07

Testando Permissões

Transcrição

Outro teste interessante de ser feito é verificar se a página de cadastro de produto em `/produtos/form` é realmente acessada apenas pelos administradores. Para tal tipo, é necessário configurar uma nova dependência no arquivo `pom.xml`. Esta dependência torna possível realizar testes no contexto do *Spring Security*. Adicinaremos a seguinte dependência em nosso projeto.

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <version>4.0.0.RELEASE</version>
    <scope>test</scope>
</dependency>
```

Após isso, podemos voltar para a classe `ProdutosControllerTest` e criar um novo método chamado `somenteAdminDeveAcessarProdutosForm` no qual realizaremos um novo *request* para o caminho `/produtos/form`.

```
@Test
public void somenteAdminDeveAcessarProdutosForm(){
    mockMvc.perform(MockMvcRequestBuilders.get("/produtos/form"));
}
```

Mas o teste não se resume a fazer uma requisição. Neste caso é necessário fazer uma tentativa de autenticação, assim, poderemos verificar se usuários diferentes dos administradores são redirecionados para outra página, que é a de login, neste caso.

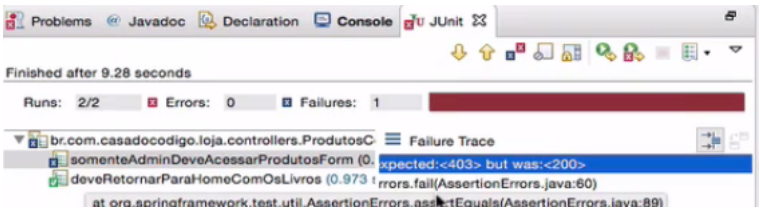
Para testar uma tentativa de autenticação, precisamos que o *Mock MVC* faça a requisição um *Post Processor*. Ou seja, um processo de *POST* antes de executar o *GET* da página, passando neste *Post Processor* os dados de autenticação do usuário. Observe o código abaixo:

```
@Test
public void somenteAdminDeveAcessarProdutosForm() throws Exception{
    mockMvc.perform(MockMvcRequestBuilders.get("/produtos/form")
            .with(SecurityMockMvcRequestPostProcessors
                .user("user@casadocodigo.com.br").password("123456")
                .roles("USUARIO")))
            .andExpect(MockMvcResultMatchers.status().is(403));
}
```

Note que para podermos usar o *Post Processor*, precisaremos fazer uso do método `with` que adiciona dados adicionais à requisição. A classe `SecurityMockMvcRequestPostProcessors` do *Spring Security Test* nos permite simular os dados de um usuário, com senha e *Role* configurados para a requisição. Por último, verificamos na resposta da requisição se o *status* da mesma foi um código 403 que significa um redirecionamento.

Observação: Caso o `roles` recebesse o valor `ADMIN` o teste falharia, pois usuários com este valor de *role* podem acessar o formulário de produtos. Para resolver este caso, o código de *status* da resposta, deveria ser posto com o valor **200**.

Mas algo estranho acontece ao tentar executar os testes. O teste de autenticação de usuários falha.



O esperado que era o código **403** não foi coorespondido. O sistema retornou o código **200**. O que aconteceu? Parece estranho, mas na verdade, cometemos um pequeno deslize.

Acontece que nas configurações da classe de testes, não especificamos as classes de configurações do *Spring Security*, desta forma, sem que este seja carregado, fica impossível de realizar os testes de segurança. Vamos resolver isso fazendo os seguintes passos.

Na anotação `@ContextConfiguration` da classe `ProdutosControllerTest` adicionaremos a classe de configuração do *Spring Security* `SecurityConfiguration.class` . Criaremos um noto atributo do tipo `Filter` que chamaremos de `springSecurityFilterChain` anotado com `@Autowired` e por fim, adicionaremos este filtro ao `mockMvc` através do método `AddFilter` .

```
@ContextConfiguration(classes = {JPAConfiguration.class, AppWebConfiguration.class, DataSourceC
@ActiveProfiles("test")
public class ProdutosControllerTest {

    // codigo suprimido

    @Autowired
    private Filter springSecurityFilterChain;

    @Before
    public void setup(){
        mockMvc = MockMvcBuilders.webAppContextSetup(wac).addFilter(springSecurityFilterChain).
    }

  // codigo suprimido

}
```

Agora, ao executarmos o teste novamente, este passa. Significando que o *Spring Security* foi carregado e fez a verificação de autenticação do usuário.