

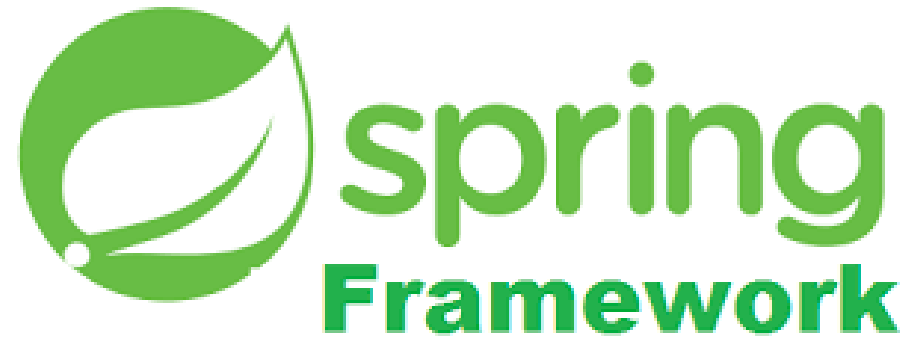
# Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 02 – REQUISIÇÕES POST

## Enviando Datos Para API



- Com a aplicação criada, vamos desenvolver as funcionalidades do projeto.
  - Quais as funcionalidades do projeto?
  - O que vamos precisar implementar?
  - Quais as validações e regras de negócio?

## Funcionalidade Cadastro de Médicos



- O sistema deve possuir uma funcionalidade de cadastro de médicos, na qual as seguintes informações deverão ser preenchidas:

Nome

E-mail

Telefone

CRM

## Funcionalidade Cadastro de Médicos



Ainda mais:

- *Especialidade* (Ortopedia, Cardiologia, Ginecologia ou Dermatologia)
- *Endereço completo* (logradouro, *número*, *complemento*, bairro, cidade, UF e CEP)
- *Todas as informações são de preenchimento \*\*obrigatório\*\*, exceto o número e o complemento do endereço.*

## Funcionalidade Cadastro de Médicos



- Temos os campos obrigatórios e a regra de negócio que devem ser aplicados no projeto.
- Para facilitar o entendimento, dado que vamos desenvolver somente **API Back-end** (não teremos interface gráfica)
- Agora, vamos **focar na funcionalidade de cadastrar médicos**.



Novo perfil

Médicos



Pacientes

Consultas

## Profissional

Nome completo

Especialidade



CRM

## Contatos

E-mail

Telefone ou celular

## Endereço profissional

Logradouro

Número

Complemento

Cidade

UF



CEP

Concluir cadastro

Cancelar

## Funcionalidade Cadastro de Médicos



- Ao clicarmos no botão "Concluir cadastro", o aplicativo irá enviar uma requisição para a nossa API. Nela, receberemos, validaremos e salvaremos essas informações em um banco de dados
- Com esse layout, conseguimos visualizar melhor o funcionamento, já que não implementaremos a parte das telas. Com as funcionalidades descritas e os layouts nos auxiliando, vamos iniciar a implementação pela parte de disparar as requisições.



## Funcionalidade Cadastro de Médicos



- Como não temos um aplicativo mobile e nem aplicação Front-end, como testaremos a API? Como vamos enviar as requisições?
- Usaremos uma ferramenta de testes de API, as duas mais usadas são:

*Postman*

*Insomnia*



*utilizaremos a ferramenta Insomnia*

<https://insomnia.rest/download>

# Spring

Insomnia

Application File Edit View Window Tools Help



## Welcome to Insomnia

Discover Local, Cloud or Git storage for your projects.



Continue with Google



Continue with GitHub



Continue with Email



Continue with SSO

By signing up or using Insomnia, you agree to the [terms of service](#) and [privacy policy](#).



Use the local Scratch Pad



Export data and more

Ativar o Windows

Acesse Configurações para ativar o Windows.

# Spring

Insomnia

Application File Edit View Window Tools Help

Search..

Ctrl + P

Star

32,499

+

< | Scratch Pad ▾

Base Environment ▾

Add Cookies

Add Certificates

Filter

+

New Request

Ctrl + N

Switch Requests

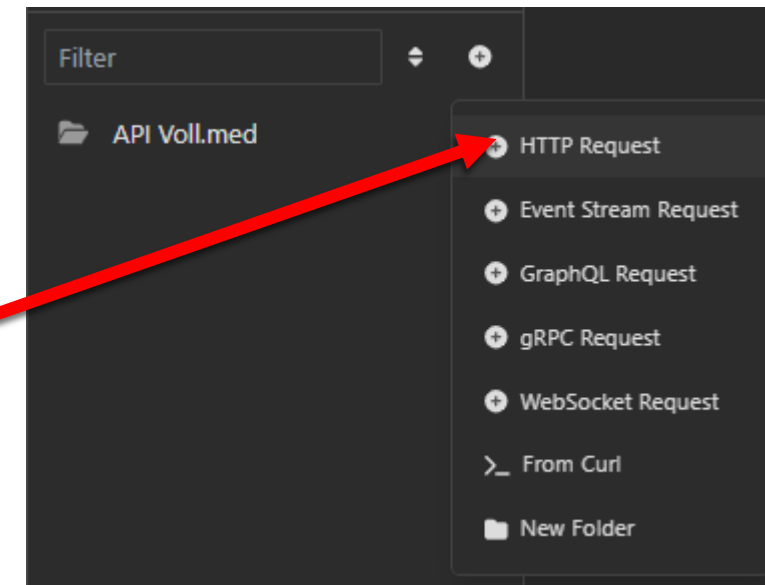
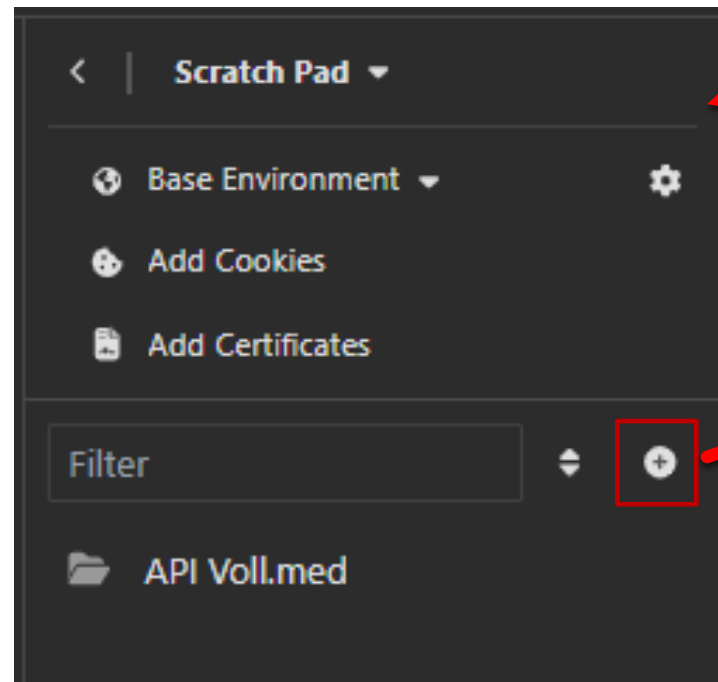
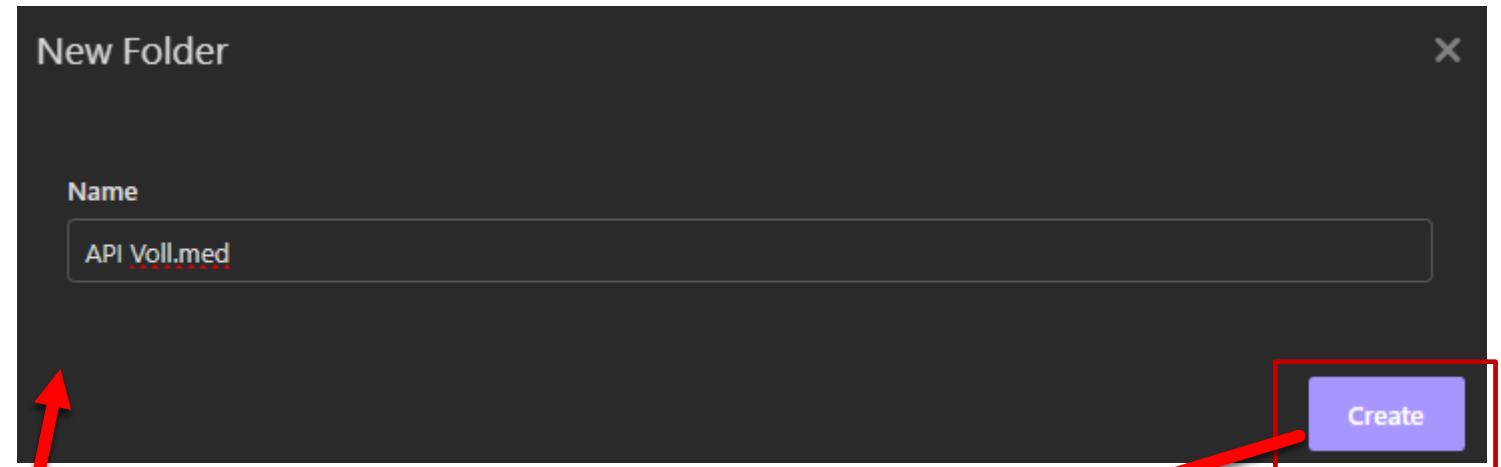
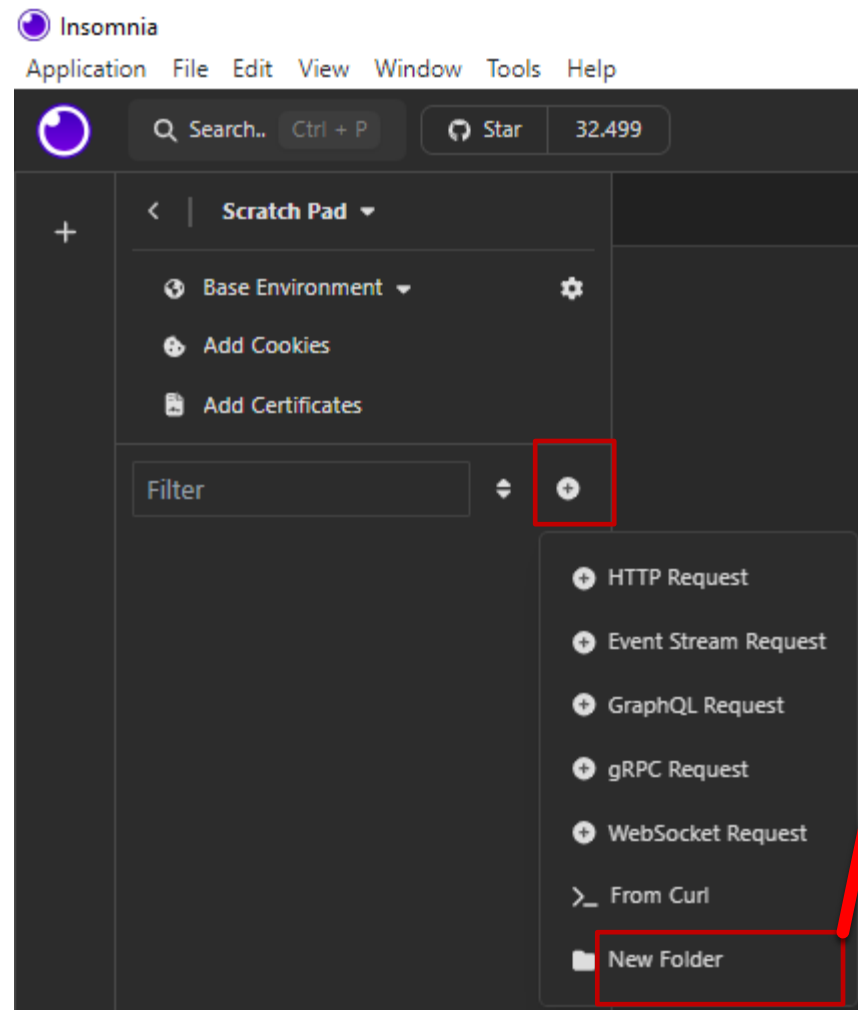
Ctrl + P

Edit Environments

Ctrl + E

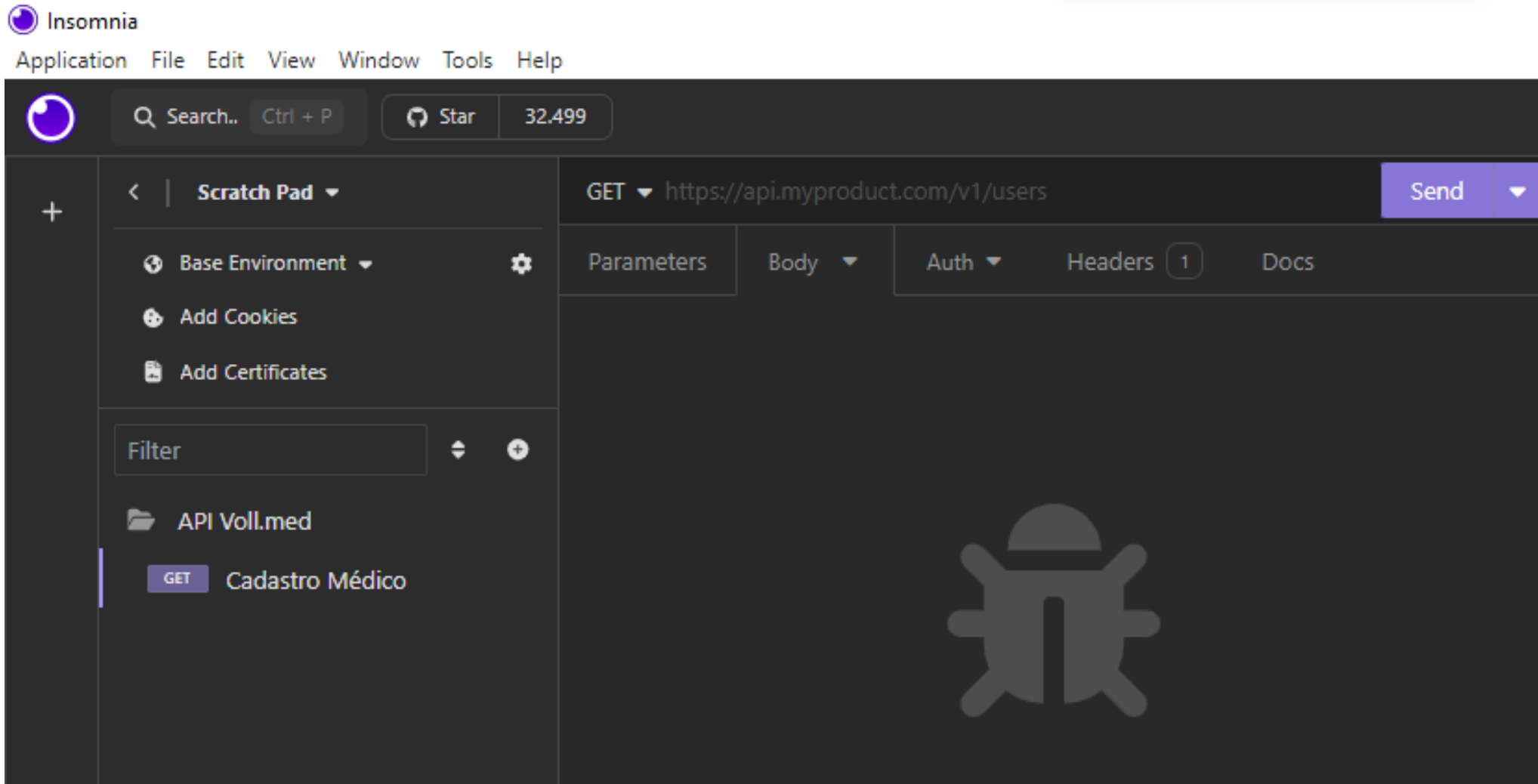
New HTTP Request

# Spring



# Spring

## Vamos Simular uma Requisição



Como queremos enviar dados para Para a API, ou seja queremos Cadastrar MÉDICOS, Temos que mudar o tipo da REQUISIÇÃO de GET (recebe/leitura) para POST (enviar dados para a API )

# Spring

## Vamos Simular uma Requisição

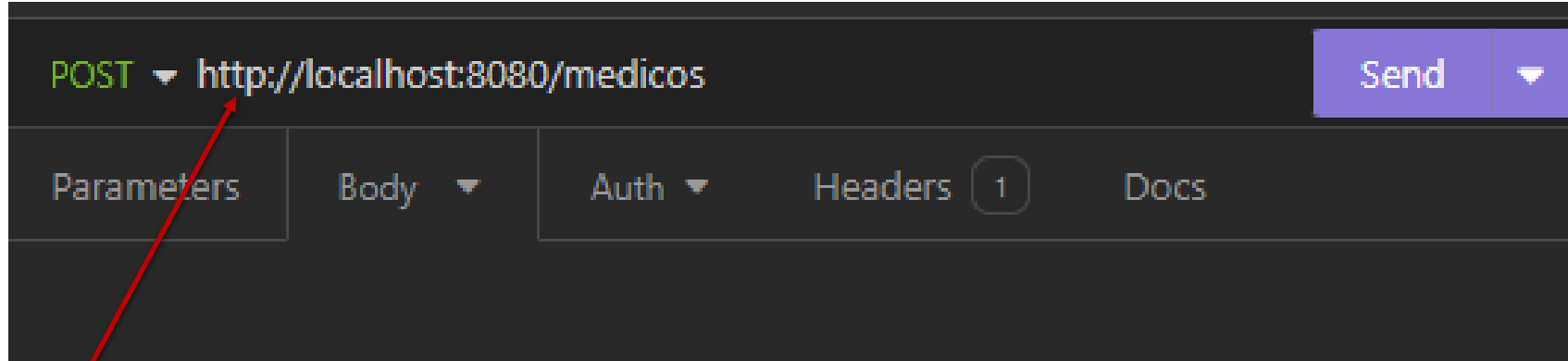
Application File Edit View Window Tools Help

The screenshot displays the Postman application interface. At the top, there is a search bar with the text "Search.." and a keyboard shortcut "Ctrl + P". To the right of the search bar is a "Star" button and a star count of "32.499".

The main interface is divided into three main sections:

- Left Sidebar:** Contains a "+" button for adding new requests. Below it, there is a "Scratch Pad" section with a dropdown arrow. Underneath, there are three options: "Base Environment" (with a gear icon), "Add Cookies", and "Add Certificates". At the bottom of the sidebar, there is a "Filter" input field and a list of API collections. One collection, "API Voll.med", is expanded, showing a single endpoint: "POST Cadastro Médico".
- Top Bar:** Displays the request method "POST" with a dropdown arrow, followed by the URL "https://api.myproduct.com/v1/users". To the right of the URL is a "Send" button and a dropdown arrow.
- Main Panel:** Contains tabs for "Parameters", "Body", "Auth", "Headers" (which is currently selected and has a count of "1"), and "Docs". The main area below these tabs is mostly empty, with a large, faint, stylized bug icon centered in the background.

## Vamos Simular uma Requisição

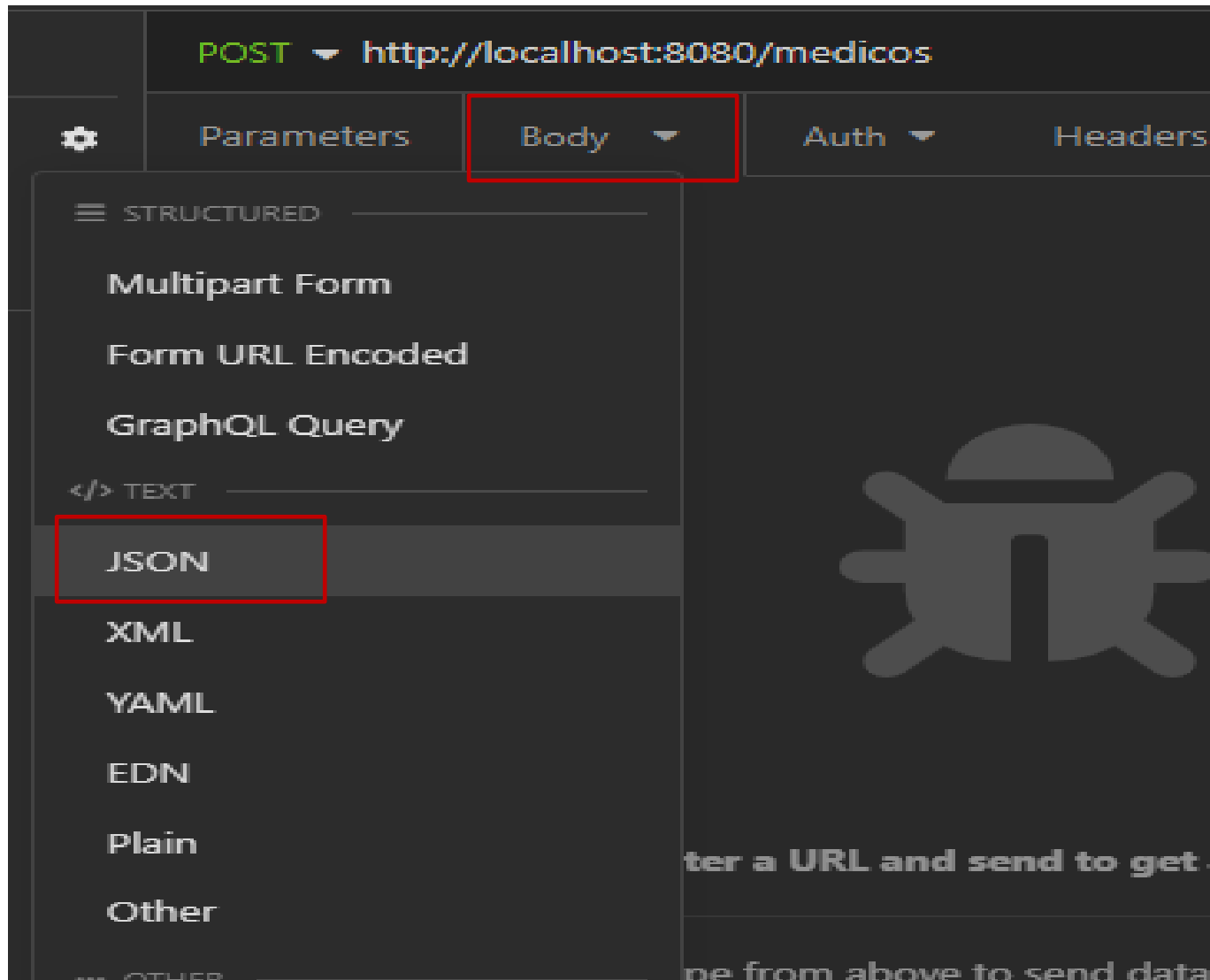


- Passando a Requisição na URL.
- Faltando passar os dados para a API

Como quer  
enviar dados  
Para a API,  
queremos  
Cadastrar M  
Temos que  
tipo da REC  
de GET  
(recebe/leit  
POST (envi  
para a API )

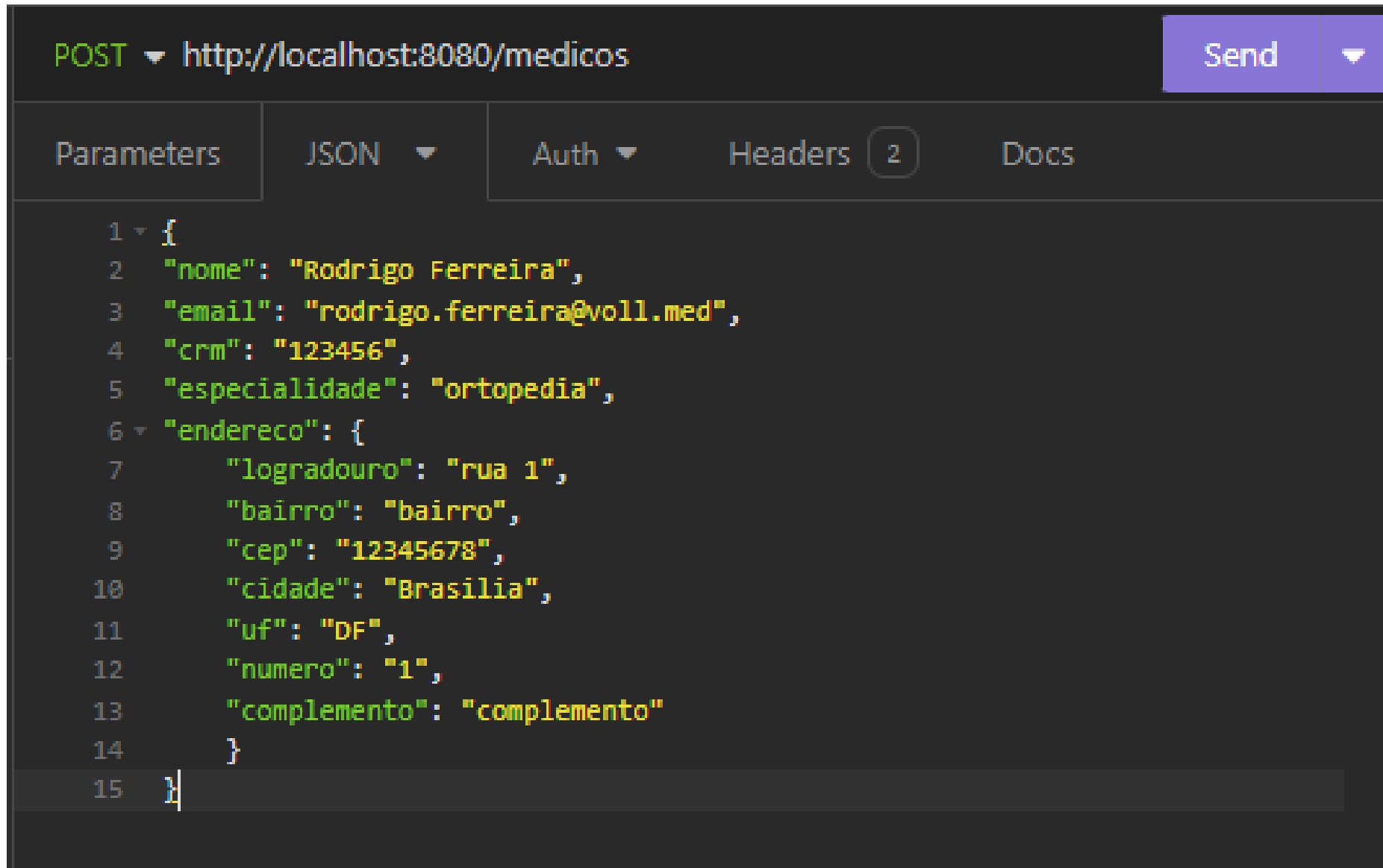
# Spring

Vamos Simular uma **Requisição**





Vamos Simular uma **Requisição**



The screenshot shows a REST client interface with a dark theme. At the top, the method is 'POST' and the URL is 'http://localhost:8080/medicos'. A 'Send' button is on the right. Below the URL bar are tabs for 'Parameters', 'JSON', 'Auth', 'Headers' (which has a '2' in a circle), and 'Docs'. The 'JSON' tab is selected, displaying a JSON body with 15 lines of code. The JSON represents a doctor with fields for name, email, CRM, specialty, and address (including street, neighborhood, CEP, city, state, number, and complement).

```
1 {  
2   "nome": "Rodrigo Ferreira",  
3   "email": "rodrigo.ferreira@voll.med",  
4   "crm": "123456",  
5   "especialidade": "ortopedia",  
6   "endereco": {  
7     "logradouro": "rua 1",  
8     "bairro": "bairro",  
9     "cep": "12345678",  
10    "cidade": "Brasilia",  
11    "uf": "DF",  
12    "numero": "1",  
13    "complemento": "complemento"  
14  }  
15 }
```

## Vamos Simular uma Requisição – COPIAR CÓDIGO

```
{  
  "nome": "Rodrigo Ferreira",  
  "email": "rodrigo.ferreira@voll.med",  
  "crm": "123456",  
  "especialidade": "ortopedia",  
  "endereco": {  
    "logradouro": "rua 1",  
    "bairro": "bairro",  
    "cep": "12345678",  
    "cidade": "Brasilia",  
    "uf": "DF",  
    "numero": "1",  
    "complemento": "complemento"  
  }  
}
```

O Json que irá representar os dados do cadastro do médico

Vamos montar a requisição via método POST

Uma vez copiado, clique no Botão SEND

## Vamos Simular uma Requisição

```
POST http://localhost:8080/medicos
```

Send

404 Not Found

329 ms

4.1 KB

Preview

Headers 6

Cookies

Timeline

```
1 {
2   "timestamp": "2024-02-25T17:29:44.190+00:00",
3   "status": 404,
4   "error": "Not Found",
```

Isso aconteceu porque estamos enviando uma requisição para o endereço `/medicos` que não está mapeado no controller.

Lembrando que temos no projeto somente o `/hello`.

```
"message": "No static resource medicos.",
"path": "/medicos"
}
```

Vamos Entender um pouco do **JSON**

- A palavra é um acrônimo de **JavaScript Object Notation**
- JSON é **um formato utilizado para representação de informações**, assim como **XML** e **CSV**.
- JSON é um padrão de formatação de dados **para troca de informações entre sistemas**.
- Principalmente usado **para transferir dados entre um servidor e um cliente**

Vamos Entender um pouco do **JSON**

- Uma **API** precisa receber e devolver informações em algum formato, que representa os recursos gerenciados por ela.
- O JSON é um desses formatos possíveis, tendo se **popularizado devido a sua leveza, simplicidade, facilidade de leitura por pessoas e máquinas, bem como seu suporte pelas diversas linguagens de programação.**

Vamos Entender um pouco do **JSON**

- Um exemplo de representação de uma informação no **formato XML** seria:

```
<produto>
  <nome> Mochila </nome>
  <preco> 89.90 </preco>
  <descricao> Mochila para notebooks de até 17 polegadas </descricao>
</produto>
```

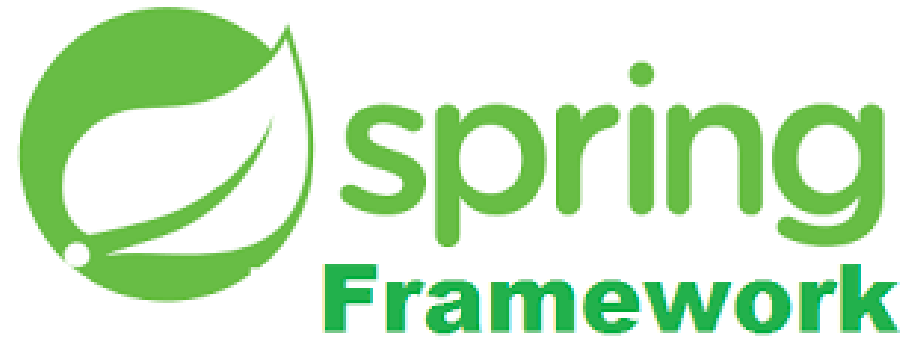
- Já a mesma informação poderia ser representada no formato JSON da seguinte maneira:

```
{
  "nome" : "Mochila",
  "preco" : 89.90,
  "descricao" : "Mochila para notebooks de até 17 polegadas"
}
```

Vamos Entender um pouco do **JSON**

- Perceba como o formato JSON é **muito mais compacto e legível**.
- Justamente por isso **se tornou o formato universal utilizado em comunicação de aplicações**, principalmente no caso de APIs REST.

## Recebendo Dados na API





Vamos Criar a Classe Médico

- É necessário criarmos uma classe controller,
- Seguindo o padrão do Spring MVC,
- O controller é o arquivo que mapeamos as requisições enviadas para nossa API.
- MedicoController

## Vamos Criar a Classe Médico

```
package med.voll.api.controller;

import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

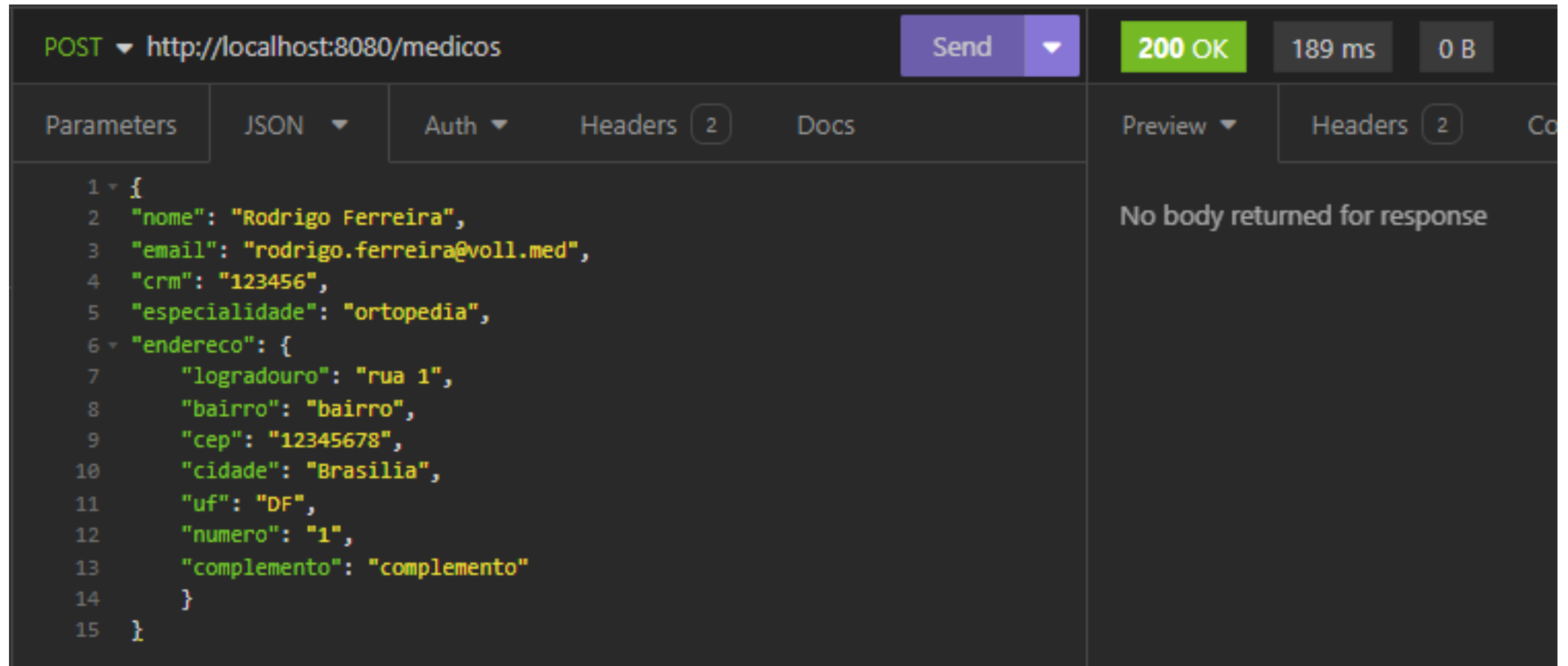
no usages
@RestController
@RequestMapping("medicos")
public class MedicoController {

    no usages
    @PostMapping
    public void cadastrar() {
    }
}
```

## Vamos Criar a Classe Médico

- Acima do método, é necessário especificarmos o verbo do protocolo HTTP que ele vai lidar.
- No caso, estamos enviando as requisições via verbo post, por isso, incluiremos a anotação @PostMapping.
- Estamos comunicando o Spring que ao chegar uma requisição do tipo post para a URL /medicos, ele deve chamar o método cadastrar da classe MedicoController.
- Vamos iniciar o servidor, depois vamos no insomnia, para disparar a requisição clicando no botão SEND

# Spring



- em "**Preview**", temos a mensagem: "No body returned for response" (em português, "**Nenhum corpo retornado para resposta**"). Isso significa que deu certo e que a requisição foi processada com sucesso!

# Spring

- Temos que passar o “**JSON**”, para o método cadastrar do Controller.

POST ▼ http://localhost:8080/medicos

Parameters

JSON ▼

Auth ▼

Headers

```
1 {  
2   "nome": "Rodrigo Ferreira",  
3   "email": "rodrigo.ferreira@voll.med",  
4   "crm": "123456",  
5   "especialidade": "ortopedia",  
6   "endereco": {  
7     "logradouro": "rua 1",  
8     "bairro": "bairro",  
9     "cep": "12345678",  
10    "cidade": "Brasilia",  
11    "uf": "DF",  
12    "numero": "1",  
13    "complemento": "complemento"  
14  }  
15 }
```

```
@PostMapping  
public void cadastrar(@RequestBody String json) {  
    System.out.println(json);  
}
```

- Roda a app / insomnia / Terminal
- Notem que temos o corpo da requisição no terminal

## Vamos Criar a Classe Médico

- Dentro do método vamos imprimir uma sequência de '\*\*\*\*'
- Vamos salvar, observe se o servidor reiniciou???
- Infelizmente não... Temos que configurar o DevTools
- Uma vez configurado, pare o servidor e reinicie.
- A ideia do DevTools é ao salvar o controller automaticamente ele reinicia o servidor sem que haja a necessidade do programador reiniciar o servidor

## Vamos Criar a Classe Médico

- Se quisermos receber os campos separados, não podemos receber como string.
- Será necessário criarmos uma classe, e nela declarar os atributos com os mesmos nomes que estão sendo recebidos pelo JSON.
- Faremos isso, pois queremos trabalhar com cada campo de forma separada.

## DTO com Java Record





- Conseguimos receber as informações enviadas pelo Insomnia no controller.
- Mas precisamos encontrar uma maneira de **não recebermos esses dados como string** e **sim receber o JSON inteiro como string**.
- Uma forma de recebermos cada campo isoladamente, **é não usar uma string como parâmetro do método cadastrar** e **sim uma classe**.
- Nela, declaramos os atributos com os mesmos nomes que constam no JSON.

# Spring

- Uma forma de recebermos cada campo isoladamente, **é não usar uma string como parâmetro do método cadastrar e sim uma classe.**
- Nela, declaramos os atributos com os mesmos nomes que constam no JSON.
- Por isso, **no método cadastrar** vamos alterar o parâmetro de string json para uma classe que criaremos para representar os dados enviados pela requisição.
- Chamaremos essa classe de DadosCadastroMedico, nomearemos o parâmetro de dados e no system out ao invés de json será dados.

# Spring

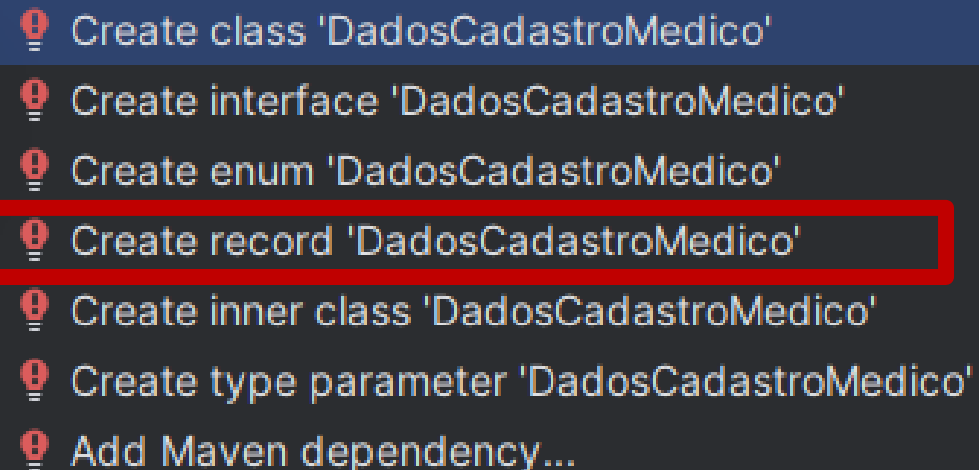
```
no usages
@PostMapping
public void cadastrar(@RequestBody DadosCadastroMedico dados) {
    System.out.println(dados);
}
}
```

- Alter + Enter
- Uma forma de recebermos cada campo isoladamente, é não usar uma string como parâmetro do método cadastrar e sim uma classe.
- Nela, declaramos os atributos com os mesmos nomes que constam no JSON.

# Spring

```
no usages
@PostMapping
public void cadastrar(@RequestBody DadosCadastroMedico dados) {
    System.out.println(dados);
}
}
```

- Alter + Enter



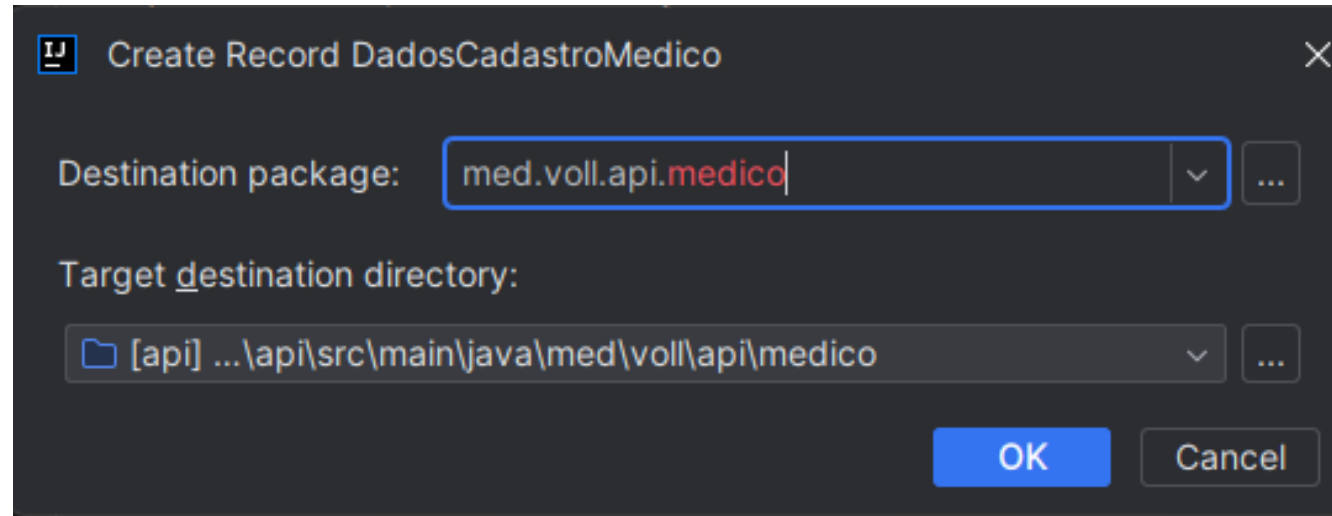
A screenshot of the IntelliJ IDEA suggestion menu that appears when pressing 'Alter + Enter' on the variable 'dados' in the code above. The menu lists several options to create or add 'DadosCadastroMedico'. The option 'Create record 'DadosCadastroMedico'' is highlighted with a red rectangular border.

- ⚠ Create class 'DadosCadastroMedico'
- ⚠ Create interface 'DadosCadastroMedico'
- ⚠ Create enum 'DadosCadastroMedico'
- ⚠ Create record 'DadosCadastroMedico'
- ⚠ Create inner class 'DadosCadastroMedico'
- ⚠ Create type parameter 'DadosCadastroMedico'
- ⚠ Add Maven dependency...

# Spring

- Note que DadosCadastroMedico está escrito na cor vermelha, isso significa que está com erro de compilação, porque ainda não criamos essa classe.
- Vamos criá-la, para isso podemos selecionar `DadosCadastroMedico` e depois usar o atalho "`Alt + Enter`". Será exibido um menu com várias opções, em que clicaremos na "`create record 'DadosCadastroMedico'`".
- Esses dados que estão chegando na API, usaremos o recurso de record (disponível nas últimas versões do Java). Este recurso funciona como se fosse uma classe imutável, para deixarmos o código simples.

# Spring



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

```
2 usages
```

```
public record DadosCadastroMedico() {  
}
```

# Spring

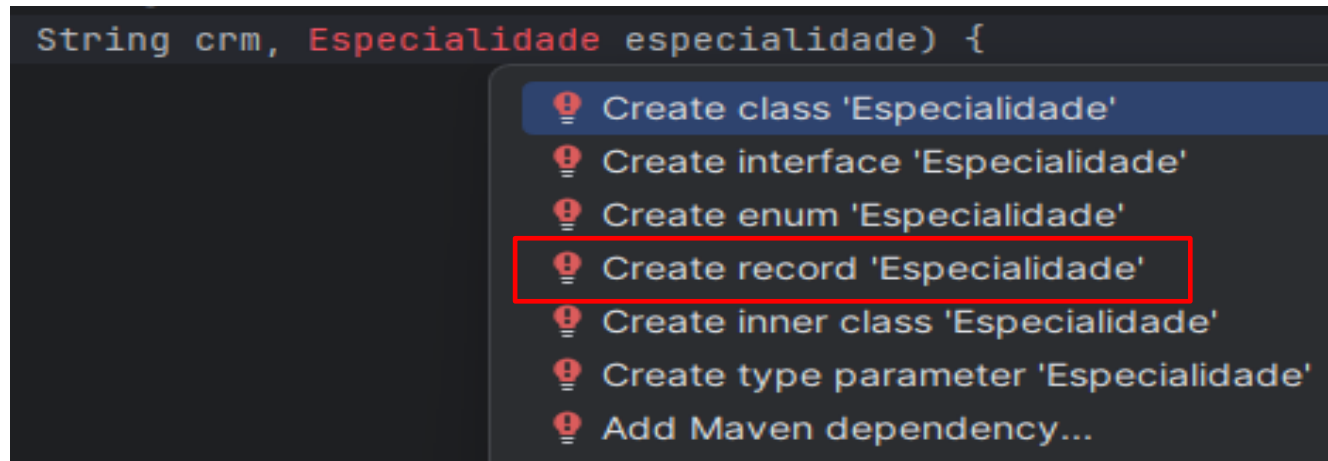
- No **parêntese do método record**, precisamos inserir os campos enviados pela requisição:

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

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

- Como o campo "**especialidade**" é fixo (temos quatro opções para a pessoa selecionar), **não usaremos string** e **sim um enum**.
- Podemos colocar após o **String crm**, o **Especialidade especialidade**.

# Spring



- Perceba que a palavra "`Especialidade`" está na cor vermelha, isso significa que não existe esse enum especialidade.
- Selecionando "`Especialidade`" usaremos o atalho "`Alt + Enter`", que exibirá um menu com diversas opções. Nele, escolheremos a opção "`Create enum 'Especialidade'`".
- No pop-up seguinte, com o título "`Create Enum Especialidade`", podemos simplesmente clicar no botão "Ok".



# Spring

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

1 usage
public enum Especialidade {
}
```

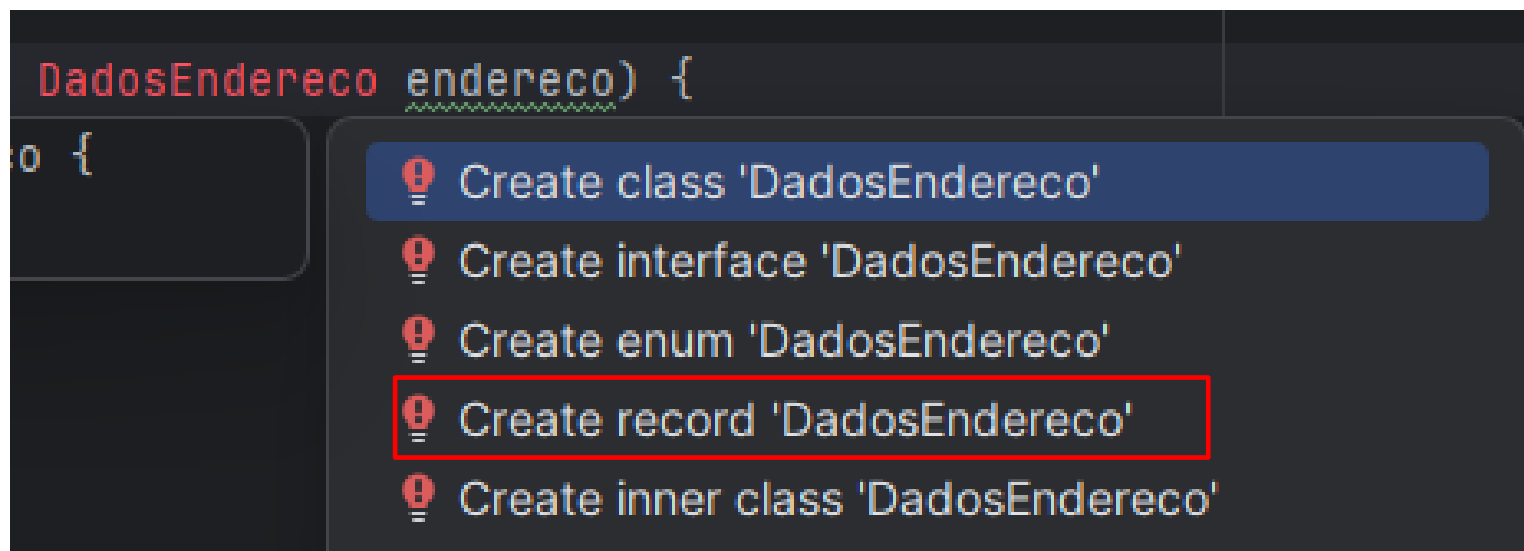
- Dentro das chaves do enum, vamos declarar as constantes, que são as opções do campo especialidade:

```
public enum Especialidade {
    no usages
    ORTOPEDIA,
    no usages
    CARDIOLOGIA,
    no usages
    GINECOLOGIA,
    no usages
    DERMATOLOGIA;
}
```

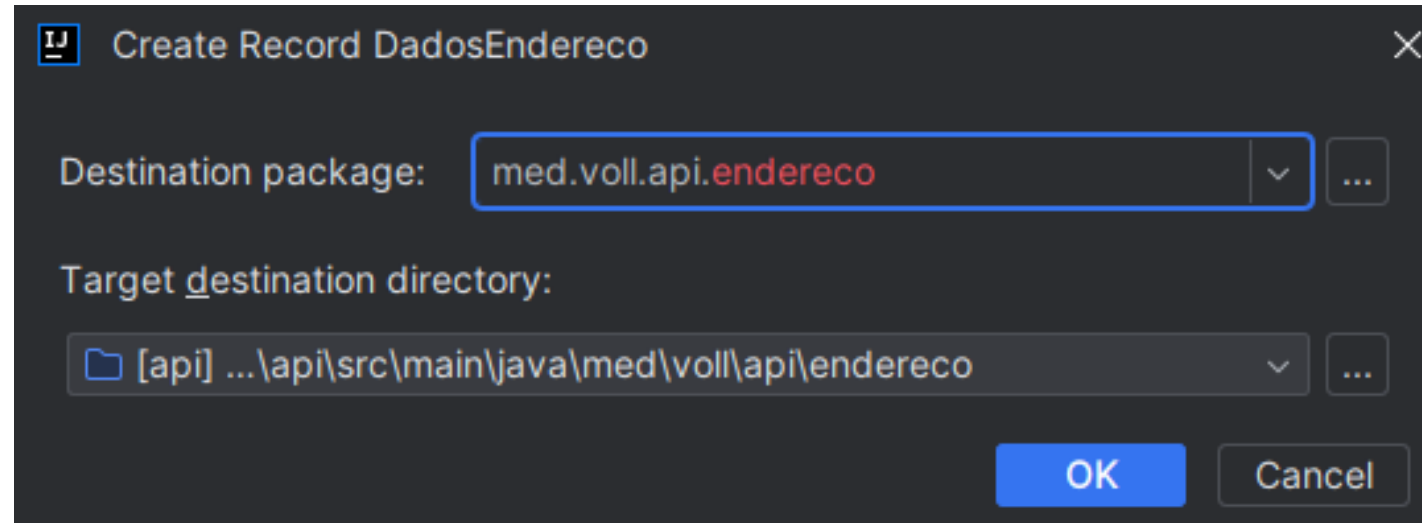
# Spring

- Após isso, voltaremos para o arquivo `DadosCadastroMedico` para **incluir** os outros campos no parâmetro do método cadastrar.
- O campo “**DadosEndereco endereço**” .

```
Especialidade especialidade, DadosEndereco endereco) {
```



# Spring



```
public record DadosEndereco(String logradouro, String bairro, String cep, String cidade,  
                             no usages  
                             String uf, String complemento, String numero) {  
}
```

String logradouro, String bairro, String cep, String cidade, String uf, String complemento, String numero

## DTO - Data Transfer Object

- É um padrão usado em APIs para representar os dados que chegam na API e também os dados que devolvemos dela.
- Isto é, sempre que precisarmos **receber ou devolver dados da API**, **criaremos um DTO** - sendo uma classe ou record que contém apenas os campos que desejamos receber ou devolver da API.
- Assim, recebemos os dados na nossa API e agora precisamos pegá-los para executarmos as validações, isso para verificar se os campos estão chegando corretamente, e depois salvar essas informações em um banco de dados.