

# Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 12 – DEVOLVENDO O CÓDIGO HTTP 201

## Devolvendo o Código 201



# Spring

- Começamos a realizar as alterações no controller, nos métodos de excluir, atualizar e listar, para padronizar os retornos para responder ResponseEntity. No entanto, faltou o método cadastrar.
- O método de cadastrar possui um detalhe a mais devido ao protocolo HTTP, e faremos mais uma requisição para detalhar o médico.
- Conforme o protocolo HTTP, ao cadastrarmos uma informação ou recurso em uma API, o código HTTP que deve ser devolvido, neste cenário, é o código 201 chamado created.
- Esse código significa que um registro foi criado na API.

# Spring

- **Código 201:** devolve no **corpo** da resposta os dados do novo recurso/registro criado
- E tem que devolver **um cabeçalho** do protocolo HTTP (**Location**).
- Esse cabeçalho mostra o endereço para que o front-end, ou aplicativo mobile consiga acessar o recurso cadastrado.
- Logo, no cadastro não devolvemos apenas o **código 200 OK** e nem apenas o **201**.
- Precisa ser o código 201, com os **dados no formato JSON** e **um cabeçalho**, na **resposta**.
- Para fazer isso, **usaremos alguns recursos do Spring Boot**.

# Spring

```
@PostMapping
@Transactional
public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico
dados) {
    repository.save(new Medico(dados));
}
```

- No método de cadastrar do arquivo **MedicoController**, note que ele já aponta um erro de compilação, já que alteramos o retorno de **void** para **ResponseEntity**.

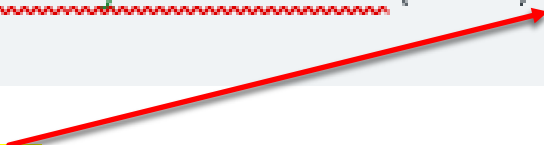
# Spring

- Após chamarmos o `repository.save`, como fazemos para `acionar o código 201`?
- Incluiremos `return ResponseEntity` com o método `.created( )`, que passaremos como parâmetro a `uri`: `return ResponseEntity.created(uri)`.
- Essa `uri` representa o endereço, e o Spring cria o cabeçalho `location` de forma automática conforme a `uri` que passamos como parâmetro.
- Ainda vamos criar `essa URI`.

# Spring

```
@PostMapping
@Transactional
public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico
dados) {
    repository.save(new Medico(dados));

    return ResponseEntity.created(uri)
}
```



- Na sequência colocamos `.body( )`, para incluir as informações que queremos devolver no corpo da resposta, como parâmetro colocamos `dto`. Assim, ele cria o `objeto ResponseEntity`.
- Temos o erro de compilação em `uri` e `dto`, isso porque essas variáveis não existem neste método cadastrar.

# Spring

```
@PostMapping
@Transactional
public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico
dados) {
    repository.save(new Medico(dados));

    return ResponseEntity.created(uri).body(dto);
}
```

- Temos o erro de compilação em **uri** e **dto**, isso porque essas variáveis não existem neste método cadastrar.



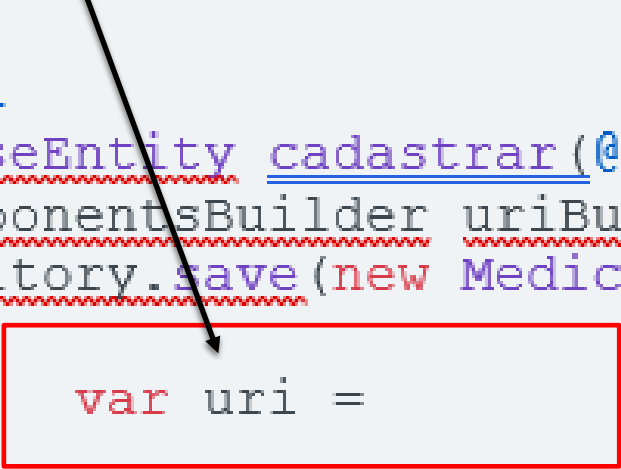
# Spring

- Vamos **criar um objeto uri**, na **linha de cima do return**, vamos criar a variável, **var uri =**.

```
@PostMapping
@Transactional
public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico
dados, UriComponentsBuilder uriBuilder) {
    repository.save(new Medico(dados));

    var uri =

    return ResponseEntity.created(uri).body(dto);
}
```





- URI significa "Uniform Resource Identifier" (Identificador Uniforme de Recurso) e é uma sequência de caracteres que **identifica um recurso na web**.
- É uma forma de endereço que permite localizar e acessar recursos, como **páginas da web, imagens, arquivos**, etc.
- Por exemplo, a URI de uma página da web pode ser algo como "https://www.exemplo.com/pagina".

# Spring

- Já a URL, que significa "Uniform Resource Locator" (Localizador Uniforme de Recurso), é um tipo específico de URI que inclui o protocolo de comunicação utilizado, como HTTP ou HTTPS, além do domínio, caminho e outros componentes.
- É basicamente o endereço completo de um recurso na web. Por exemplo, a URL completa de uma página da web pode ser "https://www.exemplo.com/pagina".

# Spring

---



# Spring

- A **URI** deve ser o endereço da **API**, no caso é o `http://localhost:8080/medicos/id`, sendo o **ID do médico** que acabou de ser criado no banco de dados.
- Lembrando que **está rodando local** e **ainda faremos o deploy** para **rodar no servidor**. Logo, não será mais `http://localhost:8080`, **será alterado**.
- Para não precisarmos ter que dar muita atenção para esse ponto no controller, o **Spring possui uma classe** que **encapsula o endereço da API**.
- Essa classe realiza a construção da URI de forma automática.

# Spring

- Para usarmos essa classe, incluiremos mais um parâmetro no método cadastrar.
- Atualmente, estamos recebendo o DadosCadastroMedico.
- Colocaremos uma vírgula (",") e, na sequência, um objeto do tipo UriComponentsBuilder, sendo a classe que criar/gera a URI.

```
public ResponseEntity cadastrar(@RequestBody @Valid DadosCadastroMedico  
dados, UriComponentsBuilder uriBuilder)
```

# Spring

- Voltando para a variável `uri`, após o sinal de igual podemos colocar `uriBuilder.`, e chamaremos o método `path` para passarmos o complemento da URL: `var uri = uriBuilder.path( )`.
- Isso porque ele cria somente a `URI localhost:8080`, e precisamos incluir o complemento `/medicos/id`.
- Portanto, no parênteses do método `path` vamos passar `"/medicos/{id}"`.
- O `id` entre chaves é um parâmetro dinâmico.

```
var uri = uriBuilder.path("/medicos/{id}")
```



# Spring

```
var uri = uriBuilder.path("/medicos/{id}")
```

- Esse ID, é o ID do médico que foi criado no banco de dados..
- Para isso, digitamos `.buildAndExpand( )`.
- Nele, precisamos passar, como parâmetro, o ID do médico criado no banco de dados.
- Esse ID está no `repository.save` que chamamos para salvar no banco de dados.

```
repository.save(new Medico(dados));
```

# Spring

- Contudo, passamos como parâmetro o new Medico para o método save. Vamos precisar desse médico na linha seguinte, por isso, criaremos uma variável para o médico: `var medico =`.

- Faremos dessa forma.

- Antes

```
repository.save(new Medico(dados));
```

- depois

```
var medico = new Medico(dados)  
repository.save();
```

# Spring

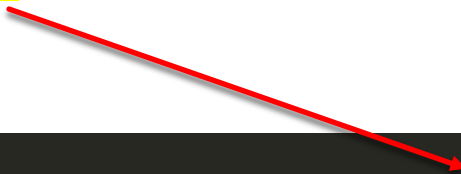
- No `repository.save()` passamos o `medico` como parâmetro, e no `buildAndExpand()` o `medico.getId()`.
- O ID será gerado pelo banco de dados na linha anterior de forma automática.
- Logo após o build and expand, colocamos `.toUri()` para criar o objeto URI.

```
var uri = uriBuilder.path("/medicos/{id}").  
    buildAndExpand(medico.getId()).toUri();
```

# Spring

- Criamos o **objeto URI**.
- Agora, **no return**, perceba que o **parâmetro dto** está na cor vermelha, significa que precisamos criá-lo.

```
return ResponseEntity.created(uri).body(dto);
```



- Usaremos o mesmo dto que utilizamos no método de atualizar, **o new DadosDetalhamentoMedico(medico)**.
- Copiaremos esse método e colaremos no parâmetro do método .body()..

```
return ResponseEntity.created(uri).  
    body(new DadosDetalhamentoMedico(medico));
```

# Spring

- O método de cadastrar possui diversos detalhes, porque precisamos devolver o código 201, o cabeçalho location com a URI e no corpo da resposta é necessário ter uma representação do recurso recém criado.
- Agora, vamos testar o cadastro, listagem, atualização e exclusão!
- No Insomnia faça um requisição:
- de excluir médico, note que retornou o código 204 No Content, sem corpo de retorno, deu certo!!!!
- De alteração, no que retornou o código 200 OK, com JSON no corpo de resposta/retorno, deu certo!!!!

# Spring

- No Insomnia faça um requisição:
- de Listagem médico, esse método não mudou, continua devolvendo o código **200 Ok**, com JSON no corpo de resposta/retorno, deu certo!!!!
- De alteração, no que retornou o código **200 OK**, com corpo de retorno, deu certo!!!! O **ResponseEntity** não alterou em nada, isso já era esperado.
- Para finalizar vamos realizar um cadastro, para isso basta alterar o nome, email e crm. Antes ele retornava o código **200 OK**, sem corpo de retorno!!!
- Agora retorna o código 201 Created, com JSON no corpo de resposta/retorno, deu certo!!!!

# Spring

---

- Por enquanto, temos somente o CRUD (Create, Read, Update e Delete), mas em uma API geralmente temos uma quinta funcionalidade para detalhar as informações de um médico.
- Na próxima aula, vamos aprender a criar essa funcionalidade.

# Spring

- Do lado direito da aba "Preview", temos a aba "Header", sendo os cabeçalhos que foram devolvidos. Ao clicarmos na aba "Header", temos:

NAME	VALUE
Location	http://localhost:8080/medicos/30
Content-Type	application/json
Transfer-Encoding	chunked
Date	Mon, 13 May 2024 04:00:51 GMT

- Se tentarmos entrar no endereço `/medicos/30`, será devolvido o código 404. Isso porque **não configuramos uma requisição para detalhar os dados de um médico.**



# Spring

