

Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 14 – LIDANDO COM ERROS NA API

Lidando com Erros na API



Spring

- Fizemos as alterações **no controller**, para devolver os códigos do protocolo HTTP de forma adequada, conforme a funcionalidade. E, também, padronizamos para retornar Response Entity nos métodos do controller.
- **Cadastro** devolve **201 Created** (Cabeçalho , Corpo)
- **Excluir** devolve **204 No Content**.(Sem Corpo).
- **Alteração** devolve **200 OK** (Com Corpo)
- **Exibir e Listagem** devolve **200 OK** (Com Corpo)

Spring

- Continuaremos com as melhorias referente à questão do protocolo HTTP e dos códigos HTTP devolvidos.
- Contudo, agora, vamos analisar outros cenários.
- No **Insomnia**, temos a requisição "Detalhar médico" em que disparamos uma **requisição do tipo get** para o endereço **http://localhost:8080/medicos/6**, passando como parâmetro o ID.
- Por exemplo, no caso, temos a **médica Juliana Queiroz** como sendo a médica correspondente ao **ID número 6** do banco de dados.
- Logo, se dispararmos essa **requisição /medicos/6**, é retornado um JSON com as informações da Juliana, em "Preview".

Spring

- Porém, e se passarmos um ID que não possui registro no banco de dados após o `medicos/`?
- Por exemplo, vamos alterar o ID para `6999`
- `http://localhost:8080/medicos/6999`
- Em seguida, **cliquemos no botão "Send"**. Note que o código devolvido foi o `500 Internal Server Error` e, em "Preview", temos uma mensagem com os campos timestamp, status, error e trace.

- Um erro parecido assim...

```
{  
  "timestamp": "2024-05-19T21:42:15.460+00:00",  
  "status": 500,  
  "error": "Internal Server Error",  
  "trace":  
    "jakarta.persistence.EntityNotFoundException:  
    Unable to find med.voll.api.medico.Medico with id  
    666\r\n\tat  
    org.hibernate.jpa.boot.internal.EntityManagerFact  
    oryBuilderImpl$JpaEntityNotFoundDelegate.handleEn
```

Código ou erro 500: Erro no servidor interno.

Spring

- No caso, seria um erro na API back-end. Contudo, o que o Spring faz quando ocorre um erro no código do back-end? Por padrão, ele retorna essas informações em formato JSON que podemos visualizar na aba "Preview".
- Nas informações do JSON há um objeto com algumas propriedades. A primeira é o timestamp que comunica data e hora do erro, o status que é o código HTTP 500 e o error é o nome do erro referente ao código.
- Por fim, a propriedade trace, sendo a stack trace do erro - sendo a mesma exibida no console, informando qual a exception que ocorreu. Nesta propriedade, temos a mensagem "Unable to find med.voll.api.medico.Medico with id 6999", que significa que o ID que digitamos não existe no banco de dados.

Spring

- No caso, **seria um erro** na **API back-end**.
- **Contudo, o que o Spring faz quando ocorre um erro** no **código do back-end**?
- **Por padrão**, **ele retorna essas informações em formato JSON** que podemos visualizar na aba **"Preview"**.
- Nas informações do JSON há um objeto com algumas propriedades.
- A primeira é o **timestamp** que **comunica data e hora do erro**, o **status** que é o **código HTTP 500** e o **error** é **o nome do erro referente ao código**.
- O erro mostrado pela propriedade do **trace**, sendo a **stack trace do erro**.

Spring

```
"message": "Unable to find  
med.voll.api.medico.Medico with id 666",  
"path": "/medicos/666"  
}
```

- Nesta propriedade, temos a mensagem "Unable to find med.voll.api.medico.Medico with id 6999", que significa que o ID que digitamos **não existe no banco de dados**.

Spring

- Nesta propriedade, temos a mensagem "Unable to find med.voll.api.medico.Medico with id 6999", que significa que o ID que digitamos **não existe no banco de dados**.
- Não é uma boa prática devolver uma **trace de erro** para **o cliente**.
- Estamos expondo nomes **de classes, métodos**. Nisso pode haver informações **sensíveis, o que pode se tornar uma brecha de segurança**.
- Desejamos padronizar as mensagens devolvidas pela API quando ocorre erro. Há diversas situações de erro que podem ocorrer da nossa API, nem sempre tudo funcionará conforme o esperado.

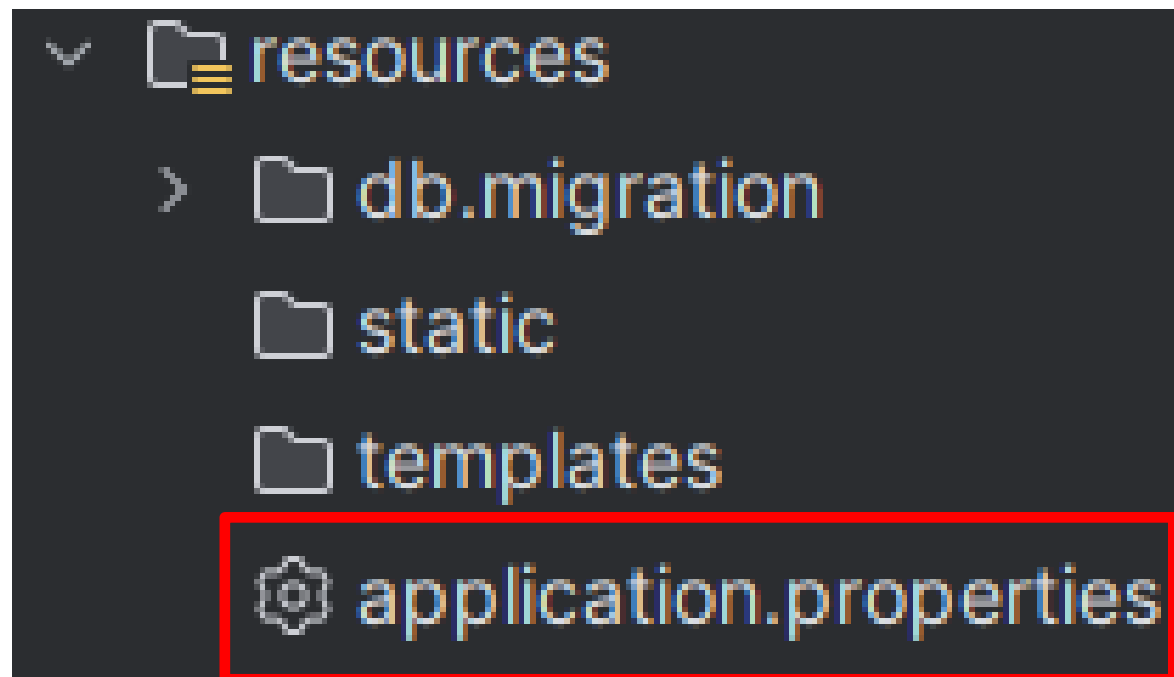
Spring

- Por isso, em situações de erros é uma boa prática colocar respostas mais apropriadas para esses cenários.
- Para ajustar isso, voltaremos ao Insomnia para analisarmos uma coisa. Perceba que o próprio tratamento do Spring já traz as informações que desejamos, porém, a **propriedade trace** nos devolve dados desnecessários e sensíveis.

```
"timestamp": "2024-05-19T21:42:15.460+00:00",  
"status": 500,  
"error": "Internal Server Error",  
"trace":  
"jakarta.persistence.EntityNotFoundException:"
```

Spring

- O **JSON** devolvido no Insomnia **é o padrão do Spring Boot**, mas podemos configurar no projeto para não retornar os dados do campo trace.
- Voltando ao IntelliJ, no arquivo **application.properties** em "**src** > **main** > **resources**".



Spring

- Nele temos a configuração do BD.
- Agora, incluiremos mais uma propriedade para o Spring não enviar a stack trace em caso de erro.
- Para sabermos quais as propriedades do Spring Boot, **basta consultar a documentação.**
- <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>
- A propriedade que buscamos está relacionada a servidores, item 11 da documentação, que esta no menu.

Spring

- Se clicarmos no item 11 em "Server Properties", do lado esquerdo da documentação.
- Nela, temos a propriedades ideal para o nosso caso, sendo a `server.error.include-stacktrace`.

Name	Description	Default Value
<code>server.error.include-stacktrace</code>	When to include the "trace" attribute.	<code>never</code>

Spring

- Vamos inserir esse propriedade na ultima linha no arquivo `application.properties`

```
server.error.include-stacktrace=never
```

- Basta salvar.
- Retorne para o Insomnia.
- Tente realizar a requisição `Detalhar Medico`.
- Passe na requisição em médico que não existe.

Spring

- Perceba que o JSON, não retornou o campo trace.

```
{  
  "timestamp": "2024-05-19T22:35:00.494+00:00",  
  "status": 500,  
  "error": "Internal Server Error",  
  "message": "Unable to find  
med.voll.api.medico.Medico with id 666",  
  "path": "/medicos/666"  
}
```

- Configuração realizada com SUCESSO!!!!
- O cliente da nossa API, basta saber que deu um erro 500.

Spring

- O código 500 ocorreu porque estamos usando o Spring Data, e fazendo uma consulta no banco de dados com a interface repository.
- E o padrão do Spring Data JPA ao passar um ID inexistente é retornar uma exception. Assim, ao gerar uma exception, nos retorna o código ou erro 500.
- Podemos configurar isso no nosso projeto.
- Em casos de determinadas exceptions, não retornar erro 500 e sim o 404.
- Vamos personalizar o tratamento de erros na API.



Spring

Tratando Erro 404

Spring

- Retorne ao IntelliJ no arquivo `MedicoController`.
- O método que está sendo chamado nessa **requisição é o detalhar**.
- Neste método, ele chama o `repository.getReferenceById(id)`.

```
@GetMapping("/{id}")  
public ResponseEntity detalhar(@PathVariable Long id){  
    var medico = repository.getReferenceById(id);  
    return ResponseEntity.ok(new DadosDetalhamentoMedico(medico));  
}
```

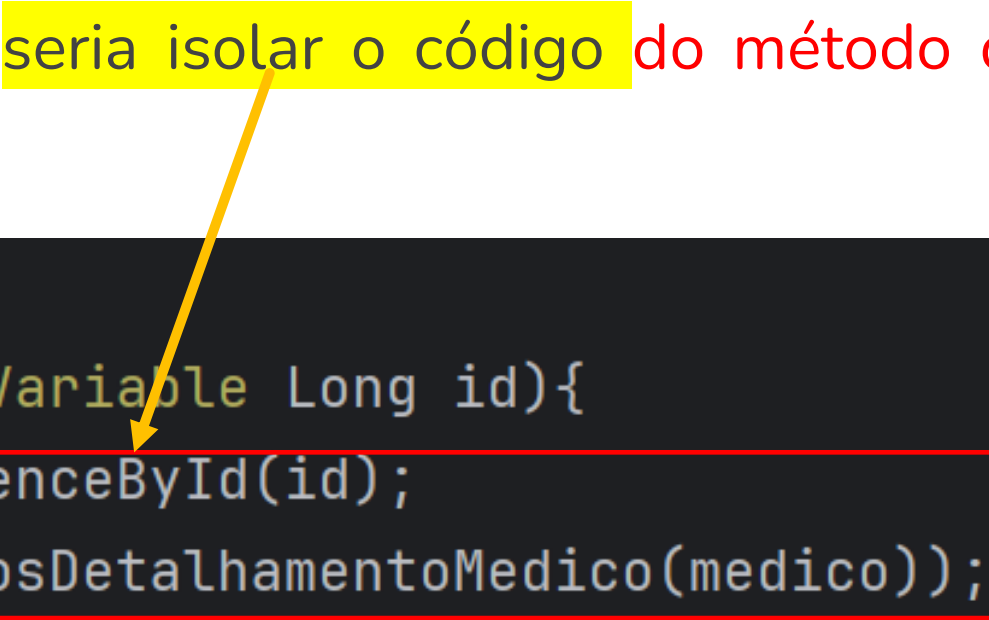
Spring

- É justamente nesta linha que está o problema.
- O método `getReferenceById()`, ao passarmos um **ID inexistente** na tabela do banco de dados, **ele envia uma exception** do tipo `EntityNotFoundException`.
- E perceba que **não fizemos o tratamento**, **não temos o try-catch**.
- Assim, **a exception aconteceu nessa linha** e **foi lançada para o Spring**.
- **O Spring**, **por padrão**, **se ocorrer uma exception** que **não foi tratada no código**, **ele trata como uma exceção gerando o erro 500**.

Spring

- No caso, não desejamos esse comportamento padrão, queremos que ao dar essa exception em específico, seja devolvido o erro 404.
- Uma forma de lidar com essa situação, seria isolar o código do método de detalhar, dentro de um try-catch.

```
@GetMapping("/{id}")  
public ResponseEntity detalhar(@PathVariable Long id){  
    var medico = repository.getReferenceById(id);  
    return ResponseEntity.ok(new DadosDetalhamentoMedico(medico));  
}
```



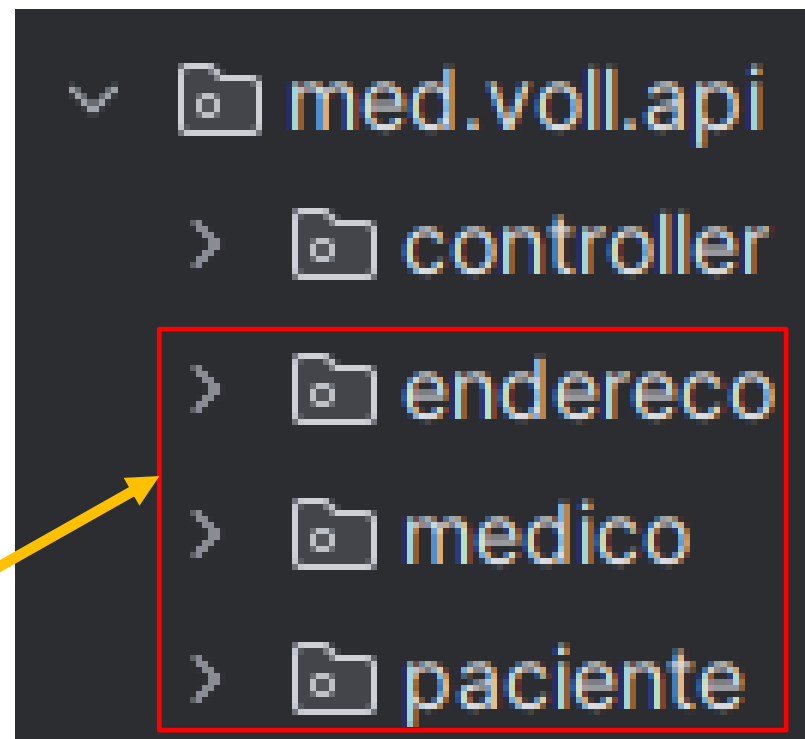
- Faremos a captura da exception, e devolveremos o código 404.

Spring

- Ao fazermos isso, estamos tratando o erro somente no método de detalhar da classe MedicoController.
- Porém, essa exceção pode acontecer em outros métodos e controllers.
- A ideia é criarmos uma classe e nela termos o método responsável por tratar esse erro em específico.
- Antes disso, faremos um ajuste. No pacote `med.voll.api`, **temos quatro sub - pacotes: controller, endereco, medico e paciente.**

Spring

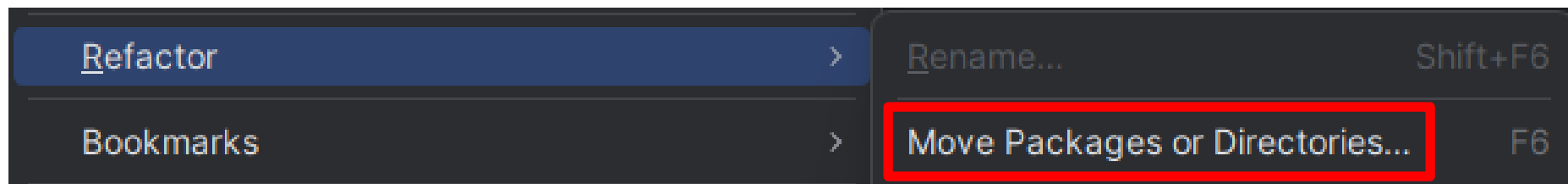
- Os sub-pacotes `endereco`, `medico` e `paciente` são referentes ao domínio da aplicação. Portanto, vamos isolar esses três pacotes em um único chamado domain.



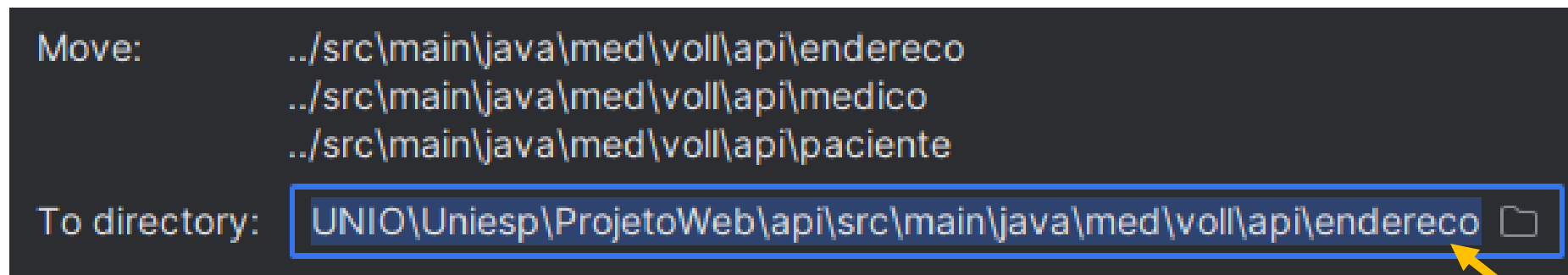
- Para isso, vamos selecionar os três pacotes segurando a tecla "`Ctrl`" e clicar com o botão direito do mouse.

Spring

- Nas caixas seguintes exibidas, escolhemos as opções "Refactor > Move packages or directories".



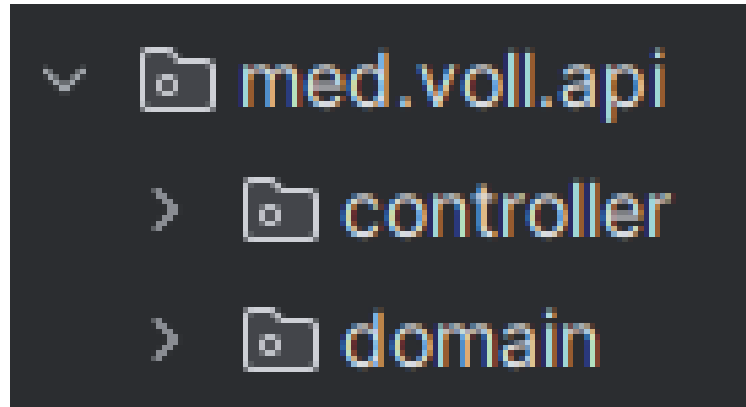
- Será exibido um pop-up, em que vamos escolher para onde desejamos mover esses três pacotes que selecionamos.



- Para mover, no final do caminho vamos alterar de "endereco" para "domain".

Spring

- Após isso, podemos clicar no botão "Refactor", no canto inferior direito. Assim, ficamos com as seguintes pastas do lado esquerdo do IntelliJ:



- Pronto! Agora a estrutura está mais organizada. No pacote principal temos os pacotes **controller** e o **domain**.
- Além desses dois pacotes, criaremos um terceiro chamado "infra", em que ficarão os códigos relacionados à infraestrutura.

Spring

- Para isso, clicamos com o **botão direito do mouse** na pasta `med.voll.api` e escolhemos a opção "**Package**".
- Na caixa exibida digitaremos o nome "**infra**", que ficará: `med.voll.api.infra`.
- Selecionando o pacote `infra`, usaremos o atalho "**Alt + Insert**" ou **botão direito do mouse** escolheremos a opção "**Java Class**".
- No pop-up exibido, digitaremos "**TratadorDeErros**", sendo a classe que vai fazer o isolamento do tratamento de erros.

```
package med.voll.api.infra;  
  
no usages  
  
public class TratadosDeErros {  
      
}
```

Spring

- Por enquanto, é uma classe em Java e não há nada em Spring. Isto é, o Spring não irá carregar essa classe automaticamente quando recarregarmos o projeto.
- Para ele carregar essa classe, é necessário termos à anotação `@RestControllerAdvice`.

```
@RestControllerAdvice  
public class TratadosDeErros {  
}
```

Spring

- Nesta classe, criaremos um método responsável por lidar com a exceção `EntityNotFoundException`.
- Para criar o método digitaremos `public return`, que chamaremos de `tratarErro404()` e, depois, abrimos e fechamos chaves ("{}").

```
public class TratadosDeErros {  
  
    public return tratarErro404(){  
        }  
  
}
```

Spring

- Uma linha anterior ao método, precisamos informar ao Spring para qual exceção esse método será chamado.
- No caso, usaremos a anotação `@ExceptionHandler`.

```
@ExceptionHandler  
public ResponseEntity tartarErro404()  
{  
}
```



Spring

- O Spring Boot emprega um mecanismo de manipulação de exceções centralizado chamado "ExceptionHandler".
- Esse mecanismo captura exceções lançadas por um controlador (controller) e trata-as de acordo com a lógica definida pelo desenvolvedor.

Spring



Spring

- Acrescentaremos um abre e fecha parênteses "(")" na anotação e dentro vamos especificar a classe exception:

```
@ExceptionHandler(EntityNotFoundException.class)  
public ResponseEntity tratarErro404()  
{
```

- Dessa forma, o Spring sabe que se em qualquer parte do controller do projeto for lançado uma exceção `EntityNotFoundException`, é para chamar o método `tratarErro404()`.
- E o que devolvermos, é o que será devolvido como resposta na requisição.

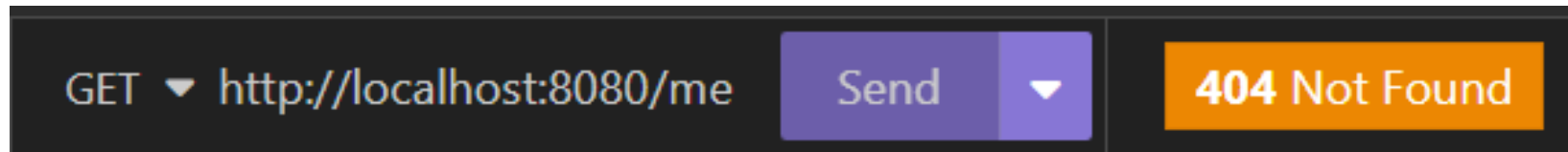
Spring

- No método `tratarErro404()`, colocaremos um `return ResponseEntity.notFound()`.
- No final, incluiremos um `.build()` para ele criar o objeto Response Entity.

```
@ExceptionHandler(EntityNotFoundException.class)
public ResponseEntity tratarErro404(){
    return ResponseEntity.notFound().build();
}
```

Spring

- Levante o servidor
- Faça o teste no Insomnia, passando o ID que não existe
- Note que o método retornou o código 404.



- Na próxima aula, faremos o tratamento do erro 400, usado quando disparamos uma requisição para cadastrar um médico ou paciente com erros na validação.

Spring

