



PRW3 - Programação para a WEB III

API REST com Spring (finalização)

Spring Security parte 1

Conteúdo 10



Avaliação da disciplina



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

AVALIAÇÃO DA DISCIPLINA

- Questionário em

<https://forms.gle/Rj6Mp8cKp1PdQ58X6>

- Completamente anônimo!
- Respostas enviadas para o Coordenador do Curso
- **RESPONDAM AGORA, POR FAVOR!!**



Um pouco de polêmica...



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

10 verdades difíceis de engolir sobre ser Engenheiro de Software

1 Faculdade é teoria, trabalho é sobrevivência

Se formar em TI é tipo aprender a nadar na teoria e, no primeiro emprego, você se joga direto no mar sem boia. Boa sorte pra não se afogar.

2 Adeus projetos perfeitinhos do zero

Esquece essa vida de criar do nada. A real é pegar sistemas velhos, cheios de bugs, e tentar não fazer o Frankenstein digital morrer.



3 Código limpo? Só seu colega liga

Ninguém vai dar parabéns pelo código organizado. A galera só quer saber se a função roda sem bug – nem que seja "colada com fita adesiva".

4 Sim, tem gente perdida na TI também

Você acha que só gênio trabalha com software? Pior que não. Tem muito colega que vai testar sua paciência.

5 Prepare-se para muitas reuniões (inúteis)

Você acha que vai passar o dia todo programando? Não! Parte do seu trabalho é sobreviver a reuniões que só servem para provar que alguém "tem trabalho".

6 “Pra quando você entrega isso?” Sempre

Seu chefe vai pedir estimativas o tempo todo. Regra de ouro: prometa pouco e entregue antes.

7 Bugs são os vilões eternos

Não importa o quanto você testa, bugs são como pernilongos: sempre aparecem. E quando não são seus, são da biblioteca que você usou.

8 A incerteza será sua “amiga” constante

Novas tecnologias, prazos surpresas e mudanças de projeto... A única certeza é que você nunca vai saber de tudo.

9 Desligar do trabalho?

É quase impossível não pensar no bug daquele código enquanto toma banho ou quando está na praia. O truque é aprender a desconectar antes que seu cérebro queime.

10 Soft skills valem mais que código

Saber programar é essencial, mas quem se dá bem é quem também sabe se comunicar, trabalhar em equipe e não surtar na pressão.

Ser engenheiro de software é mais que só escrever código – é como um jogo de sobrevivência com bugs, reuniões e incertezas. Mas se você aprender a lidar com tudo isso, pode até achar divertido.

Continuando...



Lembrando o que foi feito até agora...

- Criação do projeto
 - site start.spring.io
 - dependências
 - dev tools / lombok / spring web
- Hello World
- POST : enviando dados para criar um recurso
 - classes embedded
- Inserção de novas dependências no projeto
 - site start.spring.io
 - BD H2 / Flyway migrations / Validation / Spring Data JPA
- Configuração do BD no projeto



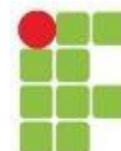
Lembrando o que foi feito até agora...

- DTO : Record
- DAO : JPA Repository
- Migration com Flyway
- Implementação de recursos de validação de dados
- GET
 - Listagem (todos os dados)
 - Listagem (dados parciais)
 - Paginação, ordenação
- GET : dados detalhados de uma única entidade
- PUT : atualizar



Lembrando o que foi feito até agora...

- DELETE : apagar entidades
 - deleção física
 - deleção lógica
- Criando um novo método de recuperação de dados (com filtro) no Repository
- Códigos de respostas das requisições
 - ResponseEntity
 - criando um link para um recurso recém-criado



“Endpoints” criados até agora:

- POST /medicos
 - cadastrar()
 - criar médico
- GET /medicos
 - listar()
 - retorna listagem de todos os dados de todos os médicos
- GET /medicos/algunsdados
 - listarAlgunsDados()
 - retorna listagem de médicos, só alguns dados, paginada



“Endpoints” criados até agora:

- GET /medicos/{id}
 - getMedicoById()
 - retorna todos os dados do médico ID
- PUT /medicos
 - atualizar()
 - altera os dados de um médico
- DELETE /medicos/{id}
 - excluir()
 - realiza a deleção lógica do médico ID



Revisitando alguns pontos
levantados por um colega de vocês...



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

equals() e hashCode() gerados pelo Lombok:

*"Using **@EqualsAndHashCode** for JPA entities is not recommended.
It can cause severe performance and memory consumption issues."*

"Usar **@EqualsAndHashCode em entidades JPA não é recomendado.
Isso pode causar sérios problemas de desempenho e consumo de
memória."**



Ao usar a anotação `@EqualsAndHashCode` do Lombok em entidades JPA, é necessário **muito cuidado**, pois ela influencia diretamente o comportamento das entidades em coleções (`Set` , `Map` , etc.) e na comparação de objetos. Aqui vão os principais pontos:

⚠ Problemas comuns

1. Uso do `id` gerado automaticamente (ex: `@GeneratedValue`):

- Se o `id` for incluído no `equals` e `hashCode`, ele ainda estará `null` antes de o objeto ser persistido. Isso causa problemas, por exemplo:
 - Dois objetos recém-criados e ainda não salvos no banco terão `id = null`, portanto `equals` pode considerar que são iguais mesmo sendo diferentes.
 - Se usados num `Set`, um objeto pode "sumir" ou ser sobreescrito após ser salvo (quando o `id` muda de `null` para um valor).

2. Uso de relações bidirecionais:

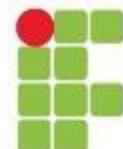
- Se o `@EqualsAndHashCode` incluir atributos de relação (`@ManyToOne` , `@OneToOne` , etc.), pode haver **loop infinito** durante a chamada de `equals` / `hashCode`.



Esse *warning* indica um problema real: usar `@EqualsAndHashCode` (do Lombok) em entidades JPA pode causar **problemas sérios de desempenho e consumo de memória**, especialmente quando o `equals` e o `hashCode` incluem relações como `@OneToMany` , `@ManyToOne` , etc.

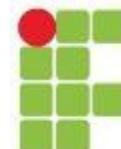
Motivo:

- 1. Ciclos de referência:** entidades JPA frequentemente têm relacionamentos bidirecionais. Se o `equals` ou `hashCode` acessarem campos que apontam para outras entidades, pode ocorrer **recursão infinita**.
- 2. Persistência preguiçosa (lazy loading):** acessar campos que ainda não foram carregados do banco pode acionar *fetches* indesejados.
- 3. Identidade vs. igualdade:** JPA compara entidades pela identidade (`==`) enquanto o `equals` gerado pelo Lombok pode comparar campos, o que conflita com o modelo do Hibernate.



✓ Boas práticas

- Use somente atributos imutáveis ou identificadores naturais (ex: CPF, email, username) para `equals` e `hashCode`.
- Evite incluir o campo `id` se ele for gerado automaticamente.
- Use `@EqualsAndHashCode(onlyExplicitlyIncluded = true)` e inclua manualmente os campos com `@EqualsAndHashCode.Include`.



Um parênteses: UUID



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

UUID

UUID (Universally Unique Identifier) é um identificador único de 128 bits, comumente representado por uma string de 32 caracteres hexadecimais, divididos em 5 grupos, separados por hífens. Ele é usado para identificar de forma única um objeto ou entidade em sistemas distribuídos, sem depender de um banco de dados centralizado.

O formato típico de um UUID é:

 Copiar

 Editar

```
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

Onde cada `x` é um dígito hexadecimal (0-9 ou a-f).

Principais características do UUID:

- Unicidade Global:** O principal benefício de um UUID é garantir que o valor gerado seja único, mesmo em sistemas diferentes e sem necessidade de um controle centralizado. Isso é útil em sistemas distribuídos, onde você não quer que duas instâncias diferentes gerem o mesmo identificador.
- Independência de Contexto:** Você pode gerar um UUID em qualquer lugar (em um banco de dados, em um código ou em uma máquina) e ele será único, sem a necessidade de um servidor central para garantir sua singularidade.
- Formato fixo:** O UUID tem um comprimento fixo de 128 bits (16 bytes), geralmente representado em 36 caracteres no formato string.

Alternativa segura: campo `uuid` imutável

java

Copiar

Editar

```
@Entity  
@EqualsAndHashCode(onlyExplicitlyIncluded = true)  
public class Pessoa {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @Column(nullable = false, unique = true, updatable = false)  
    @EqualsAndHashCode.Include  
    private final String uuid = UUID.randomUUID().toString();  
  
    // demais campos, construtores, getters, etc.  
}
```

Vantagens:

- O `uuid` é definido no momento da criação da entidade.
- Permite comparar entidades com `equals` mesmo antes de persistir.
- Evita problemas com `null id`, ciclos e lazy loading.

Essa abordagem é considerada segura e compatível com JPA, Lombok e Hibernate.

Vamos implementar essa solução no projeto.

Migration para criar o campo UUID:



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

V4__add_uuid_to_medicos.sql ×

```
1
2  -- V2__add_uuid_to_medicos.sql
3
4  -- Adiciona a coluna 'uuid' com valor default usando RANDOM_UUID()
5  ALTER TABLE medicos
6  ADD COLUMN uuid VARCHAR(36) NOT NULL DEFAULT RANDOM_UUID();
7
8  -- Atualiza registros existentes, atribuindo um UUID único para cada linha
9  UPDATE medicos
10 SET uuid = RANDOM_UUID()
11 WHERE uuid IS NULL;
12
13 -- Garante unicidade
14 ALTER TABLE medicos
15 ADD CONSTRAINT uk_medicos_uuid UNIQUE (uuid);
```

```
-- Adiciona a coluna 'uuid' com valor default usando RANDOM_UUID()
ALTER TABLE medicos
ADD COLUMN uuid VARCHAR(36) NOT NULL DEFAULT RANDOM_UUID();

-- Atualiza registros existentes, atribuindo um UUID único para cada linha
UPDATE medicos
SET uuid = RANDOM_UUID()
WHERE uuid IS NULL;

-- Garante unicidade
ALTER TABLE medicos
ADD CONSTRAINT uk_medicos_uuid UNIQUE (uuid);
```

Ao rodar o projeto,
a migration é processada,
e o Banco atualizado:



H2 Console

localhost:8080/h2-console/login.do?jsessionid=aeedd2533da625a5f4ef34568e4a6754

Auto commit | Max rows: 1000 | Auto complete | Off | Auto select On | ?

jdbc:h2:file:./DATA/pw3api

MEDICOS

- ID
- NOME
- EMAIL
- CRM
- ESPECIALIDADE
- LOGRADOURO
- BAIRRO
- CEP
- COMPLEMENTO
- NUMERO
- UF
- CIDADE
- TELEFONE
- ATIVO
- UUID

Indexes

flyway_schema_history

INFORMATION_SCHEMA

Users

H2 2.2.224 (2023-09-17)

SQL statement:

```
SELECT * FROM MEDICOS
```

ENTO	NUMERO	UF	CIDADE	TELEFONE	ATIVO	UUID
	null	SP	Ibaté	16992482222	0	0bdf8b62-297e-45ce-8bcb-8774e0575758
	333	BB	BBBB	16992483333	0	b3a68ee5-5ff0-42d1-a123-8ea279087a76
	null	SP	Limeira	16992484444	1	36e32239-64d4-4504-8fee-ee0850ce0249
	555	SP	São Carlos	16992225599	1	f42a927a-d07d-4ba0-bb2f-a2a588b935d4



Nova classe Medico (trecho):

```
@Table(name = "medicos")
@Entity(name = "Medico")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@EqualsAndHashCode(onlyExplicitlyIncluded = true)
public class Medico {

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

    @EqualsAndHashCode.Include
    @Column(nullable = false, unique = true, updatable = false)
    private final String uuid = UUID.randomUUID().toString();

    private String nome;
    private String email;
    private String telefone;
    private String crm;
```



Nova classe Medico (trecho):

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

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

    @EqualsAndHashCode.Include
    @Column(nullable = false, unique = true, updatable = false)
    private final String uuid = UUID.randomUUID().toString();

    private String nome;
    private String email;
    private String telefone;
    private String crm;
```

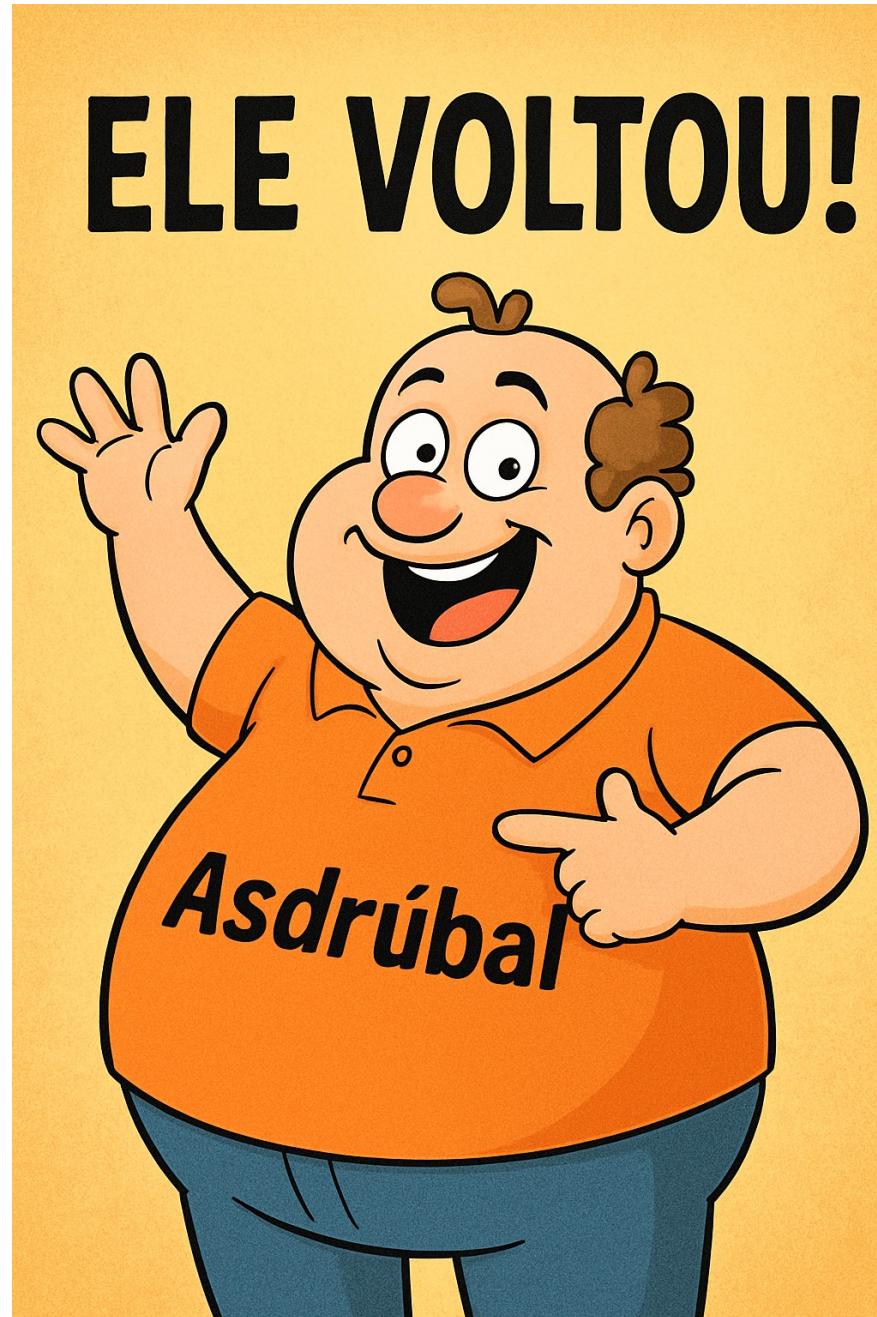


Testando...



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

Testando...



{

```
"nome": "Asdrubal Almeida",
"email": "asd@gmail.com",
"crm": "111111",
"telefone": "16992481111",
"especialidade": "ORTOPEDIA",
"endereco": {
    "logradouro": "rua blabla",
    "bairro": "santa felicia",
    "cep": "13563264",
    "cidade": "São Carlos",
    "uf": "SP",
    "numero": "50",
    "complemento": "casa 99"
}
```

}



POST



localhost:8080/medicos

Params

Authorization

Headers (8)

Body ●

Scripts

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON ▾

```
1  {
2      "nome": "Asdrubal Almeida",
3      "email": "asd@gmail.com",
4      "crm": "111111",
5      "telefone": "16992481111",
6      "especialidade": "ORTOPEDIA",
7      "endereco": {
8          "logradouro": "rua blabla",
9          "bairro": "santa felicia",
10         "cep": "13563264",
11         "cidade": "S\u00e3o Carlos",
12         "uf": "SP",
13         "numero": "50",
14         "complemento": "casa 99"
15     }
```



INSTITUTO FEDERAL DE
EDUCA\u00c7AO, CI\u00d4NCIA E TECNOLOGIA
S\u00e3O PAULO
C\u00e3mpus S\u00e3o Carlos

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	ATIVO	UUID
2	Lupita Pereira	lupy@gmail.com	222222	DERMATOLOGIA	RUA DOIS	BAIRRO DOIS	22222222	null	null	SP	Ibaté	16992482222	0	0bdf8b62-297e-45ce-8bcb-8774e0575758
3	Zoroastro Almeida	zorro@gmail.com	333333	CARDIOLOGIA	BBBB	BBBB	99999	BBB	333	BB	BBBB	16992483333	0	b3a68ee5-5ff0-42d1-a123-8ea279087a76
5	Anita Garibaldi	ag@gmail.com	444444	ORTOPEDIA	RUA QUATRO	BAIRRO QUATRO	44444444	null	null	SP	Limeira	16992484444	1	36e32239-64d4-4504-8fee-ee0850ce0249
6	Dilermando Reis	dil@gmail.com	667777	ORTOPEDIA	RUA DIL	BAIRRO DIL	12345678	DILDILDIL	555	SP	São Carlos	16992225599	1	f42a927a-d07d-4ba0-bb2f-a2a588b935d4
7	Asdrubal Almeida	asd@gmail.com	111111	ORTOPEDIA	RUA UM	BAIRRO UM	11111111	null	null	SP	Araraquara	16992481111	1	2eceeab4-d7b9-4b34-9d06-d41e7bbe8608

(5 rows, 6 ms)



Resumo:

- **UUID ajuda a reduzir problemas de desempenho** porque é uma chave única e imutável, ao contrário do `id`, que pode ser gerado automaticamente pelo banco de dados (e pode ter a sobrecarga de se referir a outros objetos).
- **A recomendação geral** ainda é evitar o uso de `@EqualsAndHashCode` em entidades JPA (mesmo com UUID), especialmente se houver relações complexas entre entidades. Uma alternativa seria a implementação manual de `equals()` e `hashCode()`, considerando apenas os atributos essenciais, como `id` ou `uuid`, e não as associações completas.

Portanto, o uso de UUID pode atenuar o problema, mas a abordagem recomendada é ter cautela ao usar `@EqualsAndHashCode` em entidades JPA e avaliar a necessidade de implementações personalizadas de `equals()` e `hashCode()`.



Referências extras:

<https://web.archive.org/web/20171211235806/http://www.onjava.com/pub/a/onjava/2006/09/13/dont-let-hibernate-steal-your-identity.html>

<https://vladmirhalcea.com/how-to-implement-equals-and-hashcode-using-the-jpa-entity-identifier/>

<https://jpa-buddy.com/blog/lombok-and-jpa-what-may-go-wrong/>



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

Sobre injeção de dependência:

Uso de **@Autowired** no atributo para injetar automaticamente uma instância da classe no atributo.



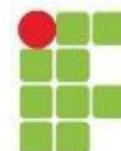
Formas de fazer injeção de dependência:

- Injeção via construtor
- Injeção via Anotação @Autowired no Construtor
- Injeção Direta com @Autowired no Campo (a que usamos)



Formas de fazer injeção de dependência:

- Injeção via construtor
- Injeção via Anotação @Autowired no Construtor
- Injeção Direta com @Autowired no Campo (a que usamos)
 - Menos recomendada 😐



1. Injeção via Construtor (Recomendada)

java

Copiar

Editar

```
@RestController  
@RequestMapping("/medicos")  
public class MedicoController {  
  
    private final MedicoRepository repository;  
  
    public MedicoController(MedicoRepository repository) {  
        this.repository = repository;  
    }  
  
    // seus métodos aqui  
}
```

Essa forma é a mais recomendada pois favorece imutabilidade e facilita testes unitários.



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

2. Injeção via Anotação `@Autowired` no Construtor

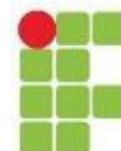
java

Copiar

Editar

```
@RestController  
@RequestMapping("/medicos")  
public class MedicoController {  
  
    private final MedicoRepository repository;  
  
    @Autowired  
    public MedicoController(MedicoRepository repository) {  
        this.repository = repository;  
    }  
  
    // seus métodos aqui  
}
```

Funciona igual à anterior, mas a anotação é opcional desde o Spring 4.3, se a classe tiver só um construtor.



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

3. Injeção Direta com @Autowired no Campo

java

 Copiar  Editar

```
@RestController  
@RequestMapping("/medicos")  
public class MedicoController {  
  
    @Autowired  
    private MedicoRepository repository;  
  
    // seus métodos aqui  
}
```

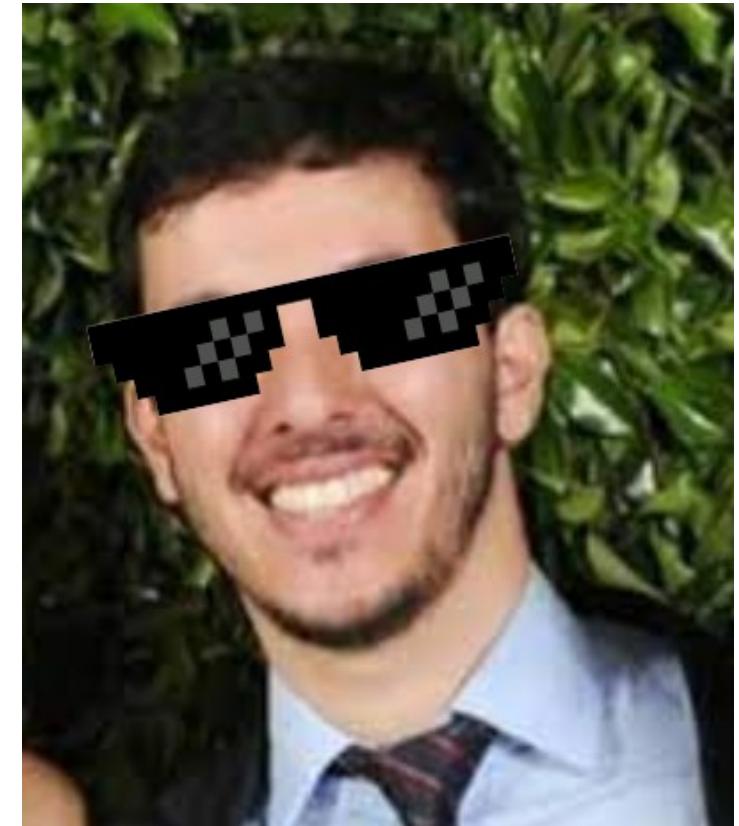
Essa forma é funcional, mas menos recomendada porque dificulta testes e reduz a clareza das dependências.



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

"Spring recomenda o uso de um construtor que tenha como parâmetros todas as dependências necessárias, porque a prioridade dele é chamar esse cara.

Se você usa injeção via atributo, ele só vai fazer o 'binding' (ligação) quando você precisar do atributo. Nesta forma, por exemplo, ao criar um service, vai dar new na classe sem nenhuma dependência. Só quando for usar o atributo (com @Autowired) ele vai tentar fazer o binding. Se não encontrar, não vai fazer a injeção, e você vai tomar um null pointer exception.



Quando você usa construtor, na hora que está criando a dependência, ele já precisa passar as dependências no construtor. Quando o Spring sobe, ele sobe já montando tudo. Não seria possível criar um service, sem o repository dele ter sido passado como parâmetro no construtor.

É melhor o construtor porque ele garante que o objeto sendo criado já é criado com as dependências certas. Via atributo não, ele cria com estado padrão, e lá na frente, conforme a necessidade, vai tentar injetar."

Revisitando nosso controller...



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

MedicoController.java

```
1 package br.edu.ifsp.prw3.api20251c.controller;  
2  
3 import ...  
17  
18 @RestController  
19 @RequestMapping("medicos")  
20 public class MedicoController {  
21  
22     @Autowired  
23     private MedicoRepository repository;  
24  
25     @PostMapping  
26     @Transactional  
27     @  
28     @  
29     @  
30     @  
31     @  
32     @  
33     @  
34     @  
35     @  
36     @  
37     @  
38     @  
39     @  
40     @  
41     @  
42     @  
43     @  
44     @  
45     @  
46     @  
47     @GetMapping  
48     public ResponseEntity listar() {...}  
52
```



Fica...

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

    //@Autowired
    //private MedicoRepository repository;

    private final MedicoRepository repository;

    public MedicoController(MedicoRepository repository) {
        this.repository = repository;
    }

    ...
}
```



Para mais detalhes...

<https://www.baeldung.com/java-spring-field-injection-cons>



Why Is Field Injection Not Recommended?

Last updated: May 11, 2024



Written by:
Ana Peterlić



Reviewed by:
Saajan Nagendra

Spring +

Autowired

Spring Core Basics

Spring DI

1. Overview

When we run the code analysis tool in the IDE, it may issue the "*Field injection is not recommended*" warning for fields with the `@Autowired` annotation.

In this tutorial, we'll explore why field injection isn't recommended and what alternative approaches we can use.

Obrigado Matheus!!
(ADS 2025.1)



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



Calendário da reta final!

- 29.out Desenvolvimento de uma API REST (parte 4). Introdução Spring Security
- 5.nov Spring Security parte 2.
- 12.nov Spring Security parte 3.
- 19.nov Dia para trab. na Avaliação 4. **Entrega até 23:59.**
- 26.nov Demonstração: Microservices com Spring (vídeo-aula, s/ controle presença)
- 3.dez Revisão. Preparação para IFA...
- 10.dez Revisão. Preparação para IFA...

**Possivelmente teremos mais um conteúdo (fora de avaliação),
especificamente sobre Spring Data.
Talvez na próxima semana, ou dia 26,
empurrando Microservices pra semana seguinte.**



Tratamento de erros



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

Vamos tentar apagar os dados
de um ID que não existe:

Versões passadas

The screenshot shows a Postman request for a DELETE operation on the URL `http://localhost:8080/medicos/7645`. The response status is **500 Internal Server Error**, with a timestamp of `2023-10-19T17:46:01.889+00:00`, status code `500`, error message `Internal Server Error`, and a detailed traceback. A red arrow points from the error status in the header to the error message in the body.

DELETE http://localhost:8080/medicos/7645

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (4) Test Results 500 Internal Server Error 27 ms 7.18 KB

Pretty Raw Preview Visualize JSON

```
1 {  
2   "timestamp": "2023-10-19T17:46:01.889+00:00",  
3   "status": 500,  
4   "error": "Internal Server Error",  
5   "trace": "jakarta.persistence.EntityNotFoundException: Unable to find br.edu.ifsp.pw3.  
api.medico.Medico with id 7645\n\tat org.hibernate.jpa.boot.internal.  
EntityManagerFactoryBuilderImpl$JpaEntityNotFoundDelegate.handleEntityNotFound  
(EntityManagerFactoryBuilderImpl.java:183)\n\tat org.hibernate.proxy.  
AbstractLazyInitializer.checkTargetState(AbstractLazyInitializer.java:286)\n\tat org.  
hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:175)  
\n\tat org.hibernate.proxy.AbstractLazyInitializer.getImplementation  
(AbstractLazyInitializer.java:310)\n\tat org.hibernate.proxy.pojo.bytebuddy.
```

{

```
"timestamp": "2023-10-19T11:47:28.399+00:00",
"status": 500,
"error": "Internal Server Error",
"trace": "jakarta.persistence.EntityNotFoundException: Unable to
find br.edu.ifsp.pw3.api.medico.Medico with id 7645\n\tat org.hibernate
.jpa.boot.internal.EntityManagerFactoryBuilderImpl$JpaEntityNotFoundDe
legate.handleEntityNotFound(EntityManagerFactoryBuilderImpl.java:183) \
n\tat org.hibernate.proxy.AbstractLazyInitializer.checkTargetState
(AbstractLazyInitializer.java:286)\n\tat org.hibernate.proxy.Abstract
LazyInitializer.initialize(AbstractLazyInitializer.java:175)\n\tat
org.hibernate.proxy.AbstractLazyInitializer.getImplementation(Abstract
LazyInitializer.java:310)\n\tat org.hibernate.proxy.pojo.bytebuddy.
ByteBuddyInterceptor.intercept(ByteBuddyInterceptor.java:44)\n\tat
*
*
*
*
org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThr
ead.java:61)\n\tat
java.base/java.lang.Thread.run(Thread.java:1623)\n",
"message": "Unable to find br.edu.ifsp.pw3.api.medico.Medico with
id 7645",
"path": "/medicos/7645"
}
```

zilhões de linhas!!!!

Este semestre:

DELETE ▼ | http://localhost:8080/medicos/7645 Send ▼

Params Auth Headers (8) Body ● Scripts Tests Settings Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body ▼ ⌚ 500 Internal Server Error • 147 ms • 267 B • 🌐 | ✉️ Save Response ...

{ } JSON ▼ ▶ Preview ⌚ Visualize ▼ ≡ ✖ 🔍 🔗

```
1 {  
2   "timestamp": "2025-05-07T12:41:18.910+00:00",  
3   "status": 500,  
4   "error": "Internal Server Error",  
5   "path": "/medicos/7645"  
6 }
```

No log de execução...

```
-  
m1_0.uuid  
from  
medicos m1_0  
where  
m1_0.id=?
```

```
2025-05-07T09:41:18.901-03:00 ERROR 12309 --- [api20251c] [nio-8080-exec-2] o.a.c.c.C.[.[][.][dispatcherServlet] : Servlet.ser
```

```
jakarta.persistence.EntityNotFoundException Create breakpoint : Unable to find br.edu.ifsp.prw3.api20251c.medico.Medico with id 7645  
at org.hibernate.jpa.boot.internal.EntityManagerFactoryBuilderImpl$JpaEntityNotFoundDelegate.handleEntityNotFound(EntityManage  
at org.hibernate.proxy.AbstractLazyInitializer.checkTargetState(AbstractLazyInitializer.java:304) ~[hibernate-core-6.6.11.Final  
at org.hibernate.proxy.AbstractLazyInitializer.initialize(AbstractLazyInitializer.java:188) ~[hibernate-core-6.6.11.Final.jar:  
at org.hibernate.proxy.AbstractLazyInitializer.getImplementation(AbstractLazyInitializer.java:328) ~[hibernate-core-6.6.11.Fin  
at org.hibernate.proxy.pojo.bytebuddy.ByteBuddyInterceptor.intercept(ByteBuddyInterceptor.java:44) ~[hibernate-core-6.6.11.Fin  
at org.hibernate.proxy.ProxyConfiguration$InterceptorDispatcher.intercept(ProxyConfiguration.java:102) ~[hibernate-core-6.6.11
```



O código de retorno HTTP 500, conhecido como "Internal Server Error" (Erro Interno do Servidor), é um dos códigos de status HTTP que o servidor envia como parte de uma resposta a uma solicitação HTTP. Esse código é usado para indicar que ocorreu um erro interno no servidor ao processar a solicitação do cliente. Ele sinaliza que algo deu errado durante o processamento da solicitação, e o servidor não conseguiu cumprir a requisição do cliente devido a um problema interno.

Aqui estão alguns pontos importantes sobre o código de retorno HTTP 500:

- 1. Erro no Lado do Servidor:** O código 500 é usado exclusivamente para indicar erros no lado do servidor. Isso significa que o problema está relacionado a problemas no servidor ou na aplicação que está sendo executada no servidor.
- 2. Natureza Genérica:** O código 500 é genérico e não fornece muitos detalhes sobre a natureza específica do erro. Ele serve como um indicativo de que algo não está funcionando corretamente, mas não especifica o motivo exato.

Não seria melhor devolver um erro
“404 not found” ??

Vamos analisar o código
que trata essa requisição...

```
@DeleteMapping("/{id}")
@Transactional
public ResponseEntity excluir(@PathVariable Long id) {

    Medico medico = repository.getReferenceById(id);
    // Chama método na classe Medico que coloca false no atributo 'ativo':
    medico.excluir();

    // Lembrando, não precisamos regravar o objeto no BD.
    // A JPA automaticamente atualiza o objeto no BD.
    return ResponseEntity.noContent().build();
}
```

- Quando não encontra o ‘id’, o método lança uma “**EntityNotFoundException**” exception.
- Por padrão, exceções não tratadas no código são interpretadas pelo Spring como **Erro 500**.
- Poderíamos tratar com “try...catch”.
 - Teríamos que “poluir” nosso código com esse código extra, em todo lugar que usarmos o método getReferenceById().
- Queremos que essa exception retorne o erro 404, em qualquer lugar que o método **getReferenceById** for utilizado.

vale
a pena
ver
de novo



vale
a pena
ver de
novo

Spring AOP

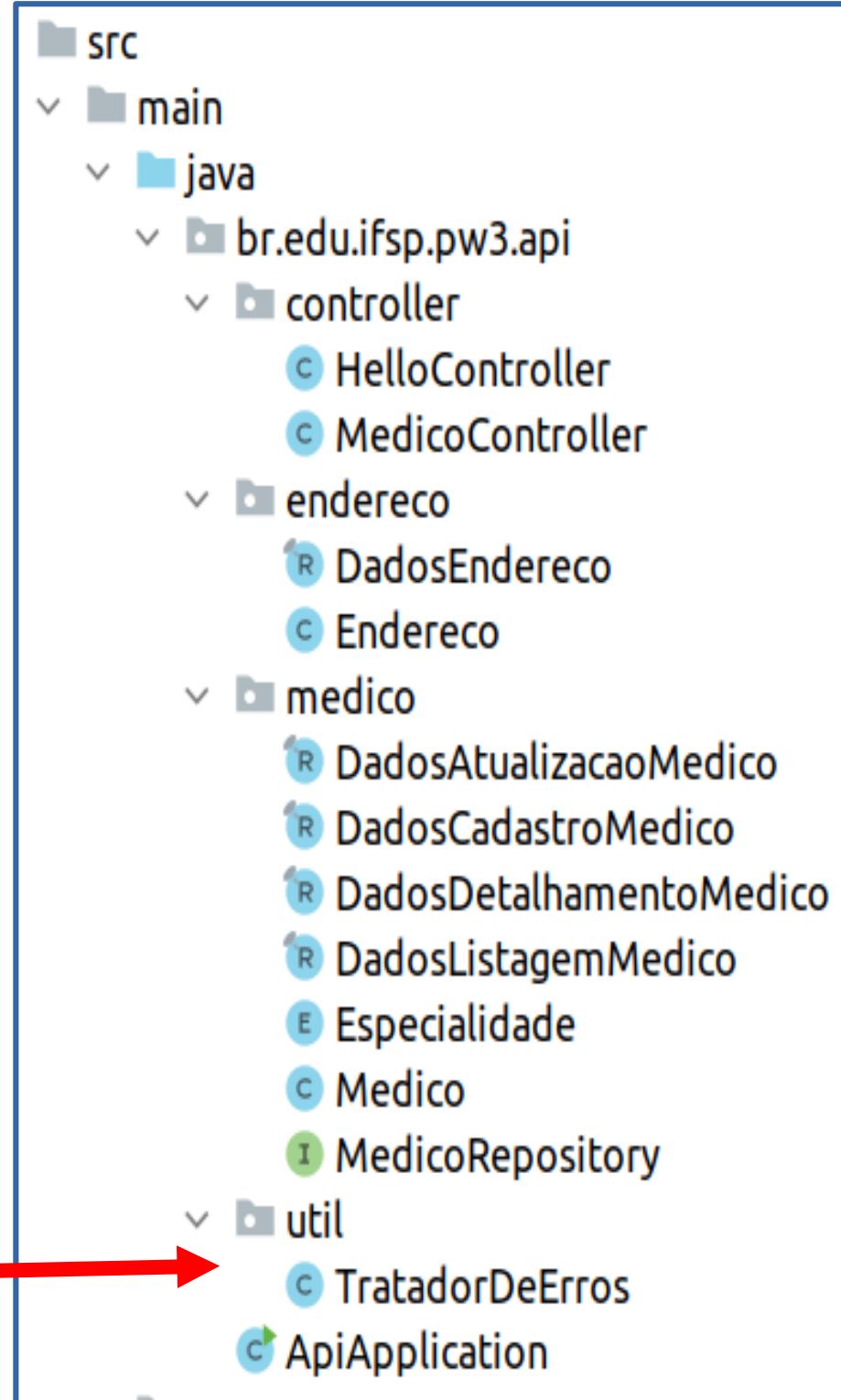
- No contexto do framework Spring, o "Spring AOP" (Aspect-Oriented Programming) é um módulo que permite a implementação da programação orientada a aspectos em um aplicativo.
- A programação orientada a aspectos é um paradigma que ajuda a **separar preocupações transversais do código principal de um programa**, como logging, segurança, tratamento de exceções e outras funcionalidades que se aplicam a várias partes do código.
- O Spring AOP fornece uma maneira de modularizar essas preocupações e **aplicar comportamentos específicos (conhecidos como "aspects") sem precisar incorporá-los no código principal**. Isso resulta em código limpo, organizado e focado.

Um exemplo simples

- Uma operação sensível, que só pode ser executada por usuário autorizado e autenticado.
- O código da operação sensível fica misturado com o código que valida as questões de segurança.
- Essa situação pode ocorrer em vários locais da aplicação.
- O Spring AOP vai permitir **organizar essas operações em um único local**, podendo ser **aplicado em várias partes da aplicação**.

```
public void sensitiveOperation() {  
    // check if user is authenticated and has the correct role  
    if ( ... ) {  
  
        // do sensitive operation  
  
    } else {  
        // raise an unauthorized error  
        // log failed attempt  
        // redirect user to login page  
    }  
}
```

Pacote: util
Classe: TratadorDeErros



```
import jakarta.persistence.EntityNotFoundException;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice
public class TratadorDeErros {

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity tratarErro404() {

        return ResponseEntity.notFound().build();
    }
}
```

```
@RestControllerAdvice
public class TratadorDeErros {

    @ExceptionHandler(EntityNotFoundException.class)
    public ResponseEntity tratarErro404() {

        return ResponseEntity.notFound().build();

    }

}
```

A anotação `@RestControllerAdvice` é uma anotação do Spring Framework que desempenha um papel importante no tratamento de exceções em aplicativos Spring Boot que utilizam controladores REST. Ela é usada em classes que desejam fornecer tratamento global para exceções lançadas por controladores que retornam respostas RESTful.

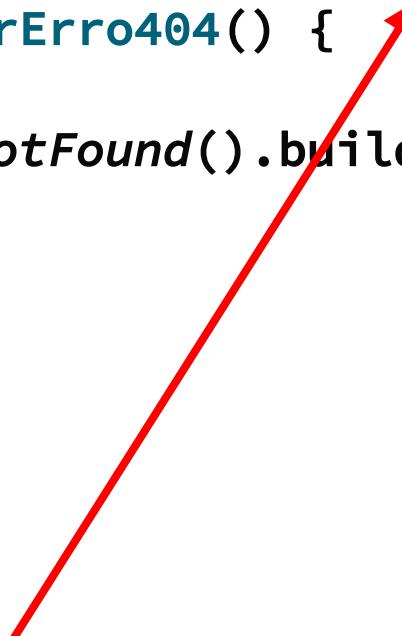
A principal função do `@RestControllerAdvice` é permitir que você centralize o tratamento de exceções em um único local, em vez de lidar com exceções em cada controlador individualmente. Isso é especialmente útil em aplicativos RESTful, onde é comum retornar respostas no formato JSON, XML ou outro formato de dados, em vez de páginas HTML.

```
@RestControllerAdvice  
public class TratadorDeErros {  
  
    @ExceptionHandler(EntityNotFoundException.class)  
    public ResponseEntity<Entity> tratarErro404() {  
  
        return ResponseEntity.notFound().build();  
    }  
  
}
```

A anotação `@ExceptionHandler` é usada no Spring Framework para tratar exceções de maneira específica em controladores (controllers). Ela permite que você defina métodos em seus controladores ou em classes anotadas com `@ControllerAdvice` que lidarão com exceções específicas lançadas durante o processamento de solicitações HTTP.

A principal finalidade da anotação `@ExceptionHandler` é personalizar o tratamento de exceções de forma granular, fornecendo respostas específicas quando ocorrem exceções. Isso é particularmente útil em aplicativos Spring, onde você deseja fornecer respostas amigáveis e significativas em caso de erros, em vez de simplesmente permitir que a exceção seja propagada até o tratamento padrão.

```
@RestControllerAdvice  
public class TratadorDeErros {  
  
    @ExceptionHandler(EntityNotFoundException.class)  
    public ResponseEntity tratarErro404() {  
  
        return ResponseEntity.notFound().build();  
  
    }  
  
}
```



Dentro dos parênteses logo após a anotação `@ExceptionHandler`, você deve especificar o tipo de exceção que você deseja capturar e tratar. Isso significa que você coloca a classe da exceção que deseja manipular.

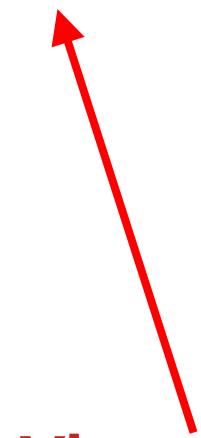


A classe EntityNotFoundException é uma exceção lançada pelo framework Java Persistence API (JPA) quando uma entidade não é encontrada na base de dados. Essa exceção é subclasse da classe `PersistenceException`, que é lançada por qualquer problema relacionado ao mapeamento de entidades no JPA.

A classe `EntityNotFoundException` pode ser lançada em várias situações, como:

- Quando um objeto de entidade é buscado pelo ID e o objeto não existe na base de dados;
- Quando um objeto de entidade é atualizado e o objeto não existe na base de dados;
- Quando um objeto de entidade é excluído e o objeto não existe na base de dados.

```
@RestControllerAdvice  
public class TratadorDeErros {  
  
    @ExceptionHandler(EntityNotFoundException.class)  
    public ResponseEntity tratarErro404() {  
  
        return ResponseEntity.notFound().build();  
    }  
  
}
```



Visto na aula passada.

Testando a requisição novamente...

DELETE



http://localhost:8080/medicos/7645

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Body

Cookies

Headers (4)

Test Results



404 Not Found

Pretty

Raw

Preview

Visualize

Text ▾



1

DELETE



http://localhost:8080/medicos/7645

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Body

Cookies

Headers (4)

Test Results



404 Not Found

Pretty

Raw

Preview

Visualize

Text ▾



1



Erros de validação

Vamos relembrar a criação de um médico no BD,
e o erro gerado quando os requisitos dos dados
não são respeitados
(problema na validação).

Dados do médico a criar (corretos):

```
{  
  "nome": "Roberval Taylor",  
  "email": "robtay@gmail.com",  
  "crm": "989898",  
  "telefone": "16992223344",  
  "especialidade": "DERMATOLOGIA",  
  "endereco": {  
    "logradouro": "RUA BLABLABLABLA",  
    "bairro": "BAIRRO BLABLABLA",  
    "cep": "88666444",  
    "numero": "9999999",  
    "complemento": "BLOCO 9 CASA 999",  
    "cidade": "New York",  
    "uf": "NY"  
  }  
}
```

Lembrando as validações...

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

Dados do médico a criar (errados!):

```
1  {
2    "nome": "Roberval Taylor",
3    "email": "robtay@gmail",
4    "crm": "989898111",
5    "telefone": "10992223344",
6    "especialidade": "DERMATOLOGIA",
7    "endereco": {
8      "logradouro": "RUA BLABLABLABLA",
9      "bairro": "BAIRRO BLABLABLA",
10     "cep": "88666444",
11     "numero": "9999999",
12     "complemento": "BLOCO 9 CASA 999",
13     "cidade": "New York",
14     "uf": "NY"
15   }
16 }
```

Teremos 4 erros
de validação.

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

POST



localhost:8080/medicos

Enviando...

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

```
1  {
2      "email": "robtay@gmail",
3      "crm": "989898111",
4      "especialidade": "DERMATOLOGIA",
5      "endereco": {
6          "logradouro": "RUA BLABLABLA",
7          "bairro": "BAIRRO BLABLABLA",
8          "cep": "88666444",
9          "numero": "9999999",
10         "complemento": "BLOCO 9 CASA 999",
11         "cidade": "New York",
12         "uf": "NY"
13     }
14 }
```

POST



localhost:8080/medicos

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

Body

Cookies

Headers (4)

Test Results



Status: 400 Bad Request

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2     "timestamp": "2023-10-19T18:32:52.388+00:00",  
3     "status": 400,  
4     "error": "Bad Request",  
5     "message": "Validation failed for object='dadosCadastroMedico'. Error count: 5",  
6     "errors": [  
7         {  
8             "codes": [  
9                 "NotBlank.dadosCadastroMedico.telefone",  
10                "NotBlank.telefone",  
11                "NotBlank.java.lang.String",  
12                "NotBlank"  
13            ],  
14            "arguments": [  
15                {  
16                    "codes": [  
17                        "dadosCadastroMedico.telefone",  
18                        "telefone"  
19                    ],  
20                    "arguments": null,  
21                    "defaultMessage": "telefone",  
22                }  
23            ]  
24        }  
25    ]  
26}  
27
```



O erro está ok!!



O erro 400 Bad Request (Solicitação Inválida) é um código de status HTTP que indica que o servidor não conseguiu entender ou processar a solicitação feita pelo cliente, porque a solicitação estava malformada, incompleta ou continha dados inválidos. Em outras palavras, o servidor não pode ou não irá processar a solicitação devido a problemas com a própria solicitação, e não por um erro interno do servidor.

Aqui estão algumas situações comuns em que o erro 400 Bad Request pode ocorrer:

- 
1. **Sintaxe inválida:** A solicitação HTTP não está formatada corretamente. Isso pode incluir erros de digitação, cabeçalhos malformados ou falta de informações obrigatórias.
 2. **Parâmetros inválidos:** Quando a solicitação contém parâmetros que são inválidos, fora dos limites ou em um formato não aceitável para o servidor.
 3. **Payload (corpo da solicitação) inválido:** Quando os dados enviados no corpo da solicitação (por exemplo, em solicitações POST, PUT ou PATCH) não estão em um formato aceitável ou não passam pela validação.

O código de erro está ok,
mas o corpo da resposta...

Novamente, a coisa da mudança de versão.

Versões anteriores:



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

POST

▼

localhost:8080/medicos

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

S

Body

Cookies

Headers (4)

Test Results

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2     "timestamp": "2023-10-19T18:32:52.388+00:00",  
3     "status": 400,  
4     "error": "Bad Request",  
5     "message": "Validation failed for object='dadosCadastroMedico'. Error count: 5",  
6     "errors": [  
7         {  
8             "codes": [  
9                 "NotBlank.dadosCadastroMedico.telefone",  
10                "NotBlank.telefone",  
11                "NotBlank.java.lang.String",  
12                "NotBlank"  
13            ],  
14            "arguments": [  
15                {  
16                    "codes": [  
17                        "dadosCadastroMedico.telefone",  
18                        "telefone"  
19                    ],  
20                    "arguments": null,  
21                    "defaultMessage": "telefone",  
22                }  
23            ]  
24        }  
25    ]  
26 }
```



Muito complexo!!!
Qual o erro, afinal de contas?!?

```
{  
  "timestamp": "2024-05-  
07T16:03:21.033+00:00",  
  "status": 400,  
  "error": "Bad Request",  
  "message": "Validation failed for  
object='dadosCadastroMedico'.  
Error count: 4",  
  "errors": [  
    {  
      "codes": [  
        "NotBlank.dadosCadastroMedico.nome"  
e,  
        "NotBlank.nome",  
        "NotBlank.java.lang.String",  
        "NotBlank"  
      ],  
      "arguments": [  
        {  
          "codes": [  
            "dadosCadastroMedico.email",  
            "email"  
          ],  
          "arguments": null,  
          "defaultMessage": "email",  
          "code": "email"  
        },  
        [],  
        {  
          "codes": [  
            ".*"  
          ],  
          "defaultMessage": ".*",  
          "arguments": null  
        },  
        {  
          "codes": [  
            "dadosCadastroMedico.email",  
            "Email"  
          ],  
          "arguments": [  
            {  
              "codes": [  
                "Pattern.dadosCadastroMedico.crm",  
                "Pattern.crm",  
                "Pattern.java.lang.String",  
                "Pattern"  
              ],  
              "arguments": [  
                {  
                  "codes": [  
                    "dadosCadastroMedico.crm",  
                    "crm"  
                  ],  
                  "arguments": null,  
                  "defaultMessage": "crm",  
                  "code": "crm"  
                },  
                [],  
                {  
                  "codes": [  
                    "\d{4,6}"  
                  ],  
                  "defaultMessage": "\d{4,6}",  
                  "arguments": null  
                },  
                {  
                  "codes": [  
                    "dadosCadastroMedico",  
                    "dadosCadastroMedico",  
                    "field": "email",  
                    "rejectedValue": "robhay@gmail",  
                    "bindingFailure": false,  
                    "rejectedValue": "989898111",  
                    "bindingFailure": false,  
                    "code": "Pattern"  
                  ],  
                  "arguments": [  
                    {  
                      "codes": [  
                        "dadosCadastroMedico.telefone",  
                        "NotBlank.telefone",  
                        "NotBlank.java.lang.String",  
                        "NotBlank"  
                      ],  
                      "arguments": [  
                        {  
                          "codes": [  
                            "dadosCadastroMedico.telefone",  
                            "telefone"  
                          ],  
                          "arguments": null,  
                          "defaultMessage": "telefone",  
                          "code": "telefone"  
                        },  
                        {  
                          "defaultMessage": "must not be  
blank",  
                          "objectName":  
                          "dadosCadastroMedico",  
                          "field": "telefone",  
                          "rejectedValue": null,  
                          "bindingFailure": false,  
                          "code": "NotBlank"  
                        }  
                      ],  
                      "path": "/medicos"  
                    }  
                  ]  
                }  
              ]  
            ]  
          ]  
        }  
      ]  
    }  
  ]  
}
```



Neste semestre...



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

POST

localhost:8080/medicos

Send

Params

Authorization

Headers (8)

Body

Scripts

Tests

Settings

Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Beautify

```
1 {  
2   "email": "robtay@gmail.com",  
3   "crm": "989898111",  
4   "especialidade": "DERMATOLOGIA",  
5   "endereco": {
```

Body Cookies Headers (4) Test Results

🌐 400 Bad Request 138 ms 242 B

💾 Save as example ⚙️

Pretty

Raw

Preview

JSON



```
1 {  
2   "timestamp": "2024-10-30T18:13:07.087+00:00",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "path": "/medicos"  
6 }
```

**Agora simples demais (tem mais coisa no log de execução)...
A questão principal persiste:**

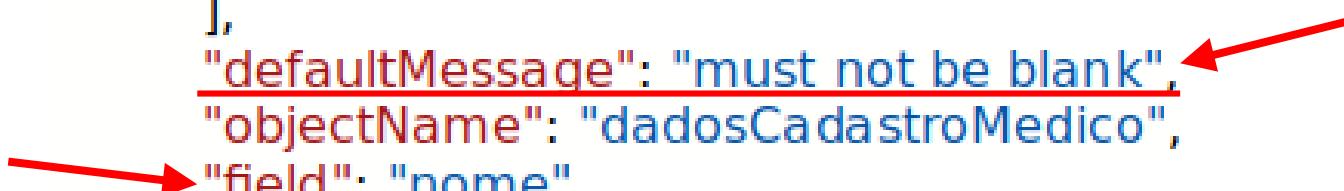
Qual o erro, afinal de contas?!?

Que tal retornar apenas
o campo que deu erro,
e qual foi o erro encontrado ?

Será que esses dados existem
nesse JSON gigantesco
que ele gera por default ?

Vamos ver os dados
de apenas um dos erros:

```
{  
    "codes": [  
        "NotBlank.dadosCadastroMedico.nome",  
        "NotBlank.nome",  
        "NotBlank.java.lang.String",  
        "NotBlank"  
    ],  
    "arguments": [  
        {  
            "codes": [  
                "dadosCadastroMedico.nome",  
                "nome"  
            ],  
            "arguments": null,  
            "defaultMessage": "nome",  
            "code": "nome"  
        }  
    ],  
    "defaultMessage": "must not be blank",  
    "objectName": "dadosCadastroMedico",  
    "field": "nome",  
    "rejectedValue": null,  
    "bindingFailure": false,  
    "code": "NotBlank"  
}
```



Precisamos converter
esse “**objeto** erro” cheio de informações,
em um **objeto mais simples (DTO)**
com menos informações
(só o campo e o texto do erro).

Então o que temos que fazer ?

- Criar um DTO apenas com os dados que queremos devolver:
 - campo que gerou o erro
 - mensagem de erro
- Obter a lista de todos os erros
- Criar uma nova lista, “**mapeando**” cada item da lista de erros original para um objeto do DTO que criamos acima, que será o elemento da nova lista
 - field → campo
 - defaultMessage → mensagem de erro
- Devolver a nova lista assim simplificada no corpo da resposta

```
"defaultMessage": "must not be blank",
"objectName": "dadosCadastroMedico",
"field": "nome",
"message": "Nome deve ser informado",
```

Lembrete... já fizemos isso antes!



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

Método do controller que cria a lista simplificada,
antes de implementarmos a paginação:

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

OBS:

Vamos criar o DTO com os dados
dos erros de validação
localmente aqui na classe TratadorDeErros,
já que só vamos usar aqui
(não vamos criar em um arquivo próprio).

Será um “**record interno**” na classe!

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getFieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}

private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```

Colocar DENTRO da classe “TratadorDeErros”.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getFieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}
```

```
private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400(MethodArgumentNotValidException ex) {

    var erros = ex.getFieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}

private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```



A classe `MethodArgumentNotValidException` é uma exceção que faz parte do ecossistema Spring Framework, em particular, da biblioteca Spring Boot. Ela é usada para representar exceções lançadas quando os argumentos de um método anotado com `@Valid` ou `@Validated` não passam na validação.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getFieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}

private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```

Parâmetro que recebe o objeto “exception”, contendo dados da exceção, como por exemplo os vários campos que não foram validados, e o motivo da não-validação.



```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getfieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}
```

```
private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```

A classe `FieldError` faz parte do ecossistema Spring Framework, em particular, da biblioteca Spring Validation (parte do pacote `org.springframework.validation`). Ela é usada para representar informações sobre erros de validação específicos em campos de um objeto. Em geral, a classe `FieldError` é utilizada em conjunto com a validação de dados, como em formulários da web, para identificar quais campos não atendem aos critérios de validação definidos.

construtor



Aqui estão os principais detalhes sobre a classe **`'FieldError'`**:

1. Atributos Principais:

- `'ObjectName'`: Representa o nome do objeto que contém o campo afetado pelo erro de validação.
- `'field'`: Representa o nome do campo específico que não passou na validação.
- `'rejectedValue'`: Contém o valor que causou o erro de validação no campo.
- `'defaultMessage'`: Armazena a mensagem de erro associada à validação que falhou. Esta mensagem geralmente é definida nas anotações de validação (por exemplo, `'@NotBlank(message = "Este campo não pode ser vazio")'`).
- Outros atributos, como `'codes'`, `'arguments'`, e `'bindingFailure'`, que fornecem informações adicionais sobre o erro.

Se quiser traduzir as mensagens,
ou criar mensagens customizadas.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getFieldErrors();
    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}

private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```





No código que você forneceu, `ex.getFieldErrors()` é usado para obter a lista de erros de validação que ocorreram quando `MethodArgumentNotValidException` foi lançada. Vou explicar com mais detalhes o que esse método faz:

`ex` é a instância de `MethodArgumentNotValidException` que foi passada como argumento para o método anotado com `@ExceptionHandler`.

`getFieldErrors()` é um método disponível na classe `MethodArgumentNotValidException`, que é usada quando os argumentos de um método anotado com `@Valid` ou `@Validated` não passam na validação. Essa exceção contém informações detalhadas sobre os erros de validação, e getFieldErrors() é usado para obter uma lista de objetos `FieldError`.

Cada objeto `FieldError` representa um erro de validação ocorrido em um campo específico. Um `FieldError` normalmente contém informações, como:

1. O nome do campo afetado.
2. O valor que causou o erro.
3. A mensagem de erro associada à validação que falhou.
4. Outras informações relevantes, como o código de erro e o objeto em que o erro ocorreu.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getFieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();
    return ResponseEntity.badRequest().body(lista);
}

private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```





vale
a pena

Na linha `var lista = erros.stream().map(DadosErroValidacao::new).toList();`, está sendo utilizado um fluxo (stream) para processar a lista de erros de validação (`erros`) e mapeá-la para uma lista de objetos `DadosErroValidacao`. Vamos examinar cada parte dessa linha em detalhes:

1. `erros.stream()`: `erros` é uma lista de objetos `FieldError` que representa os erros de validação. O método `stream()` é chamado nessa lista, convertendo-a em um fluxo (stream) de elementos.
2. `.map(DadosErroValidacao::new)`: O método `map` é usado para aplicar uma função a cada elemento do fluxo e obter um novo fluxo com os resultados. Neste caso, a função é `DadosErroValidacao::new`, que cria uma instância da classe `DadosErroValidacao` para cada `FieldError` na lista. Isso efetivamente mapeia os objetos `FieldError` para objetos `DadosErroValidacao`.
3. `.toList()`: O método `toList()` é usado para coletar os elementos do fluxo resultante em uma lista. Portanto, após o mapeamento, você obtém uma lista de objetos `DadosErroValidacao`.

No geral, a linha de código `var lista = erros.stream().map(DadosErroValidacao::new).toList();` cria uma lista de objetos `DadosErroValidacao` a partir da lista de erros de validação. Cada objeto `DadosErroValidacao` é criado a partir de um `FieldError` correspondente, e a lista resultante contém informações detalhadas sobre os erros de validação que ocorreram. Esta lista de objetos é então retornada como o corpo de uma resposta de erro 400 (Bad Request), fornecendo informações detalhadas sobre os erros de validação aos clientes da sua API.

```
@ExceptionHandler(MethodArgumentNotValidException.class)
public ResponseEntity tratarErro400( MethodArgumentNotValidException ex ) {

    var erros = ex.getFieldErrors();

    var lista = erros.stream().map(DadosErroValidacao::new).toList();

    return ResponseEntity.badRequest().body(lista);
}

private record DadosErroValidacao(String campo, String msgErro) {

    public DadosErroValidacao(FieldError erro) {
        this(erro.getField(), erro.getDefaultMessage());
    }
}
```





A linha `return ResponseEntity.badRequest().body(lista);` é a parte final do método `tratarErro400` e é responsável por criar e retornar uma resposta HTTP com status 400 (Bad Request) que contém a lista de erros de validação (a variável `lista`) no corpo da resposta. Vamos analisar cada parte dessa linha:

1. `ResponseEntity.badRequest()`: Aqui, `ResponseEntity` é uma classe do Spring Framework que representa uma resposta HTTP personalizada. O método `badRequest()` é um método estático da classe `ResponseEntity` que cria uma instância que representa uma resposta HTTP com o status 400 (Bad Request). Isso indica que algo no pedido do cliente não está correto, normalmente devido a erros de validação, como é o caso aqui.
2. `.body(lista)`: O método `body` é usado para definir o corpo da resposta HTTP. Neste caso, ele define o corpo da resposta como a lista de erros de validação (`lista`). A lista de erros de validação será serializada em formato JSON ou outro formato apropriado para ser enviada como parte do corpo da resposta HTTP. Isso permite que o cliente receba informações detalhadas sobre os erros que ocorreram durante a validação.



Refazendo a requisição...



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

POST



localhost:8080/medicos

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

Body

Cookies

Headers (4)

Test Results



Status: 400 Bad Request

Pretty

Raw

Preview

Visualize

JSON



```
1 [
2   {
3     "campo": "nome",
4     "msgErro": "must not be blank"
5   },
6   {
7     "campo": "telefone",
8     "msgErro": "must not be blank"
9   },
10  {
11    "campo": "crm",
12    "msgErro": "must match \\"\\d{4,6}\\\""
13  },
14  {
15    "campo": "email",
16    "msgErro": "must be a well-formed email address"
17  }
18 ]
```

POST



localhost:8080/medicos

Params

Authorization

Headers (9)

Body ●

Pre-request Script

Tests

Settings

Body

Cookies

Headers (4)

Test Results



Status: 400 Bad Request

Pretty

Raw

Preview

Visualize

JSON



```
1  [
2  {
3      "campo": "nome",
4      "msgErro": "must not be blank"
5  },
6  {
7      "campo": "telefone",
8      "msgErro": "must not be blank"
9  },
10 {
11     "campo": "crm",
12     "msgErro": "must match \"\\d{4,6}\""
13  },
14 {
15     "campo": "email",
16     "msgErro": "must be a well-formed email"
17  }
]
```

```
public record DadosCadastroMedico(
```

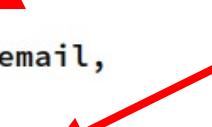
```
    @NotBlank
    String nome,
```



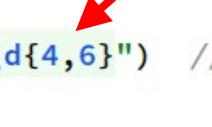
```
    @NotBlank
    @Email
    String email,
```



```
    @NotBlank
    String telefone,
```



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



```
    @NotNull
    Especialidade especialidade,
```

```
    @NotNull
    @Valid
    DadosEndereco endereco) {
```

```
}
```



Body • Pre-request Script Tests Settings

Status: 400 Bad Request

JSON ▾



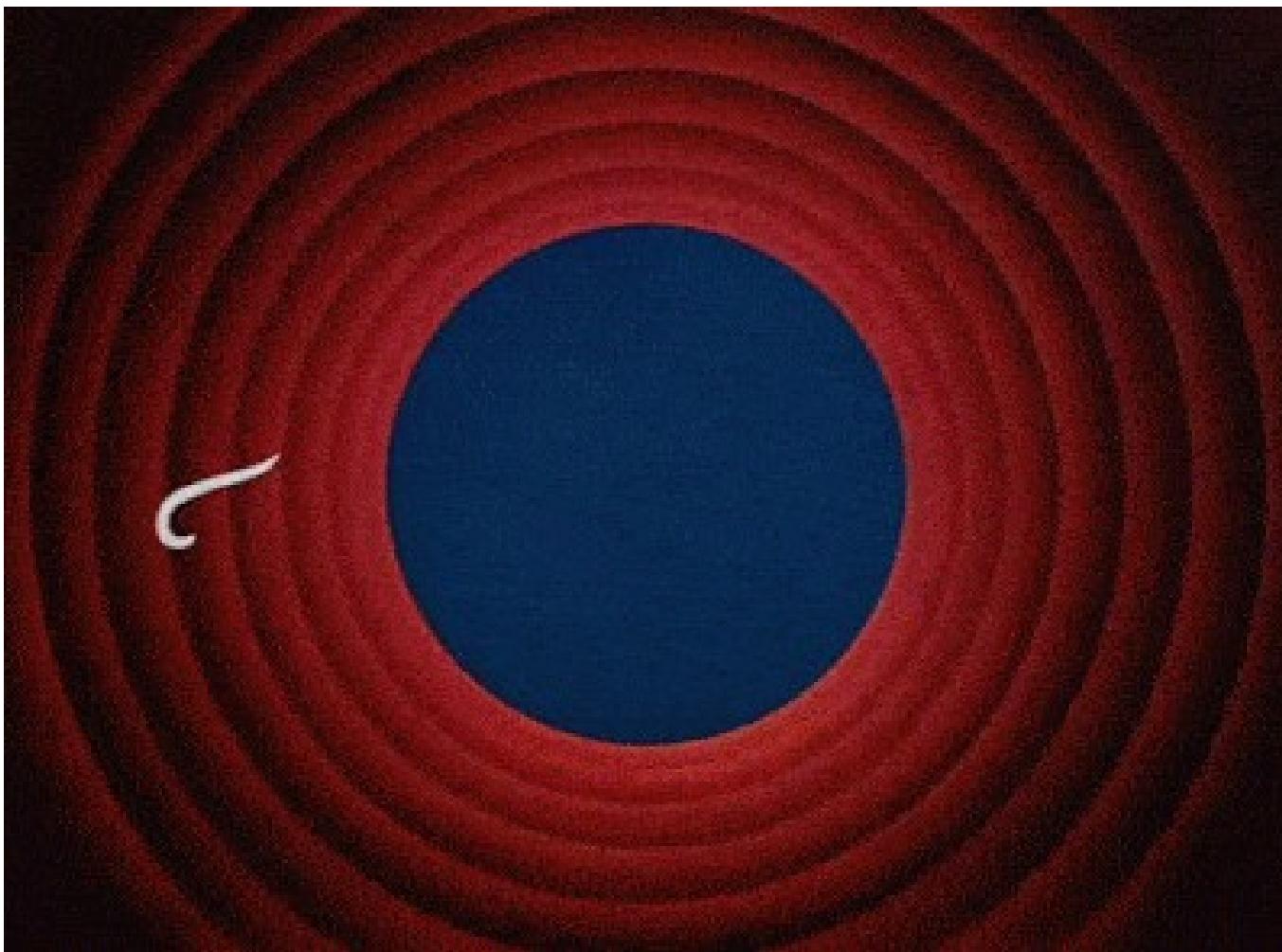
not be blank"

```
6  {
7      "campo": "telefone",
8      "msgErro": "must not be blank"
9  },
10 {
11     "campo": "crm",
12     "msgErro": "must match \\"\\d{4,6}\\"
13 },
14 {
15     "campo": "email",
16     "msgErro": "must be a well-formed email"
17 }
]
```

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

Com relação a essa api básica (crud básico)...

Com relação a essa api básica (crud básico)...



Projeto, no estado até este momento,
no github,
commit “final api basica antes security”

https://github.com/carlaopereirasanca/prw3_api_2025_2



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

Começando um tema bem simples...





ATENÇÃO

- Spring Security **é complexo!!**
- Como já vimos, poderia ser muito mais, o Spring (Boot) faz muita coisa pra nós.
 - Mas serão muitas classes, muitos detalhes a serem configurados.
- O último trabalho (Avaliação 4) é basicamente repetir as coisas mostradas aqui.
 - As configurações de segurança são genéricas, não dependem dos dados sendo manipulados (se médicos ou consertos de carros)
- DICA: já vai implementando, a cada aula, no projeto da mecânica.

Parte Teórica

Sobre o Spring Security

- Framework de segurança amplamente utilizado na plataforma Spring para aplicações Java.
- Ele fornece uma série de recursos e funcionalidades que ajudam a proteger suas aplicações contra ameaças de segurança e a controlar o acesso a recursos protegidos.



Características

- Autenticação
 - O Spring Security oferece suporte à autenticação de usuários.
 - Isso permite que os usuários ingressem em um sistema por meio de várias formas de autenticação, como login e senha, autenticação de token, autenticação de terceiros (OAuth, SAML, etc.), entre outras.
- Autorização
 - O Spring Security permite controlar o acesso dos usuários aos recursos da aplicação.
 - Isso é feito por meio de configuração de regras de autorização que determinam quem pode acessar quais partes do aplicativo.

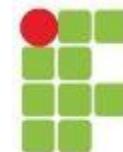


Características

- Integração com Spring Framework
 - O Spring Security é altamente integrado com outros projetos do ecossistema Spring, como o Spring MVC e o Spring Boot, facilitando a construção de aplicativos seguros.
- Suporte a autenticação baseada em anotações
 - O Spring Security permite a autenticação baseada em anotações, o que significa que você pode usar anotações Java para proteger métodos ou endpoints de API com facilidade.
- Proteção contra ameaças comuns
 - O Spring Security ajuda a proteger sua aplicação contra ameaças comuns, como ataques de injeção SQL, ataques CSRF (Cross-Site Request Forgery), ataques XSS (Cross-Site Scripting), entre outros.

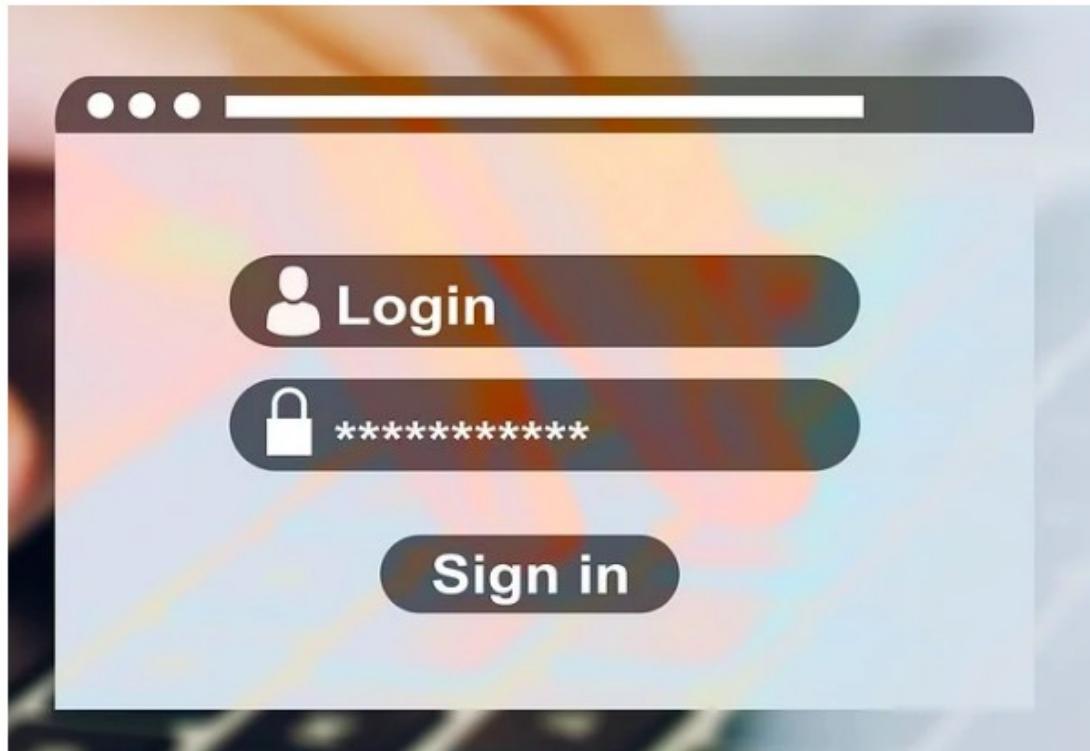


Tipos de autenticação (em geral)



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

Autenticação por usuário e senha



A utilização de PINs, usuário ou senha, é uma das formas mais básicas e simples de implementarmos autenticação em sistemas, partindo do pressuposto que somente o usuário possui essas informações de acesso.

Neste modelo depende do usuário a proteção dos seus dados, já que cada um escolhe qual será a sua senha ou PIN, por exemplo. Um problema é que hackers podem conseguir capturar essas informações, abrindo uma brecha no esquema de segurança proposto.

Autenticação por Biometria

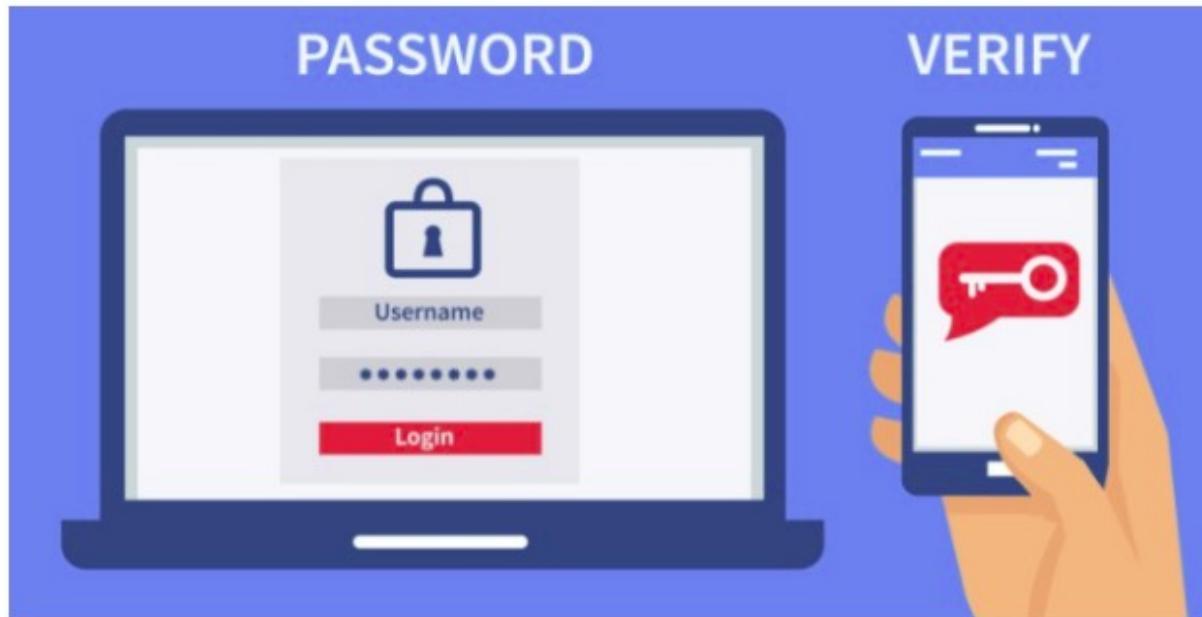


Este tipo de autenticação baseia-se na leitura de alguma característica física única do indivíduo, como uma digital, verificação de íris, ou até mesmo a voz. Esta é uma maneira bastante efetiva para os sistemas validarem se a pessoa que solicita o acesso é quem realmente diz ser.



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

Autenticação em dois fatores



Neste método, a ideia principal para a autenticação é a adição de mais uma camada de segurança ao acesso de nossas aplicações. Sendo assim, o usuário precisa de informações adicionais para terminar de acessar.

Por exemplo, em uma primeira etapa, a aplicação solicita um pin. Além disso, encaminha um código que vai complementar a autenticação para algum canal de comunicação do usuário, como celular ou e-mail.

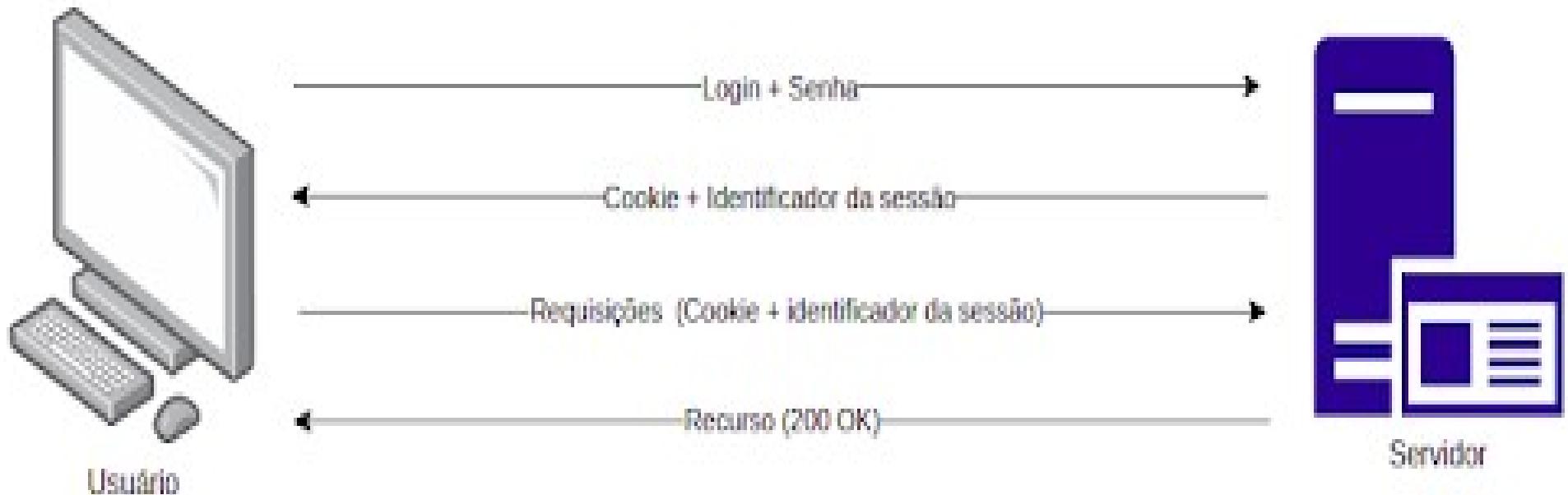
Um ponto de atenção nesta abordagem é que o usuário deve necessariamente ter o canal adicional disponível no processo de login.

Autenticação por sessão

Este foi um dos primeiros métodos de autenticação, criado no início do desenvolvimento das aplicações web. Muito empregado até hoje, neste modelo o usuário pode se autenticar com usuário e senha ou por algum outro método.

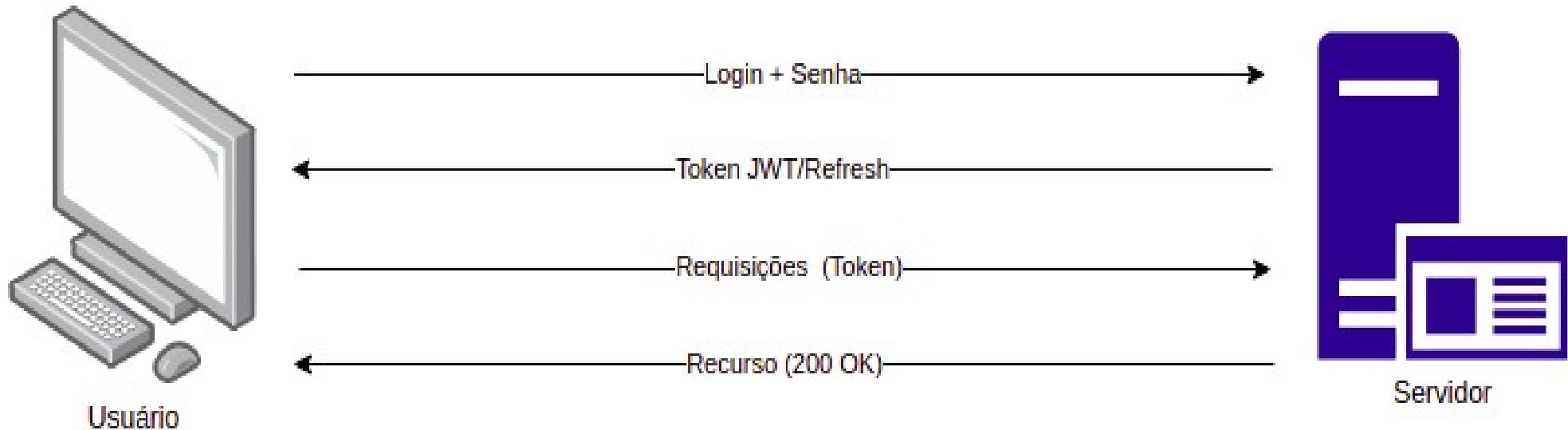
O servidor, por sua vez, cria uma sessão em sua memória ou banco e devolve a informação de usuário através de um cookie com o identificador da sessão criada.

Na próxima requisição, é passado o identificador da sessão e o servidor devolve o acesso ao recurso solicitado ou realiza algum outro tipo de manipulação referente à conta autenticada.



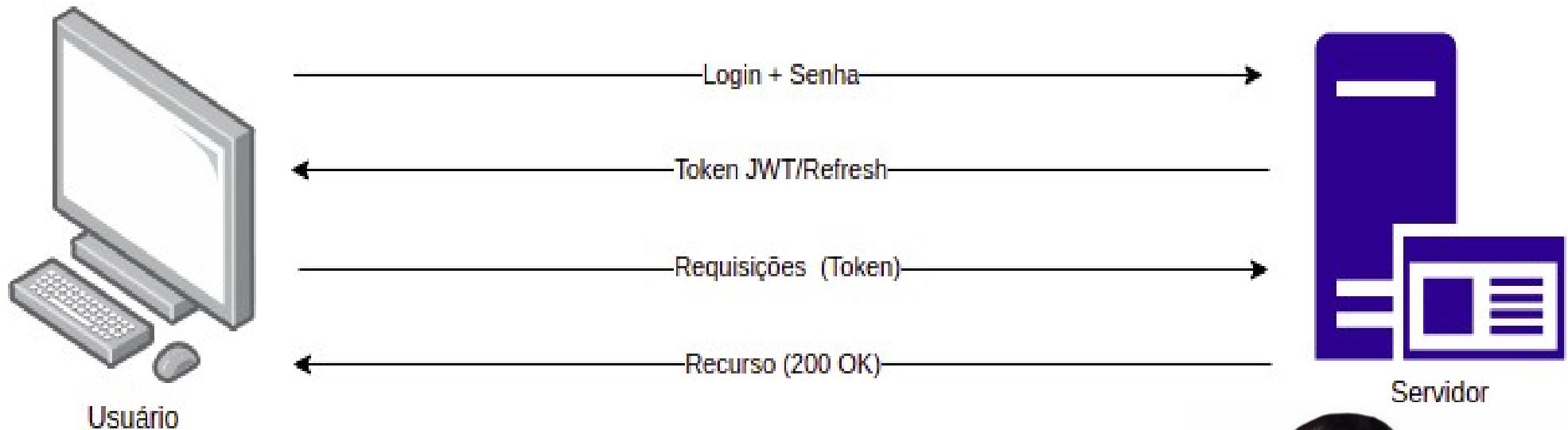
Autenticação por token

Neste método, após ter um login e senha validados pelo servidor, será criado um token que o usuário receberá em resposta e que permitirá o acesso a algum recurso. O padrão adotado por grande número das aplicações web hoje é o formato JWT (JSON Web Token) e ele fará com que o token seja assinado da forma correta para haver a autenticação da requisição a um recurso no servidor.



Autenticação por token

Neste método, após ter um login e senha validados pelo servidor, será criado um token que o usuário receberá em resposta e que permitirá o acesso a algum recurso. O padrão adotado por grande número das aplicações web hoje é o formato JWT (JSON Web Token) e ele fará com que o token seja assinado da forma correta para haver a autenticação da requisição a um recurso no servidor.



Lembrando: O que o padrão REST nos diz ?



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



vale
a pena
ver de
novo

REST

- Representational State Transfer
- **Estilo arquitetural** para projetar sistemas de software distribuídos, especialmente sistemas que operam na internet.
- Foi introduzido por Roy Fielding em sua **tese de doutorado** em 2000, onde ele descreveu os princípios e as características que definem a **arquitetura REST**.
 - OBS: um dos principais princípios é a separação entre cliente e servidor.
 - REST é um **conjunto** de princípios que permitem a criação de sistemas web que são interoperáveis entre si.
 - Essa abordagem facilita a criação de serviços web, APIs e integração entre diferentes sistemas.

1. **Cliente-Servidor:** Separação de preocupações entre cliente e servidor, permitindo independência e escalabilidade.
2. **Stateless (Sem Estado):** Cada solicitação do cliente para o servidor deve conter todas as informações necessárias. O servidor não mantém informações de estado entre solicitações.
3. **Cache:** As respostas do servidor podem ser cacheadas localmente pelos clientes para reduzir a latência e a carga no servidor.
4. **Interface Uniforme:** Uma interface padronizada para interações entre cliente e servidor, incluindo identificação de recursos, manipulação de recursos através de representações, mensagens autodescritivas e hipermídia como estado da aplicação (HATEOAS).
5. **Sistema em Camadas:** A arquitetura é organizada em camadas independentes, onde cada camada tem funções específicas e interage apenas com camadas adjacentes.
6. **Código sob Demanda (Opcional):** O servidor pode enviar código executável (como JavaScript) para o cliente para estender a funcionalidade. No entanto, essa restrição é opcional e nem sempre é utilizada.





vale
a pena
ver de
novo

REST

- Representational State Transfer
- **Estilo arquitetural** para projetar sistemas de software distribuídos, especialmente sistemas que operam na internet.
- Foi introduzido por Roy Fielding em sua **tese de doutorado** em 2000, onde ele descreveu os princípios e as características que definem a **arquitetura REST**.
 - OBS: um dos principais princípios é a separação entre cliente e servidor.
 - REST é um **conjunto** de regras para sistemas web que devem ser interoperáveis.
 - Essa abordagem faz parte da web 2.0, serviços web, APIs e frameworks modernos.

1. **Cliente-Servidor:** Separação de preocupações entre cliente e servidor, permitindo independência e escalabilidade.
2. **Stateless (Sem Estado):** Cada solicitação do cliente para o servidor deve conter todas as informações necessárias. O servidor não mantém informações de estado entre solicitações.
3. **Cache:** As respostas do servidor podem ser cacheadas localmente pelos clientes para reduzir a latência e a carga no servidor.
4. **Interface Uniforme:** Uma interface padronizada para interações entre cliente e servidor, incluindo identificação de recursos, manipulação de recursos através de representações, mensagens autodescritivas e hipermídia como estado da aplicação (HATEOAS).
5. **Sistema em Camadas:** A arquitetura é organizada em camadas independentes, onde cada camada tem funções específicas e interage apenas com camadas adjacentes.
6. **Código sob Demanda (Opcional):** O servidor pode enviar código executável (como JavaScript) para o cliente para estender a funcionalidade. No entanto, essa restrição é opcional e nem sempre é utilizada.



vale
a pena
ver
de novo

Stateless (“sem estado”)

A restrição "Stateless" (Sem Estado) é um dos princípios fundamentais da arquitetura REST (Representational State Transfer). Ela estabelece que cada solicitação que um cliente faz para um servidor deve conter todas as informações necessárias para que o servidor a entenda e responda adequadamente, sem depender de nenhum contexto ou estado anterior.

Isso significa que o servidor não mantém informações de estado entre as solicitações do cliente. Cada solicitação é tratada de forma isolada, e todas as informações necessárias para processar a solicitação devem ser fornecidas pelo cliente na própria solicitação.



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



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

Aplicação WEB tradicional x API Rest

■ Tradicional

- “Stateful”
- Quando usuário faz login, servidor guarda o “estado”
 - Cria e armazena “sessões”, com dados dos usuários
 - Consegue identificar nas requisições quem é o usuário

■ API Rest

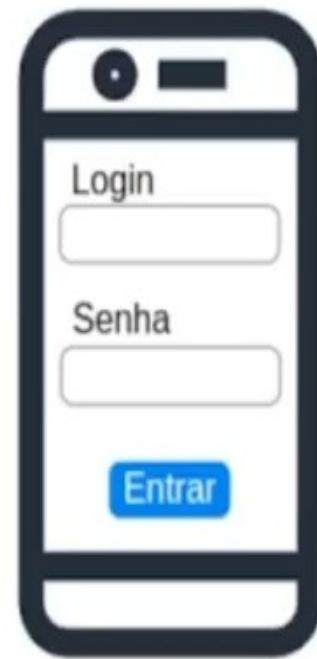
- “Stateless”
 - Servidor não guarda o “estado”
 - Não há ligação entre requisições, cada requisição é única



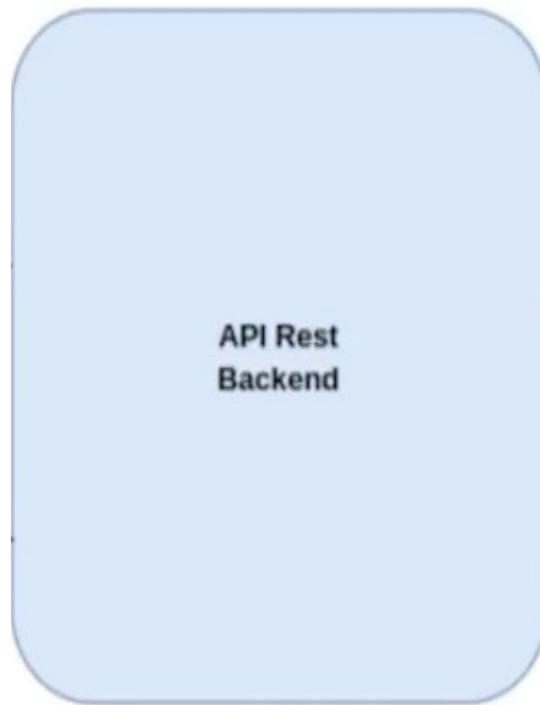
Processo de Autenticação



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

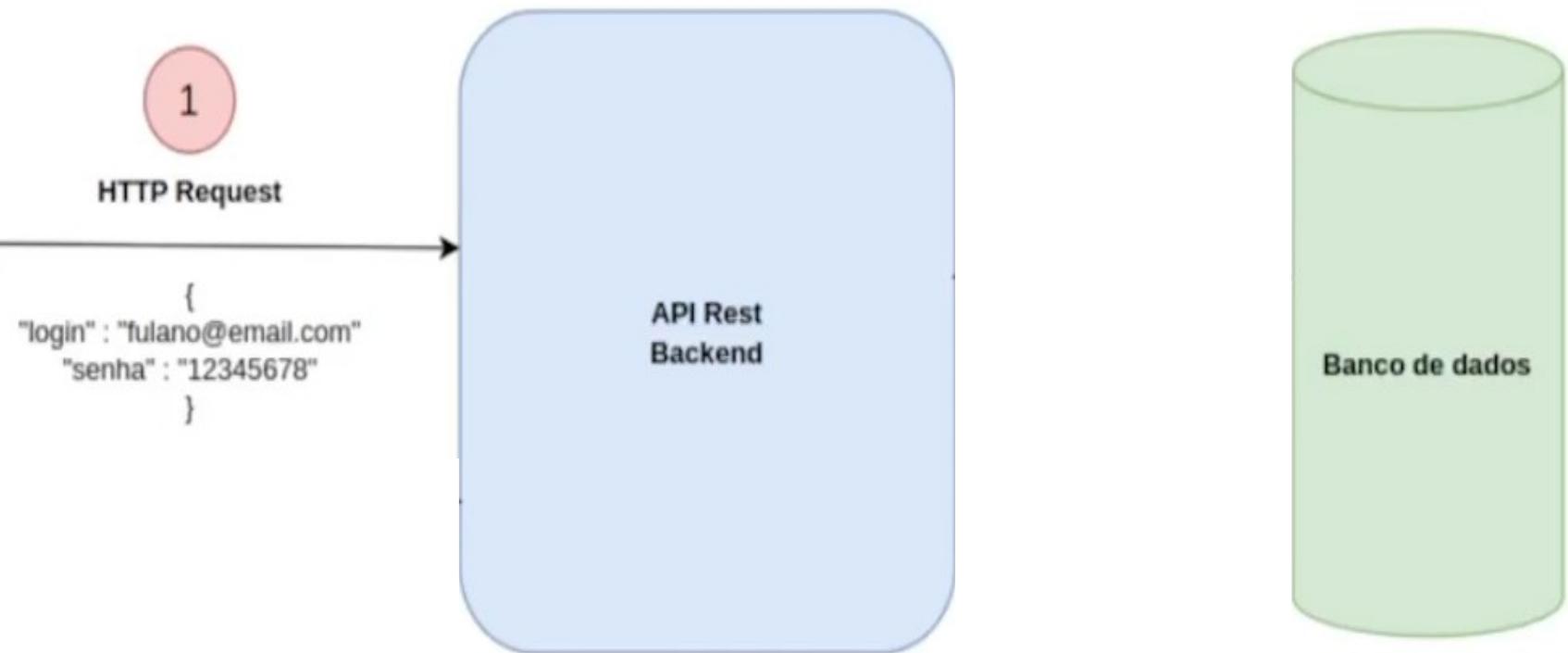
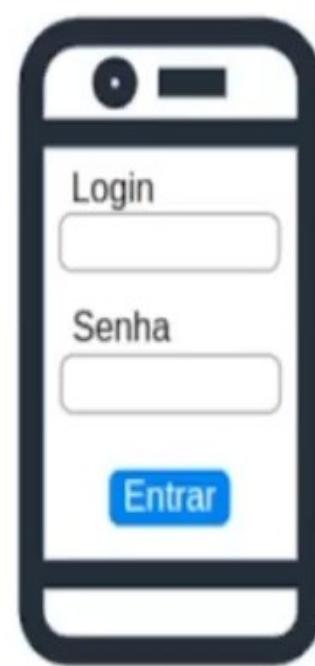


APP Cliente
Front-end / Mobile



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

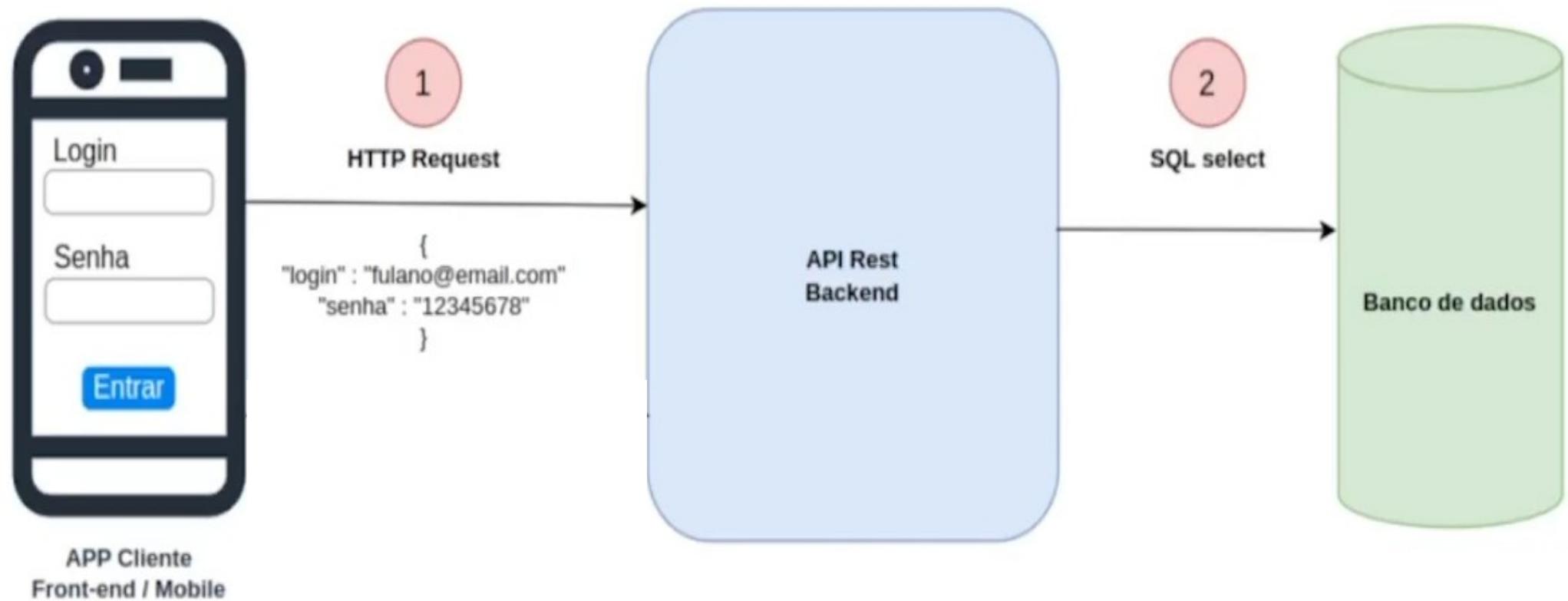
login

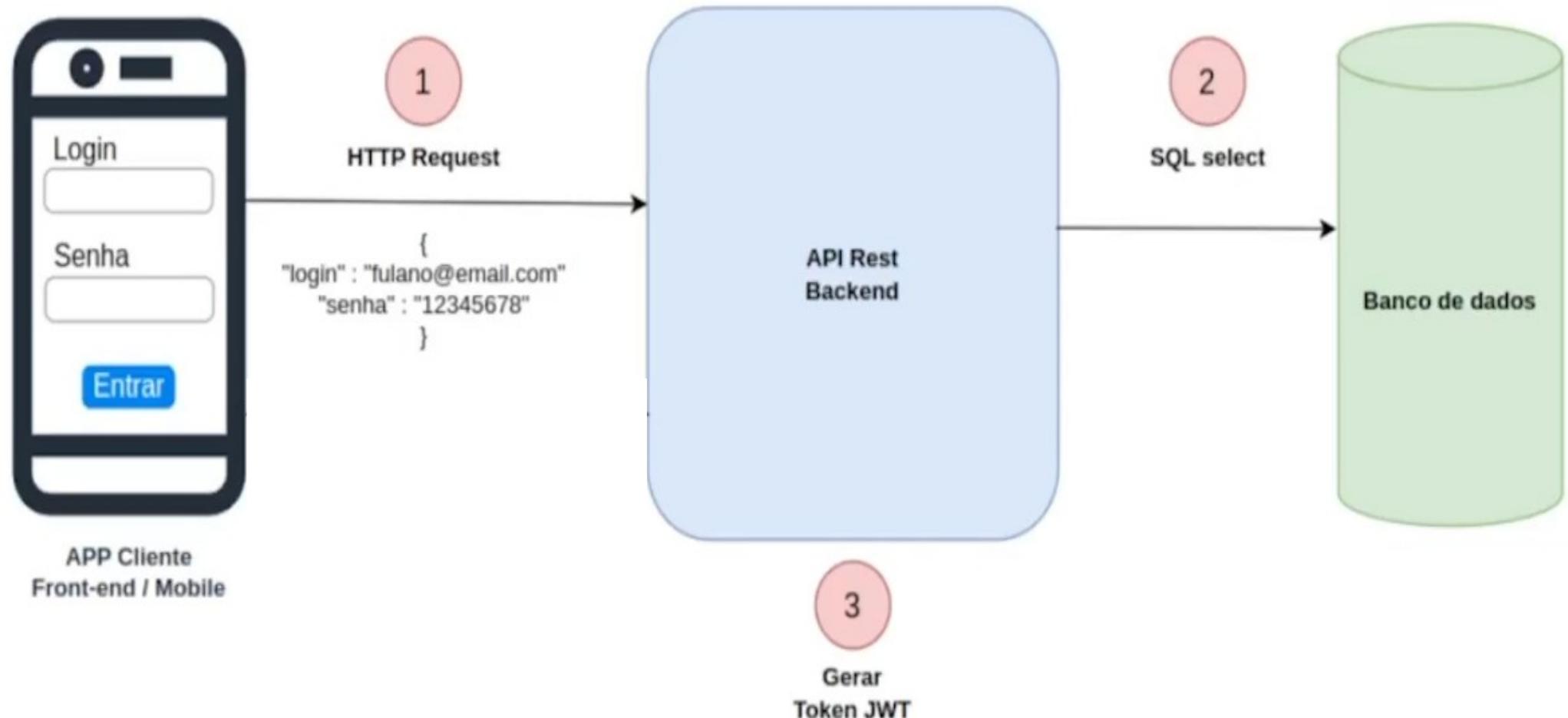


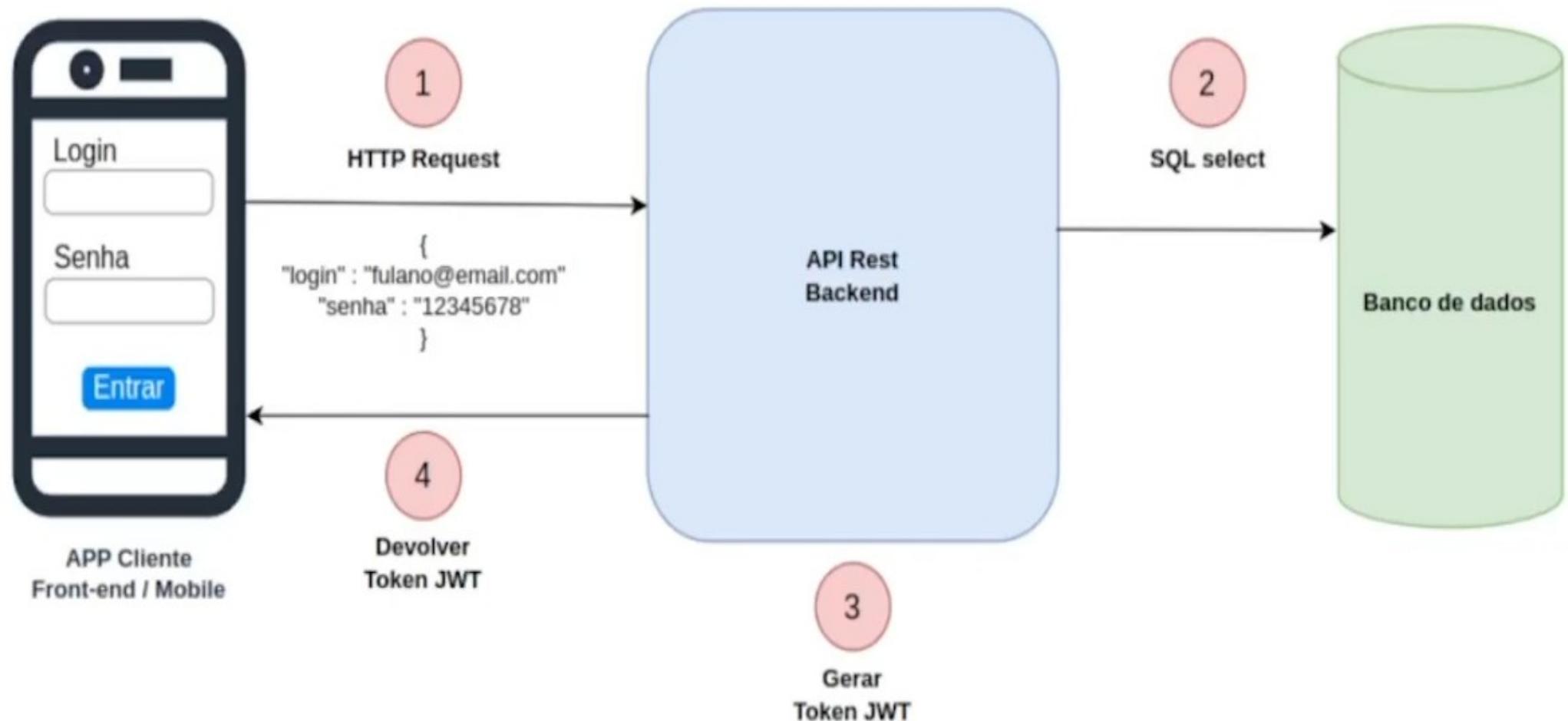
APP Cliente
Front-end / Mobile



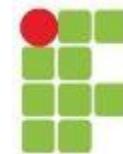
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos



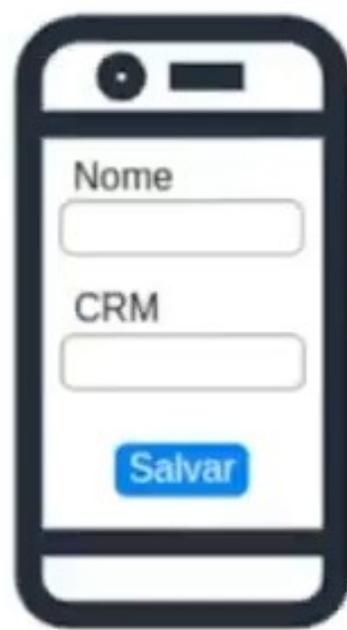




Próximas requisições



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



1

HTTP Request

HEADER

Authorization: Bearer TOKEN_JWT

BODY

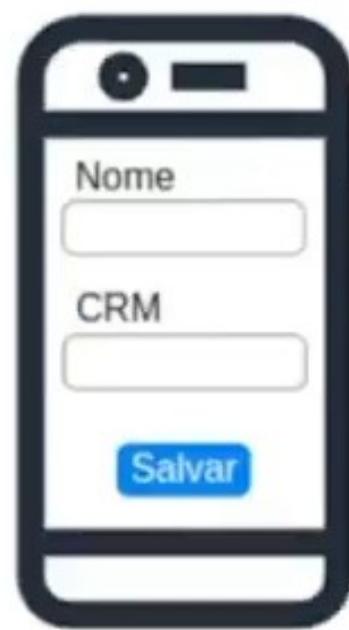
```
{  
    "nome": "Fulano Silva"  
    "crm": "000111"  
}
```

API Rest
Backend

APP Cliente
Front-end / Mobile



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



APP Cliente
Front-end / Mobile

1

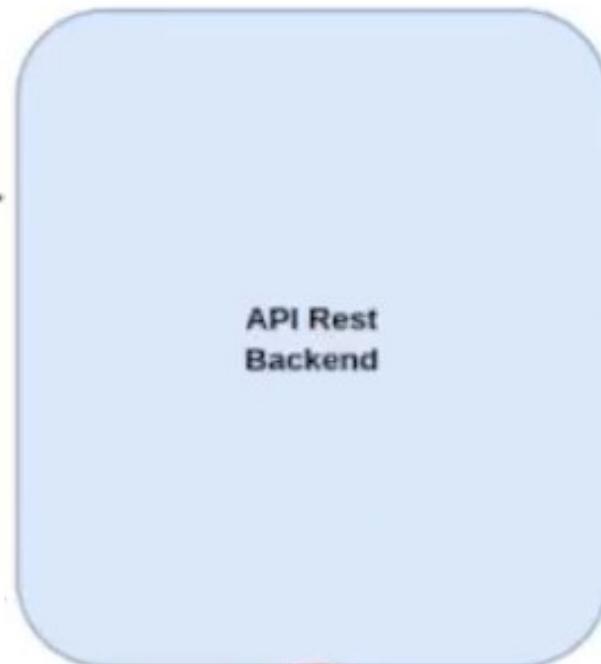
HTTP Request

HEADER

Authorization: Bearer TOKEN_JWT

BODY

```
{  
    "nome": "Fulano Silva"  
    "crm": "000111"  
}
```

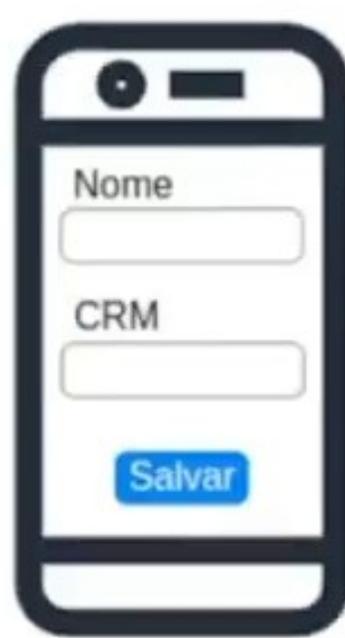


2

Validar
Token JWT



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



HTTP Request

HEADER

Authorization: Bearer TOKEN_JWT

BODY

```
{  
    "nome": "Fulano Silva"  
    "crm": "000111"  
}
```

API Rest
Backend

2

Validar
Token JWT

Não

Bloquear
requisição

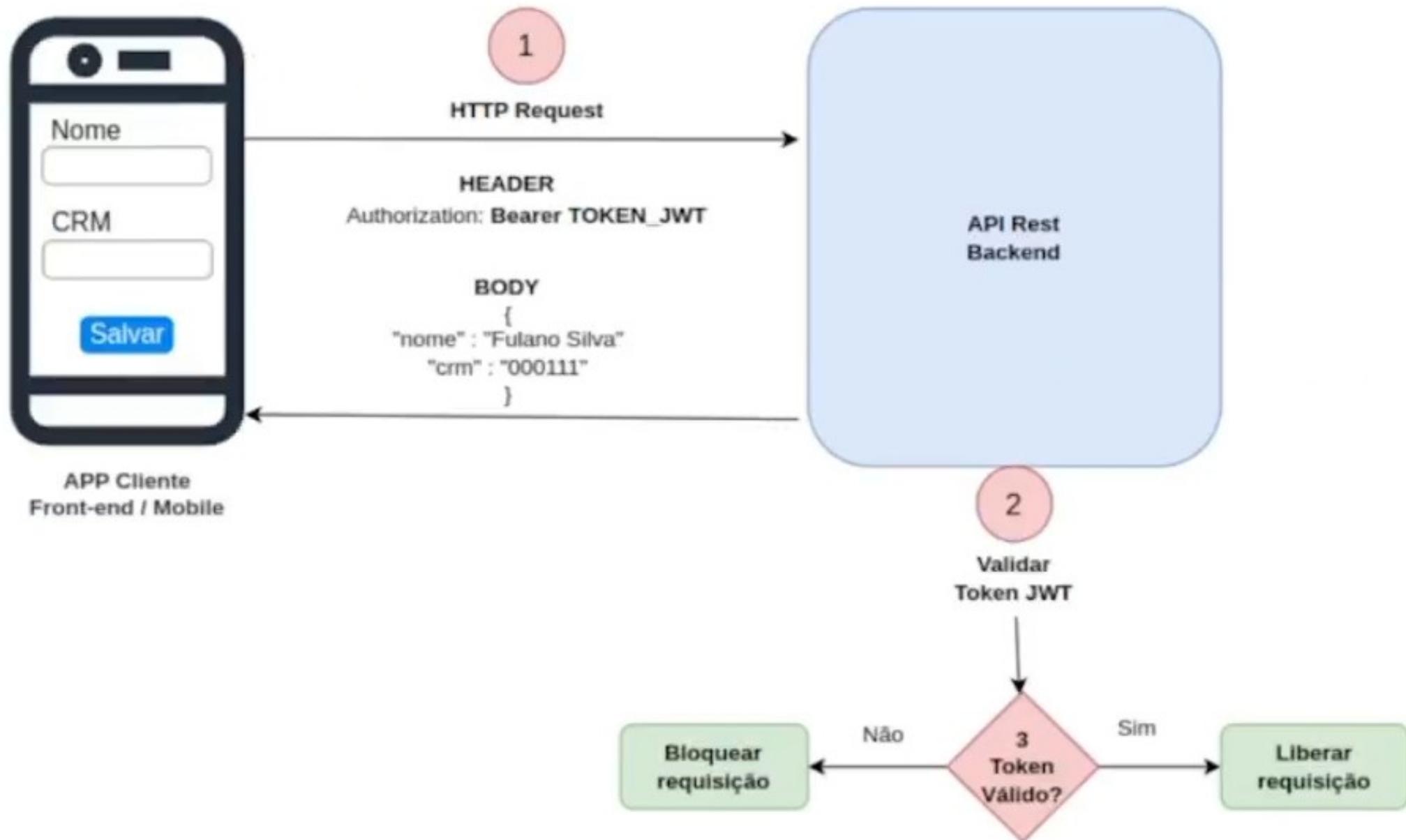
Sim

3
Token
Válido?

Liberar
requisição



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



Vamos lá!!!!



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

Adicionando a dependência do Spring Security no projeto



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

Site start.spring.io

The screenshot shows the configuration interface for a Spring Boot project on start.spring.io. Several options are highlighted with red arrows:

- A red arrow points from the "Maven" option under "Project" to the "Maven" button.
- A red arrow points from the "Java" option under "Language" to the "Java" button.
- A red arrow points from the "3.2.5" option under "Spring Boot" to the "3.2.5" button.

Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Dependencies

No dependency selected

ADD DEPENDENCIES... CTRL + B

Spring Boot

- 3.3.0 (SNAPSHOT)
- 3.3.0 (RC1)
- 3.2.5
- 3.2.6 (SNAPSHOT)
- 3.1.11



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

Site start.spring.io



Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.3.0 (SNAPSHOT)
- 3.3.0 (RC1)
- 3.2.6 (SNAPSHOT)
- 3.2.5
- 3.1.12 (SNAPSHOT)
- 3.1.11

Dependencies

No dependency selected

ADD DEPENDENCIES... CTRL + B



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

security

Press Ctrl for multiple adds

Spring Security SECURITY

Highly customizable authentication and access-control framework for Spring applications.

OAuth2 Client SECURITY

Spring Boot integration for Spring Security's OAuth2/OpenID Connect client features.

OAuth2 Resource Server SECURITY

Spring Boot integration for Spring Security's OAuth2 resource server features.

2.7 **Okta** SECURITY

Okta specific configuration for Spring Security/Spring Boot OAuth2 features. Enable your Spring Boot application to work with Okta via OAuth 2.0/OIDC.

OAuth2 Authorization Server SECURITY

Spring Boot integration for Spring Authorization Server.

Spring LDAP SECURITY

Makes it easier to build Spring based applications that use the Lightweight Directory Access Protocol.





Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.2.0 (SNAPSHOT)
- 3.1.6 (SNAPSHOT)
- 3.0.13 (SNAPSHOT)
- 2.7.18 (SNAPSHOT)
- 3.1.5
- 3.2.0 (RC1)
- 3.0.12
- 2.7.17

Project Metadata

Owner: com.example

GENERATE CTRL + ↵

EXPLORE CTRL + SPACE

SHARE...

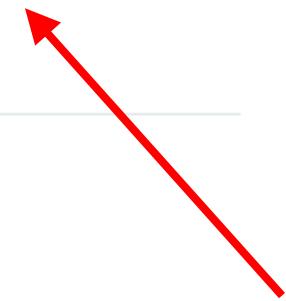
Dependencies

ADD ... CTRL + B

Spring Security

SECURITY

Highly customizable authentication and access-control framework for Spring applications.

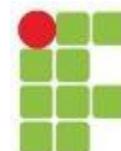


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

►	📁 .mvn
►	📄 HELP.md
►	📄 mvnw
►	📄 mvnw.cmd
►	📄 pom.xml

►	📁 src
---	-------

```
20 <dependency>
21   <groupId>org.springframework.boot</groupId>
22   <artifactId>spring-boot-starter-security</artifactId>
23 </dependency>
24
25 <dependency>
26   <groupId>org.springframework.boot</groupId>
27   <artifactId>spring-boot-starter-test</artifactId>
28   <scope>test</scope>
29 </dependency>
30 <dependency>
31   <groupId>org.springframework.security</groupId>
32   <artifactId>spring-security-test</artifactId>
33   <scope>test</scope>
34 </dependency>
35 </dependencies>
36
```



No IntelliJ...

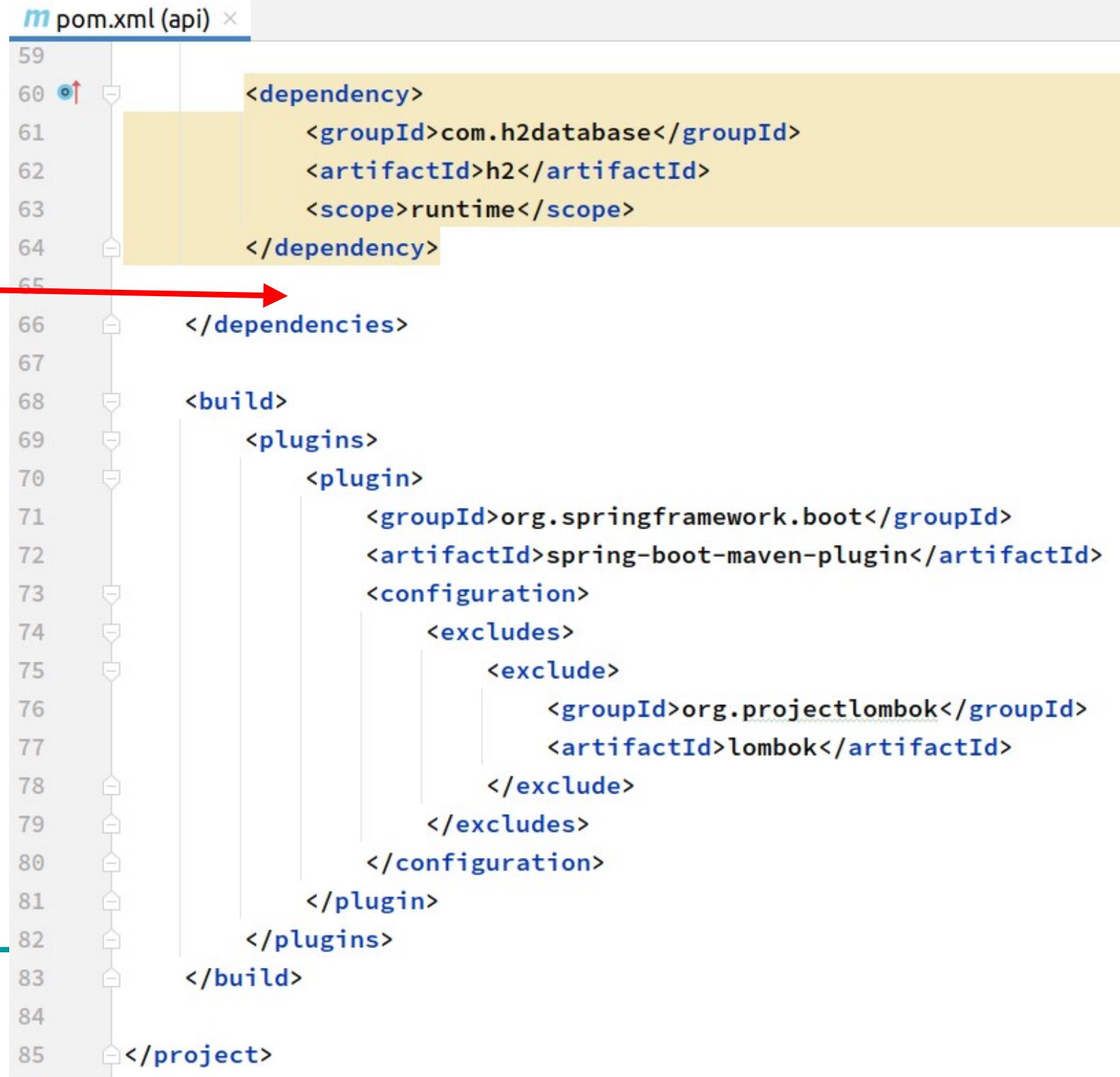
No pom.xml ...



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

Antes:

```
m pom.xml (api) ×
59
60   <dependency>
61     <groupId>com.h2database</groupId>
62     <artifactId>h2</artifactId>
63     <scope>runtime</scope>
64   </dependency>
65
66   </dependencies>
67
68   <build>
69     <plugins>
70       <plugin>
71         <groupId>org.springframework.boot</groupId>
72         <artifactId>spring-boot-maven-plugin</artifactId>
73         <configuration>
74           <excludes>
75             <exclude>
76               <groupId>org.projectlombok</groupId>
77               <artifactId>lombok</artifactId>
78             </exclude>
79           </excludes>
80         </configuration>
81       </plugin>
82     </plugins>
83   </build>
84
85 </project>
```



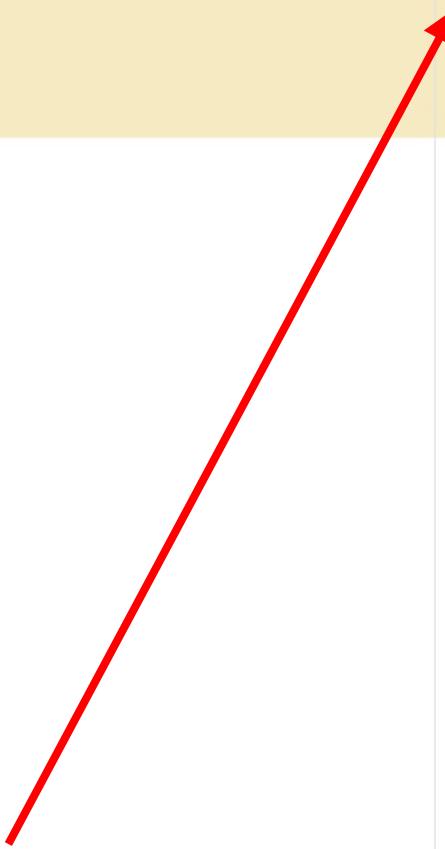
Depois:

pom.xml (api) ×

```
59
60 60 60 60 <dependency>
61      <groupId>com.h2database</groupId>
62      <artifactId>h2</artifactId>
63      <scope>runtime</scope>
64 64 64 64 </dependency>
65
66 66 66 <dependency>
67      <groupId>org.springframework.boot</groupId>
68      <artifactId>spring-boot-starter-security</artifactId>
69 69 69 </dependency>
70
71 71 71 <dependency>
72      <groupId>org.springframework.security</groupId>
73      <artifactId>spring-security-test</artifactId>
74      <scope>test</scope>
75 75 75 </dependency>
76
77 77 </dependencies>
78
79 79 <build>
80      <plugins>
81          <plugin>
```

0 2 2 1

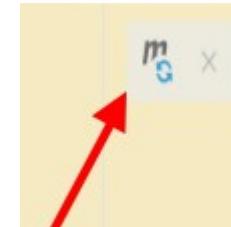
m g ×



```
59  
60 <dependency>  
61     <groupId>com.h2database</groupId>  
62     <artifactId>h2</artifactId>  
63     <scope>runtime</scope>  
64 </dependency>  
65  
66 <dependency>  
67     <groupId>org.springframework.boot</groupId>  
68     <artifactId>spring-boot-starter-security</artifactId>  
69 </dependency>  
70  
71 <dependency>  
72     <groupId>org.springframework.security</groupId>  
73     <artifactId>spring-security-test</artifactId>  
74     <scope>test</scope>  
75 </dependency>  
76  
77 </dependencies>  
78  
79 <build>  
80     <plugins>  
81         <plugin>
```



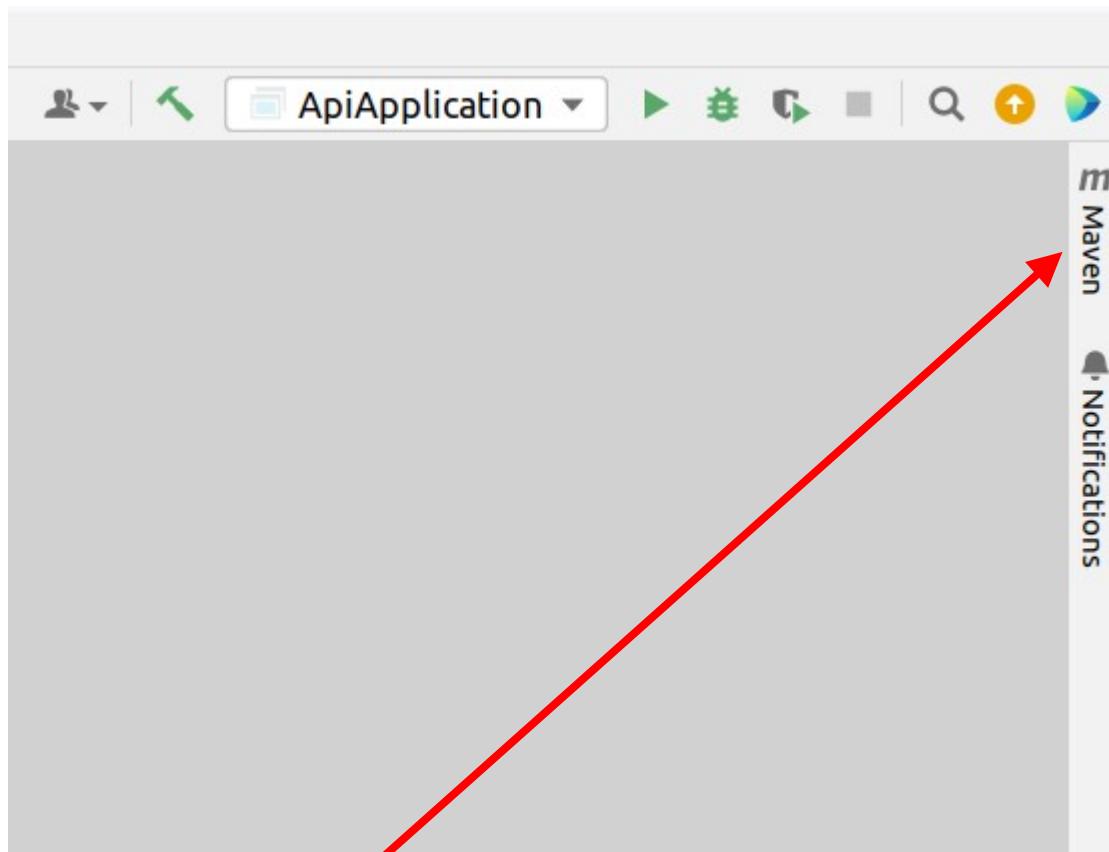
Relembrando...



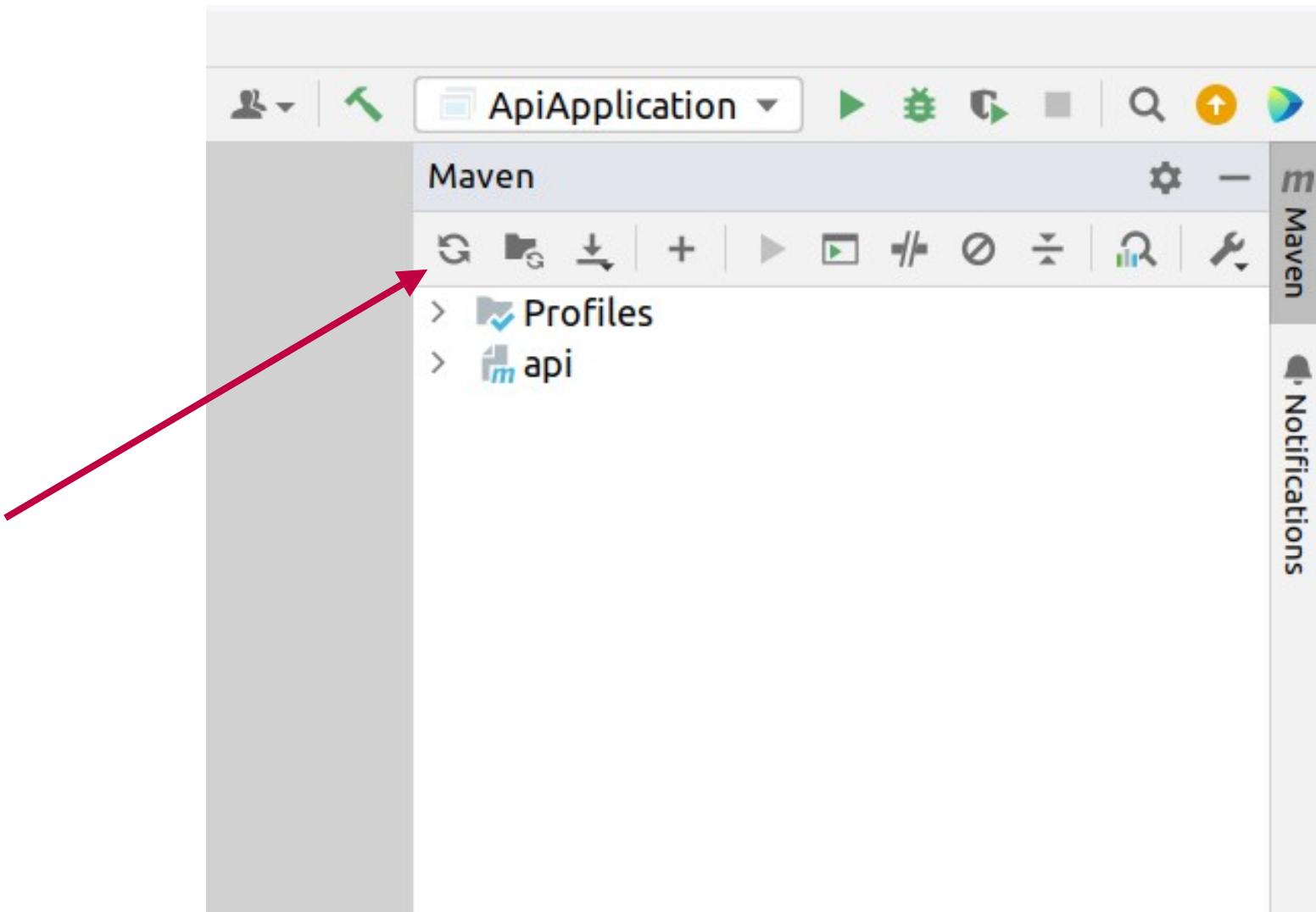
Se por acaso este botão não apareceu,
você pode dar um refresh nas
dependências do Maven...



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

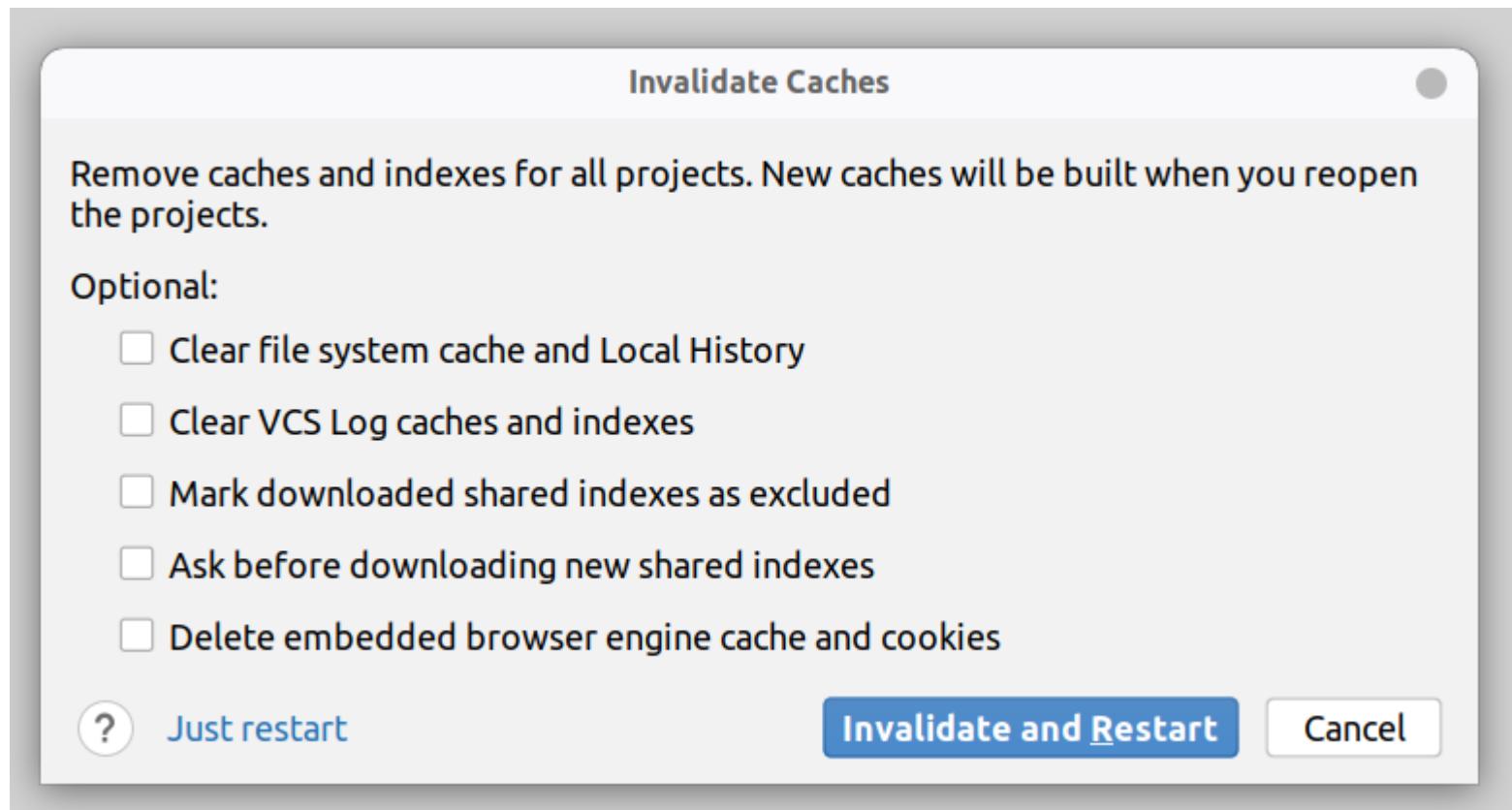


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos



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

Ou o file | invalidate caches



OBS:
Se por acaso o projeto estava em execução,
pare e reinicie o projeto.



Rodando o projeto...



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

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

```
 .   ----
 \  /   _ - _ - ( )_ -- -- - \ \ \ \
 ( ( )\__| ' | ' | ' | ' | \ / ` | \ \ \ \
 \| \| _ )| | _ )| | | | | | ( | | ) ) )
 ' | _ _ | . _ | _ | _ | _ \_, , | / / /
 ======|_|=====|_|=/_/_/_/
 :: Spring Boot ::           (v3.1.3)
```

```
2023-11-02T17:34:21.299-03:00 INFO 12274 --- [ restartedMain] br.edu.ifsp.pw3.api.ApiApplication      : Starting ApiApplication using Java 20.0.1 with PID 12
2023-11-02T17:34:21.302-03:00 INFO 12274 --- [ restartedMain] br.edu.ifsp.pw3.api.ApiApplication      : No active profile set, falling back to 1 default prof
2023-11-02T17:34:21.436-03:00 INFO 12274 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtoo
2023-11-02T17:34:21.436-03:00 INFO 12274 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting t
2023-11-02T17:34:22.713-03:00 INFO 12274 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT
2023-11-02T17:34:22.816-03:00 INFO 12274 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 81 ms. Fo
2023-11-02T17:34:23.855-03:00 INFO 12274 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
```

■ ■ ■

```
2023-11-02T17:34:24.995-03:00 INFO 12274 --- [ restartedMain] org.hibernate.cfg.Environment      : HHH000406: Using bytecode reflection optimizer
2023-11-02T17:34:25.153-03:00 INFO 12274 --- [ restartedMain] o.h.b.i.BytecodeProviderInitiator    : HHH000021: Bytecode provider name : bytebuddy
2023-11-02T17:34:25.390-03:00 INFO 12274 --- [ restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo  : No LoadTimeWeaver setup: ignoring JPA class transform
2023-11-02T17:34:25.866-03:00 INFO 12274 --- [ restartedMain] o.h.b.i.BytecodeProviderInitiator    : HHH000021: Bytecode provider name : bytebuddy
2023-11-02T17:34:26.462-03:00 INFO 12274 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator   : HHH000490: Using JtaPlatform implementation: [org.hib
2023-11-02T17:34:26.464-03:00 INFO 12274 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit
2023-11-02T17:34:26.950-03:00 WARN 12274 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefor
2023-11-02T17:34:27.318-03:00 WARN 12274 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :
```

Using generated security password: 6b2df463-3839-4ca1-9561-b7e823758a9f

This generated password is for development use only. Your security configuration must be updated before running your application in production.

```
2023-11-02T17:34:27.443-03:00 INFO 12274 --- [ restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframework.sec
2023-11-02T17:34:27.482-03:00 INFO 12274 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2023-11-02T17:34:27.504-03:00 INFO 12274 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context pa
2023-11-02T17:34:27.516-03:00 INFO 12274 --- [ restartedMain] br.edu.ifsp.pw3.api.ApiApplication      : Started ApiApplication in 6.819 seconds (process runn
```

Vamos disparar a requisição que devolve a lista de médicos:



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

GET lista geral

HTTP PW3 / lista geral

GET localhost:8080/medicos

Params Authorization Headers (8) Body Pre-request Script Tests Settings

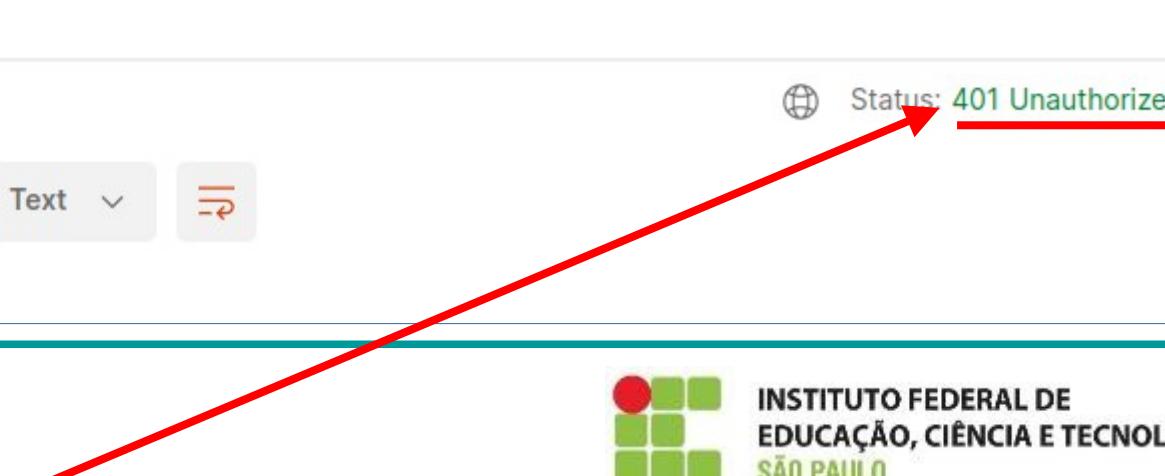
Query Params

Key	Value	Description
Key	Value	Description

Body Cookies (1) Headers (13) Test Results

Pretty Raw Preview Visualize Text 

1

Status: 401 Unauthorized 



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

GET lista geral

HTTP PW3 / lista geral

GET localhost:8081

Params Authorization Head

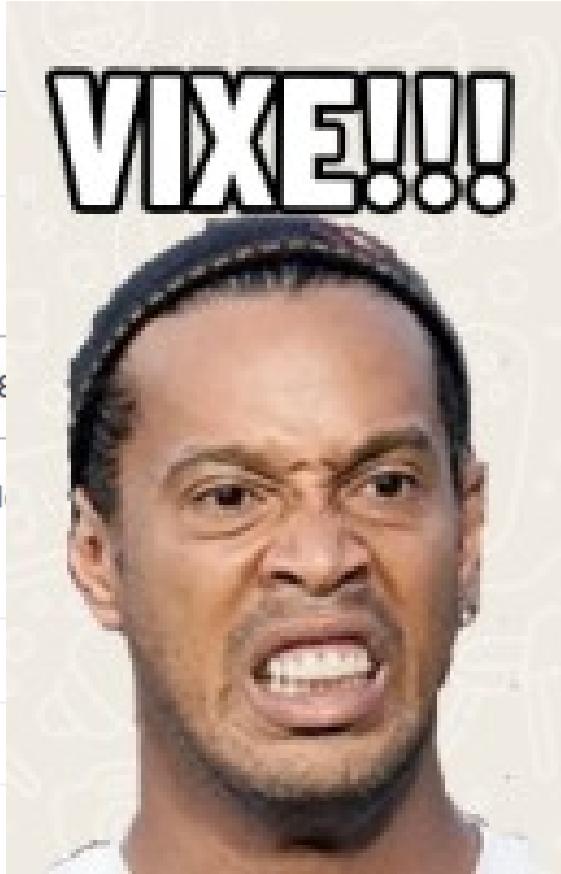
Query Params

Key	Description
Key	Description

Body Cookies (1) Headers (13) Test Results

Pretty Raw Preview Visualize Text

Status: 401 Unauthorized



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

Por padrão, bloqueia tudo!!!!

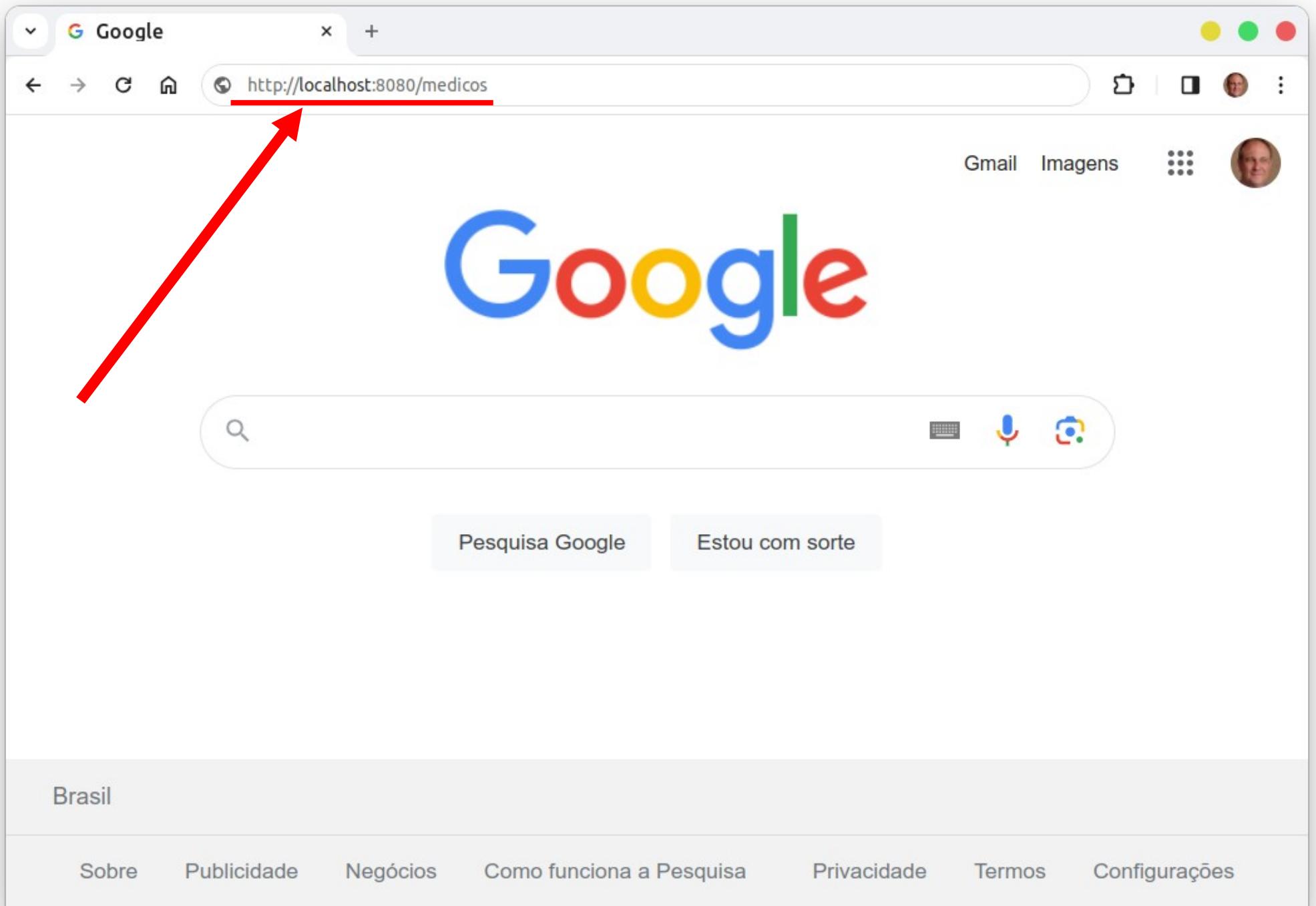


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos

Vamos tentar no navegador:



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



Please sign in

localhost:8080/login

Please sign in

Username

Password

Sign in

Please sign in

localhost:8080/login

Please sign in

Username

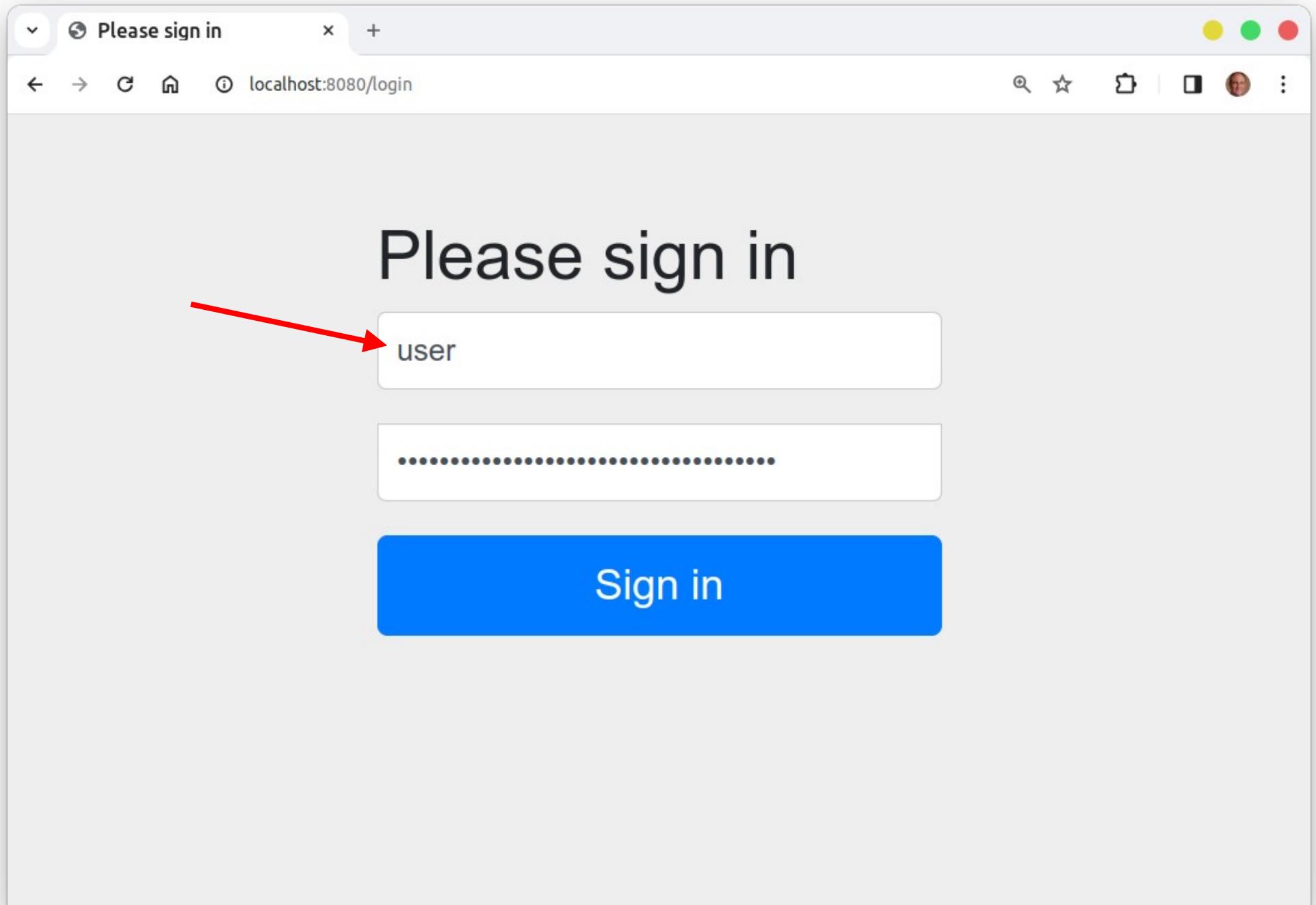
Password

Sign in



**Página gerada pelo próprio
Spring Security**

Pode usar aquela senha gerada e exibida no log do Spring



localhost:8080/medicos

```
// 20231102180453
// http://localhost:8080/medicos?continue

[
{
    "id": 2,
    "nome": "Lupita Pereira",
    "email": "lupy@gmail.com",
    "telefone": "16992482222",
    "crm": "222222",
    "especialidade": "DERMATOLOGIA",
    "endereco": {
        "logradouro": "RUA DOIS",
        "bairro": "BAIRRO DOIS",
        "cep": "22222222",
        "numero": null,
        "complemento": null,
        "cidade": "Ibaté",
        "uf": "SP"
    },
    "ativo": false
},
{
    "id": 3,
```



Isso é o que o Spring faz por default.
Mas não é isso que queremos.
Queremos autenticação Stateless.

Vamos começar a alterar o projeto
para atingir esse objetivo.



Mas fica pra próxima aula...



Só mais uma coisinha...

Só mais uma coisinha...

- Ele não te deixa mais acessar o h2 console...



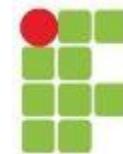
- Vai precisar usar aquele console **jar** que vimos no início da disciplina.
- Pode usar também o Datagrip, da JetBrains
 - É pago, mas pode acessar usando a licença acadêmica.

Exercícios



Exercícios

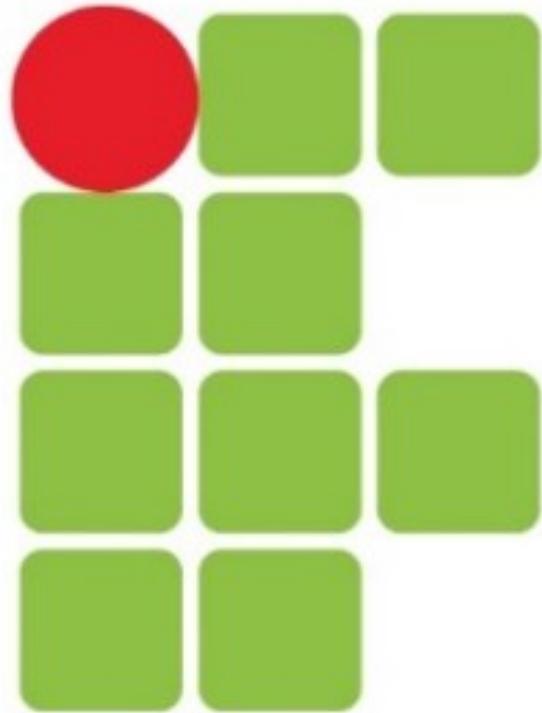
- Experimente com os códigos vistos na aula de hoje...



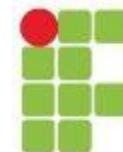
INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Câmpus São Carlos



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



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



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