Automação de testes no projeto TelEduc versão Core

Mateus S. Gasparotto, André C. da Silva, Daniela Marques

Grupo de Pesquisa Mobilidade e Novas Tecnologias de Interação
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas
Instituto Federal de Educação, Ciência e Tecnologia de São Paulo (IFSP)
Campus Hortolândia – SP – Brasil

gasparotto.mateus@aluno.ifsp.edu.br, {andre.constantino, marquesdaniela}@ifsp.edu.br

Abstract. *Software testing is an extremely important step for the quality of a product. There are different types of tests for validation and verification of different characteristics in a software, such as performance, security, availability, etc. This article presents the automation of functional tests using the Cucumber tool for the offline functionality of the Core project. Core is a Virtual Learning Environment developed by the Nucleus of Informatics Applied to Education at State University of Campinas and is an evolution of the TelEduc project that lasted from 1997 to 2017. With the automation of this step, the effort of the Core project developers could be used to develop other functionalities, as they were the ones who performed the tests manually.*

Resumo. *Teste de software é uma etapa de extrema importância para a qualidade de um produto. Existem diferentes tipos de testes para validação e verificação de diferentes características dentro de um software, tais como performance, segurança, disponibilidade, etc. Este trabalho apresenta a automação de testes funcionais utilizando a ferramenta Cucumber para a funcionalidade offline do projeto Core. O Core é um Ambiente Virtual de Aprendizagem desenvolvido pelo Núcleo de Informática Aplicada à Educação da Universidade Estadual de Campinas e é uma evolução do projeto TelEduc que durou de 1997 até 2017. Com a automação desta etapa, o esforço dos desenvolvedores do projeto Core pode ser usado para desenvolver outras funcionalidades, pois os testes eram feitos manualmente por eles.*

1. Introdução

Testes são uma etapa de extrema importância no ciclo de desenvolvimento de um *software*, pois com eles são validados as funcionalidades visando seu funcionamento e adequação aos requisitos especificados (FELIX, 2016). A utilização desta etapa é fundamental para a garantia de qualidade do produto desenvolvido. À medida que os testes são realizados, os resultados obtidos, bem sucedidos ou não, são avaliados para auxiliar na tomada de decisão, planejamento de novos testes e avaliação da qualidade do processo.

Existem diferentes tipos de testes (FELIX, 2016), por exemplo:

- Testes de caixa branca: nesse tipo de teste se tem acesso ao código fonte onde o profissional pode analisar com atenção determinadas etapas. No processo é analisado por qual caminho ocorre o fluxo de dados verificando se há passagem correta nas condições esperadas;
- Testes de caixa-preta: oposto do teste de caixa branca, nesse tipo de teste a pessoa não tem acesso ao código fonte, sendo analisado com base nos requisitos funcionais. Por conta disso, esse teste também é chamado de teste funcional;
- Performance: nesse teste é verificado o desempenho do *software*, se as requisições realizadas são rapidamente respondidas, se não há demora para carregar componentes e se a experiência do usuário é satisfatória.

Também existem diversos outros tipos de testes, tais como o de segurança, integração, usabilidade (FELIX, 2016), etc.

O fato é que a realização dos testes pode ser um processo repetitivo, requerendo muita atenção, por isso a automação dessa etapa para cenários críticos podem trazer benefícios para o projeto. Como citado no livro "Testes de *Software*" de Rafael Felix (2016) a automação de testes assegura um nível de qualidade que dificilmente seria alcançado com um trabalho manual feito por humanos.

Atualmente existem diversas ferramentas para automação de testes à disposição de uma equipe com suporte às mais variadas linguagens. Algumas delas já são bem conhecidas e utilizadas no mercado, tais como: Selenium, Visual Studio Test Professional, TestComplete, Cucumber, etc.

Como objeto de estudo, foi selecionado o ambiente Core (ARANTES; FREIRE, 2018), ambiente virtual de aprendizagem em desenvolvimento no Núcleo de Informática Aplicada à Educação (NIED) da Universidade Estadual de Campinas (UNICAMP). A equipe de desenvolvimento não está criando a automação dos testes, sendo testes manuais feitos corriqueiramente, o que acaba despendendo esforços que poderiam ser utilizados para o desenvolvimento de outras funcionalidades do sistema, bem como agilizar a identificação de problemas.

O objetivo deste trabalho colabora com os esforços da equipe, ao dispor, no final do projeto, um conjunto de testes automatizados para testar cenários da funcionalidade offline da aplicação Core. O artigo foi escrito da seguinte forma: na Seção 2 é trabalhado o referencial teórico, na Seção 3 é feito um estudo acerca dos trabalhos correlatos ao aqui desenvolvido, na Seção 4 é apresentado a metodologia utilizada para o desenvolvimento, Seção 5 relata o desenvolvimento dos testes, a Seção 6 traz os resultados da execução da automação e por fim toda a conclusão do trabalho na Seção 7.

2. Referencial teórico

As subseções a seguir apresentam definições que serão utilizadas no decorrer deste trabalho. Serão descritos os conceitos de qualidade e testes de *software*.

2.1. Qualidade de *software*

Software não é algo físico e de acordo com Galloti no livro "Qualidade de *software*" (2015) é necessário considerar dois fatores importantes para a garantia de qualidade:

1. Funcionamento do *software*, seus processos internos, geralmente inacessíveis ao usuário de modo direto;
2. Relação usuário/*software*;

Esse pensamento traz a concepção de que qualidade de *software* está diretamente relacionada a um gerenciamento rigoroso de requisitos, uma gerência efetiva de projetos e ao desenvolvimento baseado em um processo bem definido, gerenciado e em melhoria contínua (GALLOTI, 2015).

Diferente de um processo de produção linear como de uma montadora, o processo para o desenvolvimento de um *software* contém diferentes etapas. Da concepção até a finalização de um produto, são realizados diversos passos e cada um deles precisam da garantia de que aquilo que está sendo feito contém qualidade.

Nesse pensamento, normas foram criadas com a finalidade de garantir essa característica. Temos por exemplo a ISO 12207, norma regulamentadora para a qualidade do processo de desenvolvimento; a ISO 9126, com características da qualidade de produtos de *software*; e também a ISO 14598 que contém guias para avaliação de produtos de software baseados na utilização da ISO 9126.

A ISO 9126, por exemplo, fornece um modelo de propósito geral dividido em características e subcaracterísticas para o controle de qualidade (Figura 1).

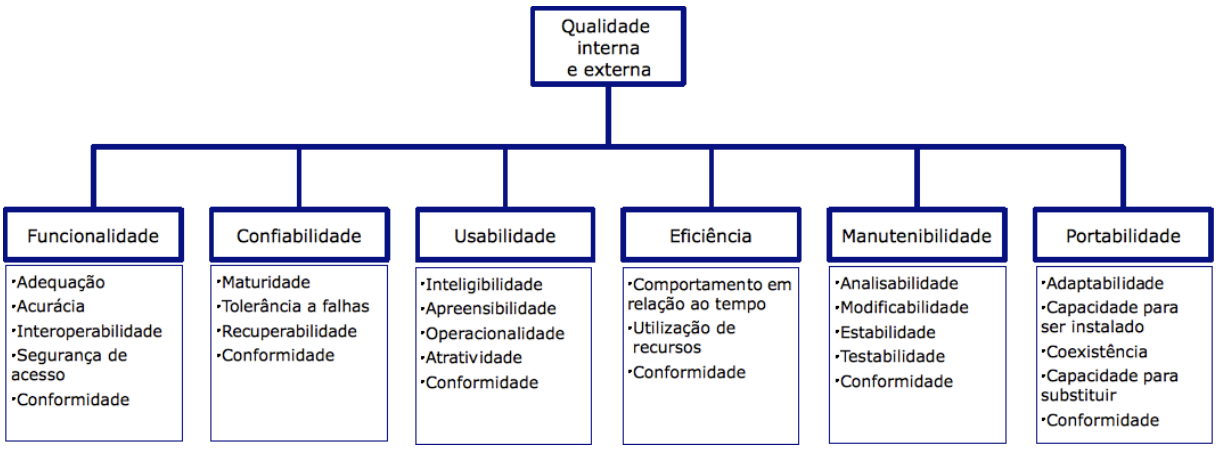


Figura 1. Característica e Subcaracterísticas de funcionalidade da norma ISO 9126 (imagem com licença *creative commons*)

Para garantir cada uma dessas características são realizados testes no *software*, sejam eles automatizados ou manuais. Como destacado em *Software Testing - Testing Across the Entire Software Development Life Cycle* (2007), de acordo com o governo dos EUA, estimou-se que houve perdas de negócios de 59,5 bilhões de dólares de 2000 a 2007 devido a baixa qualidade de *software*. Tal perda poderia ser evitada com a utilização de testes completos e minuciosos.

2.2. Testes

Testes, no âmbito de *software*, são uma etapa importante no fluxo de um projeto onde o objetivo central se resume ao de descobrir erros no sistema durante as etapas de desenvolvimento.

Conforme comentado no texto da Seção de Introdução, existem diversas formas de testes. Testes unitários, testes de caixa-preta/caixa-branca, testes de performance e diversos outros são utilizados no mercado nos mais variados fluxos de desenvolvimento trazendo a garantia de qualidades como funcionamento, segurança, disponibilidade e performance do produto. Existem técnicas avançadas de desenvolvimento que são baseadas em testes, como ocorre com o caso de *Test driven development* (TDD) onde o desenvolvimento é guiado a partir da execução de testes unitários.

2.3. *Behavior Driven Development* (BDD)

Criado por Dan North em 2003, temos o BDD (*Behavior Driven Development*) que, por tradução livre, pode ser lido como Desenvolvimento Dirigido a Comportamento.

O intuito do BDD é de melhorar a comunicação dentro de um time de desenvolvimento, desde analistas de requisitos a desenvolvedores, isso para que todos tenham o mesmo entendimento de uma funcionalidade (ANDERLE, 2015). Sendo assim, se faz de

um processo ágil de desenvolvimento que fecha a lacuna entre o pessoal técnico e o pessoal não técnico, pois encoraja a colaboração entre o time melhorando o entendimento sobre o que deve ser feito integrando regras de negócios com linguagem de programação.

No BDD, o desenvolvimento é guiado a partir de comportamentos esperados pelo sistema escritos em cenários seguindo uma estrutura chamada *Gherkin* (*Given/When/Then*), descrevendo o caminho que o usuário irá executar em uma aplicação e o comportamento esperado por ele.

Conforme John Smar no livro "*BDD in Action: Behavior-Driven development for the whole software lifecycle*" (2014), os principais benefícios ao utilizar a técnica BDD são:

- **Redução de desperdícios**: BDD foca esforço de desenvolvimento na identificação e entrega de funcionalidades que trazem valor ao negócio. BDD ajuda a evitar o desperdício de esforço em funcionalidades que não estão alinhadas com o negócio;
- **Comunicação**: o BDD melhora a comunicação entre analistas de negócios, desenvolvedores e testadores permitindo expressar requisitos de maneira mais testável, o que traz um entendimento facilitado da funcionalidade;
- **Redução de custos**: a redução de custos é consequência direta da redução de desperdícios, como o esforço será empregado apenas ao que é necessário ao negócio, não haverá gastos desperdiçados;
- **Mudanças facilitadas e seguras**: BDD facilita mudanças e evoluções de um projeto, por exemplo as especificações geradas com *Gherkin* fornece um conjunto abrangente de casos de testes que podem ser automatizados e consequentemente reduzir o risco de regressão;
- **Aumento na velocidade de entregas**: já que o BDD descreve um conjunto abrangente de testes automatizados, os testadores deixam de aplicar esforço manual repetitivo para cada nova entrega.

2.3.1. *Gherkin Steps*

North (2003) sugeriu que a linguagem utilizada no BDD poderia ser extraída já no levantamento de requisitos propondo o uso da técnica de histórias de usuário para sua composição. O modelo foi estruturado para quebrar a história em seus fragmentos constituintes e automatizá-las (ANDERLE, 2015). Os critérios de aceite foram divididos em cenários seguindo a seguinte estrutura:

***Given* (Dado)** - Dado um contexto inicial

***When* (Quando)** - Quando acontecer um determinado evento

***Then* (Então)** - Então verificar os resultados

Para demonstração, será usado de exemplo um login em sistema de rede social. Na Figura 2 é mostrado a comparação entre o formato de história de usuário tradicional para o formato proposto com utilização do BDD.

2.4. Cucumber

Inicialmente desenvolvido para a linguagem Ruby, Cucumber é uma ferramenta de testes *open-source* que dá suporte ao BDD (Cucumber, 2019). Diversas linguagens de programação reescreveram o Cucumber para suas linguagens, como o CucumberJVM do Java, utilizado

neste trabalho. Esta implementação do Cucumber foi escolhida para o trabalho devido ao conhecimento prévio da ferramenta.

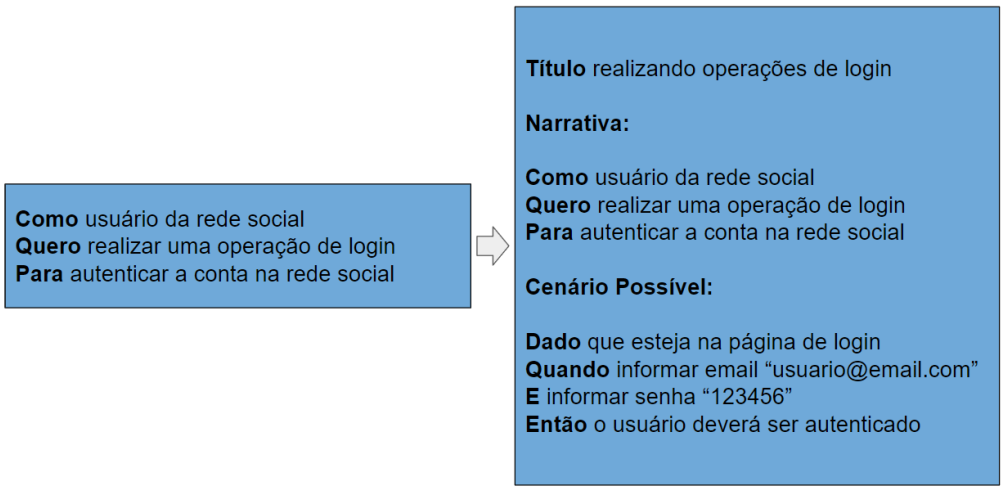


Figura 2. Comparação entre formato tradicional e formato BDD para História de Usuário.

Conforme documentação oficial (Cucumber, 2019), o Cucumber lê cada passo dos cenários especificados escritos em texto simples e valida se o *software* faz o que essas especificações dizem executando o método de código (Java) relativo ao passo do cenário.

Considere o cenário presente na Figura 3. Este é um arquivo de extensão *.feature* utilizado no Cucumber. Na linha 1 é definido o idioma que será utilizado no arquivo, no caso foi definida a língua portuguesa. É definida a funcionalidade na linha 3 "Login suap" e, em sequência, seguindo a estrutura do *Gherkin*, na linha 5 começa a descrição do cenário "Login com senha errada".

```
1  #language:pt
2
3  Funcionalidade: Login suap
4
5    Cenário: Login com senha errada
6      Dado que esteja na página: "https://suap.ifsp.edu.br/"
7      Quando fizer login com o prontuário "HT3001521" e a senha
       "senhaerrada"
8      Então o site deverá retornar erro dizendo: "Por favor, entre
       com um usuário e senha corretos. Note que ambos os
9          campos diferenciam maiúsculas e minúsculas."
```

Figura 3. Exemplo cenário Cucumber (*Gherkin*)

Na execução do Cucumber ele irá procurar as anotações (*annotations*) que representa cada um desses passos nas classes Java (*Steps Definitions*) para, então, realizar a execução do teste.

Cada um dos elementos do *Gherkin* possui sua própria anotação (@Dado, @Quando, @Então) e o texto de conteúdo de cada elemento é usado como parâmetro da anotação (Ex: linhas 26, 32 e 44 da Figura 4). O desenvolvedor de testes deve criar os diferentes métodos que executarão, na linguagem de programação adotada, o desejável em cada elemento do *Gherkin* e anotá-los com a correta anotação e texto. Por meio da análise da anotação e da comparação de *strings* do conteúdo da anotação, o Cucumber solicitará a execução dos métodos.

```java
@Dado("que esteja na página do suap: {string}")
public void que_esteja_na_página(String url) {
    webDriver.get(url);

}

@Quando("fizer login com o prontuário {string} e a senha {string}")
public void fizer_login_com_o_prontuário_e_a_senha(String login, String password) {
    String PRONTUARIO_INPUT = "//*[@id=\"id_username\"]";
    String PASSWORD_INPUT = "//*[@id=\"id_password\"]";
    String LOGIN_BUTTON = "//*[@id=\"login\"]/form/div[5]/input";

    webDriver.findElement(By.xpath(PRONTUARIO_INPUT)).sendKeys(login);
    webDriver.findElement(By.xpath(PASSWORD_INPUT)).sendKeys(password);
    webDriver.findElement(By.xpath(LOGIN_BUTTON)).click();
}


@Então("o site deverá retornar erro dizendo: {string}")
public void o_site_deverá_retornar_erro_dizendo(String errorMessage) {
    String ERROR_MESSAGE = "//*[@id=\"login\"]/p";
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    String pageReturn = webDriver.findElement(By.xpath(ERROR_MESSAGE)).getText();
    Assert.assertEquals(errorMessage, pageReturn);
}
```

Figura 4. Exemplo cenário Cucumber (Java)

Por fim, o Cucumber então verifica se o *software* está em conformidade com a especificação indicando sucesso ou falha do passo de acordo com a implementação do método.

2.5. Projeto Core

Iniciado no começo de 2018 por Flávia Arantes e Fernanda M. P. Freire, o projeto Core é um Ambiente Virtual de Aprendizagem (AVA) construído aproveitando o *know-how* do projeto TelEduc. Ele ainda está em evolução, desenvolvendo ferramentas para Educação a Distância para a criação de um ambiente de aprendizado eletrônico que funcione com pouca largura de banda ou em conexões intermitentes.

O TelEduc foi desenvolvido entre 1997 e 2017 pelo Núcleo de Informática Aplicada à Educação (NIED) com parceria do Instituto de Computação (IC) também da Universidade Estadual de Campinas (UNICAMP). É um projeto *open-source* que visa a criação, participação e administração de cursos web.

O objetivo do projeto Core é o de projetar um ambiente apenas com as ferramentas essenciais e com uma melhor UI/UX (*User Interface*/*User Experience*, em português, Interface de Usuário/Experiência de Usuário) para usuários habituados ao uso das redes sociais. Seu propósito é o de alcançar um público que vive em áreas isoladas ou com acesso limitado à Internet, como escolas rurais, populações indígenas e/ou quilombolas, etc. (ARANTES; FREIRE, 2018). O Anexo III contém capturas de algumas das telas do Core.

Como destacado pelas autoras, um ambiente *e-learning* que atenda esse público deve possuir a funcionalidade de trabalhar com pouco ou nenhum acesso a Internet, ter versão para dispositivos móveis, ser simples e ter um conjunto enxuto de funcionalidades. Ou seja, o Core está sendo construído com um conjunto de requisitos relacionados a ausência de conexão para permitir que, estando sem conexão, o usuário utilize a aplicação e, ao retomar a conexão com a Internet, haja uma sincronização dos dados locais com dados do servidor. Tal recurso foi batizado pela equipe de *offline*.

A fim de verificar as funcionalidades relacionadas ao uso da aplicação sem conexão e a sincronização dos dados, a equipe se organizou definindo "momentos" (que foi como foi chamado cada caminho de usuário) considerando a diversidade de dispositivos e navegadores, visto que a aplicação é multidispositivo. Os testes são executados manualmente e demandam muito tempo. A cada modificação do código, testes são necessários para validar que outras funcionalidades continuam funcionando. Assim, a automação desses testes funcionais na plataforma *web* (Google Chrome) se tornaria útil para a equipe de desenvolvimento, otimizando o tempo de desenvolvimento dos programadores.

2.5.1. Testes no core funcionalidade *offline* até o desenvolvimento deste trabalho

Até o desenvolvimento deste trabalho, os testes do Core para a funcionalidade *offline* estavam documentados em um arquivo compartilhado entre os desenvolvedores onde foram descritos os momentos. Segundo relato dos desenvolvedores, devido a carga alta de trabalho para estes testes foi necessário o envolvimento de toda a equipe para dividir atividades entre eles, pois é uma equipe pequena de cerca de aproximadamente 8 pessoas. Esses testes manuais demandam um grande esforço por conta da necessidade de simular a falta de conexão, sendo necessário alterar configurações de dispositivos para isso. A Tabela 1 apresenta um exemplo, onde é necessário ter a visualização do material de apoio com pontos a serem interrompida a conexão.

Tabela 1. Visualizando Material de Apoio com pontos a interromper a conexão

Tarefa A4.1. O aluno deve acessar o mural do curso "Teste Offline".
Tarefa A4.2. O aluno deve mudar as configurações do seu dispositivo para se desconectar da rede (sugere-se mudar as configurações para o modo avião). Observar: A aplicação Core emitiu alguma informação visual em relação a indisponibilidade de conexão (R2)?.
Tarefa A4.3. O aluno deve acessar a ferramenta Material de Apoio.
Tarefa A4.4. O aluno deve visualizar e ler os materiais publicados. Observar: Foi possível visualizar os arquivos em formato PDF e PNG? Não foi possível acessar a página do link?
Tarefa A4.5. O aluno deve escrever uma postagem no mural respondendo as questões: 1. Foi possível acessar os arquivos em formato PDF e PNG disponíveis na ferramenta de Material de Apoio enquanto você estava sem conexão?
Tarefa A4.6. O aluno deve mudar as configurações do seu dispositivo para se conectar à rede. Observar: A aplicação Core emitiu alguma informação visual em relação a disponibilidade de conexão (R4).
Tarefa A4.7. O aluno deve observar se a sua postagem permanece no Mural (R8).

Este é um bom exemplo da complexidade dos testes. Na tarefa A4.2 e A4.6 se faz necessário alterar configurações dos dispositivos para simular falta de conexão. Isso pode variar desde desligar o wi-fi do dispositivo até desligar o roteador padrão da rede, gerando dificuldades para os que estão testando.

Visando o alto custo de esforço para realização desses momentos, este trabalho se propôs a automatizar os testes, transcrevendo cada momento para BDD e gerando automatizações com uso de tecnologias de automação.

3. Trabalhos correlatos

Trabalhos correlatos são projetos que contém similaridades em relação ao projeto aqui desenvolvido. Eles são utilizados para o levantamento de ideias partindo da análise, teste ou pesquisa sobre os mesmos. Assim, nesta Seção, são destacados trabalhos que contêm essas semelhanças com a proposta aqui desenvolvida.

3.1. Automação de testes utilizando a ferramenta Cucumber

Neste trabalho, Felipe Curti e Felipe Dallilo (2022) trazem a automação de um cenário para cadastro de usuário em um *site*, onde é detalhado toda a implementação do desenvolvimento com a utilização de ferramentas e tecnologias como Cucumber, Java e JUnit.

Ao final do trabalho eles concluem que a automação com Cucumber reduz muito o tempo que um analista de testes empreenderia em uma tarefa em comparação com o processo manual (CURTI; DALLILO, 2022).

3.2. Automação de Testes em Aplicações de BPMS:um Relato de Experiência

Desenvolvido por Jéssica Moura e Andrea Charão (2015), este trabalho também traz como objeto de estudo a automação de testes. As autoras relatam no artigo a experiência de automatizar os testes de uma aplicação desenvolvida com apoio de sistemas de gestão de processos de negócio (*Business Process Management Systems – BPMS*).

Para isso, implementou-se um mesmo processo usando dois diferentes BPMS: Bonita e Activiti. Submeteu-se as aplicações Web resultantes a testes de carga e testes funcionais, utilizando-se as ferramentas Apache JMeter, Selenium e Cucumber. Os resultados evidenciam a viabilidade e as limitações na automação de testes deste tipo de aplicação (MOURA; CHARÃO, 2015).

4. Materiais e Métodos

Neste trabalho foi estudado a ferramenta Core para conhecimento da funcionalidade *offline* do sistema para automação de testes e também realizado um estudo sobre a automação de testes para embasamento geral.

A ferramenta base utilizada para a automação foi o CucumberJVM, uma implementação do Cucumber em Java, utilizando-se da IDE IntelliJ da JetBrains para desenvolvimento do código fonte. Para implementação dos testes em navegador *web* se utilizou a biblioteca selenium.

Com o projeto configurado foi realizada uma análise dos testes que eram feitos manualmente entendendo a necessidade de cada história de usuário. Com isso, cada história foi transcrita para um cenário seguindo o padrão BDD e validado com André Constantino da Silva (participante do projeto Core dentro do NIED/UNICAMP).

Projeto configurado e cenários identificados, iniciou-se o desenvolvimento da automação utilizando os recursos do Cucumber, Java e Selenium, o qual é retratado com detalhes na Seção Desenvolvimento. Automação codificada, os testes foram executados no navegador Chrome coletando os resultados por meio de um relatório gerado pelo próprio Cucumber para sucesso ou falha dos cenários.

5. Desenvolvimento

Nesta Seção é retratada toda a implementação da automação dos testes.

5.1. *Setup* projeto

Neste projeto as seguintes tecnologias foram utilizadas: Apache Maven 3.8.3, Openjdk 11.0.14.1, CucumberJVM 6.10.4 e Selenium 4.0.0.

5.2. Prova de Conceito

Para mostrar o funcionamento do projeto utilizando as tecnologias destacadas na Subseção 5.1 será utilizado o exemplo da Figura 3 escrita em BDD no Cucumber.

5.2.1. *Step Definitions*

Uma *Step Definition* (Definição de etapa) é um método com uma expressão que relaciona uma ou mais *Gherkin Steps*. Quando o Cucumber executar um passo escrito em *Gherkin* em algum cenário ele irá procurar por sua *Definition* correspondente.

No caso do exemplo da Figura 3 temos 3 passos *Gherkin*, por isso o cucumber irá procurar pela definição dessas etapas.

Dentro do método, é colocado todo o código necessário para a execução da *Step* (vide Figura 5). Como o cenário em questão se faz do uso do navegador, foi utilizado a ferramenta Selenium, onde foi previamente baixado para o projeto o driver do *Chrome* sendo utilizado via Interface *WebDriver* do Selenium.

A forma como Selenium foi utilizado segue uma lógica de funcionamento simples:

- Primeiro é feito um trabalho manual do teste identificando informações de *Uniform Resource Locator* (URL) e identificador xpath dos elementos da tela que há interação (obtido à partir da inspeção de elementos do navegador);
- É criada uma instância do WebDriver do Selenium para automatizar operações no navegador web (no caso o Google Chrome);
- Para navegar para o site do Core utilizou-se do método get do WebDriver (já instanciado) passando a URL como parâmetro (como visto na linha 28 da Figura 5);
- Com o xpath dos elementos já obtido foi utilizado métodos do WebDriver para escrever textos (exemplo linha 38 da Figura 5) e acionar eventos de *click* em botões (linha 40 da Figura 5).

A forma como os cenários foram implementados seguem a mesma dinâmica explicada acima com as informações adaptadas conforme o passo. A implementação de todos os cenários pode ser vista no anexo ao final do artigo com todo o código fonte.

```java
@Dado("que esteja na página do suap: {string}")
public void que_esteja_na_página(String url) {
    webDriver.get(url);

}

@Quando("fizer login com o prontuário {string} e a senha {string}")
public void fizer_login_com_o_prontuário_e_a_senha(String login, String password) {
    String PRONTUARIO_INPUT = "//*[@id=\"id_username\"]";
    String PASSWORD_INPUT = "//*[@id=\"id_password\"]";
    String LOGIN_BUTTON = "//*[@id=\"login\"]/form/div[5]/input";

    webDriver.findElement(By.xpath(PRONTUARIO_INPUT)).sendKeys(login);
    webDriver.findElement(By.xpath(PASSWORD_INPUT)).sendKeys(password);
    webDriver.findElement(By.xpath(LOGIN_BUTTON)).click();
}


@Então("o site deverá retornar erro dizendo: {string}")
public void o_site_deverá_retornar_erro_dizendo(String errorMessage) {
    String ERROR_MESSAGE = "//*[@id=\"login\"]/p";
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    String pageReturn = webDriver.findElement(By.xpath(ERROR_MESSAGE)).getText();
    Assert.assertEquals(errorMessage, pageReturn);
}
```

Figura 5. Exemplo *Steps Definitions*

5.2.2. *Cucumber Report*

Com os cenários escritos no padrão Cucumber e com suas respectivas definições codificadas, podemos executar os passos e gerar um relatório com o *status* da execução utilizando as opções que o cucumber nos dá.

Com as devidas configurações e com utilização do comando maven: *mvn test*, é gerado um relatório com extensão .html na pasta target do repositório (Figura 6).

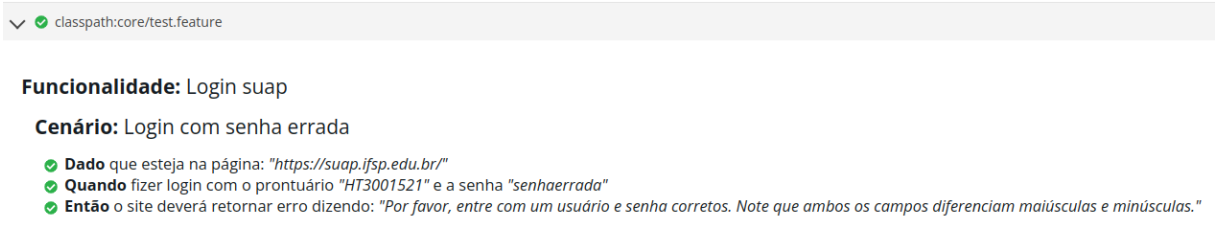


Figura 6. Exemplo de relatório emitido pelo Cucumber após a execução dos testes automatizados.

O relatório mostra os cenários que foram executados e marca sucesso para as etapas que executaram conforme esperado.

5.3. Funcionalidade Navegação *Offline*

Conforme já comentado, o foco principal deste trabalho está em verificar as funcionalidades da navegação *offline* na plataforma Core, principal proposta da plataforma. Nos momentos de testes desenvolvidos pela equipe foi feito um trabalho também de validação de usabilidade. Neste trabalho limita-se apenas os testes focados para validação das funcionalidades. Será discorrido sobre os cenários e definições de cada passo para cada um dos 5 momentos criados pela equipe.

5.3.1. Contexto

Dentro de uma funcionalidade é comum que se tenha passos iguais entre os diferentes cenários, assim é possível criar um bloco de passos chamado de contexto. O contexto é uma forma de encapsular esses passos para que sejam executados em todos os cenários na automação com Cucumber. Em nossa funcionalidade existem 2 passos comuns para todos os cenários que é o de estar autenticado no Core para execução dos testes. Por isso os passos de acessar a aplicação e se autenticar foram encapsulados nas linhas 6 e 7 da Figura 7. O código elaborado está disponível no GitHub pelo link <https://github.com/Gasparott0/teleduc-core-tests.git>.

```
5    Contexto:
6      Dado que o aluno esteja na página do teleduc: "https://proteo.nied.unicamp.br/prod"
7      E esteja conectado
```

Figura 7. Contexto

5.3.2. Cenário 1

A equipe de desenvolvimento planejou o primeiro momento de testes chamado de “Boas vindas e ambientação”, apresentado na Tabela 2. Este cenário envolve o aluno entrar em um curso existente, visualizar as mensagens postadas e publicar uma mensagem. Destaca-se que não há, neste cenário, interrupção da conexão ou sincronização.

Tabela 2. Momento - Boas Vindas e Ambientação

Tarefa A1.1. O aluno deve entrar no curso “Teste *Offline*”
Tarefa A1.2. O aluno deve ler o post publicado que está fixado pelo professor.
Tarefa A1.3. O aluno deverá publicar um comentário no post com um nome da música de Natal ou Ano Novo.

Considerando o momento do documento de testes mostrado na Tabela 2, foi feito um entendimento da necessidade do teste e sua transcrição para o BDD seguindo o formato Gherkin ao qual é possível observar na Figura 8.

```
9    Cenário: Boas vindas e ambientação
10     Quando o aluno entrar no curso Teste Offline
11     E o aluno ler o post fixado pelo professor
12     Então o aluno deverá publicar um comentário em um post escrito "jingle bell"
```

Figura 8. Cenário - Boas Vindas e Ambientação

5.3.3. Cenário 2

O segundo momento planejado pelo time de desenvolvimento foi chamado de "Habilitação dos recursos *offline*". Como o nome sugere, neste o objetivo é validar a habilitação dos recursos *offline*, veja na Tabela 3.

Tabela 3. Momento - Habilitação dos recursos *offline*

Tarefa A2.1. O aluno deve ir para a tela de cursos.
Tarefa A2.2. O aluno deve marcar o curso "Teste *Offline*" como ativo para sincronização, ou seja, acessível sem internet. Observar: Você encontrou facilmente onde acionar a funcionalidade para possibilitar navegação no curso "Teste *Offline*" quando estiver sem conexão à Internet?
Tarefa A2.3. O aluno deverá entrar no curso "Teste *Offline*" e visualizar o Mural. Observar: Você encontrou algo diferente na tela ou ela continua semelhante?
Tarefa A2.4. O aluno deverá navegar até a página de cursos e verificar se há uma informação sobre o curso configurado para ser acessível sem rede (R3). Observar: Você consegue identificar quais cursos estão acessíveis sem conexão à Internet?
Tarefa A2.5. O aluno deverá entrar no curso "Teste *Offline*" novamente.
Tarefa A2.6. O aluno deve escrever uma postagem no mural respondendo a questão "Você encontrou facilmente onde acionar a funcionalidade para possibilitar navegação no curso "Teste *Offline*" quando estiver sem conexão à Internet? E encontrou alguma informação na interface de usuário que indica quais cursos estão configurados para serem acessíveis sem acesso a Internet?"

Como comentado na Seção 5.3, os momentos gerados pela equipe de desenvolvimento também trazem um viés de usabilidade, mas o trabalho deste artigo se dá apenas para testes de funcionalidade. Visto isso, após validação com André Constantino (desenvolvedor do Core dentro do NIED), o cenário foi elaborado conforme exposto na Figura 9.

```
14    Cenário: Habilitação dos recursos offline
15      Quando marcar o curso Teste Offline como ativo para sincronização
16      Então o curso estará habilitado para utilização offline
```

Figura 9. Cenário - Habilitação dos recursos *offline*

5.3.4. Cenário 3

No terceiro momento (Tabela 4) se dá início a dinâmica de validar a funcionalidade sem a presença de conexão com a Internet. Neste momento é testado a interação do usuário com a aplicação fazendo comentários e curtindo publicações do mural do curso sem acesso a Internet. Quando retomada a conexão a aplicação deve sincronizar as alterações feitas *offline* com o servidor.

Este momento foi transcrito para BDD criando o cenário exposto na Figura 10. O cenário também foi validado pelo orientador André Constantino da Silva.

Tabela 4. Momento - Discussão no mural com pontos a interromper a conexão

Tarefa A3.1. O aluno deve acessar o mural do curso “Teste *Offline*”.
Tarefa A3.2. O aluno deve mudar as configurações do seu dispositivo para se desconectar da rede (sugere-se mudar as configurações para o modo avião). Observar: A aplicação Core emitiu alguma informação visual em relação a indisponibilidade de conexão (R2)?.
Tarefa A3.3. O aluno deve ler os comentários publicados, curtindo e comentando os que desejar (sendo obrigatório curtir um comentário e postar um comentário) (R5 e R8). Observar: A aplicação Core destaca, de alguma forma, quais as mensagens que serão enviadas quando a internet for restabelecida (R4)?.
Tarefa A3.4. O aluno deve olhar as atividades a serem realizadas (R5).
Tarefa A3.5. O aluno deve mudar as configurações do seu dispositivo para se conectar à rede. Observar: A aplicação Core emitiu alguma informação visual em relação a disponibilidade de conexão (R4).
Tarefa A3.6. O aluno deve visualizar o Mural.
Tarefa A3.7. O aluno deve observar se as suas curtidas e postagens estão consistentes com as ações que ele realizou (as postagens publicadas estão visíveis; as postagens curtidas continuam marcadas como curtidas; as postagens não curtidas continuam não marcadas como curtidas) (R8).
Tarefa A3.8. O aluno deve observar se novas postagens foram realizadas por outros participantes.
Tarefa A3.9. O aluno deve mudar as configurações do seu dispositivo para se desconectar da rede (sugere-se mudar as configurações para o modo avião). Observar: A aplicação Core emitiu alguma informação visual em relação a indisponibilidade de conexão (R2)?
Tarefa A3.10. O aluno deve escrever uma postagem no mural respondendo as questões: 1. A aplicação Core emitiu alguma informação visual em relação a indisponibilidade de conexão? 2. A aplicação Core emitiu alguma informação visual em relação a disponibilidade de conexão? 3. A aplicação Core destaca, de alguma forma, quais as mensagens que serão enviadas quando a internet for restabelecida? 4. A aplicação Core continua com os dados consistentes em relação às ações tomadas durante o período sem conectividade (as postagens publicadas estão visíveis; as postagens curtidas continuam marcadas como curtidas; as postagens não curtidas continuam não marcadas como curtidas)?

```
18    Cenário: Discussão no mural com pontos a interromper a conexão
19      Dado que o curso Teste Offline esteja habilitado para utilizacao offline
20      Quando o aluno entrar no curso Teste Offline
21      E o aluno perder acesso a rede
22      Então o aluno deve curtir e comentar "Li essa publicação e comentei sem conexao (teste automatizado)"
23      E se conectar novamente a rede
```

Figura 10. Cenário - Discussão no mural com pontos a interromper a conexão

5.3.5. Cenário 4

O quarto momento também se faz interação entre os *status* de *online* e *offline*. Neste momento o teste é para o de conseguir fazer *download* de um arquivo na página "Material de apoio" sem nenhuma largura de banda (momento na Tabela 5).

Tabela 5. Momento - Visualizando Material de Apoio com pontos a interromper a conexão

Tarefa A4.1. O aluno deve acessar o mural do curso "Teste *Offline*".
Tarefa A4.2. O aluno deve mudar as configurações do seu dispositivo para se desconectar da rede (sugere-se mudar as configurações para o modo avião). Observar: A aplicação Core emitiu alguma informação visual em relação a indisponibilidade de conexão (R2)?.
Tarefa A4.3. O aluno deve acessar a ferramenta Material de Apoio.
Tarefa A4.4. O aluno deve visualizar e ler os materiais publicados. Observar: Foi possível visualizar os arquivos em formato PDF e PNG? Não foi possível acessar a página do link?
Tarefa A4.5. O aluno deve escrever uma postagem no mural respondendo as questões: 1. Foi possível acessar os arquivos em formato PDF e PNG disponíveis na ferramenta de Material de Apoio enquanto você estava sem conexão?
Tarefa A4.6. O aluno deve mudar as configurações do seu dispositivo para se conectar à rede. Observar: A aplicação Core emitiu alguma informação visual em relação a disponibilidade de conexão (R4).
Tarefa A4.7. O aluno deve observar se a sua postagem permanece no Mural (R8).

Entendido a necessidade desse teste, o momento foi transcrito e validado para o cenário apresentado na Figura 11 para que fosse validado a funcionalidade de *download* de arquivo.

```
25    Cenário: Visualizando material de apoio com pontos a interromper a conexão
26      Dado que o curso Teste Offline esteja habilitado para utilizacao offline
27      Quando o aluno entrar no curso Teste Offline
28      E o aluno perder acesso a rede
29      E acessar a ferramenta Material de Apoio
30      Então o aluno deve baixar o arquivo ano novo.png sem conexão
31      E se conectar novamente a rede
```

Figura 11. Cenário - Visualizando Material de Apoio com pontos a interromper a conexão

5.3.6. Cenário 5

Assim como o quarto cenário, o quinto cenário vem para fazer o teste de acessar um arquivo dentro da aplicação. Neste caso o arquivo está presente em uma atividade dentro da página de "Atividades".

Tabela 6. Momento - Realizando atividades com interrupção da conexão apenas para leitura da atividade

Tarefa A5.1. O aluno deve mudar as configurações do seu dispositivo para se desconectar da rede (sugere-se mudar as configurações para o modo avião). Observar: A aplicação Core emitiu alguma informação visual em relação a indisponibilidade de conexão (R2)?.
Tarefa A.5.2. O aluno deverá ler a descrição da atividade "Histórias de Natal de Ano Novo". Observar: Foi possível acessar a descrição da atividade?
Tarefa A5.3. O aluno deve visualizar o anexo da atividade "Histórias de Natal e Ano Novo" (R5). Observar: Foi possível visualizar o arquivo em anexo a atividade?
Tarefa A5.4. O aluno deve mudar as configurações do seu dispositivo para se conectar à rede. Observar: A aplicação Core emitiu alguma informação visual em relação a disponibilidade de conexão (R4).
Tarefa A.5.5. O aluno deverá ir à ferramenta Mural.
Tarefa A.5.6. O aluno deverá enviar uma resposta à atividade "Histórias de Natal de Ano Novo".
Tarefa A5.7. O aluno deve escrever uma postagem no mural respondendo as questões: 1. Você conseguiu ler a descrição da atividade e ler o anexo mesmo sem conexão à rede? 2. O conteúdo da resposta enviada por você é o que se encontra visível na atividade após o restabelecimento da conexão?

Esse último momento foi transcrito para o cenário da Figura 12.

```
33      Cenário: Realizando atividades com interrupção da conexão apenas para leitura da atividade
34        Dado que o curso Teste Offline esteja habilitado para utilizacao offline
35        Quando o aluno entrar no curso Teste Offline
36        E o aluno perder acesso a rede
37        E acessar a ferramenta Atividades
38        E o aluno ir para a atividade Histórias de Natal e Ano Novo
39        Então o aluno baixará o anexo para realizar a atividade
40        E se conectar novamente a rede
```

Figura 12. Cenário - Realizando atividades com interrupção da conexão apenas para leitura da atividade

5.4. Implementação dos cenários

Os cenários foram implementados com o *framework* Cucumber e utilizando da linguagem Java e biblioteca Selenium, todo o código para automação dos cenários descritos na Seção 5.3 pode ser lido no Anexo I.

Para identificar qual método executa qual passo em um cenário basta fazer a comparação do passo do cenário com a anotação em cima do método. Por exemplo, considerar a Figura 13 onde na linha 15 do cenário diz "Quando marcar o curso Teste *Offline* como ativo para sincronização". Quando o Cucumber ler esse passo, a ferramenta irá procurar

na classe Java um método com anotação @Quando e que a *String* seja igual ao passo que está sendo executado no momento.

Passo Cenário

```
14    Cenário: Habilitação dos recursos offline
15      Quando marcar o curso Teste Offline como ativo para sincronização
16      Então o curso estará habilitado para utilização offline
```

Método Java

```
72    @Quando("marcar o curso Teste Offline como ativo para sincronização")
73    public void marcar_o_curso_teste_offline_como_ativo_para_sincronização() {
```

Figura 13. Passo x Método

5.4.1. Cenário - Discussão no mural com pontos a interromper a conexão

O cenário " Discussão no mural com pontos a interromper a conexão" é o primeiro cenário de teste com dinâmica de alternar estado de conexão e não conexão da Internet. Os primeiros passos a serem executados serão os passos do contexto mostrado na Figura 7.

A Figura 14 contém o código para os passos do contexto. O primeiro método a ser executado é o da linha 34 que se refere ao passo "Dado que o aluno esteja na página do teleduc: "https://proteo.nied.unicamp.br/prod"". Nele a linha 35 usa o método *get* do *WebDriver* do Selenium para navegar até a página do Core (*link* passado por parâmetro) e então na linha 36 interrompe a execução por 5 segundos para dar tempo do Selenium carregar a página. O método da linha 40 é executado para o segundo passo do contexto onde da linha 41 à 43 são definido variáveis locais para o valor *xpath* dos elementos que terão interação nesse passo (esses valores foram obtidos previamente a partir da ferramenta de inspecionar elementos do navegador). Com esses valores é utilizado o método *findElement* do *WebDriver* que retorna o elemento da tela e nas linhas 44 e 45 são enviados caracteres para o elemento encontrado (caixa de texto de *email* e senha respectivamente). Na linha 46 é chamado o evento de click para o elemento de botão de *login* retornado e na linha 47 novamente é feito interrupção de 5 segundos para ser processado a requisição de *login*.

```
33    @Dado("que o aluno esteja na página do teleduc: {string}")
34    public void que_o_aluno_esteja_na_página_do_teleduc(String url) {
35        webDriver.get(url);
36        waitSeconds(5);
37    }
38
39    @Dado("esteja conectado")
40    public void esteja_conectado() {
41        String emailInput = "//*[@id=\"userEmail\"]";
42        String passwordInput = "//*[@id=\"password\"]";
43        String loginButton = "/html/body/app-root/div/app-login/div/div[2]/form/div[3]/div/button";
44        webDriver.findElement(By.xpath(emailInput)).sendKeys(email);
45        webDriver.findElement(By.xpath(passwordInput)).sendKeys(password);
46        webDriver.findElement(By.xpath(loginButton)).click();
47        waitSeconds(5);
48    }
```

Figura 14. Definição passos contexto

5.4.1.1. Passo “Dado que o curso Teste *Offline* esteja habilitado para utilizacao *offline*”

A Figura 15 demonstra a implementação do primeiro passo para o cenário da Figura 7.

```
91      @Dado("que o curso Teste Offline esteja habilitado para utilizacao offline")
92      public void que_o_curso_teste_offline_esteja_habilitado_para_utilizacao_offline() {
93          String optionsButton = "/html/body/app-root/div/app-course/div/div[1]/div/app-home/div[2]/div/a
94          webDriver.findElement(By.xpath(optionsButton)).click();
95
96          waitSeconds(2);
97
98          String toggleOfflineButton = "/html/body/app-root/div/app-course/div/div[1]/div/app-home/div[2]
99          webDriver.findElement(By.xpath(toggleOfflineButton)).click();
100
101         waitSeconds(15);
102     }
```

Figura 15. Passo “Dado que o curso Teste *Offline* esteja habilitado para utilizacao *offline*”

Neste passo é realizada a mesma dinâmica mostrada na implementação do contexto de procurar um elemento por valor *xpath* e acionar o evento deste elemento. No caso na linha 94 é acionado o click do botão de opções de um curso para que na linha 98 seja acionado o click do botão que habilita o curso para navegação *offline*. Vide Figura 16.

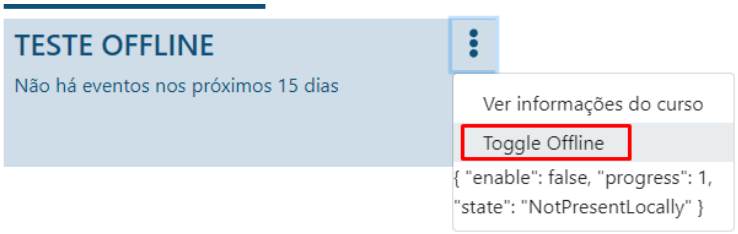


Figura 16. Habilitação recurso *Offline*

5.4.1.2. Passo “Quando o aluno entrar no curso Teste *Offline*”

A implementação do segundo passo para o cenário da Figura 7 pode ser visto na Figura 17. Esse é um passo mais simples e as únicas etapas que são feitas é clicar no botão do curso (conforme linha 53) e esperar 15 segundos (conforme linha 54).

```
50      @Quando("o aluno entrar no curso Teste Offline")
51      public void que_o_aluno_entre_no_curso_teste_offline() {
52          String offlineCourse = "/html/body/app-root/div/app-course/div/di
53          webDriver.findElement(By.xpath(offlineCourse)).click();
54          waitSeconds(15);
55      }
```

Figura 17. Passo “Quando o aluno entrar no curso Teste *Offline*”

5.4.1.3. Passo “E o aluno perder acesso a rede”

A implementação do terceiro passo para o cenário da Figura 7 é demonstrado na Figura 18.

```
105     @Quando("o aluno perder acesso a rede")
106     public void o_aluno_perder_acesso_a_rede() {
107         setToOfflineMode();
108         waitSeconds(10);
109
110     }
```

Figura 18. Passo “E o aluno perder acesso a rede”

No passo é chamado o método privado para setar a execução para o modo *offline* (linha 107) e na linha 108 é esperado 10 segundos. A implementação do método *setToOfflineMode* é visto na Figura 19.

```
192     private void setToOfflineMode() {
193         try {
194             Runtime.getRuntime().exec("nmcli radio wifi off");
195         } catch (IOException e) {
196             e.printStackTrace();
197         }
198     }
```

Figura 19. Passo “E o aluno perder acesso a rede”

Para simular a falta de conexão foi utilizado a funcionalidade do Java de poder a partir do *runtime* fazer uma chamada ao sistema operacional. Como é possível observar na Figura 19 na linha 194 é obtido o *runtime* e executado o comando “nmcli radio wifi off” que no Ubuntu (Sistema Operacional onde o teste automatizado está sendo executado) desliga a conexão wifi. Com isso é possível executar os cenários que visam a ausência de largura de banda no Core.

5.4.1.4. Passo “Então o aluno deve curtir e comentar ‘Li essa publicação e comentei sem conexao (teste automatizado)’ na publicação de testes *offline*”

A implementação do quarto passo para o cenário da Figura 7 é exposta na Figura 20. Neste passo, sem conexão com internet, é navegado até um post de testes *offlines* criado previamente seguindo a dinâmica de procurar elementos por *xpath* e chamar o evento desses elementos. Detalhe especial para a linha 116 onde é utilizado a execução de um *script* para fazer *scroll* da tela até o post.

5.4.1.4. Passo “E se conectar novamente a rede”

A implementação do quinto e último passo para o cenário da Figura 7 é exposta na Figura 21. Esse último passo do cenário segue a lógica do passo de se desconectar mudando apenas o método chamado que é o *setOnlineMode* onde a mudança está apenas no comando executado, como pode ser visto na Figura 22.

Devido a limitações de espaço, não foi detalhado a implementação dos demais cenários de testes, que seguem lógica semelhante ao apresentado.

```
112     @Então("o aluno deve curtir e comentar {string} na publicação de testes offline")
113     public void curtir_e_comentar_em_alguma_publicação(String message) {
114         String offlinePost = "/html/body/app-root/div/app-course/div/div[1]/div/app-wall/div/div[3]";
115         WebElement offlinePostElement = webDriver.findElement(By.xpath(offlinePost));
116         ((JavascriptExecutor) webDriver).executeScript("arguments[0].scrollIntoView(true);", offlinePostElement);
117         waitSeconds(2);
118 
119         String likeButton = "/html/body/app-root/div/app-course/div/div[1]/div/app-wall/div/div[3]/app-wall-item/d
120         webDriver.findElement(By.xpath(likeButton)).click();
121 
122         waitSeconds(2);
123 
124         String textArea = "/html/body/app-root/div/app-course/div/div[1]/div/app-wall/div/div[3]/app-wall-item/div
125         webDriver.findElement(By.xpath(textArea)).sendKeys(message);
126 
127         waitSeconds(2);
128 
129         String commentButton = "/html/body/app-root/div/app-course/div/div[1]/div/app-wall/div/div[3]/app-wall-ite
130         webDriver.findElement(By.xpath(commentButton)).click();
131 
132         waitSeconds(5);
133     }
```

Figura 20. Passo “Então o aluno deve curtir e comentar "Li essa publicação e comentei sem conexao (teste automatizado)" na publicação de testes *offline*”

```
135     @Entao("se conectar novamente a rede")
136     public void se_conectar_novamente_a_rede() {
137         setToOnlineMode();
138         waitSeconds(10);
139     }
```

Figura 21. Passo “E se conectar novamente a rede”

```
200     private void setToOnlineMode() {
201         try {
202             Runtime.getRuntime().exec("nmcli radio wifi on");
203         } catch (IOException e) {
204             e.printStackTrace();
205         }
206     }
207 
208 }
```

Figura 22. Passo “E se conectar novamente a rede”

6. Resultados

Com os cenários validados e implementados, foi realizada a execução de toda a automação. A execução durou em torno de 5 minutos e todos os cenários foram executados com sucesso. Como descrito no início do artigo, o Cucumber disponibiliza a opção de gerar um relatório documentando a execução dos cenários em um arquivo. O relatório está disponível no Anexo I (no Anexo II é possível observar um exemplo de como seria um relatório gerado com erro de alguns passos). Observa-se que todos os cenários foram executados com sucesso.

7. Conclusão

Antes da realização deste trabalho os desenvolvedores do Core empregavam esforços para realização de testes manuais que acabavam consumindo um tempo que poderia estar sendo utilizado para o desenvolvimento de outras funcionalidades. Motivado em minimizar esse problema, o presente trabalho trouxe um conjunto de cenários de testes automatizados com uso das tecnologias Cucumber e Selenium.

Os testes desenvolvidos fazem a cobertura da funcionalidade *Offline* do sistema Core, para serem utilizados basta que, com o código fonte em uma máquina Linux, seja executado o comando do Maven "mvn test" e, assim, todos os cenários serão executados e validados de acordo com a execução. Outra forma de realizar a etapa de execução pode ser também com a criação de uma pipeline no jenkins, podendo inclusive adotar o método de integração contínua, com a execução dos testes sendo realizada cada vez que um novo código sobe para o repositório.

Atualmente a automação é limitada a plataforma *web* rodando em cima de um sistema operacional Linux, isso abre abertura para trabalhos futuros podendo ser realizada a automatização desses cenários também para outras plataformas, como Android, por exemplo.

Por fim, durante a execução deste trabalho foi possível aprender muito acerca da prática de testes de *software*, adquirindo conhecimento em técnicas e ferramentas da área. Além disso, ao longo do desenvolvimento do projeto foram articuladas as disciplinas de Engenharia de *Software*, Qualidade de *Software* e Linguagem de Programação que compõem a grade curricular do curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia de São Paulo, Câmpus Hortolândia.

7. Referências

ANDERLE, A. Introdução de BDD (Behavior Driven Development) como melhoria de processo no desenvolvimento ágil de *software*. 2015. 44 f. Tese (Doutorado) - Curso de Qualidade de *Software*, São Leopoldo, São Leopoldo, 2015.

ARANTES, F. L.; FREIRE, F. M. P. Projetando o TelEduc Core: Integração e Interação. In: SÁNCHEZ, J. (Ed). Nuevas Ideas en Informática Educativa, vol. 14, pp. 429- 434. 2018.

CURTI, F. M.; DALLILO, F. D. Automação de Testes Utilizando a Ferramenta Cucumber. RECIMA21 - Revista Científica Multidisciplinar - ISSN 2675-6218, [S. l.], v. 3, n. 2, p. 321133, 2022. DOI: 10.47820/recima21.v3i2.1133. Disponível em: <https://www.recima21.com.br/index.php/recima21/article/view/1133>. Acesso em: 30 jun. 2022.

DE MOURA, J. L.; CHARÃO, A. S. Automação de Testes em Aplicações de BPMS: um Relato de Experiência. In: Simpósio Brasileiro de Qualidade de *Software* (SBQS), 14. , 2015, Manaus. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2015. p. 212-219. DOI: https://doi.org/10.5753/sbqs.2015.15225.

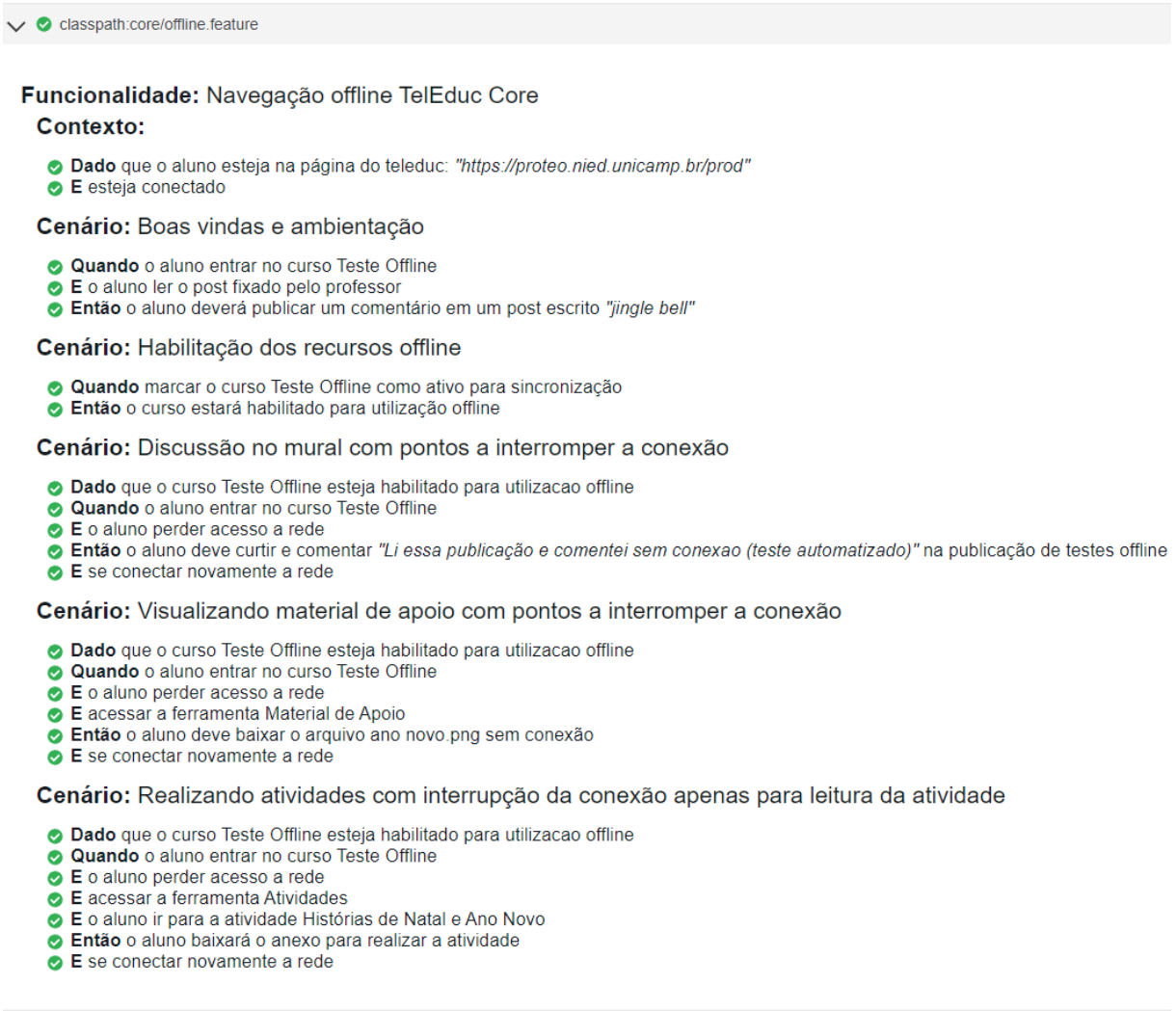
FELIX, R. Teste de *Software*. São Paulo: Pearson, 2016.

GALLOTI, G. Qualidade de *Software*. São Paulo: Pearson, 2015.

Cucumber.io (2019). Introduction. Disponível em: <https://cucumber.io/docs/guides/overview/>. Acesso em: 07 de jul. 2022.

Anexo I

Relatório de execução do Cucumber dos cenários de testes automatizados



classpath:core/offline.feature

Funcionalidade: Navegação offline TelEduc Core

Contexto:

- **Dado** que o aluno esteja na página do teleduc: *"https://proteo.nied.unicamp.br/prod"*
- **E** esteja conectado

Cenário: Boas vindas e ambientação

- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno ler o post fixado pelo professor
- **Então** o aluno deverá publicar um comentário em um post escrito *"jingle bell"*

Cenário: Habilitação dos recursos offline

- **Quando** marcar o curso Teste Offline como ativo para sincronização
- **Então** o curso estará habilitado para utilização offline

Cenário: Discussão no mural com pontos a interromper a conexão

- **Dado** que o curso Teste Offline esteja habilitado para utilizacao offline
- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno perder acesso a rede
- **Então** o aluno deve curtir e comentar *"Li essa publicação e comentei sem conexao (teste automatizado)"* na publicação de testes offline
- **E** se conectar novamente a rede

Cenário: Visualizando material de apoio com pontos a interromper a conexão

- **Dado** que o curso Teste Offline esteja habilitado para utilizacao offline
- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno perder acesso a rede
- **E** acessar a ferramenta Material de Apoio
- **Então** o aluno deve baixar o arquivo ano novo.png sem conexão
- **E** se conectar novamente a rede

Cenário: Realizando atividades com interrupção da conexão apenas para leitura da atividade

- **Dado** que o curso Teste Offline esteja habilitado para utilizacao offline
- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno perder acesso a rede
- **E** acessar a ferramenta Atividades
- **E** o aluno ir para a atividade Histórias de Natal e Ano Novo
- **Então** o aluno baixará o anexo para realizar a atividade
- **E** se conectar novamente a rede

Anexo II

Exemplo relatório de execução do Cucumber com erro de alguns passos

classpath:core/offline.feature

Funcionalidade: Navegação offline TelEduc Core

Contexto:

- **Dado** que o aluno esteja na página do teleduc: *"https://proteo.nied.unicamp.br/prod"*
- **E** esteja conectado

Cenário: Boas vindas e ambientação

- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno ler o post fixado pelo professor
- **Então** o aluno deverá publicar um comentário em um post escrito *"jingle bell"*

Cenário: Habilitação dos recursos offline

- **Quando** marcar o curso Teste Offline como ativo para sincronização
- **Então** o curso estará habilitado para utilização offline

Cenário: Discussão no mural com pontos a interromper a conexão

- **Dado** que o curso Teste Offline esteja habilitado para utilizacao offline

```
java.lang.RuntimeException:
        at core.steps.offline.OfflineStepDefinitions.que_o_curso_teste_offline_esteja_habilitado_para_utilizacao_offline(OfflineStepDefinitions.java:103)
        at ✽.que o curso Teste Offline esteja habilitado para utilizacao offline(classpath:core/offline.feature:19)
```

- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno perder acesso a rede
- **Então** o aluno deve curtir e comentar *"Li essa publicação e comentei sem conexao (teste automatizado)"* na publicação de testes offline
- **E** se conectar novamente a rede

Cenário: Visualizando material de apoio com pontos a interromper a conexão

- **Dado** que o curso Teste Offline esteja habilitado para utilizacao offline

```
java.lang.RuntimeException:
        at core.steps.offline.OfflineStepDefinitions.que_o_curso_teste_offline_esteja_habilitado_para_utilizacao_offline(OfflineStepDefinitions.java:103)
        at ✽.que o curso Teste Offline esteja habilitado para utilizacao offline(classpath:core/offline.feature:26)
```

- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno perder acesso a rede
- **E** acessar a ferramenta Material de Apoio
- **Então** o aluno deve baixar o arquivo ano novo.png sem conexão
- **E** se conectar novamente a rede

Cenário: Realizando atividades com interrupção da conexão apenas para leitura da atividade

- **Dado** que o curso Teste Offline esteja habilitado para utilizacao offline

```
java.lang.RuntimeException:
        at core.steps.offline.OfflineStepDefinitions.que_o_curso_teste_offline_esteja_habilitado_para_utilizacao_offline(OfflineStepDefinitions.java:103)
        at ✽.que o curso Teste Offline esteja habilitado para utilizacao offline(classpath:core/offline.feature:34)
```

- **Quando** o aluno entrar no curso Teste Offline
- **E** o aluno perder acesso a rede
- **E** acessar a ferramenta Atividades
- **E** o aluno ir para a atividade Histórias de Natal e Ano Novo
- **Então** o aluno baixará o anexo para realizar a atividade
- **E** se conectar novamente a rede

Anexo III

Telas do Ambiente Core

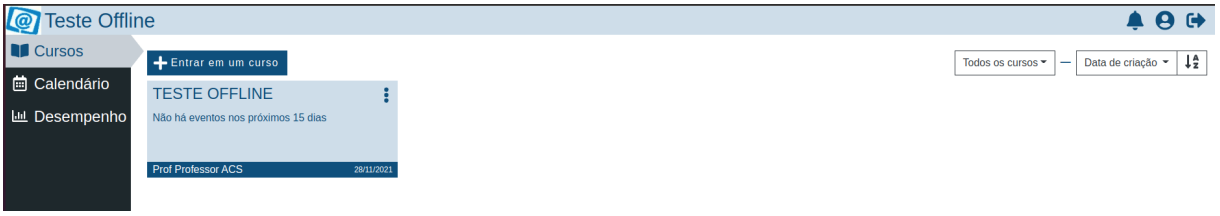


Figura 1. Tela de Cursos, apresentada assim que o usuário loga no ambiente

Figura 2. Tela de cursos visualizada por um aluno exibindo as opções do menu para um curso.

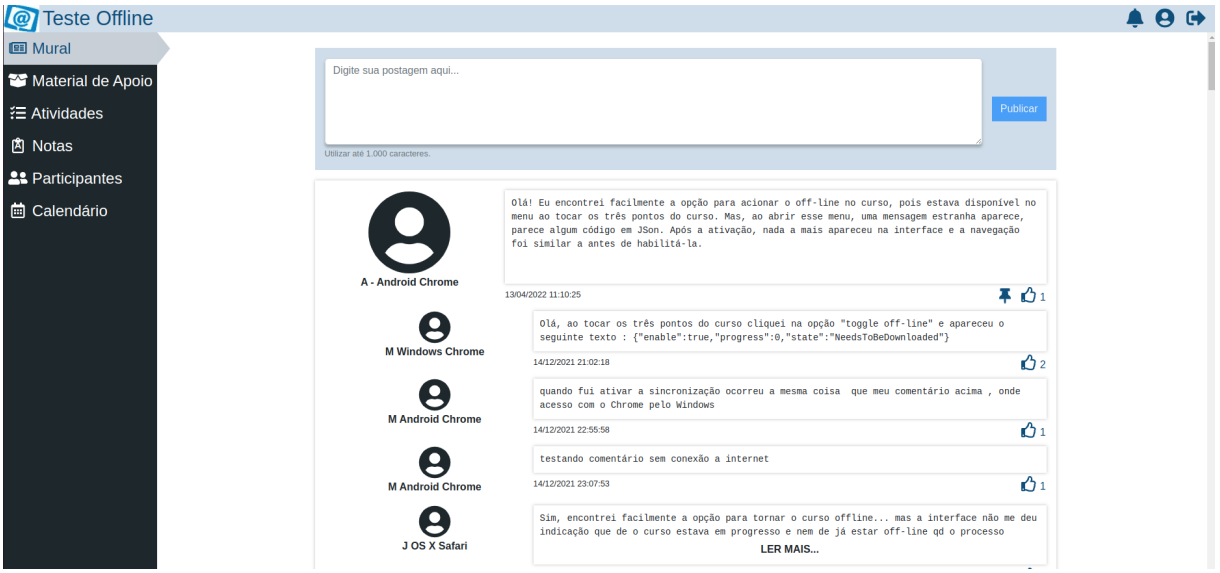


Figura 3. Tela da ferramenta Mural visualizada por um aluno com conexão a Internet

Figura 4. Tela da Ferramenta Materiais de Apoio visualizada por um aluno conectado a internet

Figura 5. Tela da ferramenta Atividades de um curso visualizada por um aluno conectado a internet.

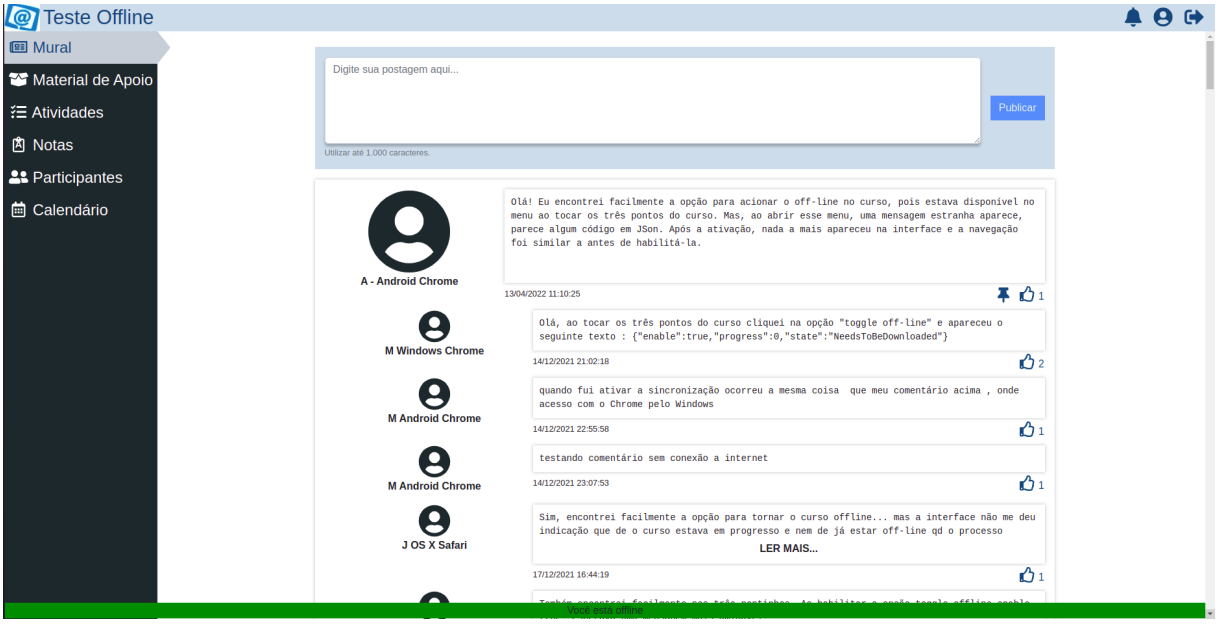


Figura 6. Tela do Mural de um curso com informação no rodapé dizendo que o usuário está sem conexão com internet.

Documento Digitalizado Público

Anexo I - artigo - TCC

Assunto: Anexo I - artigo - TCC
Assinado por: Andre Constantino
Tipo do Documento: Relatório
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Documento Digital

Documento assinado eletronicamente por:

- **Andre Constantino da Silva**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 24/07/2022 11:38:23.

Este documento foi armazenado no SUAP em 24/07/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1045732
Código de Autenticação: 3f484a38d5