Aula 4: Implementando o Endpoint de Detalhamento de Médicos

September 18, 2025

1 Introdução

Na aula anterior, ajustamos o método de cadastro no `MedicoController` para retornar o código HTTP 201 (Created), incluindo o cabeçalho `Location` e os dados do médico criado no formato JSON. Nesta aula, implementaremos um novo endpoint para detalhar as informações de um médico específico, complementando o CRUD (Create, Read, Update, Delete) com a funcionalidade de leitura detalhada. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida, e os testes serão realizados no Insomnia.

2 Revisão do Método de Cadastro

Na Aula 3, ajustamos o método de cadastro para retornar o código HTTP 201, com o cabeçalho `Location` e os dados do médico criado. O código final foi:

```
@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));
}
```

Testamos no Insomnia, e a requisição `POST http://localhost:8080/medicos` retornou o código 201 com o corpo da resposta e o cabeçalho `Location: http://localhost:8080/medicos/6`. Porém, ao tentar acessar `http://localhost:8080/medicos/6`, recebemos o código 404 (Not Found), pois o endpoint de detalhamento ainda não existe. Vamos criá-lo agora.

3 Criando o Endpoint de Detalhamento

Para permitir o detalhamento de um médico específico, criaremos um novo método no `MedicoController` que responde a requisições `GET` na rota `/medicos/id`. Este endpoint retornará o código HTTP 200 (OK) com os dados completos do médico, usando o DTO `DadosDetalhamentoMedico`.

O código do novo método é:

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

3.1 Explicação do Código

- `@GetMapping("/id")`: Define a rota `/medicos/id` (ex.: `/medicos/6`) para requisições GET.
- `@PathVariable Long id`: Captura o ID da URL como parâmetro.
- `repository.getReferenceById(id)`: Busca o médico no banco de dados pelo ID.
- `ResponseEntity.ok(new DadosDetalhamentoMedico(medico))`: Retorna o código 200 com os dados do médico no formato do DTO `DadosDetalhamentoMedico`.

4 Reutilizando o DTO DadosDetalhamentoMedico

O DTO `DadosDetalhamentoMedico`, criado na Aula 2, é reutilizado aqui para garantir que as informações do médico sejam retornadas de forma consistente, sem expor a entidade JPA diretamente. O código do DTO é:

```
package med.voll.api.medico;

import med.voll.api.endereco.Endereco;

public record DadosDetalhamentoMedico(Long id, String nome, String email, String
    crm, String telefone, Especialidade especialidade, Endereco endereco) {
    public DadosDetalhamentoMedico(Medico medico) {
        this(medico.getId(), medico.getNome(), medico.getEmail(), medico.getCrm(),
            medico.getTelefone(), medico.getEspecialidade(), medico.getEndereco());
    }
}
```

5 Testando no Insomnia

Após salvar as alterações no `MedicoController`, o Spring DevTools atualiza o projeto automaticamente. No Insomnia, testamos o novo endpoint e revisamos os existentes:

5.1 Requisição de Detalhamento

- **Requisição**: `GET http://localhost:8080/medicos/6`
- **Resultado**: Código 200 (OK) com os dados do médico:

```
{
    "id": 6,
    "nome": "Nome do Médico",
    "email": "medico@voll.med",
    "crm": "233444",
    "telefone": "61999998888",
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "numero": null,
        "complemento": null,
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

5.2 Revisão das Outras Requisições

- **Exclusão** (`DELETE http://localhost:8080/medicos/2`): Retorna 204 (No Content).
- **Atualização** (`PUT http://localhost:8080/medicos`):

```
{
    "id": 1,
    "telefone": "2111112222"
}
```

  Retorna 200 (OK) com os dados atualizados do médico.

- **Listagem** (`GET http://localhost:8080/medicos`): Retorna 200 (OK) com os dados paginados.
- **Cadastro** (`POST http://localhost:8080/medicos`):

```
{
    "nome": "Nome do Médico",
    "email": "medico@voll.med",
    "crm": "233444",
    "telefone": "61999998888",
    "especialidade": "ORTOPEDIA",
    "endereco": {
        "logradouro": "rua 1",
        "bairro": "bairro",
        "cep": "12345678",
        "complemento": null,
        "cidade": "Brasil",
        "uf": "DF"
    }
}
```

  Retorna 201 (Created) com o cabeçalho `Location: http://localhost:8080/medicos/6`.

6 Próximos Passos

Com o endpoint de detalhamento implementado, completamos as funcionalidades principais da API. Na próxima aula, abordaremos o tratamento de erros para personalizar as respostas em cenários de falha, como quando um médico não é encontrado. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

7 Dica do Professor

- **Aprofunde-se em endpoints REST**: Consulte a documentação do Spring Boot sobre controladores REST (`https://spring.io/guides/gs/rest-service/`) para entender como estruturar endpoints como o de detalhamento.
- **Comunidade no X**: Siga perfis como @RestAPIPro e @SpringDev no X para dicas sobre design de APIs e melhores práticas com Spring Boot.
- **Pratique**: Adicione validações ao endpoint de detalhamento para tratar casos em que o ID do médico não existe, retornando o código 404 (Not Found).