04

Associando o usuário ao pedido

Transcrição

[00:00] Já temos uma requisição para salvarmos um pedido novo do usuário, só que esse pedido tem um relacionamento novo agora - que é com o usuário. Esse usuário é o usuário logado aqui. Então quando recebemos uma requisição para criar um novo pedido, precisamos dar um jeito de descobrir quem é o usuário que está logado, que está fazendo essa requisição.

[00:26] E o Spring tem uma classe chamada de `SecurityContextHolder`, que nos dá essa informação através desse `getAuthentication()`. Então `getContext` de segurança, `getAuthentication`, os dados do usuário mesmo e você pode pedir, através do método `getName()` o *username* do usuário. Vou criar até uma variável aqui chamada de `username`.

[00:53] O que eu vou fazer com isso? Eu preciso passar esse `username` ao `Pedido`, então vou fazer `pedido.setUser(username);`, só que `username` não existe.

[01:10] Se você olhar no pedido, esse `setUser` e `getUser` se referem a esse `private User` aqui, que é um objeto, não `username`. Então precisamos buscar o usuário associado a esse `username` e isso nós fazemos indo no banco de dados, indo na tabela "user" do banco.

[01:31] Para acessarmos o banco de dados, já sabemos que precisamos de um *repository*, então vou criar aqui um `UserRepository`. Basicamente, nós vamos copiar essas informações que já colocamos aqui, no `PedidoRepository`, vamos estender aqui JPA também.

[01:55] Eu vou só criar aqui um método que retorna `User` chamado `findByUsername`, que recebe uma `(String username);`. Ele já vai entender isso daqui, só precisamos alterar o `public interface UserRepository extends JpaRepository<Pedido, Long>{` para `public interface UserRepository extends JpaRepository< User, Long>{`. E eu vou apertar as teclas "Ctrl + Shift + O" aqui para importarmos esse *user*.

[02:13] E se você olhar, esse `Long` se refere ao tipo do ID. Nós sabemos que o ID é o *username* e o tipo dele é *string*, então vou colocar `public interface UserRepository extends JpaRepository< User, String>{` ao invés de `public interface UserRepository extends JpaRepository< User, Long>{`. Então já temos o `UserRepository` configurado corretamente e eu posso simplesmente injetar ele aqui.

[02:30] Então eu digito `private UserRepository userRepository;` e dou um `@Autowired` nele. Pronto, agora só precisamos fazer o `findByUsername(username);` do usuário logado, pegando o `user` aqui e associando esse `user` ao pedido! Agora só precisamos salvar e ele já vai estar salvo com o usuário logado.

[02:59] Outra coisa que podemos fazer agora, que é bem interessante, é a seguinte: no `form` tem uma `action` chamada de `/pedido/novo`. Isso é bem ok, isso está funcionando, mas imagine o seguinte: estamos rodando nossa aplicação em `localhost:8080`, se essa *action* está `/pedido/novo` significa que a requisição que ele vai fazer é essa, `localhost:8080/pedido/novo`.

[03:23] Só que existe uma coisa chamada contexto da aplicação - que podemos, por exemplo, mudar aqui, adicionar um contexto e dizer que o contexto dela vai ser `/mudi`.

[03:35] Se eu quiser criar um novo pedido eu tenho que fazer `/mudi/pedido/novo`, só que essa requisição não vai fazer isso. Esse contexto é configurável no servidor, podemos mudar esses contextos aqui e tal...

[03:50] E toda vez que mudarmos de contexto, temos que lembrar de colocar aqui, em todos os links. Obviamente não precisamos fazer isso. Dinamicamente o Thymeleaf pode preencher o contexto para nós, se utilizarmos essa sintaxe aqui.

[04:05] Você usa o `th:action`, que é o *action* do Thymeleaf, com a inteligência dada pelo Thymeleaf, usa esse arroba e coloca o *path* dentro do arroba. E se a aplicação tiver um contexto, ele vai adicionar o contexto aqui automaticamente, na hora da requisição.

[04:22] Outro problema aqui é o seguinte: se eu fizer uma requisição para cadastrar e a minha aplicação estiver rodando, eu vou tomar um erro 403, um *forbidden*. Por que isso acontece? Porque automaticamente o Spring habilita uma funcionalidade de segurança chamada de CSRF, que não vamos entrar em detalhes aqui.

[04:43] E eu vou só desabilitar essa funcionalidade vindo na configuração do `WebSecurityConfig` e adicionando aqui uma informação que é `.csrf().disable();`, para simplificar, porque isso não é contexto desse treinamento especificamente. Vamos desabilitá-lo. Mas é um recurso bem legal também, vale a pena dar uma estudada na documentação.

[05:09] Se eu tentar fazer a requisição agora, eu vou conseguir. Eu vejo que a requisição bateu na aplicação, a validação foi executada e tudo mais. Agora nós já associamos o usuário logado ao pedido, e os pedidos agora vão ser salvos com o usuário associado.

[05:31] Então é isso, até o próximo vídeo!