07

Implementando o Logout

Transcrição

Segue a dependência do **Thymeleaf Extra**, para adicionar no `pom.xml` :

```
<dependency>
    <groupId>org.thymeleaf.extras</groupId>
    <artifactId>thymeleaf-extras-
springsecurity5</artifactId>
</dependency>
```

COPIAR CÓDIGO

[00:00] Fizemos a nossa tela de login, esse é o HTML que representa a tela de login, fizemos na aula passada. Estamos acessando aqui a nossa página *home* e na hora que tentamos acessar, se o usuário não estiver logado, ele vem para essa tela de login e ele pode se logar.

[00:17] O que acontece? Precisamos só ajustar o menu para conseguirmos acessar tanto a página de login quanto a requisição para fazermos um logout. Ao me logar eu quero poder ter um logout.

[00:34] Por exemplo: digito aqui na tela de login usuário: "joao", e senha: "joao". Me loguei e vim para a página *home*. Continua esse "login" aqui em cima, à direita. Isso não está servindo de nada, vamos fazer isso realmente funcionar.

[00:44] É bem simples fazer isso, a princípio. Pelo menos o login é fácil, é um link para a página de login. Como vamos fazer isso? Onde esse "login" é criado? Ele fica nesse *fragment* que colocamos no nosso *template*, no "base.html". Eu vou abri-lo.

[01:02] Aqui no "base.html" ele tem esse *fragment* da logo, onde ele tem o "login". Agora, o que vamos fazer? Transformar esse login em um link, usando o `<a href="/Login">login</a>`. Fechei aqui, pronto.

[01:21] Se eu atualizar a página vejo que funciona, só que ele fica feio. Nós podemos melhorar um pouco o *layout* do login se adicionarmos uma classe do Bootstrap chamada de *text-light*. Então vou vir aqui, antes do `href`, criar um `class="text-light"`. Vou salvar. Vou abrir novamente, já ficou melhor. Pronto, está bom assim!

[01:56] Clico nele, ele vem para a página de login. Eu não preciso necessariamente agora, porque eu já estou logado - tanto que eu consigo acessar a página *home*. Só que quando eu faço o login, como é o momento agora, não deveria aparecer este link para login, mas deveria aparecer um link para logout.

[02:12] Então vou copiar e colar esse link que criamos, `<a class="text-light href="/Login">login</a>`, e ao invés de "Login" eu vou colocar "Logout" `<a class="text-light href="/Login">Logout</a>`. Vou atualizar a página. Os dois links estão aparecendono canto superior direito, tanto o "login" quanto o "logout".

[02:28] O que precisamos melhorar aqui é o seguinte: se eu já estou logado, eu não deveria ver mais a página de login, deveria ver a página de logout. E se eu ainda não estou logado, é só o login que deveria aparecer.

[02:40] Só que para isso acontecer eu preciso conseguir identificar e acessar de alguma forma o usuário. Se eu conseguir acessar o usuário, se o usuário que está fazendo a requisição estiver autenticado, eu mostro logout; senão eu mostro o login.

[02:55] Já existe um recurso legal para nós do próprio Thymeleaf. Eu vou abrir a documentação do Thymeleaf acessível neste link

(https://www.thymeleaf.org/documentation.html) e acessando a documentação, você vai rolando aqui, tem os tutoriais, usando o Thymeleaf.

[03:10] Aqui tem o Thymeleaf com o Spring, estendendo recursos do Thymeleaf e tem alguns artigos aqui embaixo, nessa área de artigos com Spring, "With Spring". Tem esse terceiro artigo, "Thymeleaf + Spring Security integration basics".

[03:28] Lá embaixo ele mostra um recurso bem legal do Thymeleaf, que é esse atributo `sec:authorize`, e ele tem essa invocação de método chamado `isAuthenticated()`. Aqui na explicação ele diz, o `sec:authorize` vai renderizar o conteúdo se o valor dessa chamada de método retornar *true*. Então se retornar verdadeiro, eu mostro esse conteúdo.

[03:54] O que eu vou fazer com isso? Vou copiar esse `sec:authorize="isAuthenticated()"`, vou no nosso código do "base.html" e vou fazer o seguinte: se o usuário já estiver autenticado, se `isAuthenticated` for verdadeiro, eu vou renderizar esse link com o link para o logout.

[04:14] Se não estiver, vou colocar uma exclamação para negar o resultado dessa chamada de método, vou negar isso. Então se não estiver autentificado, eu mostro o link de login. Vamos ver se isso funciona?

[04:30] Vou voltar para a nossa página e atualizar. Já atualizou o HTML, vamos dar uma olhada nesse "login", à direita, vou clicar com o botão direito e em "Inspecionar". Ele tem, o atributo `sec:authorize`, está lá. Só que ele não está sendo processado, esse atributo está sendo totalmente ignorado.

[04:48] Por que ele está sendo ignorado? Para conseguirmos utilizar esse recurso, que é um recurso extra do Spring, precisamos adicionar esses recursos extras. Aqui em cima dessa documentação, bem no início, ele mostra um repositório, ele tem um link para um GitHub com um exemplo de um projeto que usa esse recurso.

[05:08] Eu já vou direto no `pom`, que eu sei que precisamos adicionar uma dependência a mais para podermos utilizar esse recurso. E esse recurso está nas dependências dele, está aqui, é essa dependência, `thymeleaf.extras`.

[05:23] Vou copiar essa dependência, colocar no `pom` do nosso projeto, embaixo do que já colocamos aqui do Spring Security, vou apertar as teclas "Ctrl + Shift + F" para alinhar isso e deletar `version`. Deletei `version` e o `scope`.

[05:38] Por quê? Porque ele já vem no Spring Boot, eu só estou pedindo para o Spring Boot, porque ele está gerenciando a versão para nós, então só estou pedindo para ele adicionar isso no nosso projeto.

[05:46] Vou salvar, esperá-lo construir o *workspace* de novo, vou derrubar o projeto, subir de novo já com ele compilado e vou subir novamente. Eu quero ver se agora vai funcionar. Eu sei que ele realmente conseguiu importar essa dependência; senão estaria quebrado aqui no próprio `pom`, ele nos mostraria.

[06:07] A aplicação está subindo, eu já vou atualizar a página *home*. Abri a página de login, o usuário não está logado, porque eu acabei de subir o servidor e só está aparecendo o link para login.

[06:21] Se eu apertar aqui em "login", ele vai vir sempre para a página de login e o usuário realmente não está logado. Se eu tentar entrar na *home* ele volta para cá. E após eu me logar, o que acontece? Aparece "logout". Pronto, então está praticamente tudo funcionando!

[06:38] Só tem uma questão agora que temos que fazer funcionar, que é o link do logout. Então como o Spring Security sabe que ao receber uma requisição, quando a nossa aplicação receber uma requisição para logout, ele vai realmente *deslogar* o usuário? Como ele sabe fazer isso?

[06:54] Vamos voltar para o nosso "WebSecurityConfig.java", assim como configuramos o `formLogin`, dizendo que ao fazer a requisição para `/login`,

isso aqui é a página de login, tem que todo mundo permitir.

[07:07] Eu também tenho outro chamado de `.logout`. Esse logout eu posso customizar, então vou pegar esse aqui que ele me passa, `(logout -> logout.` e um dos atributos é o `logoutUrl`, então vou adicionar aqui `logoutUrl("/logout"));`.

[07:25] Ou seja, toda vez que o usuário fizer uma requisição para logout, o Spring Security, através dessa configuração que estamos fazendo, vai deslogar o usuário. Será que isso funciona? Vamos lá!

[07:36] Vou derrubar, subir de novo para ter certeza que o código está atualizado, rapidamente ele vai subir aqui. Eu já vou voltar para a aplicação. Pedi para ele atualizar, mas usuário não vai estar logado. Então vamos ter que logar novamente o usuário e clicar naquele link de logout. Vamos ver se vai funcionar?

[07:55] Então eu digito no login "joao" e "joao" (usuário e senha), me logo e aparece o link de logout. Faço a requisição de logout e ele continua não encontrando. Por quê?

[08:05] Vou voltar para o nosso projeto, no arquivo "WebSecurityConfig.java" e eu vou colocar o mouse bem no `logoutUrl`, porque ele normalmente tem um Javadoc legal. Olhe só o que ele está fazendo, está dizendo que é considerado uma boa prática usar *HTTP POST*. Se você não quiser usar *HTTP POST*, tem que usar outro método chamado de `logoutRequestMatcher`, passando essa sintaxe aqui.

[08:26] Então, por questão de segurança, o ideal é utilizar o POST, não o GET. Podemos fazer isso, podemos dar um jeito de ao invés desse link que colocamos aqui no "base.html", `<a class="text-light" sec:authorize="isAuthenticated()" href="/Logout">logout</a>`. Porque o link, automaticamente quando você clicar, ele fará uma requisição do tipo GET, n
do tipo POST.

[08:42] O que eu posso fazer é basicamente criar um `form` cuja `th:action`, vamos adicionar aqui, vai para `@{/Logout}`. E o método é igual a `post`.

[09:11] Por que eu fiz isso? Eu quero continuar utilizando um link simples. Se eu utilizar o formulário desse jeito, eu vou ter que fazer um *input, submit* e tudo mais. O estilo visual que o Thymeleaf dá para *input* não é esse estilo de link bonito. *Input* é assim, um botão. Só que eu quero continuar utilizando o link.

[09:37] O que eu vou fazer aqui? Eu vou invocar esse formulário, vou até fechá-lo aqui, dando um "id" para ele. Nós vamos chamar isso de `form-Login`. Eu vou utilizar esse ID e quando eu clicar nesse link, esse `href` não vai ser mais para `/Logout`. Vou colocar cerquilha `"#"` e aqui no início, só para ficar fácil de ver, eu vou colocar o `onclick`.

[10:11] Ou seja, ao clicar, ele não vai ter mais uma url para receber, ele vai executar essa ação aqui, `onclick`. Na verdade, ele executaria essa também, se eu tivesse colocado aqui outro link; mas eu coloquei cerquilha (`#`), então eu evito que esse link seja invocado para alguma página.

[10:28] No `onclick`, o que eu vou fazer? `document.querySelector()`, então eu vou buscar um elemento dentro dessa página chamado `#form-login` e vou invocar o `submit()`. Isso aqui é JavaScript puro, eu posso fazer isso. Não estou utilizando nenhum recurso de Thymeleaf nem de nada, isso aqui é JavaScript mesmo para o HTML.

[10:57] Vamos atualizar a página e veremos se isso funciona. Atualizo, vamos ver se o `form-Logout` já está com o link. Beleza!
`onclick="document.querySelector` busca um comando chamado `#form-login` - que é exatamente esse formulário, cuja `action` vai para `/logout`, que é exatamente onde queremos. Como ele vai usar o método *post*, vamos conseguir invocar corretamente. Vou clicar aqui, "logout". E deu certo!

[11:19] Pena que nosso *layout* está quebrado, por algum motivo ele ficou tosco. O ideal é que esse `<form id="form-Login" th:action="@(/Logout)" method+"post"/>` não fosse visível, talvez o fato de eu estar fechando aqui, ele tenha ficado esquisito. É isso mesmo.

[11:36] Se eu tentar acessar a *home*, eu não consigo, porque eu já desloguei. Realmente funcionou o logout. Se eu tentar me logar, apertar aqui o "login", aparece o "logout" aqui. Então aquele atributo `sec:authorize` está funcionando.

[11:48] Se eu clicar no "logout", veja que o link que está aparecendo no canto inferior da tela é `/home#`, ou seja, ele não vai chamar nenhuma página, ele vai ficar essa página, só que ele vai executar a ação do `onclick`, que é exatamente o *submit* do formulário. Eu deslogo o usuário. E realmente, para confirmar de novo, eu desloguei - tanto que eu vim para essa página de login.

[12:11] Então é isso! Montamos nossa tela de login, com nosso menu funcionando para a página de login e para a página de logout, o logout funcionando. Até o próximo vídeo!