



Spring

Conceitos básicos para o desenvolvimento de APIs





Ecossistema Spring



- É um framework para Java;
- Agiliza e padroniza o desenvolvimento de
 - Aplicações WEB
 - Microserviços
 - Cloud
- Referência: <https://spring.io/>



Ecossistema Spring

(Principais frameworks)

Spring Web: O Spring Web é um framework que oferece suporte para o desenvolvimento de aplicativos **web** e **APIs RESTful**. Baseia-se no padrão MVC;

Spring Data JPA: Fornece suporte ao **acesso a dados relacionais (OO x Tabelas)** usando a API Java Persistence (JPA). Ele simplifica o desenvolvimento de camadas de acesso a dados, abstraindo grande parte do código repetitivo necessário para realizar operações comuns de persistência. Oferece recursos como consultas dinâmicas, paginação e ordenação.

Spring Security: framework de segurança que oferece recursos abrangentes para proteger aplicativos Java contra ameaças e garantir a **autenticação e autorização** corretas. Ele permite a configuração de autenticação baseada em formulário, autenticação baseada em token e integração com provedores de autenticação externos.

Arquitetura dos Laboratório INFO1



- Windows 7
- JDK 21
- Maven 3.9.9
- TomCat 10.1.39
- MySQL (XAMP)
- VSCode 1.70.2 (Última versão para Windows 7)
 - Extension Pack for Java
 - **Spring Boot Extension Pack (Instalar e Reiniciar)**
- Subdiretório de instalação dos aplicativos
 - Geral: c:\dev\pac2025
 - Maven: c:\dev\pac2025\maven
 - TomCat c:\dev\pac2025\tomcat

JDK Development Kit 21.0.4 downloads



Novo Projeto

Passos

- 1. Criar o projeto na WEB (<https://start.spring.io/>);**
2. Descompactar o arquivo em um subdiretório e
3. No VSCode: *File / Open Folder.*

OU

- CTRL + Shift + P;
- **Create a Initializr: Create a Maven Project.**

Novo Projeto - Spring Initializr (<https://start.spring.io/>)

 **spring initializr**

Project
 Gradle - Groovy
 Gradle - Kotlin Maven

Language
 Java Kotlin
 Groovy

Spring Boot
 4.0.0 (SNAPSHOT) 3.5.4 (SNAPSHOT) 3.5.3
 3.4.8 (SNAPSHOT) 3.4.7 3.3.13

Project Metadata

Group	br.cefeet
Artifact	sisvendaspring
Name	sisvendaspring
Description	Sisndda
Package name	br.cefeet.sisvendaspring
Packaging	<input type="radio"/> Jar <input checked="" type="radio"/> War
Java	<input type="radio"/> 24 <input checked="" type="radio"/> 21 <input type="radio"/> 17

Dependencies [ADD DEPENDENCIES... ⌘ + B](#)

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

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

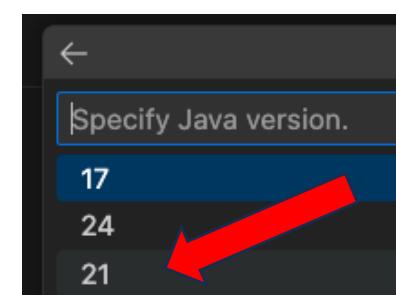
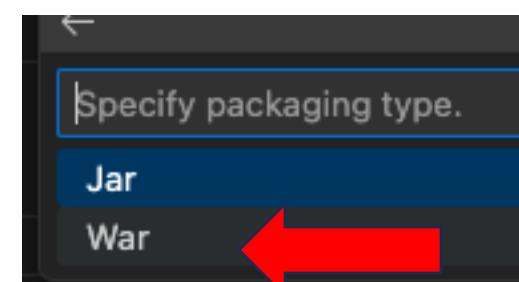
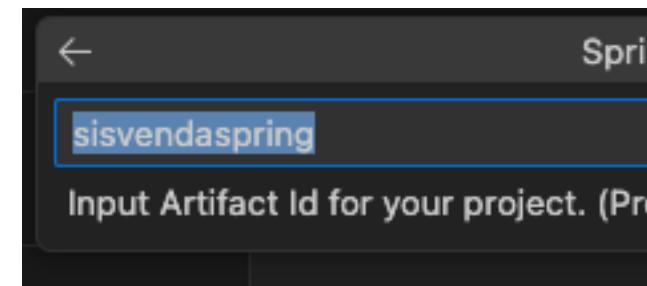
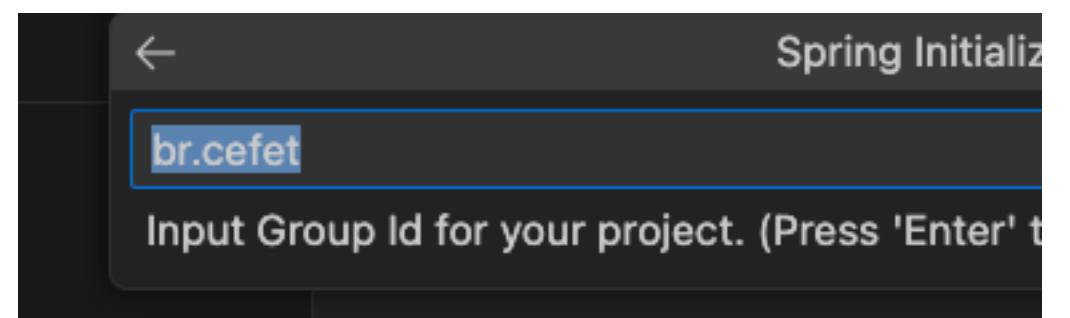
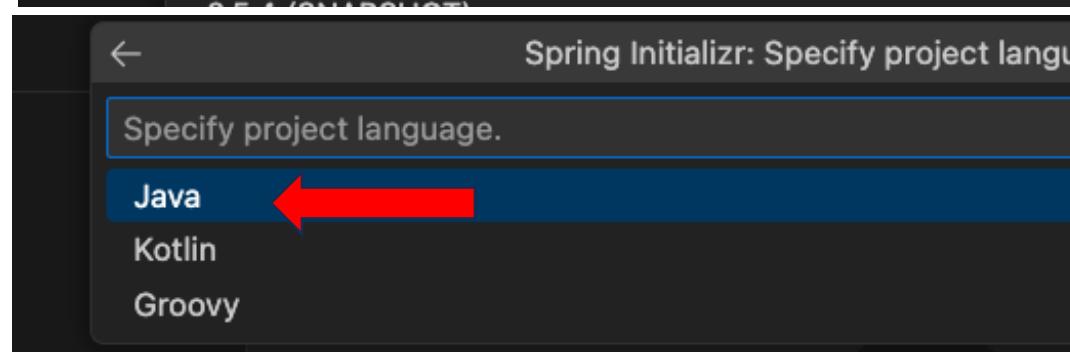
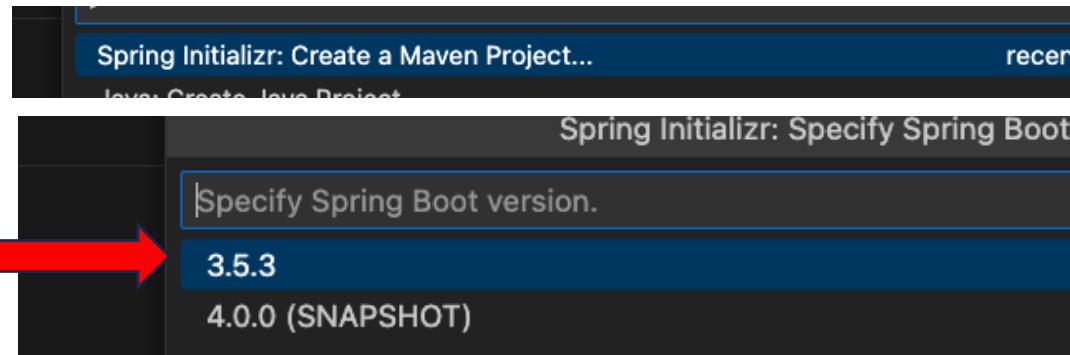
MySQL Driver SQL
MySQL JDBC driver.

[GENERATE ⌘ + ↵](#) [EXPLORE CTRL + SPACE](#) [...](#)

Novo Projeto



- CTRL + Shift + P
- Create a Initializr: Create a Maven Project



Novo Projeto



Spring Initializr: Choose dependencies

Search for dependencies.

Selected 3 dependencies

Press <Enter> to continue.

← Spring Initializr: Choose dependencies

spring

Spring Web Web
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as t...

Spring gRPC I/O

Search for dependencies.

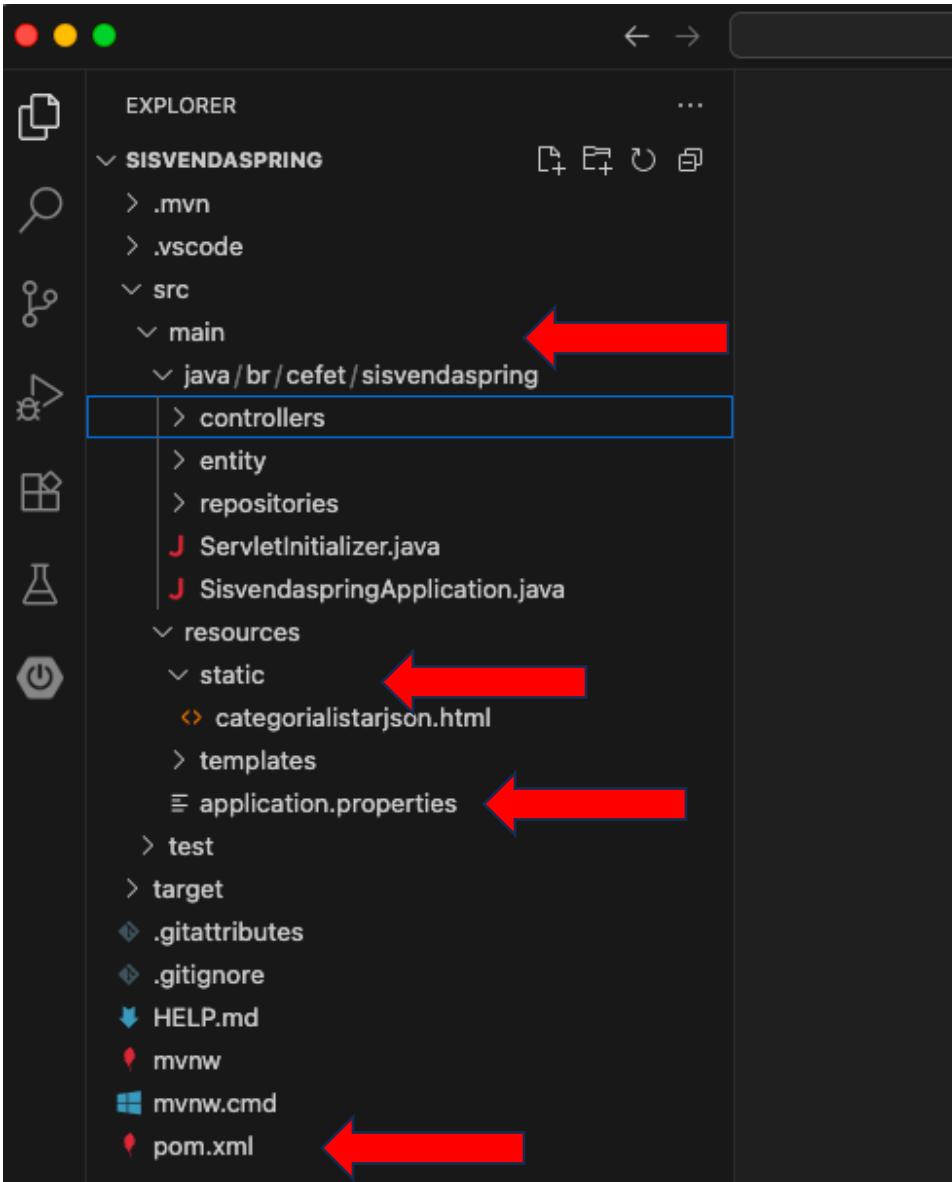
mysql

MySQL Driver SQL
MySQL JDBC driver.

jpa

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Ambiente - VSCode



Principais Subdiretórios e arquivos

- **main:** Código Java
JavaBeans
Controllers
- **Resource / Static:**
Páginas WEB (html+css+js)
- **Resource / application.Properties**
Arquivo de configuração ao Banco de dados
- **Pom.xml**
Arquivo de configuração do Maven

Maven

- É uma ferramenta para gerenciar projetos Java e outras linguagens;
- É responsável por configurar e baixar todas as dependências externas utilizadas no projeto;
- Ele automatiza a compilação/deploy/publicação do sistema;
- O arquivo pom.xml contém todas as dependências utilizadas no projeto.

Maven

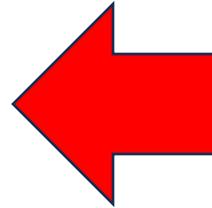
Arquivo pom.xml (Exemplo)

```
<properties>
    <java.version>21</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
```

Maven

Arquivo pom.xml (Exemplo)

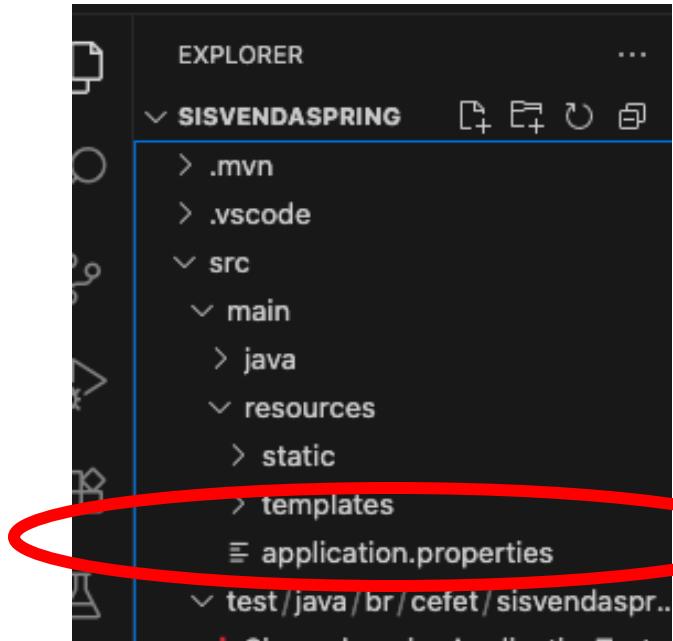
```
<build>
    <finalName>sisvendaspring</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```



Inclua essa linha, para que o arquivo final tenha esse nome + .war;

Configurando acesso ao BD

- Acesse o arquivo application.properties;
- Cole o conteúdo;



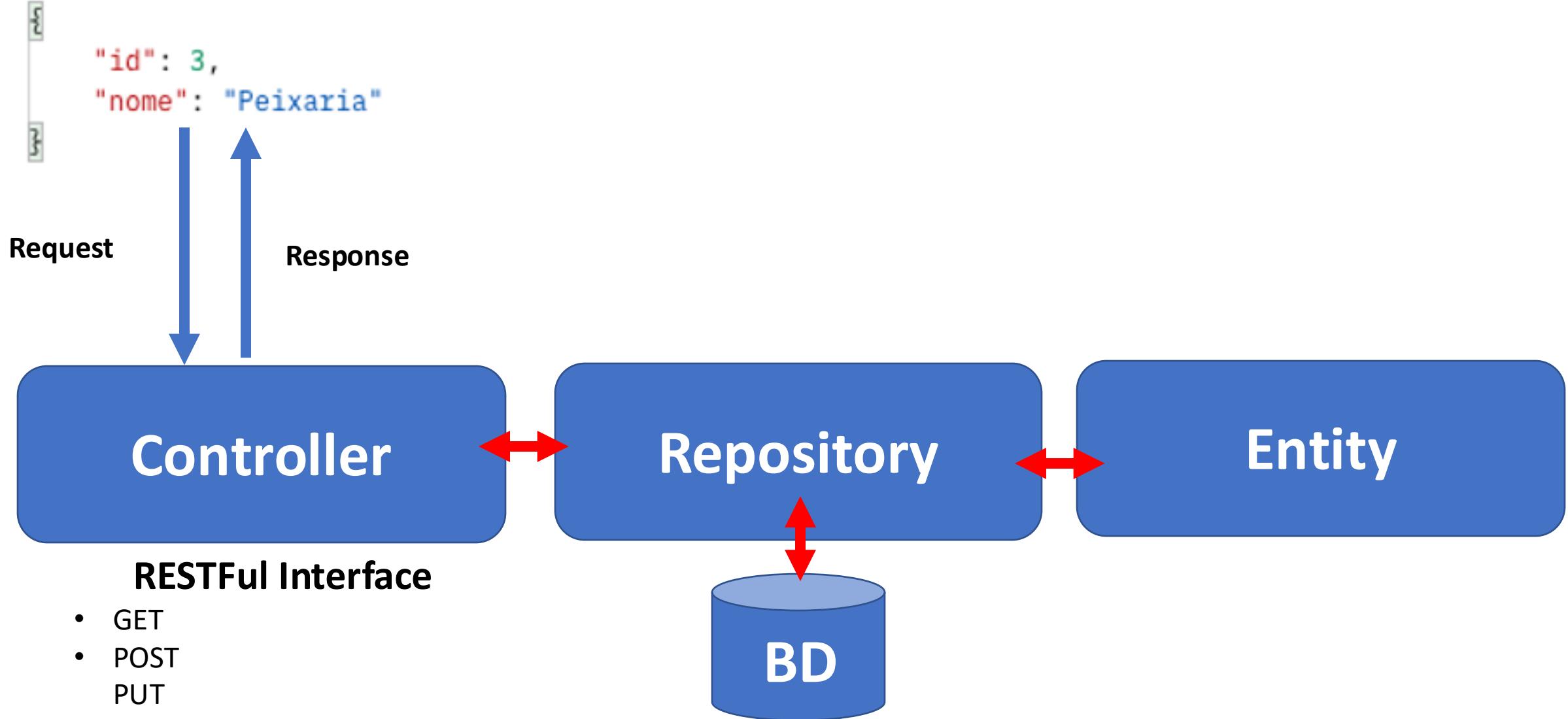
```
spring.application.name=sisvendaspring
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/dbspring
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.open-in-view:false
spring.jpa.show-sql: true
```

- **Propriedades:**
 - **ddl-auto:** cria /altera a estrutura de tabelas automaticamente no banco;
 - **dbspring** é o database
 - **show-sql:** mostra no terminal o sql executado;
- **Na interface do MySQL, execute, apenas:**

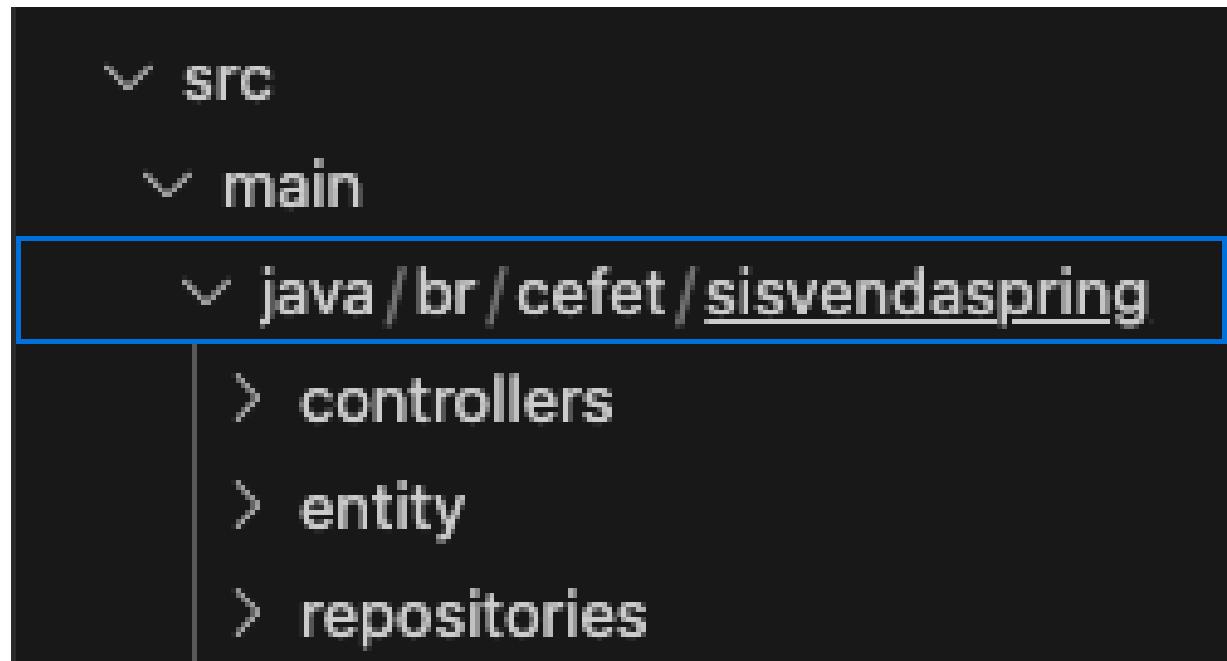
*CREATE DATABASE dbspring
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;*



Arquitetura da API (MVC)



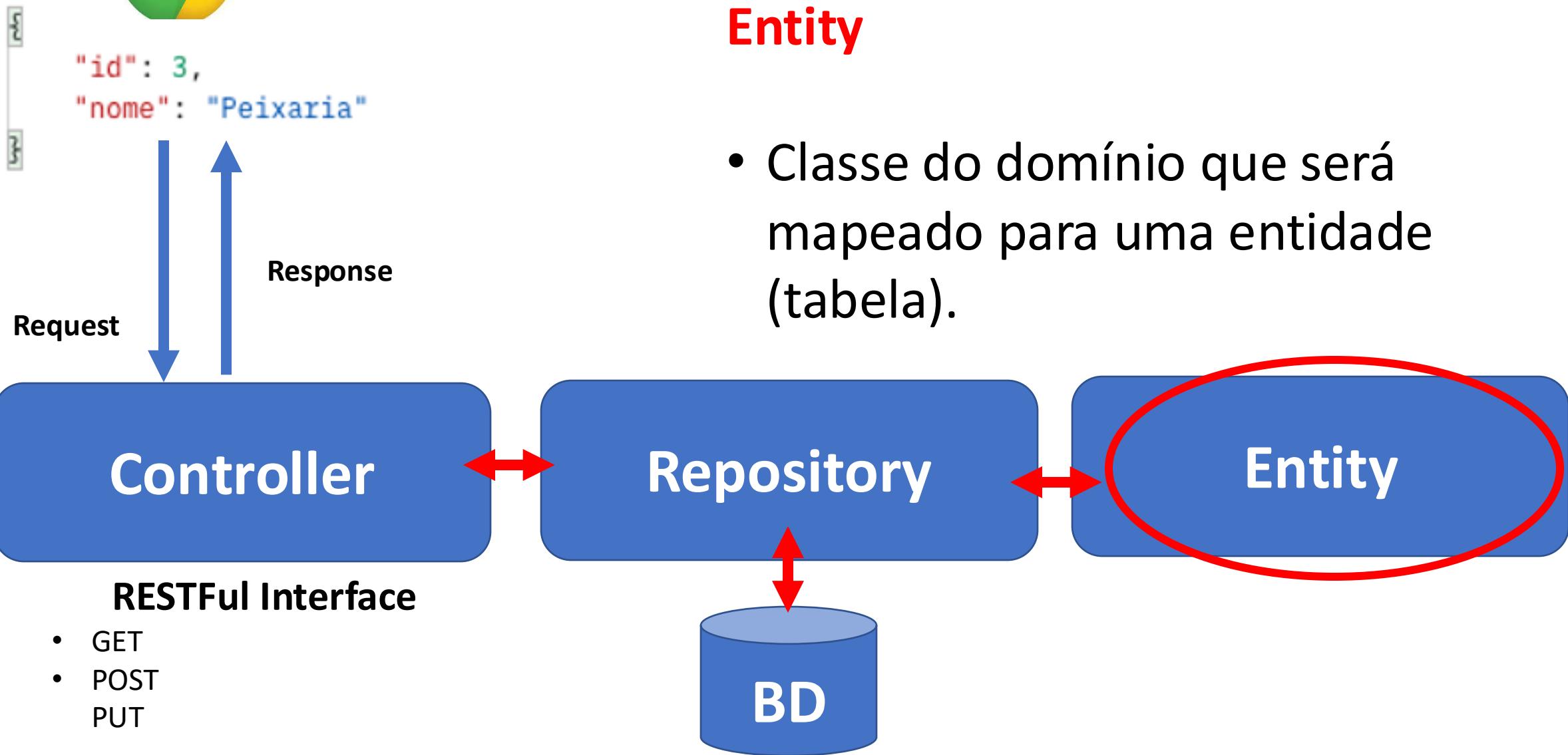
VSCode – Criar os pacotes



- **Entity**
 - Receberá as classes de modelo;
- **Repository**
 - Contém as interfaces que implementam (herdam) os recursos JPA;
 - Um REPOSITORY inclui todos os métodos, como classificação, paginação de dados e operações CRUD.
 - A notação @Repository é usada para realizar a especificação;
- **Controller**
 - Contém as classes com os métodos RESTFUL de acesso;



Arquitetura da API (MVC)

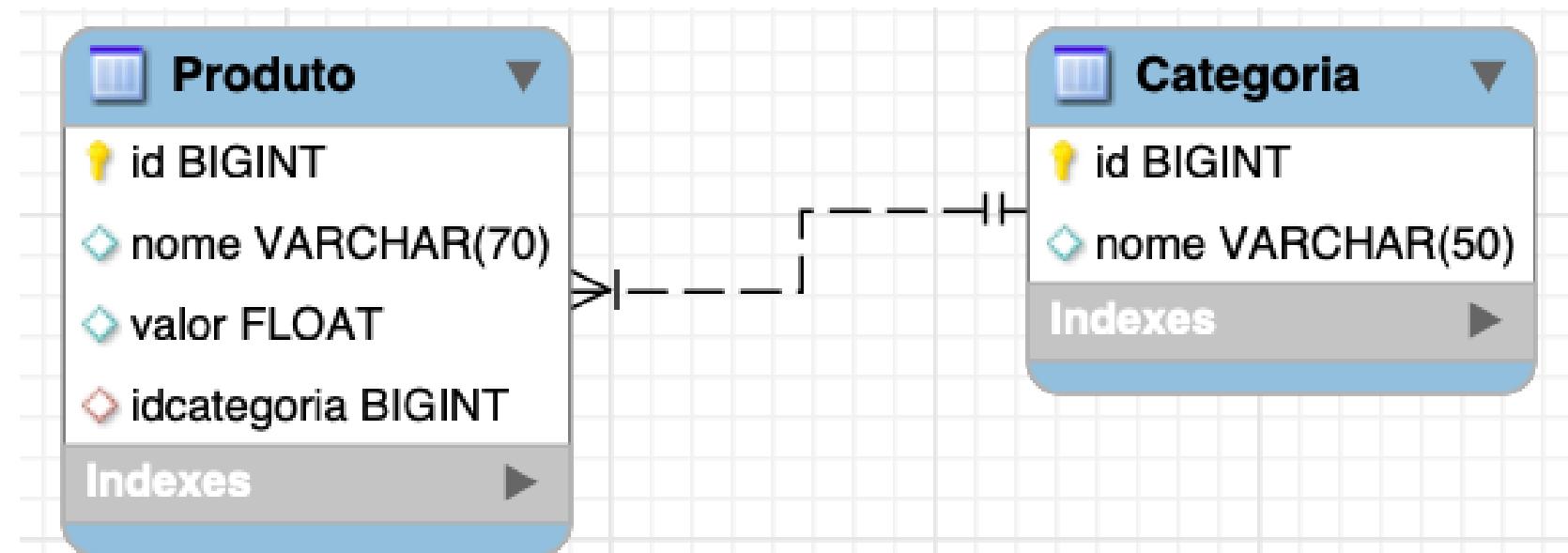
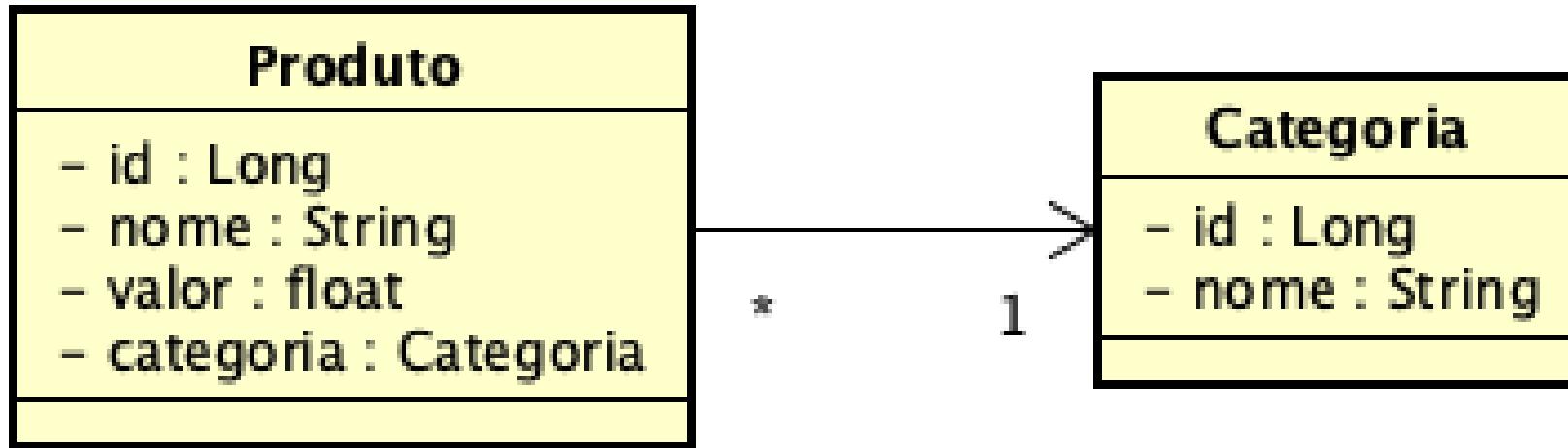


Spring Data JPA



- JPA (Java Persistence API) é um framework que tem o objetivo de persistir objetos Java em banco de dados;
- Operacionaliza as funções de acesso ao SGBD (Ex.: insert, delete ou select...);
- Permite o Mapeamento Objeto-Relacional (ORM - Object-Relational Mapping);
- Spring JPA facilita a implementação de repositórios JPA e
- As classes são mapeadas com anotações (@);

Spring Data JPA - Mapeamento



Spring Data JPA - Anotações



\\Indica que a classe será um entidade

@Entity

\\Indica o nome da tabela

@Table(name = "categoria")

public class Categoria {

\\Indica a primary key da entidade indicando auto_increment

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

public Long id;

public String nome;

* Lembre-se realizar o import de cada anotação utilizada.

Spring Data JPA - Anotações



```
@Entity  
public class Produto {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    private String nome;  
    private float valor;
```

\\Indica o relacionamento entre entidades

```
@ManyToOne  
@JoinColumn(name = "idcategoria")  
private Categoria categoria;
```

@OneToOne
@ManyToOne
@OneToMany
@ManyToMany

Spring Data JPA - Exemplo



```
@Entity  
@Table(name = "usu") \\\ Quando o nome da tabela for diferente  
public class Usuario {  
  
\\\ Quando o nome do campo for diferente do atributo  
@Column(name = "nome_usuario")  
@Column(length = 80) // Cria um VARCHAR(80)  
private String nome;
```

SpringToolSuite – Criar as classes model

```
src
└── main
    └── java
        └── br
            └── cefet
                └── sisvendaspring
                    ├── controllers
                    └── entity
                        ├── Categoria.java
                        └── Produto.java
                    └── repositories
```

```
package br.cefet.sisvenda.model;
```

```
public class Produto {
    private Long id;
    private String nome;
    private float valor;
    private Categoria categoria;
    ...
}
```

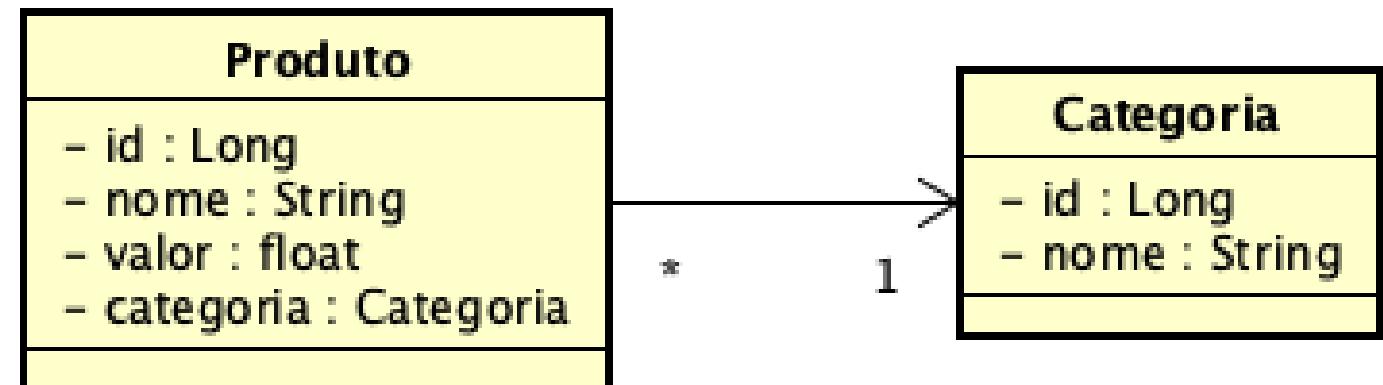
```
package br.cefet.sisvenda.model;
```

```
public class Categoria {
```

```
    public Long id;
```

```
    public String nome;
```

```
....
```



SpringToolSuite – Mapeamento Objeto Relacional - Categoria

```
package br.cefe.sisvendaspring;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
@Table(name = "categoria")
public class Categoria {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public Long id;
    public String nome;
```

SpringToolSuite – Mapeamento Objeto Relacional - Produto

```
@Entity
@Table(name = "produto")
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private float valor;
    @ManyToOne
    @JoinColumn(name = "idcategoria")
    private Categoria categoria;
```

VSCode – Executar o projeto

No terminal,

Compilar: mvn package

Copiar o arquivo: copy .\target\sisvendaspring.war c:\dev\pac2025\tomcat\webapps

Executar o TomCat: startup.bat

MySQL – Criar o banco de dados

Compile o projeto

mvn package

Se as tabelas forem criadas, provavelmente o mapeamento objeto-relacional foi feito corretamente.

MySQL – Criar o banco de dados – Código completo

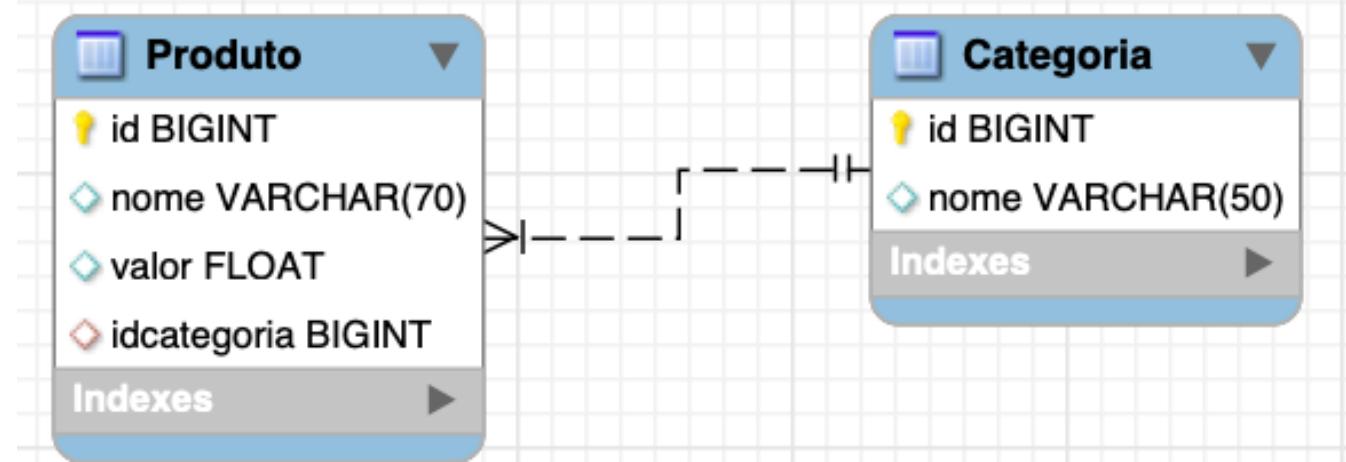
```
CREATE DATABASE dbspring CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
use dbspring;
```

```
create table categoria(  
id bigint auto_increment not null primary key,  
nome varchar(50));
```

```
create table produto(  
id bigint auto_increment,  
nome varchar(100),  
valor float,  
idcategoria bigint,  
primary key (id),  
foreign key (idcategoria) references categoria(id));
```

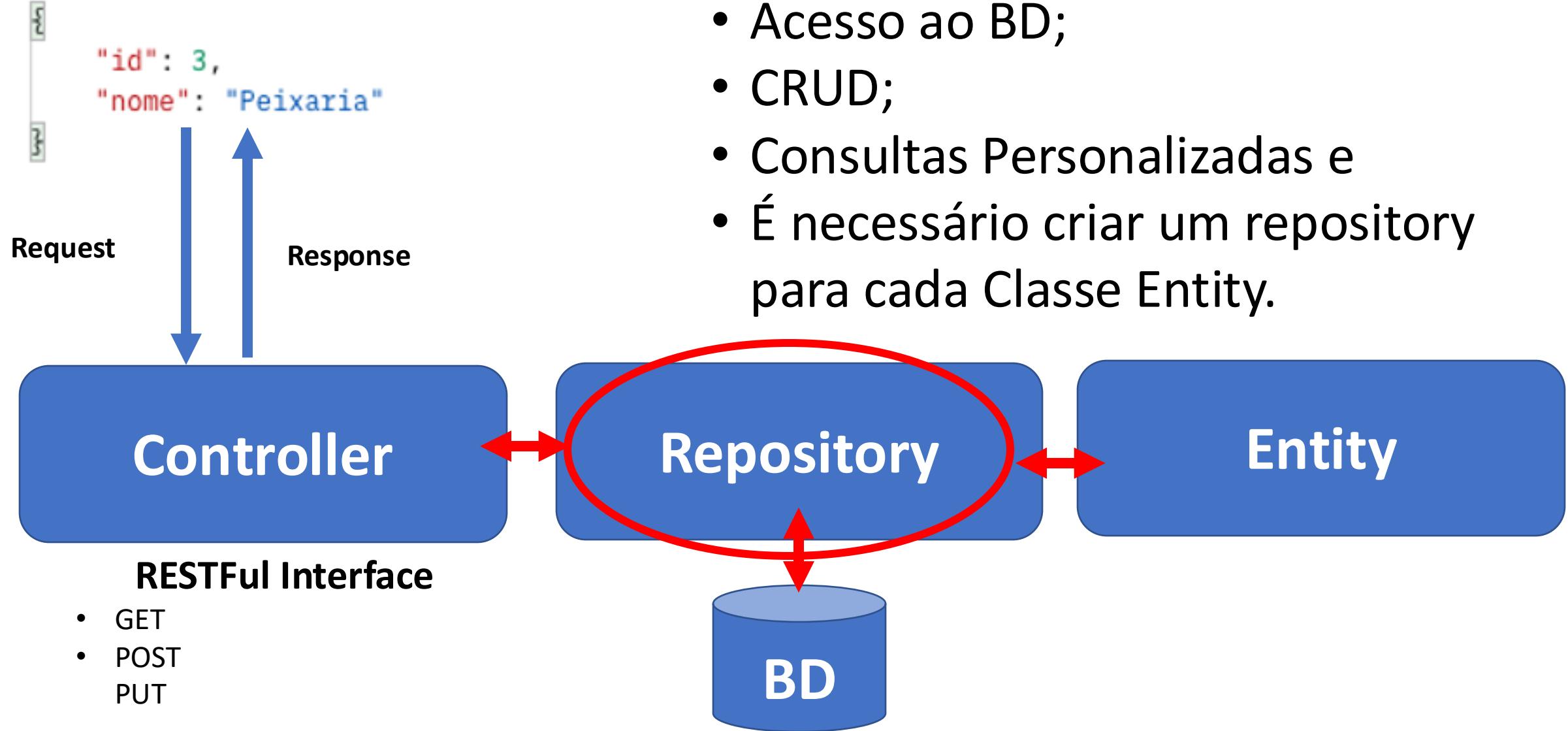
-- Popule as Tabelas

```
insert into categoria (nome) values ('Limpeza');  
insert into categoria (nome) values ('Padaria');  
insert into categoria (nome) values ('Peixaria');  
insert into produto(nome, valor, idcategoria) values ('Cloro',4.99,1);  
insert into produto(nome, valor, idcategoria) values ('Pão Plustiva',6.99,2);  
insert into produto(nome, valor, idcategoria) values ('Filé de Tilápia',6.99,2);
```

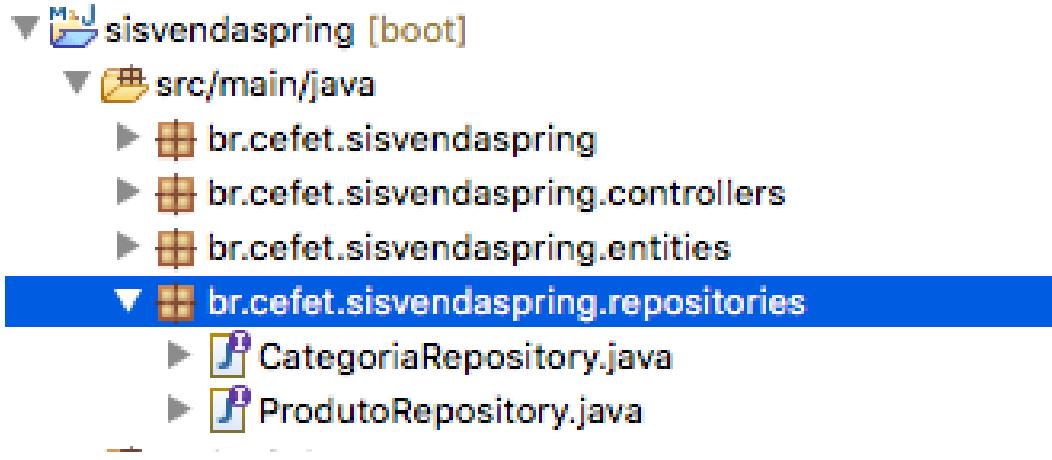




Arquitetura da API (MVC)



Repository - Categoria



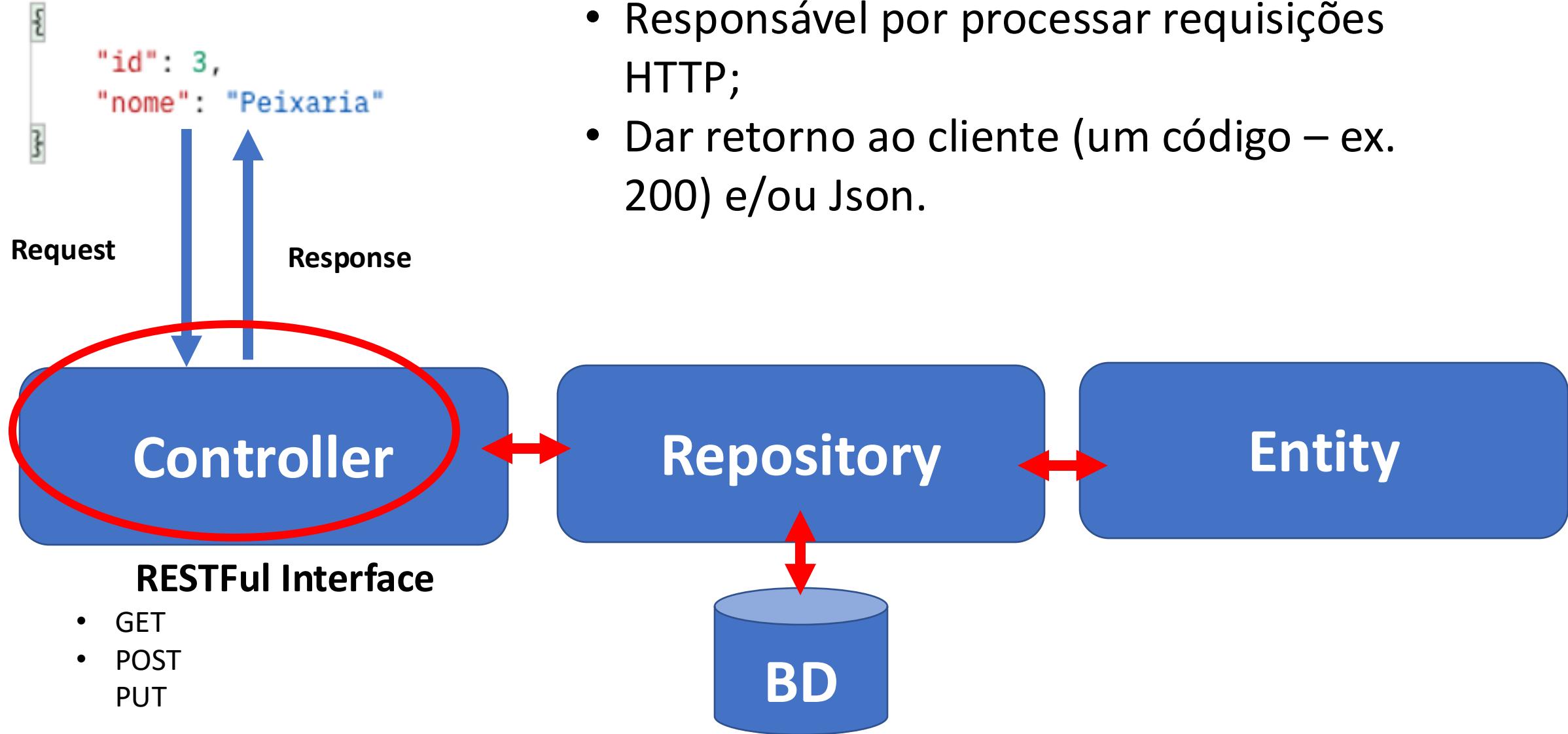
```
package br.cefet.sisvendaspring.repositories;
import org.springframework.data.jpa.repository.JpaRepository;
import br.cefet.sisvendaspring.entities.Categoria;

public interface CategoriaRepository extends JpaRepository <Categoria, Long> {

}
```



Arquitetura da API (MVC)



Controller

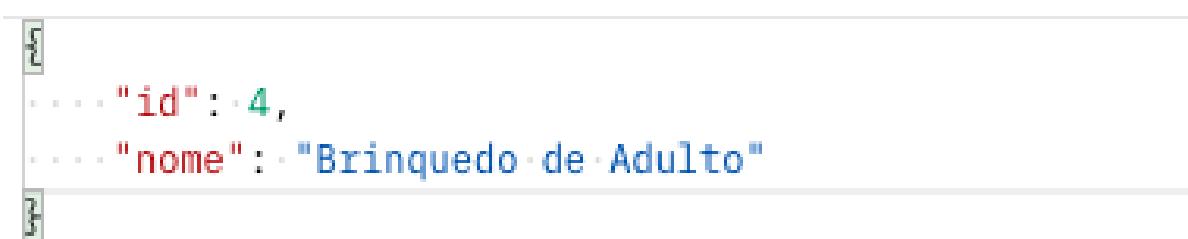
- Responsável por processar requisições HTTP;
- Dar retorno ao cliente (um código – ex. 200) e/ou Json.

Padrão RESTful

Arquitetura padrão

VERBO HTTP	URL (RESTful)	OPERAÇÃO
GET	sisvendaspring/categorias	Lista todas as categorias
GET	sisvendaspring/categorias/3	Retorna uma categoria
DELETE	sisvendaspring/categorias/3	Exclui uma categoria
POST	sisvendaspring/categorias/	Insere uma categoria
PUT	sisvendaspring/categorias/	Atualiza uma categoria

```
1  {
2    "id": 1,
3    "nome": "Limpeza"
4  },
5  {
6    "id": 2,
7    "nome": "Padaria"
8  },
9  {
10   "id": 4,
11   "nome": "Brinquedo de Adulto"
12 }
13 }
```

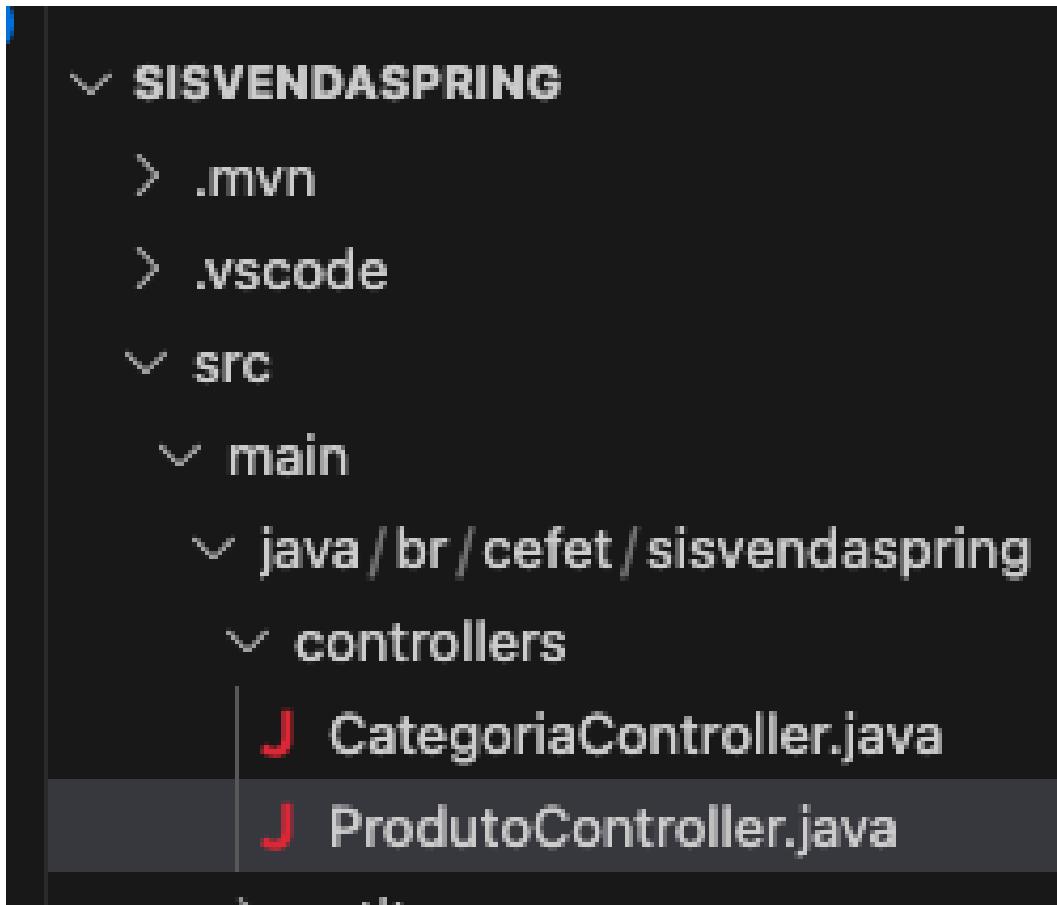


A screenshot of a JSON viewer application. It displays a list of four categories. Each category is represented by an object with two properties: 'id' and 'nome'. The 'id' property is highlighted in red, and the 'nome' property is highlighted in blue. The categories are: 1. Limpeza, 2. Padaria, 3. (empty object), and 4. Brinquedo de Adulto.

```
... "id": 4,
... "nome": "Brinquedo de Adulto"
```

Exemplos JSON

Controller



Controller - Principais Anotações do Spring Web

@Controller

Define uma classe como controladora no padrão MVC, permitindo a manipulação de solicitações HTTP.

@RestController

Combina @Controller e @ResponseBody, simplificando o retorno de dados diretamente em formato JSON ou XML.

@RequestMapping

Configura mapeamentos de URLs para métodos ou classes, definindo como as solicitações HTTP são tratadas.

@GetMapping, @PostMapping, @PutMapping, @DeleteMapping, @PatchMapping

Específicas para métodos HTTP

Controller - Categorias

```
@RestController  
 @RequestMapping(value = "/categorias")  
 public class CategoriaController {  
     @Autowired  
     private CategoriaRepository repository;  
  
     @GetMapping  
     public List<Categoria> findAll(){  
         return repository.findAll();  
     }  
 }
```

Controller - Categorias

```
@GetMapping(value = "/{id}")
public Categoria findAll(@PathVariable Long id) {
    return repository.findById(id).get();
}

@PostMapping
public Categoria insert(@RequestBody Categoria categoria) {
    return repository.save(categoria);
}

@DeleteMapping("/{id}")
public void delete(@PathVariable Long id){
    repository.deleteById(id);
}

@PutMapping("/")
public void update(@RequestBody Categoria categoria) {
    repository.save(categoria);
}
```

Controller – Categoria – Código Final

```
@RestController
@RequestMapping(value = "/categorias")
public class CategoriaController {
    @Autowired
    private CategoriaRepository repository;

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

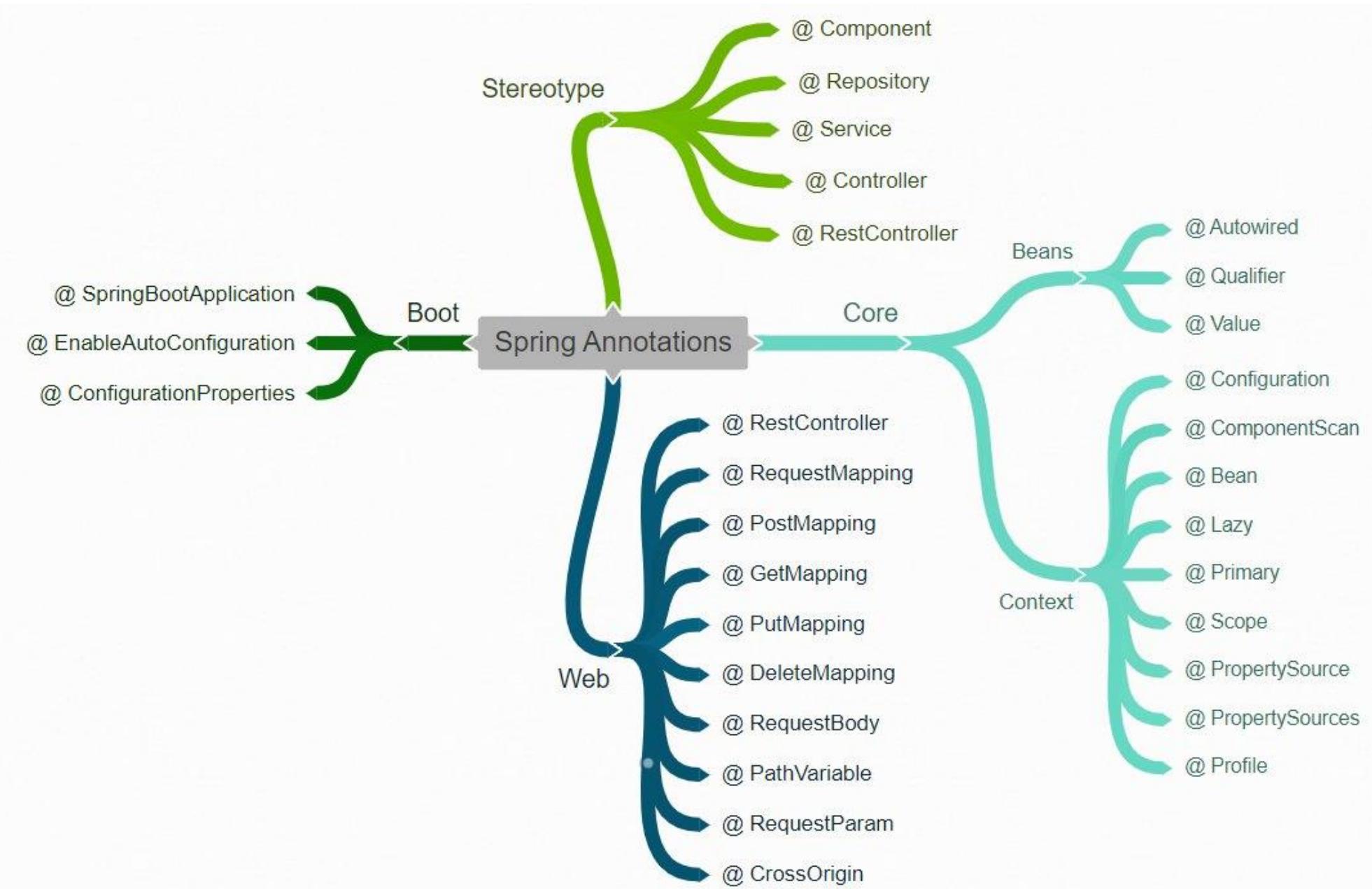
    @GetMapping(value = "/{id}")
    public Categoria findAll(@PathVariable Long id) {
        return repository.findById(id).get();
    }

    @PostMapping
    public Categoria insert(@RequestBody Categoria categoria) {
        return repository.save(categoria);
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id){
        repository.deleteById(id);
    }

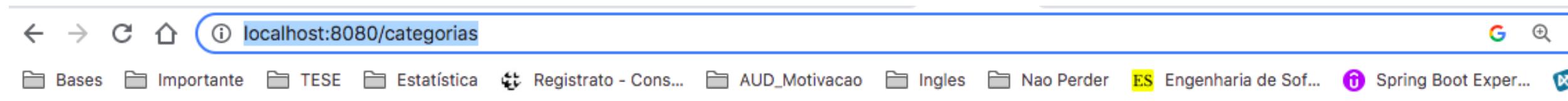
    @PutMapping("/")
    public void update(@RequestBody Categoria categoria) {
        repository.save(categoria);}}
```

Resumo - Annotations



Testar - CATEGORIA

GET: `http://localhost:8080/sisvendaspring/categorias`



```
[{"id":1,"nome":"Limpeza"}, {"id":2,"nome":"Padaria"}, {"id":3,"nome":"Peixaria"}]
```

Testar Listar Todas

Sisvenda - Spring GET http://localhost:8080/categorias + ... No Environment

http://localhost:8080/categorias

GET http://localhost:8080/categorias

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

GET: http://localhost:8080/sisvendaspring/categorias

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 22 ms Size: 244 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [ { 2   "id": 1, 3   "nome": "Limpeza" 4 }, 5 { 6   "id": 2, 7   "nome": "Padaria" 8 }, 9 { 10   "id": 3, 11   "nome": "Peixaria" 12 } 13 ] 14
```

Testar – GET {ID}

GET Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Ed
	Key	Value	Description		

GET: http://localhost:8080/sisvendaspring/categorias/3

Body Cookies Headers (5) Test Results Status: 200 OK Time: 188 ms Size: 190 B Save Response

Pretty Raw Preview Visualize JSON  

```
1 {  
2   "id": 3,  
3   "nome": "Peixaria"  
4 }
```

Testar - DELETE

The screenshot shows the Postman application interface. At the top, the URL `http://localhost:8080/categorias/3` is entered. In the top right corner, there are buttons for **Save**, **Send**, and other options. Below the URL, the method is set to **DELETE** and the target URL is again `http://localhost:8080/categorias/3`. The **Params** tab is selected, showing a table for Query Params:

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

DELETE: http://localhost:8080/sisvendaspring/categorias/3

The screenshot shows the Postman interface after sending the DELETE request. The status bar at the top indicates **Status: 200 OK**, **Time: 390 ms**, and **Size: 123 B**. The **Body** tab is selected, showing the response body which contains the number **1**.

Testar - Inserir

http://localhost:8080/categorias/

POST http://localhost:8080/categorias/

Body (8) **JSON**

```
1
2   "nome": "Brinquedo"
3
```

Params Authorization Headers (8) Body **JSON** Pre-request Script Tests Settings Cookies Beautify

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

http://localhost:8080/sisvendaspring/categorias/

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON **JSON**

```
1
2   "id": 4,
3   "nome": "Brinquedo"
4
```

Status: 200 OK Time: 189 ms Size: 191 B Save Response

Testar - Atualizar

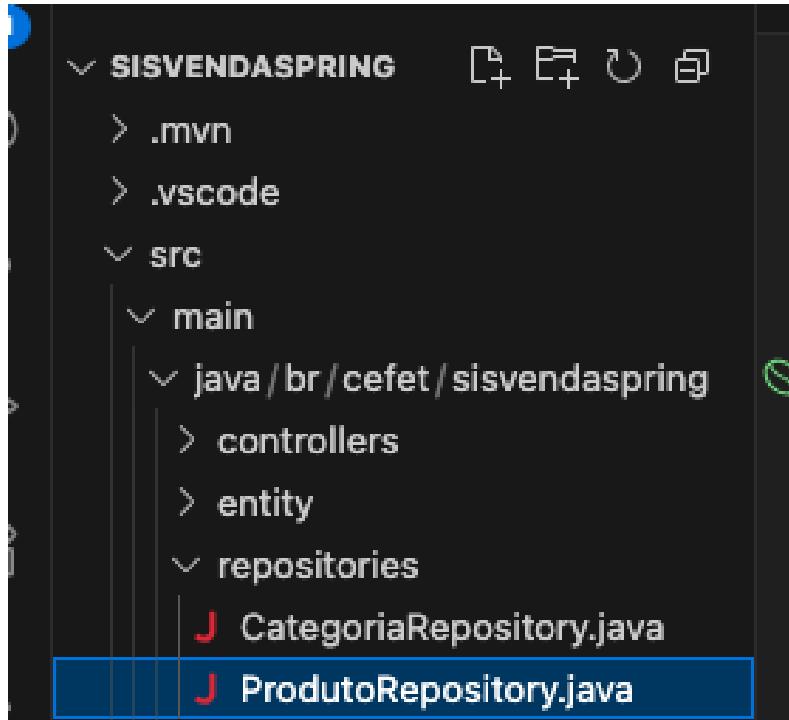
The screenshot shows the Postman application interface for testing a REST API. A red oval highlights the method 'PUT' and the URL 'http://localhost:8080/categorias/'. Another red oval highlights the 'Body' tab, which is selected (indicated by a green dot). Within the 'Body' tab, a third red oval highlights the 'JSON' option under the media type dropdown. The JSON payload in the body editor is:

```
1
2 ... "id": 4,
3 ... "nome": "Brinquedo de Adulto"
4
```

Below the request details, the URL `http://localhost:8080/sisvendaspring/categorias/` is displayed.

The screenshot shows the Postman interface after sending the request. The 'Body' tab is active at the bottom left. The response details are shown at the top right, including the status code 'Status: 200 OK', time 'Time: 65 ms', and size 'Size: 123 B'. Below the tabs, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'Text' (with a dropdown arrow), along with icons for copy and search. The response body contains a single digit '1'.

Repository - Produto



```
package br.cefet.sisvendaspring.repositories;  
import br.cefet.sisvendaspring.entity.Produto;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface ProdutoRepository extends JpaRepository<Produto, Long> {  
}
```

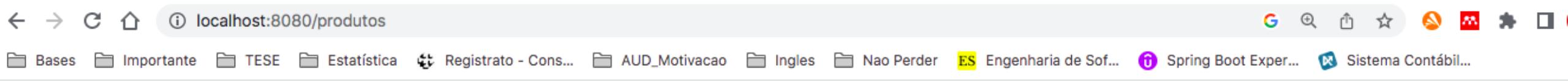
Controller - Produto

```
@RestController
@RequestMapping(value ="/produtos")
public class ProdutoController {
    @Autowired
    private ProdutoRepository repository;

    @GetMapping
    public List<Produto> findAll(){
        List<Produto> result = repository.findAll();
        return result;
    }
    @GetMapping(value = "/{id}")
    public Produto findAll(@PathVariable Long id) {
        return repository.findById(id).get();
    }
    @PostMapping
    public Produto insert(@RequestBody Produto produto) {
        return repository.save(produto);
    }
    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id){
        repository.deleteById(id);
    }
    @PutMapping("/")
    public void update(@RequestBody Produto produto) {
        repository.save(produto);
    }
}
```

Testar - Produto

GET: http://localhost:8080/sisvendaspring/produtos



```
[{"id":1,"nome":"Cloro","valor":4.99,"categoria":{"id":1,"nome":"Limpeza"}}, {"id":2,"nome":"Pão Plustiva","valor":6.99,"categoria":{"id":2,"nome":"Padaria"}}, {"id":3,"nome":"Filé de Tilápia","valor":6.99,"categoria":{"id":2,"nome":"Padaria"}}, {"id":4,"nome":"Cloro","valor":4.99,"categoria":{"id":1,"nome":"Limpeza"}}]
```

Testar – Buscar UM

GET: http://localhost:8080/sisvendaspring/produtos/1

GET ▼ http://localhost:8080/produtos/1 Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (5) Test Results Save Response ▼

Pretty Raw Preview Visualize JSON ▼ ≡ □ Q

```
1 {  
2   "id": 1,  
3   "nome": "Cloro",  
4   "valor": 4.99,  
5   "categoria": {  
6     "id": 1,  
7     "nome": "Limpeza"  
8   }  
9 }
```

Testar – Inserir

GET: http://localhost:8080/sisvendaspring/produtos/

The screenshot shows the Postman application interface. At the top, the URL `http://localhost:8080/produtos/` is entered. Below it, a POST request is selected. The 'Body' tab is active, showing a JSON object representing a product:

```
1 {
2     "id": 0,
3     "nome": "Agua Sanitária",
4     "valor": 4.99,
5     "categoria": {
6         "id": 1,
7         "nome": "Limpeza"
8     }
9 }
```

Below the request, the response is displayed. The status is 200 OK, and the response body is identical to the one sent:

```
1 {
2     "id": 5,
3     "nome": "Agua Sanitária",
4     "valor": 4.99,
5     "categoria": {
6         "id": 1,
7         "nome": "Limpeza"
8     }
9 }
```

Springdoc OpenAPI



- Suporta o **padrão OpenAPI 3.0** (antigo Swagger Specification);
- Gera a especificação (documentação)automaticamente de toda APIs REST;
- Gera uma **interface web interativa** para testar endpoints, por exemplo:
 - <http://localhost:8080/sisvendaspring/swagger-ui/index.html>
- Por meio da interface WEB é possível acessar e manipular todas as rodas da API;

Ex.: pom.xml

```
<dependency>
<groupId>org.springdoc</groupId>
<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
<version>2.5.0</version>
</dependency>
```

Springdoc OpenAPI



- Exemplo do uso de comentários;

```
import io.swagger.v3.oas.annotations.tags.Tag;  
import org.springframework.web.bind.annotation.*;
```

```
@RestController  
@RequestMapping("/api/usuarios")  
@Tag(name = "Usuários", description = "Operações relacionadas a usuários")  
public class UsuarioController {  
    @GetMapping("/{id}")  
    @Operation(summary = "Buscar usuário por ID")  
    public Usuario getUsuario(@PathVariable Long id) {  
        return new Usuario(id, "João");  
    }  
}
```

produto-controller

PUT /produtos/

GET /produtos

Parameters

Try it out

GET /produtos

Parameters

No parameters

Execute

Responses

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:8080/sisvendaspring/produtos' \
-H 'accept: */*'
```



Request URL

```
http://localhost:8080/sisvendaspring/produtos
```

Server response

Code Details

200

Response body

```
[  
  {  
    "id": 2,  
    "nome": "Cloro",  
    "valor": 4.99,  
    "categoria": {  
      "id": 1,  
      "nome": "Limpeza"  
    }  
  },  
  {  
    "id": 3,  
    "nome": "Vassoura",  
    "valor": 1.99,  
    "categoria": {  
      "id": 2,  
      "nome": "Limpeza"  
    }  
  }]
```

POST

/categorias



Parameters

No parameters

Cancel

Reset

Request body required

application/json



```
{  
  "nome": "Açougue"  
}
```

Execute

Clear

Responses

Curl

```
curl -X 'POST' \
'http://localhost:8080/sisvendaspring/categorias' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
  "nome": "Açougue"
}'
```



Request URL

<http://localhost:8080/sisvendaspring/categorias>

Server response

Code	Details
------	---------

200

Response body

```
{  
  "id": 5,  
  "nome": "Açougue"  
}
```



[Download](#)

Response headers

```
connection: keep-alive  
content-type: application/json  
date: Mon, 30 Jun 2025 09:19:47 GMT  
keep-alive: timeout=20  
transfer-encoding: chunked
```