05

Página de Login

Transcrição

> **Atenção:** neste vídeo, ocorre um erro `403`, que acontece quando se tenta adicionar um novo pedido. O erro é resolvido no terceiro vídeo da próxima aula, "Provedor de autenticação (https://cursos.alura.com.br/course/spring-mvc-security-rest-vuejs-ajax/task/81134)".

[00:00] Continuando. Fizemos a instalação inicial da página de *login*, só que precisamos ter a página de *logout* também, então vamos fazer duas evoluções aqui: vamos fazer o formulário de *login* e ao invés de trabalharmos com `httpBasic`, vamos trabalhar com o formulário de login mesmo.

[00:25] E ao invés de deixarmos o usuário logado aqui, sem nada, vamos colocar um menu aqui em cima, à direita, onde aparece "login" caso ele não esteja logado, e colocar "logout" caso ele já esteja logado e queira se deslogar.

[00:40] Vou voltar aqui nas configurações. Nós já estávamos aqui nas referências, nós abrimos o *guide* aqui para usarmos como exemplo inicial, mas podemos evoluir a partir da própria referência mesmo. Na própria documentação do Spring Security fomos atrás do "Basic Authentication", no item 10.10.2, só que logo acima ele tem o "Form Login".

[01:09] E como já fizemos a configuração inicial, já conseguimos continuar a implementação a partir daqui. Veja que nesse "Example 51. Form Log In", ao invés do método `configure()`, que já vimos, temos o método `configure()` implementado usando o `httpBasic`.

[01:26] Só que ao invés de usar `httpBasic`, ele está utilizando o `formLogin` com as configurações *default*. Só que logo ele demonstra um detalhamento melhor usando uma página de login chamada `/login`. Aqui embaixo ele mostra um exemplo do que é uma página "login".

[01:44] A primeira coisa que eu vou fazer é copiar isso aqui e explicar com detalhes onde colocar e do que se trata. Esse `.formLogin` nos permite dizer qual é a *url* da página de login. Então página de login vai estar em `/login`.

[02:03] Depois nós configuramos dizendo que todo mundo é permitido acessar essa página de login, porque realmente o usuário não tem que estar autenticado quando ele vai fazer o acesso à página de login, senão ele não consegue nem se autenticar na aplicação.

[02:15] Logo abaixo ele coloca um exemplo do formulário em HTML, utilizando o Thymeleaf mesmo, o Thymeleaf *template* que produz aqui a tela de login.

[02:28] E o que ele tem de principal? Ele tem um formulário onde ele tem três *inputs*, sendo que o último é o *input* de `submit`. Então isso aqui é um formulário para `/login`, usando o método `post`. Ele tem um *input* do tipo texto com o `username` e outro *input* do tipo `password` com o *name* `Password` mesmo.

[02:56] Nós vamos fazer a mesma coisa, vamos usar esses *inputs* como modelo, pelo menos para entendermos quais são os *inputs* que temos que colocar. Só que eu vou utilizar para fazer essa `.loginPage("/login")`, dentro de "templates". Veja que na própria documentação ele põe o caminho também, `src/main/resources/templates/login.html`.

[03:17] Vamos fazer exatamente igual, só que eu vou utilizar a página *home* como modelo. Antes de mais nada eu vou deletar esse "HelloController.java", porque isso foi do início, não precisamos mais dele para nada. Tiramos o que é inútil. Eu copiei o `/home`, vou colar aqui na pasta, vou chamá-lo de "login.html" e vou tirar as coisas que não são necessárias.

[03:47] Aqui em cima temos o `head`, que tem aqueles *imports* que usamos, tem a `logo`, logo em seguida tem "Meus Pedidos", que é o título. Ao invés de chamar de "Meus Pedidos" vamos chamar de "Login".

[04:02] Logo abaixo ele tem o menu de navegação.

[04:06] Esse menu de navegação corresponde à barra horizontal na parte de cima em que se lê "Todos", "Aguardando", "Aprovado", "Entregue", "Novo".

[04:10] E esse menu de navegação não vai ter na página de login, então eu vou tirá-lo.

[04:15] E aqui no "card", onde tem o conteúdo principal da aplicação, ele tem um monte de coisas que não vamos usar, esse topo aqui, não vamos usar. Esse `card-body` também não vou utilizar. Podemos colocar tudo aqui direto mesmo, no formulário direto.

[04:35] Vou tirar esse `th:each`, que eu também não espero que tenha pedido, não vou também preencher nada de `pedidos` aqui e vou fazer o que ele está colocando ali na documentação. Então para isso podemos, sem problema nenhum, copiar o `form`. Vou fechar o formulário aqui embaixo, `form` para login.

[04:56] Ele tem alguns *inputs,* só que aqui ele não segue o nosso padrão de colocar um *label* e tal, então eu vou pescar aqui do "formulário.html" que já temos. Temos `label` e o `input`. Então vou copiá-lo e vou colar na minha página de login, simplificando.

[05:15] Então ele precisa da classe `form-control`, o `nome do produto` na verdade é nome de usuário, `placeholder` é o `usuário`. Vamos deixar assim. `th:field` - não temos essa questão aqui, então vamos escrever só o `name` mesmo. Isso vai ficar igual, `username`. Esse é um *input*.

[05:41] Os nossos *inputs*, se você olhar aqui, eu não copiei, eles ficam dentro de um `form-group`.

[05:49] Então isso eu preciso copiar também e colocar o nosso *input* dentro de um `form-group`. Vou fechá-lo logo em seguida e vou envolver esses dois aqui, `label` e `input`.

[06:10] Vamos ajustar aqui também o `label`: `<label for="username">Usuário</label>`. Vamos agora colocar aqui `<label for="password">Produto</label>`. Então: `<input type="password" name="password" class="form-control" placeholder="usuário"/>`.

[06:32] Podemos deixar aqui `label for="password">Password</label>` ou `label for="password">Senha</label>`. Estamos trabalhando com o português. Vamos colocar `"senha"` aqui também. Pronto, eu vou salvar!

[06:43] Então já temos a tela de login, só que já sabemos que se quisermos abrir uma tela - inclusive, ele até indica aqui na própria referência. Ele diz: "crie um *controller* para poder abrir a tela de login".

[07:00] O que o Spring Security faz é cuidar do `post` para `/Login`, então se você fizer uma requisição *post* para login passando dois atributos (`username` e `password`), isso ele resolve. Agora o *get*, que é para mostrar essa tela, ele não vai resolver.

[07:14] Então vou criar aqui uma classe *controller* chamada de `LoginController`. Vou digitar `@Controller`. Vou fazer um *get* para ele: `@GetMapping` e `@RequestMapping("/login")`. E aqui eu só crio o método `public String login() { . return "login";` - representando o nome da página mesmo, que é essa página de login que nós temos agora.

[08:01] Pelo menos a página de login já vamos conseguir acessar e já fizemos a configuração do formulário de login. Então, devemos pelo menos conseguir abrir o formulário de login.

[08:12] Será que vai precisar de mais alguma coisa? Vamos olhar na referênciada documentação. Diz que precisamos do formulário de login, esse formulário eu acabei de lembrar que eu acho que eu não criei. É esse aqui: `<button class="btn btn-primary" type="submit">Cadastrar</button>`, esse não copiamos. É importante ter o *input* de `submit`. Vamos lá, `<button class="btn btn-primary" type="submit">Login</button>`.

[08:38] Nós vamos testar agora, porque já fizemos a tela de login, o usuário já tem os *inputs* para preencher usuário e senha, o Spring Security já sabe receber um *post* para `/Login`, ele sabe ler os atributos *username* e *password* e ele vai fazer a autenticação como se fosse o `httpBasic`, a mesma funcionalidade agora.

[08:58] Eu vou subir a aplicação e nós vamos ver se ele vai funcionar. A ideia é que agora esteja tudo ok e que não tenhamos mais nenhum problema com relação a isso. Podemos ver na aba Console que deu problema, vamos ver. "EntityManager is closed". Ele já está rodando, eu esqueci de derrubar esse aqui.

[09:34] Beleza, subiu na porta 8080, inicializou. 11 segundos, foi rápido. Vou abrir a página "login" novamente. Ele está reclamando acentuação, nós adicionamos no base o meta UTF-8, já está aqui, mas ele não pegou. Vamos adicionar de novo, no "login.html", não custa nada.

[10:03] Pronto, agora usuário é "usuário" e a senha é "senha". Ele ainda está desformatado aqui, os campos estão muito perto um do outro. Eu não sei se eu perdi alguma coisa na hora de copiar. Deixe-me voltar para o código da página "login.html".

[10:27] Vamos lá. `<div class="card mb-3">` está dentro de um *container*, ele consegue adicionar o *head*. Vamos só conferir. Então ele está conseguindo implementar tudo.

[10:40] Dei um *replace*, esse aqui é um `class="container"` mesmo, com o título de `Login`. `<div class="form-group">` é um *input*. Isso aqui é um *form*. Na verdade, faltou o `class="card-body"` no `form`, então fica: `<form th:action="@{/login}" class="card-body" method="post">`.

[11:17] Vamos voltar para a tela, ver se ele atualiza e fica legal. Beleza, está respeitando as margens, agora está legal. Se eu tentar acessar "/home", será que eu consigo? Ele está me redirecionando já para a tela de login, ótimo! Eu vou preencher aqui no usuário e senha, `joao`, `joao`, e vou me logar. Deu certo, ele entrou para a página *home*.

[11:39] Nós já conseguimos nos logar usando uma página bonita mesmo, de login, ao invés daquele *pop-up* que estava abrindo do navegador, mas precisamos agora ajustarmos para termos aqui um link para login e logout. Vamos dar uma ajustada nisso para conseguirmos logar e deslogar o usuário corretamente, antes de passarmos a fazer as pesquisas de pedidos e tal, do usuário que está logado.

[12:05] Depois vamos aprender como fazemos para identificarmos no nosso código quem é o usuário, qual é o nome do usuário, como buscar informações daquele usuário específico e tudo mais. Vamos evoluir nossa aplicação agora para lidarmos com dados específicos do usuário. Até a próxima aula!