04

Boas práticas no cadastro

Transcrição

[00:00] Continuando então. Fizemos o cadastro, está tudo ok. Funciona, ele me devolve o código 200. Só que tem um problema: não queremos deixar o método como `void` e nem devolver 200, embora a requisição - se não der nenhum erro, não estiver fora do ar o banco de dados - vai acontecer com sucesso. Mas segundo o modelo REST, existe a família de código HTTP, a família 200. E o 200 é o código de "ok" genérico. Ou seja, a requisição foi processada com sucesso, mas não significa nada demais.

```
@PostMapping
public void cadastrar(@RequestBody TopicoForm form) {
    Topico topico = form.converter(cursoRepository);
        topicoRepository.save(topico);
}
```

COPIAR CÓDIGO

[00:40] Quando estou cadastrando uma informação, postando uma nova informação no sistema, o ideal seria devolver outro código da família de sucesso, que é o código 201, de quando tenho uma requisição processada com sucesso, mas um novo recurso foi criado no servidor.

[01:04] O ideal seria que devolvêssemos 201, pois é o caso mais apropriado. Para devolver o código 201 no `Controller` do Spring, o método não pode ser `void`. Ele tem que ter outro retorno. E aí o Spring tem uma classe chamada `ResponseEntity<>` que recebe um *generic*. Esse *generic* é o tipo de objeto que vou devolver no corpo da resposta, que no caso, seria o `topico`. Mas lembre

se, nós não devolvemos a entidade, as classes de domínio, então na verdade vai ser um `topicoDto` (vamos devolver no corpo da resposta um DTO representando um recurso que acabou de ser criado).

```
@PostMapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
TopicoForm form) {
    Topico topico = form.converter(cursoRepository);
        topicoRepository.save(topico);
}
```

COPIAR CÓDIGO

[01:48] A partir deste momento, o método já não compila, porque preciso ter um retorno, e aí como faço para criar um `ResponseEntity` , dizer que é o código 201? Essa classe `ResponseEntity` tem alguns métodos estáticos para você criar um objeto `ResponseEntity` . Então, no `return` , teremos `ResponseEntity.created()` que devolve o 201. Só que esse método `created()` recebe um parâmetro `uri` e que eu preciso passar para ele.

```
@PostMapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
TopicoForm form) {
    Topico topico = form.converter(cursoRepository);
        topicoRepository.save(topico);

return ResponseEntity.created(uri)
}
```

COPIAR CÓDIGO

Isso acontece porque toda vez que devolvo 201 para o cliente, além de devolver o código, tenho que devolver mais duas coisas: uma delas é um cabeçalho HTTP, chamado "Location", com a URL desse novo recurso que acabou de ser criado; a segunda coisa é que, no corpo da resposta, eu tenho que devolver uma representação desse recurso que acabei de criar. Então, quando eu chamo

o método `created()`, ele fica esperando a `uri` do recurso que criamos para adicioná-la no cabeçalho `Location`.

[03:15] Esse `uri` que ele recebe é a classe URI do Java (vem do pacote `java.net`). Mas na hora em que crio um objeto `uri`, tenho que passar o caminho completo dessa `uri`: "http://localhost:8080/topico/id (http://localhost:8080/topico/id) do tópico". Para não ter que passar esse endereço completo - até porque quando eu colocar o sistema em produção não vai ser mais localhost: 8080 - o Spring vai nos ajudar novamente.

[03:46] No método `cadastrar()`, estou recebendo o `form` e posso colocar uma vírgula e depois uma classe do Spring chamada `UriComponentsBuilder`. Seguindo, vou chamar esse parâmetro de `uriBuilder`. É só declarar o `UriComponentsBuilder` como parâmetro que o Spring vai injetar no método para você.

```
@PostMapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
TopicoForm form, UriComponentsBuilder uriBuilder) {
    Topico topico = form.converter(cursoRepository);
        topicoRepository.save(topico);

URI uri =
return ResponseEntity.created(uri)
}
```

COPIAR CÓDIGO

[04:06] Nós vamos usar o `uriBuilder.` depois de `URI uri =`. Existem alguns métodos para ele criar um objeto `uri`. O método que vamos chamar é o `path()`. E aqui está o segredo: não vou passar o caminho completo, o caminho do servidor. Só vou passar o caminho do recurso.

[04:31] O caminho do recurso será `"/topicos/"`. Mas, se eu passar só `"/tópicos"`, é a lista de tópicos, sendo que aqui, na verdade, criei só um único

tópico. No geral, costumamos adicionar um `id`, que é dinâmico - é o `id` do tópico que acabei de criar. Coloco ele entre chaves para dizer que é um parâmetro dinâmico `{id}`.

Por último `.buildAndExpand()`, método que temos que chamar, passando como parâmetro um valor a ser substituído no espaço do `{id}`, que é dinâmico. No caso, vou puxar o `id` do tópico que acabei de criar no banco de dados e ele vai substituir esse `{id}` e jogar na `uri`.

No final, tem um método `.toUri()`, que converte e transforma na URL completa, com endereço do servidor e com valores dinâmicos que posso passar como parâmetro no `buildAndExpand`.

```
@PostMapping
public ResponseEntity<TopicoDto> cadastrar(@RequestBody
TopicoForm form, UriComponentsBuilder uriBuilder) {
    Topico topico = form.converter(cursoRepository);
        topicoRepository.save(topico);

URI uri =
uriBuilder.path("/topicos/{id}").buildAndExpand(topico.getId()

return ResponseEntity.created(uri)
}
```

COPIAR CÓDIGO

[05:30] Dessa maneira criamos o `uri`. É o `uri` que vou passar quando for chamar o método `created()`. Na sequência, `.body(body)`, porque tenho que passar o corpo - com o 201, além de devolver a `uri`, preciso devolver um "*body*", corpo, na resposta. A ideia é criarmos um DTO, então, `(new TopicoDto(topico)`. Lembrando que quando dou um `new` no `TopicoDto()` posso passar o `topico` como parâmetro. Dentro dele tem todas as informações que o DTO precisa. Feito isso, terminamos. `return`

`ResposeEntity.created(uri)` , com a `uri` do recurso que acabei de criar `"/topicos/{id}"` seguido do corpo da reposta com `TopicoDto()` .

```
URI uri =
uriBuilder.path("/topicos/{id}").buildAndExpand(topico.getId())

return ResponseEntity.created(uri).body(new
TopicoDto(topico));
}
```

COPIAR CÓDIGO

[06:13] O endereço `"/tópicos/{id}"` não existe ainda. A ideia é que, mais para frente, montemos essa nova `uri` do nosso projeto que vai ser o endereço para detalhar um tópico. Por enquanto não temos, mas quando tivermos, esse caminho que será devolvido quando crio um recurso, o endereço do recurso.

[06:38] Com isso agora estamos seguindo as boas práticas do modelo REST, devolvendo o código mais apropriado para esse cenário de criação de um novo recurso. Agora, só tem um detalhe. Como eu testo? Eu implementei o código, mas como eu sei que está funcionado? Até então estávamos testando pelo browser, né? Só que no browser só consigo testar requisições do tipo GET, que disparo pela barra de endereços do navegador. Mas para cadastrar um novo tópico, a requisição tem que ser do tipo POST.

[07:14] No próximo vídeo vamos ver que ferramenta posso utilizar para testar uma requisição do tipo POST na minha API REST.