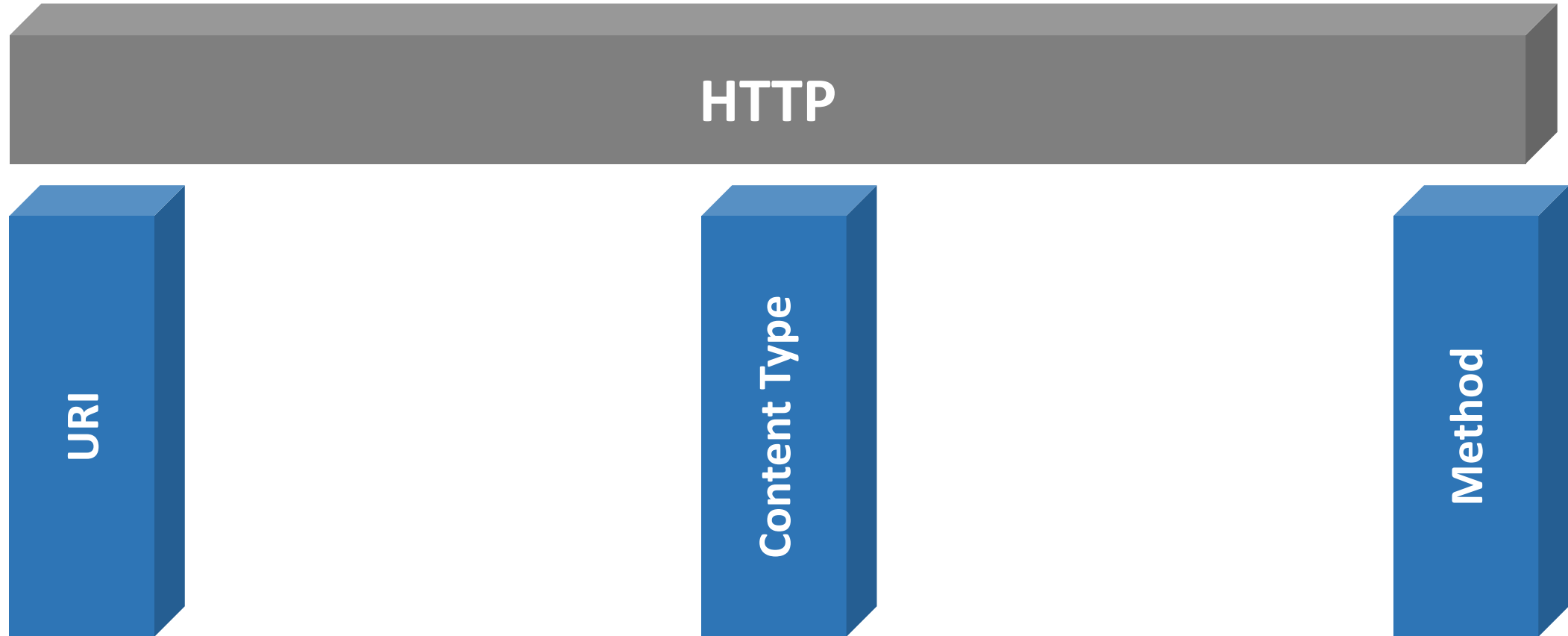


# Laboratório **REST**

Disciplina: <b>Sistemas Distribuídos</b>	Semestre: <b>2022/1</b>
Professor: <b>Vitor Figueiredo</b>	Versão: <b>6.0</b>



Servidor rodando localhost ouvindo a porta 8080

## Aplicação A

arquivo\_a1

arquivo\_a2

arquivo\_a3

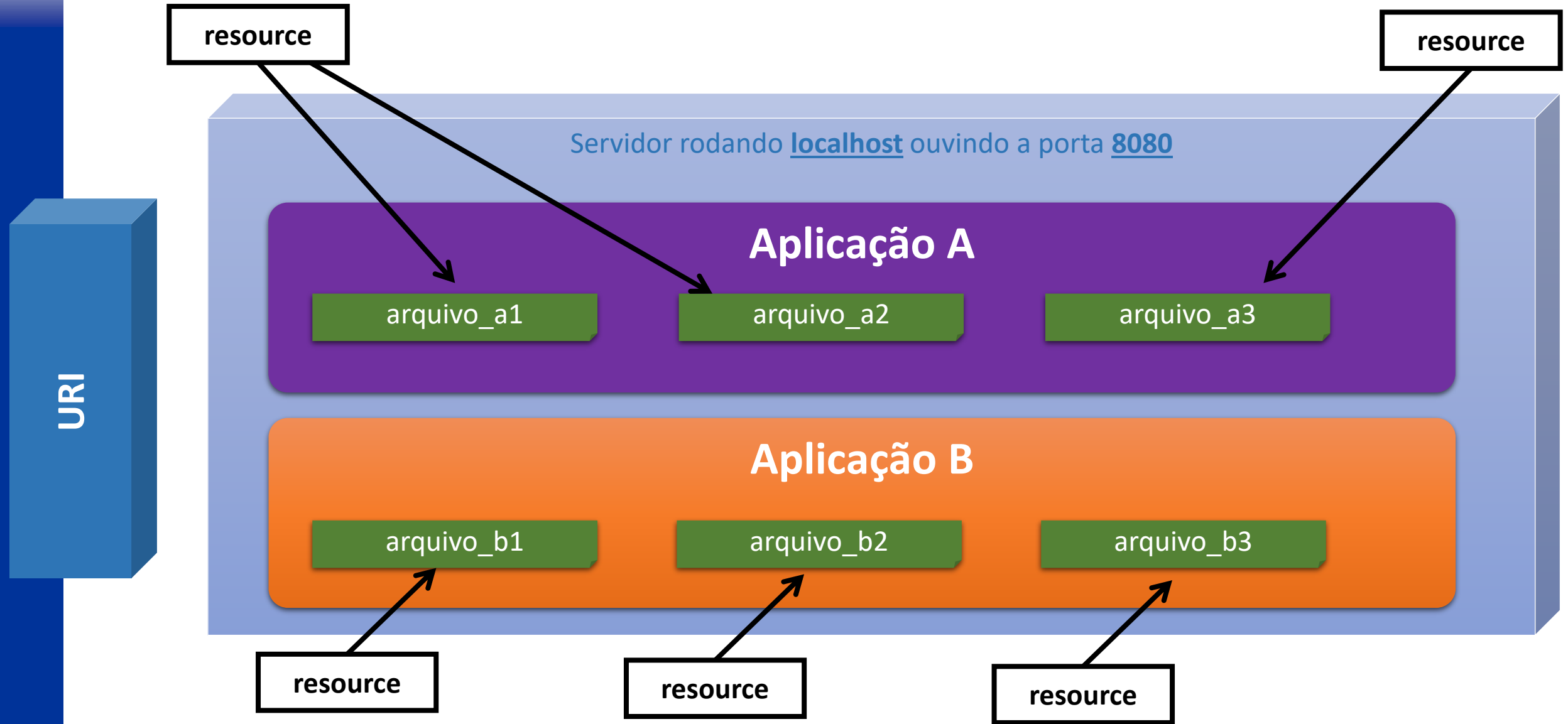
## Aplicação B

arquivo\_b1

arquivo\_b2

arquivo\_b3

URI



`http://localhost:8080/estoque/relatorio.html`

Servidor rodando **localhost** ouvindo a porta **8080**

## estoque

home.html

fornecedor.html

relatorio.html

## marketing

index.html

cliente.html

relatorio.html

URI

`http://localhost:8080/marketing/relatorio.html`

`http://localhost:8080/estoque/relatorio.html`

Servidor rodando **localhost** ouvindo a porta **8080**

**estoque**

`home.html`

`fornecedor.html`

`relatorio.html`

**marketing**

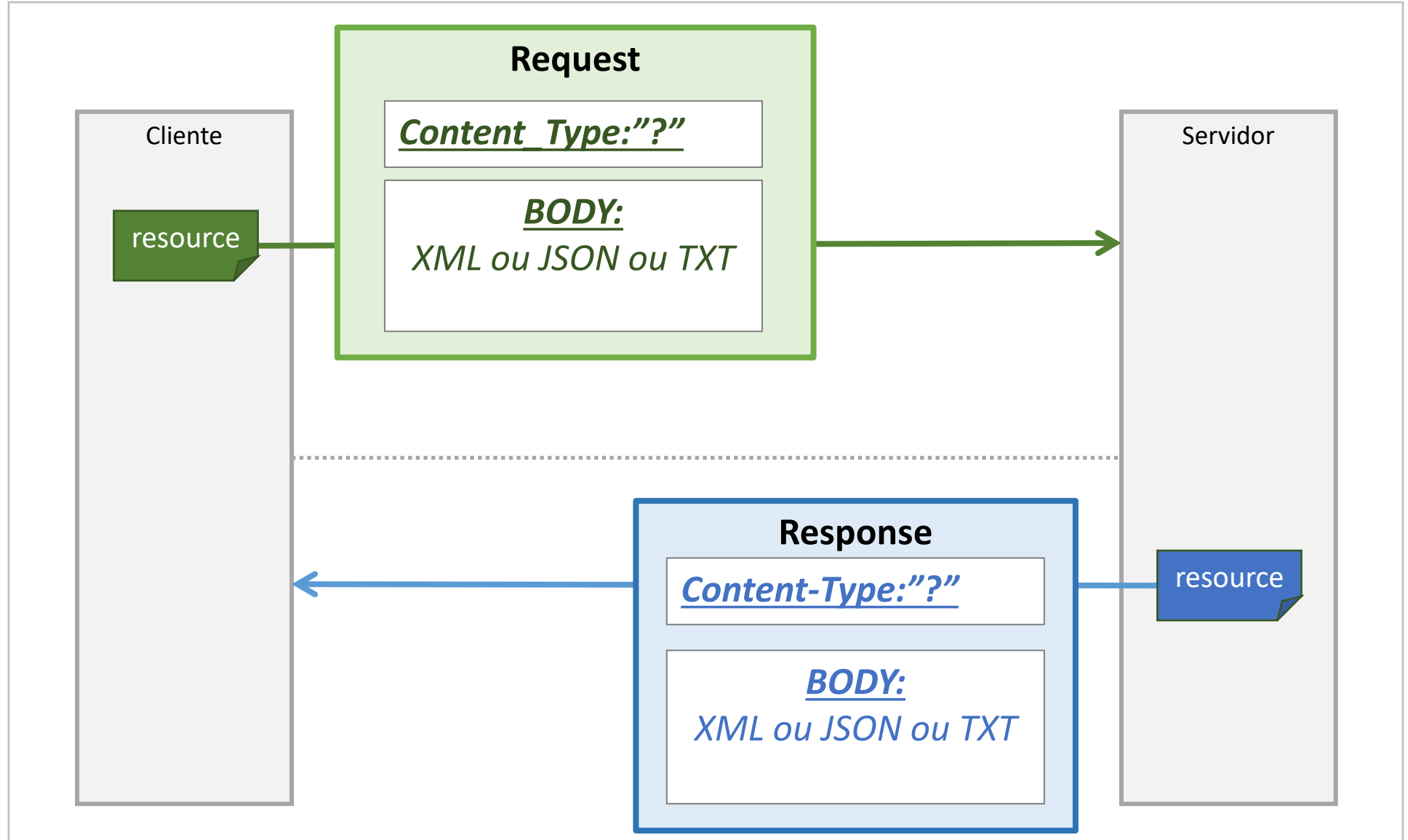
`index.html`

`cliente.html`

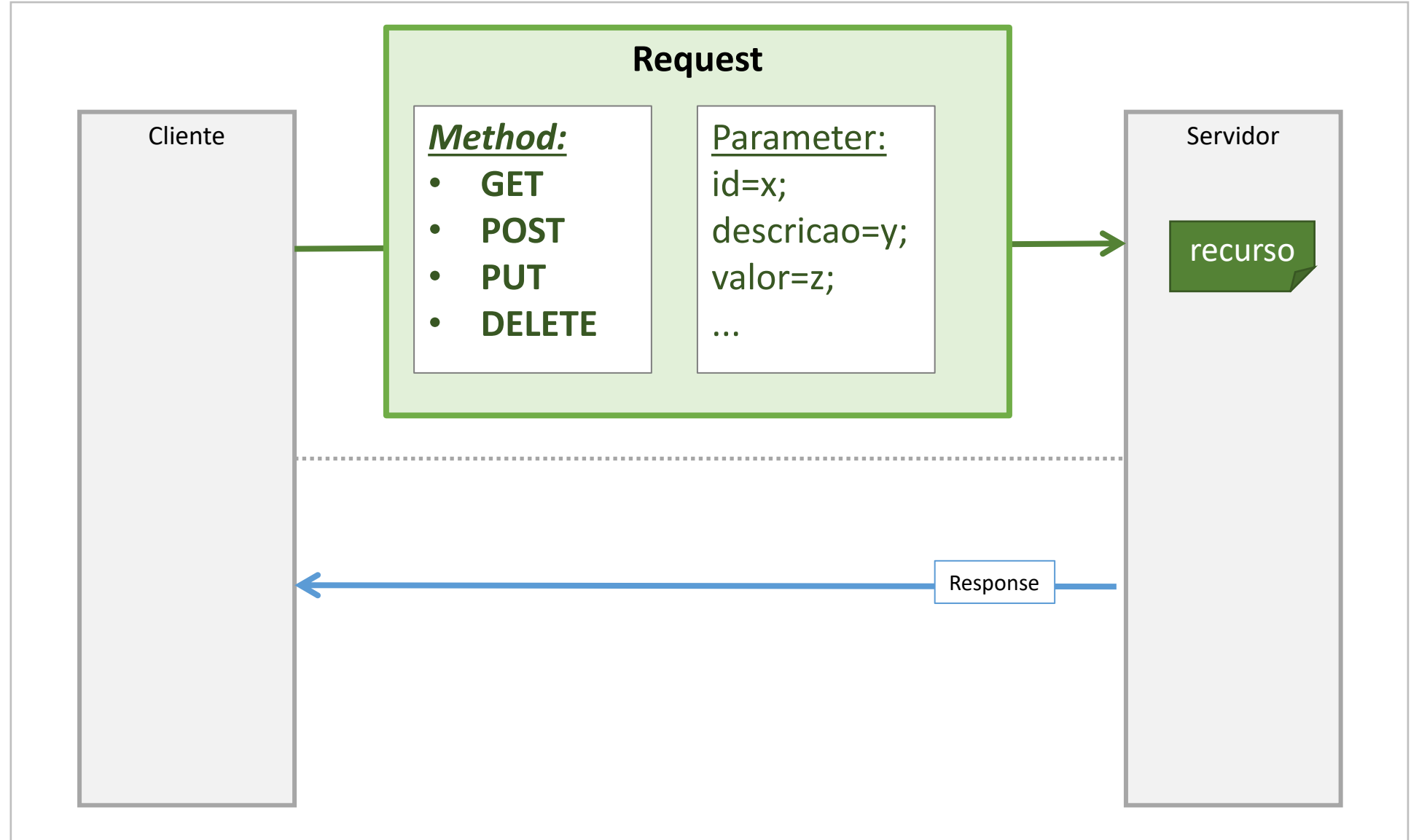
`relatorio.html`

`http://localhost:8080/marketing/relatorio.html`

URI



## Method



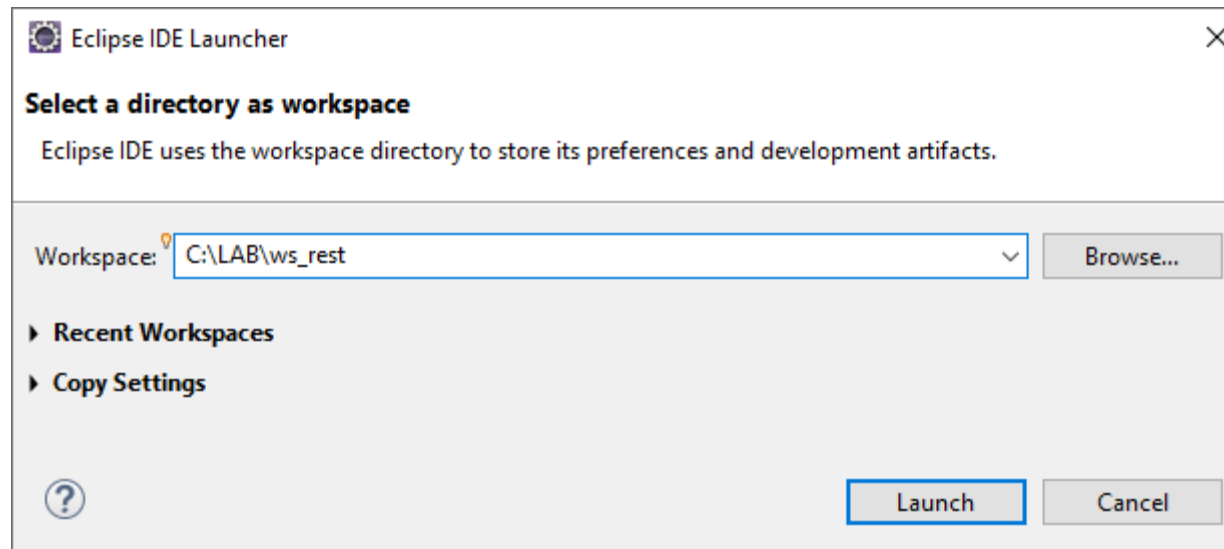


Método HTTP	Descrição
<b>GET</b>	Requisita a <b>consulta</b> de recurso. Para consultar um recurso específico, o pacote do REQUEST deve conter o respectivo identificador (chave primária).
<b>POST</b>	Requisita a <b>criação</b> de uma recurso. Por exemplo, o registro de um produto no banco de dados. O pacote de REQUEST deve conter os dados para criação do recurso. Estes dados podem estar no corpo do pacote de REQUEST
<b>PUT</b>	Requisita a <b>atualização</b> dos dados de um recurso já existente. Também leva os dados no corpo do pacote de REQUEST
<b>DELETE</b>	Requisita a <b>remoção</b> do recurso. O identificador do recurso (chave primária) deve estar presente no pacote REQUEST, seja no corpo do pacote ou na própria URI

Classe	Significado	Exemplos
<b>100</b>	Informacional	100 Continue 101 Switching Protocols 102 Processing
<b>200</b>	Sucesso	200 OK 201 Created 202 Accepted 204 No Content
<b>300</b>	Redirecionamento	300 Multiple Choices 301 Moved Permanently
<b>400</b>	Erro no cliente	400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
<b>500</b>	Erro no servidor	500 Internal Server Error 501 Not Implemented 502 Bad Gateway 503 Service Unavailable

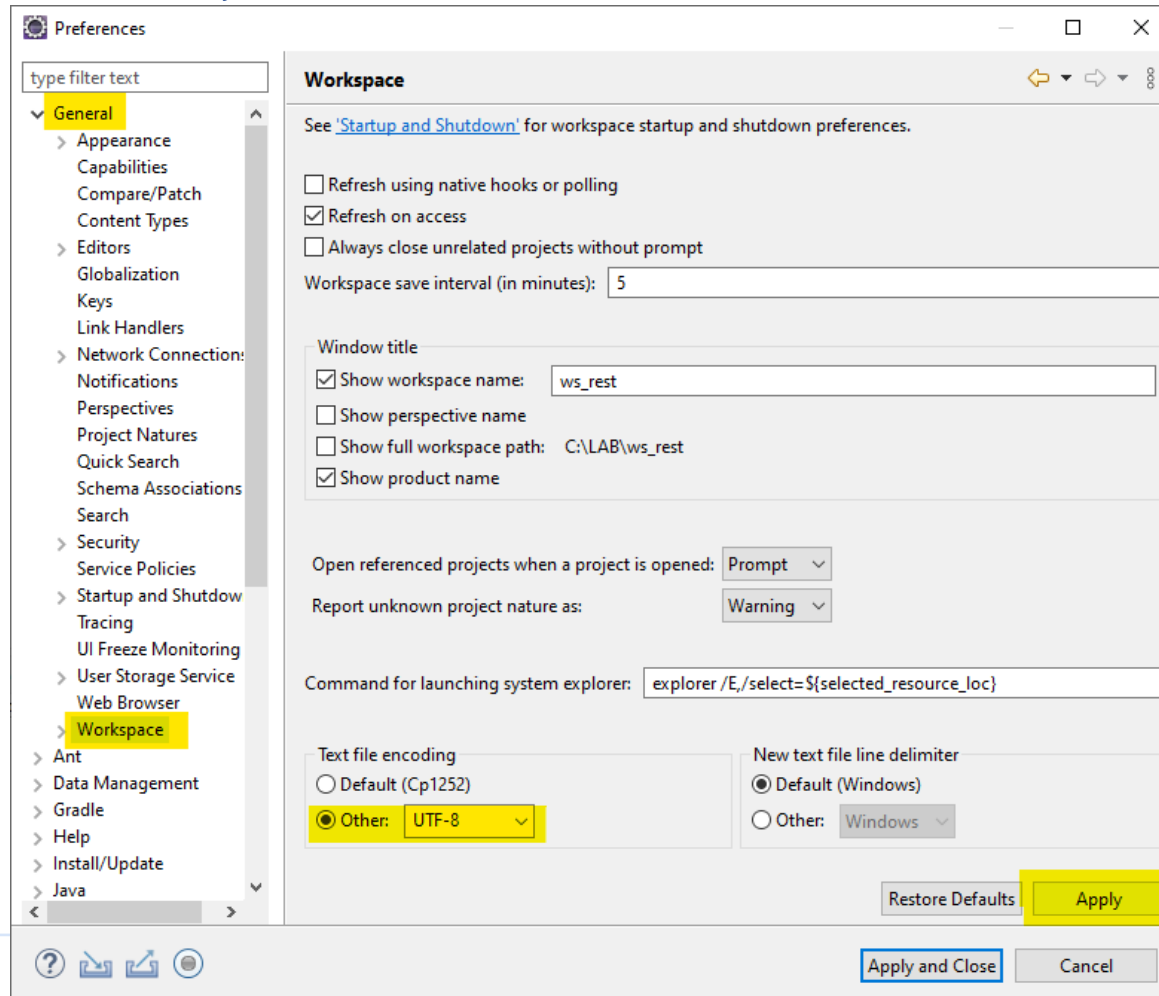
- > Criar pasta c:\LAB
- > Baixar e instalar Java Development Kit 11
- > Baixar última versão Eclipse: **Eclipse IDE for Enterprise Java and Web Developers**
- > Descompactar o zip do Eclipse na pasta criada
- > Iniciar o Eclipse

## >Selecionar o workspace:



## >Configuração inicial no Eclipse (1 de 3):

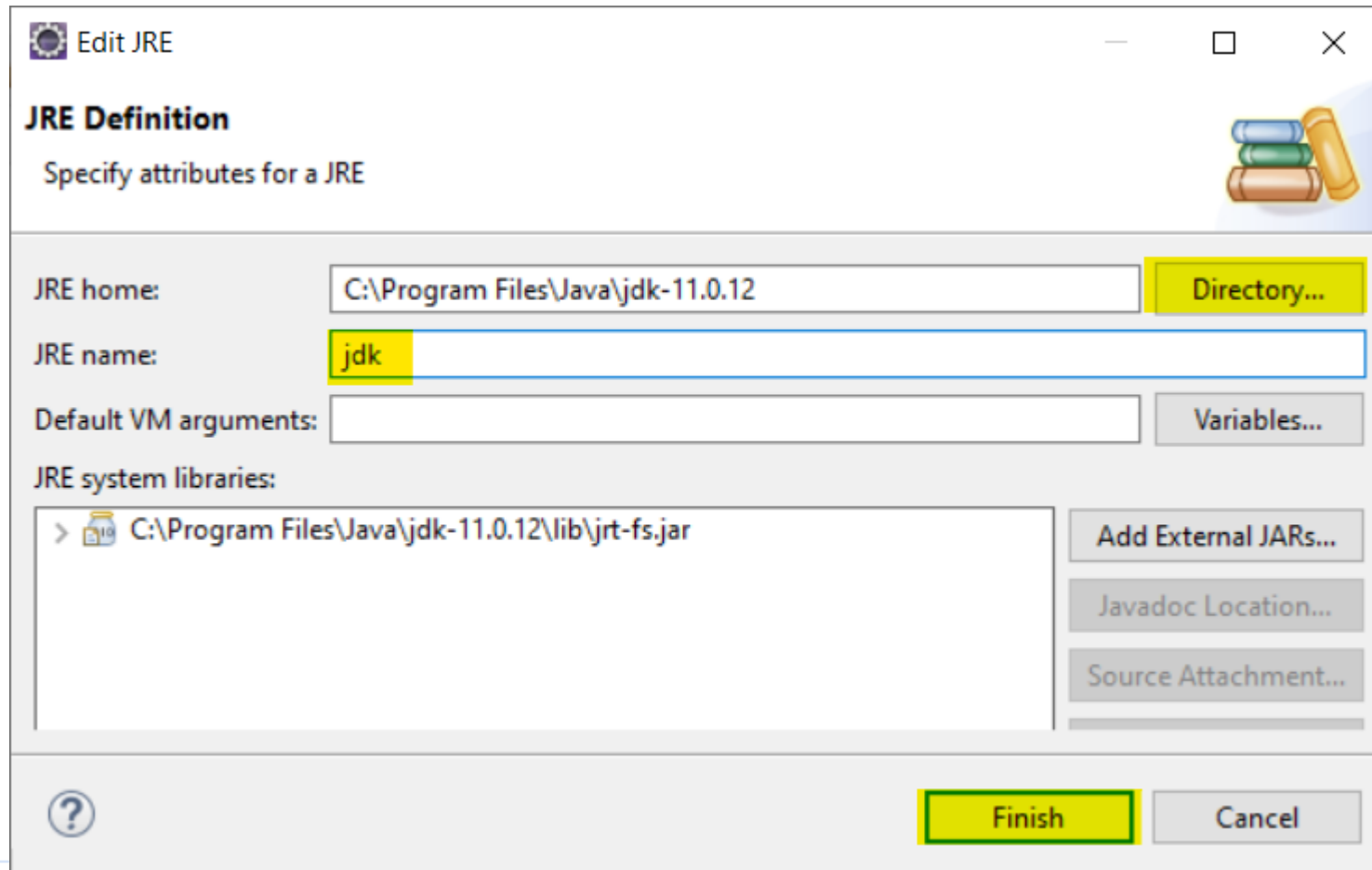
- Precisamos configurar o *Encoding* do Workspace. Clique em **Window > Preferences**.
- Nas opções da esquerda, expandir **General**, e clique em **Workspace**:



- Em Text file encoding: selecionar **Other: UTF-8**
- Clicar **Apply**

## >Configuração inicial no Eclipse (2 de 3):

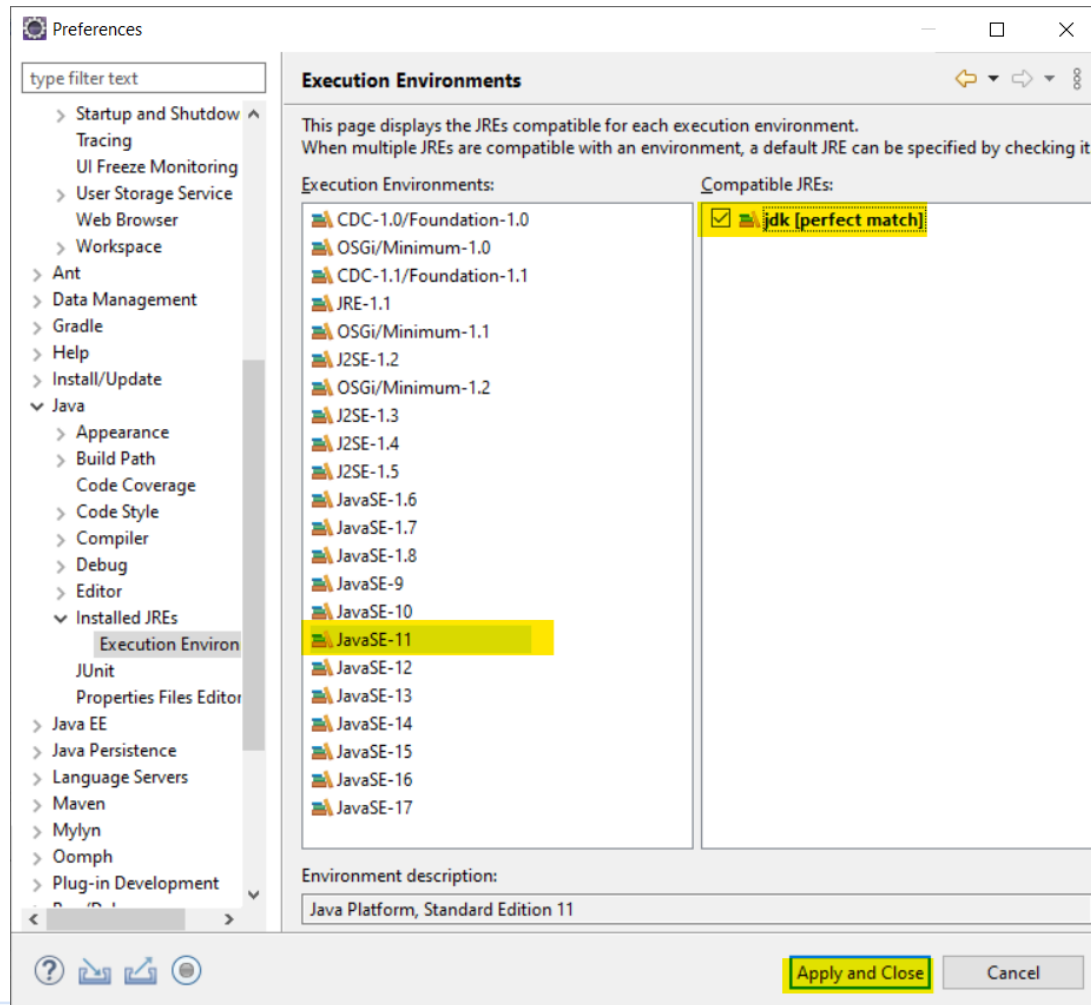
- Agora, precisamos configurar o Eclipse para usar o JDK ao invés do JRE.
- Nas opções da esquerda, expanda **Java** e clique em **Installed JRE's**:



- Clicar **Directory** para selecionar a pasta onde o JDK foi instalado
- No campo JRE Name, digitar **jdk**
- Clicar **Finish**

## >Configuração inicial no Eclipse (3 de 3):

- Na coluna lateral da esquerda, expanda “Installed JREs” e selecione “Execution Environment”




- Na área “Execution Environments”, selecionar **JavaSE-11**.
- Na área “Compatible JREs”, marcar **jdk (perfect match)**
- Clicar **Apply and Close**





- > Projeto do **Spring Framework** para tornar o **desenvolvimento extremamente simples**, tanto Web tanto REST
- > Basea-se no conceito CoC -> **Convention Over Configuration**
- > **Configuração de projeto intuitivo**: selecionamos as dependências, um zip é gerado e importamos pelo Eclipse
- > **Evita toda confusão do XML do Maven**, bibliotecas, versões, configurações.
- > Vem com Tomcat embutido


**spring** initializr

**Project**

☒ Maven Project
 ☐ Gradle Project

**Language**

☒ Java ☐ Kotlin
 ☐ Groovy

**Spring Boot**

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M1) ☐ 2.7.0 (SNAPSHOT)
 ☐ 2.7.0 (M2) ☐ 2.6.5 (SNAPSHOT) ☒ 2.6.4
 ☐ 2.5.11 (SNAPSHOT) ☐ 2.5.10

**Project Metadata**

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 17 ☒ 11 ☐ 8

**Dependencies**

ADD DEPENDENCIES... CTRL + B

**Spring Web** WEB
 

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot DevTools** DEVELOPER TOOLS
 

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

GENERATE CTRL + G

EXPLORE CTRL + SPACE

SHARE...

Executando esta classe, o Spring Boot é iniciado

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Run As > Java Application

- Desde o Spring Boot 1.2.0, podemos usar somente a anotação **@SpringBootApplication**

Ela é combinação de outras três anotações:

**@Configuration,**

**@EnableAutoConfiguration,**

**@ComponentScan,**

com atributos default

<https://start.spring.io/>

<https://spring.io/learn>

<https://www.baeldung.com/>

<https://www.alura.com.br/>

>Acessar **start.spring.io**

>Preencher o formulário segundo a tabela abaixo:

Project	<b>Maven Project</b>
Language	<b>Java</b>
Spring Boot	<b>2.6.4</b>

<b>Dependencies</b>
Spring Web
Spring Boot Dev Tools

Project Metadata:	Group	<b>br.inatel.myrestapi</b>
	Artifact	<b>MyRestAPI</b>
	Name	<b>MyRestAPI</b>
	Description	<b>Minha API Rest</b>
	Package name	<b>br.inatel.myrestapi</b>
	Packaging	<b>Jar</b>
	Java	<b>11</b>

>Clicar em **Generate**

>Descompactar zip para a pasta do repositório do Eclipse

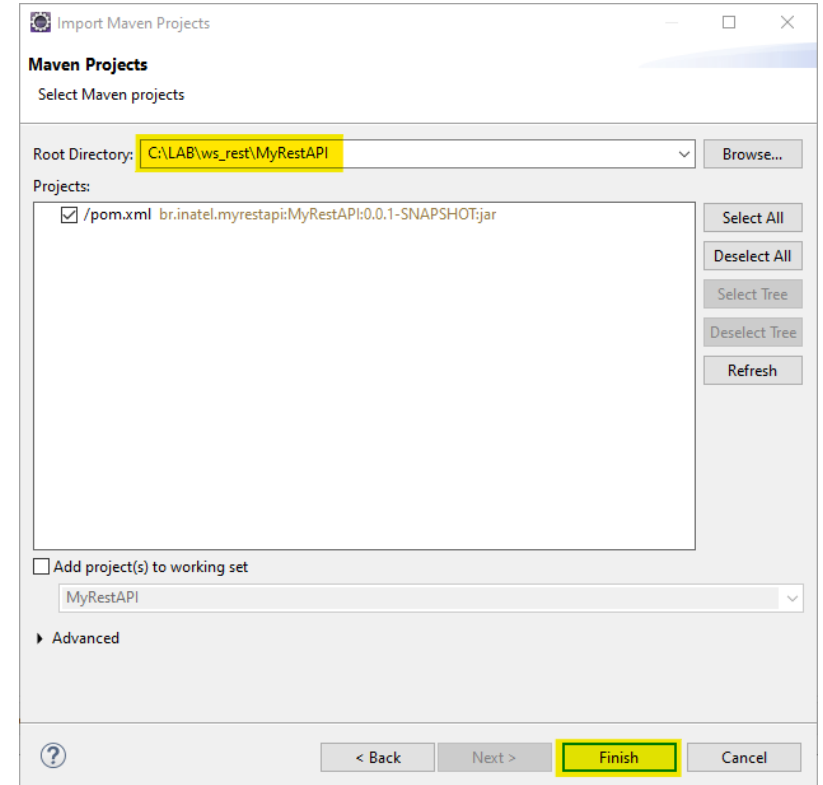
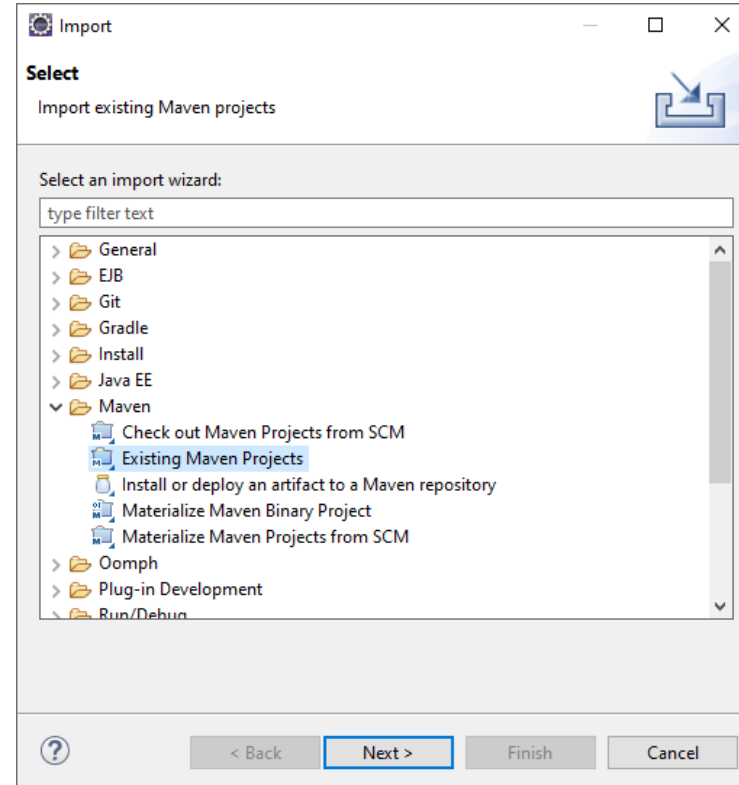
>No Eclipse, importar como projeto Maven:

**File > Import**

>Expandir **Maven > Existing Maven Projects**

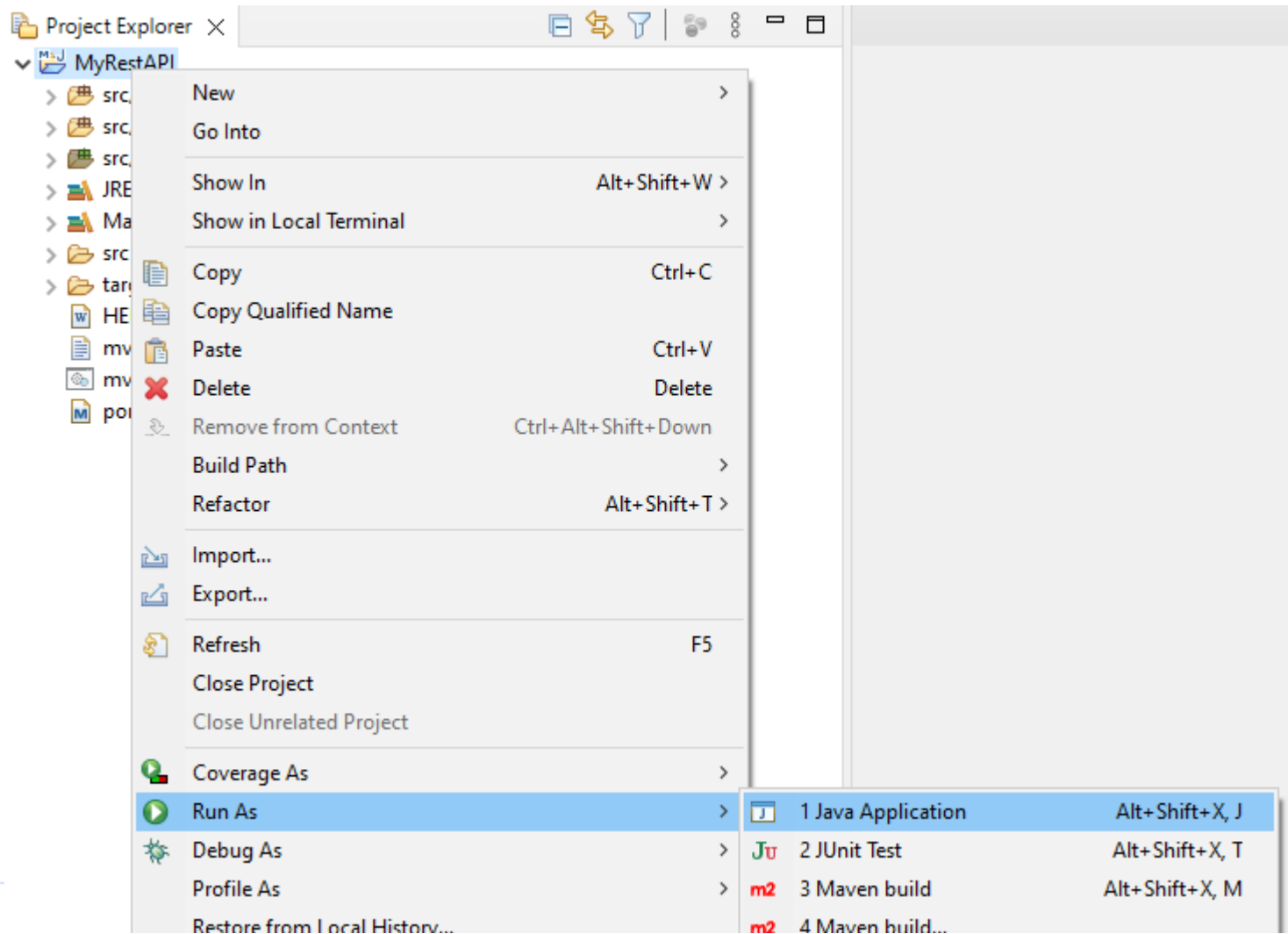
>Clicar **Finish**

**Importante:**  
Aguardar o build do projeto



>Subir a aplicação:

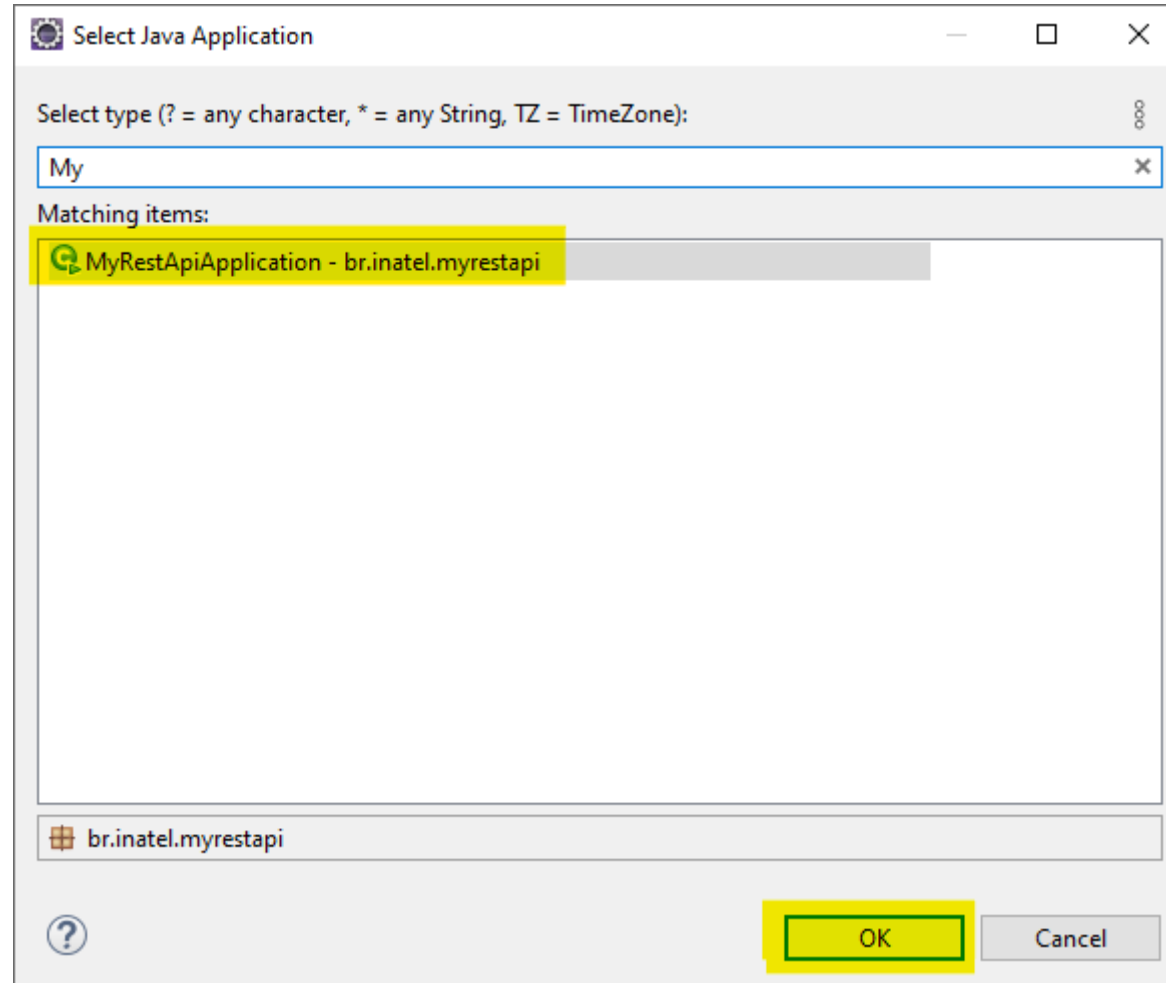
a) Na aba Project Explorer, clique da direita no projeto > **Run As** > **Java Application**



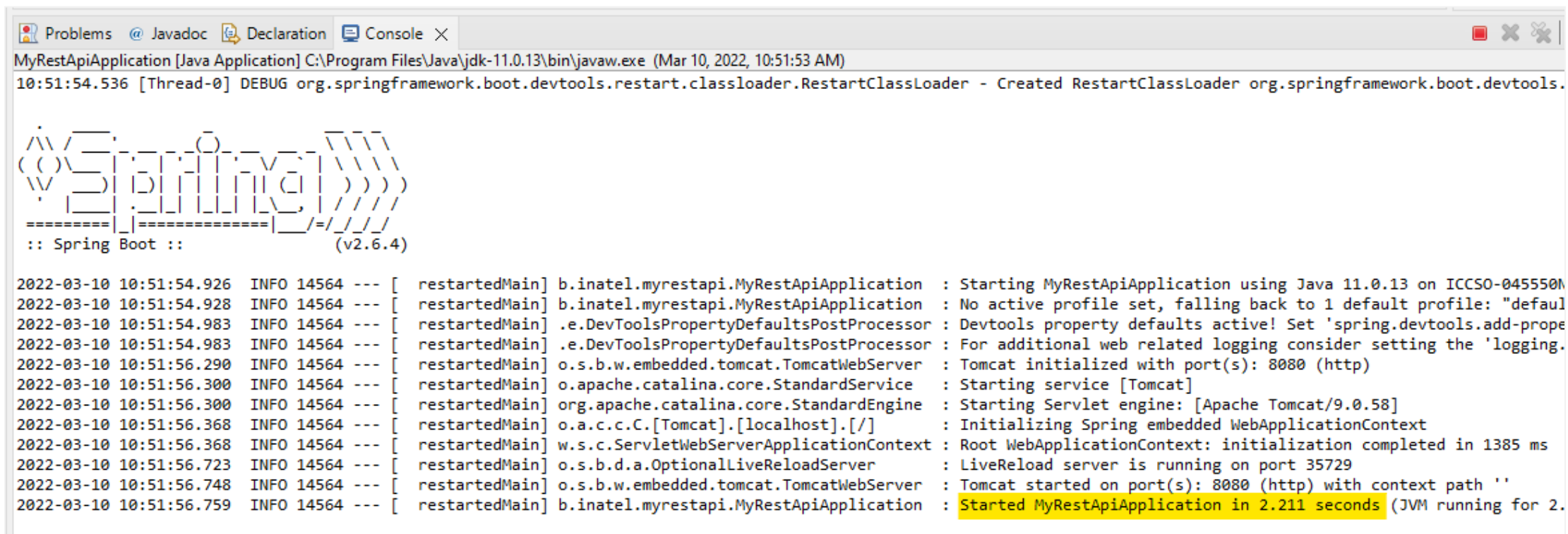


b) Selecionar **MyRestApiApplication** (use o filtro)

c) Clicar **OK**



## d) Observar a saída do console:



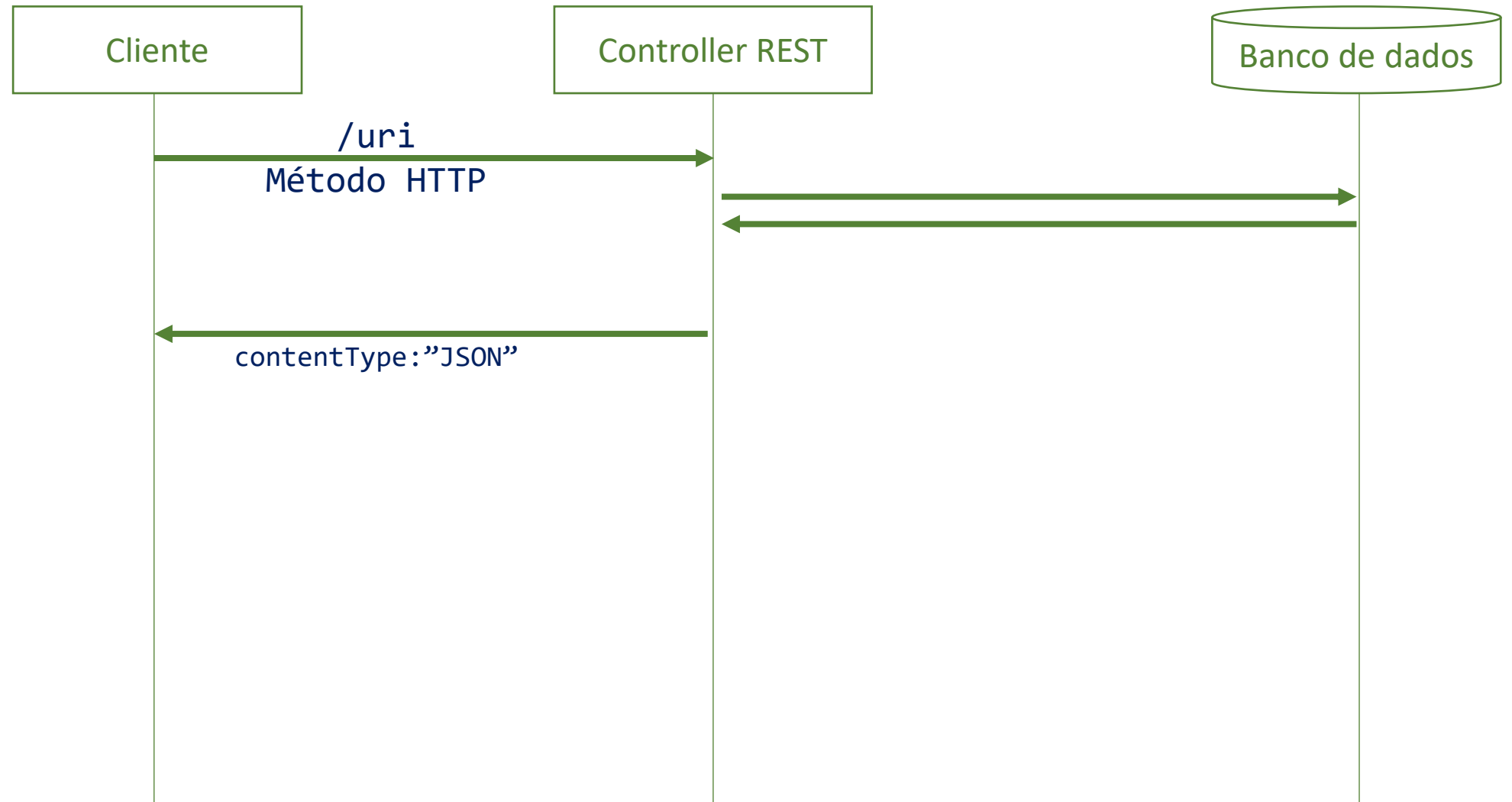
```

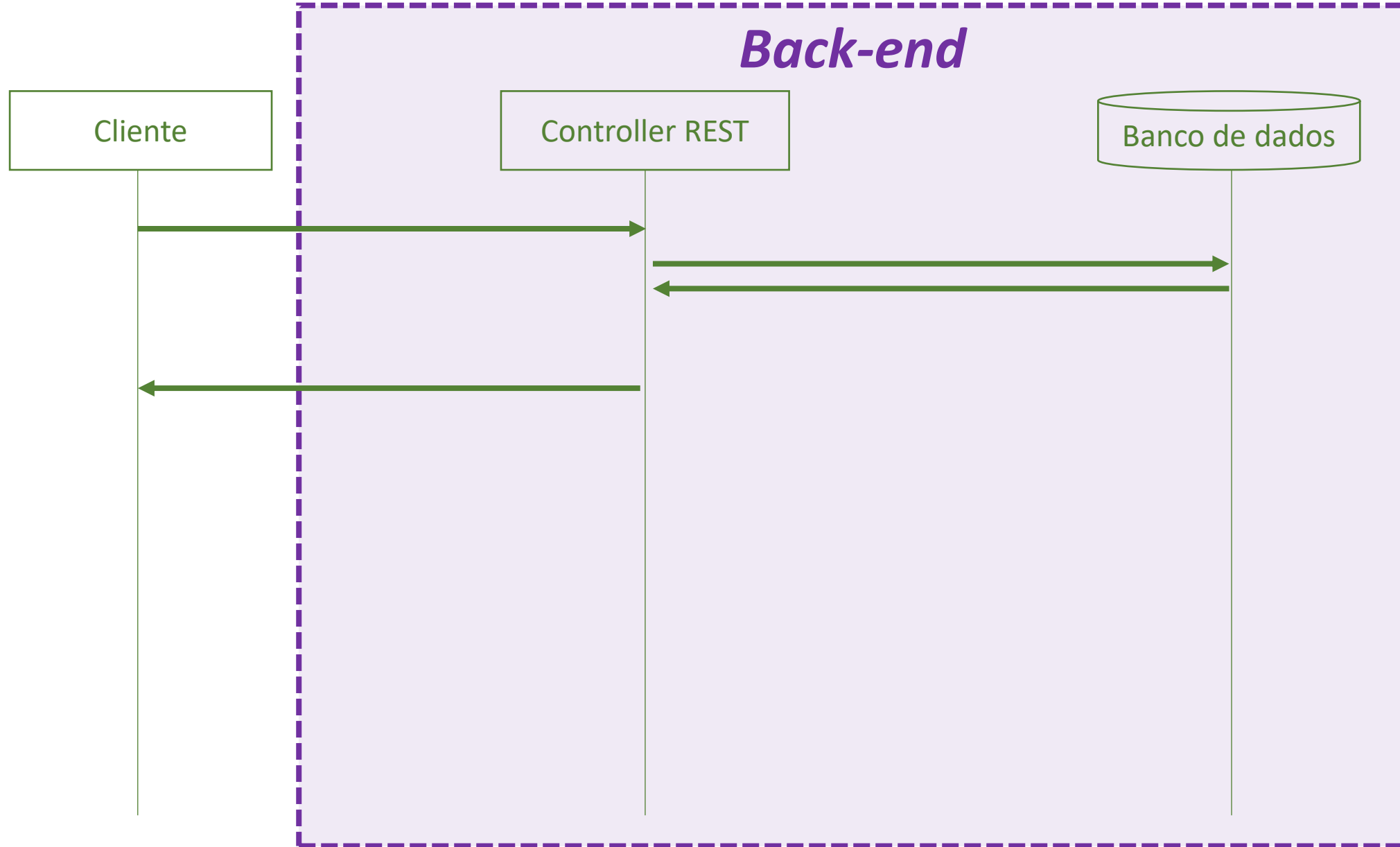
Problems @ Javadoc Declaration Console ×
MyRestApiApplication [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (Mar 10, 2022, 10:51:53 AM)
10:51:54.536 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.

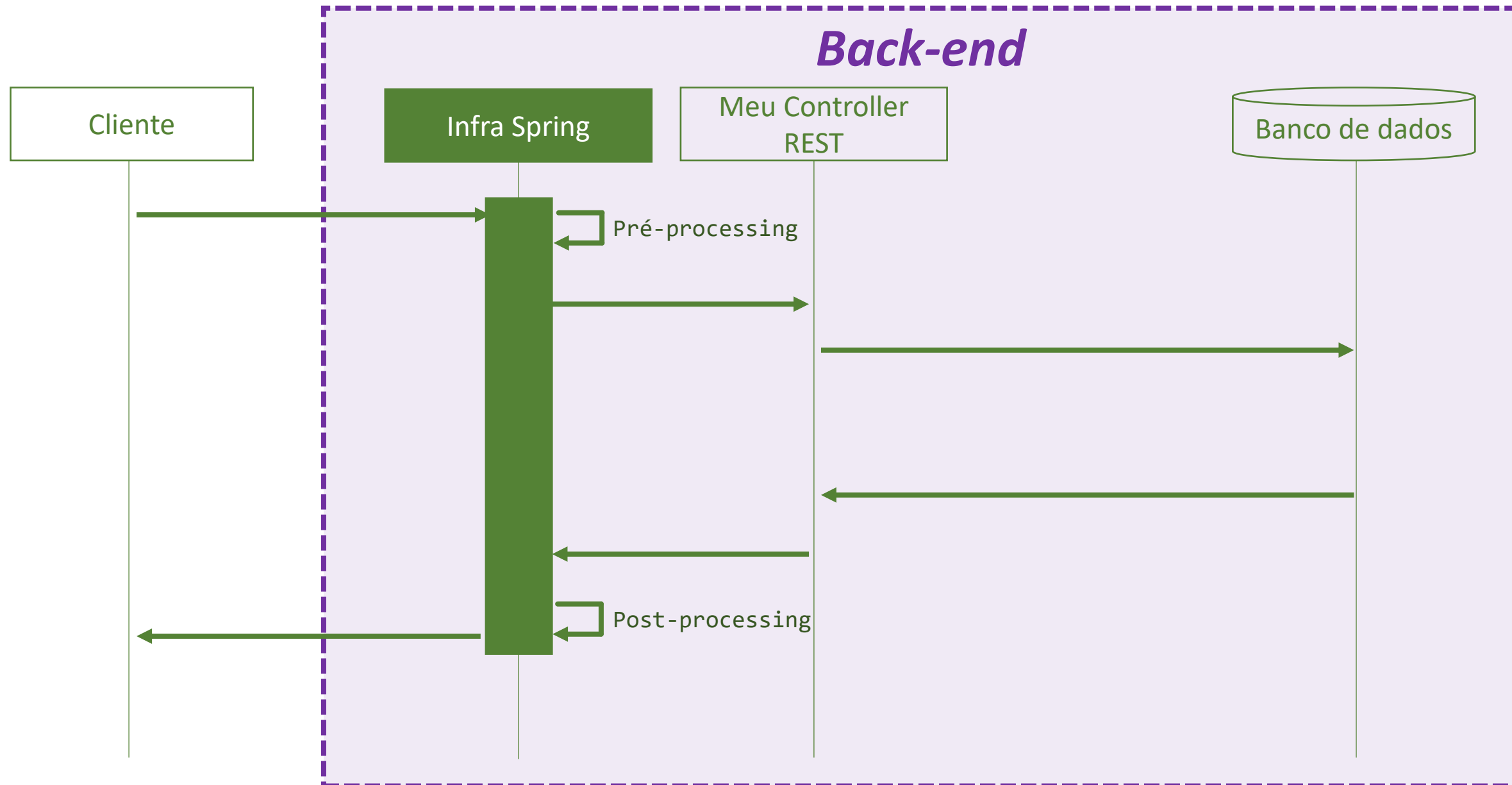
:: Spring Boot :: (v2.6.4)

2022-03-10 10:51:54.926 INFO 14564 --- [ restartedMain] b.inatel.myrestapi.MyRestApiApplication : Starting MyRestApiApplication using Java 11.0.13 on ICCSO-045550M
2022-03-10 10:51:54.928 INFO 14564 --- [ restartedMain] b.inatel.myrestapi.MyRestApiApplication : No active profile set, falling back to 1 default profile: "default"
2022-03-10 10:51:54.983 INFO 14564 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-prope
2022-03-10 10:51:54.983 INFO 14564 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.
2022-03-10 10:51:56.290 INFO 14564 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-03-10 10:51:56.300 INFO 14564 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-03-10 10:51:56.300 INFO 14564 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.58]
2022-03-10 10:51:56.368 INFO 14564 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2022-03-10 10:51:56.368 INFO 14564 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1385 ms
2022-03-10 10:51:56.723 INFO 14564 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2022-03-10 10:51:56.748 INFO 14564 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-03-10 10:51:56.759 INFO 14564 --- [ restartedMain] b.inatel.myrestapi.MyRestApiApplication : Started MyRestApiApplication in 2.211 seconds (JVM running for 2.
  
```

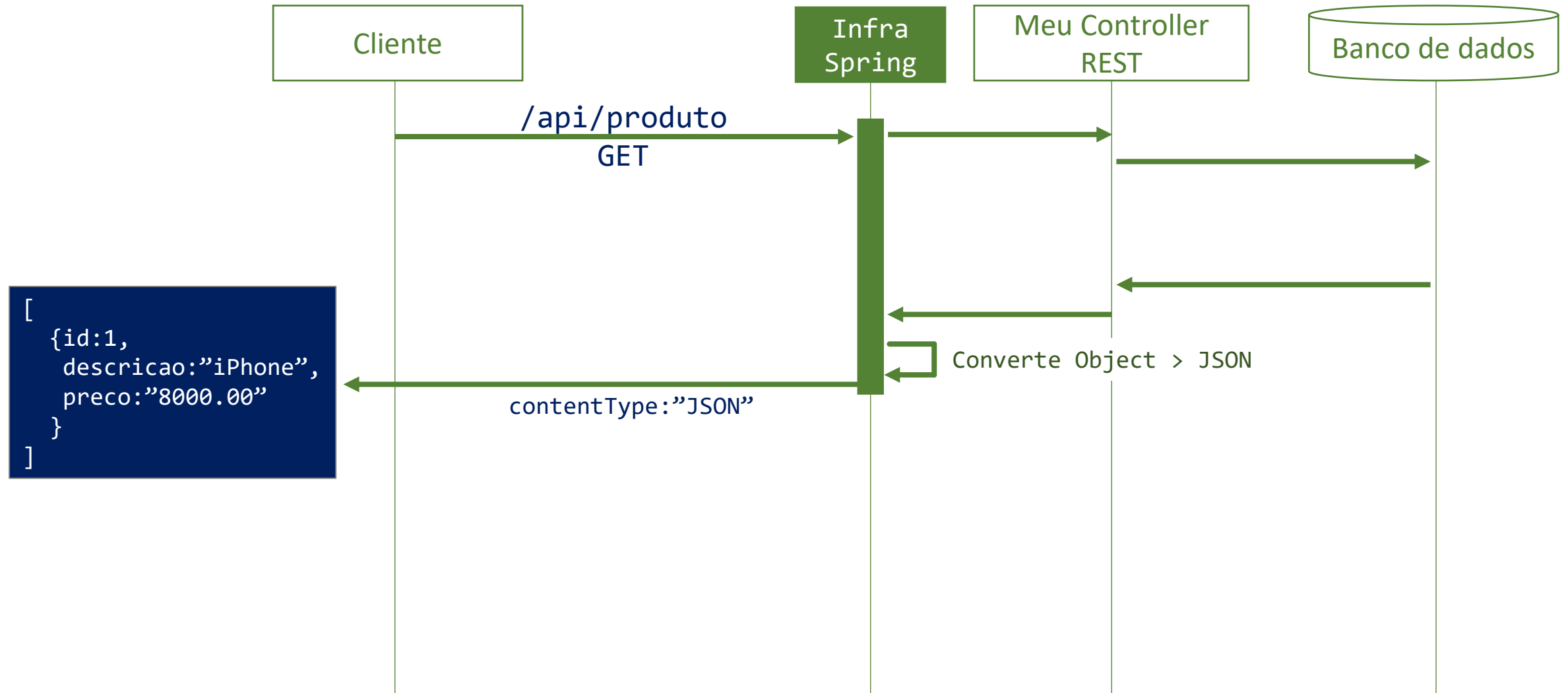
# *Arquitetura REST no Spring*

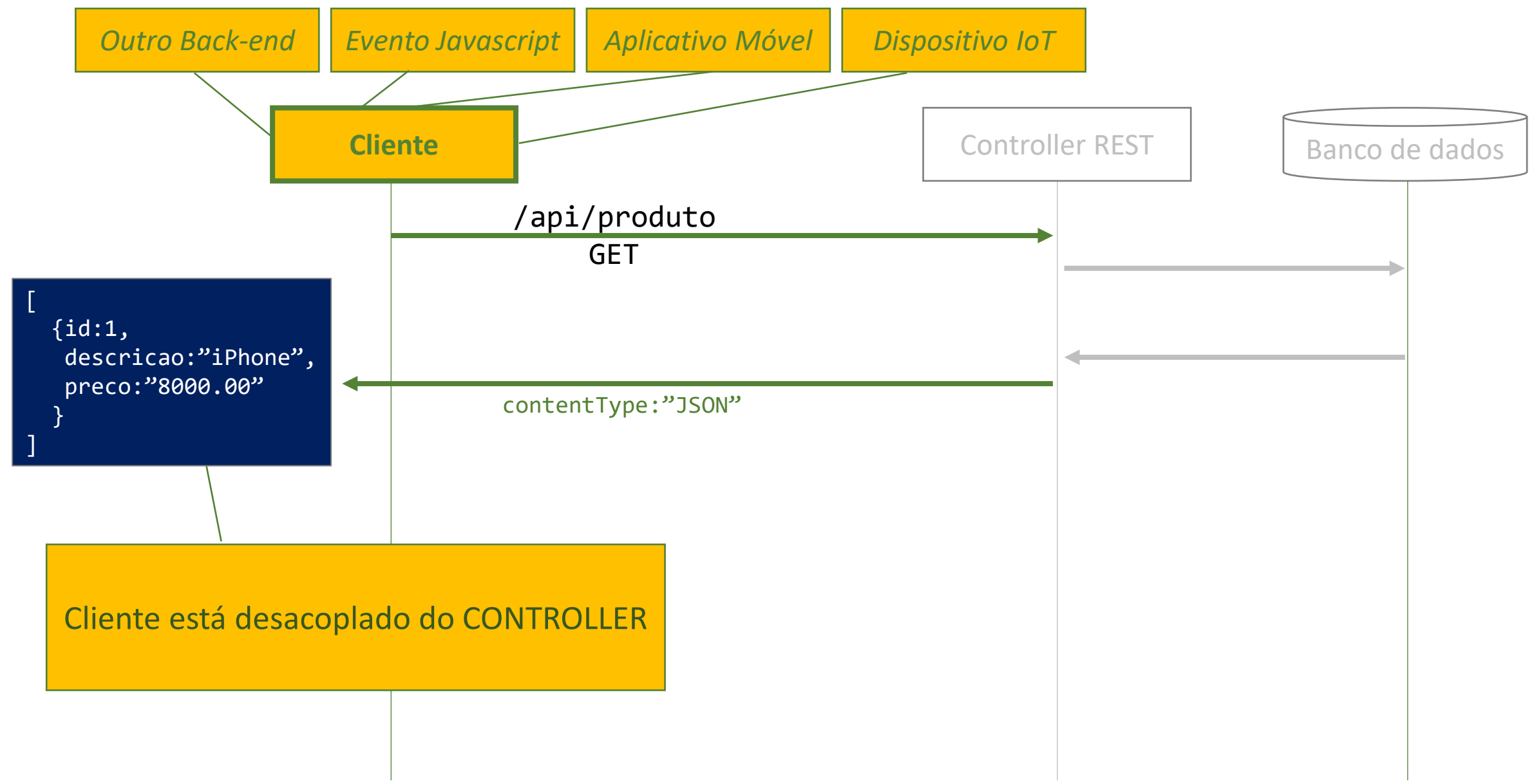




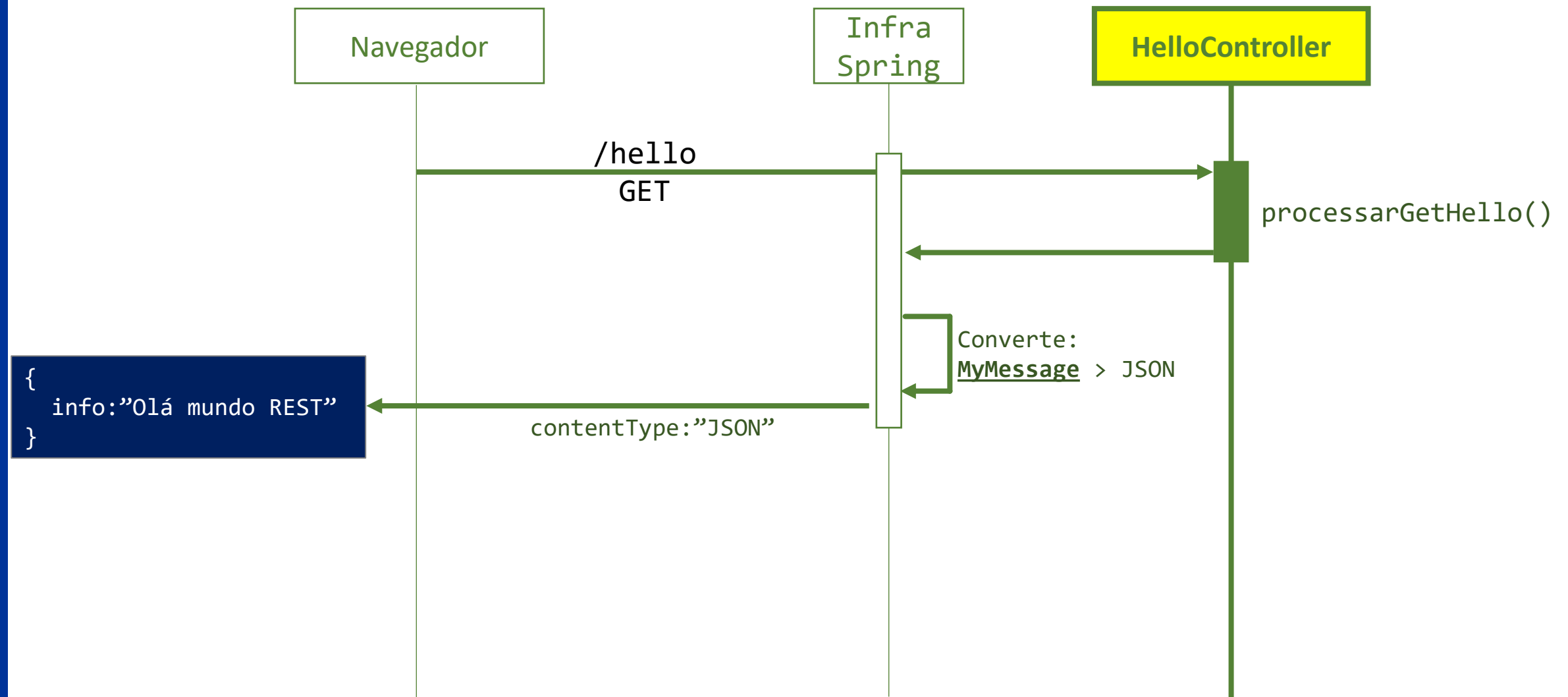


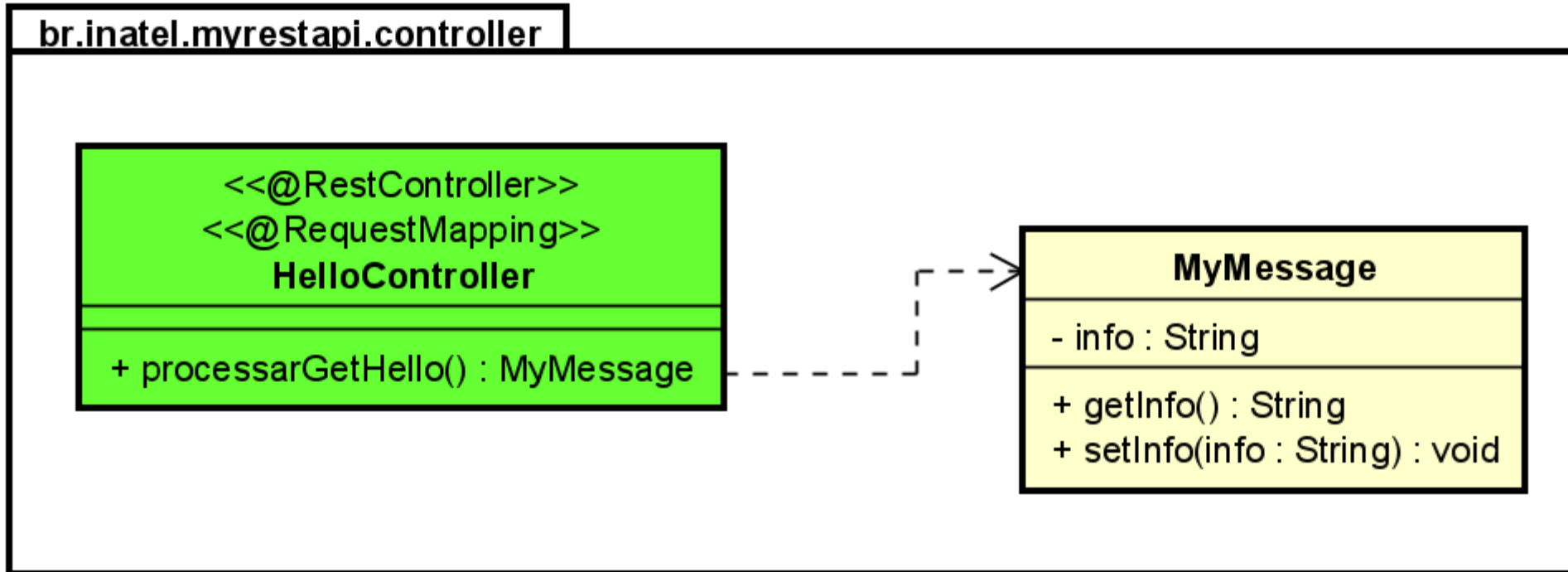
## Controller REST











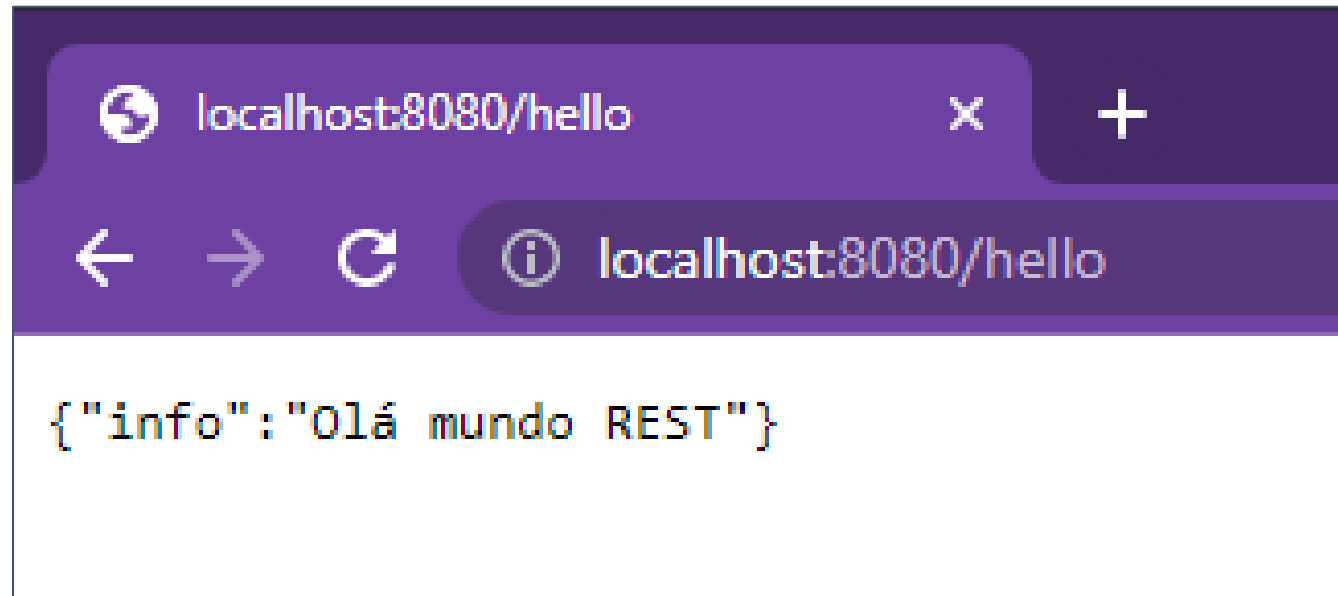
> Criar pacote **br.inatel.myrestapi.controller**:

a) Criar classe **MyMessage** e codificar conforme UML

b) Criar classe **HelloController** conforme abaixo:

```
HelloController.java ×
1 package br.inatel.myrestapi.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 @RequestMapping("/hello")
9 public class HelloController {
10
11     @GetMapping
12     public MyMessage processarGetHello() {
13         MyMessage msg = new MyMessage();
14         msg.setInfo("Olá mundo REST");
15         return msg;
16     }
17
18 }
19
```

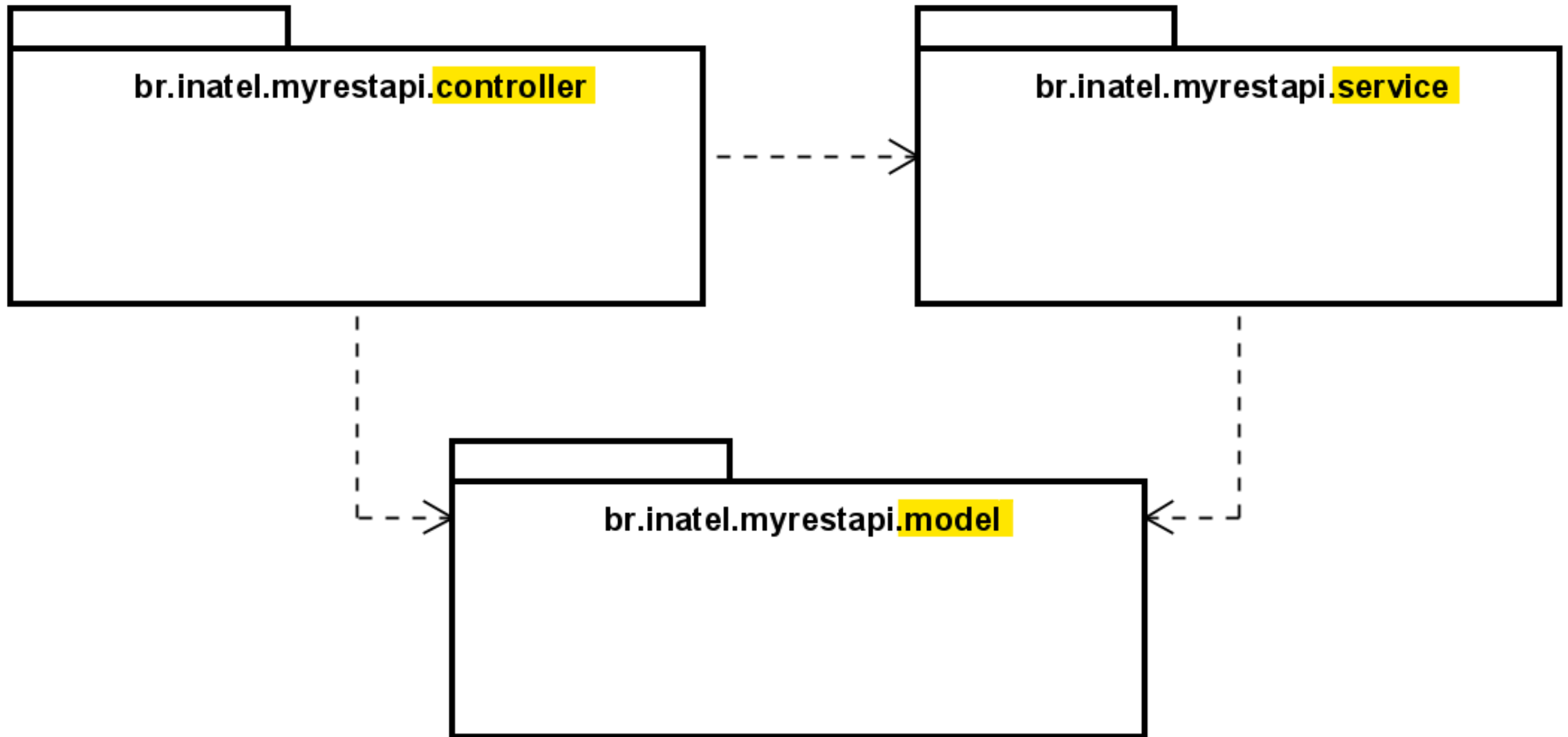
>Abrir o navegador e acessar: **localhost:8080/hello**

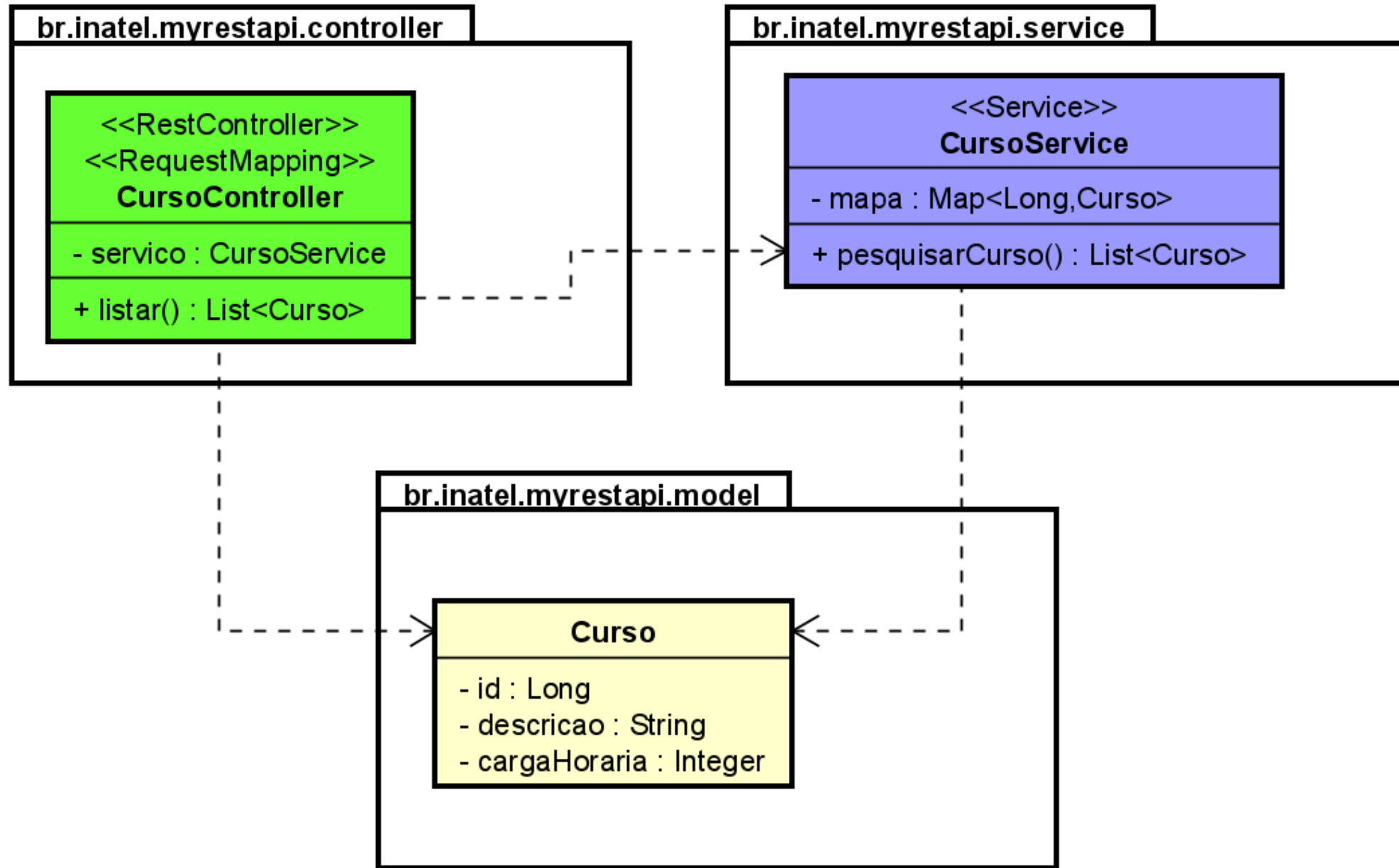


**Questão: Quem fez a conversão MyMessage > JSON?**

**Resposta: A infraestrutura do Spring**

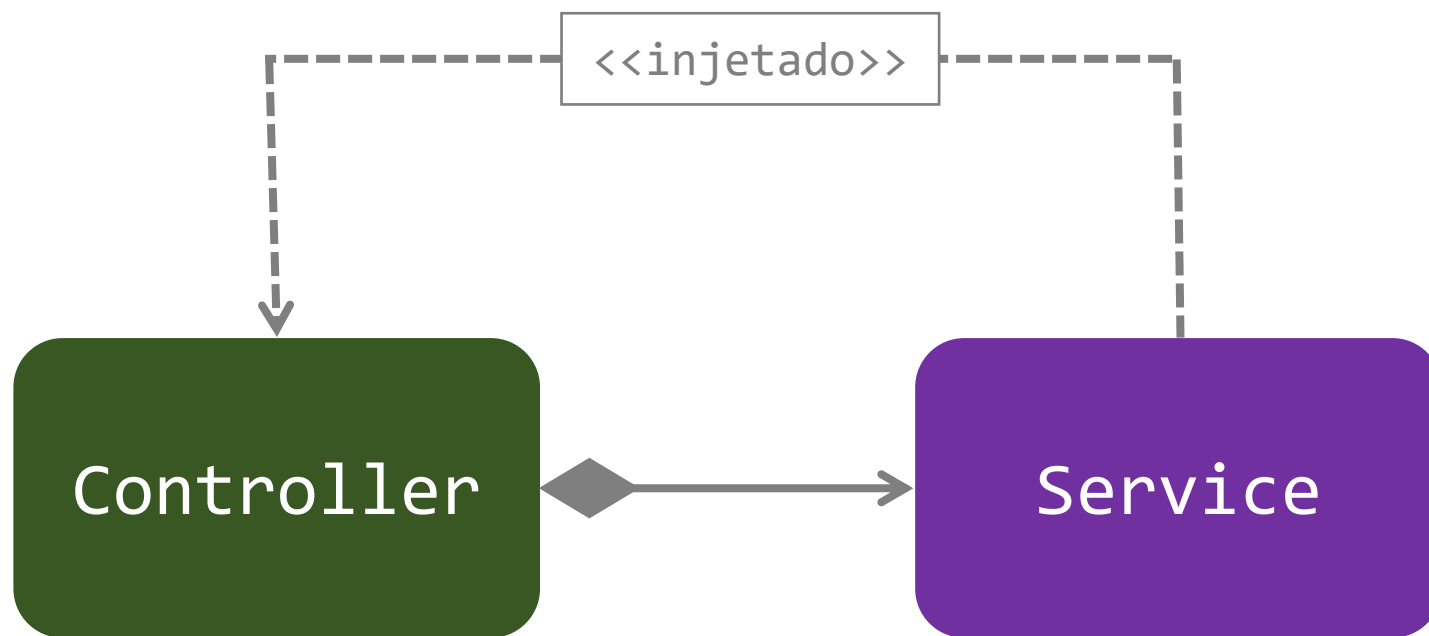
*Implementando um back-end  
completo de catálogo de cursos*





Sempre que 2 componentes dependentes são gerenciados pelo Spring, um deve ser injetado pelo outro

**Injeção de dependência** no Spring é feita pela anotação @AutoWired





## model.Curso

```
Curso.java X
1 package br.inatel.myrestapi.model;
2
3 public class Curso {
4
5     private Long id;
6
7     private String descricao;
8
9     private Integer cargaHoraria;
10
11     //construtor gerado com Ctrl + 3 > 'gcuf' (Generate Constructor Using Fields)
12
13     public Curso(Long id, String descricao, Integer cargaHoraria) {
14         super();
15         this.id = id;
16         this.descricao = descricao;
17         this.cargaHoraria = cargaHoraria;
18     }
19
20     //getters e setters gerados com Ctrl + 3 > 'ggas' (Generate Getters And Setters)
21 }
```

Tarefa: gerar  
construtor default

## service.CursoService

```
CursoService.java X
14
15 @Service
16 public class CursoService {
17
18     private Map<Long, Curso> mapa = new HashMap<>();
19
20
21     public List<Curso> pesquisarCurso() {
22         return mapa.entrySet().stream()
23             .map(m -> m.getValue() )
24             .collect(Collectors.toList());
25     }
26
```

## controller.CursoController

```
CursoController.java ×
12
13 @RestController
14 @RequestMapping("/curso")
15 public class CursoController {
16
17     @Autowired
18     private CursoService servico;
19
20
21     @GetMapping
22     public List<Curso> listar() {
23         List<Curso> listaCurso = servico.pesquisarCurso();
24         return listaCurso;
25     }
26
27 }
28
```

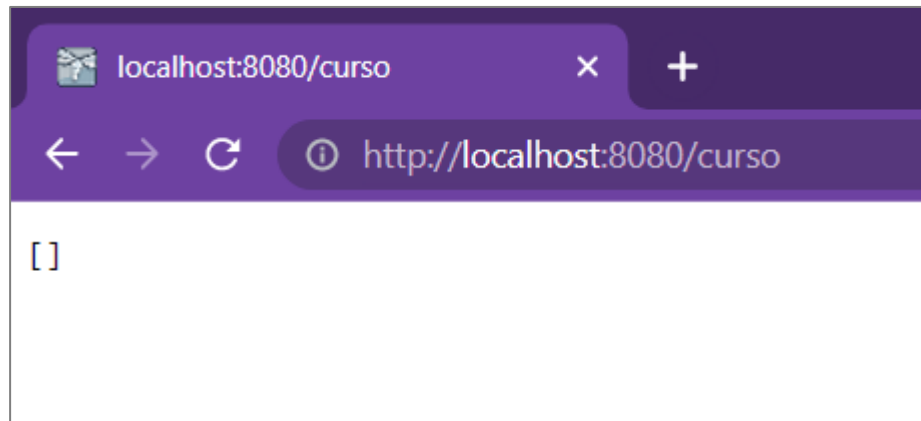
>Seguindo os slides anteriores:

a)Criar os sub-pacotes **model** e **service**

b)Implementar as 3 classes do catálogo de cursos

c)Subir o Spring Boot

d)Abrir navegador e acessar: **localhost:8080/curso**



É esperado que não retorne  
nenhum resultado

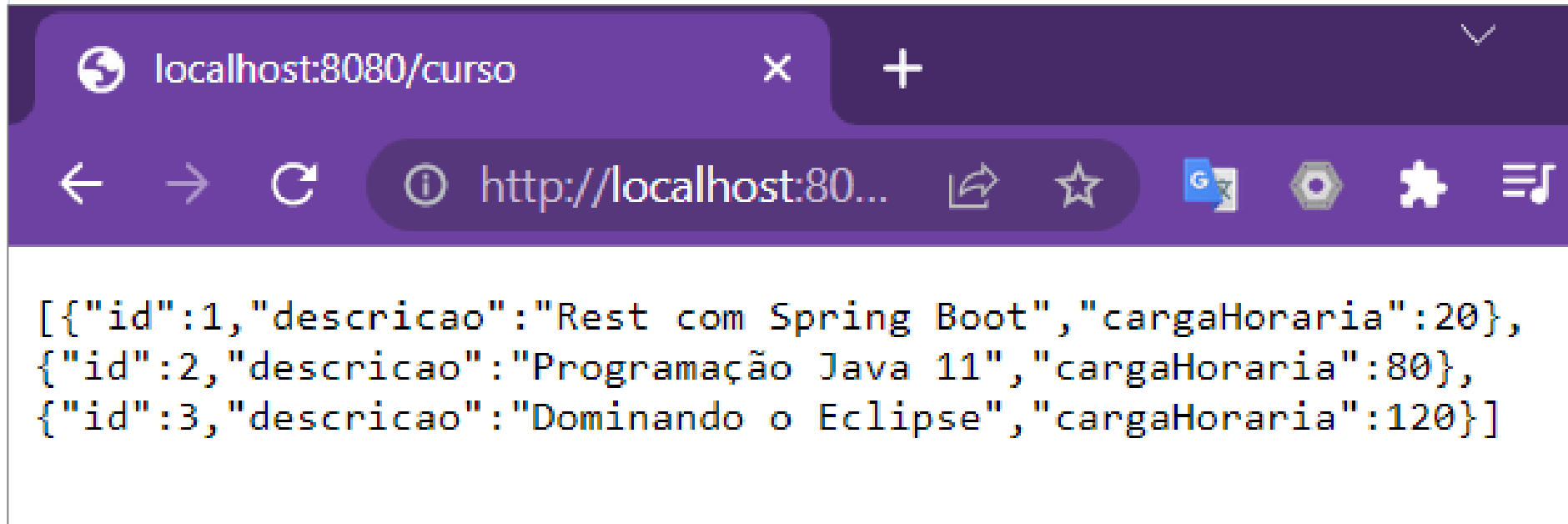
>Vamos implementar um método na classe service para **inicializar o mapa com alguns cursos**:

```

CursoService.java ×
14 @Service
15 public class CursoService {
16
17     private Map<Long, Curso> mapa = new HashMap<>();
18
19     @PostConstruct
20     private void inicializarMapa() {
21         Curso c1 = new Curso(1L, "Rest com Spring Boot", 20);
22         Curso c2 = new Curso(2L, "Programação Java 11", 80);
23         Curso c3 = new Curso(3L, "Dominando o Eclipse", 120);
24
25         mapa.put(c1.getId(), c1);
26         mapa.put(c2.getId(), c2);
27         mapa.put(c3.getId(), c3);
28     }
29

```

>No navegador, novamente acessar: **localhost:8080/curso**



*Buscando um curso pela chave primária*

> Outra possível operação de leitura seria a **busca de um curso através de sua chave primária**

> Vamos implementar esta funcionalidade:

a) Na classe service, criamos um método que recebe o parâmetro referente ao ID de curso e retorna o curso guardado no mapa:

```
15 @Service
16 public class CursoService {
17
18     private Map<Long, Curso> mapa = new HashMap<>();
19
20
21     public Curso buscarCursoPeloId(Long cursoId) {
22         Curso curso = mapa.get( cursoId );
23         return curso;
24     }
25
26 }
```




b)Na classe controller, mapeamos outro método com @GetMapping e uma variável de path:

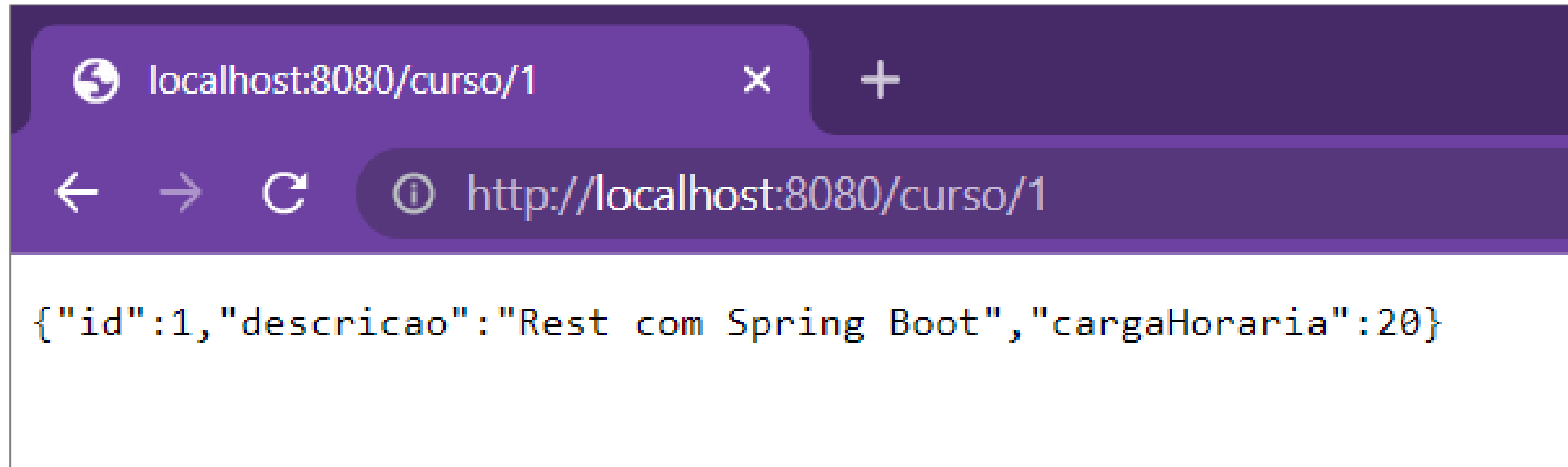
```
16 public class CursoController {  
17  
18     @Autowired  
19     private CursoService servico;  
20  
21  
22     @GetMapping("/{id}")  
23     public Curso buscar(@PathVariable("id") Long cursoId) {  
24         Curso curso = servico.buscarCursoPeloId(cursoId);  
25         return curso;  
26     }  
27
```

b) Na classe controller, mapeamos outro método com @GetMapping e uma variável de path:

```
16 public class CursoController {
17
18     @Autowired
19     private CursoService servico;
20
21
22     @GetMapping("/{id}")
23     public Curso buscar(@PathVariable("id") Long cursoId) {
24         Curso curso = servico.buscarCursoPeloId(cursoId);
25         return curso;
26     }
27 }
```



c) Usando o navegador, concatenamos o id do curso na própria URI:



>Seguindo os slides:

a)Implementar em **CursoService** o método **buscarCursoPeloid(...)**

b)Implementar em **CursoController**, o método **buscar(...)**

>No navegador, acessar diferente IDs:

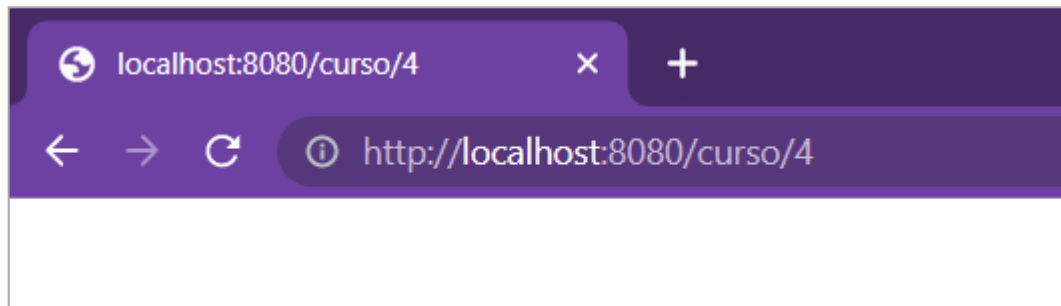
<http://localhost:8080/curso/1>

<http://localhost:8080/curso/2>

<http://localhost:8080/curso/3>

>O que acontece ao acessar <http://localhost:8080/curso/4> ?

> Quando acessamos um ID inexistente, uma resposta vazia é devolvida.



> Isso pode causar **confusão** para o cliente da API

*> ele pode interpretar que aconteceu um erro no servidor.*

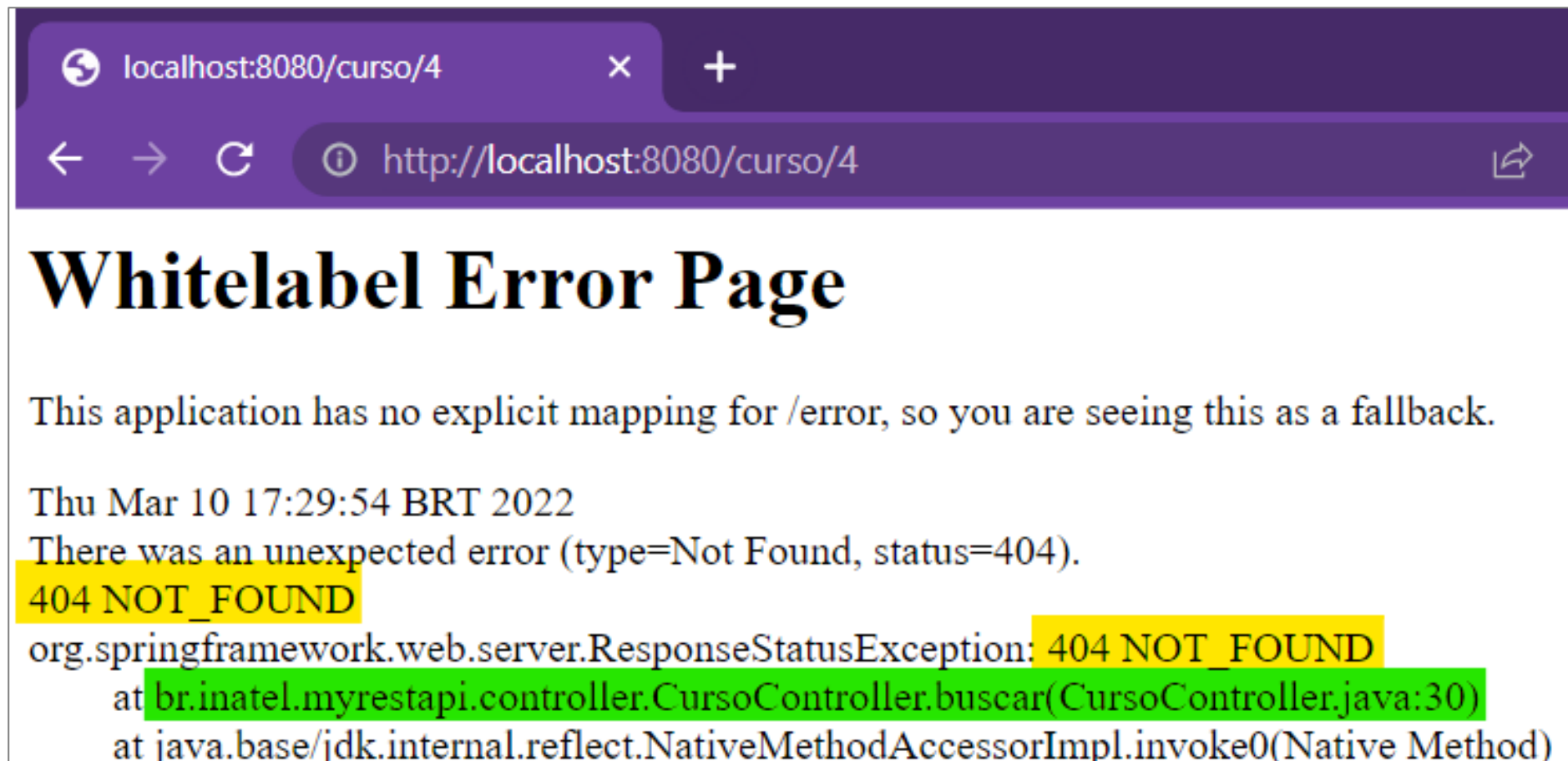
> Podemos adicionar um **status na resposta** sinalizando que tudo ocorreu bem, mas nada foi encontrado!

> O status **404 (NOT\_FOUND)** é o ideal para este cenário

>A maneira mais simples de retornar um status 404 é lançar uma exception própria para tal: **ResponseStatusException**

```
@GetMapping("/{id}")
public Curso buscar(@PathVariable("id") Long cursoId) {
    Curso curso = servico.buscarCursoPeloId(cursoId);
    if (curso != null) {
        return curso;
    }
    throw new ResponseStatusException(HttpStatus.NOT_FOUND);
}
```

>Ao acessar a API com um ID inexistente, receberá esta resposta:



- >No controller, alterar o método **buscar** para retornar o status 404 quando o ID não tem referência a um curso existente.
- >No navegador, acessar a URI com ID inexistente e inspecionar o resultado
- >Exercício avançado: Explorar os outros construtores de **ResponseStatusException**.



## *Completando o back-end de cursos*

A gestão completa de cursos consiste ainda em operações REST para:

- a) criar curso
- b) atualizar curso
- c) remover um curso específico

Este um conjunto básico de operações e comumente chamamos de **CRUD** (**C**reate, **R**etrieve, **U**ppdate, **D**elete)

## a) Criar um curso:

controller.**CursoController**

```
@PostMapping
public Curso criar(Curso curso) {
    curso = servico.criarCurso(curso);
    return curso;
}
```



Retorna o curso para o cliente

service.**CursoService**

```
public Curso criarCurso(Curso curso) {
    Long cursoId = gerarCursoIdUnico();
    curso.setId(cursoId);

    mapa.put(cursoId, curso);
    return curso;
}
```



Como gerar IDs únicos?

## b) Atualizar um curso:

controller.**CursoController**

```
@PutMapping  
public void atualizar(Curso curso) {  
    servico.atualizarCurso(curso);  
}
```

Não precisa de retorno na atualização

service.**CursoService**

```
public void atualizarCurso(Curso curso) {  
    mapa.put(curso.getId(), curso);  
}
```

## c) Remover um curso:

controller.**CursoController**

```
@DeleteMapping("/{id}")  
public void remover(@PathVariable("id") Long idParaRemover) {  
    servico.removerCursoPeloId(idParaRemover);  
}
```

service.**CursoService**

```
public void removerCurso(Long cursoId) {  
    mapa.remove(cursoId);  
}
```

>Se guiando pelos slides anteriores:

- a) **Implementar os métodos na classe service**
- b) **Implementar os métodos na classe controller**

## *Acessando APIs REST com Spring*

- >O navegador somente dá suporte para o método HTTP GET
- >Para acessar todas as operações da nossa API, precisamos de uma das opções:
  - a)Ferramenta específica para acessar APIs
  - b)Escrever código que acessam nossa API -> usando uma biblioteca



## RestTemplate

Bloqueante  
(sincrôna)

*Antiga: < Spring 4*

*Deprecated*

## WebClient

Não-bloqueante  
(assíncrona)

Spring 5+

Java 9 Reactive Streams API

Tipo reativos: **Mono** e **Flux**

>A classe WebClient faz parte da biblioteca **Spring WebFlux**

>Para adicionar esta biblioteca no projeto:

1)Adicionar dependência no **pom.xml**:

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

## Spring WebFlux:

### 2)Codificar:

```
List<Curso> listaCurso = new ArrayList<Curso>();
```

```
Flux<Curso> flux = WebClient.create()  
    .get()  
    .uri("localhost:8080/curso")  
    .retrieve()  
    .bodyToFlux(Curso.class)  
    ;
```

```
flux.subscribe(c -> listaCurso.add(c) );  
flux.blockLast();
```

Este trecho é executado  
numa **thread** separada

## Spring WebFlux:

### 2)Codificar:

```
List<Curso> listaCurso = new ArrayList<Curso>();

Flux<Curso> fluxCurso = WebClient.create("localhost:8080/curso")
    .get()
    .retrieve()
    .bodyToFlux(Curso.class)
    ;

fluxCurso.subscribe( c -> listaCurso.add(c) );

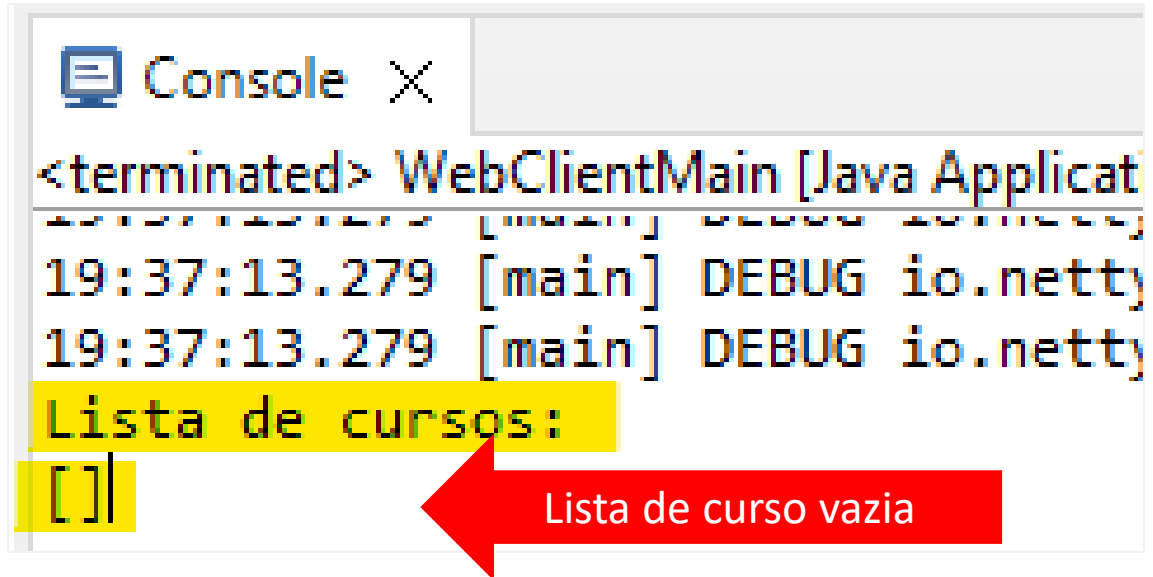
System.out.println("Lista de cursos: ");
System.out.println( listaCurso );
```

Este trecho é executado  
numa **thread** separada

- > Buscar em [start.spring.io](http://start.spring.io) a dependência do WebFlux
- > Copiar o trecho e colar no pom.xml
- > Em `src/test/java`, no pacote `br.inatel.myrestapi`, criar a classe **WebClientMain**
- > No método **main**, codificar usando o seguinte código:

```
13 public static void main(String[] args) {  
14  
15     List<Curso> listaCurso = new ArrayList<Curso>();  
16  
17     Flux<Curso> flux = WebClient.create()  
18         .get()  
19         .uri("localhost:8080/curso")  
20         .retrieve()  
21         .bodyToFlux(Curso.class)  
22         ;  
23  
24     flux.subscribe(c -> listaCurso.add(c) );  
25  
26  
27     System.out.println("Lista de Cursos:");  
28     System.out.println( listaCurso );  
29 }  
30
```

>Ao executar, temos o resultado:



```
Console X
<terminated> WebClientMain [Java Applicat
19:37:13.279 [main] DEBUG io.netty
19:37:13.279 [main] DEBUG io.netty
Lista de cursos:
[]
```

Lista de curso vazia

*O que aconteceu?*

>**Solução:** adicionar na linha 24 a instrução `flux.blockLast()` para bloquear até o último registro chegar:

```
14
15     List<Curso> listaCurso = new ArrayList<Curso>();
16
17     Flux<Curso> fluxCurso = WebClient.create("localhost:8080/curso")
18         .get()
19         .retrieve()
20         .bodyToFlux(Curso.class)
21         ;
22
23     fluxCurso.subscribe( c -> listaCurso.add(c) );
24     fluxCurso.blockLast();//bloqueia até ultimo curso chegar...
25
26     System.out.println("Lista de cursos: ");
27     System.out.println( listaCurso );
28
```

*Funcionou!*

> Usamos **Flux** quando o retorno do endpoint é **zero ou muitos**: **[0 , \*]**

> Usamos **Mono** quando o retorno do endpoint é **zero ou 1**: **[0 , 1]**

> Código com **Mono**:

```
Mono<Curso> monoCurso = WebClient.create("localhost:8080/curso/2")
    .get()
    .retrieve()
    .bodyToMono(Curso.class)
    ;

monoCurso.subscribe();
Curso curso = monoCurso.block();

System.out.println("Curso encontrado: ");
System.out.println(curso);
```



- >Renomear a classe **WebClientMain** para **WebClientFluxMain**
- >Criar no mesmo pacote a classe **WebClientMonoMain**
- >Codificar o método **main** com:

```
Mono<Curso> monoCurso = WebClient.create("localhost:8080/curso/2")  
    .get()  
    .retrieve()  
    .bodyToMono(Curso.class)  
    ;  
  
monoCurso.subscribe();  
Curso curso = monoCurso.block();  
  
System.out.println("Curso encontrado: ");  
System.out.println(curso);
```

- > Faça um teste usando um ID inexistente. O que acontece?
- > Implemente um código para resolver

> Para requisições **POST** que retornam um valor, usamos **Mono**

> Código para criar um curso:

```
Curso novoCurso = new Curso();  
novoCurso.setDescricao("Dominando Spring WebFlux");  
novoCurso.setCargaHoraria(80);  
  
Curso cursoCriado = WebClient.create("localhost:8080/curso")  
    .post()  
    .bodyValue(novoCurso)  
    .retrieve()  
    .bodyToMono(Curso.class)  
    .block();  
  
System.out.println("Curso criado:");  
System.out.println(cursoCriado); // com ID preenchido
```

> Criar a classe **WebClientPostMain**

> No método main, codificar:

```
Curso novoCurso = new Curso();  
novoCurso.setDescricao("Dominando Spring WebFlux");  
novoCurso.setCargaHoraria(80);  
  
Curso cursoCriado = WebClient.create("localhost:8080/curso")  
    .post()  
    .bodyValue(novoCurso)  
    .retrieve()  
    .bodyToMono(Curso.class)  
    .block();  
  
System.out.println("Curso criado:");  
System.out.println(cursoCriado); //com ID preenchido
```

>Para atualizar um curso existente, realizamos um request **PUT**.

```
Curso cursoExistente = new Curso(1L, "REST com Spring Boot e Spring WebFlux", 120);

ResponseEntity<Void> responseEntity = WebClient.create("localhost:8080/curso")
    .put()
    .bodyValue(cursoExistente)
    .retrieve()
    .toBodilessEntity()
    .block();

HttpStatus statusCode = responseEntity.getStatusCode();

System.out.println("Curso atualizado. Status: " + statusCode);
```

> Para remover um curso, invocamos um request **DELETE**

```
ResponseEntity<Void> responseEntity = WebClient.create("localhost:8080/curso/3")  
    .delete()  
    .retrieve()  
    .toBodilessEntity()  
    .block();  
  
HttpStatus statusCode = responseEntity.getStatusCode();  
  
System.out.println("Curso removido. Status: " + statusCode );
```

- >No mesmo pacote dos exercícios anteriores:
- >Criar a classe **WebClientPutMain**
- >Codifique o método main:

```
Curso cursoExistente = new Curso(1L, "REST com Spring Boot e Spring WebFlux", 120);

ResponseEntity<Void> responseEntity = WebClient.create("localhost:8080/curso")
    .put()
    .bodyValue(cursoExistente)
    .retrieve()
    .toBodilessEntity()
    .block();

HttpStatus statusCode = responseEntity.getStatusCode();

System.out.println("Curso atualizado. Status: " + statusCode);
```

>Criar a classe **WebClientDeleteMain**

>Codifique o método main:

```
ResponseEntity<Void> responseEntity = WebClient.create("localhost:8080/curso/3")  
    .delete()  
    .retrieve()  
    .toBodilessEntity()  
    .block();  
  
HttpStatus statusCode = responseEntity.getStatusCode();  
  
System.out.println("Curso removido. Status: " + statusCode );
```



# Testando APIs com **WebTestClient**

- >A biblioteca Spring WebFlux fornece a classe **WebTestClient** para testar APIs REST.
- >Similar ao WebClient
- >Contém métodos para checar status da resposta, header e body

<code>.expectStatus().isOk()</code>	Status é 200?
<code>.expectStatus().isNotFound()</code>	Status é 404?
<code>.expectHeader().contentType(MediaType.APPLICATION_JSON)</code>	Cabeçalho tem contentType JSON?
<code>.expectBody().returnResult()</code>	Corpo tem algum resultado?
<code>.expectBody().isEmpty()</code>	Corpo está vazio?
<code>.expectBody().jsonPath("\$.descricao").isNotEmpty()</code>	O JSON do corpo tem atributo 'descricao'?

## Como escrever as classes de teste:

- a) Anotar a classe com `@SpringBootTest` e configurar `webEnvironment`
- b) A instância de `WebTestClient` deve ser injetada via `@AutoWired`
- c) Escrever os métodos de teste de maneira fluente

## > Testando endpoint /curso com método **GET**

```

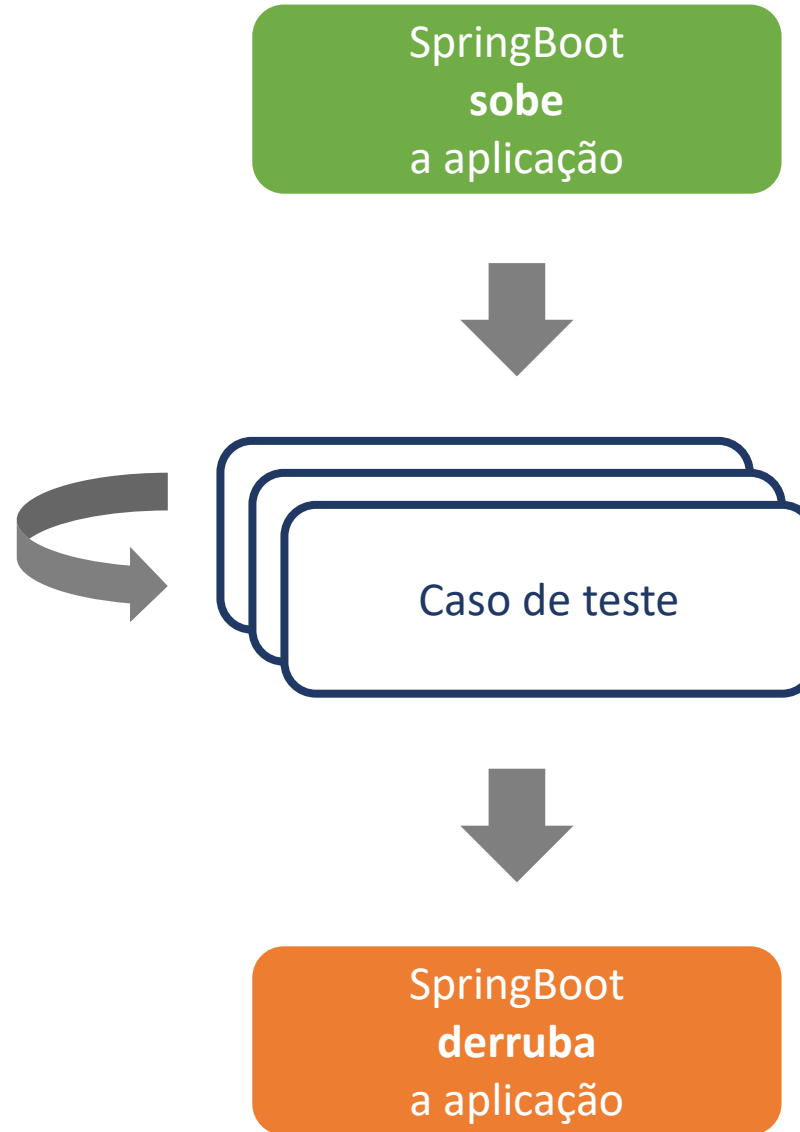
9  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
10 class CursoControllerTest {
11
12     @Autowired
13     private WebTestClient webTestClient;
14
15     @Test
16     void deveListarCursos() {
17
18         webTestClient.get()
19             .uri("/curso")
20             .exchange()
21             .expectHeader().contentType(MediaType.APPLICATION_JSON)
22             .expectStatus().isOk()
23             .expectBody().returnResult()
24             ;
25     }
26 }

```

a)

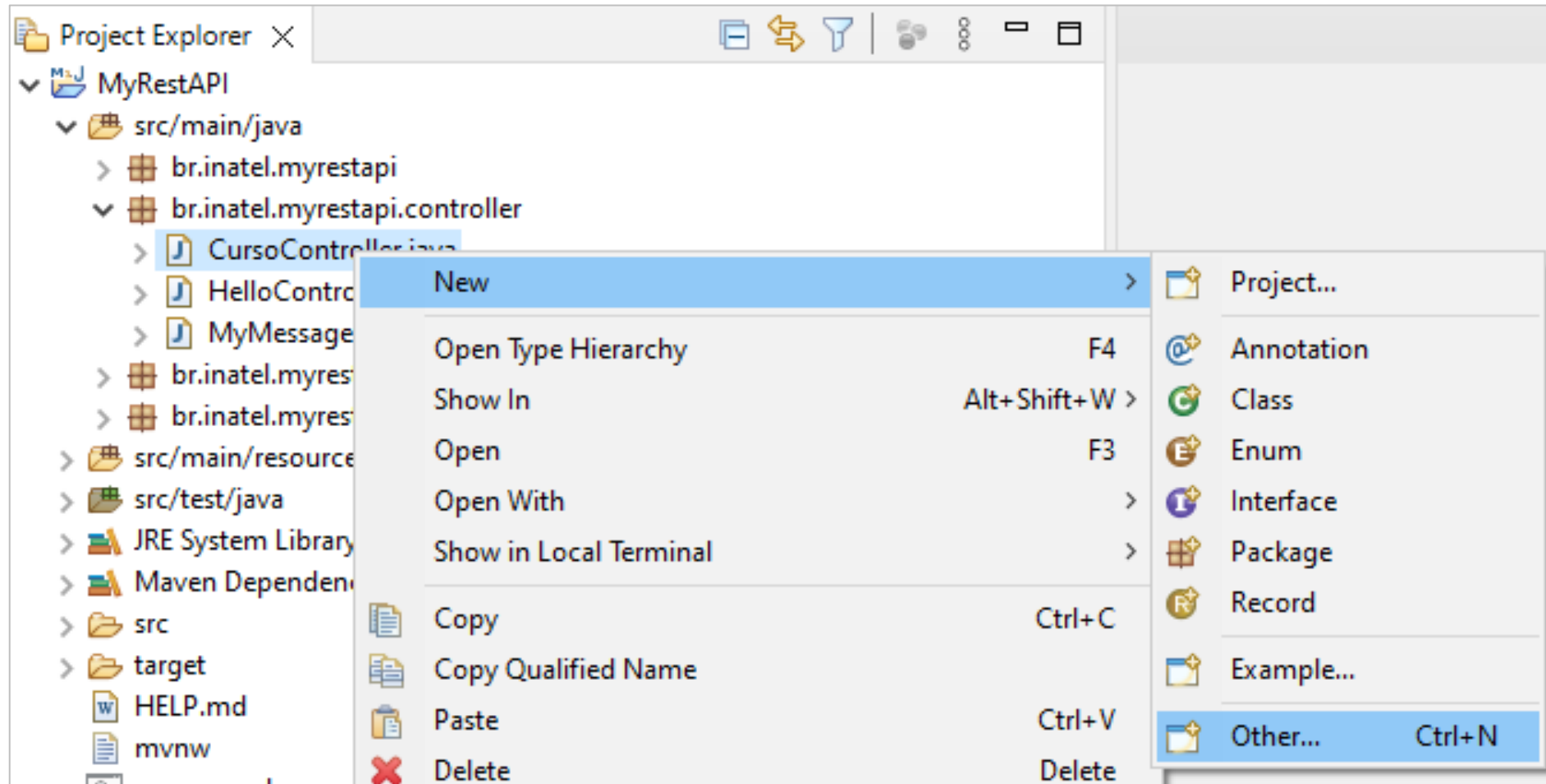
b)

c)

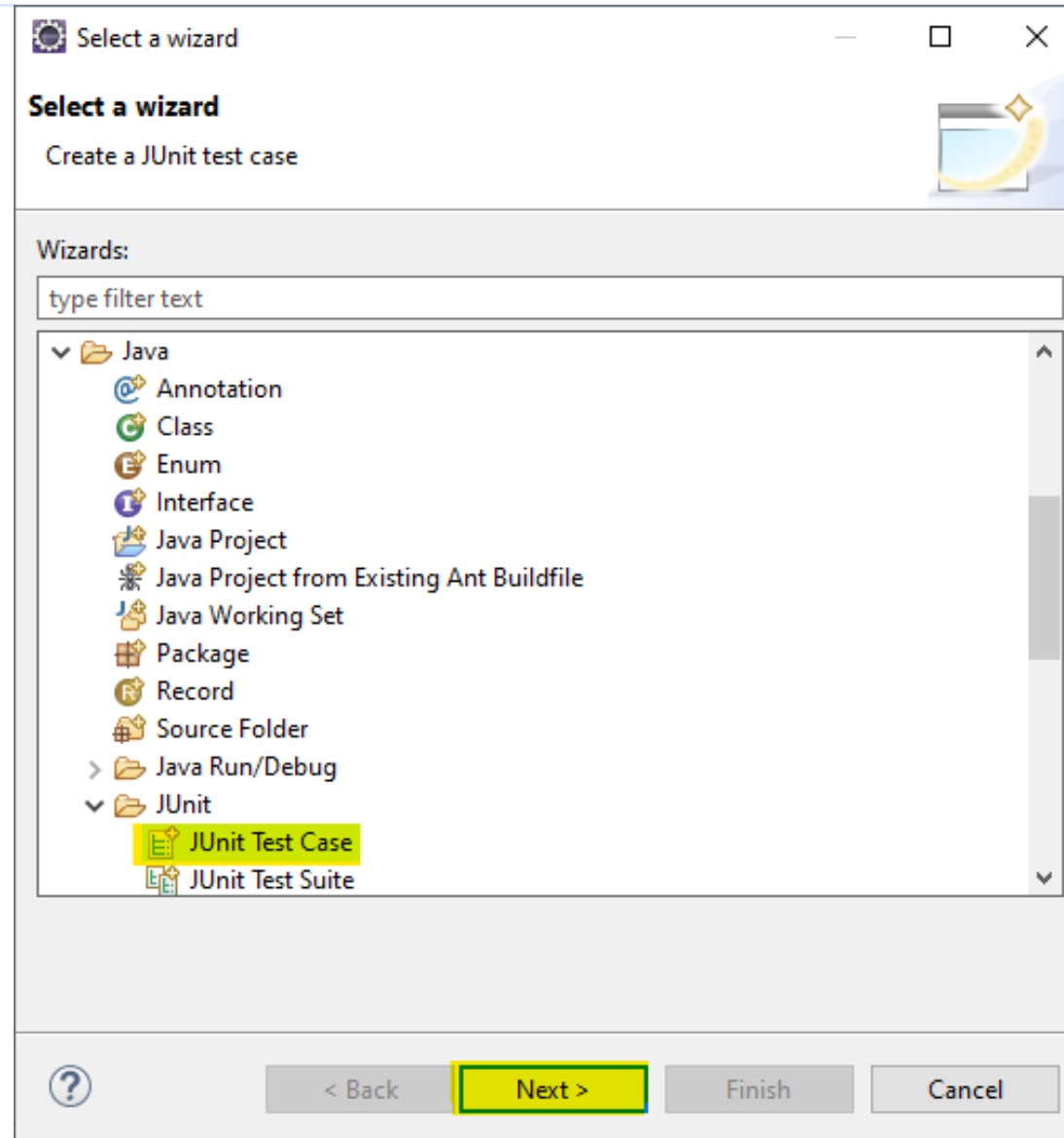


>Vamos criar uma *classe de teste* para **CursoController**:

>Na view Package Explorer, clique dar direita em CursoController > New > Other



- >Expandir Java > **JUnit**
- >Seleciona **JUnit Test Case**
- >Clicar **Next**



>Na próxima janela, deixar tudo default

>Clicar **Finish**

**New JUnit Test Case**

Select the name of the new JUnit test case. Specify the class under test to select methods to be tested on the next page.

☐ New JUnit 3 test ☐ New JUnit 4 test ☒ New JUnit Jupiter test

Source folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

☐ @BeforeAll setUpBeforeClass() ☐ @AfterAll tearDownAfterClass()  
☐ @BeforeEach setUp() ☐ @AfterEach tearDown()  
☐ constructor

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

Class under test:



>Na classe CursoControllerTest, codificar:

a)Anotar a classe com:

```
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
```

b)Declarar o atributo privado e injetado:

```
@Autowired
```

```
private WebTestClient webTestClient;
```

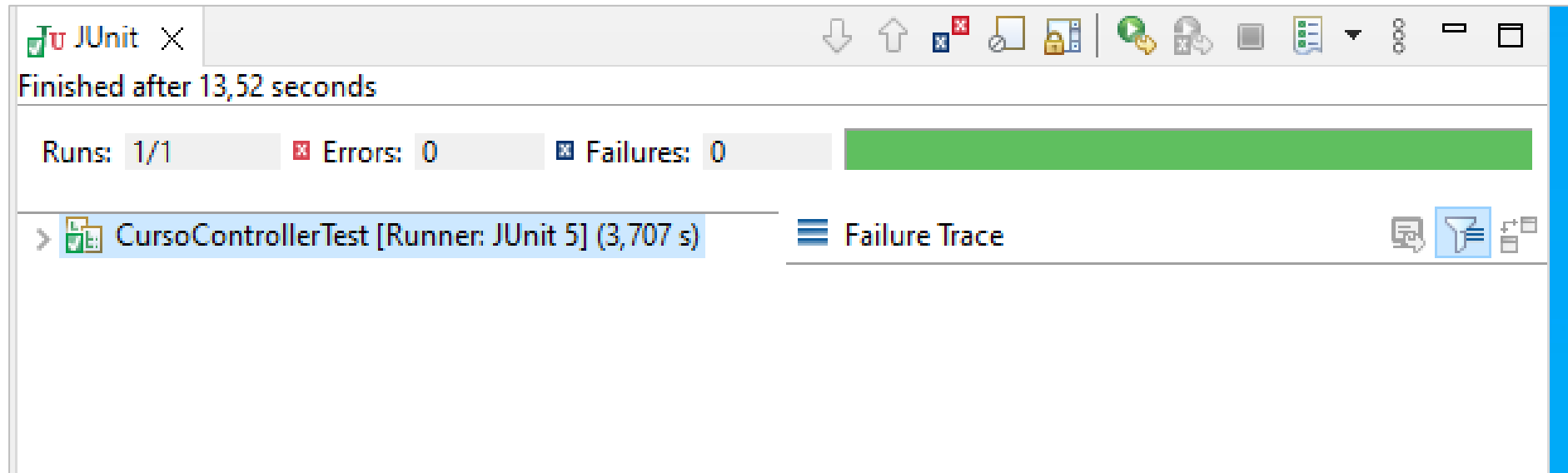
c)Declarar o método `void` `deveListarCursos()` e anotar com `@Test`

d)Codificar o método de teste (próximo slide)

>A classe completa deve ficar assim:

```
8  @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
9  class CursoControllerTest {
10
11      @Autowired
12      private WebTestClient webTestClient;
13
14      @Test
15      void deveListarCursos() {
16
17          webTestClient.get()
18              .uri("/curso")
19              .exchange()
20              .expectStatus().isOk()
21              .expectBody().returnResult()
22              ;
23      }
24  }
```

> Para rodar o teste: Clique da direita no método: Run As > **JUnit Test**



>Abrir um terminal na pasta do projeto e executar `mvnw test`

[illegible]

```
2022-03-19 10:46:00.249 INFO 8640 --- [main] b.i.myrestapi.MyRestApiAppl
t\MyRestAPI)
2022-03-19 10:46:00.253 INFO 8640 --- [main] b.i.myrestapi.MyRestApiAppl
2022-03-19 10:46:01.872 INFO 8640 --- [main] b.i.myrestapi.MyRestApiAppl
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.768 s - in
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 26.113 s
[INFO] Finished at: 2022-03-19T10:46:05-03:00
[INFO] -----
```

## >Behaviour Driven Develpment



***givenIDValido\_whenGetCurso\_thenRespondeComCursoValido***

>Em português também é possível:



*dadoIDValido\_quandoGetCurso\_entadoRespondeComCursoValido*

Vantagens:

- > **Compartilhamento de conhecimento:** aproxima desenvolvedores e testadores
- > **Documentação dinâmica:** os próprios testes servem como documentação
- > **Comunicação entre times:** analistas de requisitos conseguem se comunicar com os desenvolvedores e testadores usando a abordagem BDD

> Para o endpoint **GET** `/curso/{id}`, escreva 2 testes usando BDD

Sugestão de nome de método:

`dadoCursoIDValido_quandoGetCursoId_entaoRespondeComCursoValido`

`dadoCursoIDInvalido_quandoGetCursoID_entaoRespondeComStatusNotFound`



## >Solução possível:

```
@Test
void dadoCursoIDValido_quandoGetCursoId_entaoRespondeComCursoValido() {
    Long idValido = 1L;

    Curso cursoRespondido = webTestClient.get()
        .uri("/curso/" + idValido)
        .exchange()
        .expectStatus().isOk()
        .expectBody(Curso.class)
            .returnResult()
            .getResponseBody();

    assertEquals(cursoRespondido.getId(), idValido);
}
```

> Para o endpoint **PUT /curso**, escreva 1 teste usando BDD

> Para o endpoint **DELETE** /curso/{id}, escreva 1 teste usando BDD

*Obrigado*

