08

Tratando o erro 404

Transcrição

[00:00] Nós implementamos então a parte de excluir (a remoção do tópico) no último vídeo. Nesta aula, vamos só fazer um tratamento no caso do usuário passar um id inválido, tanto no método `remover()`, `atualizar()` e `detalhar()`, que são os métodos que recebem um id na barra de endereços.

[00:22] Para entendermos o que acontece, vamos fazer um teste. Eu vou disparar no Postman uma requisição "GET" para o http://localhost:8080/topicos/3 (http://localhost:8080/topicos/3). Existe um tópico de id "3" no banco de dados, então ele carregou os dados corretamente.

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

COPIAR CÓDIGO

Mas e se eu passar por exemplo http://localhost:8080/topicos/999 (http://localhost:8080/topicos/999)? A URL é válida, mas não existe um tópico cc esse id no banco de dados. Disparando a requisição, ele vai dar um erro (deu um

erro no servidor e esse erro voltou para o cliente) e voltará uma *stack trace*. Não deveríamos devolver esse erro para o usuário.

[01:10] Isso aconteceu porque no exemplo anterior estávamos chamando o método `detalhar()`. O erro aconteceu exatamente na linha `Topico topico = topicoRepository.getOne(id)`. Quando usamos o `topicoRepository` para buscar pelo `id`, estávamos utilizando o método `getOne()`. O `getOne()` funciona assim: passamos um `id` como parâmetro e ele considera que existe um registro, um tópico com esse `id` no banco de dados. Se ele não encontrar esse registro, ele não devolve nulo. Ele joga uma *exception*. Aconteceu uma exception no meio do caminho e essa *exception* vazou para o cliente.

```
@GetMapping("/{id}")
public DetalhesDoTopicoDto detalhar(@PathVariable Long id) {
    Topico topico = topicoRepository.getOne(id);
        return new DetalhesDoTopicoDto(topico);
    }
```

COPIAR CÓDIGO

[01:48] O ideal aqui seria fazermos um tratamento. Antes de `detalhar()`, `atualizar()` e `remover()`, primeiro vou verificar se existe um tópico com esse `id`. Se existir, ok, executo. Se não existir, ao invés de voltar um erro imenso, devolvo "404", que é o código de "Not Found" - e que faz sentido neste caso. O cliente mandou uma URL com o id de um recurso que não existe, que não foi encontrado.

[02:16] Para fazer isso, vamos fazer a seguinte alteração. No `detalhar()`, não vou mais usar o método `getOne()`, porque ele, por padrão, se não achar o elemento vai dar erro. Até poderia fazer um *Try/Catch*, se cair no *catch* devolver o "404", mas tem outro jeito. Existe outro método nos *Repositorys* chamado `findById()`. Ele também tem a mesma lógica. Você passa um `id` e ele vai fazer uma consul

filtrando pelo `id` no banco de dados, só que se ele não encontrar, ele não joga *exception*.

```
@GetMapping("/{id}")
public DetalhesDoTopicoDto detalhar(@PathVariable Long id) {
    Topico topico = topicoRepository.findById(id);
        return new DetalhesDoTopicoDto(topico);
    }
```

COPIAR CÓDIGO

[02:52] Está até dando erro de compilação, porque o retorno do `findById()` não é a entidade `topico`. Ele devolve um `Optional`, que é uma classe que veio na API do Java 8. Tenho que mudar meu retorno para ser um `Optional<topico>` (isto é, para ser um *Optional* de tópico). Vou importar o `Optional` (abrindo o atalho, em "Choose type to import", seleciono "java.util.Optional" e ele devolverá um `Optional`).

```
@GetMapping("/{id}")
public DetalhesDoTopicoDto detalhar(@PathVariable Long id) {
    Optional<Topico> topico = topicoRepository.findById(id);
        return new DetalhesDoTopicoDto(topico);
    }
```

COPIAR CÓDIGO

[03:24] Como o próprio nome já diz, o `Optional` é opcional. Tenho que verificar se nesse `Optional` tem um registro. Se não tiver, devolvo "404". Se tiver, eu retorno esse `TopicoDto`, conforme estava funcionando antes. Então vou fazer um `if`. Ou seja, `if (topico.isPresent())`. Se existe um registro de fato presente, vou retornar um `return new DetalhesDoTopicoDto(topico)`, passando como parâmetro `topico`.

```
@GetMapping("/{id}")
public DetalhesDoTopicoDto detalhar(@PathVariable Long id) {
    Optional<Topico> topico = topicoRepository.findById(id);
        if (topico.isPresent()) {
        return new DetalhesDoTopicoDto(topico);
    }
}
```

COPIAR CÓDIGO

[04:06] Vai dar erro de compilação, porque o parâmetro `topico` agora é o `optional`, mas, nesse caso, não quero o `optional`. Então, tenho que chamar o método `get()`, que é para pegar o `topico` de fato que está dentro do `Optional`. Enão, eu só vou chamar o `get()` dentro do `if (topico.isPresent())`, isto é, se estiver presente.

```
@GetMapping("/{id}")
public DetalhesDoTopicoDto detalhar(@PathVariable Long id) {
    Optional<Topico> topico = topicoRepository.findById(id);
        if (topico.isPresent()) {
        return new DetalhesDoTopicoDto(topico.get());
    }
}
```

COPIAR CÓDIGO

[04:28] Se entrou no `if`, `if (topico.isPresent())`, vai devolver o DTO. Se não entrou, é porque ele não está presente. Ou seja, não encontrei um tópico com esse id. Se chegou, preciso devolver o "404". Para fazer isso, vou retornar o `ResponseEntity.notFound().build` para montar o objeto `ResponseEntity`.

```
return ResponseEntity.notFound().build();
```

COPIAR CÓDIGO

[04:50] Só que agora vai dar um problema, porque o método `detalhar()` está devolvendo um `DetalhesDoTopicoDto`, não o `ResponseEntity`. Para eu devolver um `notFound()`, preciso que o retorno do método seja o `ResponseEntity`. Então precisamos alterar para que ele devolva um `ResponseEntity` de `<DetalhesDoTopicoDto>`.

```
@GetMapping("/{id}")
public ResponseEntity<DetalhesDoTopicoDto>
detalhar(@PathVariable Long id) {
    Optional<Topico> topico = topicoRepository.findById(id);
        if (topico.isPresent()) {
        return new DetalhesDoTopicoDto(topico.get());
    }
}
```

COPIAR CÓDIGO

Essa parte funcionou, mas agora está dando problema no `if`, porque dentro do `if` não estou devolvendo um `ResponseEntity`. Então, vou devolver um `ResponseEntity` e, no caso do `detalhar()`, vou devolver um `ok()`, e como parâmetro do `ok()`, no corpo da requisição é que mando o DTO.

```
if (topico.isPresent()) {
    return ResponseEntity.ok(new
DetalhesDoTopicoDto(topiico.get());
}
```

COPIAR CÓDIGO

[05:36] Fiz essa alteração, salvei, vamos testar voltando ao Postman. Primeiro vou testar um cenário correto, em que passei um id válido, http://localhost:8080/topicos/3 (http://localhost:8080/topicos/3) e ele devolveu os detalhes do tópico "3" normalmente. Agora, se eu passar o "999", que não existe,

http://localhost:8080/topicos/999 (http://localhost:8080/topicos/999), no "Status" teremos "404" e nada aparece no corpo, porque ele não encontrou o recurso. Pereceberemos que ele não devolveu aquela *stack trace* imensa, devolveu o "404", que é o jeito apropriado.

[06:07] A ideia agora é fazer a mesma coisa, tanto no `atualizar()` quanto no `remover()`. Vou até copiar algumas linhas de código, e depois vou adaptar.

```
Optional<Topico> topico = topicoRepository.findById(id);
if (topico.isPresent()) {
    return ResponseEntity.ok(new
DetalhesDoTopicoDto(topico.get()));
}

return ResponseEntity.notFound().build();
```

COPIAR CÓDIGO

Primeiro, vamos adaptar para o método `atualizar()`. Nele, busquei por `id`, se eu encontrei, vou só trocar o `return`: pegar o `return ResponseEntity.ok(new TopicoDto(topico))` e substituir pelo `return ResponseEntity.ok(new DetalhesDoTopicoDto(topioc.get()))`. Preciso fazer também o `atualizar`, colocando a linha `Topico topico = form.atualizar(id, topicoRepository)` acima do `return`. Agora basta corrigir um último probema, a variável `topico` deverá ser trocada por `optional`. Isto é, `optional.isPresent()`, que busca pelo `id` ou devolve "404".

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

```
            Topico topico = form.atualizar(id, topicoRepository);
            return ResponseEntity.ok(new TopicoDto(topico));
        }

        return ResponseEntity.notFound().build();
    }
```

COPIAR CÓDIGO

No `remover()`, vou fazer a mesma coisa. Primeiro, vou verificar: fazer a consulta pelo `optional`, trazendo a linha `Optional<Topico> optional = topicoRepository.findById(id);`. Verificar se o `Optional` está presente, `if (optional.isPresent())`. Se estiver presente, então posso trazer as linhas `topicoRepository.deleteById(id)` e `return ResponseEntity.ok().build()`. Se não estiver presente, vou devolver um "404", como nos outros métodos. Agora basta colar o `return ReponseEntity.notFound().build()` e salvar.

```
    @DeleteMapping("/{id}")
    @Transactional
    public ResponseEntity<?> remover(@PathVariable Long id) {
        Optional<Topico> optional = topicoRepository.findById(id);
            if (optional.isPresent()) {
                topicoRepository.deletById(id);
                    return ResponseEntity.ok().build();
              }
            return ReponseEntity.notFound().build();

        }
```

COPIAR CÓDIGO

Agora os três métodos estão fazendo o tratamento quando o id é inválido.Vamos testar no Postman com o método "DELETE", passando para excluir o tópico de id "1", http://localhost:8080/topicos/1 (http://localhost:8080/topicos/1). Deu certo: c

"Status" mostra "200 OK". Se eu mandar o "1" de novo, já não existe, por isso, agora o "Status" mostra "404 Not Found".

[07:53] Essa foi só uma melhoria, um ajuste, para fazer esse tratamento no caso de mandar um id que não existe. Ao invés de dar um erro, devolvo o "404". Com isso finalizamos nossa API.