06

Teste de URL restrita

Transcrição

Na última aula implementamos os testes referentes aos dois cenários de autenticação: login bem-sucedido com dados válidos e login inválido. Fechamos a aula organizando o código e deixando-o mais limpo, sem repetições. Nesse vídeo fecharemos os testes relacionados a login com um último cenário.

Quando um usuário faz login no sistema, são exibidos dois botões na tela de leilões: "editar" e "dar lance". Ao clicarmos em "dar lance", somos enviados a uma tela com os dados do leilão, incluindo a listagem de lances já efetuados. O endereço para essa página é http://localhost:8080/leiloes/2 (http://localhost:8080/leiloes/2), onde `2` representa o ID do leilão.

Entretanto, essa é uma página registra que só deverá ser acessada caso o usuário esteja autenticado no sistema. Ou seja, se fizermos logout e tentarmos acessar essa URL, o sistema deve detectar que não estamos logados e nos redirecionar para a página de login. Esse é o cenário que testaremos agora pensando nessa parte de autorização: o usuário não deve acessar um recurso restrito sem estar autenticado.

Voltaremos à classe `LoginTest` para criarmos esse teste. Começaremos construindo o método `naoDeveriaAcessarPaginaRestritaSemEstarLogado()` com a anotação `@Test`. Como anteriormente criamos alguns métodos que são executados antes e depois de cada teste, não precisaremos nos preocupar com setar o driver, abrir o browser ou mesmo fechá-lo.

Aqui temos um problema. Antes de cada teste estamos navegando para a página de login, mas neste em específico queremos navegar para outra página. Sendo assim, adicionaremos a linha responsável por essa navegação, desta vez para a URL http://localhost:8080/leiloes/2 (http://localhost:8080/leiloes/2).

```
@Test
public void naoDeveriaAcessarPaginaRestritaSemEstarLogado() {
    this.browser.navigate().to("http://localhost:8080/leiloes/2");
}
```

COPIAR CÓDIGO

Desta forma, o teste abrirá o navegador, entrará na tela de login e na sequência tentará acessar a tela de leilões sem ter se autenticado. A ideia é que ele não consiga acessar esta página, sendo redirecionado de volta para a tela de login. Portanto, faremos as assertivas do JUnit verificando se a página atual permanece sendo a de login com `assertTrue()`, da mesma maneira que fizemos antes.

```
@Test
public void naoDeveriaAcessarPaginaRestritaSemEstarLogado() {
    this.browser.navigate().to("http://localhost:8080/leiloes/2");
    Assert.assertTrue(browser.getCurrentUrl().equals("http://loca
}
```

COPIAR CÓDIGO

Para garantirmos que realmente não acessamos a tela de leilão, podemos verificar se o título "Dados do Leilão" não está presente no código fonte da página.

```
@Test
public void naoDeveriaAcessarPaginaRestritaSemEstarLogado() {
```

```
        this.browser.navigate().to("http://localhost:8080/leiloes/2");
        Assert.assertTrue(browser.getCurrentUrl().equals("http://loca
        Assert.assertFalse(browser.getPageSource().contains("Dados do
    }
```

COPIAR CÓDIGO

Ao executarmos, o teste passará corretamente. Com isso concluímos os testes da funcionalidade de login, incluindo a autenticação com dados válidos, a falha de autenticação por dados inválidos e a tentativa de acessar um recurso restrito sem autorização. Perceba que a API do Selenium simplifica bastante o desenvolvimento desses testes, além de ser bastante intuitiva e legível.

Na próxima aula vamos conhecer outros recursos que melhorarão a qualidade do teste e tornarão o código ainda mais fácil de entender.