01

Validações no controller

Transcrição

[00:00] No vídeo anterior aprendemos a utilizar o BootStrap em conjunto com um pacote externo, com uma biblioteca externa que funciona como um envelope dos métodos ajudantes que vimos que existem no Play para estilizar automaticamente os nossos formulários.

[00:17] Vimos que o pacote chama B3 e que podemos usar os campos de texto e ele já deixa o formulário bonitinho, que nem vemos aqui.

[00:27] Mas nós agora precisamos garantir que o nosso produto seja salvo no banco de modo adequado, por exemplo, ele tem que ter um título, ele tem que ter um tipo, o preço precisa ser acima de 0, o código dele precisa ser realmente único.

[00:43] Então precisamos fazer uma validação do nosso formulário e vamos fazer isso utilizando um sistema de validação que o Play tem embutido já direto nos formulários dele.

[00:53] Para saber se tem algum erro, precisamos vim aqui no ProdutoController e precisamos perguntar para o formulário, “tem algum erro no seu formulário?”, então, “if formulário.hasErrors”, se tiver um erro no nosso formulário, que que vamos fazer? Vamos devolver o usuário para o formulário de cadastro de produto mas nós não vamos retornar um HTTP Status ok, vamos falar que a requisição dele foi inválida, foi inadequada.

[01:28] Vamos retornar um bad request e renderizamos a página de novo, “formularioDeNovoProduto.render” e devolvemos o formulário com os erros para ele não perder nenhum dado caso ele tenha escrito alguma descrição super complexa.

[01:53] No caso, caso tenhamos escrito uma descrição complexa. Eu não gosto de perder o meu trabalho. Mas se nós só fizermos isso, nada vai acontecer porque nós não temos realmente nenhuma restrição no nosso produto.

[02:08] Então antes de testar, vamos criar alguma restrição. Viemos aqui no produto e vamos criar uma restrição para o título, o título tem que ser obrigatório. Vamos colocar a notação required nele, do pacote play data validation constraints e, não só vamos falar que o título é obrigatório, como vamos colocar uma mensagem caso o usuário falhe em colocar o título. Vamos dizer que o “campo título é obrigatório”.

[02:49] Quais outros campos são obrigatórios? Código certamente é obrigatório porque é o identificador do nosso produto, o tipo é obrigatório também porque ele que vai definir que produto estamos vendendo, que tipo de produto estamos vendendo. Nós não podemos não saber se estamos vendendo uma roupa ou um e-book, um livro ou um e-book.

[03:14] A descrição não é obrigatória, é de bom tom sempre ter uma descrição, mas podemos só vender um produto sem descrição. Agora o preço também é obrigatório, nós não podemos vender um produto sem preço, vai quebrar todo o nosso sistema.

[03:33] Vamos ver agora o que acontece? Atualizamos aqui, garante que ele compilou tudo bonitinho, parece que compilou, atualizamos. Não tem nada de diferente, olha só, se eu tentar cadastrar aqui, ele mostra uma mensagem de erro de validação HTML.

[03:52] Pois é, quando criamos uma restrição no Play, no código Play, ele já cria a restrição HTML também, olha só, "required=true", eu vou remover esse cara aqui e eu vou enviar o formulário. Vamos tirar isso daqui, eu vou deixar sem título, eu vou colocar um código livro, eu vou colocar um tipo, eu vou colocar uma descrição "um livro" e eu vou colocar um preço, 10 reais.

[04:26] Agora se eu enviar o cadastro, ele volta o nosso formulário com o código livro, descrição e preço já preenchidos e o título com a mensagem que cadastramos aqui, "campo título é obrigatório". É incomum esse erro acontecer, mas se por acaso alguma coisa der errado na renderização do HTML, temos uma segurança no backend também.

[04:58] Mas tem algumas outras validações que eu quero fazer, por exemplo, saber se o preço do produto é maior do que 0. Eu não quero vender produtos com preços negativos e eu também quero saber se o produto que eu estou cadastrando, se o código do produto que eu estou cadastrando já está ocupado porque ele tem que ser único.

[05:17] Então vamos começar pelo preço, viemos aqui no ProdutoController e antes de perguntar se o formulário tem erros, vamos pegar o produto e vamos perguntar se o produto tem um preço adequado, "if produto.getPreco" for menor ou igual a 0, aliás, menor do que zero porque eu posso vender um produto de graça.

[05:47] E eu vou, que que eu faço nesse caso? Eu preciso rejeitar o campo do preço. Então eu vou pegar o meu formulário e eu vou falar "rejeita o valor do campo do preço", então, "reject" preço, e eu vou passar uma mensagem pra ele. No caso eu não vou simplesmente escrever a mensagem aqui, eu vou criar uma "reject" e eu vou passar o campo do preço, mas como eu passo o campo do preço? Tenho que criar uma mensagem de validação.

[06:20] "new ValidationError" e eu falo, "rejeita", o erro está no campo do preço, então preço aqui e aqui eu vou escrever a minha mensagem, "o preço do produto tem que ser maior que 0", eu criei uma mensagem de validação.

[06:48] Vamos ver o que acontece? Eu vou escrever aqui o título, livro de play, o código já está ok, o preço eu vou colocar -1, ou eu vou colocar 0. Cadastrar e ele cadastrou porque eu permito preços iguais a 0, eu vou dar um voltar aqui e eu vou cadastrar um produto com preço -1. Cadastrar, olha só a mensagem aqui, "o preço do produto tem que ser maior do que 0".

[07:22] E agora como é que verificamos o código? Temos que buscar no banco se já existe um produto com aquele código e para isso vamos precisar fazer algum objeto, criar algum objeto que tenha acesso aos dados do banco, comumente chamado de Data Access Object ou DAO, então vamos fazer um produto DAO, um objeto de acesso aos dados do produto.

[07:49] Então vamos lá, "Ctrl + N", eu vou fazer uma classe, eu vou colocar ela no pacote DAOs e eu vou chamar ela de ProdutoDAO. E para buscar um produto com um dado código, eu vou criar um método aqui que retorne um produto "comCodigo (String codigo)", então caso exista um produto com esse código, ele vai me retornar o produto, se não, ele vai retornar um objeto nulo.

[08:22] No caso, eu vou retornar um new produto para ele sempre dar erro, vou importar aqui a classe produto. Então toda vez que eu perguntar se já existe um produto com dado código, ele vai me retornar um produto novo.

[08:40] Então vamos lá, aqui depois do preço, temos que perguntar para esse DAO se já existe um produto com esse código. Como que eu acesso o DAO? Podemos usar a injeção de dependências de novo, viemos aqui em cima e pede o nosso DAO injetado, então "inject private ProdutoDAO produtoDAO".

[09:10] Agora temos uma variável do ProdutoDAO, eu vou importar aqui a classe e podemos perguntar para ele se já existe um produto com esse código. Então, "produtoDAO.comCodigo produto.getCodigo", se esse cara não for nulo, temos que rejeitar o campo do código também. Então eu vou copiar aqui e o erro de validação vai ser no campo do código e a mensagem vai ser "já existe um produto com este código".

[09:54] Vamos testar? Que códigos nós já temos cadastrados no banco? Nós já temos aqui um código repetido, eu vou apagar ele, “delete from produto where preco = 0;”, aqui agora temos só códigos únicos.

[10:17] Vamos tentar cadastrar um produto com o código “livro play”. Vamos lá, atualizamos aqui, podemos clicar aqui no link de novo produto também e vamos cadastrar aqui um livro, eu vou cadastrar que ele é um e-book, com a descrição “um e-book de play” e preço 1.

[10:46] Se tentarmos enviar, ele reconhece que já existe um produto com esse código, mas na verdade é uma mentira porque o nosso produto DAO está retornando um produto sempre. Como que fazemos para realmente conferir se existe um produto no banco? Temos que buscar no banco. E como buscamos? O Ebean tem um objeto que é um buscador, um finder.

[11:11] Então vamos declarar aqui que queremos um finder que recebe um produto, mas o finder só trabalha com produtos que tenham id e nós não cadastramos um id. Então vamos terminar isso aqui, “produtos = new Finder(Produto.class)”.

[11:46] Ele ainda está dando erro, deixa eu importar aqui o finder, ele está dando erro aqui porque ele precisa de um id, só que nós não temos um id no produto. Então vamos cadastrar um id no produto, “@id” do javax persistence e “@GeneratedValue”, eu não quero me preocupar com preencher o valor do meu id. “PRIVATE long id” e geramos um getter e um setter para ele com ggas, id, after preço.

[12:27] Agora todos os getters e setters estão na mesma ordem, agora temos um id, agora podemos falar aqui que a classe do id é um long e ele vai ficar feliz, sem erros de compilação, mas vai acontecer uma coisa que nós não esperávamos.

[12:48] Olha só, se eu tentar acessar a minha página, ele vai falar que o banco de dados precisa de uma evolução. Lembra que eu falei que quando alterávamos o banco, alterava os nossos modelos, o Play registrava essa alteração e fazia uma alteração no banco? Pois é, é exatamente isso que está acontecendo aqui.

[13:06] Olha só, ele quer que evoluamos o banco de dados porque a tabela produto não tem um id, então ele quer adicionar o id na tabela produto, só que como nós não estamos em produção, ele simplesmente vai apagar a tabela e recriar do zero, apagando todos os produtos que tenham lá. Por mim tudo bem, eu tenho dois produtos lá, eu vou aceitar.

[13:34] Mais para frente vamos ver exatamente como criar uma evolution que não apague tudo e recrie do zero. Então eu vou aplicar aqui e agora a nossa tabela tem um id, mas não tem nada lá dentro, “desc produto;”, agora podemos ver que ela tem um id, que ele é a chave primária e não pode ser nula e que é auto increment.

[14:03] Então agora podemos fazer a busca no banco e procurar um produto que tenha um código, então vamos lá, fazemos assim, “produtos.” estou criando uma consulta, uma query, onde “.where” tenhamos um código que seja igual ao código que recebemos ali, então “.eq(“codigo”, codigo")”.

[14:32] E queremos encontrar um único objeto, se encontrarmos mais de um tem alguma coisa errada. Então, “findUnique”, encontre um único objeto cuja o código seja o código que recebemos, ele vai retornar um produto.

[14:52] Vamos cadastrar um produto aqui então, eu vou cadastrar o livro de play com modelagem, “um livro de play”, com o preço 11, cadastrar. Agora eu vou tentar cadastrar outro, eu vou cadastrar um livro de play, mas eu sem querer coloco “livro-play-modelagem” como tipo, descrição “livro de play” e preço 10.

[15:23] Eu clico em cadastrar e ele, olha só, ele reconhece que já existe um produto com esse código, agora se eu apagar aqui e fizer o cadastro com outro código, cadastrou direitinho. Agora se eu selecionar, ele criou os dois, está tudo ok.

[15:41] Por agora é só, vimos o que? Vimos como fazer uma validação de campos obrigatórios simplesmente colocando uma notação e perguntando pro formulário se ele tinha algum erro, como ele tinha algum erro, nós só renderizávamos ele de novo. Aprendemos a fazer validações customizadas e fazer até buscas no banco para fazer validações mais complexas.

[16:11] Na próxima aula vamos dar uma refatorada nisso para adequar um pouquinho melhor as boas práticas, já que estamos usando Java 8, vamos usar objetos que se adequem a linguagem sem precisar ficar comparando com null, por exemplo.