Aula 4: Implementando o Endpoint de Detalhamento de Médicos

September 18, 2025

1 Introdução

Nas aulas anteriores, padronizamos os retornos do `MedicoController` usando `ResponseEntity`, implementando códigos HTTP apropriados: 201 (Created) para cadastro, 200 (OK) para listagem e atualização, e 204 (No Content) para exclusão. Nesta aula, implementaremos o endpoint de detalhamento de médicos, completando as funcionalidades do CRUD com a leitura detalhada de um médico específico. As alterações serão feitas no `Visual Studio Code` ou sua IDE preferida, com testes no Insomnia.

2 Revisão do Método de Cadastro

Na Aula 3, ajustamos o método de cadastro para retornar o código HTTP 201 (Created), incluindo o cabeçalho `Location` (ex.: `http://localhost:8080/medicos/6`) e os dados do médico criado. O código 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));
}
```

Ao testar no Insomnia, a requisição `POST http://localhost:8080/medicos` retornou 201 com o cabeçalho `Location`. Porém, ao tentar acessar `http://localhost:8080/medicos/6` com uma requisição `GET`, recebemos o código 405 (Method Not Allowed), pois a rota `/medicos/id` está mapeada apenas para `DELETE`. Vamos criar o endpoint `GET` para detalhamento.

3 Criando o Endpoint de Detalhamento

Criaremos um método no `MedicoController` para responder a requisições `GET` na rota `/medicos/id`, retornando o código 200 (OK) com os dados do médico no DTO `DadosDetalhamentoMedico`. O código 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")`: Mapeia a rota `/medicos/id` (ex.: `/medicos/6`) para requisições `GET`.
- `@PathVariable Long id`: Captura o ID da URL.
- `repository.getReferenceById(id)`: Busca o médico no banco de dados pelo ID.
- `ResponseEntity.ok(new DadosDetalhamentoMedico(medico))`: Retorna 200 (OK) com os dados do médico no formato do DTO.

O método reutiliza o DTO `DadosDetalhamentoMedico`, já criado para os métodos de cadastro e atualização:

```
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());
    }
}
```

4 Testando no Insomnia

Após salvar as alterações, o Spring DevTools atualiza o projeto automaticamente. No Insomnia, criamos uma nova requisição `GET`:

- **Requisição**: `GET http://localhost:8080/medicos/6`
- Acesse o menu clicando no botão "+" e selecione "HTTP Request".
- Renomeie a requisição para "Detalhar Médico".
- Insira a URL `http://localhost:8080/medicos/6` e clique em "Send".

4.1 Resultado da Requisição

- **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"
    }
}
```

Testando com `GET http://localhost:8080/medicos/1`, obtemos o mesmo código 200 com os dados do médico correspondente ao ID 1. Se testarmos um ID inválido (ex.: 999), o método `getReferenceById` lançará uma exceção, resultando em erro 500 (Internal Server Error). Este problema será resolvido na próxima aula com tratamento de erros.

5 Aplicando a Lógica ao Controlador de Pacientes

A mesma lógica do endpoint de detalhamento pode ser aplicada ao `PacienteController`. Como atividade prática, crie um método semelhante ao de detalhamento de médicos, usando `ResponseEntity` e um DTO `DadosDetalhamentoPaciente` (a ser criado com base em `DadosDetalhamentoMedico`). O endpoint deve responder a `GET /pacientes/id` e retornar 200 (OK) com os dados do paciente.

6 Próximos Passos

Com o endpoint de detalhamento implementado, completamos as funcionalidades principais da API. Na próxima aula, abordaremos o tratamento de erros, como retornar 404 (Not Found) para IDs inválidos, conforme os objetivos do curso. Continue praticando no `Visual Studio Code` ou sua IDE preferida!

7 Dica do Professor

- **Aprofunde-se em design de APIs**: Consulte a documentação do Spring Boot sobre controladores REST (`https://spring.io/guides/gs/rest-service/`) para entender como estruturar endpoints eficientes.
- **Comunidade no X**: Siga perfis como @RestAPIPro e @SpringDev no X para dicas sobre design de APIs e melhores práticas com Spring Boot.
- **Pratique**: Implemente o endpoint de detalhamento no `PacienteController` e teste no Insomnia, verificando os códigos HTTP retornados.