PROJETO DE TESTE AUTOMATIZADO PARA E-COMMERCE

Guilherme Sousa Leão
guilhermesousaleao@outlook.com
Stefano Schwenk Borges Vale Vita
stefano.vita@uniube.br

RESUMO

O presente trabalho possui como tema o estudo da importância da qualidade aplicada no desenvolvimento de softwares, bem como o desenvolvimento e aplicabilidade da automação de testes. A automação será desenvolvida utilizando a linguagem de programação Java, e usando o framework de automação front-end Selenium, bem como a ferramenta Cucumber. É necessário que haja o esforço e o compromisso para garantir a qualidade desses softwares, visto que o seu mau funcionamento pode acarretar diversos impactos na vida das pessoas. Existem diversas formas e métodos para garantir a qualidade do que está sendo desenvolvido e entregue, tais como fundamentos de teste de software, quando os testes devem ser feitos, e técnicas de teste que cobrem o ciclo de desenvolvimento e após a entrega do produto. Sendo assim, a automação de testes é uma importante ferramenta aliada a necessidade de garantir a qualidade, visto que consegue cobrir cenários repetitivos com rapidez e consistência.

Palavras-chave: Automação. Desenvolvimento. Software. Teste.

ABSTRACT

Software use is very present in people’s lives, and sometimes it can go unnotice. Effort and compromisse is needed to assure de quality of those softwares, because the malfunctioning of it can trigger many problems in people’s lives. There are a number of methods to ensure the quality in the development stages, like software test basic principles, when and how to test, and test, and methods to cover the software after the ending of the development. Test automation is very useful tool to assure the software’s quality, since it can cover repetitive case tests with ease.

Key-words: Software. Test. Automation. Development.

1. INTRODUÇÃO

Com o desenvolvimento tecnológico, o homem criou diversos métodos a fim de sanar suas necessidades e resolver problemas do cotidiano. Posteriormente, com a criação do computador, esse se expandiu e passou a fazer parte de todos os espaços da sociedade moderna, desde empresas até residências. A partir da criação do computador, o mundo vivenciou uma transformação digital que impulsionou o surgimento de *softwares*, *smartphones*, aplicativos para assistência a esses e, em última instância, a inteligência artificial passou a ocupar um lugar central nesse processo de desenvolvimento.

No que tange ao âmbito empresarial, a partir dessa revolução tecnológica, as empresas precisam modificar os métodos de vendas, bem como o atendimento a seus clientes e usuários. De acordo com Vasconcelos, Santos e Baldochi (2016), uma característica de negócios de sucesso é oferecer o que o cliente necessita no momento certo, sendo o atendimento fora do ambiente físico da organização uma das necessidades primordiais.

Nesse sentido, as plataformas web se tornaram essenciais, mormente enquanto modelos de negócios de muitos empreendimentos. Assim, garantir que todos os programas e sistemas funcionem corretamente é essencial, embora seja uma tarefa árdua, sobretudo quando o tamanho e a complexidade do projeto a ser desenvolvido são muito grandes (LUFT, 2012).

Plataformas web que apresentam defeitos e baixa qualidade provocam a insatisfação dos clientes, diversos gastos com manutenção e, ainda, comprometem a imagem da empresa. Visando evitar o cometimento de tais desvios, surge a ferramenta do teste de *software*, compreendida enquanto uma atividade essencial que visa assegurar a qualidade no funcionamento do sistema no desenvolvimento de um sistema. Por meio da realização do teste, ou seja, do processo de executar o sistema e observar seu comportamento em relação aos requisitos acordados em contrato, torna-se possível identificar as falhas nos sistemas e evitar erros futuros.

De acordo com Braga (2018), em alguns casos, quando os testes são repetitivos, pode ser apropriado aplicar automação de testes. Bernardo e Kon (2008) afirmam que a automação de testes é amplamente utilizada pelo mercado, principalmente em

metodologias ágeis, com o foco de agilizar o processo de testes, garantindo que tarefas repetitivas e que levariam uma estimativa maior de tempo sejam realizadas em um curto período, permitindo ao *tester* focar em outras tarefas que não podem ser automatizadas.

2. QUALIDADE DE SOFTWARE E AUTOMAÇÃO DE TESTES

A cada dia que passa, o mundo moderno está mais imerso no universo computacional. Durante o seu cotidiano, o ser humano se depara constantemente com softwares em quase todos os lugares. Assim, é importante constatar que cada aparelho ou dispositivo portador de um sistema de software teve, durante o seu processo de desenvolvimento, centenas ou até milhares de defeitos encontrados e corrigidos por seus desenvolvedores e testadores.

Em que pese a realização das referidas correções, é comum que, durante a sua utilização, um usuário encontre possíveis erros e defeitos nesses sistemas. Ante tal cenário, percebeu-se que, com o aumento dos sistemas, a depuração de todos os possíveis erros a cada modificação feita no sistema torna-se uma tarefa de esforço muito elevado. Assim, surgiu o conceito da automatização de testes.

2.1 TESTES DE *SOFTWARE*

Os testes de *software* consistem em um processo usado na avaliação de um sistema ou um componente de um sistema por meio de atividades manuais ou automáticas a fim de verificar se ele atende aos critérios especificados ou ainda identificar possíveis falhas e incoerências entre os resultados esperados e obtidos (LIMA, 2014).

Conforme leciona Pressman (2011), os testes de *software* são atividades construídas e aplicadas de modo metódico ainda na fase de desenvolvimento do sistema, ou seja, antes desse ser concluído, visando identificar se o *software* realiza as atividades requisitadas pelo cliente, desde a interface e *layout* até o código fonte.

Quanto aos tipos de testes de *software*, Braga (2018, p. 26) apresenta:

> Teste de componente/unitário: Se concentra em testar componentes e trechos de código de maneira separada, independente do restante do sistema.
> Teste funcional: Este teste é caracterizado por possuir o objetivo de validar se as funções disponíveis pelo software estão de acordo com o especificado pelo cliente na documentação do produto, como por exemplo requisitos de negócios e requisitos técnicos.
> Teste de carga: É o teste que tem como objetivo simular a carga em condições de normais de uso.
> Teste de instalação: Tipo de teste que verifica se o software é instalado corretamente em diferentes hardwares sob condições adversas como pouca memória, internet ruim e interrupções durante a instalação.
> Teste de segurança: São testes focados em verificar se o sistema funciona de maneira segura e se os dados estão protegidos.
> Teste de performance: É o teste que visa validar a forma que o software performa durante a realização da tarefa a que se propõe
> Teste de carga: Visa verificar a forma que o sistema se comporta quando é submetido a níveis diferentes de cargas, seja de usuários ou transações.
> Teste de stress: Teste que tem como objetivo verificar o comportamento do software quando se atinge o máximo (ou além disso) do tráfico de dados suportados.
> Teste de regressão: Um dos testes considerados mais importantes, é o reteste de funcionalidades anteriormente desenvolvidas para verificar se alguma modificação ou nova funcionalidade impactou nas mesmas.
> Teste de usabilidade: É o teste que foca na utilização da aplicação pelo usuário, se o layout está correto, se a interface está de acordo com as necessidades do cliente, entre outros.
> Smoke test: É o teste que verifica os pontos mais críticos do software, é executado após a aplicação estar pronta no ciclo de desenvolvimento, para ser encaminhada para o ciclo de testes.

2.2 LINGUAGEM DE PROGRAMAÇÃO

A linguagem de programação consiste em uma linguagem forma que, por meio de diversas instruções, permite que um indivíduo programador descreva um conjunto de ações consecutivas, algoritmos e dados a fim de criar programas que controlam o comportamento lógico e físico de um determinado dispositivo (MAGALHÃES, 2016).

Fuhuyoshi (2012) explica que a linguem de programação consiste em um sistema de comunicação estruturado, formado por palavras-chave, símbolos e regras que promovem a relação de entendimento entre o programador e máquina.

Assim, todo *software* é desenvolvido por meio de uma linguagem de programação, a fim de rodar em um dispositivo móvel, computador ou qualquer equipamento que permita sua execução. Insta salientar que existem diversas linguagens. De acordo com

Zapalowski (2011), são mais de 700 linguagens de programação diferentes, de modo que cada uma serve para diversos propósitos.

2.3 LINGUAGEM DE PROGRAMAÇÃO JAVA

O Java consiste em uma linguagem de programação orientada a objetos e multiplataforma, com funcionamento em qualquer sistema operacional, desde que o interpretador esteja instalado. O interpretador, chamado de máquina virtual Java (JVM), trata-se de um programa capaz de converter o código Java em comandos que o sistema operacional seja capaz de executar. O Java faz uso de um conceito diferenciado, em vez que gerar um código binário único para cada plataforma, gera-se um binário que pode ser executado em qualquer plataforma, em um equipamento virtual. Denomina-se esse código binário único de bytecode (SILVA, 2011).

Uma vez que o Java é uma linguagem que não se baseia em uma plataforma, ou a um sistema operacional, é aplicada em diversos ambientes de desenvolvimento. Dentre esses ambientes, os mais conhecidos podem ser elencados como web, dispositivos e desktop, possuindo então categorias respectivas por Java SE, ME e EE. Assim, todo dispositivo que tenha uma JVM instalada, pode executar programas escritos por meio da linguagem Java. Com isso, os trabalhos do programador é facilitado, pois esse pode usar códigos de um ambiente para outro com menos preocupação, sendo que essa preocupação fica com os desenvolvedores e criadores Java (SILVA, 2011).

Conforme expõe Deitel (2003), os programas em Java são formados em partes denominadas como classes. Essas classes são subdivididas em métodos que executam tarefas e informam quando as tarefas são completadas. Porém, os programadores aproveitam das coleções de classes disponíveis em bibliotecas de classes Java. Essas bibliotecas de classes são chamadas de Java *Applications Programming Interfaces* – APIs.

Silveira (2008) explica que a linguagem Java e seu ambiente foram criados com a finalidade de solucionar problemas decorrentes na programação. Assim, apresentam características que a tornam diferentes de outras linguagens, de modo que sua existência

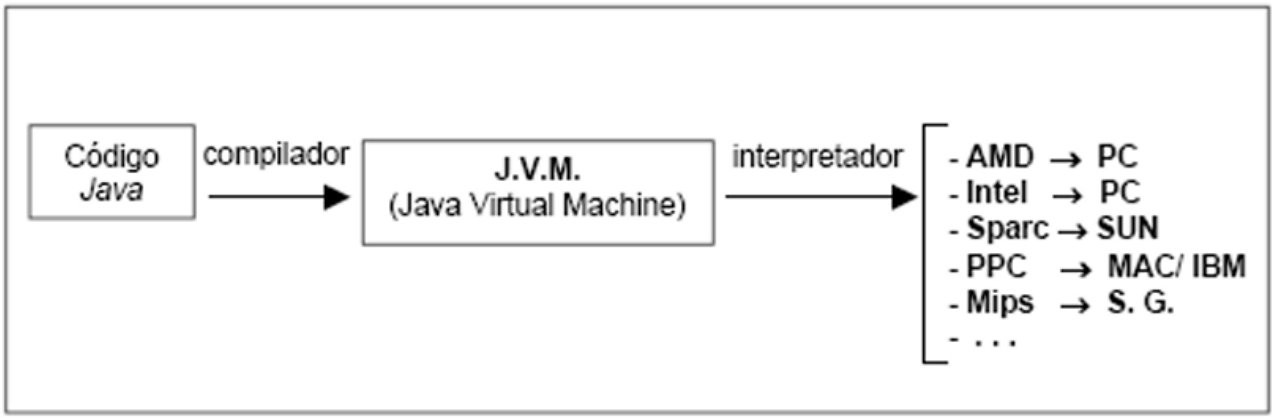
visa atender aos seguintes objetivos: simplicidade, orientação a objeto, portabilidade, recursos de rede e segurança.

Quanto à simplicidade, a linguagem foi criada com intuito de facilitar e minimizar o tempo de aprendizado, porém, com base as práticas mais atuais de desenvolvimento. Essa simplicidade permite que os *softwares* sejam concebidos a fim de rodarem em máquinas pequenas (GRIM, 2010).

No que tange a orientação a objeto, o sistema é baseado nos modelos Smalltalk e Simula67. Essa característica é muito relevante, uma vez que o projeto orientado a objeto permite uma facilidade na definição mais clara de interfaces e possibilita o reuso de códigos (GRIM, 2010).

Por meio da portabilidade, o *software* criado dentro do Java possui a independência de plataforma, conforme Figura 1. Isso é possível pelo fato do código ser compilado a um processador virtual, ou seja, o JVM, que posteriormente torna-se bytecodes (GRIM, 2010).

Figura 1 – Exemplo de interpretação e portabilidade de um código Java



Fonte: Grim, 2010

Quanto aos recursos de rede, a linguagem Java possui uma grande biblioteca de práticas que visam facilitar a cooperação com protocolos TCP/IP, como HTTP e FTP, com isso os aplicativos Java podem executar e acessar objetos por meio de rede via endereços de páginas da internet, de modo fácil como o acesso a qualquer sistema de arquivo local (GRIM, 2010).

Diante desta construção, sistema Java garante segurança, pois é possível executar programas via rede com restrições de execução. Durante as transferências de dados são realizadas verificações de código de byte, evitando acréscimo de vírus e cavalos de Tróia. Ainda, atende padrões de criptografia de chave pública do algoritmo RSA, garantindo a proteção de privacidade (GRIM, 2010).

Mediante todas essas características, a linguagem Java tem sido usada em diversos tipos de soluções, assegurando um ganho de tempo e maior desempenho ás aplicações (GRIM, 2010).

2.4 FRAMEWORK

O *Framework* consiste em uma coleção de classes abstratas colaborativas, representando um design abstrato, que é empregado no desenvolvimento de uma família de aplicações similares, com foco no reuso. Portanto, consiste em uma aplicação quase completa, demandando por configurações e implementações em determinados aspectos específicos para uma determinada aplicação do domínio. Esses sistemas possuem um elevado nível de reuso de *software*, permitindo que padrões arquitetônicos possam ser reutilizados em aplicações, melhorando de modo significativo a qualidade destas (JOHNSON e FOOTE, 1988; WOLFGANG,1994).

Olegário (2019) explica que o framework atua na orientação do programador quanto a forma que ele deve desenvolver seu projeto, ou uma parcela dele, de modo a indicar ao programador onde e como o código deve ser desenvolvido.

De acordo com Freitas (2017), um framework se diferencia de padrões de projetos em razão de sua especificidade em relação a um domínio de aplicações. Por sua vez, os padrões de projetos são empregados em qualquer tipo de aplicação. Quanto a uma aplicação orientada a objetos, há diferença. Por ser uma aplicação orientada a objetos apresenta uma implementação completa e executável de modo específico, enquanto os frameworks capturam a funcionalidade da aplicação e são executáveis por não atenderem de início o comportamento específico da aplicação desejada.

O framework propõe elementos flexíveis que devem ser modificados para o funcionamento de uma determinada aplicação e pontos que imutáveis e de alterações complicadas. Os elementos de flexibilidade de um framework são chamados de *hot spots*, elementos abstratos ou métodos que podem ser implementados para gerar uma aplicação executável. Por sua vez, os pontos imutáveis são chamados de *frozen spots*, que, diferentemente dos *hot spots*, são classes de um código já implementados, que em geral representam o núcleo da aplicação a ser gerada e ainda, sendo responsáveis por usar os *hot spots* implementados (MARKIEWICZ e LUCENA, 2001).

Conforme Olegário (2019), esse estilo de programação oferece uma estrutura mais sólida ao projeto, uma vez que todos os programadores seguem o mesmo padrão, além de seguir a mesma filosofia das bibliotecas, onde o objetivo principal é o reuso de código entre projetos que compartilham o mesmo domínio de problema.

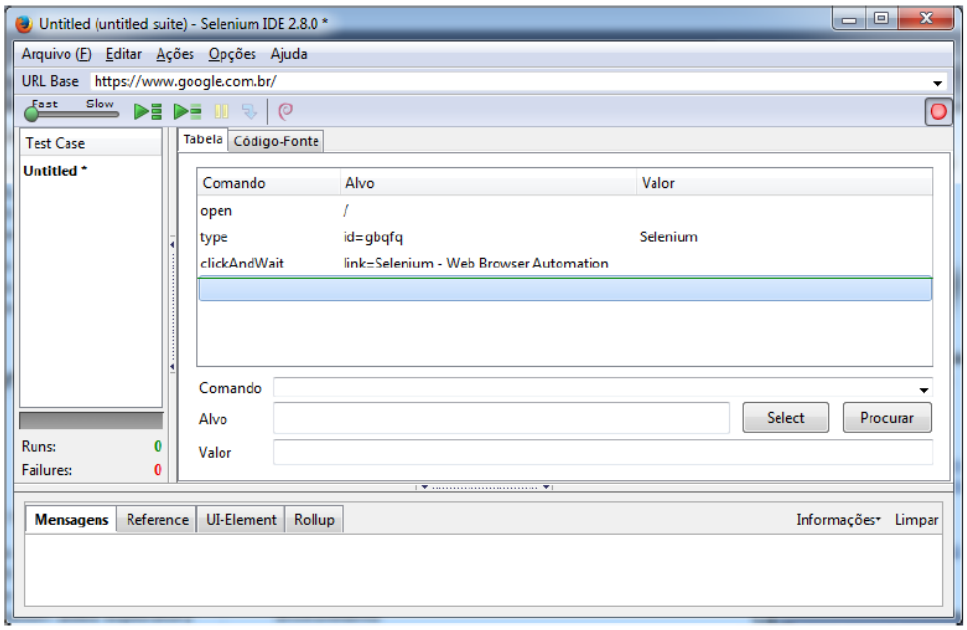
2.4.1 Framework de automação Selenium

O Selenium é uma ferramenta *open source* usada para criar testes automatizados funcionais para dispositivos executados sob protocolos *web* como HTTP ou HTTPS. O uso dessa ferramenta permite gerar *scripts* de teste, através da técnica *Record&Playback*. Diversos navegadores web suportam o Selenium, como por exemplo o Internet Explorer e o Mozilla Firefox (BRAGA, 2018).

Segundo Costa (2012, p. 104), outra importante característica desta ferramenta “está relacionada à sua compatibilidade com diversas plataformas. Pelo fato de ser desenvolvido na linguagem Java, pode ser executado em sistemas operacionais Windows, Linux e MacOS”.

O Selenium 2.0, ou Selenium WebDriver, como também denominado, consiste em um framework multi-plataforma usado para testar interface em aplicações web, foi criado em 2004 por Jason Huggins. O *software* possui sua própria Selenium IDE – IDE capaz de gravar e executar scripts de teste, podendo ser usado por usuários que desconhecem conhecimentos de linguagens de programação. O Selenium IDE consiste em um plugin do navegador Firefox. A Figura 2 apresenta a interface do Selenium IDE (IZABEL, 2014).

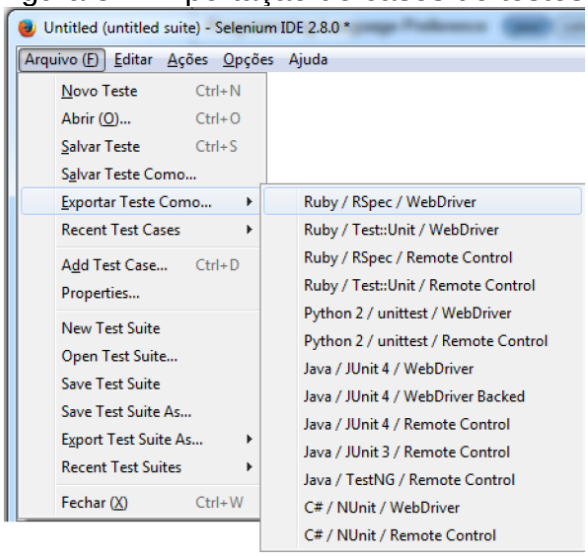
Figura 2 - Selenium IDE



Fonte: Izabel, 2014

A gravação de testes é simplificada, bastando que seja executado o caso de teste de modo manual enquanto o IDE faz o armazenamento de todos os passos seguidos. Em seguida, o *tester* usa o próprio IDE a fim de executar o plano de teste usando o Firefox como navegador de teste ou exportar os testes em alguma das linguagens possíveis e executá-lo no seu próprio IDE, com o navegador que desejar (Figura 3) (IZABEL, 2014).

Figura 3 – Exportação de casos de testes



Fonte: Izabel, 2014

O IDE permite gerenciar diversos casos de testes, além de garantir a viabilização do desenvolvimento de um plano de teste com vários casos de testes inclusos (IZABEL, 2014).

2.5 BEHAVIOR DRIVEN DEVELOPMENT

O *Behavior Driven Development* (ou BDD) é uma metodologia colaborativa que envolve todos os membros do time buscando o entendimento do sistema ou funcionalidade. O BDD visa trazer esse entendimento das regras de negócio forma acessível a todos os envolvidos.

Podemos utilizar o BDD na automação como uma forma de estruturar o desenvolvimento dos testes. Dessa forma, a cobertura dos testes fica de forma acessível para o time inteiro. Para aplicar o BDD na automação, podemos utilizar uma ferramenta bastante comum, chamada *Cucumber.*

2.5.1 Cucumber

O Cucumber é uma ferramenta que suporta a chamada de métodos utilizando a linguagem natural. Ele segue o formato de escrita do BDD (*Given/When/Then)*, e permite que façamos as chamadas dos métodos na automação de forma descritiva, permitindo que o fluxo de teste seja acessível e de fácil entendimento para o time.

2.6 VANTAGENS DA AUTOMAÇÃO DE TESTES

A automação de testes é aplicada a fim de reduzir a fator humano em atividades manuais que impactam no custo e tempo da demanda. Assim, seu objetivo é usar *softwares* para controlar as execuções dos *softwares* por intermédio de aplicações e estratégias. As ferramentas de teste automatizadas executam testes, reportam resultados e comparam resultados com testes anteriores. Testes realizados com essas

ferramentas podem ser executados repetidamente, a qualquer hora do dia (SILVA e CORRÊA, 2021).

Conforme Hooda (2012) a automação de testes aplica *software* a fim de controlar a execução de testes de *software*, comparando os resultados esperados com os resultados obtidos, a configuração das pré-condições de teste e outras funções de controle e relatório de teste.

De acordo com Fewster e Graham (1999 *apud* Lima, 2014, p. 21), os principais benefícios da automação de testes de software podem ser elencados como:

> a) Facilitam a execução de testes de regressão, tendo em vista que os casos de testes automatizados para uma versão do sistema podem ser reproduzidos nas versões seguintes com um esforço manual mínimo;
> b) Permite executar um maior número de testes em menor tempo, aumentando a frequência dos testes e trazendo mais confiança ao sistema;
> c) Realizar testes difíceis ou impossíveis de serem executados manualmente, como verificação de cálculos complexos ou testes de carga com um grande número de acessos;
> d) Melhorar a utilização de recursos, permitindo que os testadores reduzam o esforço em tarefas repetitivas, dando maior atenção ao planejamento dos testes e a execução de testes cuja automação é inviável;
> e) Permite aumentar a consistência dos resultados dos testes entre diferentes plataformas, já que as entradas e condições esperadas são as mesmas;
> f) Reduzir o *time to market,* tendo em vista a redução do tempo do projeto dedicado aos testes;
> g) Aumentar a confiança na qualidade do sistema. Um grande conjunto de testes automatizados bem planejados executados com êxito pode dar a equipe um sentimento de confiança na qualidade do produto a ser liberado.

A partir desses benefícios, é possível perceber o potencial da automação de testes na melhoria das atividades de teste de software (LIMA, 2014).

3. RESULTADOS E DISCUSSÕES

Baseado nos conceitos apresentados, foi desenvolvido um projeto de automação de teste de software, implementando as ferramentas e técnicas apresentadas.

3.1 DESCREVENDO A ESTRUTURA DO PROJETO

O padrão de projeto utilizado foi o PageObjects. Esse padrão separa as classes tendo como base as páginas que serão automatizadas. Geralmente divide-se conforme as seguintes etapas:

● O pacote *maps* agrega todas as classes relativas ao mapeamento dos elementos;

● O pacote *pages* agrega todas as classes relativas à implementação dos mapeamentos;

● O pacote steps agrega todos os *steps* relativos às páginas;

● O pacote *utils* agrega as classes referentes a elementos que são base da automação, como por exemplo criação de métodos de espera e suas variáveis globais, instanciamento de drivers e etc;

● O pacote *tests* agrega as classes de execução dos testes. No caso em estudo, possui somente uma classe de execução;

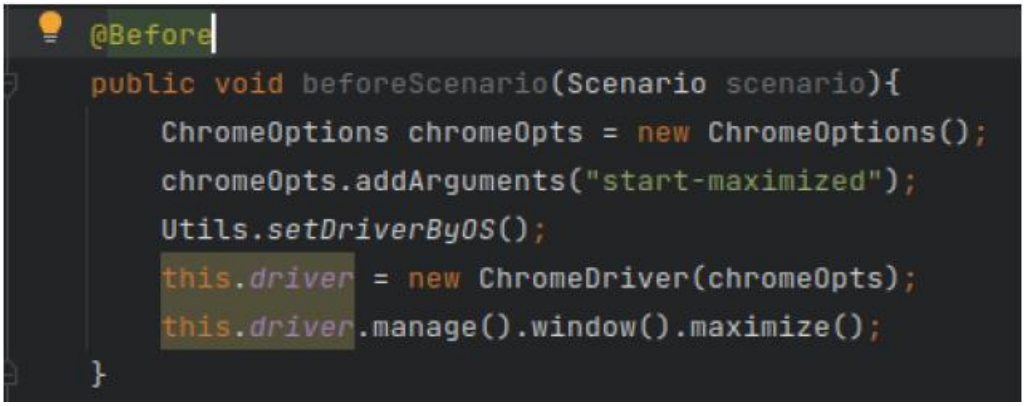
● O pacote *resources* agrupa todos os arquivos *.feature*, definem o fluxo de teste.

3.2 DETALHANDO O DESENVOLVIMENTO

A priori foram criados dois métodos na classe *TestRule:* o beforeScenario e o afterScenario.

O método beforeScenario é responsável por instanciar o driver do *browser*, abrir o *browser* antes do teste começar. Esse método utiliza a anotação @Before, do Cucumber. Essa anotação também é conhecida por *tagged hook*, e define esse método vai ser executado antes do teste iniciar (Figura 4).

Figura 4 - Método beforeScenario



Fonte: Autor, 2022

O método afterScenario é responsável por fechar o *browser* e encerrar o processo do mesmo. Esse método usa a anotação @After, que indica que ele vai ser executado após o teste, independentemente do ser resultado (Figura 5).

Figura 5 - Método afterScenari

```
@After
public void afterScenario(){
    if(driver != null){
        driver.close();
        driver.quit();
        driver = null;
    }
}
```

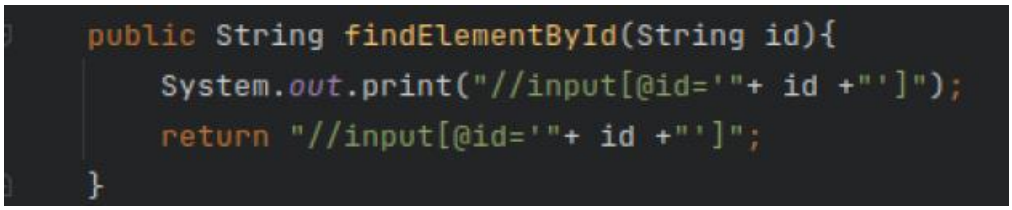
Fonte: Autor, 2022

Quando o browser for iniciado, é preciso interagir com os elementos HTML. Existem diversas formas para se fazer isso, sendo que, no presente estudo, foi escolhido o XPath.

O XPath é uma linguagem de consulta que busca e seleciona nós dentro de um documento XML. Ele é composto da seguinte forma: barras duplas (//), seguidas pela tag pai do elemento que queremos localizar. Abriu-se colchetes, e dentro deles definiu-se uma condição que irá ser localizada no documento XML. Esses elementos agrupados ficam da seguinte forma: //input[@id='input-search'].

Após elaborar o XPath do elemento que precisa-se interagir, criou-se uma classe com o nome da página que contém esse elemento, e dentro dessa classe, foi criado um método que retorna o XPath. Tem-se uma opção para melhorar o código: existem cenários onde a estrutura do XPath se repete para vários elementos dentro da página. Pode-se criar um xpath dinâmico, que possui uma variável dentro da condição. Essa variável foi definida pelo método utilizador do XPath, conforme Figura 6.

Figura 6 – XPath



Fonte: Autor, 2022

Em seguida foram criados todos os xpath's para os elementos que precisam interagir, separando-os pela página que eles estão.

Após a criação dos xpath's, foi criada a classe page relativa a classe map criada anteriormente. Os métodos da classe page são responsáveis por ações específicas da automação, como por exemplo: clicar em um elemento, ou escrever um texto em um campo de busca. Esses métodos vão utilizar os mapeamentos criados anteriormente para completar as ações definidas.

Figura 7 - Xpath's

Fonte: Autor, 2022

A Figura 7 ilustra método pesquisar ´produto. Esse método utiliza o xpath findElementById, que é um xpath dinâmico. Assim, foi preciso adicionar o id do elemento (input-search), e definir qual a ação que vai ser realizada com esse elemento. A ação é o sendKeys, que escreve um texto em um campo de busca. O texto a ser escrito não é definido nesse método: usou-se uma outra variável (produto), que será usada posteriormente.

Existem outros métodos para interagir com os elementos numa página web., destacam-se:

- click(): clica em um elemento;
- clear(): limpa os valores em algum elemento de texto (edit box ou text box);
- findElement(): encontra o elemento na página web.

Com os métodos da classe page finalizados, desenvolveu-se os *steps* do teste. Foi então criada uma classe com o nome da página na qual se pretende interagir, seguido de “Steps”. Por exemplo: homePageSteps. Dentro da classe criada, foram usadas anotações do Cucumber para definir a ordem dos steps. Seguindo o formato BDD (Behaviour Driven Development) juntamente com o Gherkin, destacam-se três momentos importantes:

- @Dado(): definição do contexto do teste;
- @Quando(): quando acontecer um evento;
- @Então(): saída esperada.

Dentro das anotações, usou-se a linguagem natural para descrever o step: @Dado (que acesso a homePage do site). Após a definição da anotação, desenvolveu-se o método que executa as ações definidas nas classes page, que por sua vez utiliza os mapeamentos desenvolvidos nas classes *map* (Figura 8).

Figura 8 - @Dado

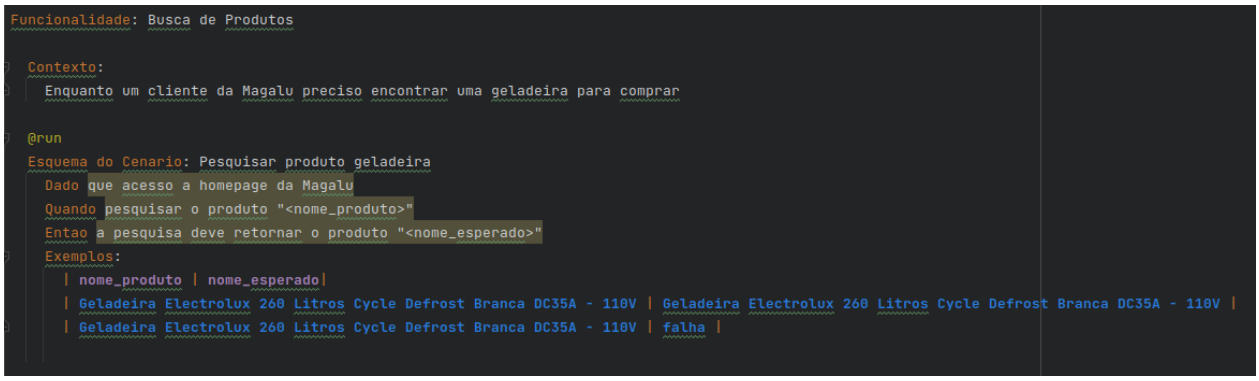
```
@Dado("que acesso a homepage da Magalu")
public void acessarEcommerce(){
    String ecommerceUrl = "https://www.magazineluiza.com.br/";
    TestRule.abrirNavegador(ecommerceUrl);
   }
```

Fonte: Autor, 2022

Por fim, após definição dos steps, estruturou-se a execução seguindo o cenário de teste a ser automatizado. Foi preciso estruturar esse cenário de teste usando o BDD, e criá-lo no arquivo .feature, que agrupa os cenários por funcionalidade.

Novamente, faz-se o uso de anotações, agora usando a @run, para definir que aquele cenário precisa ser executado (Figura 9).

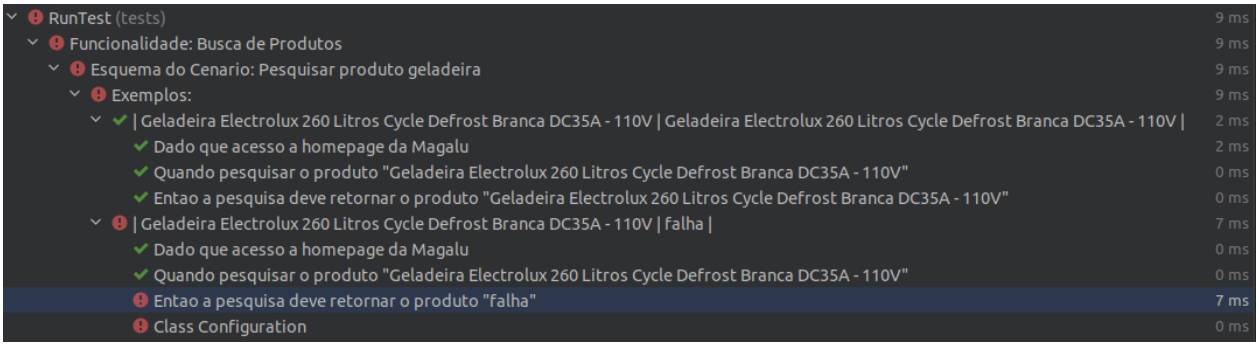
Figura 9 - Definição dos Steps



Fonte: Autor, 2022

Após a execução na IDE de desenvolvimento IntelliJ, obteve-se os resultados apresentados na Figura 10.

Figura 10 – Resultados



Fonte: Autor, 2022

Os cenários de sucesso são apresentados com o *check* na cor verde. Os cenários que falharam são apresentados com o *warning* na cor vermelha. Nos casos de falha, é necessário que aconteça uma avaliação, pois esse é um momento crucial para a elaboração do relatório dos resultados da execução. Nem sempre as falhas mostradas pela IDE são sinônimos de defeitos encontrados.

Em automações de *front-end*, expõe-se a elementos que não estão sob nosso controle. Mudanças no layout da página, instabilidade e lentidão podem gerar impacto

nas execuções. Esses cenários podem causar casos de testes com falha, mas não são defeitos. Nesses casos, executou-se os cenários novamente. Caso seja necessário, alterações no mapeamento ou nos métodos de execução são feitos para se adequarem melhor ao fluxo de teste.

Caso a execução encontre um defeito, esse defeito é reportado para o time analisar sua causa, os impactos gerados pelo defeito e, consequentemente, para implementar a correção.

CONCLUSÕES

O presente trabalho visou desenvolver um teste automatizado para sites de e-commerce. Assim, o objetivo foi alcançado a partir da criação de um código para automatização do sistema de teste para o e-commerce.

Por meio dos resultados obtidos, constatou-se a existência de cenários de sucesso e cenários de falhas. Os cenários que falharam são apresentados com o *warning* na cor vermelha. Nos casos de falhas, é necessário realizar uma avaliação, pois esse é um momento crucial para a elaboração do relatório dos resultados da execução. Nem sempre as falhas mostradas pela IDE são sinônimos de defeitos encontrados.

Em automações de front-end, expõem-se a elementos que não estão sob controle. Mudanças no layout da página, instabilidade e lentidão podem gerar impacto nas execuções. Esses cenários podem causar casos de testes com falha, mas não são defeitos. Nesses casos, executamos os cenários novamente. Caso seja necessário, alterações no mapeamento ou nos métodos de execução são feitos para se adequarem melhor ao fluxo de teste. Caso a execução encontre um defeito, esse defeito é reportado para o time analisar sua causa, os impactos gerados pelo defeito e para implementar a correção.

Em continuidade ao presente trabalho, serão implementados mais cenários para uma maior cobertura dos testes. Também poderão ser implementadas formas para lidar com massas de teste, e acesso a banco de dados para testes mais abrangentes no sistema.

REFERÊNCIAS

BERNARDO, P. C; KON, F. **A Importância dos Teste Automatizados**. Engenharia de Software Magazine, nº 3. 2008.

BRAGA, F. A. **Qualidade de software: proposta de automação de testes e de um processo ágil para uma empresa de software.** Monografia (Bacharel em Sistemas de Informação). Universidade do Sul de Santa Catarina. Florianópolis. 2018.

COSTA, Leandro Teodoro. **Conjunto de características para teste de desempenho: Uma visão a partir de ferramentas**. Porto Alegre, 2012.

DEITEL, H. M. **Java, Como Programar.** trad. Carlos Arthur Lang Lisbôa. – 4 ed. Porto Alegre: Bookman, 2003.

FREITAS, B. B. **Um framework para coleta de dados ambientais em cidades inteligentes.** Monografia (Bacharel em Engenharia de Software). Universidade Federal do Ceará. Quixadá. 2017.

FUKUYOSHI, B. H. **Programação para gerenciamento de rebanho leiteiro.** Monografia (Bacharel em Engenharia Elétrica). Universidade Estadual Paulista. Guaratinguetá. 2012.

GRIM, L. F. **Integração de tecnologias utilizando Java.** Monografia (Bacharel em Processamento de Dados). Faculdade de Tecnologia de Americana. Americana. 2010.
HOODA, R. **An Automation of Software Testing: A Foundation for the Future.** *In*: International Journal Of Latest Research In Science And Technology, v. 1, n. 2, p. 152–154, 2012.

IZABEL, L. R. P. **Testes automatizados no processo de desenvolvimento de softwares.** Monografia (Bacharel em Engenharia Eletrônica e de Computação). Universidade Federal do Rio de Janeiro. Rio de Janeiro. 2014.

JOHNSON, R. E.; FOOTE, B. **Designing reusable classes.** Journal of object-oriented programming, v. 1, n. 2, p. 22–35, 1988.

LIMA, A. G. S. **Automação de testes funcionais em ambientes web: um estudo de caso no laboratório de sistema e bancos de dados da universidade federal do Ceará.** Monografia (Bacharel em Engenharia de Software). Universidade Federal do Ceará. Quixadá. 2014.

LUFT, C. C. **Teste de software: uma necessidade das empresas.** Monografia (Bacharel em Ciência da Computação). Universidade Regional do Noroeste do Estado do Rio Grande do Sul. Santa Rosa. 2012.

MAGALHÃES, G. G. **Como Linguagens de Programação e Paradigmas Afetam Desempenho e Consumo Energético em Aplicações Paralelas.** Monografia (Bacharel em Ciência da Computação). Universidade Federal do Rio Grande do Sul. Porto Alegre. 2016.

MARKIEWICZ, M. E.; LUCENA, C. J. de**. Object oriented framework development.** Crossroads, ACM, v. 7, n. 4, p. 3–9, 2001.

OLEGÁRIO, G. F. **Um framework para geração de testes automatizados para aplicações mobile.** Monografia (Bacharel em Ciências da Computação). Brasil. 2019.

PRESSMAN, Roger S. **Engenharia de Software: uma abordagem profissional**. 7. ed. São Paulo: Pearson Makron Books, 2011.

SILVA, E. N. S. **Desenvolvimento do framework java-fácil.** Monografia (Bacharel). Instituto Municipal de Ensino Superior de Assis IMESA. Assis. 2011.

SILVA, W. T. V.; CORRÊA, J. S. **Elaboração de um mínimo processo viável para automação de testes funcionais em fábricas de software.** Monografia (Bacharel em Engenharia de Computação). Universidade Evangélica de Anápolis – UniEVANGÉLICA. Anápolis. 2021.

WOLFGANG, P. **Design patterns for object-oriented software development.** [S.l.]: Reading, Mass.: Addison-Wesley, 1994.