

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

GUILHERME SOUZA FERNANDES

**AUTOMAÇÃO DE TESTES COM SELENIUM WEBDRIVER APLICADA NO
SISTEMA DE DISTRIBUIÇÃO DE DISCIPLINAS**

UBERLÂNDIA – 2020

GUILHERME SOUZA FERNANDES

**AUTOMAÇÃO DE TESTES COM SELENIUM WEBDRIVER APLICADA NO
SISTEMA DE DISTRIBUIÇÃO DE DISCIPLINAS**

Monografia apresentada ao Curso de Graduação em Gestão da Informação, da Universidade Federal de Uberlândia, como exigência parcial para a obtenção do título de Bacharel em Gestão da Informação.

Orientador: Prof. Dr. Bruno Augusto Nassif Travençolo

UBERLÂNDIA 2020

RESUMO

A fase de testes de um sistema é a fase final do seu desenvolvimento, pois é nessa fase que são encontrados os problemas que poderão ocorrer em produção, pois são feitas várias simulações de situações que o usuário faria no sistema. É considerada sucesso uma simulação que alcançou o resultado esperado, e falha caso o resultado ou funcionalidade do sistema não tenha apresentado o resultado esperado. Apesar dessas práticas demandarem tempo, conhecimento, planejamento, infraestrutura e pessoal especializado, devido à sua importância, as empresas vêm adotando cada vez mais a prática de testes no desenvolvimento de sistemas. Este trabalho tem a finalidade de apresentar o que é um teste de software e a sua importância no desenvolvimento dos sistemas. São explicadas como funciona uma automação de testes na prática e como é feita uma automação utilizando Eclipse, Cucumber, linguagem Java e o Selenium WebDriver. Para aplicação da automação foi utilizado o sistema de distribuição de disciplinas utilizado pela Faculdade de Computação da UFU, o qual designa para cada professor, qual disciplina por ele será ministrada, o ano letivo e a turma para qual o mesmo irá ministrar as aulas.

Palavras-chave: Automação, Eclipse, Teste, Qualidade, Cucumber, WebDriver.

Sumário

1. INTRODUÇÃO	6
1.1. OBJETIVOS	6
1.2. ORGANIZAÇÃO DO TRABALHO	6
2. REFERENCIAL TEÓRICO	8
2.1. DIMENSÕES DE QUALIDADE	8
2.2. CICLO DE VIDA DOS TESTES	8
2.3. CUSTO DA QUALIDADE DE SOFTWARE	9
2.4. TIPOS DE TESTE DE SOFTWARE	9
3. MATERIAIS E MÉTODOS	11
3.1. SOFTWARE ANALISADO: SODD	11
3.2. FERRAMENTAS	11
3.3. FERRAMENTAS DE AUTOMAÇÃO	11
JUnit	12
Cucumber	12
Selenium WebDriver	12
3.4. AMBIENTE DE AUTOMAÇÃO	12
4. RESULTADOS	18
5. CONCLUSÃO	30
6. REFERÊNCIAS.....	31

Lista de Figuras

Figura 1 - Gráfico que representa o custo de um problema encontrado em diferentes etapas do desenvolvimento de um software	9
Figura 2 - Tela de login do sistema	13
Figura 3 - Arquivo Feature utilizado para descrever os casos testes.....	13
Figura 4 – Arquivo Page Objects utilizado para mapear os elementos de uma tela.	14
Figura 5 - Arquivo Step utilizando para definir o passo a passo do cenário de teste que será automatizado.	15
Figura 6 - Arquivo Step utilizando para definir o passo a passo do cenário de teste que será automatizado.	16
Figura 7 - Arquivo Json utilizado para armazenar valores em variáveis que serão utilizadas em vários lugares, facilitando a manutenção.	17
Figura 8 – Tela obtida após um login Válido	18
Figura 9 – Tela resultante de um login Inválido. Indicando o erro obtido.	18
Figura 10 - Professor com turma – são listadas as turmas do professor consultado.....	19
Figura 11 - Professor sem turma (realização da busca)	19
Figura 12 - Professor sem turma (resultado da busca).....	20
Figura 13 - Fila de professores por turma.....	20
Figura 14 - Fila por professor (sem as turmas). Consulta com retorno.	21
Figura 15 - Fila por professor (sem as turmas), realização da busca.	21
Figura 16 - Fila por professor (sem as turmas), resultado da busca	22
Figura 17 - Tela de pesquisa, edição, exclusão e botão de acesso para cadastrar um curso	23
Figura 18 - Tela de cadastro de curso	23
Figura 19 - Tela de pesquisa de curso por nome completo	24
Figura 20 - Tela de pesquisa de curso por nome incompleto	24
Figura 21 - Acesso a tela de atribuição de turmas aos professores.	24
Figura 22 - Tela atribuição de turmas aos professores.....	25
Figura 23 - Tela de pesquisa para atribuição de turmas aos professores.....	25
Figura 24 - Tela para realizar atribuição de turmas aos professores	26
Figura 25 - Tela de pesquisa, edição, exclusão e botão de acesso à tela para cadastrar professor	27
Figura 26 - Modal de cadastro de professor	27
Figura 27 - Excluir registro do sistema.....	28
Figura 28 - Confirmar exclusão de registro	28
Figura 29 - Mensagem de nenhum registro encontrado.....	29
Figura 30 - Comando "AssertEquals" utilizado para comparação de variáveis	29

1. INTRODUÇÃO

A prática de se realizar testes em sistemas vem sendo adotada cada vez mais nas empresas de tecnologia, uma vez que, o uso deste minimiza os erros encontrados dentro do sistema, além de minimizar também as chances de ocorrer falhas quando o usuário final estiver utilizando o programa.

Para se realizar os testes em um sistema, é feito todo um planejamento para que o sistema possa ser validado. Ao encontrar qualquer tipo de erro nessa fase, ele é passado imediatamente para o desenvolvedor, para que este realize a correção e retorne para o analista de teste poder então validar novamente aquela funcionalidade. De acordo com (OLIVEIRA, 2010) essa fase de testes é de extrema importância uma vez que gera economia para a empresa, pois gera menos retrabalho, reduzindo assim os gastos.

Quando se fala em teste de software, muito se ouve sobre testes automatizados. Mas afinal, o que é e quais as vantagens de se utilizar um teste automatizado quando comparado com um teste manual? De acordo com Bernardo e Kon (2008: p.55), os testes automatizados são programas ou scripts simples que exercitam funcionalidades do sistema testado e fazem verificações automáticas nos efeitos colaterais obtidos. A grande vantagem dessa abordagem, é que todos os casos de teste podem ser facilmente e rapidamente repetidos a qualquer momento e com pouco esforço.

Logo, podemos inferir que um teste automatizado é a utilização de um software para gerenciar a execução de um teste pré-programado pelo testador. Dentre as vantagens existentes ao se utilizar o teste automatizado, a principal que se pode destacar é o tempo ganho para execução dos testes. Testes que poderiam demorar horas ou dias para serem feitos, passam a ser executados em minutos ou horas, o que representa um ganho muito significativo de tempo.

Há diferentes formas de se fazer um teste automatizado, variando de testes apenas do *backend*, que nesse caso não leva em consideração a interface do sistema, e os testes automatizados que leva em consideração não só o *backend* como o *frontend* do sistema.

1.1. OBJETIVOS

Este trabalho tem a finalidade de apresentar alguns dos conceitos utilizados em testes de softwares como também mostrar a aplicação de um teste automatizado em um sistema web.

1.2. ORGANIZAÇÃO DO TRABALHO

No início do trabalho é apresentado uma breve introdução sobre conceitos de testes, testes automatizados e os benefícios de se aplicar a prática de testes no ciclo de vida de um software. A introdução é seguida por um referencial teórico, que descreve a base teórica utilizada no desenvolvimento do trabalho. Em seguida, no capítulo de materiais e métodos são apresentados os softwares que foram utilizados para desenvolver a automação, como também os softwares utilizados para a criação de um ambiente de testes. Além disso, é apresentado a estrutura de um projeto de automação criado no

Eclipse, utilizando Selenium WebDriver e a linguagem Java. O penúltimo capítulo do trabalho é o Resultado. Nesse capítulo são exibidas as automações feitas nas principais funcionalidades do sistema e como elas foram feitas, seguidas de ilustrações para uma melhor compreensão. E por fim, a Conclusão, que baseado nos temas apresentados no início do trabalho, juntamente com a aplicação da automação no sistema nos dá uma conclusão sobre o trabalho feito, baseado nesses dados.

2. REFERENCIAL TEÓRICO

Neste capítulo serão descritos os principais conceitos relacionados aos testes de software. São apresentados alguns métodos utilizados em um processo de testes, ferramentas utilizadas na automação utilizando Selenium WebDriver, tipos de testes que podem ser feitos em um determinado sistema e também as vantagens de se implantar em uma empresa a prática de testes de software.

2.1. DIMENSÕES DE QUALIDADE

Existem três tipos de dimensões de qualidade que se devem ser abordadas nos testes, a Confiança, Funcionalidade e Performance (CUNHA, 2010).

- os testes de confiança têm o intuito de garantir que em nenhum fluxo do sistema, o mesmo irá entrar em um *loop* ou ter seu fluxo interrompido por algum tipo de falha, o que acaba atrapalhando a experiência do usuário.

- os testes de funcionalidade garantem que o sistema está funcionando de acordo com o que foi definido nos requisitos do sistema, sendo esses requisitos funcionais ou não.

- os testes de performance visam garantir que o sistema desenvolvido tenha um tempo de resposta que é aceitável, mesmo que o número de acessos simultâneos ao sistema seja alto.

2.2. CICLO DE VIDA DOS TESTES

Segundo Rios (2007), o ciclo de vida de testes é composto pelas seguintes etapas: (i) Planejamento, que é a etapa em que se estabelece o que vai ser testado, em quanto tempo e em que momento os testes serão interrompidos; (ii) Preparação, em que o objetivo é preparar toda a estrutura do ambiente de testes como equipamentos, configuração de hardware e softwares usados (sistemas operacionais, navegadores, etc.), criação da massa de dados de teste, pessoal, ferramentas de automação, entre outros; (iii) Especificação, cuja atividade principal é elaborar e revisar os cenários e roteiros de testes. Na Execução são feitos os testes planejados e registra-se os resultados obtidos; (iv) por fim, na Entrega, é arquivada toda a documentação e são descritas todas as ocorrências do projeto relevantes para a melhoria do processo.

Segundo (OLIVEIRA, 2010), para se obter resultados positivos nos projetos de testes é necessário que ele se inicie desde a especificação dos requisitos do sistema a ser implementado, ou seja, tão logo comece o projeto de desenvolvimento do software inicia-se também em conjunto o projeto de testes de software.

2.3. CUSTO DA QUALIDADE DE SOFTWARE

A falta de testes de softwares pode aumentar os custos de um projeto, uma vez que o desenvolvedor terá mais trabalho, pois além de criar o software ele precisa testar e corrigir os erros encontrados, e isso causará uma sobrecarga de trabalho, aumentando o risco de atrasos no projeto e falhas no produto após ser entregue ao cliente.

Todos esses problemas podem ser resolvidos de forma eficiente e com redução de custos, utilizando os testes como forma de validar o sistema e encontrando possíveis problemas antes do mesmo ir para produção. De acordo com o *Systems Sciences Institute* da IBM (DAWSON, Maurice et al. 2010), o custo de correção de um bug em produção pode chegar a 100 vezes mais quando se comparado com o custo do mesmo bug encontrado na fase de requisitos, ou seja, quando se encontra um problema logo no início do desenvolvimento, o custo para sua correção é exponencialmente menor do que quando o mesmo problema é encontrado quando o software já está em produção, como pode ser observado (Figura 1).

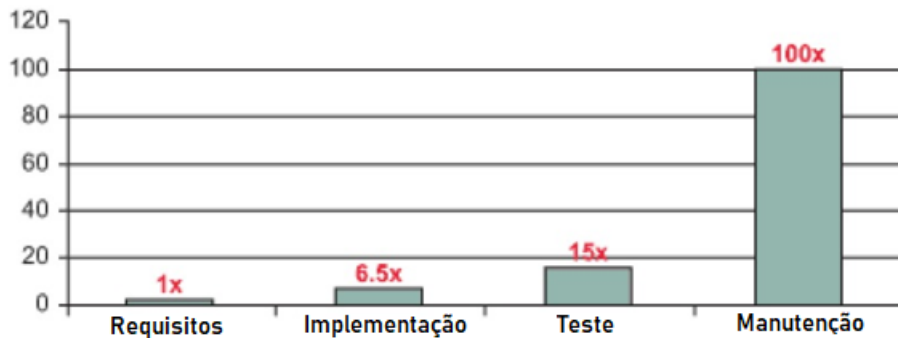


Figura 1 - Gráfico que representa o custo de um problema encontrado em diferentes etapas do desenvolvimento de um software

Figura adaptada de (DAWSON, Maurice; BURREL, Darrell; RAHIM, Emad; BEWSTER, Stephen, 2010).

2.4. TIPOS DE TESTE DE SOFTWARE

Existem diversos tipos de testes de softwares e durante o planejamento de testes de um sistema deve-se analisar quais os tipos que serão executados naquele determinado projeto, levando em consideração as características do sistema, quantidade de usuário que irão utilizá-lo, entre outras, para assim averiguar quais serão mais eficientes.

A seguir são mostrados os tipos de testes mais utilizados e a abordagem que cada um realiza (PERES, 2009):

- um teste de unidade valida a classe ou componente do sistema de forma isolada, já o teste integrado valida todos os componentes de uma tela de forma combinada, com o intuito de certificar que esses componentes correspondem de forma satisfatória e esperada. Em poucas palavras o teste integrado é a junção de vários testes de unidade.

- os testes de configuração certificam de que o software funciona corretamente no hardware que foi instalado.

- os testes de integridade validam a resistência do software á falhas, certificando de que ele está funcionando de forma correta.

- o teste de instalação tem como intuito validar o processo de instalação do sistema levando em consideração diferentes hardwares e sistemas operacionais levando também em consideração interrupções de rede e interrupções na instalação.

- o teste de performance é dividido em dois: teste de carga e teste de stress. O teste de carga certifica de que o sistema suporta usuário simultâneos e que o tempo de resposta dele continuará aceitável, em condições normais. Já o teste de stress valida o comportamento do sistema com um número grande de usuários e seu tempo de resposta, porem em uma situação extrema, certificando assim a sua estabilidade em determinadas situações.

- testes de usabilidade foca no layout, no acesso as funcionalidades e na interface dos sistemas;

- os testes de Regressão, são retestes utilizados para certificar que objetos que foram modificados recentemente não estejam causando impactos no funcionamento do sistema.

- os testes funcionais podem ser realizados manualmente ou automaticamente. O método manual é realizado sem o auxílio de ferramentas, apenas com as informações inseridas pelo testador. Já os testes automáticos são executados com o auxílio de uma ferramenta que irá simular as ações realizadas pelos usuários de forma automática.

3. MATERIAIS E MÉTODOS

Neste capítulo são apresentadas as ferramentas utilizadas para a configuração do ambiente para realizar a automação, bem como as ferramentas que foram utilizadas para o desenvolvimento da automação de testes em questão.

Os principais materiais utilizados nessa seção foram o Eclipse, Cucumber, TomCat, PgAdmin, WebDriver e JUnit.

3.1. SOFTWARE ANALISADO: SODD

O SODD – Sistema Online de Distribuição de Disciplinas – é um sistema web desenvolvido por alunos do curso de Sistemas de Informação da FACOM e o mesmo tem a finalidade de possibilitar o cadastro de novos professores, turmas, disciplinas como também de manter um registro das distribuições semestrais das disciplinas ofertadas na UFU durante cada semestre. O sistema foi desenvolvido utilizando a linguagem Java e para o banco de dados foi utilizado o SQL (OLIVEIRA, A. 2017).

O sistema SODD possui os módulos Administrativos e Usuário, e ambos foram utilizados no trabalho em questão. O módulo usuário permite o cadastro e visualização de filas de professores por turmas, filas de disciplinas para um determinado professor, entre outros. Já o módulo Administrador permite que sejam cadastrados novos professores, novas disciplinas e também que sejam atribuídas determinadas turmas ao professor desejado.

3.2. FERRAMENTAS

No trabalho realizado foi levado em consideração tanto o *frontend* quando o *backend* do sistema testado. Para a realização dos testes foi necessário a criação de um ambiente local do sistema e dos softwares necessários para a execução dos testes automatizados. O ambiente local foi criado utilizando o Eclipse, sendo necessário o projeto do sistema importado no mesmo, o TomCat 7.0 como servidor web e o PgAdmin para conexão do banco de dados com o sistema. Após a importação do sistema no Eclipse, foi necessário fazer a conexão do sistema com o banco de dados. A conexão é feita no próprio projeto do sistema que está no PgAdmin, por meio de um arquivo já inserido no projeto, em que é informado o usuário, senha e o caminho do banco a ser conectado. Após configurar a conexão basta vincular o sistema no servidor web criado no Eclipse, inicializar o servidor e acessar o sistema pelo navegador, utilizando o endereço *http://localhost:8080*.

3.3. FERRAMENTAS DE AUTOMAÇÃO

Com o ambiente local criado e funcionando, foi dado início à automação de testes do sistema. Para desenvolvimento e execução dos testes automatizados os softwares utilizados foram o Eclipse para o desenvolvimento do código, o Cucumber que é uma ferramenta que permite o desenvolvimento orientado por comportamentos e para

execução do código anteriormente informado foi utilizado o Selenium WebDriver, que é uma ferramenta responsável por automatizar todas as entradas de dados no navegador. Também foi utilizado o JUnit que é um framework de teste voltado para testes de ponta a ponta, testes unitários e também testes de integração utilizando a linguagem Java. Nas próximas seções serão descritas essas ferramentas

JUnit

JUnit é um framework de código aberto que possibilita o desenvolvimento de testes automatizados utilizando a linguagem Java. O JUnit possibilita a execução de teste unitários, ou seja, cada componente do sistema é testado de maneira isolada, permitindo assim uma grande cobertura de testes. O JUnit traz consigo uma série de vantagens, sendo algumas delas:

- Criação rápida de código de testes;
- Verificação dos resultados dos testes e indicação de teste aprovado ou não;
- É um framework gratuito;
- Após desenvolvido os testes os mesmos podem ser executados sem que haja necessidade de interromper o desenvolvimento do sistema.

Cucumber

O Cucumber é uma ferramenta utilizada para executar testes de aceitação com o método BDD (*Behavior-Driven Development* - Desenvolvimento Orientado por Comportamento). Uma das principais vantagens que se pode destacar do Cucumber é sua linguagem utilizada, que é uma linguagem natural, podendo o código ser escrito em Português, Inglês, etc. O que o torna também de fácil compreensão e escrita até mesmo por algum outro membro do projeto.

Selenium WebDriver

O Cucumber não consegue interagir diretamente com a aplicação, com isso foi necessário utilizar o Selenium WebDriver que é uma ferramenta utilizada para automatizar as entradas de dados no navegador, permitindo também executar ações como cliques, seleções, escritas, etc.

3.4. AMBIENTE DE AUTOMAÇÃO

Unindo as ferramentas citadas no capítulo anterior é possível criar um ambiente completo e funcional de testes automatizados.

A primeira tela apresentada ao acessar o link do sistema, é a tela de *login* (Figura 2). Com isso, essa foi a primeira tela em que a automação foi realizada. Em uma automação buscamos sempre testar os possíveis cenários existentes de testes para aquela tela ou componente. Tomando como exemplo a tela de *login*, temos apenas o campo SIAPE. Em um caso como esse, tomamos como cenários de teste, os seguintes cenários: Login válido (informando um SIAPE existente no banco de dados) e Login inválido (informando um SIAPE inexistente no banco de dados)

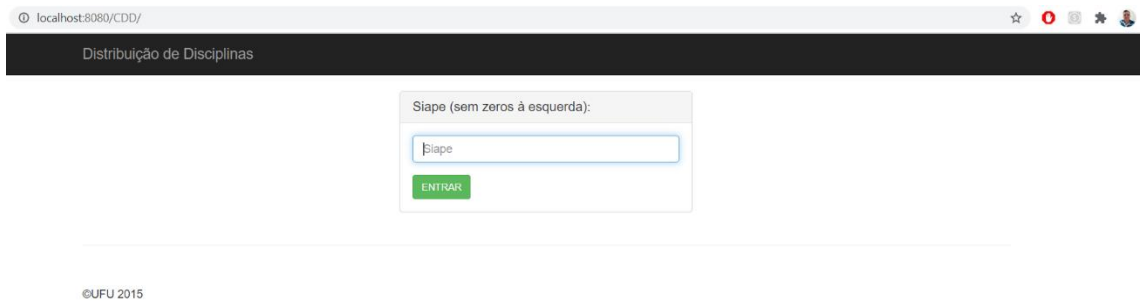


Figura 2 - Tela de login do sistema.

Para fazer as validações dos cenários informados devemos observar o comportamento da página ao realizar o *login*. No sistema de distribuição ao informar um SIAPE válido, o sistema direciona o usuário para a página inicial do sistema. Como validação, optou-se por validar algum componente que só existe naquela página em específico, garantindo assim que após informar um SIAPE existente no banco de dados, o sistema direcionou o usuário para a página correta. Em caso de falha de login, o sistema apresenta um *pop-up* informando que o SIAPE é inválido. Porém, por ser um *pop-up* a automação não consegue identificar esse elemento na tela, tornando assim impossível de utilizá-lo na validação. Como medida alternativa, validamos que o campo SIAPE ainda permanecia na tela, mesmo após a tentativa de *login*, conseguindo assim realizar a validação de que o login não foi realizado com sucesso.

O desenvolvimento de uma automação é baseado em cada caso de teste criado pelo testador. Cada caso de teste será um código único a ser desenvolvido.

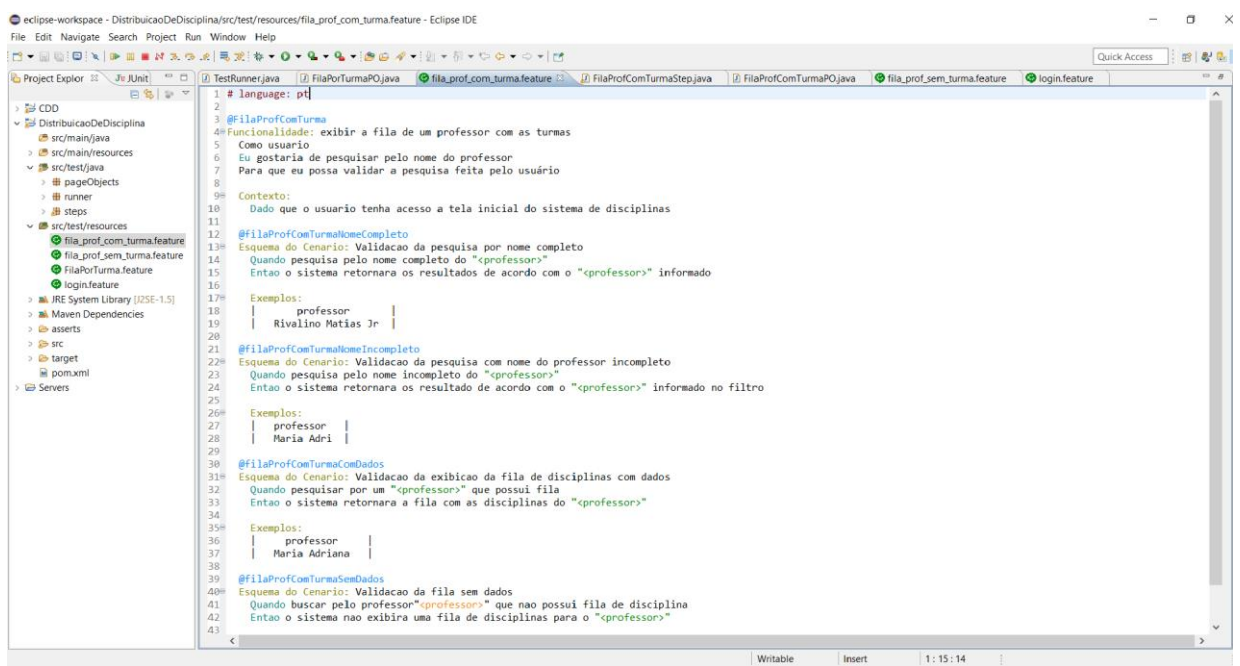
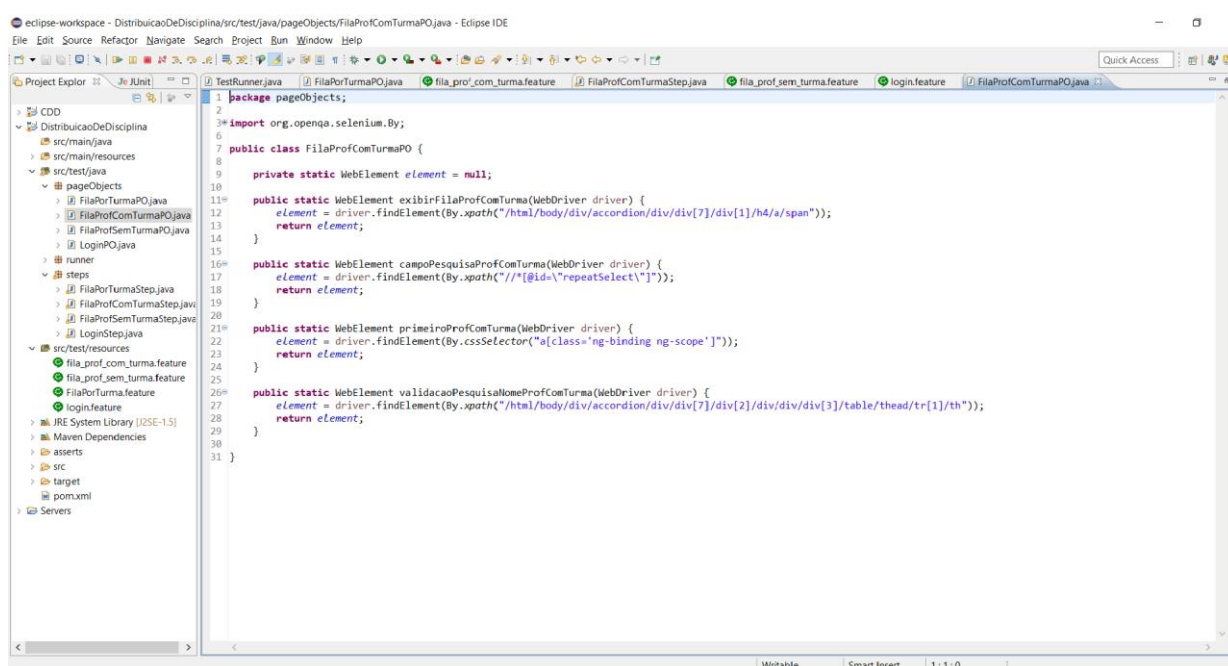


Figura 3 - Arquivo Feature utilizado para descrever os casos testes.

A Figura 3 representa um arquivo *feature* do projeto de automação, em que são especificados todos os cenários de testes e os dados que serão passados em um determinada variável.

Nesse exemplo podemos ver que para a funcionalidade Pesquisa de fila de professores com turma, foram feitos 4 tipos de cenários no teste, validando assim pesquisa com nome completo, incompleto, professor com dados e sem dados.

No arquivo *feature* são definidas as variáveis que serão utilizadas nesse teste e o valor que cada uma irá passar, como podemos ver na Figura 2, esses valores são definidos na tabela “Exemplos”.



```
1 package pageObjects;
2
3 import org.openqa.selenium.By;
4
5 public class FilaProfComTurmaPO {
6
7     private static WebElement element = null;
8
9     public static WebElement exibirFilaProfComTurma(WebDriver driver) {
10         element = driver.findElement(By.xpath("/html/body/div/accordion/div/div[7]/div[1]/h4/a/span"));
11         return element;
12     }
13
14     public static WebElement campoPesquisaProfComTurma(WebDriver driver) {
15         element = driver.findElement(By.xpath("//*[@id='repeatSelect']"));
16         return element;
17     }
18
19     public static WebElement primeiroProfComTurma(WebDriver driver) {
20         element = driver.findElement(By.cssSelector("a[class='ng-binding ng-scope']"));
21         return element;
22     }
23
24     public static WebElement validacaoPesquisalomeProfComTurma(WebDriver driver) {
25         element = driver.findElement(By.xpath("/html/body/div/accordion/div/div[7]/div[2]/div/div[3]/table
```

Figura 4 – Arquivo Page Objects utilizado para mapear os elementos de uma tela.

A Figura 4 apresenta um exemplo do arquivo PageObjects de um projeto de automação. Nesse arquivo fica armazenado o “mapeamento” dos elementos da tela, ou seja, é a posição na tela do sistema em que aquele elemento se encontra. Após realizar esse mapeamento, no arquivo Step basta chamar a variável referente ao elemento que deseja interagir.

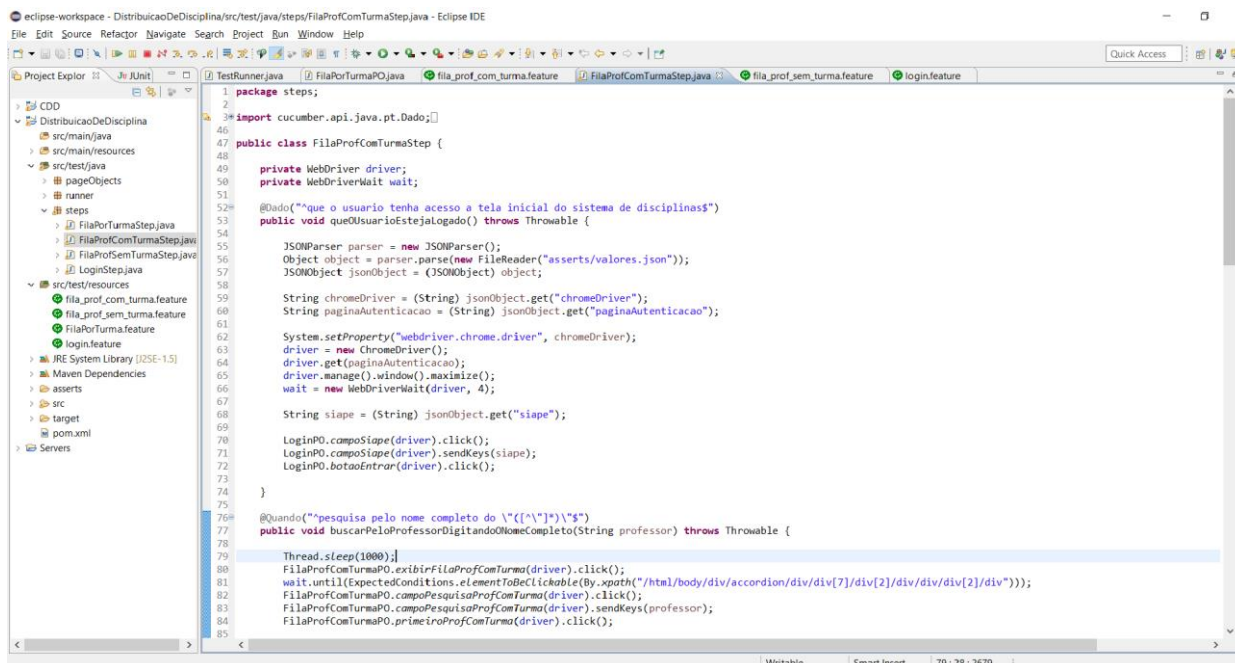


Figura 5 - Arquivo Step utilizando para definir o passo a passo do cenário de teste que será automatizado.

Na Figura 5 temos o arquivo “Step” que também faz parte do projeto de automação. Nesse arquivo é definido o passo a passo do teste, ou seja, o testador irá programar onde a automação irá clicar, escrever e também irá definir as validações, como por exemplo se um determinado campo, botão, escrita está presente na tela. No arquivo “Step” seguimos uma estrutura dividida em “Dado”, “Quando” e “Então”.

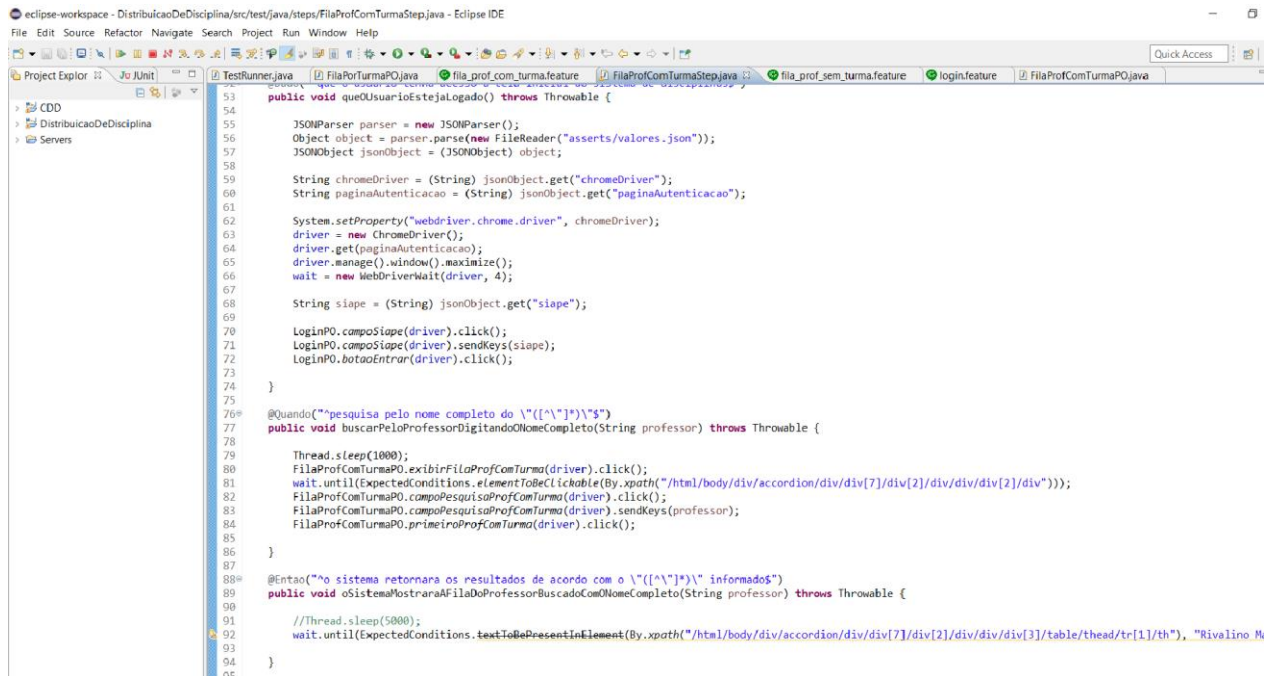
Basicamente em “Dado” definimos sempre o *Login* no sistema, ou seja, toda a parte de clicar nos campos, informar as credenciais necessárias e clicar no botão entrar, são definidas nessa parte. Em “Quando” é definido o passo a passo do teste, por exemplo, após entrar no sistema a automação deve clicar no campo de pesquisa, preencher o filtro com alguma informação e clicar no botão pesquisar. E por último, no “Então” (Figura 4) definimos qual validação será feita e como ela é feita, o que irá definir se o teste passou ou não.

Alguns comandos básicos de uma automação em Java podem ser observados na imagem acima como por exemplo:

O comando `click()` é utilizado para clicar em algum elemento da tela cujo a posição do mesmo foi informada no arquivo Page Objects (Figura 2). Para utilizá-lo basta chamar seu arquivo Page Objects seguido por ponto e o nome do elemento na tela. Exemplo: `ArquivoPageObjects.BotaoLogin.Click();`

O comando `sendKeys(Variável)` tem a função de escrever no campo o valor que é definido na variável declarada no arquivo “Feature” (Figura 1). Esse comando deve ser utilizado sempre após o comando de `click()`, pois para se escrever em um campo, é necessário clicar nele primeiro para depois escrever. Caso seja utilizado apenas o “sendKeys”, a automação irá falhar, pois não irá encontrar o campo para escrever.

Outro comando muito importante utilizado é o comando `wait.until` (Figura 5, Linha 81 do código). Sua função é pausar a automação por alguns milésimos de segundos ou até segundo até que um determinado elemento apareça na tela. Como a automação roda o código em uma velocidade alta, muitas vezes o navegador não consegue carregar a tempo um botão ou campo na tela, fazendo com que a automação acuse erro no sistema. Para evitar isso, utilizamos esse comando para pausar o código até que esse determinado elemento se torne visível na tela. Para determinar o tempo máximo de espera, utilizamos o comando “`wait = new WebDriverWait(driver, 4);`”, em que, nesse caso o valor 4, representa o valor máximo em segundos que será aguardado.



```
eclipse-workspace - DistribuicaoDeDisciplina/src/test/java/steps/FilaProfComTurmaStep.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer: Jo JUnit
TestRunner.java | FilaPorTurmaPO.java | fila_prof_com_turma.feature | FilaProfComTurmaStep.java | fila_prof_com_turma.feature | login.feature | FilaProfComTurmaPO.java
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
public void queUsuarioEstejaLogado() throws Throwable {
    JSONParser parser = new JSONParser();
    Object object = parser.parse(new FileReader("asserts/valores.json"));
    JSONObject jsonObject = (JSONObject) object;
    String chromeDriver = (String) jsonObject.get("chromeDriver");
    String paginaAutenticacao = (String) jsonObject.get("paginaAutenticacao");
    System.setProperty("webdriver.chrome.driver", chromeDriver);
    driver = new ChromeDriver();
    driver.get(paginaAutenticacao);
    driver.manage().window().maximize();
    wait = new WebDriverWait(driver, 4);
    String siape = (String) jsonObject.get("siape");
    LoginPO.compoSiape(driver).click();
    LoginPO.compoSiape(driver).sendKeys(siape);
    LoginPO.botoaEntrar(driver).click();
}
@Quando("pesquisa pelo nome completo do \"{[*]*}\"")
public void buscarPeloProfessorDigitandoNomeCompleto(String professor) throws Throwable {
    Thread.sleep(1000);
    FilaProfComTurmaPO.exibirFilaProfComTurma(driver).click();
    wait.until(ExpectedConditions.elementToBeClickable(By.xpath("/html/body/div/accordion/div/div[7]/div[2]/div/div[2]/div"));
    FilaProfComTurmaPO.compoPesquisaProfComTurma(driver).click();
    FilaProfComTurmaPO.compoPesquisaProfComTurma(driver).sendKeys(professor);
    FilaProfComTurmaPO.primeiroProfComTurma(driver).click();
}
@Entao("o sistema retornara os resultados de acordo com o \"{[*]*}\" informado")
public void oSistemaMostraraAFilaDoProfessorBuscadoComNomeCompleto(String professor) throws Throwable {
    //Thread.sleep(5000);
    wait.until(ExpectedConditions.textToBePresentInElement(By.xpath("/html/body/div/accordion/div/div[7]/div[2]/div/div[3]/table/thead/tr[1]/th"), "Rivalino M
```

Figura 6 - Arquivo Step utilizando para definir o passo a passo do cenário de teste que será automatizado.

Nesta quarta imagem podemos notar a parte do “Então” da estrutura definida acima. Nela, como foi dito, definimos qual validação será feita e como ela será feita. Nesse exemplo, trouxemos o comando `wait.until` (Figura 6, Linha 92 do código) que compara as informações de um elemento na tela com algum dado que foi passado para comparação. Nesse caso ele comparou o retorno do nome do professor presente no grid de pesquisa com o nome “Rivalino Matias Jr.”, validando assim que o nome informado na pesquisa foi o mesmo nome apresentando no grid após a pesquisa realizada, validando assim um retorno correto do sistema.

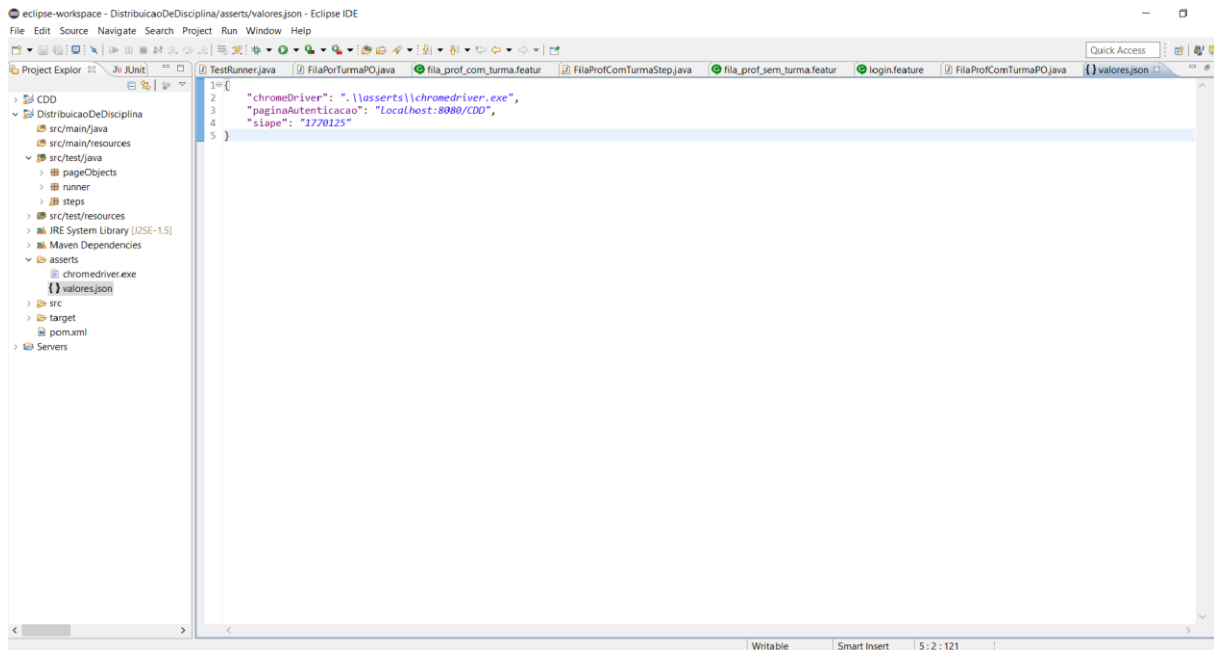


Figura 7 - Arquivo Json utilizado para armazenar valores em variáveis que serão utilizadas em vários lugares, facilitando a manutenção.

A imagem acima (Figura 7) mostra um arquivo no formato “Json” presente no projeto de automação. A função desse arquivo é armazenar alguma informação em uma determinada variável definida pelo testador. Sua principal finalidade é evitar a replicação de código e facilitar a manutenção do código caso se faça necessário. Vamos tomar como exemplo uma senha. Caso alguma senha, que é utilizada em vários lugares seja alterada, basta alterá-la apenas no arquivo Json, já que sempre que for necessário informar essa senha, a variável declarada no Json será chamada.

4. RESULTADOS

Levando em consideração os itens no capítulo anterior, foi desenvolvido uma automação de testes para cada funcionalidade do sistema individualmente, e para cada uma dessas partes foi utilizado uma forma diferente para que a validação do resultado do teste fosse feita. A partes automatizadas e a forma de como as validações foram desenvolvidas, foram feitas da seguinte maneira:

Login: A automação do login realizou a validação de dois cenários. Login válido e login inválido, em que no login válido foi passado o código correto, levando o usuário para a página inicial (Figura 8) e o login inválido permaneceu na tela de login (Figura 9), conseguindo assim, realizar a validação do campo

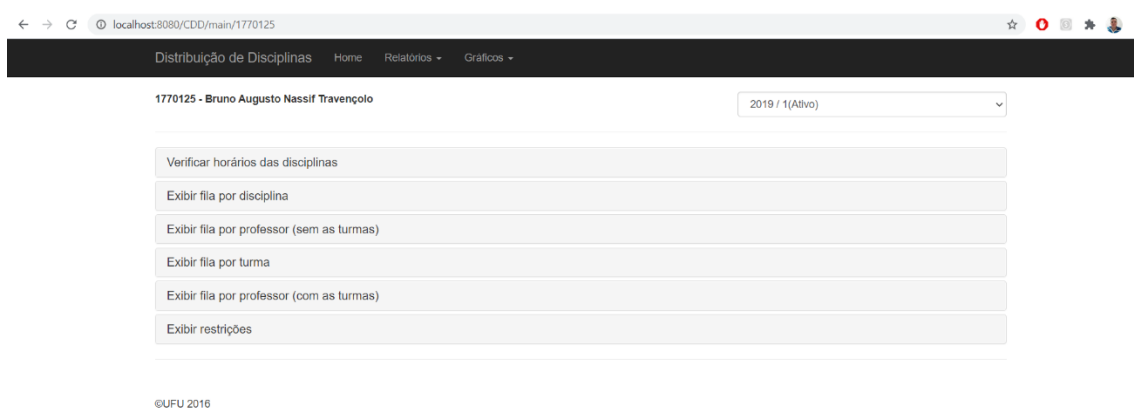


Figura 8 – Tela obtida após um login válido

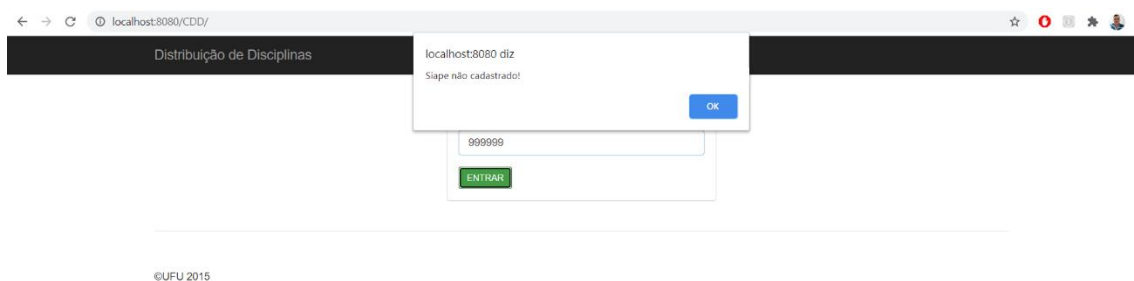
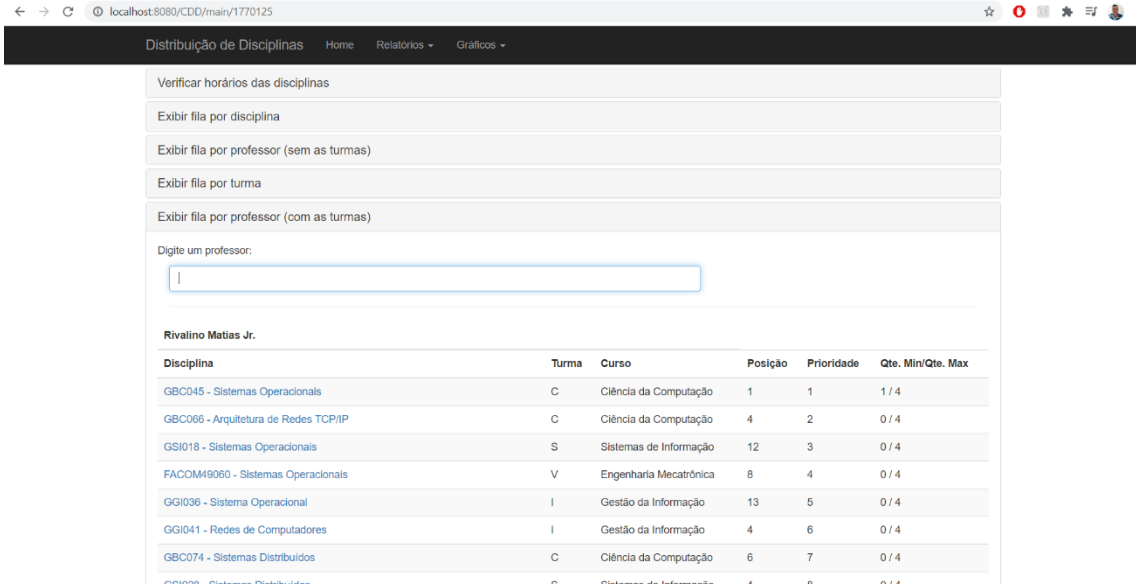


Figura 9 – Tela resultante de um login inválido. Indicando o erro obtido.

Exibir fila de professor com turma: A validação dessa funcionalidade se baseou em entrar no sistema, acessar o campo de pesquisa referente a essa funcionalidade, digitar o nome de um professor e validar que as informações foram trazidas de acordo com o professor informado. No sistema quando se realiza essa pesquisa é exibido o nome do

professor juntamente com a lista de disciplinas (Figura 10), com isso foi possível validar que o nome do professor pesquisado é o mesmo que o apresentado na tela, após realizar a pesquisa. Foi realizada também a validação de professor que não contém turma (Figura 11).. Para a validação em questão foi comparado se o nome do professor informado no campo de pesquisa não aparecia na tabela de resultados, com isso, nenhuma informação sobre o mesmo foi carregada pois ele não possui turma (Figura 12).



localhost:8080/CDD/main/1770125

Distribuição de Disciplinas Home Relatórios Gráficos

Verificar horários das disciplinas

Exibir fila por disciplina

Exibir fila por professor (sem as turmas)

Exibir fila por turma

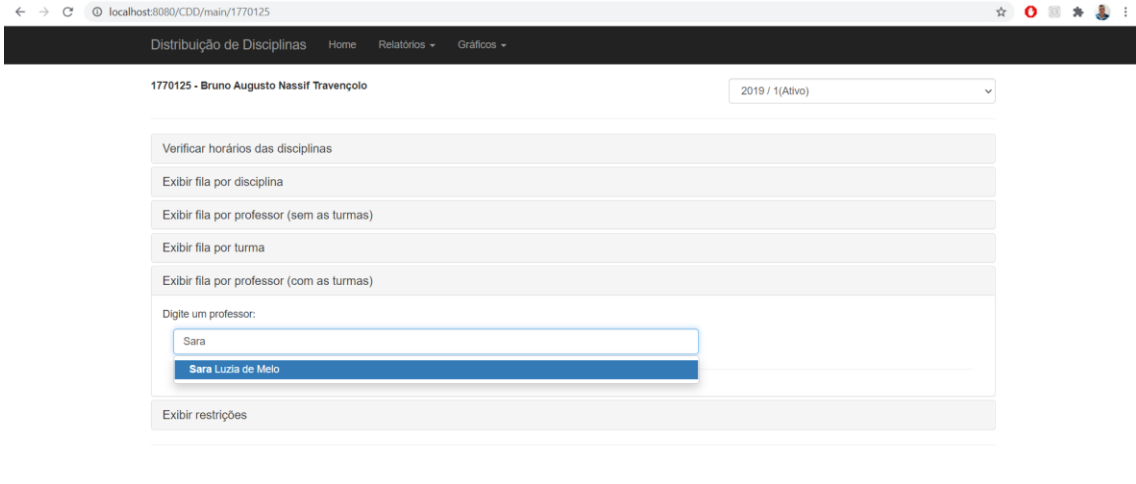
Exibir fila por professor (com as turmas)

Digite um professor:

Rivalino Matias Jr.

Disciplina	Turma	Curso	Posição	Prioridade	Qte. Min/Qte. Max
GBC045 - Sistemas Operacionais	C	Clência da Computação	1	1	1 / 4
GBC066 - Arquitetura de Redes TCP/IP	C	Clência da Computação	4	2	0 / 4
GSI018 - Sistemas Operacionais	S	Sistemas de Informação	12	3	0 / 4
FACOM49060 - Sistemas Operacionais	V	Engenharia Mecatrônica	8	4	0 / 4
GGI036 - Sistema Operacional	I	Gestão da Informação	13	5	0 / 4
GGI041 - Redes de Computadores	I	Gestão da Informação	4	6	0 / 4
GBC074 - Sistemas Distribuídos	C	Clência da Computação	6	7	0 / 4
GSI028 - Sistemas Distribuídos	S	Sistemas de Informação	4	8	0 / 4

Figura 10 - Professor com turma – são listadas as turmas do professor consultado.



localhost:8080/CDD/main/1770125

Distribuição de Disciplinas Home Relatórios Gráficos

1770125 - Bruno Augusto Nassif Travençolo 2019 / 1(Abivo)

Verificar horários das disciplinas

Exibir fila por disciplina

Exibir fila por professor (sem as turmas)

Exibir fila por turma

Exibir fila por professor (com as turmas)

Digite um professor:

Sara

Sara Luzia de Melo

Exibir restrições

©UFU 2016

Figura 11 - Professor sem turma (realização da busca).

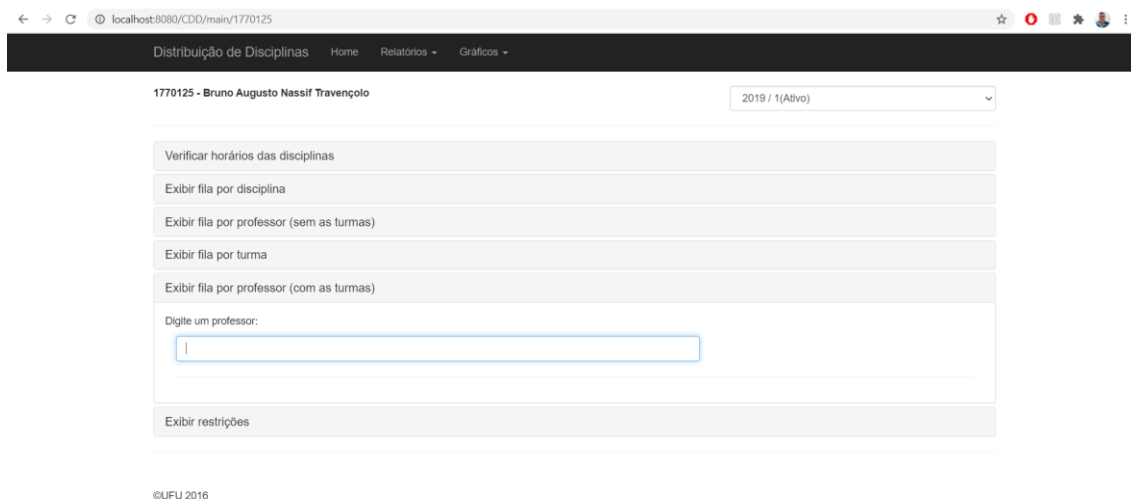


Figura 12 - Professor sem turma (resultado da busca).

Exibir fila de professor por turma: Para realizar a validação da busca de fila de professor de uma determinada turma foi feita utilizando o nome da disciplina pesquisada. Após realizar o login no sistema, expandir a opção “Buscar fila de professor por turma” e inserir uma disciplina no campo de pesquisa, o sistema apresenta os nomes dos professores que estão na fila para ministrar as aulas daquela disciplina. Para validar o retorno correto da disciplina, foi feita a comparação do nome que foi passado no campo de pesquisa, para realizar a busca com o nome apresentado no grid de resultados da pesquisa. Como o nome da disciplina informada no campo da pesquisa é igual ao nome da disciplina apresentado no grid de pesquisa, conclui-se que o retorno da pesquisa foi correto (Figura 13).

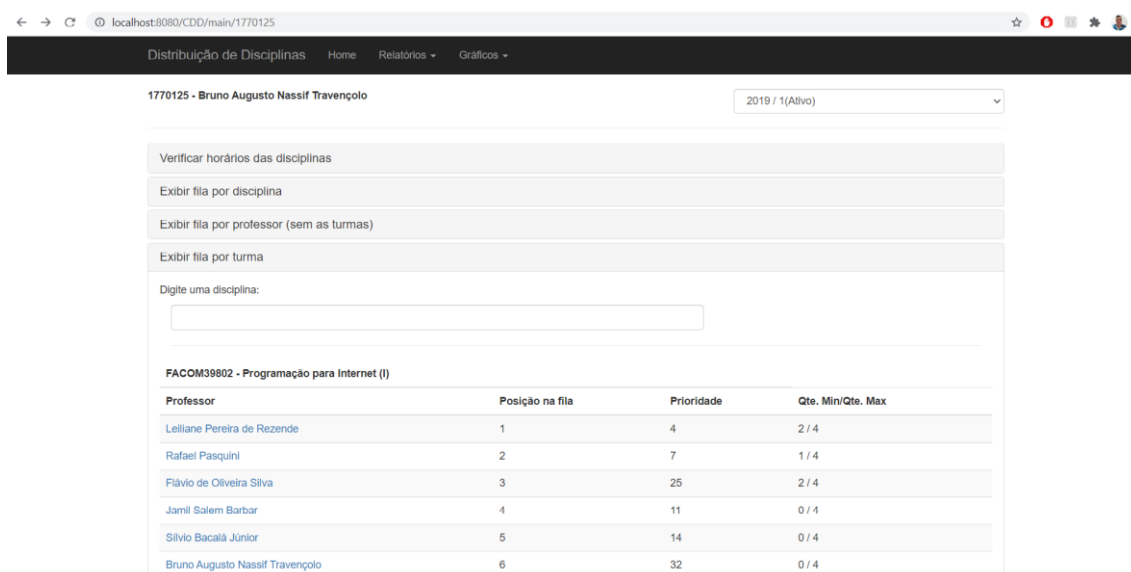


Figura 13 - Fila de professores por turma.

Exibir fila de professor sem turma: Para validar essa funcionalidade do sistema foi criado todo o fluxo do sistema, desde o login até a pesquisa de professor. Para realizar a validação dessa funcionalidade foi utilizado a mesma lógica que a automação da funcionalidade “Exibir fila de professor com turma”. Após a pesquisa, foi realizado a verificação do nome apresentado na tabela de resultados, validando assim que as informações apresentadas na pesquisa, foram do professor informado no campo de pesquisa (Figura 14 - Figura 16).

1770125 - Bruno Augusto Nassif Travençolo

2019 / 1(Ativo)

Verificar horários das disciplinas

Exibir fila por disciplina

Exibir fila por professor (sem as turmas)

Digite um professor:

Bruno Augusto Nassif Travençolo

Disciplina	Curso	Prioridade	Posição	Qte. Min/Qte. Max
GSI006 - Estrutura de Dados 1	Sistemas de Informação	1	10	0/4
FACOM49080 - Bancos de Dados	Engenharia Mecatrônica	2	6	0/4
FACOM49010(V) - Algoritmos e Programação de Computadores	Engenharia Mecatrônica	3	15	0/4
FACOM49010(U) - Algoritmos e Programação de Computadores	Engenharia Mecânica	4	12	0/4
GSI016 - Bancos de Dados 1	Sistemas de Informação	5	14	0/4
GBT017 - Informática para Biotecnologia	Biotecnologia	6	6	0/4
GSI011 - Estrutura de Dados 2	Sistemas de Informação	7	16	0/4
GSI021 - Bancos de Dados 2	Sistemas de Informação	8	4	0/4

Figura 14 - Fila por professor (sem as turmas). Consulta com retorno.

1770125 - Bruno Augusto Nassif Travençolo

2019 / 1(Ativo)

Verificar horários das disciplinas

Exibir fila por disciplina

Exibir fila por professor (sem as turmas)

Digite um professor:

Adriano Mendonça Rocha

Bruno Augusto Nassif Travençolo

Disciplina	Curso	Prioridade	Posição	Qte. Min/Qte. Max
GSI006 - Estrutura de Dados 1	Sistemas de Informação	1	10	0/4
FACOM49080 - Bancos de Dados	Engenharia Mecatrônica	2	6	0/4
FACOM49010(V) - Algoritmos e Programação de Computadores	Engenharia Mecatrônica	3	15	0/4
FACOM49010(U) - Algoritmos e Programação de Computadores	Engenharia Mecânica	4	12	0/4
GSI016 - Bancos de Dados 1	Sistemas de Informação	5	14	0/4
GBT017 - Informática para Biotecnologia	Biotecnologia	6	6	0/4
GSI011 - Estrutura de Dados 2	Sistemas de Informação	7	16	0/4
GSI021 - Bancos de Dados 2	Sistemas de Informação	8	4	0/4

Figura 15 - Fila por professor (sem as turmas), realização da busca.

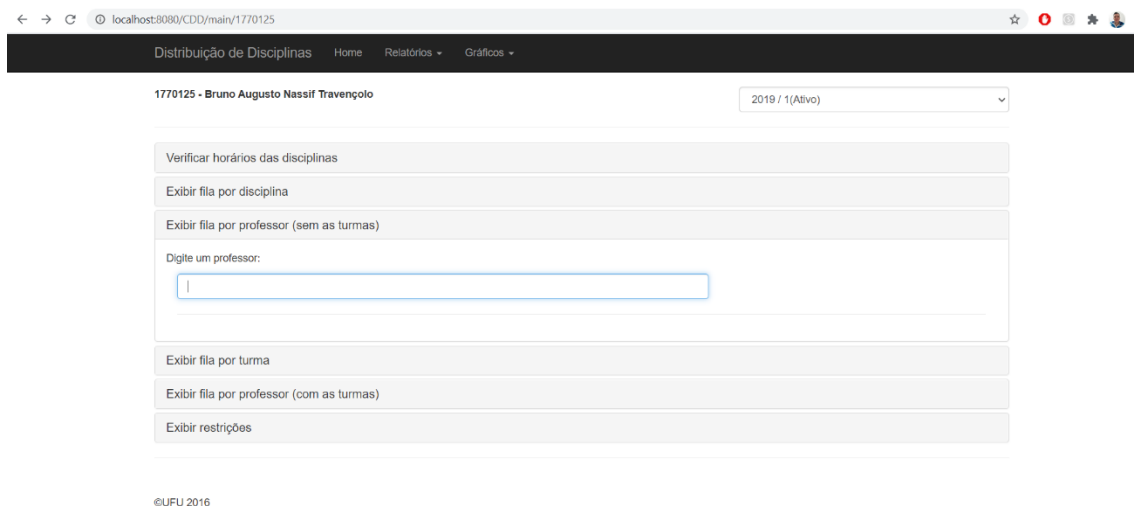


Figura 16 - Fila por professor (sem as turmas), resultado da busca

Cadastrar curso: Para automatizar o cadastro de um novo curso, foi necessário entrar diretamente no sistema na parte de Administrador. Para isso, ao acessar o navegador, ao invés de acessar o link “Localhost:8080/CDD” nós acessamos o link “Localhost:8080/CDD/Admin/fila” que nos leva diretamente para a parte de administração no sistema.

Após realizar esse acesso foi necessário seguir o seguinte fluxo para cadastro: Acessar o menu Administrador e Clicar no botão “Curso”. Ao clicar no botão, o sistema nos direciona para a tela de pesquisa de curso, e nessa mesma tela é apresentado o botão “Novo Curso”. Com isso, após acessar a tela nós clicamos no botão “Novo Curso” para que a tela de cadastro seja apresentada. Com a apresentação da tela a automação irá preencher os campos “Código”, “Nome”, “Unidade” e “Campus” de acordo com os valores que nós passamos, e irá clicar no botão salvar, criando assim um novo registro.

O passo a passo que foi seguido, pode ser observado na Figura 17 e na Figura 18 abaixo.

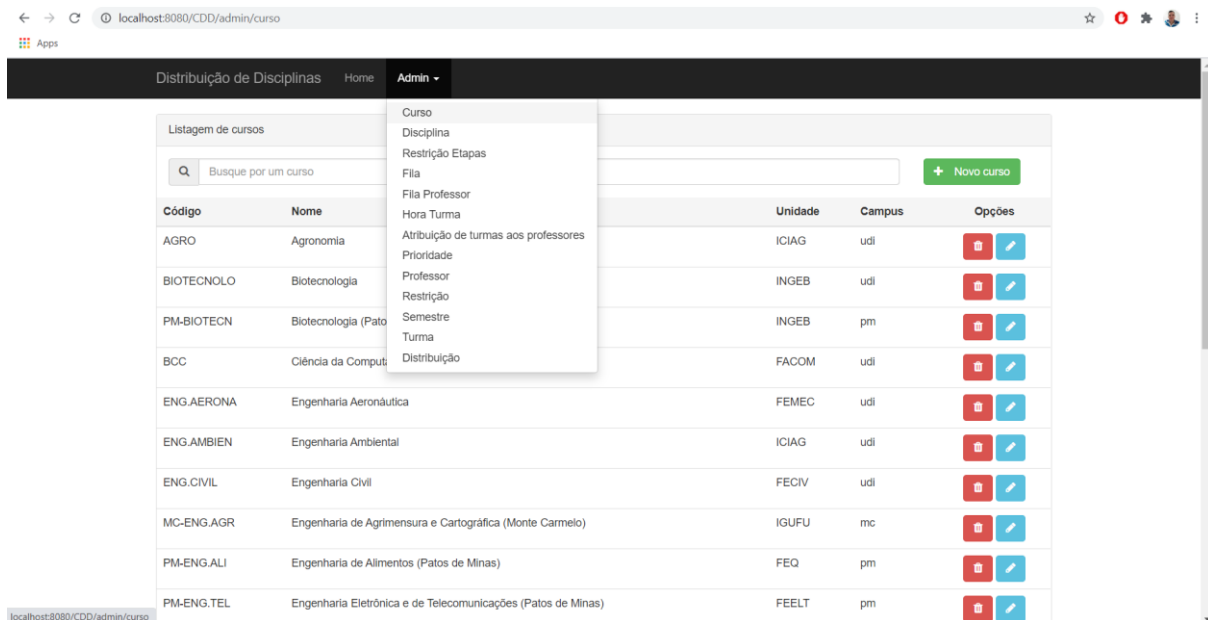


Figura 17 - Tela de pesquisa, edição, exclusão e botão de acesso para cadastrar um curso

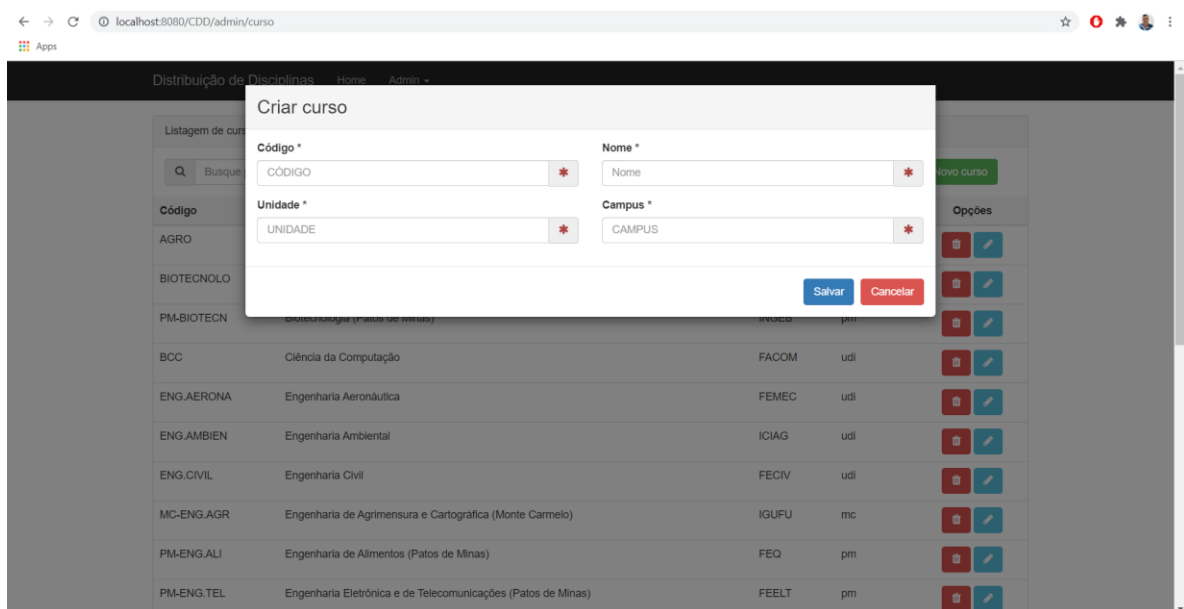


Figura 18 - Tela de cadastro de curso

Pesquisa de curso: O fluxo da automação de pesquisa de curso segue os mesmos passos da automação de cadastro de curso, porém na pesquisa não é necessário clicar no botão “Novo Curso”. Além disso, ao entrar na tela “Curso” nós clicamos no campo de pesquisa, informamos os dados e validamos o retorno.

Na automação feita a validação foi feita por meio da comparação da disciplina informada na pesquisa com a disciplina apresentada no grid de pesquisa. O teste foi realizado com dois cenários: Uma pesquisa informando um nome incompleto de curso e

uma pesquisa informando o nome completo do curso, garantindo assim que independente da maneira que o usuário informar o nome do curso, o sistema irá apresentar o resultado correto.

Os dois cenários informados acima podem ser observados na Figura 19 e Figura 20 abaixo.

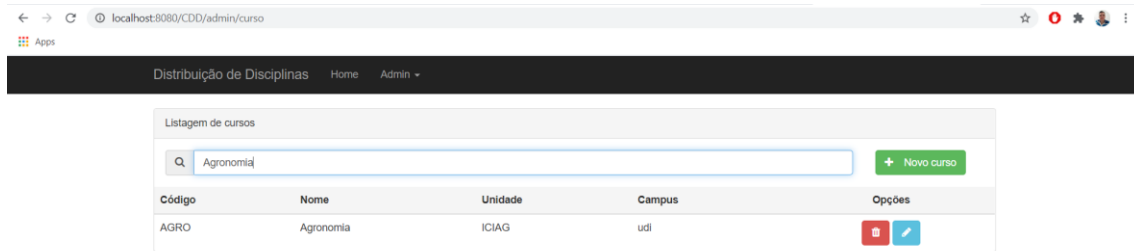


Figura 19 - Tela de pesquisa de curso por nome completo.

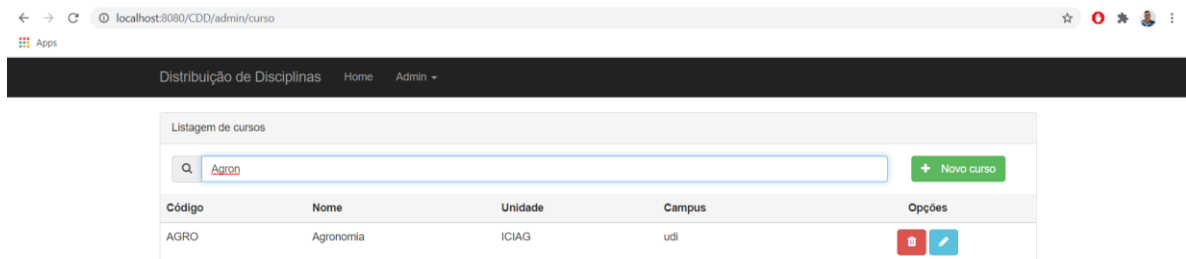
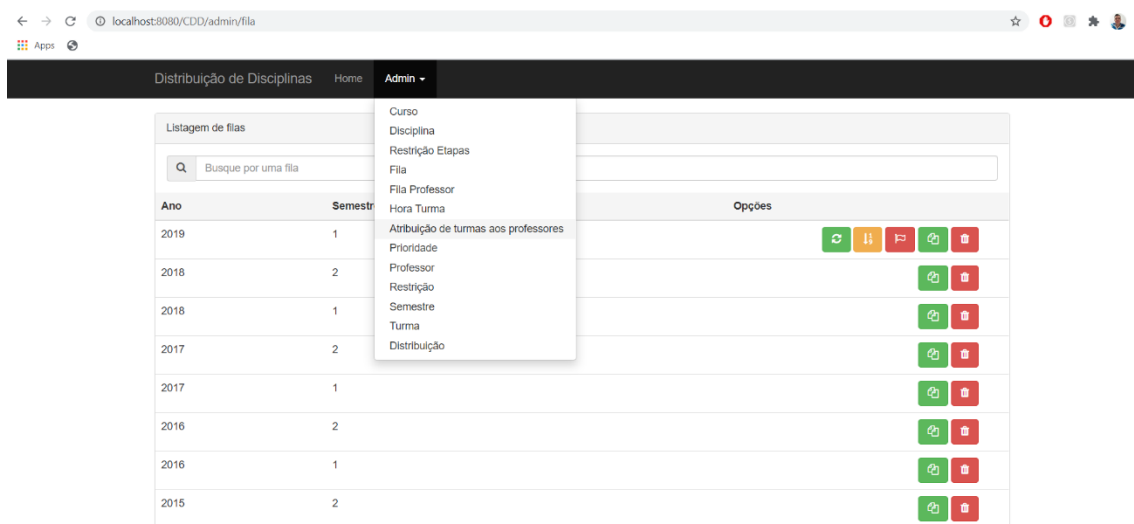


Figura 20 - Tela de pesquisa de curso por nome incompleto.



localhost:8080/CDD/admin/ministraDisciplina

Figura 21 - Acesso a tela de atribuição de turmas aos professores.

Atribuir turma ao professor: Para acessar a tela de atribuição de disciplina para um determinado professor é preciso acessar o link do sistema como administrador e no menu de “Admin”, selecionar a opção “Atribuição de turmas aos professores, como apresentado na Figura 21.



Figura 22 - Tela atribuição de turmas aos professores.

Acessando o menu de “Atribuição de turmas aos professores” o sistema nos direciona para uma tela que possui duas finalidades, “Atribuir turma ao professor” e “Exibir turmas por professor” como mostra a Figura 22.

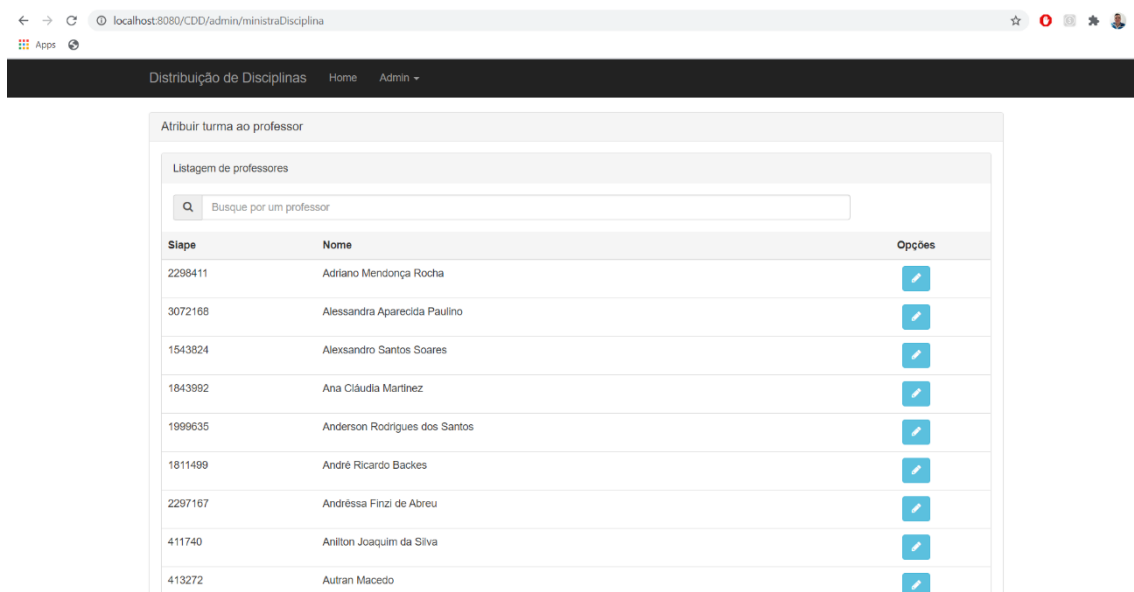


Figura 23 - Tela de pesquisa para atribuição de turmas aos professores.

Acessando a opção desejada, que no nosso caso seria “Atribuir turma ao professor”, é apresentado a lista de professores, juntamente com um campo de busca, em

que informamos o nome do professor a qual desejamos atribuir determinada disciplina e um botão ao lado de seu nome, para que a atribuição seja feita, conforme Figura 23.

Ao clicar no botão ao lado, para atribuir a disciplina desejada ao professor, é apresentado uma janela para que informações como Semestre e Turma, sejam preenchidas. Essa janela apresentada pode ser observada na Figura 24.

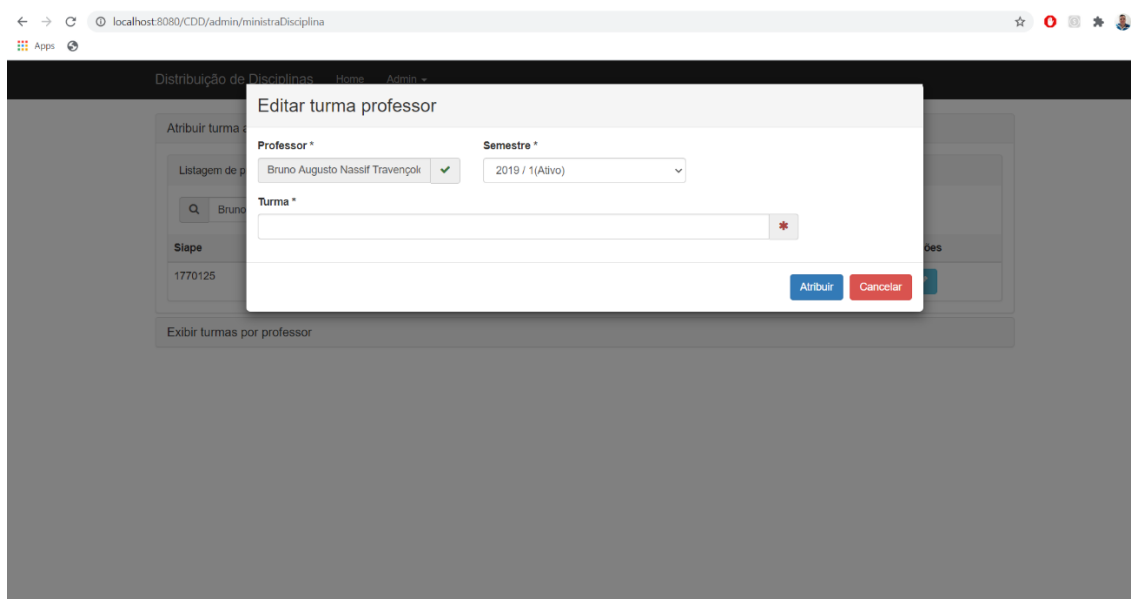


Figura 24 - Tela para realizar atribuição de turmas aos professores

Após preencher todos os campos obrigatórios o sistema atribui a disciplina ao professor e fecha a janela de atribuição de disciplina. Caso falte alguma informação obrigatória, o sistema não salva e mantém a janela aberta. Na automação para essa validação ser feita, foi levado em consideração que, caso a automação clique no botão salvar e o título “Editar turma professor” ainda esteja sendo apresentado em tela, a edição não foi realizada visto que a janela continuou aberta. Dessa forma, conseguimos validar que os campos foram preenchidos e a disciplina atribuída.

Cadastrar e excluir professor: No sistema de distribuição de disciplinas é possível tanto cadastrar quanto excluir um professor que já está cadastrado no sistema. Para cadastrar devemos acessar o sistema pelo portal Administrador, acessar o menu Admin e entrar na opção Professor. Na tela professor teremos então o botão Novo professor (Figura 25), o qual nos possibilita realizar o cadastro de um novo professor no sistema.

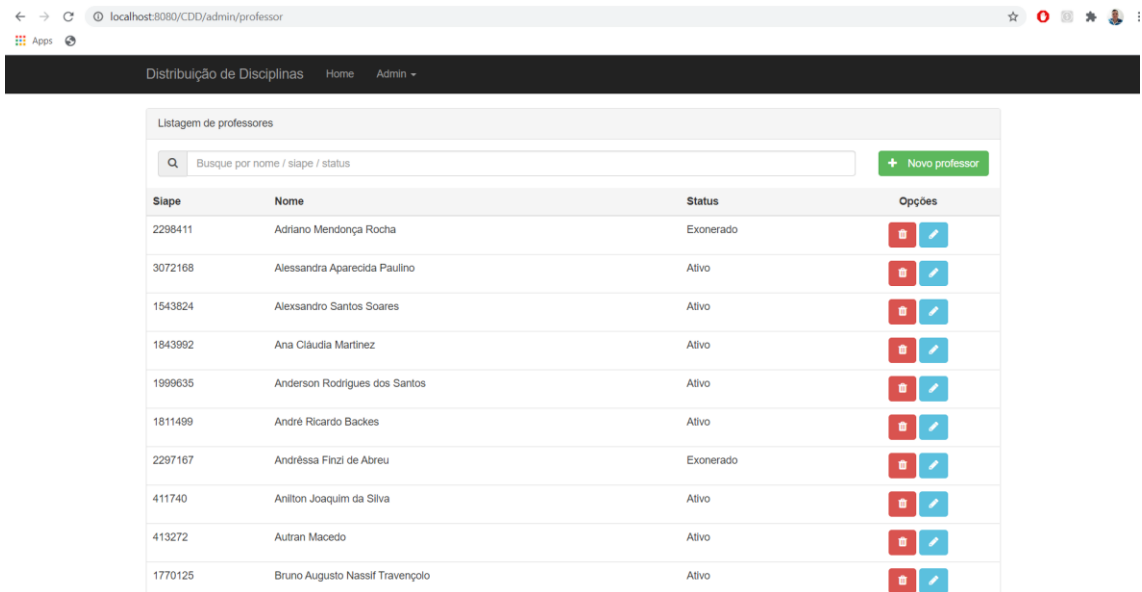


Figura 25 - Tela de pesquisa, edição, exclusão e botão de acesso à tela para cadastrar professor.

Quando clicamos no botão Novo professor o sistema nos apresenta um modal para preencher algumas informações sobre o novo professor a ser cadastrado, como por exemplo: Siape, Nome, Data de Ingresso, Data de Nascimento, etc. Esse modal pode ser observado na Figura 26.

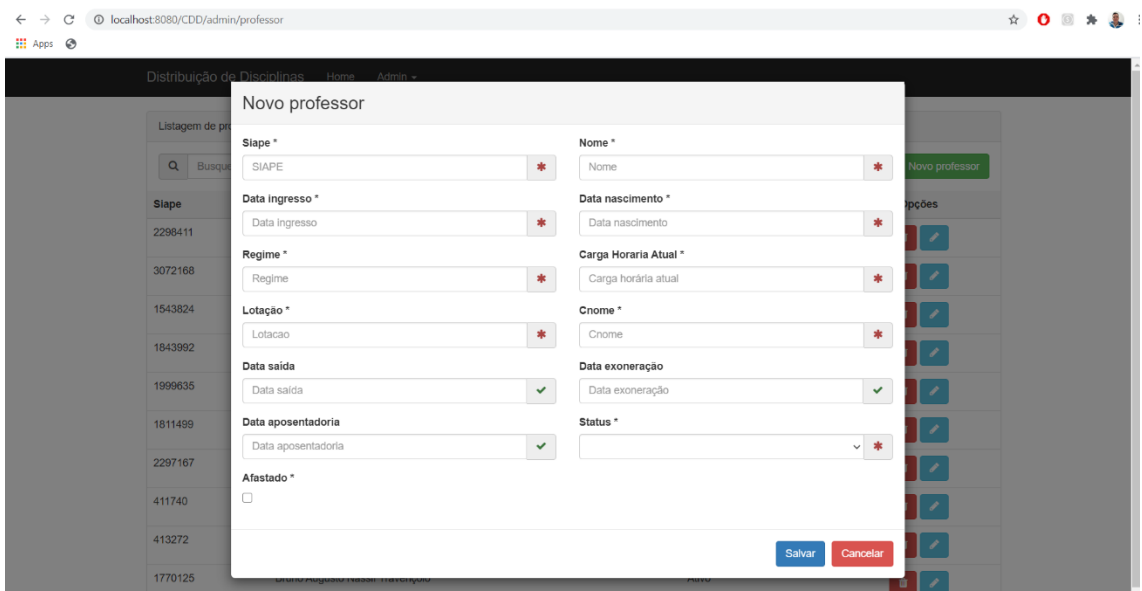


Figura 26 - Modal de cadastro de professor

Como teste, é preciso validar que após preencher as informações necessárias, o sistema irá salvar o registro corretamente. Na automação, como forma de validar isso utilizamos a seguinte lógica, preenchamos todas as informações sobre o novo professor a

ser cadastrado no sistema, clicamos no botão salvar, pesquisamos esse novo registro pelo nome do professor e validamos que as informações que são apresentadas no grid sobre esse novo professor, são as mesmas que informamos durante o cadastro, e desta forma nós conseguimos validar que as informações passadas durante o cadastro foram salvas corretamente no banco de dados do sistema.

Para realizar a exclusão de um registro usamos o mesmo registro utilizado nas informações acima. O passo a passo para acesso a tela de exclusão é praticamente o mesmo, nós acessamos o sistema pelo portal Administrador, pesquisamos o registro pelo nome que desejamos, clicamos no botão excluir que é apresentado ao lado das informações no grid (Figura 27) e confirmamos a exclusão pelo modal apresentado para confirmação (Figura 28).



Figura 27 - Excluir registro do sistema.

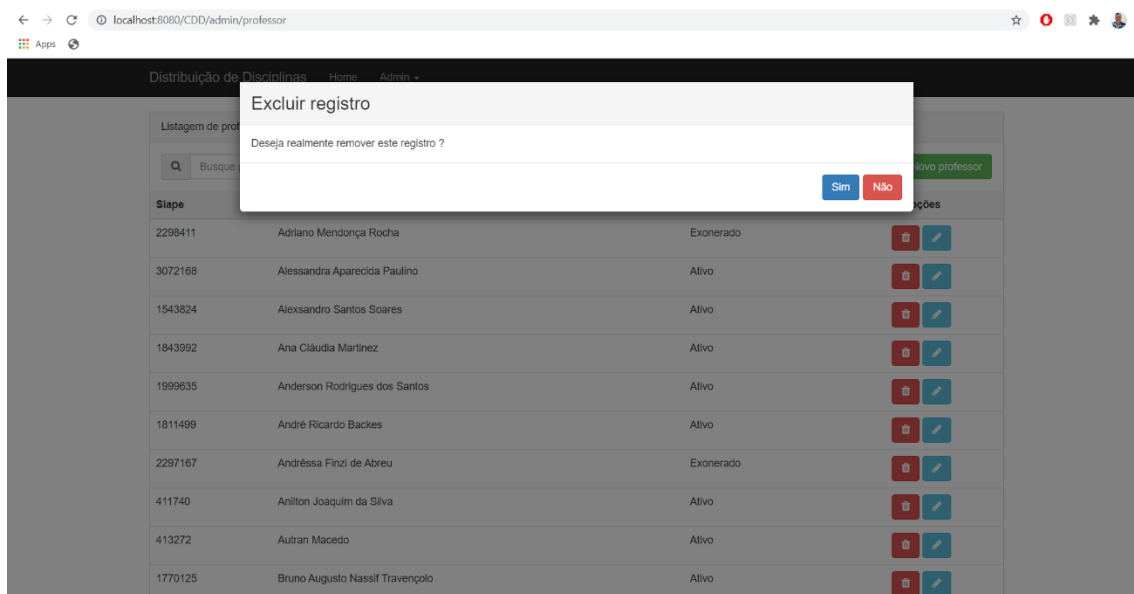


Figura 28 - Confirmar exclusão de registro.

Após confirmar a exclusão do registro, é preciso validar que o registro realmente foi excluído do sistema. Para validar a função de exclusão do sistema na automação, basicamente após a exclusão, nós pesquisamos novamente pelo nome do professor, o registro que acabamos de excluir. Por padrão no sistema, quando pesquisamos um registro

inexistente ele nos apresenta no grid a seguinte mensagem, “Nenhum registro encontrado” (Figura 29), informando o usuário de que o registro que ele busca, não existe no sistema. Logo, com essa mensagem conseguimos validar que o registro foi realmente excluído, nós guardamos essa mensagem em uma variável e comparamos ela com a mensagem utilizando o comando *Assert.assertEquals* no código (Figura 30). Caso a mensagem seja apresentada, ou seja, caso o registro tenha sido excluído, a variável irá receber a mensagem de que nenhum registro foi encontrado e assim validamos que ele realmente foi excluído.

Caso falhe a exclusão, a variável não irá receber a mensagem “Nenhum registro encontrado” pois a mesma não será apresentada na tela, com isso certificamos de que a exclusão falhou.

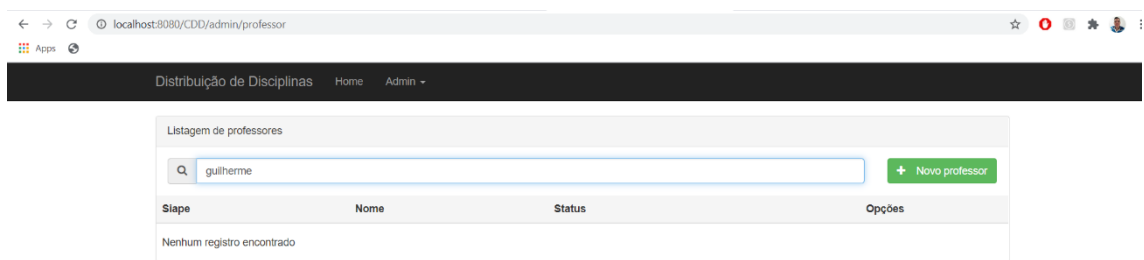


Figura 29 - Mensagem de nenhum registro encontrado

```
131  
132     String nenhumRegistro = CadastrarProfessorPO.nenhumRegistro(driver).getIext();  
133  
134     Assert.assertEquals(nenhumRegistro, "Nenhum registro encontrado");  
135     System.out.println("Registro excluído com sucesso !! ");  
136 }  
137
```

Figura 30 - Comando "AssertEquals" utilizado para comparação de variáveis

5. CONCLUSÃO

Este trabalho teve a finalidade de apresentar sobre o processo de testes em softwares, sua importância e seus pontos teóricos, assim como apresentar também uma aplicação da automação em um sistema web utilizando o Eclipse e a linguagem Java.

Com base no que foi apresentado, podemos inferir que o teste de forma geral é de suma importância quando se trata de desenvolvimento de software. Notamos que essa prática, quando aplicada dentro de uma empresa, traz grandes benefícios para mesma, podendo, esses benefícios, serem financeiros ou não. Podemos notar também os benefícios dos testes automatizados. Como podemos ver, todos os testes que foram automatizados poderão ser executados novamente em qualquer momento quantas vezes forem necessárias, o que implica em ganhos de tempo visto que o testador não precisará pausar o seu atual serviço para realizar validações em itens que estão automatizados.

Durante o desenvolvimento dos testes foram encontrados alguns *bugs*, como por exemplo, pesquisas incorretas, campos de pesquisa com erro na função de auto complete, entre outros, os quais foram repassados para outros alunos que também trabalharam no projeto realizando as correções destes, e com isso pode-se concluir que o desenvolvimento do trabalho foi de grande valia já que durante o seu processo de desenvolvimento alguns problemas foram corrigidos além dos testes de alguns itens importantes estarem automatizados, o que irá garantir em menor tempo, se comparado a testes manuais, um bom funcionamento do sistema em atualizações futuras.

6. REFERÊNCIAS

BASTOS, Anderson; RIOS Emerson; CRISTALLI Ricardo; MOREIRA Trayahú. **Base de conhecimento em teste de Software**. São Paulo: Martins, 2007.

BERNARDO, Paulo; KON, Fabio. **A Importância dos Testes Automatizados**. Publicado na Engenharia de Software Magazine, 1(3), pp. 54-57. 2008.

BUI, Trong. **Testes de automação com Cucumber BDD em times ágeis**. <<https://www.infoq.com/br/articles/cucumber-bdd-automation-testing/>> Acesso em 22/07/2020

CUNHA, Simone. **ENGENHARIA DE SOFTWARE – UMA ABORDAGEM À FASE DE TESTES**. Disponível em: <https://repositorio.ufmg.br/bitstream/1843/BUBD-B8XKUP/1/espinform_tica_simonemoreiracunha_monografia.pdf> Acesso em: 22/07/2020

DAWSON, Maurice; BURREL, Darrell; RAHIM, Emad; BEWSTER, Stephen. **Integrating software assurance into the software Development life cycle (SDLC)**.

Disponível em:

<<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.668.9372&rep=rep1&type=pdf>> Acesso em 01/12/2020.

IZABEL, Leonardo R. P. **TESTES AUTOMATIZADOS NO PROCESSO DE DESENVOLVIMENTO DE SOFTWARES**. www.monografias.poli.ufrj.br, 2014. Disponível em: <<http://monografias.poli.ufrj.br/monografias/monopoli10012548.pdf>> Acesso em: 22/07/2020

MYERS, Glenford J. **The art of software testing**. New York: John Wiley & Sons, 1979, 177p.

OLIVEIRA, André. **Novas Funcionalidades e Manutenção no Sistema Online de Distribuição de Disciplinas**. Disponível em:

<<http://repositorio.ufu.br/bitstream/123456789/19437/3/NovasFuncionalidadesManutencao.pdf>> Acesso em 03/12/2020.

OLIVEIRA, Ângela. **Testes de Software para reduzir custos em um projeto**. BAGUETE. Disponível em: <<http://www.baguete.com.br/artigos/856/angela-oliveira/23/06/2010/testes-de-software-para-reduzir-custos-em-um-projeto>> Acesso em: 15/08/2020

PERES, Paulo. **Teste de Software: Os primeiros passos em busca da qualidade de Software**. Disponível em: <<http://pt.slideshare.net/pauloperes2009/testes-de-software-uma-viso-geral>> Acesso em 25/11/2020