

# Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 07 – REQUISIÇÕES GET....

**Spring**

## **Listagem Médicos**



## Listagem Médicos

- O sistema deve possuir uma funcionalidade de listagem de médicos, na qual as seguintes informações, de cada médico cadastrado, deverão ser exibidas:
  - Nome
  - E-mail
  - CRM
  - Especialidade
- A listagem deve ser **ordenada** pelo **nome do médico**, de maneira **crescente**, bem como ser **paginada**, trazendo **10 registros** por página

## Listagem Médicos

- Vamos acessar "src > main > java > med.voll.api > controller > MedicoController".
- Até então, havíamos implementado apenas o método `cadastrar`. Agora criaremos o **método public**, responsável pela listagem.
- O retorno dele será `List<Medico> listar ()`. Acima do método, vamos adicionar a anotação `@GetMapping`, para informar o verbo do **protocolo HTTP**.

## Listagem Médicos

- Como precisamos acessar o banco de dados, passaremos a classe `repository.findAll()` como retorno:

```
@GetMapping  
public List<Medico> listar() {  
    return repository.findAll();  
}
```

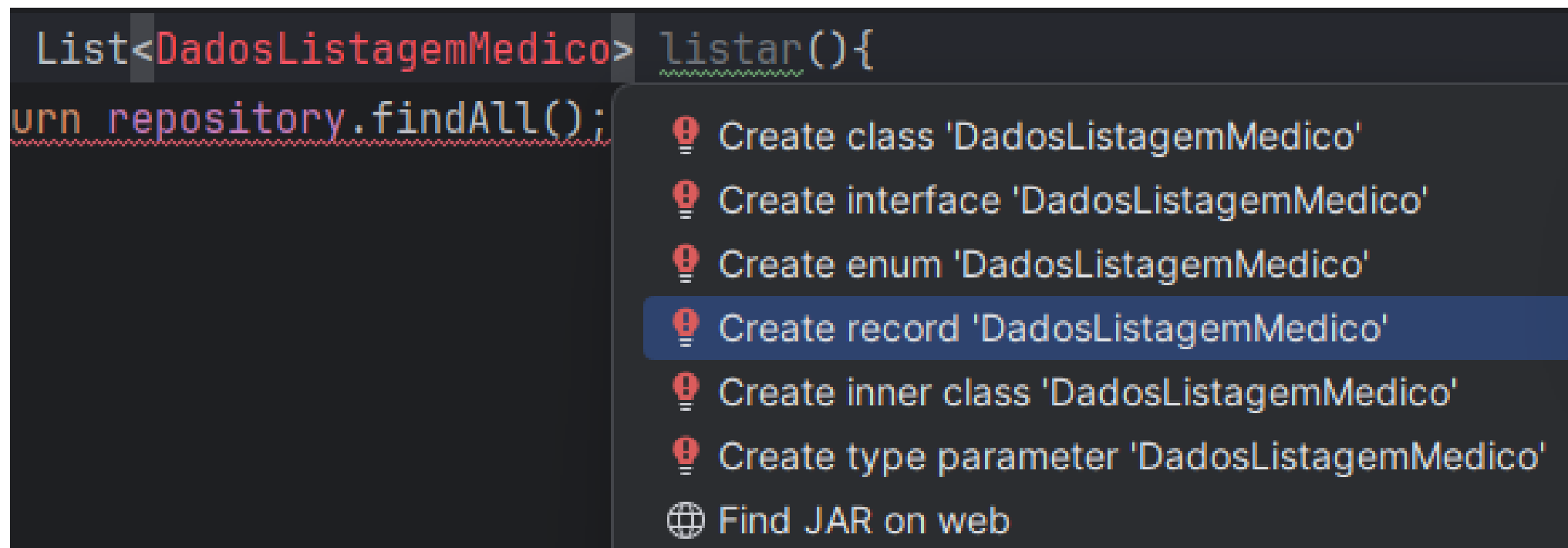
## Listagem Médicos

- Porém, não podemos devolver uma lista de Medico, porque não queremos devolver todos os atributos dela, apenas nome, email, CRM e especialidade.
- Por isso, criaremos um DTO que devolverá dados da API. Substituiremos Medico por DadosListagemMedico.

```
@GetMapping
public List<DadosListagemMedico> listar(){
    return repository.findAll();
}
```

## Listagem Médicos

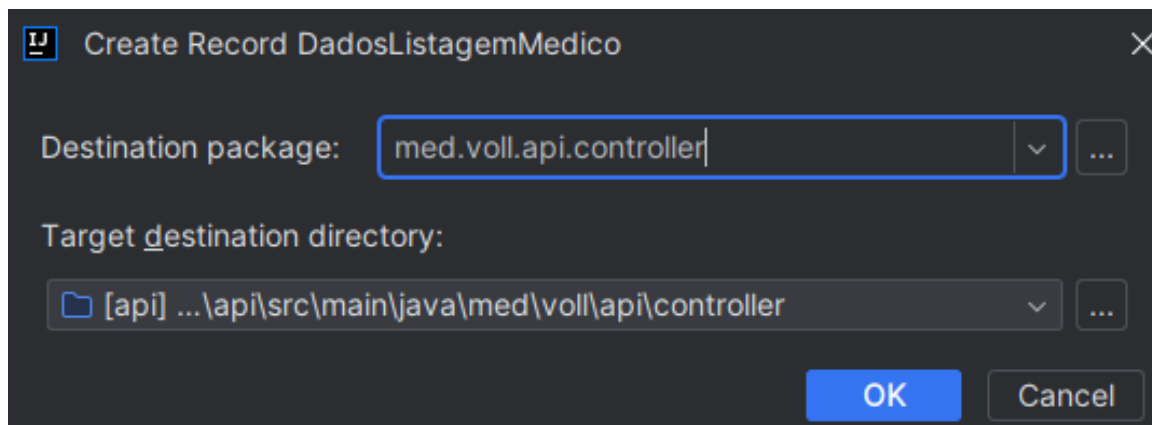
- Vamos usar o atalho "**Alt + Enter**" e selecionar a opção "**Create record 'DadosListagemMedico'**".



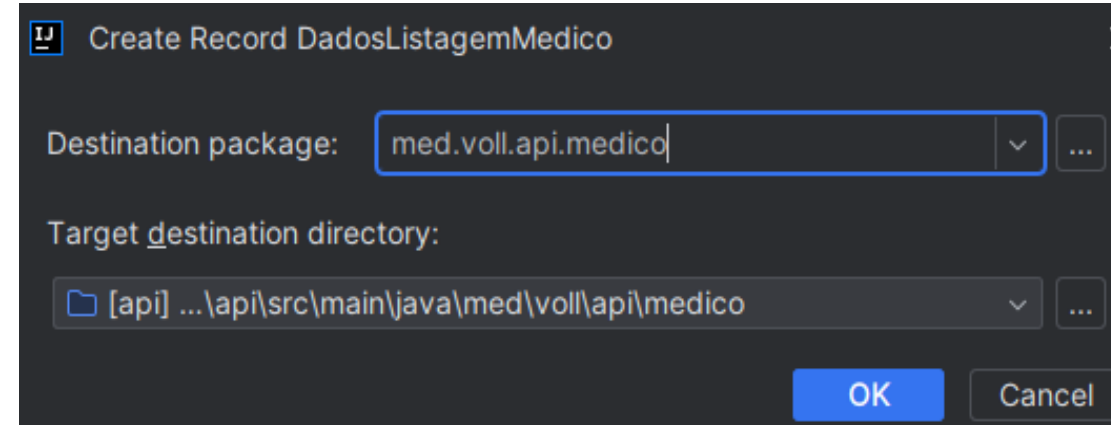
# Spring

## Listagem Médicos

- Em "Destination package", informaremos "med.voll.api.medico" e clicaremos em "OK".



Mude para.



- Agora precisamos informar as propriedades que serão trabalhadas pelo DTO, que serão **String nome**, **String email**, **String crm** e **Especialidade especialidade**:



# Spring

```
public record DadosListagemMedico(String nome,  
                                     no usages  
                                     String email,  
                                     no usages  
                                     String crm,  
                                     no usages  
                                     Especialidade especialidade) {  
}
```

- Esses são os campos que pretendemos devolver para o Front-end, app mobile, para o cliente da nossa api, Insomnia.
- Com o **DTO é possível** realizar essa **filtragem**, temos esse **controle** de não devolver todos os atributos da **entidade JPA**.

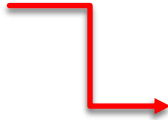
## Listagem Médicos

- Devolvemos apenas as informações necessárias, com o DTO ganhamos com a flexibilidade.
- **Retorne** para **MedicoController**.

# Spring

## Listagem Médicos

- O porquê desse erro??



```
@GetMapping
public List<DadosListagemMedico> listar(){
    return repository.findAll();
}
```

- Resposta:

```
@GetMapping
public List<DadosListagemMedico> listar(){
    return repository.findAll();
}
```

Required type: List <DadosListagemMedico>

Provided: List <Medico>

# Spring

- Temos que converter de **Medico** para **DadosListagemMedico**.
- **Antes da conversão**, vamos criar um método **DadosListagemMedico**, dentro do **record DadosListagemMedico**.
- Depois faremos a devida conversão!!!!

# Spring

```
1 usage
public record DadosListagemMedico(String nome,
                                   no usages
                                   String email,
                                   no usages
                                   String crm,
                                   no usages
                                   Especialidade especialidade) {

    no usages
    public DadosListagemMedico(Medico medico) {
        this(medico.getNome(), medico.getEmail(), medico.getCrm(), medico.getEspecialidade());
    }
}
```

## Listagem Médicos

- Vamos converter de Medico para DadosListagemMedico

```
@GetMapping
public List<DadosListagemMedico> listar(){
    return repository.findAll().stream().map(DadosListagemMedico::new).toList();
}
```

- Dessa forma conseguimos converte uma Lista de Médicos para uma Lista de DadosListagemMedico que é o nosso DTO.
- Agora esta pronto nossa listagem!!! O teste será realizado no Insomnia.

## Listagem Médicos

- Para esse método `listar()` não estamos usando o `@Transactional` , pois não precisa ter uma transação ativa porque ele é apenas de leitura!!

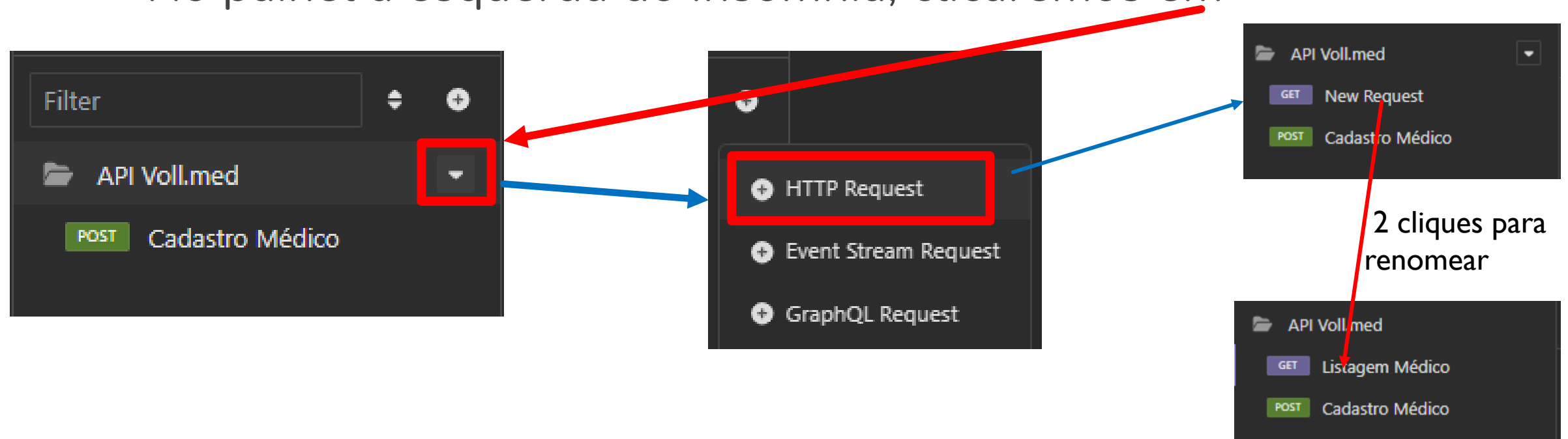
## Testando a Listagem





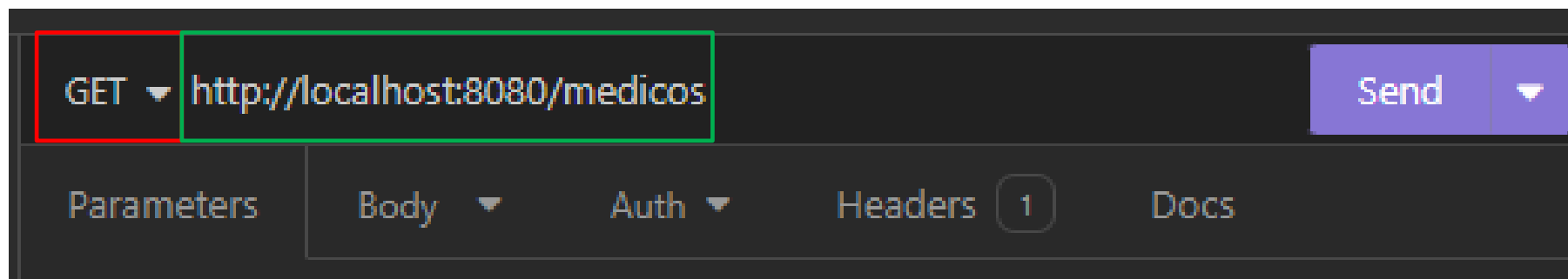
# Spring

- Vamos acessar o Insomnia para testar nossa API.
- Como **não queremos** mais **testar o cadastro de médicos**, precisaremos **criar uma nova requisição**.
- No painel à esquerda do Insomnia, clicaremos em



# Spring

- O verbo será o padrão, "GET".




- Em seguida, digitaremos a URL da requisição para o mesmo endereço do cadastro: "<http://localhost:8080/medicos>".

Obs: Não haverá conflito porque os verbos das duas requisições da mesma URL são diferentes.

# Spring

- Retornando para o IntelliJ lá no controller.
- Em cima da classe controller temos:



```
@RequestMapping("medicos")
public class MedicoController {
    // ...
}
```

- Estamos informando para o Spring, que as requisições para a URL médicos, quem irá responder será o “controller MedicoController”

# Spring

- Repare que no **controller**, temos dois métodos

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

```
public List<DadosListagemMedico> listar() {  
    return repository.findAll().stream().map(Da  
}
```

- Como o **Spring** irá saber **qual método usar**??? Quando chegar uma requisição para o **endereço médicos**???

# Spring

- Simples!!! Olhando o verbo da requisição HTTP!!!!
- Se for POST, ele irá chamar o método cadastrar, devido a anotação

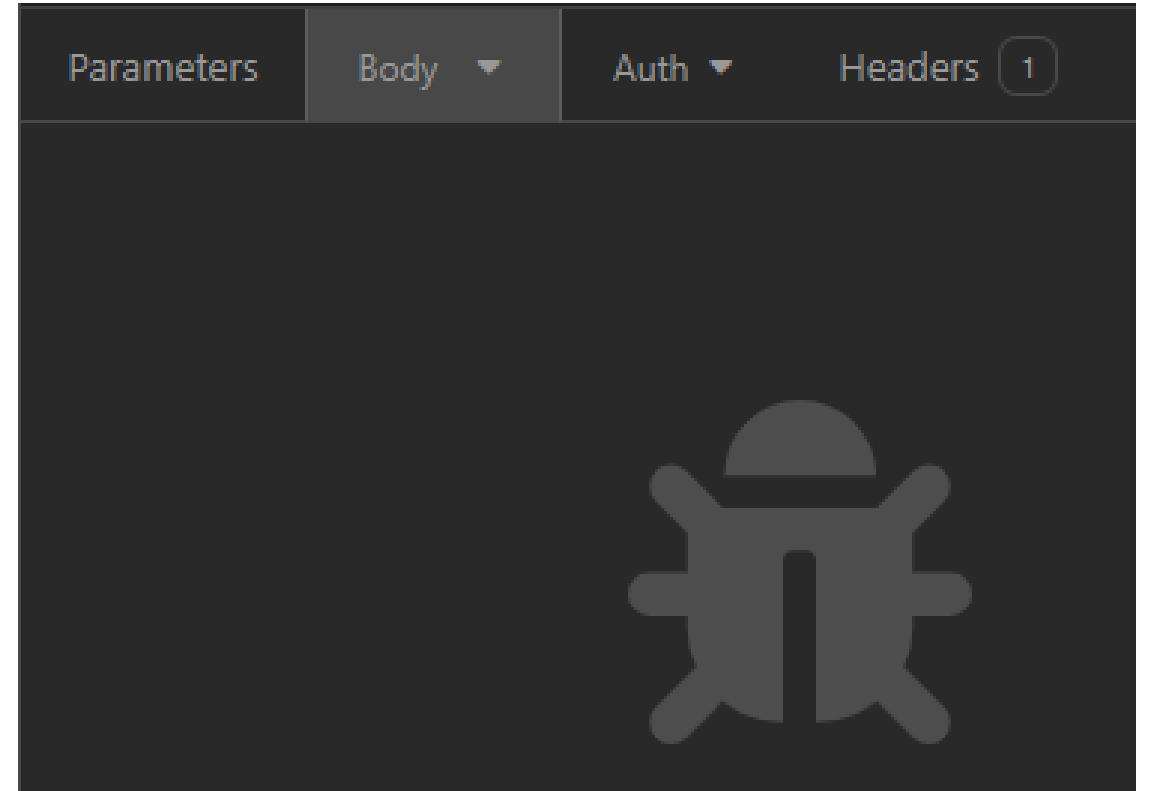
```
@PostMapping
@Transactional
public void cadastrar(@RequestBody @Valid
    repository.save(new Medico(dados));
```

- Se for GET, ele irá chamar o método listar, devido a anotação

```
@GetMapping
public List<DadosListagemMedico> listar(){
    return repository.findAll().stream().map
}
```

# Spring

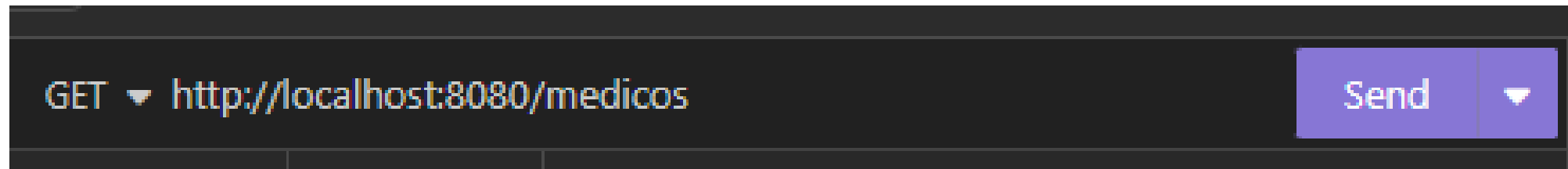
- Retorne ao Insomnia.
- O body



- O body da requisição irá vazio, porque não estamos cadastrando informações.

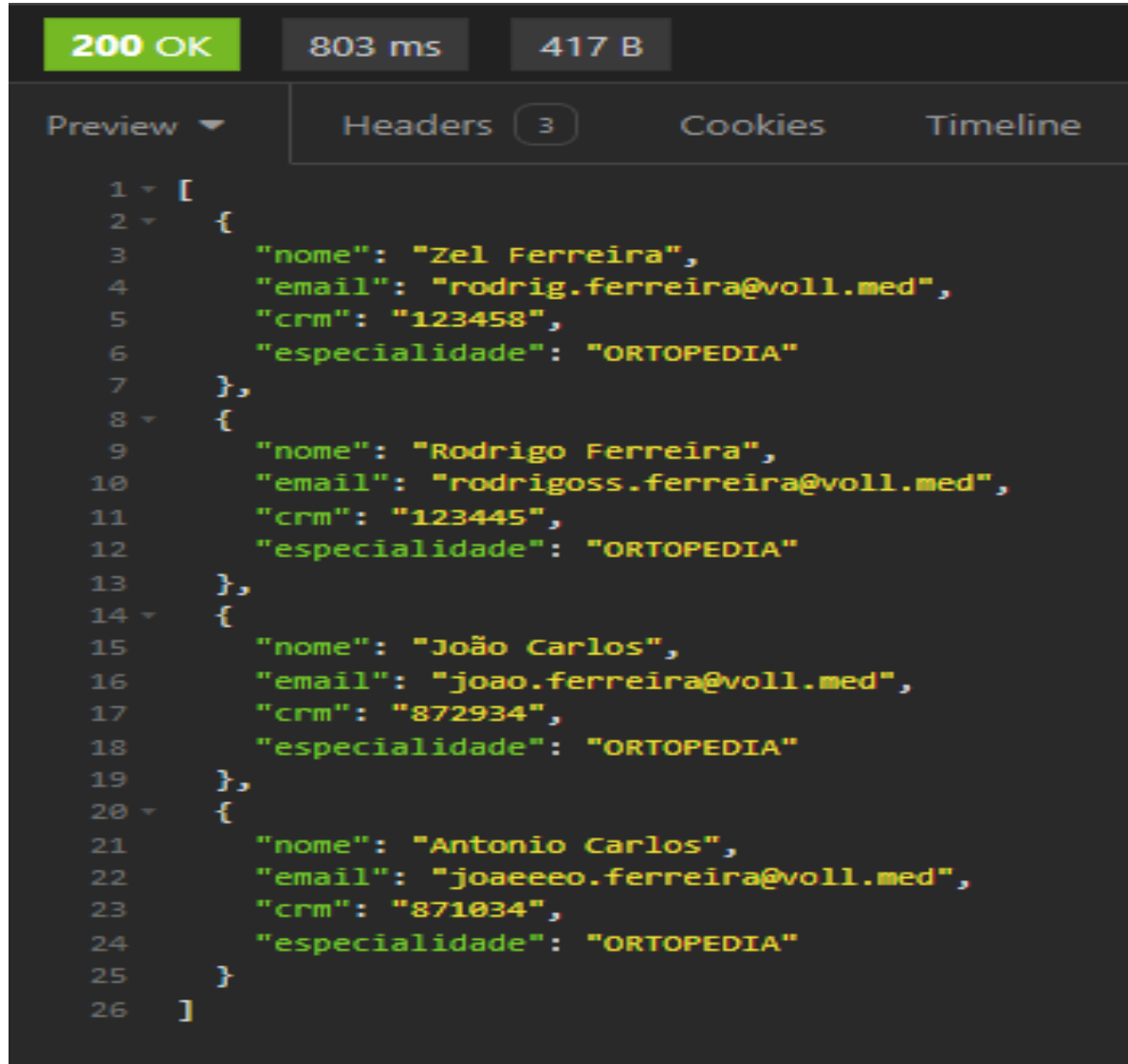
# Spring

- Clicando no botão "Send", vamos disparar a requisição.



# Spring

- No painel, receberemos os dados devolvidos pela API.



The screenshot displays a web browser's developer console with the 'Preview' tab selected. At the top, the status bar indicates a '200 OK' response, a duration of '803 ms', and a size of '417 B'. Below this, the 'Headers' tab is active, showing '3' headers. The main area displays a JSON array of four doctor records, each with fields for 'nome', 'email', 'crm', and 'especialidade'. The records are for Zel Ferreira, Rodrigo Ferreira, João Carlos, and Antonio Carlos, all specializing in 'ORTOPEDIA'.

```
1 [
2   {
3     "nome": "Zel Ferreira",
4     "email": "rodrig.ferreira@voll.med",
5     "crm": "123458",
6     "especialidade": "ORTOPEDIA"
7   },
8   {
9     "nome": "Rodrigo Ferreira",
10    "email": "rodrigoss.ferreira@voll.med",
11    "crm": "123445",
12    "especialidade": "ORTOPEDIA"
13  },
14  {
15    "nome": "João Carlos",
16    "email": "joao.ferreira@voll.med",
17    "crm": "872934",
18    "especialidade": "ORTOPEDIA"
19  },
20  {
21    "nome": "Antonio Carlos",
22    "email": "joaeeee.ferreira@voll.med",
23    "crm": "871034",
24    "especialidade": "ORTOPEDIA"
25  }
26 ]
```



# Spring

---

- O Spring Boot assumiu automaticamente que queremos converter a lista para um JSON.
- Isso faz com que eles no devolva um arquivo desse tipo, contendo um array listando todos os médicos cadastrados no banco de dados.

# Spring

