04

Refactor - ValidadorDeProduto e Optionals

Transcrição

[00:00] No vídeo anterior, aprendemos a utilizar os formulários de Play para fazer validações direto do modelo com o formulario.hasErrors, viemos nesse nosso modelo e adicionamos as anotações required, fazendo assim com que esses campos sejam obrigatórios, isso inclui uma validação feita direto no HTML pelos ajudantes do Play.

[00:27] Agora, nós também aprendemos a fazer algumas validações externas, como essas duas que comparam se o preço é maior do que 0 e se já existe um produto com aquele código único. E para isso, chegamos até a criar um DAO, que faz consultas no banco para retornar os nossos produtos.

[00:47] Agora, o que seria legal é pegarmos esses trechos de código de validação e extrair para alguma classe de validação para seguir um pouco as boas práticas e desacoplar essa lógica do controlador.

[01:00] Vamos fazer isso? Então, eu vou recortar isso aqui e vamos criar uma nova classe. Então, vamos criar essa classe? Eu vou dar aqui “Ctrl + N”, class e colocar essa classe no pacote validadores, a classe vai chamar “ValidadorDeProduto” porque ela vai validar um produto.

[01:25] O que vamos fazer aqui? Vamos criar um método que contenha a lógica que removemos do controlador, mas eu vou adicionar um retorno a esse método, então vamos lá, “public boolean temErros” e eu vou receber aqui um formulário de produto, “return false” por enquanto, não tem erros, vamos importar as classes form e produto.

[02:04] Agora pegamos nossa lógica aqui de volta, mas a variável produto ainda não existe, então temos “formulario.get” e vamos ter o nosso produto de volta aqui. Então a nossa lógica já está aqui, estamos adicionando os erros no formulário, nós ainda não temos o DAO, como removemos a lógica do DAO aqui do controlador, podemos remover ele daqui também. Então, vamos tirar ele daqui e jogar no nosso validador. Agora parece tudo ok.

[02:46] O que eu quero adicionar aqui é esse retorno, ao invés de fazer aqui, se o formulário tem erros, eu vou chamar direto o tem erros do validador e para isso, eu vou pegar esse formulário temErros e jogar aqui porque adicionamos os erros do formulário aqui dentro, então aqui, esse mesmo formulário já vai ter os mesmos erros que ele teria do lado de fora.

[03:17] Mas como eu faço para acessar o nosso validador aqui? Do mesmo jeito que fizemos com o DAO, injetamos ele, “inject private ValidadorDeProduto validadorDeProduto”, então aqui, se nós ao invés de perguntarmos se o formulário tem erros, perguntarmos para o validador, “tem erros no formulário?”, estamos recebendo exatamente a mesma resposta.

[03:47] Então, vamos ver se tudo continua funcionando para garantir que não fizemos nada errado. Atualizar aqui, deixa eu conferir um produto que nós já tenhamos no banco, eu vou selecionar os produtos, temos um produto com o código “livro play”, então vou cadastrar um produto com código “livro play”, descrição “um livro de play” e preço 10 reais. Olha só, a validação continua funcionando, já existe um produto com este código.

[04:23] Então, o que mais podemos fazer aqui para deixar o código um pouco mais adequado às boas práticas? Eu vou abrir aqui o nosso validador e tem uma comparação aqui que, é ruim ficar comparando objeto com nulo, especialmente se você estiver usando Java 8 e no caso, estamos usando Java 8.

[04:45] Eu vou trocar a comparação do produtoDAO aqui, do retorno do método com código, ao invés de retornar um produto ou retornar um objeto nulo, eu vou retornar um objeto opcional, que pode conter o produto ou não.

[05:02] Então vamos lá, eu vou abrir o produtoDAO e substituímos aqui, ao invés de retornar um produto, vamos retornar um “Optional”, importar aqui a classe optional do Java util e agora temos que encapsular essa lógica aqui, que retorna um produto, dentro de um optional, então primeiro eu vou extrair para uma variável “produto” e aqui eu vou retornar um “optional.ofNullable” porque o objeto pode ser nulo e entregar o nosso produto para esse optional. Aqui é só.

[05:51] Agora lá no validador, nós ainda estamos comparando com nulo, mas esse cara aqui já não é mais nulo ou não, ele sempre vai estar válido, o que ele pode não ser é, ele pode estar presente ou não. No caso, se ele já estiver presente, se ele existir, se o produto dentro dele for válido, rejeitamos o formulário porque o código já foi utilizado.

[06:17] Acho que é só, essas são as duas mudanças que eu achei que eram importantes de fazer para podermos continuar o nosso projeto. Vamos só conferir se ele está funcionando, eu vou tentar recadastrar aqui e ele continua com o mesmo erro, quer dizer que o código ainda está sendo conferido de maneira correta.

[06:36] Que que vimos aqui? Vimos como criar o nosso próprio validador, injetar ele no controller e demos uma adequada para que o projeto ficasse um pouco mais de acordo com as boas práticas de programação.

[06:50] Na próxima aula, vamos ver como interagir um pouco mais diretamente com o usuário, com o cliente, com nós mesmos mostrando uma mensagem de erro ou de sucesso de acordo com a ação que executarmos.