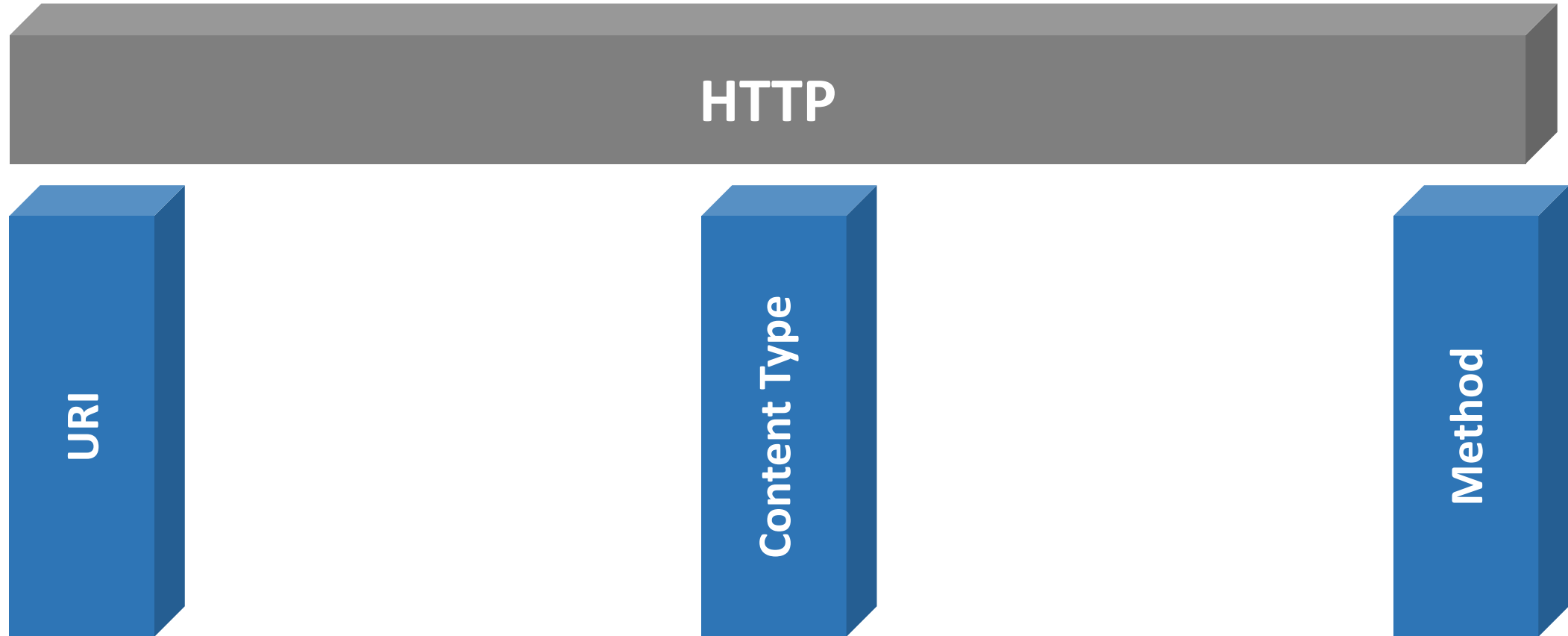


# Laboratório ***REST***

**Sistemas Distribuídos**

2022/01

Professor Vitor Figueiredo



Servidor rodando localhost ouvindo a porta 8080

## Aplicação A

arquivo\_a1

arquivo\_a2

arquivo\_a3

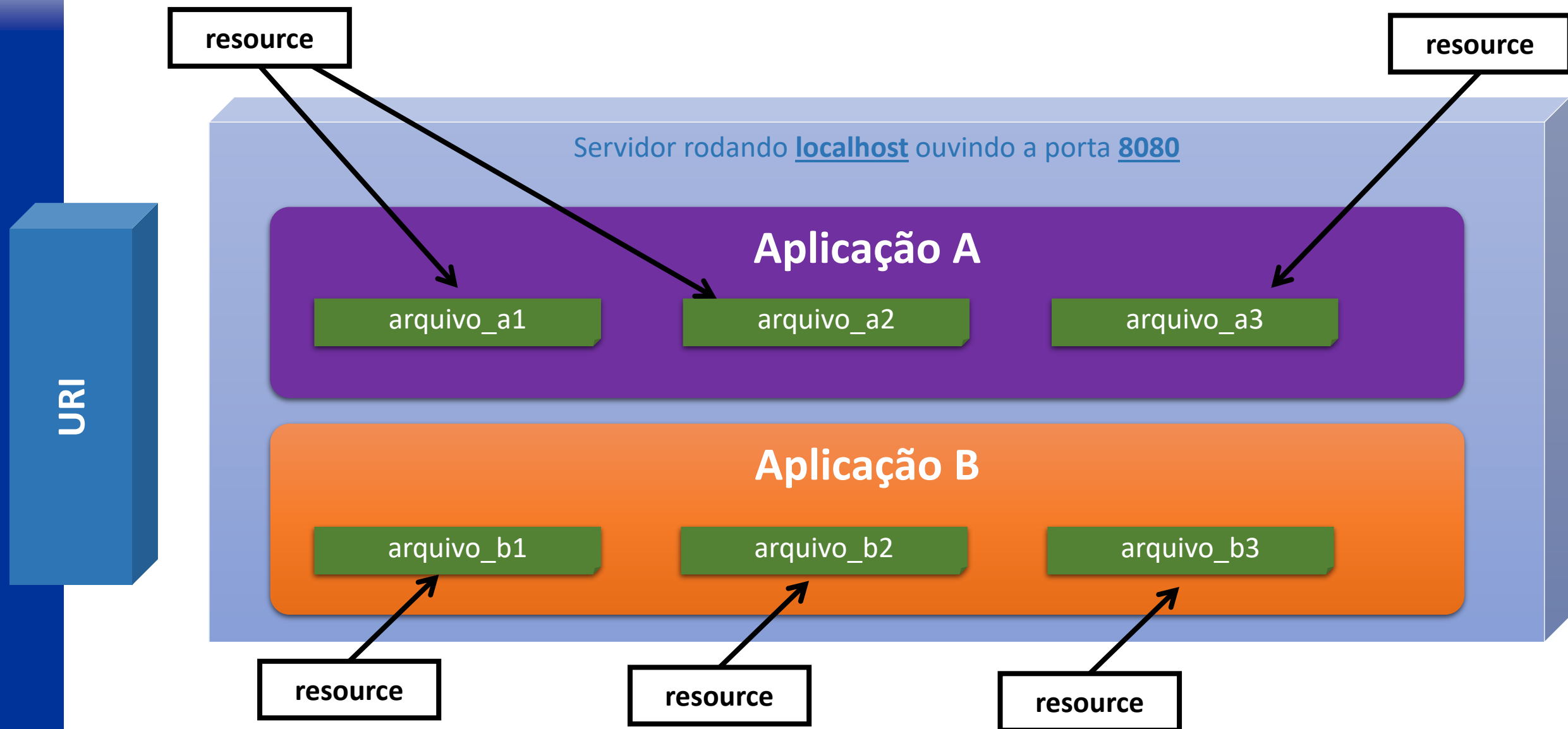
## Aplicação B

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**

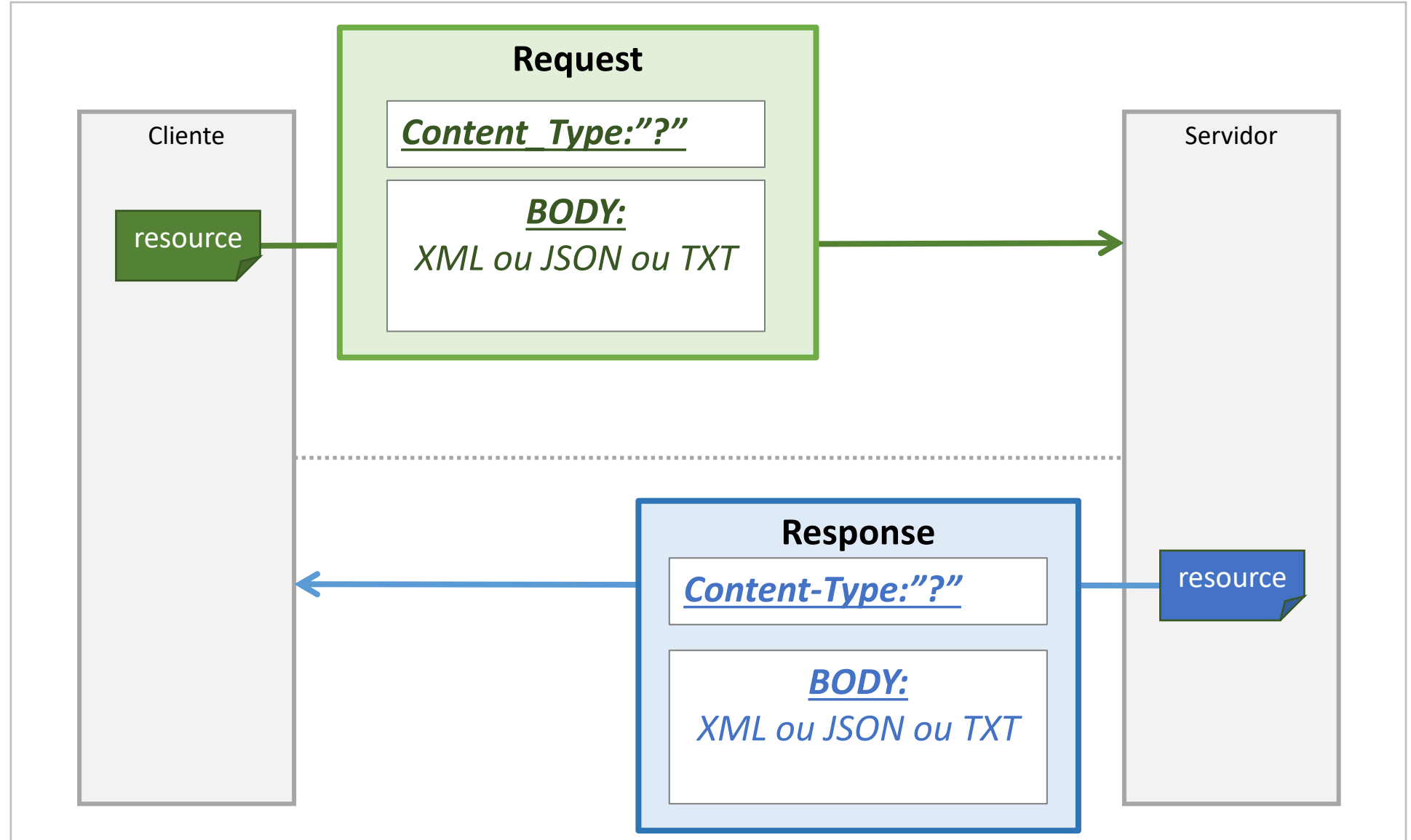
`index.html`

`cliente.html`

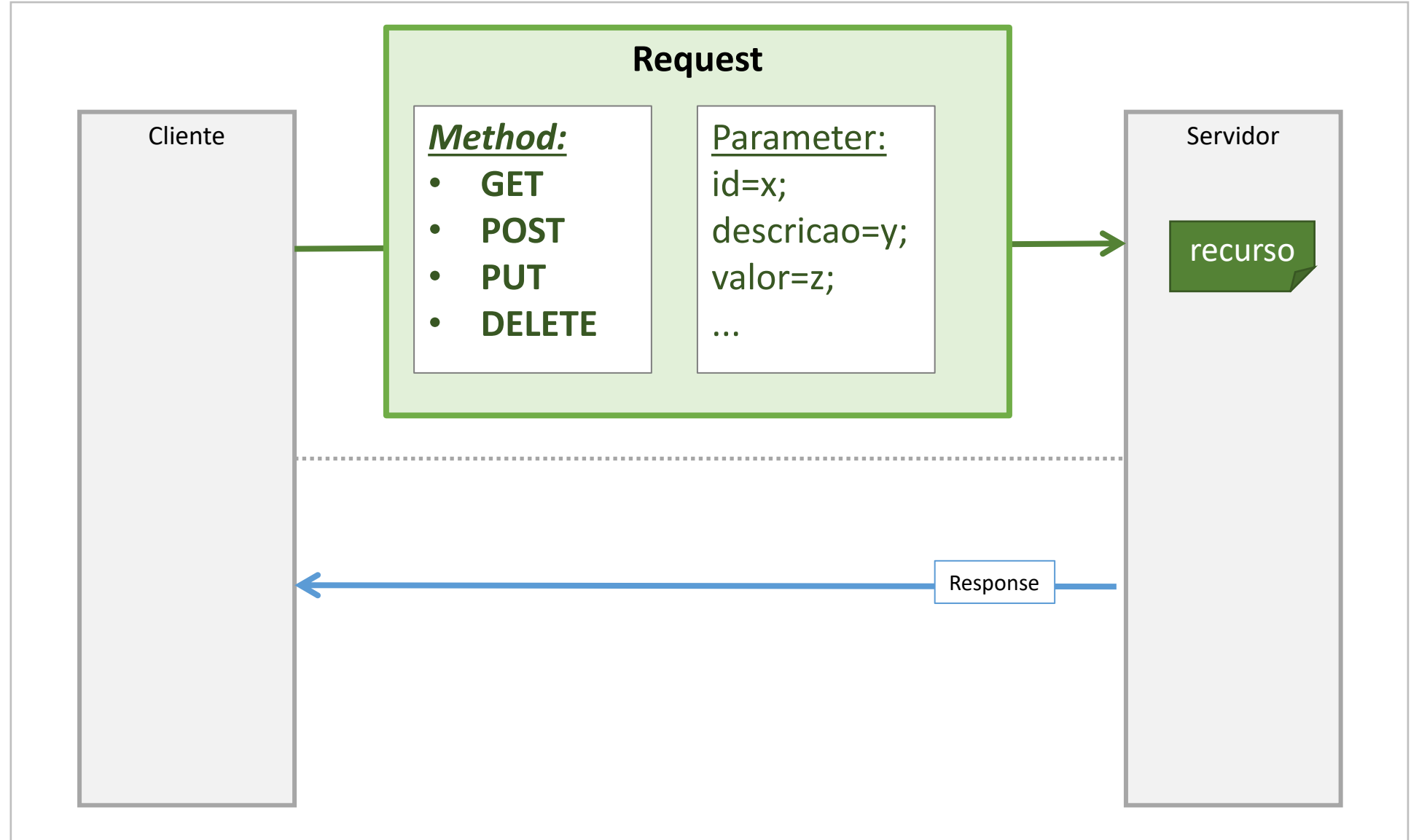
`relatorio.html`

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

URI



## Method

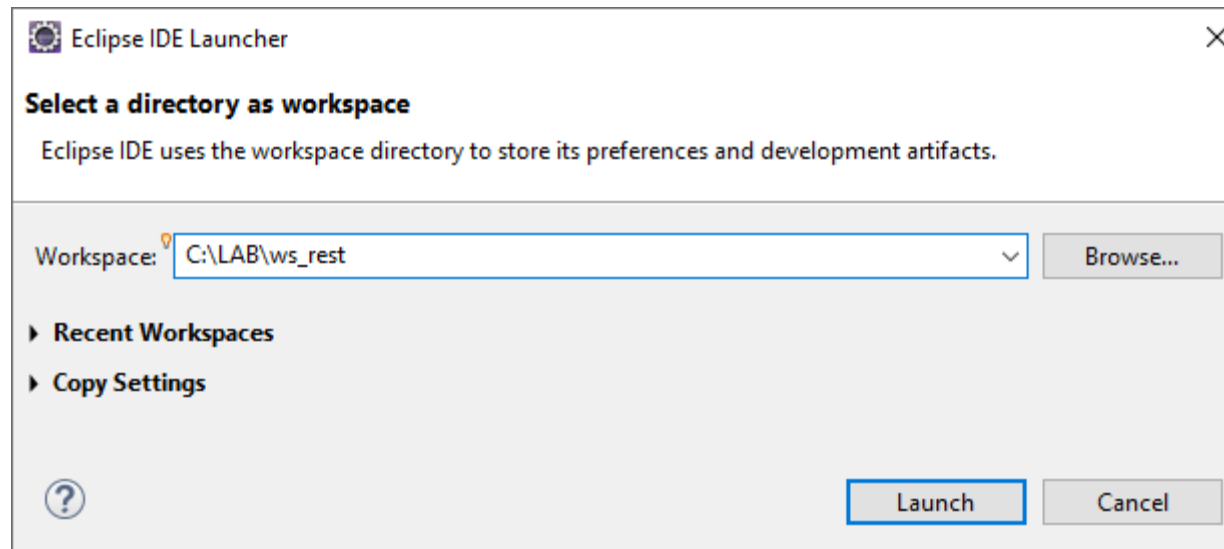




| Método HTTP   | Descrição   |
|---------------|---|
| <b>GET</b>    | Requisita a <b>consulta</b> de recurso. Para consultar um recurso específico, o pacote do REQUEST deve conter o respectivo identificador (chave primária).  |
| <b>POST</b>   | Requisita a <b>criação</b> de uma recurso. Por exemplo, 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   |

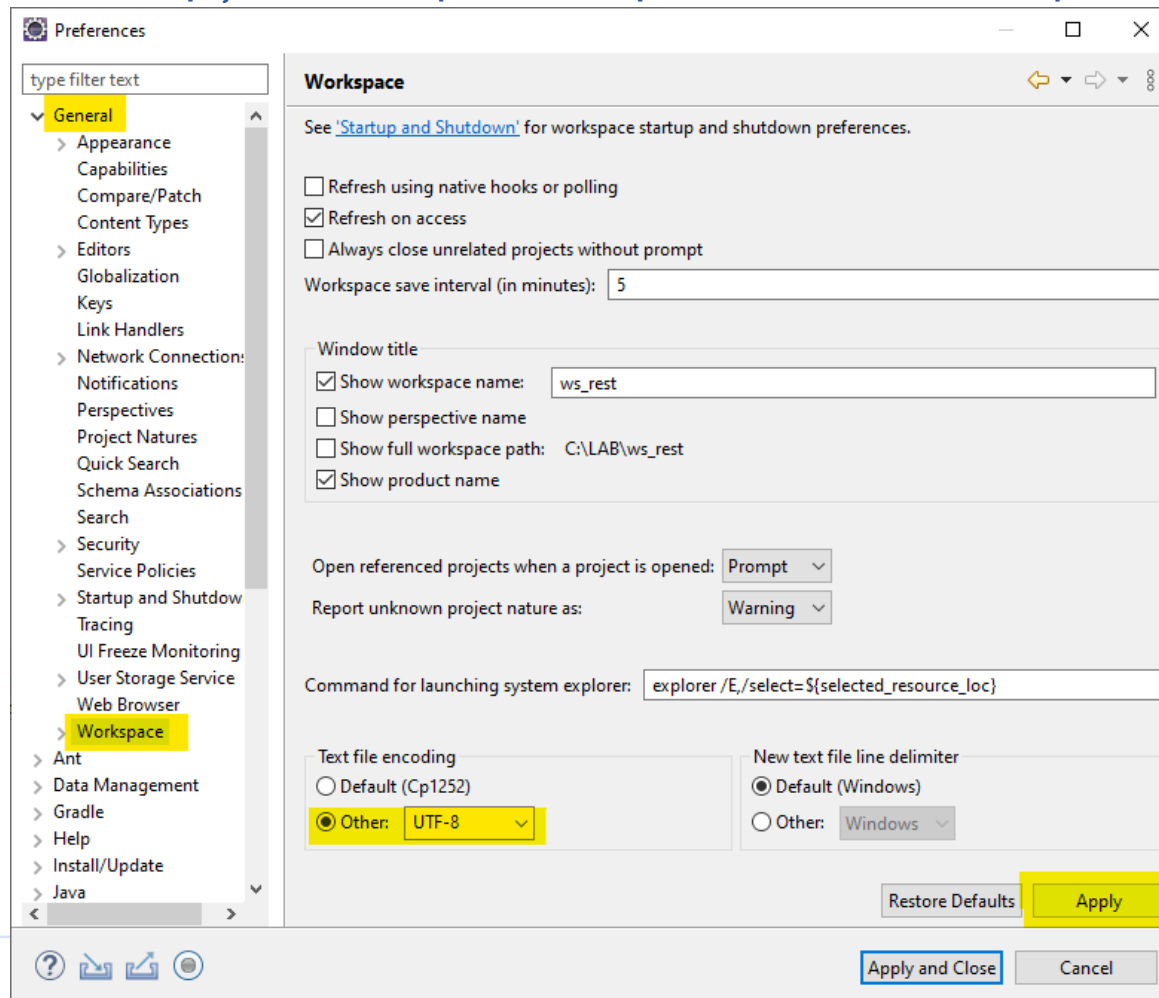
- > Criar pasta c:\LAB
- > Baixar e instalar Java Development Kit 11
- > Baixar última versão Eclipse: **Eclipse IDE for Enterprise Java and Web Developers**
- > Descompactar o zip do Eclipse na pasta criada
- > Iniciar o Eclipse

## >Selecionar o workspace:



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

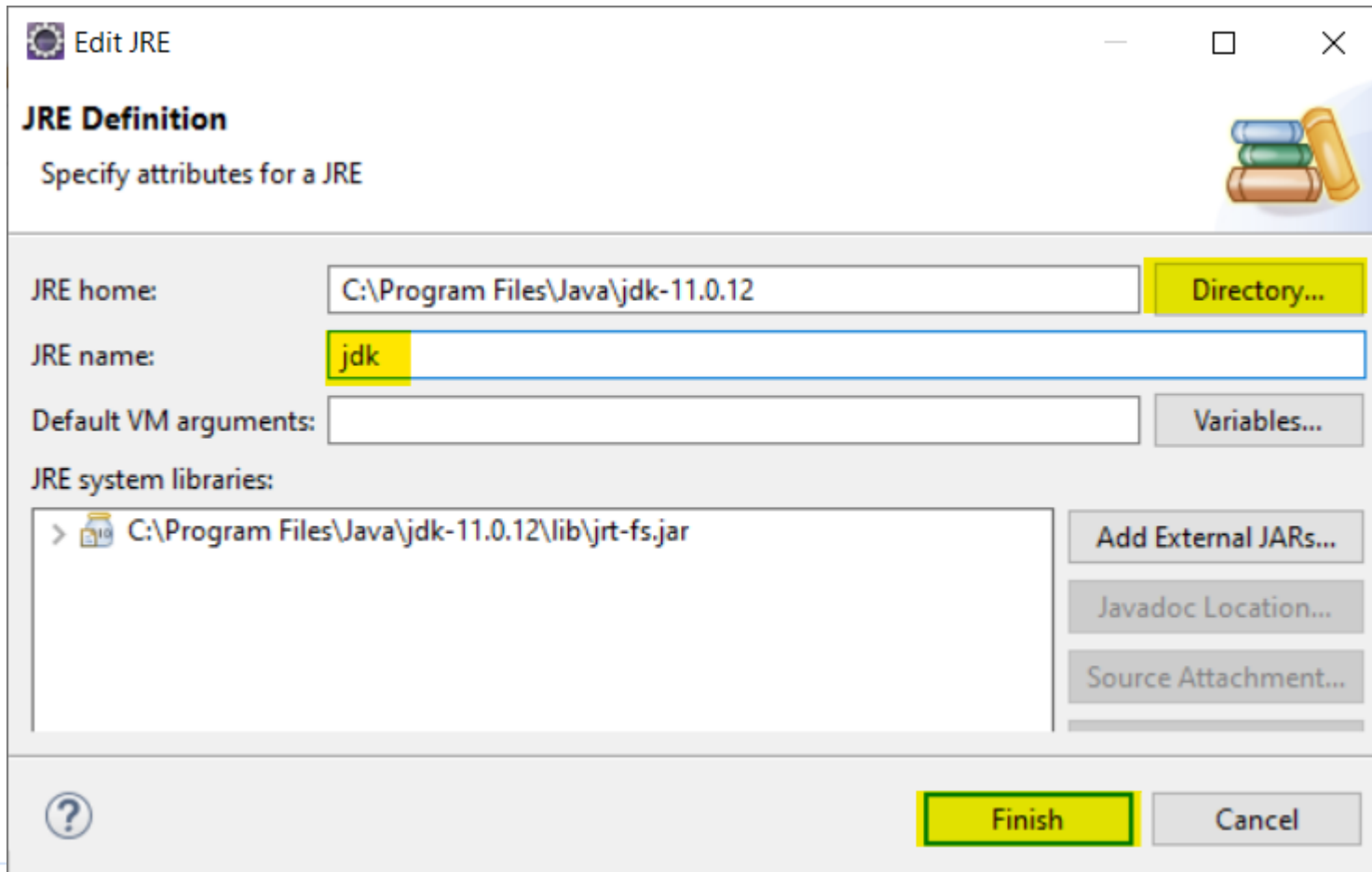
- Precisamos configurar o *Encoding* do Workspace. Clique em **Window > Preferences**.
- Nas opções da esquerda, expandir **General**, e clique em **Workspace**:



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

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

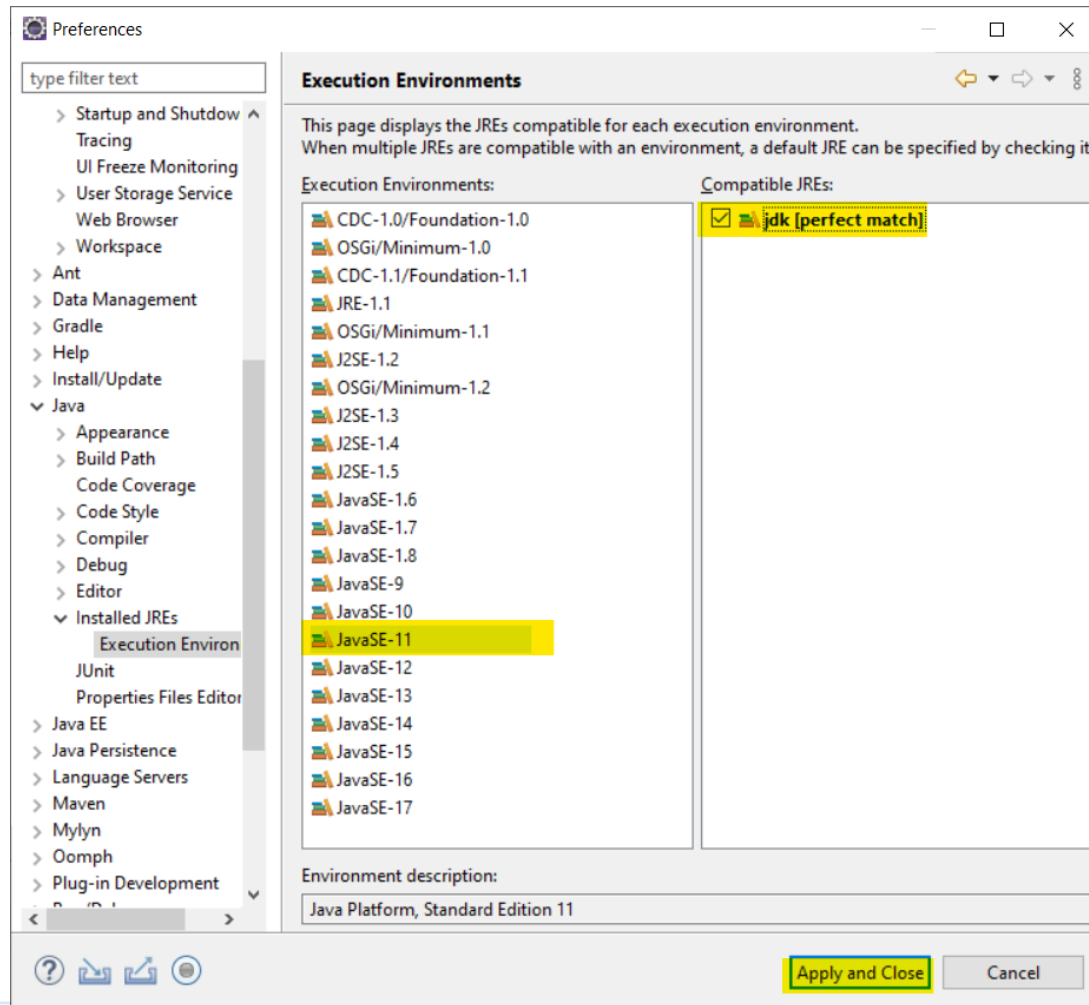
- Agora, precisamos configurar o Eclipse para usar o JDK ao invés do JRE.
- Nas opções da esquerda, expanda **Java** e clique em **Installed JRE's**:



- 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** inicializr

**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

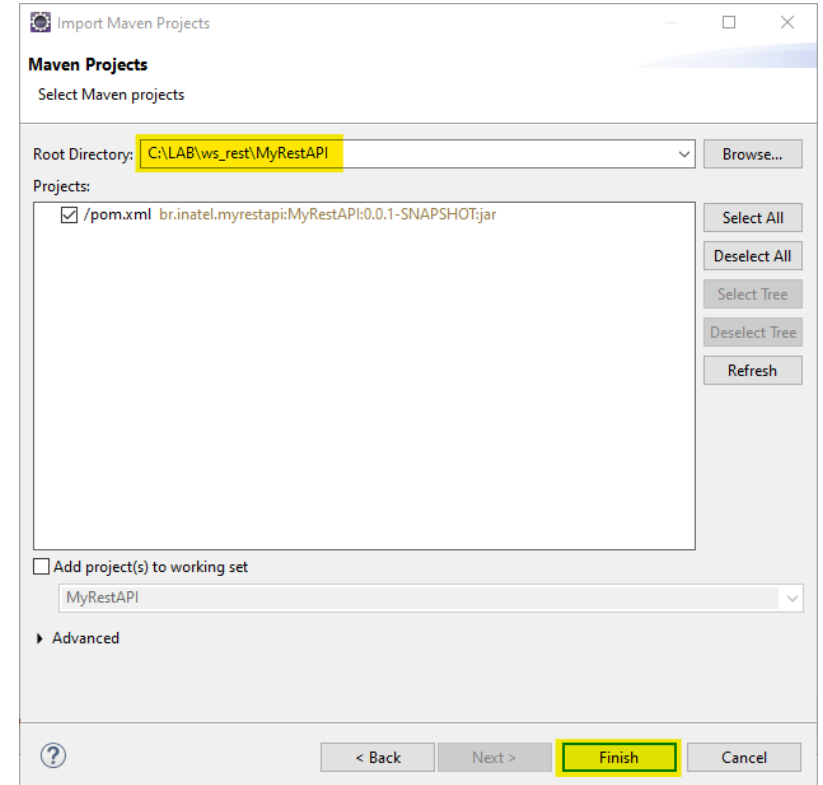
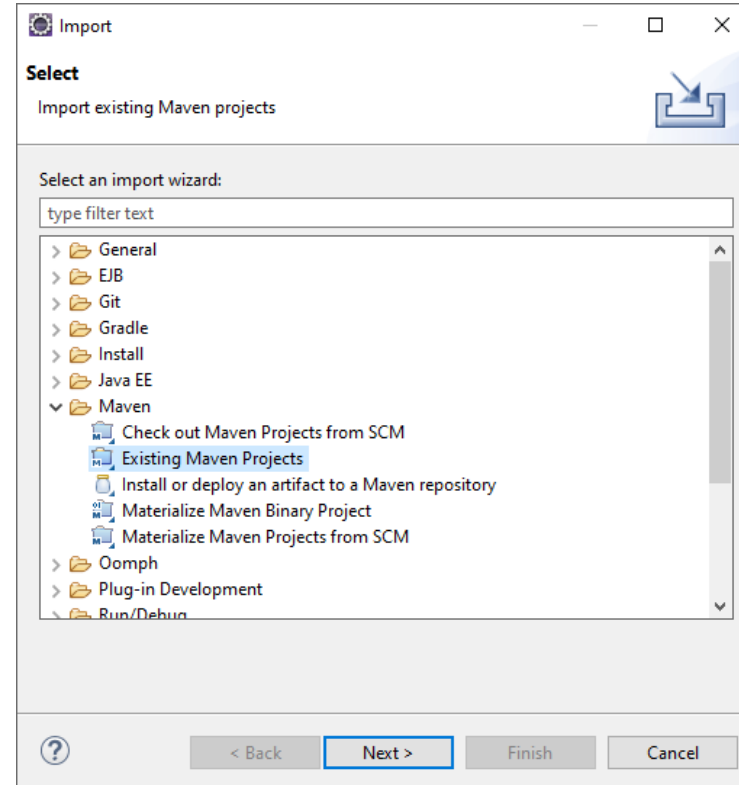
>No Eclipse, importar como projeto Maven:

**File > Import**

>Expandir **Maven > Existing Maven Projects**

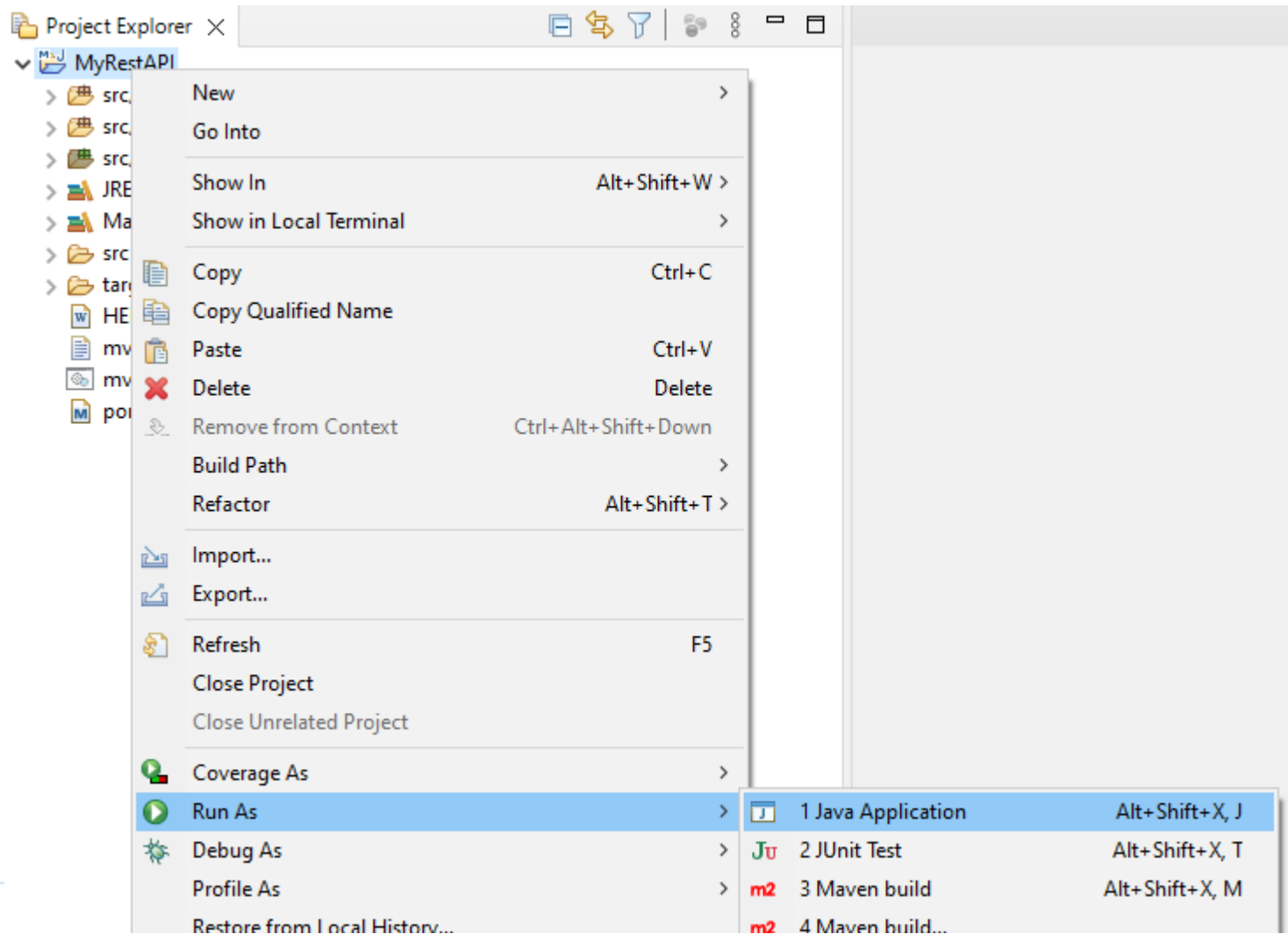
>Clicar **Finish**

**Importante:**  
Aguardar o build do projeto



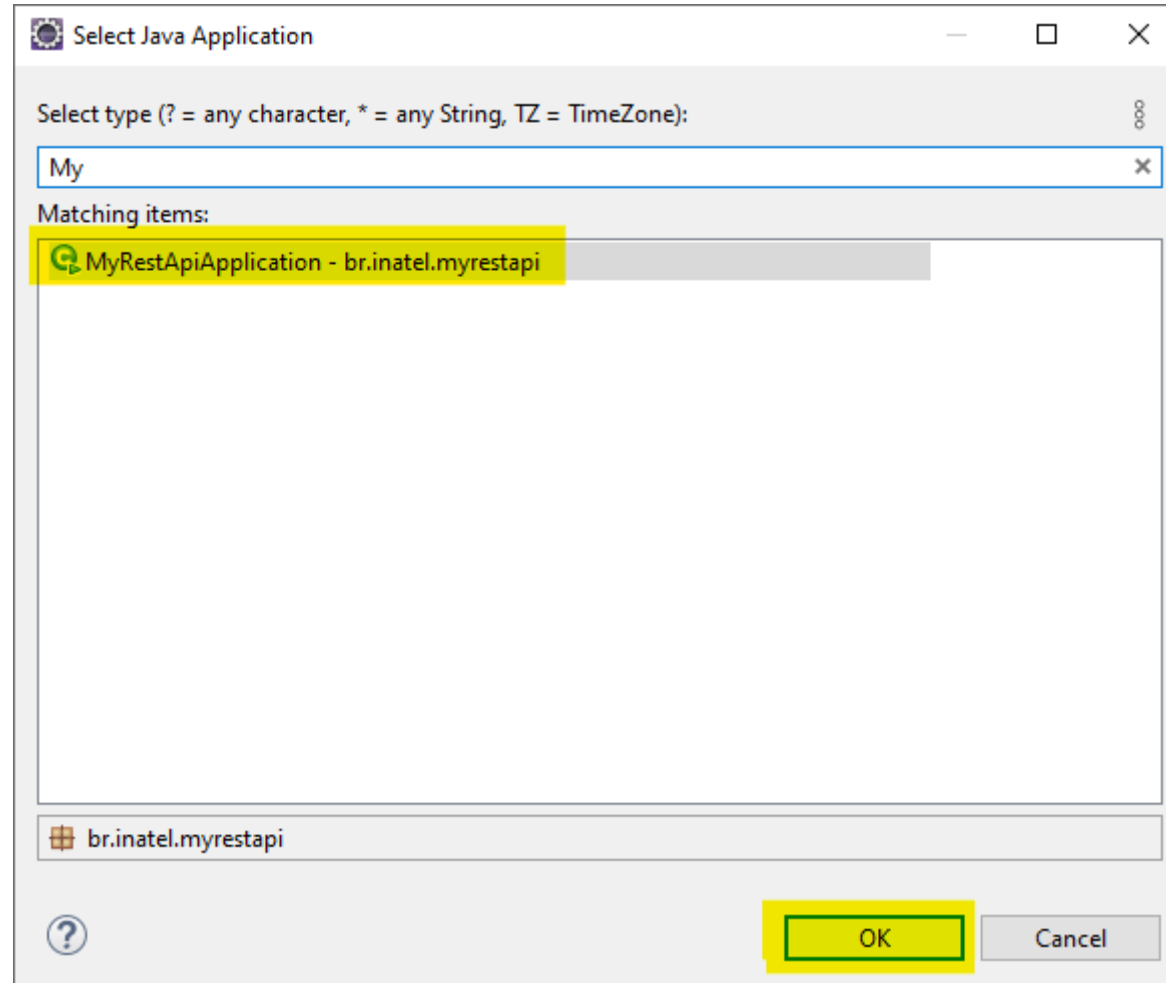
>Subir a aplicação:

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



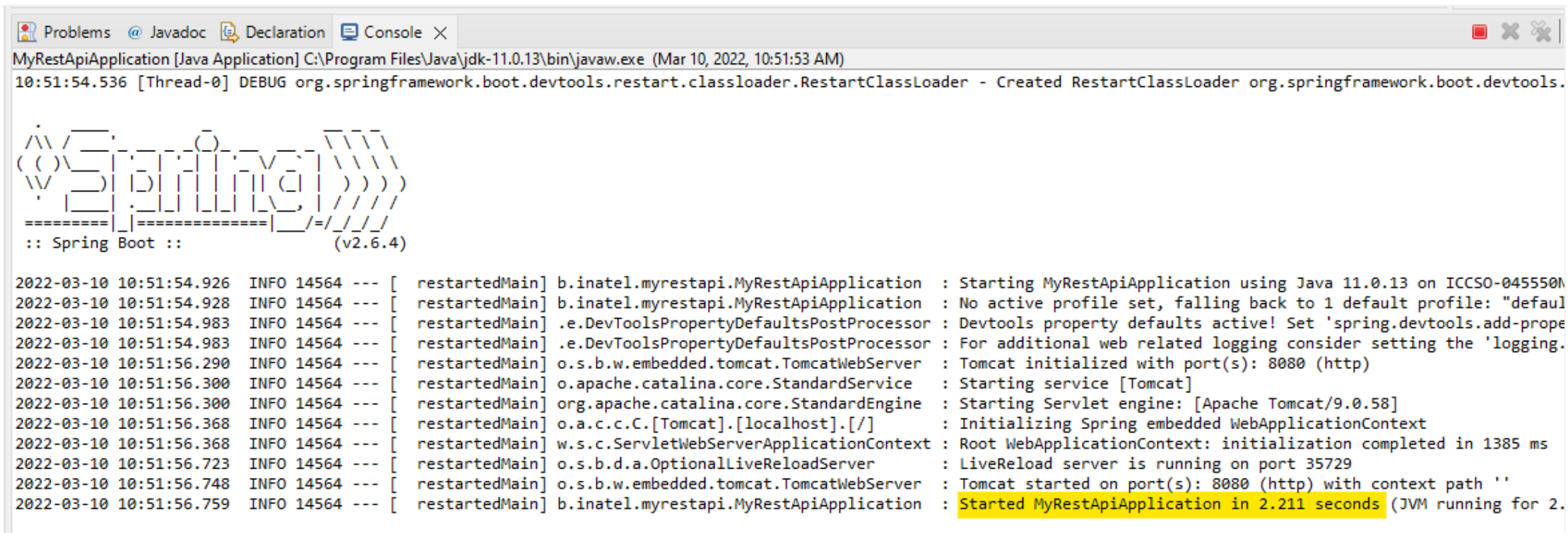
b) Selecionar **MyRestApiApplication** (use o filtro)

c) Clicar **OK**





## d) Observar a saída do console:



```

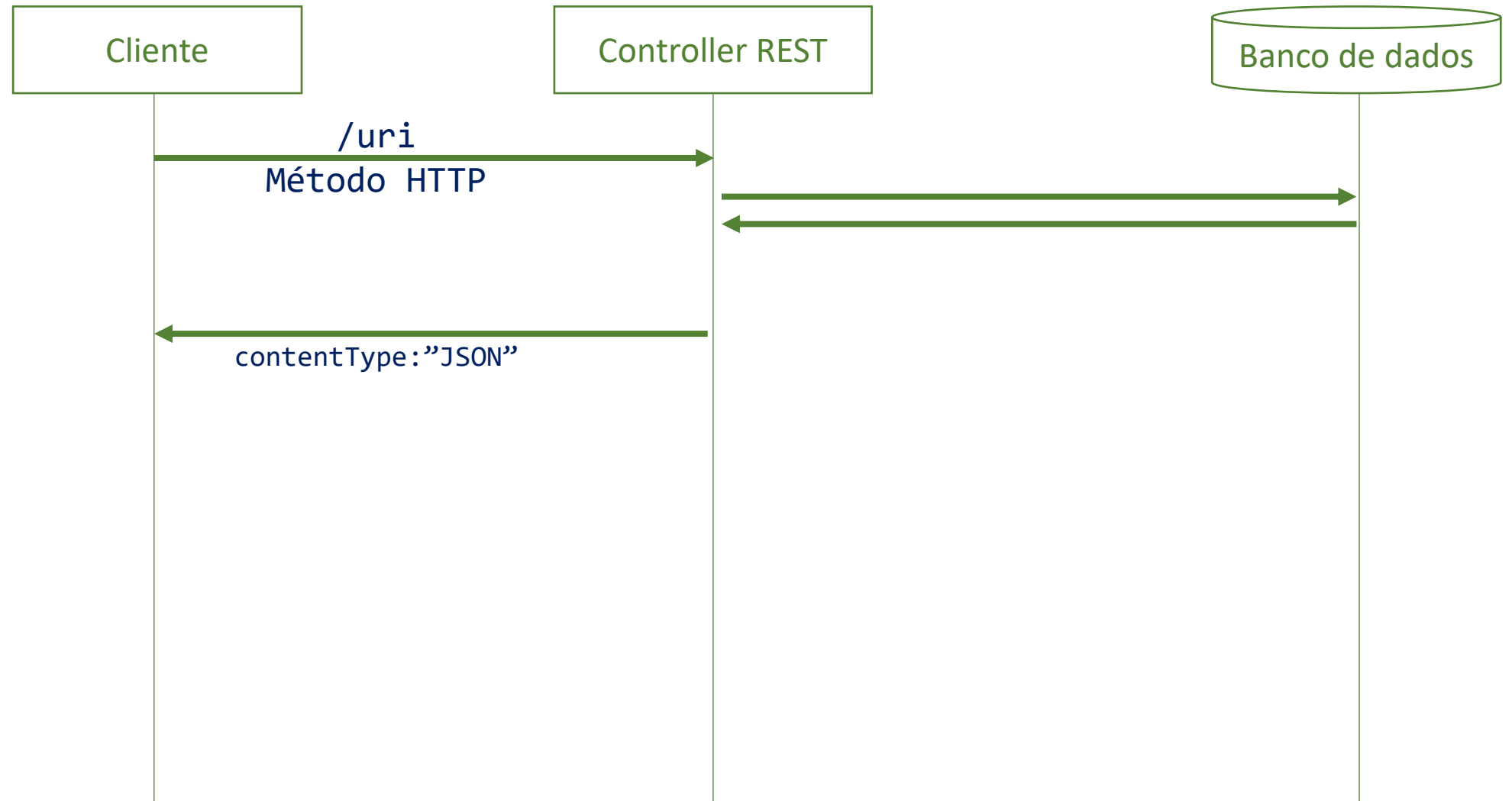
Problems @ Javadoc Declaration Console ×
MyRestApiApplication [Java Application] C:\Program Files\Java\jdk-11.0.13\bin\javaw.exe (Mar 10, 2022, 10:51:53 AM)
10:51:54.536 [Thread-0] DEBUG org.springframework.boot.devtools.restart.classloader.RestartClassLoader - Created RestartClassLoader org.springframework.boot.devtools.

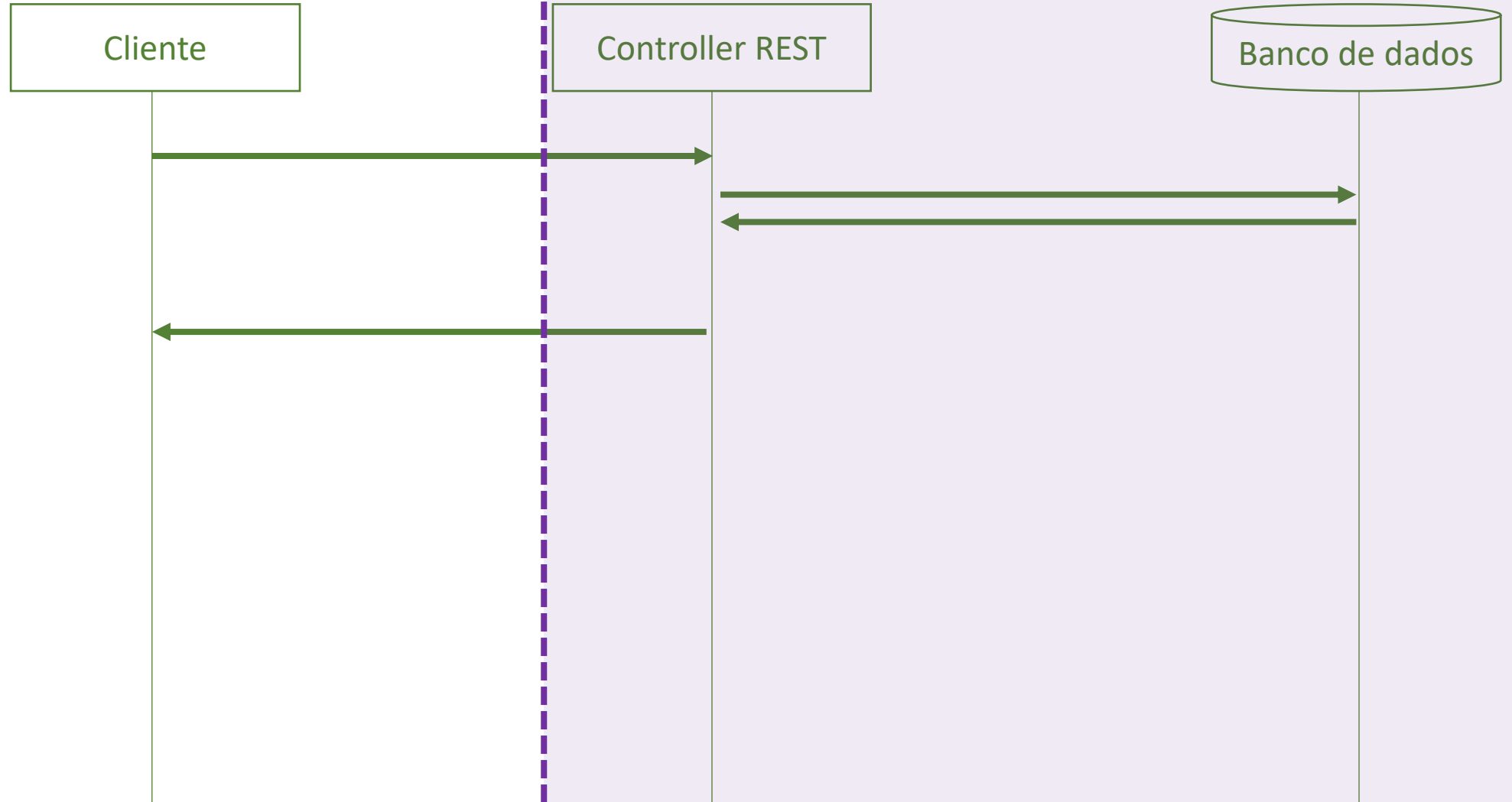
:: Spring Boot :: (v2.6.4)

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.

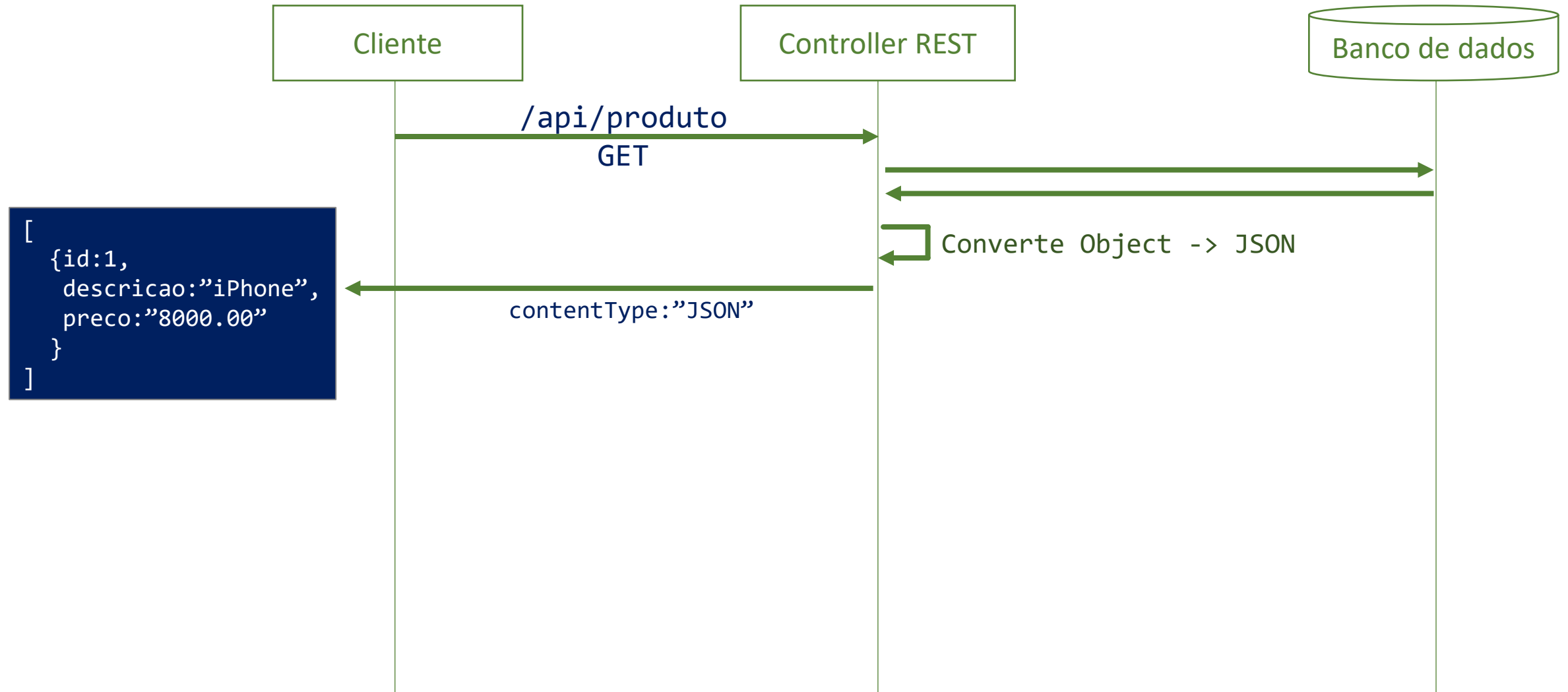
```

# *Arquitetura REST no Spring*



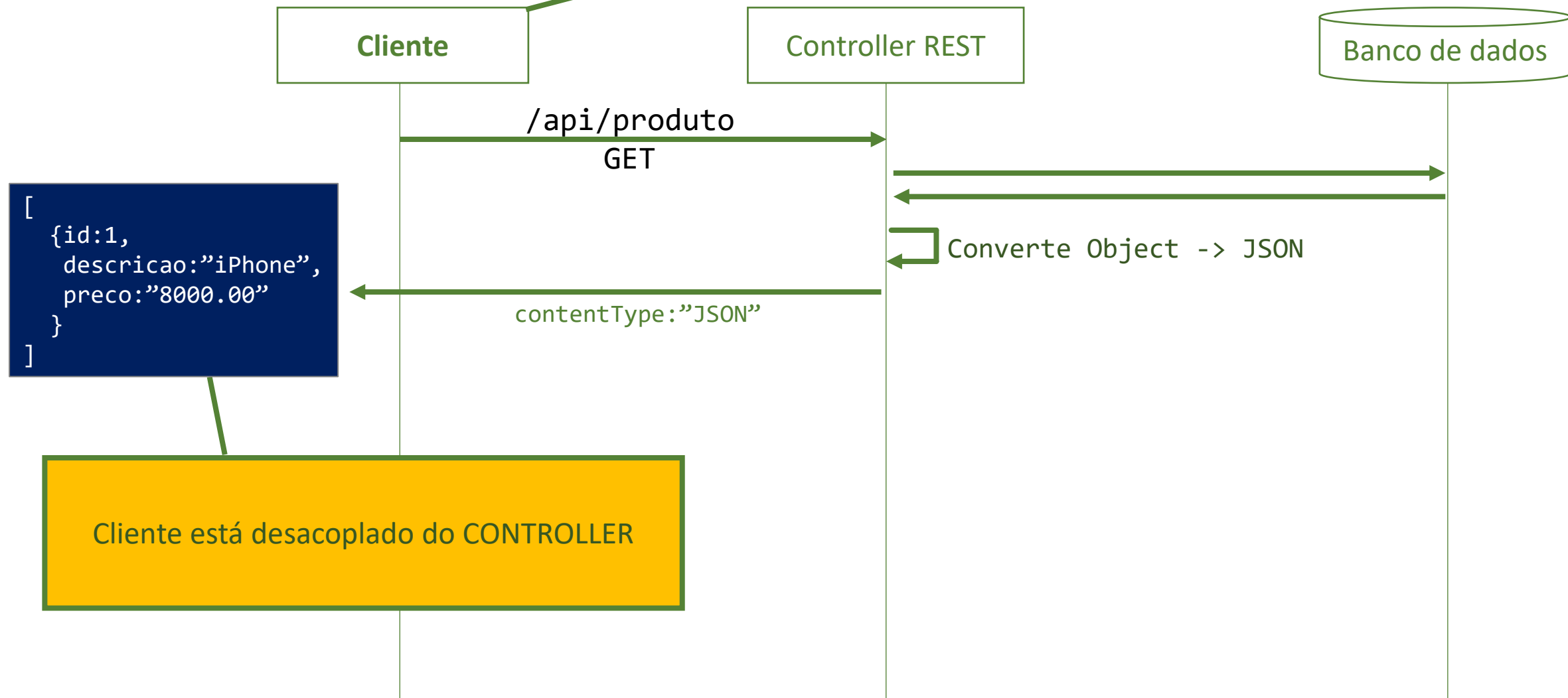


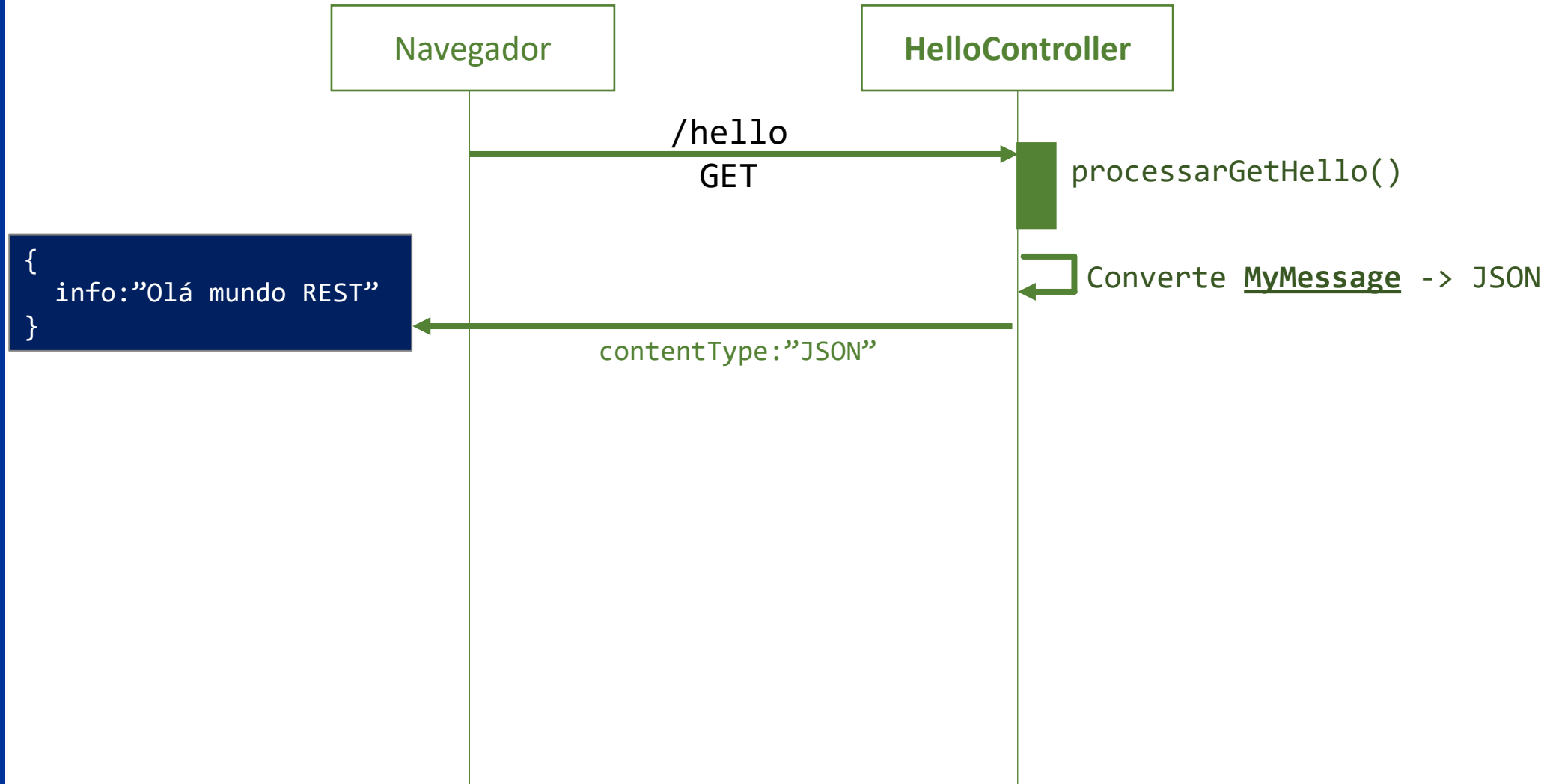
## Controller REST

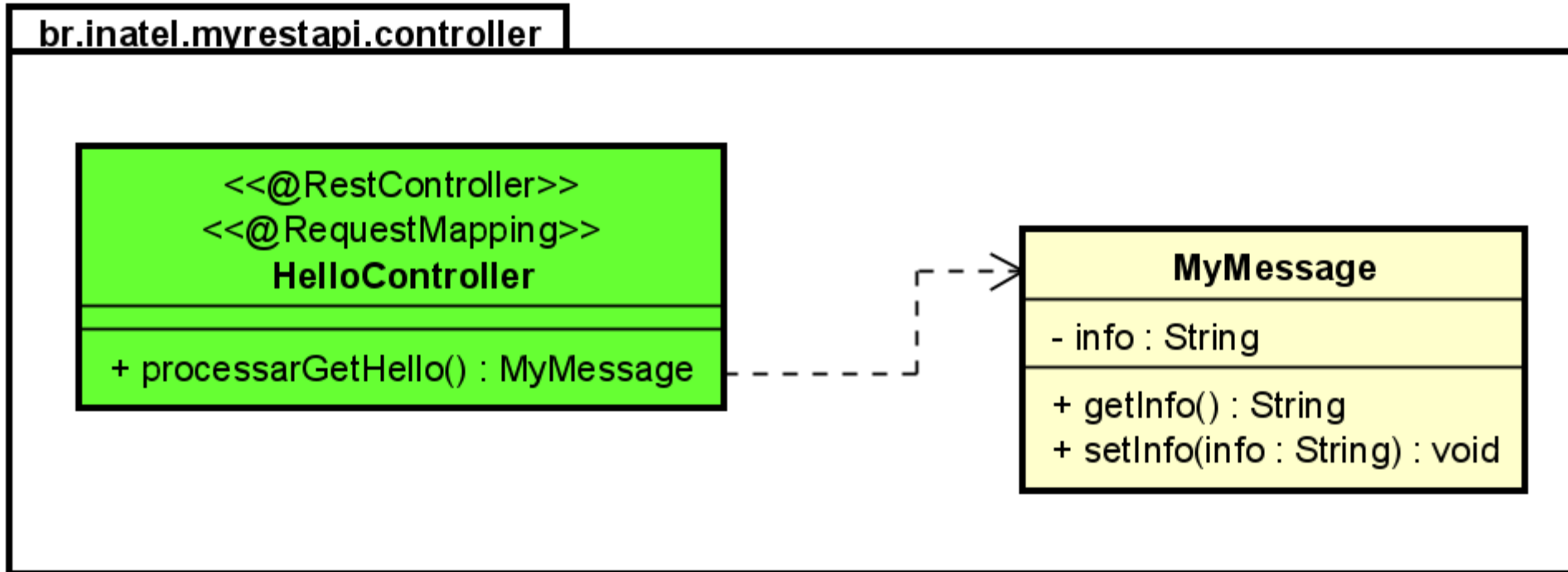


## Controller REST

Cliente pode ser outro Sistema, Evento JavaScript, Aplicativo Móvel, Dispositivo IoT, ...









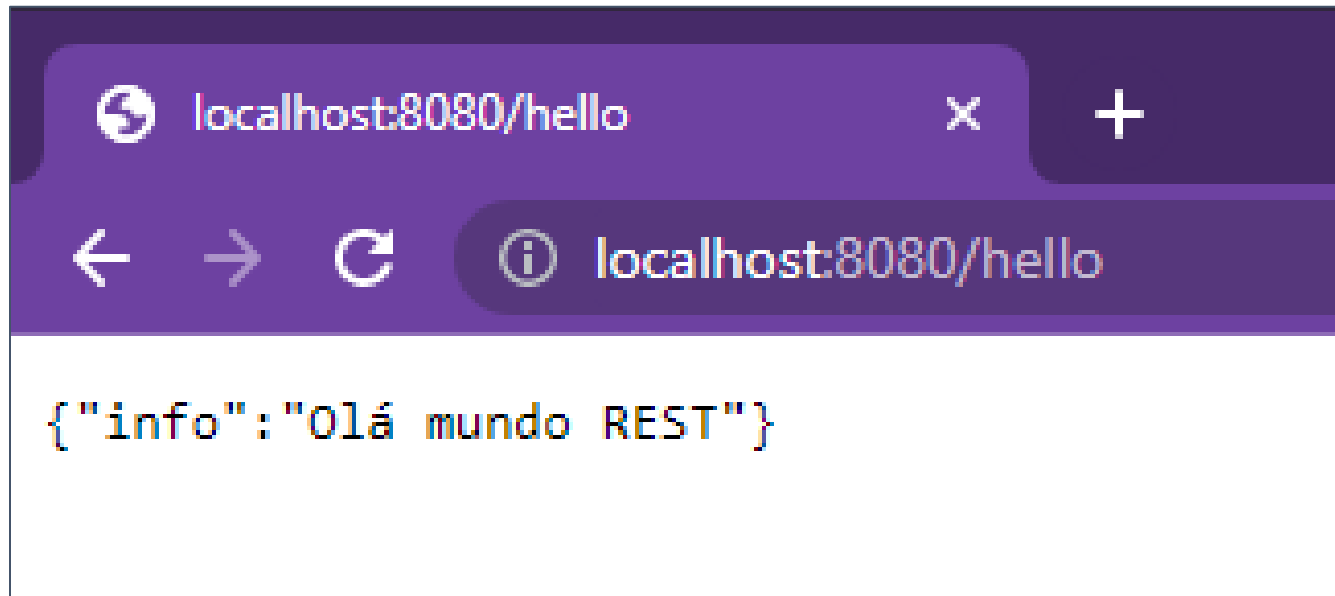
> Criar pacote **br.inatel.myrestapi.controller**:

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

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

```
HelloController.java X
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
```

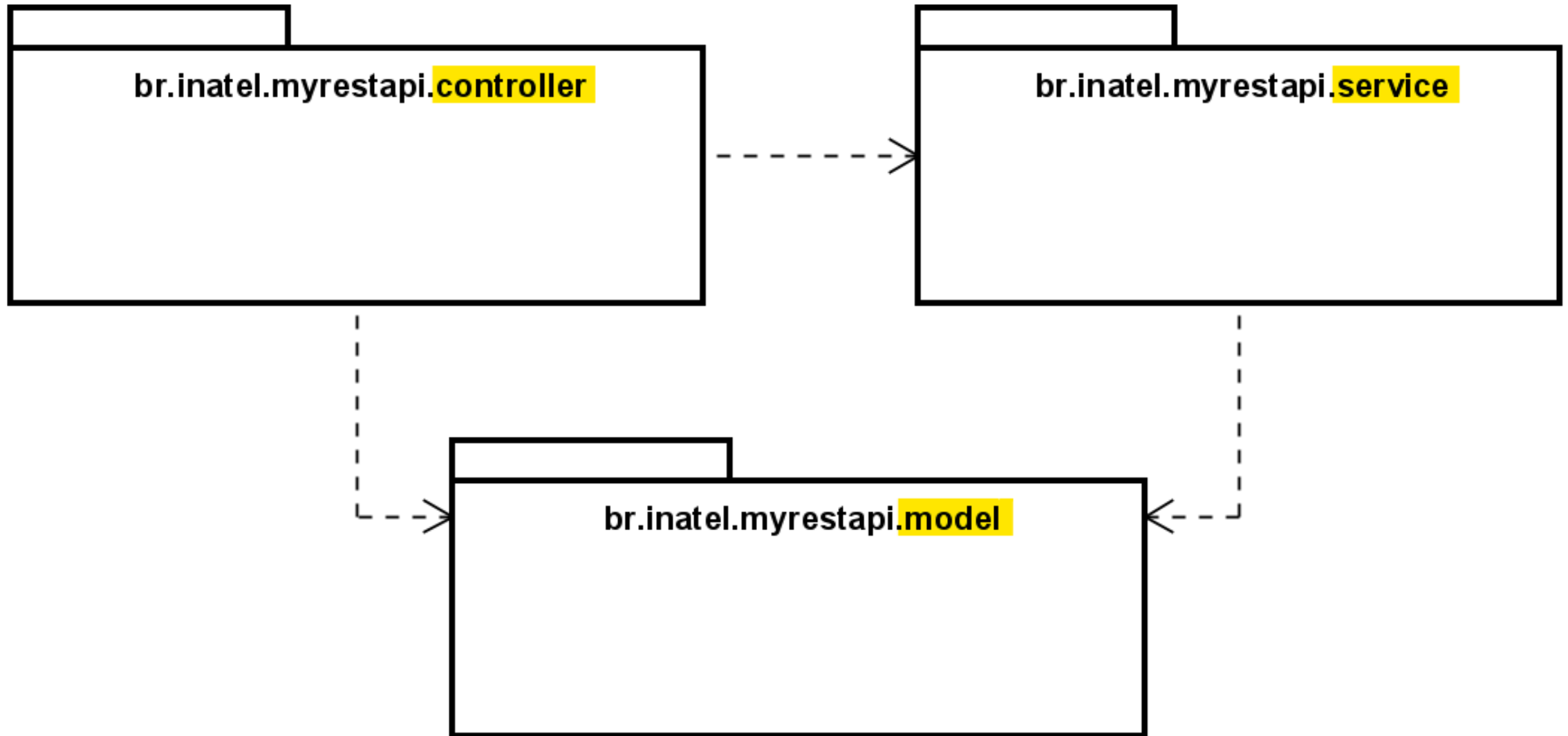
>Abrir o navegador e acessar: **localhost:8080/hello**

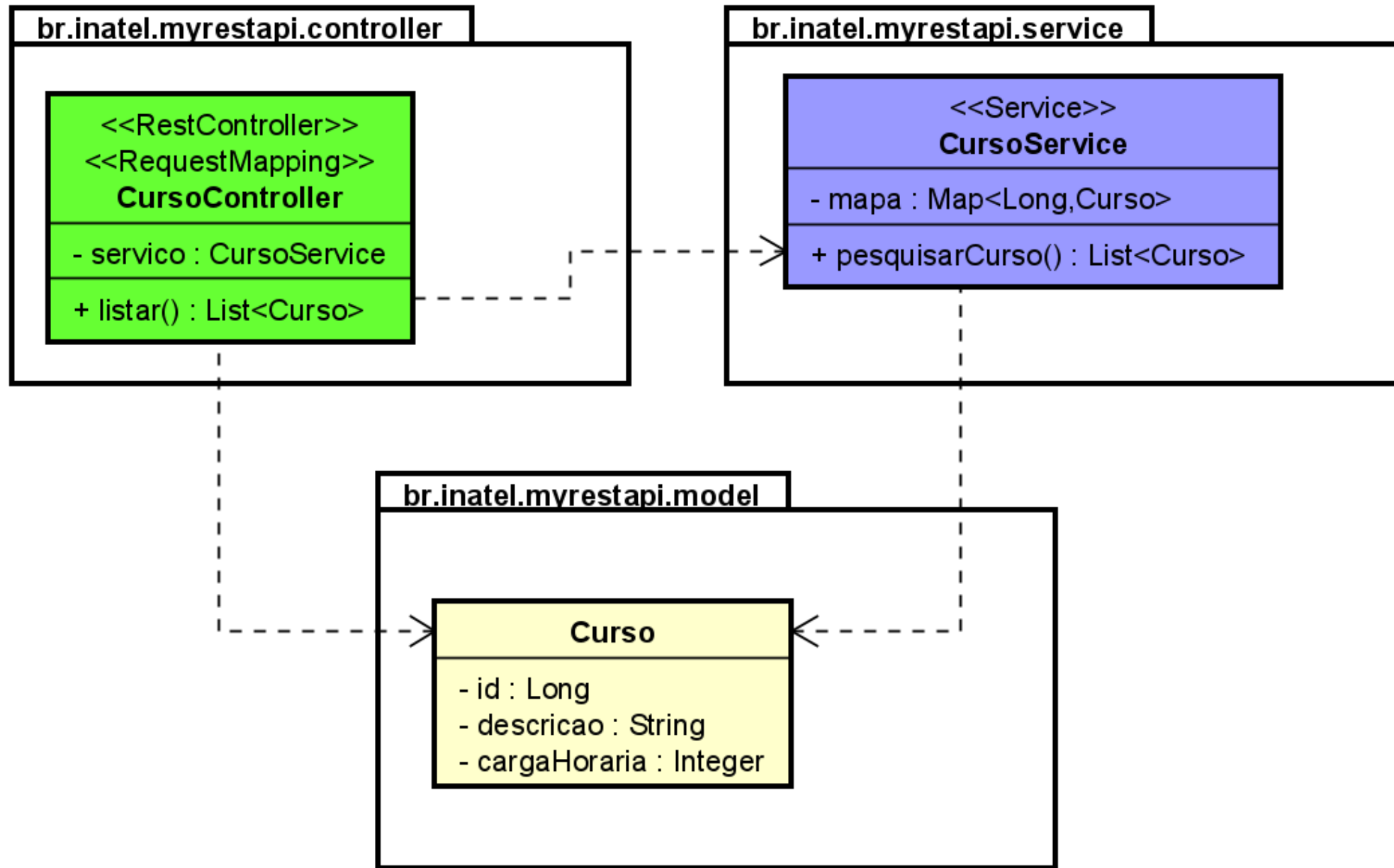


**Questão:**

**Quem fez a conversão MyMessage -> JSON?**

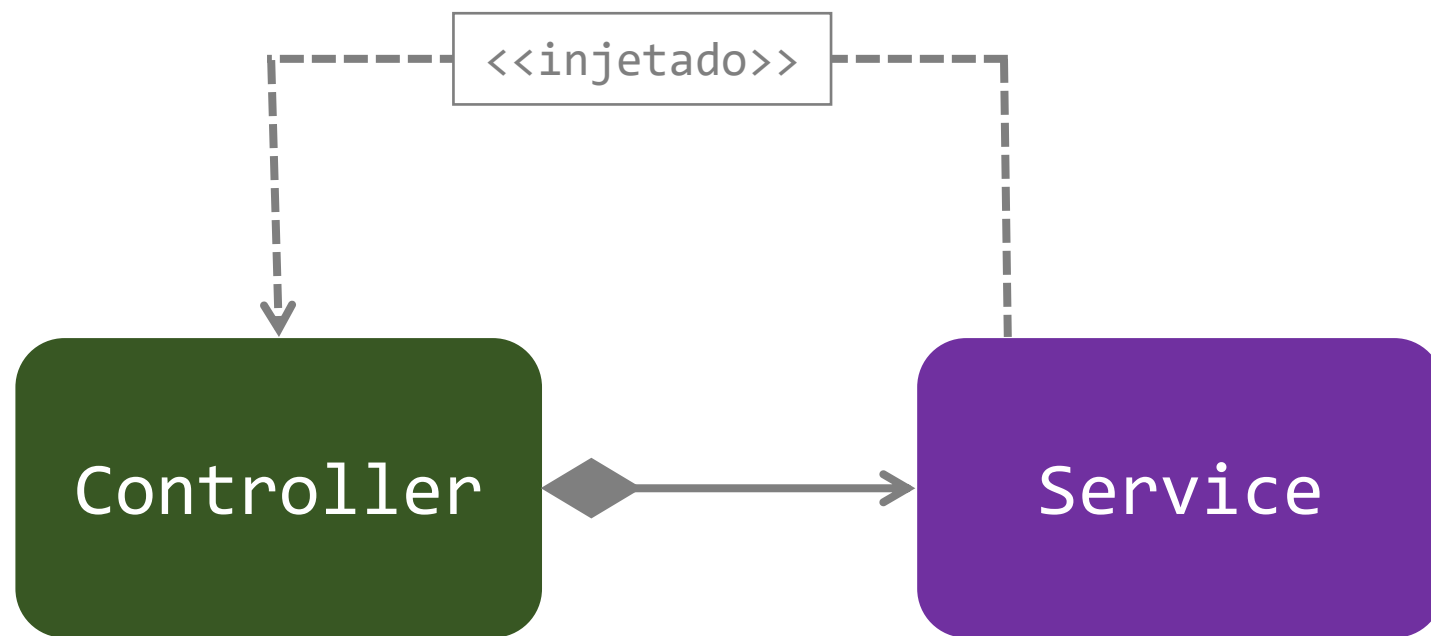
*Implementando um back-end  
completo de catálogo de cursos*





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
```

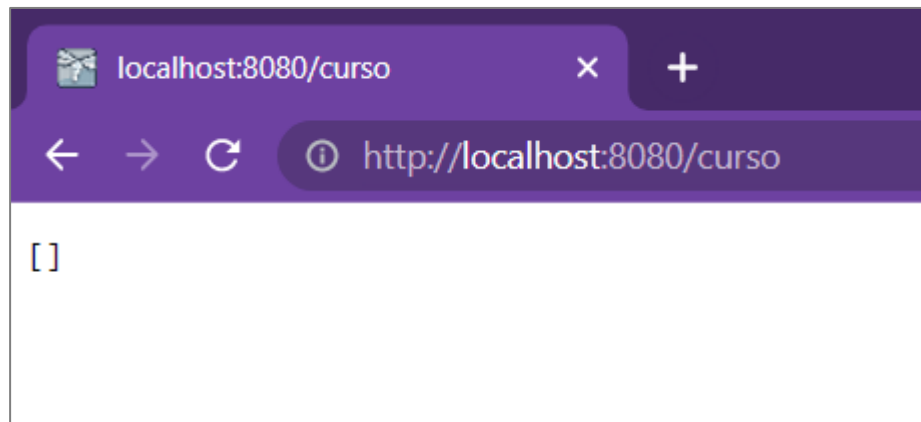
>Seguindo os slides anteriores:

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**



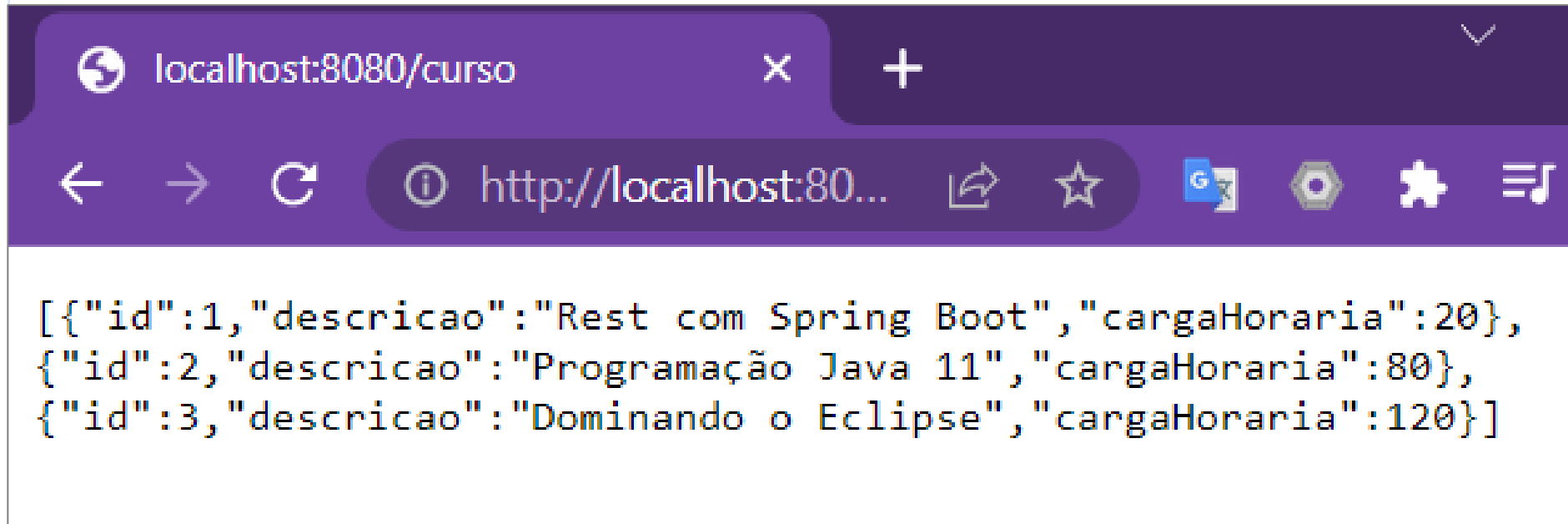
É esperado que não retorne  
nenhum resultado

>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:

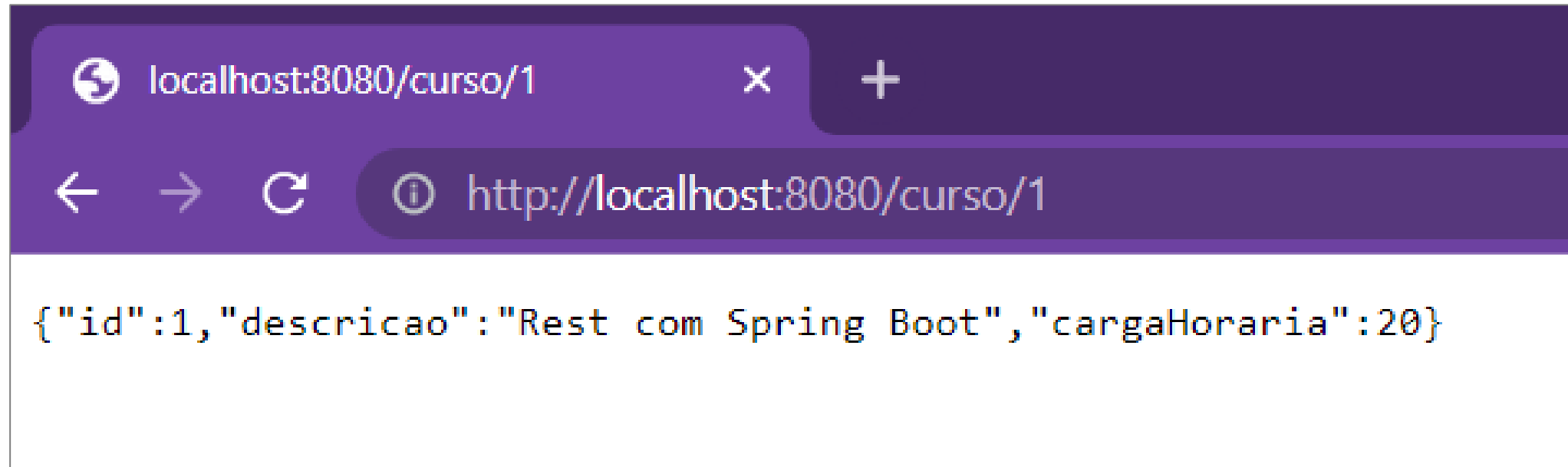
a) Na classe service, criamos um método que recebe o parâmetro referente ao ID de curso e retorna o curso guardado no mapa:

```
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 }
```

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:

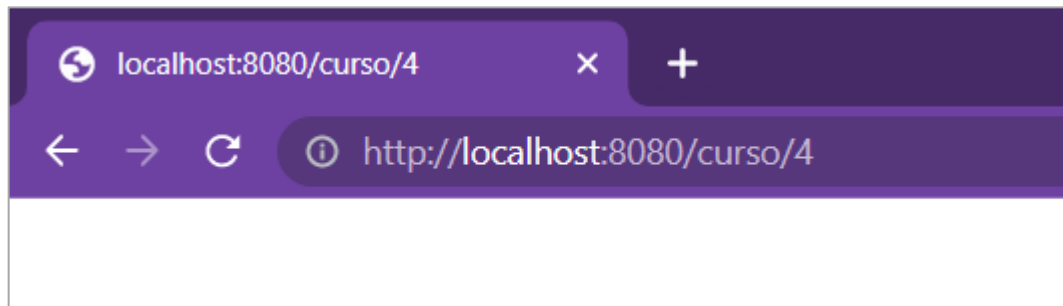
`http://localhost:8080/curso/1`

`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.*

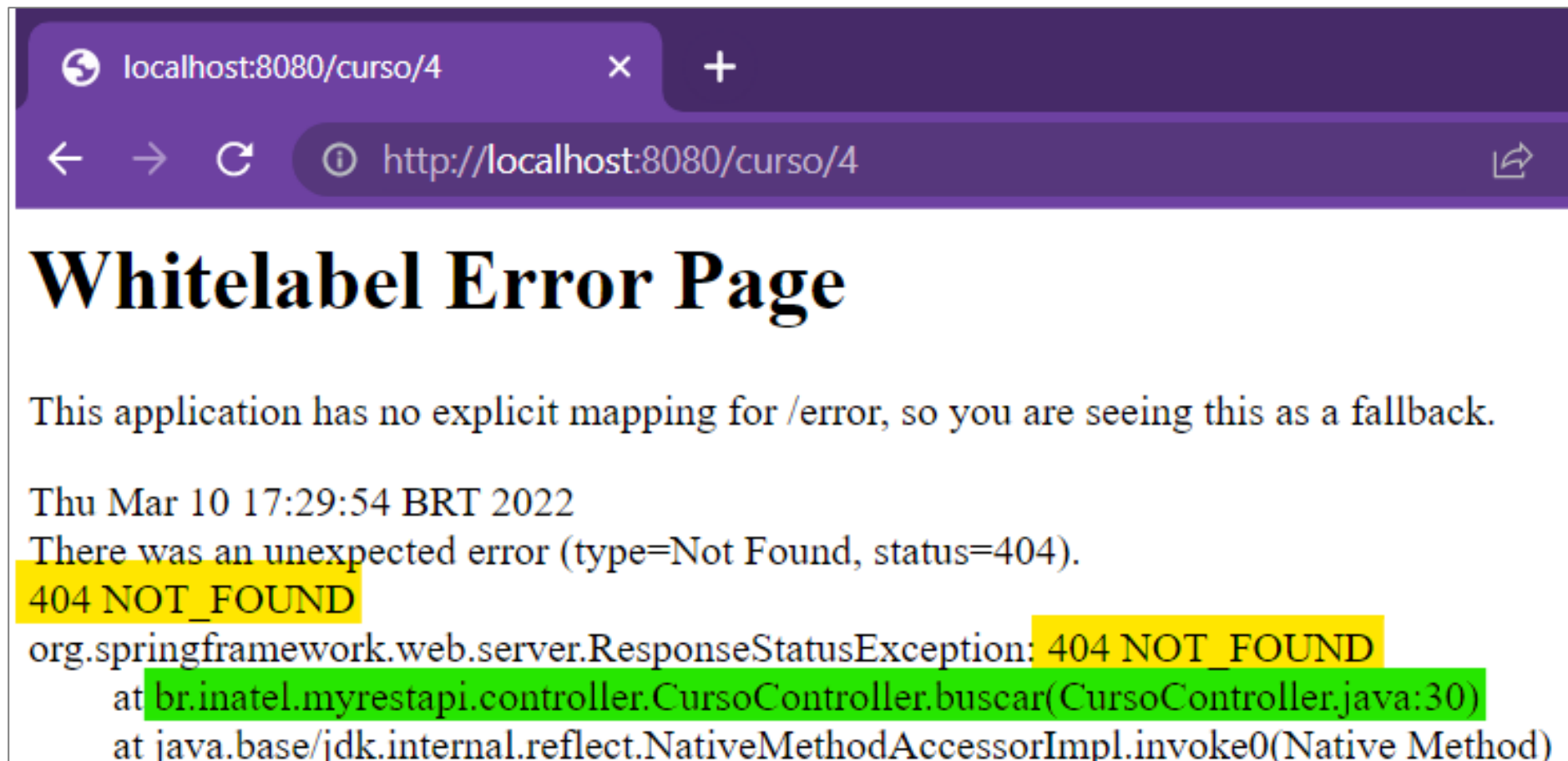
>Podemos adicionar um **status na resposta** sinalizando que tudo ocorreu bem, mas nada foi encontrado!

>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 consistem ainda em operações REST para:

- >criar curso
- >atualizar curso
- >remover um curso específico

Este um conjunto básico de operações e comumente chamamos de **CRUD** (Create, Retrieve, Update, Delete)

>Inicialmente, vamos implementar estas operações na classe **service**:

a)Criar um curso:

```
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:

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

c)Remover um curso:

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



>Na classe **controller**: declaramos os respectivos métodos:

a) Criar um curso:

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

Tem retorno para o cliente receber o ID do novo curso

b) Atualizar um curso:

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

Não precisa de retorno na atualização

c) Remover um curso:

```
@DeleteMapping("{id}")
public void remover(@PathVariable("id") Long cursoIdRemover) {
    servico.removerCurso(cursoIdRemover);
}
```

>Se guiando pelos slides anteriores:

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

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

>Com **Spring WebFlux**, é possível fazer requisições não-bloqueantes

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();
```

- > Buscar em [start.spring.io](http://start.spring.io) a dependência do WebFlux
- > Copiar o trecho e colar no pom.xml
- > Criar uma classe de teste chamada WebFluxTest
- > No método main, codificar usando o seguinte código:

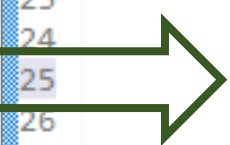
```
13 public static void main(String[] args) {  
14  
15     List<Curso> listaCurso = new ArrayList<Curso>();  
16  
17     Flux<Curso> flux = WebClient.create()  
18         .get()  
19         .uri("localhost:8080/curso")  
20         .retrieve()  
21         .bodyToFlux(Curso.class)  
22        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



O que aconteceu?

>Solução: adicionar na linha 25 a instrução `flux.blockLast();`

```
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     flux.blockLast();//bloqueia até a resposta chegar  
26  
27  
28     System.out.println("Lista de Cursos:");  
29     System.out.println( listaCurso );  
30 }  
31
```



## ***Exercício Desafio:***

Invoque todas as operações da API de maneira consistente:

- 1) Buscar todos os cursos
- 2) Inserir um curso
- 3) Conferir se realmente foi inserido
- 4) Atualizar um curso
- 5) Assegurar que foi atualizado
- 6) Remover um curso
- 7) Verificar se foi removido



>Configurar projeto para rodar no Docker

*Obrigado*