Revista de Sistemas de Informação da FSMA

n. 29 (2022) pp. 45-59

SISTEMAS DE INFORMAÇÃO

http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

Automação de Testes de Interface Web: Uma análise comparativa entre Cypress e Playwright

Marco A. S. Segundo, Ana Paula Furtado, Departamento de Computação,
Universidade Federal Rural de Pernambuco - UFRPE

Resumo **- O mercado de software tem buscado cada vez mais a otimização em seus processos, assim como uma qualidade superior em seus produtos. Dentre os processos em busca de aprimoração, está o de Testes de Software, que contribui de forma considerável para a qualidade dos sistemas. No meio deles, há o Teste de Interface que é baseado na formação de uma sequência de eventos nos componentes de uma interface gráfica a fim de encontrar falhas. Para auxiliar neste processo, é comum a utilização de ferramentas de automação com capacidade de gerenciamento e controle de testes. Entre as principais ferramentas do mercado, está o Selenium WebDriver, que possui limitações em sua aplicação. Diante disso, o objetivo deste trabalho é a realização de uma análise comparativa entre o Cypress e o Playwright, duas ferramentas que possuem arquiteturas diferentes do Selenium, a fim de auxiliar profissionais na escolha de frameworks de automação relacionados à interface gráfica. A partir desta análise, foi possível verificar um equilíbrio entre as ferramentas, com o Cypress obtendo uma vantagem em relação a eficiência dos testes e o Playwright garantindo uma vantagem nos critérios de tempo de execução e cobertura de testes.**

Palavras chave **— Automação de Testes; Ferramenta de automação de testes; Teste de Interface Web; Cypress; Playwright;**

Abstract **- Nowadays the software market is increasingly looking to optimize in its processes and to achieve higher quality in its products. Among the processes under improvement, there is the Software Testing one, which contributes considerably to systems quality. Among them is the Interface Test, which is based on the definition of a sequence of events in the components of a graphical interface in order to search for faults.It is common to use automation tools with the ability to manage and control tests to assist this process. Among the main tools in the market there is Selenium WebDriver, which has limitations in its application. Therefore, the objective of this work is to carry out a comparative analysis between Cypress and Playwright, two tools whose architectures are different from Selenium, in order to assist professionals to choose automation frameworks related to the graphical interface. Based on this analysis, it was possible to verify a balance between the tools, with Cypress obtaining an advantage in relation to the efficiency of the tests and Playwright ensuring an advantage in the criteria of execution time and coverage of the tests.**

Index terms **— Automation Testing; Automation Testing Tools; Web Interface Test; Cypress; Playwright;**

I. INTRODUÇÃO

Como os sistemas de software estão cada vez mais presentes na vida das pessoas, tendo um alto grau de importância, o seu mau funcionamento pode acarretar sérios problemas, tais como perda de dinheiro, reputação e até mesmo a morte. Dessa forma, o Teste de software atua para diminuir esses riscos de falha do software [1]. Segundo Delamaro et al. [2], garantir que o software esteja de acordo com o que foi especificado é a finalidade dos Testes de Software.

As atividades de teste podem ser realizadas em cada etapa do processo de desenvolvimento do software. Segundo o ISTQB [1] elas são agrupadas em níveis de acordo com a etapa em que serão executadas. Dentre esses níveis, existe o teste de sistema que é responsável por verificar a conformidade de todo o sistema com os requisitos definidos no escopo do projeto. Dentro deste nível há o teste de interface, que investiga a *Graphical User Interface* (GUI). Esta representa a aparência e o comportamento dos elementos visuais da aplicação, sendo considerada essencial para o sucesso de uma aplicação, tornando crucial a sua verificação [3].

De acordo com Sommerville [4], testar o software é uma atividade custosa e árdua do processo de software com repetidas ações e grande número de possibilidades. Com isso, surgiu uma necessidade de diminuir o tempo e esforço para a realização desta ação, e os testes automáticos surgiram, simulando a interação com a aplicação através do uso de ferramentas [5].

Para a realização do processo de automação de testes é necessário o uso de ferramentas específicas capazes de gerenciar e controlar determinadas ações. Com o uso correto desses softwares é possível agilizar e facilitar a execução dos testes, que é uma das etapas mais importantes do processo de desenvolvimento de software [5].

Dentro do mercado das ferramentas de automação para testes de interface web, Selenium WebDriver é a mais popular [6]. Por ser uma ferramenta estável e poderosa, a sua arquitetura é utilizada como base para outras ferramentas e se

tornou um padrão para os frameworks de automação [7]. No entanto, ela possui limitações na manipulação dos elementos nas aplicações web mais modernas [8]. Por causa destas limitações, abre-se espaço para novas ferramentas com novas metodologias e arquiteturas como o Cypress, ferramenta de automação de testes feita para a web moderna [9] e que utiliza a técnica de injetar Javascript diretamente no browser [8], e o Playwright, ferramenta criada recentemente pela Microsoft que utiliza o protocolo DevTools para testar aplicações atuais [10].

Dentro deste contexto, a pergunta de pesquisa deste trabalho é: Entre Cypress e Playwright, qual delas apresenta o melhor resultado no contexto de testes de interface?

Por conseguinte, podemos dizer que este trabalho tem como objetivo geral realizar uma análise comparativa entre Cypress e Playwright, com o intuito de auxiliar os profissionais na tomada de decisão de qual framework utilizar no contexto de testes de interface. Ademais, como objetivos específicos, nós nos propomos a avaliar qual destes frameworks possui um menor tempo de execução dos seus scripts, melhor eficiência de testes, isto é, qual das ferramentas requer menor esforço, medido em linhas de código, para criação dos testes. E por fim, se as ferramentas contemplam as principais interações com elementos web.

Para tanto, este trabalho está estruturado em seis seções, além da introdução aqui apresentada. Na Seção 2, a metodologia proposta no trabalho é descrita de acordo com os critérios utilizados para sua definição. Já a Seção 3, possui uma breve revisão sobre testes de software e ferramentas de automação. A Seção 4 aborda os trabalhos relacionados ao escopo deste projeto, citando cerca de seis publicações, para melhor entendimento da abordagem deste estudo. Na Seção 5 são apresentadas as etapas de Planejamento desta pesquisa. Na Seção 6 apresentamos as etapas de execução da análise comparativa. Por fim, na Seção 7, temos as contribuições deste trabalho e as possibilidades de trabalhos futuros.

II. METODOLOGIA

A. Este trabalho tem como objetivo realizar um estudo comparativo entre duas ferramentas de automação de testes Web no contexto de interface. Foram definidas sete etapas para a realização deste estudo, seis das quais agrupadas em duas fases: Planejamento e Execução. As etapas e os artefatos gerados em cada uma delas estão ilustrados na Figura 1 e são descritos com mais detalhes a seguir. *Realizar revisão bibliográfica exploratória*

Segundo Gil [11], a revisão bibliográfica visa permitir ao pesquisador uma compreensão mais ampla sobre o que está sendo estudado. Durante esta etapa, foi realizada uma pesquisa exploratória em artigos científicos encontrados nas bibliotecas digitais IEEE Xplore Digital Library, Google Acadêmico e Springer Link, além de sites e livros, com o intuito de criar um referencial teórico sobre os conceitos abordados neste estudo. Esta pesquisa foi seguida por uma análise dos trabalhos comparativos descobertos que versavam sobre ferramentas de automação de testes relacionados a essa pesquisa.

B. Definir critérios de avaliação

A partir dos trabalhos relacionados encontrados na etapa anterior, foram definidos os critérios de avaliação das ferramentas de automação que foram utilizados na realização da análise comparativa.

C. Definir ferramentas

Nesta etapa foram selecionadas as duas ferramentas de automação de testes de interface web que seriam consideradas no desenvolvimento dos projetos. Onde foram consideradas ferramentas que utilizam a mesma técnica de criação de scripts de teste. Como citado anteriormente, o Selenium é a ferramenta de testes mais popular, mas possui limitações, o que abre espaço para outras ferramentas, como as duas que serão selecionadas neste projeto. Nesta etapa nós explicamos como chegamos nelas duas.

D. Definir aplicação

Levando em consideração os testes de interface web, nesta etapa foi realizada a busca por uma aplicação que contemplasse várias interações com elementos web.

E. Definir casos de teste

Com base na aplicação selecionada na etapa anterior, foram levantados os casos de teste que definiram os scripts que seriam implementados.

F. Desenvolver scripts

A partir das ferramentas, aplicação e casos de teste definidos nas etapas anteriores, foi desenvolvido um projeto em cada framework selecionado, a fim de comparar os seus resultados.

G. Realizar análise comparativa

Baseando-se nos critérios de avaliação definidos, como também nos scripts desenvolvidos, nesta etapa é realizada a análise comparativa com o intuito de avaliar a resposta obtida de cada cenário.

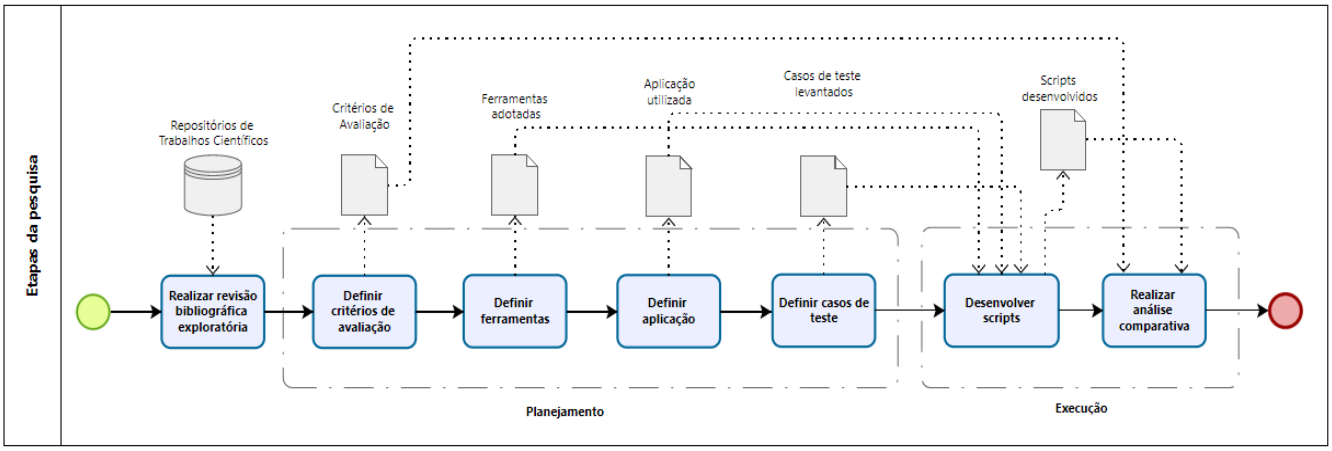
III. REVISÃO BIBLIOGRÁFICA

A. Teste de software

Atualmente, existem vários conceitos para o teste de software, mas de uma forma direta, testar um software significa verificar através de uma execução controlada se o seu comportamento transcorre de acordo com o especificado. O seu principal objetivo é garantir que o software atendeu às especificações, ou seja, mostrar aos desenvolvedores se os resultados estão ou não de acordo com os critérios estabelecidos [12].

Quando lidamos com testes, nos referimos também à qualidade do software. Não que sejam sinônimos, mas sem dúvida o nível de qualidade dos testes de um software é um dos principais fatores para definir a qualidade do produto final, que depende do processo de desenvolvimento desse software [13].

Figura 1 – Etapas da Pesquisa



Fonte: Elaborado pelo autor (2022)

B. Níveis de Teste

Os níveis de testes são conjuntos de atividades de teste que estão relacionados com as outras atividades e etapas do ciclo de desenvolvimento de software [1]. A seguir são descritos os níveis de teste de software existentes, definidos de acordo com Rios e Moreira [13].

- **Testes unitários:** também conhecido como teste de Unidade. São realizados com acesso ao código e pertencem ao nível mais baixo da escala de testes. Verificam o funcionamento de um pedaço do sistema, com o intuito de provocar falhas em cada módulo separadamente, e geralmente são realizados pelos próprios desenvolvedores.
- **Testes de integração:** caracterizado por verificar a comunicação entre os componentes, como módulos ou aplicações distintas e visam assegurar que as partes funcionem juntas corretamente.
- **Testes de sistemas:** são executados no sistema como um todo, com o intuito de verificar se o comportamento do software corresponde ao definido na especificação. Neste nível se atua fortemente na verificação dos requisitos funcionais e não funcionais.
- **Testes de aceitação:** são os testes finais e geralmente são executados pelos usuários do sistema com o objetivo de validar se o software atende à finalidade.

C. Teste de Interface

Teste de Interface, ou Teste de GUI, se baseia na realização de uma sequência de eventos - podendo ser o "Clicar no botão", o "Digitar", o "Abrir menu", entre outros - nos Widgets da interface gráfica com o intuito de encontrar falhas e/ou comportamentos adversos. A maioria dos sistemas baseados em UI dispõem de uma grande quantidade de sequência de eventos que podem ser executados. Podendo ser de maneira manual ou automática, as técnicas de teste de interface gráfica focam na redução da quantidade de entradas no intuito de viabilizar a execução desses testes [14].

Segundo Choudhary et. al. [15], é possível dividir as diversas abordagens de exploração de interface gráfica em três grupos, são eles:

- **Randômica:** Basicamente é uma estratégia que apenas provoca eventos UI, o que pode ser bastante ineficiente, uma vez que há muitos eventos que podem ser produzidos. Entretanto, pode ter utilidade para testes de estresse. Em resumo, não apresentam um critério de parada que indique o sucesso da exploração [16].
- **Baseada em Modelos:** Consistem em ferramentas que utilizam modelos de interface gráfica, geralmente máquinas de estados finitos que possuem *activities* como estados e eventos como transições. Quanto à cobertura de código, esta estratégia se mostra mais eficaz e precisa, porém, sua principal limitação é referente à maneira como as transições entre estados são realizadas, deixando de considerar eventos que alteram o estado da aplicação, mas não mudam a GUI [16].
- **Sistemática:** Por meio de técnicas mais sofisticadas, esta estratégia busca através da utilização de técnicas como execução simbólica e algoritmos evolucionários, conduzir a exploração para áreas do código as quais ainda não foram exploradas pelas outras estratégias. Entretanto, esta se mostra menos escalável quando comparada às estratégias randômicas [16].

D. Automação de testes

Segundo Tian [17], o teste automático é o processo onde ferramentas realizam os testes executando ações predefinidas, conferindo se os resultados encontrados correspondem àqueles esperados. Ademais, de acordo com Fewster e Graham [18], a automação dos testes pode influenciar diretamente no aumento do escopo dos testes, já que pode aumentar a quantidade de testes realizados em determinado espaço de tempo e ajudar a reduzir o esforço necessário para realização das atividades de teste.

Hanna et al. [19], relatam que o teste automático tem o script de teste como seu elemento básico, onde o script é um arquivo que armazena uma série de eventos e comandos necessários para executar os testes e registrar os seus resultados.

Segundo Collins e Lobão [20], as principais técnicas de criação de testes automáticos são Gravação-Execução (*Record and Playback)* e Programação de scripts (*Script Programming):*

- Gravação-Execução: consiste em utilizar uma ferramenta de automação de testes para gravar as ações executadas pelo usuário na interface gráfica da aplicação, convertendo essas ações em scripts que podem ser executados posteriormente e repetidamente.
- Programação de scripts: é necessário um conhecimento em programação para criar os scripts nesta técnica. Tem a vantagem de poder criar testes mais simples e produtivos do que os gerados pelas ferramentas Gravação-Execução, e traz benefícios como a capacidade de adicionar lógicas, cálculos e integrações entre sistemas. Por outro lado, ao ser programado, ele está sujeito a todos os problemas de uma programação, como bugs, erros de lógica etc.

E. Ferramentas de automação

Os testes de automação estão cada vez mais avançados e mais eficientes, e um dos principais motivos desse crescimento é a grande quantidade de ferramentas disponíveis no mercado, que com metodologias e arquiteturas diferentes, dão um leque de opções que podem se adequar às mais diversas necessidades dos usuários [6]. A seguir serão apresentadas algumas ferramentas de automação que foram selecionadas de acordo com a importância para este estudo:

1) Selenium

Considerada a ferramenta de automação mais popular [6], o Selenium é uma ferramenta robusta e poderosa que possui um conjunto de componentes com diferentes abordagens, fazendo com que ela atenda às necessidades da maioria dos usuários [21]. Esses componentes são:
- Selenium WebDriver: Componente bastante consolidado, é utilizado como protocolo base de diversos frameworks de automação [7]. O Selenium WebDriver é uma interface que utiliza o driver específico de cada navegador para realizar a interação entre o script de teste desenvolvido pelo usuário e o respectivo browser, permitindo a descoberta e manipulação dos elementos no documento web [22].
- Selenium IDE: Componente que utiliza a técnica de Gravação-Execução, permitindo a criação de testes de forma rápida e simples [23].
- Selenium Grid: Componente voltado para realização dos testes em máquinas diferentes de forma remota. Seu objetivo é fornecer uma maneira simples de executar os testes de forma paralela em várias máquinas [24].

2) SAHI

Segundo Naidu et al. [21], o SAHI é uma ferramenta de código aberto voltada para automação de testes utilizando a técnica de Gravação-Execução tanto em aplicações web como em desktop. Foi desenvolvida nas linguagens Java e Javascript e possui recursos como controle do navegador, scripts baseados em texto, além de relatórios e log integrados. Na gravação o SAHI funciona como um servidor proxy que intercepta o tráfego do navegador e registra as ações de sua navegação. Já na Execução, ele injeta Javascript no navegador para que os elementos possam ser acessados na página web.

3) TestComplete

O TestComplete é uma ferramenta que permite a criação, gerenciamento e execução de testes para aplicações web, desktop e mobile. Ele suporta tanto a técnica de Gravação-Execução, quanto a de Programação de scripts. O TestComplete é um framework com interface de usuário agradável, facilitando a sua utilização, e possui reprodução eficiente. [25].

4) Watir Webdriver

Watir é uma biblioteca de código aberto que oferece suporte a vários navegadores em diferentes plataformas e possui arquitetura baseada no Selenium WebDriver. Ela permite a escrita de testes com facilidade de leitura e manutenção, credenciando-se como uma ferramenta de utilização intuitiva [25].

5) Katalon Studio

O Katalon Studio (KS) é uma ferramenta que suporta a criação de testes desktop, web e mobile. Tem o Selenium como uma de suas bases arquiteturais e oferece uma interface amigável, além de possuir uma flexibilidade para criação de scripts, suportando as duas principais técnicas [26].

6) Robot Framework

Ferramenta de automação de código aberto voltada para testes de aceitação, o Robot Framework (RF) utiliza a técnica de programação de scripts, é fácil de criar testes e suporta as plataformas web, desktop e mobile [26].

7) Puppeteer

Ferramenta que utiliza a técnica de programação de scripts por meio da linguagem Javascript, o Puppeteer é um framework de automação de testes web que foi desenvolvido pela equipe do Chrome DevTools e devido a isso possui a vantagem de explorar melhor os recursos do navegador Google Chrome [10].

8) Playwright

O Playwright é um framework criado pela Microsoft, que utiliza a técnica de programação de scripts. Como parte de sua equipe de criação também participou do desenvolvimento do Puppeteer, é possível verificar diversas semelhanças entre as duas ferramentas, com o Playwright tendo como principal diferencial o suporte a múltiplas linguagens [10].

9) Cypress

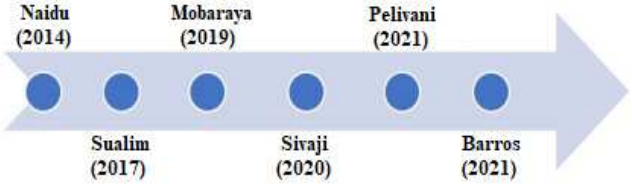
Framework que possui uma arquitetura inovadora de execução de scripts de teste em aplicativos web. Utiliza o Javascript como sua linguagem de desenvolvimento, e se destaca pela simplicidade na criação dos seus testes e na execução de testes assíncronos [8].

IV. TRABALHOS RELACIONADOS

Foi realizada uma busca nas bibliotecas digitais IEEE Xplore Digital Library, Google Acadêmico e Springer Link. com o objetivo de encontrar estudos comparativos entre

ferramentas de automação de testes de maneira aplicada, ou seja, estudos em que scripts de testes foram criados e analisados. Essa busca tem a finalidade de apresentar as diferentes táticas utilizadas em comparativos de ferramentas, auxiliando na seleção da melhor abordagem para o escopo deste projeto. A Figura 3 mostra uma linha do tempo da data de publicação desses trabalhos.

Figura 3 - Linha do tempo dos trabalhos relacionados com seus principais autores



Fonte: Elaborado pelo Autor (2022)

No trabalho de Naidu et al. [21] foi realizado um estudo comparativo das ferramentas SAHI e Selenium IDE. O experimento foi feito através da GUI de configuração de um Access Point (AP) por meio de um navegador web, onde foram levadas em consideração 11 funcionalidades, dentre elas a configuração do SSID, a alteração do nome do dispositivo e a mudança de IP. Neste estudo foram avaliados oito critérios, entre eles tempo de execução, tempo de criação e facilidade de aprendizado e os autores demonstraram que em seis deles o Selenium se sobressaiu, em somente um o SAHI levou vantagem, e no critério restante ocorreu um empate.

O objetivo de Sualim et al [25] em seu estudo foi realizar a comparação entre três frameworks de automação de testes de aceitação para aplicações web. Sendo eles: TestComplete, Selenium Webdriver e Watir Webdriver. Essa comparação foi baseada em critérios de usabilidade aplicados às ferramentas, que incluem eficiência, eficácia, satisfação e erro. A eficiência é o tempo necessário para o aprendizado e conclusão da atividade. A eficácia consiste em o usuário conseguir concluir a tarefa atingindo o objetivo. A Satisfação é o quão amigável é a ferramenta, ou seja, é o grau de dificuldade em utilizá-la e encontrar as informações importantes da aplicação. O erro consiste na taxa de identificação de erros cometidos pelo usuário ao tentar inserir alguma entrada ou não inserir quando a entrada é inválida. As funcionalidades utilizadas no experimento foram: o login em uma aplicação web e a atualização das informações pessoais neste sistema. A fim de realizar a avaliação das ferramentas, foram utilizados um conjunto de 13 perguntas. No qual foi possível para o Sualim concluir que a melhor ferramenta foi a Watir Webdriver, pois ela tem o tempo de execução mais curto e a possibilidade de desenvolvimento de scripts de teste simples, além de cumprir os critérios de usabilidade definidos.

Em sua pesquisa, Morabaya e Ali [8] realizam uma comparação entre os frameworks Selenium Webdriver e Cypress com o intuito de observar qual dos dois possui melhor eficiência na escrita e execução dos scripts de automação, e se os 2 conseguem ser eficazes cobrindo os requisitos estabelecidos para uma aplicação web moderna. O estudo é feito sobre 2 cenários de um site de e-commerce, um referente à conta do usuário e outro relativo à realização de compras de produtos, tendo sido criados 5 scripts de teste para cada um deles. Morabaya e Ali concluem que as 2 ferramentas não possuem muitas diferenças na medição do tempo de execução dos testes e conseguem cobrir os requisitos estabelecidos, porém possuem uma diferença significativa na eficiência do desenvolvimento dos scripts, com o Cypress levando vantagem.

Sivaji et al. [26], realizaram um estudo comparativo em aplicações web que são a interface com o usuário em sistemas Smart Manufacturing (SM), no qual foram utilizadas as ferramentas Katalon Studio (KS) e Robot Framework (RF). Nessa pesquisa focada em produtividade, 10 especialistas em automação de testes foram questionados sobre o tempo gasto para realizar determinadas atividades em RF, e após isso foram introduzidos em um treinamento da ferramenta KS onde eles responderam um questionário antes e depois desta capacitação a fim de medir o conhecimento adquirido e explorando essa nova ferramenta. Após isso tiveram que responder novamente o questionário sobre o tempo gasto para realizar certas atividades, desta vez na ferramenta KS. Foram levados em consideração processos como instalação, escrita de scripts, execução, geração de relatórios, entre outros. Por fim, Sivaji concluiu que o Katalon Studio se saiu melhor na taxa de produtividade considerando o contexto de SM.

Em seu estudo, Pelivani e Cico [6] fizeram uma comparação entre Selenium Webdriver e Katalon Studio (KS), aplicando técnicas de criação de testes diferentes, utilizando Gravação-Execução com o KS e Programação de Scripts com o Selenium. Foram automatizados 5 casos de teste de GUI, onde foram analisados tempo de execução, geração de relatórios, facilidade de configuração, capacidade de gravação em comparação com habilidades de programação e linguagens suportadas. Pelivani e Cico [6] concluíram o estudo apresentando uma tabela com características e recursos de cada ferramenta, pois seu intuito era analisar as diferenças e não escolher o melhor framework.

No estudo de Barros et al. [27], foi realizada uma comparação técnica entre os frameworks Selenium Webdriver e Cypress, onde foram avaliados critérios como tempo de execução e quantidade de linhas de código, sendo também abordadas diferenças mais técnicas como validação de requisições HTTP, suporte a múltiplas abas, geração de artefatos, entre outros. O que diferencia o trabalho de Barros et al. [27] do estudo de Morabaya e Ali [8], é que o primeiro abordou os testes de interface e utilizou uma aplicação web que possui vários tipos de elementos e interações voltadas para este tipo de teste, enquanto que o segundo abordou os

testes das funcionalidades da aplicação. Por mais que o intuito do trabalho seja apresentar as diferenças entre as ferramentas para ajudar na tomada de decisão dos profissionais, Barros et al. [27] relatam que o Cypress teve um melhor desempenho,é mais simples e menos custoso na criação de scripts, além de suportar nativamente várias ações que o Selenium não suporta. Porém, também destaca que o Cypress não possui suporte a múltiplas abas e janelas, limitando o uso da ferramenta em determinadas aplicações.

No Quadro 1 é possível verificar os critérios de comparação e as ferramentas utilizadas em cada trabalho relacionado a este estudo.

Os trabalhos mencionados no Quadro 1 têm em comum a utilização de pelo menos uma ferramenta que tenha relação com o Selenium, sendo o próprio framework ou alguma ferramenta que o utilize como base em sua arquitetura. Além disso, os trabalhos também apresentam comparativos entre ferramentas que possuem técnicas de escrita de scripts distintas, ou utilizam linguagens de programação diferentes. Essa distinção resulta em uma diferença considerável na métrica de eficiência dos testes, que considera o esforço necessário para a criação dos testes. Diante disso, o foco deste trabalho é apresentar uma comparação entre ferramentas que surgiram como alternativas ao Selenium, e que suportam a mesma técnica de criação de scripts e linguagem de programação.

Quadro 1 – Critérios de comparação e ferramentas utilizadas nos trabalhos relacionados

Autor Principal	Trabalho	Critérios de comparação	Ferramentas
Naidu	**Sahi vs Selenium - A comparative analysis**	1 - Tempo de criação do script; 2 - Tempo de execução do script; 3 - Facilidade de aprendizado; 4 - Instalação e configuração; 5 - Gravação e execução; 6 - Registros de execução e geração de relatórios; 7 - Custos de licença; 8 - Compatibilidade com plataformas.	- Sahi; - Selenium IDE.
Sualim	**Comparative Evaluation of Automated User Acceptance Testing Tool for Web Based Application**	1 - Eficiência da ferramenta; 2 - Eficácia da ferramenta; 3 - Satisfação do usuário com a ferramenta; 4 - Taxa de erros detectados.	- TestComplete; - Selenium; - Watir WebDriver.
Morabaya	**Technical analysis of Selenium and Cypress as functional automation framework for modern web application testing**	1 - Tempo de execução; 2 - Eficiência do teste; 3 - Cobertura do teste.	- Selenium; - Cypress.
Sivaji	**Software Testing Automation - A Comparative Study on Productivity Rate of Open Source Automated Software Testing Tools For Smart Manufacturing**	1 - Tempo de instalação; 2 - Tempo de escrita dos testes; 3 - Tempo de verificação; 4 - Tempo de execução; 5 - Tempo de interpretação dos relatórios; 6 - Tempo de manutenção do script; 7 - Tempo do teste de regressão.	- Katalon Studio; - Robot Framework.
Pelivani	**A comparative study of automation testing tools for web applications**	1 - Tempo de execução; 2 - Geração de relatórios; 3 - Facilidade de configuração; 4 - Capacidade de gravação vs habilidade de programação; 5 - Linguagens suportadas.	- Selenium; - Katalon Studio.
Barros	**Automação de Testes Funcionais Uma Análise Técnica dos Frameworks Cypress e Selenium**	1 - Tempo de execução; 2 - Quantidade de linhas de código; 3 - Simplicidade; 4 - Validação de requisições HTTP; 5 - Utilização do comando de espera; 6 - Suporte a métodos Javascript; 7 - Suporte a múltiplas abas; 8 - Documentação da ferramenta; 9 - Geração de artefatos; 10 - Suporte à execução dos testes.	- Selenium; - Cypress.

Fonte: Elaborado pelo Autor (2022)

V. PLANEJAMENTO

A. Definição dos critérios de avaliação

Tomando como base os critérios utilizados nos trabalhos relacionados a este estudo, foram definidos três critérios de comparação que são adequados ao escopo deste projeto: Tempo de execução, Eficiência do teste e Cobertura do teste.

Como verificado nos trabalhos relacionados, o tempo de execução foi utilizado em 5 dos 6 estudos revisados. Neste critério é avaliado que a ferramenta que obtiver o menor tempo de execução, será a que atinge o melhor resultado.

De acordo com Damm [28], a eficiência do teste é medida dividindo o número de defeitos encontrados em um teste pelo esforço necessário para executá-lo, medindo o custo-benefício para a organização. Sualim et al. [25], que analisaram ferramentas utilizando diferentes técnicas de criação de testes, adaptaram essa definição para o contexto de automação considerando-a como o tempo necessário para criar o script de teste. Porém, este estudo vai utilizar o conceito aplicado por Morabaya e Ali [8], que também compararam ferramentas utilizando a mesma técnica de criação de scripts. Eles adaptaram o conceito de eficiência para medir o esforço de escrever o script de teste automático, em que quanto menor o esforço para atingir o padrão de qualidade de software, mais eficiente o teste é, ou seja, quanto menos linhas o código tiver, melhor ele será.

Assim como Morabaya e Ali [8], este projeto avaliará a cobertura de testes considerando a eficácia do script diante do caso de teste correspondente. Segundo Muhammad e Suhaimi [29], a cobertura de testes é uma métrica de garantia de qualidade que a partir dos requisitos testados determina a eficácia dos testes.

B. Definição das ferramentas

Assim como os trabalhos de Morabaya e Ali [8] e Barros et al. [27], este trabalho utilizou ferramentas que suportem a técnica de programação de scripts, mas diferentemente deles foi decidido utilizar ferramentas que suportem a mesma linguagem de programação, a fim de retirar o impacto na eficiência dos testes causado pela diferença das linguagens.

Levando em conta os testes de automação realizados através de navegadores web, pode-se considerar que existem três maneiras de se comunicar e controlar o navegador [30]: por meio do protocolo Webdriver; através da injeção do Javascript diretamente no navegador; e por intermédio do protocolo DevTools [31].

Como o WebDriver é baseado no Selenium [7] e este trabalho busca apresentar alternativas a esta ferramenta, foram descartadas as ferramentas que são derivadas deste protocolo. Com isso foram consideradas as ferramentas baseadas nos dois protocolos restantes.

O Cypress é uma ferramenta inovadora que utiliza a técnica de injetar Javascript diretamente no browser [8], e por causa disso ela foi uma das ferramentas escolhidas para ser utilizada neste estudo. Devido ao fato do Cypress suportar somente a linguagem de programação Javascript [8], a utilização desta linguagem se tornou um requisito na escolha da segunda ferramenta utilizada neste projeto.

Para o protocolo DevTools, o Puppeteer e o Playwright são as principais ferramentas [30] e atendem ao requisito de suportar a linguagem Javascript [10]. Como o Playwright é um framework criado recentemente [32], ele foi pouco explorado em estudos (de acordo com as buscas realizadas nesta pesquisa, o Playwright não foi mencionado em nenhum artigo). Além disso, ele foi recomendado para experimentação pelo Tech Radar da Thoughtworks [33] por ser uma ferramenta que suporta os principais motores de busca de navegadores e por ter uma boa estabilidade. Assim, devido a esses fatores o Playwright foi a ferramenta selecionada para ser explorada neste estudo.

A seguir serão apresentados mais detalhes arquiteturais das duas ferramentas que serão utilizadas neste projeto.

1) Cypress

Nesta ferramenta o teste é executado diretamente no navegador, nela o teste e a aplicação compartilham o mesmo processo e domínio, ou seja, o teste pode acessar diretamente as APIs do navegador e o DOM da página. Nesse compartilhamento a aplicação e o teste revezam a execução, com a aplicação pausando enquanto o teste é executado e vice-versa, garantindo que a aplicação não mude enquanto o teste está sendo executado [30]. Isso é possível porque o código do teste e a aplicação são carregados em *iframes* diferentes na página, já que esse elemento HTML permite a inserção de conteúdo dentro dele, como objetos HTML e mídias. Na Figura 5 está ilustrada a arquitetura do Cypress elaborada por Dominic Elm [34].

2) Playwright

No Playwright o teste é executado através do protocolo DevTools, que permite que a ferramenta acesse diretamente a API do navegador, com a comunicação sendo feita por meio de WebSockets. Essa comunicação direta com o navegador é uma vantagem em relação ao WebDriver, que possui um intermediário que faz essa comunicação [30]. Outro ponto importante sobre o Playwright é que ao instanciar um navegador a ferramenta cria contextos, que são ambientes isolados que permitem a execução de testes simultâneos separadamente. Na Figura 4 é possível verificar uma ilustração da arquitetura do Playwright que foi adaptada da arquitetura do Puppeteer [35].

C. Definição da aplicação

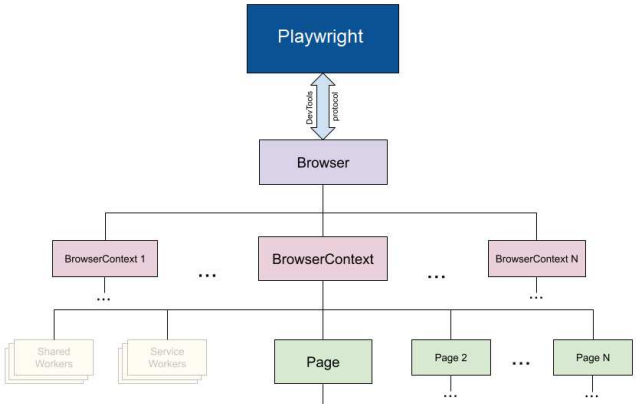
Devido ao escopo do projeto ser direcionado aos testes de interface, nesta fase foram considerados sites voltados para o aprendizado e treinamento de ferramentas de automação, pois segundo Vila et al. [36], esses sites possuem os mais diversos tipos de interações que são possíveis validar utilizando frameworks de automação. Na busca foram encontrados diversos sites como *Test Pages for Automating* [37], *UI Test Automation Playground* [38], *The-Internet* [39] e *The Automation Practice* [40], porém a aplicação selecionada foi a *ToolsQA* [41], ilustrada na Figura 6, que é uma aplicação voltada para o treinamento de certificação da ferramenta

Selenium e que também foi utilizada por Vila et al. [36] e Barros et al. [27] em seus estudos.

D. Definição dos casos de teste

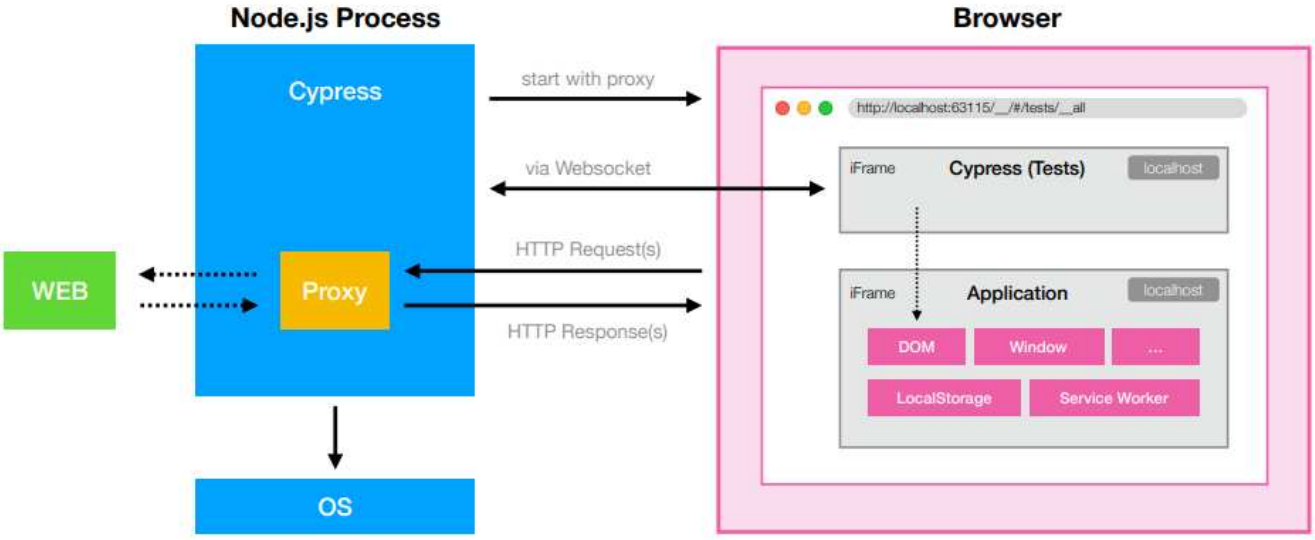
Os casos de teste foram definidos e embasados nos cenários apresentados por Barros et al. [27], que identificaram as interações mais relevantes conforme a sua utilização em aplicações reais. No entanto, este trabalho considerou como relevantes as interações de upload e download após a realização de uma abordagem randômica de exploração na interface da aplicação, sendo definido um caso de teste específico para elas. Ao todo foram definidos 24 casos de teste agrupados de acordo com a divisão de seções do site *ToolsQA*. Os casos de teste e seus respectivos grupos estão listados no Apêndice A.

Figura 4 -Arquitetura do Playwright



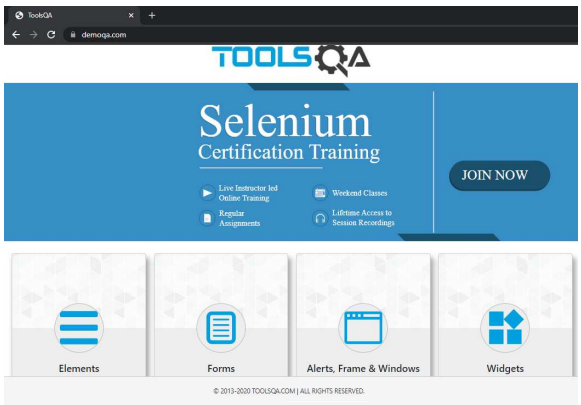
Fonte: Adaptado pelo Autor [35]

Figura 5 - Arquitetura do Cypress



Fonte: [34]

Figura 6 – Aplicação ToolsQA



Fonte: [41]

VI. EXECUÇÃO

A. Desenvolvimento dos scripts

Embasado na divisão de seções do site *ToolsQA*, foi desenvolvido um script para cada grupo de cenários, resultando em 4 arquivos em cada projeto, como mostra a Figura 7. Em cada um destes arquivos foram implementados os testes correspondentes aos cenários de cada grupo. No Apêndice B, é possível identificar a implementação do teste no framework Cypress correspondente ao grupo Forms, que possui somente o cenário de interação com formulários, como mostra o Apêndice A. Por fim, no Apêndice C, é possível verificar a implementação do mesmo teste no framework Playwright.

B. Análise comparativa

Esta etapa se iniciou com a coleta dos dados necessários para realizar a análise comparativa. A seguir serão apresentados os dados coletados.

- Tempo de execução do teste (ms)

Neste trabalho foi considerando o tempo na unidade de milissegundos (ms). Como esse valor pode variar de acordo com o equipamento em que é executado, foi utilizada a mesma máquina para execução dos scripts de teste. As especificações desta máquina estão descritas a seguir:

- **Modelo:** Dell Latitude 3410
- **Processador:** Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz 2.30 GHz
- **Memória:** 16 GB 2667 MHz DDR4
- **Placa de vídeo:** Intel UHD Graphics 620 8198 MB
- **Sistema operacional:** Windows 10 Pro 64-bit
- **Navegador:** Chrome 96

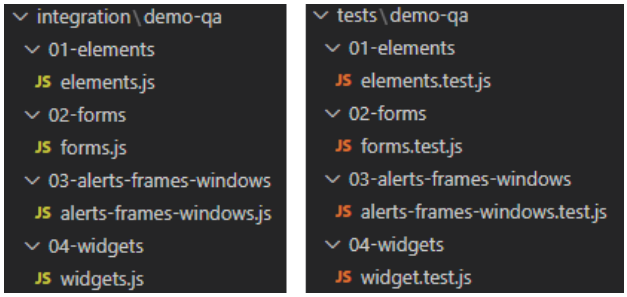
Os tempos correspondentes à execução única de cada caso de teste estão ilustrados no Quadro 2, que também contém a informação de qual framework leva a vantagem obtendo a menor execução.

Quadro 2 - Tempo de execução de cada teste em milissegundos (ms)

Caso de Teste	Playwright (ms)	Cypress (ms)	Vantagem
CT 01	5531	9746	Playwright
CT 02	4225	4896	Playwright
CT 03	4051	6738	Playwright
CT 04	6272	6979	Playwright
CT 05	4144	5184	Playwright
CT 06	10166	17537	Playwright
CT 07	4910	2250	Cypress
CT 08	4858	12272	Playwright
CT 09	4034	5542	Playwright
CT 10	10199	6627	Cypress
CT 11	3457	2695	Cypress
CT 12	3617	2343	Cypress
CT 13	4073	2528	Cypress
CT 14	4667	3737	Cypress
CT 15	5467	N/A	Playwright
CT 16	7403	6378	Cypress
CT 17	3934	5437	Playwright
CT 18	5465	10294	Playwright
CT 19	4004	2222	Cypress
CT 20	14478	14588	Playwright
CT 21	4480	4075	Cypress
CT 22	7441	7931	Playwright
CT 23	4106	2225	Cypress
CT 24	4411	4096	Cypress
Total	**135393**	**146320**	

Fonte: Elaborado pelo Autor (2022)

Figura 7 - Estrutura de arquivos Cypress X Playwright



Fonte: Elaborado pelo Autor (2022)

- Eficiência do teste

No Quadro 3 estão ilustradas as quantidades de linha de código necessárias para executar os testes correspondentes a cada cenário, além de conter a informação de qual framework leva a vantagem sendo mais eficiente.

Quadro 3 - Quantidade de linhas de código em cada teste

Caso de Teste	Playwright	Cypress	Vantagem
CT 01	12	12	Iguais
CT 02	6	6	Iguais
CT 03	5	5	Iguais
CT 04	28	20	Cypress
CT 05	9	9	Iguais
CT 06	76	34	Cypress
CT 07	11	13	Playwright
CT 08	33	39	Playwright
CT 09	8	7	Cypress
CT 10	9	8	Cypress
CT 11	8	8	Iguais
CT 12	8	8	Iguais
CT 13	8	8	Iguais
CT 14	8	9	Playwright
CT 15	16	N/A	Playwright
CT 16	48	21	Cypress
CT 17	16	16	Iguais
CT 18	19	19	Iguais
CT 19	11	9	Cypress
CT 20	18	22	Playwright
CT 21	71	32	Cypress
CT 22	18	35	Playwright
CT 23	15	12	Cypress
CT 24	20	18	Cypress
Total	**481**	**370**	

Fonte: Elaborado pelo Autor (2022)

- Cobertura do teste

O Quadro 4 detalha se os testes de cada ferramenta cobrem os cenários e informa a porcentagem de cobertura dos testes na última linha.

Quadro 4 - Cobertura de testes dos cenários

Caso de Teste	Playwright	Cypress
CT 01	Cobre	Cobre
CT 02	Cobre	Cobre
CT 03	Cobre	Cobre
CT 04	Cobre	Cobre
CT 05	Cobre	Cobre
CT 06	Cobre	Cobre
CT 07	Cobre	Cobre
CT 08	Cobre	Cobre
CT 09	Cobre	Cobre
CT 10	Cobre	Cobre
CT 11	Cobre	Cobre
CT 12	Cobre	Cobre
CT 13	Cobre	Cobre
CT 14	Cobre	Cobre
CT 15	Cobre	Não cobre
CT 16	Cobre	Cobre
CT 17	Cobre	Cobre
CT 18	Cobre	Cobre
CT 19	Cobre	Cobre
CT 20	Cobre	Cobre
CT 21	Cobre	Cobre
CT 22	Cobre	Cobre
CT 23	Cobre	Cobre
CT 24	Cobre	Cobre
Cobertura	**100,00%**	**95,83%**

Fonte: Elaborado pelo Autor (2022)

Diante dos dados coletados, pôde-se observar que em relação ao critério de tempo de execução, a comparação entre os frameworks foi bem equilibrada, com o Playwright obtendo vantagem em 13 dos 24 casos de teste e o Cypress levando vantagem em 11. Vale destacar que no cenário 15, referente a interação com diferentes janelas e abas, não foi possível implementar o script de testes em Cypress por conta de uma limitação da ferramenta [42], e devido a isso, foi considerada a vantagem para o Playwright.

Levando-se em conta o tempo de execução total o Playwright teve a vantagem por cerca de 11 segundos. Um ponto a se destacar é que em apenas 2 casos de teste a diferença de tempo foi maior que 5 segundos (valor definido *ad hoc* pela equipe com base para os testes), evidenciando uma performance equivalente entre as duas ferramentas.

Tendo em vista os grupos, o Playwright teve uma larga vantagem no de *Elements* e o Cypress teve uma ampla vantagem no de *Alerts, Frame and Windows*. No grupo de *Forms*, que contém somente um cenário o Playwright levou vantagem e no grupo de *Widgets* ocorreu um equilíbrio, com o Cypress obtendo vantagem em 5 dos 9 casos.

Referente ao critério de eficiência de teste, a comparação entre os frameworks também foi bem equilibrada com o Cypress obtendo vantagem em 9 dos 24 casos de teste, o Playwright levando vantagem em 6, e em 9 casos eles ficaram empatados. Nesse critério, também foi dada a vantagem para o Playwright no caso 15 em que não foi possível implementar o script no Cypress.

Levando-se em conta a eficiência total, o Cypress obteve uma vantagem considerável por cerca de 110 linhas de código. É importante destacar que em apenas 4 cenários a diferença de linhas foi maior do que 10 (valor base de referência, definido *ad hoc*), demonstrando que esses cenários específicos como a interação com endpoints, no qual o Playwright obteve uma grande desvantagem, influenciaram no resultado total. Entretanto, se for analisar os cenários individualmente é possível notar um equilíbrio.

Tendo em vista os grupos, o Cypress obteve uma vantagem considerável no de *Widgets*, onde foi melhor em 5 dos 9 cenários, foi pior em 2 e teve a mesma eficiência nos 2 restantes. Este apliativo também levou uma pequena vantagem no grupo de *Elements* em que se sobressaiu em 2 e ficou empatado em 4, sendo pior em apenas 1 caso. No grupo de *Forms*, o Playwright levou vantagem no único cenário e no grupo de *Alerts, Frame and Windows* ocorreu um equilíbrio, onde cada ferramenta levou vantagem em 2 cenários e ficaram empatadas nos 3 restantes.

Considerando a cobertura dos testes, o Playwright levou vantagem ao atingir 100% de cobertura. O Cypress chegou a 95,83%, e isso ocorreu por conta da limitação da ferramenta na interação com múltiplas janelas do navegador [42], fazendo com que não fosse possível implementar o script correspondente ao cenário 15. É importante destacar que essa limitação é uma decisão de projeto dos criadores do framework [42] que sugerem que nesses casos o script seja desenvolvido de modo a abrir a URL de destino na mesma aba, porém essa alternativa não cobre o caso de teste 15 já que o intuito dele é validar a abertura de uma nova aba ou janela.

VII. CONCLUSÕES

Este trabalho teve como objetivo realizar uma análise comparativa entre Cypress e Playwright, com o intuito de auxiliar os profissionais na tomada de decisão de qual framework utilizar no contexto de interface. Foram realizadas avaliações do tempo de execução dos frameworks em cenários específicos, como também da quantidade de linhas de código - medindo as eficiências dos testes. Adicionalmente, foi feita a verificação da cobertura dessas ferramentas nas principais interações utilizadas nas aplicações web modernas.

Foi constatado que entre as ferramentas analisadas existe um certo equilíbrio, com o Playwright levando a vantagem no tempo de execução, e o Cypress levando vantagem na

eficiência dos scripts. Foi possível verificar também que as ferramentas abrangem as principais interações utilizadas nas aplicações web modernas, com o Cypress não cobrindo apenas a interação de abertura de novas abas e janelas, por opção dos seus criadores. Com isto, pode-se observar que tanto o Cypress quanto o Playwright se apresentam como ferramentas sólidas e ótimas alternativas ao Selenium, cabendo ao profissional escolher qual delas é mais adequada à sua necessidade.

Entretanto, vale ressaltar que como ameaça à validade, o projeto teve sua aplicação/site voltada para o aprendizado e treinamento de ferramentas de automação, é possível que em um contexto mais real o resultado obtido seja diferente. Além disso, apesar de conter uma boa documentação [43] e um fórum oficial para resolução de dúvidas, que foi utilizado durante este trabalho [44], o Playwright é uma ferramenta nova e possui pouco suporte nos sites de dúvidas mais populares - como o Stackoverflow [45] - e poucos cursos de capacitação para o framework. Devido a isso, a implementação pode não ter sido feita da melhor maneira, influenciando diretamente nos resultados obtidos nos critérios de tempo de execução e eficiência dos testes.

A relevância deste trabalho se justifica pela apresentação de uma análise comparativa entre ferramentas de automação de teste web modernas utilizando a técnica de programação de scripts com a mesma linguagem de programação. Além de abordar o Playwright que por ser uma ferramenta criada recentemente ainda não possui trabalhos relevantes sobre ela.

Para trabalhos futuros pode-se destacar a realização da análise dessas ferramentas em uma aplicação web moderna de maior escala com utilização real, como sites de e-commerce populares; a adição de um framework que utilize a técnica de programação de scripts, que tenha o WebDriver como sua arquitetura base e que suporte à linguagem de programação Javascript, como o NightwatchJS [46]; e, por último, a adição de critérios de avaliação mais subjetivos, como facilidade de instalação e configuração, e facilidade de aprendizado.

REFERÊNCIAS

[1] International Software Testing Qualifications Board (ISTQB). Certified Tester Foundation Level Syllabus. endereço: https://bstqb.org.br/b9/doc/syllabus_ctfl_3.1br.pdf. (acesso em 29/05/2022).

[2] Delamaro, E.; Maldonado, J. C.; Jino, M. "*Introdução ao teste de software*." 1ª Edição, Elsevier, Rio de Janeiro, 2007.

[3] R. A. P. Oliveira, E. Alégroth, Z. Gao and A. Memon, "Definition and evaluation of mutation operators for GUI-level mutation analysis," em *2015 Eighth International Conference on Software Testing, Verification and Validation Workshops* (ICSTW), IEEE, 2015, pp. 1-10.

[4] Sommerville, I. "*Engenharia de software*." 9ª edição, Pearson Universidades, São Paulo, 2011.

[5] Molinari, L. "*Inovação e Automação de Testes de Software*." 1ª Edição, Érica Ltda, São Paulo, 2010.

[6] E. Pelivani e B. Cico, "A comparative study of automation testing tools for web applications," em *2021 10th Mediterranean Conference on Embedded Computing* (MECO), 2021, pp. 1-6.

[7] WebDriver. (2022). Endereço: https://w3c.github.io/webdriver/#compatibility. (acesso em 29/05/2022).

[8] F. Mobaraya e S. Ali, "Technical Analysis of Selenium and Cypress as Functional Automation Framework for Modern Web Application Testing", em *2019 9th International Conference on Computer Science, Engineering and Applications* (ICCSEA), 2019, pp. 27-46.

[9] Cypress. (2022). "Why Cypress? Cypress Documentation", Endereço: https://docs.cypress.io/guides/overview/why-cypress. (acesso em 29/05/2022).

[10] "Playwright vs. Puppeteer: Which should you choose? - LogRocket Blog" endereço: https://blog.logrocket.com/playwright-vs-puppeteer/ (acesso em 29/05/2022).

[11] Gil, A. C. "*Como elaborar projetos de pesquisa*". 6ª Edição, Atlas, São Paulo, 2017.

[12] Pressman, R. S. "*Software Engineering: A Practitioner's Approach*". 6ª Edição, McGraw-Hill, Nova York, 2005.

[13] Rios, E.; Moreira, T. "*Teste de Software*". 3ª Edição, Alta Books, Rio de Janeiro, 2013.

[14] I. Banerjee, B. Nguyen, V. Garousi e A. Memon, "Graphical user interface (GUI) testing: Systematic mapping and repository", *Information and Software Technology*, v. 55, n. 10, pp. 1679-1694, 2013.

[15] S. R. Choudhary, A. Gorla e A. Orso, "Automated test input generation for android: Are we there yet? (e)", em *2015 30th IEEE/ACM International Conference on Automated Software Engineering* (ASE), IEEE, 2015, pp. 429-440.

[16] Wanderley, Arthur. *"Análise comparativa das abordagens para teste de interface gráfica em dispositivos móveis"*. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal de Pernambuco, Recife, 2019.

[17] Tian, J. "*Software Quality Engineering*". 1. Edição. [S.l.]: Wiley-IEEE Press, 2005.

[18] Fewster, M. e Graham, D. "*Software Test Automation: Effective Use of Test Execution Tools*". Addison-Wesley, New York, 1999.

[19] M. Hanna et al. "A Review of Scripting Techniques Used in Automated Software Testing", *International Journal of Advanced Computer Science and Applications*, vol. 5, n. 1, pp. 194-202, 2014.

[20] E. Collins e L. Lobão. "Experiência em Automação do Processo de Testes em Ambiente Ágil com SCRUM e ferramentas OpenSource ", em *2010 Anais do IX Simpósio Brasileiro de Qualidade de Software, Belém*, 2010, pp. 303-310.

[21] T. J. Naidu et al., "SAHI vs. Selenium: A comparative analysis," em *2014 International Conference on Contemporary Computing and Informatics* (IC3I), 2014, pp. 967-970.

[22] Selenium WebDriver. (2022) "Getting started | Selenium WebDriver" endereço: https://www.selenium.dev/documentation/webdriver/getting_started/ (acesso em 29/05/2022).

[23] Selenium IDE. (2022) "Open source record and playback test automation for the web" endereço: https://www.selenium.dev/selenium-ide/ (acesso em 29/05/2022).

[24] Selenium Grid. (2022) "Selenium Grid 4 | Selenium" endereço: https://www.selenium.dev/documentation/grid/ (acesso em 29/05/2022).

[25] Sualim, S. A. et al. "Comparative Evaluation of Automated User Acceptance Testing Tool for Web Based Application." *Software Engineering and Technology*, v. 2, n. 2, 2017.

[26] A. Sivaji et al., "Software Testing Automation: A Comparative Study on Productivity Rate of Open Source Automated Software Testing Tools For Smart Manufacturing," em *2020 IEEE Conference on Open Systems* (ICOS), 2020, pp. 7-12.

[27] Barros, D. de S. et al., "Automação de Testes Funcionais: Uma Análise Técnica dos Frameworks Cypress e Selenium". *Even3 Publicações*, 2021.

[28] Damm, Lars-Ola."Evaluating and Improving Test Efficiency". 2002.

[29] S. Muhammad e I. Suhaimi. "An Evaluation of Test Coverage Tools in Software Testing" em *2011 International Conference on Telecommunication Technology and Applications* (IACSIT), 2011, pp. 216-222.

[30] "Cypress vs Other Test Runners | Better world by better software" endereço:https://glebbahmutov.com/blog/cypress-vs-other-test-runners/ (acesso em 29/05/2022).

[31] Chrome DevTools Protocol. (2022). endereço: https://chromedevtools.github.io/devtools-protocol/ (acesso em 29/05/2022).

[32] Release v0.10.0 · microsoft/playwright (2022). Endereço: https://github.com/microsoft/playwright/releases/tag/v0.10.0 (acesso em 29/05/2022).

[33] "Playwright | Technology Radar | Thoughtworks" endereço: https://www.thoughtworks.com/pt-br/radar/tools/playwright (acesso em 29/05/2022).
[34] "Cypress: The future of E2E testing - Dominic Elm - Speaker Deck" endereço:https://speakerdeck.com/d3lm/cypress-the-future-of-e2e-testing?slide=27 (acesso em 29/05/2022).
[35] Puppeteer – Overview. (2022). Endereço: https://github.com/puppeteer/puppeteer/blob/main/docs/api.md#overview (acesso em 29/05/2022).
[36] Vila, E. et al. "Automation Testing Framework for Web Applications with Selenium WebDriver: Opportunities and Threats." *Em 2017 Proceedings of the International Conference on Advances in Image Processing*, ACM, 2017, pp. 144-150.
[37] Selenium Test Pages. (2022). Endereço: https://testpages.herokuapp.com/styled/index.html (acesso em 29/05/2022).
[38] UI Test Automation Playground. (2022). Endereço: http://uitestingplayground.com/ (acesso em 29/05/2022).
[39] The Internet. (2022). Endereço: https://the-internet.herokuapp.com/ (acesso em 29/05/2022).
[40] My Store. (2022). http://automationpractice.com/ (acesso em 29/05/2022).
[41] ToolsQA. (2022). Endereço: https://demoqa.com/ (acesso em 29/05/2022).
[42] "Trade-offs | Cypress Documentation" endereço: https://docs.cypress.io/guides/references/trade-offs#Multiple-browsers-open-at-the-same-time (acesso em 29/05/2022).
[43] Playwright (2022). "Getting started | Playwright.v.", Endereço: https://playwright.dev/docs/intro. (acesso em 29/05/2022).
[44] "[Question] How to intercept a specific response? Issue #10045 · microsoft/playwright" endereço:https://github.com/microsoft/playwright/issues/10045 (acesso em 29/05/2022).
[45] "Stack Overflow - Where Developers Learn, Share, & Build Careers" endereço: https://stackoverflow.com/ (acesso em 29/05/2022).
[46] "Nightwatch.js | Node.js powered End-to-End testing framework" endereço: https://nightwatchjs.org/ (acesso em 29/05/2022).

APÊNDICE A – CASOS DE TESTE DA APLICAÇÃO TOOLSQA

Caso de teste	Grupo	Ação	Verificação
01 – Interação com caixas de texto	Elements	1 - Preencher as caixas de texto	1 - Se o texto corresponde ao digitado
02 – Interação com elemento checkbox		1 - Marcar os checkboxes	1 - Se o texto de confirmação da ação é exibido
03 – Interação com elemento radio button		1 - Selecionar o radio button	1 - Se o texto de confirmação da ação é exibido
04 – Interação com elementos em tabelas		1 - Filtrar a tabela por um registro; 2 - Adicionar um novo registro na tabela; 3 - Filtrar por um registro diferente da ação 1.	1 - Se o registro desejado é exibido; 2 - Se o novo registro é exibido; 3 - Se o registro da verificação 1 não é exibido.
05 – Interação com botões - botão direito e clique duplo		1 - Clicar no elemento com o botão direito do mouse; 2 - Efetuar um clique duplo no elemento; 3 - Clicar no elemento com identificador dinâmico.	1 - Se o texto de confirmação da ação é exibido; 2 - Se o texto de confirmação da ação é exibido; 3 - Se o texto de confirmação da ação é exibido.
06 – Interação com links e endpoints		1 - Clicar no link simples de redirecionamento da página; 2 - Clicar no link dinâmico de redirecionamento da página; 3 - Clicar no link que chama a API e recebe a resposta HTTP com status 201; 4 - Clicar no link que chama a API e recebe a resposta HTTP com status 204; 5 - Clicar no link que chama a API e recebe a resposta HTTP com status 301; 6 - Clicar no link que chama a API e recebe a resposta HTTP com status 400; 7 - Clicar no link que chama a API e recebe a resposta HTTP com status 401; 8 - Clicar no link que chama a API e recebe a resposta HTTP com status 403; 9 - Clicar no link que chama a API e recebe a resposta HTTP com status 404.	1 - Se a página é redirecionada corretamente; 2 - Se a página é redirecionada corretamente; 3 - Se a resposta da chamada foi o status 201; 4 - Se a resposta da chamada foi o status 204; 5 - Se a resposta da chamada foi o status 301; 6 - Se a resposta da chamada foi o status 400; 7 - Se a resposta da chamada foi o status 401; 8 - Se a resposta da chamada foi o status 403; 9 - Se a resposta da chamada foi o status 404.
07 – Interação com upload e download de arquivos		1 - Realizar o download de um arquivo; 2 - Realizar o upload de um arquivo.	1 - Se o download foi realizado corretamente; 2 - Se o arquivo foi carregado corretamente.
08 – Interação com formulários (Submissão)	Forms	1 - Prencher os campos do formulário e realizar a submissão	1 - Se os dados do formulário foram submetidos corretamente
09 – Interação com alertas - clicar no botão para exibir um alerta	Alerts, Frame & Windows	1 - Clicar no botão para exibir um alerta	1 - Se o alerta foi exibido corretamente

10 – Interação com alertas - alerta é exibido após 5 segundos		1 - Clicar no botão que exibe um alerta após 5 segundos	1 - Se o alerta foi exibido corretamente após 5 segundos
11 – Interação com alertas de confirmação - botão Cancelar		1 - Clicar no botão para exibir um alerta de confirmação; 2 - Clicar no botão Cancelar do alerta	1 - Se o alerta foi exibido corretamente; 2 - Se o texto de confirmação da ação é exibido
12 – Interação com alertas de confirmação - botão OK		1 - Clicar no botão para exibir um alerta de confirmação; 2 - Clicar no botão OK do alerta	1 - Se o alerta foi exibido corretamente; 2 - Se o texto de confirmação da ação é exibido
13 – Interação com alertas - preencher Prompt		1 - Clicar no botão para exibir um alerta de preenchimento de texto; 2 - Preencher o campo do alerta com um texto e clicar no botão OK.	1 - Se o alerta foi exibido corretamente; 2 - Se o texto digitado é exibido, confirmando a ação
14 – Interação com modais		1 - Clicar no botão que exibe um modal pequeno; 2 - Clicar no botão que exibe um modal grande.	1 - Se o modal pequeno é exibido; 2 - Se o modal grande é exibido.
15 – Interação com diferentes janelas e abas		1 - Clicar no botão que abre uma nova aba; 2 - Clicar no botão que abre uma nova janela; 3 - Clicar no botão que abre uma janela de mensagem.	1 - Se uma nova aba é aberta; 2 - Se uma nova janela é criada; 3 - Se uma nova janela de mensagem é exibida.
16 – Interação com widgets accordian	Widgets	1 - Clicar na segunda guia do elemento accordian; 2 - Clicar na primeira guia do elemento accordian; 3 - Clicar na terceira guia do elemento accordian; 4 - Clicar novamente na terceira guia do elemento accordian;	1 - Se o texto da segunda guia é expandido e os outros são fechados; 2 - Se o texto da primeira guia é expandido e os outros são fechados; 3 - Se o texto da terceira guia é expandido e os outros são fechados; 4 - Se os textos das três guias são fechados.
17 – Interação com widgets de auto-complete		1 - Digite uma letra no elemento que permite múltipla seleção; 2 - Selecione um registro; 3 - Digite a mesma letra da ação 1 no elemento que permite múltipla seleção; 4 - Selecione outro registro; 5 - Digite uma letra no elemento que só permite uma seleção; 6 - Selecione um registro; 7 - Digite a mesma letra da ação 4 no elemento que só permite uma seleção; 8 - Selecione outro registro;	1 - Se uma lista de opções é exibida; 2 - Se o registro selecionado é exibido no elemento; 3 - Se uma lista de opções sem o registro selecionado anteriormente é exibida; 4 - Se os registros selecionados nas ações 2 e 4 são exibidos no elemento; 5 - Se uma lista de opções é exibida; 6 - Se o registro selecionado é exibido no elemento; 7 - Se uma lista de opções é exibida; 8 - Se o registro selecionado é exibido no elemento e o anterior foi descartado.
18 – Interação com seletores de data		1 - Clicar no campo de data; 2 - Selecionar um ano diferente do atual; 3 - Selecionar um mês diferente do atual; 4 - Selecionar um dia do mês; 5 - Clicar no campo de data e hora; 6 - Selecionar um ano diferente do atual; 7 - Selecionar um mês diferente do atual; 8 - Selecionar um dia do mês; 9 - Selecione um horário.	1 - Se o calendário do elemento é exibido; 2 - Se o ano foi modificado no elemento; 3 - Se o mês foi modificado no elemento; 4 - Se a data é exibida corretamente no campo de data; 5 - Se o calendário e as opções de hora do elemento são exibidos; 6 - Se o ano foi modificado no elemento; 7 - Se o mês foi modificado no elemento; 8 - Se o dia foi modificado no elemento; 9 - Se a data e hora são exibidas corretamente no campo de data e hora.
19 – Interação com elementos de slider		1 - Clique em uma posição do slider diferente da atual	1 - Se a posição do slider foi modificada

20 – Interação com barra de progresso		1 - Clique no botão para iniciar a ação; 2 - Clique no botão para pausar a ação quando chegar em 50%; 3 - Clique no botão para retomar a ação e espere até finalizar; 4 - Clique no botão para resetar o progresso;	1 - Se o elemento está exibindo a evolução do progresso; 2 - Se o elemento está exibindo que 50% da ação foi realizada; 3 - Se o elemento está exibindo que a ação foi finalizada com 100%; 4 - Se o elemento voltou ao seu estado inicial.
21 – Interação com elementos de abas		1 - Clicar na segunda aba do elemento; 2 - Clicar na primeira aba do elemento; 3 - Clicar na terceira aba do elemento; 4 - Clicar na quarta aba do elemento que está desabilitada.	1 - Se somente o texto referente a segunda aba é exibido; 2 - Se somente o texto referente a primeira aba é exibido; 3 - Se somente o texto referente a terceira aba é exibido; 4 - Se nenhuma açao é executada.
22 – Interação com elementos tooltip		1 - Colocar o cursor do mouse sobre o botão que possui tooltip; 2 - Colocar o cursor do mouse sobre o campo que possui tooltip; 3 - Colocar o cursor do mouse sobre o primeiro link que possui tooltip; 4 - Colocar o cursor do mouse sobre o segundo link que possui tooltip;	1 - Se o tooltip referente ao elemento é exibido corretamente; 2 - Se o tooltip referente ao elemento é exibido corretamente; 3 - Se o tooltip referente ao elemento é exibido corretamente; 4 - Se o tooltip referente ao elemento é exibido corretamente;
23 – Interação com elementos de menu		1 - Colocar o cursor do mouse sobre o segundo elemento do menu horizontal; 2 - Colocar o cursor do mouse sobre o terceiro elemento do sub-menu vertical.	1 - Se um sub-menu vertical com três elementos é exibido; 2 - Se um sub-menu vertical com dois elementos é exibido.
24 – Interação com elementos select		1 - Selecione uma opção no elemento de select com agrupamento; 2 - Selecione uma opção no elemento de select; 3 - Seleciona uma opção no elemento de select (padrão HTML); 4 - Selecionar mais de uma opção no elemento de multiselect; 5 - Seleciona uma opção no elemento de multiselect (padrão HTML);	1 - A opção selecionada é exibida no elemento; 2 - A opção selecionada é exibida no elemento; 3 - A opção selecionada é exibida no elemento; 4 - A opções selecionadas são exibidas no elemento; 5 - A opções selecionadas são exibidas no elemento;

Fonte: Elaborado pelo Autor (2022)

APÊNDICE B – SCRIPT DE AUTOMAÇÃO REFERENTE AO GRUPO "FORMS" NO FRAMEWORK CYPRESS

Script Cypress

```
/// <reference types="cypress" />
describe("Test Forms on Demo QA site", () => {
   it("Should be able to fill and submit the form", () => {
      cy.visit("https://demoqa.com/automation-practice-form");
      cy.xpath("//input[@id='firstName']").type("Lainey");
      cy.xpath("//input[@id='lastName']").type("Ross");
      cy.xpath("//input[@id='userEmail']").type("nullchar@demoqa.com");
      cy.xpath("//input[@type='radio']").check('Male', {force: true});
      cy.xpath("//input[@id='userNumber']").type("4865596142");
      cy.xpath("//input[@id='dateOfBirthInput']").click();
      cy.xpath("//select[@class='react-datepicker__month-select']").select('July');
      cy.xpath("//select[@class='react-datepicker__year-select']").select('2020');
      cy.xpath("//div[@class='react-datepicker__week']//div[contains(@aria-label,'July 7th')]").click();
      cy.xpath("//input[@id='subjectsInput']").type("Maths");
      cy.xpath("//div[contains(@class,'subjects-auto-complete__option') and text()='Maths']").click();
      cy.xpath("//label[contains(text(),'Sports')]").prev().check({force: true});
      cy.xpath("//label[contains(text(),'Music')]").prev().check({force: true});
      cy.fixture('testPicture.png').then(fileContent => {
         cy.get('input[type="file"]').attachFile({
            fileContent: fileContent.toString(),
            fileName: 'testPicture.png',
```

```
            mimeType: 'image/png'
          });
      });
      cy.xpath("//textarea[@id='currentAddress']").type("768 Gainsway Street Lawrenceville, GA 30043");
      cy.xpath("//div[@id='state']").click();
      cy.xpath("//*[contains(@tabindex,'-1')]").contains('NCR').click();
      cy.xpath("//div[@id='city']").click();
      cy.xpath("//*[contains(@tabindex,'-1')]").contains('Noida').click();
      cy.xpath("//button[@id='submit']").click();
      cy.get('table').contains('td', 'Student Name').next().contains('td', 'Lainey Ross').should('be.visible');
      cy.get('table').contains('td', 'Student Email').next().contains('td', 'nullchar@demoqa.com').should('be.visible');
      cy.get('table').contains('td', 'Gender').next().contains('td', 'Male').should('be.visible');
      cy.get('table').contains('td', 'Mobile').next().contains('td', '4865596142').should('be.visible');
      cy.get('table').contains('td', 'Date of Birth').next().contains('td', '7 July,2020').should('be.visible');
      cy.get('table').contains('td', 'Subjects').next().contains('td', 'Maths').should('be.visible');
      cy.get('table').contains('td', 'Hobbies').next().contains('td', 'Sports, Music').should('be.visible');
      cy.get('table').contains('td', 'Picture').next().contains('td', 'testPicture.png').should('be.visible');
      cy.get('table').contains('td', 'Address').next().contains('td', '768 Gainsway Street Lawrenceville, GA
30043').scrollIntoView().should('be.visible');
      cy.get('table').contains('td', 'State and City').next().contains('td', 'NCR Noida').should('be.visible');
   });
});
```

Fonte: Elaborado pelo Autor (2022)

APÊNDICE C - SCRIPT DE AUTOMAÇÃO REFERENTE AO GRUPO "FORMS" NO FRAMEWORK PLAYWRIGHT

Script Playwright

```
const { test, expect } = require("@playwright/test");
test.describe("Test Forms on Demo QA site", () => {
   test('Should be able to fill and submit the form', async ({ page }) => {
      await page.goto("https://demoqa.com/automation-practice-form");
      await page.fill("//input[@id='firstName']", 'Lainey');
      await page.fill("//input[@id='lastName']", 'Ross');
      await page.fill("//input[@id='userEmail']", 'nullchar@demoqa.com');
      await page.click("//label[contains(text(),'Male')]");
      await page.fill("//input[@id='userNumber']", '4865596142');
      await page.click("//input[@id='dateOfBirthInput']");
      await page.selectOption("//select[@class='react-datepicker__month-select']", { label: 'July' });
      await page.selectOption("//select[@class='react-datepicker__year-select']", { label: '2020' });
      await page.click("//div[@class='react-datepicker__week']//div[contains(@aria-label,'July 7th')]")
      await page.fill("//input[@id='subjectsInput']", 'Maths');
      await page.click("//div[contains(@class,'subjects-auto-complete__option') and text()='Maths']");
      await page.check("//label[contains(text(),'Sports')]", {force:true});
      await page.check("//label[contains(text(),'Music')]", {force:true});
      await page.setInputFiles('input[type="file"]', './fixtures/testPicture.png');
      await page.fill("//textarea[@id='currentAddress']", '768 Gainsway Street Lawrenceville, GA 30043');
      await page.click("//div[@id='state']");
      await page.click("//*[contains(@tabindex,'-1')]");
      await page.click("//div[@id='city']");
      await page.click("//*[contains(@tabindex,'-1')]");
      await page.click("//button[@id='submit']");
      await expect(page.locator("//td[contains(text(),'Student Name')]/following-sibling::td")).toHaveText(/Lainey Ross/);
      await expect(page.locator("//td[contains(text(),'Student Email')]/following-sibling::td")).toHaveText(/nullchar@demoqa.com/);
      await expect(page.locator("//td[contains(text(),'Gender')]/following-sibling::td")).toHaveText(/Male/);
      await expect(page.locator("//td[contains(text(),'Mobile')]/following-sibling::td")).toHaveText(/4865596142/);
      await expect(page.locator("//td[contains(text(),'Date of Birth')]/following-sibling::td")).toHaveText(/7 July,2020/);
      await expect(page.locator("//td[contains(text(),'Subjects')]/following-sibling::td")).toHaveText(/Maths/);
      await expect(page.locator("//td[contains(text(),'Hobbies')]/following-sibling::td")).toHaveText(/Sports, Music/);
      await expect(page.locator("//td[contains(text(),'Picture')]/following-sibling::td")).toHaveText(/testPicture.png/);
      await expect(page.locator("//td[contains(text(),'Address')]/following-sibling::td")).toHaveText(/768 Gainsway Street Lawrenceville,
GA 30043/);
      await expect(page.locator("//td[contains(text(),'State and City')]/following-sibling::td")).toHaveText(/NCR Delhi/);
   });
});
```

Fonte: Elaborado pelo Autor (2022)