Tecnologias Para Back-End

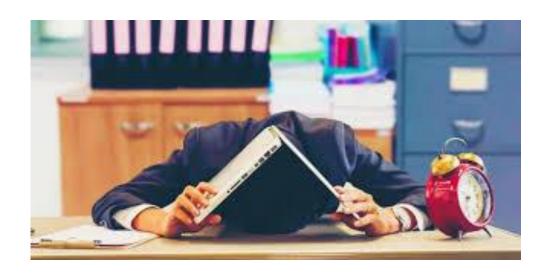
Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 15 – LIDANDO COM ERROS NA API – CONTINUAÇÃO......

Lidando com Erros na API – Erro 400





- Por fim, o último erro que vamos tratar na API será o erro 400.
- Código ou erro 400: indica que o servidor não conseguiu processar uma requisição por erro de validação nos dados enviados pelo cliente.
- Voltando ao Insomnia, isso pode acontecer tanto no cadastro de médicos quanto de pacientes.
- Vamos clicar em "Cadastro de médico" do lado esquerdo.
- Neste método temos o endereço http://localhost:8080/medicos com o verbo post, e no corpo do JSON os seguintes dados:

```
"nome": "Renato Amoedo",
"email": "renato.amoedo@voll.med",
"crm": "333444",
"telefone": "61999998888",
"especialidade": "ORTOPEDIA",
"endereco": {
    "logradouro": "rua 1",
    "bairro": "bairro",
    "cep": "12345678",
    "cidade": "Brasilia",
    "uf": "DF"
```

Estamos usando o bean validation na API para realizar as validações dos campos obrigatórios, entre outros dados.

 Logo, se removermos os campos nome, email, crm e telefone do JSON e dispararmos a requisição com os campos de especialidade e endereco,

deveria retornar o código 400.

```
"nome": "Renato Amoedo",
"email": "renato.amoedo@voll.med",
"crm": "333444",
"telefone": "61999998888",
```

- Isso porque todos os campos excluídos são obrigatórios, como not null.
- Após essa remoção, clicaremos no botão "Send". Note que foi devolvido o erro 400 Bad Request. Por padrão, ao acontecer um erro 400, o Spring faz a integração com o bean validation, e roda as validações.

Caso ocorra alguma falha, é devolvido o código 400 e no JSON da resposta
nos envia vários campos, como timestamp, status, error, message, entre
outros campos. Há, também, um objeto chamado errors, que contém o array
com cada erro de validação ocorrido.

```
"codes": [
    "NotBlank.dadosCadastroMedico.telefone"
    "NotBlank.telefone",
    "NotBlank.java.lang.string",
    "NotBlank"
],
```

- No log do servidor veja o nome da exceção.
- Na última linha

```
.w.s.m.s.DefaultHandlerExceptionResolver
```

- Bem, já sabemos o nome da EXCEÇÃO e os ERROS.
- Contudo esse retorno poderia ser mais SIMPLIFICADO
- Podemos retornar um JSON com uma lista dos campos que geraram erro e,
 para cada campo, devolver qual o campo e a mensagem de erro.
- Por exemplo: "campo nome é obrigatório", "o e-mail está com formato inválido", etc.

- Esses dados bastam para o usuário da nossa API, já que ele saberá qual o campo inválido e o motivo.
- Esse será o tratamento que faremos na API.
- Voltando ao IntelliJ, incluiremos um novo método na classe TratadorDeErros.

```
public class TratadosDeErros {
    no usages
    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity tartarErro404(){
        return ResponseEntity.notFound().build();
```

Após o penúltimo fecha chaves do return do método tratarErro404(),
 criaremos mais um método chamado tratarErro400().

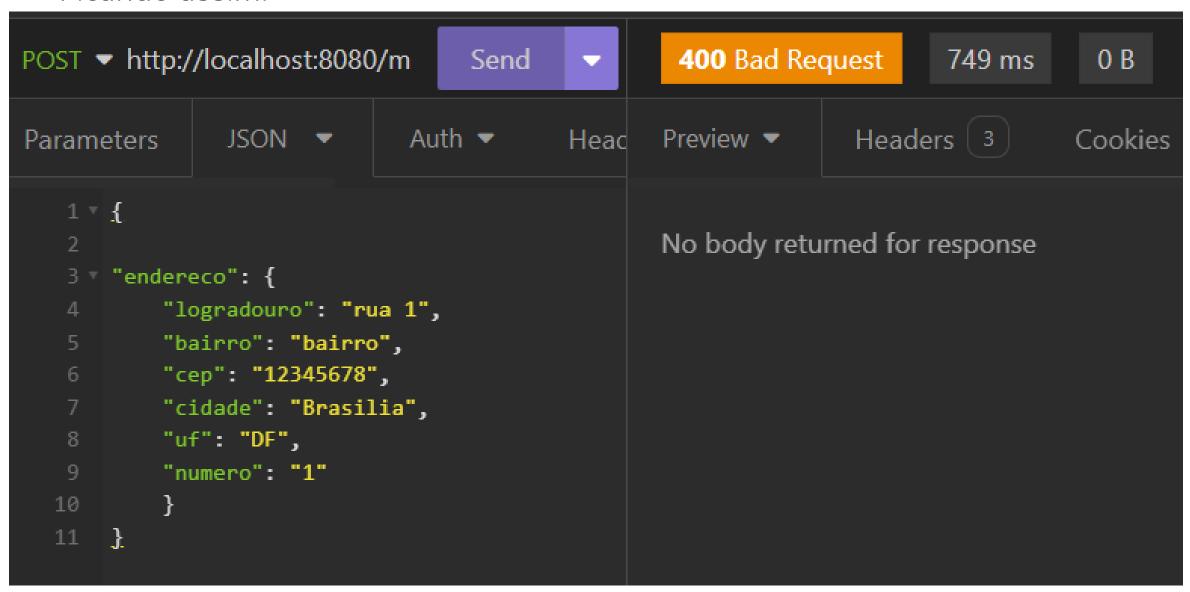
```
@ExceptionHandler(EntityNotFoundException.class)
public ResponseEntity tartarErro404(){
    return ResponseEntity.notFound().build();
no usages
public ResponseEntity trata
    return
```

Ficando assim.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400() {
    return ResponseEntity.badRequest().build();
}
```

• Salve e teste...Veja como ficou a resposta no Insomnia

Ficando assim.



- Ao dispararmos arequisição, nos é devolvido o código 400 Bad Request.
- No entanto, em "Preview", n\u00e3o retornou nenhum dado, somente a mensagem "No body returned for response".
- Conseguimos fazer o tratamento personalizado, mas faltou o corpo da resposta.
- Precisamos mostrar agora para o usuário o erro.
- Para levar um corpo com as informações, no método tratarErro400(), precisamos passar o objeto que desejamos retornar no parênteses do bad request().

- Porém, diferente do código 404, no 400 é necessário sabermos quais erros ocorreram.
- Portanto, precisamos capturar a exceção lançada, pois é nela que teremos acesso a quais campos estão inválidos conforme as regras do bean validation.
- Teremos que pegar a exception lançada, e passar como parâmetro no método tratarErro400() e acrescentar o parâmetro **ex** (abreviação de exception).

- Porém, diferente do código 404, no 400 é necessário sabermos quais erros ocorreram.
- Portanto, precisamos capturar a exceção lançada, pois é nela que teremos acesso a quais campos estão inválidos conforme as regras do bean validation.
- Teremos que pegar a exception lançada, e passar como parâmetro no método tratarErro400() e acrescentar o parâmetro ex (abreviação de exception).

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400 (MethodArgumentNotValidException ex) {
    return ResponseEntity.badRequest().build();
}
```

- Para capturar os erros que ocorreram no bean validation,
- Existe um objeto exception, que possui um método que retorna a lista com os campos inválidos.
- Vamos colocar isso em uma variável chamada erros dentro do método, antes do return

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400(MethodArgumentNotValidException ex) {
    var erros = ex.getFieldErrors();
    return ResponseEntity.badRequest().build();
}
```

- Salve e execute.
- Ele irá devolver o mesmo erro do JSON que foi devolvido anteriormente, com informações desnecessárias.
- No caso, desejamos retornar somente o nome do campo e a mensagem.
- Para isso, vamos personalizar aquela lista, transformando-a em um DTO.
- Assim como fizemos na listagem de médicos, precisaremos criar um DTO que contém somente os campos e as mensagens.
- E teremos que converter a lista do bean validation, para a lista do nosso DTO.

- Voltando para o IntelliJ para criar o DTO record.
- Contudo, ao invés de criarmos em um arquivo separado, montaremos dentro da classe TratadorDeErros.

```
private record DadosErroValidacao(String campo, String mensagem) {
}
```

- Agora, no body do método tratarErro400 não passaremos mais a lista de erros, precisamos convertê-la para uma lista de dados, erro e validação.
- No parêntese do body passaremos erros.stream().map().
- Essa parte solicita para erros me dê um stream e mapeie cada objeto FieldError para um objeto DadosErroValidacao.
- Em DadosErroValidacao acrescentamos ::new para chamar o construtor.
- Por fim, usaremos o método .toList() para convertermos para uma lista.

- Erro de compilação. Para isso, vamos declarar um construtor em record.
- Na linha seguinte colocaremos public DadosErroValidacao() que vai receber um objeto do tipo FieldError chamado erro.

```
private record DadosErroValidacao(String campo, String mensagem) {
    1usage
    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```

- Chamaremos o construtor padrão
 - do record passando erro.getField() e erro.getDefaultMessage().
 - erro.getField(): nos devolve o nome do campo
 - erro.getDefaultMessage(): nos devolve a mensagem para um campo específico.

- Deste modo, conseguimos converter a lista de FieldError para uma lista de DadosErroValidacao.
- Pronto!!! Vamos testar...



