Tecnologias Para Back-End

Prof. JUNIO FIGUEIRÊDO

JUNIOINF@GMAIL.COM

AULA 09 – REQUISIÇÕES PUT....

Requisições Put

Spring

- Agora vamos aprender a funcionalidade de atualização.
- Quando editarmos os perfis dos médicos, nossa API receberá uma nova requisição com os novos dados.
- Precisamos, porém, saber qual médico precisa ter seus dados atualizados no banco de dados.
- A forma que temos para identificar os registros é o ID.
- O problema é que, por enquanto, o ID não está sendo devolvido na funcionalidade de listagem.
- Por isso, vamos alterar o DTO de listagem, para que ele inclua o ID.

Spring

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


    1 usage
    public DadosListagemMedico(Medico medico) {
        this(medico.getNome(), medico.getEmail(), medico.getCrm(),
    }
}
```

Spring

- No DTO "DadosListagemMedico.java".
- Vamos adicionar Long id dentro dos parênteses.

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

- Para que não haja erro de compilação no construtor, vamos passar o ID no this, passando medico.getId() entre parênteses:

```
public DadosListagemMedico(Medico medico) {
    this(   medico.getId(),
            medico.getNome(),
            medico.getEmail(),
            medico.getCrm(),
            medico.getEspecialidade());
}
```

Spring

- Agora basta salvar.
- Vamos adicionar o método public_void atualizar()
- Dentro do "MedicoController.java" .

```
@PutMapping
@Transactional
public void atualizar(@RequestBody @Valid DadosCadastroMedico dados) {
```

- Porém, não poderemos usar DadosCadastroMedico.
- Precisaremos criar outro DTO.

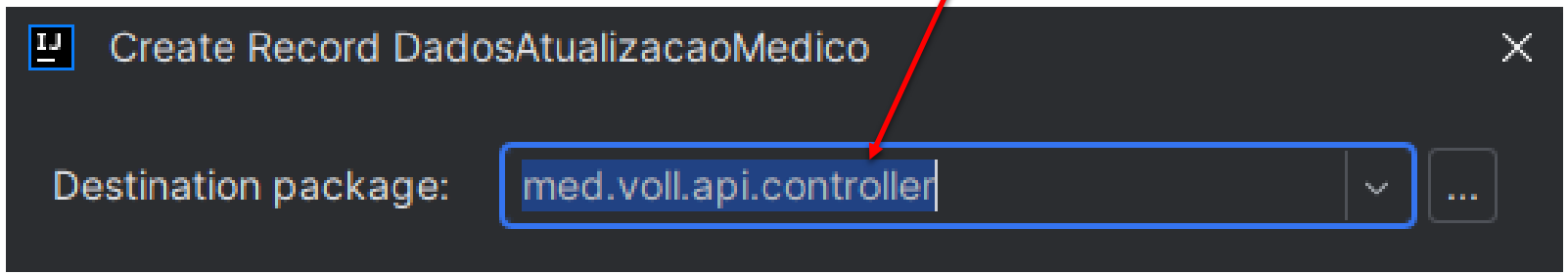
Atualizando Dados

Spring

- Ao invés de DadosCadastroMedico, usaremos DadosAtualizacaoMedico.

```
@PutMapping
@Transactional
public void atualizar(@RequestBody @Valid DadosAtualizacaoMedico dados) {

}
```

- Com o atalho "Alt + Enter > Create record 'DadosAtualizacaoMedico'", vamos substituir o pacote para "med.voll.api.medico" e clicar no OK.



- Receberemos apenas o ID e os dados atualizáveis, nome, telefone e endereço.
- Passaremos todas como strings, exceto o endereço, para o qual usaremos o DTO DadosEndereco endereco.
- As validações dela, como logradouro e CEP, permanecerão.
- Reaproveitamos o DTO.
- Caso contrário teríamos que criar um novo DTO, por exemplo (não é possível alterar a UF), Nesta situação teríamos que criar um DTO para endereço aonde não existiria o atributo UF !!!!!

Spring

- No campo ID, adicionaremos a anotação @NotNull, para informar que ele é obrigatório:

```
public record DadosAtualizacaoMedico(
        no usages
        @NotNull
        Long id,
        no usages
        String nome,
        no usages
        String telefone,
        no usages
        DadosEndereco endereco) {
}
```

```
public record DadosAtualizacaoMedico(
        @NotNull
        Long id,
        String nome,
        String telefone,
        DadosEndereco endereco) {
}
```

Spring

- No campo ID, adicionaremos a anotação @NotNull, para informar que ele é obrigatório:

```
public record DadosAtualizacaoMedico(
        no usages
        @NotNull
        Long id,
        no usages
        String nome,
        no usages
        String telefone,
        no usages
        DadosEndereco endereco) {
}
```

Spring

- De volta a "MedicoController.java", no método atualizar.
- Vamos carregar o objeto no banco de dados, criando a variável.
- var medico = repository.getReferenceById(dados.id());
- busca a referencia dessa entidade no bd pelo id
- já o id está dentro do nosso DTO, dessa forma carrego o médico pelo id, que está chegando pelo DTO

```
public void atualizar(@RequestBody @Valid DadosAtualizacaoMedico dados) {
    var medico = repository.getReferenceById(dados.id());
}
```

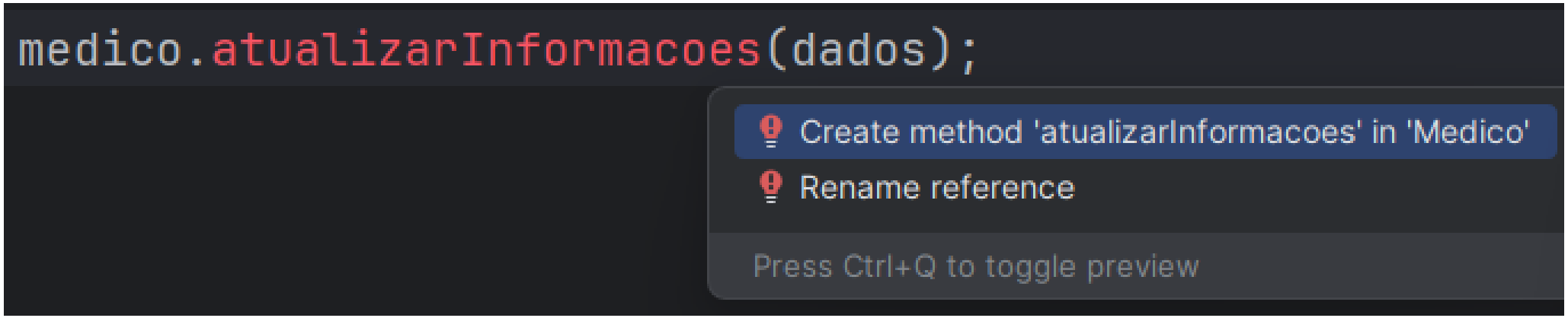
Spring

- E agora o que irei fazer com esse objeto médico????
- Que está com as informações desatualizadas!!!!
- Precisamos atualizar/sobrescrever : nome, telefone e endereço.
- Conforme dados que estão vindo do DTO.
- Ainda no método atualizar, vamos inserir um método para atualizar os dados, passando como parâmetro dados, que está chegando via DTO.
- Esse método chamado de atualizarInformações, ele irá substituir os campos que posem ser atualizados

Spring

```java
public void atualizar(@RequestBody @Valid DadosAtualizacaoMedico dados) {
    var medico = repository.getReferenceById(dados.id());
    medico.atualizarInformacoes(dados);
}
```

- Vamos registrar o novo método clicando em "Alt + Enter > Create method".



Spring

- Seremos redirecionados para a classe "Medico.java".
- Lá, será criada a assinatura.

```
public void atualizarInformacoes(DadosAtualizacaoMedico dados) {
}
```

- Vamos adicionar a implementação dentro das chaves.
- Passaremos this.nome = dados.nome().
- Pegaremos o nome do médico atual e substitui-lo pelo nome que chega via DTO.

- Como queremos atualizar apenas se o campo for enviado, precisaremos adicionar um if.

```
public void atualizarInformacoes(DadosAtualizacaoMedico dados) {
    if (dados.nome() != null) {
        this.nome = dados.nome();
    }
    if (dados.telefone() != null) {
        this.telefone = dados.telefone();
    }
    if (dados.endereco() != null) {
        this.endereco.atualizarInformacoes(dados.endereco());
    }
}
```

- No endereço, porém, criaremos o método atualizarInformacoes() na classe Endereco,, passando dentro do if e selecionando "Alt + Enter > Create method":

```
if (dados.endereco() != null) {
    this.endereco.atualizarInformacoes(dados.endereco());
}
```

Create method 'atualizarInformacoes' in 'Endereco'

Rename reference

```
public void atualizarInformacoes(DadosEndereco endereco) {
}
```

Spring

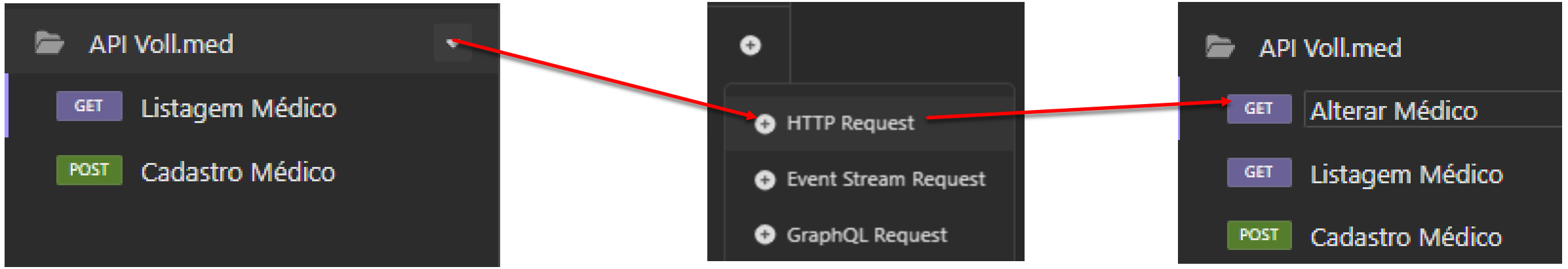
```java
public void atualizarInformacoes(DadosEndereco dados) {
    if (dados.logradouro() != null) {
        this.logradouro = dados.logradouro();
    }
    if (dados.bairro() != null) {
        this.bairro = dados.bairro();
    }
    if (dados.cep() != null) {
        this.cep = dados.cep();
    }
    if (dados.uf() != null) {
        this.uf = dados.uf();
    }
    if (dados.cidade() != null) {
        this.cidade = dados.cidade();
    }
    if (dados.numero() != null) {
        this.numero = dados.numero();
    }
    if (dados.complemento() != null) {
        this.complemento = dados.complemento();
    }
}
```

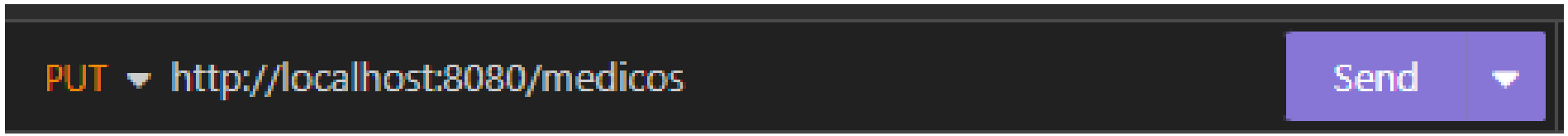
```
public void atualizarInformacoes(DadosEndereco dados) {

    if (dados.logradouro() != null) {
        this.logradouro = dados.logradouro();
    }
    if (dados.bairro() != null) {
        this.bairro = dados.bairro();
    }
    if (dados.cep() != null) {
        this.cep = dados.cep();
    }
    if (dados.uf() != null) {
        this.uf = dados.uf();
```

```
    if (dados.cidade() != null) {
        this.cidade = dados.cidade();
    }
    if (dados.numero() != null) {
        this.numero = dados.numero();
    }
    if (dados.complemento() != null) {
        this.complemento = dados.complemento();
    }
}
```

Spring

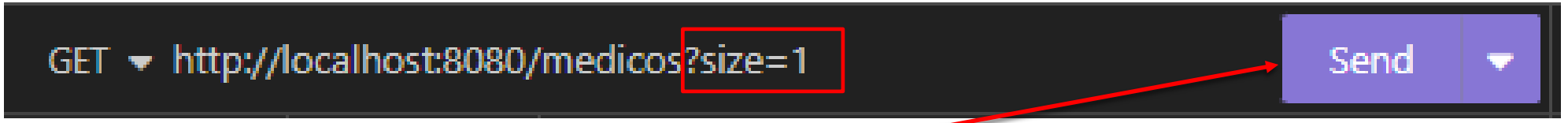
- Agora basta salvar.
- Vamos ao Insomnia, Vamos criar uma nova requisição.





Spring

- Ainda no Insomnia, selecione a requisição Listagem Médico.
- Caso na requisição tenha algum parâmetro de paginação exclua.

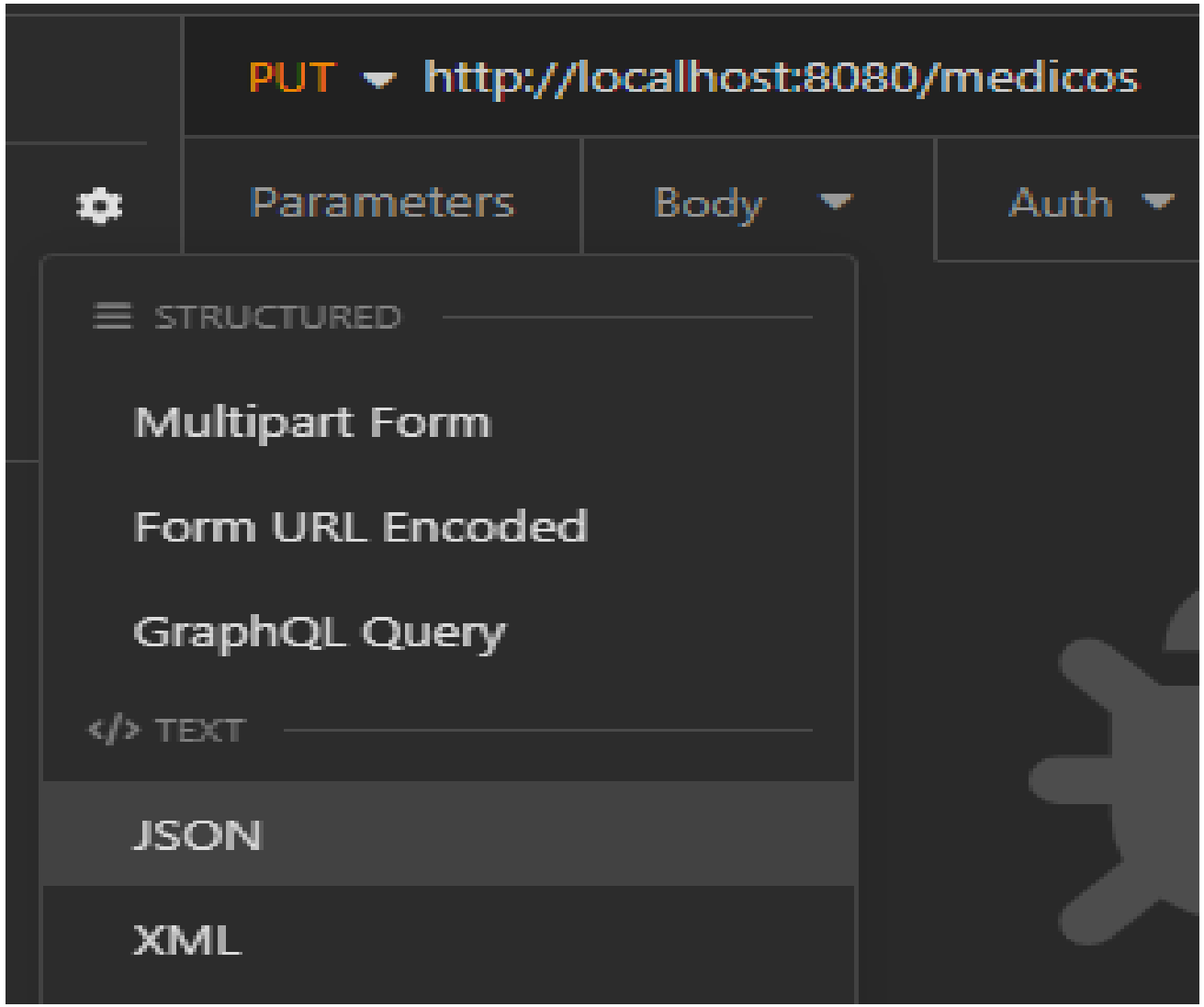


- Realize a requisição
- Por exemplo, vamos atualizar o id = 12.
- Vá na requisição de Atualizar


```
  "id": 14,
  "nome": "Ana Julia Maria Luiza Carlo",
  "email": "jsanamajoaeeeo.ferreira@voll.med",
  "crm": "571034",
  "especialidade": "ORTOPEDIA"
},
{
  "id": 12,
  "nome": "Ana Maria Luiza Carlo",
  "email": "anamajoaeeeo.ferreira@voll.med",
  "crm": "471034",
  "especialidade": "ORTOPEDIA"
},
{
  "id": 8,
  "nome": "Antonio Carlos",
  "email": "joaeeeo.ferreira@voll.med",
  "crm": "871034",
  "especialidade": "ORTOPEDIA"
},
```

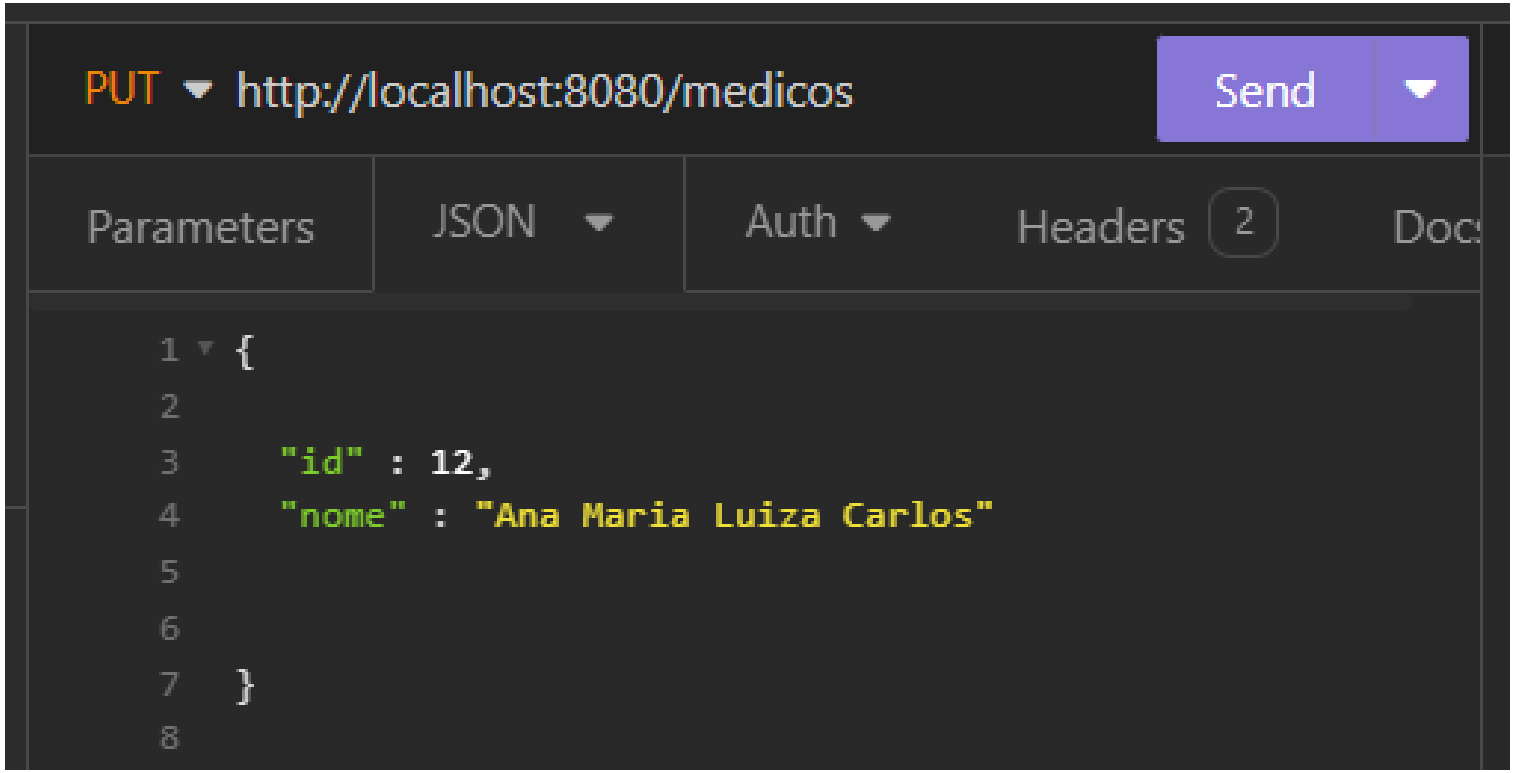
Spring

- No Body escolha JSON
-



Spring

- Na requisição de Atualizar.



- Clique em Send.
- Veja que a resposta foi 200

Spring

- Vamos conferir se realmente o id = 12, foi alterado.
- Execute a requisição Listagem Médico.
- Veja que o nome foi alterado.


```
{
  "id": 12,
  "nome": "Ana Maria Luiza Carlos",
  "email": "anamajoaeeeo.ferreira@voll.med",
  "crm": "471034",
  "especialidade": "ORTOPEDIA"
},
```

Put vs Patch

Spring

Escolher entre o método HTTP PUT ou PATCH é uma dúvida comum que surge quando estamos desenvolvendo APIs e precisamos criar um endpoint para atualização de recursos. Vamos entender as diferenças entre as duas opções e quando utilizar cada uma.

PUT

O método PUT substitui todos os atuais dados de um recurso pelos dados passados na requisição, ou seja, estamos falando de uma atualização integral. Então, com ele, fazemos a atualização total de um recurso em apenas uma requisição.

PATCH

O método PATCH, por sua vez, aplica modificações **parciais** em um recurso. Logo, é possível modificar apenas uma parte de um recurso. Com o PATCH, então, realizamos atualizações parciais, o que torna as opções de atualização mais flexíveis.

Qual escolher?

Na prática, é difícil saber qual método utilizar, pois nem sempre saberemos se um recurso será atualizado parcialmente ou totalmente em uma requisição - a não ser que realizemos uma verificação quanto a isso, algo que não é recomendado.

O mais comum então nas aplicações é utilizar o método PUT para requisições de atualização de recursos em uma API

Spring