Sumário

Introdução 3
Objetivos 3
Pré requisitos 4
1 - Criação do projeto 5
Spring initializr 5
Estrutura do projeto 5
Executando o projeto 6
Configuração do IntelliJ: Build automático 7
Criando nosso Controller 7
2 - Requisições POST 8
Criação do cadastro de médicos 8
Fazendo a requisição 9
3 - Spring Data JPA 10
Adicionando dependências 10
Executando a aplicação sem configurações de banco de dados 11
Definindo configurações com o banco de dados 11
Criando banco de dados 12
Executando a aplicação com configurações de banco de dados 13
Criação de entidades JPA 13
Interfaces Repository 14
Executando o projeto a aplicação: 16
Migrations com Flyway 16
Criação do diretório das Migrations: 17
Criação de arquivos Migration: 18
Executando o projeto com a Migration configurada: 18
Validando no MySQL Workbench: 18
Validando MySQL terminal: 19
Executando a aplicação e fazendo a requisição: 20
Bean Validation 21
Anotando o DTO com o Validation: 21
Anotando controller 22
Testando as validações 22
Nova migration: Alteração propriedades do projeto 23
Criar nova Migration 24
4 – Requisição Get 26
Criando DTO de listagem 26
Testando a listagem 27

Paginação 27
Ordenação.....28
Ordenando Decrescente: 29
Plus: Mostrar queries de banco de dados executadas 29
Alteração dos nomes dos parâmetros de paginação.....30
5 - PUT e DELETE..... 31
Criando a requisição PUT 31
Criando o método de atualização de médico: 32
Criando DTO para definir os dados que serão atualizados: 32
Testando a requisição 34
Criando a requisição DELETE 35
Exclusão no banco de dados: 35
Exclusão lógica: 36

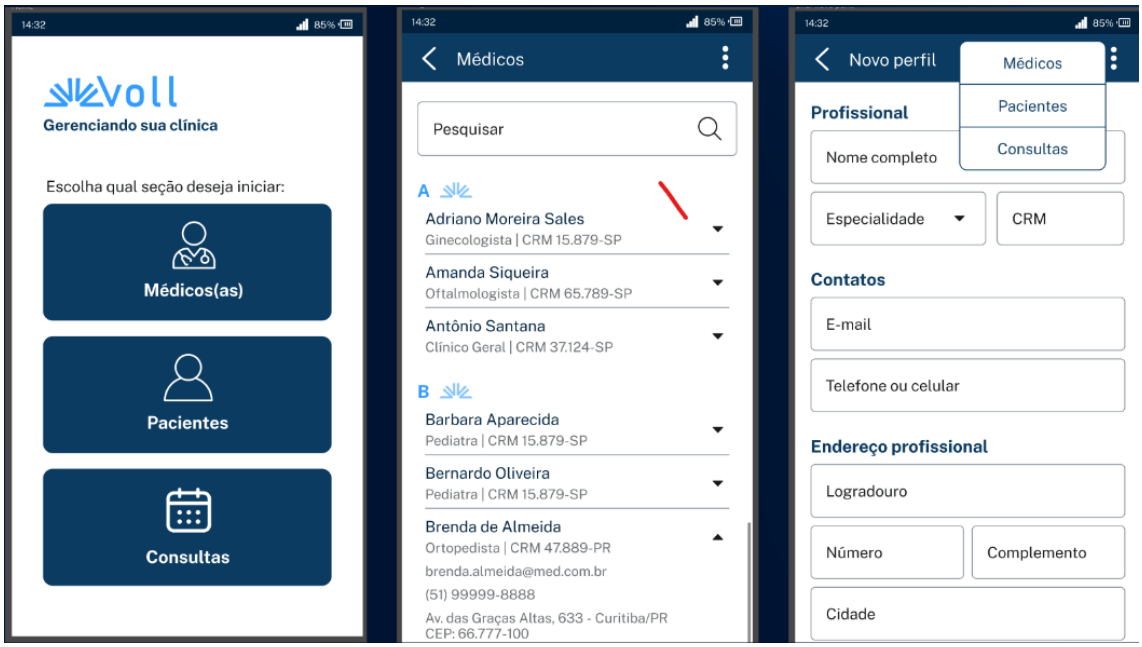
Introdução

Objetivos

- Desenvolvimento de uma API Rest
- CRUD (Create, Read, Update e Delete)
- Validações
- Paginação e ordenação

Trabalharemos em um projeto de uma clínica médica fictícia. Temos uma empresa chamada Voll Med, que possui uma clínica que precisa de um aplicativo para monitorar o cadastro de médicos, pacientes e agendamento de consultas.

Será um aplicativo com algumas opções, em que a pessoa que for usar pode fazer o CRUD, tanto de médicos quanto de pacientes e o agendamento e cancelamento das consultas.



Pré requisitos

Java JDK 17:

https://www.oracle.com/java/technologies/downloads/

IntelliJ IDE:

https://www.jetbrains.com/pt-br/idea/

Insomnia:

https://insomnia.rest/download

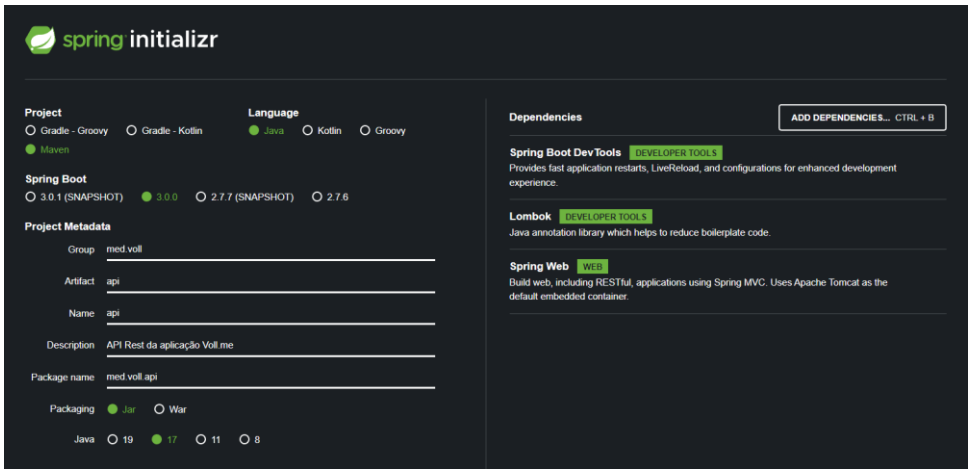
MySQL:

https://www.alura.com.br/artigos/mysql-instalacao-configuracao

1 - Criação do projeto

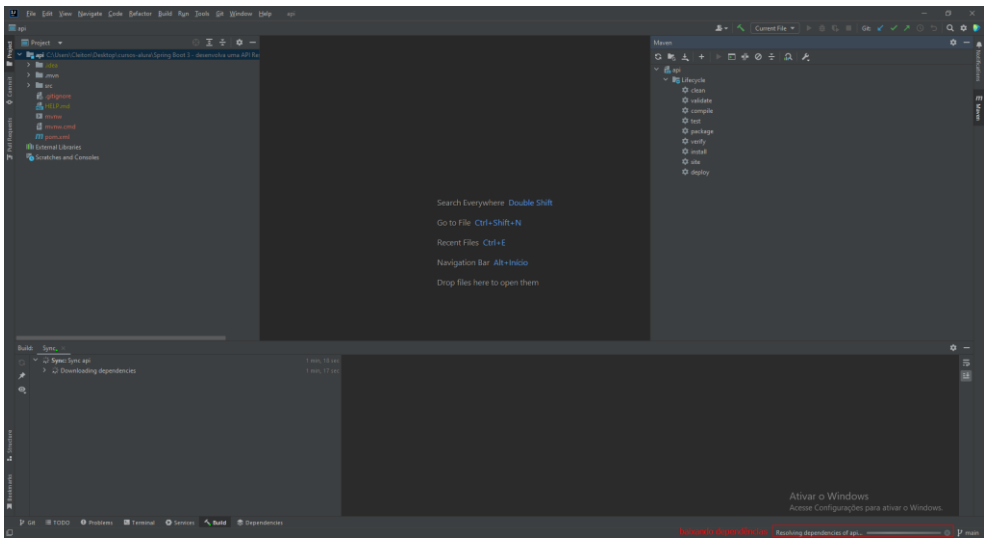
Spring initializr

https://start.spring.io/

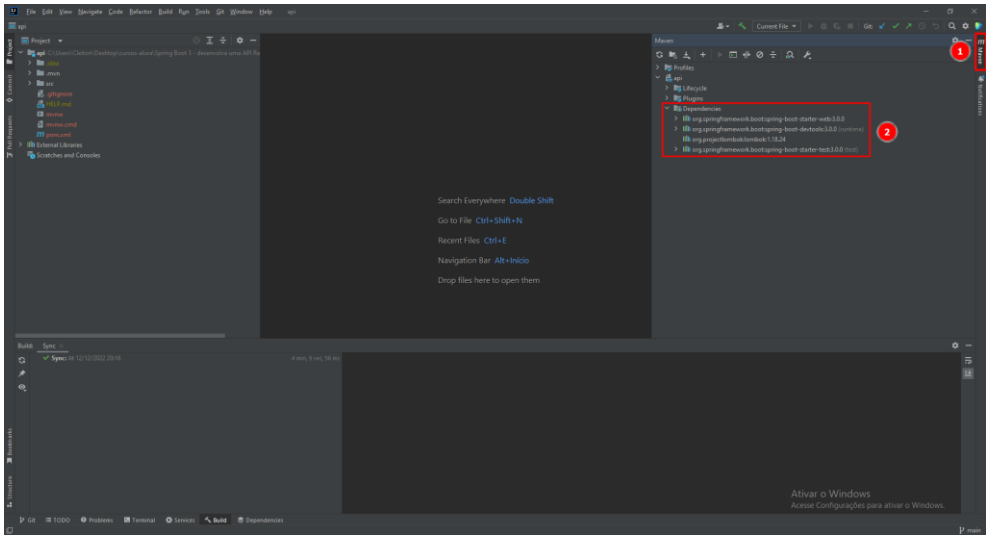


Estrutura do projeto

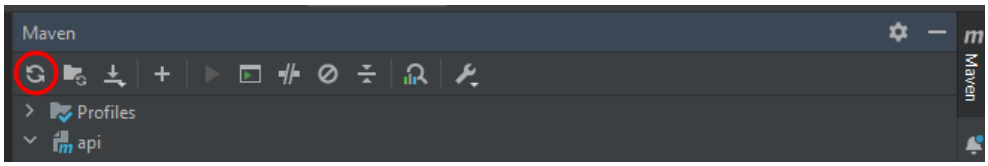
Validar se dependências já foram baixadas



Todas as dependências que foram adicionadas devem estar listadas

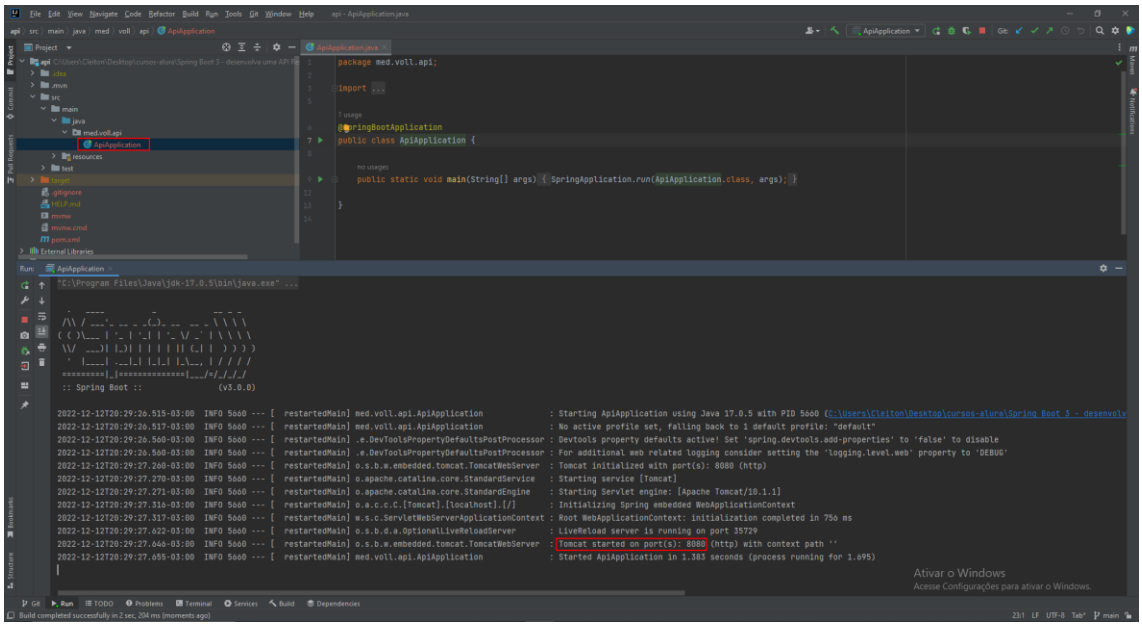


Caso der algum problema, basta clicar em reload que o projeto será recarregado e tentará baixas as dependências que falta.

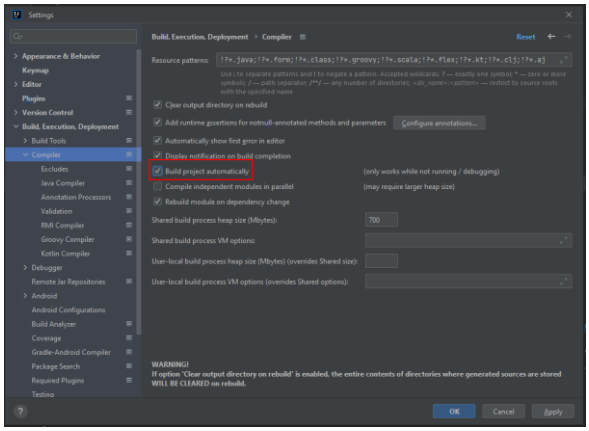


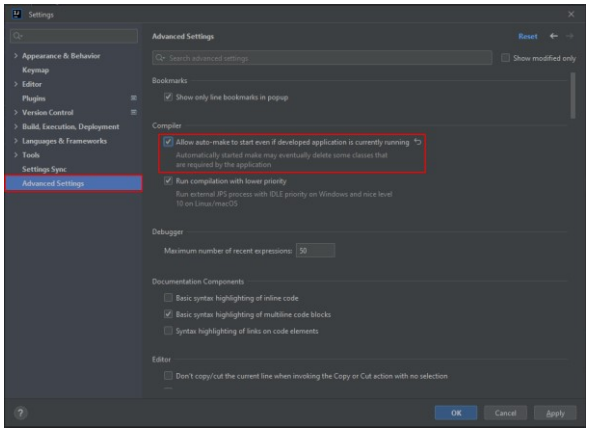
Executando o projeto

Antigamente para executar um projeto Spring, era necessário criar um servidor e colocar o projeto dentro. Agora com o SpringBoot, o servidor está dentro do projeto, incluso.

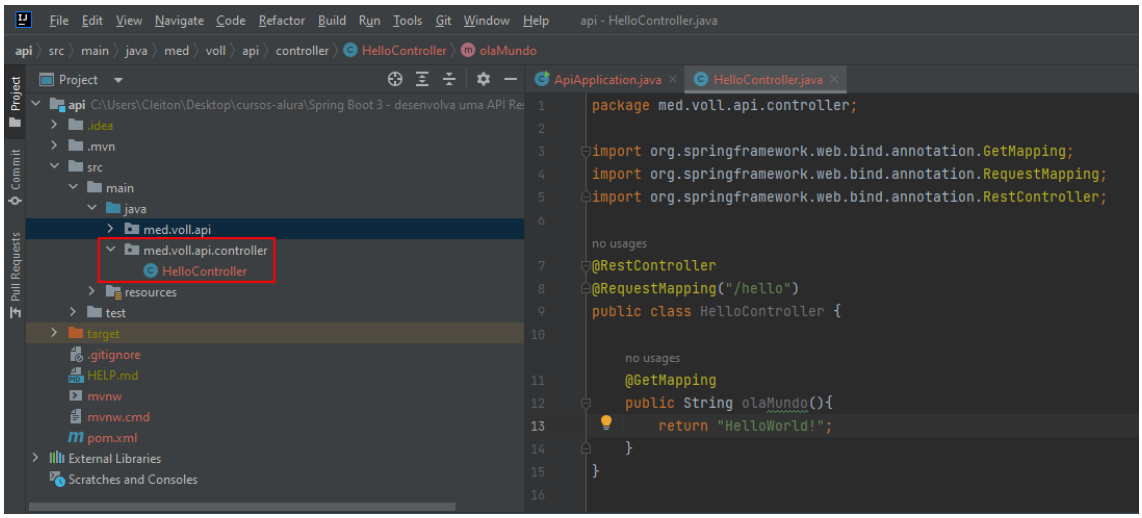


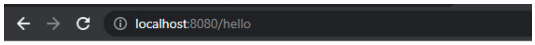
Configuração do IntelliJ: Build automático





Criando nosso Controller





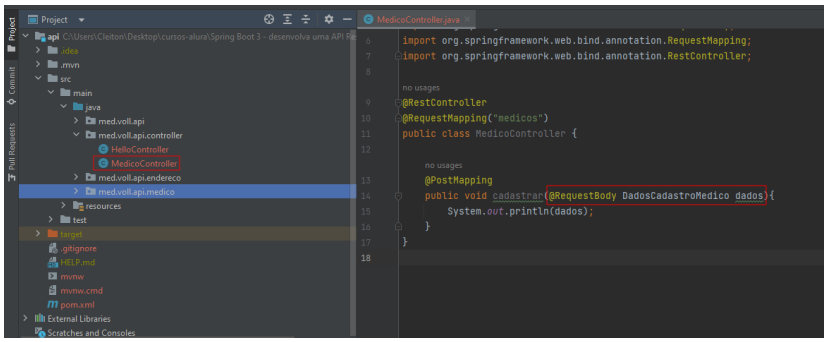
HelloWorld!

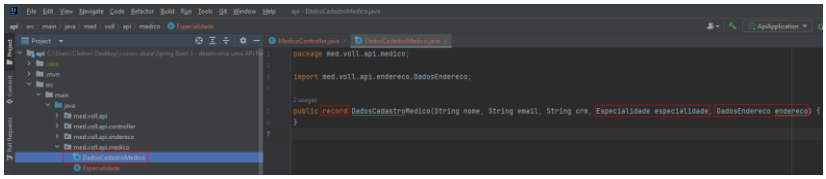
2 - Requisições POST

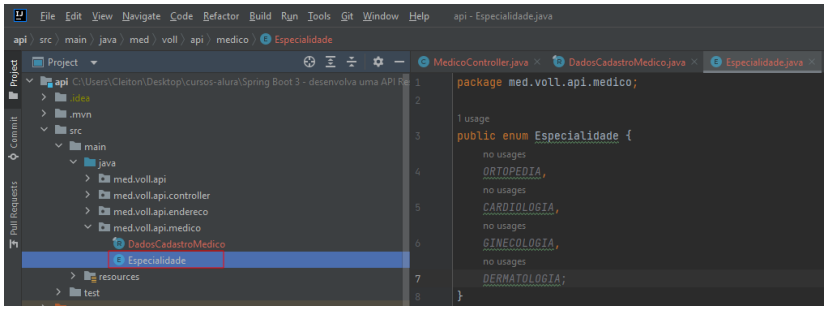
Criação do cadastro de médicos

https://trello.com/b/O0lGCsKb/api-voll-med

https://www.figma.com/file/N4CgpJqsg7gjbKuDmra3EV/Voll.med

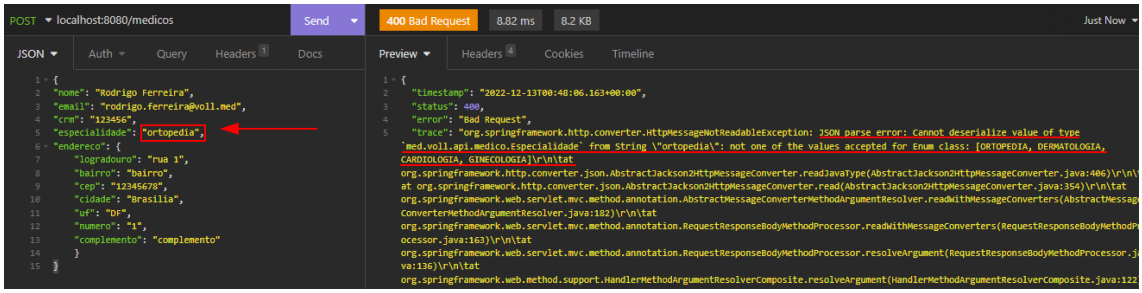


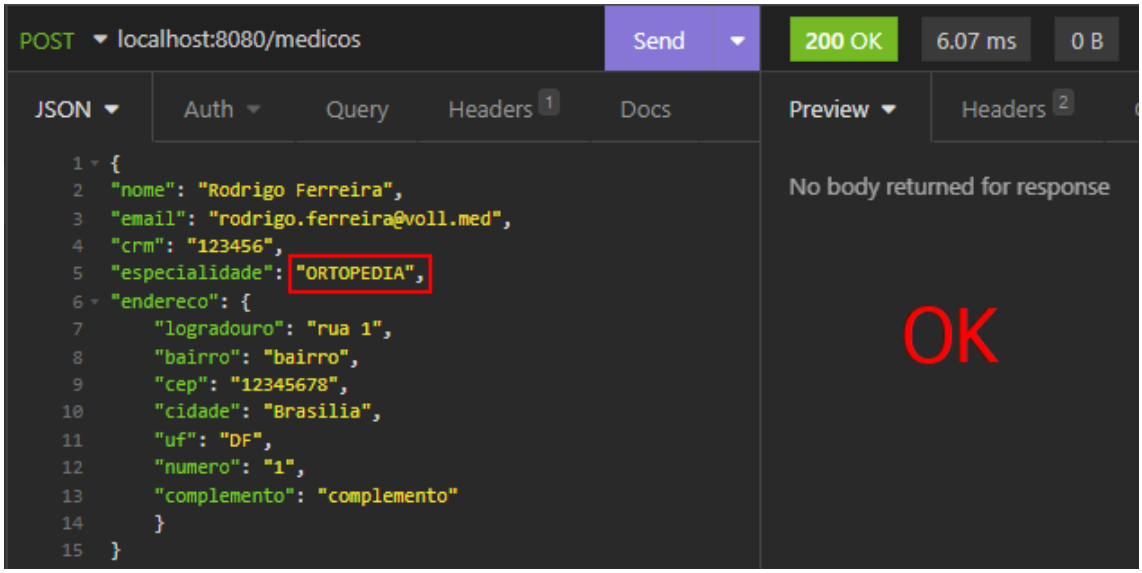


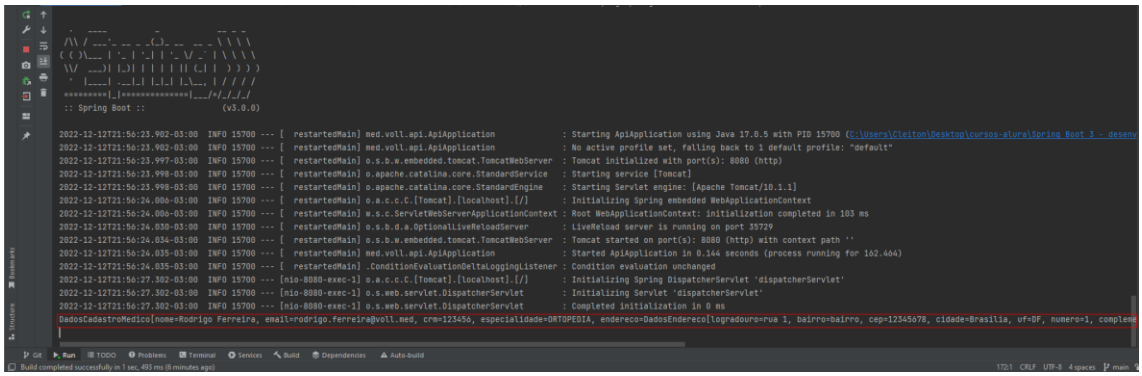


Fazendo a requisição

Observe que deu um erro por conta do enum que está minúsculo. Temos que colocar exatamente igual ao codificado.



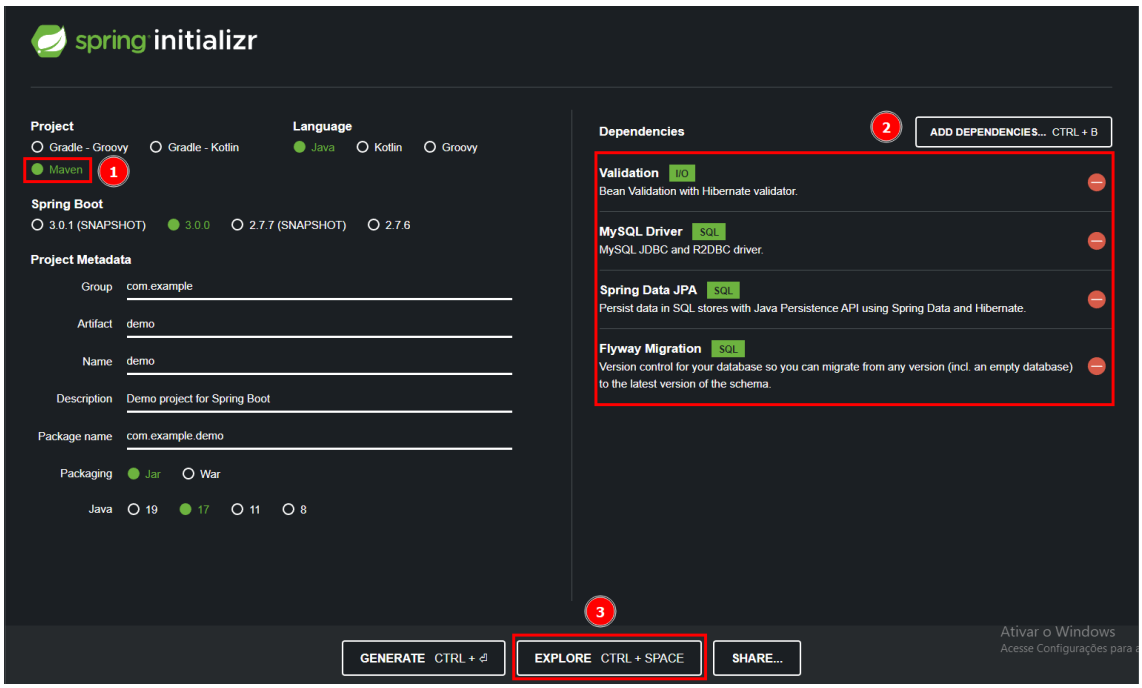


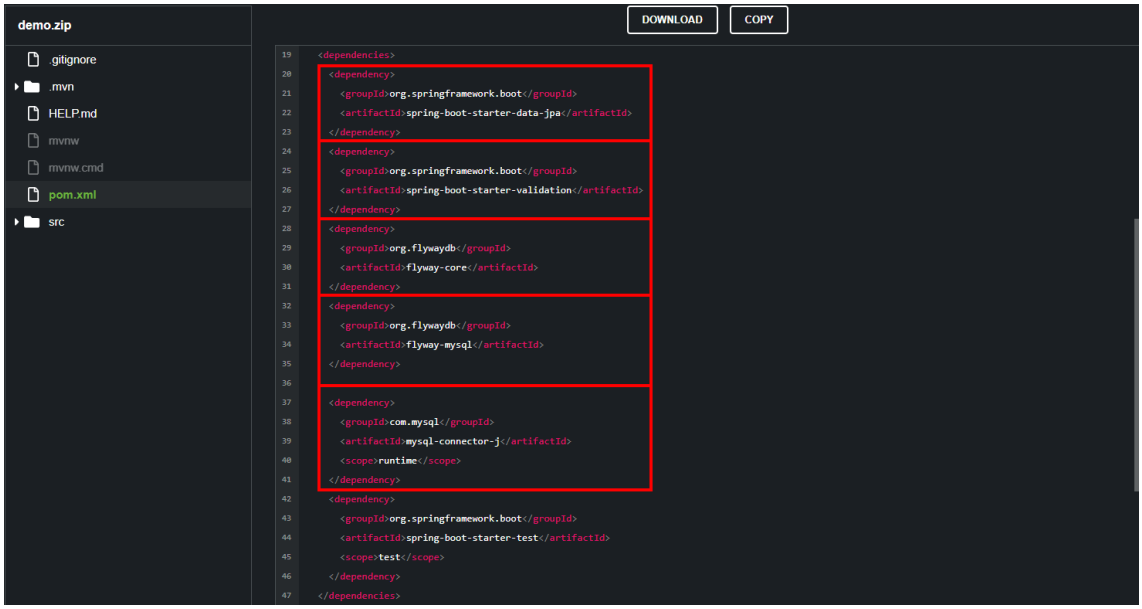


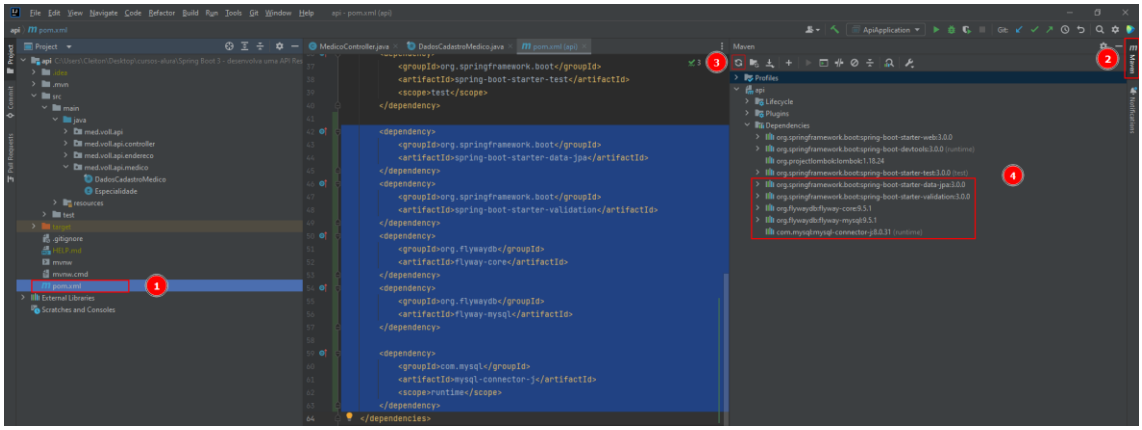
Obs.: Certifique que TODOS os atributos estão com os mesmos nomes do Json

3 - Spring Data JPA

Adicionando dependências

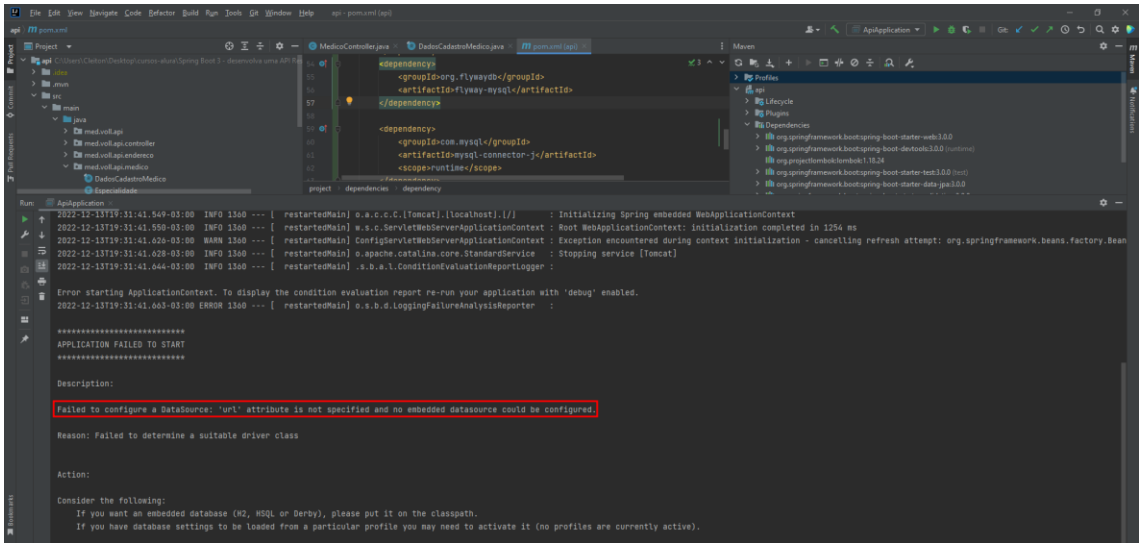






Executando a aplicação sem configurações de banco de dados

Quando adicionamos as dependências do JPA, o Spring tenta pegar as configurações para fazer a conexão com o banco de dados, mas como não adicionamos, o seguinte erro é exibido:

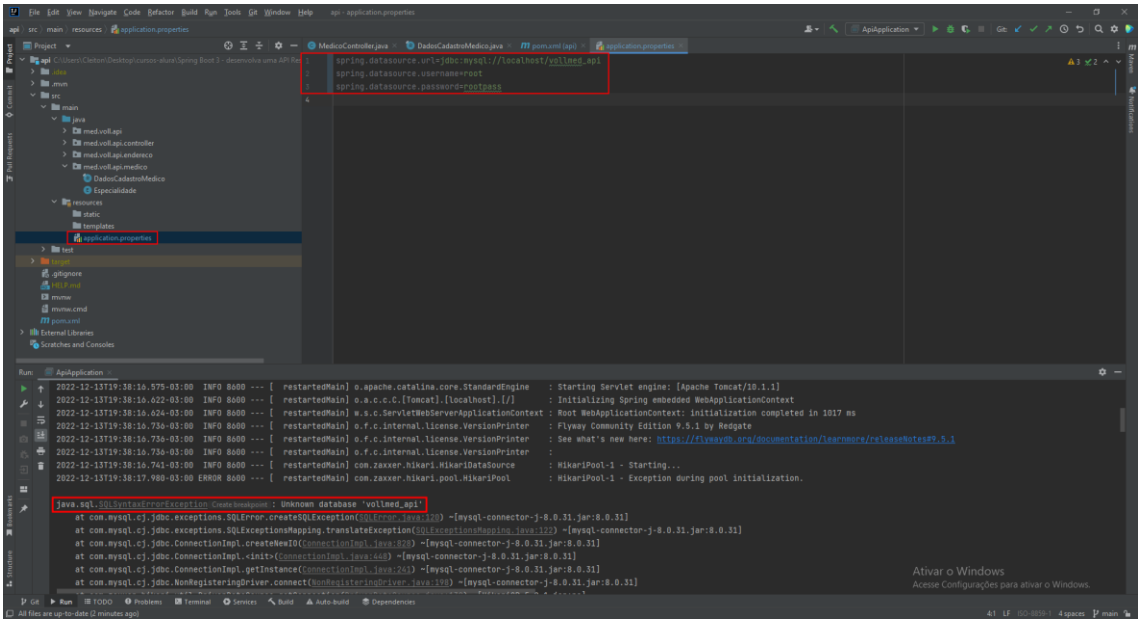


Definindo configurações com o banco de dados

Para passar a URL de conexão com o banco de dados, vamos inserir o código: spring.datasource.url=jdbc:mysql://localhost/vollmed_api. Passaremos também spring.datasource.username=root, para definir o username, e spring.datasource.password=root, para definir a senha.

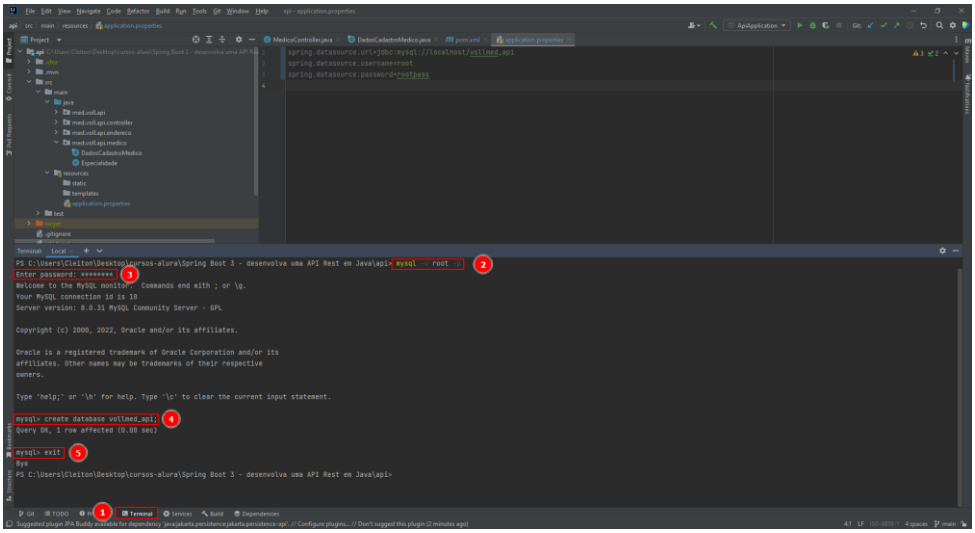
Ao executar aparece o seguinte erro: nknown database 'vollmed_api'

Faz sentido, já que não criamos o nosso banco de dados.

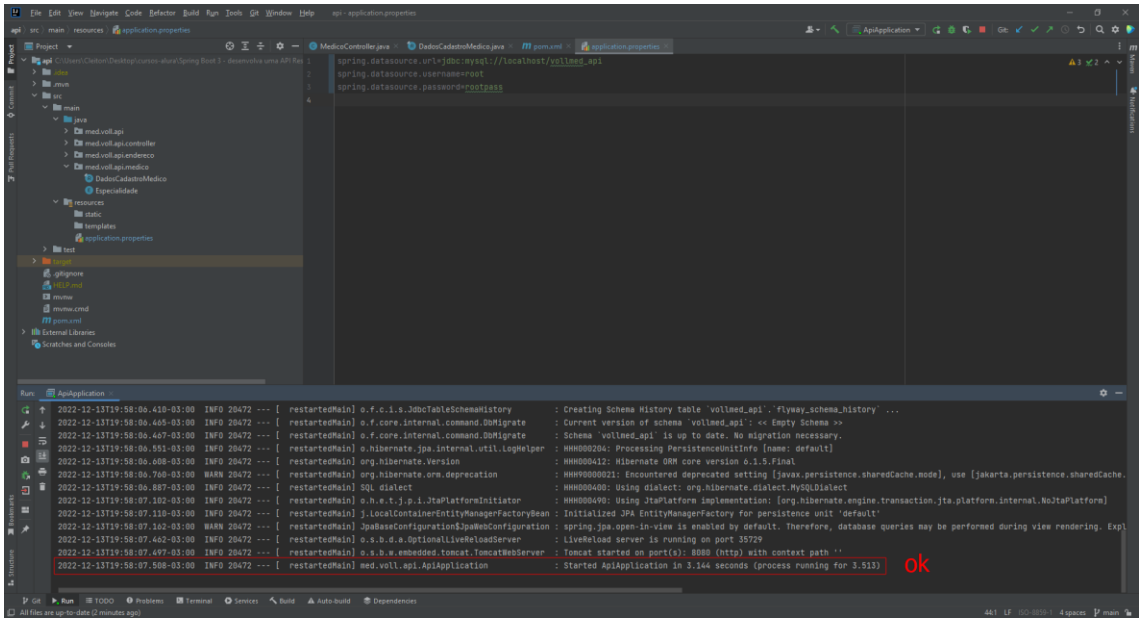


Criando banco de dados

Caso o comando *mysql* não seja reconhecido, adicionar nas variáveis do sistema!



Executando a aplicação com configurações de banco de dados



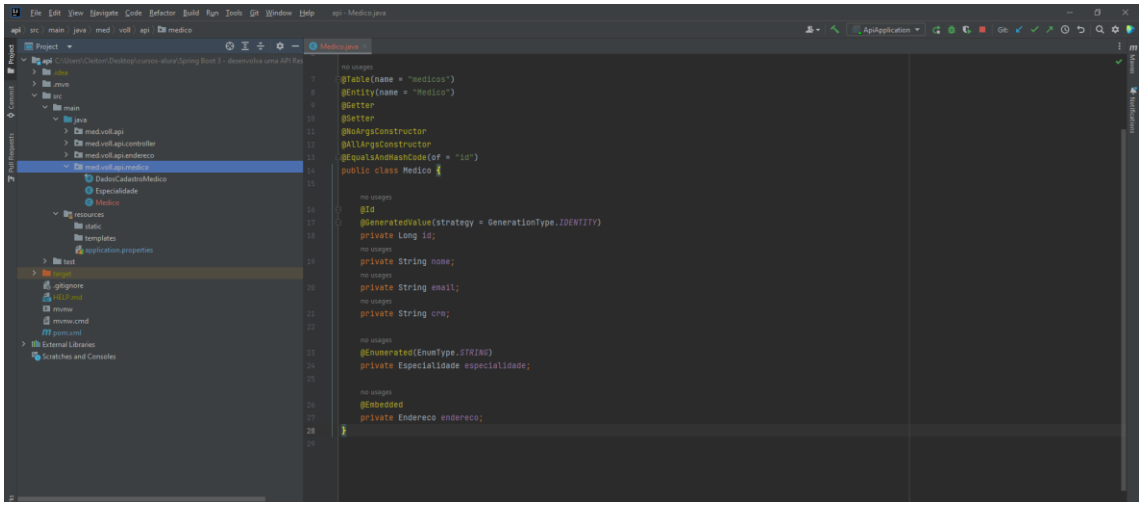
Criação de entidades JPA

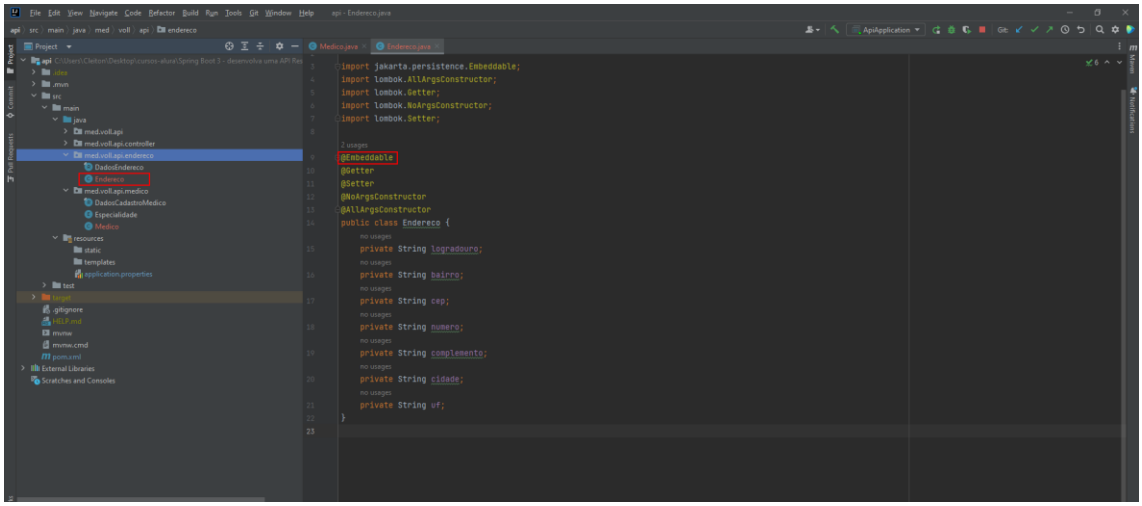
Criamos as nossas Records para o DTO (Data Transfer Object), agora precisamos de nossas entidades JPA (Java Persistence API), ou seja, as classes que irão mapear as entidades do banco de dados.

Para facilitar nossa vida, vamos usar o lombok par acriação de getters, setters, construtores, equals e hash code.

Nota: @Embedded em Medico

Vamos usar Embeddable Attribute da JPA para que Endereco fique em uma classe separada, mas faça parte da mesma tabela de Medicos junto ao banco de dados.

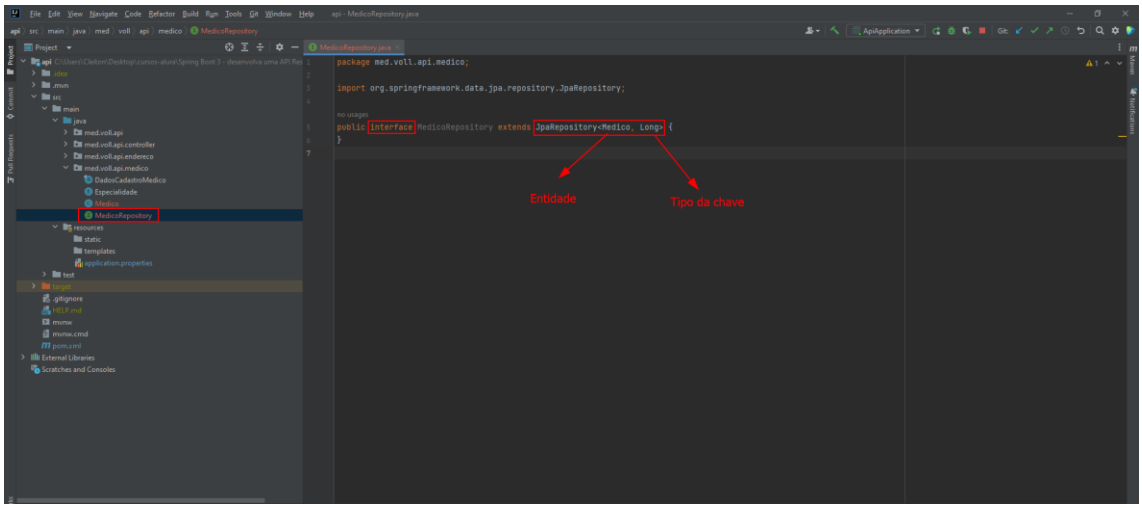




Interfaces Repository

Para fazer a persistência, pegar o objeto Medico e salvar no banco de dados, o Spring Data tem o Repository, que são interfaces. O Spring já nos fornece a implementação.

Repository:



Controller:

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

import med.voll.api.medico.DadosCadastroMedico;
import med.voll.api.medico.Medico;
import med.voll.api.medico.MedicoRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

    @Autowired
    private MedicoRepository repository;

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

Repository com injeção do Spring

Passando o nosso DTO dados no construtor da entidade Medico

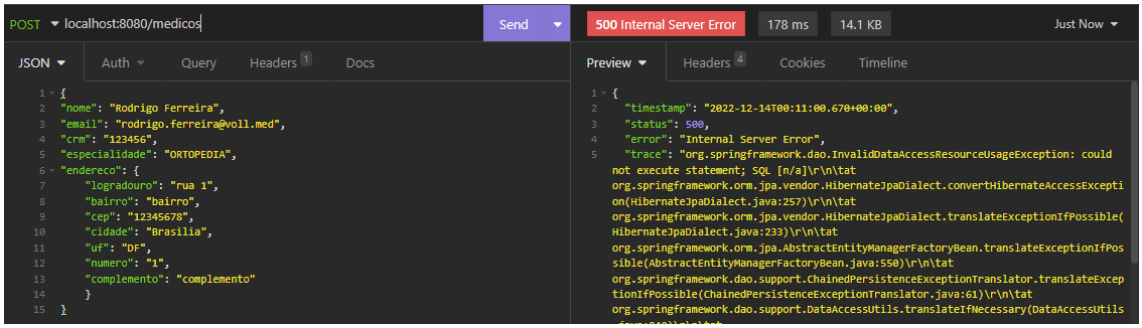
Médico:

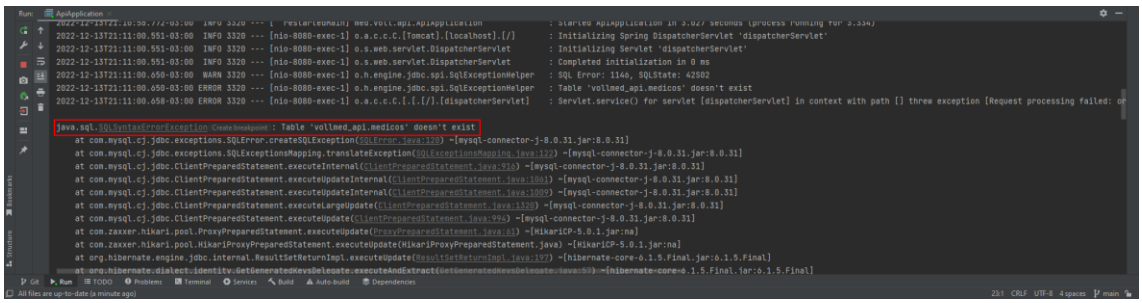
```java
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nome;

    private String email;

    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.crm = dados.crm();
        this.especialidade = dados.especialidade();
        this.endereco = new Endereco(dados.endereco());
    }
}
```

Endereço:

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Endereco {

    private String logradouro;

    private String bairro;

    private String cep;

    private String numero;

    private String complemento;

    private String cidade;

    private String uf;

    public Endereco(DadosEndereco dados) {
        this.logradouro = dados.logradouro();
        this.bairro = dados.bairro();
        this.cep = dados.cep();
        this.uf = dados.uf();
        this.cidade = dados.cidade();
        this.numero = dados.numero();
        this.complemento = dados.complemento();
    }
}
```

Executando o projeto a aplicação:





Um erro esperado, já que apenas criamos o banco de dados mas não a tabela médicos, como mapeado na entidade Medico

```
7   @Table(name = "medicos")
8   @Entity(name = "Medico")
9   @Getter
10  @Setter
11  @NoArgsConstructor
12  @AllArgsConstructor
13  @EqualsAndHashCode(of = "id")
14  public class Medico {
```

Migrations com Flyway

Usaremos migrations, ou ferramentas de migrações, para **registrar as atualizações no banco de dados**. Já registramos o Flyway nas dependências, uma dessas ferramentas suportadas pelo Spring Boot. Dessa forma iremos ter todo um histórico de banco de dados, assim como o histórico de nossos códigos no Git.

Para cada mudança que quisermos executar no banco de dados, precisamos criar um arquivo .sql no projeto e, nele, escrever o trecho do comando SQL que será executado no banco de dados.

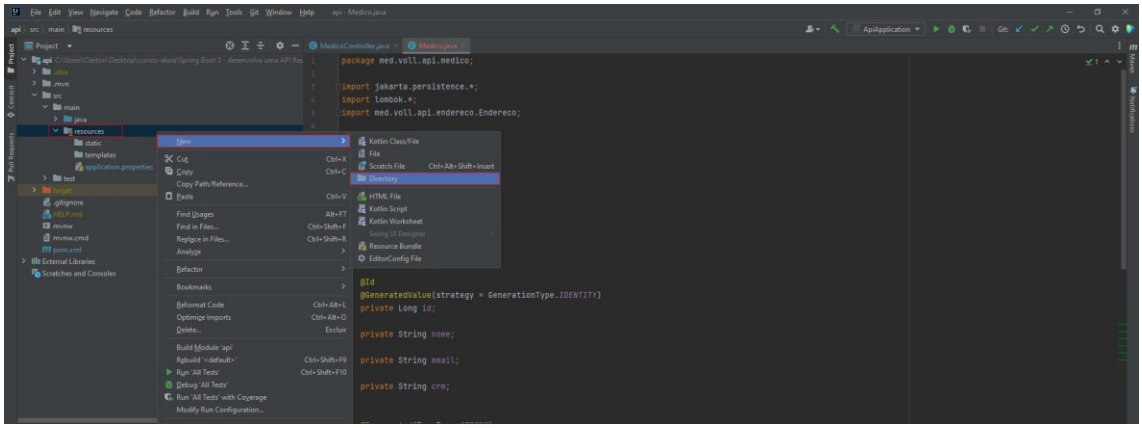
Precisamos salvá-los em um diretório específico. Criaremos essa nova pasta em "main > resources". Com "Alt + Insert", vamos escolher a opção "Directory" e digita o nome da pasta: "db/migration".

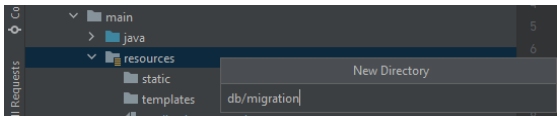
Obs: A barra significa que será criada uma subpasta, "migration", dentro da pasta "db".

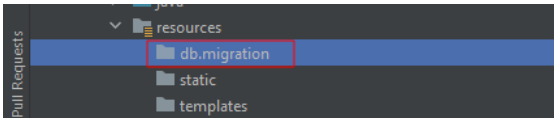
Dentro da pasta, criaremos um arquivo SQL que servirá como nossa primeira migration, responsável por criar a tabela de médicos. Antes disso, é preciso interromper o projeto antes de usar migrations.

Obs: Sempre interrompa o projeto ao usar migrations.

Criação do diretório das Migrations:







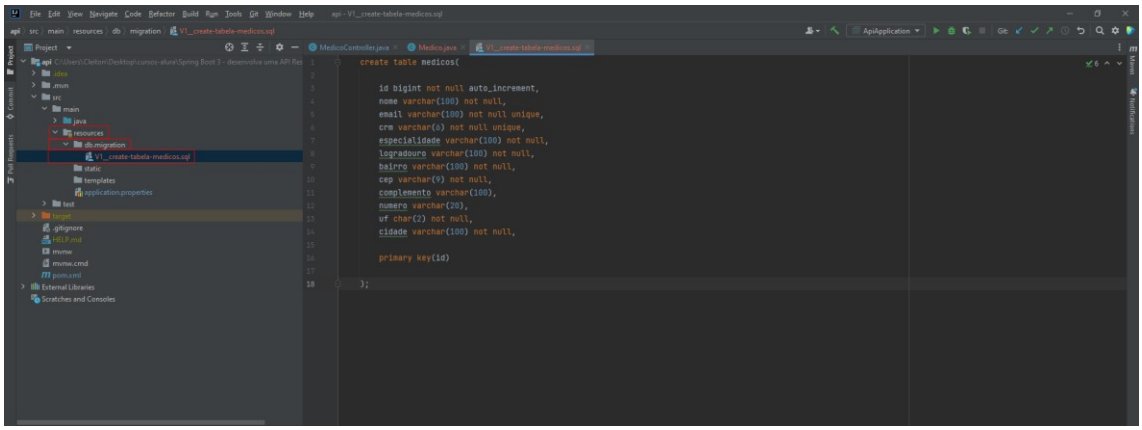
*** OBS: SEMPRE INTERROMPA O PROJETO AO USAR MIGRATIONS. SE AINDA NÃO PAROU, PARE AGORA, QUE PODE DAR RUIM ISSO AÍ!!!

Criação de arquivos Migration:

A Migration tem um padrão: V{numero da versão}_ _nome-sugestivo.sql

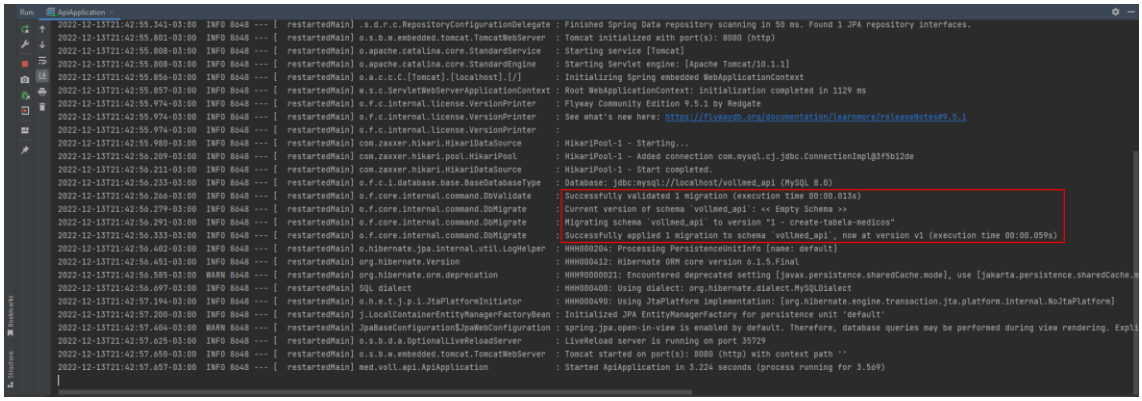
Obs.: a letra V da versão sempre maiúscula.

Ex.: V1__create-tabela-medicos.sql

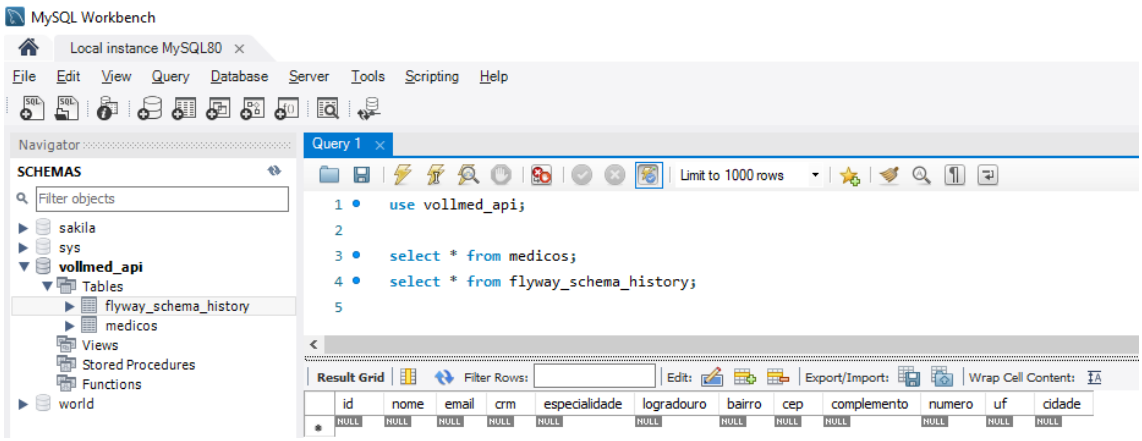


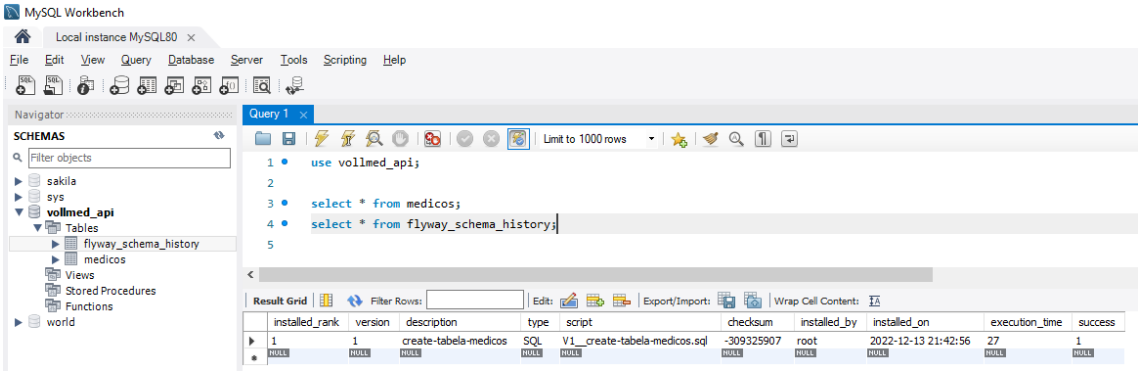
Executando o projeto com a Migration configurada:

Encontrou uma Migration, validou a versão e estava vazia, executou o script e logou que executou com sucesso



Validando no MySQL Workbench:





Validando MySQL terminal:

```
PS C:\Users\Cleiton\Desktop\cursos-alura\Spring Boot 3 - desenvolva uma API Rest em Java\api> mysql -u root -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 247
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use vollmed_api;
Database changed
```

```
PS C:\Users\Cleiton\Desktop\cursos-alura\Spring Boot 3 - desenvolva uma API Rest em Java\api> mysql -u root -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 247
Server version: 8.0.31 MySQL Community Server - GPL

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use vollmed_api;
Database changed
mysql> show tables;
+-----------------------+
| Tables_in_vollmed_api |
+-----------------------+
| flyway_schema_history |
| medicos               |
+-----------------------+
2 rows in set (0.00 sec)

mysql>
```

Git | Run | TODO | Problems | Terminal | Services | Build | Auto-build | Dependencies

Executando a aplicação e fazendo a requisição:

```
no usages  marquescleiton
20    @PostMapping
21    @Transactional
22    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados){
23        repository.save(new Medico(dados));
24    }
25 }
26
```

Impede que multiplas aplicações manipulem o banco ao mesmo tempo, garantindo as transações e evitando travar devido ter outro processo em andamento.

POST localhost:8080/medicos | Send | 200 OK | 166 ms | 0 B | 2 Minutes Ago

JSON | Auth | Query | Headers 1 | Do | Preview | Headers 2 | Cookies | Timeline

```
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 }
```

No body returned for response

MySQL Workbench

Local instance MySQL80

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- sakila
- sys
- vollmed_api
  - Tables
    - flyway_schema_history
    - medicos
  - Views
  - Stored Procedures
  - Functions
- world

Query 1

Limit to 1000 rows

```
1 • use vollmed_api;
2
3 • select * from medicos;
4 • select * from flyway_schema_history;
5
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

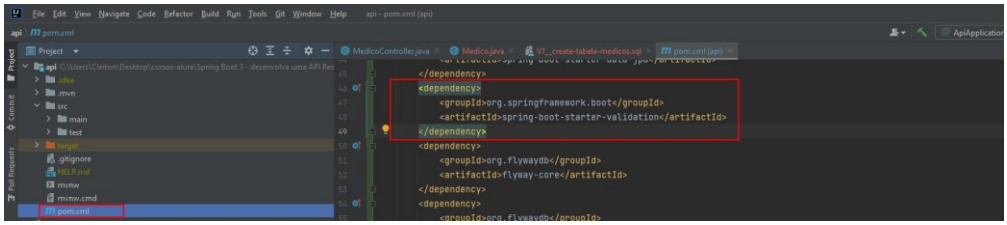
id	nome	email	crm	especialidade	logradouro	bairro	cep	complemento	numero	uf	cidade
1	Rodrigo Ferreira	rodrigo.ferreira@voll.med	123456	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Bean Validation

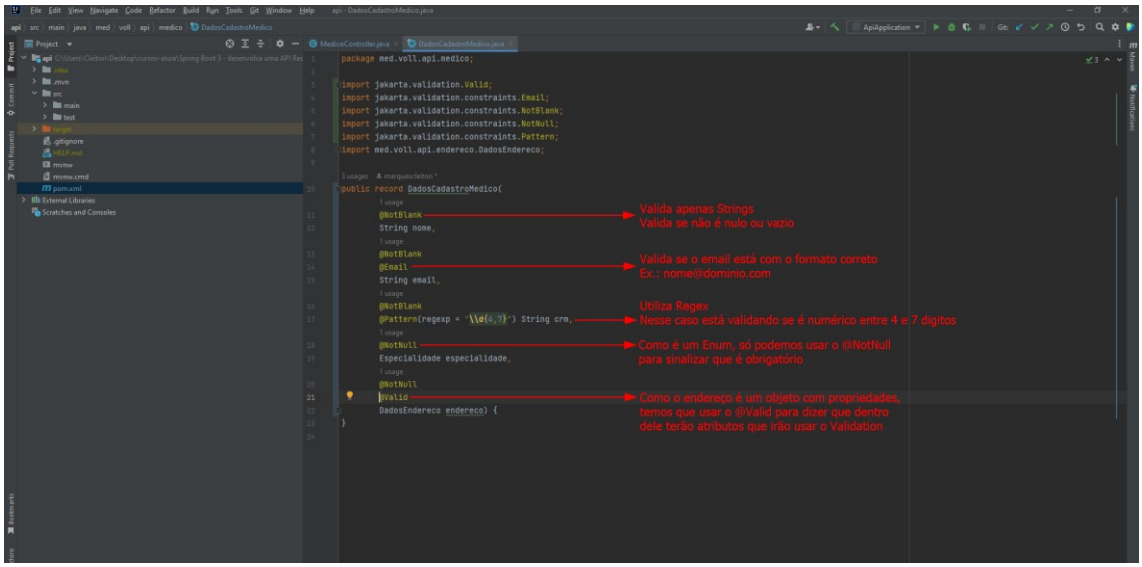
As validações nos ajuda a colocar restrições em nossos DTOs. Dessa forma podemos validar se a informações como:

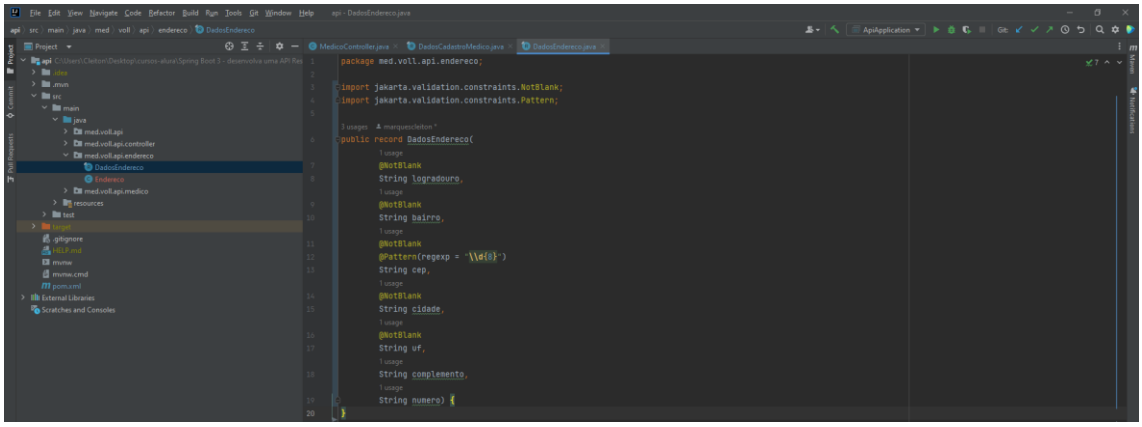
- Campo não pode ser nulo;
- Apenas valor numérico;
- Apenas letras;
- Valor mínimo;
- Valor máximo;
- Patters para restringir formatos específicos: CPF, CNPJ, Email...

Dependência foi adicionado na construção do projeto



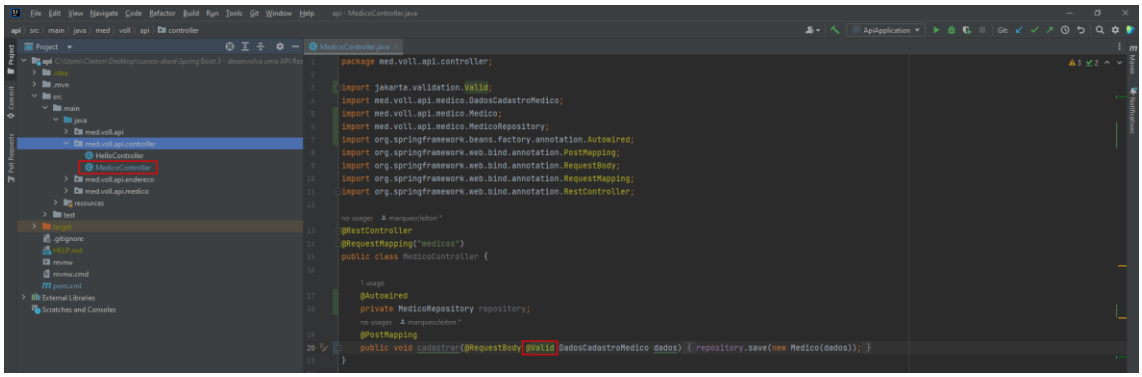
Anotando o DTO com o Validation:



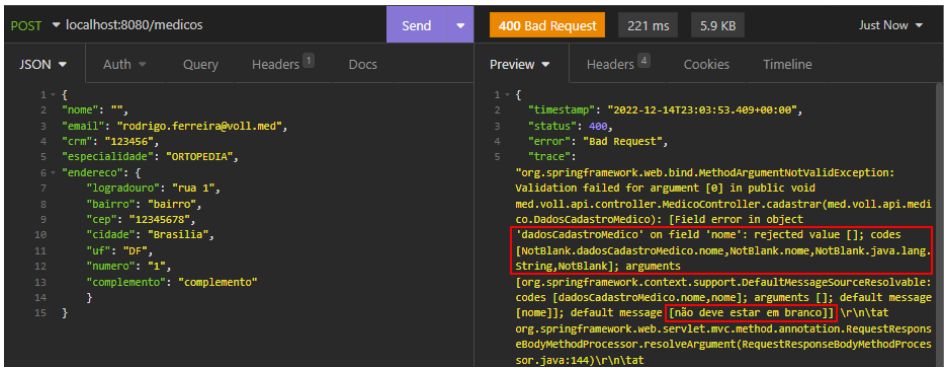


Anotando controller

É importante adicionaremos @Valid ao nosso Controller, para solicitar que o Spring se integre ao Bean Validation e execute as validações.



Testando as validações



POST localhost:8080/medicos Send

JSON Auth Query Headers 1 Docs

```
1 {
2 "nome": "Rodrigo Ferreira",
3 "email": "rodrigo.ferreira@voll.med",
4 "crm": "123",
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 }
```

400 Bad Request 5.96 ms 6 KB Just Now

Preview Headers 4 Cookies Timeline

```
1 {
2   "timestamp": "2022-12-14T23:06:23.587+00:00",
3   "status": 400,
4   "error": "Bad Request",
5   "trace":
"org.springframework.web.bind.MethodArgumentNotValidException:
Validation failed for argument [0] in public void
med.voll.api.controller.MedicoController.cadastrar(med.voll.api.medi
co.DadosCadastroMedico): [Field error in object
'dadosCadastroMedico' on field 'crm': rejected value [123]; codes
[Pattern.dadosCadastroMedico.crm,Pattern.crm,Pattern.java.lang.Strin
g,Pattern]; arguments
[org.springframework.context.support.DefaultMessageSourceResolvable:
codes [dadosCadastroMedico.crm,crm]; arguments []; default message
[crm],
[Ljakarta.validation.constraints.Pattern$Flag;@304378cf,\\d{4,7}];
default message [deve corresponder a \"\\d{4,7}\"]] \r\n\tat
org.springframework.web.servlet.mvc.method.annotation.RequestRespons
```

POST localhost:8080/medicos Send

JSON Auth Query Headers 1 Docs

```
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 }
```

Como o CRM e o email foram configurados como únicos quando criamos a tabela, ao salvar outro médico com esses mesmos dados a aplicação levanta uma exception de duplicidade

500 Internal Server Error 9.02 ms 14.2 KB Just Now

Preview Headers 4 Cookies Timeline

```
1 {
2   "timestamp": "2022-12-14T23:11:40.791+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "trace": "org.springframework.dao.DataIntegrityViolationException:
could not execute statement; SQL [n/a]; constraint
[medicos.email]\r\n\tat
org.springframework.orm.jpa.vendor.HibernateJpaDialect.convertHibern
ateAccessException(HibernateJpaDialect.java:270)\r\n\tat
org.springframework.orm.jpa.vendor.HibernateJpaDialect.translateExce
ptionIfPossible(HibernateJpaDialect.java:233)\r\n\tat
org.springframework.orm.jpa.AbstractEntityManagerFactoryBean.transla
teExceptionIfPossible(AbstractEntityManagerFactoryBean.java:550)\r\n
\tat
org.springframework.dao.support.ChainedPersistenceExceptionTranslato
r.translateExceptionIfPossible(ChainedPersistenceExceptionTranslator
.java:61)\r\n\tat
org.springframework.dao.support.DataAccessUtils.translateIfNecessary
(DataAccessUtils.java:242)\r\n\tat
org.springframework.dao.support.PersistenceExceptionTranslationInter
```

POST localhost:8080/medicos Send

JSON Auth Query Headers 1 Docs

```
1 {
2 "nome": "Sérgio Mallandro",
3 "email": "sergio.malandro@voll.med",
4 "crm": "654321",
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 }
```

200 OK 22.5 ms 0 B Just Now

Preview Headers 2 Cookies Timeline

No body returned for response

```
3 • select * from medicos;
4
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	nome	email	crm	especialidade	logradouro	bairro	cep	complemento	numero	uf	cidade
1	Rodrigo Ferreira	rodrigo.ferreira@voll.med	123456	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia
5	Sérgio Mallandro	sergio.malandro@voll.med	654321	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

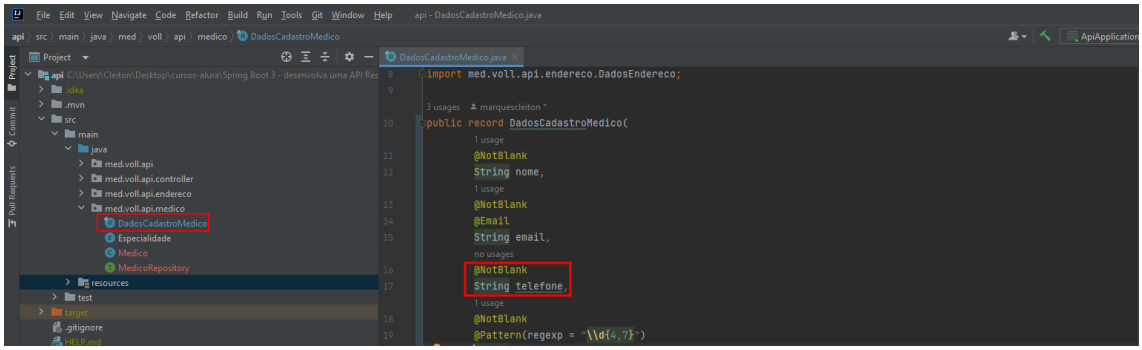
Nova migration: Alteração propriedades do projeto

Ao criar a entidade e a tabela no banco, acabamos não colocando o telefone

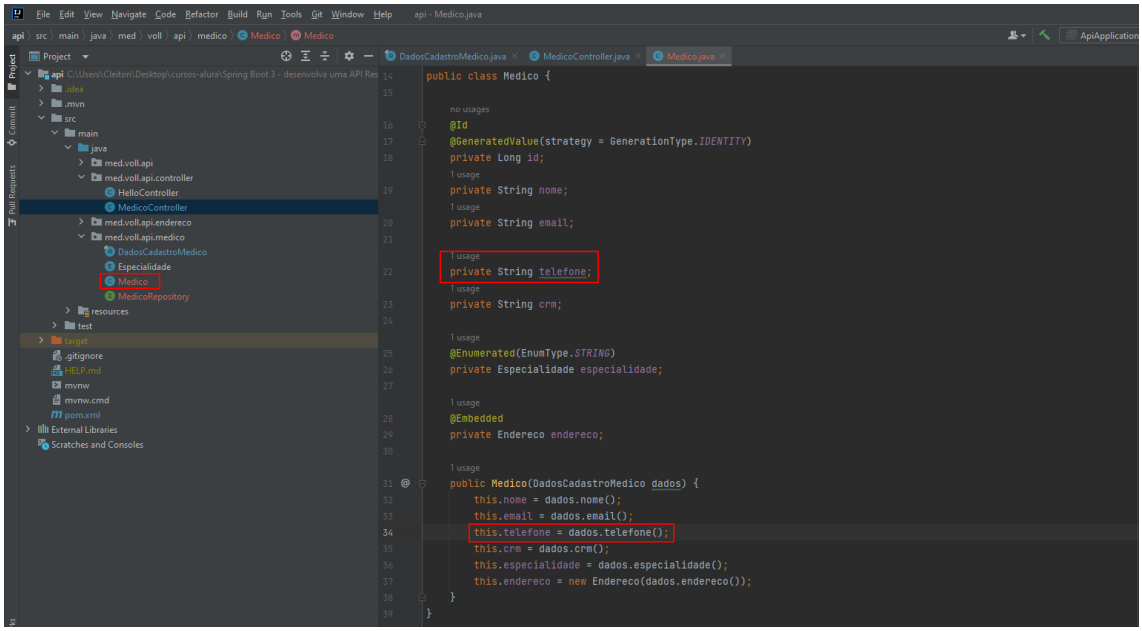
O sistema deve possuir uma funcionalidade de cadastro de médicos, na qual as seguintes informações deverão ser preenchidas:

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

Alterando DTO

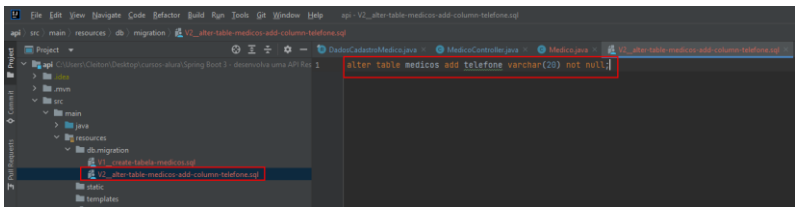


Alterando JPA

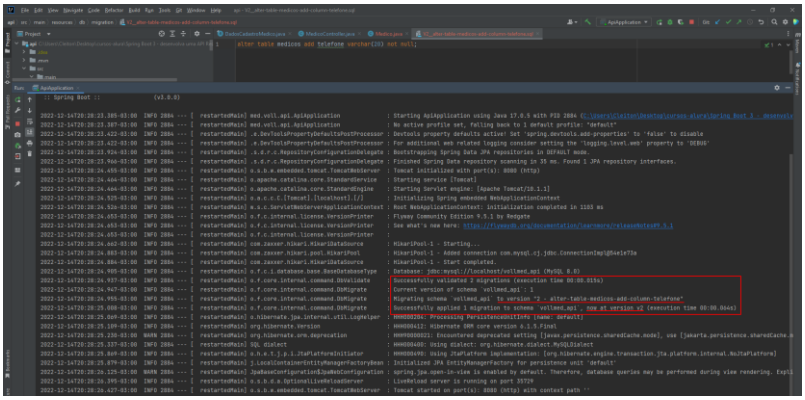


Criar nova Migration

Obs: Sempre interrompa o projeto ao usar migrations.



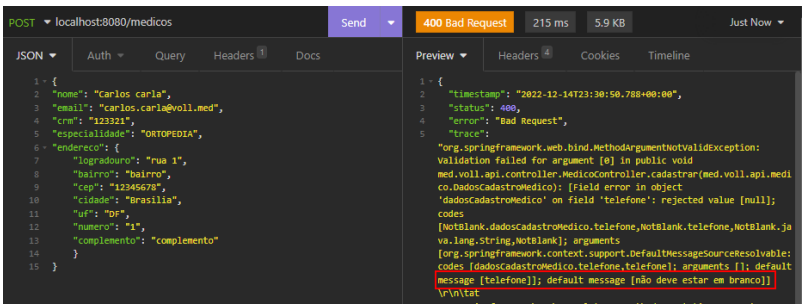
Executando o projeto...



```
select * from medicos;
```

Result Grid

id	nome	email	crm	especialidade	logradouro	bairro	cep	complemento	numero	uf	cidade	telefone
1	Rodrigo Ferreira	rodrigo.ferreira@voll.med	123456	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia	
5	Sérgio Mallandro	sergio.malandro@voll.med	654321	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia	



POST localhost:8080/medicos — Send

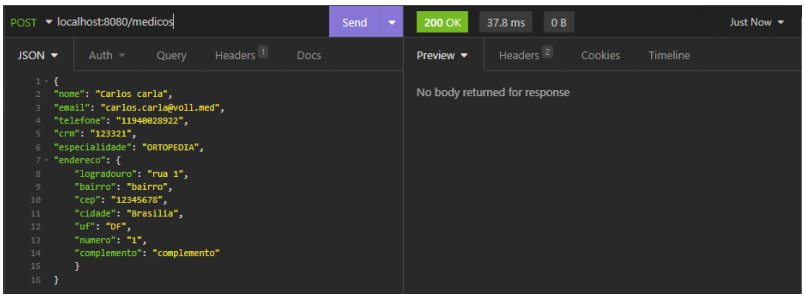
400 Bad Request | 215 ms | 5.9 KB | Just Now

JSON | Auth | Query | Headers 1 | Docs

```
{
  "nome": "Carlos carla",
  "email": "carlos.carla@voll.med",
  "crm": "123321",
  "especialidade": "ORTOPEDIA",
  "endereco": {
      "logradouro": "rua 1",
      "bairro": "bairro",
      "cep": "12345678",
      "cidade": "Brasilia",
      "uf": "DF",
      "numero": "1",
      "complemento": "complemento"
      }
}
```

Preview | Headers 4 | Cookies | Timeline

```
{
  "timestamp": "2022-12-14T23:30:50.788+00:00",
  "status": 400,
  "error": "Bad Request",
  "trace":
  "org.springframework.web.bind.MethodArgumentNotValidException:
  Validation failed for argument [0] in public void
  med.voll.api.controller.MedicoController.cadastrar(med.voll.api.medi
  co.DadosCadastroMedico): [Field error in object
  'dadosCadastroMedico' on field 'telefone': rejected value [null];
  codes
  [NotBlank.dadosCadastroMedico.telefone,NotBlank.telefone,NotBlank.ja
  va.lang.String,NotBlank]; arguments
  [org.springframework.context.support.DefaultMessageSourceResolvable:
  codes [dadosCadastroMedico.telefone,telefone]; arguments []; default
  message [telefone]]; default message [não deve estar em branco]]
  \r\n\tat
```



POST localhost:8080/medicos — Send

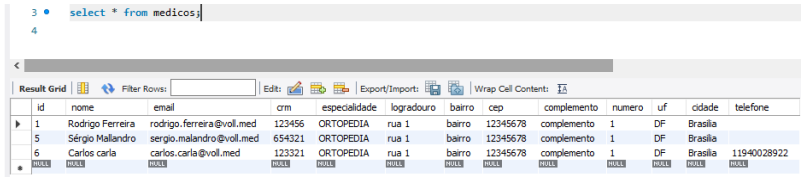
200 OK | 37.8 ms | 0 B | Just Now

JSON | Auth | Query | Headers 1 | Docs

```
{
  "nome": "Carlos carla",
  "email": "carlos.carla@voll.med",
  "telefone": "11940028922",
  "crm": "123321",
  "especialidade": "ORTOPEDIA",
  "endereco": {
      "logradouro": "rua 1",
      "bairro": "bairro",
      "cep": "12345678",
      "cidade": "Brasilia",
      "uf": "DF",
      "numero": "1",
      "complemento": "complemento"
      }
}
```

Preview | Headers 2 | Cookies | Timeline

No body returned for response

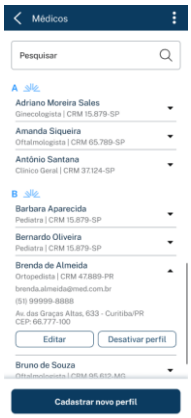


```
select * from medicos;
```

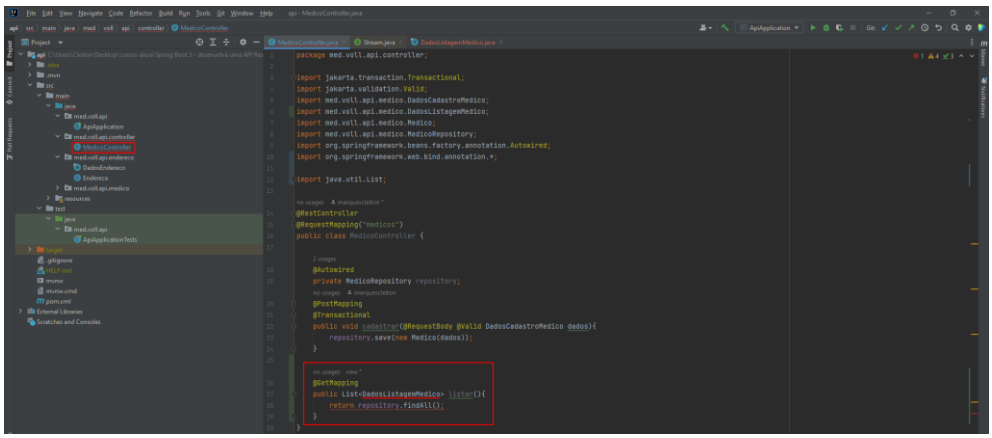
Result Grid

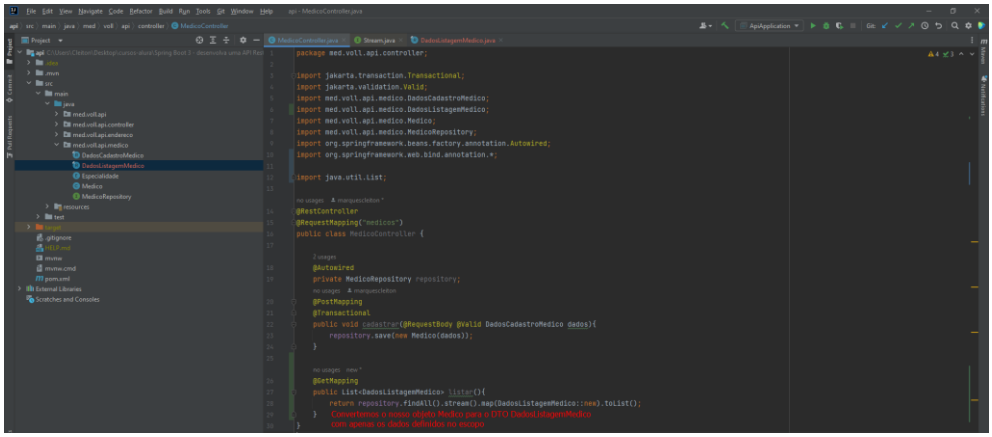
id	nome	email	crm	especialidade	logradouro	bairro	cep	complemento	numero	uf	cidade	telefone
1	Rodrigo Ferreira	rodrigo.ferreira@voll.med	123456	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia	
5	Sérgio Mallandro	sergio.malandro@voll.med	654321	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia	
6	Carlos carla	carlos.carla@voll.med	123321	ORTOPEDIA	rua 1	bairro	12345678	complemento	1	DF	Brasilia	11940028922

4 – Requisição Get

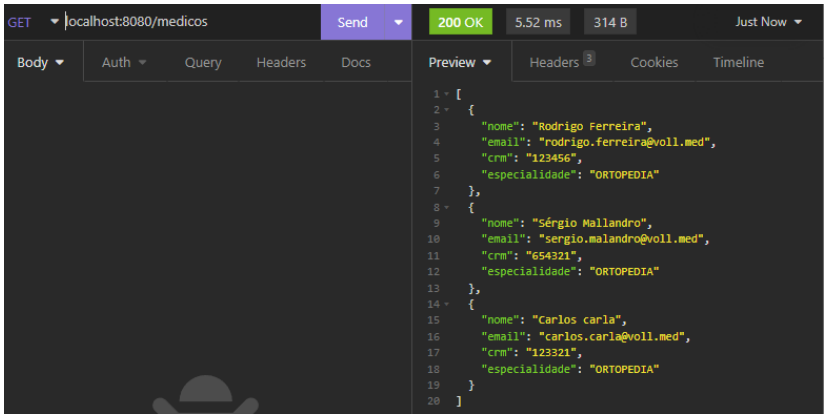


Criando DTO de listagem





Testando a listagem

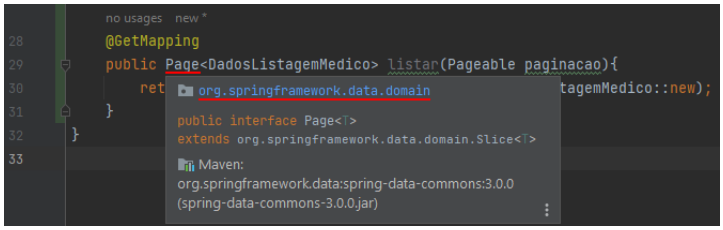


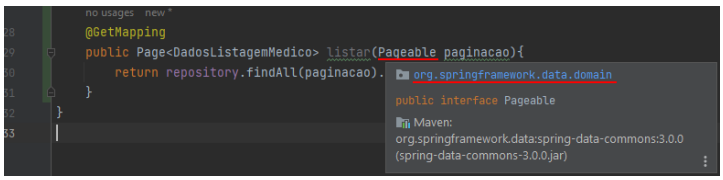
Obs: Não haverá conflito porque os verbos das duas requisições da mesma URL são diferentes.

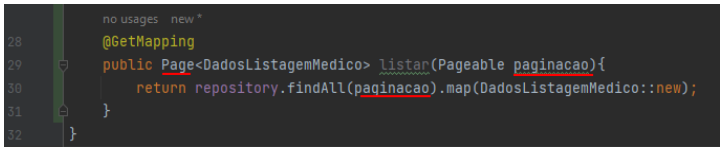
Paginação

Alterando Controller

Aqui iremos utilizar a biblioteca de paginação do Spring





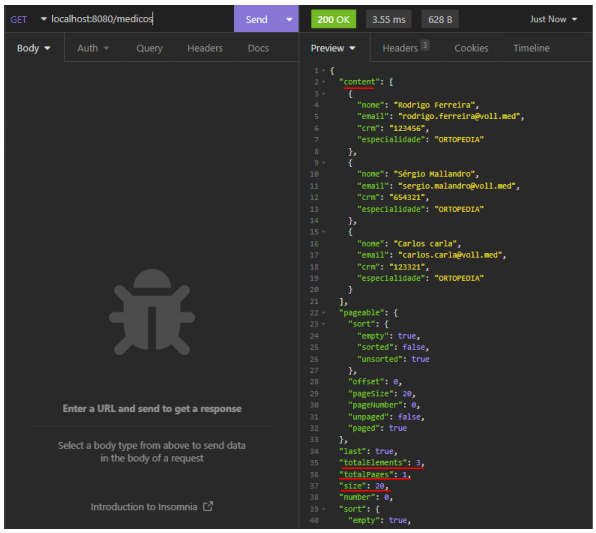


Como fazer a requisição:

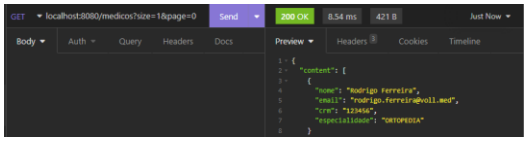
http://localhost:8080/medicos?size=1?page=0

size: quantidade de registros por página

page: navega entre as páginas



Por **padrão** o spring define 20 elementos por página, sendo que a página começa no índice zero.

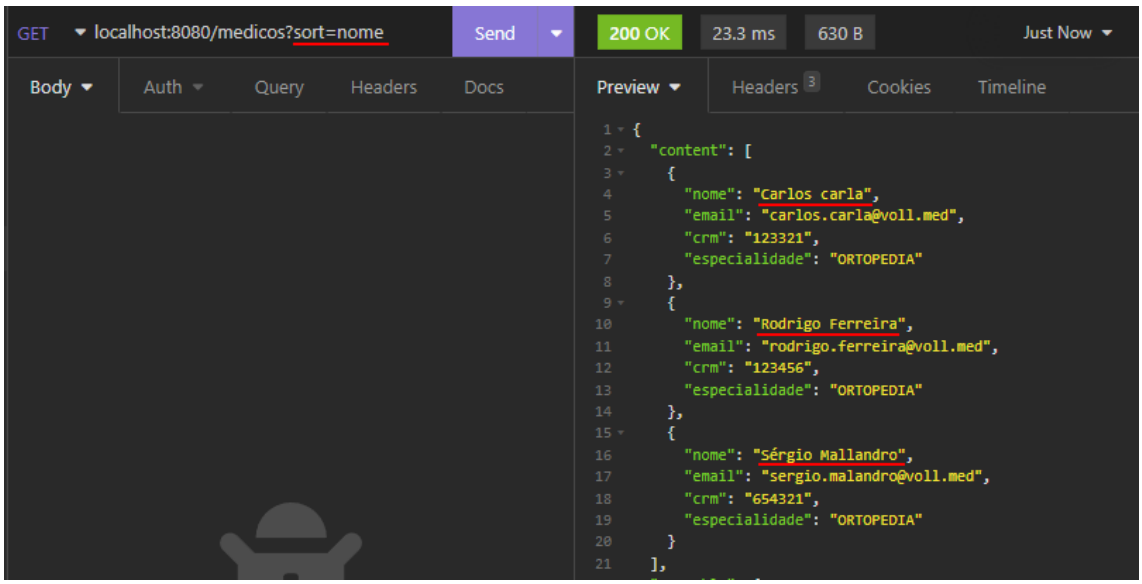


Caso queiramos mudar o padrão, basta adicionar a anotação: @PageableDefoult

Ordenação

Para mudar a ordenação, também usaremos um parâmetro na URL, chamado *sort*.

Sendo *nome* o atributo da entidade médico, mas poderia ser pelo crm, especialidade...



Ordenando Decrescente:

GET localhost:8080/medicos?sort=crm,desc Send 200 OK 4.1 ms 630 B Just Now

Body Auth Query Headers Docs Preview Headers 3 Cookies Timeline


```
1  {
2    "content": [
3      {
4        "nome": "Sérgio Mallandro",
5        "email": "sergio.malandro@voll.med",
6        "crm": "654321",
7        "especialidade": "ORTOPEDIA"
8      },
9      {
10       "nome": "Rodrigo Ferreira",
11       "email": "rodrigo.ferreira@voll.med",
12       "crm": "123456",
13       "especialidade": "ORTOPEDIA"
14     },
15     {
16       "nome": "Carlos carla",
17       "email": "carlos.carla@voll.med",
18       "crm": "123321",
19       "especialidade": "ORTOPEDIA"
20     }
```


Se passarmos lista(@PageableDefault(size = 10, sort = {"nome"}), isso significa que, caso não passemos parâmetros na URL, no Insomnia, o novo padrão será a exibição de 10 resultados por página, ordenados a partir do nome.

Plus: Mostrar queries de banco de dados executadas

application.properties

spring.jpa.show-sql=true

Com isso, os SQLs disparados no banco de dados serão impressos.

spring.jpa.properties.hibernate.format_sql=true

Formata as queries para melhor exibição, já que por padrão são exibidas em um única linha.

```
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 limit ?,
        ?
```

Alteração dos nomes dos parâmetros de paginação

O Spring Boot permite que os nomes de tais parâmetros sejam modificados via configuração no arquivo application.properties

spring.data.web.pageable.page-parameter=pagina

spring.data.web.pageable.size-parameter=tamanho

spring.data.web.sort.sort-parameter=ordem

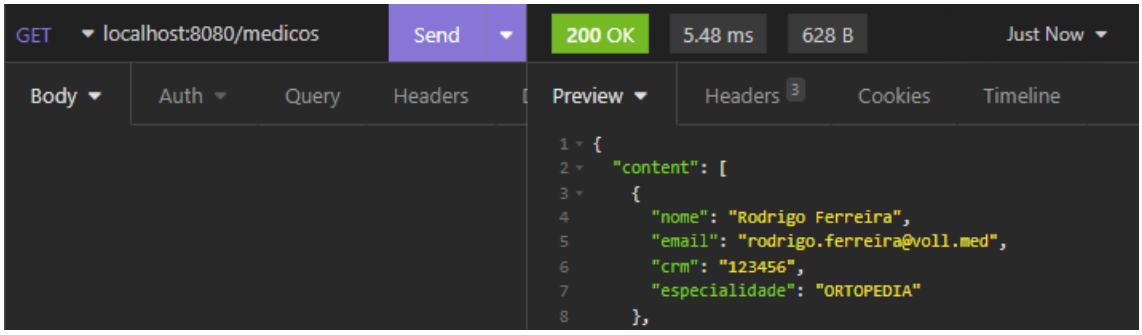
http://localhost:8080/medicos?tamanho=5&pagina=1&ordem=email,desc

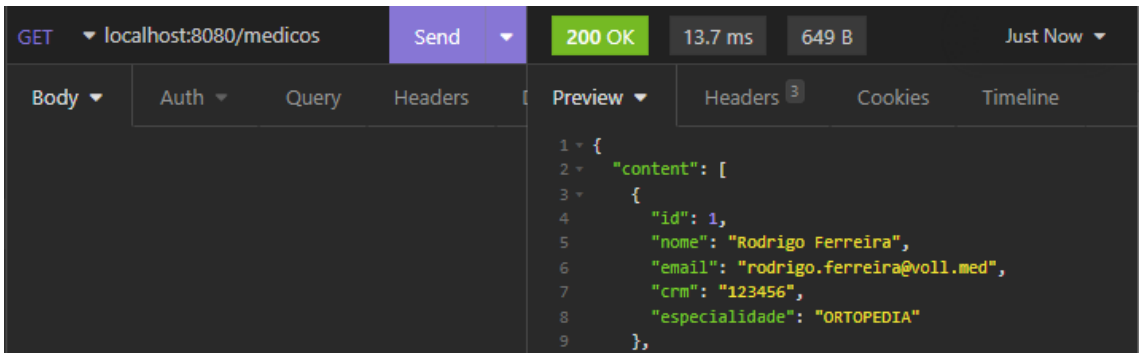
5 - PUT e DELETE

Criando a requisição PUT

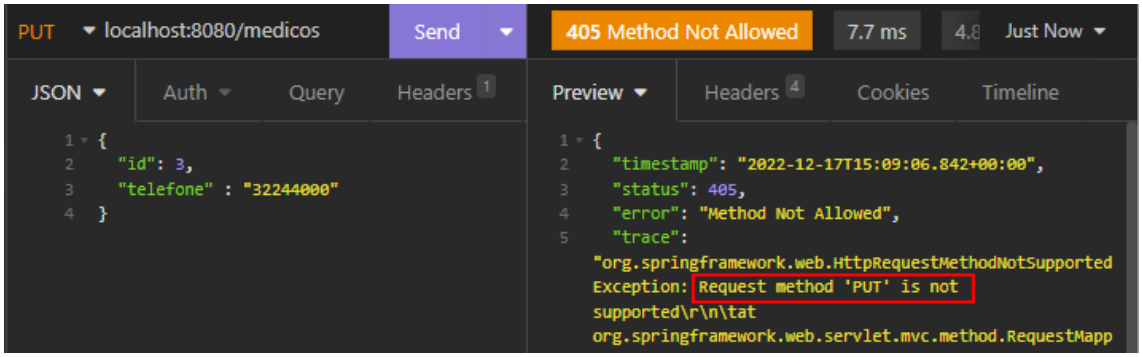
Quando o usuário apertar em Editar ele será redirecionado para um formulário de edição.

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.





Com a busca já trazendo os médicos com seus respectivos IDs, podemos implementar o método de atualização de médico.



Criando o método de atualização de médico:

Criando DTO para definir os dados que serão atualizados:

Obs: como a regra de negócios diz quais dados devem, ser atualizados, no DTO basta não colocar os atributos que NÃO devem ser alterados.

Consulta do médico que será atualizado:

Para fazer a consulta usamos o nosso repository buscando pelo ID. Logo após usaremos um método de atualizar as informações dentro da entidade Medico, mas como não existe, teremos que criar.

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

no usages  new *
@GetMapping
public Page<DadosListagemMedico> listar(@PageableDefault(size=10) Pageable paginacao){
    return repository.findAll(paginacao).map(DadosListagemMedico::new);
}

no usages  new *
@PutMapping  Anotação do verbo PUT
@Transactional  Como vamos atualizar os dados é bom assegurar a transação
public void atualizar(@RequestBody @Valid DadosAtualizacaoMedico dados){  Criar um novo DTO de atualização
    Medico medico = repository.getReferenceById(dados.id());  Buscar o médico pelo ID
    medico.atualizarInformacoes(dados);  Atualizar médico passando os dados vindos do DTO
}
}
```

Criação do métodos de atualização de informações em Médico:

```
private Endereco endereco;

1 usage  marquescleiton
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());
}

1 usage  new *
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.atualizarEndereco(dados.endereco());
    }
}
```

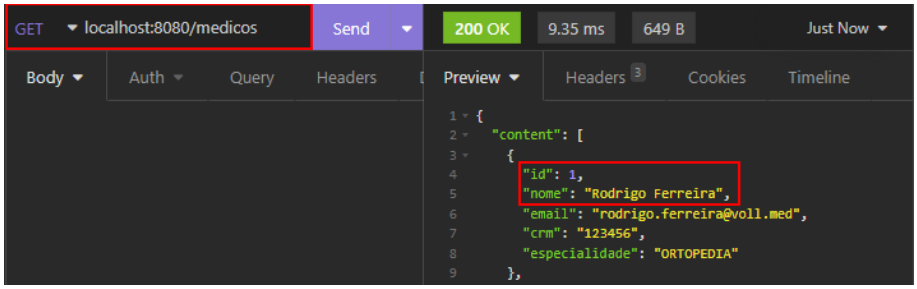
Como em endereço não é diferente, e recebemos um DTO de endereço, precisamos criar um método dentro da entidade Endereço e atualizar os dados.

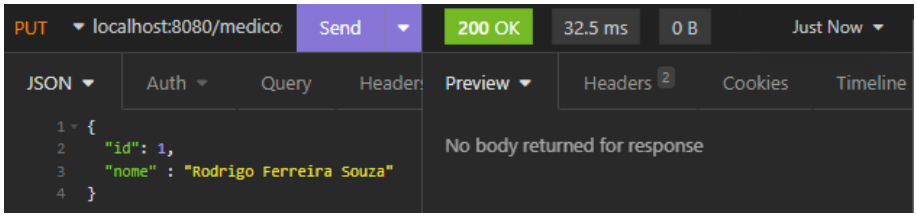
```
public void atualizarInformacoes(DadosAtualizacaoMedico dados) {

    if(dados.nome() != null) {
        this.nome = dados.nome();
    }
    if(dados.telefone() != null){
        this.telefone = dados.telefone();
    }
    if(dados.endereco() != null) {
        this.endereco.atualizarEndereco(dados.endereco());
    }
}
}
```

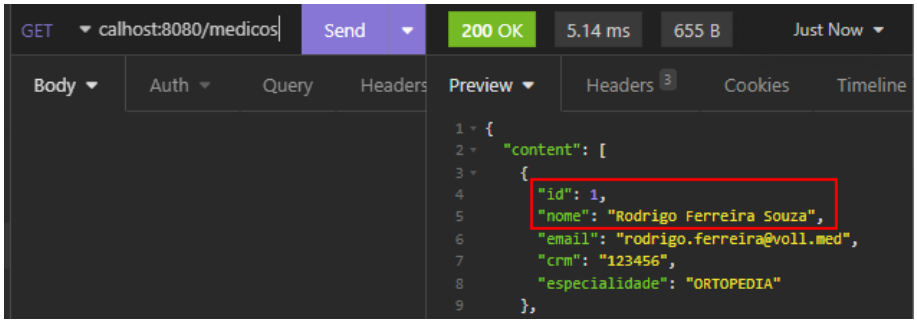
```java
        this.numero = dados.numero();
        this.complemento = dados.complemento();
    }

    1 usage  new *
    public void atualizarEndereco(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();
        }
    }
```

Testando a requisição

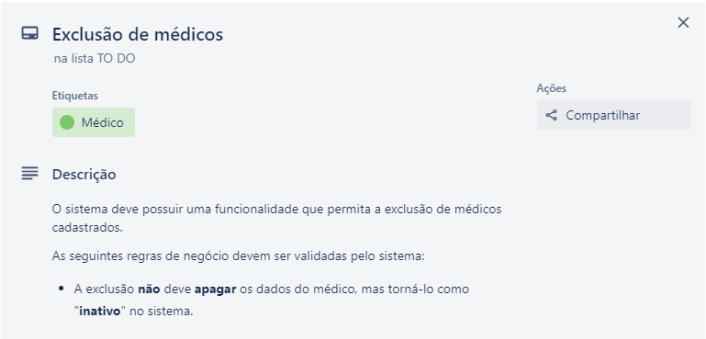
Buscando um médico, pegando o ID e atualizando os dados:







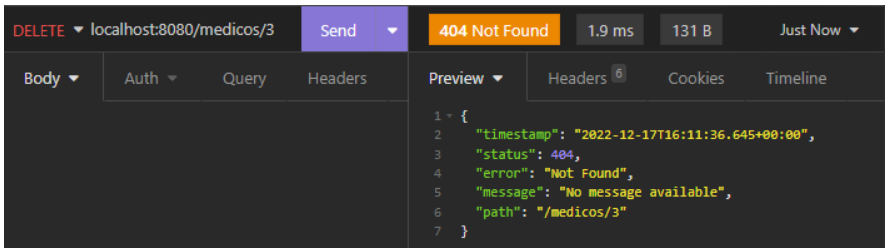
Criando a requisição DELETE

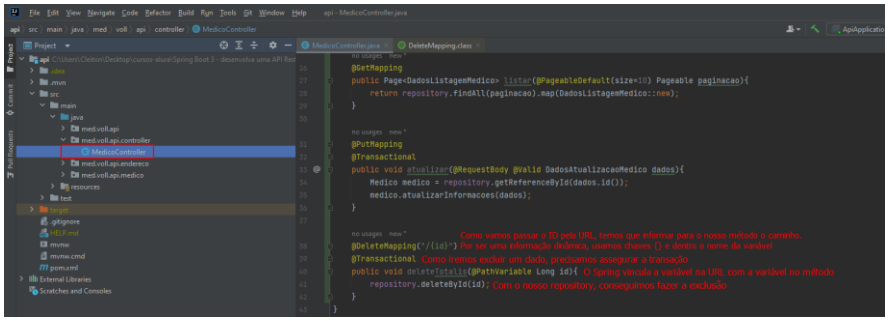


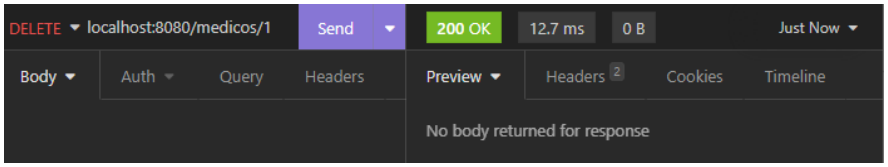
Em desativar, vamos criar uma exclusão lógica, e não de fato excluir do sistema. Assim teremos todo o histórico do médico registrado e evitaremos problemas com chaves estrangeiras.

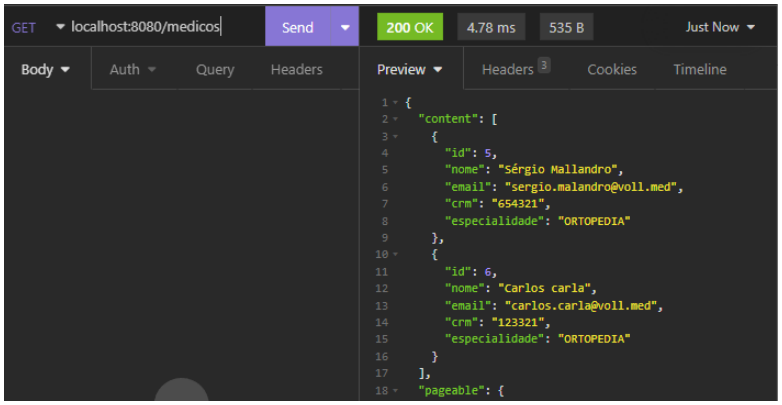
Exclusão no banco de dados:

Para questões de aprendizagem, vamos ver como excluir um dado no banco de dados.







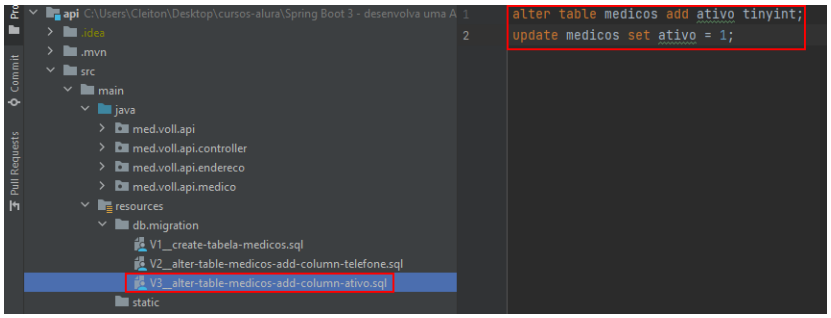


Exclusão lógica:

Para isso, precisaremos de um novo campo em nossa entidade Medico, ativo, que terá como valor verdadeiro ou falso.

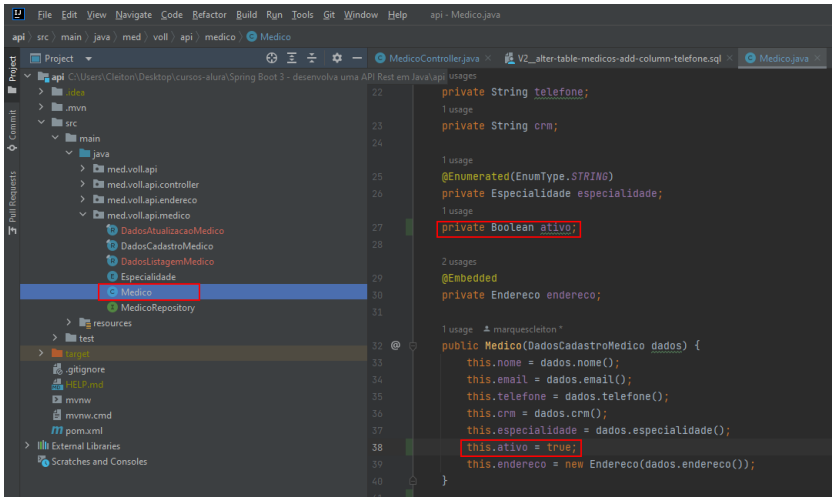
Nova Migration incluindo nova coluna:

Criar uma nova coluna chamada ativo e já setar TODOS os dados já existentes na base como ativo.



Alterar entidade Medico incluindo novo campo

Quando um novo médico for criado, já setaremos ativo como verdadeiro.



Incluir exclusão lógica no Controller

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

no usages  new *
@DeleteMapping("/{id}")
@Transactional
public void deleteTotalis(@PathVariable Long id){
    repository.deleteById(id);
}

no usages  new *
@DeleteMapping("exclusaologica/{id}")
@Transactional
public void deleteLogico(@PathVariable Long id){
    Medico medico = repository.getReferenceById(id);
    medico.excluir();
}
}
```

Criando método de exclusão na entidade Medico

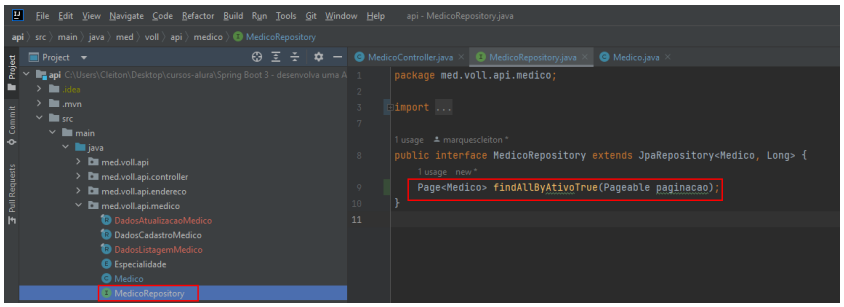
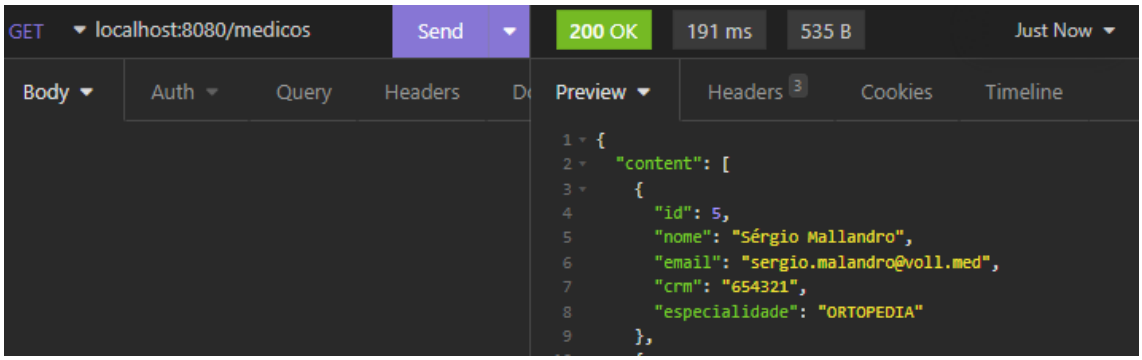
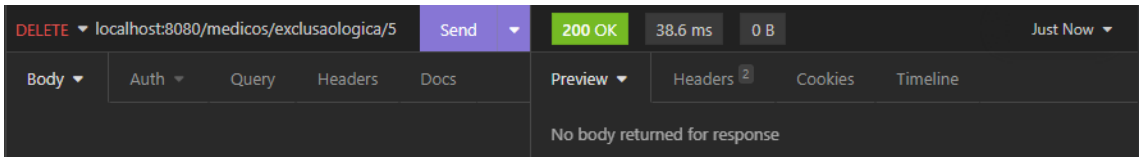
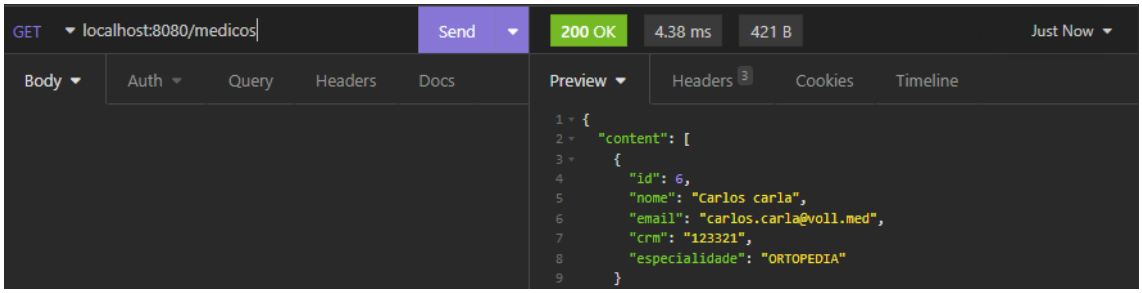
```
}

1 usage  new *
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.atualizarEndereco(dados.endereco());
    }
}

1 usage  new *
public void excluir() {
    this.ativo = false;
}
```

Alterar método listar para buscar apenas médicos ativos

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

    5 usages
    @Autowired
    private MedicoRepository repository;
    no usages  marquescleiton
    @PostMapping
    @Transactional
    public void cadastrar(@RequestBody @Valid DadosCadastroMedico dados) { repository.save(new Medico(dados)); }

    no usages  new *
    @GetMapping
    public Page<DadosListagemMedico> listar(@PageableDefault(size=10) Pageable paginacao){
        return repository.findAllByAtivoTrue(paginacao).map(DadosListagemMedico::new);
    }
```

Criando método no repository

O Spring tem um esquema de nomenclatura de métodos que abstrai para nós algumas pesquisas. Ele consegue identificar os nomes dos atributos e pelo o que estamos pesquisando.



Testando exclusão lógica