03

Mostrando os pedidos do usuário

Transcrição

[00:00] Temos a nossa tela de login, onde ele busca os dados dos usuários no banco de dados e agora precisamos fazer a ligação entre esse login que estamos criando agora com os pedidos que os usuários criaram ou vão criar posteriormente.

[00:18] Já temos aqui o mapeamento para pedidos e temos uma tabela de pedidos, mas também temos a tabela de usuário e não temos o mapeamento do usuário. Precisamos do mapeamento do usuário para podermos fazer a ligação entre pedido e usuário.

[00:37] Eu vou começar criando aqui uma classe chamada de "User" e eu vou dizer que ela é uma "@Entity" e mapeá-la com um nome de tabela específico - que tem que ser o mesmo nome da tabela que estamos usando aqui, porque usamos aquele *script* que pegamos na documentação. Então o nome da tabela é "users". `@Table(name="users")` .

[01:01] Essa tabela tem três colunas: `username` , `password` e tem um Boolean chamado `enabled` . Então é `username` , `password` e `enabled` , para dizer se o usuário está habilitado ou não - porque ele pode estar desabilitado. Estou usando Lombok aqui e vou criar os três. Beleza, foram criados!

[01:30] Agora temos que fazer o mapeamento de usuário para pedidos. Então o usuário, qual o relacionamento no banco de dados que ele vai ter com pedidos? Um usuário tem vários pedidos, então `private List<Pedido> pedidos;` e aqui nós colocamos o `@OneToMany` .

[01:54] Nesse mapeamento que vamos fazer, `OneToMany`, eu vou adicionar aqui, sem me preocupar com muitos detalhes, algumas coisas que eu quero colocar. Eu vou dizer que ele vai ser mapeado pelo "user", que eu vou mapear na tabela de pedido, senão ele acaba criando outra tabela. E vou digitar `fetch = FetchType.LAZY)` também, para se eu buscar o pedido, ele não carregar os pedidos.

[02:22] Lá na tabela "Pedido.java" eu vou fazer o mapeamento também. Vou criar aqui um "user" só, então o pedido tem um usuário, e vou colocar. Nesse caso, muitos pedidos vão ter apenas um usuário, então na tabela "Pedido.java" vamos ter um `@ManyToOne`.

[02:46] E aqui eu não vou colocar muita coisa, só vou dizer que ele vai ser `LAZY` também, porque eu posso querer só listar pedidos, então deixo `LAZY` para ele não carregar os dados do usuário quando eu fizer um *select* com JPA - que ele pode acabar fazendo isso automaticamente.

[03:03] A aplicação já subiu e está dando erro. Ele está dizendo que não tem o identificador específico para *user*, e realmente não existe na tabela, mas eu não coloquei aqui no JPA. Vou salvar aqui, vê-lo subindo de novo. Agora deve funcionar.

[03:21] E se olharmos aqui na tabela "pedido", ele já foi incrementado com a coluna chamada "user_username", que já tem uma *foreign key* associada para ele com a tabela de *user*, onde o ID dele. A *primary key* dele é "username". Vai ficar assim mesmo, simplifica, fica até mais fácil de ver.

[03:41] E o que precisamos fazer é o seguinte: na hora em que abrimos a página *home*, vou até me logar, vou colocar aqui usuário e senha, "joao", "joao" e vou logar. Os pedidos que aparecem aqui, apesar de só ter um pedido no banco de dados, podemos até olhar os dados aqui, só tem um, mas ele está buscando todos os pedidos.

[04:08] Ele não deveria estar buscando todos os pedidos, deveria estar buscando só o pedido do usuário logado, que é o João. De fato, foi ele quem criou, foi com esse usuário. Na verdade, na época nem tinha o usuário, obviamente. Então o "username" aqui está vazio, então esse pedido aqui não pertence à usuário nenhum. Ele não deveria nem estar aparecendo aqui.

[04:28] O que vou fazer é alterar esse *select* no "HomeController.java" aqui para não fazer um `findAll` vazio. Vou colocar aqui um `findAllByUsuario`. Esse aqui não existe ainda, não é tão simples fazer essas queries assim. Então toda vez que precisarmos criar uma query manualmente mesmo, existe uma anotação chamada de `@Query` do Spring Data JPA.

[04:53] E dentro de `@Query` podemos escrever a query que queremos. Então, como eu quero fazer o *select* de pedido só que de um determinado *username*? Eu vou até adicionar um parâmetro nesse aqui: `findAllByUsuario` e vou receber uma `(String username);`.

[05:18] Eu vou mostrar para vocês que ainda falta mais uma coisa aqui, mas vou mostrar o porquê disso. Então vamos fazer aqui um `@Query("select de p from Pedido p join p.user u where u.username = :username")`.

[06:08] Então como ele vai fazer, na hora em que estivermos chamando esse método? Se eu estiver, por exemplo, recebendo vários parâmetros, aqui que só tem um é fácil; mas se eu estivesse recebendo vários parâmetros, como o Spring Data iria conseguir mapear esse parâmetro "username" com esse aqui, "username"?

[06:25] Nós utilizamos uma anotação chamada `@Param` do Spring Data também. E só passamos o nome, "username", o nome desse aqui. Agora ele sabe mapear esse "username" para esse parâmetro. Vou salvar esse *select*. Aqui vai dar erro agora. O que acontece? Qual é o nome do usuário logado?

[06:52] Para fazermos isso existe uma maneira bem simples, eu vou mostrar outra depois, com uma classe chamada de SecurityContextHolder. Mas existe

um código chamado de `Principal`, do próprio Java Security, não é nada do Hibernate, só que é integrado, porque é um padrão JAAS, `Principal principal`. E o que acontece? Eu peço para o `Principal` o nome do usuário.

[07:18] Quando você adiciona ele aqui, automaticamente, como é o Spring que sabe mapear uma requisição para um determinado método e tudo mais, e já sabemos que podemos pedir coisas e ele injeta para nós; se você pedir o principal, ele vai te injetar os dados do usuário logado.

[07:34] Então com `Principal`, eu não sei se abrirmos vamos ver muita coisa aqui, acho que não. Mas com o `Principal` nós pegamos os dados do usuário, temos as *rows* também, conseguimos pegar informações do usuário com esse aqui.

[07:48] Vou salvar esse "Controller", ele vai recarregar para mim e eu vou tentar fazer a requisição. Ele não deveria mostrar nada aqui, porque esse pedido especificamente não está associado ao usuário logado. Beleza, funcionou! O *select* realmente funcionou.

[08:05] O que eu vou fazer agora? Qual é o nome do usuário que eu me loguei? É João, esse é o ID dele. Vou só copiar esse valor de ID, vou jogar no "username". Ele me permite associar. Eu acho que salvou. Vou carregar de novo essa página. Pronto, agora eu consigo recuperar os dados do usuário!

[08:30] Já fizemos a associação no *select*, estamos buscando aqui os dados do usuário logado. Temos que também corrigir um problema, que é fazer essa busca por *status*, então tem que ser *status* e o usuário logado.

[08:48] E o mais importante de tudo é no PedidoController, na hora em que ele vai criar um novo pedido. Eu estou criando um pedido, salvando esse pedido e não estou associando esse pedido com o usuário logado. Isso nós vamos ver na próxima aula, até lá!