Aula 7: Tratamento de Erros com Código 400

September 18, 2025

1 Introdução

Na Aula 6, personalizamos o tratamento de erros para retornar o código 404 (Not Found) em vez de 500 (Internal Server Error) ao buscar um médico com ID inexistente, usando a classe `TratadorDeErros`. Nesta aula, trataremos o erro 400 (Bad Request) para falhas de validação no cadastro de médicos, simplificando a resposta para incluir apenas os campos inválidos e suas mensagens de erro. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida, com testes no Insomnia.

2 Revisão do Erro 400 no Cadastro

O método de cadastro no `MedicoController` usa validações do Bean Validation:

```
@PostMapping
@Transactional
public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico dados,
    UriComponentsBuilder uriBuilder) {
    var medico = new Medico(dados);
    repository.save(medico);
    var uri =
        uriBuilder.path("/medicos/{id}").buildAndExpand(medico.getId()).toUri();
    return ResponseEntity.created(uri).body(new DadosDetalhamentoMedico(medico));
}
```

Testando no Insomnia com uma requisição `POST http://localhost:8080/medicos` e um JSON válido, obtemos o código 201 (Created):

```
{
    "nome": "Nome do Médico",
    "email": "medico@voll.med",
    "crm": "124580",
    "telefone": "61999998888",
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

Resposta:

```
{
    "id": 7,
    "nome": "Nome do Médico",
    "email": "medico@voll.med",
    "crm": "124580",
    "telefone": "61999998888",
```

```
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "numero": null,
        "complemento": null,
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

Porém, ao enviar um JSON com campos obrigatórios ausentes (ex.: sem `nome`, `email`, `crm` e `telefone`):

```
{
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

O Spring retorna 400 (Bad Request) com um JSON complexo, incluindo um array `errors` com informações detalhadas do Bean Validation, como `codes` e `message`. Queremos simplificar isso para retornar apenas o nome do campo e a mensagem de erro.

3 Personalizando o Tratamento do Erro 400

Na classe `TratadorDeErros`, adicionaremos um método para tratar a `MethodArgumentNotValidException`, que é lançada pelo Bean Validation quando há falhas de validação. Criaremos também um DTO interno para formatar a resposta.

O código atualizado é:

```
package med.voll.api.infra;

import jakarta.persistence.EntityNotFoundException;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class TratadorDeErros {

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

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity tratarErro400(MethodArgumentNotValidException ex) {
        var erros = ex.getFieldErrors();
        return
            ResponseEntity.badRequest().body(erros.stream().map(DadosErroValidacao::new).toList())
    }

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

```
        public DadosErroValidacao(FieldError erro) {
            this(erro.getField(), erro.getDefaultMessage());
        }
    }
}
```

3.1 Explicação do Código

- `@ExceptionHandler(MethodArgumentNotValidException.class)`: Associa o método `tratarErro400` à `MethodArgumentNotValidException`.
- `ex.getFieldErrors()`: Obtém a lista de erros de validação (`FieldError`) da exceção.
- `erros.stream().map(DadosErroValidacao::new).toList()`: Converte a lista de `FieldError` em uma lista de `DadosErroValidacao` usando Java Streams.
- `DadosErroValidacao`: DTO interno com os campos `campo` e `mensagem`, construído a partir de `FieldError.getField()` e `FieldError.getDefaultMessage()`.
- `ResponseEntity.badRequest().body(...)`: Retorna 400 (Bad Request) com a lista de erros no corpo da resposta.

4 Testando no Insomnia

Após salvar as alterações, o Spring DevTools reinicia o projeto. Testamos no Insomnia com a requisição `POST http://localhost:8080/medicos`:

4.1 Requisição com Dados Inválidos

- **Requisição**: JSON sem campos obrigatórios:

```
{
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

- **Resultado**: Código 400 (Bad Request) com o corpo:

```
[
    {
        "campo": "nome",
        "mensagem": "must not be blank"
    },
    {
        "campo": "email",
        "mensagem": "must not be blank"
    },
    {
        "campo": "telefone",
        "mensagem": "must not be blank"
    },
    {
        "campo": "crm",
        "mensagem": "must not be blank"
```

```
    }
]
```

4.2 Requisição com Erro Parcial

- **Requisição**: JSON sem o campo `crm`:

```
{
    "nome": "Nome do Médico",
    "email": "medico@voll.med",
    "telefone": "61999998888",
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

- **Resultado**: Código 400 (Bad Request) com o corpo:

```
[
    {
        "campo": "crm",
        "mensagem": "must not be blank"
    }
]
```

4.3 Requisição Válida

- **Requisição**: JSON completo:

```
{
    "nome": "Nome do Médico",
    "email": "medico@voll.med",
    "crm": "124580",
    "telefone": "61999998888",
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

- **Resultado**: Código 201 (Created) com os dados do médico.

5 Próximos Passos

Com o tratamento do erro 400 implementado, a API agora retorna respostas claras e seguras para falhas de validação. Na próxima aula, continuaremos o desenvolvimento do projeto, possivelmente explorando autenticação com Spring Security e tokens JWT, conforme os objetivos do curso. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

6 Dica do Professor

- **Aprofunde-se em validações**: Consulte a documentação do Bean Validation (`https://beanvalidation.org/2.0/`) para aprender sobre anotações personalizadas e mensagens de erro.
- **Comunidade no X**: Siga perfis como @SpringPro e @RestAPIExpert no X para dicas sobre validações e tratamento de erros em APIs REST.
- **Pratique**: Aplique o mesmo tratamento de erro 400 ao `PacienteController` para o endpoint de cadastro, testando no Insomnia com dados inválidos.