03

Criando ofertas

Transcrição

[00:00] Nós conseguimos acessar o clique do botão `enviarOferta` e estamos até logando aqui o pedido associado a esse botão. No vídeo anterior conseguimos ver até as informações que o usuário preencheu nos `inputs`, de valor, data de entrega e comentário. Nós colocamos aqui como `v-model`, fizemos isso que chamamos de *bind* (ligação) entre a variável dentro do pedido e o próprio `input`, o próprio *text area* etc.

[00:30] E qual é a ideia? Precisamos fazer mesmo a requisição para aquele serviço Rest que já criamos chamado de `criarOferta` e essas requisições que, inclusive, já fizemos anteriormente. Fizemos com o Axios, que baixamos, colocamos aqui embaixo na "home.html" e nós vamos usá-lo de novo. O importamos aqui.

[00:51] Vamos usá-lo de novo para fazermos não uma requisição GET, mas uma requisição do tipo POST. E não vai ser para `/api/pedidos/aguardando`, mas para `ofertas`. Aqui, `OfertaRest`, `/api/ofertas`. O método associado ao POST é o que criamos, `criarOferta`.

[01:11] Além disso, podemos até colocar aqui um `.then(response =>`, só para sabermos se deu certo ou não, podemos colocar aqui `console.log(response));`, a título de teste mesmo.

[01:26] E o que mais nós precisamos? A diferença do GET para o POST não é só o fato de agora o nome do método ser `.post`, mas o fato de passarmos um objeto, com alguns atributos aqui dentro.

[01:43] E esses valores que nós colocamos aqui são os valores que serão mapeados para essa requisição de nova oferta. O que `RequisicaoNovaOferta` recebe? Recebe alguém chamado `pedidoId`, então vou colocar aqui, `pedidoId:`. Igual ao `pedido` que estamos fazendo a oferta, `.id,`.

[02:06] O que mais vamos ter que passar para ele? Valor, data da entrega e comentário. Vamos colocar aqui: `valor: ,`, `dataDaEntrega: ,` e `comentario:`. Essas informações vêm de onde?

[02:30] O valor vem de dentro do pedido, que é o `input` de valor que o usuário digita aqui, `valorNegociado`. Então tem que ser o mesmo nome que recebemos no pedido, que não veio preenchido, obviamente, porque o usuário ainda não fez nenhuma oferta para o pedido e nem foi aceito. Então ele basicamente não tem.

[02:52] Mas nós colocamos esse `pedido.valorNegociado` - não é o valor negociado que veio da consulta, mas o que o usuário preencheu aqui no `input`. Nós vimos isso no vídeo anterior.

[03:05] Além disso, tem data da entrega, que está nesse outro atributo, `pedido.dataDaEntrega`. Estamos reaproveitando o próprio pedido para fazer isso, você pode fazer de várias outras formas também, mas por praticidade eu estou usando o próprio objeto que eu já recebo e que está associado.

[03:28] Se você olhar, esse *card* aqui está associado àquele pedido, um determinado pedido da lista de pedidos, esse está associado ao outro pedido, então estou aproveitando o fato de que cada um já tem um objeto associado. Estou alimentando com esse objeto e agora estou preenchendo aqui o objeto da oferta.

[03:45] Então, pronto! `pedidoId`, `valor`, `dataDaEntrega` e `comentario`, é isso que precisamos passar, não precisamos passar mais nada além disso. Eu vou derrubar a aplicação e subi-la com o modo *debug*, para acompanharmos a

requisição. Eu já até coloquei aqui um *breakpoint* e vamos fazer um teste, vamos ver se isso funciona.

[04:10] Esperandoa aplicação subir, porta 8080, beleza. Eu acho que já dá para tentarmos acessar. Ele vai cair, obviamente, na tela de login. Nós nos logamos novamente. Digitamos "joao" e "joao" no usuário e senha, logamos e vamos para a página "Faça sua Oferta".

[04:36] E aqui ele preencheu de novo a listinha com o que ele consultou de lá. Eu vou colocar aqui no valor de "123123", data da entrega de "10/10/2020", "algum comentário", que você pode não fazer. Vamos clicar no "Enviar Oferta" e vamos receber isso aqui. Vamos ver se a requisição já foi preenchida... Ainda não.

[04:58] Nós passamos os dados corretos com os nomes certos, mas ele não fez o *bind*, ele não conseguiu receber. Por que isso? Porque temos que indicar para o Spring que ele tem que preencher esse objeto com o que veio na requisição, então `@RequestBody` é a anotação que indicamos exatamente isso, pega os dados da requisição e adiciona nesse aqui.

[05:26] Ele não vai funcionar, vou derrubar e subir de novo, acho que agora ele vai subir um pouco mais rápido e vamos acessar de novo lá. Tentar acessar de novo a página de login. De novo ele dá essa travada. Beleza, já entrou logado!

[06:08] Vamos preencher aqui, valor de "123123", data da entrega de "10/10/2020" e "algum comentário", como preenchi anteriormente. Envio a oferta. Recebemos agora com o `RequestBody`, vamos ver se ele preencheu a requisição.

[06:21] Pronto, agora tem todas as informações, o ID do pedido. A primeira coisa que ele vai pegar é o pedido associado a esse ID no banco de dados. O pedido foi buscado? Já tem o valor aqui, então ele não está nulo.

[06:35] Agora vou pegar o valor do pedido mesmo, aqui tem todas as informações do pedido, inclusive tem as ofertas - mas ele está vazio, não só porque não tem oferta, mas porque está *lazy*. Ele não ia pegar de qualquer jeito, só se invocássemos isso.

[06:49] E mesmo invocando, está limpo aqui o *log*. Se eu fizer um `getOfertas` aqui no `OfertasRest` ele não vai trazer as ofertas e não vai fazer um *select* para buscar, porque ele está configurado como *lazy*. Só para confirmar, está vendo? `FetchType.LAZY`, ele não iria buscar de qualquer jeito.

[07:08] E vamos salvar esse pedido no banco de dados com a oferta nova associado a ele, vamos retornar isso para a tela e nada vai mudar. Mas vamos ver no banco de dados, aqui na tabela de ofertas, se conseguimos visualizar o conteúdo novo.

[07:24] Temos uma oferta já registrada associada ao pedido com o ID 6, que é esse pedido aqui, "Video Baby Monitor". De quem é? Da Maria. Já conseguimos criar uma oferta.

[07:36] O que não podemos deixar ficar assim, porque precisamos dar uma resposta para o usuário. Então ele fez uma requisição, criou, mas para o usuário nada aconteceu. O que podemos fazer é alterarmos o botão. Na hora que ele clicar em "Enviar Oferta", recebeu a resposta. Nós mudamos para oferta enviada, por exemplo. É o que vamos fazer mesmo.

[08:03] Aqui na "home.html" temos o botão de enviar oferta. Eu vou copiar e colar isso aqui: `<button v-on:click="enviarOferta(pedido)" class="btn btn-primary mt-2">Enviar Oferta</button>` e vou fazer o seguinte: esse primeiro botão vai ficar invisível e ele vai ter o conteúdo de `Oferta Enviada`.

[08:19] Podemos até mudar um pouco o estilo desse botão, para ele mudar de cor. Temos o `btn-primary`, mas temos também o `btn-success`. No caso do `Oferta Enviada`, vamos utilizar esse outro estilo do Bootstrap.

[08:36] E se abrirmos a tela desse jeito, os dois botões vão estar aparecendo lado a lado: "Oferta Enviada" verde e "Enviar Oferta" em azul. Eu não quero que o primeiro apareça, eu quero que o "Enviar Oferta" só apareça depois que eu clicar nesse botão. O botão "Enviar Oferta" tem que desaparecer.

[08:54] Podemos fazer isso com outro recurso do Vue que é o `v-if`, que podemos dizer que se um determinado valor for verdadeiro, apresentamos esse botão; senão tem outro bem legal. Fica uma sintaxe bem interessante, que é o `v-else`. Ele já vai entender que isso está associado ao `if` de cima ainda.

[09:21] O `v-else` é só isso, você não precisa mudar nada. Mas o `v-if` é o seguinte: eu vou pegar o valor do pedido, vou digitar `pedido.ofertaEnviada`. Quando esse valor for verdadeiro, ele vai mostrar esse `v-if="pedido.ofertaEnviar"` e vai desaparecer com o de baixo.

[09:47] Só tem uma questão: quando ele fizer o *bind* desse `if` com o atributo `ofertaEnviada` do pedido, ele não vai encontrar esse atributo `ofertaEnviada`. Então mais uma vez, eu estou aproveitando o retorno da consulta que fazemos de pedidos, estou aproveitando esses objetos que são criados em JavaScript através do JSON que recebemos, para fazer mais uma lógica aqui na tela.

[10:12] Óbvio que você pode criar outro objeto, fazer uma lógica diferente, mas eu acho essa mais prática e talvez para cá funcione, está funcionando.

[10:21] E o que precisamos fazer é o seguinte: como esse atributo `pedido.ofertaEnviada` não existe, vamos criá-lo - e para criar não é tão difícil, nós recebemos o objeto `pedidos`. Deixe-me fechar isso aqui, jogar esse `this.pedidos = response.data` para baixo.

[10:36] E o que vamos fazer, com uma implementação mais resumida, é o seguinte: antes de associarmos os dados da resposta ao pedido, vamos alimentar os dados da resposta com mais informações.

[11:01] Para isso eu preciso percorrer esses dados aqui e eu vou usar o `forEach()`. E o que acontece? No `forEach` vai me entregar cada "pedido" contido dentro do `data`, que é basicamente esse *array* de pedidos. Por isso que estamos conseguindo utilizar aqui o `forEach`, porque é um *array* que recebemos de resposta.

[11:23] E eu vou criar o atributo aqui, vou fazer `pedido.ofertaEnviada` e já vou inicializar. Eu estou criando e inicializando um atributo novo, que não existe, não veio na resposta de `pedidos/aguardando`; eu simplesmente criei para facilitar. Mais uma vez, estamos reaproveitando um dado que já recebemos. E eu vou deixá-lo como falso.

[11:44] E o que eu faço? Na resposta do `.post`, em vez de colocar `console.log(response));` aqui, que eu não preciso, eu digito `pedido.ofertaEnviada` recebendo verdadeiro. Será que vamos conseguir? Vamos lá!

[12:01] Eu vou abrir a oferta de novo, vou criar mais uma oferta aqui, "321321" no valor, vou colocar data da entrega de "20/10/2020", fazer "outro comentário" e vou enviar a oferta. Vou soltar aqui o modo *debug*, porque já vimos que está funcionando, e fazer a requisição.

[12:19] Ele foi lá, criou e alterou esse botão para "Oferta Enviada". Essa é uma maneira de mostrarmos para o usuário que a ação aconteceu ali. Se formos de novo para o banco de dados, vamos ver que ele já criou isso daqui.

[12:35] A próxima coisa que precisamos fazer é validar esses dados. Imagine que eu passo qualquer coisa aqui, "outro comentário 2", e faço o envio. Se bem que ele não vai funcionar para esse "Video Baby Monitor", eu vou usar o de baixo, "Wireless Earbuds TaoTronics". Vou colocar letras no valor, números desformatados na data de entrega etc., vou enviar a oferta. O que acontece? Acho que aconteceu alguma coisa.

[12:56] Primeiro que ele não conseguiu converter o dado, pois estávamos esperando outro tipo de informação, então ele não conseguiu parsear isso para data.

[13:07] Vamos deixar uma data legal, de "10/10/2020", e vamos enviar a oferta. Será que ele conseguiu? Não, porque provavelmente não conseguiu parsear outro valor, `NumberFormatException` . Precisamos validar esses dados que o usuário está mandando, para não ficar dando erro lá no back-end.

[13:22] É isso que vamos fazer no próximo vídeo, até lá!