03

Paginação

Transcrição

[00:00] Vamos começar então. No último curso montamos nossa API REST usando Spring Boot. Basicamente, o que fizemos foi o CRUD, cadastro, listagem, alteração e exclusão de tópicos que são postados no fórum da Alura. Basicamente, a principal classe que temos é o TopicosController, e dentro está toda a lógica para fazer o CRUD do recurso de tópico. Temos vários métodos com as operações.

[00:33] Na aula de hoje, vamos fazer uma mudança no método lista. Esse método devolve todos os tópicos cadastrados no banco de dados. O problema é que no nosso banco de dados só temos três registros, por causa daquele arquivo que está no source/main/Resources, data.sql, que toda vez que rodamos o projeto o Spring executa. Eu tinha deixado fixo para ele sempre cadastrar três tópicos no banco de dados.

[01:00] Mas imagine que o banco de dados começa a crescer e daqui a pouco tem milhares de registros. Toda vez que esse método lista for chamado, vamos devolver todas essas informações do banco de dados, e com muitos registros pode começar a ficar lento.

[01:17] Para resolver esse problema, o pessoal usa a ideia de paginação, onde posso controlar. Ao invés de devolver todos os registros, posso devolver de pouco em pouco.

[01:38] Tínhamos colocado um parâmetro no método lista, que é o nome do curso, onde eu podia filtrar o curso pelo nome. Podemos seguir a mesma ide Já tenho que um parâmetro aqui, posso ter outros parâmetros, que no caso

serão usados para paginação. Posso ter um parâmetro int pagina, em que vou dizer em que página estou, e também preciso dizer a quantidade, quantos elementos quero por página. Se eu estiver na página um e a quantidade for dez, ele vai trazer os registros do primeiro ao décimo.

[02:18] Um detalhe importante. Nesse tipo de método, cujo mapeamento é do tipo GET, e temos parâmetros, que são aqueles parâmetros de url, diferente do método POST, que tem um parâmetro, mas está com @RequestBody, ou seja, o parâmetro vem no corpo da requisição, no formato JSON, no método lista esses parâmetros não vêm no corpo da requisição. Eles vêm na própria url, como parâmetros de url.

[02:47] No geral, para esse tipo de parâmetro é interessante colocar uma anotação, que é o @RequestParam, para avisar ao Spring que é um parâmetro de request. Automaticamente, quando você coloca essa anotação, o Spring considera que o parâmetro é obrigatório. Se chamarmos esse endereço e não passarmos o parâmetro, o Spring vai jogar um erro 400 para o cliente dizendo que tem um parâmetro obrigatório que ele não enviou.

[03:16] Só que no caso, o nome do curso vai ser opcional. Então, nessa anotação tem um atributo chamado required, que vai ser igual a falso.

[03:25] Já o int página e int quantidade também vou colocar o request param, para avisar para o Spring que são parâmetros de url, mas vou colocar como obrigatórios. Quero obrigar o cliente que está chamando a API a sempre passar os parâmetros de página e quantidade, para sempre termos a paginação.

[03:48] Nosso método agora tem três parâmetros, o nome do curso, a página e a quantidade. Na consulta não mudamos nada na implementação. Os parâmetros vão chegar, mas ele vai ignorar. Como faço para pegar esses parâmetros e na hora de chamar o repository, chamar o findall, ou o findbycurso, quero fazer a paginação.

[04:17] Se você já trabalhou com JPA, ela tem suporte para paginação. Se você usa aquela API do entitymanager, na hora de criar sua query tem métodos para setar o maxResult, firstResult, e aí você consegue controlar qual o primeiro registro e qual o número máximo de registros. Dessa maneira, você faria a paginação com JPA na mão. Mas aqui estamos usando o Spring data. Lembre-se que ele abstrai algumas coisinhas para nós, inclusive paginação.

[04:44] Se quisermos fazer paginação, o Spring data já tem um esquema pronto para facilitar nossa vida. Para fazer paginação, precisamos criar uma variável do tipo pageable, que é uma classe do Spring data que serve justamente para fazer paginação. Vou criar essa variável, vou chamar de paginação = page.

[05:07] Para criar essa interface pageable, vou precisar importar. Mas cuidado, tem três opções. A que queremos é a do org.springframework.data. Para criar esse cara, usamos outra classe chamada pageRequest. Nela, tem um método estático chamado of, em que passamos a página e a quantidade. Com isso, ele cria um objeto do tipo pageable.

[05:37] E aí o que eu faço com esse objeto pageable que está na minha variável paginação? Se você olhar o método findall, você vai ver que na verdade existem vários findall. Dentre eles têm um que recebe um pageable como parâmetro. Então, podemos passar esse paginação como parâmetro para o método, que aí o Spring data automaticamente vai saber que você quer fazer paginação e vai usar os parâmetros página e quantidade para saber qual o primeiro registro e quantos registros ele vai carregar para você. Então, não precisamos implementar nada da consulta com paginação. Ele faz isso automaticamente.

[06:20] Porém, como você pode observar, deu um erro de compilação. Quando usamos paginação, o retorno do método findall não é mais um list, porque o list apenas traz a lista com todos os tópicos, mas quando usamos paginação é interessante sabermos em qual página estamos no momento, quantas tenho no total, quantos registros tenho. Pense no cliente da nossa API REST. Ele dispara uma requisição usando paginação, mas recebe só os registros daquela paginação. Ele pode ficar meio perdido. Esse tipo de informação é muito útil

para o cliente. Por isso o Spring não devolve um list. Ele devolve outra classe chamada page, que tem um generics para você dizer qual é o tipo de classe com que esse page vai trabalhar.

[07:14] Dentro desse page tem a lista com os registros. Além dela, tem essas informações do número de páginas, qual a página atual, quantos elementos tem no total. Ele já dispara umas consultas para fazer o count de quantos registros tem no banco sozinho.

[07:33] Isso se entrou no if, se eu não trouxe o parâmetro do nome do curso. Se eu trouxer, nessa consulta também tenho que passar a paginação. Só que esse findbycursonome fomos nós que criamos. Então, vou entrar na classe tópicoRepository. Como é um método que nós definimos, temos que receber como parâmetro um pageable, que vou chamar de paginação. Já vou alterar o retorno, que não é mais list, é page. Vou importar o pageable e o page. Volto para o meu controller.

[08:18] Só que agora temos um problema. No dto o método converter não recebe um page de tópico. Ele recebe um list. Só que agora, como está usando paginação, vamos ter que alterar o método. Vou entrar no método, e ao invés de receber um list, vou trocar para receber um page. O retorno do método também vai mudar. Não posso devolver mais um list, tem que ser um page.

[08:44] Dá um problema na implementação, porque estávamos usando o API de strings, e agora não vai mais ficar assim. Só que aquela API de string vem do list do Java, mas agora estou recebendo um page. O pessoal do Spring também pensou nisso, e no tópicos existe um método chamado map, que é o mesmo map de strings do Java 8, e aí posso passar para ele que é para fazer um map utilizando o topicoDto::new. Ele vai pegar cada um dos registros que está dentro do page de tópico e transformar em um page de topicoDto.

[09:28] Faltou mudar o retorno do método, que agora não vai ser mais um list, vai ser um page.

[09:36] Vou iniciar o projeto. Sem nenhum erro, iniciou normalmente. No Postman, vamos fazer os testes. Vou disparar uma requisição do tipo GET para tópicos sem passar nenhum parâmetro. Ele me dá erro, como deveria, porque o parâmetro página é obrigatório. Eu tenho que passar topicos?pagina=1&qtd=1. Por exemplo, estou na página um e quero trazer apenas um registro. Vou disparar a requisição. Ele me devolve um JSON com content, e nesse content estão os dados. Vem outro parâmetro no JSON, pageable, com as informações da paginação.

[11:08] O cliente pode ler essas informações do JSON e montar o esquema de paginação automático, os números da página, tudo certo, que além dos dados ele tem as informações da paginação.

[11:20] Só um detalhe. Na verdade, o registro que veio coloquei quantidade um, então só veio um registro. Mas não veio o primeiro registro, veio o segundo, porque coloquei página um. Na verdade, ele começa no zero.

[11:40] Não deu erro no parâmetro nome curso porque coloquei como opcional. Se eu não filtrar pelo nome do curso, não tem problema, não é obrigatório. Só é obrigatório fazer o filtro pela paginação.

[11:52] Com isso, conseguimos fazer nossa paginação usando Spring Boot, Spring data, de maneira relativamente simples, nosso código ficou tranquilo, usando esse tal de pageable, e devolvendo page ao invés de list. No próximo vídeo vamos aprender a fazer ordenação também. Os registros estão voltando de acordo com a ordem que vem do banco de dados, pela chave primeira, mas e se eu quiser ordenar pela data, pelo status, ou pelo autor? No próximo vídeo vamos aprender como fazer ordenação usando Spring data.