UNIVERSIDADE ESTADUAL PAULISTA - UNESP

Faculdade de Engenharia- campus de Ilha Solteira

THIAGO GARCIA PRADO MARTINS DE OLIVEIRA

Desenvolvimento de metodologia para automação de aplicações web usando proxy e requisições HTTP

Ilha Solteira

2025

GRADUAÇÃO EM ENGENHARIA ELÉTRICA

THIAGO GARCIA PRADO MARTINS DE OLIVEIRA

Desenvolvimento de metodologia para automação de aplicações web usando proxy e requisições HTTP

Trabalho de graduação, apresentado à Universidade Estadual Paulista (UNESP), Faculdade de Engenharia, Ilha Solteira, para obtenção do título de grau acadêmico Bacharel em Engenharia Elétrica.

Área de Concentração: Engenharia Elétrica

Orientador: Profº Dr Leandro Oliveira Salviano

Ilha Solteira
2025

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

O48d

Oliveira, Thiago Garcia Prado Martins de.
Desenvolvimento de metodologia para automação de aplicações web usando proxy e requisições HTTP / Thiago Garcia Prado Martins de Oliveira. -- Ilha Solteira: [s.n.], 2024
49 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista (UNESP), Faculdade de Engenharia, Ilha Solteira, 2024

Orientador: Leandro Oliveira Salviano

Inclui bibliografia

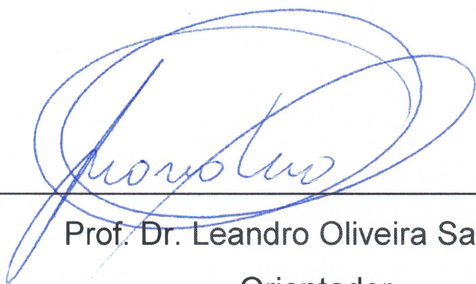
1. Engenharia de simulação. 2. Interceptação de tráfego. 3. Integração de sistemas. 4. Execução programada.

Sandra Montibeller - CRB-8/060

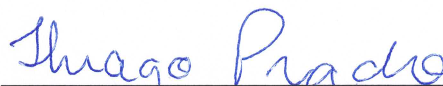
unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

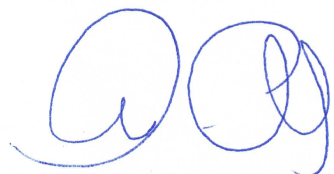
Aos doze dias do mês de junho do ano de dois mil e vinte e cinco, o discente ***Thiago Garcia Prado Martins de Oliveira***, matriculado sob o nº **191053041**, tendo como banca examinadora o seu orientador, *o Prof. Dr. Leandro Oliveira Salviano, o Prof. Dr. Carlos Antonio Alves e o Prof. Dr. Jean Marcos de Souza Ribeiro*, apresentou o Trabalho de Graduação intitulado **"Desenvolvimento de metodologia para automação de aplicações web usando proxy e requisições HTTP"**, obtendo a nota 10 (DEZ) e conceito APROVADO.



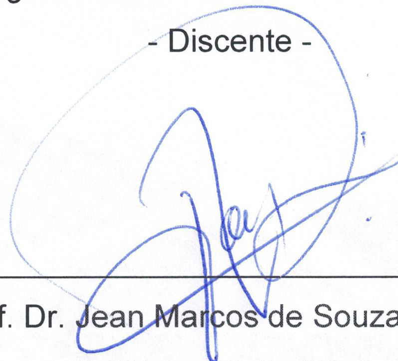
Prof. Dr. Leandro Oliveira Salviano
- Orientador-



Thiago Garcia Prado Martins de Oliveira
- Discente -



Prof. Dr. Carlos Antonio Alves
- Membro da Banca -



Prof. Dr. Jean Marcos de Souza Ribeiro
- Membro da Banca -

Faculdade de Engenharia – Departamento de Engenharia Elétrica
Avenida Professor José Carlos Rossi, 1370 - CEP 15.385-000, Ilha Solteira - SP - Brasil
Tel (18) 37431150 - Fax (18) 3742-2735 - www.feis.unesp.br

Dedico este trabalho à minha mãe, Claudia Garcia Prado, por estar presente e ser meu suporte em todos os momentos da minha vida e por ter-se dedicado incansáveis anos para que eu alcançasse esse objetivo. Ao meu avô, Wilson Garcia Prado, e à minha avó, Aparecida Cardamone Garcia Prado, por todo o conhecimento transmitido e por terem sido o suporte da minha mãe. Aos meus irmãos, Matheus Garcia Prado e Eloah Garcia Prado, por estarem presentes em todos os momentos da minha vida e me ajudarem a ver a vida de uma forma mais leve. O apoio e a presença de todos vocês foram essenciais para essa conquista.

AGRADECIMENTOS

Concluir essa etapa não seria possível sem a enorme rede de apoio que existiu em diversos momentos da minha vida, desta forma, dedico minha imensa gratidão a todos que contribuíram direta ou indiretamente.

À minha família, meu alicerce e minha maior fonte de força. Agradeço pelo amor incondicional, pelo apoio em todos os momentos e pela confiança depositada em mim, mesmo quando eu duvidei de mim mesmo. Cada palavra de incentivo, cada gesto de carinho e cada sacrifício feito por mim foram fundamentais para que eu chegasse até aqui. Este trabalho também é de vocês.

Aos meus amigos, Maria, Gabriel, Enzo, Eric, Vitor, Mariana, Nanda, Gustavo, Lucas Orlando e Fabrício. Dedico também um agradecimento especial para meu amigo Luís que me acompanhou por toda a trajetória, bem como me auxiliou em todos os momentos. Obrigado pelas conversas, explicações, ensinamentos, risadas e principalmente por também terem se tornado minha família.

À Isabele, pelo apoio, risadas, carinho e paciência ao longo dessa jornada. Sua presença foi fundamental nos momentos de dificuldade, seja com palavras de incentivo, seja simplesmente estando ao meu lado nos dias mais cansativos.

Ao meus amigos, Alan Brum, Marccio Alcaide e Guilherme Duarte por todo auxílio e ensinamento fornecidos para minha carreira profissional, vocês foram essenciais para que eu pudesse crescer como profissional e finalizar essa etapa.

Ao meu orientador, Prof. Leandro Oliveira Salviano, minha sincera gratidão pela orientação dedicada, pela paciência e por todo o conhecimento compartilhado ao longo desse trabalho. Sua orientação foi fundamental para o desenvolvimento deste trabalho e para o meu crescimento acadêmico e pessoal.

"Quero deixar uma marca no universo."
(Jobs, 1994, tradução nossa)

RESUMO

Com o crescimento do uso de aplicações web, a automação de processos que interagem com essas aplicações se tornou essencial para garantir eficiência, escalabilidade e redução de esforços manuais. Tradicionalmente, ferramentas como o Selenium são amplamente utilizadas para essa finalidade, porém operam em nível de interface gráfica, o que pode comprometer o desempenho em cenários com grande volume de operações. Este trabalho propõe uma metodologia alternativa baseada na automação via requisições HTTP, eliminando a necessidade de simulação de navegação, o que resulta em maior eficiência computacional e significativa melhoria no tempo de execução. A metodologia foi aplicada em um estudo de caso envolvendo a simulação de uma planta de gaseificação no ambiente IPSE GO. Para isso, somente as requisições necessárias para o processo de automação foram capturadas por meio de um *proxy*, analisadas e reproduzidas em um *script* escrito em Python. A automação foi capaz de realizar *login*, modificar parâmetros da planta, solicitar simulações e coletar resultados de forma programada. Como principais resultados, observou-se uma redução média de $81,08\%$ no tempo de execução em comparação ao processo manual, que demorava, em média, $37,00s$ na execução manual e $7,27s$ na execução realizada pelo *script*. Além disso, foram identificadas oportunidades de aprimoramento na estrutura da automação, como o armazenamento prévio dos dados da planta e a reutilização dos *tokens* de autenticação para contribuir com ganhos adicionais de performance. Esses resultados indicam que a metodologia é eficaz e aplicável a contextos que demandam repetição, agilidade e integração com outras ferramentas automatizadas.

Palavras-Chave: Engenharia de Simulação; Interceptação de Tráfego; Integração de Sistemas; Execução Programada.

ABSTRACT

With the growing use of web applications, automating processes that interact with these systems has become essential to ensure efficiency, scalability, and reduced manual effort. Traditionally, tools such as Selenium are widely used for this purpose, although they operate at the graphical interface level, which can compromise performance in scenarios involving a high volume of operations. This work proposes an alternative methodology based on automation through HTTP requests, eliminating the need for navigation simulation, which results in greater computational efficiency and a significant improvement in execution time. The approach was applied in a case study involving the simulation of a gasification plant in the IPSE GO environment. For this purpose, only the HTTP requests required for the automation process were captured using a proxy, analyzed, and reproduced in a Python script. The automation was able to perform login, modify plant parameters, trigger simulations, and collect results in a programmatic manner. As key results, an average reduction of $81,08\%$ in execution time was observed, decreasing from $37,00s$ in the manual process to $7.27s$ using the script. Furthermore, opportunities for improvement were identified, such as pre-storing plant data and reusing authentication tokens, aiming at additional performance gains. The results demonstrate that the proposed methodology is effective and feasible for applications requiring high repetition of operations, integration with automated pipelines, and increased task execution speed.

Keywords: Simulation Engineering; Traffic Interception; Systems Integration; Programmatic Execution.

LISTA DE ILUSTRAÇÕES

Figura 1 Exemplo de protocolo de comunicação humano. 14
Figura 2 Arquitetura de camadas de uma companhia aérea. 15
Figura 3 Conjunto de protocolos da Internet. 16
Figura 4 Exemplo de socket. 17
Figura 5 Exemplo de endereço IP. 19
Figura 6 Exemplo de requisição HTTP. 20
Figura 7 Exemplo de requisição HTTP. 21
Figura 8 Exemplo de requisição HTTP. 22
Figura 9 Modelo de funcionamento de um proxy. 24
Figura 10 Exemplo de planta. 27
Figura 11 Burp Suite página de configuração. 28
Figura 12 Fluxo de automação. 29
Figura 13 Fluxo de automação. 31
Figura 14 Filtragem por termo no BurpSuite. 33
Figura 15 Requisição de login. 34
Figura 16 Requisição de login. 34
Figura 17 Aquisição de planta. 35
Figura 18 Estrutura para solicitação de cálculo. 37
Figura 19 Criação dos dados para cálculo. 38
Figura 20 Alteração de dados da planta. 38
Figura 21 Conexão com websocket. 40
Figura 22 Solicitação e registro do cálculo. 40
Figura 23 Aquisição dos resultados. 41
Figura 24 Planta de gaseificação. 42

SUMÁRIO

1 INTRODUÇÃO . . . 10

2 FUNDAMENTAÇÃO TEÓRICA . . . 12
2.1 PROCESSO . . . 12
2.2 MODELO CLIENTE SERVIDOR . . . 13
2.3 PROTOCOLOS DE COMUNICAÇÃO . . . 13
2.4 ARQUITETURA DE CAMADAS . . . 14
2.4.1 CAMADA DE APLICAÇÃO . . . 16
2.4.2 CAMADA DE TRANSPORTE . . . 17
2.4.3 CAMADA DE REDE . . . 18
2.4.4 CAMADA DE ENLACE . . . 19
2.4.5 CAMADA FÍSICA . . . 19
2.5 SERVIDOR WEB . . . 19
2.6 PROTOCOLO HTTP . . . 20
2.7 PROXY . . . 23
2.8 WEBSOCKET . . . 24
2.9 PYTHON . . . 24

3 METODOLOGIA . . . 26
3.1 IPSE GO . . . 26
3.2 BURP SUITE . . . 27
3.3 PYTHON . . . 28
3.4 DESENVOLVENDO O FLUXO DE AUTOMAÇÃO . . . 28
3.5 O FLUXO DE DESENVOLVIMENTO . . . 30

4 RESULTADOS . . . 33
4.1 PROCESSO DE LOGIN . . . 33
4.2 PROCESSO DE AQUISIÇÃO DA PLANTA . . . 35
4.3 MONTAGEM DOS DADOS PARA CÁLCULO . . . 36
4.4 ALTERAÇÃO DOS DADOS PARA CÁLCULO . . . 38
4.5 SOLICITAÇÃO DE CÁLCULO . . . 39
4.6 AQUISIÇÃO DOS RESULTADOS . . . 41
4.7 DESEMPENHO DA AUTOMAÇÃO NO IPSE GO . . . 41
4.8 MELHORIAS . . . 44

5 CONCLUSÃO . . . 45

REFERÊNCIAS **46**

1 INTRODUÇÃO

Conforme apresentado por Ribeiro (2014), a partir de 2007, com a popularização das redes sociais, o número de websites cresceu intensamente. Nos anos subsequentes, o barateamento do acesso à internet viabilizou a migração de softwares, antes desenvolvidos exclusivamente para execução local, para plataformas baseadas na web. Essa transição tornou-se ainda mais acentuada com a popularização dos sistemas em nuvem (*cloud computing*) a partir de 2019 (Ansys, 2025), permitindo que desenvolvedores adotassem modelos de cobrança como o "*pay as you go*", nos quais os custos variam conforme a demanda de uso. Isso barateou significativamente diversos serviços. Do ponto de vista do usuário, essa mudança trouxe vantagens como a eliminação da necessidade de infraestrutura local especializada e a possibilidade de uso simultâneo por múltiplos usuários.

Entre os exemplos de plataformas que surgiram nesse contexto, destacam-se o IPSE GO, utilizado para simulações termodinâmicas e de gaseificação, e o SimScale, que permite simulações de fluidodinâmica computacional (CFD), análise de elementos finitos (FEA) e transferência de calor.

Com essa migração, a necessidade de automatizar tarefas recorrentes — como a coleta de dados, o envio de formulários e a realização de testes — tornou-se mais evidente. Essas automações são essenciais para aumentar a eficiência, reduzir o esforço manual e facilitar a integração com fluxos de trabalho automatizados. Diversos estudos investigaram o desempenho de ferramentas de automação aplicadas a sistemas *web*, como apresentado em (Gamido, 2019) e (Pelivani, 2021).

Apesar de sua popularidade, a automação baseada em interface gráfica apresenta limitações em termos de desempenho e escalabilidade, especialmente em cenários que exigem alto volume de execuções ou integração com sistemas automatizados. Como alternativa, destaca-se a automação baseada diretamente no protocolo *Hypertext Transfer Protocol* (HTTP) — a base da comunicação na web. Essa abordagem opera em um nível mais baixo, utilizando requisições diretas entre cliente e servidor, sem a necessidade de renderização gráfica. Isso reduz significativamente a sobrecarga computacional e permite maior controle sobre os dados manipulados.

O HTTP segue o modelo requisição-resposta, no qual o cliente envia uma requisição e o servidor responde com os dados solicitados. Essa estrutura, aliada à possibilidade de interceptar e analisar o tráfego por meio de ferramentas como *proxies* (e.g., Burp Suite ou mitmproxy), permite mapear e reproduzir interações com aplicações *web* de forma programada. Essa técnica tem sido explorada principalmente em contextos de segurança e testes (PortSwigger, 2025), mas seu uso como base para automação de processos ainda é pouco sistematizado na literatura.

Este trabalho propõe uma metodologia para automação de aplicações web baseada na comunicação HTTP, dispensando a simulação de navegadores. A abordagem explora a captura e

reprodução de requisições por meio de proxies, e a execução automatizada de interações com aplicações web através de scripts em Python. Como estudo de caso, a metodologia é aplicada ao IPSE GO, validando sua eficácia na simulação automatizada de uma planta de gaseificação.

Dentre os objetivos, destaca-se a avaliação da eficiência em comparação ao processo manual, a análise do desempenho e a identificação de oportunidades de otimização na arquitetura da automação. Os resultados obtidos demonstram reduções significativas no tempo de execução, além da manutenção da precisão dos dados, indicando que a metodologia é uma alternativa viável e eficiente para ambientes que demandam escalabilidade, repetição de processos e integração com *pipelines* de automação.

2 FUNDAMENTAÇÃO TEÓRICA

A automação de processos *web* utilizada neste trabalho baseia-se na interação entre diferentes sistemas por meio de requisições HTTP, que funcionam como um meio de comunicação entre processos em computadores independentes, caracterizando um sistema distribuído (MacLeod; Rodd, 2021). Assim, torna-se fundamental compreender conceitos como processos, comunicação interprocessos, o modelo cliente-servidor e as camadas do protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*).

2.1 PROCESSO

Um processo é, basicamente, um programa em execução. Na década de 1950, os computadores eram projetados para executar apenas um programa por vez. Com a evolução dos sistemas operacionais na década de 1960, surgiram técnicas como multiprogramação, permitindo que vários processos fossem carregados na memória e executados alternadamente pelo processador. Isso criou a ilusão de execução simultânea, conhecida como pseudoparalelismo. Já nos anos 1970 e 1980, com a popularização dos sistemas de tempo compartilhado e a introdução dos primeiros sistemas multitarefa, tornou-se comum que múltiplos processos rodassem concorrente e eficientemente nos computadores pessoais (Tanenbaum, 2015).

Em sistemas baseados em Unix como o Linux e o macOS, ao iniciar o sistema, um processo pai é gerado; dentre suas responsabilidades, está a de criar processos filhos por meio de chamadas de sistema como o *fork()*. Quando essa chamada é executada, é gerada uma cópia do processo pai. Após, o processo filho pode ser modificado para executar um programa diferente usando a chamada *exec()*, o que permite que ele se torne um processo completamente diferente. Esse processo de geração de filhos cria uma estrutura em árvore, com o processo pai no topo e seus filhos abaixo, podendo gerar novos filhos e assim por diante (Tanenbaum, 2015).

A cada processo criado é associado um espaço de endereçamento, o qual armazena de forma isolada na memória, todas as informações necessárias para o funcionamento do programa, como o programa executável, seus dados e sua pilha (Tanenbaum, 2015). Este isolamento permite que cada processo atue de forma independente e impede a interferência de um processo em outro.

A comunicação interprocessos (IPC) é um conceito fundamental na área de sistemas operacionais. Ela é essencial no desenvolvimento de sistemas complexos pois permite o compartilhamento de recursos e informações entre processos (MacLeod; Rodd, 2021). Além disto, ela também possibilita a comunicação de processos em diferentes máquinas, viabilizando assim a criação de sistemas distribuídos.

Processos que rodam no mesmo sistema se comunicam por meio de regras definidas pelo próprio sistema operacional. Já processos que rodam em sistemas diferentes se comunicam utilizando mensagens, as quais são enviadas por meio da rede de computadores (Tanenbaum,

2015).

2.2 MODELO CLIENTE SERVIDOR

Em redes de computadores, os dispositivos conectados à rede são chamados de sistemas finais. Para que dois programas ou processos em sistemas finais distintos possam se comunicar, é necessário adotar um modelo que defina as regras dessa interação. Um exemplo comum dessa comunicação é a conexão entre um navegador e um site, que são executados em computadores distintos.

Atualmente, o modelo mais amplamente utilizado para esse tipo de comunicação é o cliente-servidor, nele, os sistemas finais são classificados em dois papéis principais: o cliente, responsável por iniciar o processo de comunicação, e o servidor, que responde às solicitações do cliente. Normalmente, dispositivos como computadores pessoais, celulares e tablets desempenham o papel de clientes, enquanto os servidores são máquinas mais robustas, projetadas para executar serviços específicos, como aplicações web ou servidores de e-mail (Rocha, 2002).

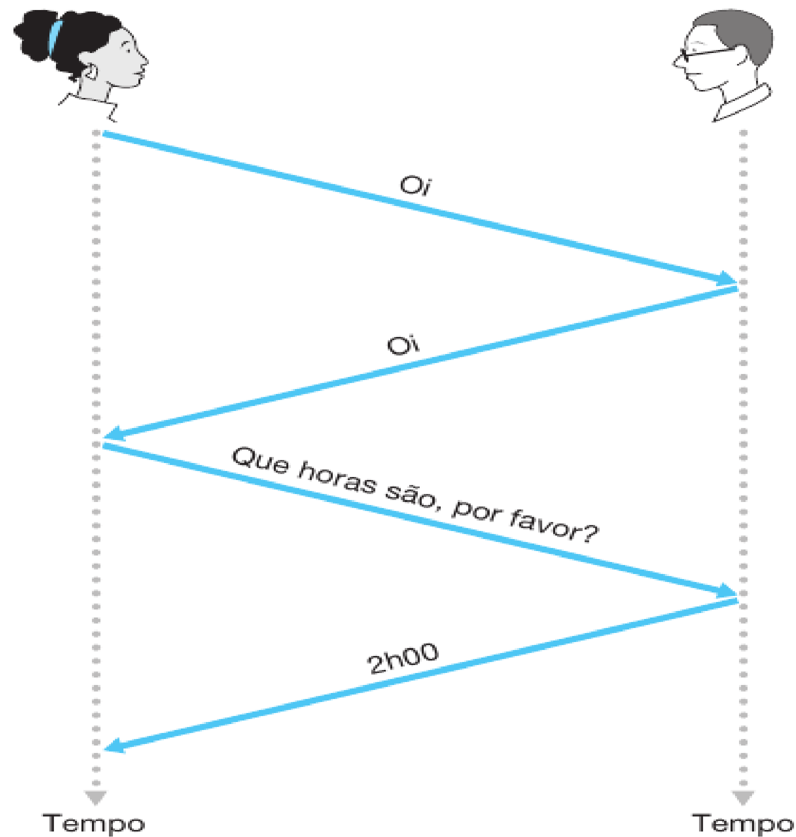
Uma analogia simples para esse modelo é a de uma pizzaria, onde a pizzaria representa o servidor, responsável por atender às solicitações de diversos clientes. Assim como na internet, o cliente é quem inicia a comunicação ao realizar o pedido da pizza, e o servidor passa a processar e entregar a solicitação. Além disso, destaca-se que uma pizzaria é capaz de atender a vários clientes simultaneamente, de maneira semelhante a como os servidores lidam com múltiplas conexões (Rocha, 2002).

2.3 PROTOCOLOS DE COMUNICAÇÃO

Um protocolo de comunicação define o formato, a ordem das mensagens trocadas entre duas entidades e o método de transmissão e recepção dessas mensagens. Essas entidades podem ser humanas ou sistemas computacionais (Kurose; Ross, 2021).

Por exemplo, no caso dos seres humanos, é considerado uma boa prática iniciar uma conversa com um cumprimento, como um “Oi”. A resposta do interlocutor, como “Oi” ou “Bom dia”, sinaliza que a comunicação pode prosseguir. Já respostas como “Não falo português” ou “Não perturbe” indicam que o receptor não deseja ou não pode continuar a interação. Se os comunicantes não utilizarem o mesmo "protocolo", como ao falarem idiomas diferentes, a comunicação se torna inviável — o mesmo princípio se aplica aos sistemas computacionais. Um exemplo de um protocolo aplicado à comunicação humana é apresentado na Figura 1 (Kurose; Ross, 2021).

Figura 1 – Exemplo de protocolo de comunicação humano.



Fonte: (Kurose; Ross, 2021)

Na internet, todas as atividades que envolvem entidades remotas comunicantes são regidas por protocolos. Por exemplo, protocolos *web* controlam o formato e a estrutura das mensagens enviadas entre dois computadores, e protocolos em roteadores definem o caminho de um pacote da origem ao destino (Kurose; Ross, 2021).

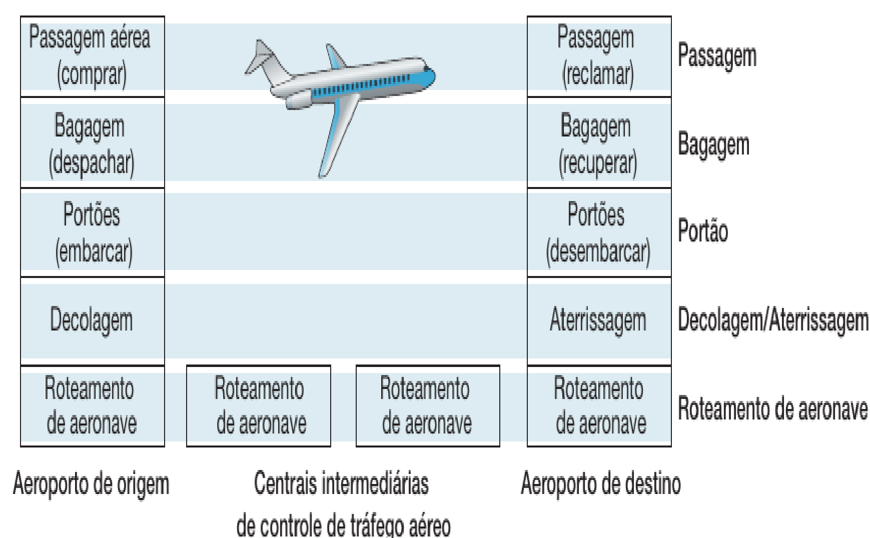
2.4 ARQUITETURA DE CAMADAS

Arquiteturas de sistemas complexos, como a da Internet, costumam dividir as responsabilidades de cada etapa do processo em camadas, criando um sistema modular, mais fácil de alterar, expandir e manter (Micheli, 2019).

Para facilitar a compreensão da arquitetura em camadas da Internet, pode-se utilizar elementos do funcionamento de outros sistemas complexos, como o de uma companhia aérea, apresentado na Figura 2. Nesse contexto, o processo pode ser comparado ao envio de um pacote na Internet. Inicialmente, é necessário comprar uma passagem, em seguida despacham-se as malas, realiza-se o embarque, viaja-se até o destino, desembarca-se e, por fim, recuperam-se as malas. Além disso, há também a possibilidade de registrar reclamações com a empresa responsável pelo voo, caso o serviço não tenha atendido às expectativas. Assim como na Internet, cada etapa desse processo é

análoga a uma camada que desempenha uma função específica no envio de um pacote de dados de um ponto a outro (Kurose; Ross, 2021).

Figura 2 – Arquitetura de camadas de uma companhia aérea.



Fonte: (Kurose; Ross, 2021)

A figura demonstra que, cada camada, juntamente com as camadas inferiores a ela, fornece um serviço. Por exemplo, na camada de bagagem e nas camadas abaixo dela, é realizada a transferência "despacho-de-bagagem-recuperação-de-bagagem"de um passageiro. É importante destacar que esse serviço só pode ser executado após o passageiro apresentar o bilhete de embarque, ou seja, depois que o serviço de compra de passagem aérea tiver sido concluído. Nesse contexto, cada camada oferece seu serviço ao realizar ações específicas dentro de si e ao utilizar os serviços fornecidos pela camada imediatamente inferior. Por exemplo, nos portões de embarque, utiliza-se o serviço da camada de decolagem/aterrissagem (Kurose; Ross, 2021).

Este modelo facilita a manutenção do sistema, pois, desde que uma camada continue fornecendo o mesmo serviço para a camada imediatamente superior, o método pelo qual ela realiza esse serviço pode ser alterado sem que sejam necessárias outras modificações no restante do sistema. Por exemplo, se as funções dos portões forem modificadas, o restante do sistema permanecerá inalterado, já que essa camada continuará a prover o mesmo serviço, mas de forma diferente (Kurose; Ross, 2021).

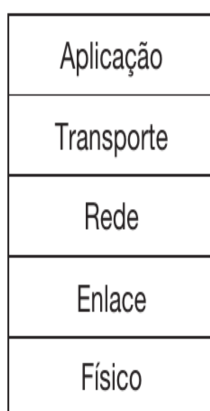
Assim, a arquitetura em camadas possibilita a modularização e a abstração dos processos, garantindo maior flexibilidade, escalabilidade e facilidade de manutenção nos sistemas complexos, como a Internet. Cada camada desempenha um papel essencial, interagindo com as camadas vizinhas para garantir a entrega eficiente dos serviços. Essa abordagem permite que avanços tecnológicos e otimizações sejam implementados em camadas individuais sem impactar o funcionamento geral do sistema, tornando-o mais adaptável a novas demandas e desafios.

Dessa forma, a compreensão desse modelo não apenas facilita o entendimento da arquitetura da Internet, mas também auxilia no desenvolvimento de soluções robustas e eficientes em diversos domínios da computação e da engenharia de software (Kurose; Ross, 2021).

A Internet organiza os protocolos em camadas de forma que cada protocolo pertence a uma camada, assim como cada função na arquitetura de linha aérea pertencia a uma camada. Por exemplo, os serviços da camada 'n' podem incluir uma entrega confiável de mensagens entre os dois extremos da rede, enquanto o protocolo da camada 'n-1' não; isto adiciona uma funcionalidade na camada 'n' que detecta e retransmite mensagens perdidas (Kurose; Ross, 2021).

Uma camada de protocolo pode ser implementada em um programa, em hardware, ou nos dois. O conjunto de protocolos da Internet é formado por cinco camadas, apresentadas na Figura 3: física, enlace, rede, transporte e aplicação. A camada de aplicação é sempre implementada em um programa, tendo como protocolos mais utilizados o HTTP. A camada de transporte também é implementada em software, possuindo protocolos como o UDP (*User Datagram Protocol*) e TCP. Já a camada de rede é uma implementação mista de software e hardware, e seu principal protocolo é o Protocolo de Internet (IP). Já a camada física e de enlace são implementadas em hardware e possuem como principal responsabilidade o manuseio da comunicação por um enlace.

Figura 3 – Conjunto de protocolos da Internet.



Fonte: (Kurose; Ross, 2021)

2.4.1 CAMADA DE APLICAÇÃO

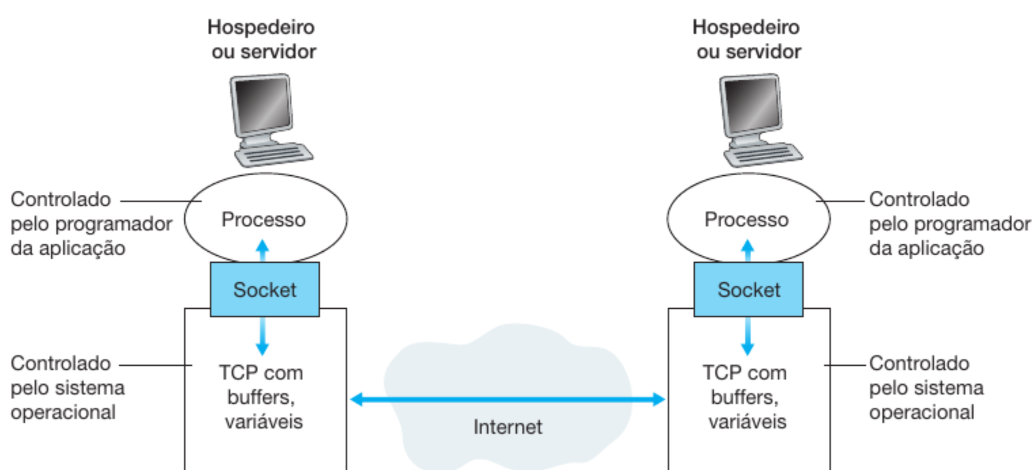
A camada de aplicação é uma das camadas de rede que possui o maior número de protocolos. Por ser a camada mais alta do modelo de comunicação, ela oferece aos desenvolvedores maior controle sobre as características dos protocolos utilizados. Esse nível de controle diminui progressivamente à medida que se desce pelas camadas em direção à camada física, que lida com a transmissão de dados no nível mais básico (Kurose; Ross, 2021).

Entre os protocolos da camada de aplicação estão o HTTP, amplamente utilizado na web; o *Simple Mail Transfer Protocol* (SMTP), responsável pelo envio de e-mails; o *File Transfer Protocol* (FTP), empregado na transferência de arquivos; e o *Secure Shell* (SSH), utilizado para acesso remoto seguro (Murkomen, 2024).

Esses protocolos possibilitam a troca de mensagens entre aplicações em diferentes sistemas finais. Essa comunicação é realizada a partir de uma interface de software chamada socket, conforme apresentado na Figura 4. Para entender esse conceito, pode-se utilizar como exemplo uma casa e uma porta; nesta analogia, a casa seria um processo e a porta seria o socket. Quando uma aplicação deseja se comunicar com um processo em outro sistema final, ela empurra essa mensagem pela sua porta. Ao chegar ao hospedeiro destinatário, a mensagem passa pela porta (socket) do processo receptor que então executa alguma ação sobre a mensagem (Kurose; Ross, 2021).

Para que haja uma comunicação entre dois sistemas finais distintos, é necessário que sejam definidos dois valores: o endereço IP e a porta. O endereço IP é um valor composto por 4 blocos numéricos separados por um ponto e é utilizado para identificar de forma exclusiva um computador numa rede. Já a porta indica o processo alvo com o qual o usuário deseja se comunicar. Isto é necessário, pois um servidor pode estar executando mais de uma aplicação de rede; logo, o usuário precisa indicar com qual delas deseja se comunicar (Kurose; Ross, 2021).

Figura 4 – Exemplo de socket.



Fonte: (Kurose; Ross, 2021)

2.4.2 CAMADA DE TRANSPORTE

Os protocolos da camada de transporte fornecem uma comunicação lógica para os processos ou aplicações que rodam em sistemas diferentes de forma que, para a aplicação, é como se os processos estivessem conectados diretamente. Isto permite que a camada de aplicação não se preocupe com detalhes da infraestrutura física. Uma das principais funcionalidades dela é entre-

gar os pacotes de rede provenientes da camada de rede para seus respectivos processos/sockets, algo conhecido como demultiplexação (Palma, 2018).

Esta camada opera com dois protocolos: o TCP e o UDP. O primeiro oferece para a aplicação um serviço confiável, orientado a conexão, enquanto o segundo não. Também é importante citar que o protocolo utilizado na camada de rede não garante que os dados sejam entregues aos hospedeiros; portanto, caso seja necessário, essa validação e verificação de erros devem ser realizadas na camada de transporte (Nor; Alubady; Kamil, 2017).

O protocolo TCP oferece uma transferência confiável de dados, usando números de sequência, reconhecimentos, temporizadores e controle de congestionamento para assegurar que os dados sejam entregues do processo remetente ao processo destinatário corretamente e em ordem; isto o faz ser usado em aplicações que necessitam desse tipo de confiabilidade, como, por exemplo, sua conexão com um site bancário ou a transferência de arquivos (Kurose; Ross, 2021).

O UDP é um protocolo de transporte leve e rápido que não garante entrega ordenada nem retransmissão de pacotes perdidos. Ele é ideal para aplicações onde a latência é mais importante do que a confiabilidade, como, por exemplo, *streaming* de áudio, jogos online ou serviços de voz (Palma, 2018).

2.4.3 CAMADA DE REDE

Enquanto a camada de transporte tem a função de entregar os pacotes ao nível de processo, sem se preocupar com os detalhes da comunicação entre sistemas, a camada de rede é responsável por fornecer a comunicação entre sistemas localizados em diferentes redes, garantindo que os dados sejam entregues de um ponto a outro, independentemente das diferenças na infraestrutura física e na topologia da rede (Kurose; Ross, 2021). A principal função desta camada é o roteamento, ou seja, determinar o melhor caminho para os pacotes de dados, com base nas informações de endereçamento e nas condições da rede (Cunha, 2018).

O protocolo fundamental dessa camada é o IP, que realiza o endereçamento e o roteamento dos pacotes de dados de uma rede a outra. O IP é responsável por garantir que os pacotes sejam enviados ao destino correto, mas não oferece garantias quanto à entrega. Ou seja, o IP apenas se preocupa em encaminhar os pacotes, sem verificar se chegaram corretamente ou se houve perda de dados (Cunha, 2018).

Entre outras funções, o protocolo define que cada dispositivo na rede deve ter um endereço IP, sendo ele formado por quatro números, onde cada um deles varia entre 0 e 255, conforme apresentado na Figura 5. Cada dispositivo costuma ter dois endereços IP: o primeiro, que o identifica dentro da rede à qual está conectado, seja uma rede corporativa ou doméstica; o segundo, que identifica essa rede na Internet, sendo esse endereço atribuído ao roteador e não ao dispositivo. Estes endereços são chamados de IP interno e IP externo, respectivamente (Cunha, 2018).

Figura 5 – Exemplo de endereço IP.

255 . 255 . 255 . 255

Fonte: Elaborado pelo autor.

Embora o IP não forneça confiabilidade, ele desempenha um papel crucial no endereçamento e na entrega dos pacotes. A confiabilidade e a ordem dos pacotes são gerenciadas por protocolos da camada de transporte, como o TCP, que depende do endereçamento realizado pelo IP para assegurar que os dados cheguem corretamente ao destino (Kurose; Ross, 2021).

2.4.4 CAMADA DE ENLACE

A camada de enlace tem como responsabilidade organizar os dados da camada de rede em quadros e lidar com o controle de erros, como a detecção de falhas na transmissão. A camada de enlace de dados é responsável pela comunicação direta entre dispositivos que estão conectados na mesma rede física (Glabbeek, 2019).

Uma das principais funções dessa camada é garantir que os dados sejam corretamente encapsulados em quadros (frames) e entregues ao destino dentro da mesma rede ou segmento de rede. Ela também lida com a detecção e correção de erros de transmissão que possam ocorrer devido a interferências ou falhas de comunicação na mídia física, como fios, cabos ou ondas de rádio em redes sem fio (Glabbeek, 2019).

Existem dois tipos principais de protocolos na camada de enlace: o Ethernet (utilizado em redes locais cabeadas) e o Wi-Fi (usado em redes sem fio). Ambos são responsáveis por definir como os quadros são formados e transmitidos (Kurose; Ross, 2021).

Portanto, ela assegura que os dados possam ser transmitidos de forma segura e eficiente em uma rede, lidando com os aspectos de formatação, detecção de erros e controle de acesso.

2.4.5 CAMADA FÍSICA

A camada física é responsável pela transmissão dos dados reais através do meio físico, como cabos, fibras ópticas ou sinais de rádio. Ela converte os bits em sinais elétricos, ópticos ou de rádio, permitindo a comunicação entre dispositivos. Sua principal função é garantir que os bits sejam transmitidos ao longo do meio de comunicação, sem se preocupar com a estrutura ou o controle de erros dos dados. Ela lida com aspectos como a modulação dos sinais e a velocidade de transmissão, fazendo com que os dados cheguem fisicamente de um dispositivo ao outro (Mishra, 2021).

2.5 SERVIDOR WEB

As camadas de protocolo têm a função de permitir a comunicação entre dois sistemas. Um dos exemplos mais comuns dessa interação ocorre entre um navegador e um servidor *web*.

O servidor *web* é um software responsável por armazenar, processar e entregar páginas web aos usuários por meio da internet. Ele opera basicamente em duas etapas: primeiro, recebe requisições dos clientes (como navegadores); em seguida, processa e envia respostas, entregando os arquivos solicitados, como páginas HTML (*HyperText Markup Language*), imagens e *scripts* (Jader; Zeebaree; Zebar, 2019).

Quando um usuário digita um endereço de site no navegador, o navegador envia uma requisição por meio do protocolo HTTP (*Hypertext Transfer Protocol*) para o servidor *web*. Essa requisição pode ser para obter uma página HTML, imagens, vídeos ou até mesmo arquivos de configuração. O servidor *web*, ao receber essa requisição, processa as informações e, com base nas configurações e nos arquivos armazenados, retorna uma resposta (Jader; Zeebaree; Zebar, 2019).

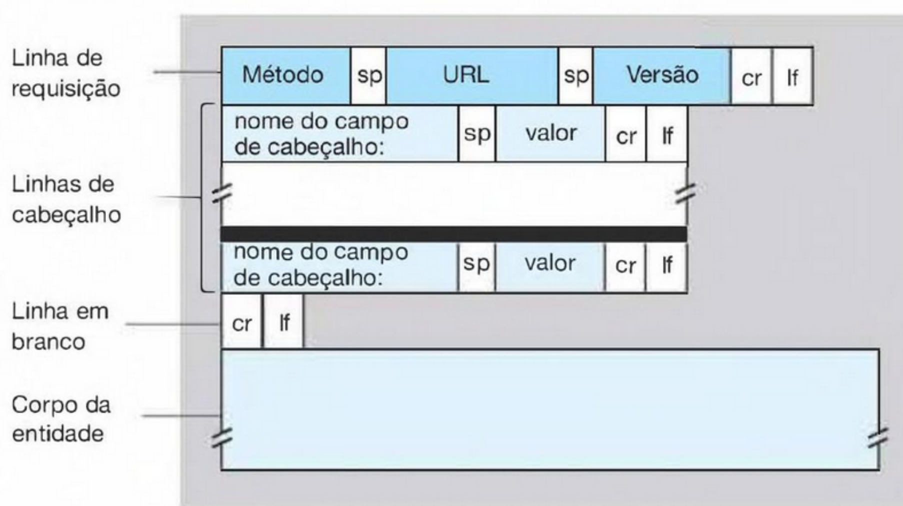
A resposta do servidor web geralmente inclui o conteúdo da página solicitada, como o código HTML, e também pode incluir cabeçalhos HTTP, que fornecem informações adicionais sobre a resposta, como tipo de conteúdo ou *status* da requisição (Jader; Zeebaree; Zebar, 2019).

2.6 PROTOCOLO HTTP

O protocolo HTTP é um protocolo sem estado da camada de aplicação; ele funciona no modelo cliente-servidor, sendo majoritariamente utilizado em aplicações *web*.

Neste protocolo ocorre a troca de mensagens entre o cliente e o servidor, onde o primeiro realiza uma requisição ou *request* e o servidor devolve uma resposta ou *response* (Zineddine, 2024). Um exemplo de *request* é apresentado na Figura 6.

Figura 6 – Exemplo de requisição HTTP.



Fonte: (Kurose; Ross, 2021)

Na primeira linha há três valores separados por um espaço, o primeiro é chamado de método da requisição HTTP, ele indica que tipo de recurso está sendo utilizado ou acessado dentro da

aplicação. Os dois valores seguintes são o *path* ou o recurso acessado e a versão do protocolo HTTP. Todas as linhas abaixo dessa são chamadas de *headers*, ou seja, cabeçalhos, elas enviam informações adicionais sobre a solicitação como o tipo de informação, a língua e o navegador utilizado (Filho, 2015).

Dentre os métodos existentes, alguns são importantes de serem explicados, sendo eles:

- *GET* : O cliente requisita algum recurso, como uma página ou uma imagem.
- *POST* : O cliente está enviando dados que estão contidos no corpo da request
- *PUT* : Parecido com o *POST*, diferindo apenas em como o servidor irá lidar com os dados enviados. Por exemplo, caso seja necessário atualizar os dados de um usuário, usa-se o método *PUT*, pois com ele o servidor irá sobrescrever os dados antigos com os novos, gerando somente um registro, com o *POST* o servidor cria vários registros, um para cada *request* feita.
- *DELETE* : O cliente requisita que algum recurso seja excluído do servidor

As utilizações supracitadas são apenas recomendações de um sistema, entretanto, eles não precisam ser seguidos necessariamente. Dentro de alguns métodos como o *POST* e o *PUT*, é necessário enviar juntamente com o método e os cabeçalhos um *body*, o qual costuma conter as informações a serem enviadas; ele é separado por uma linha em branco dos cabeçalhos da requisição, como apresentado na Figura 7. Também há outro método para enviar informações para o sistema: o *query params*; nele, as informações são enviadas dentro da *URL* a ser acessada. Esse método é muito utilizado para consultas e aquisições de valores, entretanto, não deve ser usado para a passagem de valores sensíveis, como senhas, visto que ele poderia ser facilmente encontrado no histórico de navegação do usuário ou no registro de eventos da aplicação (Filho, 2015).

Figura 7 – Exemplo de requisição HTTP.

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Content-Type: text/plain
Content-Lenght: 325
Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
Connection: keep-alive
Cookie:

body
```

Fonte: Elaborado pelo autor.

Já dentro dos cabeçalhos mais utilizados, pode-se citar:

- *Host* : Nome de domínio do servidor. Alguns exemplos são: *google.com* e *youtube.com*
- *User-Agent* : Usado pelo servidor para identificar quem está fazendo a *request*. Ela contém dados como navegador e sistema operacional do cliente.
- *Accept* : Exprime quais tipos de dados o cliente é capaz de entender. Alguns exemplos são: *text/plain* e *application/json*
- *Content-Type* : Indica qual o tipo dos dados que o servidor ou o cliente está enviando. Alguns exemplos são: *text/plain* e *application/json*
- *Accept-Language* : Indica qual linguagem o cliente entende. Usado para definir se uma página estará em português ou inglês, por exemplo.
- *Connection* : Define se a conexão com o servidor deve ser mantida para futuras requisições, no primeiro caso seu valor é *keep-alive*, no segundo *close*.
- *Cookie* : Os cookies permitem que o servidor mantenha informações sobre a sessão do usuário, garantindo que ele permaneça autenticado entre requisições. Eles são enviados pelo servidor no cabeçalho *Set-Cookie* e armazenados no navegador do cliente, que os retorna em todas as requisições subsequentes. Isto é necessário pois o HTTP é um protocolo *stateless*, ou seja, não é possível identificar um usuário em solicitações subsequentes.
- *Authorization*: Algumas aplicações não utilizam *cookies* para autenticação, preferindo o uso de *tokens*, que são enviados no cabeçalho *Authorization*. Esses *tokens* podem adotar diferentes formatos, não havendo um consenso absoluto sobre qual modelo é o mais adequado, pois a escolha do formato depende de fatores como segurança, escalabilidade e requisitos específicos da aplicação.

Como *response* da primeira requisição realizada, tem-se a Figura 8.

Figura 8 – Exemplo de requisição HTTP.

```
HTTP/1.1 200 OK
Date: Mon, 11 May 2020 19:45:35 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Strict-Transport-Security: max-age=31536000
Server: gws
X-Frame-Options: SAMEORIGIN
Set-Cookie:
Connection: close
Content-Length: 194814
```

Fonte: Elaborado pelo autor.

A primeira linha é dividida em três valores, o primeiro é a versão do protocolo HTTP utilizada, o segundo é um *status* que indica o resultado da solicitação, o terceiro é o nome do *status* (Filho, 2015). Os status ainda são inseridos em grupos que variam em ranges de 100, onde cada um deles indica uma particularidade da resposta dentro daquele grupo, alguns *status* mais comuns são:

- *1XX* : Retorna informações sobre a solicitação, se a solicitação foi aceita, ou se o processo continua em desenvolvimento.
- *2XX* : A solicitação foi executada com sucesso.
- *3XX* : Indica que há a necessidade de redirecionamento para que a solicitação possa ser concluída.
- *4XX* : Mostra que houve um erro na solicitação por parte do cliente.
- *5XX* : Indica que o servidor não pôde responder a solicitação.

Alguns cabeçalhos mais comuns na resposta são:

- *Date* : Data em que a resposta foi originada.
- *Expires* : Indica quando o conteúdo deve ser considerado desatualizado, neste caso o valor -1 significa que o conteúdo expira imediatamente após ser enviado.
- *Cache-Control* : Define políticas de cache.
- *Etag* : Identifica uma versão específica de algum recurso. Permite assim que o servidor não envie a resposta completa, proporcionando uma maior velocidade.
- *Server* : Define informações acerca do servidor.
- *Set-Cookie* : Usado para o servidor enviar *cookies* para o cliente.

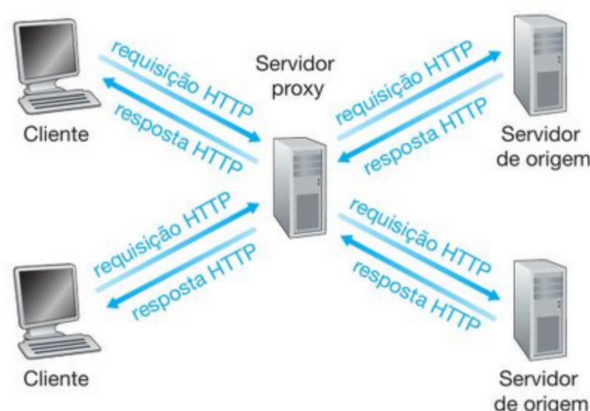
Dessa forma, o protocolo HTTP desempenha um papel fundamental na comunicação entre clientes e servidores na web. Sua estrutura baseada em requisições e respostas permite a transferência de dados de forma eficiente, enquanto os diferentes métodos e cabeçalhos proporcionam flexibilidade para atender às necessidades das aplicações modernas.

2.7 PROXY

Servidores *proxy*, também conhecidos como *cache web*, são aplicações que têm a funcionalidade de atender requisições HTTP em nome de outro servidor *web* (Kurose; Ross, 2021). Ao ser configurado no navegador, cada requisição realizada é enviada primeiro para ele. Após, ele envia a solicitação de origem para o alvo, adquire o resultado e direciona este para o navegador

de volta. Além disto, *Proxies* funcionam como cliente e servidor. Ao receber requisições de um navegador e devolver-lhe a resposta, ele atua como um servidor. Quando envia requisições para um site e recebe essa resposta, ele atua como um cliente (Abiona, 2014).

Figura 9 – Modelo de funcionamento de um proxy.



Fonte: (Kurose; Ross, 2021)

2.8 WEBSOCKET

O protocolo *WebSocket* foi desenvolvido como uma extensão do HTTP para fornecer comunicação bidirecional entre cliente e servidor. Com ele, após a conexão ser estabelecida pelo cliente, ambas as partes podem trocar mensagens livremente, sem a necessidade de múltiplas requisições HTTP (Melnikov; Fette, 2011).

Diferentemente do HTTP, que segue um modelo requisição-resposta, o WebSocket mantém uma conexão persistente, permitindo que o servidor envie mensagens ao cliente sem necessidade de uma solicitação prévia. Isso o torna ideal para aplicações que exigem comunicação em tempo real, como jogos online e sistemas de chat (Melnikov; Fette, 2011).

2.9 PYTHON

Python é uma linguagem de programação de alto nível, interpretada e multiparadigma, amplamente utilizada no desenvolvimento de aplicações web, automação, ciência de dados, inteligência artificial e diversas outras áreas. Criada por Guido van Rossum e lançada em 1991, Python se destaca por sua sintaxe simples e legível, facilitando a aprendizagem e o desenvolvimento rápido de software (Python, 2025).

Com uma vasta biblioteca padrão e um ecossistema rico em pacotes de terceiros, Python oferece suporte para manipulação de dados, redes, desenvolvimento web, aprendizado de máquina, requisições HTTP e conexões *WebSocket*.

A linguagem também é amplamente adotada na automação de tarefas e no desenvolvimento de scripts para administração de sistemas, graças à sua integração nativa com diferentes plataformas. Além disso, sua popularidade cresceu significativamente nos últimos anos, tornando-se uma das linguagens mais utilizadas no mundo da programação.

3 METODOLOGIA

A metodologia adotada neste trabalho visa estabelecer um processo sistemático para o desenvolvimento de automações baseadas em aplicações *web* que utilizam requisições HTTP. O foco está na interceptação, análise e reprodução dessas requisições com o auxílio de ferramentas específicas, permitindo a interação programática com sistemas que, originalmente, foram projetados para o uso humano via interface gráfica. Para demonstrar a aplicabilidade da abordagem proposta, escolheu-se o *software* IPSE GO como estudo de caso, dada sua natureza *web* e dependência de simulações computacionais intensivas. A seguir, são descritos os principais componentes envolvidos na automação, as ferramentas utilizadas e o fluxo de desenvolvimento adotado.

3.1 IPSE GO

O IPSE GO é um *software* de simulação de processos baseado em nuvem, desenvolvido pela Simtech. Ele permite a modelagem e análise de sistemas complexos, sendo amplamente utilizado em setores como geração de energia, dessalinização e refrigeração, entre outros. Diferente de *softwares* locais, onde os cálculos são processados diretamente na máquina do usuário, no IPSE GO as simulações são executadas remotamente nos servidores da Simtech, garantindo alto desempenho e acesso centralizado aos dados. Outra vantagem da ferramenta é sua interface gráfica que permite, por meio de blocos, a criação do fluxo do processo a ser simulado (Symtech, 2025).

Neste trabalho, o IPSE GO foi adotado como estudo de caso para o desenvolvimento de uma automação capaz de interagir com a plataforma de forma programática. A automação implementada é responsável por enviar requisições diretamente ao IPSE GO, processar as respostas recebidas e extrair informações relevantes provenientes das simulações. Além disso, quando necessário, o sistema permite modificar os dados da planta, o que se mostra especialmente útil em contextos de análise paramétrica, otimização de desempenho e integração entre múltiplas plantas de simulação.

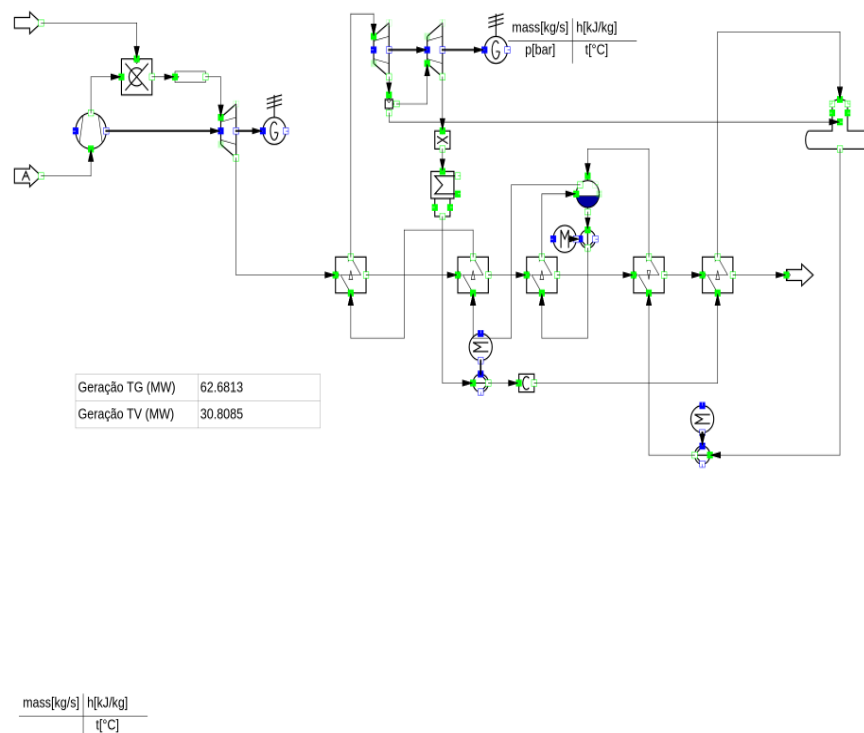
Essa capacidade de alterar dinamicamente os parâmetros da planta permite realizar simulações iterativas com diferentes configurações, sem a necessidade de intervenção manual, o que amplia significativamente o potencial da ferramenta para análises complexas. Um exemplo prático dessa aplicação pode ser observado no trabalho de Paula e Salviano (2025) , no qual a automação foi utilizada para integrar duas plantas distintas. Nessa integração, 12 parâmetros foram alterados de forma coordenada entre as duas configurações, permitindo que a saída de uma planta fosse utilizada como entrada para a outra, viabilizando um fluxo contínuo de simulações.

O objetivo central dessa abordagem é reduzir a dependência da interação humana no processo de execução das simulações, aumentando a eficiência, a reprodutibilidade dos resultados e a capacidade de integração com outras ferramentas ou pipelines automatizados. Dessa forma,

a metodologia proposta amplia as possibilidades de aplicação do IPSE GO em cenários mais robustos e automatizados, como otimizações numéricas, simulações em série e ambientes de engenharia integrados.

Um exemplo de uma planta de processo criada no IPSE GO foi apresentada na Figura 10.

Figura 10 – Exemplo de planta.



Fonte: Elaborado pelo autor.

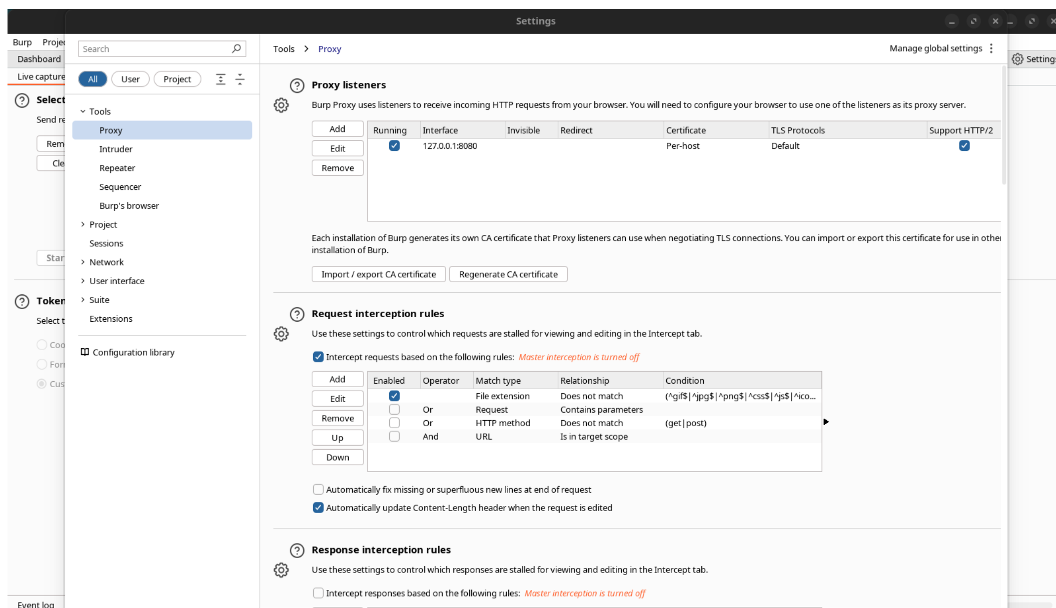
3.2 BURP SUITE

Para reproduzir as requisições HTTP necessárias no processo de automação, é fundamental adquiri-las previamente. Isso pode ser realizado por meio de um *proxy* que intercepta e registra o tráfego entre o navegador e o site-alvo, permitindo a captura tanto das requisições enviadas quanto das respostas recebidas.

Dentre as ferramentas disponíveis, destaca-se o Burp Suite, amplamente utilizado na área de segurança de aplicações web. Essa ferramenta possibilita a interceptação, modificação e reenvio de requisições, além de fornecer recursos avançados para análise de tráfego. Alternativamente, outras soluções, como mitmproxy, também podem ser empregadas dependendo do cenário e dos requisitos específicos da automação (PortSwigger, 2025).

Após a instalação do Burp Suite, é necessário identificar o endereço IP e a porta onde seu servidor *proxy* está operando. Essas informações podem ser obtidas na página de configurações da ferramenta, conforme apresentado na Figura 11. Uma vez configurado, o tráfego do navegador pode ser redirecionado para o *proxy*, permitindo a captura das requisições.

Figura 11 – Burp Suite página de configuração.



Fonte: Elaborado pelo autor.

No Mozilla Firefox, por exemplo, essa configuração pode ser ajustada na seção de configurações de rede, onde a opção "Configuração manual de proxy"permite inserir os valores de IP e porta obtidos. Com o proxy devidamente configurado, todas as requisições HTTP enviadas pelo navegador e suas respectivas respostas podem ser acessadas na aba "HTTP history"do Burp Suite, possibilitando sua análise e posterior reprodução no processo de automação.

3.3 PYTHON

Para desenvolver a automação que solicita e adquire os dados da simulação, foi utilizada a linguagem Python, devido à sua versatilidade e ampla gama de bibliotecas para manipulação de requisições HTTP e conexões *socket*. A automação implementada é capaz de interagir com o IPSE GO de forma programática, enviando requisições para iniciar simulações, monitorar o progresso dos cálculos e coletar os resultados gerados pelo software.

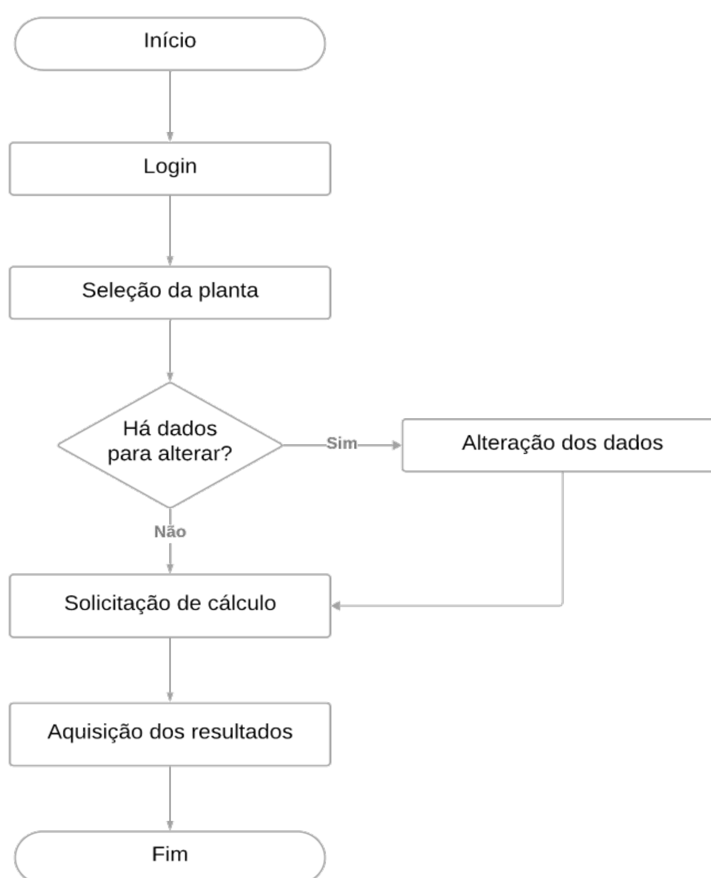
Por meio de bibliotecas como *requests* e *websockets* para a comunicação com o sistema do IPSE GO, a automação permite que todo o processo seja executado de maneira eficiente e sem necessidade de intervenção manual. Além disso, a modularidade do código possibilita adaptações futuras para diferentes cenários de simulação, tornando a solução flexível e escalável.

3.4 DESENVOLVENDO O FLUXO DE AUTOMAÇÃO

Para automatizar um processo em uma aplicação *web*, é fundamental compreender previamente todas as ações necessárias para a sua execução completa. No caso do estudo realizado com a plataforma IPSE GO, o fluxo inicia-se com o processo de autenticação do usuário, ou seja,

o *login* na aplicação. Após essa etapa, realiza-se a seleção da planta desejada para simulação. Caso haja necessidade, os dados da planta podem ser ajustados ou reconfigurados conforme os parâmetros definidos pelo usuário. Por fim, aciona-se a funcionalidade de simulação através do botão “Calcule”, e os resultados gerados são coletados para posterior análise. O detalhamento desse fluxo é apresentado na Figura 12, que ilustra as etapas necessárias para a realização completa do processo de forma sequencial e lógica.

Figura 12 – Fluxo de automação.



Fonte: Elaborado pelo autor.

Cada aplicação *web* pode apresentar fluxos de operação distintos, variando em complexidade e número de etapas. Esses fluxos podem ser mais simples ou significativamente mais elaborados do que o apresentado neste estudo. Portanto, a definição clara e precisa do fluxo de execução é essencial para identificar quais ações realmente precisam ser automatizadas. Esse mapeamento contribui para evitar o desperdício de esforço em etapas irrelevantes e garante que a automação foque apenas nos pontos críticos e recorrentes do processo.

É importante destacar que a automação de processos *web* nem sempre é mais simples que a execução manual. As interfaces gráficas frequentemente oferecem abstrações que tornam a experiência do usuário mais intuitiva e fluida. No entanto, ao automatizar essas etapas, o desenvolvedor precisa reconstruir essas abstrações manualmente no código, utilizando requisições

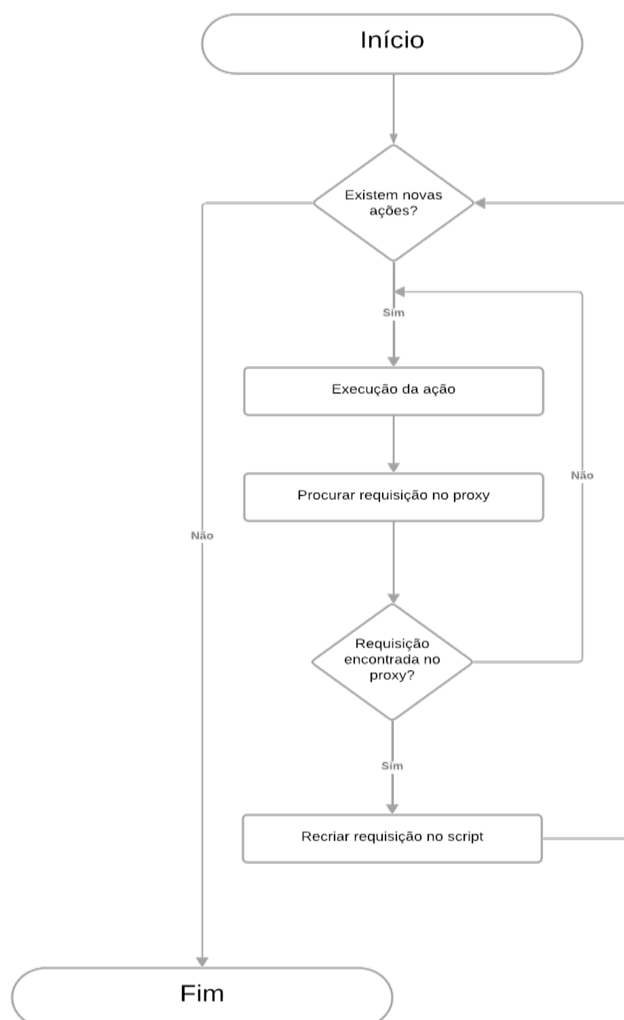
HTTP, manipulação de *tokens*, identificação de *endpoints* e construção dos dados de entrada, o que exige conhecimento técnico aprofundado.

Uma abordagem que pode facilitar o processo de modelagem do fluxo é entender que cada interação do usuário com a interface — como um clique — representa uma ação lógica. No entanto, representar todos os cliques de forma literal em um fluxograma pode tornar a visualização redundante e pouco eficiente. Assim, recomenda-se interpretar a intenção por trás de cada clique, abstraindo essas interações em ações significativas, como “Selecionar planta”, “Modificar parâmetros” ou “Executar simulação”. Essa abstração torna o fluxo mais claro, genérico e reaproveitável, o que é particularmente vantajoso em projetos com potencial de expansão ou adaptação para outras aplicações *web* semelhantes.

3.5 O FLUXO DE DESENVOLVIMENTO

Desenvolvido o fluxo geral de execução dos processos, tornou-se necessário definir como cada etapa seria analisada para possibilitar sua reprodução no *script*. Para isso, considerou-se que cada etapa do fluxo apresentado na Figura 12 representa uma "ação"distinta. A partir dessa definição, foi criado um fluxo de desenvolvimento aplicável a qualquer processo de automação baseado em requisições HTTP. Esse fluxo, apresentado na Figura 13, descreve a lógica utilizada para identificar, capturar e reproduzir cada ação de forma automatizada, garantindo a fidelidade da implementação e facilitando a escalabilidade do sistema.

Figura 13 – Fluxo de automação.



Fonte: Elaborado pelo autor.

O funcionamento desse fluxo inicia-se com a verificação da existência de novas ações a serem implementadas na automação. Caso existam, realiza-se a execução manual da ação no navegador com o *proxy* devidamente configurado para capturar o tráfego HTTP. Em seguida, é feita uma análise no histórico do *proxy* para verificar se a requisição gerada pela ação está registrada. Se a requisição for encontrada, ela é interpretada e reproduzida dentro do *script* de automação, considerando seus parâmetros, cabeçalhos, método e corpo da mensagem, quando aplicável. Caso a requisição não seja localizada ou seja inválida para o propósito da automação, o processo retorna à etapa anterior, possibilitando a reexecução da ação e nova análise no *proxy*. Após a conclusão de uma ação, o ciclo se reinicia para a próxima, até que todas as etapas definidas tenham sido processadas e convertidas em requisições automatizadas.

Esse fluxo permite que a automação seja construída de forma modular, segura e verificável, uma vez que cada ação é validada individualmente e transcrita com base em uma análise concreta do tráfego gerado pela aplicação. Além disso, garante que o *script* final reproduza fielmente as

interações originais da aplicação *web*, sem a necessidade de simulação gráfica via navegador, promovendo maior eficiência e escalabilidade no processo automatizado.

4 RESULTADOS

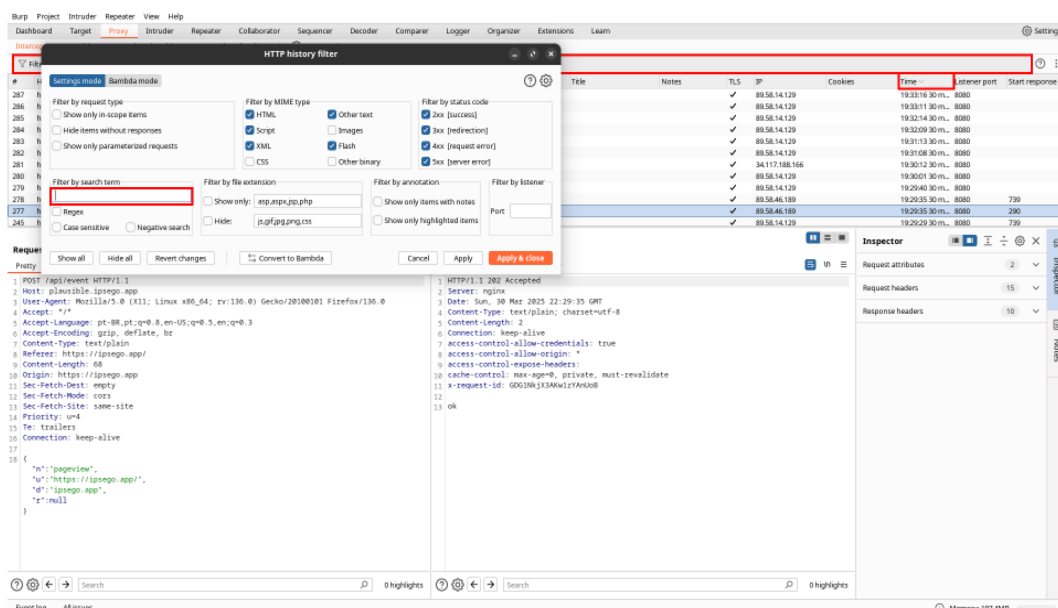
Realizada a montagem da automação, avaliou-se o desempenho da solução desenvolvida em comparação com o processo manual, considerando métricas como o tempo de execução, a reprodutibilidade dos dados e a viabilidade da aplicação em contextos de maior escala ou repetição. Também são discutidos aspectos relacionados à eficiência da abordagem proposta e possíveis oportunidades de otimização identificadas durante os testes.

4.1 PROCESSO DE LOGIN

Com o proxy devidamente configurado no navegador, o primeiro passo para automatizar o processo de autenticação consiste na realização manual do *login* na aplicação. Durante essa etapa, as requisições e respostas geradas são registradas, permitindo a análise do fluxo de comunicação entre o cliente e o servidor. A identificação da requisição correspondente ao *login* pode ser facilitada pelo uso de filtros e ordenações no Burp Suite, os quais permitem segmentar o tráfego com base em critérios como método HTTP, host, endereço IP e momento da requisição.

Neste estudo, a abordagem adotada para localizar a requisição de login consistiu em aplicar uma ordenação cronológica e um filtro baseado no e-mail inserido no formulário de *login*. Como o processo de autenticação exige credenciais (e-mail e senha), a busca por requisições contendo o endereço de e-mail utilizado facilitou a identificação da requisição específica, conforme ilustrado na Figura 14.

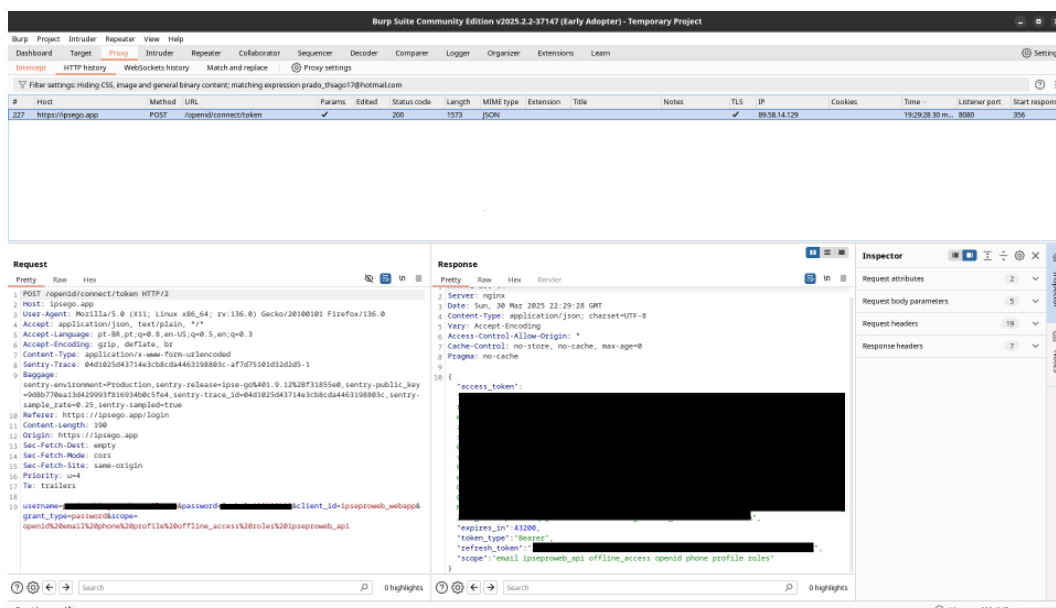
Figura 14 – Filtragem por termo no BurpSuite.



Fonte: Elaborado pelo autor.

A requisição resultante da aplicação do filtro corresponde ao processo de autenticação, visto que contém os parâmetros de e-mail e senha, além de um *token* de acesso na resposta. Esse *token*, geralmente encontrado no corpo da resposta, é um elemento essencial para a automatização, pois permite que as requisições subsequentes sejam autenticadas sem necessidade de interação manual. A requisição identificada é apresentada na Figura 15.

Figura 15 – Requisição de login.



Fonte: Elaborado pelo autor.

Visando a escalabilidade e a flexibilidade da aplicação, realizou-se o desenvolvimento de uma função em Python para automatizar este processo; ela deve receber como entrada o e-mail e a senha do usuário e, como saída, deve retornar o seu *token* de autenticação, permitindo assim uma interação programática com a aplicação. Esta função foi apresentada na Figura 16.

Figura 16 – Requisição de login.

```python
1  baseURL = "https://ipsego.app"
2
3
4  def doLogin(user: str, password: str) → str:
5      data = {
6          "username": user,
7          "password": password,
8          "client_id": "ipseproweb_webapp",
9          "grant_type": "password",
10         "scope": "openid email phone profile offline_access roles ipseproweb_api",
11     }
12     res = requests.post(f"{baseURL}/openid/connect/token", data=data)
13     if res.status_code ≠ 200:
14         raise ValueError(f"Error login: {res.status_code}; {res.text}")
15
16     res = res.json()
17     return f"Bearer {res['access_token']}"
```

Fonte: Elaborado pelo autor.

4.2 PROCESSO DE AQUISIÇÃO DA PLANTA

Após a aquisição do *token* responsável por identificar o usuário dentro do IPSE GO, a próxima etapa do processo de automação é a aquisição dos dados da planta. Ela é necessária pois, ao realizar a solicitação de cálculo, os dados do projeto devem ser enviados como parte da solicitação. Desta forma, destaca-se a importância do entendimento prévio do funcionamento do sistema, o qual é adquirido analisando as requisições realizadas na execução do processo manual.

É comum que sistemas *web* usem um ID (identificador) único para diferenciar itens dentro do sistema, podendo eles serem valores numéricos inteiros ou alfanuméricos (Texas, 2025).

Este ID é importante pois, com base nele, pôde-se adquirir ou alterar informações sobre o item usando requisições HTTP.

No contexto da automação para adquirir dados da planta, deve-se, inicialmente, adquirir seu ID, o qual pode ser encontrado diretamente na aplicação web do IPSE GO ou pelas requisições no *proxy*. No primeiro caso, basta acessá-la e seu ID estará na URL. Já no segundo caso, deve-se filtrar as requisições pela única informação acessível da planta: o seu nome. Ainda é importante frisar que pode-se utilizar um filtro via método HTTP, neste caso, como deseja-se adquirir um recurso, ele deve ser do tipo GET, diferentemente do processo de *login* que, por necessitar do envio de dados, utiliza-se o método POST.

No código Python, desenvolveu-se uma função que recebe como entrada o cabeçalho HTTP, o qual deve conter o *token* de autenticação do usuário e o ID da planta e deve retornar, como resposta, seus dados. Como um usuário pode ter mais de um projeto, forneceu-se para o IPSE GO um filtro por meio do seu ID no *query params*. O *token* do usuário também deve ser fornecido, pois é com base nele que a aplicação permite ou não a aquisição dos dados da planta. Por fim, também validou-se o status da resposta da requisição HTTP, conforme apresentado em 2.6; os *status* entre 200 e 299 informam que a solicitação foi atendida com sucesso.

Figura 17 – Aquisição de planta.

```python
def getProjectData(headers: dict, projectId: str) -> dict:
    params = {"id": projectId}
    res = requests.get(f"{baseURL}/api/v1/projects", params=params, headers=headers)
    if res.status_code != 200:
        raise ValueError(f"Error getting project;{res.text};{res.status_code}")

    return res.json()
```

Fonte: Elaborado pelo autor.

4.3 MONTAGEM DOS DADOS PARA CÁLCULO

Na Figura 12, foram apresentadas as ações manuais necessárias para o processo de simulação de uma planta. É importante destacar que, embora a interface do IPSE GO permita editar elementos da simulação a qualquer momento, sem a necessidade de um conjunto de dados previamente estruturado, o mesmo não ocorre na automação.

No desenvolvimento da automação, observou-se que é mais simples e eficiente montar previamente o conjunto completo de dados e, somente após isso, realizar alterações específicas. Essa diferença impõe um desafio adicional: o fluxo automatizado não pode simplesmente replicar, de forma linear, as ações realizadas pela interface. É necessário antecipar etapas e estruturar dados com antecedência, mesmo que, na experiência do usuário, essas informações só sejam exibidas mais adiante.

A detecção dessa necessidade exige uma análise detalhada, mas resulta em um ganho significativo de performance no sistema. Isso só pôde ser observado ao analisar o processo por completo, não focando apenas na etapa atual. Por exemplo, na interface web, é necessário salvar manualmente as alterações antes de solicitar o cálculo, o que implica em uma requisição adicional. No entanto, ao examinar o corpo da requisição responsável pelo cálculo, percebe-se que todas as informações da planta são enviadas diretamente nela, eliminando a necessidade de um salvamento prévio.

Esse descompasso entre a experiência visual e a lógica de funcionamento da automação torna o desenvolvimento mais complexo, exigindo uma compreensão aprofundada do comportamento interno da aplicação. Além disso, destaca-se a importância de testes no ambiente real, uma vez que a interface não evidencia essa particularidade: a eliminação da etapa de salvamento só foi descoberta por meio de tentativas de envio direto da solicitação de simulação.

O IPSE GO utiliza a estrutura base apresentada na Figura 18 para realizar as solicitações de cálculo dos projetos. Dentro dessa estrutura, alguns campos se destacam por sua importância, como o "LibGUID"e o "LibName". Para compreender seu papel, é necessário entender como a plataforma organiza determinadas informações.

Figura 18 – Estrutura para solicitação de cálculo.

```
{
    "arguments": [
        {
            "LibGUID": "",
            "LibName": "",
            "Task": "stat",
            "solverParameters": {
                "bExtendedProtocol": false,
                "bUseDamping": false,
                "nSteps": 10,
                "xTolerance": 0.001,
                "yTolerance": 0.001
            },
            "analysisOptions": {
                "bModelAnalysisEnabled": false,
                "bSettingsAnalysisEnabled": true
            },
            "dataFrameCells": []
        }
    ],
    "invocationId": "1",
    "target": "RequestCalculation",
    "type": 1
}
```

Fonte: Elaborado pelo autor.

O IPSE GO oferece diferentes tipos de análises, como simulações de plantas de usinas de energia com turbinas a gás e plantas de gaseificação. Cada tipo de planta possui parâmetros específicos — como valores de temperatura de componentes, massa e propriedades dos materiais — necessários para o cálculo da simulação. Devido à natureza invariável desses dados, eles são armazenados em bibliotecas, que funcionam como repositórios de modelos prontos, desta forma, toda planta é atrelada diretamente a uma biblioteca. Ao solicitar o cálculo da planta na interface *web*, a própria aplicação une os dados da planta com os da biblioteca e solicita seu cálculo. Entretanto, na automação, esse valor foi retornado ao solicitar os dados da planta e adquiridos com o auxílio do *proxy* tendo sido utilizado na montagem da solicitação.

Os campos "LibGUID"e "LibName"representam, respectivamente, o identificador único e o nome da biblioteca utilizada. Esses valores são fundamentais para que o sistema reconheça corretamente os parâmetros técnicos que compõem a simulação solicitada. Campos como "Task", "solverParameters", "analysisOptions"e "dataFrameCells"são invariantes e independem da planta. Por essa razão, não foi realizada uma análise aprofundada sobre suas funções.

Por fim, a construção do restante da estrutura necessária foi realizada por meio de um processo de engenharia reversa. Para isso, analisou-se a requisição gerada pela interface da plataforma, e cada campo presente nessa solicitação foi replicado no código da automação, conforme apresentado na Figura 19.

Figura 19 – Criação dos dados para cálculo.

```python
def prepareData(projectData: dict) -> dict:
    data = json.load(open("utils/payload.json"))

    content = json.loads(projectData["content"])
    data["arguments"].insert(0, projectData["id"])
    data["arguments"].insert(1, content["LibGUID"])

    payload = data["arguments"][2]
    payload["LibGUID"] = content["LibGUID"]
    payload["LibName"] = content["LibName"]
    payload["globals"] = createGlobals(content)
    payload["connections"] = createConnectionsPayload(content)
    payload["units"] = createUnitsPayload(content)
    data["arguments"].append(0)
    return data
```

Fonte: Elaborado pelo autor.

4.4 ALTERAÇÃO DOS DADOS PARA CÁLCULO

Com a estrutura de dados devidamente montada para a solicitação de cálculo, a automação também permite, quando necessário, a alteração de valores específicos antes do envio. Para isso, foi implementada uma função responsável por iterar sobre os elementos da estrutura e identificar aquele que corresponde ao item a ser modificado. Uma vez localizado, o valor é atualizado conforme o novo parâmetro desejado. Como exemplo, tem-se a Figura 20, a qual apresenta a composição dos elementos na referência global *syngas*.

Figura 20 – Alteração de dados da planta.

```json
{
    "SYNGAS": {
        "CH4": "",
        "CO": "",
        "CO2": "",
        "H2": "",
        "H2O": ""
    }
}
```

Fonte: Elaborado pelo autor.

É importante destacar que, mesmo após a modificação dos valores, a estrutura da requisição

deve permanecer intacta. Ou seja, não devem ser adicionados ou removidos campos — apenas seus conteúdos podem ser atualizados para refletir as novas configurações.

4.5 SOLICITAÇÃO DE CÁLCULO

Após a montagem da estrutura de dados para a solicitação de cálculo, iniciou-se o desenvolvimento da penúltima etapa da automação. Como ilustrado na Seção 3.4, é comum que interfaces *web* abstraiam certas complexidades para proporcionar uma melhor experiência ao usuário — e este caso é um exemplo claro disso.

Quando os processos da aplicação envolvem múltiplas etapas, identificá-las pode representar um desafio considerável. No entanto, uma abordagem eficaz consiste em ordenar as requisições capturadas por data de origem e iniciar a análise a partir da última etapa do fluxo. Essa ordenação permite compreender a sequência completa de ações realizadas pela aplicação, facilitando o rastreamento das etapas necessárias até a conclusão do processo.

Ao iniciar pela última requisição, torna-se possível analisar quais dados foram enviados e levantar questionamentos estratégicos, como: "É possível gerar todos esses dados com as informações disponíveis atualmente?" ou "De onde vieram os dados que ainda não consigo montar?". Com a requisição em mãos e as etapas ordenadas cronologicamente, as respostas geralmente estão em uma ou mais interações anteriores. Além disso, o uso de filtros por termos-chave nas requisições pode acelerar significativamente esse processo investigativo.

Por exemplo, durante o desenvolvimento da automação com o IPSE GO, inicialmente buscou-se identificar o envio dos dados do projeto por meio das requisições HTTP. No entanto, como tais informações não foram encontradas nessa camada de comunicação, a investigação foi direcionada para as conexões via *webSocket*.

O primeiro ponto de atenção foi a identificação de um valor até então desconhecido: o "connectionToken", utilizado como um identificador de sessão nas comunicações por *webSocket*. A partir da análise das requisições anteriores, localizou-se uma que retornava esse "connectionToken"na resposta. Após testes, confirmou-se que esse valor era necessário para o estabelecimento correto da conexão com o *webSocket*.

Com base nessa descoberta, foram implementadas duas funções específicas: a primeira responsável por obter o "connectionToken", e a segunda encarregada de estabelecer a conexão com o *webSocket*. Além do "connectionToken", essa segunda função também requer o fornecimento do token de autenticação do usuário, previamente obtido no processo de *login*. Por fim, essa função deve retornar a conexão estabelecida com o *webSocket*, a qual será utilizada nas etapas subsequentes da automação.

Figura 21 – Conexão com websocket.

```python
def getConnectionToken(headers: dict) -> str:
    res = requests.post(
        f"{baseURL}/signalr/negotiate?negotiateVersion=1", headers=headers
    )
    if res.status_code != 200:
        raise ValueError(f"Error login: {res.status_code}; {res.text}")

    res = res.json()
    return res["connectionToken"]


async def startConnectionWebSocket(
    connectionToken: str, headers: dict
) -> websockets.WebSocketClientProtocol:
    auth = headers["Authorization"].replace("Bearer ", "")
    url = f"wss://ipsego.app/signalr?id={connectionToken}&access_token={auth}"
    websocket = await websockets.connect(url)
    await websocket.send('{"protocol":"json","version":1}\x1E')
    return websocket
```

Fonte: Elaborado pelo autor.

É importante observar que, após o estabelecimento da conexão com o *webSocket*, é enviada automaticamente uma mensagem contendo informações sobre o protocolo e a versão utilizados na comunicação. Testes demonstraram que a conexão não pode ser mantida sem essa interação inicial, tornando esse envio um passo obrigatório no processo de autenticação do canal.

Por fim, com a conexão estabelecida e validada, os dados referentes à solicitação devem ser enviados e devidamente registrados para que o processo de simulação seja iniciado corretamente.

Figura 22 – Solicitação e registro do cálculo.

```python
async def requestCalculation(
    payload: dict, websocket: websockets.WebSocketClientProtocol
) -> int:
    await websocket.send(json.dumps(payload) + "\x1E")
    while True:
        res = await websocket.recv()
        if "calculationId" in res:
            res = res[:-1]
            return json.loads(res)["result"]["calculationId"]


async def registerCalculation(
    websocket: websockets.WebSocketClientProtocol, id: int
) -> None:
    payload = {
        "arguments": [{"id": id}],
        "invocationId": "2",
        "target": "RegisterCalculation",
        "type": 1,
    }
    await websocket.send(json.dumps(payload) + "\x1E")
```

Fonte: Elaborado pelo autor.

4.6 AQUISIÇÃO DOS RESULTADOS

O processo de aquisição dos resultados baseou-se na identificação, nas respostas recebidas via *webSocket*, de trechos que indicassem o término do cálculo. Essa detecção foi possível por meio da análise da comunicação entre a interface *web* e os servidores do IPSE GO.

Nesse contexto, vale destacar o uso do mecanismo de "ping/pong", comum em conexões *webSocket*, no qual o cliente envia um "ping"e o servidor responde com um "pong". Esse mecanismo é utilizado para monitorar e manter a conexão ativa, permitindo que ambas as partes identifiquem a necessidade de encerrá-la ou reiniciá-la.

Dessa forma, durante a etapa de obtenção dos resultados, foi implementado um laço de repetição infinito para aguardar as respostas do servidor. No entanto, essa abordagem pode causar problemas como *loops* infinitos, caso o servidor não envie a resposta esperada. Para contornar essa situação, foi adicionado um contador que monitora a quantidade de mensagens recebidas. Caso cinco respostas sejam recebidas sem que nenhuma contenha os dados esperados, o laço é encerrado e a automação finalizada de forma segura.

Figura 23 – Aquisição dos resultados.

Fonte: Elaborado pelo autor.

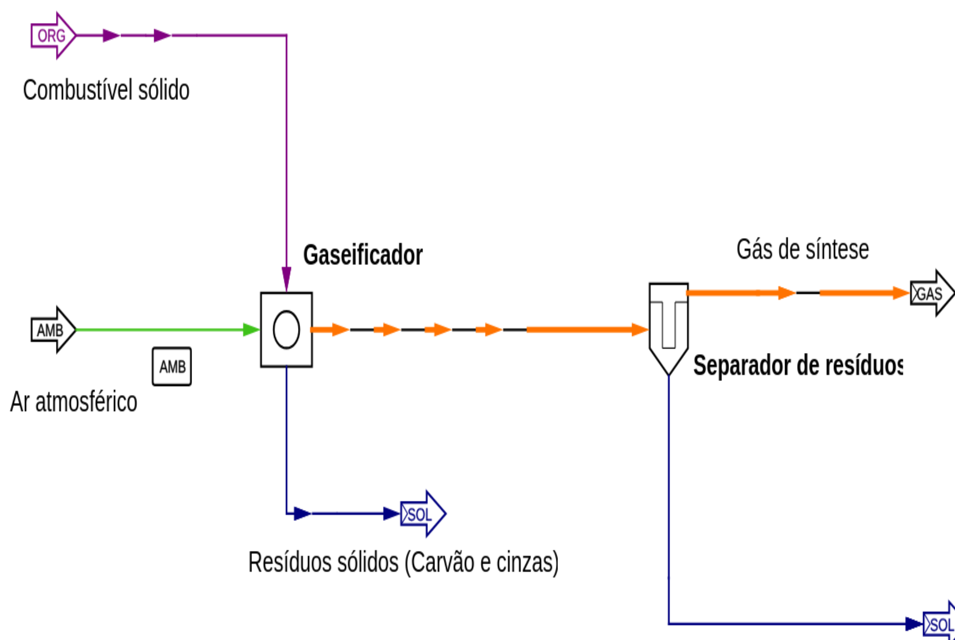
4.7 DESEMPENHO DA AUTOMAÇÃO NO IPSE GO

Os resultados obtidos a partir da aplicação da metodologia proposta evidenciam a eficácia da automação desenvolvida para a interação com aplicações web por meio de requisições HTTP. A automação implementada atingiu os objetivos estabelecidos, demonstrando um desempenho consistente e eficiente no contexto analisado.

Para a validação da metodologia, foi selecionada uma planta de gaseificação cujo processo envolvia a conversão de biomassa sólida em um gás combustível por meio de uma reação química

pobre em oxigênio, a qual foi apresentada na Figura 24.

Figura 24 – Planta de gaseificação.



Fonte: Elaborado pelo autor.

Para o cálculo do tempo de execução do processo, o fluxo da Figura 12 foi reproduzido quatro vezes a partir do *script* desenvolvido, bem como a partir da interface do IPSE GO. Neste caso, foi realizada a alteração de quatro parâmetros da planta de forma que o processo seja mais verossímil com um caso de uso real. O tempo de execução do fluxo manual foi apresentado na Tabela 1.

Tabela 1 – Tempo de execução manual.

Execução	Tempo (s)
1	43,00
2	36,00
3	35,00
4	34,00

Fonte: Elaborado pelo autor.

Já o fluxo automatizado é apresentado na Tabela 2.

Tabela 2 – Tempo de execução automatizado.

Execução	Tempo (s)
1	6,78
2	7,13
3	7,18
4	7,99

Fonte: Elaborado pelo autor.

A partir das tabelas, encontrou-se que o tempo médio de execução para o método manual e automático são, respectivamente: $37,00s$ e $7,27s$. Sendo esta uma diferença percentual de $81,08\%$, ou seja, para cada execução manual pode-se realizar, em média, 5 automáticas.

Ainda é importante frisar que o tempo para a execução manual poderia crescer consideravelmente caso a quantidade de parâmetros alterados fosse maior; esse aumento não ocorreria na automática, visto que essa operação não é custosa computacionalmente.

A utilização do *script* também permite o desenvolvimento de processos que necessitam de um grande número de repetições, como é o caso de análises de sensibilidade e otimizações numéricas, e seriam portanto, muito custosos de serem realizados manualmente. Em Paula e Salviano (2025), o *script* desenvolvido neste trabalho foi utilizado para realizar um processo de otimização numérica do acoplamento de duas plantas de bibliotecas diferentes do IPSE GO. Os autores utilizaram alguns dados de saída da primeira planta como dados de entrada para a segunda. Se este trabalho fosse realizado manualmente, o usuário teria que obter os resultados da primeira planta, retirar os desejados, inseri-los na segunda planta, simulá-la e, por fim, aquisitar os dados desejados. Esse processo é considerado ineficiente para a execução em larga escala como a realizada pelos autores, que desenvolveram uma otimização multiobjetivo com um total de doze variáveis distribuídas entre as duas plantas, que necessitou de mais de 10000 execuções até atingir a convergência.

Nesse caso, a integração das plantas com o algoritmo de otimização foi realizada por meio do software modeFRONTIER. A partir de uma amostra inicial (combinações das variáveis de entrada produzidas pelo próprio software através de um algoritmo e faixas de variação escolhidas pelo usuário), o algoritmo de otimização, no caso os algoritmos genéticos, produz novos indivíduos, ou combinações das variáveis de entrada, que em conjunto são chamadas de geração. Esses novos casos são simuladas novamente e os dados de saída são comparados com os anteriores, de forma a testar sua aptidão de acordo com o objetivo desejado, em geral, maximizar ou minimizar uma ou mais funções objetivo. Esse processo é repetido até que o melhor resultado seja encontrado (convergência) ou até 5000 gerações. Esse processo ainda pode contar com restrições de projeto, que limitam ainda mais a solução, necessitando de ainda mais casos para atingir a convergência.

Em síntese, os resultados obtidos evidenciam que a automação proposta não apenas reduziu significativamente o tempo de execução das tarefas viabilizou a aplicação da metodologia em

cenários de alta repetição e análise iterativa.

4.8 MELHORIAS

A automação de processos web, além de replicar ações manuais de forma programada, também possibilita a implementação de diversas estratégias de otimização que visam reduzir o tempo de execução e tornar a interação com o sistema mais eficiente. Essas otimizações são particularmente relevantes em cenários onde múltiplas execuções são necessárias, como em análises iterativas, otimizações numéricas ou simulações em larga escala.

Uma das otimizações mais eficazes envolve o gerenciamento inteligente da autenticação. Em vez de realizar o processo de *login* a cada execução, é possível armazenar os *cookies* de sessão, reutilizando-os enquanto estiverem válidos. O *login* só seria executado novamente caso o *cookie* expire ou se torne inválido, o que reduz significativamente o número de requisições de autenticação e, consequentemente, o tempo total de execução.

Outra otimização relevante consiste no armazenamento local dos dados da planta. Esses dados, uma vez adquiridos do servidor, podem ser reutilizados em interações subsequentes, evitando a repetição de requisições desnecessárias. Isso é especialmente útil quando os parâmetros da planta não variam entre as execuções ou quando pequenas variações podem ser tratadas localmente no *script*.

Além disso, também é possível armazenar informações da biblioteca do IPSE GO, como componentes, equações ou propriedades físicas, que normalmente não sofrem alterações frequentes. Manter essas informações localmente reduz ainda mais a dependência de requisições ao servidor e melhora a responsividade da automação.

Outra possibilidade, embora mais complexa, é a reutilização de conexões abertas, como as conexões via *WebSocket*. Em alguns sistemas, manter uma conexão persistente pode reduzir drasticamente a latência entre ações. No entanto, essa abordagem depende das características específicas da aplicação em questão. Por exemplo, no estudo de Paula e Salviano (2025), que utilizou a plataforma modeFRONTIER para executar algoritmos de otimização, isso não seria viável visto que o modeFRONTIER analisa cada indivíduo da população de forma independente, reiniciando o ambiente de execução a cada avaliação. Nesses casos, conexões e estados mantidos entre execuções não podem ser preservados, tornando a abordagem de *WebSocket* inaplicável.

De forma geral, a adoção dessas otimizações deve considerar o comportamento do sistema automatizado, os requisitos de desempenho do processo e os limites impostos pela infraestrutura da aplicação *web*. Quando bem aplicadas, essas estratégias elevam significativamente a performance da automação, tornando-a mais adequada a contextos de alta repetição e grande volume de dados.

5 CONCLUSÃO

O projeto de desenvolvimento de uma metodologia para a automação de processos web por meio de requisições HTTP foi concluído com êxito. A abordagem desenvolvida demonstrou ser capaz de reduzir significativamente o tempo de execução das tarefas, atingindo um ganho percentual de $508,94\%$ em comparação ao método manual, além de manter a precisão dos resultados.

A aplicação da automação em um cenário específico de simulação de uma planta de gaseificação evidenciou a robustez da metodologia ao replicar, de forma consistente, os cálculos obtidos manualmente. Esse resultado reforça a viabilidade do uso da automação em processos iterativos e de alta repetição, como otimizações numéricas e análises de sensibilidade, conforme demonstrado no estudo paralelo realizado por Paula e Salviano (2025).

Adicionalmente, durante a execução dos testes, foram identificadas oportunidades de aprimoramento na estrutura da automação. A implementação de estratégias de otimização, como o armazenamento prévio dos dados da planta e a reutilização dos *tokens* de autenticação, configuram-se como medidas viáveis para reduzir o tempo de execução e aumentar a eficiência do sistema.

Em síntese, o presente trabalho atingiu os objetivos propostos, validando a aplicação da automação baseada em requisições HTTP no contexto estudado. Os resultados obtidos consolidam a eficácia da metodologia e sugerem novas possibilidades de aplicação da automação em cenários que demandam alta escalabilidade, minimização do tempo de execução e precisão nos resultados.

REFERÊNCIAS

ABIONA, O. Proxy server experiment and network security with changing nature of the web. **International Journal of Communications, Network and System Sciences**, [S.L.], v. 07, p. 519–528, 2014.

ANSYS. **Study Cloud Computing Engineering Simulation**. Ansys, 2025. Disponível em: https://www.ansys.com/resource-center/white-paper/study-cloud-computing-engineering-simulation. Acesso em: 25 mai. 2025.

CUNHA, J. de S. **Protocolos de roteamento dinâmico RIP e OSPF** Orientador(a): Fábio Éder Cardoso: Fundamentos e simulação. 2018. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) — Instituto Municipal de Ensino Superior de Assis – IMESA/Fundação Educacional do Município de Assis – FEMA, Assis, 2018. Disponível em: https://cepein.femanet.com.br/BDigital/arqTccs/1511420434.pdf. Acesso em: 7 mai. 2025.

FILHO, A. B. Q. D. P. D. S. **Ataques de negação de serviço na camada de aplicação** Orientador(a): Edna Dias Caned: Estudo de ataques lentos ao protocolo http. 2015. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) — Faculdade de Tecnologia da Universidade de Brasília, Brasília, 2015. Disponível em: https://bdm.unb.br/bitstream/10483/15693/1/2015_AlejandroBarriosQuintanilla_DeijavalPereiraFilho.pdf. Acesso em: 7 mai. 2025.

GAMIDO, H. V. Comparative review of the features of automated software testing tools. **International Journal of Electrical and Computer Engineering (IJECE)**, [S.L.], v. 9, 2019.

GLABBEEK, R. van. A process algebra for link layer protocols. **Lecture Notes in Computer Science**, [S.L.], p. 668–693, 2019.

JADER, O. H.; ZEEBAREE, S. R. M.; ZEBAR, R. R. A state of art survey for web server performance measurement and load balancing mechanisms. **International Journal of Scientific & Technology Research**, [S.L.], v. 8, p. 535–543, 2019.

JOBS, S. Steve jobs: The lost interview. 1994.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top-down**. São Paulo: Pearson; Porto Alegre: Bookman, 2021.

MACLEOD, I. M.; RODD, M. G. Inter-process communication (ipc) in distributed environments: An investigation and performance analysis of some middleware technologies. **International Journal of Modern Education and Computer Science**, [S.L.], p. 1–6, 2021.

MELNIKOV, A.; FETTE, I. **The WebSocket Protocol**. RFC Editor, 2011. Disponível em: https://datatracker.ietf.org/doc/html/rfc6455. Acesso em: 7 mai. 2025.

MICHELI, G. J. Modularization as a system life cycle management strategy: Drivers, barriers, mechanisms and impacts. **International Journal of Engineering Business Management**, [S.L.], v. 11, p. 1–6, 2019.

MISHRA, S. Osi model the basics structure of network communication. [S.L.], v. 9, n. 5, p. 66–69, 2021.

MURKOMEN, T. Performance, privacy, and security issues of tcp/ip at the application layer. **GSC Advanced Research and Reviews**, [S.L.], v. 18, n. 3, p. 234–264, 2024.

NOR, S. A.; ALUBADY, R.; KAMIL, W. A. Simulated performance of tcp, sctp, dccp and udp protocols over 4g network. **Procedia Computer Science**, [S.L.], v. 111, p. 2–7, 2017.

PALMA, D. N. **Camada de transporte modelo TC/IP.** Orientador(a): Jéssica Lopes. 2018. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) — Faculdade Anhanguera, Sorocaba, 2018. Disponível em: https://repositorio.pgsscogna.com.br/bitstream/123456789/35932/1/DANIEL%20NOVAES%20PALMA_1493703_assignsubmission_file_DANIEL_N_PALMA_ATIVIDADE4.pdf. Acesso em: 7 mai. 2025.

PAULA, I. O. de; SALVIANO, L. O. Integration of a power thermal plant under operation to a biomass gasification system: An approach through sensitivity analysis and optimization procedure. **Energy Conversion and Management**, [S.L.], v. 325, 2025.

PELIVANI, E. A comparative study of automation testing tools for web applications. **2021 10th Mediterranean Conference on Embedded Computing (MECO)**, [S.L.], p. 1–6, 2021.

PORTSWIGGER. **Burp Suite - Application Security Testing Software - PortSwigger**. PortSwigger, 2025. Disponível em: https://portswigger.net/burp. Acesso em: 7 mai. 2025.

PYTHON. **History and License**. 2025. Disponível em: https://docs.python.org/3/license.html. Acesso em: 7 mai. 2025.

RIBEIRO, B. Modeling and predicting the growth and death of membership-based websites. **Proceedings of the 23rd international conference on World wide web**, [S.L.], p. 653–664, 2014.

ROCHA, C. A. d. S. **Análise de desempenho em ambientes cliente/servidor 2-camadas e 3-camadas** Orientador(a): Paulo José de Freitas Filho. 2002. Dissertação (Mestrado em Ciência da Computação) — Centro Tecnológico, Universidade Federal de Santa Catarina, Santa Catarina, 2002.

SYMTECH. **The Future of Process Simulation**. Symtech, 2025. Disponível em: https://about.ipsego.app/. Acesso em: 7 mai. 2025.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. São Paulo: Pearson Education, 2015.

TEXAS, U. of N. **About Unique Identifiers - University Libraries - UNT**. 2025. Disponível em: https://library.unt.edu/digital-projects-unit/partners/unique-identifiers/. Acesso em: 7 mai. 2025.

ZINEDDINE, A. A systematic review of cybersecurity assessment methods for https. **Computers and Electrical Engineering**, [S.L.], v. 115, 2024.