



PRW3 - Programação para a WEB III

# API REST com Spring Boot (parte 2) GET

Conteúdo 08

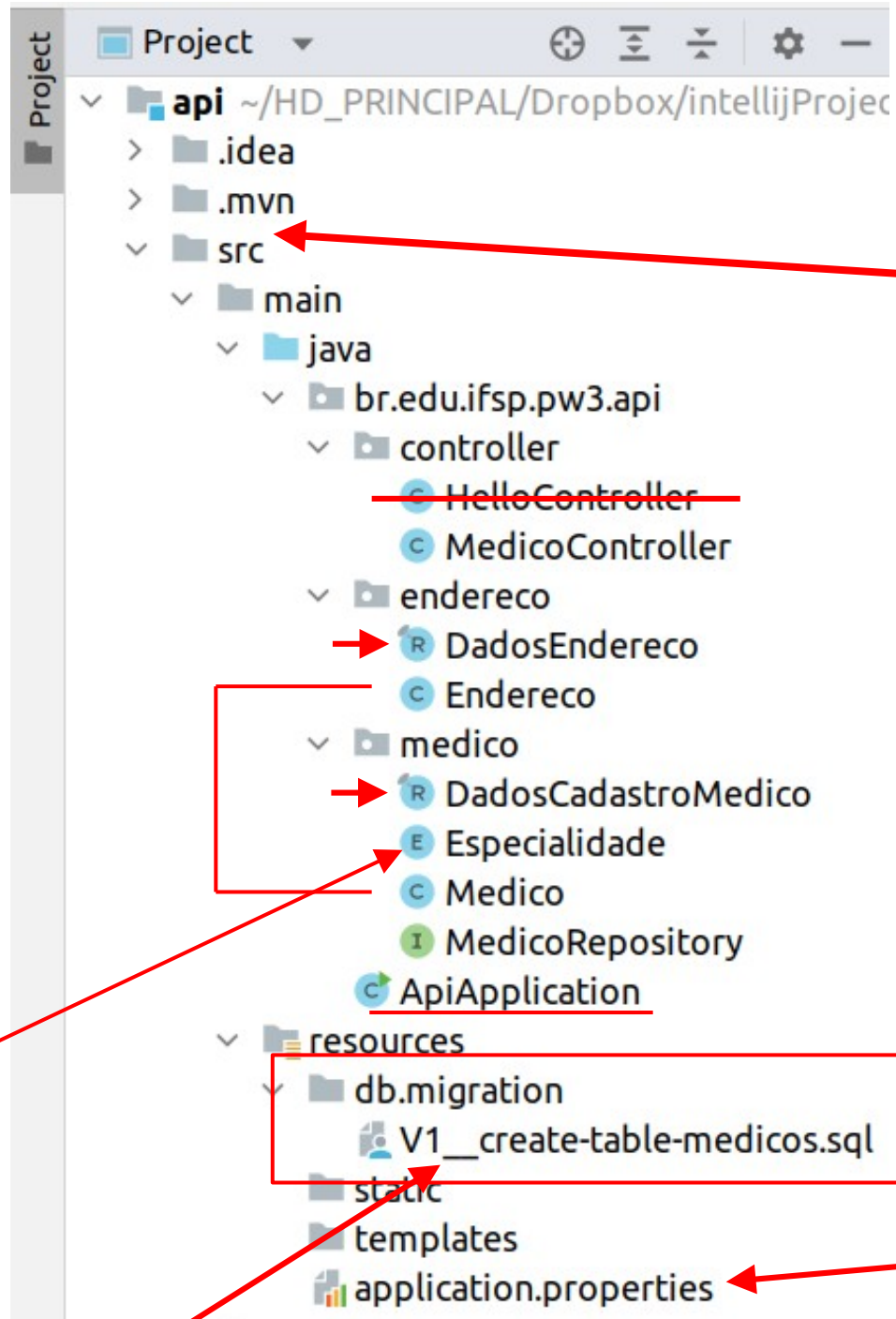


# AVISO!! (será repetido todas as aulas)

- Mantenha a disposição de cabos nos computadores dos labs.
- Se retirar um cabo de rede (para usar notebook), não esqueça de repor no computador ao final da aula.
- Mantenha o filtro de linha e as tomadas, atrás da mesa, em seu estado original, especialmente para que não possa ser esbarrado por outros alunos.
- Não altere de jeito nenhum os cabos atrás dos computadores, especialmente mexendo nas “travas”.
- **Por favor, desliguem a(s) máquina(s) ao final da aula.**
- Lembre-se! Os laboratórios são nossos!! Vamos cuidar!



# Lembrando o que foi feito até agora...



Colocamos o arquivo de dados do H2 numa pasta DATA

# Validação das informações que estão chegando

- Está vindo tudo que precisa vir ?
  - Formato correto ?
  - Tem campo obrigatório ?
  - Número ? Letra ?
  - Tamanho mínimo ?
  - Tamanho máximo ?
  - etc...
- 
- Módulo Validation, do Spring Boot
    - Inserimos a dependência
    - Integrada com uma especificação do Java, “Bean Validation”



# Validação das informações que estão chegando

- Está vindo tudo que precisa vir ?
- Formato correto ?
- Tem campo obrigatório ?
- Número ? Letra ?
- Tamanho mínimo ?
- Tamanho máximo ?
- etc...

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-validation</artifactId>  
</dependency>
```

- Módulo Validation, do Spring Boot
  - Inserimos a dependência
  - Integrada com uma especificação do Java, “Bean Validation” \*



# Bean

Em **Java** (e também no **Spring**), um *bean* é simplesmente um **objeto instanciado, configurado e gerenciado pelo contêiner de Inversão de Controle (IoC)**.

## Conceito

- No Java tradicional, "JavaBean" era uma classe com atributos privados, construtor sem argumentos e métodos *getters* e *setters*.
- No **Spring**, o termo "bean" ganhou um sentido mais amplo: qualquer objeto que o contêiner do Spring cria e gerencia dentro do contexto da aplicação.





A especificação Java "Bean Validation" é um padrão que facilita a validação de dados em aplicações Java. Ela permite definir restrições de validação usando anotações diretamente nas classes do seu modelo de negócio. Isso significa que você pode especificar se um campo é obrigatório, o tamanho mínimo ou máximo de uma string, o formato de um email, entre outras validações.

Vantagens da validação de Bean:

- **Código mais limpo e organizado:** As regras de validação ficam separadas da lógica da aplicação, deixando o código mais fácil de ler e manter.
- **Reuso de validações:** As anotações podem ser aplicadas em diferentes camadas da aplicação, evitando código duplicado.
- **Flexibilidade:** A especificação oferece suporte a diversas restrições prontas e permite a criação de validações customizadas.

Alguns frameworks, como JSF e Hibernate, integram-se com a validação de Bean, simplificando ainda mais o processo.

# Como fazer ?

if campo tal...

if campo tal...

if campo tal...

if campo tal...

...





# Como fazer ?

if campo tal...

if campo tal...

if campo tal...

if campo tal...

...



Os dados chegam através de nosso DTO  
(record)



R DadosCadastroMedico.java x

```
1 package br.edu.ifsp.pw3.api.medico;
2
3 import br.edu.ifsp.pw3.api.endereco.DadosEndereco;
4
5 public record DadosCadastroMedico(String nome, String email, String crm,
6                                     Especialidade especialidade,
7                                     DadosEndereco endereco) {
8
9 }
```



Vamos colocar anotações nos campos  
deste Record/DTO.

Que anotações ?



## ■ @NotNull

- Esta anotação é usada para marcar um campo como obrigatório (não nulo). Se um valor nulo for atribuído a um campo marcado com @NotNull, uma exceção de validação será lançada.

## ■ @NotBlank

- Essa anotação é usada para validar se uma string não está em branco (não consiste apenas de espaços em branco) **e não é nula**. Ela é frequentemente usada para garantir que campos de texto não estejam vazios.

## ■ @Email

- A anotação @Email é usada para validar se uma string representa um endereço de e-mail válido de acordo com um conjunto de regras. Ela ajuda a garantir que o valor fornecido seja um endereço de e-mail válido **com relação ao formato**.



## ■ @Pattern

- Esta anotação permite que você especifique um padrão regular que uma string deve seguir. Ela é usada para validar se o valor de um campo atende a um padrão específico. Você fornece uma **expressão regular\*** como argumento para essa anotação.

## ■ @Valid

- A anotação @Valid é usada em associação com **validação de objetos aninhados**. Ela indica que o Spring deve **validar o objeto aninhado** (geralmente uma propriedade de outra classe) durante a validação do objeto principal. Isso é útil **quando você tem objetos compostos e deseja validar todos os seus componentes**.
- Vamos usar para dizer que o “Endereço” (outro DTO) também terá anotações de validação, que devem ser respeitadas.



# Expressão Regular

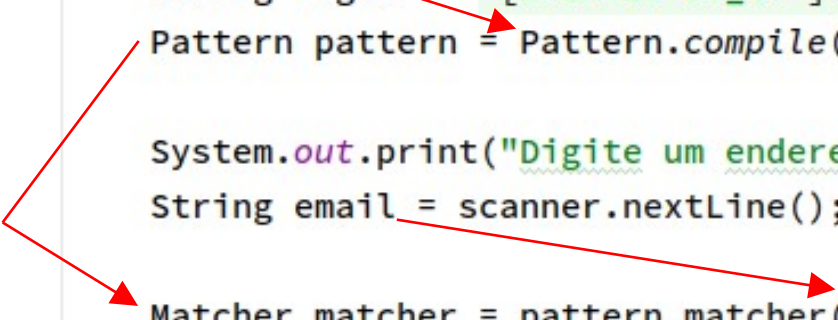
- Também conhecida como regex ou regexp, é uma **sequência de caracteres** que define um padrão de pesquisa. Ela é usada para procurar, corresponder e **validar strings com base nesse padrão**. As expressões regulares são úteis para **verificar se uma string se encaixa em um formato específico** ou contém um padrão desejado.
- Aqui estão exemplos de expressões regulares:
  - Validação de Endereço de E-mail:
    - `^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`
  - Validação de Número de Telefone (Formato Internacional):
    - `^\+\d{2} \d{2} \d{5}-\d{4}$`
  - Validação de CPF (Cadastro de Pessoa Física) Brasileiro:
    - `^\d{3}\.\d{3}\.\d{3}-\d{2}$`
- Lembre-se de que as expressões regulares podem variar em complexidade e detalhes dependendo dos requisitos específicos de validação. **Esses exemplos são simplificados para fins ilustrativos e podem não cobrir todos os casos possíveis.** Em situações reais, é importante adaptar as expressões regulares de acordo com as necessidades de validação específicas de seu aplicativo.

# Exemplo simples de expressão regular ...

(Java puro, nada a ver com Spring...)



```
public class ValidacaoEmail {  
  
    public static void main(String[] args) {  
  
        Scanner scanner = new Scanner(System.in);  
  
        // Padrão de expressão regular para validar o e-mail:  
        String regex = "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$";  
        Pattern pattern = Pattern.compile(regex);  
  
        System.out.print("Digite um endereço de e-mail: ");  
        String email = scanner.nextLine();  
  
        Matcher matcher = pattern.matcher(email);  
  
        if (matcher.matches()) {  
            System.out.println("O endereço de e-mail é válido.");  
        } else {  
            System.out.println("O endereço de e-mail não é válido.");  
        }  
    }  
}
```



```
public class Val
```

```
public stati
```

```
Scanner
```

```
// Padrã
```

```
String r
```

```
Pattern
```

```
System.o
```

```
String e
```

```
Matcher
```

```
if (matc
```

```
Syst
```

```
} else {
```

```
Syst
```

```
}
```

```
}
```

Run: ValidacaoEmail x



```
/usr/lib/jvm/jdk-20/bin/java ...
```



```
Digite um endereço de e-mail: carlao2005@ifsp.edu.br
```

```
O endereço de e-mail é válido.
```



```
Process finished with exit code 0
```

Run: ValidacaoEmail x



```
/usr/lib/jvm/jdk-20/bin/java ...
```



```
Digite um endereço de e-mail: carlao2005.ifsp.edu.br
```

```
O endereço de e-mail não é válido.
```



```
Process finished with exit code 0
```

Run: ValidacaoEmail x



```
/usr/lib/jvm/jdk-20/bin/java ...
```



```
Digite um endereço de e-mail: carlao2005@ifsp
```

```
O endereço de e-mail não é válido.
```



```
Process finished with exit code 0
```

Voltando aos DTO's ...

```
package br.edu.ifsp.pw3.api.medico;

import br.edu.ifsp.pw3.api.endereco.DadosEndereco;
import jakarta.validation.Valid;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Pattern;

public record DadosCadastroMedico(

    @NotBlank
    String nome,

    @NotBlank
    @Email
    String email,

    @NotBlank
    @Pattern(regexp = "\\d{4,6}") // 4 a 6 dígitos
    String crm,

    @NotNull
    Especialidade especialidade,

    @NotNull
    @Valid
    DadosEndereco endereco) {

}
```

```
package br.edu.ifsp.pw3.api.medico;
```

```
import br.edu.ifsp.pw3.api.endereco.DadosEndereco;  
import jakarta.validation.Valid;  
import jakarta.validation.constraints.Email;  
import jakarta.validation.constraints.NotBlank;  
import jakarta.validation.constraints.NotNull;  
import jakarta.validation.constraints.Pattern;
```

```
public record DadosCadastroMedico(  
    @NotBlank  
    String nome,  
    @NotBlank  
    @Email  
    String email,  
    @NotBlank  
    @Pattern(regexp = "\\d{4,6}") // 4 a 6 dígitos  
    String crm,  
    @NotNull  
    Especialidade especialidade,  
    @NotNull  
    @Valid  
    DadosEndereco endereco) {  
}
```

```
public record DadosCadastroMedico(  
    @NotBlank  
    String nome,  
    @NotBlank  
    @Email  
    String email,  
    @NotBlank  
    @Pattern(regexp = "\\d{4,6}") // 4 a 6 dígitos  
    String crm,  
    @NotNull  
    Especialidade especialidade,  
    @NotNull  
    @Valid  
    DadosEndereco endereco) {  
}
```

@NotBlank

String

@NotBlank (jakarta.validation.constraints)

@NotBlank (org.hibernate.validator.constraints) Maven

@Email

String email,

As strings no enum  
já são validadas!

Para usar a validação que será configurada  
lá no record DadosEndereco

E no record do endereço...



```
package br.edu.ifsp.pw3.api.endereco;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;

public record DadosEndereco(

    @NotBlank
    String logradouro,

    @NotBlank
    String bairro,

    @NotBlank
    @Pattern(regexp = "\\d{8}")
    String cep,

    @NotBlank
    String cidade,

    @NotBlank
    String uf,

    String complemento, // opcional

    String numero) {    // opcional

}
```

Precisa configurar no controller também:

@Valid





```
@RestController
@RequestMapping("medicos")
public class MedicoController {

    @Autowired
    private MedicoRepository repository;

    @PostMapping
    @Transactional
    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados){

        repository.save( new Medico(dados) );

    }

}
```



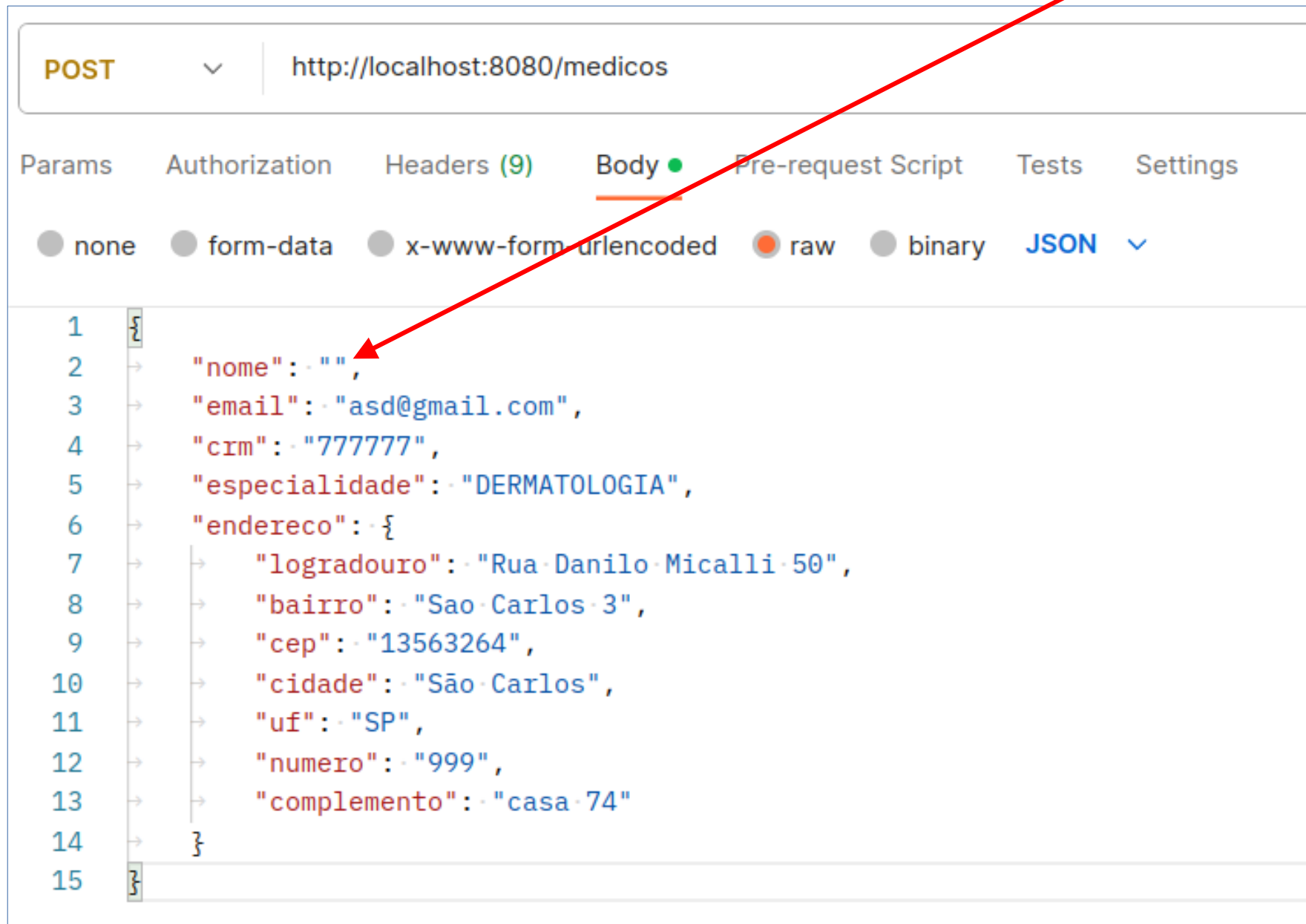
# Testando...



# JSON Completo:

```
{  
  "nome": "Asdrubal Zoroastro",  
  "email": "asd@gmail.com",  
  "crm": "777777",  
  "especialidade": "DERMATOLOGIA",  
  "endereco": {  
    "logradouro": "Rua Danilo Micalli 50",  
    "bairro": "Sao Carlos 3",  
    "cep": "13563264",  
    "cidade": "São Carlos",  
    "uf": "SP",  
    "numero": "999",  
    "complemento": "casa 74"  
  }  
}
```

# Vou deixar o nome vazio:



Body Cookies Headers (4) Test Results

Status: 400 Bad Request

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "timestamp": "2024-10-04T19:38:39.140+00:00",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "path": "/medicos"  
6 }
```

## No log de execução:

2024-10-04T16:38:39.127-03:00 WARN 8908 --- [api\_2024\_2] [nio-8080-exec-1] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved

[org.springframework.web.bind.MethodArgumentNotValidException: **Validation failed** for argument [0] in public void

br.edu.ifsp.prw3.api\_2024\_2.controller.MedicoController.cadastrar(br.edu.ifsp.prw3.api\_2024\_2.medico.DadosCadastroMedico): **[Field error in object 'dadosCadastroMedico' on field 'nome': rejected value []]**; codes

[NotBlank.dadosCadastroMedico.nome,NotBlank.nome,NotBlank.java.lang.String,NotBlank]; arguments [org.springframework.context.support.DefaultMessageSourceResolvable: codes [dadosCadastroMedico.nome,nome]; arguments []; default message [nome]]; default message **[must not be blank]**] ]

Outro teste, email mal formatado:

POST

http://localhost:8080/medicos

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

JSON



```
1 {  
2   "nome": "Asdrubal Zoroastro",  
3   "email": "asd@gmail.com",  
4   "crm": "777777",  
5   "especialidade": "DERMATOLOGIA",  
6   "endereco": {  
7     "logradouro": "Rua Danilo Micalli 50",  
8     "bairro": "Sao Carlos 3",  
9     "cep": "13563264",  
10    "cidade": "São Carlos",  
11    "uf": "SP",  
12    "numero": "999",  
13    "complemento": "casa 74"  
14  }  
15 }
```

No log de execução:

```
2024-10-04T16:42:44.536-03:00 WARN 8908 --- [api_2024_2] [nio-8080-
exec-3] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved
[org.springframework.web.bind.MethodArgumentNotValidException:
Validation failed for argument [0] in public void
br.edu.ifsp.prw3.api_2024_2.controller.MedicoController.cadastrar(br.edu.ifs
p.prw3.api_2024_2.medico.DadosCadastroMedico): [Field error in object
'dadosCadastroMedico' on field 'email': rejected value [asd.gmail.com];
codes
[Email.dadosCadastroMedico.email,Email.email,Email.java.lang.String,Email
]; arguments
[org.springframework.context.support.DefaultMessageSourceResolvable:
codes [dadosCadastroMedico.email,email]; arguments []; default message
[email],[Ljakarta.validation.constraints.Pattern$Flag;@db7d955,*]; default
message [must be a well-formed email address]] ]
```



# Tirando os campos opcionais (complemento e número)



POST

http://localhost:8080/medicos

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON

```
1 {  
2   "nome": "Asdrubal Zoroastro",  
3   "email": "asd@gmail.com",  
4   "crm": "777777",  
5   "especialidade": "DERMATOLOGIA",  
6   "endereco": {  
7     "logradouro": "Rua Danilo Micalli 50",  
8     "bairro": "Sao Carlos 3",  
9     "cep": "13563264",  
10    "cidade": "São Carlos",  
11    "uf": "SP",  
12  }  
13 }
```

Não esquece de  
tirar a vírgula

🌐 Status: 200 OK Time: 41 ms Size: 123 B

SELECT \* FROM MEDICOS;

ID	NOME	EMAIL	CRM	ESPECIALIDADE	LOGRADOURO	BAIRRO	CEP	COMPLEMENTO	NUMERO	UF	CIDADE
1	Asdrubal Jose	asd.jos@ifsp.med.br	222222	ORTOPEDIA	rua blabla	santa felicia	13563264	null	50	SP	São Carlos
2	Asdrubal Zoroastro	asd@gmail.com	777777	DERMATOLOGIA	Rua Danilo Micalli 50	Sao Carlos 3	13563264	null	null	SP	São Carlos


(2 rows, 2 ms)

Edit

# Alterando o BD



# Ops.... esquecemos um campo!!!!

 **Cadastro de médicos**

in list [DOING](#)

Labels

Médico

+

☰

**Description**

Edit


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

- **Nome**
- **E-mail**
- **Telefone**
- **CRM**
- **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.




# Ops.... esquecemos um campo!!!!

 **Cadastro de médicos**  
in list [DOING](#)

Labels

● Médico

+

 **Description** Edit

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

- Nome
- E-mail
- Telefone
- CRM
- Especialidade (*Ortopedia, Cardiologia, Ginecologia ou Dermatologia*)
- Endereço completo (logradouro, número, complemento, bairro, cidade, UF e CEP)

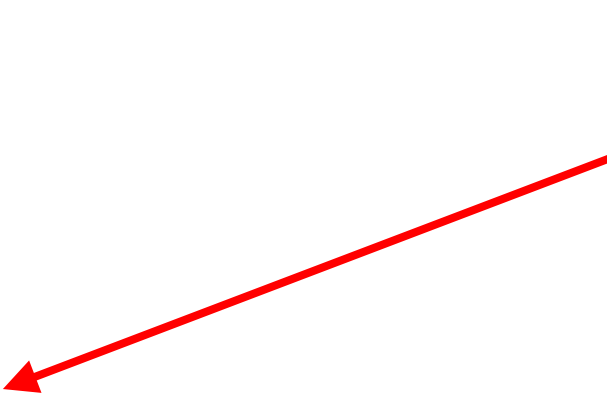
Todas as informações são de preenchimento **obrigatório**, exceto o número e o complemento do endereço.

E agora??? 🤨

**"Vamos por partes."  
Já dizia Jack o Estripador.**

# Inserir no DTO do médico:

```
public record DadosCadastroMedico(  
    @NotNull  
    @NotBlank  
    String nome,  
  
    @NotBlank  
    @Email  
    String email,  
  
    @NotBlank  
    String telefone,  
  
    @NotBlank  
    @Pattern(regexp = "\\d{4,6}") // 4 a 6 dígitos  
    String crm,  
  
    @NotNull  
    Especialidade especialidade,  
  
    @NotNull  
    @Valid  
    DadosEndereco endereco) {  
}
```





# Inserir na entidade (Classe) Medico:

Alterar também o construtor que recebe o DTO com o telefone, transferindo o valor para o atributo.

```
@Table(name = "medicos")
@Entity(name = "Medico")
@Getter
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(of = "id")
public class Medico {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private String email;
    private String telefone;
    private String crm;
    @Enumerated(EnumType.STRING)
    private Especialidade especialidade;
    @Embedded
    private Endereco endereco;

    public Medico(DadosCadastroMedico dados) {
        this.nome = dados.nome();
        this.email = dados.email();
        this.telefone = dados.telefone();
        this.crm = dados.crm();
        this.especialidade = dados.especialidade();
        this.endereco = new Endereco(dados.endereco());
    }
}
```

E a tabela no Banco de Dados?

Altero a migration que fizemos? (V1)

E a tabela no Banco de Dados?

Altero a migration que fizemos? (V1)



**Migrations executadas  
não podem ser modificadas !**

Vamos criar uma nova migration.

O banco vai para a versão 2 (V2).

# Boa prática!!

- É uma **boa prática** criar uma nova migration (versão 2) para alterar a tabela em vez de modificar o arquivo da migration V1 por várias razões:
  - **Rastreabilidade e Histórico:** Ao criar uma nova migration para cada alteração no esquema do banco de dados, você mantém um histórico claro e rastreável de todas as alterações que foram aplicadas ao longo do tempo. Isso é fundamental para entender como o banco de dados evoluiu e para reverter ou reaplicar alterações específicas, se necessário.
  - **Colaboração e Controle de Versão:** Se você estiver trabalhando em equipe ou usando um sistema de controle de versão como o Git, ter migrations separadas permite que várias pessoas trabalhem em diferentes partes do banco de dados ao mesmo tempo, sem conflitos no arquivo de migração.
  - **Consistência em Ambientes:** Migrar para um ambiente de desenvolvimento, teste e produção é mais fácil quando você tem migrations individuais. Você pode aplicar as migrations em ordem, garantindo que todos os ambientes tenham o mesmo esquema.
  - **Reversão Segura:** Se você precisar reverter uma alteração específica, é mais fácil fazê-lo quando essa alteração está contida em uma única migration. Você pode simplesmente reverter ou reaplicar essa migration em vez de tentar desfazer alterações em um arquivo de migração mais longo.
  - **Evitar Problemas de Conflito:** Modificar uma migration existente pode causar problemas de conflito quando outros membros da equipe também estão trabalhando no mesmo arquivo. Isso pode levar a situações complicadas de mesclagem (merge) em sistemas de controle de versão.
- Portanto, é uma prática recomendada criar uma nova migration (versão 2) para adicionar ou modificar campos na tabela, em vez de modificar diretamente a migration V1. Cada migration deve representar uma única alteração no esquema do banco de dados para manter o controle e a ordem das alterações de forma organizada e rastreável.

É claro que em um projeto simples como este, poderia apagar o banco, alterar a migration, etc...

Mas a idéia do flyway não é essa!

É você ir desenvolvendo seu aplicativo e seu banco, gerando versões dele, e ele se atualiza automaticamente.

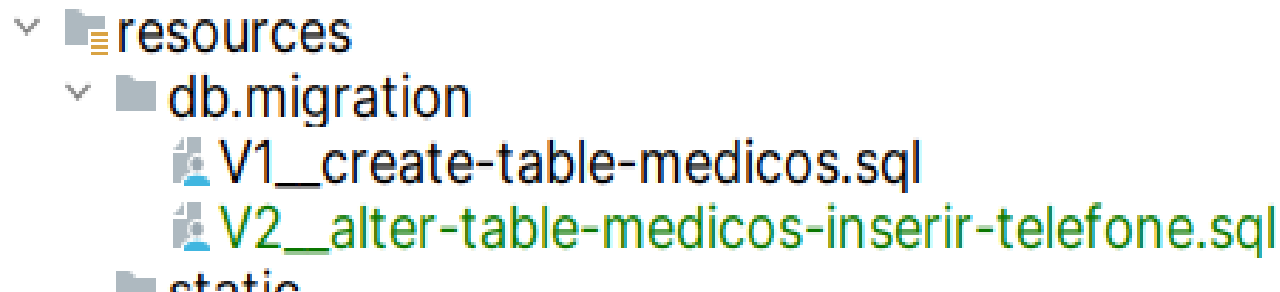
Lembre-se:  
Se vai mexer em migrations,  
**PARE** a execução da aplicação!

(se instalou aquela biblioteca  
que reinicia a aplicação  
toda vez que altera e salva o código).

Vamos criar o arquivo

V2 \_ \_ alter-table-medicos-inserir-telefone.sql

V2\_\_alter-table-medicos-inserir-telefone.sql



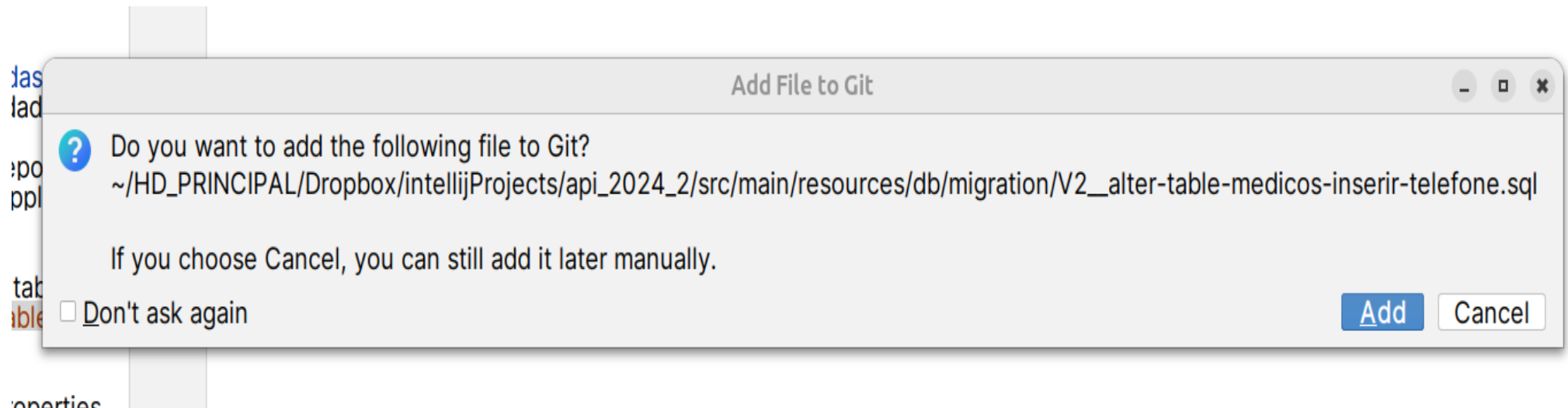
```
resources
└─ db.migration
   ├── V1_create-table-medicos.sql
   └── V2__alter-table-medicos-inserir-telefone.sql
```





# OBS!!

## Como estou usando GIT, apareceu:



D\_PRINCIPAL/Dropbox/intelliProjects/api

1 ;

2 alter table medicos add telefone VARCHAR(20);

3

4

ain

java

resources

db.migration

V1\_\_create-table-medicos.sql

V2\_\_alter-table-medicos-inserir-telefone.sql

**alter table medicos add telefone VARCHAR(20);**

Executando novamente...

OBS: Obviamente, não precisa apagar o arquivo do banco do H2!!!

Vai ser alterado corretamente.



```
: Database: jdbc:h2:file:./DATA/pw3api (H2 2.2)
: Flyway upgrade recommended: H2 2.2.224 is newer than this version of Flyway and
: Successfully validated 2 migrations (execution time 00:00.016s)
: Current version of schema "PUBLIC": 1
: Migrating schema "PUBLIC" to version "2 - alter-table-medicos-inserir-telefone"
: Successfully applied 1 migration to schema "PUBLIC", now at version v2 (executio
```

```
..... - . - . . . . . - . - . .
```

🔗 | 🔑 | ☒ Auto commit 🔑

🗄️ jdbc:h2:file:/home/carla02005/pw

- [-] 📄 MEDICOS
  - + 📄 ID
  - + 📄 NOME
  - + 📄 EMAIL
  - + 📄 CRM
  - + 📄 ESPECIALIDADE
  - + 📄 LOGRADOURO
  - + 📄 BAIRRO
  - + 📄 CEP
  - + 📄 COMPLEMENTO
  - + 📄 NUMERO
  - + 📄 UF
  - + 📄 CIDADE
  - + 📄 TELEFONE
  - + 📄 Indexes
- + 📄 flyway\_schema\_history
- + 📁 INFORMATION\_SCHEMA

UF	CIDADE	TELEFONE
SP	São Carlos	<i>null</i>
SP	São Carlos	<i>null</i>

Nova requisição POST

```
{  
  "nome": "Lupita Zoroastro",  
  "email": "lupy@gmail.com",  
  "telefone": "16992487777",  
  "crm": "666666",  
  "especialidade": "DERMATOLOGIA",  
  "endereco": {  
    "logradouro": "Rua Danilo Micalli 50",  
    "bairro": "Sao Carlos 3",  
    "cep": "13563264",  
    "cidade": "São Carlos",  
    "uf": "SP",  
    "numero": "888",  
    "complemento": "casa 99"  
  }  
}
```



Run Run Selected Auto complete Clear SQL statement:

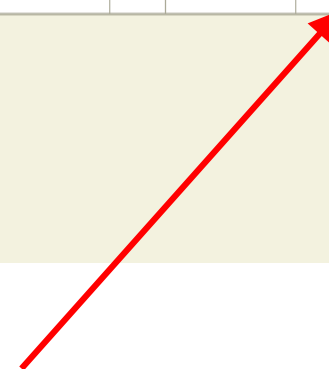
SELECT \* FROM MEDICOS

SELECT \* FROM MEDICOS;

ID	NOME	EMAIL	CRM	ESPECIALIDADE	LOGRADOURO	BAIRRO	CEP	COMPLEMENTO	NUMERO	UF	CIDADE	TELEFONE
1	Asdrubal Jose	asd.jos@ifsp.med.br	222222	ORTOPEDIA	rua blabla	santa felicia	13563264	null	50	SP	São Carlos	null
2	Asdrubal Zoroastro	asd@gmail.com	777777	DERMATOLOGIA	Rua Danilo Micalli 50	Sao Carlos 3	13563264	null	null	SP	São Carlos	null
3	Lupita Zoroastro	lupy@gmail.com	666666	DERMATOLOGIA	Rua Danilo Micalli 50	Sao Carlos 3	13563264	casa 99	999	SP	São Carlos	16992487777

(3 rows, 2 ms)

Edit





# OBS:

Cuidado ao inserir vários médicos,  
para testar.

Precisa mudar pelo menos o email e crm!

```
create table medicos(  
  
    id bigint not null auto_increment,  
    nome varchar(100) not null,  
    email varchar(100) not null unique,  
    crm varchar(6) not null unique,  
    especialidade varchar(100) not null,  
    logradouro varchar(100) not null,  
    bairro varchar(100) not null,  
    cep varchar(9) not null,  
    complemento varchar(100),  
    numero varchar(20),  
    uf char(2) not null,  
    cidade varchar(100) not null,  
  
    primary key(id)  
  
);
```

Apaguei o banco.

Dados para os próximos exemplos:

```
{
  "nome": "Asdrubal Almeida",
  "email": "asd@gmail.com",
  "telefone": "16992481111",
  "crm": "111111",
  "especialidade": "ORTOPEDIA",
  "endereco": {
    "logradouro": "RUA UM",
    "bairro": "BAIRRO UM",
    "cep": "11111111",
    "cidade": "Araraquara",
    "uf": "SP"
  }
}

{
  "nome": "Lupita Almeida",
  "email": "lupy@gmail.com",
  "telefone": "16992482222",
  "crm": "222222",
  "especialidade": "DERMATOLOGIA",
  "endereco": {
    "logradouro": "RUA DOIS",
    "bairro": "BAIRRO DOIS",
    "cep": "22222222",
    "cidade": "Ibaté",
    "uf": "SP"
  }
}
```

```
{
  "nome": "Zoroastro Almeida",
  "email": "zorro@gmail.com",
  "telefone": "16992483333",
  "crm": "333333",
  "especialidade": "CARDIOLOGIA",
  "endereco": {
    "logradouro": "RUA TRES",
    "bairro": "BAIRRO TRES",
    "cep": "33333333",
    "cidade": "São Carlos",
    "uf": "SP"
  }
}
```

[Run](#)[Run Selected](#)[Auto complete](#)[Clear](#)

SQL statement:

```
select * from medicos;
```

```
select * from medicos;
```

ID	NOME	EMAIL	CRM	ESPECIALIDADE	LOGRADOURO	BAIRRO	CEP	COMPLEMENTO	NUMERO	UF	CIDADE	TELEFONE
1	Asdrubal Almeida	asd@gmail.com	111111	ORTOPEDIA	RUA UM	BAIRRO UM	11111111	<i>null</i>	<i>null</i>	SP	Araraquara	16992481111
2	Lupita Almeida	lupy@gmail.com	222222	DERMATOLOGIA	RUA DOIS	BAIRRO DOIS	22222222	<i>null</i>	<i>null</i>	SP	Ibaté	16992482222
3	Zoroastro Almeida	zorro@gmail.com	333333	CARDIOLOGIA	RUA TRES	BAIRRO TRES	33333333	<i>null</i>	<i>null</i>	SP	São Carlos	16992483333

(3 rows, 9 ms)

# Listagem de Medicos

## Método GET

# MedicoController

```
@RestController
@RequestMapping("medicos")
public class MedicoController {

    @Autowired
    private MedicoRepository repository;

    @PostMapping
    @Transactional
    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados){

        repository.save( new Medico(dados) );

    }

    @GetMapping
    public List<Medico> listar() {

        return repository.findAll();

    }

}
```



# MedicoController



```
@RestController
@RequestMapping("medicos")
public class MedicoController {

    @Autowired
    private MedicoRepository repository;

    @PostMapping
    @Transactional
    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados){

        repository.save( new Medico(dados) );

    }
```

```
    @GetMapping
    public List<Medico> listar() {

        return repository.findAll();

    }
```

```
}
```

O método `findAll()` é um dos métodos de consulta fornecidos pelo Spring Data JPA. Ele é usado para recuperar todos os registros de uma entidade específica no banco de dados.



http://localhost:8080/medicos

GET

http://localhost:8080/medicos



http://localhost:8080/medicos

GET

http://localhost:8080/medicos

Params Authorization Headers (9) Body ●

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

JSON

```
1 [
2   {
3     "id": 1,
4     "nome": "Asdrubal Almeida",
5     "email": "asd@gmail.com",
6     "telefone": "16992481111",
7     "crm": "111111",
8     "especialidade": "ORTOPEDIA",
9     "endereco": {
10      "logradouro": "RUA UM",
11      "bairro": "BAIRRO UM",
12      "cep": "11111111",
13      "numero": null,
14      "complemento": null,
15      "cidade": "Araraquara",
16      "uf": "SP"
17    }
18  },
```

```
19 {
20   "id": 2,
21   "nome": "Lupita Almeida",
22   "email": "lupy@gmail.com",
23   "telefone": "16992482222",
24   "crm": "222222",
25   "especialidade": "DERMATOLOGIA",
26   "endereco": {
27     "logradouro": "RUA DOIS",
28     "bairro": "BAIRRO DOIS",
29     "cep": "22222222",
30     "numero": null,
31     "complemento": null,
32     "cidade": "Ibaté",
33     "uf": "SP"
34   }
35 },
```

```
41 {
42   "id": 3,
43   "nome": "Zoroastro Almeida",
44   "email": "zorro@gmail.com",
45   "telefone": "16992483333",
46   "crm": "333333",
47   "especialidade": "CARDIOLOGIA",
48   "endereco": {
49     "logradouro": "RUA TRES",
50     "bairro": "BAIRRO TRES",
51     "cep": "33333333",
52     "numero": null,
53     "complemento": null,
54     "cidade": "São Carlos",
55     "uf": "SP"
56   }
57 },
```





http://localhost:8080/medicos

GET

http://localhost:8080/medicos

**Está retornando  
Todos os atributos  
da Classe Medico**



http://localhost:8080/medicos

GET

http://localhost:8080/medicos

Params Authorization Headers (9) Body ●

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

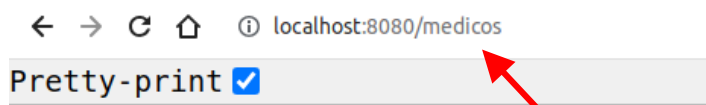
Visualize

JSON

```
1 [
2   {
3     "id": 1,
4     "nome": "Asdrubal Almeida",
5     "email": "asd@gmail.com",
6     "telefone": "16992481111",
7     "crm": "111111",
8     "especialidade": "ORTOPEDIA",
9     "endereco": {
10      "logradouro": "RUA UM",
11      "bairro": "BAIRRO UM",
12      "cep": "11111111",
13      "numero": null,
14      "complemento": null,
15      "cidade": "Araraquara",
16      "uf": "SP"
17    }
18  },
```

```
19 {
20   "id": 2,
21   "nome": "Lupita Almeida",
22   "email": "lupy@gmail.com",
23   "telefone": "16992482222",
24   "crm": "222222",
25   "especialidade": "DERMATOLOGIA",
26   "endereco": {
27     "logradouro": "RUA DOIS",
28     "bairro": "BAIRRO DOIS",
29     "cep": "22222222",
30     "numero": null,
31     "complemento": null,
32     "cidade": "Ibaté",
33     "uf": "SP"
34   }
35 },
```

```
41 {
42   "id": 3,
43   "nome": "Zoroastro Almeida",
44   "email": "zorro@gmail.com",
45   "telefone": "16992483333",
46   "crm": "333333",
47   "especialidade": "CARDIOLOGIA",
48   "endereco": {
49     "logradouro": "RUA TRES",
50     "bairro": "BAIRRO TRES",
51     "cep": "33333333",
52     "numero": null,
53     "complemento": null,
54     "cidade": "São Carlos",
55     "uf": "SP"
56   }
57 },
```



```
[
  {
    "id": 1,
    "nome": "Asdrubal Almeida",
    "email": "asd@gmail.com",
    "telefone": "16992481111",
    "crm": "111111",
    "especialidade": "ORTOPEDIA",
    "endereco": {
      "logradouro": "RUA UM",
      "bairro": "BAIRRO UM",
      "cep": "11111111",
      "numero": null,
      "complemento": null,
      "cidade": "Araraquara",
      "uf": "SP"
    }
  },
  {
    "id": 2,
    "nome": "Lupita Almeida",
    "email": "lupy@gmail.com",
    "telefone": "16992482222",
    "crm": "222222",
    "especialidade": "DERMATOLOGIA",
    "endereco": {
      "logradouro": "RUA DOIS",
      "bairro": "BAIRRO DOIS",
      "cep": "22222222",
      "numero": null,
      "complemento": null,
      "cidade": "Ibat\u00E9",
      "uf": "SP"
    }
  },
  {
    "id": 3,
    "nome": "Zoroastro Almeida",
    "email": "zorro@gmail.com",
    "telefone": "16992483333",
```

**Direto no browser...**

Front-end Python...

```
import requests

headers = {'Accept' : 'application/json'}

r = requests.get('http://localhost:8080/medicos', headers=headers)

lista = r.json()

# 'lista' é lista de dicionários:

for d in lista:

    print("ID:", d['id'])
    print("Nome:", d['nome'])
    print("Email:", d['email'], " ", "Telefone:", d['telefone'])
    print("CRM:", d['crm'], "-", "Especialidade:", d['especialidade'])
    print("Endereço:", d['endereco']['logradouro'])
    print(d['endereco']['bairro'], "- CEP:", d['endereco']['cep'])
    print(d['endereco']['cidade'], "/", d['endereco']['uf'])
    print()

##fimFOR
```

===== RESTART: /home/carlao2005/pw3.py

ID: 1

Nome: Asdrubal Almeida

Email: asd@gmail.com      Telefone: 16992481111

CRM: 111111 - Especialidade: ORTOPEDIA

Endereço: RUA UM

BAIRRO UM - CEP: 11111111

Araraquara / SP

ID: 2

Nome: Lupita Almeida

Email: lupy@gmail.com      Telefone: 16992482222

CRM: 222222 - Especialidade: DERMATOLOGIA

Endereço: RUA DOIS

BAIRRO DOIS - CEP: 22222222

Ibaté / SP

ID: 3

Nome: Zoroastro Almeida

Email: zorro@gmail.com      Telefone: 16992483333

CRM: 333333 - Especialidade: CARDIOLOGIA

Endereço: RUA TRES

BAIRRO TRES - CEP: 33333333

São Carlos / SP

E se não quisermos todos os atributos ?

Só queremos:


nome

email

crm

especialidade

Vamos criar um novo método  
na classe controller,  
para devolver apenas alguns dados.

O nome não deve ser muito "REST"..   
Mas pra exemplo de aula tá bom...

# MedicoController

```
@GetMapping
public List<Medico> listar() {

    return repository.findAll();

}

@GetMapping("algunsdados")
public List< ? > listarAlgunsDados() {

    return repository.findAll();

}

}
```





Precisamos criar um novo DTO (Record)  
apenas com os dados que queremos.

# DTO (Record): DadosListagemMedico

```
package br.edu.ifsp.pw3.api.medico;  
  
public record DadosListagemMedico(String nome, String email,  
                                   String crm, Especialidade especialidade) {  
}
```



## Voltando no controller...

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

O DTO que  
acabamos  
de criar



erro!!



# Voltando no controller...

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

return repository.findAll();

Required type: List <DadosListagemMedico>

Provided: List <Medico>

Make 'listarAlgunsDados()' return 'java.util.List<br.edu.ifsp.pw3.api.medico.Medico>' Alt+Shift+Enter More actions... Alt+Enter

org.springframework.data.repository.ListCrudRepository<T, ID>

public abstract java.util.List<T> findAll()

Overrides: findAll in interface CrudRepository

Maven: org.springframework.data:spring-data-commons:3.1.3 (spring-data-commons-3.1.3.jar)

CA

estou usando framework spring. a variável repository é um JpaRepository<Medico, Long>. Por que o código abaixo está dando erro ?

```
@GetMapping
@RequestMapping("algunsdados")
public List<DadosListagemMedico> listarAlgunsDados() {

    return repository.findAll();

}
```

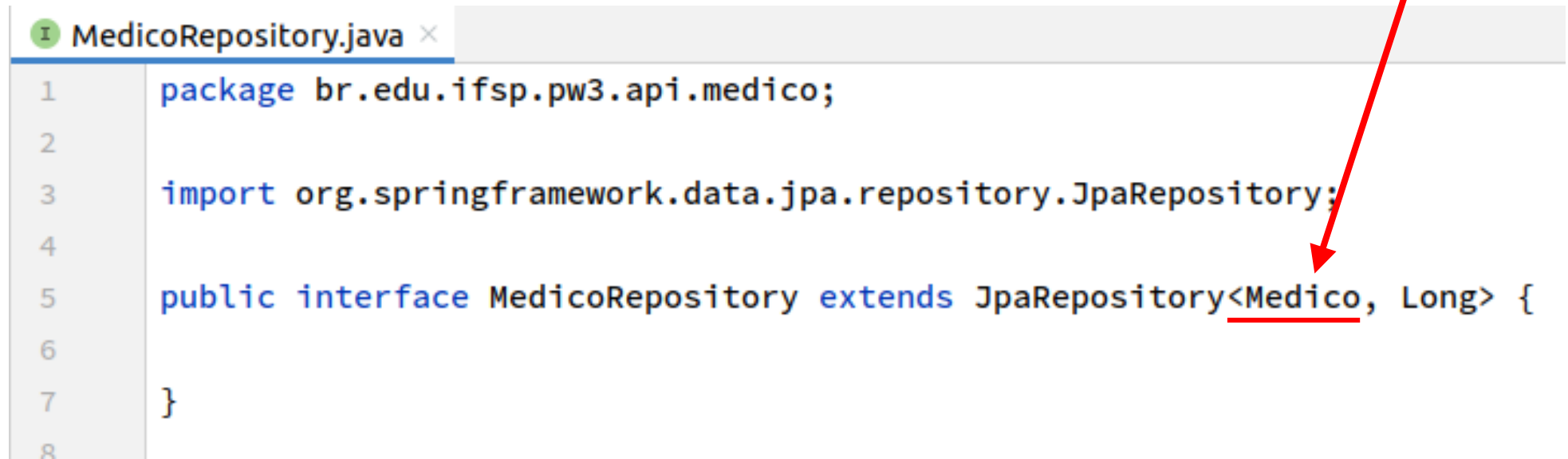


O erro que você está enfrentando provavelmente ocorre porque o método `findAll()` do `JpaRepository` retorna uma lista de objetos do tipo `Medico`, mas o método `listarAlgunsDados()` está declarado para retornar uma lista de `DadosListagemMedico`. Isso causa uma incompatibilidade de tipos.



Se você deseja retornar uma lista de `DadosListagemMedico`, precisará realizar uma conversão dos objetos `Medico` para `DadosListagemMedico`. Isso geralmente é feito através do uso de mapeamento ou transformação dos dados. Você pode fazer isso usando o Stream API do Java ou qualquer outra técnica de mapeamento de objetos que preferir.

# Lembrando nosso repository de Medicos:



```
MedicoRepository.java x
1 package br.edu.ifsp.pw3.api.medico;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5 public interface MedicoRepository extends JpaRepository<Medico, Long> {
6
7 }
8
```

O repository que fizemos, trabalha com a entidade Medico.

O método `findAll()` do repository, vai devolver objetos “Medico”.

Mas agora não queremos retornar objetos Medico,  
mas sim objetos `DadosListagemMedico`.

Precisaremos converter objetos Medico em `DadosListagemMedico`.

Comando para converter  
uma lista de objetos **Medico**,  
em uma lista de objetos **DadosListagemMedico**:

# MedicoController:

```
@RestController
@RequestMapping("medicos")
public class MedicoController {

    @Autowired
    private MedicoRepository repository;

    @PostMapping
    @Transactional
    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados){

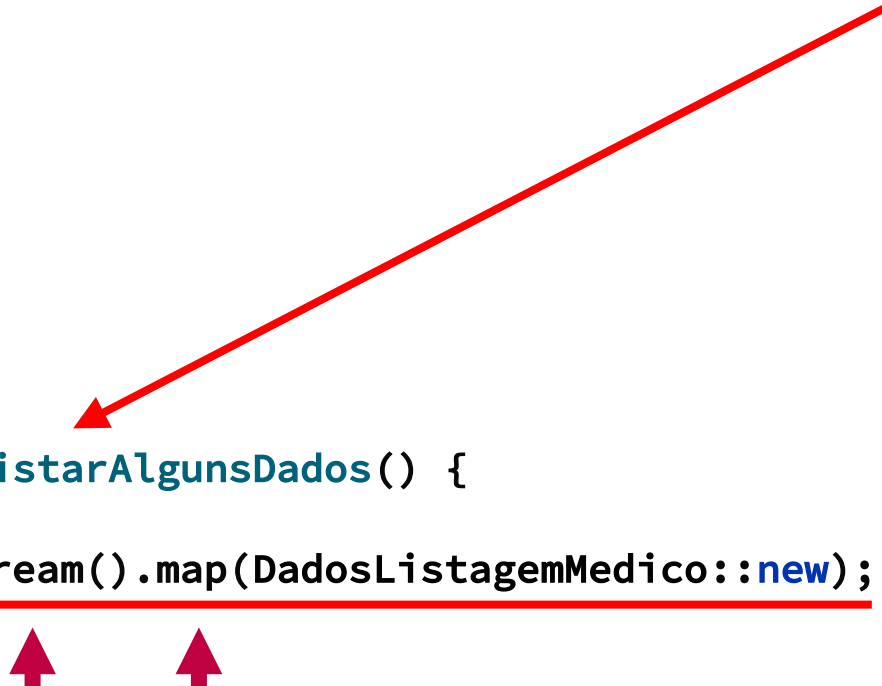
        repository.save( new Medico(dados) );
    }

    @GetMapping
    public List<Medico> listar() {

        return repository.findAll();
    }

    @GetMapping("algunsdados")
    public List<DadosListagemMedico> listarAlgunsDados() {

        return repository.findAll().stream().map(DadosListagemMedico::new);
    }
}
```



The diagram illustrates the flow of data and annotations in the MedicoController. A red arrow points from the `@GetMapping("algunsdados")` annotation to the `listarAlgunsDados()` method. Two red arrows point upwards from the bottom towards the `stream()` and `map()` methods in the `listarAlgunsDados()` method, highlighting the stream processing logic.



# MedicoController:

```
@RestController
@RequestMapping("medicos")
public class MedicoController {

    @Autowired
    private MedicoRepository repository;

    @PostMapping
    @Transactional
    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados){

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

    @GetMapping
    public List<Medico> listar() {

        return repository.findAll();
    }

    @GetMapping("algunsdados")
    public List<DadosListagemMedico> listarAlgunsDados() {

        return repository.findAll().stream().map(DadosListagemMedico::new);
    }
}
```





Os métodos `stream()` e `map()` fazem parte da API Java Stream, que foi introduzida a partir do Java 8 como parte do pacote `java.util.stream`. Essa API é usada para realizar operações de processamento de coleções de forma mais concisa e funcional, permitindo que você execute operações como filtragem, mapeamento, redução e outras em coleções de dados, como listas ou conjuntos, de maneira mais eficiente e expressiva. Ela é frequentemente usada em combinação com lambdas para escrever código mais legível e conciso ao trabalhar com coleções em Java.



O método `stream()` da API Java Stream cria um "fluxo" (stream) de elementos a partir de uma coleção, como uma lista ou conjunto. Esse fluxo permite que você realize várias operações de processamento de dados, como filtragem, mapeamento e redução, de forma eficiente e funcional. É uma maneira poderosa de manipular e transformar coleções de dados em Java de maneira concisa e expressiva.



O método `map()` da API Java Stream é usado para transformar (mapear) cada elemento de um fluxo (stream) em um novo elemento, aplicando uma função especificada a cada elemento do fluxo. Ele cria um novo fluxo contendo os resultados dessas transformações. Em resumo, o `map()` permite converter cada elemento do fluxo original em outro elemento com base em uma função de mapeamento.

Estou trabalhando com framework spring. tenho um JpaRepository na variável repository. Esse JpaRepository está programado para trabalhar com objetos da classe Medico. Tenho também um DTO (tipo record do java) chamado DadosListagemMedico. O que o seguinte comando faz?

```
repository.findAll().stream().map(DadosListagemMedico::new);
```



- O comando **repository.findAll().stream().map(DadosListagemMedico::new);** parece estar realizando **uma operação de mapeamento (mapping)** em **uma lista de objetos Medico recuperados do seu repositório JPA (repository)** para **objetos do tipo DadosListagemMedico**. Vou explicar em detalhes como isso funciona:
  - **repository.findAll():** Esta parte do comando chama o método **findAll()** do seu repositório JPA. Isso provavelmente **recupera todos os objetos da classe Medico do banco de dados**, retornando-os como uma lista (ou outra coleção) de objetos.
  - **.stream():** O método **stream()** **é chamado na lista de objetos Medico, convertendo-a em um fluxo de objetos**. Isso **permite** que você realize **operações de fluxo (como mapeamento, filtragem, etc.)** nos elementos da lista.
  - **.map(DadosListagemMedico::new):** Aqui, a função **map()** é usada para transformar cada objeto **Medico** no **fluxo** em um objeto **DadosListagemMedico**. **O DadosListagemMedico::new é uma referência a um construtor de DadosListagemMedico**. Portanto, para cada objeto **Medico** no fluxo, um novo objeto **DadosListagemMedico** é criado usando o construtor correspondente.
- No final dessa operação, você terá um **fluxo de objetos DadosListagemMedico**, que pode ser **coletado em uma lista** ou processado de outras maneiras, dependendo do que você deseja fazer com esses objetos.
- **Basicamente, esse comando está transformando uma lista de objetos Medico em uma lista de objetos DadosListagemMedico usando a função de mapeamento**. Isso pode ser útil, por exemplo, quando você deseja apresentar os dados dos médicos em um formato diferente, como para exibição em uma lista ou tabela.





vou explicar essa sintaxe específica em mais detalhes:

No comando `repository.findAll().stream().map(DadosListagemMedico::new);`, a parte `DadosListagemMedico::new` está usando uma expressão de referência a construtor (constructor reference).

- `DadosListagemMedico` é o nome da classe cujo construtor estamos referenciando.
- `::` é a notação de referência a construtor em Java.
- `new` é a palavra-chave que cria uma nova instância de um objeto.

Isso significa que você está dizendo ao método `map()` para usar o construtor da classe `DadosListagemMedico` para criar novas instâncias de objetos `DadosListagemMedico` para cada elemento do fluxo original. É uma maneira concisa e funcional de criar essas novas instâncias durante o processo de mapeamento.

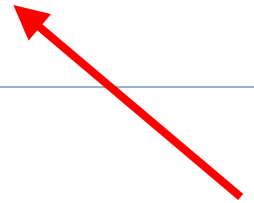
Por exemplo, se você tiver uma lista de objetos `Medico` e desejar criar uma lista correspondente de objetos `DadosListagemMedico` a partir dos objetos `Medico`, você pode usar essa expressão de referência ao construtor para fazer isso de forma eficiente e legível. O construtor `DadosListagemMedico` deve aceitar um objeto `Medico` como argumento para que a correspondência funcione corretamente.

```
@GetMapping("algunsdados")
```

```
public List<DadosListagemMedico> listarAlgunsDados() {
```


```
    return repository.findAll().stream().map(DadosListagemMedico::new);
```

```
}
```



```
@GetMapping("algunsdados")
```

```
public List<DadosListagemMedico> listarAlgunsDados() {  
  
    return repository.findAll().stream().map(DadosListagemMedico::new);  
}
```



CA

ainda falando do comando

```
repository.findAll().stream().map(DadosListagemMedico::new);
```

está dando erro na parte "DadosListagemMedico::new".

Segue a definição de DadosListagemMedico:

```
public record DadosListagemMedico(String nome, String email, String crm, Especialidade  
especialidade) { }
```



O erro que você está enfrentando no trecho `DadosListagemMedico::new` está relacionado ao fato de que um construtor de `DadosListagemMedico` não pode ser encontrado que corresponda à assinatura esperada pelo método `map()`. Isso ocorre porque, por padrão, quando você cria um registro (record) em Java, um construtor é gerado automaticamente para você, mas ele é limitado a inicializar os componentes do registro. Ele não oferece um construtor que possa ser usado para criar um novo objeto `DadosListagemMedico` a partir de um objeto `Medico`.

Para resolver esse problema, você precisará criar um construtor personalizado em `DadosListagemMedico` que aceite um objeto `Medico` e faça a conversão apropriada.

Precisamos dotar nosso DTO  
de um construtor para  
realizar essa conversão!

Isto é, um construtor  
que receba um objeto Médico  
e chame o construtor padrão do Record,  
passando os dados desejados  
do Medico.



# DTO (Record): DadosListagemMedico

```
public record DadosListagemMedico(String nome, String email,  
                                   String crm, Especialidade especialidade) {  
  
    // Construtor que recebe um objeto Medico, pega só os dados necessários,  
    // para criar este objeto DadosListagemMedico.  
  
    // OBS: Você pode criar diversos construtores específicos,  
    //      porém eles TEM QUE chamar o construtor padrão (this).  
  
    public DadosListagemMedico(Medico medico) {  
  
        this( medico.getNome(), medico.getEmail(),  
              medico.getCrm(), medico.getEspecialidade() );  
    }  
}
```



# DTO (Record): DadosListagemMedico

```
public record DadosListagemMedico(String nome, String email,  
                                String crm, Especialidade especialidade) {  
  
    // Construtor que recebe um objeto Medico, pega só os dados necessários,  
    // para criar este objeto DadosListagemMedico.  
  
    // OBS: Você pode criar diversos construtores específicos,  
    //      porém eles TEM QUE chamar o construtor padrão (this).  
  
    public DadosListagemMedico(Medico medico) {  
  
        this( medico.getNome(), medico.getEmail(),  
              medico.getCrm(), medico.getEspecialidade() );  
    }  
}
```

**‘Getters’ gerados automaticamente pelo Lombok, na classe Medico, pela anotação @Getter.**

```
@Table(name = "medicos")  
@Entity(name = "Medico")  
@Getter  
@NoArgsConstructor  
@AllArgsConstructor  
@EqualsAndHashCode(of = "id")  
public class Medico {  
  
    @Id  
    @GeneratedValue(strategy = G  
    private Long id;
```



Voltando ao controller...

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



Tá piorando...



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



**Dissemos na assinatura do método  
que iríamos retornar uma lista...**

**Então precisamos retornar uma lista!!!**

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

Código completo da classe:



# MedicoController

```
@RestController
@RequestMapping("medicos")
public class MedicoController {

    @Autowired
    private MedicoRepository repository;

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

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

    @GetMapping("algunsdados")
    public List<DadosListagemMedico> listarAlgunsDados() {

        return repository.findAll().stream().map(DadosListagemMedico::new)
            .toList();
    }
}
```

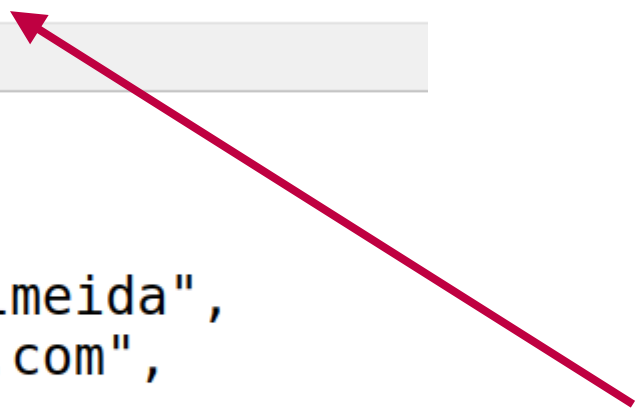
Two red arrows are present in the image. The first arrow points from the top right towards the `@RequestMapping("medicos")` annotation. The second arrow points from the bottom right towards the `@GetMapping("algunsdados")` annotation.



Testando a requisição...

Pretty-print ☒

```
[
  {
    "nome": "Asdrubal Almeida",
    "email": "asd@gmail.com",
    "crm": "111111",
    "especialidade": "ORTOPEDIA"
  },
  {
    "nome": "Lupita Almeida",
    "email": "lupy@gmail.com",
    "crm": "222222",
    "especialidade": "DERMATOLOGIA"
  },
  {
    "nome": "Zoroastro Almeida",
    "email": "zorro@gmail.com",
    "crm": "333333",
    "especialidade": "CARDIOLOGIA"
  }
]
```



GET

localhost:8080/medicos/algunsdados

Params

Authorization

Headers (8)

Body ●

Pre-request Script

Tests

Settings

Body

Cookies

Headers (5)

Test Results



Status: 200 OK

Pretty

Raw

Preview

Visualize

JSON



```
1  [
2    {
3      "nome": "Asdrubal Almeida",
4      "email": "asd@gmail.com",
5      "crm": "111111",
6      "especialidade": "ORTOPEDIA"
7    },
8    {
9      "nome": "Lupita Almeida",
10     "email": "lupy@gmail.com",
11     "crm": "222222",
12     "especialidade": "DERMATOLOGIA"
13   },
14   {
15     "nome": "Zoroastro Almeida",
16     "email": "zorro@gmail.com",
17     "crm": "333333",
18     "especialidade": "CARDIOLOGIA"
19   }
20 ]
```

Vamos inserir mais um médico,  
pra testar a próxima funcionalidade...



```
{  
  "nome": "Anita Garibaldi",  
  "email": "ag@gmail.com",  
  "telefone": "16992484444",  
  "crm": "444444",  
  "especialidade": "ORTOPEDIA",  
  "endereco": {  
    "logradouro": "RUA QUATRO",  
    "bairro": "BAIRRO QUATRO",  
    "cep": "44444444",  
    "cidade": "Limeira",  
    "uf": "SP"  
  }  
}
```



🔗 | 💰 | ☒ Auto commit 🔑 | 📝 | Max rows: 1000 ▼ | ▶️ ▶️ ⏹️ | 📄 | Auto complete Off ▼ Auto select

📁 jdbc:h2:file:./DATA/pw3api

+ 📄 MEDICOS

+ 📄 flyway\_schema\_history

+ 📁 INFORMATION\_SCHEMA

+ 👤 Users

📘 H2 2.3.232 (2024-08-11)

Run Run Selected Auto complete Clear SQL statement:

SELECT \* FROM MEDICOS |

SELECT \* FROM MEDICOS;

ID	NOME	EMAIL	CRM	ESPECIALIDADE	LOGRADOURO
1	Asdrubal Almeida	asd@gmail.com	111111	ORTOPEDIA	RUA UM
2	Lupita Almeida	lupy@gmail.com	222222	DERMATOLOGIA	RUA DOIS
3	Zoroastro Almeida	zorro@gmail.com	333333	CARDIOLOGIA	RUA TRES
4	Anita Garibaldi	ag@gmail.com	444444	ORTOPEDIA	RUA QUATRO

(4 rows, 7 ms)



Estamos retornando todos os registros  
no Banco de Dados.

E se existirem muitos registros ?

Seria bom receber isso por “páginas”.

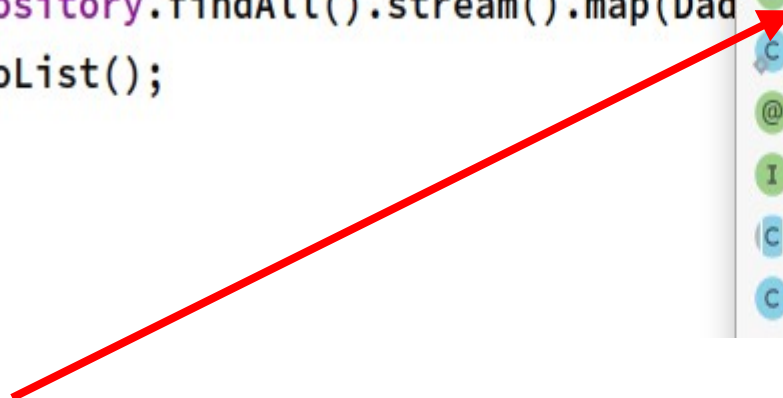
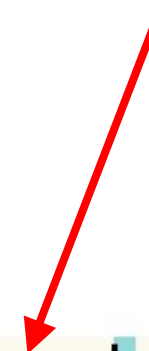


```
@GetMapping("algunsdados")
```

```
public List<DadosListagemMedico> listarAlgunsDados(Pageable) {
```

```
    return repository.findAll().stream().map(Dad  
        .toList();
```

```
}
```



- ~~Pageable~~ java.awt.print
- Pageable org.springframework.data.domain
- Pageable org.springframework.boot.autoconfigure
- @PageableDefault org.springframework.data
- PageableArgumentResolver org.springframework
- PageableExecutionUtils org.springframework
- PageableHandlerMethodArgumentResolver org

**cuidado com o import!!!**





```

@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}

```

```


@GetMapping
@RequestMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados
    Page<T> org.springframework.data.domain
    Page org.hibernate.query
    Pageable org.springframework.data.dom...
    Pageable java.awt.print
    PageImpl<T> org.springframework.data....
    PageRequest org.springframework.data....
    PageAttributes java.awt
    PageFormat java.awt.print

```



```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```



A classe `Pageable` é uma interface do Spring Data que representa informações sobre a paginação de dados em uma consulta. Ela encapsula detalhes como:

- **Número da página** (qual página está sendo requisitada).
- **Tamanho da página** (quantos registros devem ser retornados por página).
- **Ordenação** (se os dados devem ser ordenados e por qual campo).

No seu código, o parâmetro `Pageable paginacao` é automaticamente preenchido pelo Spring quando a requisição HTTP inclui parâmetros como `page`, `size` e `sort`.



```
@GetMapping("algunsdados")  
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {  
    return repository.findAll(paginacao).map(DadosListagemMedico::new);  
}
```

O retorno do método é um objeto do tipo `Page<DadosListagemMedico>`, que faz parte da API do **Spring Data** e representa uma página de resultados paginados.

O que significa `Page<DadosListagemMedico>`?

- `Page<T>` é uma **interface** do Spring Data que encapsula uma lista de elementos do tipo `T` e fornece informações sobre a paginação.
- `DadosListagemMedico` é o tipo dos objetos contidos nessa página, que provavelmente é um **DTO** (Data Transfer Object) usado para expor apenas os dados necessários na resposta.



```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```

removeu o stream()  
(já tem no Pageable)



```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```

removeu o toList()



```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```

### 1. repository.findAll(paginacao)

Este trecho faz a busca paginada no banco de dados.

- `repository`: é o repositório da sua entidade, provavelmente uma interface que estende `JpaRepository<Medico, Long>`.
- `findAll(Pageable paginacao)`: esse método é fornecido pelo `JpaRepository` e retorna um objeto do tipo `Page<Medico>`. Ele executa a consulta no banco trazendo apenas um subconjunto de registros (uma página).
- `paginacao`: um objeto `Pageable`, que contém informações sobre número da página, tamanho da página e ordenação.



```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(Pageable paginacao) {

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```

## 2. `.map(DadosListagemMedico::new)`

Agora, o resultado do `findAll(paginacao)`, que é um `Page<Medico>`, precisa ser convertido para `Page<DadosListagemMedico>`.

- `map(DadosListagemMedico::new)`: aplica a conversão de cada objeto `Medico` para um `DadosListagemMedico` usando o construtor.
- O `Page<T>` tem um método `.map(Function<T, R>)` que transforma os elementos da página em outro tipo de objeto.
- `DadosListagemMedico::new`: é uma referência ao construtor da classe `DadosListagemMedico`, equivalente a:



Do jeito que está já funciona,  
mas ele retorna um objeto interno do Spring,  
e emite um warning,  
dizendo que isso não é seguro,  
pois não é garantido que este objeto vai manter seu  
formato em próximas versões.

Isto é, um front-end criado esperando certas  
informações, pode quebrar no futuro, se essas  
informações mudarem !





```
2025-03-29T19:25:09.114-03:00 WARN 15284 --- [api20251c] [nio-8080-exec-2]
```

```
ration$PageModule$WarningLoggingModifier :
```

Serializing PageImpl instances as-is is not supported, meaning that **there is no guarantee about the stability of the resulting JSON structure!**

**For a stable JSON structure**, please use Spring Data's PagedModel (globally via **@EnableSpringDataWebSupport(pageSerializationMode = VIA DTO)**)

or Spring HATEOAS and Spring Data's PagedResourcesAssembler as documented in ...



Para resolver:

No próprio warning, como vimos,  
ele já sugere uma anotação que  
resolve o problema:



# Anotação necessária na classe principal da aplicação:

**@EnableSpringDataWebSupport**(pageSerializationMode = **EnableSpringDataWebSupport.PageSerializationMode.VIA\_DTO**)

Api20251cApplication.java ×

```
package br.edu.ifsp.prw3.api20251c;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.web.config.EnableSpringDataWebSupport;

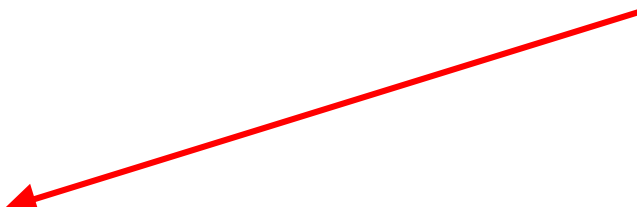
@SpringBootApplication
@EnableSpringDataWebSupport(pageSerializationMode = EnableSpringDataWebSupport.PageSerializationMode.VIA_DTO)
public class Api20251cApplication {

    public static void main(String[] args) {

        SpringApplication.run(Api20251cApplication.class, args);

    }

}
```



# Pra quem gosta de mais informação...

O Spring Data usa a classe `PageImpl` para representar os resultados paginados de consultas a bancos de dados. Quando o Spring Boot serializa esse objeto `PageImpl` para JSON, ele inclui várias informações sobre a paginação, como `totalPages`, `totalElements`, `size`, etc.

O problema é que o formato JSON dessa serialização pode mudar entre versões do Spring. Ou seja, se você estiver consumindo essa API em um frontend, uma mudança na estrutura do JSON pode quebrar sua aplicação.

A mensagem sugere usar `PagedModel` do Spring HATEOAS ou ativar a opção de serialização `VIA_DTO` para garantir que o JSON gerado seja estável.

## Por que usar

```
@EnableSpringDataWebSupport(pageSerializationMode =  
VIA_DTO) ?
```

Essa anotação garante que o Spring converta automaticamente `Page<T>` para um formato mais estável ao ser serializado em JSON.

Com isso:

- A estrutura do JSON gerado será sempre a mesma, mesmo que o Spring mude internamente a implementação de `PageImpl`.
- Seu frontend não será impactado por mudanças futuras no Spring Boot.
- Você evita warnings e possíveis incompatibilidades.

Na requisição, dizer quantos registros por página quer recuperar:

**localhost:8080/medicos/algunsdados?size=2**



```
{
```

```
  "content": [
```

```
    {
```

```
      "nome": "Asdrubal Almeida",
```

```
      "email": "asd@gmail.com",
```

```
      "crm": "111111",
```

```
      "especialidade": "ORTOPEDIA"
```

```
    },
```

```
    {
```

```
      "nome": "Lupita Almeida",
```

```
      "email": "lupy@gmail.com",
```

```
      "crm": "222222",
```

```
      "especialidade": "DERMATOLOGIA"
```

```
    }
```

```
  ],
```

```
  "page": {
```

```
    "size": 2,
```

```
    "number": 0,
```

```
    "totalElements": 4,
```

```
    "totalPages": 2
```

```
  }
```

```
}
```

{

```
"content": [  
  {  
    "nome": "Asdrubal Almeida",  
    "email": "asd@gmail.com",  
    "crm": "111111",  
    "especialidade": "ORTOPEDIA"  
  },  
  {  
    "nome": "Lupita Almeida",  
    "email": "lupy@gmail.com",  
    "crm": "222222",  
    "especialidade": "DERMATOLOGIA"  
  }  
],
```

```
"page": {  
  "size": 2,  
  "number": 0,  
  "totalElements": 4,  
  "totalPages": 2  
}
```

### 1. Estrutura Geral da Resposta

A resposta contém dois principais blocos:

- "content": uma lista com os objetos retornados, ou seja, os médicos.
- "page": metadados sobre a paginação.

}

{

## 2. Campo "content"

Este campo contém os objetos que foram mapeados para o DTO `DadosListagemMedico`. No seu caso, são dois médicos:

```
"content": [  
  {  
    "nome": "Asdrubal Almeida",  
    "email": "asd@gmail.com",  
    "crm": "111111",  
    "especialidade": "ORTOPEDIA"  
  },  
  {  
    "nome": "Lupita Almeida",  
    "email": "lupy@gmail.com",  
    "crm": "222222",  
    "especialidade": "DERMATOLOGIA"  
  }  
],
```

```
"page": {  
  "size": 2,  
  "number": 0,  
  "totalElements": 4,  
  "totalPages": 2  
}
```

}



```
{
```

```
"content": [  
  { ... },  
  { ... }  
],
```

```
"page": {  
  "size": 2,  
  "number": 0,  
  "totalElements": 4,  
  "totalPages": 2  
}
```

```
}
```

### 3. Campo "page"

Este campo contém informações sobre a paginação da resposta:

- `size: 2` → O tamanho da página solicitado na requisição (`?size=2`). Isso indica que cada página contém até **2 registros**.
- `number: 0` → O número da página atual. Como você não especificou `?page=X` na URL, o padrão é a **primeira página (0)**.
- `totalElements: 4` → O total de registros na tabela do banco de dados, ou seja, **há 4 médicos no total**.
- `totalPages: 2` → O número total de páginas necessárias para exibir todos os registros. Como cada página pode ter no máximo 2 registros, o total de 4 médicos resulta em **2 páginas ( $4 \div 2 = 2$ )**.

Se não informar nada, por default  
a página tem 20 elementos.

Como pedir uma página específica?

?size=x&page=x



GET  <http://localhost:8080/medicos/algunsdados?size=2&page=1>

Params  Authorization Headers (8) Body  Pre-request Script Test

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON  

```
2      "content": [  
3          {  
4              "nome": "Zoroastro Almeida",  
5              "email": "zorro@gmail.com",  
6              "crm": "333333",  
7              "especialidade": "CARDIOLOGIA"  
8          },  
9          {  
10             "nome": "Anita Garibaldi",  
11             "email": "ag@gmail.com",  
12             "crm": "444444",  
13             "especialidade": "ORTOPEDIA"  
14         }  
15     ],
```

**Por exemplo,  
pedindo a  
2a página  
(começa em 0!)**

**OBS!!!!!!  
Isto é só um  
exemplo!!!!!!**

Para recuperar os dados em alguma ordem,  
não precisa mexer no código,  
apenas na requisição...

← → ↻ 🏠 ⓘ localhost:8080/medicos/algunsdados?sort=nome

📁 DIVERSOS

Pretty-print ☒

```
{
  "content": [
    {
      "nome": "Anita Garibaldi",
      "email": "ag@gmail.com",
      "crm": "444444",
      "especialidade": "ORTOPEDIA"
    },
    {
      "nome": "Asdrubal Almeida",
      "email": "asd@gmail.com",
      "crm": "111111",
      "especialidade": "ORTOPEDIA"
    },
    {
      "nome": "Lupita Almeida",
      "email": "lupy@gmail.com",
      "crm": "222222",
      "especialidade": "DERMATOLOGIA"
    },
    {
      "nome": "Zoroastro Almeida",
      "email": "zorro@gmail.com",
      "crm": "333333",
      "especialidade": "CARDIOLOGIA"
    }
  ]
}
```

Por padrão, ordem ascendente (asc).

Ordem descendente: desc

← → ↻ 🏠 ⓘ localhost:8080/medicos/algunsdados?sort=nome,desc

📖 booky ⚙️ Limpar ZAP 📞 WhatsApp Web 📺 Best Screen R... 📄 a F

Pretty-print ☒

```
{
  "content": [
    {
      "nome": "Zoroastro Almeida",
      "email": "zorro@gmail.com",
      "crm": "333333",
      "especialidade": "CARDIOLOGIA"
    },
    {
      "nome": "Lupita Almeida",
      "email": "lupy@gmail.com",
      "crm": "222222",
      "especialidade": "DERMATOLOGIA"
    },
    {
      "nome": "Asdrubal Almeida",
      "email": "asd@gmail.com",
      "crm": "111111",
      "especialidade": "ORTOPEDIA"
    },
    {
      "nome": "Anita Garibaldi",
      "email": "ag@gmail.com",
      "crm": "444444",
      "especialidade": "ORTOPEDIA"
    }
  ]
}
```

Combinando as solicitações...



Pretty-print ☒

```
{
  "content": [
    {
      "nome": "Lupita Almeida",
      "email": "lupy@gmail.com",
      "crm": "222222",
      "especialidade": "DERMATOLOGIA"
    },
    {
      "nome": "Asdrubal Almeida",
      "email": "asd@gmail.com",
      "crm": "111111",
      "especialidade": "ORTOPEDIA"
    }
  ],
  "page": {
    "size": 2,
    "number": 1,
    "totalElements": 4,
    "totalPages": 2
  }
}
```

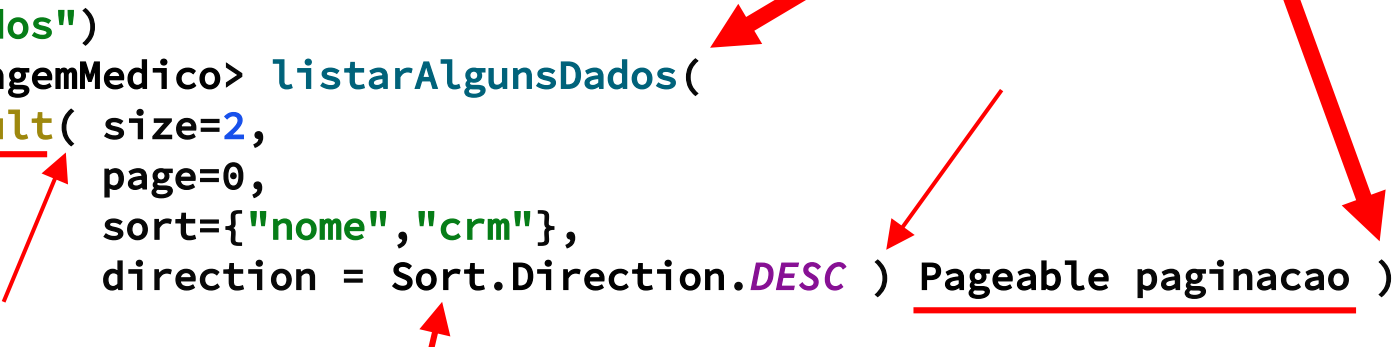
Por default, a página tem 20 elementos,  
**começa na página 0**,  
sem ordenação  
(vem na ordem que está no banco).

Como mudar isso?

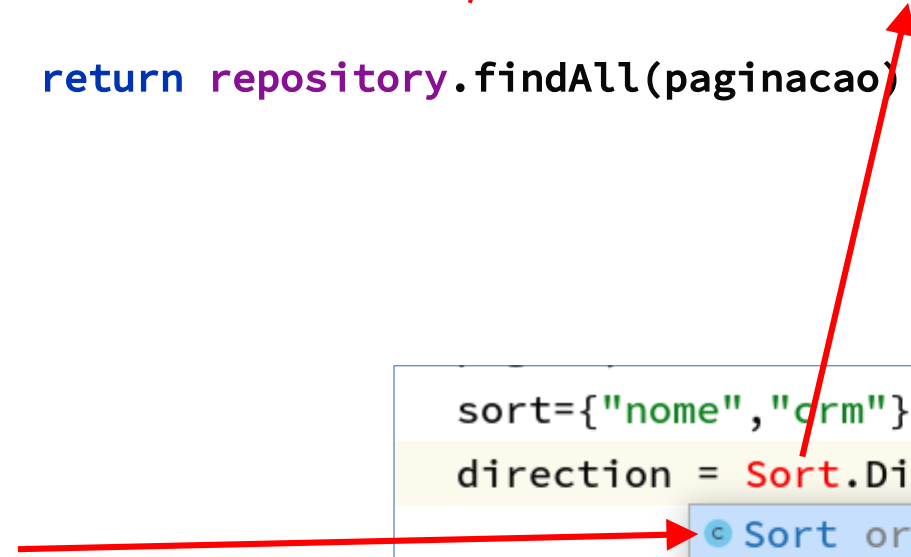
Como definir um “default” diferente ?  
(é apenas um default, isto é,  
se colocar diferente na requisição,  
**vale o da requisição!**)

# MedicoController

```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(
    @PageableDefault( size=2,
        page=0,
        sort={"nome","crm"},
        direction = Sort.Direction.DISC ) Pageable paginacao )
{
    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```



```
sort={"nome","crm"},
direction = Sort.Direction.DISC ) Pageable paginacao )
repository.find
```



- Sort org.springframework.data.domain
- Sort org.springframework.boot.autoconfigure
- Direction.ASC (org.springframework.data.do
- Direction.DISC (org.springframework.data.d
- SortedMap<K, V> java.util
- SortedSet<E> java.util

# MedicoControler

```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(
    @PageableDefault( size=2,
                      page=0,
                      sort={"nome","crm"},
                      direction = Sort.Direction.DISC ) Pageable paginacao )
{

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```

A anotação `@PageableDefault` define valores padrão para a paginação quando o cliente da API não fornece parâmetros explícitos na requisição. Vamos analisar cada um dos atributos utilizados:

1. `size = 2`

- Define o **tamanho padrão da página** como **2 registros por página**.
- Se a requisição não especificar `size`, esse valor será usado.

2. `page = 0`

- Define o **número da página inicial** como **0**.
- Se a requisição não especificar `page`, a API retornará os registros da **primeira página (0)**.

# MedicoControler

```
@GetMapping("algunsdados")
public Page<DadosListagemMedico> listarAlgunsDados(
    @PageableDefault( size=2,
                      page=0,
                      sort={"nome","crm"},
                      direction = Sort.Direction.DESC ) Pageable paginacao )
{

    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}
```

3. `sort = {"nome", "crm"}`

- Define a **ordenação padrão** dos registros.
- Primeiro, os dados serão ordenados pelo campo `nome`.
- Se houver nomes iguais, será usado um critério secundário: o campo `crm`.

4. `direction = Sort.Direction.DESC`

- Define a **direção da ordenação** como **decrecente (DESC)**.
- Isso significa que os registros serão listados do maior para o menor valor em relação aos campos de ordenação.

Não custa registrar novamente...  
isto é se você quer definir novos defaults.  
Não é obrigatório!



Pra fechar...

Como ver os comandos SQL no log ?

# Arquivo resources/application.properties

```
application.properties x
1  spring.application.name=api
2
3  # Configuração do banco de dados H2
4  spring.datasource.url=jdbc:h2:file:./DATA/pw3api
5  spring.datasource.username=sa
6  spring.datasource.password=
7  spring.h2.console.enabled=true
8  spring.h2.console.path=/h2-console
9
10 spring.jpa.show-sql=true
11 spring.jpa.properties.hibernate.format_sql=true
12
```

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true



Fazendo requisição para recuperar  
“alguns dados” ...





```
2023-09-25T12:24:15.631-03:00 INFO 15554 --- [ restartedMain] br.edu.ifsp.pw3.api.ApiApplication
2023-09-25T12:24:22.836-03:00 INFO 15554 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]
2023-09-25T12:24:22.836-03:00 INFO 15554 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
2023-09-25T12:24:22.837-03:00 INFO 15554 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet
```

Hibernate:

```
select
    m1_0.id,
    m1_0.crm,
    m1_0.email,
    m1_0.bairro,
    m1_0.cep,
    m1_0.cidade,
    m1_0.complemento,
    m1_0.logradouro,
    m1_0.numero,
    m1_0.uf,
    m1_0.especialidade,
    m1_0.nome,
    m1_0.telefone
from
    medicos m1_0
order by
    m1_0.crm desc offset ? rows fetch first ? rows only
```

Hibernate:

```
select
    count(m1_0.id)
from
    medicos m1_0
```

```
: Started ApiApplication in 2.671 seconds (
: Initializing Spring DispatcherServlet 'di
: Initializing Servlet 'dispatcherServlet'
: Completed initialization in 1 ms
```

Estado final do projeto,  
após a aula de hoje  
disponível no github

**[https://github.com/carlaopereirasanca/prw3\\_api\\_2025\\_2](https://github.com/carlaopereirasanca/prw3_api_2025_2)**  
(*“apos aula 2 valendo”*)



# Exercícios



# Exercícios

- Experimente com os códigos vistos na aula de hoje...



## Avaliação 3, parte 2



# Avaliações

## ~~■ Avaliação 1~~

~~■ Prática - Conteúdo: JPA.~~

~~■ Vale **15%** da média.~~

## ~~■ Avaliação 2~~

~~■ Escrita - Conteúdo: parte teórica (api rest, reqs. http, arquit. spring).~~

~~■ Vale **10%** da média.~~

## ■ Avaliação 3

■ Prática - Conteúdo: API Rest, CRUD, com Spring.

■ A nota final vale **30%** da média da disciplina.

## ■ Avaliação 4

■ Prática - Conteúdo: dotar a api criada na **av.3** de autenticação (com Spring Security).

■ Vale **45%** da média.

# Calendário

- ~~■ 17.set Desenvolvimento de uma API REST (parte 1).~~
- 24.set Desenvolvimento de uma API REST (parte 2). ←
- 1.out Desenvolvimento de uma API REST (parte 3). Enunciado da Avaliação 3
- 8.out **\*\* WECOMP \*\***
- 15.out Dia para trab. na Avaliação 3.
- 22.out Dia para trab. na Avaliação 3. **Entrega até 23:59.**





## Avaliação 3 parte 2

- Retome o projeto pedido na parte 1.
- Realize a validação dos dados enviados:
  - datas: formato “xx/xx/xxxx”
    - OBS: só o formato, não precisa se preocupar com número de dias, ano bissexto... (isto é, uma expressão regular simples dá conta!)
  - Dados obrigatórios:
    - nome do mecânico
    - marca do veículo
    - modelo do veículo
    - ano do veículo, no formato “xxxx”
- Crie uma **nova “migration”** para inserir o campo “cor” no Veículo
  - será um campo não obrigatório
- Crie na API **duas** repostas para a requisição GET:
  - retornando **todos** os dados (**com** recurso de paginação, **não** precisa configurar pageabledefault)
  - retornando **apenas** as datas, o nome do mecânico, a marca e o modelo do veículo (**sem paginação**)

## Avaliação 3 parte 2

- Retome o projeto pedido na parte 1.
- Realize a validação dos dados enviados:
  - datas: formato “xx/xx/xxxx”
    - OBS: só o formato, não precisa se preocupar com número de dias, ano bissexto... (isto é, uma expressão regular simples dá conta!)
  - Dados obrigatórios:
    - nome do mecânico
    - marca do veículo
    - modelo do veículo
    - ano do veículo, no formato “xxxx”
- Crie uma **nova “migration”** para inserir o campo “cor” no Veículo
  - será um campo não obrigatório
- Crie na API **duas** repostas para a requisição GET:
  - retornando **todos** os dados (**com** recurso de paginação, **não** precisa configurar pageabledefault)
  - retornando **apenas** as datas, o nome do mecânico, a marca e o modelo do veículo (**sem paginação**)

**Obrigatório!!!**

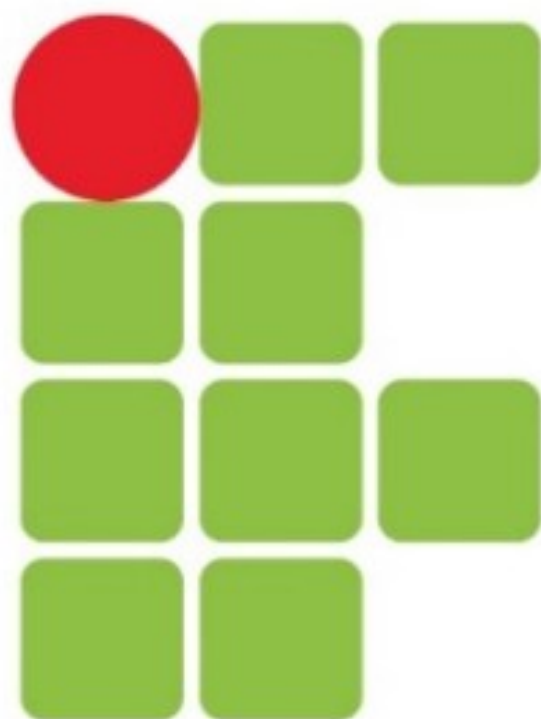


# AVISO!! (será repetido todas as aulas)

- Mantenha a disposição de cabos nos computadores dos labs.
- Se retirar um cabo de rede (para usar notebook), não esqueça de repor no computador ao final da aula.
- Mantenha o filtro de linha e as tomadas, atrás da mesa, em seu estado original, especialmente para que não possa ser esbarrado por outros alunos.
- Não altere de jeito nenhum os cabos atrás dos computadores, especialmente mexendo nas “travas”.
- **Por favor, desliguem a(s) máquina(s) ao final da aula.**
- Lembre-se! Os laboratórios são nossos!! Vamos cuidar!







**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**  
SÃO PAULO  
Câmpus São Carlos



**INSTITUTO FEDERAL DE  
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**  
SÃO PAULO  
Câmpus São Carlos