04

Atualizando tópicos

Transcrição

[00:00] Continuando então. Fizemos o detalhe do tópico no último vídeo. Agora a próxima funcionalidade vai ser a de atualização. Como de costume, precisamos ter um novo método no nosso `TopicosController.java`, que vai ser o método responsável por atualizar um tópico. Vai ser parecido com o de `cadastrar()`. O método vai devolver um `ResponseEntity`, e vou chamá-lo de `atualizar()`.

```
public ResponseEntity<TopicoDto> atualizar()
```

COPIAR CÓDIGO

[00:36] Qual vai ser o mapeamento desse método? Não pode ser POST, porque já tenho o `"/tópicos"` com o método POST. Vamos utilizar o método PUT, `@PutMapping`. Na verdade, existe uma discussão. Quando vamos atualizar um recurso no modelo REST existem dois métodos para fazer atualização: o método PUT e o método PATCH. Os dois tem a ideia de atualização, mas o PUT seria para quando você quer sobrescrever o recurso. Isto é, criei um recurso com determinadas informações e quero sobrescrever aquele recurso inteiro - quero atualizar todas as informações. Já o PATCH seria a ideia de fazer apenas um pequeno "*patch*", uma pequena atualização - quando quero mudar só alguns campos.

```
@PutMapping()
public ResponseEntity<TopicoDto> atualizar() {

}
```

COPIAR CÓDIGO

[01:37] Só que na aplicação não consigo saber se ele está atualizando o recurso inteiro ou um ou outro campo. Só se eu fizesse essa verificação. Nós vamos utilizar o PUT, que é o mais comum do mercado.

[02:11] Na URL, preciso dizer também qual o recurso que quero atualizar, porque não são todos os tópicos. Eu vou passar o `id` do recurso que quero atualizar, `("/{id}")`, parecido com o que fizemos no `detalhar()`. Vou até copiar `@PathVariable`, porque precisaremos de um parâmetro `id` com um `@PathVariable`.

```
@PutMapping("/{id}")
public ReponseEntity<TopicoDto> atualizar(@PathVariable Long id)
{

    }
```

COPIAR CÓDIGO

[02:34] Só que nesse método `atualizar()`, além do `id`, vou precisar dos dados do tópico que quero atualizar. Ou seja, vou precisar receber um DTO com os dados, parecido com o que fizemos no `cadastrar()`. Só que no `cadastrar()` não estávamos usando o padrão de nomenclatura DTO e, sim, o padrão *Form*. No `atualizar()` também preciso receber o `TopicoForm form`, ele vem no corpo da requisição, e tem que ter validação, porque não posso atualizar com algo inválido. Então, vou copiar o `TopicoForm form` que vem do método `cadastrar()`.

```
@PutMapping("/{id}")
public ReponseEntity<TopicoDto> atualizar(@Pa  Variable Long id,
@RequestBody @Valid TopicoForm form) {
```

```
    }
```

COPIAR CÓDIGO

[03:11] Mas tem um detalhe. Eu não vou receber um `TopicoForm`, porque também posso querer ter flexibilidade. Tem algumas situações em que no cadastro preencho algumas informações, mas na hora de atualizar não posso atualizar todas as informações. Tem um ou outro campo que é somente leitura que não posso atualizar. Se eu receber o mesmo *Form*, nele poderiam vir campos que não deveria atualizar.

No geral, o ideal seria ter outra classe `Form` que representa a atualização. Vou criar outra classe, por exemplo, `AtualizacaoTopicoForm`, e nele só estarão os dados que quero fazer a atualização. Então, seleciono `AtualizacaoTopicoForm`, abro o atalho com o comando "Ctrl + 1", pressiono "Create class 'AtualizacaoTopicoForm'". Na próxima tela, aperto "Browser", depois "br.com.alura.forum.controller.form" e "OK". Agora basta pressionar "Finish".

[04:04] Agora, no `AtualizacaoTopicoForm.java`, colocarei só os campos que eu quero de fato atualizar. Vai ser um pouco parecido com o `TopicoForm.java`.

```
    @NotNull @NotEmpty @Length(min = 5)
        private String titulo;

        @NotNull @NotEmpty @Length(min = 10)
        private String mensagem;
}
```

COPIAR CÓDIGO

Só para exemplificar, ter o exemplo da variação, imagine que na atualização o usuário não pode mudar o curso, só o título ou a mensagem. Se ele criou o tópico naquele curso, é daquele curso e acabou. Por isso tenho dois *Forms*, um que só

permite alterar o título e a mensagem e outro que permite cadastrar todas as informações.

```
package br.com.alura.forum.controller.form;

import javax.validation.contraints.NoEmpty;

public class AtualizacaoTopicoForm {

    @NotNull @NotEmpty @Length(min = 5)
        private String titulo;

        @NotNull @NotEmpty @Length(min = 10)
        private String mensagem;
}
```

COPIAR CÓDIGO

[04:48] Aqui também vou precisar dos *Getters* e *Setters*. Vou gerar os *Getters* e *Setters* pelo atalho do Eclipse, apertando "Source > Generate Getters and Setters", depois, na próxima tela, "Sellect All" com "mensagem" e "titulo" selecionados. Para finalizar a operação, basta apertar "Generate".

Voltando para o `TopicosController.java`, chegou a requisição para atualizar um tópico, vai vir o `id` e o `AtualizacaoTopicoForm` com o título e a mensagem que quero atualizar. Agora posso implementar a lógica para atualização.

```
@PutMapping("/{id}")
public ResponseEntity<TopicoDto> atualizar(@PathVariable Long id,
@RequestBody @Valid AtualizacaoTopicoForm form) {

}
```

COPIAR CÓDIGO

[05:27] Nessa lógica, tenho que fazer o seguinte: está chegando como parâmetro o `id` do tópico que quero atualizar, então preciso carregá-lo do banco de dados. Depois preciso sobrescrever com as informações que foram enviadas pelo usuário, pelo cliente no `AtualizacaoTopicoForm`. Para não ficar com essa lógica no `Controller`, vou encapsular essa lógica dentro da classe *Form*. Eu vou chamar de `form.atualizar()`.

Vou ter um método em que passo as informações que não consigo ter dentro do *Form*, por exemplo, o `id`, que vai chegar no `Controller`. Vou precisar também do `topicoRepository`, porque no *Form* não consigo fazer injeção de dependência, e o método `form.atualizar()` vai atualizar e vai me devolver o `topico` como parâmetro, como resposta. Sendo que o tópico que vai ser devolvido é o tópico com as informações atualizadas.

```
@PutMapping("/{id}")
public ResponseEntity<TopicoDto> atualizar(@PathVariable Long id,
@RequestBody @Valid AtualizacaoTopicoForm form) {
    Topico topico = form.atualizar(id, topicoRepository);
}
```

COPIAR CÓDIGO

[06:33] Ele está reclamando, porque o método `atualizar()` não existe. Vou mandar ele criar selecionando `atualizar()` e, com o atalho, apertar "Create method 'atualizar(Long, TopicoRepository)'". Depois vou colocar a lógica de buscar o tópico pelo `id` no banco de dados e atualizar as informações que foram modificadas.

Para buscar um tópico pelo id, já vimos como fazer, `topicoRepository.getOne(id)`, passo o `id`, que chegou como parâmetro do método. Pego o tópico, que veio do banco de dados com as informações desatualizadas. Agor u sobrescrever tudo que eu quiser sobrescrever `topico.setTitulo()`, e passo o `this.titulo`, que é o que veio no JSON que o usuário enviou. Mesma coisa com a mensagem,

`topico.setmensagem(this.mensagem)` . E aí eu retorno o tópico que foi atualizado `return topico` .

```
public Topico atualizar(Long id, TopicoRepository
topicoRepository) {
    Topico topico = topicoRepository.getOne(id);

        topico.setTitulo(this.titulo);
        topico.setMensagem(this.mensagem);

        return topico;
}
```

COPIAR CÓDIGO

Então, nesse método `atualizar()` , chega o `id` do tópico, chega o `Repository` . Eu carreguei o tópico com as informações atuais, sobrescrevi com o que veio no JSON e devolvi o tópico atualizado para ser enviado para o para o `TopicosController.java` .

[07:46] Para atualizar no banco de dados, não precisamos chamar nenhum método do `Repository` , porque a partir do momento em que carreguei ele do banco de dados pelo `id` , pela JPA ele já está sendo gerenciado. Qualquer atributo que eu setar, no final do méotodo, o Spring roda dentro de uma transação. Então eu vou carregar o tópico do banco de dados, no final do método ele vai commitar a transação, a JPA vai detectar que foram alterados os atributos e ela vai disparar o update no banco de dados automaticamente, não preciso chamar.

[08:22] Quando chegar na linha `Topico topico = form.atualizar(id, topicoRepository)` , ele já atualizou em memória e, quando acabar o método, ele atualiza no banco de dados.

[08:30] Preciso devolver uma resposta. No caso, vou usar a classe `ResponseEntity` para montar uma resposta. Eu não estou criando um novo recurso, então não vou

chamar o método `created()`, vou chamar só o `ok()`, e aí passo como parâmetro um `new TopicoDto(topico)`, com esse tópico que foi atualizado. Esse parâmetro do `ok()` é o corpo que vai ser devolvido como resposta pelo servidor.

```
return ResponseEntity.ok(new TopicoDto(topico));
```

COPIAR CÓDIGO

[09:03] Vamos testar agora no Postman. No primeiro campo (que está no na parte centro esqueda da tela) vou trocar o método de "GET" para "PUT" e quero atualizar o tópico de id "3". Só para verificar antes, vou disparar uma requisição "GET" para trazer os dados do tópico de id "3". No banco de dados está assim:

```
{
    "id": 3,
        "titulo": "Dúvida 3",
        "mensagem": "Tag HTML",
        "dataCriacao": "2019-05-05T20:00:00",
        "nomeAutor": "Aluno",
        "status": "NAO_RESPONDIDO",
        "repostas": []
}
```

COPIAR CÓDIGO

[09:32] Agora vou trocar o primeiro campo de "GET" para "PUT". No campo a frente, o endereço é: http://localhost:8080/topicos/3 (http://localhost:8080/topicos/3). Logo abaixo, no "Body", lembre-se que temos que selecionar "raw" e mandar o JSON que vai ser enviado para o servidor. No JSON, só posso mandar dois campos: `"titulo"` (o novo título é `“Atualizado”`); e a `"mensagem"` (que agora é `ensagem nova”`).

```
{
    "titulo":"Atualizado",
        "mensagem":"Mensagem nova"
}
```

COPIAR CÓDIGO

Além disso, precisamos mandar o cabeçalho, que é o "Content:Type" (Na parte centro esquerda da tela, encontraremos o campo "Headers", nele precisamos selecionar "Content-Type"). Preencheremos o campo "Value" - que está à frente do campo "Headers" - com "application/json" para informar ao servidor que estou mandando os dados no formato JSON.

[10:30] Disparando a requisição (apertando "Send"), ele devolveu o código "200 OK" e, como retorno, o JSON foi devolvido com as informações atualizadas. Mas será que atualizou mesmo no banco de dados?

```
{
    "id": 3,
        "titulo": "Atualizado",
        "mensagem": "Mensagem nova",
        "dataCriacao": "2019-05-05T20:00:00"
}
```

COPIAR CÓDIGO

Só para garantir, vou disparar a requisição "GET" de novo para http://localhost:8080/topicos/3 (http://localhost:8080/topicos/3) . De fato, ele não atualizou (está mostrando os valores antigos).

```
{
    "id": 3,
        "titulo": "Dúvida 3",
```

```
        "mensagem": "Tag HTML",
        "dataCriacao": "2019-05-05T20:00:00",
        "nomeAutor": "Aluno",
        "status": "NAO_RESPONDIDO",
        "repostas": []
}
```

COPIAR CÓDIGO

[11:20] Vamos até o `AtualizacaoTopicoForm.java` analisar o que aconteceu. Eu carreguei o tópico pelo `id`, setei o título e a mensagem, devolvi o tópico para o `Controller`, devolvi o DTO com o tópico atualizado. Parece que vamos precisar colocar, no `TopicosController.java`, o `@Transactional`, que é para avisar para o Spring que é para commitar a transação no final do método.

```
@PutMapping("/{id}")
@Transactional
public ResponseEntity<TopicoDto> atualizar(@PathVariable Long id,
@RequestBody @Valid AtualizacaoTopicoForm form) {
    Topico topico = form.atualizar(id, topicoRepository);

        return ResponseEntity.ok(new TopicoDto(topico));
}
```

COPIAR CÓDIGO

Vamos voltar no Postman e disparar a requisição "PUT" com endereço http://localhost:8080/topicos/3 (http://localhost:8080/topicos/3). O "Body" será `"Atualizado novamente"`, e a mensagem agora `"Mensagem nova 2"`. Está com o cabeçalho "Content-Type", vou disparar com a requisição "GET" pressionando "Send" e ele foi atualizado de fato no banco de dados.

```
{
    "id": 3,
```

```
    "titulo": "Atualizado novamente",
    "mensagem": "Mensagem nova 2",
    "dataCriacao": "2019-05T20:00:00",
    "nomeAutor": "Aluno",
    "status": "NAO_RESPONDIDO",
    "respostas": []
}
```

COPIAR CÓDIGO

[12:17] Com isso, finalizamos a funcionalidade para atualizar o tópico no banco de dados utilizando outro *Form*, `AtualizacaoTopicoForm`, para ter essa separação do que eu cadastro e do que atualizo, porque às vezes pode ser que você tenha menos campos que podem ser atualizados. Se você usar o mesmo *Form*, você tem que tomar cuidado, porque o cliente pode estar mandando campos que ele não pode atualizar.

[12:44] Com isso, fechamos a atualização. No próximo vídeo vamos fazer o DELETE, para fazer a exclusão, por exemplo, querer apagar um tópico que cadastrei errado.