04

Teste de login inválido

Transcrição

Agora que já testamos a funcionalidade de login com sucesso, testaremos o caso de tentar uma autenticação com um usuário ou senha incorretos. Começaremos simulando esse percurso no browser. Deslogaremos da aplicação clicando no botão "Sair", clicaremos no botão "Entrar" e preencheremos os campos com o usuário "invalido" e uma senha aleatória.

Ao clicarmos em "Login", permaneceremos na mesma página e será exibida uma mensagem de erro, "Usuário e senha inválidos". Esse é o cenário que desejamos testar no Selenium, portanto precisamos traduzi-lo para código.

Ainda na classe `LoginTest`, criaremos o método `naoDeveriaLogarComDadosInvalidos()` com a anotação `@Test` e incluindo todo o código do teste anterior - como serão bem parecidos, partiremos do mesmo esqueleto.

```
@Test
public void naoDeveriaLogarComDadosInvalidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chromed
    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://local
    Assert.assertEquals("fulano", browser.findElement(By.id("usuari
    browser.quit();
```

COPIAR CÓDIGO

Passaremos para as adaptações. O caminho inicial continua o mesmo: setamos o driver do navegador, abrimos o navegador, acessamos a página de login e preenchemos o campo `username`, mas desta vez com outro texto, `invalido`. Também alteraremos o preenchimento do campo `password` para uma senha qualquer, como `123123`.

Enviaremos o formulário, e em seguida faremos novas assertivas. Dessa vez, desejamos permanecer na página de login - portanto, ao invés de `assertFalse()`, verificaremos com `assertTrue()` se o browser se mantêm no endereço correto.

Além disso, precisamos verificar se a mensagem de erro é exibida na página. Para isso, precisaremos recuperar o elemento, algo que poderíamos fazer com o `findElement(By.Id())`. Porém, para conhecermos outros métodos da API do Selenium, vamos tentar outra forma.

Outra maneira de recuperar elementos em uma página é com o método `browser.getPageSource()`, que devolve uma string com todo o código fonte da página. A partir dele, podemos usar o `contains()` para verificarmos se a mensagem "Usuário e senha inválidos." está presente na página.

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

```
    browser.findElement(By.id("password")).sendKeys("123123");
    browser.findElement(By.id("login-form")).submit();

    Assert.assertTrue(browser.getCurrentUrl().equals("http://local
    Assert.assertTrue(browser.getPageSource().contains("Usuário e
    browser.quit();
}
```

COPIAR CÓDIGO

Além disso,, quando o usuário não loga, o seu nome não é exibido no `<span>` da página. Sendo assim, copiaremos o `Assert` do teste anterior, que buscava o nome do usuário nesse elemento. Dessa vez, podemos verificar se o conteúdo de `usuario-logado` é vazio.

```
public void naoDeveriaLogarComDadosInvalidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chror
    WebDriver browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");
    browser.findElement(By.id("username")).sendKeys("invalido");
    browser.findElement(By.id("password")).sendKeys("123123");
    browser.findElement(By.id("login-form")).submit();

    Assert.assertTrue(browser.getCurrentUrl().equals("http://loca
    Assert.assertTrue(browser.getPageSource().contains("Usuário e
    Assert.assertEquals("", browser.findElement(By.id("usuario-lo
    browser.quit();
}
```

COPIAR CÓDIGO

E está pronto o nosso novo teste! Perceba que, depois que fazemos o primeiro, os próximos saem bem mais rapidamente, já muitas vezes dependem somente de

adaptação. Recapitulando: estamos abrindo o navegador, acessando a página de login e preenchendo os campos com usuário e senha inválidos. Após enviarmos as informações, verificamos se a página atual é a própria página de login, se a mensagem de erro está sendo exibida e se o campo `usuario-logado` está vazio.

Ao executarmos o teste, entretanto, receberemos um `AssertionError`. Isso ocorre porque quando a autenticação falha, nós permanecemos na página de login, mas a URL é http://localhost:8080/login?error (http://localhost:8080/login?error) - ou seja, há a inclusão de um novo parâmetro. Como nosso `Assert` verifica se as URLs são iguais, ele retorna um erro.

Nesse caso temos duas opções: ou adicionamos o parâmetro na verificação, ou utilizamos, ao invés do `equals()`, o método `contains()` para verificarmos se parte da URL está correta. Opteramos pela inclusão do parâmetro.

```
public void naoDeveriaLogarComDadosInvalidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chror
    WebDriver browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");
    browser.findElement(By.id("username")).sendKeys("invalido");
    browser.findElement(By.id("password")).sendKeys("123123");
    browser.findElement(By.id("login-form")).submit();

    Assert.assertTrue(browser.getCurrentUrl().equals("http://loca
    Assert.assertTrue(browser.getPageSource().contains("Usuário e
    Assert.assertEquals("", browser.findElement(By.id("usuario-lo
    browser.quit();
}
```

COPIAR CÓDIGO

Entretanto, mesmo após essa correção, a execução do código continuará nos retornando um problema - dessa vez uma `NoSuchElementException`. Isso acontece porque, quando o usuário não loga, o elemento `usuario-logado` não aparece na página HTML (ao invés de ser exibido vazio). Quando chamamos o método `findElement()` do Selenium, ele devolve o elemento encontrado, mas lança uma exceção quando não o encontra.

Sendo assim, uma maneira de fazermos nosso teste é verificando se uma exceção está sendo lançada, algo que conseguiremos com o `assertThrows()`. Como a exceção é lançada diretamente pelo `findElement()`, podemos remover o `getText()`. Também precisamos passar qual exceção esperamos que seja lançada, nesse caso a `NoSuchElementException` do pacote `org.openqa.selenium`. Por fim, passaremos o lambda do Java 8 (`() ->`).

```
public void naoDeveriaLogarComDadosInvalidos() {
    System.setProperty("webdriver.chrome.driver", "/drivers/chror
    WebDriver browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");
    browser.findElement(By.id("username")).sendKeys("invalido");
    browser.findElement(By.id("password")).sendKeys("123123");
    browser.findElement(By.id("login-form")).submit();

    Assert.assertTrue(browser.getCurrentUrl().equals("http://loca
    Assert.assertTrue(browser.getPageSource().contains("Usuário e
    Assert.assertThrows(NoSuchElementException.class, () -> brows
    browser.quit();
}
```

COPIAR CÓDIGO

Feitas essas alterações, nosso teste passará com sucesso. Conseguimos testar o segundo cenário, de login inválido, que é bem parecido com o primeiro. Algum

diferenças foram o parâmetro adicionado à URL, a verificação da string direto no código fonte e a verificação de uma exceção lançada pelo próprio Selenium.

Antes de fecharmos essa aula, repare que temos bastante código duplicado em nossa classe `LoginTest`. Praticamente todos os métodos possuirão trechos de código repetidos, e que podem ser extraídos para melhorarmos a organização. Por exemplo, em todo teste precisamos setar o driver do Chrome, abrir o navegador e chamar o método `quit()`.

Podemos utilizar os recursos do JUnit para executarmos um método antes ou depois de cada teste, isolando os trechos de código que se repetem. Começaremos criando o método `beforeEach()` com a anotação `@BeforeEach`, de modo que o JUnit o execute antes de cada um dos métodos dessa classe. No corpo do método incluiremos as duas linhas que fazem a configuração do driver e abrem o navegador. Como desejamos utilizar a variável `browser` em mais de um método, vamos torná-la um atributo da classe.

```
public class LoginTest {

    private WebDriver browser;

    @BeforeEach
    public void beforeEach(){
        System.setProperty("webdriver.chrome.driver", "/drivers/c
        this.browser = new ChromeDriver();
    }
//...
```

COPIAR CÓDIGO

Em seguida, criaremos um método `afterEach()` com a anotação `@AfterEach`, fazendo com que o código seja executado após cada um dos métodos. No corpo,

incluiremos a chamada de `this.browser.quit()` para sempre fecharmos o navegador.

```
public class LoginTest {

    private WebDriver browser;

    @BeforeEach
    public void beforeEach(){
        System.setProperty("webdriver.chrome.driver", "/drivers/
        this.browser = new ChromeDriver();
    }

    @AfterEach
    public void afterEach(){
        this.browser.quit();
    }
```

COPIAR CÓDIGO

Agora não precisaremos abrir e fechar o navegador em cada um dos testes. Inclusive, não é necessário setarmos o driver em cada um dos métodos, sendo necessário fazê-lo só uma vez. Sendo assim, extrairemos esse trecho para outro método estático, `beforeAll()`, com a anotação `@BeforeAll`.

```
public class LoginTest {

    private WebDriver browser;

    @BeforeAll
    public static void beforeAll() {
        System.setProperty("webdriver.chrome.driver", "/driver
    }
```

```
    @BeforeEach
    public void beforeEach(){
        this.browser = new ChromeDriver();
    }

    @AfterEach
    public void afterEach(){
        this.browser.quit();
    }
```

COPIAR CÓDIGO

Como todos os métodos começam na página de login, também podemos passar essa etapa para o método `beforeEach()`.

```
@BeforeEach
public void beforeEach(){
    this.browser = new ChromeDriver();
    browser.navigate().to("http://localhost:8080/login");

}
```

COPIAR CÓDIGO

Por fim, a URL de login também se repete e pode ser extraída para uma constante privada que chamaremos de `URL_LOGIN`. Agora nosso código está mais organizado, e cada método de teste só possui os códigos respectivos aos seus cenários.

```
public class LoginTest {

    private static final String URL_LOGIN = "http://localhost:
    private WebDriver browser;
```

```
@BeforeAll
public static void beforeAll() {
    System.setProperty("webdriver.chrome.driver", "/drivers/c
}

@BeforeEach
public void beforeEach(){
    this.browser = new ChromeDriver();
    browser.navigate().to(URL_LOGIN);

}

@AfterEach
public void afterEach(){
    this.browser.quit();
}

@Test
public void deveriaEfetuarLoginComDadosValidos() {
    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(URL_LOG
    Assert.assertEquals("fulano", browser.findElement(By.id('
}

@Test
public void naoDeveriaLogarComDadosInvalidos() {
    browser.findElement(By.id("username")).sendKeys("invalid
    browser.findElement(By.id("password")).sendKeys("123123")
    browser.findElement(By.id("login-form")).submit();

    Assert.assertTrue(browser.getCurrentUrl().equals("http
```

```
        Assert.assertTrue(browser.getPageSource().contains("Usuár
        Assert.assertThrows(NoSuchElementException.class, () -> b
    }
}
```

COPIAR CÓDIGO

Para garantirmos que essa refatoração não incorreu em nenhum erro, executaremos novamente os testes. Todos eles passarão corretamente. Com isso concluímos o objetivo desta aula, que era testar um novo cenário (o login inválido), conhecendo outros métodos da API do Selenium, testar as exceções e começar a organizar nosso código de teste para evitar redundâncias.