07

Filtro dinâmico e validação

Transcrição

[00:00] Na aula anterior, aprendemos a criar rotas customizadas com parâmetros, como por exemplo, o tipo e usamos isso para criar um filtro para os nossos produtos, para a nossa API, para que o usuário consiga buscar só os produtos de um dado tipo.

[00:20] Mas e se ele quiser, por exemplo, procurar todos os livros com um dado tipo e um dado preço? Por exemplo, os audiobooks que custem 5 reais. Vamos criar uma rota que seja /tipo e o tipo, /preço e o preço, para criar dois filtros ao mesmo tempo? Vai ficar feio e não é muito escalável, se tivermos 8 atributos, vamos fazer uma rota que vai até o infinito, não é legal, vai ficar um caos.

[00:49] E o que que podemos fazer para resolver isso? Nós já vimos como receber dados de um formulário, podemos pegar esses parâmetros do formulário, que são escondidos, que são dinâmicos e usar eles para fazer o nosso filtro. Pode ser.

[01:09] Como isso vai ser uma rota super genérica, que podemos pegar um filtro ou n, seria legal usar a rota mais genérica, /api/produtos, para isso. Então, eu vou alterar aqui, a nossa rota que retorna todos os produtos e deixar ela explicita falando, "você retorna todos os produtos". E agora fazemos a nossa rota mais genérica "/api/produtos" e deixa ela aí.

[01:44] Para onde que ela vai apontar? Vamos fazer uma rota com filtros dinâmicos, então eu vou apontar ela para o ApiController para um método chamado "comFiltros". Vamos criar esse método, como que ele funciona? Viemos aqui, "public result comFiltros", o que esse método vai retornar? A mesma coisa de sempre, vai retornar um return, um ok, que vai ser algo convertido para JSON.

[02:23] Então vamos, se tivermos produtos, vamos criar um envelope, "new EnvelopeDeProdutos", que vai receber uma lista de produtos, essa lista de produtos por enquanto eu vou deixar aqui fora, guarda isso em uma variável, "EnvelopeDeProdutos envelope" e transforma isso em um "Json.toJson envelope".

[02:52] Mas de onde que vem essa variável produtos? Vamos pegar ela utilizando os filtros de um formulário, então precisamos da nossa fábrica de formulário. Vamos injetar ela aqui então, "@inject private FormFactory", fábrica de formulários, "formularios" e aí vamos pegar os dados do formulário aqui. "Formularios.form" e vamos atrelar esse formulário à nossa request com "bindFromRequest", não precisa de parâmetro nenhum, guarda isso em uma variável.

[03:36] Temos os dados do formulário, mas do modo que fazíamos antes, tínhamos que fazer "formulário.get" e precisávamos passar o nome do nosso parâmetro do formulário. Só que nós não sabemos os parâmetros que o usuário vai passar, então que que fazemos? Fazemos "formulário.data" e recebemos um mapa de parâmetros com todos os parâmetros desse formulário.

[04:07] Conseguimos iterar nesse mapa, que é exatamente o que precisamos, vamos passar por cada um dos nossos parâmetros e adicionar como filtro no nosso DAO. Vamos chamar isso aqui do produtoDAO, um método, como fizemos em cima com o tipo, vai ser o mesmo nome "comFiltro" e vamos passar aqui os nossos parâmetro.

[04:37] Vamos criar esse método para ver como ele funciona? Vamos precisar fazer uma consulta, então vamos declarar a nossa consulta, "produtos.where", só que invés de já concatenar as nossas condições aqui, eu vou guardar isso em uma variável.

[04:59] Eu vou chamar essa variável de “consulta” e quando terminarmos de incluir os parâmetros, vamos retornar “consulta.findList”, que queremos a lista de todos os produtos com os filtros que vamos criar aqui.

[05:17] Se deixarmos esse método assim, ele vai retornar sempre todos os nossos produtos, que nem aqui, o “produtos.all”, mas vamos criar os nossos filtros aqui a partir dos parâmetros. Como que fazemos isso? Vamos percorrer os parâmetros e para cada um deles, adicionamos o parâmetro na nossa consulta.

[05:38] E como vai ser isso? Quando fazemos um get com parâmetros, colocamos aqui a interrogação e coloca o nome do parâmetro e o valor dele. O nome do parâmetro vai representar o nosso atributo e o valor vai ser o valor que queremos para esse atributo.

[05:57] Então, para cada um dos parâmetros, “parametros.entrySet”, para cada uma das entradas desse parâmetro, “foreach”, vamos pegar a entrada e vamos fazer alguma coisa com ela.

[06:14] O que vamos fazer com ela? Vamos adicionar na consulta, vamos adicionar como o que? Como uma restrição de que seja igual, que nem fizemos com o tipo. Então, “equals”, e vamos falar que a nossa variável, o nosso atributo, como fizemos aqui com tipo, o nosso atributo era o que? Era a chave do nosso parâmetro.

[06:41] Então, “entrada.getKey”, vamos falar que esse atributo tem que ter valor igual ao valor da nossa entrada, aqui era 10, é o valor da entrada, “entrada.getValue”. Para cada um dos parâmetros que tivermos, vamos adicionar essa condição na consulta. Se tivermos aqui, por exemplo, dois parâmetros, “tipo = e-book”, vamos ter duas chaves, preço e tipo, que vão ser entradas aqui na nossa consulta, cada uma com seu valor apropriado.

[07:22] Vou salvar aqui e vamos fazer o teste, eu tenho no momento, três livros, três produtos no nosso banco, dois e-books, cada um com um preço e dois produtos de preço 10, cada um com um tipo. Vamos ver como é que isso funciona.

[07:39] Se eu falar aqui que eu quero todos os produtos com preço 10, eu vou pegar o livro de play com modelagem e o livro de JavaScript, olha só os dois aqui com preço 10, agora se eu quiser com preço 10 e com tipo igual a livro, eu vou pegar só o livro de play com modelagem, que é o único que se encaixa nas condições.

[08:02] Mas e se, por exemplo, eu sem querer errar aqui e escrever “precos=10”, que que acontece? Tomamos uma exceção na cara. Por que isso? Porque a coluna “precos” não existe na nossa tabela, só que estamos passando ela aqui direto na nossa consulta, está passando preços aqui direto na nossa consulta e ele vai procurar por esse atributo na tabela.

[08:25] Ele não existe, tomamos uma exceção e não é legal jogar uma exceção na cara do usuário, seria mais legal retornar um JSON com a lista de erros e com status bad request 400, ao invés dessa exceção feia, que o usuário não vai nem entender o que que aconteceu. Vamos fazer isso.

[08:49] Como que fazemos isso? Temos que validar esse formulário e se tiver algum campo que não seja parte dos atributos do nosso produto, temos que retornar um bad request e falar para o usuário que ele fez uma requisição inválida.

[09:06] Então vamos fazer uma validação do nosso formulário, poderíamos fazer um modelo, eu vou fazer aqui só um método “validaFormulario”, vai receber um formulário, vamos criar o método, ele já recebe um formulário e vamos pegar os nossos parâmetros aqui no formulário.

[09:29] Para cada um desses parâmetros, temos que ver se ele pertence aos atributos do nosso produto, então precisamos de uma lista dos atributos, “List ATRIBUTOS = Arrays.asList”, vamos fazer essa lista aqui hardcoded mesmo, depois eu explico porquê.

[09:55] Quais são os atributos que temos? Temos id, temos título, temos código, temos descrição, temos tipo e temos preço. Então, essa é a nossa lista de atributos.

[10:15] Como ela vai ser sempre igual, podemos transformar ela em uma constante, então eu vou transformar ela aqui em uma constante, eu vou arrastar ela para o começo do nosso arquivo, “private static final” e eu colo ela aqui.

[10:39] Temos uma constante com a lista dos nossos atributos, por que que eu fiz uma constante? Se mudarmos o nosso produto, se adicionarmos um campo no nosso produto, vai dar problema, vamos ter que vim aqui na lista e adicionar o campo nela.

[10:57] Por que que eu fiz assim? Porque poderíamos utilizar a reflexão para fazer essa lista, mas é um assunto que eu não quero tocar agora, eu não vou entrar nele.

[11:07] Você pode, se quiser, fazer o curso de Java Reflexion, magic e metaprogramação para aprender sobre Reflexion. Caso você conheça, você pode vim aqui e usar direto, não tem problema, mas se você não conhecer, vai lá, faz o curso e você pode adaptar o nosso programa para usar.

[11:28] Eu acho que é bem mais interessante se você tiver um curso inteiro sobre isso, que é um curso bem legal de aprender, mas que não é o nosso foco aqui, então eu vou deixar só mencionado.

[11:41] Então pegamos esses parâmetros e, para cada um deles, temos que ver se ele está na lista de atributos, “parametros.”, os atributos são a representação das chaves, então nós não precisamos pegar as entradas que nem fizemos no DAO, podemos pegar só as chaves. Então, “keySet”, para cada uma das chaves, “forEach(chave)”, vamos fazer uma ação.

[12:07] Poderia fazer um filtro aqui, mas eu quero realmente fazer uma ação diferente para cada chave. Eu vou fazer o que? Eu vou adicionar nosso formulário, aliás, caso a chave não esteja dentro dos atributos, então “ATRIBUTOS.contains” a nossa chave, se essa condição não for verdadeira, vamos rejeitar esse formulário, “formulario.reject”.

[12:41] E vamos fazer um erro de validação, “new ValidationError”, que vai rejeitar esse atributo, “atributo inválido” e vamos passar aqui o nome do atributo, no caso, a chave.

[13:00] Vamos ver como que isso funcionou, aliás, nós ainda não estamos retornando isso, como rejeitamos campos do formulário, podemos aqui perguntar se o formulário tem erros, “hasErrors”, caso ele tenha erros, retornamos um bad request, “return badRequest” com os erros do nosso formulário, então “formulario.errorsAsJson”, olha só, ele já converte para JSON para nós.

[13:34] Vamos ver como é que isso fica na prática, damos uma atualizada aqui, olha só, atributo inválido e o atributo preços, se errarmos mais de um, “tipos”, ele vai adicionar aqui na lista, todos os atributos inválidos são os preços e os tipos, como pode ter mais de um, eu vou alterar aqui para o plural.

[14:03] Mas, ficou meio feio porque fica sem dizer que isso é efetivamente um erro, então vamos adicionar isso como um erro. Vamos converter para um JSON que tenha um pai, podemos fazer um envelope ou eu vou ensinar um novo método aqui que é, você cria um objeto JSON, “newObject” e você adiciona um campo chamado “erros” com o valor dos erros do formulário, eu vou guardar isso em uma variável, “erros” e eu vou retornar aqui, “erros”.

[14:49] Um objeto JSON, que tem a raiz erros e contenha os nossos erros do formulário. Vamos ver se ficou um pouco mais bonito, olha só, “erros” e aí quais são os erros? “Atributos inválidos” e os atributos que tínhamos.

[15:04] Mas e aquele bad request? Vamos conferir se foi mesmo um bad request, eu vou abrir aqui o nosso console e eu vou refazer a requisição, já estamos na aba Network, então vamos lá e vamos ver se ele lança, aqui, “400 bad request”.

[15:21] Agora nós já temos um filtro dinâmico permitindo que o cliente faça pesquisas livremente e, caso dê erro, nós já mostramos quais foram os erros que ele deu, quais foram os atributos errados que ele passou para nós e nós já retornamos tudo isso em formato JSON. Então agora terminamos a nossa API.

[15:40] No próximo vídeo, vamos fazer uma pequena revisão de todo o conteúdo do curso, para ver tudo que vimos até agora.