02

# Teste do formulário de login

## Transcrição

Na última aula aprendemos a baixar e configurar as dependências do Selenium, utilizando o Maven, e do driver do Chrome. Além disso, escrevemos e executamos nosso primeiro teste, o `helloworld`.

Agora começaremos a realmente testar as funcionalidades do Selenium, a começar pela funcionalidade de login. Quando entramos na aplicação, temos no canto superior direito um link "Entrar" que nos envia a um formulário de login bem clássico: nome de usuário, senha e um botão para enviar as informações. Após preencher as informações e clicar no botão, a autenticação é feita e, na página de leilões, o nome do usuário logado passa a ser exibido no canto superior direito, junto de um botão "Sair".

Essa é a funcionalidade que queremos testar. Escrever um teste end-to-end é justamente traduzir esse passo a passo que fizemos manualmente para código. Sendo assim, antes de escrevermos um teste, precisamos entender qual é a funcionalidade que queremos testar e simular os passos que normalmente são realizados. Essa é, inclusive, a ideia de um teste E2E: simular a utilização de um software do ponto de vista do usuário.

Ou seja, para efetuar um login na aplicação, os passos são: entrar na tela de login, preencher o formulário com usuário e senha, clicar no botão de logar e ser redirecionado para a tela de leilões com o nome de usuário sendo exibido no

canto superior direito. É isso que desejamos verificar se está funcionando como esperado, mas agora de maneira automatizada usando a API do Selenium.

Como este será um novo teste, criaremos uma classe `LoginTest` no pacote `br.alura.leilao.login`. Existem vários cenários de autenticação, mas o que desejamos executar nessa aula é aquele em que a autenticação é efetuada normalmente: o usuário digita um usuário e senha corretamente, clica no botão de logar e a autenticação é bem-sucedida.

Criaremos um método `deveriaEfetuarLoginComDadosValidos()` que simulará esse cenário. Adicionaremos a anotação `@Test` e importaremos o seu pacote. O primeiro passo do teste será abrir o navegador e entrar na página de login, algo que copiaremos do `helloTest()`, substituindo a navegação para a página de leilões pela página correta.

```
public class LoginTest {
    @Test
    public void deveriaEfetuarLoginComDadosValidos() {
        System.setProperty("webdriver.chrome.driver", "/drivers/c
        WebDriver browser = new ChromeDriver();
        browser.navigate().to("http://localhost:8080/login");
    }
}
```

COPIAR CÓDIGO

O próximo passo é preencher os campos "Usuário" e "Senha". Mas como fazemos para que o Selenium encontre um elemento em uma página HTML e digite um valor em um campo de *input*? Como queremos manipular a página, usaremos a variável `browser` e, a partir dela, chamaremos o método `findElement()`, que nos permite encontrar um elemento na página.

Esse método recebe como parâmetro uma outra classe do Selenium, chamada `By`. Essa classe não é instanciada, pois possui diversos métodos estáticos. Existem várias maneiras de referenciarmos um elemento de uma página HTML, por exemplo pelo nome da classe CSS, pelo nome da tag, pelo ID e assim por diante - ou seja, pelos recursos e atributos do próprio HTML. A escolha de qual utilizar vai depender de como a sua página está configurada.

Geralmente, recomenda-se utilizar o ID, e não a classe. Isso porque as classes geralmente são utilizadas pela equipe de front-end, que cuida do CSS, e elas posteriormente podem ser modificadas gerando um impacto nos testes. O mesmo vale para os nomes das tags ou o xpath, pois qualquer mudança na estrutura da página pode quebrar os testes.

Sendo assim, utilizaremos o ID também no nosso treinamento. Mas qual o ID dos nossos inputs? Uma das desvantagens dos testes de interface, seja com o Selenium ou com qualquer outra biblioteca, é que eles são muito frágeis devido a serem acoplados à tela. A ideia é que, para escrevermos o teste, precisamos conhecer coisas externas ao teste - como os IDs de uma página HTML. Da mesma forma, se as IDs forem alteradas, ao invés de termos um erro de compilação, nossos testes irão quebrar.

Os testes E2E, portanto, são bem frágeis e costumam quebrar com facilidade conforme os times de desenvolvimento trabalham nas páginas. Isso significa que você e o seu time devem ter isso em mente para tomar determinados cuidados de modo a evitar que os testes quebrem.

Existem duas maneiras de descobrirmos o ID dos nossos inputs: inspecionando pelo browser ou abrindo o arquivo HTML do projeto. Inspecionando pelo navegador ("botão direito do mouse > inspect element"), descobriremos que nosso input na realidade não possui um ID. Portanto, às vezes pode ser

necessário alterarmos o HTML para escrevermos nossos testes da maneira que estamos planejando.

No projeto existe um arquivo `login.html` localizado no diretório "leilao/src/main/rescources/tempaltes". Nele encontraremos os inputs que representam o usuário e a senha, e adicionaremos os ids `username` e `password`, respectivamente. Note que eles serão idênticos ao atributo `for` da tag `<label>`, o que é uma boa prática.

```
<div class="form-group">
    <label for="username">Usuário</label>
    <input id="username" name="username"  class="form-control" p
</div>

<div class="form-group">
    <label for="password">Senha</label>
    <input id="password" type="password" name="password" class="
</div>
```

COPIAR CÓDIGO

Agora que incluídos o ID nos dois campos, voltaremos ao teste. Passaremos para o método `findElement(By.id())` a string "username". Nesse ponto temos duas escolhas: guardar essa informação em uma variável ou encadear a chamada de outro método. Como não usaremos esse elemento posteriormente, simplesmente encadearemos a chamada de `sendKeys()`, um método que permite a entrada de um texto em um campo, com o argumento `fulano`.

```
public void deveriaEfetuarLoginComDadosValidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chro
    WebDriver browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");
```

```
        browser.findElement(By.id("username")).sendKeys("fulano");

    }
```

COPIAR CÓDIGO

Repetiremos esse procedimento para a senha, substituindo `username` por `password` e `fulano` por `pass`.

```
    public void deveriaEfetuarLoginComDadosValidos() {
        System.setProperty("webdriver.chrome.driver", "/drivers/chror
        WebDriver browser = new ChromeDriver();
        browser.navigate().to("http://localhost:8080/login");
        browser.findElement(By.id("username")).sendKeys("fulano");
        browser.findElement(By.id("password")).sendKeys("pass");
    }
```

COPIAR CÓDIGO

O próximo passo será submetermos o formulário, o que pode ser feito clicando no botão de login, fazendo um *submit* na tag `form` ou a partir de um campo, dentre outras maneiras. Em nosso caso, recuperaremos a tag `<form>`, cujo ID é `login-form`, e chamaremos o método `submit()`.

```
    public void deveriaEfetuarLoginComDadosValidos() {
        System.setProperty("webdriver.chrome.driver", "/drivers/chror
        WebDriver browser = new ChromeDriver();
        browser.navigate().to("http://localhost:8080/login");
        browser.findElement(By.id("username")).sendKeys("fulano");
        browser.findElement(By.id("password")).sendKeys("pass");
        browser.findElement(By.id("login-form")).submit();
    }
```

COPIAR CÓDIGO

Outra alternativa seria usarmos `findElement()` para encontrarmos o botão de login e chamarmos o método `click()`, que dispara um clique.

Recapitulando, em nosso teste estamos abrindo o navegador, acessando a página de login, preenchendo os campos "username" e "password" e enviando o formulário. Agora precisamos verificar se tudo funcionou conforme o esperado, algo que também pode ser feito de diferentes maneiras: é possível, por exemplo, verificar se a URL do navegador mudou (deixou de apresentar a URL de login e navegou até a URL de leilões), e se o nome "fulano" apareceu no campo superior direito, indicando que o usuário foi logado no sistema.

Agora utilizaremos os *Asserts* do JUnit para fazermos as assertivas. Com `Assert.assertFalse()`, verificaremos se a URL na qual estamos no momento não é mais a de login - afinal, se a autenticação foi feita com sucesso, deveríamos ter sido redirecionados. Conseguiremos a página em que estamos no momento com `browser.getCurrentUrl()`, e a compararemos usando `equals()` recebendo a URL da página de login (algo que poderíamos ter extraído para uma variável, por exemplo).

```
public void deveriaEfetuarLoginComDadosValidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chror
    WebDriver browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");
    browser.findElement(By.id("username")).sendKeys("fulano");
    browser.findElement(By.id("password")).sendKeys("pass");
    browser.findElement(By.id("login-form")).submit();

    Assert.assertFalse(browser.getCurrentUrl().equals("http://loc
}
```

COPIAR CÓDIGO

A próxima assertiva deverá verificar se o nome do usuário aparece no elemento do canto superior direito da tela, algo que precisaremos recuperar no HTML. Inspecionando a página, veremos que é uma tag `<span>` que não possui um ID. Note que esse elemento não está na página de leilões nem na página de login, mas sim em um template. Ou seja, sempre é necessário prestar atenção em como as páginas foram montadas no seu projeto.

Acessaremos o arquivo `base.html`, que funciona como um template geral das páginas HTML, incluindo o cabeçalho superior. Aqui buscaremos o `<span>` responsável por exibir o nome do usuário logado, e incluiremos o ID `usuario-logado`.

```
<span class="mt-3 mr-2">
    <span id="usuario-logado" sec:authorize="isAuthenticated()" s
    <a class="text-light" sec:authorize="!isAuthenticated()" hre
    <a onclick="document.querySelector('#form-login').submit()" c
    <form id="form-login" th:action="@{/logout}" method="post"></
</span>
```

COPIAR CÓDIGO

No teste, adicionaremos um `Assert.assertEquals()` que verifica se o texto `fulano` é igual ao conteúdo do elemento, que conseguiremos com `browser.findElement(By.Id("usuario-logado"))`. Como esse é um conteúdo de texto, podemos chamar o método `getText()` para acessá-lo. Por fim, podemos incluir um `browser.quit()` ao final do teste para fecharmos o navegador.

```
public void deveriaEfetuarLoginComDadosValidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chror
    WebDriver browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");
```

```
        browser.findElement(By.id("username")).sendKeys("fulano");
        browser.findElement(By.id("password")).sendKeys("pass");
        browser.findElement(By.id("login-form")).submit();

        Assert.assertFalse(browser.getCurrentUrl().equals("http://loc
        Assert.assertEquals("fulano", browser.findElement(By.id("usua
        browser.quit();
    }
```

COPIAR CÓDIGO

Com isso, finalizamos a construção do nosso teste, e ele reflete exatamente o passo-a-passo que fizemos manualmente na página. Ao rodarmos o código, o teste será executado e passará com sucesso.

Neste vídeo aprendemos a encontrar elementos na página utilizando alguns recursos, como o ID, a preencher formulários na página com o método `sendKeys()` e a enviar formulários usando o `submit()` ou algum outro tipo de input, além de realizar algumas assertivas para verificarmos se é possível encontrar o cenário esperado. Assim, concluímos nosso primeiro teste.

No próximo vídeo continuaremos testando outros cenários da funcionalidade de login.