UNIVERSIDADE FEDERAL DO CEARÁ

***CAMPUS* SOBRAL**

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

RAÍSSA ELLEN DE SOUSA

UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS DE AUTOMAÇÃO DE TESTES: SELENIUM E CYPRESS

SOBRAL

2023

RAÍSSA ELLEN DE SOUSA

UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS DE AUTOMAÇÃO DE TESTES: SELENIUM E CYPRESS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação do *Campus* Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia da Computação.

Orientador: Prof. Dr. Fischer Jônatas Ferreira.

SOBRAL

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S698e Sousa, Raíssa Ellen de.
UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS DE AUTOMAÇÃO DE TESTES: SELENIUM E CYPRESS / Raíssa Ellen de Sousa. – 2023.
89 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2023.
Orientação: Prof. Dr. Fischer Jônatas Ferreira..

1. Teste de software. 2. Automação de Testes. 3. Cypress. 4. Selenium. 5. Qualidade de software. I. Título.
CDD 621.39

RAÍSSA ELLEN DE SOUSA

UM ESTUDO COMPARATIVO ENTRE FERRAMENTAS DE AUTOMAÇÃO DE TESTES: SELENIUM E CYPRESS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação do *Campus* Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia da Computação.

Aprovada em: 12 de Julho de 2023

BANCA EXAMINADORA

Prof. Dr. Fischer Jônatas Ferreira (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Iális Cavalcante de Paula Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Wendley Souza da Silva
Universidade Federal do Ceará (UFC)

Dedico este trabalho, primeiramente Àquele que me mantém de pé todos os dias, Deus. Às minhas avós: Maria Lúcia (*in memoriam*) e Maria Paula (*in memoriam*). A minha família, meus pais e minha irmã, os grandes incentivadores dos meus sonhos.

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus, fonte de toda minha força e determinação, pois sem Ele e sua graça não teria conseguido alcançar minha posição.

Aos meus familiares, pelo apoio incondicional, inspiração e encorajamento. Agradeço à minha mãe, Maria Inês, por sempre ser o meu pilar. Sei o quanto foi difícil me ver sair do "ninho", obrigada por seus preciosos conselhos e todo amor. Minha gratidão, também, ao meu pai, José Valmir, por todo o esforço e ajuda que me incentivaram a não desistir, e por ser o maior entusiasta dessa fase da minha vida. E a minha irmã, Tainara Kelly, por me motivar a ser melhor todos os dias. Obrigada por acreditarem em mim quando até eu mesma duvidava.

Agradeço, especialmente, ao meu orientador Prof. Fischer Jônatas Ferreira por todo o conhecimento compartilhado e pela paciência durante a orientação acadêmica. Seu profissionalismo, comprometimento e gentileza foram uma inspiração para mim. Esse período sob sua orientação contribuiu para o meu crescimento pessoal e profissional. Obrigada por tudo.

Agradeço de modo geral aos professores do curso de Engenharia da Computação pelo comprometimento com o aprendizado. Essa conquista não seria possível sem vocês.

Aos membros da banca examinadora, Prof. Dr. Iális Cavalcante e Prof. Dr. Wendley Silva, meus sinceros agradecimentos por terem aceitado o convite e por toda a compreensão.

Agradeço também, ao meu namorado, Israel Andrade, por todo seu companheirismo, apoio incondicional e incentivo durante todos esses anos. Obrigada pela sua presença nesta etapa tão importante da minha vida meu amor.

As minhas colegas de quarto, Beatriz Dias, Mikaelly Costa e Ívina Araújo por me proporcionarem a sensação de estar em casa, mesmo estando a quilômetros dela. A companhia e a amizade de vocês foram essenciais para a minha adaptação e a largada dessa trajetória.

Agradeço em especial aos meus amigos, João Gomes e Fabrício Moura, por permanecerem comigo em todos os momentos, pela irmandade construída e por terem me ajudado a acreditar que eu seria capaz de tanto. Obrigada, parte dessa conquista também é de vocês.

Agradeço também, à minha “panelinha", Laiana, Juliana, Willian, Carlos Eduardo e Carlos Roberto por todos os momentos partilhados durante essa jornada, pelas noites de estudo e as noites de pizzas. Obrigada pelo apoio e suporte durante todo o curso.

"Eu posso não estar onde gostaria de estar, mas estou feliz por saber que estou a caminho."
(Joyce Meyer)

RESUMO

O desenvolvimento de um *software* pode ser um processo complexo e propenso a apresentar problemas que afetem o seu funcionamento. Para alcançar a qualidade desejada e minimizar a ocorrência destes problemas é imprescindível que o *software* seja submetido a testes. O teste de *software* é um processo integral e contínuo no ciclo de vida de desenvolvimento, onde *bugs*, erros e vulnerabilidades de uma aplicação podem ser identificados previamente e corrigidos. Nesse contexto, a área de automação de testes vem se destacando, pois proporciona agilidade e eficiência aos processos de testes. Diversas ferramentas são utilizadas para auxiliar nesta atividade de automação. Dentre elas se destacam o `Selenium`, uma ferramenta já consolidada no mercado, e o `Cypress`, uma solução mais recente. Ambas são referência na área da automação de testes para aplicações Web. Entretanto, não existem estudos que apontem um referencial comparativo completo quanto à performance e aplicação de cada uma destas ferramentas. Logo, este trabalho realiza uma avaliação entre `Selenium` e `Cypress` quanto às suas aplicações, recursos, vantagens, desvantagens e outras métricas de desempenho. Dessa forma, objetiva-se orientar a escolha de cada uma destas ferramentas em diferentes aplicações e sistemas. Para isto, fora realizada uma revisão da literatura acerca das ferramentas em estudo. Além disso, foi realizado um estudo empírico com a implementação e execução de *scripts* de testes em uma aplicação Web, para promover uma comparação prática e obter resultados dos aspectos funcionais de ambas as ferramentas. Como resultado desta pesquisa, podemos concluir alguns pontos acerca do comparativo realizado. Em um contexto geral, embora o `Selenium` apresente muitos diferenciais quanto a liberdade de uso em ambientes, ferramentas e linguagens, também apresenta problemas de complexidade na configuração e uma arquitetura propensa a diminuir seu desempenho. Já o `Cypress`, apesar de não ser tão adaptável quanto o `Selenium`, possui alguns recursos que se destacam em termos de escrita de testes, configuração e depuração. Os resultados deste estudo serão úteis para desenvolvedores, testadores e pesquisadores para determinar um comparativo entre uma ferramenta bastante utilizada (`Selenium`) e uma nova (`Cypress`) com um futuro promissor na área de automação de testes para aplicações dinâmicas na Web. Contribuindo para uma escolha, segundo seus recursos, em ambientes corporativos e educacionais.

Palavras-chave: Teste de *software*, Automação de Testes, Selenium, Cypress

ABSTRACT

The development of software can be a complex process and prone to problems that affect its operation. In order to achieve the desired quality and minimize the occurrence of these problems, the software must be submitted for tests. Therefore, software testing is an integral and continuous process in the development lifecycle, where an application's bugs, errors, and vulnerabilities can be previously identified and corrected. In this context, the test automation area has stood out, providing agility and efficiency to the testing processes. Several tools are used to assist in this automation activity. Among them, we highlight Selenium, a tool already consolidated in the market, and Cypress, a more recent solution. Both are references in the area of test automation for Web applications. However, studies need to point to a complete comparative reference regarding the performance and application of each of these tools. This work evaluates Selenium and Cypress regarding their applications, features, advantages, disadvantages, and other performance metrics. Therefore, the objective is to guide the choice of these tools in different applications and systems. For this, Ad hoc research was carried out on the tools under study. In addition, we build the implementation and execution of test scripts in a Web application to promote a practical comparison and obtain results of the functional aspects of both tools. As a previous result of this research, we can conclude that both Selenium and Cypress have advantages and disadvantages concerning each other. In general, although Selenium presents many differentials in terms of freedom of use in environments, tools, and languages, it also presents problems of complexity in configuration and an architecture prone to decreasing its performance. On the other hand, despite not being as adaptable as Selenium, Cypress has some features that stand out in writing tests, configuration, and debugging. The results of this study will help developers, testers, and researchers to compare a widely used tool and a new one with a promising future in the area of test automation for dynamic applications on the Web. This work contributed to choosing corporate and educational environments according to resources.

Keywords: Software Testing, Test Automation, Selenium, Cypress

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RUP	*Rational Unified Process*
SUT	*System Under Test*
BLANK	*Blank Lines of Code*
CI/CD	*Continuous Integration/Continuous Delivery*
CLOC	*Comment Lines of Code*
CMM	*Capability Maturity Model*
DOM	*Document Object Mode*l
IDE	*Integrated Development Environment*
ISTQB	*International Software Testing Qualifications Board*
NPM	*Node Package Manager*
PLOC	*Physical Lines of Code*
SLOC	*Source Lines of Code*
SO	Sistema Operacional
TIC	Tecnologias da Informação e da Comunicação
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Nas últimas décadas, os contínuos avanços da indústria computacional e a grande difusão das Tecnologias da Informação e da Comunicação (TIC), acarretaram em uma transformação significativa na sociedade (BAHRINI; QAFFAS, 2019). A tecnologia está cada vez mais presente na vida cotidiana das pessoas, seja em aplicativos, sites ou programas de computadores. Diante disto, as exigências estabelecidas para a criação de um *software* confiável, intuitivo, menos suscetível à falhas e eficiente quanto aos seus requisitos funcionais tem aumentado. Para (PRESSMAN; MAXIM, 2021), conforme cresce a importância do *software* e a exigência por qualidade, existe uma preocupação para se desenvolver abordagens para melhorar o seu desenvolvimento e produção. Consequentemente, nos últimos anos o interesse pela atividade de teste de *software* vem aumentando (BARBOSA *et al.*, 2000).

Em (BARTIÉ, 2002), é definido qualidade de *software* como "um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando defeitos". Logo, o processo de planejamento dos testes deve ocorrer em diferentes níveis e em paralelo ao processo de desenvolvimento de um *software*. O que pode significar melhorias na qualidade em geral do projeto, reduzindo custos, evitando retrabalhos, contribuindo com a segurança e a confiabilidade, como também, antecipando a descoberta de falhas e incompatibilidades.

Diversas áreas da nossa sociedade, desde negócios, educação, até setores de entretenimento e vida pessoal foram impactadas pela utilização da Web (DOĞAN *et al.*, 2014). Segundo (KAPPEL *et al.*, 2004), aplicações Web são sistemas de *software* baseados em tecnologias e padrões do *World Wide Web Consortium* (W3C)[1] que detém, através de uma interface de usuário, recursos específicos da Web. Para (MANSOUR; HOURI, 2006), aplicativos Web são sistemas complexos, em constante evolução e atualização. Logo, com aumento do uso de tais sistemas, a confiabilidade desse tipo de *software* tornou-se cada vez mais importante (DOĞAN *et al.*, 2014). Portanto, justificando a importância da área de testes aplicada em tais aplicações atualmente.

A execução de testes em um projeto de *software* pode ser dividida em níveis, tipos e técnicas. Em (ROCHA, 2001), podemos encontrar a definição dos principais níveis de teste de *software*, são eles: testes de unidade, de integração, de sistema e de aceitação. Assim, em

[1] O W3C desenvolve padrões e diretrizes para auxiliar na construção de uma Web baseada nos princípios de acessibilidade, internacionalização, privacidade e segurança (https://www.w3.org/).

cada nível podemos executar diferentes tipos de testes, como testes de regressão, de desempenho e outros. Além disso, a execução dos mesmos pode ser realizada de maneira manual ou automatizada. Os testes manuais consistem na interação manual por parte de um colaborador com as funcionalidades do projeto. Já, os testes automatizados utilizam ferramentas e tecnologias para automação deste processo. Considerando que a execução manual de um teste pode representar uma atividade redundante, sujeita à falhas humanas e apresentar um custo elevado de tempo, a automação passou a ser vista como uma alternativa atraente para a aplicação de testes de *software*.

A automação de testes de *software* refere-se a prática de revisar e validar automaticamente um produto de *software*, podendo ser aplicada em diferentes níveis de testes. Para (BARTIÉ, 2002), os testes automatizados são definidos como "[...] a utilização de ferramentas de testes que possibilitem simular usuários ou atividades humanas de forma a não requerer procedimentos manuais no processo de execução dos testes". Logo, a automatização de testes pode significar a redução de custo e tempo ao longo do ciclo de vida de um *software*, além de otimizar processos de entrega e validação. Entretanto, é evidente que existe um esforço inicial para definir e consolidar a automatização dos testes em um sistema. Segundo (BARTIÉ, 2002), uma das motivações para seu uso é que "a automação exige um esforço inicial de criação, porém possibilita uma incomparável eficiência e confiabilidade, impossível de ser atingida com procedimentos manuais".

Dessa forma, o presente trabalho propõe abordar um estudo comparativo entre duas ferramentas utilizadas para a realização de testes automatizados em aplicações Web atualmente, o `Selenium` e `Cypress`. O `Selenium` é uma das ferramentas de automação de testes mais populares. De acordo com (MOBARAYA *et al.*, 2019), a ferramenta `Selenium` é considerada confiável e robusta devido à sua longa existência no mercado. Já o `Cypress`, surge como uma solução relativamente nova e promissora para atuar na automatização. Ambas as ferramentas são utilizadas principalmente em testes de aplicações Web, onde os *scripts* de testes interagem de forma automatizada simulando a utilização e o comportamento de sistemas. Portanto, o objetivo geral deste estudo é apresentar as ferramentas `Selenium` e `Cypress`, realizar um levantamento completo de suas características e funcionalidades, bem como, exemplificar a aplicação de ambas ao implementar testes funcionais automatizados em um sistema Web.

Durante este trabalho foi observado que a ferramenta `Selenium` apresenta mais possibilidades para sua utilização, uma vez que, a mesma oferece suporte para a execução em

diferentes ambientes. Dessa forma, o `Selenium` pode ser usado em uma grande variedade de sistemas operacionais e navegadores, além de oferecer uma maior liberdade para a escolha da linguagem de programação dos testes. Entretanto, apresenta algumas desvantagens quanto à complexidade para configuração e características de sua arquitetura e recursos, podendo impactar no desempenho da execução dos testes. Por sua vez, o `Cypress` apresenta algumas limitações relacionadas à execução em diferentes ambientes, limitando as suas possibilidades, principalmente em relação ao suporte às linguagens de programação. Por outro lado, a ferramenta apresenta uma série de recursos interessantes para a aplicação de testes na Web, em termos de escrita, configuração, depuração e execução. Além de apresentar uma arquitetura simplificada e um processo de configuração facilitado.

A principal contribuição deste estudo consiste em criar um embasamento teórico e prático para guiar a escolha de tais ferramentas para uso de empresas e desenvolvedores de *software* que buscam melhorar a qualidade dos seus produtos. Assim como, apresentar métricas e aspectos para justificar o uso do `Selenium` ou `Cypress` dentro da realidade de pesquisadores, estudantes e profissionais da área.

1.1 Justificativa

A etapa de testes é muito importante no processo de desenvolvimento de um *software*. Atualmente, existem diversas ferramentas usadas para a automatização de testes de *software* no mercado, cada uma com características e recursos específicos. Ao avaliar ou pesquisar uma ferramenta de automatização de testes, é importante elencar recursos e requisitos que precisam ser atendidos. Não realizar este comparativo pode implicar em um custo de tempo, pois será necessário um esforço para configurar, instalar e aprender uma ferramenta que pode não atender aos requisitos específicos que buscamos nos testes. Além disso, podem existir outras alternativas de ferramentas no mercado que atendam aos critérios almejados. Este estudo avalia duas principais ferramentas para automatização de testes em aplicações Web, o `Selenium` e o `Cypress`, realizando um comparativo em termos de recursos das ferramentas, execução de testes, e entre outras métricas. Assim espera-se contribuir para a análise e escolha das ferramentas de teste em estudo.

1.2 Objetivos

Nesta seção, são descritos os objetivos gerais e específicos deste estudo.

1.2.1 Objetivos Gerais

O objetivo geral deste trabalho é analisar e comparar as ferramentas de automação de testes `Selenium` e `Cypress`. Ressaltando as principais diferenças, recursos disponíveis, arquiteturas e vantagens na utilização de ambas as ferramentas. Assim como, relatar o aprendizado e resultados obtidos ao utilizar as ferramentas para realizar a automação de testes em uma aplicação Web.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Descrever as ferramentas de testes em estudo, `Selenium` e `Cypress`;
- Realizar uma revisão na literatura (ad hoc) das características das ferramentas em estudo;
- Implementar testes automatizados em uma aplicação Web utilizando as ferramentas `Selenium` e `Cypress`;
- Relatar resultados, para que por fim seja realizada uma comparação entre as ferramentas em relação às suas características e resultados de um estudo empírico.

1.3 Organização do Trabalho

A organização deste trabalho consiste na divisão e apresentação dos Capítulos 2, 3, 4, 5 e 6. No Capítulo 2, são apresentados os termos, conceitos e exemplos necessários para a compreensão da temática deste trabalho. Abordando a definição de teste de *software* e detalhando nomenclaturas e principais técnicas de teste, além de contextualizar a automação de testes de *software* e apresentar previamente as ferramentas `Selenium` e `Cypress`. No Capítulo 3, a metodologia que foi utilizada na realização deste estudo é detalhada. Já no Capítulo 4, são apresentados os resultados e discussões acerca das descobertas e implicações. No Capítulo 5, são apresentados alguns trabalhos relacionados, enfatizando suas contribuições e ressaltando semelhanças e diferenças quanto ao trabalho aqui proposto. Por fim, no Capítulo 6 são apresentados as conclusões e discussões acerca de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece embasamento para os assuntos relacionados ao contexto teórico envolvido na proposta deste trabalho. Portanto, tem como objetivo elucidar os seguintes tópicos: teste de *software* (Seção 2.1), tipos de testes de *software* (Seção 2.2), automação de testes (Seção 2.3), bem como apresentar as ferramentas de automação `Selenium` (Seção 2.4) e `Cypress` (Seção 2.5).

2.1 Teste de *software*

Teste de *software* pode ser definido como um processo, ou uma série de processos, realizados com a finalidade de garantir que o *software* atenda as funcionalidades que foi projetado para executar (MYERS *et al.*, 2011). Segundo (NETO; CLAUDIO, 2007), testar um *software* "significa verificar através de uma execução controlada se o seu comportamento corre de acordo com o especificado". Para (PRESSMAN; MAXIM, 2021), "teste de *software* é um elemento crítico da garantia de qualidade de *software* e representa a revisão final da especificação, projeto e geração de código". Assim, ao realizar os testes em um *software* são analisados itens e recursos para detectar possíveis diferenças entre as condições existentes e necessárias (IEEE, 1990).

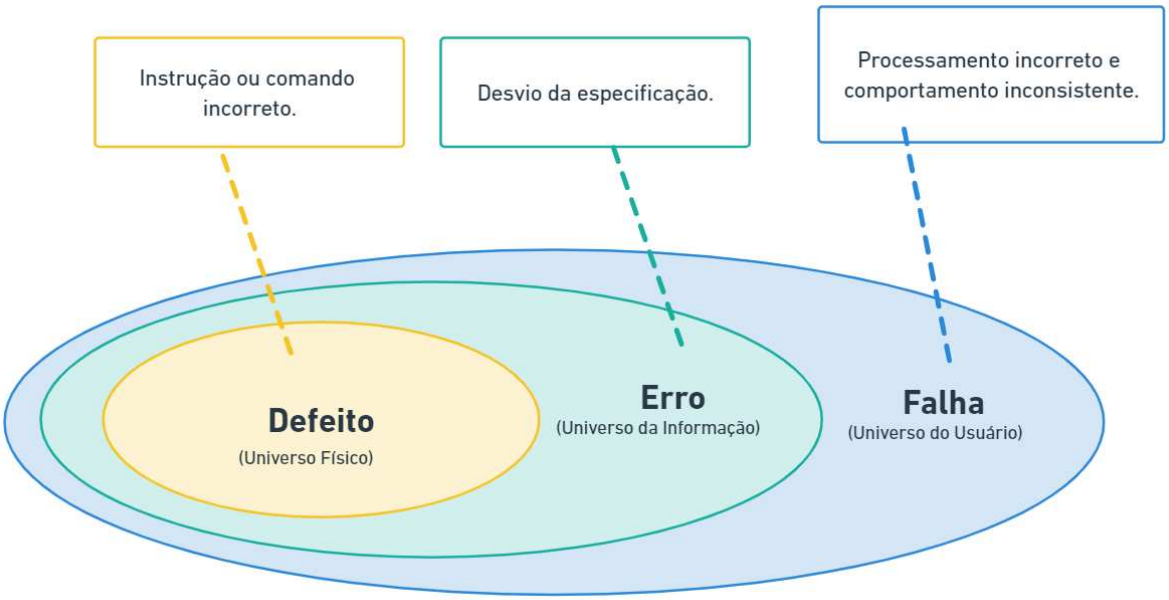
Para compreender melhor a atividade de teste de *software* e estabelecer uma linguagem comum para debater a temática, é importante definir alguns conceitos. Na literatura tradicional são estabelecidos alguns destes termos como: defeito, erro, falha e engano (DELAMARO *et al.*, 2017). As definições utilizados neste trabalho estão de acordo com o documento *"Standard Glossary of Terms used in Software Testing"* (BOARD, 2014), referência do *International Software Testing Qualifications Board* (ISTQB)[2] e a terminologia padrão para Engenharia de *Software* do *Institute of Electrical and Electronics Engeneers* (IEEE, 1990).

- **Defeito** (do inglês, *fault*), pode ser conceituado como um passo ou definição de dados incorretos (DELAMARO *et al.*, 2017). Pode ser encontrado durante a execução do *software*, podendo causar falhas em componentes ou sistema (BOARD, 2014).
- **Erro** (do inglês, *error*), pode ser caracterizado como um estado inconsistente ou inesperado, ocasionado pela manifestação concreta de um defeito durante a execução de um programa (DELAMARO *et al.*, 2017). Ou seja, qualquer estado intermediário incorreto ou resultado inesperado (IEEE, 1990).

[2] O ISTQB®(*International Software Testing Qualifications Board*) é uma associação sem fins lucrativos, que se tornou líder mundial na certificação de competências em testes de *software* (https://www.istqb.org/).

– **Falha** (do inglês, *failure*), é quando o comportamento produzido pelo *software* difere do resultado esperado (DELAMARO *et al.*, 2017). Uma falha pode ter sido causada por diversos erros e alguns erros podem nunca causar falhas (IEEE, 1990).
– **Engano** (do inglês, *mistake*), pode ser definido como uma ação humana que produz um defeito (DELAMARO *et al.*, 2017), um resultado incorreto (IEEE, 1990).

Figura 1 – Diferença entre os conceitos de defeito, erro e falha



Fonte: Adaptada de (NETO; CLAUDIO, 2007).

A Figura 1 expressa a diferença entre esses conceitos. Os defeitos estão relacionados à aplicação propriamente dita, e podem ser causados por usuários, por exemplo, através do mal uso de uma tecnologia. Assim, os mesmos podem refletir no surgimento de erros em um produto, ou seja, a construção de um *software* de forma diferente ao que foi especificado. Por fim, os erros podem ocasionar em falhas, que consistem em comportamentos inesperados que podem inviabilizar a utilização de uma aplicação, afetando diretamente o usuário final. O foco deste trabalho é em falhas, já que estas são detectadas através do comportamento do sistema. Desvios na especificação (erros) e erros no código (defeitos) não fazem parte do escopo deste trabalho.

Os testes geralmente são aplicados em diferentes níveis, e devem ser considerados em paralelo ao desenvolvimento do *software*. Geralmente na literatura são definidos os principais níveis de teste de *software*, são eles: teste de unidade, teste de integração, teste de sistema e teste de aceitação. A Tabela 1 apresenta um comparativo entre tais níveis de testes, considerando determinados critérios. Assim, na primeira linha consideramos a finalidade de cada um dos tipos

de testes, por exemplo, os testes de unidade tem a finalidade de garantir o funcionamento correto de cada unidade.

Tabela 1 – Níveis de testes de *software*

Critério	Unidade	Integração	Sistema	Aceitação
Finalidade	O funcionamento correto da unidade /módulo	O funcionamento correto das unidades integradas	Todo o sistema funciona bem quando integrado	As expectativas do cliente são atendidas
Foco	Menor parte testável	Interface e interação de módulos	Interação e funciona - mento de todos os módulos como um	*Software* funcionando de acordo com as especificações fornecidas
Tempo de teste	Uma vez que um novo código é escrito	Uma vez que novos componentes são adicionados	Assim que o *software* estiver completo	Uma vez que o *software* é lido operacional- mente
Executado por	Desenvolvedor	Equipe de desenvolvimento.	Equipe de teste	A equipe de desenvolvi- mento e os usuários finais
Automação	Automatizável	Automatizável	Automatizável	Automatizável

Fonte: Adaptada de (UMAR, 2019).

O **Teste de Unidade**, também conhecido como testes unitários, enfatiza a unidade individual ou módulo isoladamente. Durante o testes são verificadas possíveis falhas ocasionadas por problemas de lógica e/ou de implementação em cada módulo separadamente. O **Teste de Integração** consiste em analisar unidades combinadas da estrutura do *software* e a integração geral do projeto. Já o **Teste de Sistema** consiste em testar o *software* completo integrado com o intuito de verificar sua conformidade com seus requisitos e a interação geral dos componentes, garantindo assim o funcionamento unânime de todos os módulos e programas sem erros. E por fim, o **Teste de Aceitação** enfatiza a validação do *software* em relação aos requisitos do cliente e sua aceitabilidade.

2.2 Tipos de teste de *software*

Para evitar o surgimento de problemas durante o ciclo de vida de desenvolvimento de um *software* são empregados diferentes tipos de testes, uma vez que, o objetivo principal ao desenvolver um *software* é garantir a qualidade em todas as suas camadas. Existem muitos tipos de testes, cada um propósito distinto, por exemplo, testes de funcionalidades, testes de performance, testes de usabilidade, de regressão e entre outros (VALENTE, 2020).

Nos testes funcionais, os requisitos, requerimentos e regras de negócio da aplicação são testados, para validar as funcionalidades documentadas do *software* (BARBOSA; TORRES, 2011). O teste de performance ou desempenho, pode ser definido como um processo que avalia se os requisitos de performance do sistema são atendidos, considerando um volume de acessos/transações dentro do esperado e avaliando métricas como o tempo de resposta (SOUZA; GASPAROTTO, 2013). Já os testes de usabilidade são desempenhados pela perspectiva do usuário final, determinando se a aplicação é intuitiva e fácil para o uso do público-alvo. O objetivo desse tipo de teste é verificar a usabilidade da interface do sistema (VALENTE, 2020). Os testes de regressão consistem na execução de testes que anteriormente já foram executados com sucesso, verificando se as alterações na aplicação introduziram novos *bugs* e analisando se os resultados correspondem ao esperado (SOMMERVILLE, 2011). Esse tipo de teste é realizado a cada nova alteração ou atualização do sistema (BARBOSA; TORRES, 2011).

2.3 Automação de testes

A execução de testes é uma das atividades que buscam contribuir para a melhoria da qualidade do *software* (BARBOSA *et al.*, 2000). Logo, são atividades importantes desde o levantamento de requisitos, definição de arquitetura, codificação e, principalmente, nos momentos de

integração de componentes do sistema e validação das funcionalidades existentes. A importância do uso de teste automático de *software* é definida por (DUARTE *et al.*, 2010) como ferramenta essencial ao desenvolvimento de aplicações de grande porte. Logo, a automatização dos testes de *software* pode se tornar essencial quando se deseja executar todos os testes em um sistema de maneira otimizada, simples e a qualquer momento (BARTIÉ, 2002).

A automação de testes pode ser definida como o uso de ferramentas, *softwares* e *scripts* que possibilitam a execução automática de testes. Com isto, os testes são realizados de forma controlada comparando os resultados esperados com os resultados reais obtidos. Segundo Watts S. Humphrey, criador do *Capability Maturity Model* (CMM)[3], a qualidade do produto final é diretamente proporcional à qualidade do processo utilizado no seu ciclo de vida. Logo, a eficiência da implantação da automação de testes depende de processos, durante todo o ciclo de desenvolvimento, consistentes e bem definidos. Assim, a automação de testes pode proporcionar uma maior segurança para a realização de mudanças em uma aplicação, sejam elas de refatoração, manutenção ou desenvolvimento de novas funcionalidades, pois realiza execução dos testes de forma rápida, reduzindo o esforço e possibilitando a reprodução (MOLINARI, 2008).

Devido ao crescimento da área de testes de *softwares*, cada vez mais são discutidas a utilização de ferramentas que auxiliem o trabalho dos profissionais da área em tarefas como o planejamento de casos de testes, execução de testes, abertura de *bugs* entre outros (PATUCI, 2011). Existem diversas ferramentas que podem ser utilizadas para realizar a automatização de testes de *software*. Com o objetivo de apresentar a estratégia de automação de testes utilizada neste trabalho, as ferramentas em uso serão apresentadas detalhadamente.

[3] O CMM - *Capability Maturity Model* é um modelo para avaliação da maturidade dos processos de *software* de uma organização.

2.4 SELENIUM

O `Selenium`[4] consiste em um conjunto de ferramentas utilizadas para automatizar testes de forma prática e eficiente (Selenium, 2023). Criado em 2004 por Jason Huggins, foi idealizado para otimizar o processo de testes de um projeto interno na empresa ThougthWork[5]. Desenvolvido como uma biblioteca em JavaScript com a finalidade de interagir com a aplicação e executar os testes em navegadores, a ferramenta começou a ser difundida dentro da empresa. Logo, o `Selenium` recebeu destaque pelo seu desempenho, facilidade e potencial de crescimento como uma estrutura de teste reutilizável para outros aplicativos da Web. O `Selenium` causou grande impacto na época, e com o crescimento na área de testes automatizados, essa biblioteca se transformou mais tarde no núcleo das ferramentas `Selenium`.

O `Selenium` permite realizar operações altamente adaptáveis, possibilitando encontrar elementos da interface, manipulá-los, validar resultados, preencher valores e simular com precisão o comportamento de um usuário ao navegar por algum sistema Web. Um recente estudo sobre testes de *software* indica o `Selenium` como uma das principais ferramentas de testes da atualidade, seguido por JUnit e Cucumber (CERIOLI *et al.*, 2020). Essa popularidade pode ser justificada por algumas características que se apresentam como vantagens importantes da ferramenta, como por exemplo, o código-fonte aberto, flexibilidade e suporte quanto a diferentes sistemas operacionais e linguagens de programação. Além disso, permite a interação com distintos navegadores como o Chrome, Firefox, Edge ou Opera de forma automatizada.

Segundo o estudo realizado por (VELOSO *et al.*, 2010), o `Selenium` é uma das ferramentas mais adequadas para a realização de testes funcionais automatizados. Além disso, muitas outras ferramentas se baseiam no `Selenium`, como mostrou os estudos de (FOWLER, 2018).

[4] https://www.selenium.dev/
[5] https://www.thoughtworks.com/

As ferramentas que compõem o Selenium são: o Selenium IDE (do inglês *Integrated Development Environment*), o Selenium RC (do inglês *Remote Control*), o Selenium WebDriver e o Selenium Grid, onde cada uma possui uma abordagem distinta e se concentra em atender necessidades específicas.

2.4.1 Selenium IDE

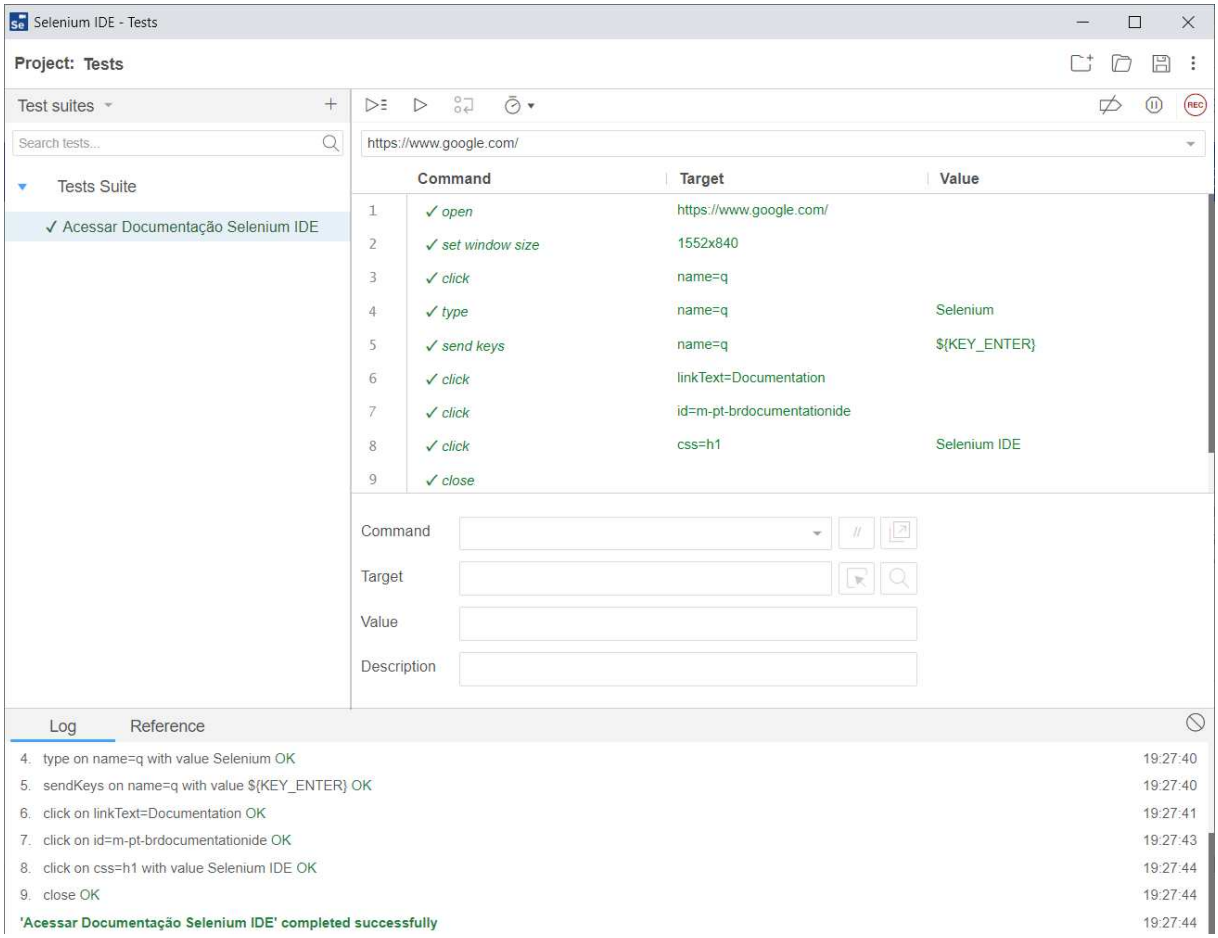
O Selenium IDE é uma ferramenta usada para construir *scripts* de testes automatizados através de um *plugin* para navegadores. A ferramenta fornece uma interface amigável e possui uma abordagem *record-and-playback*, ou seja, permite capturar as ações executadas pelo testador ou usuário em um *script*, e posteriormente reproduzir o caso de teste de forma automatizada. A ferramenta está disponível para os navegadores Google Chrome, Mozilla Firefox e Microsoft Edge e não requer nenhuma configuração adicional além de instalar a extensão no navegador.

O processo de criação de testes com o Selenium IDE pode ser exemplificado com as seguintes etapas: **a) Definir passos:** O utilizador interage diretamente com o sistema seguindo o caso de teste. Para cada ação realizada, a ferramenta adiciona um passo no *script* de testes; **b) Definir critérios de validação:** Com base nos critérios de aceitação do caso de teste em questão, o utilizador seleciona parâmetros e asserções; **c) Executar o *script*:** Por fim deve-se executar o *script* de teste para avaliar se a ferramenta capturou todos os passos do caso de teste e as verificações correspondem aos resultados esperados. Assim, são replicadas as ações realizadas pelo testador durante a gravação do *script*. Ao final da execução, é avaliado o critério de validação, onde será informado o sucesso ou a falha do caso de teste.

Na Figura 2 é exibida a interface do Selenium IDE com a definição de um caso de

teste. O teste exemplificado consiste em abrir o navegador Chrome, pesquisar por "Selenium"e abrir a documentação referente ao `Selenium IDE`. A ferramenta permite gravar os testes, editar, executar e exportar em diferentes linguagens. No Listing 2.4.1, é apresentado o *script* do mesmo teste criado na IDE, exportado para a linguagem JavaScript. Na linha 10 deste *script* é instanciado o *driver* para acessar o navegador; das linhas 16 à 19 é realizada a pesquisa no Google. Em seguida, das linhas 20 à 22 a documentação do `Selenium` é acessada. Por fim, na linha 23 a instância do navegador é fechada.

Figura 2 – Interface do plugin Selenium IDE para o Chrome



Fonte: Elaborada pelo autor.

```
// Generated by Selenium IDE
const { Builder, By, Key, until } = require('selenium-webdriver')
const assert = require('assert')

describe('Acessar Documentação Selenium IDE', function() {
  this.timeout(30000)
  let driver
  let vars
  beforeEach(async function() {
    driver = await new Builder().forBrowser('chrome').build()
  })
  afterEach(async function() {
    await driver.quit();
  })
  it('Acessar Documentação Selenium IDE', async function() {
    await driver.get("https://www.google.com/")
    await driver.findElement(By.name("q")).click()
    await driver.findElement(By.name("q")).sendKeys("Selenium")
    await driver.findElement(By.name("q")).sendKeys(Key.ENTER)
    await driver.findElement(By.linkText("Documentation")).click()
    await driver.findElement(By.id("m-pt-brdocumentationide")).click()
    await driver.findElement(By.css("h1")).click()
    await driver.close()
  })
})
```

Listing 2.4.1 – Teste exportado do Selenium IDE para JavaScript

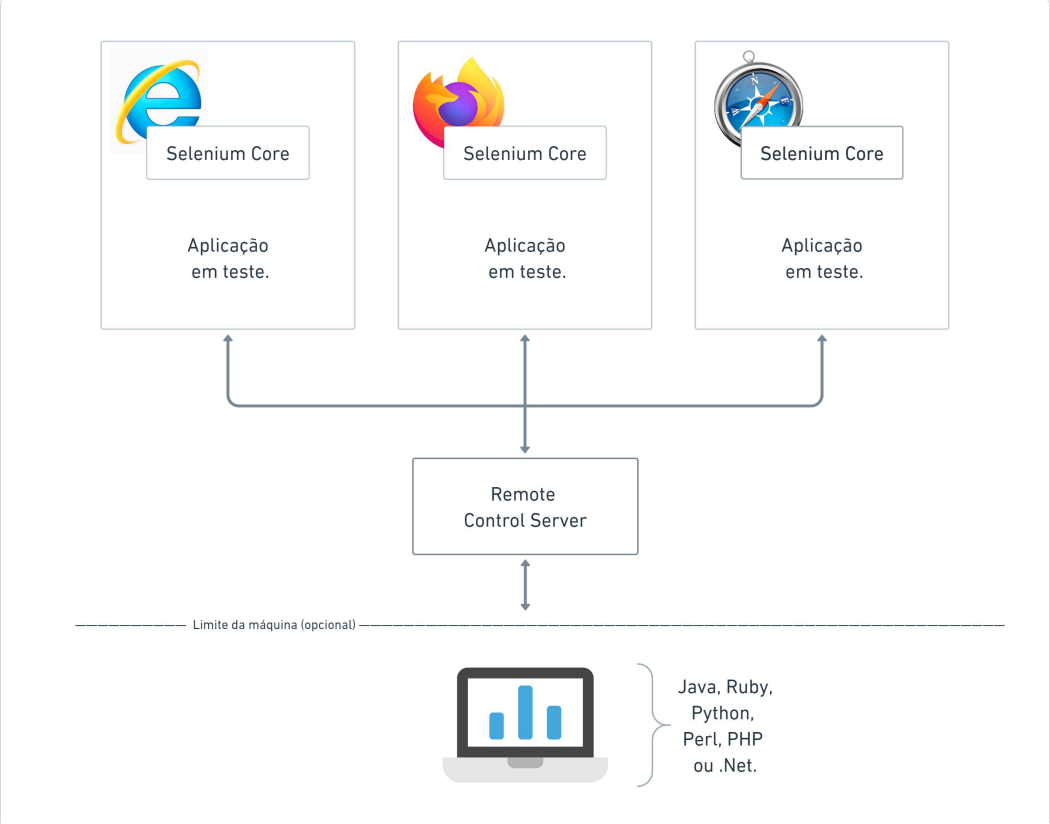
Os *scripts* gerados são construídos usando comandos `Selenium` com parâmetros definidos conforme o contexto de cada elemento. Após a criação destes *scripts*, a ferramenta permite exportá-los para as linguagens de programação suportadas, como por exemplo, JavaScript, Ruby, Java, Python e outras. Permitindo ao testador a possibilidade de aprimorar os *scripts* de testes através da programação na linguagem desejada. Essa é uma importante característica da ferramenta `Selenium IDE`, pois permite contornar as possíveis limitações.

O `Selenium IDE` é considerado por muitas literaturas como uma ferramenta de prototipagem rápida. Recomenda-se utilizar outras soluções do `Selenium` para criação de testes mais robustos que necessitem de recursos mais avançados.

2.4.2 Selenium RC

Segundo (MAHAJAN *et al.*, 2016) o `Selenium RC` é uma solução para testes entre navegadores. Essa ferramenta foi desenvolvida com base em uma biblioteca JavaScript, e possibilita automatizar testes escritos em variadas linguagens de programação. De acordo com (GARCíA *et al.*, 2020), o `Selenium RC` apresenta uma arquitetura cliente-servidor, e dispõe de dois principais componentes: o *Selenium Server* (servidor) e a *Client Librarie* (bibliotecas do cliente). O *Selenium Server* é responsável por inicializar e finalizar os navegadores, como também, interpretar e executar os comandos `Selenium`. Já a *Client Librarie* é responsável por disponibilizar uma interface entre o programa de automação de testes e o servidor.

Figura 3 – Arquitetura do Selenium RC



Fonte: Adaptada de (Selenium, 2023)

A Figura 3 apresenta uma representação da arquitetura do `Selenium RC`. As bibliotecas do cliente são encarregadas de enviar instruções do `Selenium` para o servidor, que

repassam os comandos para o *Selenium Core*, que consiste em uma biblioteca de comandos em JavaScript responsável por executar os comandos dentro do navegador. Logo, tornando possível a automatização das ações sobre a aplicação Web em teste.

Segundo (GARCíA *et al.*, 2020), como a tecnologia base do `Selenium RC` era o JavaScript, existiam alguns recursos que não eram suportados nas interações automatizadas da Web. Como exemplo, *upload* e *download* de arquivos, ou *pop-ups* e manuseio de diálogos. Além disso, uma sobrecarga considerável apresentada pelo servidor `Selenium RC` afeta o desempenho final dos testes.

Embora o `Selenium RC` tenha se tornado um dos principais projetos do `Selenium` por um tempo e tenha feito contribuições significativas no espaço de automação de testes em navegadores, a ferramenta apresenta limitações significativas. Diante disso, em 2008, Simon Stewart da empresa Thinkworks, desenvolveu uma nova ferramenta para automatização de testes com recursos mais poderosos, chamada `Selenium WebDriver` (GARCíA *et al.*, 2020).

2.4.3 Selenium WebDriver

O `Selenium WebDriver` pode ser definido como uma coleção de APIs (*Application Programming Interface*) usadas para automatizar testes em aplicações Web. A ferramenta oferece suporte para os principais navegadores do mercado. Segundo (GONZALEZ *et al.*, 2022), a comunicação da ferramenta é bidirecional: o WebDriver passa comandos para o navegador através do *driver* e recebe informações pela mesma rota.

O `Selenium WebDriver`, também conhecido como `Selenium 2.0`, fornece uma interface de programação mais simples e concisa. Diferentemente do `Selenium RC`, que exige que um servidor específico opere durante a utilização da ferramenta, o mesmo interage dire-

tamente com o navegador para a realização de testes de uma aplicação Web (WARDHAN; MADAN, 2021).

De acordo com (WARDHAN; MADAN, 2021) o `Selenium WebDriver` é uma ferramenta muito útil em navegações avançadas de usuários. Suporta de maneira eficiente as páginas dinâmicas da Web, onde os elementos de uma página podem variar sem que a própria página seja recarregada (ANGMO; SHARMA, 2014). Logo, ao utilizar a ferramenta é possível navegar e realizar testes automatizados em páginas mais complexas.

A ferramenta também suporta a execução em modo *Headless*, um modo de execução para navegadores baseados em Firefox e Chromium. Com este recurso é possível executar *scripts* automatizados no modo sem cabeçalho, ou seja, durante a execução a janela do navegador não ficará visível. Com isto, o `Selenium WebDriver` possibilita a execução de testes automatizados em ambientes baseados em linha de comando, como também facilita a utilização da ferramenta em ambientes de integração contínua.

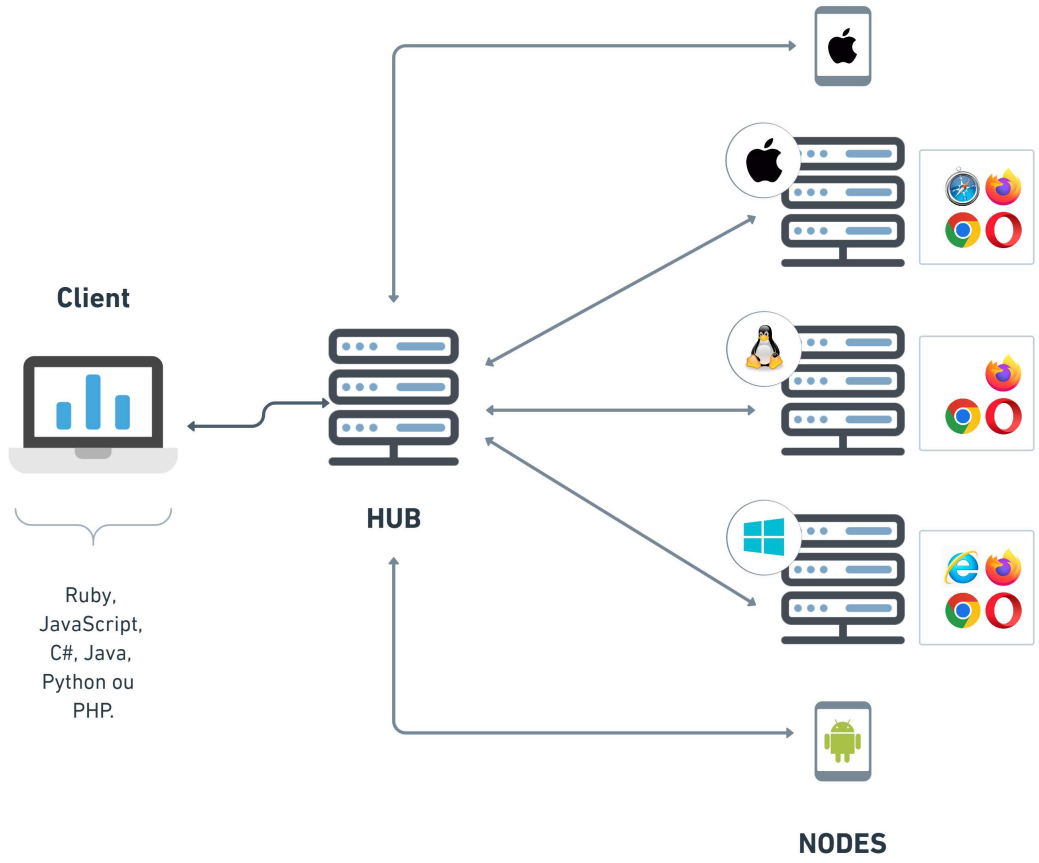
Segundo (RAMYA *et al.*, 2017), o `Selenium WebDriver` é considerado o mais rápido entre todos os componentes do `Selenium`. Um dos fatores que contribuem para tal é o não envolvimento de um servidor. Os comandos fornecidos no código são traduzidos em métodos de serviço da Web, e o *driver* remoto aceita solicitações HTTP e as executa no próprio navegador. Portanto, apresenta ganhos relacionados à eficiência e otimização no tempo de execução dos testes.

2.4.4 Selenium Grid

O `Selenium Grid` é uma ferramenta que possibilita a execução de testes em paralelo em várias máquinas remotas. Suporta a utilização de diferentes tipos e versões de navegadores

e sistemas operacionais. Como também, possibilita executar os *scripts* de testes em várias instâncias do mesmo navegador. Essa paralelização no `Selenium Grid` pode ser um fator importante na otimização do tempo de execução, principalmente, quando existe uma grande quantidade de casos de testes para serem realizados.

Figura 4 – Arquitetura do Selenium Grid



Fonte: Adaptada de (Selenium, 2023)

A arquitetura do `Selenium Grid` pode ser exemplificada em dois principais componentes: um *hub* (ou servidor) e *nodes* (nós ou dispositivos). O *Selenium Hub* é um servidor que rastreia os nós e as solicitações realizadas pelos *scripts* do `Selenium`. Conforme ilustrado na Figura 3, o *hub* recebe instruções do cliente WebDriver e a execução paralela do teste é realizada em uma máquina diferente chamada *nodes* (nós), que pode ser configurada em diferentes navegadores e/ou sistemas operacionais (GARCíA *et al.*, 2020).

2.4.5 Exemplo prático com Selenium

O `Selenium` oferece, por meio do uso do WebDriver, suporte para a automação de testes nos principais navegadores do mercado como Chrome, Firefox, Internet Explorer, Edge e Safari. Para esse exemplo prático será considerado a utilização do `Selenium WebDriver` com o navegador Google Chrome no Windows 10. O exemplo será implementado utilizando a linguagem de programação JavaScript.

2.4.5.1 Preparação do ambiente

Inicialmente para preparar o ambiente para a utilização do `Selenium WebDriver`, será necessário instalar sua biblioteca para a linguagem de programação selecionada, neste caso, JavaScript. A instalação é realizada pelo terminal através do *Node Package Manager* (NPM)[6], o Listing 2.4.2 apresenta o comando de instalação do `Selenium WebDriver`.

```
1 npm install --save selenium-webdriver
```

Listing 2.4.2 – Instalação do Selenium Webdriver com npm

Logo após, será necessário instalar algumas bibliotecas para configurar o ambiente de execução dos testes. Para este exemplo, será necessário instalar os *drives* do navegador e uma estrutura de teste JavaScript, o Mocha[7]. Os Listing 2.4.3 e 2.4.4 exibem os comandos para instalação de tais dependências. Portanto, após esses passos o ambiente já estará pronto para a execução de testes.

```
1 npm install chromedriver
```

Listing 2.4.3 – Instalação do chromedriver com npm

[6] O NPM é uma ferramenta do Node.js para o gerenciamento de pacotes (https://www.npmjs.com/).
[7] https://mochajs.org/

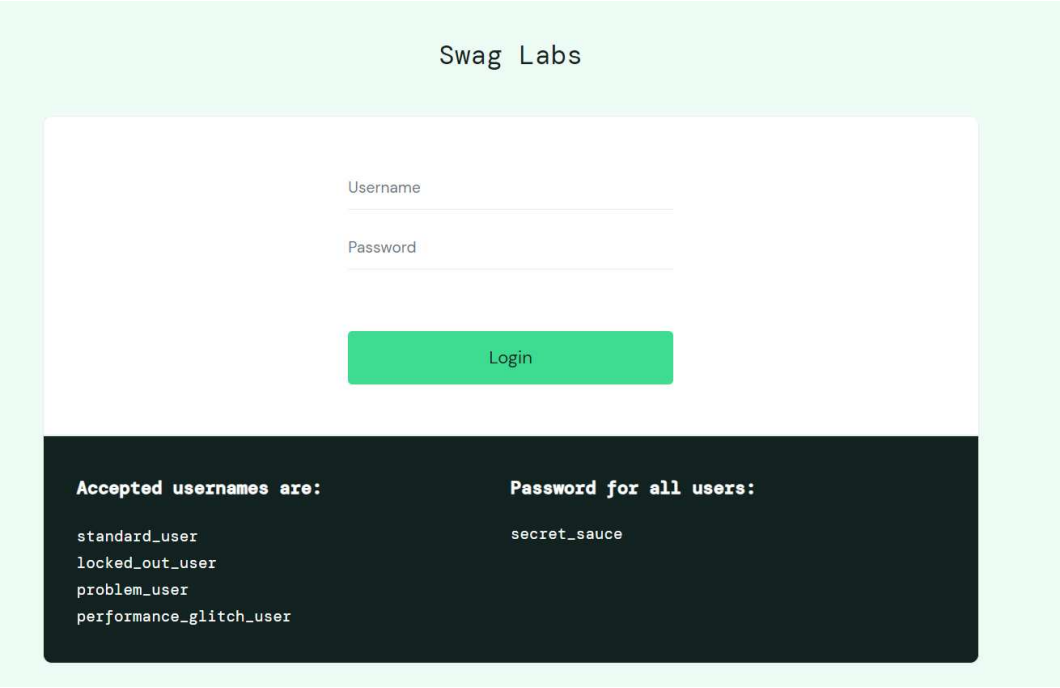
```
1 npm install mocha
```

Listing 2.4.4 – Instalação do mocha com npm

2.4.5.2 Explicação do Exemplo

Para este exemplo prático foi utilizado um site de demonstração para automação de teste de interface do usuário, o SauceDemo[8]. Uma página Web que simula um *e-commerce*. Nas Figuras 5 e 6 são exibidas as telas de login e de produtos da plataforma, respectivamente. A solução fornece algumas opções de perfis usuários para realização dos testes. Neste exemplo, serão utilizadas as credenciais de acesso do *standard_user*, um usuário normal e funcional para simular compras no site normalmente. Os demais perfis apresentam alguns problemas para serem explorados e usados para validação de outros testes, porém, serão desconsiderados durante esse exemplo.

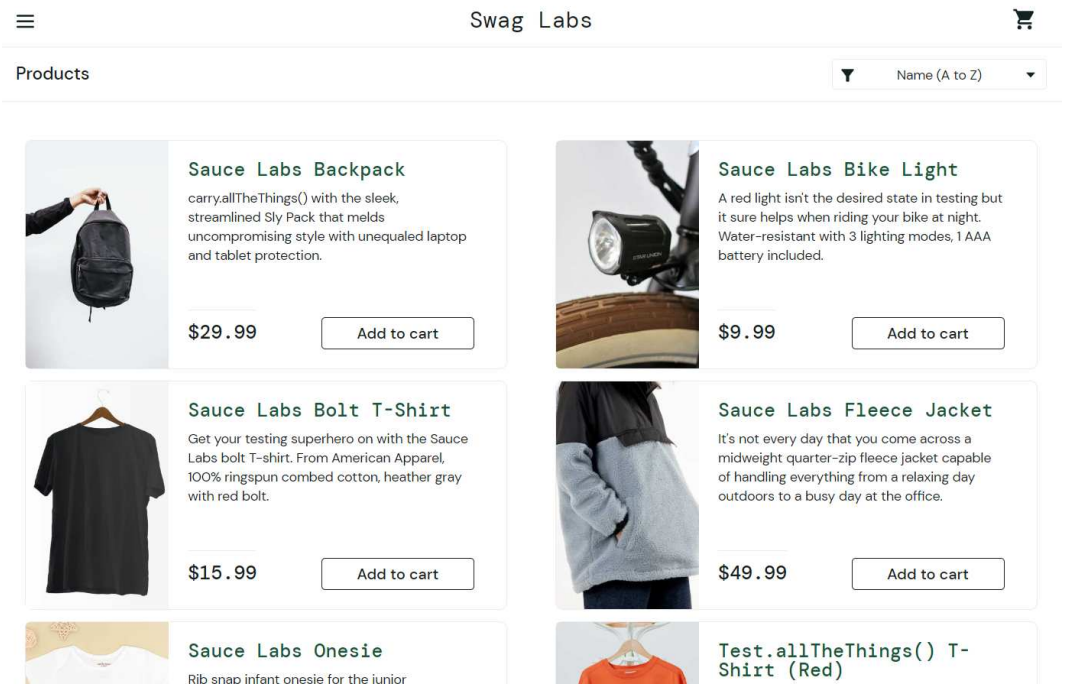
Figura 5 – Interface da tela de login da aplicação SauceDemo



Fonte: Elaborada pelo autor.

[8] https://www.saucedemo.com/

Figura 6 – Interface da tela de produtos da aplicação SauceDemo



Fonte: Elaborada pelo autor.

A Tabela 2 apresenta a especificação completa do caso de teste que será realizado como exemplo prático. O caso de teste consiste em acessar a página de login da aplicação (Figura 5), inserir os dados de acesso (*username* e *password*) do usuário *standard_user* e verificar o sucesso do login ao trocar para a tela inicial com os produtos (Figura 6).

Tabela 2 – Especificação do caso de teste para o exemplo prático

Item de teste	Realização de um login padrão.
Especificações de entrada	**Usuário:** standard_user **Senha:** secret_sauce Clicar no botão “Login”
Procedimentos	1. Acesse a página Web https://www.saucedemo.com/ 2. Insira o dado de entrada “Usuário” no campo Username. 3. Insira o dado de entrada “Senha” no campo Password. 4. Localizar e clicar no botão de “Login”.
Especificações de saída	Após a realização dos procedimentos anteriores, a tela da aplicação deve ser alterada para a exibição inicial da aplicação (contendo os produtos do *e-commerce*).

Fonte: Elaborada pelo autor.

2.4.5.3 *Implementação do Exemplo*

Após configurar o ambiente, como mencionado anteriormente na seção 2.4.5.1, será necessário criar um novo arquivo em JavaScript. Neste arquivo, serão adicionados os comandos para a realização do teste automatizado. Antes de organizar a estrutura do teste, os principais comandos usados são brevemente apresentados. Primeiramente é necessário iniciar uma sessão para abrir o navegador Google Chrome, o Listing 2.4.5 demonstra o comando utilizado. Logo após, a página Web da aplicação SauceDemo deverá ser acessada. O Listing 2.4.6 apresenta o comando *driver.get*() para acessar a URL da aplicação.

```
1 driver = await new Builder().forBrowser('chrome').build();
```

Listing 2.4.5 – Instanciar o navegador Google Chrome no Selenium Webdriver

```
1 await driver.get("https://www.saucedemo.com/")
```

Listing 2.4.6 – Abrir página Web no Selenium Webdriver

Para a adição dos dados de acesso do usuário, é necessário identificar na página os campos de entrada correspondentes. A função *driver.findElement*() é usada para encontrar elementos no *Document Object Mode*l (DOM). Os parâmetros usados podem ser *ids*, classes, características de estilos, e outros métodos de identificar os elementos da página Web. O Listing 2.4.7 apresenta os comandos a serem utilizados para identificação dos campos de **Username**, **Password**, bem como, do botão de Login.

```
1 let userBox = await driver.findElement(By.id("user-name"));
2 let passBox = await driver.findElement(By.id("password"));
3 let loginButton = await driver.findElement(By.id("login-button"));
```

Listing 2.4.7 – Localizar elementos na página Web com Selenium

Após encontrar e definir os elementos principais é necessário interagir com os mesmos para a realização do teste. Existem alguns comandos usados para realizar essa inte-

ração, contudo, para esta automação serão utilizados principalmente os comandos *.click*() e *.sendKeys*(). O comando *.click*() clica nos elementos de uma página Web, e o *.sendKeys*() insere dados fornecidos em elementos editáveis. Com os elementos já definidos anteriormente nas variáveis *userBox* e *passBox*, é necessário clicar e adicionar os dados do usuário nos campos correspondentes. Em seguida, clicar no botão, definido na variável *loginButton*. O Listing 2.4.8 apresenta o processo descrito.

```
1 await userBox.click();
2 await userBox.sendKeys("standard_user");
3 await passBox.click();
4 await passBox.sendKeys("secret_sauce");
5 await loginButton.click();
```

Listing 2.4.8 – Inserindo dados em campos de entradas em Selenium

Por fim, será realizada a verificação das especificações de saída, através dos comandos de solicitação de dados *.getTex*() e o *.getCurrentUrl*(). O comando *.getTex*() obtém da tela um texto visível de um elemento. Já o comando *.getCurrentUrl*(), obtém uma *string* que representa a URL atual do navegador. O Listing 2.4.9 apresenta a verificação da mudança de tela e a presença do título “Products”.

```
1 let productsTitle = await
  ↪  driver.findElement((By.className("title"))).getText();
2 assert.equal("Products", productsTitle);
3 let urlPage = await driver.getCurrentUrl();
4 assert.equal("https://www.saucedemo.com/inventory.html", urlPage);
```

Listing 2.4.9 – Verificação de asserts em Selenium

Portanto, no Listing 2.4.10 é apresentado o *script* de teste automatizado em `Selenium WebDriver`, completo e reorganizado. O resultado após a execução do caso de teste é ilustrado na Figura 7.

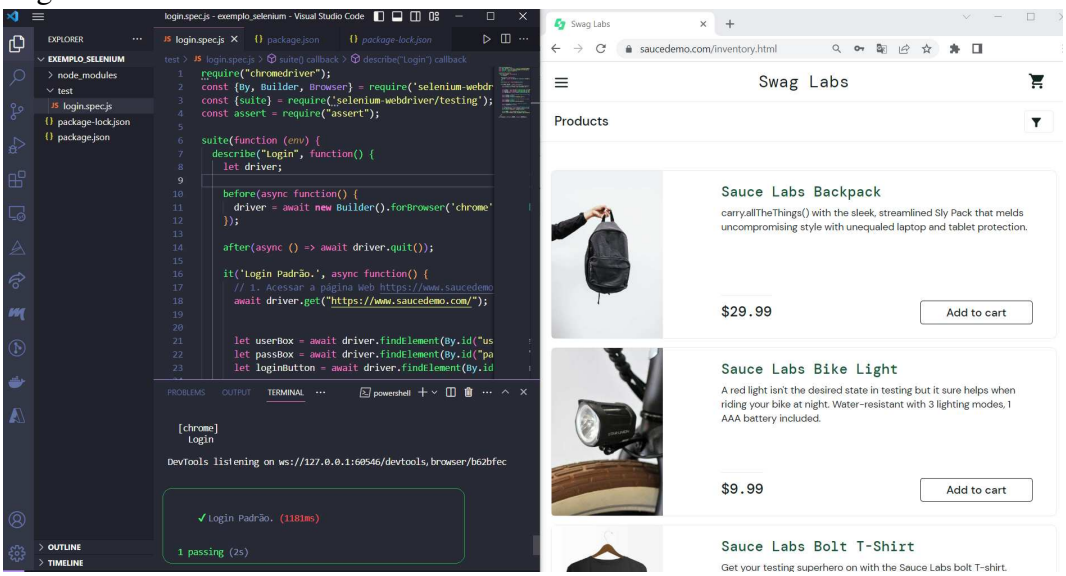
```
require("chromedriver");
const {By, Builder, Browser} = require('selenium-webdriver');
const {suite} = require('selenium-webdriver/testing');
const assert = require("assert");

suite(function (env) {
  describe("Login", function() {
    let driver;
    before(async function() {
      driver = await new Builder().forBrowser('chrome').build();
    });
    after(async () => await driver.quit());
    it('Login Padrão.', async function() {
      await driver.get("https://www.saucedemo.com/");
      let userBox = await driver.findElement(By.id("user-name"));
      let passBox = await driver.findElement(By.id("password"));
      let loginButton = await driver.findElement(By.id("login-button"));
      await userBox.click();
      await userBox.sendKeys("standard_user");
      await passBox.click();
      await passBox.sendKeys("secret_sauce");
      await loginButton.click();

      let productsTitle = await
      ↪  driver.findElement((By.className("title"))).getText();
      assert.equal("Products", productsTitle);
      let urlPage = await driver.getCurrentUrl();
      assert.equal("https://www.saucedemo.com/inventory.html", urlPage);
    })
  })
},{ browsers: [Browser.CHROME, Browser.FIREFOX]});
```

Listing 2.4.10 – Código completo do exemplo em Selenium

Figura 7 – Resultado do caso de teste com Selenium



Fonte: Elaborada pelo autor.

2.5 CYPRESS

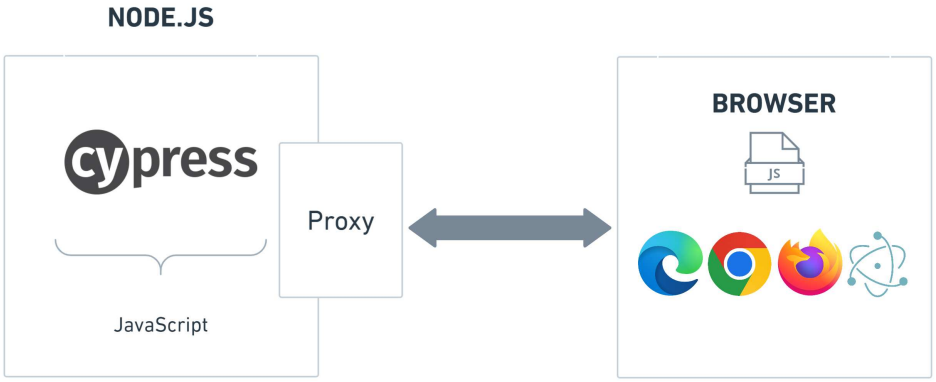
O Cypress[9] é um *framework* utilizado para testes automatizados de aplicações Web *end-to-end*[10], desenvolvido por Brian Man. É uma ferramenta relativamente nova atuando no cenário de testes automatizados quando comparada ao Selenium. Segundo (MACHADO, 2017) a primeira versão foi liberada em 2015, mas o lançamento oficial do Cypress foi realizado somente em 2017. De acordo com (TAKY, 2021), a ferramenta foi projetada com a finalidade de auxiliar o trabalho de desenvolvedores e engenheiros de garantia de qualidade.

O Cypress é construído em Node.js e empacotado como um módulo npm. Utiliza Javascript para o desenvolvimento dos *scripts* de testes e é um projeto *open source*. Além disso, a ferramenta herda muitos dos métodos jQuery para identificação de componentes de interface do usuário e simplifica a travessia e manipulação da árvore HTML DOM, bem como manipulação de eventos, animação CSS e Ajax. O Cypress funciona como um aplicativo autônomo e pode ser instalado nos sistemas operacionais MacOS, Unix/Linux e Windows usando programas ou ferramentas de linha de comando (MWAURA, 2021).

Diferente da maioria das ferramentas de testes para aplicações Web, que operam interagindo com o navegador e executam comandos remotos pela rede, o Cypress opera diretamente dentro do navegador. O navegador executa o código de teste e permite que o Cypress ouça e modifique o comportamento do navegador em tempo de execução, manipulando o DOM e alterando as solicitações e respostas da rede em tempo real (KHETARPAL, 2021). A Figura 8 exemplifica a arquitetura de funcionamento da ferramenta.

[9] https://www.cypress.io/
[10] https://circleci.com/blog/what-is-end-to-end-testing/

Figura 8 – Arquitetura do Cypress



Fonte: Elaborada pelo autor.

Os navegadores que o `Cypress` suporta são: Chrome, Electron (padrão), Chromium, Mozilla Firefox (suporte beta) e navegadores Microsoft Edge (baseado em Chromium). Segundo a documentação oficial da ferramenta, também podem ser adicionados alguns *plugins* via NPM, o que estende a funcionalidade da ferramenta e acrescenta novos comandos e possibilidades de personalização. Além disso, existem alguns outros recursos disponíveis em versões pagas, como por exemplo, a detecção de *flaky tests*[11], integrações com ferramentas de gerenciamento de projetos e algumas outras funcionalidades.

O `Cypress` também possibilita a paralelização dos testes, distribuindo automaticamente os testes buscando otimizar o tempo de execução. Apresenta uma interface amigável com um *dashboard* onde é possível verificar as execuções dos testes. Com a ferramenta é possível realizar diferentes tipos de testes, como por exemplo, testes unitários, funcionais, de integração e de ponta a ponta.

[11] Flaky Tests são casos de testes automatizados que apresentam um comportamento não-determinístico.

2.5.1 Exemplo prático com Cypress

O Cypress é uma ferramenta *desktop* compatível com alguns sistemas operacionais, como por exemplo: MacOS, Linux Ubuntu, Fedora, Debian e Windows. Para esse exemplo prático será considerado a utilização do Cypress com o navegador Google Chrome no Windows 10. O exemplo será implementado utilizando a linguagem de programação JavaScript.

2.5.1.1 Preparação do ambiente

A ferramenta Cypress pode ser instalada utilizando o NPM. Além de facilitar a preparação do ambiente, também pode manter e atualizar a ferramenta. Caso o testador ou desenvolvedor opte por não utilizar o Node.js ou npm, a ferramenta também pode ser instalada através do *download* direto. Desta forma, a versão mais recente disponível da ferramenta é baixada, o site detecta a plataforma que realizou a solicitação de *download* automaticamente, então não é necessário escolher a versão segundo o ambiente. Após finalizar o *download*, descompactar e instalar a ferramenta, o Cypress estará pronto para ser utilizado. Neste exemplo, a instalação será realizada com o npm, executando portando, o Listing 2.5.1.

```
1 npm install cypress --save-dev
```

Listing 2.5.1 – Comando para instalar o Cypress com npm

Após concluir a instalação, além dos arquivos e pastas do Cypress é gerado automaticamente um arquivo chamado *package-lock.json*. Este arquivo controla as possíveis alterações que o npm realiza na árvore *node_modules* ou no *package.json*. Descrevendo a árvore exata que foi gerada, e retornando árvores idênticas mesmo após instalações subsequentes. Com o Cypress instalado, para abrir a ferramenta a partir da raiz do projeto, é necessário apenas executar no terminal o comando do Listing 2.5.2.

```
1 npx cypress open
```

Listing 2.5.2 – Comando de execução do Cypress

2.5.1.2 Explicação do Exemplo

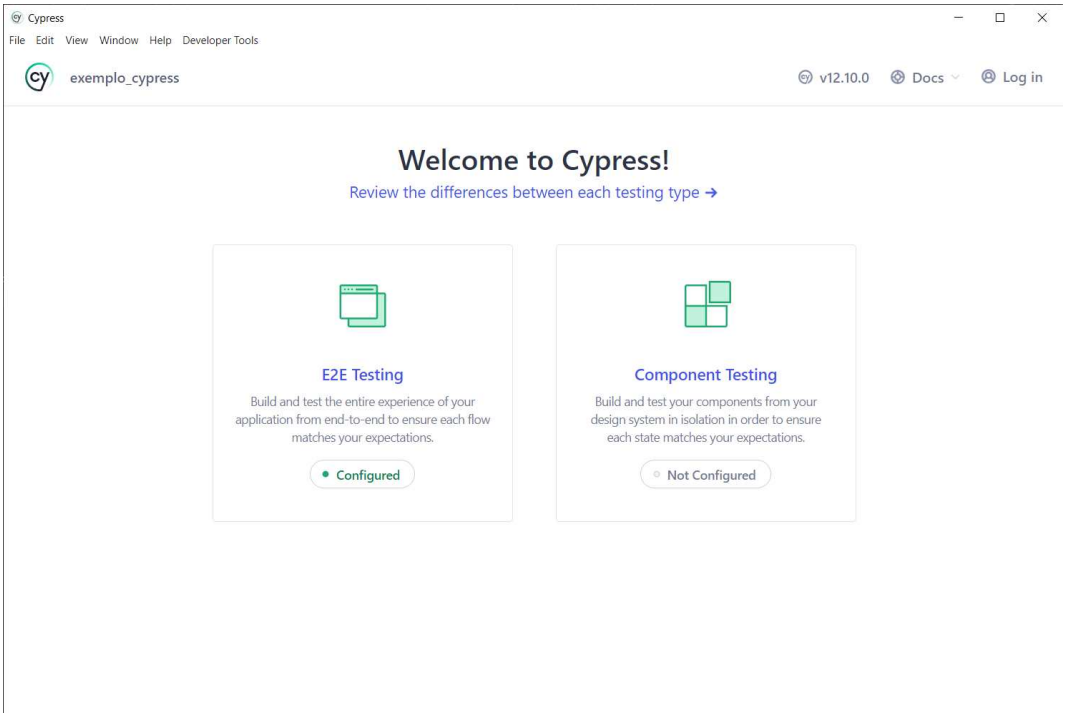
Para ilustrar a utilização do `Cypress`, será realizado o mesmo exemplo anterior descrito na Seção 2.4.5. Usando as credenciais do perfil *standard_user* no site de demonstração, SauceDemo. Portanto, será realizado o acesso à página da aplicação, o preenchimento dos dados do usuário e a verificação do sucesso do login. As especificações do caso de teste são apresentados na Tabela 2 da Seção 2.4.5.

2.5.1.3 Implementação do Exemplo

Após realizar a instalação da ferramenta através do npm, como mencionado anteriormente, é necessário criar o projeto e o *script* de testes em JavaScript. Para isto, é executado o comando do Listing 2.5.2 em uma interface de linha de comando. Logo após, a ferramenta exibe algumas opções de configurações para a elaboração dos *scripts* de testes (Figura 9). Inicialmente será definido o tipo de teste realizado, neste exemplo será selecionada a opção *E2E Testing*. O teste E2E é realizado para testar uma aplicação desde o navegador Web até o *back-end*. Segundo a documentação oficial da ferramenta, neste tipo de teste o `Cypress` realiza o comportamento fiel aos usuários interagindo diretamente com a aplicação

Posteriormente, a ferramenta elenca um conjunto de arquivos para a criação prévia da configuração para realizar o tipo de teste escolhido (Figura 10). Em seguida, são apresentadas as opções de navegadores, será selecionado o Google Chrome. Após selecionar o navegador, será necessário criar uma nova especificação vazia. E neste arquivo os passos do teste são

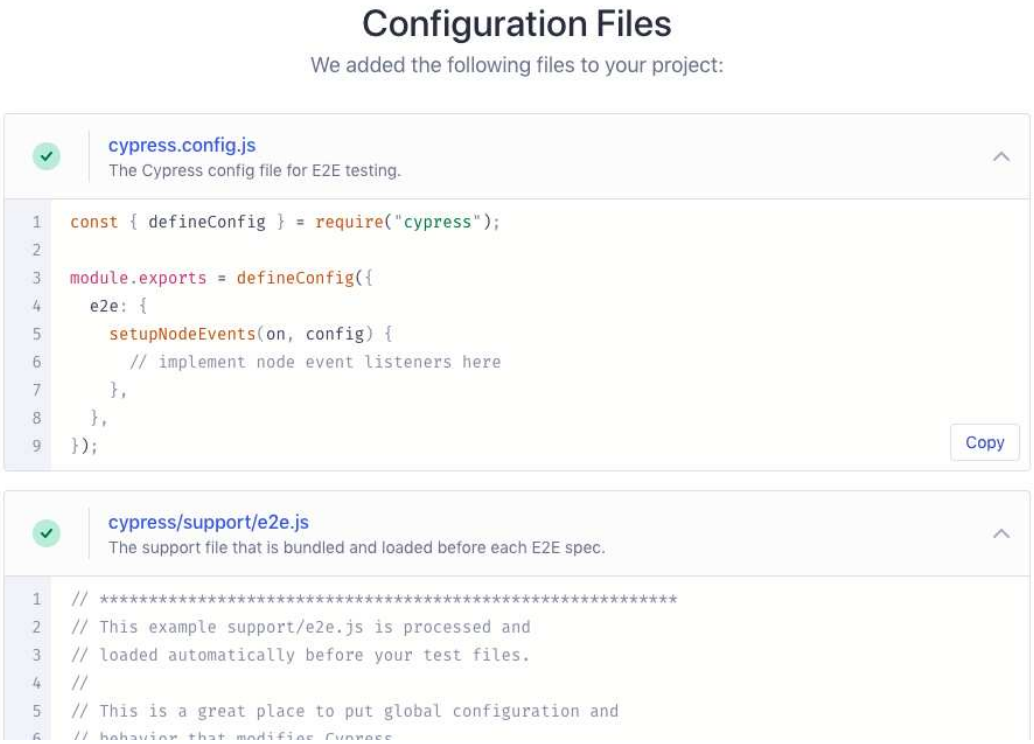
Figura 9 – Interface da tela inicial do Cypress



Fonte: Elaborada pelo autor.

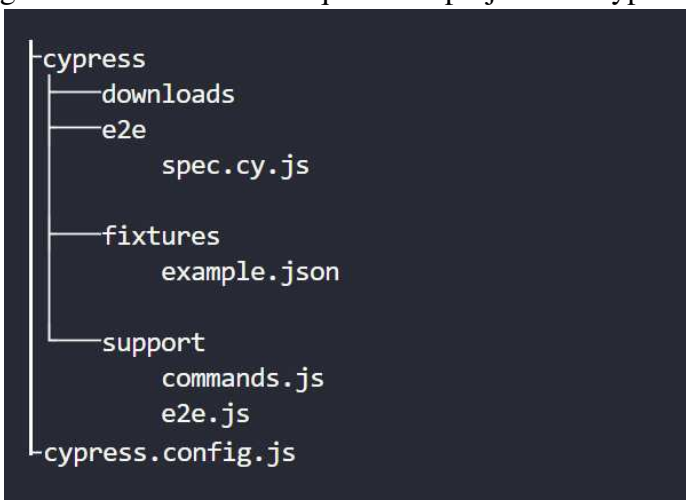
configurados. Assim, a estrutura inicial de pastas e arquivos necessários para o desenvolvimento e execução dos testes é ilustrada na Figura 11.

Figura 10 – Interface da tela de configuração do Cypress



Fonte: Elaborada pelo autor.

Figura 11 – Estrutura de arquivos do projeto em Cypress



Fonte: Elaborada pelo autor.

Assim com a estrutura do projeto gerada inicialmente pelo `Cypress`, é preciso adicionar no arquivo *spec.cy.js* os comando para desenvolvimento do *script* de teste. Antes de organizar a estrutura do teste, será discutido brevemente os principais comandos usados neste exemplo. O comando *cy.visit*() é usado para acessar URLs no navegador, logo o Listing 2.5.3 apresenta a instrução para abrir a página Web da aplicação em teste. É importante notar, que diferente de outras ferramentas, não foi necessário realizar nenhuma configuração para a ferramenta fazer uso do navegador.

```
1 cy.visit("https://www.saucedemo.com/")
```

Listing 2.5.3 – Abrir página Web usando Cypress

Os comandos *.get*() e *.type*(), são utilizados para localizar e inserir os dados de acesso nos campos de entrada de **Username** e **Password**. No Listing 2.5.4 os dados são atribuídos nos respectivos campos de entrada. Com o comando *.get*(), são especificadas as características dos elementos da página Web, estes parâmetros podem ser *ids*, classes, características de estilos, e outros métodos de identificar um elemento DOM. Através do comando *.type*() são inseridos os dados de texto a serem digitados no elemento.

```
cy.get('#user-name').type("standard_user")
cy.get('#password').type("secret_sauce")
```

Listing 2.5.4 – Localizar e preencher os campos de entrada em Cypress

Em seguida, para localizar o botão de Login na tela, o mesmo comando *.get*() é usado com argumentos referentes ao botão. Por fim, o comando *.click*() clica no botão para solicitar a realização do login na página.

```
cy.get('#login-button').click()
```

Listing 2.5.5 – Clicar em elemento DOM usando Cypress

Na validação das especificações de saída, é utilizado o comando *.should*() para determinar as asserções. Assim, é verificado a mudança de tela comparando se a URL atual *.url*().*should*() corresponde ao endereço da página que deveria aparecer. Isto é feito atribuindo 'eq' (de *equals*) e a *string* contendo a URL como parâmetro. Também é verificado a presença do elemento com o título "Products"na tela atual da página, usando o atributo de visibilidade *.should*('*be.visible*'). O Listing 2.5.6 apresenta a validação desses passos do teste.

```
cy.url().should('eq', 'https://www.saucedemo.com/inventory.html')
cy.get('span:contains(Products)').should('be.visible')
```

Listing 2.5.6 – Verificação dos asserts em Cypress

Portanto, o *script* de teste automatizado em `Cypress` completo e reorganizado é exibido no Listing 2.5.7. O resultado após a execução do *script* pode ser visto na Figura 10.

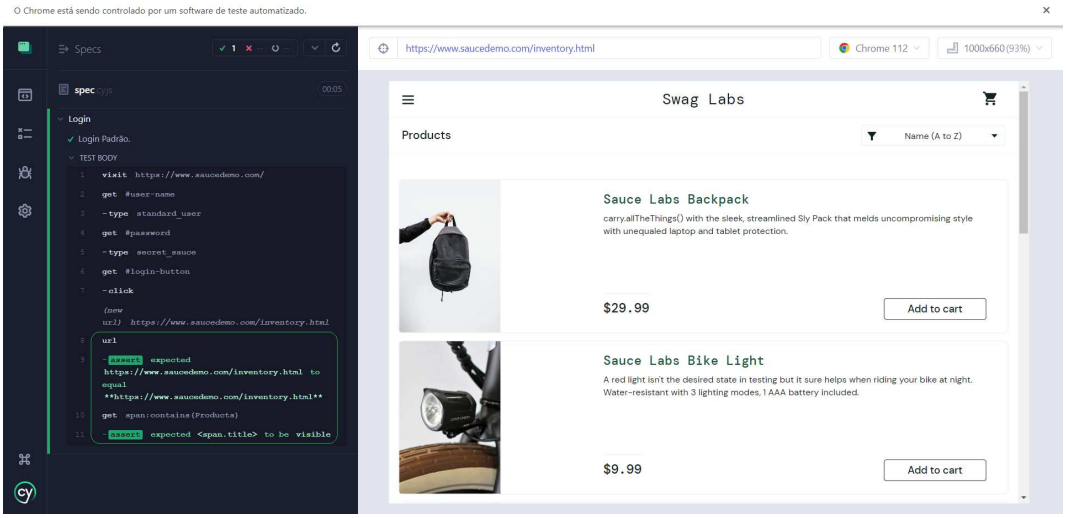
```
/// <reference types="cypress" />

describe("Login", () => {
  it("Login Padrão.", function () {
    cy.visit("https://www.saucedemo.com/")
    cy.get('#user-name').type("standard_user")
    cy.get('#password').type("secret_sauce")
    cy.get('#login-button').click()

    cy.get('span:contains(Products)').should('be.visible')
  });
});
```

Listing 2.5.7 – Código completo do exemplo em Cypress

Figura 12 – Resultado do caso de teste com Cypress

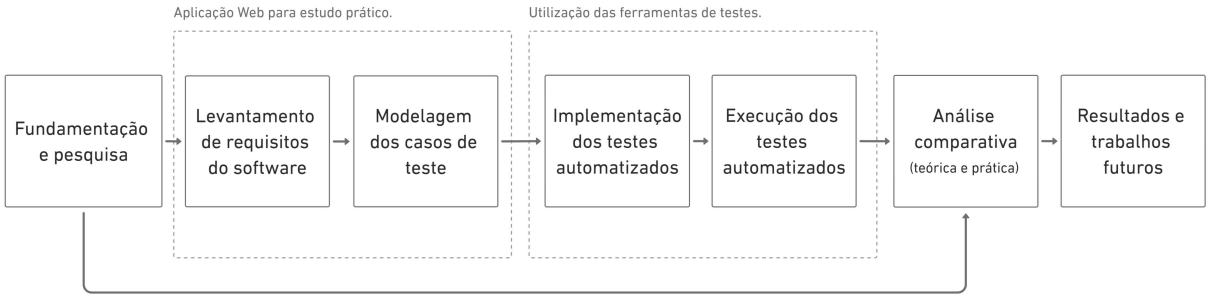


Fonte: Elaborada pelo autor.

3 METODOLOGIA

Neste capítulo será abordado a metodologia utilizada para a realização do presente estudo comparativo entre as ferramentas de testes `Selenium` e `Cypress`. Definindo, portanto, as etapas necessárias para alcançar os objetivos deste trabalho. Para compreender melhor a metodologia proposta será detalhado cada uma das seguintes etapas: fundamentação e pesquisa, levantamento de requisitos do *software*, modelagem dos casos de teste, implementação dos testes automatizados, execução dos testes automatizados, análise comparativa e, por fim, resultados e trabalhos futuros. A Figura 13 resume as etapas e as próximas seções descrevem cada uma delas.

Figura 13 – Etapas metodológicas do trabalho



Fonte: Elaborada pelo autor.

3.1 Fundamentação e pesquisa

Inicialmente, na primeira fase deste estudo, foi considerada uma metodologia de pesquisa exploratória e descritiva. Segundo (GIL *et al.*, 2002) a pesquisa exploratória tem como objetivo proporcionar maior familiaridade com o problema, e envolvem levantamentos bibliográficos e análise de exemplos que estimulem a compreensão. Já, a pesquisa descritiva tem como objetivo primordial a descrição das características de determinado fenômeno ou estabelecimento de relações entre variáveis (GIL *et al.*, 2002).

Diante disso, nesta etapa foram abordados o estudo de outros trabalhos científicos,

documentação oficial das ferramentas, artigos e livros relacionados à temática. Como também, as ferramentas de teste foram estudadas segundo o levantamento de suas características, funcionalidades, recursos e outros aspectos. Como resultado teremos um aprofundamento e comparação de dados entre as ferramentas propostas para estudo, o `Selenium` e o `Cypress`, com base em estudos bibliográficos. Contribuindo assim, com uma documentação detalhada para trabalhos que venham utilizar as ferramentas, bem como auxiliar escolhas mais assertivas em relação ao uso das mesmas para a automação de testes.

3.2 Levantamento de requisitos do *software*

Para a avaliação prática deste estudo foi selecionada uma aplicação Web para a realização dos testes automatizados em ambas as ferramentas, `Selenium` e `Cypress`. Logo, a mesma assume o papel de sistema em teste, do inglês *System Under Test* (*SUT*)[12]. Para selecionar o SUT foi realizado algumas pesquisas de aplicações disponíveis para estudo de testes, levando em consideração as seguintes condições: a) deve ser uma aplicação Web sob licença de código aberto; b) deve simular um sistema real comumente usado, para facilitar a compreensão das funcionalidades; c) deve ser uma aplicação de configuração e acesso simplificado para facilitar a automatização e replicação dos experimentos. Com base nesses requisitos, o SUT selecionado para o estudo e aplicação prática dos testes automatizados foi o ParaBank.

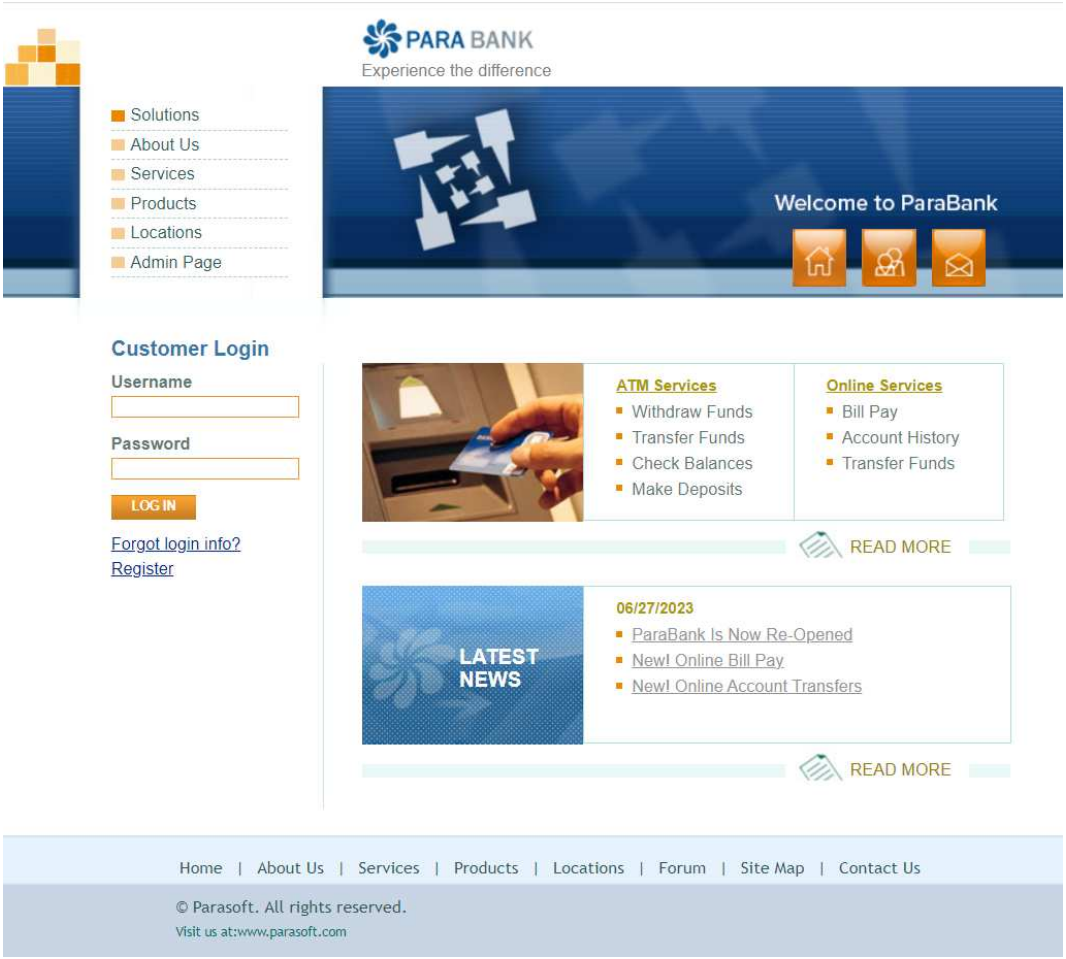
O ParaBank é uma aplicação Web que simula um site de banco online, desenvolvida pela empresa *Parasoft*[13] para demonstração de suas soluções de *software*. O ParaBank é um projeto de código aberto implementado em Java no lado do servidor, dentre suas principais

[12] *System under test*, refere-se a um sistema que está sendo testado ou validado.

[13] *Parasoft* é uma empresa de tecnologia da informação, que ajuda as organizações a fornecer continuamente *software* de qualidade com seu conjunto integrado e comprovado de mercado de ferramentas automatizadas de teste de *software* (https://www.parasoft.com/).

funcionalidades podemos elencar o gerenciamento de clientes, transações, pagamentos e outros serviços bancários. Na Figura 14 podemos observar a tela inicial do sistema, que dispõem de alguns menus laterais para navegação, como também, os campos direcionados para realizar o login e cadastro de clientes.

Figura 14 – Tela inicial da aplicação Web Parabank



Fonte: Elaborada pelo autor.

Assim, com a definição e escolha do SUT, foi realizado um levantamento dos aspectos funcionais da aplicação. Segundo (WAZLAWICK, 2011), o levantamento de requisitos é uma etapa significativa, onde se utiliza de todas as informações disponíveis do sistema para gerar os requisitos e com isso identificar as principais funções daquele sistema. A aplicação Parabank possui muitas funcionalidades e possibilidades para simular um sistema bancário, entretanto, para o escopo deste trabalho foram consideradas apenas algumas dessas funcionalidades para

realização dos testes. Essas definições serão explicadas no Capítulo 4.

3.3 Modelagem dos casos de teste

Antes de executar qualquer tipo de teste em um *software*, é necessário analisar e elencar os possíveis passos que deverão ser executados, assim como, os resultados que deverão ser obtidos de acordo com os requisitos da aplicação em teste. A documentação destas informações consiste na criação de um caso de teste. Segundo (SOMMERVILLE, 2011), os casos de teste são especificações das entradas e das saídas esperadas do sistema, além de uma declaração do que está sendo testado.

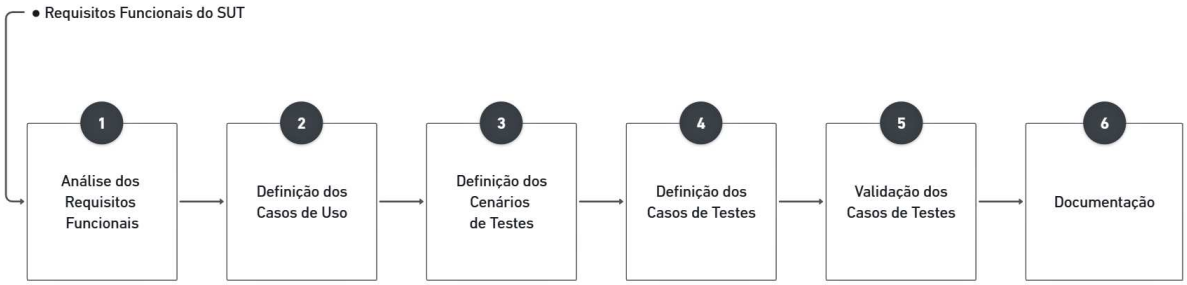
Para a criação dos casos de teste foi proposto uma abordagem de especificação a partir de casos de uso. Segundo (LEFFINGWELL, 2010), um caso de uso é a descrição de uma sequência de ações que um usuário pode realizar para interagir com o sistema, visando alcançar um determinado resultado esperado. De acordo com o *Rational Unified Process* (*RUP*)[14], um caso de uso é uma sequência de ações executadas para fornecer um resultado observável em um sistema. Assim, os casos de testes derivados de casos de uso podem garantir a conformidade da aplicação em relação aos seus requisitos funcionais (KOSINDRDECHA; DAENGDEJ, 2010).

O SUT escolhido neste estudo apresenta uma quantidade significativa de funcionalidades, logo podem existir um grande número de possíveis casos de uso. No entanto, o escopo deste trabalho se limita aos requisitos funcionais obtidos na etapa anterior. A Figura 15 ilustra as etapas que foram realizadas para definir a modelagem dos casos de testes.

Em (HEUMANN, 2001) é proposto um processo de três etapas para gerar casos de teste a partir de casos de uso: (a) para cada caso de uso é definido um conjunto de cenários; (b)

[14] *Rational Unified Process (RUP)* é uma abordagem de engenharia de *software* para delegar atividades e responsabilidades dentro de uma organização de desenvolvimento. Seu objetivo principal é permitir a criação de *software* de alta qualidade que satisfaça os requisitos do usuário final.

Figura 15 – Modelagem dos casos de testes



Fonte: Elaborada pelo autor.

para cada cenário é identificado pelo menos um caso de teste; e (c) para cada caso de teste são determinados os valores dos dados com os quais testar. Logo, as etapas realizadas neste estudo e ilustradas na Figura 15 foram baseadas na proposta de (HEUMANN, 2001), com o acréscimo de outros processos, tais como a análise dos requisitos funcionais e a validação dos casos testes. Assim, podemos descrever cada uma das etapas:

i. **Análise dos Requisitos Funcionais:** Inicialmente foi realizado uma análise dos requisitos definidos do SUT, obtidos na Seção 3.2, para definir a abrangência e priorização das funcionalidades e requisitos usados neste trabalho.
ii. **Definição dos Casos de Uso:** Foram identificados, de acordo com os requisitos, os casos de uso que definem as ações e interações com o SUT para atender as funcionalidades elencadas para os testes. Para cada caso de uso foi definido os dados de título, descrição, precondições e pós-condições.
iii. **Definição dos Cenários:** Para cada caso de uso foram definidos um conjunto de cenários, identificados como diferentes combinações de fluxos básicos e alternativos que podem ser executados nas funcionalidades do SUT.
iv. **Definição dos Casos de Testes:** Com os cenários obtidos foram definidos os casos de testes. Cada caso de teste representa uma especificação dos fluxos a serem percorridos,

para estes foi determinado um identificador, título, conjunto de entradas, condições de execução (ou passos) e resultados esperados.

v. **Validação dos Casos de Testes:** Para avaliar a viabilização de implementar os casos de testes criados, cada um deles foi executado de maneira manual no SUT. Realizando portanto os fluxos definidos detalhadamente, considerando os dados de entrada, passos e resultados esperados.

vi. **Documentação:** Por fim, foram documentados todos os casos de uso, cenários e casos de testes definidos após realizar as etapas anteriores.

3.4 Implementação dos testes automatizados

Durante esta etapa foram desenvolvidos *scripts* de testes utilizando as duas ferramentas em estudo, `Selenium` e `Cypress`. Para dar equidade ao estudo empírico, para os testes em ambas as ferramentas, foram utilizados o mesmo ambiente de produção, aplicação Web, requisitos e casos de testes, além da linguagem de programação e *Integrated Development Environment* (IDE)[15]. Utilizando portanto, a linguagem JavaScript e o VSCode[16] como IDE. Como resultado, foi alcançado o desenvolvimento de *suites* de testes automatizados para as duas ferramentas, devidamente documentadas e armazenadas no GitHub, plataforma de hospedagem e versionamento de código-fonte.

[15] IDE, do inglês *integrated development environment* ou ambiente de desenvolvimento integrado.

[16] *Visual Studio Code* é um editor de código redefinido e otimizado para criar e depurar aplicativos modernos da Web e da nuvem (https://code.visualstudio.com/).

3.5 Execução dos testes automatizados

Após a finalização da etapa anterior (Seção 3.4), foi realizado a execução das *suites* de testes automatizados em ambas as ferramentas em estudo. Como resultado, foi obtido a geração de um relatório de execução exibindo o *status* de sucesso ou falha para cada caso de teste. Assim, além de obter a geração do relatório, a execução automatizada dos testes em cada uma das ferramentas forneceu os dados necessários para auxiliar no embasamento do estudo empírico. As métricas de avaliação utilizadas são detalhadas na próxima etapa.

3.6 Análise comparativa

Utilizando os dados da revisão bibliográfica e do estudo empírico realizados anteriormente, nesta etapa foi definido um conjunto de critérios para a avaliação das ferramentas `Selenium` e `Cypress`. Desta forma, a análise comparativa entre as ferramentas foi baseada nos resultados das características e do estudo empírico das ferramentas alvo deste estudo.

3.6.1 Características das ferramentas alvo do estudo

Adotando uma metodologia baseada na análise documental das ferramentas em estudo, foram consideradas algumas características importantes para as ferramentas de automação de testes. Assim, são elencadas as características e critérios escolhidos para este estudo:

i. **Sistemas Operacionais suportados:** Conhecer em qual Sistema Operacional (SO) uma ferramenta de automação de testes Web pode ser utilizada é muito importante. Sabendo que atualmente existe uma grande variedade de sistemas operacionais e distribuições, é crucial que estas ferramentas possam suportar a execução em diferentes ambientes. Segundo (SOMMERVILLE, 2011), “a confiabilidade de um sistema depende do contexto

em que ele é usado", portanto, é necessário avaliar o sistema nos mais diversos contextos possíveis. Logo, neste estudo, são considerados o suporte do `Selenium` e `Cypress` aos principais sistemas operacionais da atualidade: Microsoft Windows, Apple MacOS, Linux, Android e iOS (WGU, 2021).

ii. **Linguagens de Programação suportadas:** Segundo (GOTARDO, 2015), uma linguagem de programação é um método padronizado com um conjunto de regras sintáticas e semânticas, usadas para expressar instruções de um programa a um computador. A escolha de qual linguagem utilizar pode estar vinculada a diferentes variáveis, como por exemplo, a curva de aprendizagem, tempo de experiência ou até mesmo preferência. Neste contexto, para uso e escrita de *scripts* de testes, é importante avaliar o suporte das linguagens disponíveis de uma ferramenta de automação de testes. Assim, quanto maior a variedade de linguagens de programação que uma ferramenta dá suporte, mais fácil pode se tornar o processo de aprendizado e utilização dessa ferramenta, uma vez que, os testadores podem utilizar a linguagem que possuem mais familiaridade. Neste trabalho, foram consideradas para efeitos de comparação o suporte para algumas das linguagens mais usadas: JavaScript, Python, Java, PHP e Ruby (iMasters, 2023).

iii. **Navegadores suportados:** Aplicações Web executam diretamente em navegadores, e como citado anteriormente em (SOMMERVILLE, 2011) é crucial avaliar as possibilidades em diversos contextos. Logo, para uma ferramenta de automação de testes, é importante oferecer possibilidades para a execução de testes em diferentes navegadores. Assim, neste estudo consideramos o suporte das ferramentas aos diferentes navegadores do mercado: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera e Safari (SANTOS, 2023).

iv. **Instalação e configuração:** Algumas ferramentas podem possuir dependências de outras tecnologias para funcionar adequadamente. Configurações complexas e mal documentadas,

podem ser um fator limitante de algumas ferramentas. Neste contexto, foi avaliado as dependências das ferramentas em relação a outros recursos ou tecnologias, como por exemplo o uso de *drivers* ou bibliotecas.

v. **Recursos disponíveis:** Foram avaliados também alguns recursos disponíveis ao utilizar as ferramentas, como por exemplo, a possibilidade de usar múltiplas instâncias do navegador durante os testes, execução de testes em paralelo, execução remota, *Time Travel*[17] e espera automática[18]. Como também, foram avaliados aspectos da geração de relatórios e suporte ao *Continuous Integration/Continuous Delivery* (CI/CD)[19].

3.6.2 Estudo empírico com as ferramentas alvo do estudo

Para realizar o estudo empírico das ferramentas foram desenvolvidos *scripts* de testes automatizados utilizando o `Selenium` e o `Cypress`. As métricas selecionadas levam em consideração as etapas necessárias para a implementação dos testes em uma aplicação Web. Logo, foram analisados os processos: i) de instalação e configuração; ii) de escrita dos *scripts*; iii) de execução dos testes; e iv) de geração de relatórios. Para cada uma dessas etapas foram determinadas algumas métricas comparativas para fomentar a discussão dos resultados. São elas:

i. **Instalação e configuração:** Para o processo de instalar as ferramentas e configurar o ambiente de desenvolvimento para implementação dos testes automatizados foram considerados os aspectos de dependência de outras tecnologias.

ii. **Desenvolvimento dos *scripts*:** Nesta etapa foram analisados algumas métricas de *software* para a comparação dos *scripts* desenvolvidos nas ferramentas `Selenium` e `Cypress`. Se-

[17] *Time Travel*, em português "viagem no tempo", aplicado ao contexto de testes possibilita salvar e retornar aos estados anteriores da aplicação durante a execução dos testes.

[18] A espera automática está relacionada ao carregamento de elementos em uma aplicação Web, como um *timeout* configurado pela própria ferramenta em tempo de execução.

[19] CI/CD, abreviação de *Continuous Integration/Continuous Delivery*, trata-se de uma prática de desenvolvimento de *software* que visa tornar a integração de código mais eficiente por meio de *builds* e testes automatizados.

gundo (SOMMERVILLE, 2011), uma métrica de *software* pode ser definida como uma característica de um sistema ou processo de *software* que pode ser objetivamente medido. Existem muitos exemplos de técnicas de métricas de *software*, contudo, neste trabalho foi abordado a contagem de linha de código e a verbosidade. Logo, foram definidos atributos para avaliar os *scripts* implementados utilizando as ferramentas em estudo, são eles: a) *Source Lines of Code* (SLOC), que definem as linhas de código-fonte; b) *Comment Lines of Code* (CLOC), que indica as linhas comentadas; c) *Physical Lines of Code* (PLOC), que define as linhas totais (físicas) do código; d) *Blank Lines of Code* (BLANK), que indica o número de linhas em branco; e) a contagem de caracteres totais do código.

iii. **Execução dos testes:** Após finalizar a implementação em ambas as ferramentas, `Selenium` e `Cypress`, os *scripts* de testes automatizados foram executados. Assim, após a execução foi possível verificar o tempo necessário para finalizar cada um dos casos de testes, como também, analisar os resultados apresentados em cada ferramenta.

iv. **Relatórios de testes:** A configuração e geração dos relatórios de testes em ambas ferramentas, `Selenium` e `Cypress`, foram analisadas de acordo com o suporte para configuração, a completude e organização dos dados.

3.7 Resultados e trabalhos futuros

Para finalizar foi realizada uma discussão e revisão geral dos resultados teóricos e práticos obtidos. Nesta fase, foi detalhado uma conclusão acerca da comparação entre as ferramentas, `Selenium` e `Cypress`, em relação a todas as métricas definidas. Assim como, foram apresentados as perspectivas de melhorias para aperfeiçoar os resultados em possíveis trabalhos futuros.

4 RESULTADOS E DISCUSSÕES

Neste capítulo serão apresentados os resultados, considerando dois principais momentos: levantamento de características e estudo empírico das ferramentas alvo deste trabalho. O levantamento de características considera a adoção da metodologia exploratória para realizar uma análise documental das ferramentas `Selenium` e `Cypress`. Para isto, foram pressupostas algumas características importantes para as ferramentas de automação de testes. Assim foi realizado um comparativo das ferramentas ressaltando suas características, recursos, contextos de utilização e outros aspectos funcionais. Já no estudo empírico, foi selecionada uma aplicação Web como SUT para a avaliação de conceitos práticos e implementação de testes automatizados utilizando ambas as ferramentas. Logo, algumas métricas foram determinadas a fim de avaliar o comportamento e desempenho do `Selenium` e `Cypress` ao serem submetidos a um mesmo contexto de automação e execução de testes.

4.1 Características das ferramentas alvo do estudo

Os resultados do estudo das características das ferramentas `Selenium` e `Cypress` são apresentados nos tópicos a seguir. São consideradas informações comprovadas e obtidas por meio de documentações oficiais e estudos bibliográficos de cada ferramenta. As comparações e discussões englobam os aspectos e características previamente mencionadas no Capítulo 3.

4.1.1 Sistemas Operacionais suportados

Em relação ao suporte para utilização e execução em diferentes sistemas operacionais, ambas as ferramentas apresentam resultados satisfatórios, cobrindo a maioria dos sistemas operacionais mais populares como mostra a Tabela 3. Neste aspecto as ferramentas demonstram

uma diferença apenas em relação aos sistemas operacionais para aplicações móveis, Android e iOS.

Segundo a sua documentação oficial, o `Cypress` é uma ferramenta de teste criada para a Web moderna (CYPRESS, 2023). Portanto, não oferece suporte para execução de testes em dispositivos móveis, para realizar este tipo de teste é preciso que o aplicativo possa ser executado em um navegador de um computador pessoal e não em um dispositivo móvel. Já o `Selenium`, oferece suporte para realizar testes em aparelhos móveis (Selenium, 2023). Para isto, a ferramenta utiliza de *frameworks* específicas para automatizar os testes, como por exemplo o Appium[20]. O Appium deriva suas raízes do `Selenium` e utiliza a mesma API do WebDriver para conduzir navegadores móveis no Android e iOS. Diante disto, o `Selenium` oferece um suporte maior para diferentes SOs, incluindo projetos para dispositivos móveis.

Tabela 3 – Suporte das ferramentas Selenium e Cypress aos principais SOs

Ferramenta de Testes	**Sistemas Operacionais**				
	Microsoft Windows	**Apple MacOS**	**Linux**	**Android**	**iOS**
Selenium	✓	✓	✓	✓	✓
Cypress	✓	✓	✓	✗	✗

Fonte: Elaborada pelo autor.

4.1.2 Linguagens de Programação suportadas

A Tabela 4 apresenta uma comparação em relação ao suporte a diferentes linguagens de programação para escrita de testes nas ferramentas `Selenium` e `Cypress`. Neste aspecto as ferramentas em estudo apresentam uma grande diferença. O `Selenium` oferece mais liberdade na escolha da linguagem, dando suporte a uma maior variedade de linguagens de programação que podem ser utilizadas para a criação dos testes automatizados. Já o `Cypress` se limita ao uso do

[20] O Appium é uma *framework* projetado para facilitar a automação da interface do usuário de muitas plataformas, incluindo dispositivos móveis (https://appium.io/).

JavaScript, que é uma linguagem amplamente utilizada em aplicações Web. Segundo (CYPRESS, 2023), o `Cypress` foi criado para lidar especialmente com estruturas JavaScript, prometendo apresentar maior controle sobre a automação ao manipular tudo que está sendo executado dentro do navegador.

Tabela 4 – Suporte das ferramentas Selenium e Cypress a diferentes linguagens de programação

Ferramenta de Testes	**Linguagem de Programação**				
	JavaScript	**Python**	**Java**	**PHP**	**Ruby**
Selenium	✓	✓	✓	✓	✓
Cypress	✓	✗	✗	✗	✗

Fonte: Elaborada pelo autor.

4.1.3 Navegadores suportados

A Tabela 5 apresenta o comparativo em relação ao suporte aos diferentes tipos de navegadores das ferramentas `Selenium` e `Cypress`. É verificado portanto, que o `Selenium` pode ser utilizado em uma maior variedade de navegadores em comparação ao `Cypress`. Isto pode ser justificado principalmente pelo favorecimento que arquitetura do `Selenium` proporciona, uma vez que sua execução em diferentes tipos de navegadores pode ser realizada apenas com o uso dos *drivers* correspondentes de cada navegador. O `Cypress`, além dos navegadores listados na Tabela 5, também oferece suporte a outros, como por exemplo o Electron, que já vem embutido na ferramenta e não precisa ser instalado separadamente.

Tabela 5 – Suporte das ferramentas Selenium e Cypress a diferentes navegadores

Ferramentas de Testes	**Navegadores**				
	Google Chrome	**Mozilla Firefox**	**Microsoft Edge**	**Opera**	**Safari**
Selenium	✓	✓	✓	✓	✓
Cypress	✓	✓	✓	✗	✗

Fonte: Elaborada pelo autor.

4.1.4 Instalação e configuração

O grau de complexidade de instalação e configuração pode ser algo relativo, logo as ferramentas em estudo foram avaliadas quanto a dependência de outras tecnologias para compor o projeto de automação de testes. A Tabela 6 apresenta os resultados obtidos através das documentações de ambas as ferramentas. O `Cypress` pode ser instalado sem a necessidade de nenhuma dependência ou tecnologia adicional, através do *download* direto da versão mais recente disponível, sendo necessário apenas descompactar e instalar. Como também, pode ser instalado utilizando o *npm* ou o *yarn*[21] apenas com um comando. Já para instalar e utilizar o `Selenium`, é necessário outras tecnologias para a configuração. O `Selenium Webdriver` requer o *download* de *drivers* de navegadores e uma configuração prévia do ambiente de teste, assim como o `Selenium Grid`. Entretanto, para o `Selenium IDE` , basta realizar o *download* da extensão e instalar. Logo, neste contexto, existe uma maior complexidade para a instalação e configuração do `Selenium` em relação ao `Cypress`. Os dados da coluna de configuração prévia foram baseados nas dependências e envolvimento de outros conceitos nos tutoriais de instalação presentes nas documentações das ferramentas.

Tabela 6 – Instalação e configuração das ferramentas Selenium e Cypress

Ferramenta de Testes		**Dependência de bibliotecas, drivers ou outras tecnologias**	**Configuração prévia complexa do ambiente**
Selenium	**Grid**	SIM	SIM
	IDE	NÃO	NÃO
	Webdriver	SIM	SIM
Cypress		NÃO	NÃO

Fonte: Elaborada pelo autor.

[21] O Yarn é um dos principais gerenciadores de pacotes JavaScript (https://yarnpkg.com/).

4.1.5 Recursos disponíveis

A Tabela 7 apresenta os resultados da comparação entre as ferramentas `Selenium` e `Cypress` quanto a alguns recursos disponíveis. As ferramentas apresentam alguns desses recursos em comum, como a geração de relatórios de testes, a possibilidade de execução de teste paralelo e o uso em ambientes de integração e entrega contínua (CI/CD). O `Cypress` apresenta um diferencial em relação a disponibilidade de uma interface para acompanhar a execução dos testes, como também a possibilidade de espera automática e de salvar e retornar aos estados anteriores da aplicação durante a execução dos testes (conceito de *Time Travel*). Já o `Selenium`, permite a execução remota de testes e a utilização de múltiplas instâncias de um navegador e múltiplas abas.

Tabela 7 – Recursos disponíveis das ferramentas Selenium e Cypress

Ferramentas de Testes	Interface da Execução dos Testes	Espera Automática	CI/ CD	Execução Remota	Teste Paralelo	Múltiplas instâncias do navegador	Time Travel	Relatório de testes
Selenium	✗	✗	✓	✓	✓	✓	✗	✓
Cypress	✓	✓	✓	✗	✓	✗	✓	✓

Fonte: Elaborada pelo autor.

Existem inúmeros outros recursos ofertados pelas ferramentas `Selenium` e `Cypress`, como também formas de configurar alguns que não são disponibilizados por padrão. Deste modo, a comparação entre as ferramentas se torna algo mais subjetivo, levando em conta que cada projeto de testes têm contextos e objetivos diferentes. Portanto, cabe ao testador realizar o levantamento do que será preciso na automação dos testes e optar pela ferramenta que oferece os recursos desejáveis. No Apêndice A é apresentada uma tabela que resume os resultados teóricos discutidos nesta seção, como também outros aspectos e características das ferramentas em estudo.

4.2 Estudo empírico com as ferramentas alvo do estudo

Nesta seção são apresentados os resultados do estudo empírico das ferramentas, `Selenium` e `Cypress`, baseados na implementação de testes automatizados em uma aplicação Web selecionada como *SUT*. Seguindo as etapas previamente mencionadas no Capítulo 3. Logo, foi realizado todo o processo de instalação e configuração das ferramentas, desenvolvimento dos *scripts*, execução dos testes e observação dos resultados e relatórios.

4.2.1 Sistema em teste (SUT)

O sistema Parabank foi selecionado como SUT neste trabalho para a modelagem e execução dos testes automatizados. O Parabank é uma aplicação Web gratuita e de código aberto, o seu projeto está disponível para uso tanto em um domínio[22] na Web quanto em um repositório remoto[23]. Como mencionado anteriormente, a aplicação detém uma série de funcionalidades de um sistema bancário online, desde o cadastro de clientes até a realização de transações e pagamentos. Contudo, para delimitar o escopo e abrangência dos testes implementados foram selecionadas algumas telas e menus do sistema, são eles: login de usuários, encontrar informações de login, cadastrar usuários (menus 1, 2 e 3, respectivamente, na Figura 16), criar uma conta, visualizar painel geral das contas, atualizar dados e sair do sistema (menus 4, 5, 6 e 7, respectivamente, na Figura 17).

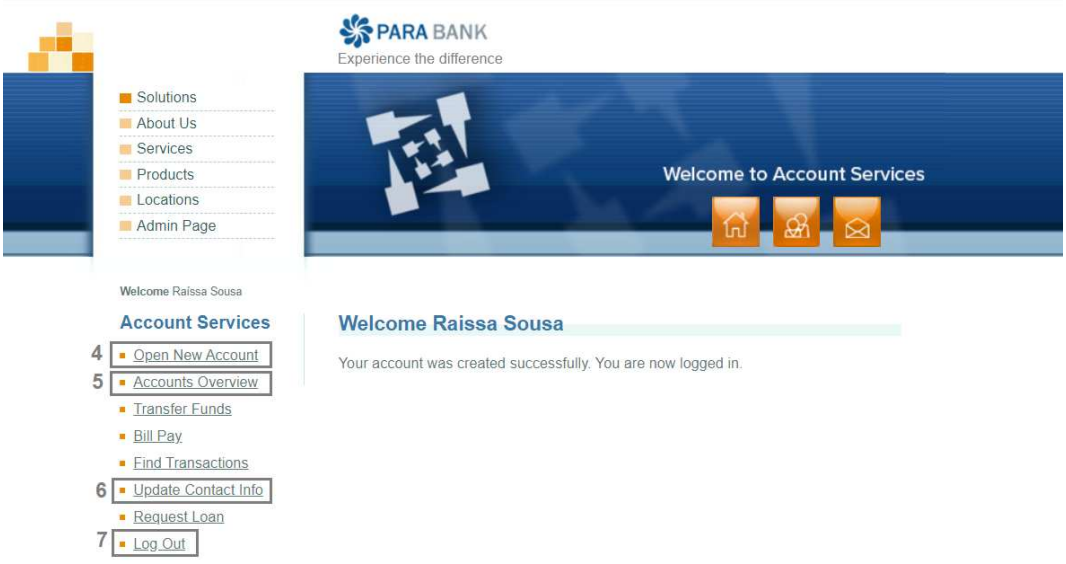
Logo, com o SUT definido determinamos os principais requisitos funcionais abordados e cobertos na implementação dos testes automatizados. Na Tabela 8 são elencados e descritos os requisitos que limitam o escopo dos testes realizados neste estudo. Os requisitos foram baseados nas funcionalidades selecionadas para a implementação dos testes.

[22] Domínio Web do Parabank:https://parabank.parasoft.com/parabank/index.htm
[23] Repositório Github do Parabank:https://github.com/parasoft/parabank

Figura 16 – Menus da tela inicial da aplicação Web Parabank

Fonte: Elaborada pelo autor.

Figura 17 – Menus da tela principal da aplicação Web Parabank



Fonte: Elaborada pelo autor.

Com os requisitos de funcionamento bem definidos, foram elaborados os casos de uso que determinam o comportamento básico da aplicação para atender a estes requisitos, estes são elencados na Tabela 9. Em seguida, para cada caso de uso foram determinados alguns casos de testes. A Tabela 10 exibe todos os casos de testes que foram documentados previamente para orientar a implementação e execução dos testes automatizados no SUT.

Tabela 8 – Requisitos funcionais do SUT

Identificador	Descrição
RF01	Permitir o cadastro de novos clientes.
RF02	Assegurar que todos os dados do cadastro de novos clientes são válidos e completos.
RF03	Permitir a validação do acesso de clientes cadastrados.
RF04	Identificar problemas com a validação do acesso de clientes cadastrados.
RF05	Permitir recuperar dados de acesso de clientes cadastrados.
RF06	Garantir a segurança dos dados dos clientes ao tentar recuperar informações de acesso.
RF07	Permitir atualizar os dados cadastrais de clientes.
RF08	Permitir a criação de contas bancárias para clientes cadastrados.
RF09	Permitir ao cliente acessar as informações de movimentação em sua(s) conta(s).
RF10	Permitir ao cliente realizar movimentações em sua(s) conta(s).

Fonte: Elaborada pelo autor.

Tabela 9 – Casos de Uso do SUT

Identificador	Título	Objetivo
UC01	Cadastro de Usuários	Cadastro de clientes na aplicação ao fornecer dados válidos e completos.
UC02	Login de Usuários	Autenticação de clientes cadastrados na aplicação.
UC03	Encontrar informações de login	Recuperação dos dados de acesso de clientes cadastrados ao fornecer dados válidos.
UC04	Atualizar dados	Atualização dos dados de clientes cadastrados.
UC05	Criar uma conta	Criar contas bancárias para clientes cadastrados.
UC06	Visualizar painel geral das contas	Monitorar dados e informações de movimentações da(s) conta(s) de clientes cadastrados.
UC07	Sair do sistema	Encerrar a sessão de clientes cadastrados.

Fonte: Elaborada pelo autor.

Para cada caso de teste foram definidos os objetivos, os dados necessários, os passos e resultados esperados de acordo com a utilização do SUT. Por exemplo, o caso de teste **TC01 - Cadastro de usuário com sucesso** que realiza o cadastro de um cliente na aplicação Parabank é exibido na Tabela 11. Após a criação e especificação de todos os casos de teste de acordo com o exemplificado na Tabela 11, os testes foram implementados e executados de forma automatizada utilizando as ferramentas `Selenium` e `Cypress`. No Apêndice B deste trabalho é apresentada uma tabela com todos os casos de testes realizados neste estudo, detalhando as especificação de cada um deles.

Tabela 10 – Casos de Testes do SUT

Casos de Uso	Requisitos Funcionais	Casos de Testes
UC01	RF01 e RF02	TC01 - Cadastro de usuário com sucesso
		TC02 - Cadastro de usuário com todos os campos vazios
		TC03 - Cadastro de usuário com um username já cadastrado
		TC04 - Cadastro de usuário com campos não obrigatórios vazios
		TC05 - Cadastro de usuário com senhas diferentes
		TC06 - Cadastro de usuário com campos obrigatórios vazios
UC02	RF03 e RF04	TC07 - Login com sucesso
		TC08 - Login com todos os campos vazios
		TC09 - Login com usuário não cadastrado
		TC10 - Login com senha errada
		TC11 - Login com campo de usuário vazio
		TC12 - Login com campo de senha vazio
UC03	RF05 e RF06	TC13 - Encontrar informações de login com sucesso
		TC14 - Encontrar informações de login com todos os campos vazios
		TC15 - Encontrar informações de login com dados inconsistentes
		TC16 - Encontrar informações de login com usuário não cadastrado
		TC17 - Encontrar informações de login com campos obrigatórios vazios
UC04	RF07	TC18 - Atualizar dados com sucesso
		TC19 - Atualizar dados com todos os campos vazios
		TC20 - Atualizar dados com campos obrigatórios vazios
		TC21 - Atualizar dados com o campo telefone (não obrigatório) vazio
UC05	RF08	TC22 - Criar uma conta com sucesso
		TC23 - Visualizar detalhes de uma nova conta
UC06	RF09	TC24 - Visualizar painel geral das contas com sucesso
		TC25 - Exibir lista de contas com sucesso
		TC26 - Visualizar detalhes de uma conta com sucesso
		TC27 - Verificar dados da conta
		TC28 - Visualizar atividades da conta
		TC29 - Visualizar lista de transações da conta
UC07	RF10	TC30 - Realizar logout

Fonte: Elaborada pelo autor.

Tabela 11 – Especificação do caso de teste TC01

TC	Descrição	Dados do Teste	Passos	Resultado Esperado
TC01	Verifica se o usuário consegue realizar o cadastro com sucesso na aplicação após fornecer todos os dados válidos.	- Acesso à aplicação ParaBank. - Dados para cadastro de clientes: Nome, Endereço (Rua, Estado, Zipe Code), Telefone e SSN.	1. Abrir a aplicação ParaBank.	1. Exibir página inicial do ParaBank.
			2. Verificar o menu Register.	2. O menu Register deve ser exibido.
			3. Clicar em Register.	3. A aplicação exibe a tela de cadastro de clientes.
			4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm.	4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos.
			5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm.	5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm.
			6. Clicar no botão Register.	6. O login deve ser feito e a aplicação exibe na tela a mensagem: Your account was created successfully. You are now logged in.

Fonte: Elaborada pelo autor.

4.2.2 Instalação e configuração

Para instalar e configurar cada uma das ferramentas em estudo foram necessários realizar alguns passos no ambiente de desenvolvimento utilizado para a implementação e execução dos testes. Para tornar a comparação prática deste estudo mais precisa, foram realizados esforços para tornar todos os processos mais semelhantes possíveis em ambas as ferramentas. Logo, foram utilizados os mesmos requisitos iniciais para a instalação: o Node.js e o editor de código *Visual Studio Code*. Bem como, as mesmas especificações de ambiente e máquina de desenvolvimento. Os detalhes são exibidos na Tabela 12.

Tabela 12 – Especificações do ambiente e máquina de desenvolvimento e execução dos testes

Máquina	**Modelo**	Dell Inc. G3 3500
	Processador	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz (12 CPUs), ∼2.6GHz
	Memória	16,0 GB
	Placa de vídeo	NVIDIA GeForce GTX 1660 Ti
	Sistema operacional	Windows 10 Pro 64-bit
Outros programas	**Navegador**	Google Chrome, Versão 114.0.5735.134 ∼64 bits
	IDE	Visual Studio Code 1.79.2

Fonte: Elaborada pelo autor.

Como o `Cypress` não tem requisitos adicionais para uma instalação padrão, não foi preciso o uso de bibliotecas, mecanismos de teste ou *drivers*. Apenas com um comando de instalação a ferramenta já estava pronta para executar e gerar relatórios dos testes. Já com o `Selenium`, foi necessário utilizar de outras tecnologias para execução dos testes. Mesmo realizando uma configuração mais simples com o uso da estrutura de testes Mocha[24] para instalar os módulos necessários, foi preciso utilizar de outras tecnologias, como por exemplo, o *driver* do navegador. Destarte, podemos verificar que o `Cypress` apresentou durante este estudo empírico uma configuração e instalação mais simples e rápida.

[24] Mocha é uma estrutura de teste JavaScript executada em Node.js (https://mochajs.org/).

4.2.3 Desenvolvimento dos scripts

Ambas as ferramentas, `Selenium` e `Cypress`, apresentam seus próprios comandos e sintaxe para realizar a criação e execução dos testes automatizados. De acordo com as funcionalidades e casos de uso definidos anteriormente, foi desenvolvido um *script* para cada caso de uso. Assim, utilizando as ferramentas foram criados sete *scripts* de testes. Em cada um destes arquivos foram implementados os casos de testes correspondentes aos cenários de cada caso de uso.

As métricas avaliadas em relação ao processo de desenvolvimento dos *scripts* realizado neste trabalho são baseadas em uma análise do código e não na experiência de uso. Assim, como especificado anteriormente, as métricas de *software* analisadas são: a) SLOC; b) CLOC; c)PLOC; d) BLANK); e) contagem de caracteres totais do código. Para calcular as linhas de código dos *scripts* em ambas as ferramentas foi utilizada a extensão *VS Code Counter*[25] que realiza a contagem das linhas de código-fonte, em branco, comentadas e totais. Os resultados são exibidos na Tabela 13.

A contagem de linhas de código é uma métrica simples e confiável, porém determinar como realizar comparações válidas pode ser um problema (LAIRD; BRENNAN, 2006). Diante disto, foi considerada uma análise com base na quantidade de linhas de código necessários para as ferramentas executarem o mesmo conjunto de casos de testes. É importante evidenciar que todos os testes foram implementados com o mesmo número de passos e *assertions*[26].

A Tabela 13 mostra que para ambas as ferramentas os *scripts* apresentam valores semelhantes para CLOC e BLANK, mostrando que as linhas de comentários e em branco

[25] Extensão para o VSCode que oferece o suporte para a contagem de linhas de códigos em muitas linguagens de programação (https://marketplace.visualstudio.com/items?itemName=uctakeoff.vscode-counter).

[26] *Assertion* é um conceito-chave em testes, as asserções são usadas para declarar o comportamento esperado ou o resultado de um teste

Tabela 13 – Métricas do desenvolvimento dos *scripts* de testes em Selenium e Cypress

SLOC								
Ferramenta de Teste	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**	**UC06**	**UC07**	**Total**
Selenium	148	78	106	118	79	179	31	739
Cypress	137	65	96	89	52	143	21	603
CLOC								
Ferramenta de Teste	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**	**UC06**	**UC07**	**Total**
Selenium	7	8	6	9	7	12	3	52
Cypress	7	7	6	5	6	11	3	45
BLANK								
Ferramenta de Teste	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**	**UC06**	**UC07**	**Total**
Selenium	3	1	2	8	6	14	2	36
Cypress	4	2	2	6	6	13	2	35
PLOC								
Ferramenta de Teste	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**	**UC06**	**UC07**	**Total**
Selenium	158	87	114	135	92	205	36	827
Cypress	148	74	104	100	64	167	26	683
Quantidade de caracteres								
Ferramenta de Teste	**UC01**	**UC02**	**UC03**	**UC04**	**UC05**	**UC06**	**UC07**	**Total**
Selenium	12103	5377	8680	9125	5733	13070	1742	55830
Cypress	6938	3327	7879	5226	2791	8029	1048	35238

Fonte: Elaborada pelo autor.

foram equiparadas para facilitar a análise do código. As métricas CLOC e BLANK são usadas geralmente para uma análise da quantidade de documentação dentro do código e espaçamentos, são sensíveis à formatação logicamente irrelevante e convenções de estilo. Portanto, são mais subjetivas e não evidenciam necessariamente uma conclusão acerca do código. Os valores das métricas SLOC e PLOC apresentam um contraste maior. A métrica PLOC evidencia a quantidade bruta de linhas de um código, contendo linhas em branco e comentadas. Já a métrica SLOC apresenta um comparativo interessante sobre os *scripts*. Os resultados mostram que para todos os *scripts* das UCs, o `Selenium` apresenta um número maior de linhas de código. Destarte, o estudo evidencia que para realizar o mesmo conjunto de testes elaborado neste trabalho, o `Selenium` precisa de mais linhas de instruções que o `Cypress`.

Na Tabela 13 também são exibidos os resultados das quantidades de caracteres em cada *script* de teste. Para obter esses dados foram desconsiderados as linhas em branco e

comentadas. Analisando os valores individuais e totais é possível perceber que o `Selenium` possui comandos com mais verbosidade[27] para realizar os mesmos testes comparado ao `Cypress`. Em (TOOMIM *et al.*, 2004) é destacado que redundância e verbosidade podem dificultar a compreensão de um código e obscurecer informações significativas. Por outro lado, comandos pouco descritivos também podem causar essa dificuldade. No estudo empírico realizado foi observado que com o `Selenium` era necessário utilizar instruções mais longas do que no `Cypress` para realizar a mesma ação, por exemplo, nos Listings 4.2.1 e 4.2.2 são exibidos os comando para verificar a exibição da mensagem *"Welcome username"* na página.

```
1 assert.equal("Welcome "+ USERNAME, await
  ↪  driver.findElement((By.className("title"))).getText())
```

Listing 4.2.1 – Comando no Selenium para verificar a exibição de um elemento

```
1 cy.get(".title").contains("Welcome "+ USERNAME)
```

Listing 4.2.2 – Comando no Cypress para verificar a exibição de um elemento

4.2.4 Execução dos testes

O tempo total de execução de testes automatizados é uma métrica bem relativa, contudo, pode auxiliar o acompanhamento do progresso geral da execução do teste. Como os valores podem variar de acordo com o equipamento em que é executado, foi utilizada a mesma máquina e ambiente para execução dos *scripts* de teste, as especificações estão descritas na Seção 4.2.2. Existem muitos fatores que podem influenciar no resultado da medição do tempo, como por exemplo a interferência de processos executados na máquina ou até a latência da rede, tentando minimizar a ocorrência destes desvios a execução em cada uma das ferramentas foi realizada algumas vezes. A Tabela 14 mostra o tempo médio de execução para cada *script* de

[27] Verbosidade, qualidade do que é verboso, que fala muito ou usa palavras em excesso para se expressar.

teste utilizando o `Selenium` e o `Cypress`.

Tabela 14 – Tempo de execução dos testes em Selenium e Cypress

UC	**Tempo de Execução Médio (s)**	
	Selenium	**Cypress**
UC01	20	25
UC02	12	13
UC03	19	18
UC04	24	22
UC05	17	15
UC06	27	28
UC07	10	5
Total	129	126

Fonte: Elaborada pelo autor.

Logo, descobriu-se que o tempo total de execução para este conjunto de testes não apresentou uma diferença significativa entre o `Selenium` e o `Cypress`. Com base na Tabela 14, as diferenças encontradas foram uma média de apenas $3s$ no tempo total de execução. Todavia, como a aplicação em teste não está em um ambiente controlado a mesma apresentou comportamentos inesperados algumas vezes. Desta forma, o tempo de execução para este estudo pode não evidenciar de fato o desempenho real das ferramentas.

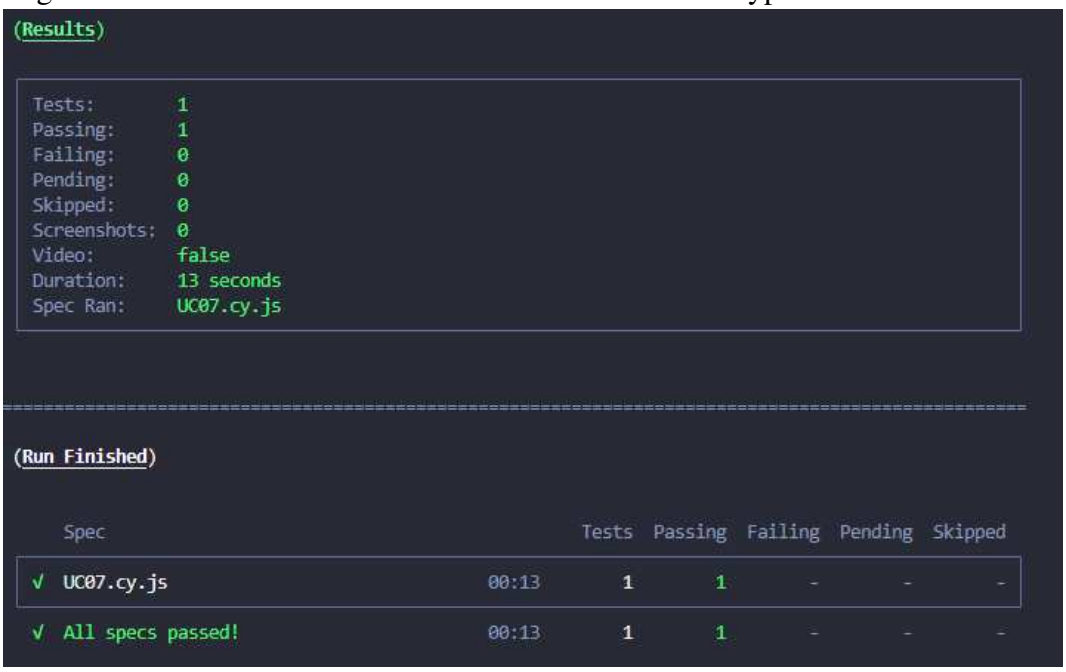
4.2.5 Relatórios de testes

Ao fim de cada ciclo de teste é escrito um relatório, que consiste na descrição de como foi a execução do teste, apontando os resultados e quais os possíveis problemas que ocorreram (MENDEZ, 2011). Tanto o `Selenium` quanto o `Cypress` exibem ao final da execução dos testes um relatório no próprio terminal da IDE do VSCode. Esses resultados gerados em ambas as ferramentas também apresentam informações úteis para rastreio de falhas no código. As Figuras 18 e 19 mostram, respectivamente, um exemplo destes relatórios de testes gerados utilizando o `Selenium` e o `Cypress`.

Figura 18 – Relatório de testes via terminal utilizando o Selenium

Fonte: Elaborada pelo autor.

Figura 19 – Relatório de testes via terminal utilizando o Cypress



Fonte: Elaborada pelo autor.

Ambas as ferramentas também oferecem suporte aos modelos de relatórios mais visuais, como por exemplo, o *Mochawesome*[28] que consiste em um modelo de relatório personalizado para uso em estrutura de teste Javascript. Logo, avaliando da perspectiva da geração de relatórios as ferramentas em estudo se mostram bastante eficientes, uma vez que, permitem obter os resultados, rastrear falhas e personalizar a visualização e estilos do relatório.

[28] https://www.npmjs.com/package/mochawesome

4.3 Lições aprendidas

Esta seção discute algumas características acerca das ferramentas `Selenium` e `Cypress` com base nas lições aprendidas durante a realização deste estudo. De acordo com os resultados obtidos foram elencados alguns tópicos para apresentar e descrever as lições. É importante mencionar que em cada tópico são abordadas perspectivas dos pontos fortes e fracos de cada ferramenta considerando os resultados deste estudo. Logo, isso não significa necessariamente que uma ferramenta seja melhor que outra.

I. **Sintaxe para desenvolvimento dos *scripts*:** As ferramentas em estudo apresentam características diferentes quanto a sintaxe adotada. Enquanto o `Cypress` utiliza instruções de código curtas e objetivas, no `Selenium` os comandos são mais densos. De fato, existem estudos que associam a verbosidade à menor legibilidade de um código de *software*, destacando que pode obscurecer informações significativas no código, tornando-o difícil de entender. Por outro lado, comandos curtos e sem objetividade também podem prejudicar a compreensão de um *script*. Nessa perspectiva, embora o `Selenium` detenha *scripts* mais verbosos que o `Cypress`, foram verificados em ambos que os códigos produzidos são legíveis e objetivos. Como por exemplo, os comandos $cy.contains(id).should('be.visible')$ e $driver.findElement(By.id()).isDisplayed()$, usados respectivamente no `Selenium` e `Cypress`, deixam claro a ação executada de procurar um elemento pelo *id* e a verificar se é exibido na página.

II. **Limitações ao testar aplicações Web dinâmicas:** Durante o estudo foram observadas algumas limitações ao utilizar ambas as ferramentas para automatizar os testes no SUT escolhido, uma aplicação Web dinâmica. Com o `Selenium` foi observado uma dificuldade maior com o manuseio de elementos dinâmicos como menus *dropdown*. Além disso, a

ferramenta também apresentou limitações relacionadas à limpeza de elementos de *input*. Já com o `Cypress` foi analisado uma demora maior ao trabalhar com formulários, fato que pode ser explicado pelos conceitos de espera automática da ferramenta. Como também, existe a dificuldade em lidar com a manipulação de várias guias e a solução para isto no `Cypress` pode ser uma tarefa complicada, contudo não foi preciso para os testes com o SUT deste trabalho.

III. **Configurações de esperas para assegurar sucesso:** Durante a execução dos testes, o `Cypress` espera automaticamente que o DOM seja carregado. Característica muito interessante para aplicação de testes otimizados sem a necessidade de implementar ou configurar esperas explícitas ou implícitas. Já com o `Selenium`, algumas vezes é necessário fazer este tipo de declaração de espera para carregar os elementos e garantir que o teste não falhe, muitas vezes aumentando o custo de tempo para o desenvolvimento.

IV. **Replicabilidade em ambientes diferentes:** O `Selenium` representa uma maior flexibilidade em termos de escolha de diferentes ambientes com suporte às linguagens, sistemas operacionais e navegadores variados. Já o `Cypress` se limita às definições de linguagem e do ambiente de execução. Contudo, quando se trata de replicação de testes em ambientes diferentes o `Cypress` apresenta uma vantagem significativa. Ao tentar replicar a execução dos testes no SUT em diferentes navegadores foi observado um grande esforço utilizando os *scripts* do `Selenium`, sendo necessário algumas modificações e configurações. Por outro lado, com o `Cypress` foi possível executar em todos os navegadores suportadas e listados na interface da ferramenta apenas com um clique, como também definido o parâmetro *–browser* na linha de comando.

5 TRABALHOS RELACIONADOS

O Selenium já é uma ferramenta bem consolidada no mercado e utilizada em muitas empresas e ambientes de aprendizagem voltados à área de testes de *softwares*. Segundo o estudo realizado por (VELOSO *et al.*, 2010) é uma das ferramentas mais adequadas para a realização de testes funcionais automatizados. Além disso, como exemplificado nos estudos de (FOWLER, 2018), muitas outras ferramentas são baseadas no Selenium, ou seja, são construídas sobre tal ferramenta. No entanto, de acordo com alguns estudos (VILA *et al.*, 2017), (BULLA; BRUNDA, 2016), (MANE *et al.*, 2016), o Selenium apresenta algumas limitações significativas ao trabalhar com aplicações web dinâmicas e modernas. Podendo implicar em problemas de desempenho e confiabilidade da ferramenta, conforme mencionado por (COLANTONIO, 2014).

Surgindo como uma alternativa ao Selenium, temos a ferramenta Cypress. De acordo com (CAPELLINI, 2021), o Cypress é uma ferramenta atual e poderosa para realizar testes, totalmente baseada em uma nova arquitetura isenta do Selenium. Segundo (CYPRESS, 2023) a ferramenta é usada principalmente por desenvolvedores ou engenheiros de controle de qualidade. Podendo ser realizados diferentes tipos de testes em aplicativos Web que utilizam estruturas JavaScript modernas, dentre eles testes de ponta a ponta, testes de componentes, testes de integração e testes de unidade. De acordo com (MOBARAYA *et al.*, 2019), o Cypress fornece uma visão promissora para o futuro da automação de testes, facilitando os processos de configuração e entregando um código melhor e mais limpo.

Em (MOBARAYA *et al.*, 2019) foi realizado uma análise técnica das ferramentas Selenium e Cypress, quanto a utilização das mesmas na automação de testes em aplicações Web modernas. Propuseram uma comparação com base nos resultados da execução do teste em Selenium e Cypress, levando em consideração parâmetros como tempo de execução do teste,

eficiência do teste e cobertura do teste baseada em requisitos. O trabalho afirma a importância do `Selenium`, consolidado no mercado há muitos anos, e o ressalta como uma ferramenta poderosa devido à sua enorme comunidade e suporte. Por outro lado, aponta o `Cypress`, como uma ferramenta bastante promissora e poderosa para testar aplicativos da Web dinâmicos, modernos e complexos.

Em (GONZALEZ *et al.*, 2022) foi realizado um estudo comparativo entre diferentes ferramentas de automação de testes para aplicações Web, com o objetivo de analisar as mesmas como uma alternativa ao `Selenium`. Confrontando a posição de destaque e preferência do `Selenium` no mercado, elencando comparações com as ferramentas: `Cypress`, `Robot Framework` e `Cucumber`. A partir disto, o estudo evidencia que o `Selenium` não é uma ferramenta insuperável, como também mostra que existem outras ferramentas que podem se destacar em alguns aspectos. Propõem também, estudos aprofundados do `Cypress` como ferramenta de automação, analisando as possibilidades de crescimento e o efeito no mercado de testes.

6 CONCLUSÕES E TRABALHOS FUTUROS

Atualmente inúmeras aplicações são desenvolvidas e utilizadas corriqueiramente pela sociedade. Na era da informação e da internet, as aplicações Web ganham pouco a pouco mais espaço e se tornam plataformas presentes nas mais variadas áreas, desde sites de entretenimento até sistemas bancários. Diante disto, como forma de assegurar a qualidade de tais aplicações, a área de testes tem atraído cada vez mais a atenção nos meios acadêmicos e empresariais. Os testes de forma geral almejam a entrega de *software* mais confiável e qualificado ao público-alvo. Logo, a otimização desse processo de garantia de qualidade é muito importante durante o desenvolvimento de um *software*, podendo significar a redução de riscos, custos e atrasos de entregas. Portanto, a automação com o uso de ferramentas geralmente é utilizada para atender tal objetivo de alcançar ciclos de testes otimizados.

Logo, o presente trabalho teve como principal objetivo realizar um estudo comparativo entres as ferramentas de automação de testes de *software*, `Selenium` e `Cypress`. A partir do levantamento de características documentadas e a realização de um estudo empírico utilizando um sistema Web proposto como ambiente de testes, foi possível identificar aspectos relevantes acerca de cada uma das ferramentas.

No primeiro momento deste estudo foram observadas algumas características das ferramentas. Os resultados evidenciaram diferenças principalmente em relação à liberdade de utilização do `Selenium` e `Cypress`. O `Cypress` oferece uma linguagem única e específica, logo sua utilização e aprendizagem se limitam aos desenvolvedores e testadores que desejam usar o JavaScript. Utilizando o `Selenium` o contexto é um pouco diferente, a ferramenta suporta as mais variadas linguagens de programação, aumentando as possibilidades para os usuários optarem pela linguagem que dominem ou tenham preferência. Já em relação ao uso em diferentes

sistemas operacionais e navegadores ambas estão equiparadas, pois oferecem suporte para os mais utilizados do mercado. Contudo, ainda existe uma vantagem para o `Selenium`, uma vez que também abrange um maior leque de possibilidades de ambientes e testes que não são suportados pelo `Cypress`, como por exemplo, testes em dispositivos móveis.

Considerando os recursos das ferramentas, foram observados que cada uma apresenta características únicas e semelhantes. O `Cypress` apresenta aspectos interessantes quanto à escrita, execução e depuração dos testes. A ferramenta possui recursos para auxiliar no rastreamento de falhas, acompanhar a execução de testes e diminuir o esforço ao trabalhar com elementos dinâmicos na Web. Por outro lado, o `Selenium` consegue utilizar múltiplas abas de um navegador e permite testes entre navegadores em escala. Ambas as ferramentas permitem execução em CI/CD e geração de relatórios de testes.

O segundo momento deste trabalho apresentou a utilização das ferramentas `Selenium` e `Cypress`, considerando um escopo de testes determinado em uma aplicação Web. Através do estudo empírico foi observado que as ferramentas atenderam a cobertura dos casos de testes levantados e apresentaram algumas distinções. Com o `Selenium` foram criados *scripts* com comandos mais verbosos, contudo bem descritivos quanto ao seu objetivo, garantindo a legibilidade do código. Já com o `Cypress` as estruturas de códigos foram curtas e objetivas, tornando o *script* de teste legível sem a necessidade de existir múltiplas palavras reservadas. Além disso, a ferramenta `Cypress` apresentou uma pequena diferença de tempo na execução sendo no geral um pouco mais rápida. Os resultados dos testes em ambas as ferramentas foram exibidos em relatórios simples e informativos.

Analisando os resultados conclui-se que o uso do `Selenium` é preferível quando demandado maior flexibilidade do ambiente de produção e execução; na realização de teste simultâneos em diferentes navegadores; testes em aplicativos móveis e recursos próprios da

ferramenta. Por outro lado, é preferível a escolha do `Cypress` quando é almejado realizar capturas de tela e de vídeo da execução do teste de forma simples; evitar instalação e configuração de dependências; replicar testes sem muito esforço e entre outros aspectos da ferramenta.

Destarte, os resultados mostraram que o `Selenium` e o `Cypress` são excelentes ferramentas para realização de testes automatizados em aplicações Web. O presente estudo não tem pretensão de determinar qual das ferramentas é mais vantajosa ou completa, uma vez que, tal conclusão está sujeita a uma série de fatores. Cabe ao utilizador averiguar qual ferramenta atende aos objetivos dos testes, ambiente de aplicação, recursos desejáveis, e outros aspectos. Logo, a contribuição deste trabalho é fornecer um embasamento para auxiliar esse processo de escolha, apresentando ambas as ferramentas de diferentes perspectivas e métricas de comparação. Deste modo, beneficiando profissionais de testes, desenvolvedores, pesquisadores e entusiastas na área de automação de testes com resultados comparativos de uma ferramenta em ascensão (`Cypress`) e outra já bem consolidada (`Selenium`).

Como planejamentos para trabalhos futuros, pode-se estender o estudo em relação a outras métricas de desempenho como a replicabilidade, execução em ambientes de integração contínua e custo de desenvolvimento e manutenção dos testes. Considerando também, um conjunto maior de testes e uma aplicação Web mais robusta. Além disso, pode-se englobar outras ferramentas de automação de testes, como por exemplo, o *Robot Framework* e *Playwright*.

REFERÊNCIAS

ANGMO, R.; SHARMA, M. Selenium tool: A web based automation testing framework. **International Journal of Emerging Technologies in Computational and Applied Science, Chandigarh**, 2014.

BAHRINI, R.; QAFFAS, A. A. Impact of information and communication technology on economic growth: Evidence from developing countries. **Economies**, MDPI, v. 7, n. 1, p. 21, 2019.

BARBOSA, E. F.; MALDONADO, J. C.; VINCENZI, A. M. R.; DELAMARO, M. E.; SOUZA, S. d. R. S. d.; JINO, M. Introdução ao teste de software. **Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software (SBES 2000)**, 2000.

BARBOSA, F.; TORRES, I. O teste de software no mercado de trabalho. **Tecnologias em Projeção**, v. 2, n. 1, 2011.

BARTIÉ, A. Garantia da qualidade de software: as melhores práticas de engenharia de software aplicadas à sua empresa. **Rio de Janeiro: Campus**, 2002.

BOARD, Q. **Standard glossary of terms used in Software Testing**. [*S. l.*]: Homepage, 2014.

BULLA, A.; BRUNDA, S. A study: Automation technique using selenium web driver. **International Journal of Research**, p. 467–470, 2016.

CAPELLINI, L. A. **Cypress: o novo conceito em testes automatizados**. 2021. Disponível em: https://atech.com.br/cypress-o-novo-conceito-em-testes-automatizados/.

CERIOLI, M.; LEOTTA, M.; RICCA, F. What 5 million job advertisements tell us about testing: a preliminary empirical investigation. In: **Proceedings of the 35th Annual ACM Symposium on Applied Computing**. [*S. l.*: *s. n.*], 2020. p. 1586–1594.

COLANTONIO, J. The# 1 killer of selenium script performance and reliability. **Retrieved from Joe Colantonio: Automation Awesomeness: https://www. joecolantonio. com/seleniumperformance-reliability**, 2014.

CYPRESS. **Why Cypress?** 2023. Disponível em: https://docs.cypress.io/guides/overview/why-cypress.

DELAMARO, M.; JINO, M.; MALDONADO, J. **Introdução ao teste de software**. [*S. l.*]: Elsevier Brasil, 2017.

DOĞAN, S.; BETIN-CAN, A.; GAROUSI, V. Web application testing: A systematic literature review. **Journal of Systems and Software**, Elsevier, v. 91, p. 174–201, 2014.

DUARTE, A. N. *et al.* Uma abordagem baseada em testes automáticos de software para diagnóstico de faltas em grades computacionais. Universidade Federal de Campina Grande, 2010.

FOWLER, M. The practical test pyramid. **martinfowler. com.[Online]. Available: https://martinfowler. com/articles/practical-test-pyramid. html**, 2018.

GARCíA, B.; GALLEGO, M.; GORTáZAR, F.; MUNOZ-ORGANERO, M. A survey of the selenium ecosystem. **Electronics**, v. 9, n. 7, 2020. ISSN 2079-9292. Disponível em: https://www.mdpi.com/2079-9292/9/7/1067.

GIL, A. C. *et al.* **Como elaborar projetos de pesquisa**. [*S. l.*]: Atlas São Paulo, 2002. v. 4.

GONZALEZ, R. S. S.; URRIBARRI, D. K.; LARREA, M. L. Automation tools for web testing. beyond selenium. **Memorias de las JAIIO**, v. 8, n. 3, 2022.

GOTARDO, R. Linguagem de programação. **Rio de Janeiro: Seses**, 2015.

HEUMANN, J. Generating test cases from use cases. **The rational edge**, v. 6, n. 01, 2001.

IEEE. Ieee standard glossary of software engineering terminology: Approved september 28, 1990, ieee standards board. In: INST. OF ELECTRICAL AND ELECTRONICS ENGINEERS. [*S. l.*], 1990.

iMasters. **Confira quais foram as linguagens de programação mais usadas em 2022**. 2023. https://imasters.com.br/noticia/confira-quais-foram-as-linguagens-de-programacao-mais-usadas-em-2022.

KAPPEL, G.; MICHLMAYR, E.; PRÖLL, B.; REICH, S.; RETSCHITZEGGER, W. Web engineering–old wine in new bottles? In: SPRINGER. **Web Engineering: 4th International Conference, ICWE 2004, Munich, Germany, July 26-30, 2004. Proceedings 4**. [*S. l.*], 2004.

KHETARPAL, A. **What is Cypress? Cypress Architecture, Features and Introduction**. TOOLSQA, 2021. Disponível em: https://www.toolsqa.com/cypress/what-is-cypress/.

KOSINDRDECHA, N.; DAENGDEJ, J. A test case generation technique and process. In: **International Workshop on Design, Evaluation and Refinement of Intelligent Systems (DERIS2010)**. [*S. l.*: *s. n.*], 2010. p. 59.

LAIRD, L. M.; BRENNAN, M. C. **Software measurement and estimation: a practical approach**. [*S. l.*]: John Wiley & Sons, 2006.

LEFFINGWELL, D. **Agile software requirements: lean requirements practices for teams, programs, and the enterprise**. [*S. l.*]: Addison-Wesley Professional, 2010.

MACHADO, O. **Ferramentas de testes de aceitação: Uma Odisséia.** Medium, 2017. Disponível em: https://medium.com/@otavio/ferramentas-de-testes-de-aceita%C3%A7%C3%A3o-uma-odiss%C3%A9ia-8eff4c96da9e.

MAHAJAN, P.; SHEDGE, H.; PATKAR, U. Automation testing in software organization. **International Journal of Computer Applications Technology and Research**, v. 5, n. 4, p. 198–201, 2016.

MANE, D.; BHADEKAR, G.; SALUKHE, S. Text and keyword driven automation testing using selenium web driver. **International Research Journal of Engineering and Technology**, p. 515–519, 2016.

MANSOUR, N.; HOURI, M. Testing web applications. **Information and Software Technology**, v. 48, n. 1, 2006. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584905000297.

MENDEZ, S. **The Test Process and Important Testing Techniques**. 2011.

MOBARAYA, F.; ALI, S. *et al.* Technical analysis of selenium and cypress as functional automation framework for modern web application testing. In: **9th International Conference on Computer Science**. [*S. l.*: *s. n.*], 2019.

MOLINARI, L. **Testes de Software: Produzindo Sistemas melhores e mais confiáveis**. [*S. l.*: *s. n.*], 2008. v. 4.

MWAURA, W. **End-to-End Web Testing with Cypress: Explore techniques for automated frontend web testing with Cypress and JavaScript**. [*S. l.*]: Packt Publishing Ltd, 2021.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. [*S. l.*]: John Wiley & Sons, 2011.

NETO, A.; CLAUDIO, D. Introdução a teste de software. **Engenharia de Software Magazine**, v. 1, p. 22, 2007.

PATUCI, G. d. O. **Ferramentas de teste de software - Revista Engenharia de Software Magazine 37**. 2011. https://www.devmedia.com.br/ferramentas-de-teste-de-software-revista-engenharia-de-software-magazine-37/21424.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de software-9**. [*S. l.*]: McGraw Hill Brasil, 2021.

RAMYA, P.; SINDHURA, V.; SAGAR, P. V. Testing using selenium web driver. **2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)**, p. 1–7, 2017.

ROCHA, A. R. C. da. **Qualidade de software: teoria e prática**. [*S. l.*]: Prentice Hall, 2001.

SANTOS, A. B. **Which web browser should you be using 2023**. 2023. https://softwarekeep.com/blog/which-web-browser-should-you-be-using-in-2021-updated.

Selenium. **The selenium browser automation project**. 2023. https://www.selenium.dev/documentation/.

SOMMERVILLE, I. **Engenharia de software**. [*S. l.*]: Editora PEB, São Paulo, 2011.

SOUZA, K. P. d.; GASPAROTTO, A. M. S. A importância da atividade de teste no desenvolvimento de software. **XXXIII Encontro Nacional de Engenharia de Produção A Gestão dos Processos de Produção e as Parcerias Globais para o Desenvolvimento Sustentável dos Sistemas Produtivos [Internet].[citado em 2019 mar. 29]**, 2013.

TAKY, M. T. Automated testing with cypress. 2021.

TOOMIM, M.; BEGEL, A.; GRAHAM, S. L. Managing duplicated code with linked editing. In: IEEE. **2004 IEEE Symposium on Visual Languages-Human Centric Computing**. [*S. l.*], 2004. p. 173–180.

UMAR, M. A. Comprehensive study of software testing: Categories, levels, techniques, and types. v. 5, 11 2019.

VALENTE, M. T. Engenharia de software moderna. **Available on**, 2020.

VELOSO, J. de S.; NETO, P. d. A. dos S.; SANTOS, I. de S.; BRITTO, R. de S. Avaliação de ferramentas de apoio ao teste de sistemas de informação. **iSys-Brazilian Journal of Information Systems**, v. 3, n. 1, 2010.

VILA, E.; NOVAKOVA, G.; TODOROVA, D. Automation testing framework for web applications with selenium webdriver: Opportunities and threats. In: **Proceedings of the International Conference on Advances in Image Processing**. [*S. l.*: *s. n.*], 2017. p. 144–150.

WARDHAN, H.; MADAN, S. Study on functioning of selenium testing tool. **International Research Journal of Modernization in Engineering Technology and Science Www. Irjmets. Com@ International Research Journal of Modernization in Engineering**, p. 2582–5208, 2021.

WAZLAWICK, R. S. Análise e projeto de sistemas de informação orientados a objetos. Elsevier Editora, 2011.

WGU. **5 most popular operating systems**. 2021. https://www.wgu.edu/blog/5-most-popular-operating-systems1910.html.

APÊNDICE A – TABELA DE COMPARAÇÃO ENTRE O SELENIUM E CYPRESS.

A Tabela consiste em um resumo dos resultados discutidos na Seção **4.1 Características das ferramentas alvo do estudo**, assim como o acréscimo de alguns outros dados acerca das ferramentas de testes Selenium e Cypress obtidos através de um estudo documental.

Tabela 15 – Comparação entre as ferramentas Selenium e Cypress.

CARACTERÍSTICA		SELENIUM	CYPRESS
Open source		SIM	SIM
Sistemas Operacionais Suportados	**Windows**	SIM	SIM
	macOS	SIM	SIM
	Linux	SIM	SIM
	Android	SIM	NÃO
	iOS	SIM	NÃO
	Outros	SIM	SIM *(Fedora e Debian)*
Linguagens de Programação Suportadas	**JavaScript**	SIM	SIM
	Python	SIM	NÃO
	Java	SIM	NÃO
	C#	SIM	NÃO
	PHP	SIM	NÃO
	Ruby	SIM	NÃO
	Outras	SIM *(Kotlin e Perl)*	NÃO
Navegadores Suportados por padrão	**Google Chrome**	SIM	SIM
	Mozila Firefox	SIM	SIM
	Microsoft Edge	SIM	SIM
	Opera	SIM	N/C
	Safari	SIM	N/C
	Brave	NÃO *(Mas configurável)*	SIM
	Outros	SIM *(Internet Explorer)*	SIM *(Electron e WebKit (Experimental))*
Dependências de Drivers		SIM	NÃO
Multi-abas		SIM	NÃO
Múltiplas instâncias do navegador		SIM	NÃO
Viagem no tempo		NÃO	SIM
Capturas de tela e vídeo disponíveis por padrão		NÃO *(Mas configurável)*	SIM
Documentação	**Completa**	NÃO	SIM
	Internacionalizada	SIM	SIM
Execução Remota		SIM	NÃO
Teste Paralelo		SIM	SIM
Uso em CI/CD		SIM	SIM
Relatório de Execução		SIM	SIM

APÊNDICE B – ESPECIFICAÇÃO DOS TESTES DO *SUT*

A Tabela consiste na documentação para especificar todos os casos de testes que foram implementados e executados neste estudo. Nas colunas **TC** (do inglês, *Test Case*) e **Título**, são exibidos o identificador e o título dos casos de testes. Em seguida são detalhados os passos para a execução de cada caso de teste e os resultados esperados de cada passo, respectivamente, em **Passos** e **Resultado Esperado**. A coluna identificada como **UC** (do inglês *User Case*) exibe o caso de uso correspondente ao teste.

UC	TC	Título	Passos	Resultado Esperado
UC01 - Cadastro de Usuários	TC01	Cadastro de usuário com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Register. 3. Clicar em Register. 4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 6. Clicar no botão Register.	1. Exibir página inicial do ParaBank. 2. O menu Register deve ser exibido. 3. A aplicação exibe a tela de cadastro de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 6. O login deve ser efetuado e a aplicação exibe na tela a mensagem: “Welcome <nome-do-usuário> Your account was created successfully. You are now logged in.”
	TC02	Cadastro de usuário com todos os campos vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Register. 3. Clicar em Register. 4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 5. Deixar vazios os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 6. Clicar no botão Register.	1. Exibir página inicial do ParaBank. 2. O menu Register deve ser exibido. 3. A aplicação exibe a tela de cadastro de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 5. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem permanecer vazios. 6. A aplicação exibe uma mensagem de “<nome-campo> is required” para os campos: First name, Last name, Address, City, State, Zip Code, Social Security Number, Username, Password e Password confirmation.
	TC03	Cadastro de usuário com um username já cadastrado	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Register. 3. Clicar em Register. 4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 6. Clicar no botão Register. 7. Localizar o menu de Log Out. 8. Clicar no menu de Log Out. 9. Verificar a exibição do menu Register. 10. Clicar em Register. 11. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 12. Inserir os mesmos dados válidos do usuário cadastrado anteriormente nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 13. Clicar no botão Register.	1. Exibir página inicial do ParaBank. 2. O menu Register deve ser exibido. 3. A aplicação exibe a tela de cadastro de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 6. O login deve ser efetuado e a aplicação exibe na tela a mensagem: “Welcome <nome-do-usuário> Your account was created successfully. You are now logged in.” 7. O menu de Log Out deve ser exibido. 8. A aplicação encerra a sessão e retorna para a tela inicial de login do cliente. 9. O menu Register deve ser exibido. 10. A aplicação exibe a tela de cadastro de clientes. 11. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 12. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 13. A aplicação exibe uma mensagem de alerta: “This username already exists.”
	TC04	Cadastro de usuário com campos não obrigatórios vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Register. 3. Clicar em Register. 4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code, SSN, Username, Password e Confirm. 6. Deixar o campo Phone não obrigatório vazio. 7. Clicar no botão Register.	1. Exibir página inicial do ParaBank. 2. O menu Register deve ser exibido. 3. A aplicação exibe a tela de cadastro de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, SSN, Username, Password e Confirm. 6. O campo Phone deve permanecer vazio. 7. O login deve ser efetuado e a aplicação exibe na tela a mensagem: “Welcome <nome-do-usuário> Your account was created successfully. You are now logged in.”
	TC05	Cadastro de usuário com senhas diferentes	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Register. 3. Clicar em Register. 4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone e Username. 6. Inserir senhas diferentes nos campos Password e Confirm. 7. Clicar no botão Register.	1. Exibir página inicial do ParaBank. 2. O menu Register deve ser exibido. 3. A aplicação exibe a tela de cadastro de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone e Username. 6. Os dados (diferentes) devem ser inseridos nos campos Password e Confirm. 7. A aplicação exibe uma mensagem de alerta “Passwords did not match.” ao lado do campo Confirm.
	TC06	Cadastro de usuário com campos obrigatórios vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Register. 3. Clicar em Register. 4. Verificar a exibição da tela de cadastro com os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm. 5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, Username, Password e Confirm. 6. Deixar o campo obrigatório vazio (SSN). 7. Clicar no botão Register.	1. Exibir página inicial do ParaBank. 2. O menu Register deve ser exibido. 3. A aplicação exibe a tela de cadastro de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code, Phone, SSN, Username, Password e Confirm devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code, Phone, Username, Password e Confirm. 6. O campo obrigatório SSN deve permanecer vazio. 7. A aplicação exibe uma mensagem de alerta “Social Security Number is required.” ao lado do campo SSN.

UC02 - Login de Usuários	TC07	Login com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida.
	TC08	Login com todos os campos vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Não inserir nenhum dado nos campos UserName e Password. 4. Clicar no botão Login.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os campos UserName e Password permanecem vazios. 4. Na tela da aplicação deve ser exibida a mensagem: "Error! Please enter a username and password."
	TC09	Login com usuário não cadastrado	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário não cadastrado. 4. Inserir uma senha qualquer. 5. Clicar no botão Login.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. Na tela da aplicação deve ser exibida a mensagem: "Error! An internal error has occurred and has been logged."
	TC10	Login com senha errada	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha errada. 5. Clicar no botão Login.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. Na tela da aplicação deve ser exibida a mensagem: "Error! An internal error has occurred and has been logged."
	TC11	Login com campo de usuário vazio	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Deixar o campo UserName vazio. 4. Inserir a senha válida. 5. Clicar no botão Login.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. O campo UserName deve permanecer vazio. 4. Os dados de Password devem ser inseridos no campo. 5. Na tela da aplicação deve ser exibida a mensagem: "Error! Please enter a username and password."
	TC12	Login com campo de senha vazio	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Deixar o campo Password vazio. 5. Clicar no botão Login.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. O campo UserName deve permanecer vazio. 4. Os dados de Password devem ser inseridos no campo. 5. Na tela da aplicação deve ser exibida a mensagem: "Error! Please enter a username and password."
UC03 - Encontrar informações de login	TC13	Encontrar informações de login com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Forgot login info. 3. Clicar em Forgot login info. 4. Verificar a exibição da tela de pesquisa de cliente com os campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 5. Inserir dados válidos de um usuário já cadastrado nos campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. Clicar no botão "Find My Login Info".	1. Exibir página inicial do ParaBank. 2. O menu Forgot login info deve ser exibido. 3. A aplicação exibe a tela de pesquisa de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code e SSN devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. A pesquisa deve ser efetuada e a aplicação exibe na tela a mensagem de “Your login information was located successfully. You are now logged in.i” e os dados de Username e Password.
	TC14	Encontrar informações de login com todos os campos vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Forgot login info. 3. Clicar em Forgot login info. 4. Verificar a exibição da tela de pesquisa de cliente com os campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 5. Deixar vazios os campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. Clicar no botão "Find My Login Info".	1. Exibir página inicial do ParaBank. 2. O menu Forgot login info deve ser exibido. 3. A aplicação exibe a tela de pesquisa de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code e SSN devem ser exibidos. 5. Os campos: First Name, Last Name, Address, City, State, Zip Code e SSN devem permanecer vazios. 6. A aplicação exibe uma mensagem de “<nome-campo> is required” para todos os campos: First Name, Last Name, Address, City, State, Zip Code e SSN.
	TC15	Encontrar informações de login com dados inconsistentes	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Forgot login info. 3. Clicar em Forgot login info. 4. Verificar a exibição da tela de pesquisa de cliente com os campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 5. Inserir dados inválidos (dados errados de um cliente cadastrado) nos campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. Clicar no botão "Find My Login Info".	1. Exibir página inicial do ParaBank. 2. O menu Forgot login info deve ser exibido. 3. A aplicação exibe a tela de pesquisa de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code e SSN devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. A aplicação exibe uma mensagem de alerta na tela: “Error! The customer information provided could not be found.”
	TC16	Encontrar informações de login com usuário não cadastrado	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Forgot login info. 3. Clicar em Forgot login info. 4. Verificar a exibição da tela de pesquisa de cliente com os campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 5. Inserir dados inválidos (de um cliente não cadastrado) nos campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. Clicar no botão "Find My Login Info".	1. Exibir página inicial do ParaBank. 2. O menu Forgot login info deve ser exibido. 3. A aplicação exibe a tela de pesquisa de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code e SSN devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 6. A aplicação exibe uma mensagem de alerta na tela: “Error! The customer information provided could not be found.
	TC17	Encontrar informações de login com campos obrigatórios vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição do menu Forgot login info. 3. Clicar em Forgot login info. 4. Verificar a exibição da tela de pesquisa de cliente com os campos: First Name, Last Name, Address, City, State, Zip Code e SSN. 5. Inserir dados válidos nos campos: First Name, Last Name, Address, City, State, Zip Code. 6. Deixar o campo obrigatório vazio (SSN). 7. Clicar no botão "Find My Login Info".	1. Exibir página inicial do ParaBank. 2. O menu Forgot login info deve ser exibido. 3. A aplicação exibe a tela de pesquisa de clientes. 4. Os campos: First Name, Last Name, Address, City, State, Zip Code e SSN devem ser exibidos. 5. Os dados devem ser inseridos nos campos: First Name, Last Name, Address, City, State, Zip Code. 6. O campo obrigatório SSN deve permanecer vazio. 7. A aplicação exibe uma mensagem de alerta “Social Security Number is required.” ao lado do campo SSN.

UC04 - Atualizar dados	TC18	Atualizar dados com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Update Contact Info”. 7. Clicar no menu “Update Contact Info”. 8. Verificar a exibição dos campos: First Name, Last Name, Address, City, State, Zip Code e Phone. 9. Atualizar os dados de endereço e telefone do usuário nos campos correspondentes (Address, City, State, Zip Code e Phone). 10. Clicar no botão de “Update Profile”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu "Update Contact Info" deve ser exibido. 7. A aplicação exibe a tela de atualizar perfil. 8. Os campos: First Name, Last Name, Address, City, State, Zip Code e Phone devem ser exibidos. 9. Os novos dados de endereço e telefone devem ser exibidos nos campos correspondentes. 10. A aplicação atualiza os dados e exibe na tela “Profile Updated”, como também uma mensagem: “Your updated address and phone number have been added to the system.”
	TC19	Atualizar dados com todos os campos vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Update Contact Info”. 7. Clicar no menu “Update Contact Info”. 8. Verificar a exibição dos campos: First Name, Last Name, Address, City, State, Zip Code e Phone. 9. Apagar todos os dados de todos os campos (First Name, Last Name, Address, City, State, Zip Code e Phone). 10. Clicar no botão de “Update Profile”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu "Update Contact Info" deve ser exibido. 7. A aplicação exibe a tela de atualizar perfil. 8. Os campos: First Name, Last Name, Address, City, State, Zip Code e Phone devem ser exibidos. 9. Os campos ficam vazios (First Name, Last Name, Address, City, State, Zip Code e Phone). 10. A aplicação exibe uma mensagem de “<nome-campo> is required” para todos os campos: First Name, Last Name, Address, City, State, Zip Code e Phone.
	TC20	Atualizar dados com campos obrigatórios vazios	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Update Contact Info”. 7. Clicar no menu “Update Contact Info”. 8. Verificar a exibição dos campos: First Name, Last Name, Address, City, State, Zip Code e Phone. 9. Atualizar os dados de endereço nos campos: Address, City, State e deixar o campo Zip Code (obrigatório) vazio. 10. Clicar no botão de “Update Profile”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu "Update Contact Info" deve ser exibido. 7. A aplicação exibe a tela de atualizar perfil. 8. Os campos: First Name, Last Name, Address, City, State, Zip Code e Phone devem ser exibidos. 9. Os novos dados de endereço devem ser exibidos nos campos correspondentes, e o campo Zip Code deve estar vazio. 10. A aplicação exibe uma mensagem de “Zip Code is required.” ao lado do campo obrigatório Zip Code.
	TC21	Atualizar dados com o campo telefone (não obrigatório) vazio	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Update Contact Info”. 7. Clicar no menu “Update Contact Info”. 8. Verificar a exibição dos campos: First Name, Last Name, Address, City, State, Zip Code e Phone. 9. Atualizar os dados de endereço nos campos: Address, City, State, Zip Code e deixar o campo Phone (não obrigatório) vazio. 10. Clicar no botão de “Update Profile”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu "Update Contact Info" deve ser exibido. 7. A aplicação exibe a tela de atualizar perfil. 8. Os campos: First Name, Last Name, Address, City, State, Zip Code e Phone devem ser exibidos. 9. Os novos dados devem ser exibidos nos campos correspondentes. 10. A aplicação atualiza os dados e exibe na tela “Profile Updated”, como também uma mensagem: “Your updated address and phone number have been added to the system.”
UC05 - Criar uma conta	TC22	Criar uma conta com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Open New Account”. 7. Clicar no menu “Open New Account”. 8. Verificar a exibição dos campos para selecionar o tipo de conta e uma conta existente para transferir fundos para a nova conta. 9. Selecionar o tipo de conta. 10. Selecionar uma conta existente para transferir fundos para a nova conta. 11. Clicar no botão de “Open New Account”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu “Open New Account” deve ser exibido. 7. A aplicação exibe a tela de abrir uma nova conta. 8. Os campos para selecionar o tipo de conta e uma conta existente para transferir fundos para a nova conta devem ser exibidos. 9. O tipo de conta selecionado é exibido no menu de seleção. 10. A conta existente selecionada para transferir fundos para a nova conta é exibida no menu. 11. A aplicação abre a conta e exibe na tela “Account Opened!”, assim como uma mensagem: “Opened! Congratulations, your account is now open ” seguida dos dados da nova conta (número da conta).

UC05 - Criar uma conta	TC23	Visualizar detalhes de uma nova conta	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Open New Account”. 7. Clicar no menu “Open New Account”. 8. Verificar a exibição dos campos para selecionar o tipo de conta e uma conta existente para transferir fundos para a nova conta. 9. Selecionar o tipo de conta. 10. Selecionar uma conta existente para transferir fundos para a nova conta. 11. Clicar no botão de “Open New Account”. 12. Clicar no número da conta criada. 13. Verificar os detalhes da conta criada.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu “Open New Account” deve ser exibido. 7. A aplicação exibe a tela de abrir uma nova conta. 8. Os campos para selecionar o tipo de conta e uma conta existente para transferir fundos para a nova conta devem ser exibidos. 9. O tipo de conta selecionado é exibido no menu de seleção. 10. A conta existente selecionada para transferir fundos para a nova conta é exibida no menu. 11. A aplicação abre a conta e exibe na tela “Account Opened!”, assim como uma mensagem: “Opened! Congratulations, your account is now open ” seguida dos dados da nova conta (número da conta). 12. A aplicação redireciona para a tela de detalhes da conta (Account Details). 13. A aplicação deve exibir os dados da conta nos campos: Account Number, Account Type, Balance e Available.
UC06 - Visualizar painel geral das contas	TC24	Visualizar painel geral das contas com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição do menu “Accounts Overview”. 7. Clicar no menu “Accounts Overview”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado. 6. O menu “Accounts Overview” deve ser exibido. 7. A aplicação exibe a tela de visão geral das contas.
	TC25	Exibir lista de contas com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição da tela geral das contas (Accounts Overview). 7. Verificar a exibição da tabela com os dados das contas do usuário (Account, Balance e Available Amount).	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. A página de Visão geral das contas deve ser exibida. 7. A aplicação exibe uma tabela com os dados das contas do usuário (Account, Balance e Available Amount).
	TC26	Visualizar detalhes de uma conta com sucesso	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição da tela geral das contas (Accounts Overview). 7. Verificar a exibição da tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. Clicar na linha correspondente à uma conta.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. A página de Visão geral das contas deve ser exibida. 7. A aplicação exibe uma tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. A aplicação redireciona para a tela de detalhes da conta (Account Details).
	TC27	Verificar dados da conta	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição da tela geral das contas (Accounts Overview). 7. Verificar a exibição da tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. Clicar na linha correspondente à uma conta. 9. Verificar os detalhes da conta exibidos.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. A página de Visão geral das contas deve ser exibida. 7. A aplicação exibe uma tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. A aplicação redireciona para a tela de detalhes da conta (Account Details). 9. A aplicação deve exibir os dados da conta nos campos: Account Number, Account Type, Balance e Available.
	TC28	Visualizar atividades da conta	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição da tela geral das contas (Accounts Overview). 7. Verificar a exibição da tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. Clicar na linha correspondente à uma conta. 9. Selecionar o período da atividade da conta no menu dropdown. 10. Selecionar o tipo de atividade (Credit ou Debit). 11. Clicar no botão “Go”.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. A página de Visão geral das contas deve ser exibida. 7. A aplicação exibe uma tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. A aplicação redireciona para a tela de detalhes da conta (Account Details). 9. A aplicação deve exibir no menu dropdown a opção selecionada para o período da atividade. 10. A aplicação deve exibir no menu dropdown a opção selecionada para o tipo de atividade (Credit ou Debit). 11. A aplicação exibe uma tabela com as informações de atividade da conta (com os campos: Date, Transaction, Debit e Credit).

UC06 - Visualizar painel geral das contas	TC29	Visualizar lista de transações da conta	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Verificar a exibição da tela geral das contas (Accounts Overview). 7. Verificar a exibição da tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. Clicar na linha correspondente à uma conta. 9. Selecionar o período da atividade da conta no menu dropdown. 10. Selecionar o tipo de atividade (Credit ou Debit). 11. Clicar no botão “Go”. 12. Clicar na linha clicável “Funds Transfer Sent” na atividade da conta.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. A página de Visão geral das contas deve ser exibida. 7. A aplicação exibe uma tabela com os dados das contas do usuário (Account, Balance e Available Amount). 8. A aplicação redireciona para a tela de detalhes da conta (Account Details). 9. A aplicação deve exibir no menu dropdown a opção selecionada para o período da atividade. 10. A aplicação deve exibir no menu dropdown a opção selecionada para o tipo de atividade (Credit ou Debit). 11. A aplicação exibe uma tabela com as informações de atividade da conta (com os campos: Date, Transaction, Debit e Credit). 12. A aplicação redireciona para a tela de detalhes da transação (Transaction Details) como os campos: Transaction ID, Date, Description, Type e Amount.
UC07 - Sair do sistema	TC30	Realizar logout	1. Abrir a aplicação ParaBank. 2. Verificar a exibição dos campos UserName e Password. 3. Inserir um nome de usuário válido. 4. Inserir a senha válida. 5. Clicar no botão Login. 6. Localizar o menu de Log Out. 7. Clicar no menu de Log Out.	1. Exibir página inicial do ParaBank. 2. Os campos UserName e Password devem ser exibidos. 3. Os dados de UserName devem ser inseridos no campo. 4. Os dados de Password devem ser inseridos no campo. 5. O login deve ser efetuado e a página de Visão geral das contas deve ser exibida. 6. O menu de Log Out deve ser exibido. 7. A aplicação encerra a sessão e retorna para a tela inicial de login do cliente.