02. Requisições POST

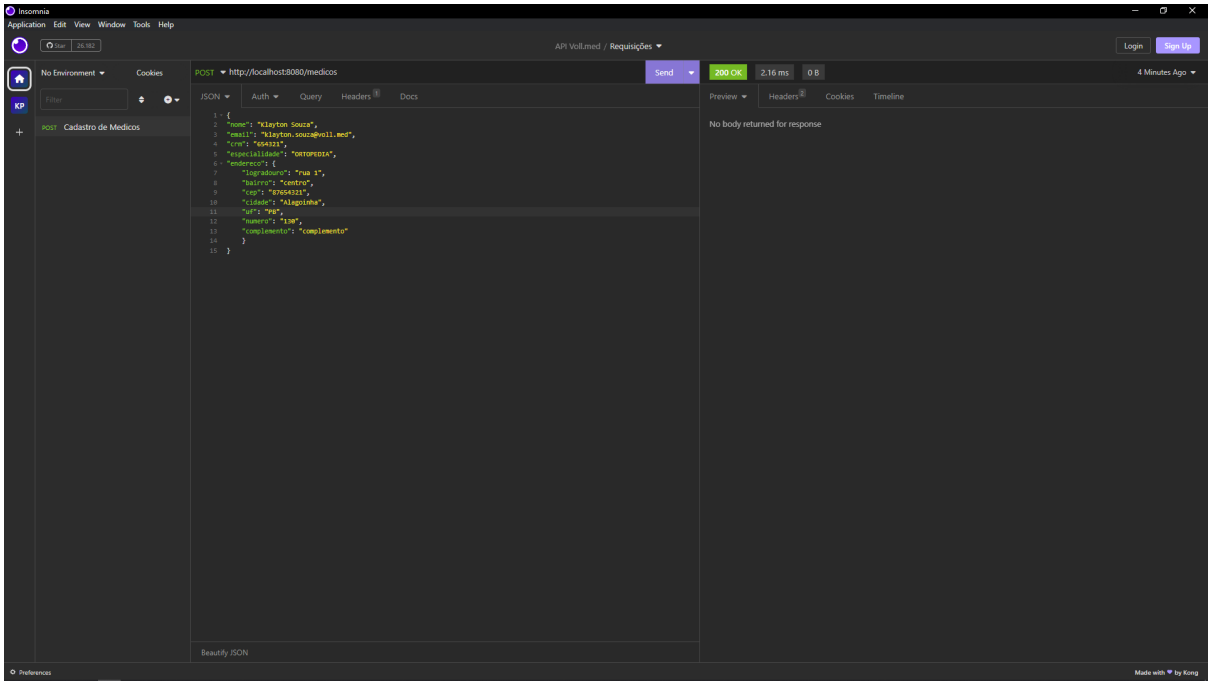
Date	@20/01/2023
Categoria	Java API
Curso	Spring Boot 3: desenvolva uma API Rest em Java

Tópicos

- Enviando dados para a API
- Preparando o ambiente: Trello e Figma
- Recebendo dados na API
- Para saber mais: JSON
- Para saber mais: lidando com CORS
- DTO com Java Record
- Para saber mais: Java Record
- JSON e DTO
- Faça como eu fiz: controller de pacientes
- O que aprendemos?

Recebendo dados na API

Aqui configuramos o insomnia com os dados que desejamos enviar, esses dados e um JSON, podemos ver o exemplo dessa requisição na imagem abaixo:



Para saber mais: JSON

JSON (*JavaScript Object Notation*) é um formato utilizado para representação de informações, assim como XML e CSV.

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.

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”
}
```

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. Mais detalhes sobre o JSON podem ser encontrados no site JSON.org.

Para saber mais: Java Record

Lançado oficialmente no Java 16, mas disponível desde o Java 14 de maneira experimental, o **Record** é um recurso que permite representar uma classe imutável, contendo apenas atributos, construtor e métodos de leitura, de uma maneira muito simples e enxuta.

Esse tipo de classe se encaixa perfeitamente para representar classes DTO, já que seu objetivo é apenas representar dados que serão recebidos ou devolvidos pela API, sem nenhum tipo de comportamento.

Para se criar uma classe DTO imutável, sem a utilização do Record, era necessário escrever muito código. Vejamos um exemplo de uma classe DTO que representa um telefone:

```
public final class Telefone {

    private final String ddd;
    private final String numero;

    public Telefone(String ddd, String numero) {
        this.ddd = ddd;
        this.numero = numero;
    }

    @Override
    public int hashCode() {
        return Objects.hash(ddd, numero);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        } else if (!(obj instanceof Telefone)) {
```

```
            return false;
        } else {
            Telefone other = (Telefone) obj;
            return Objects.equals(ddd, other.ddd)
              && Objects.equals(numero, other.numero);
        }
    }

    public String getDdd() {
        return this.ddd;
    }

    public String getNumero() {
        return this.numero;
    }
}
```

Agora com o Record, todo esse código pode ser resumido com uma única linha:

```
public record Telefone(String ddd, String numero){}
```

Muito mais simples, não?!

Por baixo dos panos, o Java vai transformar esse Record em uma classe imutável, muito similar ao código exibido anteriormente.

Mais detalhes sobre esse recurso podem ser encontrados na documentação oficial.

JSON e DTO

Em uma classe Controller existe o seguinte método declarado:

```
@PostMapping
public void cadastrar(DadosCadastroProduto dados) {
    System.out.println(dados);
}
```

E nesse projeto foi criado também o DTO `DadosCadastroProduto`:

```
public record DadosCadastroProduto(String nome, String descricao, BigDecimal preco){}
```

Você dispara uma requisição POST, enviando no corpo dela o seguinte JSON:

```
{
    “preco” : 399.99,
    “descricao” : “Wireless. Cor: branca”,
    “nome” : “Fone de ouvido”
}
```

Mas, ao verificar o console da IDE, percebe que os dados estão chegando todos como **null**. Escolha a alternativa CORRETA que indica o porquê os dados estão retornado como nulos:

- Faltou mapear no método do Controller que a requisição vai receber dados no formato JSON.
    - O Spring já identifica automaticamente quando os dados chegam para a API no formato JSON.
- A ordem dos campos no JSON não é a mesma ordem declarada no DTO.
    - Para o Spring, a ordem dos campos do JSON é indiferente, sendo que apenas os **nomes**
      é que devem ser iguais aos que foram declarados no DTO.
- Faltou anotar o parâmetro `dados`, recebido no método `cadastrar` do Controller, com **@RequestBody**.
    - Sem essa anotação o Spring não vai ler o corpo da requisição e mapear os campos dele para o DTO recebido como parâmetro.

O que aprendemos?

Nessa aula, você aprendeu como:

- Mapear requisições POST em uma classe Controller;
    - Utilizando a anotação `@PostMapping`
- Enviar requisições POST para a API utilizando o Insomnia;
    - No tópico Recebendo dados na API foi exemplificado como foi realizado essa requisição
- Enviar dados para API no formato JSON;
- Utilizar a anotação `@RequestBody` para receber os dados do corpo da requisição em um parâmetro no Controller;

- Utilizar o padrão **DTO (*Data Transfer Object*)**, via Java Records, para representar os dados recebidos em uma requisição POST.
    - O Java Records e bastante utilizado em casos que precisamos de uma classe imutável

---