operador *design*ado. Essas telas são frequentemente denominadas visões gerais do sistema e subsistema. O propósito das

telas de Nível 2 é facilitar a navegação para detalhes mais específicos fornecidos nas telas de Nível 3 e 4. Enquanto as telas de Nível 1 oferecem uma visão geral contínua da área de controle do operador, as telas de Nível 2 são projetadas com base em tarefas específicas, permitindo que o operador realize suas atividades com um número limitado de telas e uma navegação mínima. Sendo ilustrada na Figura 2 um exemplo de tela de nível 2.

Figura 2 – Tela de Nível 2



Fonte: (FAGUNDES, 2023)

Para garantir a eficácia das telas de Nível 2, é importante considerar o seguinte:

1. Incluir visões gerais da unidade de processo e telas principais para cada sistema importante, como a unidade de processo controlada pelo operador *design*ado. Essas telas são frequentemente chamadas de visões gerais do sistema e subsistema.
2. Exibir todos os alarmes de alta e média prioridade para o sistema ou subsistema específico.
3. Fornecer indicadores de navegação claros para os alarmes de baixa prioridade não exibidos.
4. Incluir informações e controladores essenciais dentro da interface principal para operar o sistema na maioria das condições.
5. Apresentar os principais controladores para a área de processo específica.

6. Exibir informações específicas da tarefa para operações de inicialização e/ou desligamento.

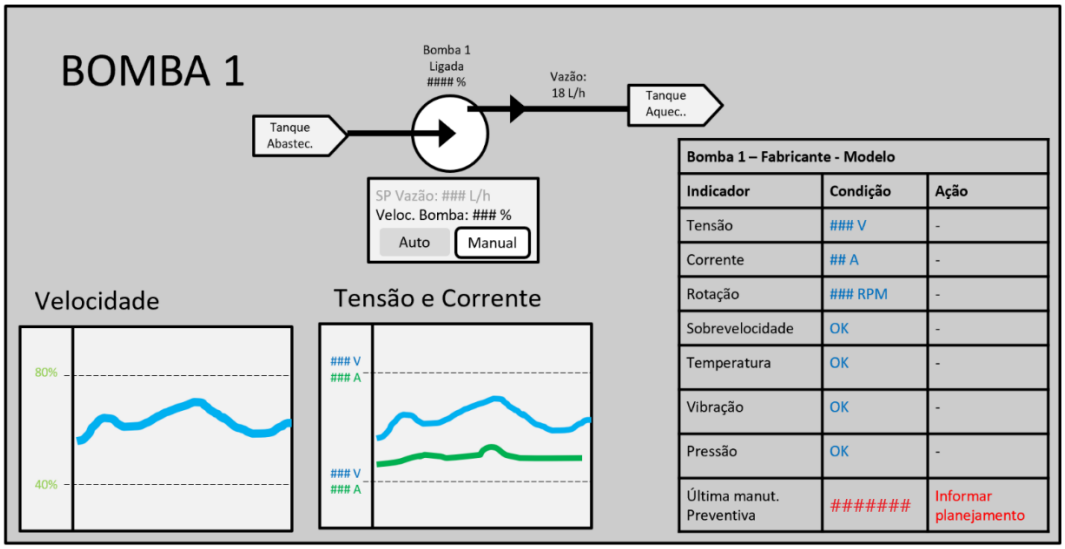
2.2.4.3. Tela de Nível 3

As telas de Nível 3 são caracterizadas como exibições detalhadas de sistemas ou subsistemas, oferecendo mais detalhes do que as telas de Nível 2 correspondentes. Elas são projetadas para serem utilizadas pelo operador em operações não rotineiras, como mudanças de configuração, trocas de equipamentos ou tarefas rotineiras complexas. As telas de Nível 3 devem fornecer informações suficientes para facilitar o diagnóstico do processo e são orientadas por tarefas, permitindo que o operador realize atividades com um número limitado de telas e uma navegação mínima. Sendo ilustrada na Figura 3 um exemplo de tela de nível 3.

Para as telas de Nível 3, as seguintes considerações são importantes:

1. Incluir os loops de controle e indicadores para os equipamentos do processo.
2. Exibir alarmes de todas as prioridades.
3. Mostrar o status de diversos bloqueios para o equipamento representado.

Figura 3 – Tela de Nível 3



Fonte: (FAGUNDES, 2023)

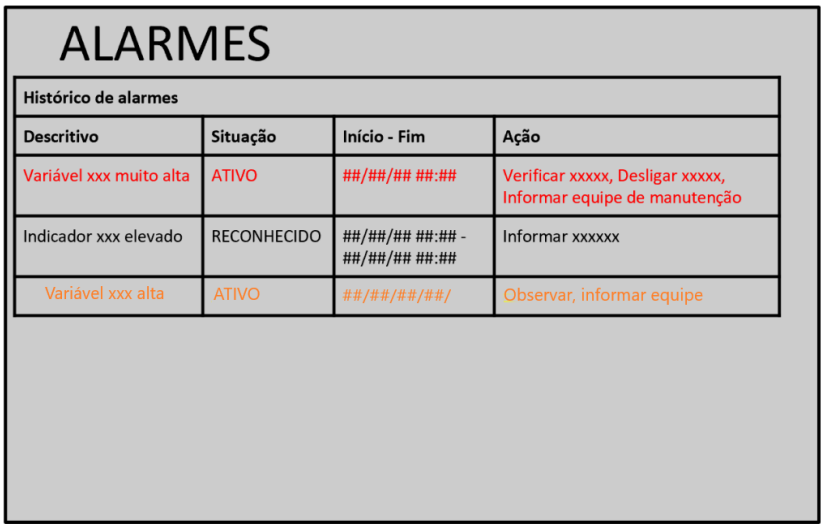
2.2.4.4. Tela de Nível 4

As telas de Nível 4 são específicas para diagnóstico, abrigando todas as informações essenciais do sistema. Embora não se destinem ao controle ativo do processo ou sistema, ainda podem oferecer funcionalidades de controle em pontos específicos. Diferentemente das outras telas, as de Nível 4 geralmente não precisam ocupar toda a tela, podendo exibir informações em *faceplates* ou *pop-ups*, devido à sua utilização breve e intermitente. Sendo ilustrada na Figura 4 um exemplo de tela de nível 4.

Essas telas são caracterizadas por:

1. Fornecer procedimentos operacionais detalhados para equipamentos individuais.
2. Oferecer informações de ajuda para o controle e diagnóstico do equipamento.
3. Incluir protocolos detalhados de desligamento de segurança.
4. Apresentar dados sobre intertravamentos e permissões do sistema.

Figura 4 – Tela de Nível 4



Fonte: (FAGUNDES, 2023)

2.3. Protocolo de Comunicação OPC

O OPC (*Open Platform Communications*) representa um padrão vital na comunicação industrial, permitindo a comunicação entre sistemas de automação e controle de fabricantes distintos. Desenvolvido pela Microsoft nos anos 90 como parte da tecnologia OLE (*Object Linking and Embedding*), o OPC surgiu em resposta à crescente diversidade de dispositivos e sistemas na indústria, simplificando a comunicação entre aplicativos em ambientes Windows (GONZÁLEZ e colab., 2019).

O primeiro padrão, conhecido como OPC *Data Access* (OPC DA), estreou em 1996, permitindo a troca instantânea de dados entre sistemas de controle e aplicativos Windows. Posteriormente, outros padrões OPC surgiram para atender às necessidades variadas da indústria, como o OPC *Alarms and Events* (OPC AE) para gerenciamento de alarmes e eventos, e o OPC *Unified Architecture* (OPC UA) para proporcionar uma solução avançada e segura de comunicação (GONZÁLEZ e colab., 2019).

As vantagens do protocolo OPC são significativas: ele garante interoperabilidade, é um padrão aberto, facilita a integração, possibilita comunicação em tempo real, gerencia alarmes e eventos, oferece segurança robusta, reduz custos

operacionais, é fácil de configurar e diagnosticar, além de ser compatível com uma variedade de dispositivos industriais. Essas características fazem do OPC uma escolha fundamental na indústria, promovendo a eficiência, segurança e integridade dos sistemas de automação e controle (GONZÁLEZ e colab., 2019).

2.3.1. OPC UA

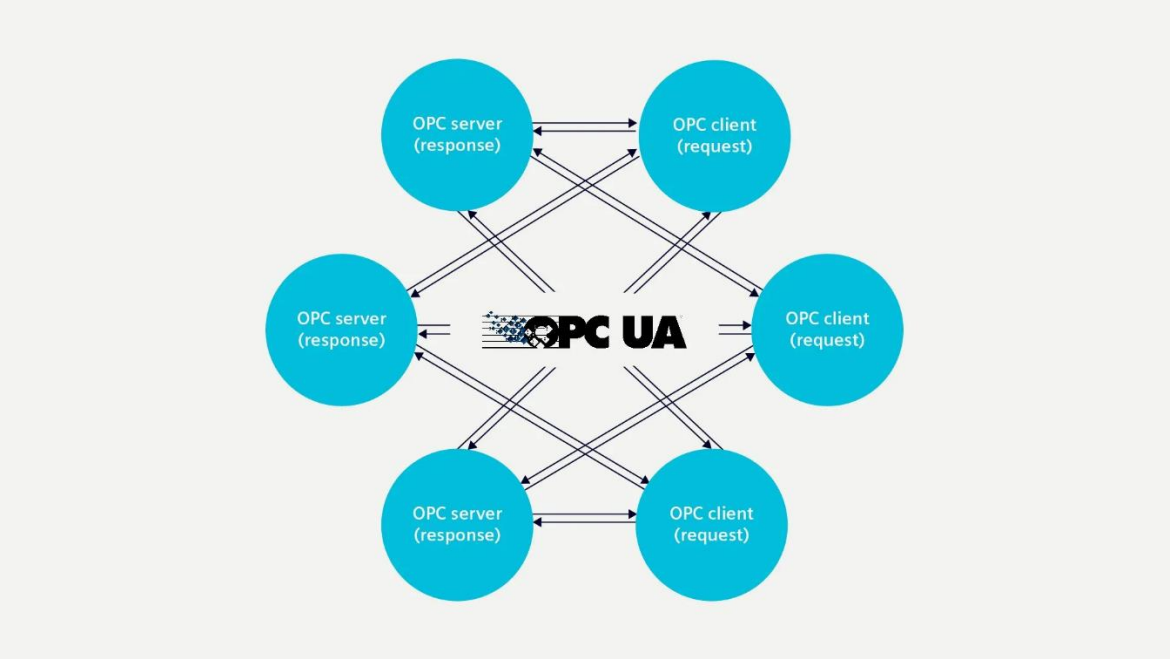
O OPC UA (*Open Platform Communications - Unified Architecture*) representa um padrão aberto para comunicação horizontal entre máquinas (M2M) e vertical, conectando máquinas à nuvem. Esse protocolo é independente de fornecedor ou plataforma, suportando mecanismos avançados de segurança. Além disso, pode ser integrado de maneira otimizada em uma rede Ethernet Industrial compartilhada (DERHAMY e colab., 2017).

Uma característica fundamental do OPC UA é sua neutralidade em relação à plataforma, permitindo comunicações contínuas com aplicativos de terceiros. Sua flexibilidade permite configurações específicas para atender a requisitos variados. Além de transmitir dados, o OPC UA também incorpora um modelo de informações exclusivo, proporcionando uma estrutura organizada (DERHAMY e colab., 2017).

Em termos de segurança, o OPC UA adota práticas cibernéticas que são consideradas avançadas dentro do meio industrial, que necessita de respostas em tempo real. Essas práticas são autenticação, autorização e criptografia, garantindo a integridade dos dados durante as trocas. Ele pode ser facilmente integrado em redes Ethernet Industrial existentes, sem prejudicar o desempenho (DERHAMY e colab., 2017).

O OPC UA opera em diferentes modelos, sendo o cliente/servidor um deles. Nesse formato, os clientes OPC UA acessam dados em um servidor OPC UA por meio de comunicações ponto a ponto, garantindo trocas de dados criptografadas, seguras e eficientes. Mesmo em ambientes de rede desafiadores, essa comunicação mantém a integridade dos dados (POGACEAN e colab., 2016). A Figura 5 contém uma ilustração do sentido dessa comunicação.

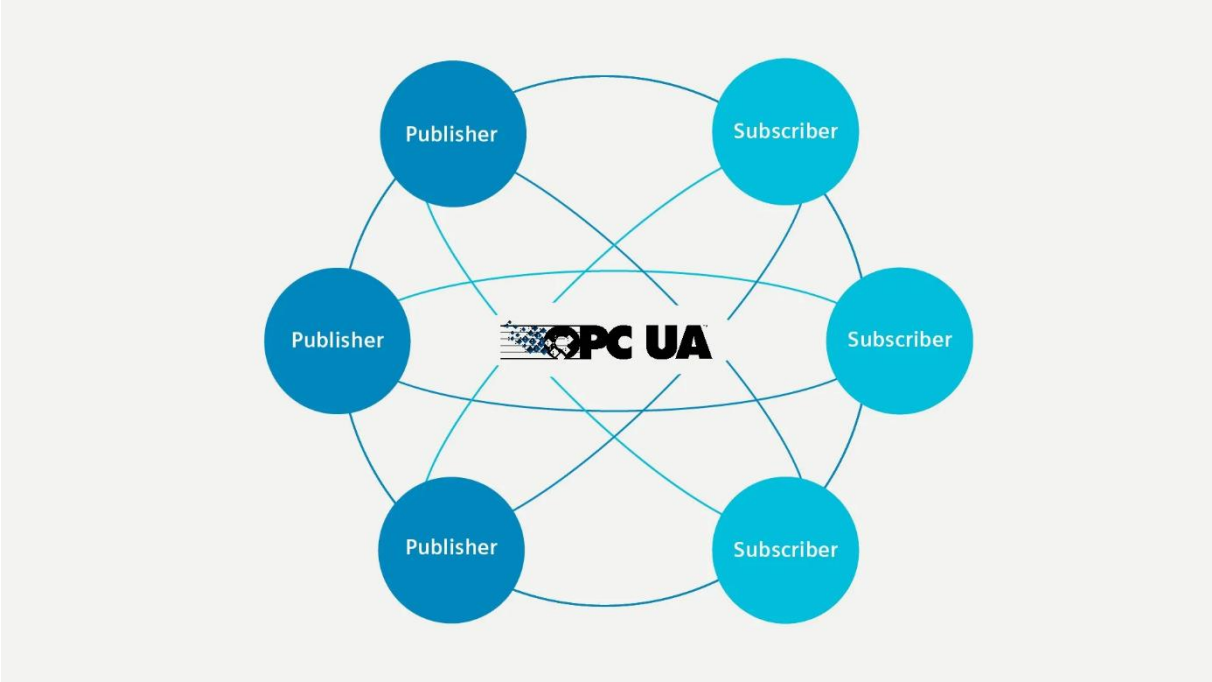
Figura 5 – OPC/UA, Cliente/Servidor



Fonte: (SIEMENS BRASIL, 2023)

Outro modelo amplamente utilizado é o OPC UA com PubSub (*publish/subscribe*), baseado no conceito um-para-muitos ou muitos-para-um. Nesse cenário, um *publisher* disponibiliza dados para uma quantidade ilimitada de *subscribers* na rede. A comunicação OPC UA PubSub pode ocorrer via *User Datagram Protocol* (UDP) ou diretamente na camada 2, resultando em ciclos de comunicação rápidos, especialmente quando combinado com o *Time-Sensitive Networking* (TSN). Isso viabiliza a comunicação em tempo real no nível de controle, atendendo às exigências de aplicações sensíveis ao tempo (POGACEAN e colab., 2016). A Figura 6 contém uma ilustração do sentido dessa comunicação.

Figura 6 – OPC/UA, PubSub



Fonte: (SIEMENS, 2023)

2.4. Tecnologias e Ferramentas *Web*

Neste subcapítulo da revisão bibliográfica, abordaremos os fundamentos essenciais da *Web*, incluindo HTML, CSS e Javascript. Exploraremos também um *framework* relevante no desenvolvimento *frontend*, o Bootstrap. Por fim, dedicaremos atenção à ferramenta utilizada para o desenvolvimento do *backend* da aplicação *web*, o .NET, e à sua linguagem associada, o C#.

2.4.1. HTML, CSS e Javascript

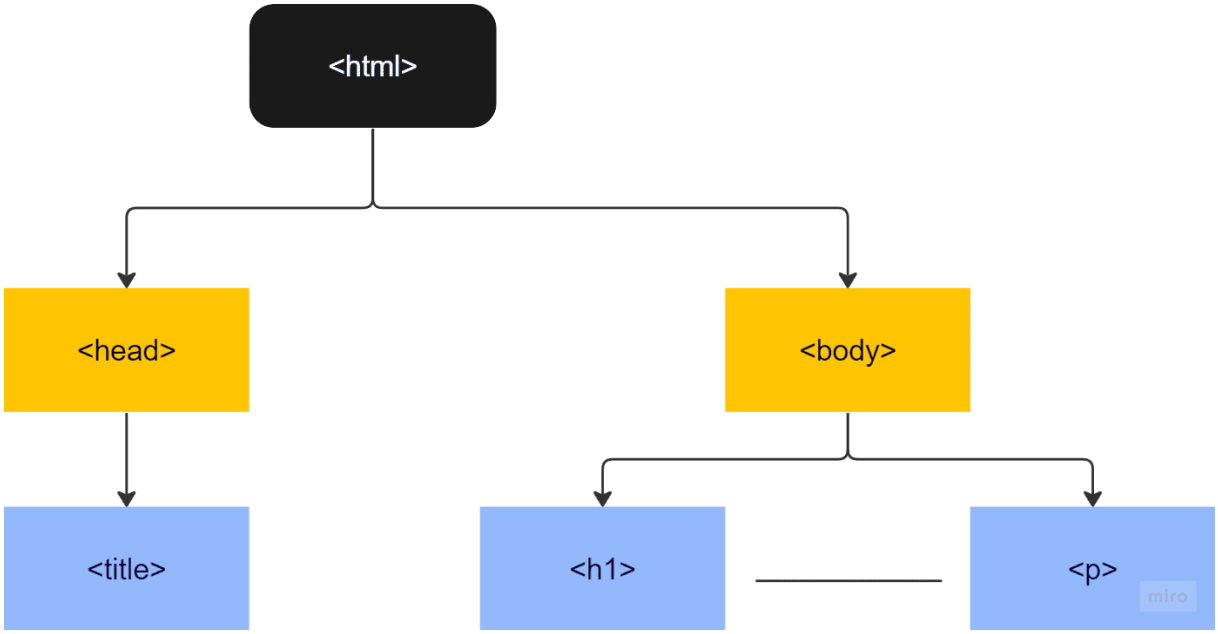
HTML (*HyperText Markup Language*) é uma linguagem fundamental para estruturar e formatar o conteúdo das páginas *web*. Para criar um projeto utilizando HTML, pode-se utilizar um arquivo com extensão .html que pode ser aberto em um navegador. O navegador interpreta o código-fonte presente no arquivo .html para dispor os elementos na tela (HILLMAN, 2021).

A estrutura de um documento HTML pode ser comparada a uma árvore, onde temos a raiz representada pelo elemento <html>. A partir da raiz, é possível declarar novos elementos, que podem ser considerados como ramos da árvore. Esses ramos podem ter ramificações adicionais, aninhando elementos, o que cria pais e filhos na hierarquia. Elementos que compartilham do mesmo ramo de origem são chamados de irmãos (MDN CONTRIBUTORS, 2023b).

Cada elemento HTML é construído com uma *tag* de abertura, cercada por colchetes angulares. Por exemplo, a *tag* de parágrafo é seguida pelo conteúdo a ser mostrado e é fechada por uma *tag* idêntica à de abertura, mas com uma barra antes do nome do elemento: (MDN CONTRIBUTORS, 2023b).

A representação estruturada do conteúdo HTML de uma página *web* é feita através do DOM (*Document Object Model*), que é uma interface de programação permitindo o acesso e manipulação dos elementos (MDN CONTRIBUTORS, 2023b). O DOM organiza os elementos em forma de árvore, onde cada elemento é representado por um nó, organizado hierarquicamente de acordo com a estrutura do documento. Na Figura 7 está ilustrada a estrutura básica do DOM:

Figura 7 – Estrutura básica do DOM



Fonte: (HILLMAN, 2021)

Antes do surgimento do CSS (*Cascading Style Sheets*), a estilização de páginas *web* era feita principalmente por meio de elementos HTML e atributos de estilo embutidos. Isso significava que o estilo estava misturado com a estrutura do documento HTML, tornando o código mais difícil de manter e modificar (NEVES, 2021).

Em 1994, o engenheiro Håkon Wium Lie introduziu uma solução para o caos visual das páginas HTML da época: o CSS, ou *Cascading Style Sheets*. Em parceria com o W3C (*World Wide Web Consortium*), ele propôs uma linguagem de estilização que permitiria aos desenvolvedores implementarem e controlar estilos visuais de maneira eficaz (NEVES, 2021).

O CSS evoluiu ao longo do tempo, progredindo em diferentes níveis em vez de versões distintas:

CSS1: Lançado oficialmente em 1996, permitiu aos desenvolvedores separarem o conteúdo do *design* em seus sites pela primeira vez, marcando um avanço significativo.

CSS2: Em 1998, trouxe funcionalidades adicionais, como suporte para tipos de mídias, possibilitando estilos específicos para impressão e visualização na tela.

CSS3: Esta fase introduziu complexidade ao ser dividida em módulos menores, abordando diversos aspectos do CSS. Inovações notáveis incluem efeitos de animação, transições, transformações e o versátil *flexbox*.

Os seletores em CSS desempenham um papel fundamental na estilização das páginas *web*, permitindo a identificação de elementos com base em padrões específicos. Esses seletores capacitam os desenvolvedores a direcionarem com precisão os elementos HTML desejados, seja com base no tipo do elemento, por meio de classes ou IDs atribuídos a eles. Essa capacidade de seleção refinada é essencial para aplicar estilos e layout de maneira personalizada, tornando o CSS uma linguagem poderosa para o *design* e apresentação de conteúdo na *web* (NEVES, 2021). Os seletores em CSS ajudam a identificar elementos com base em padrões específicos:

1. Tipo: Seleciona elementos pelo nome do elemento, como p, div, h1, etc.
2. Classe: Seleciona elementos pela classe, usando um ponto (.) seguido do nome da classe.
3. ID: Seleciona elementos pelo ID, usando uma *hashtag* (#) seguida do nome do ID.

As propriedades em CSS controlam diversos aspectos visuais, como cor do texto, tamanho da borda, fonte e espaçamento. Com centenas de propriedades disponíveis, oferece um controle minucioso sobre a aparência das páginas *web*. Cada propriedade possui valores específicos, podendo ser simples, como "*auto*" ou "*none*", ou valores mais complexos, como números, cores ou *strings* de texto (MDN CONTRIBUTORS, 2023a).

O posicionamento dos elementos em CSS é uma habilidade fundamental que implica organizar diversos blocos em um espaço restrito. Essa disposição é alcançada por meio de propriedades como "*position*", "*top*", "*right*", "*bottom*", "*left*" e "*z-index*", que oferecem controle preciso sobre a localização e empilhamento dos elementos na página. Cada elemento é visualizado como uma caixa retangular, seguindo o modelo

conhecido como *Box Model*, que é dividido em quatro áreas distintas: a margem (*margin*), que define o espaço ao redor da caixa, funcionando como um *buffer* entre o elemento e seu entorno; a borda (*border*), que é a linha que circunda o preenchimento e o conteúdo interno; e o preenchimento *(padding*), um espaço entre a borda e o conteúdo, atuando como uma almofada que protege o conteúdo da borda. Essa compreensão detalhada do posicionamento e da estrutura dos elementos é essencial para criar layouts eficazes e visualmente atraentes em páginas *web* (MDN CONTRIBUTORS, 2023a).

Uma característica crucial do CSS é sua capacidade de tornar os *design*s responsivos, o que significa que as páginas se adaptam de forma fluida a diferentes ambientes, incluindo variados tamanhos de tela e dispositivos. Esse recurso é possível graças às *media queries*, que permitem ajustar estilos conforme o tamanho da tela, proporcionando uma experiência de usuário perfeitamente adaptada ao dispositivo em uso (MDN CONTRIBUTORS, 2023a).

JavaScript, em conjunto com HTML e CSS, desempenha um papel essencial na criação de páginas *web* dinâmicas e interativas. Enquanto o HTML fornece a estrutura básica da página, definindo elementos e seu conteúdo, o CSS é responsável pelo estilo e pela apresentação visual. Já o JavaScript adiciona uma camada de dinamismo, permitindo a manipulação do conteúdo e interações do usuário.

JavaScript é uma linguagem interpretada e orientada a objetos, com funções de primeira classe, sendo amplamente reconhecida como a principal linguagem de *script* para desenvolvimento *web*. Sua arquitetura é fundamentada em protótipos, destacando-se por ser uma linguagem multi-paradigma e dinâmica, capaz de suportar abordagens variadas, incluindo orientação a objetos, imperativa e declarativa (MDN CONTRIBUTORS, 2022).

Esta linguagem se destaca por sua concisão e notável flexibilidade. Sua natureza compacta não compromete sua capacidade de abordar uma ampla gama de aplicações (MDN CONTRIBUTORS, 2022). Entre suas diversas utilidades, destacam-se a criação de interfaces de programação de aplicativos integradas a navegadores *web*, APIs de terceiros que possibilitam a incorporação de novas funcionalidades em diferentes provedores de conteúdo, e o aproveitamento de estruturas e bibliotecas de

terceiros que agilizam significativamente o desenvolvimento de *web*sites e aplicativos ao serem aplicadas ao HTML. Essa versatilidade torna possível atender às demandas de projetos diversos de forma eficiente e adaptável.

2.4.2. Bootstrap

O Bootstrap é um robusto *framework frontend* que simplifica e acelera a criação de sites e aplicações responsivas. Ele oferece uma variedade de estruturas de CSS prontas para uso, garantindo a consistência e o *design* visualmente atraente em diversas plataformas, desde desktops até dispositivos móveis (DÉRAMOND, 2023).

Este *framework* versátil oferece uma série de propriedades e classes essenciais para o desenvolvimento *web*:

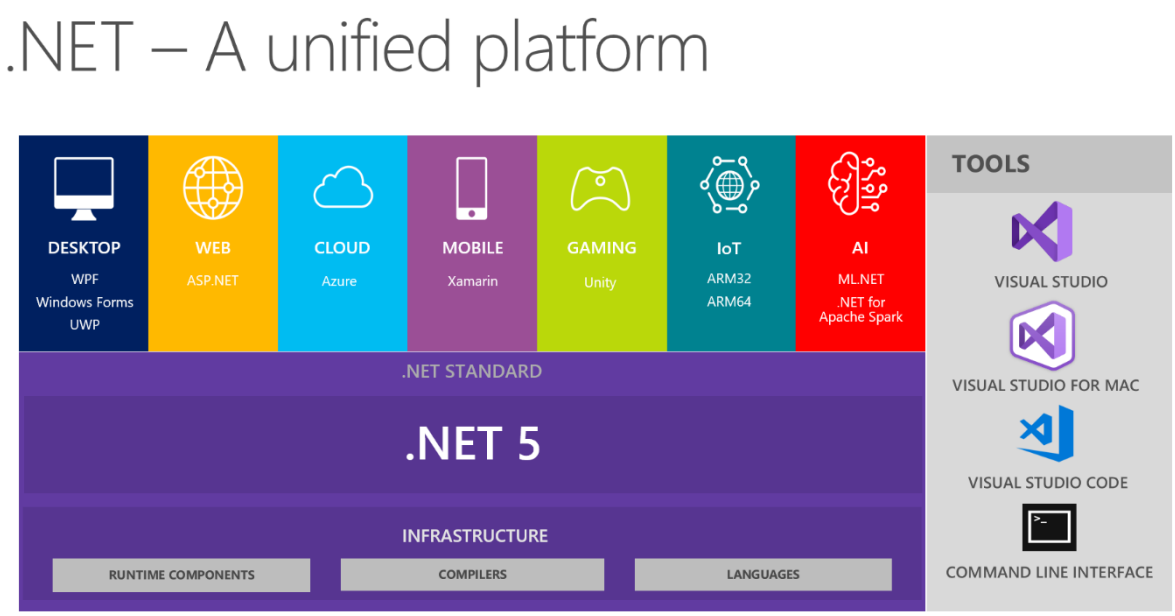
1. Sistema de *Grid*: Um sistema de *grid* responsivo baseado em colunas permite a criação fácil e flexível de *layouts* adaptáveis a diferentes tamanhos de tela e dispositivos.
2. Estilização de Texto: Classes para modificar tamanhos de fonte, negrito, itálico, entre outros, proporcionam controle completo sobre a tipografia.
3. Cores e Espaçamento: Classes predefinidas para cores e utilitários para ajustar espaçamentos interno (*padding*) e externo (*margin*) garantem um *design* harmonioso.
4. Bordas e Alertas: Classes para adição ou remoção de bordas, além de estilos específicos para alertas, como sucesso, aviso e erro, oferecem *feedback* visual claro.
5. Botões e Formulários: Estilos para criar botões com diferentes tamanhos, formas e cores, juntamente com classes para personalizar campos de formulário, como input, *checkbox* e *radio*.
6. Componentes Interativos: O Bootstrap oferece uma variedade de componentes interativos, incluindo barra de navegação responsiva, Carrossel para imagens rotativas, Modais para pop-ups personalizados e Abas (*Tabs*) para navegação intuitiva entre seções de conteúdo.
7. *Cards*, *Dropdowns* e Ícones: Estilos predefinidos para criar cartões informativos (*Cards*) com suporte a imagens, títulos e botões, além de

dropdowns elegantes e integração com bibliotecas populares de ícones, como *FontAwesome*, para inclusão fácil de ícones personalizados.

2.4.3. .NET E C#

O .NET é uma plataforma de desenvolvimento de código aberto e gratuita, que oferece suporte à criação de uma variedade de aplicações em diferentes plataformas. Esta abrangência inclui aplicações de *desktop, web, cloud, mobile, gaming*, IoT e IA, conforme ilustrado na Figura 8. Para o desenvolvimento dessas aplicações, destacam-se ferramentas essenciais como o Visual Studio, Visual Studio for Mac, Visual Studio Code, além do próprio console (MICROSOFT, 2023c).

Figura 8 – Plataforma .NET



Fonte: (RICH LANDER, 2019)

Essa plataforma oferece a flexibilidade de utilizar diversas linguagens de programação, entre as quais se destacam o C#, F# e Visual Basic. O C#, em particular, é a linguagem mais utilizada, caracterizando-se como uma linguagem orientada a objetos e componentes. A robustez e confiabilidade do C# são fundamentadas em recursos essenciais, como a coleta de lixo, que automaticamente libera a memória ocupada por objetos não acessíveis (MICROSOFT, 2023e).

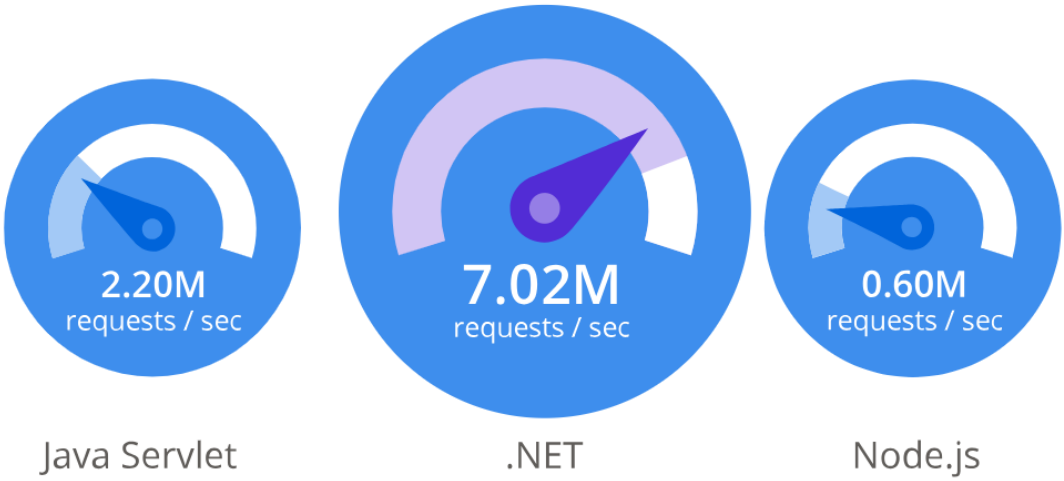
O tratamento de exceções oferece uma abordagem estruturada e extensível para detecção e recuperação de erros, enquanto expressões *Lambda* adicionam suporte a técnicas de programação funcional. O LINQ *(Language Integrated Query)* estabelece um padrão para trabalhar com dados provenientes de diversas fontes. O suporte à linguagem para operações assíncronas proporciona uma sintaxe para a criação de sistemas distribuídos(MICROSOFT, 2023e).

O C# permite a alocação dinâmica de objetos e o armazenamento eficiente de estruturas de sintaxe simples. Adicionalmente, oferece suporte a tipos de referência e tipos de valor definidos pelo usuário. A alocação dinâmica de objetos e o armazenamento eficiente de estruturas leves em linha são possíveis com o C#. Além disso, a linguagem proporciona suporte a métodos e tipos genéricos, oferecendo maior segurança e desempenho(MICROSOFT, 2023e).

Entre as vantagens da utilização do .NET, destacam-se sua versatilidade, evidenciada na Figura 8, e o suporte ao código aberto proporcionado pela .NET Foundation. Esta fundação conta com mais de 100.000 contribuições provenientes de mais de 3700 empresas distintas. Vale ressaltar ainda que o .NET é reconhecido por sua confiabilidade e segurança, respaldado pelo suporte ágil da Microsoft, que realiza atualizações de segurança de forma rápida diante de ameaças reportadas (MICROSOFT, 2023d).

Um fator de extrema importância é o desempenho oferecido pelo .NET, comprovado por testes conduzidos pelo TECHEMPOWER, que está ilustrado na Figura 9. Esses testes englobam tarefas como serialização JSON, acesso a banco de dados e renderização de modelo do lado do servidor (TECHEMPOWER, 2022).

Figura 9 – Comparação de Desempenho .NET



Fonte: (MICROSOFT, 2023a)

Para desenvolver aplicações *web* com o .NET, é necessário utilizar o ASP.NET, um *framework* que, por sua vez, se baseia no .NET. Essa estrutura abrange todas as funcionalidades do .NET e expande essa plataforma com ferramentas e bibliotecas dedicadas à criação *web*. Entre essas ferramentas, destacam-se o *framework* básico para solicitações *web* em C#, o Razor - uma sintaxe para modelagem de páginas *web*, o padrão de arquitetura MVC - Modelo-Visão-Controlador, o sistema de autenticação e as extensões do editor (MICROSOFT, 2023b).

O Razor, mencionado anteriormente, oferece uma sintaxe para criar páginas *web* dinâmicas utilizando HTML e C#. O código é avaliado pelo servidor, e o conteúdo HTML resultante é enviado ao usuário. O ASP.NET integra-se com estruturas JavaScript e inclui modelos predefinidos para estruturas SPA - aplicativos de página única, como React e Angular (MICROSOFT, 2023b).

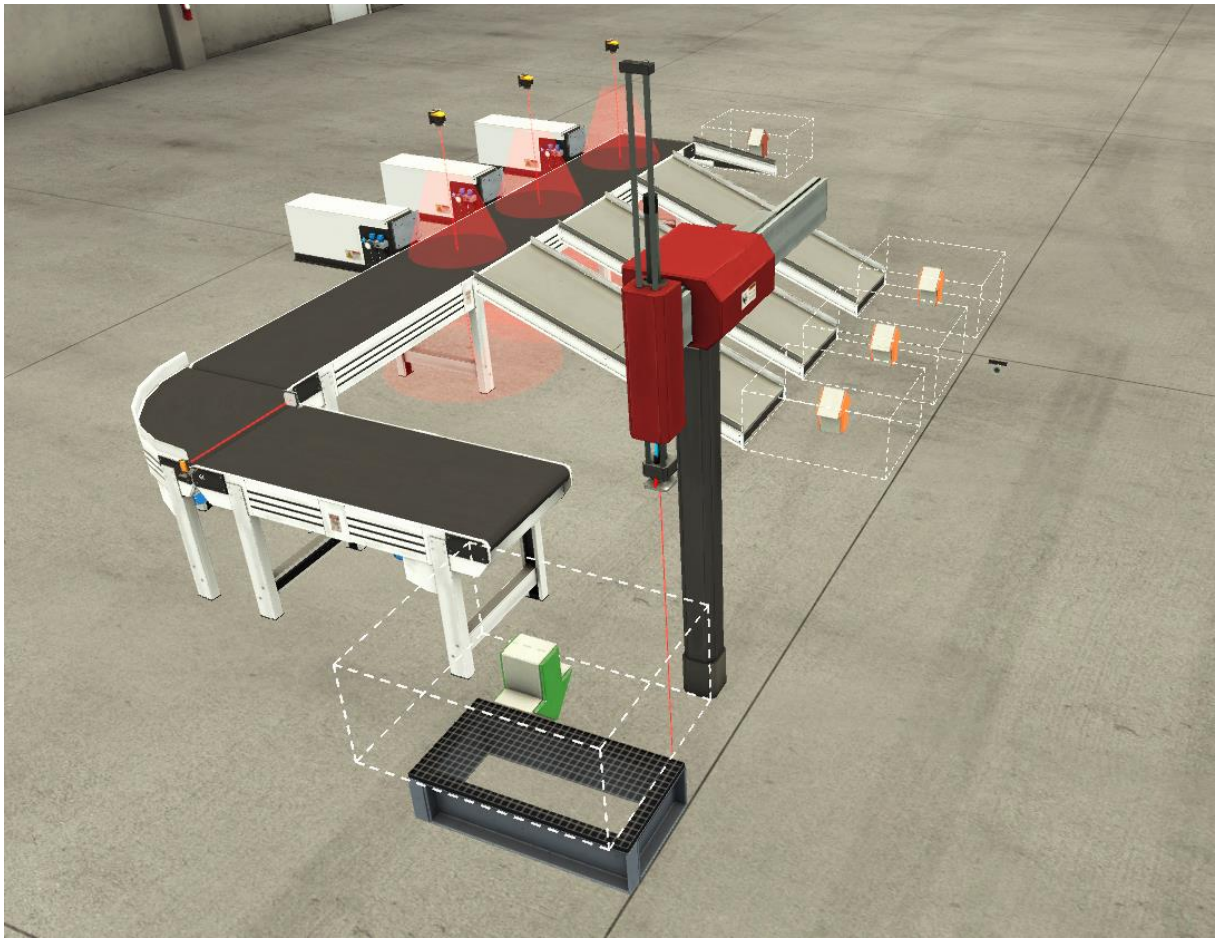
3. Metodologia

O projeto foi desenvolvido em uma sequência estruturada de etapas. Inicialmente, a planta de separação de peças foi adaptada no Factory I/O, seguida pela criação da programação em *ladder* no Codesys, que também foi utilizado para criar e configurar o servidor OPC. Na sequência, foram realizados testes de conexão e desenvolvida a API REST utilizando as bibliotecas do *framework* ASP.NET. Por fim, *o front-end* foi implementado com CSS, HTML, JavaScript e Bootstrap. Cada uma dessas etapas será detalhada nos subcapítulos a seguir

3.1. Aplicação do Factory.IO

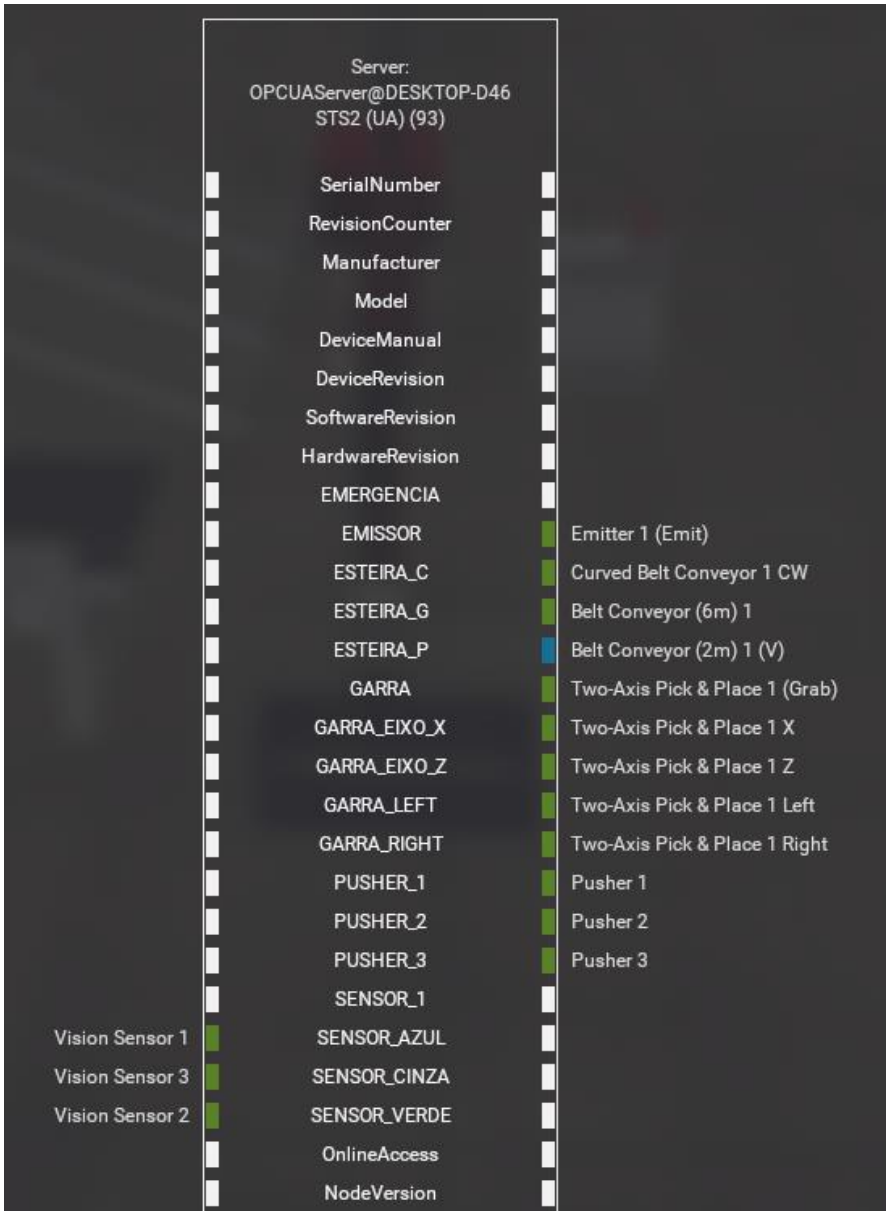
A planta de separação de peças, ilustrada na Figura 10, utilizada no projeto foi baseada nas aulas de Controladores Lógicos Digitais, ministradas pelo professor Willian, um dos orientadores do trabalho. O primeiro passo foi estabelecer a conexão com o servidor OPC e correlacionar as variáveis da lógica *ladder*, conforme ilustrado na Figura 11. Na parte superior da imagem, pode-se visualizar o nome do servidor conectado. No centro, encontram-se listadas todas as variáveis presentes no servidor. À esquerda, estão os sensores da simulação, vinculados às variáveis correspondentes do servidor, enquanto à direita, estão os elementos de saída da simulação, também associados às suas respectivas variáveis no servidor. Esse mapeamento foi essencial para garantir o correto funcionamento da integração entre a simulação e o sistema

Figura 10 – Planta de Separação de Peças Factory IO



Fonte: (Imagem do Autor)

Figura 11 – Conexão OPC/FACTORY.IO



Fonte: (Imagem do Autor)

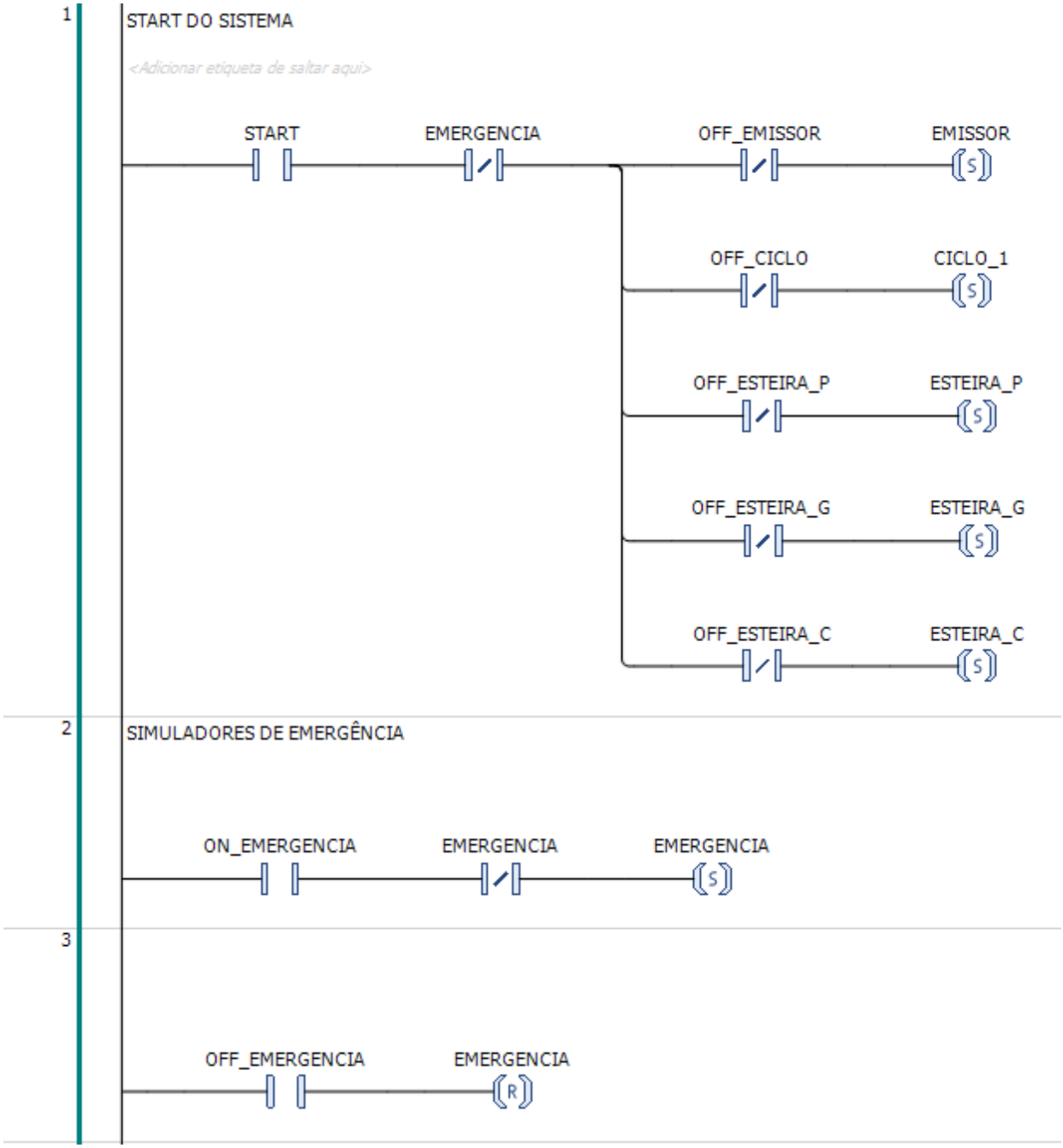
Em seguida, foi necessário ajustar a posição dos sensores e modificar o intervalo de surgimento das peças para otimizar o funcionamento da planta. Esses ajustes permitiram uma melhor adequação ao processo de classificação. A planta é composta por diversos componentes, incluindo uma garra para movimentação das peças, uma esteira grande, uma esteira pequena e uma em curva. Além disso, há três sensores, um para cada tipo de cor das peças, três empurradores e um emissor responsável por liberar as peças no sistema. Após essas modificações, a planta ficou como ilustrado na Figura 10.

3.2. Programação pelo Codesys para automação do processo

Inicialmente, foi necessário configurar o servidor OPC, o que incluiu a definição de permissões de acesso para garantir a segurança e o controle adequado da comunicação entre os dispositivos. Após essa etapa, iniciou-se o desenvolvimento da programação em *ladder*, seguindo um processo estruturado. Foram configuradas as variáveis associadas aos sensores e atuadores da planta, além de implementadas as interações entre os diferentes elementos do sistema, como os empurradores, esteiras e o emissor de peças.

A lógica do sistema pode ser dividida em sete partes distintas. A primeira é o 'Start' (Figura 12), responsável por iniciar o funcionamento da planta. Em seguida, temos os simuladores de emergência, que testam o comportamento da planta em cenários de falha (Figura 12). Os 'Sets' (Figura 13 e Figura 14) controlam a ativação de diferentes componentes, enquanto os 'Resets' (Figura 15, Figura 16 e Figura 17) realizam a função oposta, desativando-os quando necessário. Os sensores acionam os empurradores ('*pushers*') para direcionar as peças (Figura 18). O 'Ciclo 1' (Figura 19) é responsável por mover as peças até a esteira, enquanto o 'Ciclo 2' cuida do retorno da garra, reiniciando o processo (Figura 20).

Figura 12 – *Ladder* Start/Emergência

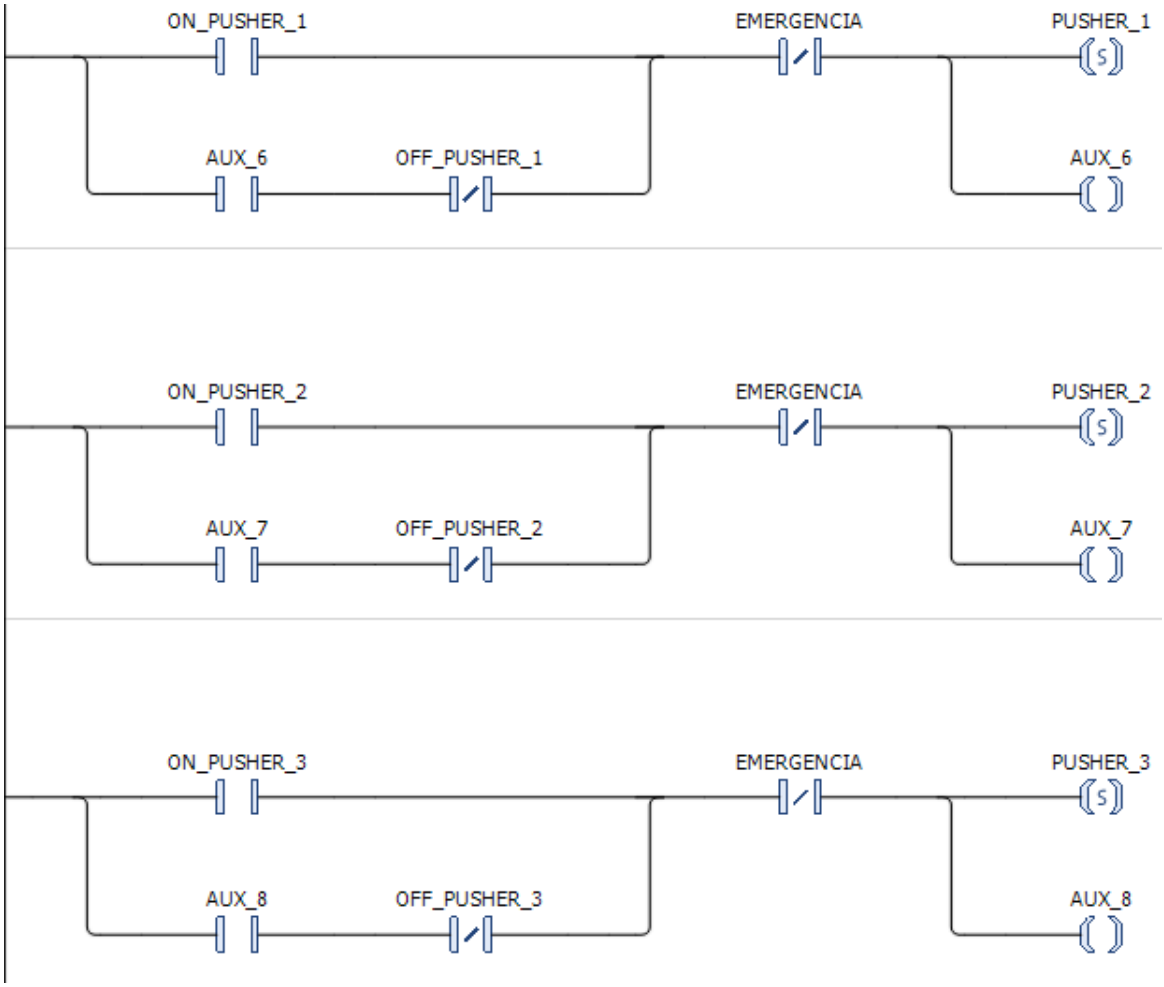


Fonte: (Imagem do Autor)

Figura 13 – *Ladder* Sets 1

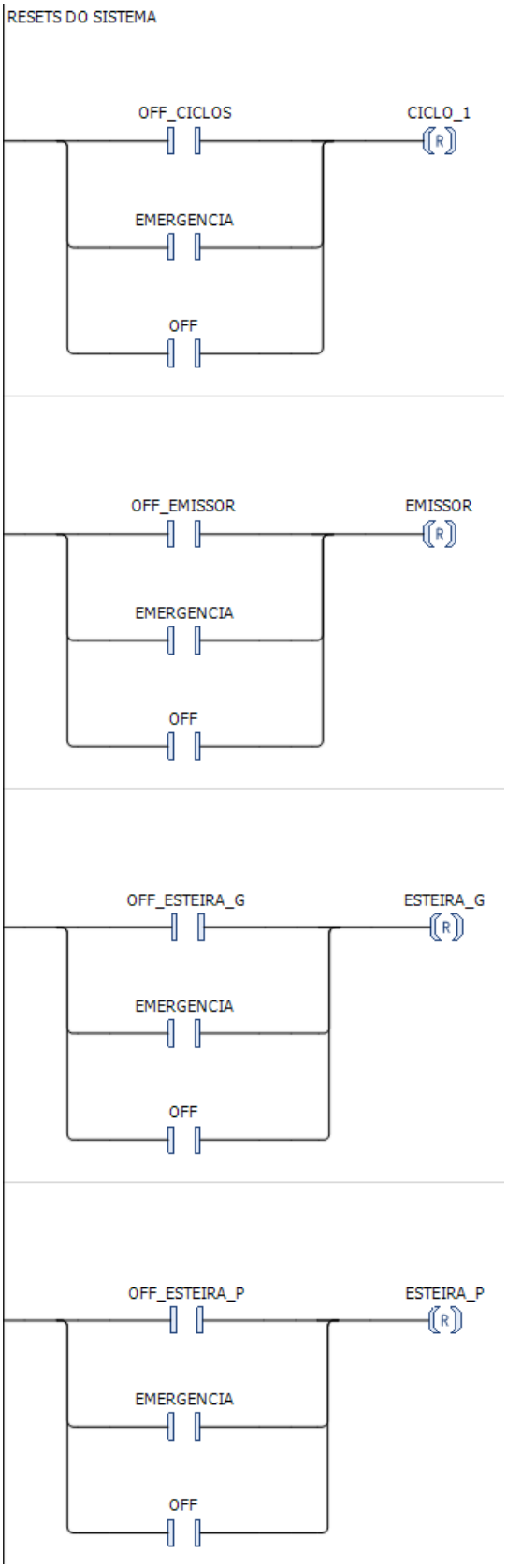
Fonte: (Imagem do Autor)

Figura 14 – *Ladder* Sets 2



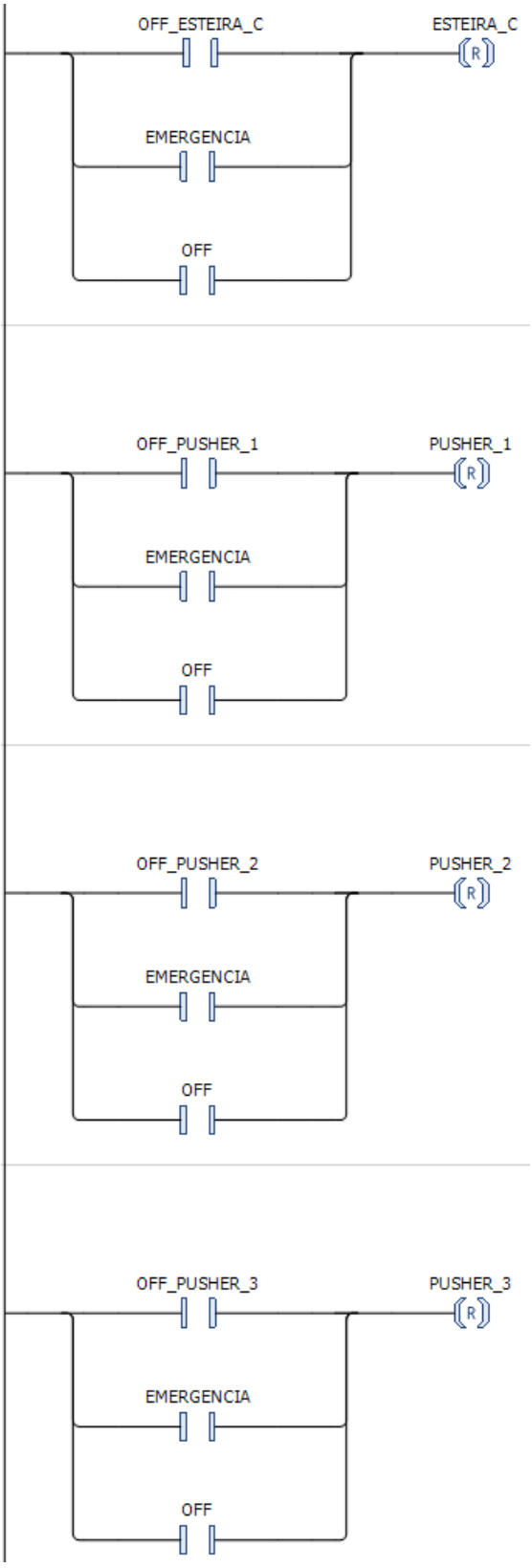
Fonte: (Imagem do Autor)

Figura 15 – *Ladder* Resets 1



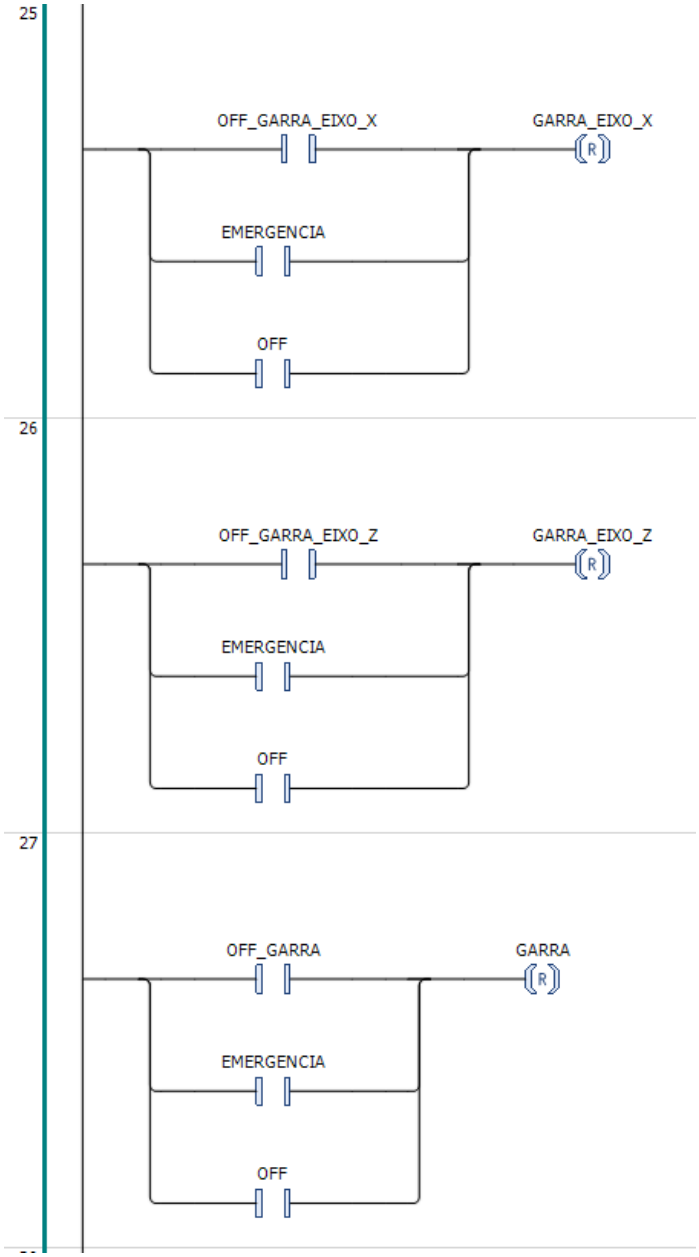
Fonte: Imagem do Autor

Figura 16 – *Ladder* Resets 2



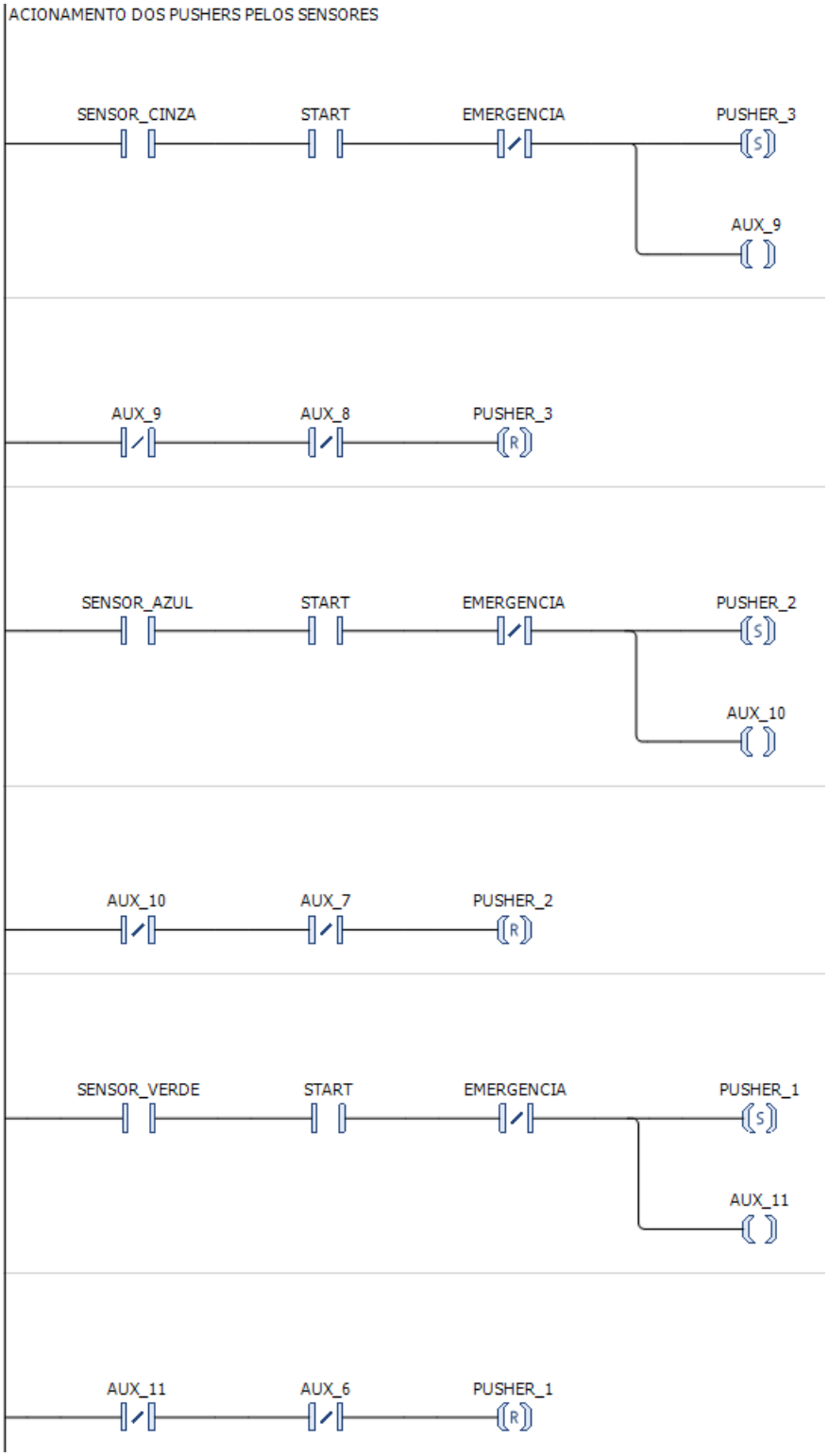
Fonte: (Imagem do Autor)

Figura 17 – *Ladder* Resets 3



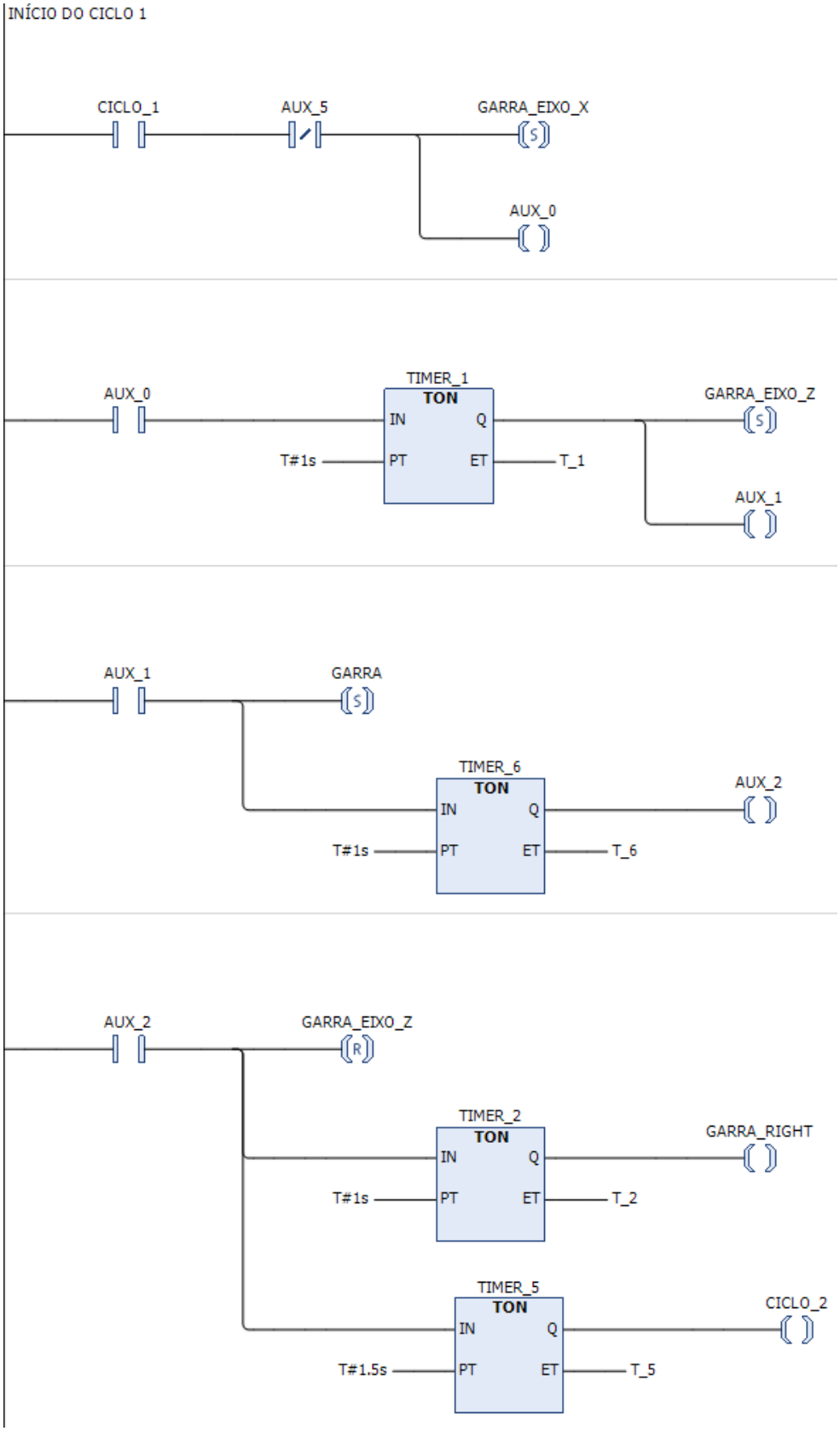
Fonte: (Imagem do Autor)

Figura 18 – *Ladder* Sensores



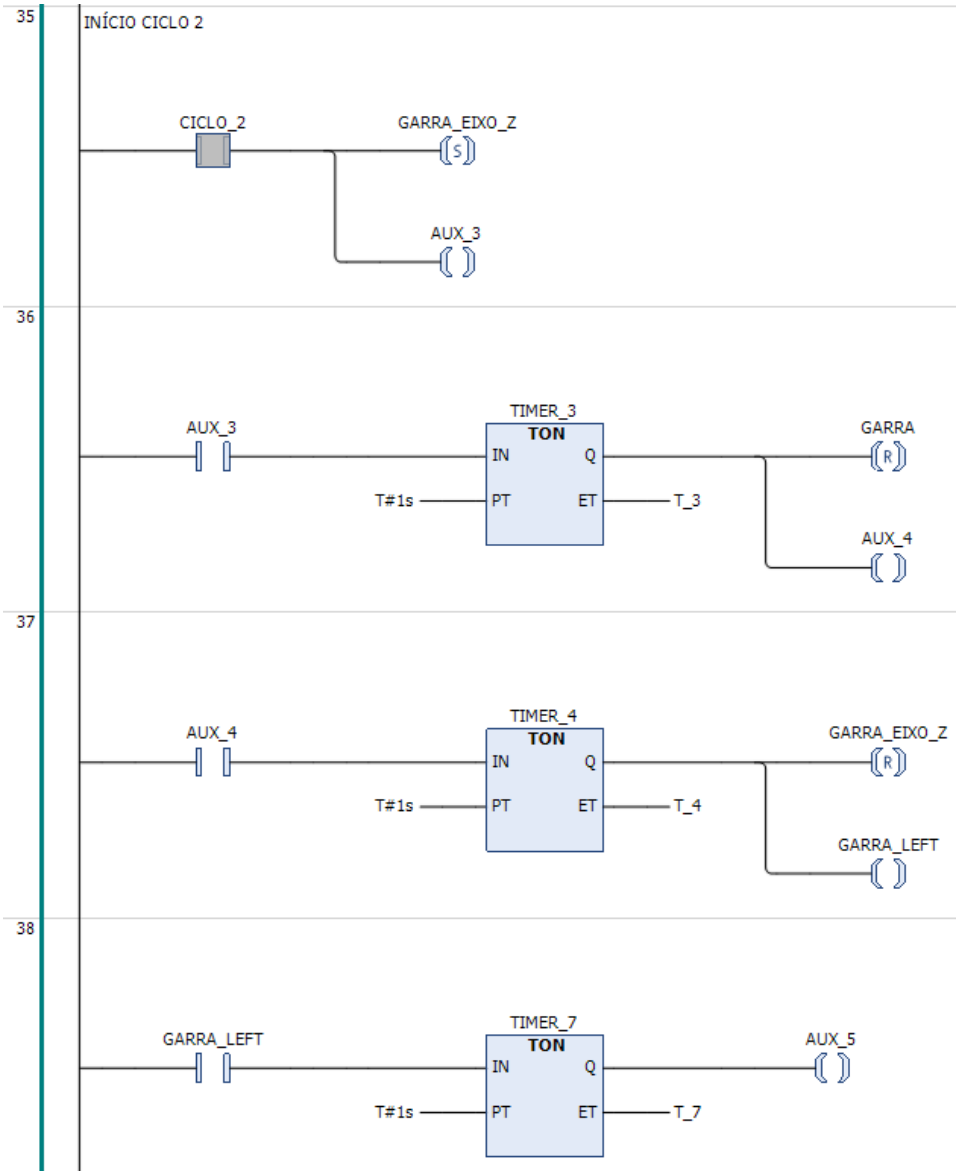
Fonte: Imagem do Autor

Figura 19 – *Ladder* Ciclo 1



Fonte: Imagem do Autor

Figura 20 – *Ladder* Ciclo 2



Fonte: (Imagem do Autor)

3.3. Desenvolvimento da API REST pelo ASP.NET

Para estabelecer a comunicação com o servidor OPC, utilizou-se a biblioteca Opc.UaFx.Client. O primeiro passo foi testar a conexão para identificar o NodeId de cada variável associada aos componentes da planta. Esse processo foi essencial para garantir que as variáveis corretas fossem acessadas e que fosse possível realizar alterações nelas conforme necessário. A partir desses testes, foi possível assegurar que o sistema poderia ler e modificar as variáveis em tempo real, permitindo uma

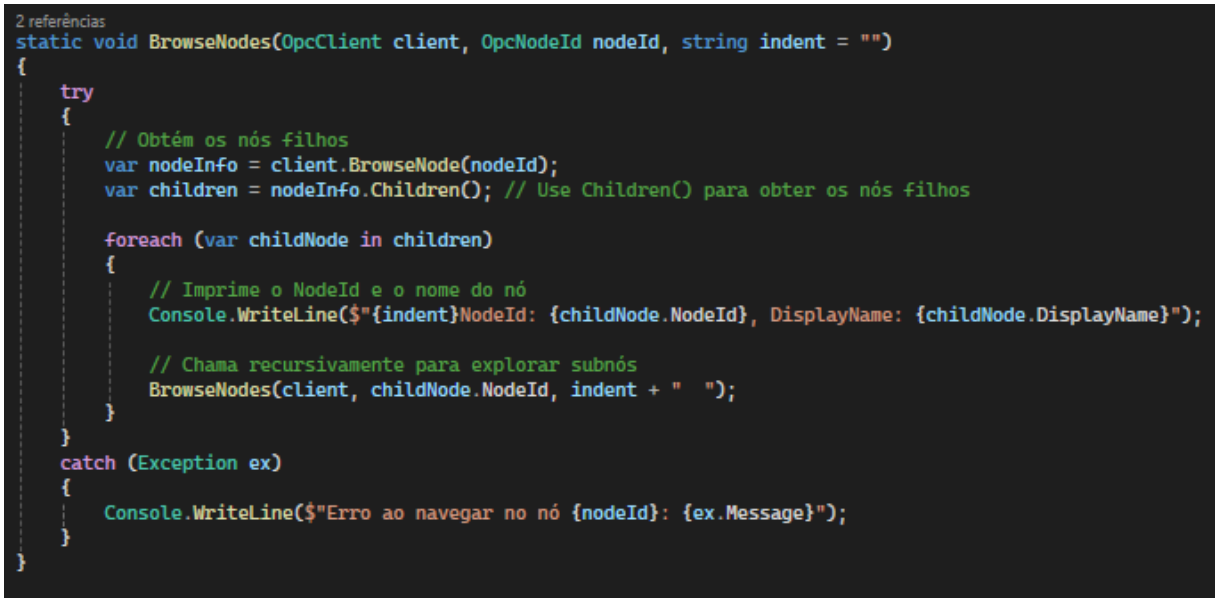
integração eficaz entre o servidor OPC e *backend*. O código utilizado nesse teste está na Figura 21 e na Figura 22.

Figura 21 – Código Teste Comunicação OPC 1

```
using Opc.UaFx;
using Opc.UaFx.Client;

0 referências
class Program
{
    0 referências
    static void Main(string[] args)
    {
        // URL do servidor OPC UA
        string serverUrl = "opc.tcp://DESKTOP-D46STS2:4840";

        // Conecta ao servidor OPC UA
        using (var client = new OpcClient(serverUrl))
        {
            client.Connect();
            Console.WriteLine("Conectado ao servidor OPC UA!");

            // Inicia a partir do nó raiz (RootFolder)
            var rootNodeId = OpcObjectTypes.ObjectsFolder;

            // Chama o método para navegar pelo servidor e imprimir os NodeIds
            BrowseNodes(client, rootNodeId);

            client.Disconnect();
            Console.WriteLine("Desconectado do servidor OPC UA.");
        }

        using (var client = new OpcClient(serverUrl))
        {
            // Conecta ao servidor
            client.Connect();
            Console.WriteLine("Conectado ao servidor OPC UA!");

            // Define o nó que deseja ler
            var nodeIdToRead = "ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.START";

            // Lê o valor da variável
            OpcValue value = client.ReadNode(nodeIdToRead);
            Console.WriteLine($"Valor lido: {value}");

            // Define o novo valor da variável
            var newValue = false;

            // Escreve o novo valor na variável
            client.WriteNode(nodeIdToRead, newValue);
            Console.WriteLine($"Valor escrito: {newValue}");

            // Desconecta do servidor
            client.Disconnect();
            Console.WriteLine("Desconectado do servidor OPC UA.");
        }
    }
```

Fonte: (Imagem do Autor)

Figura 22 - Código Teste Comunicação OPC 2



```
2 referências
static void BrowseNodes(OpcClient client, OpcNodeId nodeId, string indent = "")
{
    try
    {
        // Obtém os nós filhos
        var nodeInfo = client.BrowseNode(nodeId);
        var children = nodeInfo.Children(); // Use Children() para obter os nós filhos

        foreach (var childNode in children)
        {
            // Imprime o NodeId e o nome do nó
            Console.WriteLine($"{indent}NodeId: {childNode.NodeId}, DisplayName: {childNode.DisplayName}");

            // Chama recursivamente para explorar subnós
            BrowseNodes(client, childNode.NodeId, indent + "  ");
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Erro ao navegar no nó {nodeId}: {ex.Message}");
    }
}
```

Fonte: (Imagem do Autor)

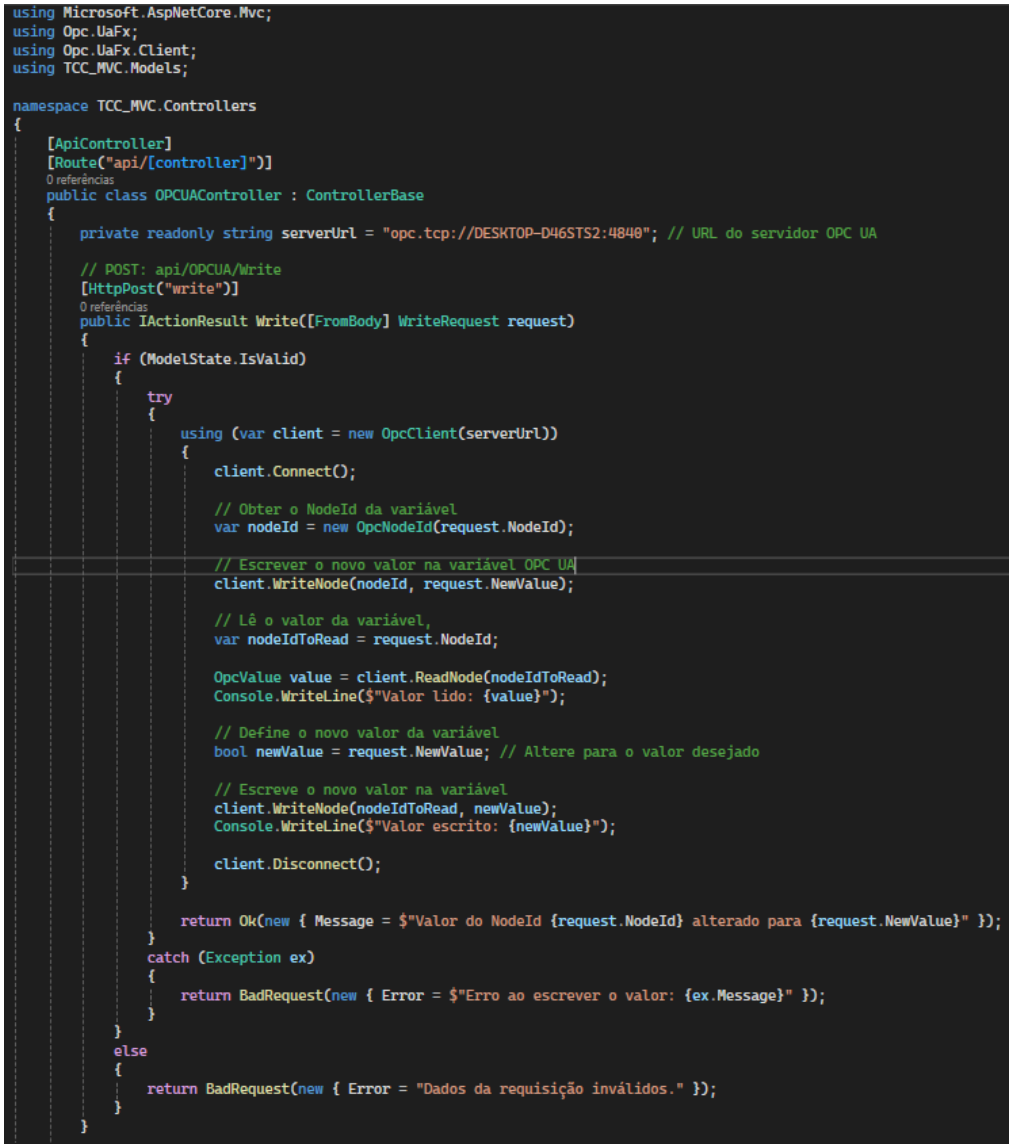
Após confirmar a viabilidade da comunicação com o servidor OPC, deu-se início ao desenvolvimento da API REST. O primeiro passo foi a criação do modelo de dados, que serviria como base para a estrutura da API. Esse modelo define as variáveis e parâmetros que serão manipulados pela controladora principal, permitindo a interação direta com a lógica de controle da planta. A partir desse modelo, a controladora foi construída para gerenciar as requisições HTTP, facilitando a comunicação entre o *front-end* e o servidor OPC de forma eficiente e segura. O código pode ser visto na Figura 23.

Figura 23 – Modelo para Controladora

Fonte: (Imagem do Autor)

Em seguida, foi desenvolvida a controladora, responsável por gerenciar as requisições GET e POST da API. A função GET (Figura 25) foi implementada para consultar os estados atuais das variáveis do servidor OPC, permitindo o monitoramento em tempo real. Já o método POST (Figura 24) foi projetado para enviar comandos e atualizações para o servidor, alterando o estado das variáveis conforme necessário.

Figura 24 – Controladora e Método Post da Rest Api



```
using Microsoft.AspNetCore.Mvc;
using Opc.UaFx;
using Opc.UaFx.Client;
using TCC_MVC.Models;

namespace TCC_MVC.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    0 referências
    public class OPCUAController : ControllerBase
    {
        private readonly string serverUrl = "opc.tcp://DESKTOP-D46STS2:4840"; // URL do servidor OPC UA

        // POST: api/OPCUA/Write
        [HttpPost("write")]
        0 referências
        public IActionResult Write([FromBody] WriteRequest request)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    using (var client = new OpcClient(serverUrl))
                    {
                        client.Connect();

                        // Obter o NodeId da variável
                        var nodeId = new OpcNodeId(request.NodeId);

                        // Escrever o novo valor na variável OPC UA
                        client.WriteNode(nodeId, request.NewValue);

                        // Lê o valor da variável,
                        var nodeIdToRead = request.NodeId;

                        OpcValue value = client.ReadNode(nodeIdToRead);
                        Console.WriteLine($"Valor lido: {value}");

                        // Define o novo valor da variável
                        bool newValue = request.NewValue; // Altere para o valor desejado

                        // Escreve o novo valor na variável
                        client.WriteNode(nodeIdToRead, newValue);
                        Console.WriteLine($"Valor escrito: {newValue}");

                        client.Disconnect();
                    }

                    return Ok(new { Message = $"Valor do NodeId {request.NodeId} alterado para {request.NewValue}" });
                }
                catch (Exception ex)
                {
                    return BadRequest(new { Error = $"Erro ao escrever o valor: {ex.Message}" });
                }
            }
            else
            {
                return BadRequest(new { Error = "Dados da requisição inválidos." });
            }
        }
```

Fonte: (Imagem do Autor)

Figura 25 – Método Get da Rest Api

```
[HttpGet("readvalue")]
0 referências
public IActionResult ReadValue([FromQuery] string nodeId)
{
    if (string.IsNullOrEmpty(nodeId))
    {
        return BadRequest("NodeId não pode ser nulo ou vazio.");
    }

    try
    {
        using (var client = new OpcClient(serverUrl))
        {
            client.Connect();

            OpcValue opcValue = client.ReadNode(nodeId);
            Console.WriteLine($"Valor lido no {nodeId}: {opcValue}");

            client.Disconnect();

            return Ok(opcValue.Value);
        }
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Erro ao ler o valor: {ex.Message}");
    }
}
```

Fonte: (Imagem do Autor)

3.4. Desenvolvimento das telas seguindo a ISA-101

A fase final do desenvolvimento foi dedicada à criação do *front-end*, começando pela definição do CSS, responsável pela estilização e aparência da página. O CSS e o Bootstrap foram utilizados para garantir um layout responsivo e visualmente agradável, ajustando elementos como cores, fontes, espaçamento e organização dos componentes na interface. Além disso, eles asseguraram que o *design* fosse consistente em diferentes dispositivos e tamanhos de tela. Tendo o código do CSS na Figura 26.

Figura 26 - CSS

Fonte: (Imagem do Autor)

Em seguida, foi desenvolvido o HTML, responsável pela estruturação das páginas do sistema. A página foi organizada em três partes principais. A primeira é a barra de navegação (Figura 27), que fornece acesso rápido às diferentes seções do sistema e facilita a navegação do usuário. A segunda parte é composta pelos *containers* que exibem os equipamentos da planta (Figura 28), permitindo o monitoramento visual dos componentes em tempo real. Por fim, a terceira parte inclui os botões interativos (Figura 29), que permitem ao usuário modificar e visualizar o estado das variáveis controladas, oferecendo uma interface direta para o gerenciamento da planta.

Figura 27 – Barra de Navegação HTML

```html
<nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
        <!-- Container de flex para garantir que cada item ocupe um quarto do espaço disponível -->
        <div class="d-flex w-100 align-items-center">
            <!-- Parte 1: Cefet -->
            <div class="d-lg-block flex-fill text-center d-flex align-items-center justify-content-center">
                <a class="navbar-brand d-block" href="#">
                    <img src="assets/768px-Logo_CEFET-MG.png" alt="Cefet" style="max-height: 80px;">
                </a>
            </div>

            <!-- Parte 2: Nome do Projeto -->
            <div class="d-none d-lg-flex flex-fill text-center align-items-center justify-content-center mt-2">
                <a class="navbar-brand" href="index.html">Sistema Supervisório</a>
            </div>

            <!-- Parte 3: Emergência -->
            <div class="d-lg-flex flex-fill text-center align-items-center justify-content-center mt-2">
                <a class="navbar-brand" id="emergencia-text" href="emergencia.html">Emergência</a>
                <img id="emergencia-icon" src="assets/exclamation-diamond.svg"
                alt="Emergência" style="width: 30px; height: auto;">
            </div>

            <!-- Parte 4: Dropdown para Telas Pequenas e Grandes -->
            <div class="col-1 d-flex align-items-center justify-content-center">
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav"
                    aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
            </div>
        </div>

        <!-- Menu de navegação (visível em telas pequenas) -->
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav ml-auto d-lg-none">
                <li class="nav-item">
                    <a class="nav-link" href="index.html">Principal</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="garra.html">Garra</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="esteiras.html">Esteiras</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="pushers.html">Pushers</a>
                </li>
                <li class="nav-item">
                    <a class="nav-link" href="sensores.html">Sensores</a>
                </li>
            </ul>
        </div>

        <!-- Dropdown para telas grandes -->
        <div class="d-none d-lg-block">
            <div class="dropdown">
                <button class="btn btn-secondary dropdown-toggle" type="button" id="dropdownMenuButton"
                    data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
                    Telas
                </button>
                <div class="dropdown-menu" aria-labelledby="dropdownMenuButton">
                    <a class="dropdown-item" href="index.html">Principal</a>
                    <a class="dropdown-item" href="garra.html">Garra</a>
                    <a class="dropdown-item" href="esteiras.html">Esteiras</a>
                    <a class="dropdown-item" href="pushers.html">Pushers</a>
                    <a class="dropdown-item" href="sensores.html">Sensores</a>
                </div>
            </div>
        </div>

    </div>
</nav>
```

Fonte: (Imagem do Autor)

Figura 28 – *Containers* HTML

```
<div class="row justify-content-center">
    <!-- Container 1 -->
    <div class="col-12 col-md-6 col-lg-3 d-flex flex-column my-3">
        <div class="flex-grow-1">
            <div class="h-100 border">Gerador de peças</div>
        </div>
        <div class="d-flex justify-content-center mt-auto">
            <button class="btn btn-primary mx-2 opc-button btn-white" data-pair="2"
                data-nodeid="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.EMISSOR"
                data-nodeidlogic="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.ON_EMISSOR"
                data-newvalue="true" data-type="primary">On</button>
            <button class="btn btn-secondary mx-2 opc-button btn-white" data-pair="2"
                data-nodeid="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.EMISSOR"
                data-nodeidlogic="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.OFF_EMISSOR"
                data-newvalue="true" data-type="secondary">Off</button>
        </div>
    </div>

    <!-- Container 2 -->
    <div class="col-12 col-md-6 col-lg-3 d-flex flex-column my-3">
        <div class="flex-grow-1">
            <div class="h-100 border">Esteira Pequena</div>
        </div>
        <div class="d-flex justify-content-center mt-auto">
            <button class="btn btn-primary mx-2 opc-button btn-white" data-pair="3"
                data-nodeid="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.ESTEIRA_P"
                data-nodeidlogic="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.ON_ESTEIRA_P"
                data-newvalue="true" data-type="primary">On</button>
            <button class="btn btn-secondary mx-2 opc-button btn-white" data-pair="3"
                data-nodeid="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.ESTEIRA_P"
                data-nodeidlogic="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.OFF_ESTEIRA_P"
                data-newvalue="true" data-type="secondary">Off</button>
        </div>
    </div>
```

Fonte: (Imagem do Autor)

Figura 29 – Botões HTML

```
<div class="d-flex justify-content-center mt-auto">
    <button class="btn btn-primary mx-2 opc-button btn-white" data-pair="2"
        data-nodeid="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.EMISSOR"
        data-nodeidlogic="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.ON_EMISSOR"
        data-newvalue="true" data-type="primary">On</button>
    <button class="btn btn-secondary mx-2 opc-button btn-white" data-pair="2"
        data-nodeid="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.EMISSOR"
        data-nodeidlogic="ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.OFF_EMISSOR"
        data-newvalue="true" data-type="secondary">Off</button>
</div>
```

Fonte: (Imagem do Autor)

Por fim, foi desenvolvido o JavaScript do *front-end*, dividido em duas partes principais. O primeiro *script* é responsável por comunicar com a API REST criada anteriormente, atualizando o estado das variáveis exibidas na tela e monitorando

continuamente o estado de emergência, independentemente da tela em uso, garantindo que quaisquer situações críticas sejam detectadas. A Figura 30 e a Figura 31, mostram esse código. O segundo *script* lida com a alteração dos valores das variáveis quando solicitado pelo usuário; sendo visto na Figura 32; passando essa alteração para a Rest Api que repassa para o servidor OPC.

Figura 30 – Botões e Emergência JavaScript 1

```javascript
// URL da API
const apiUrlBase = 'http://localhost:5195/api/OPCUA/';

// Função para verificar o valor do NodeId para a emergência
async function verificarNodeIdEmergencia() {
    const nodeIdEmergencia = "ns=4;s=|var|CODESYS Control Win V3 x64.Application.PLC_PRG.EMERGENCIA";
    const apiUrl = `${apiUrlBase}readvalue?nodeId=${encodeURIComponent(nodeIdEmergencia)}`;

    try {
        const response = await fetch(apiUrl);
        if (!response.ok) {
            throw new Error(`Erro na requisição: ${response.statusText}`);
        }

        const data = await response.json();
        const emergenciaElement = document.querySelector('.navbar-brand');
        const emergenciaIcon = document.querySelector('.navbar-brand + img');

        if (data === true) {
            // Adiciona a classe que altera a cor para vermelho
            emergenciaElement.classList.add('emergencia-ativa');
            // Altera o ícone para o ícone desejado
            emergenciaIcon.src = 'assets/exclamation-diamond-fill.svg';
        } else {
            // Remove a classe para voltar ao estilo padrão
            emergenciaElement.classList.remove('emergencia-ativa');
            emergenciaIcon.src = 'assets/exclamation-diamond.svg';
        }
    } catch (error) {
        console.error('Erro ao verificar o NodeId de emergência:', error);
    }
}

// Função para atualizar a cor de um par de botões
async function updateButtonPair(pairButtons) {
    const nodeId = pairButtons[0].getAttribute("data-nodeid");
    const apiUrl = `${apiUrlBase}readvalue?nodeId=${encodeURIComponent(nodeId)}`;

    try {
        const response = await fetch(apiUrl);
        if (!response.ok) {
            throw new Error(`Erro na requisição: ${response.statusText}`);
        }

        const value = await response.json();

        pairButtons.forEach(btn => {
            const btnType = btn.getAttribute("data-type");

            if (value) {
                if (btnType === "primary") {
                    btn.classList.remove("btn-secondary", "btn-white");
                    btn.classList.add("btn-primary");
                } else if (btnType === "secondary") {
                    btn.classList.remove("btn-primary", "btn-white");
                    btn.classList.add("btn-secondary");
                }
            } else {
                if (btnType === "primary") {
                    btn.classList.remove("btn-primary", "btn-white");
                    btn.classList.add("btn-secondary");
                } else if (btnType === "secondary") {
                    btn.classList.remove("btn-secondary", "btn-white");
                    btn.classList.add("btn-primary");
                }
            }
        });
    } catch (error) {
        console.error("Erro ao atualizar o par de botões:", error);
    }
}
```

Fonte: (Imagem do Autor)

Figura 31 - Botões e Emergência JavaScript 2

```
// Função para atualizar todas as cores dos botões
async function updateButtonColors() {
    const buttonPairs = {}; // Objeto para armazenar pares de botões

    document.querySelectorAll(".opc-button").forEach(button => {
        const pair = button.getAttribute("data-pair");
        if (!buttonPairs[pair]) {
            buttonPairs[pair] = [];
        }
        buttonPairs[pair].push(button);
    });

    // Atualize todos os pares de botões
    for (const pair in buttonPairs) {
        await updateButtonPair(buttonPairs[pair]);
    }

    // Verifica o estado da emergência
    await verificarNodeIdEmergencia();

    // Aguarde 1 segundo antes de iniciar o próximo ciclo
    setTimeout(updateButtonColors, 1000);
}

// Inicie o ciclo de atualização
document.addEventListener('DOMContentLoaded', function() {
    updateButtonColors();
});
```

Fonte: (Imagem do Autor)

Figura 32 – Requisição JavaScript

```javascript
document.addEventListener("DOMContentLoaded", function() {
    const buttons = document.querySelectorAll(".opc-button");
    const result = document.getElementById("result");

    async function sendValue(nodeId, newValue) {
        const apiUrl = "http://localhost:5195/api/OPCUA/write";
        const payload = {
            NodeId: nodeId,
            NewValue: newValue
        };

        try {
            const response = await fetch(apiUrl, {
                method: "POST",
                headers: {
                    "Content-Type": "application/json"
                },
                body: JSON.stringify(payload)
            });

            if (!response.ok) {
                throw new Error(`Erro na requisição: ${response.statusText}`);
            }

            const data = await response.json();
            result.textContent = "Sucesso: " + JSON.stringify(data);
        } catch (error) {
            result.textContent = "Erro: " + error.message;
        }
    }

    buttons.forEach(button => {
        button.addEventListener("click", async function() {
            const nodeId = button.getAttribute("data-nodeidlogic");
            const newValue = button.getAttribute("data-newvalue") === "true";

            if (!nodeId) {
                result.textContent = "Erro: NodeId é nulo ou indefinido.";
                return;
            }

            if (typeof newValue !== "boolean") {
                result.textContent = "Erro: NewValue não é um booleano válido.";
                return;
            }

            // Envia o valor true
            await sendValue(nodeId, true);

            // Espera 2 segundo e então envia o valor false
            setTimeout(async () => {
                await sendValue(nodeId, false);
            }, 2000);
        });
    });
});
```

Fonte: (Imagem do Autor)

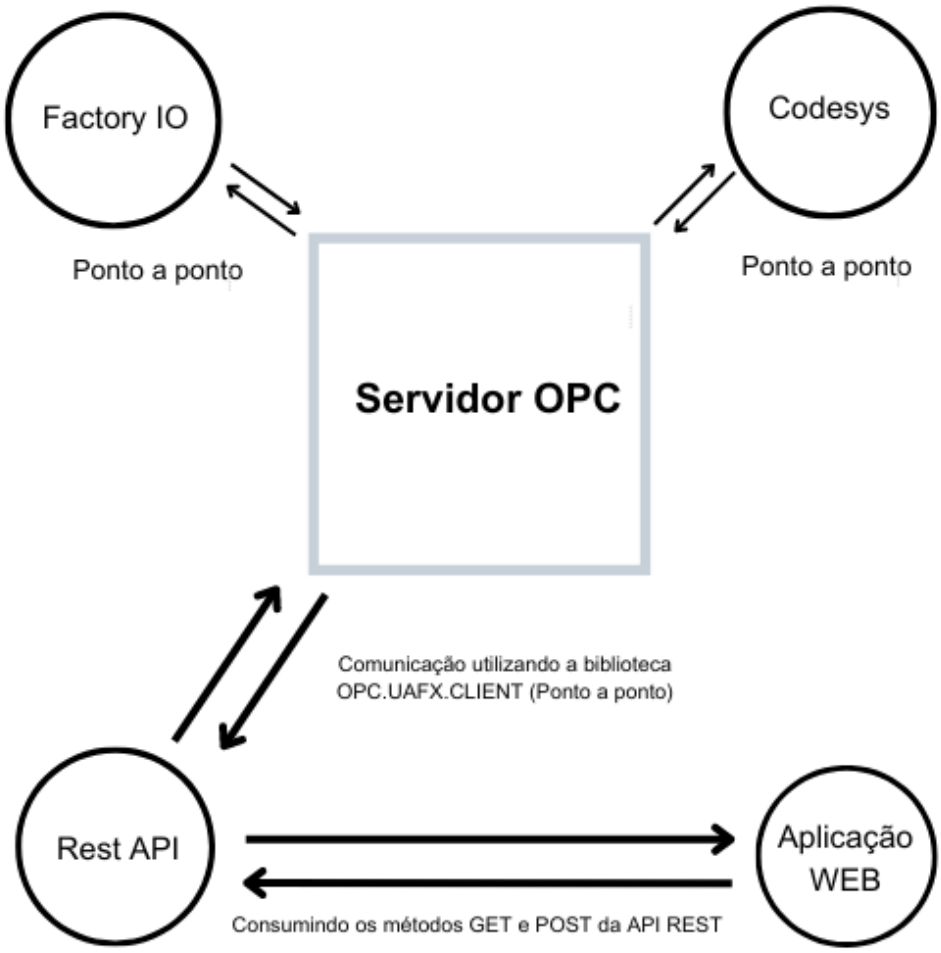
4. Resultados

A adaptação da planta utilizando o Factory I/O foi bem-sucedida, assim como a programação em *ladder* no Codesys e a criação e configuração do servidor OPC. A implementação da API REST com o .NET também foi concluída com êxito. As telas, desenvolvidas com CSS, HTML, JavaScript e, principalmente, Bootstrap, estão operacionais e responsivas, adaptando-se eficientemente a diferentes dispositivos com acesso à *web*.

4.1. Diagrama de Conexão do Sistema

A comunicação no sistema está estruturada com o servidor OPC como o núcleo central, facilitando a interação entre os diferentes componentes. O Factory I/O, o Codesys e a API REST estabelecem comunicação direta com o servidor OPC, garantindo que todos os dados e comandos sejam centralizados e gerenciados eficientemente. A aplicação *web*, por sua vez, interage exclusivamente com a API REST, que atua como intermediária entre a interface do usuário e o servidor OPC. Esse modelo garante que a comunicação entre a aplicação *web* e o servidor OPC seja bem gerenciada e simplificada, permitindo que a API REST forneça uma interface consistente e segura para a manipulação dos dados e comandos no servidor OPC. Essa relação de comunicação está ilustrada na Figura 33.

Figura 33 – Diagrama de Conexões do Sistema

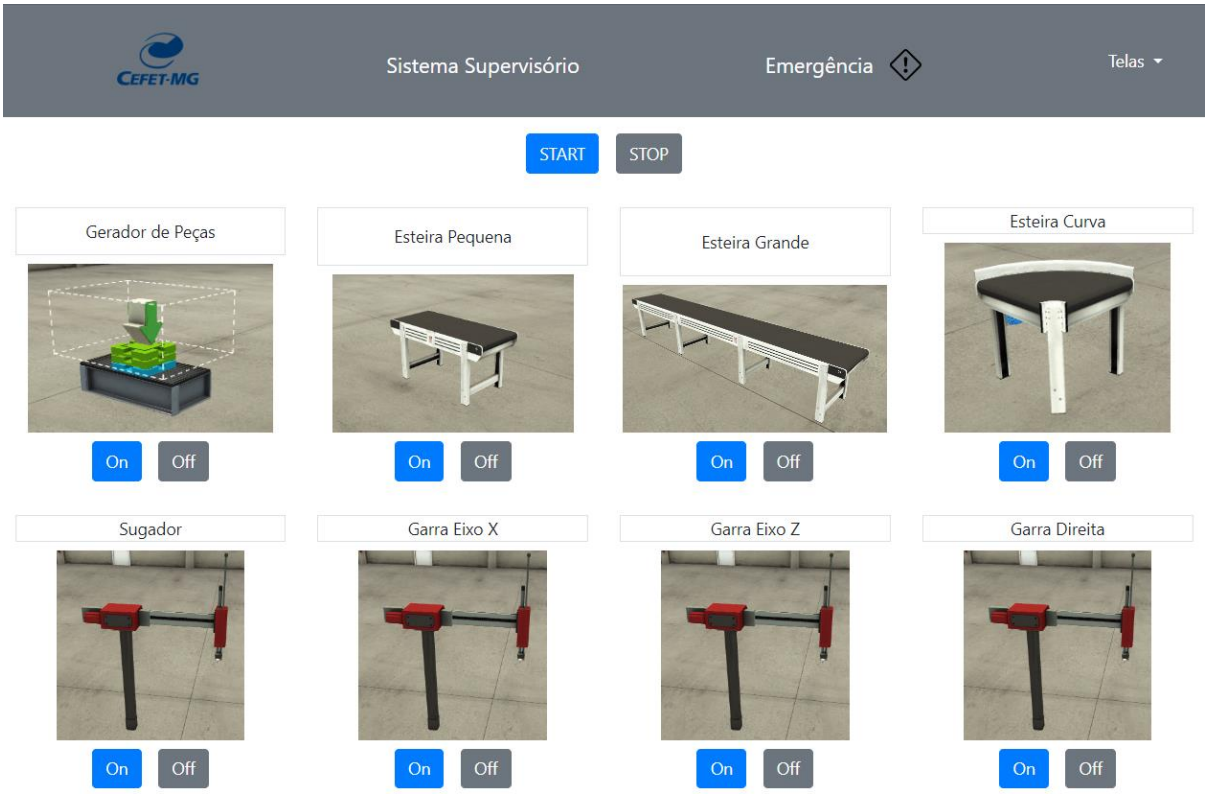


Fonte: (Imagem do Autor)

4.2. Telas do Sistema Seguindo a ISA-101

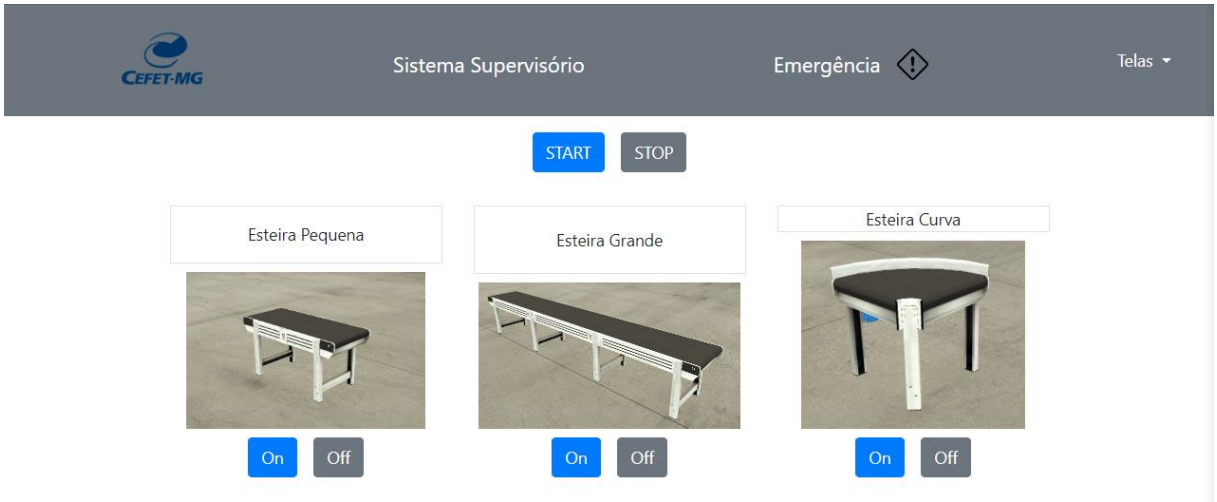
A tela principal, exibida na Figura 34, apresenta os principais elementos do sistema, incluindo o gerador de peças, as esteiras e as posições da garra. As telas secundárias, exemplificadas na Figura 35, fornecem informações detalhadas sobre partes específicas do projeto, como a seção das esteiras da planta. Apesar das diferenças entre as telas, há atributos comuns a todas elas: a barra de navegação, que facilita a transição entre diferentes seções do sistema e indica se uma emergência está em curso, e os botões de *start* e *stop*, que controlam o processo geral da planta. Esses elementos são projetados para garantir uma navegação intuitiva e um controle eficaz do sistema.

Figura 34 – Tela Principal



Fonte: (Imagem do autor)

Figura 35 – Tela Secundária

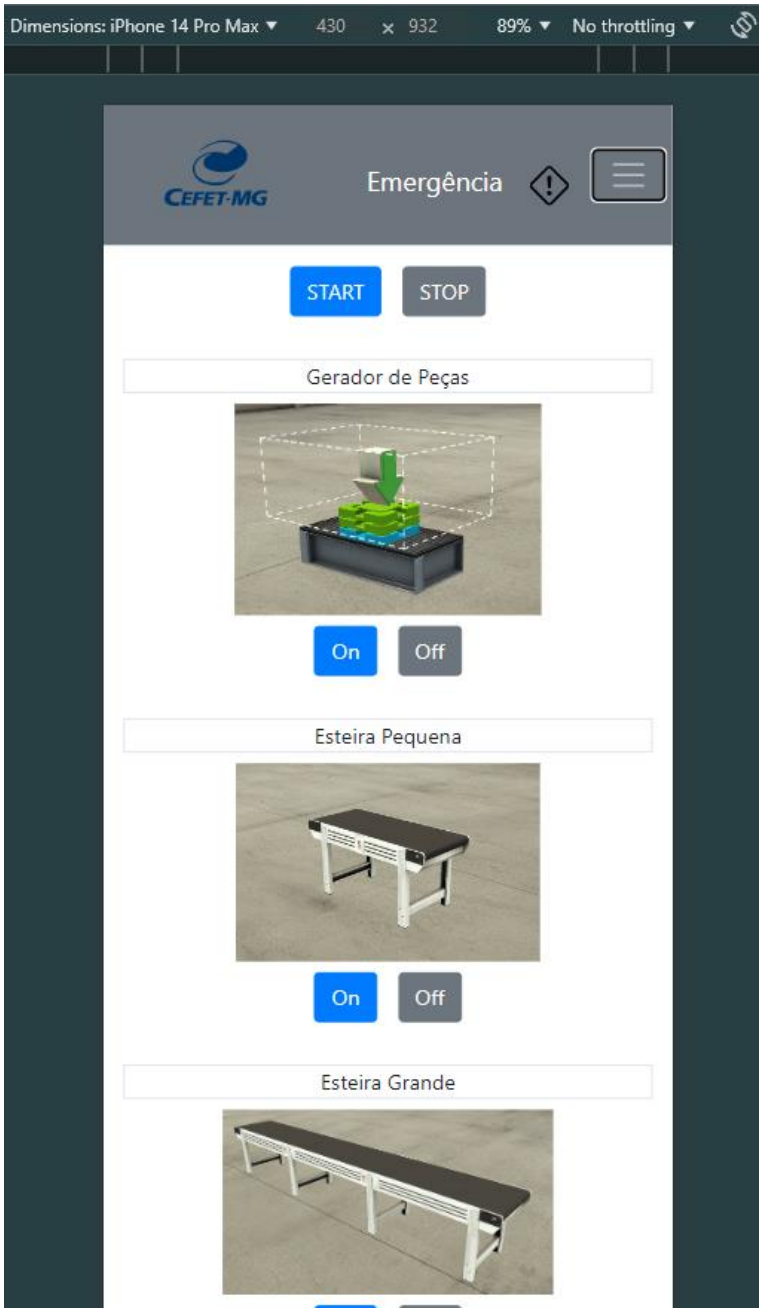


Fonte: (Imagem do autor)

Além disso, é importante destacar que a responsividade das telas está funcionando de maneira eficiente. Elas ajustam adequadamente os *containers* do projeto e garantem o acesso fluido a diferentes seções do sistema por meio da barra de navegação, independentemente do dispositivo ou tamanho da tela. Como pode ser

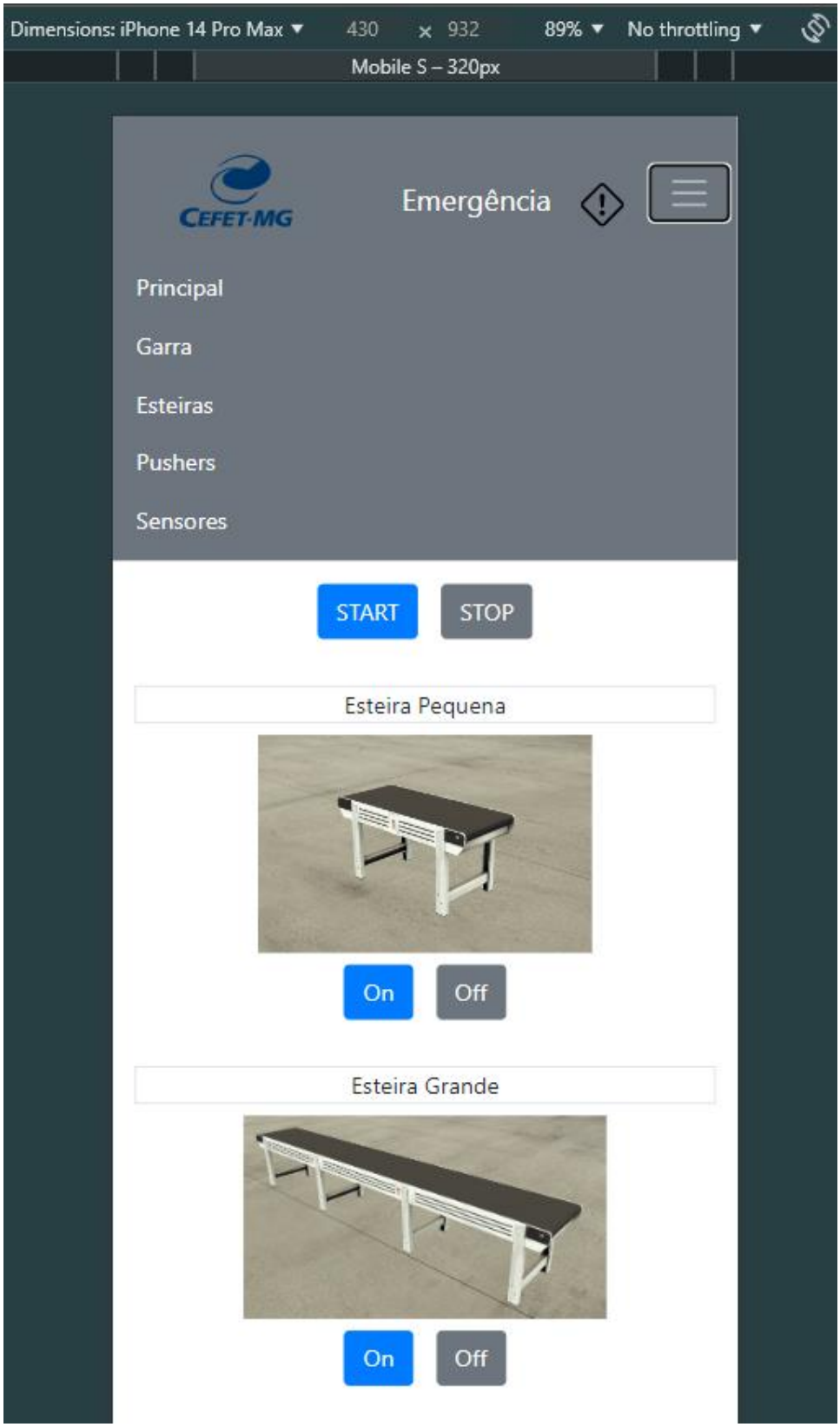
visto na Figura 36 e na Figura 37, onde ambas estão dimensionadas para o tamanho de um Iphone 14 Pro Max.

Figura 36 – Tela Principal *Mobile* 1



Fonte: (Imagem do autor)

Figura 37 – Tela Principal *Mobile* 2



Fonte: (Imagem do autor)

4.3. Funcionamento e Indicações dos Botões

Os botões "On" e "Off" de cada componente do sistema desempenham duas funções principais. Primeiramente, eles enviam para a API REST o valor que a variável correspondente deve assumir, que pode ser *true* (para ligado) ou *false* (para desligado). Em segundo lugar, esses botões fornecem uma indicação visual do estado atual do componente. Quando o botão está na posição "*On*", ele altera a cor para azul, sinalizando que o componente está ativo e funcionando corretamente. Por outro lado, quando o botão está na posição "*Off*", sua cor muda para cinza, indicando que o componente está inativo. Essa representação visual facilita a monitorização do estado dos elementos do sistema de forma rápida e intuitiva.

Um exemplo desse funcionamento pode ser visualizado na Figura 38. Nela, o gerador de peças está ativo, conforme indicado pelo botão "*On*", que está destacado em azul, enquanto o botão "*Off*" está desativado e aparece em cinza. Em contraste, a esteira pequena mostra o estado oposto; o botão "*Off*" foi selecionado, o que faz com que ele apareça em azul, enquanto o botão "*On*" está desativado e aparece em cinza. Esta representação visual ajuda a distinguir rapidamente quais componentes estão em operação e quais estão desligados, facilitando o monitoramento e controle do sistema.

Figura 38 – Exemplo de Funcionamento dos Botões

Fonte: (Imagem do Autor)

4.4. Funcionamento da Emergência

A função de simulação de emergência opera da seguinte maneira: quando ativada, ela desativa todo o sistema, impedindo que os outros elementos funcionem até que a emergência seja desativada. Além disso, um indicativo visual no topo da tela mostra um símbolo cuja cor é invertida para sinalizar a ocorrência de um problema.

A Figura 39 ilustra esse funcionamento. Na imagem, a emergência está em curso e, mesmo que o botão "Start" seja pressionado para retomar a operação, o sistema não reativa até que a emergência seja resolvida. O símbolo de emergência, localizado ao lado da palavra "Emergência" na barra de navegação, só retorna ao seu estado normal quando a emergência é finalizada. Essa abordagem assegura que a condição crítica seja resolvida antes de qualquer retorno à operação normal do sistema.

Figura 39 – Exemplo do Funcionamento da Emergência

Fonte: (Imagem do Autor)

4.5. Requisitos de Desempenho na Aplicação e Desenvolvimento

Um ponto crucial a ser analisado é a estabilidade do Codesys, que apresentou piora nas versões mais recentes, impactando negativamente o desenvolvimento e a execução do projeto. Além disso, é fundamental destacar que a eficiência na comunicação entre sistemas é vital. A máquina que executa a API REST deve possuir uma capacidade operacional robusta da CPU (processador). Da mesma forma, o dispositivo que executa a aplicação *web* precisa de uma conexão com a internet estável e rápida para garantir um desempenho otimizado e uma experiência de usuário satisfatória.

5. Considerações Finais

Concluindo, a implementação de um sistema supervisório *web* provou ser eficaz, com ampla aplicabilidade em diferentes processos produtivos. Esse tipo de sistema é especialmente interessante quando há necessidade de supervisionar as operações remotamente, permitindo que o engenheiro responsável, ou qualquer outro profissional autorizado, monitore o funcionamento da planta em tempo real, independentemente de sua localização.

Além disso, a versatilidade do sistema supervisório *web* foi um ponto de destaque, sobretudo quando combinado com uma API REST desenvolvida em .NET. A robustez dessa plataforma, que oferece uma biblioteca completa para lidar com o protocolo de comunicação OPC, facilita a integração e garante uma comunicação eficiente entre os componentes da planta. Essa solução tecnológica permite não apenas o monitoramento, mas também o controle remoto, ampliando significativamente as possibilidades de automação e gestão industrial, tornando o processo produtivo mais ágil, seguro e flexível.

Para aprimorar este trabalho, sugerem-se as seguintes direções para trabalhos futuros:

- Estabelecer uma relação entre usuários e permissões de acesso a funções específicas do sistema;
- Otimizar a eficiência dos métodos HTTP da API REST, assegurando respostas mais rápidas e eficazes;
- Aplicar o mesmo princípio de otimização aos scripts JavaScript no *front-end*, melhorando a performance da interface;
- Replicar o trabalho em uma planta física real, a fim de identificar as adaptações necessárias para uma implementação prática em um ambiente mais próximo do industrial.

Referências

ACKERMAN', William J e BLOCK, Wayne R. **Understanding Supervisory Systems**. . [S.l: s.n.], 1992.

DÉRAMOND, Julien. **Get started with Bootstrap**. Disponível em: <https://getbootstrap.com/docs/4.1/getting-started/introduction/>. Acesso em: 24 nov 2023.

DERHAMY, Hasan e colab. **Protocol interoperability of OPC UA in Service Oriented Architectures**. . [S.l: s.n.], 2017. Disponível em: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8104744>. Acesso em: 30 out 2023.

FAGUNDES, Frederico Duarte. **Apostila de Sistemas Supervisórios e Interfaces Homem-Máquina**. . [S.l: s.n.], 2023.

GONZÁLEZ, Isaías e colab. **A literature survey on open platform communications (OPC) applied to advanced industrial environments**. Electronics (Switzerland), v. 8, n. 5, 1 Maio 2019.

GREGORY DURANSO. **High-Performance HMI *Design* Basics**. Disponível em: <https://realpars.com/hmi-*design*>. Acesso em: 8 nov 2023.

HILLMAN, Mônica Mazzochi. **HTML: o que é, a importância para a *Web*** . Disponível em: <https://www.alura.com.br/artigos/html?_gl=1*m0226e*_ga*MTU2MTEzNDY5NS4xNjk5Mzk3MzQz*_ga_1EPWSW3PCS*MTY5OTM5NzM0My4xLjEuMTY5OTM5NzUxNC4wLjAuMA..*_fplc*ZHdMSzZNbndSQldlMUwlMkJzb1dBcWFEYnlQMmt5NUFiYlFTakpvNFQ1TFR1eENhUyUyQktsczNJTzJvVUt0WlBZOWt4aUxQbiUyQmpSY0Q1WVFmME85R2dXeVhnMjM2UWkyTU1vTnNkc1lWbEZkSW96Q1c2YXE0VXpSZWtFcmNXUEhRJTNEJTNE>. Acesso em: 18 nov 2023.

INTERNATIONAL SOCIETY OF AUTOMATION. **American National Standard, ANSI/ISA-101.01-2015: Human machine interfaces for process automation**

systems, approved 9 July 2015. [S.l: s.n.], 2015. Disponível em: <https://www.isa.org/products/ansi-isa-101-01-2015-human-machine-interfaces-for>. Acesso em: 23 set 2023.

LINKEDIN CONTRIBUTORS. **Brasil Empregos em alta**. Disponível em: <https://business.linkedin.com/pt-br/talent-solutions/resources/talent-acquisition/jobs-on-the-rise-cont-fact>. Acesso em: 5 set 2024.

MDN CONTRIBUTORS. **CSS Documentation**. Disponível em: <https://devdocs.io/css/>. Acesso em: 18 nov 2023a.

MDN CONTRIBUTORS. **HTML: Linguagem de Marcação de Hipertexto**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/*Web*/HTML>. Acesso em: 18 nov 2023b.

MDN CONTRIBUTORS. **JavaScript**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/*Web*/JavaScript>. Acesso em: 24 nov 2023.

MICROSOFT. **Desempenho onde é importante** . Disponível em: <https://dotnet.microsoft.com/pt-br/platform/why-choose-dotnet>. Acesso em: 10 nov 2023a.

MICROSOFT. **O que é ASP.NET ?** Disponível em: <https://dotnet.microsoft.com/pt-br/learn/aspnet/what-is-aspnet>. Acesso em: 10 nov 2023b.

MICROSOFT. **O que é o .NET?** Disponível em: <https://dotnet.microsoft.com/pt-br/learn/dotnet/what-is-dotnet>. Acesso em: 8 nov 2023c.

MICROSOFT. **Por que escolher o .NET?** Disponível em: <https://dotnet.microsoft.com/pt-br/platform/why-choose-dotnet>. Acesso em: 10 nov 2023d.

MICROSOFT. **Um tour pela linguagem C#**. Disponível em: <https://learn.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/>. Acesso em: 8 nov 2023e.

NEVES, Vinícios. **CSS: o que é, como usar no HTML**. Disponível em: <https://www.alura.com.br/artigos/css?_gl=1*sjfv9i*_ga*MTU2MTEzNDY5NS4xNjk5Mzk3MzQz*_ga_1EPWSW3PCS*MTY5OTM5NzM0My4xLjEuMTY5OTM5NzUyMC4wLjAuMA..*_fplc*ZHdMSzZNbndSQldlMUwlMkJzb1dBcWFEYnlQMmt5NUFiYlFTakpvNFQ1TFR1eENhUyUyQktsczNJTzJvVUt0WlBZOWt4aUxQbiUyQmpSY0Q1WVFmME85R2dXeVhnMjM2UWkyTU1vTnNkc1lWbEZkSW96Q1c2YXE0VXpSZWtFcmNXUEhRJTNEJTNE>. Acesso em: 18 nov 2023.

NVTEC. **Componentes de um sistema SCADA explicados**. Disponível em: <https://www.nvtecnologias.com/blog/blog-1/componentes-de-un-sistema-scada-explicados-26>. Acesso em: 8 nov 2023.

POGACEAN, Cristian e BROSCHEI, Sebatian e SUSS, Georg. **Implementando Comunicação OPC UA Determinísca Sumário**. . [S.l: s.n.], 2016. Disponível em: <www.wii.com.br>.

RICH LANDER. **Introducing .NET 5**. Disponível em: <https://devblogs.microsoft.com/dotnet/introducing-net-5/>. Acesso em: 10 nov 2023.

SAJID, Anam e ABBAS, Haider e SALEEM, Kashif. **Cloud-Assisted IoT-Based SCADA Systems Security: A Review of the State of the Art and Future Challenges**. IEEE Access. [S.l.]: Institute of Electrical and Electronics Engineers Inc. , 2016

SIEMENS BRASIL. **OPC UA acelera a digitalização da sua empresa**. Disponível em: <https://www.siemens.com/br/pt/produtos/automacao/comunicacao-industrial/opc-ua.html>. Acesso em: 30 out 2023.

TECHEMPOWER. ***Web Framework* Benchmarks**. Disponível em: <https://www.techempower.com/benchmarks/#hw=ph&test=plaintext&section=data-r21>. Acesso em: 10 nov 2023.

CÓPIA DO TRABALHO Nº 149/2024 - DELMAX (11.57.05)

(Nº do Protocolo: NÃO PROTOCOLADO)

(Assinado digitalmente em 18/09/2024 11:45)
MATEUS ANTUNES OLIVEIRA LEITE
PROFESSOR ENS BASICO TECN TECNOLOGICO
CAAAX (11.57.01)
Matrícula: ###384#0

Visualize o documento original em https://sig.cefetmg.br/documentos/ informando seu número: ***149****, ano:* ***2024****, tipo:* ***CÓPIA DO TRABALHO****, data de emissão:* ***18/09/2024*** *e o código de verificação:* ***3c6b29fc1e***