07

Guardando ofertas

Transcrição

[00:00] Nós temos a tela de listagem de ofertas e o nosso objetivo agora é realizar as ofertas. Eu estava me logando aqui antes de começar a gravar o vídeo e vi que essa tela estava saltando e eu percebi o porquê de precisarmos colocar a margem. Eu fiz isso no vídeo anterior.

[00:16] Eu não tinha notado essa barra aqui embaixo e não notei esse detalhe no nosso *template*, no "base.html". Vamos aqui na *home*. Toda vez que criamos uma *row*, que é o cara daquela diagramação, do Bootstrap, essas *rows* ficam dentro de *containers* e se comportam direitinho quando seguimos a documentação.

[00:41] Essa classe *row* que colocamos no *template*, tanto na parte da logo, que é essa parte azul, quanto na parte do menu que nós criamos, de `pedidos` e `faça sua oferta`. Essa *row* que eu acabei de deletar que estava dando problema. Realmente não deveríamos ter colocado ali. E agora isso vai resolver esse salto que ele estava dando na página de login, até aumentou a margem, mas vai ficar assim.

[01:06] Vou me logar na aplicação, para sermos bem objetivos, o nosso objetivo aqui é criarmos as nossas ofertas. Como vamos criar as ofertas e que tipo de informação temos que guardar com relação às ofertas?

[01:21] O que acontece? Aqui no nosso `PedidoRest` que criamos para essa tela mesmo, criamos um *endpoint* para buscarmos pedidos aguardando ofertas, e agora o que vamos fazer é gerar e lançar ofertas.

[01:37] Não vou fazer isso dentro do `PedidoRest`, mas eu vou criar outra classe chamada de `OfertasRest` - e já vamos começar inserindo anotações `@RestController` e `@RequestMapping`, seguindo o mesmo padrão da outra. `/api/ofertas`. Apertamos as teclas "Ctrl + Shift + O" para importarmos o outro.

[02:01] Vamos criar um método que ao ser chamado gera uma oferta, o nome dele é `criaOferta()`. Ele vai receber os dados que queremos da oferta, vamos receber aqui uma `(RequisicaoNovaOferta` chamada de `requisicao) {`.

[02:23] E qual o objetivo? O que vem dentro desse `RequisicaoNovaOferta`? Vamos criar essa classe `RequisicaoNovaOferta.java`. Eu vou colocar dentro de DTO mesmo, serve para receber os dados de requisição.

[02:36] Eu vou criar aqui alguns atributos que precisamos, um eu vou chamar de `Long` mesmo, vai ser o `pedidoID;`. Porque quando vamos criar uma oferta, vamos criar uma oferta para um determinado pedido, então precisamos pelo menos do ID dele, uma referência para o próprio pedido.

[02:52] Outra coisa que eu vou colocar aqui é a seguinte: quando eu vou fazer uma oferta, a oferta eu vou preencher o valor e a data de entrega. Vamos criar mais um campo também. Temos esse *text area* que estamos imprimindo aqui a descrição, mas não vamos imprimir descrição no *text area*, na verdade vamos escrever um comentário aqui.

[03:09] O usuário que vai fazer uma oferta também vai poder fazer comentário. Óbvio que ele vai poder ver a descrição, mas não aqui no *text area*, vamos alterar isso.

[03:18] Vamos colocar valor, data e um comentário, então `private String valor;`. Vou deixar como `String`, porque vamos fazer uma validação mais para frente e é importante que consigamos receber qualquer tipo de valor, senão ele dá um erro específico ali no atributo. `private String dataDaEntrega;` e `private String comentario;`. Vou criar aqui os *getters* e *setters* e é isso.

[03:48] Esse é o objeto que vamos receber, ou seja, dado uma requisição POST que vamos receber aqui, o Spring vai pegar todos os parâmetros da requisição. Nessa requisição POST vamos passar parâmetros que têm esses dados que o usuário preencheu, mais o ID do pedido, que representa essa área aqui, e vamos mapear para esse objeto.

[04:12] Só que os parâmetros têm que bater com o nome certinho aqui. Você vai ver, mais para frente, quando fizermos a tela, vamos ter que voltar aqui para ver se os nomes são os mesmos.

[04:21] E depois que recebermos uma requisição de nova oferta, vamos salvar essa requisição de nova oferta no banco de dados. Então vou criar a classe `Oferta` dentro do pacote "model", vou chamar esse aqui de `@Entity` e vou criar aqui `private Long id;` aqui com os valores corretos. O valor vai ser `BigDecimal valor`. O `private LocalDate` para `dataDaEntrega;`, e `private String comentario;`. Aqui está com todos os formatos certos.

[05:03] Vou gerar os *getters* e *setters* e vou apenas colocar as anotações que precisamos. `@Id`, essa é obrigatória, e a `@GeneratedValue` é importante porque esse ID vai ser gerado automaticamente, então a `strategy` dele aqui vai ser `IDENTITY`.

[05:17] O que mais vamos fazer com relação à oferta? As ofertas são criadas em cima de um pedido, então podemos criar já a associação entre a oferta e o pedido. E o que acontece? Quantas ofertas para um pedido? Muitas, então `@ManyToOne`. Eu vou só colocar aqui um `(fetch = FetchType.LAZY)`, para ele não carregar automaticamente quando buscar dados de ofertas e não carregar os dados do pedido também.

[05:50] Só que ao mesmo tempo em que eu tenho o mapeamento de oferta com pedido, eu vou fazer o mapeamento do pedido para a oferta. Quantas ofertas tem para o pedido? Muitas, então um pedido tem uma lista de ofertas. Pode ter nenhuma, pode ter várias. E o mapeamento vai ser `@OneToMany`.

[06:19] Eu vou preencher o `cascade` como `CascadeType.ALL,` . Eu posso ter vários aqui, mas quando eu vou salvar, se eu adicionei uma oferta, ele vai salvar a oferta também. `mappedBy =` , eu vou mapear pelo atributo dentro de oferta, que é `"pedido",` . E o `fetch` vai ser `FetchType.LAZY)` também. Então quando eu buscar um pedido, ele não vai automaticamente carregar também no banco as ofertas.

[06:50] Imagine que eu faço um *find* pedido por ID e ele tem 30 ofertas. Eu não vou buscar essas 30 ofertas no banco automaticamente, só se eu realmente pedir para isso acontecer, aí os dados das ofertas vêm.

[07:05] Já temos o mapeamento e agora, o que precisamos fazer? Já que eu recebi uma requisição POST, recebendo os dados da oferta nessa requisição, eu vou converter essa requisição para oferta.

[07:20] E, basicamente, a primeira coisa que eu preciso fazer antes dessa conversão é buscar o pedido no banco de dados. Vou chamar de `pedidoRepository;` , `@Autowired` . Eu vou fazer `pedidoRepository.findById` e vou passar aqui da `(requisicao.` .

[07:48] Lembra que quando eu construí a requisição eu criei o `PedidoId` ? Porque é isso. Toda oferta que fazemos é em cima de um pedido, por isso que eu estou pedindo o dado do ID do pedido. Quando construirmos a interface você vai ver de onde vai vir esse `PedidoId` .

[08:05] E aí sim eu vou ter aqui um pedido, então vou dar um *new local variable* `pedidoBuscado` . Ele veio como *optional,* então pode ser que ele simplesmente não encontre o pedido. E se ele não tiver encontrado, ou seja, se não estiver presente isso, eu vou retornar aqui alguma coisa. Mais para frente nós mudamos isso para algum erro, vou deixar `return null;` .

[08:36] E aqui embaixo nós vamos pegar essa requisição e transformá-la em uma oferta que eu vou chamar de `nova` , então `requisicao.toOferta()` . Esse

`toOferta` é basicamente um método dentro do `RequisicaoNovaOferta`, que tem como objetivo criar um objeto do tipo `Oferta`.

[08:56] E temos que tomar uma atenção aqui, que é a seguinte: quando eu for criar a oferta, por exemplo, vou setar o comentário. Aqui na requisição eu recebi o comentário do usuário. Eu também tenho que preencher a data da entrega, `setDataDaEntrega`.

[09:22] Vou pegar aqui `(this.dataDaEntrega);`. Só que a data da entrega aqui do objeto da requisição está *string* e eu preciso converter isso para um objeto do tipo `LocalDate`, porque é como mapeamos.

[09:36] Então fazemos aqui, antes do `this.dataDaEntrega`, `(LocalDate.parse`, onde passamos um `DateTimeFormatter`. O que eu vou fazer? Eu vou pegar esse `this.dataDaEntrega`, vou dizer que esse é o texto e eu vou dizer como esse texto está formatado.

[09:54] Lá na interface gráfica eu vou pedir para o usuário digitar assim: "10/10/2020", dia, mês e ano separado por barra. Eu preciso de alguém com inteligência suficiente para saber fazer essa conversão, que é um `DateTimeFormatter`.

[10:05] Eu vou criar esse como estático, `private static final DateTimeFormatter formatter =`, recebendo `DateTimeFormatter.ofPattern` e vou dizer que o padrão é `("dd/MM/yyyy");`. Esse é o formato que vamos usar aqui.

[10:41] E agora temos que *setar* também o valor. Eu vou só fazer um `(new BigDecimal` e vou passar o `(this.valor));` para ele e ele vai converter para mim. Se estiver no formato inválido ele vai travar, mas vamos fazer a validação disso para garantir que isso não aconteça.

[11:02] Agora já conseguimos converter para oferta, já temos `pedidoBuscado`, que precisamos fazer é pegar esse pedido mesmo, então `Perdido pedido =`

`pedidoBuscado.get();` - que é onde eu pego realmente um objeto. Eu faço essa `nova.setPedido` . Não criei a oferta, agora criei. E vou passar o `(pedido);` que eu busquei no banco.

[11:39] E ao pedido eu vou adicionar aqui no `pedido.getOfertas` , Criei *getters* e *setters* para os dois. Digito um `getOfertas().add(nova);` e a nova oferta. Associei um com o outro e a única coisa que eu preciso fazer é salvar o pedido. Então `pedidoRepository.save` e passo o `(pedido);` .

[12:12] "Mas eu não preciso salvar a oferta também?" Não precisa, automaticamente ele já vai na hora em que você salvar o pedido. Como ele está com *cascade all* ele já vai salvar também para *persistence*. Nós vamos persistir isso. Eu só digito um `return nova;` .

[12:28] Então já temos o *endpoint* que conseguimos salvar a oferta. O que vamos aprender no próximo vídeo é utilizar isso usando o Vue.js e o Axios, a partir da página *home*. Até o próximo vídeo!