Spring

Conceitos básicos para o desenvolvimento de APIs


BACK-END DEVELOPER
FRONT-END DEVELOPER
BACK-END
REST API
FRONT-END


BACK-END
FRONT-END
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

Apache Maven Project
http://maven.apache.org/

Maven™

MySQL

Novo Projeto

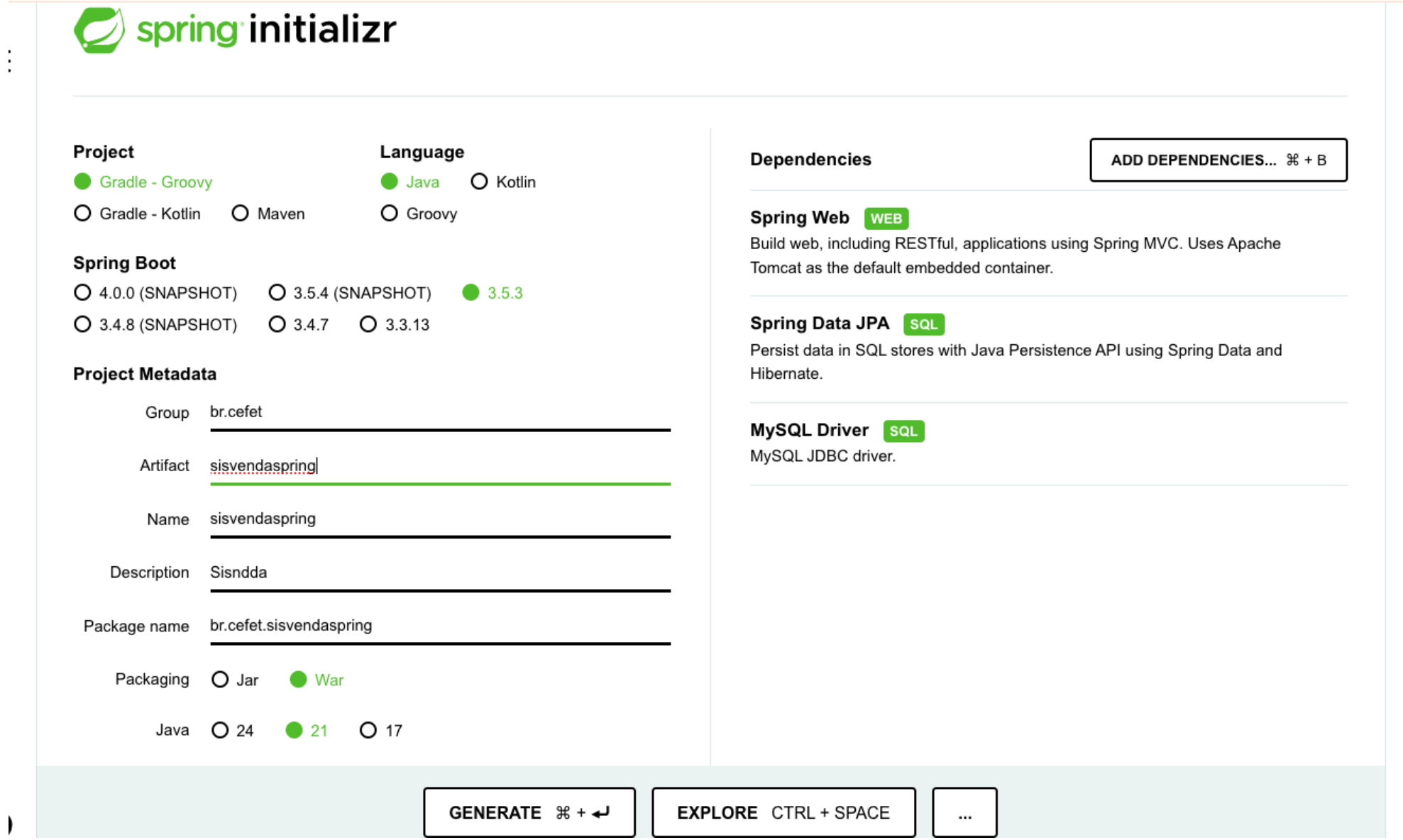
Passos

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

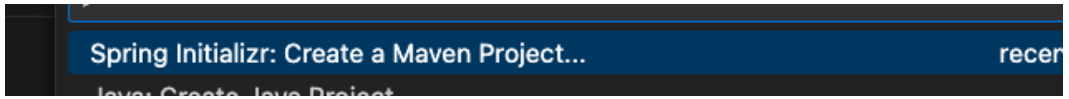
- CTRL + Shif + P;
- **Create a Initializr: Create a Maven Project.**

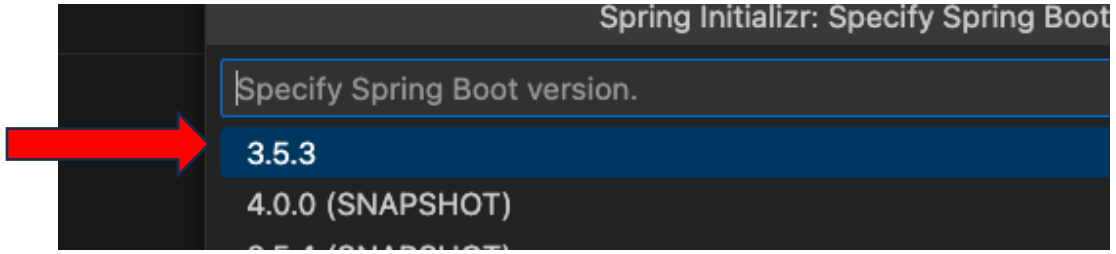
Novo Projeto - Spring Initializr (https://start.spring.io/)

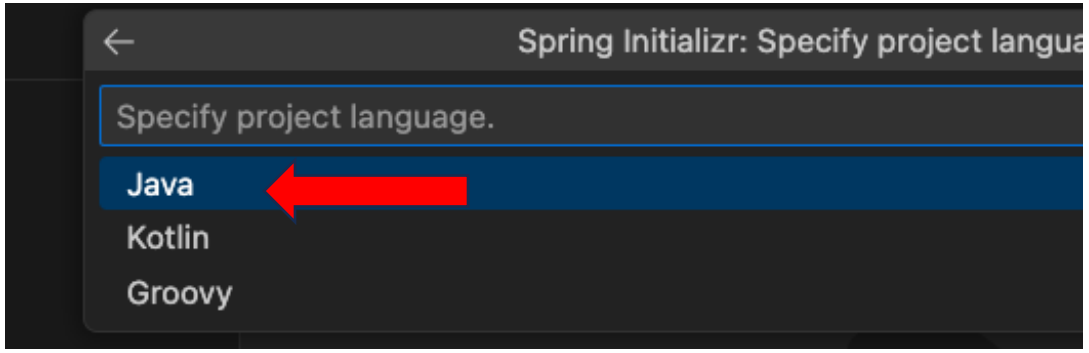


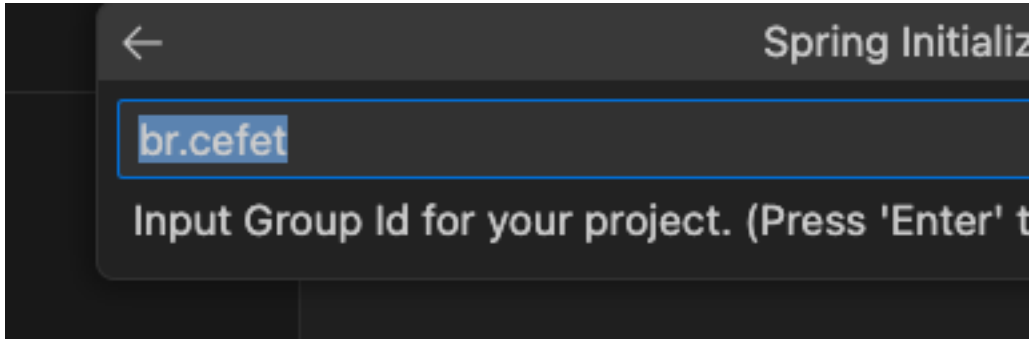
Novo Projeto

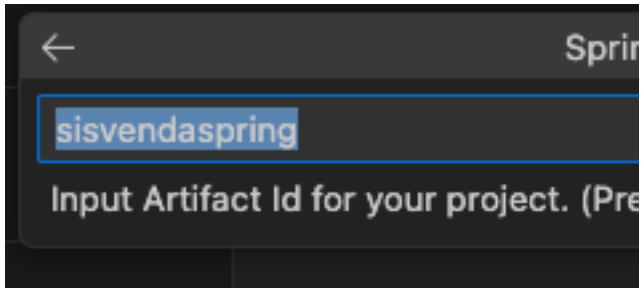
- CTRL + Shif + P
- **Create a Initializr: Create a Maven Project**

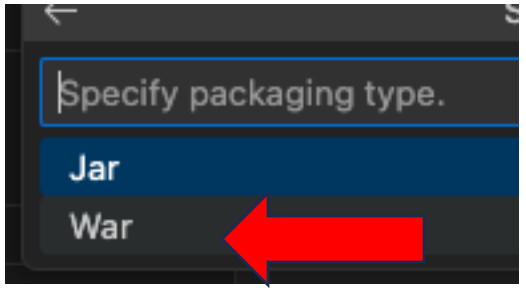


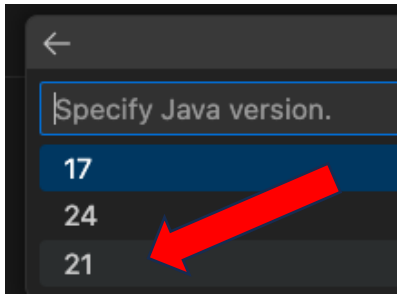




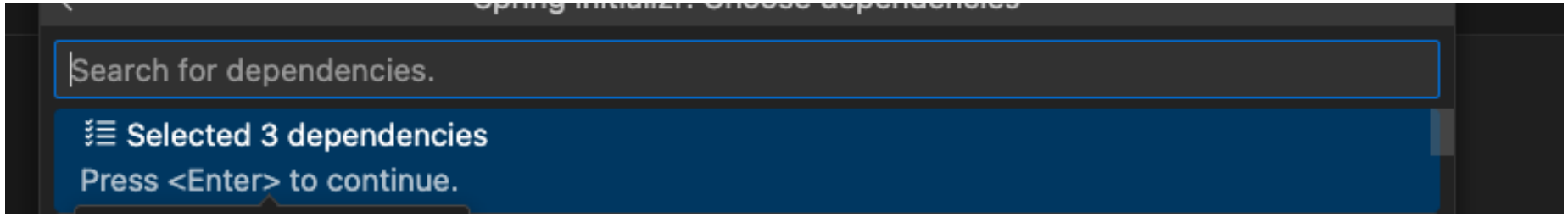


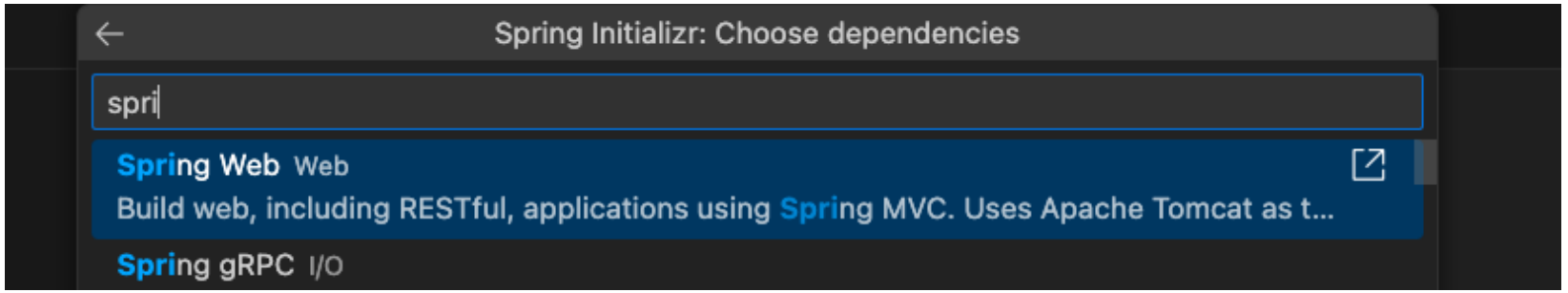


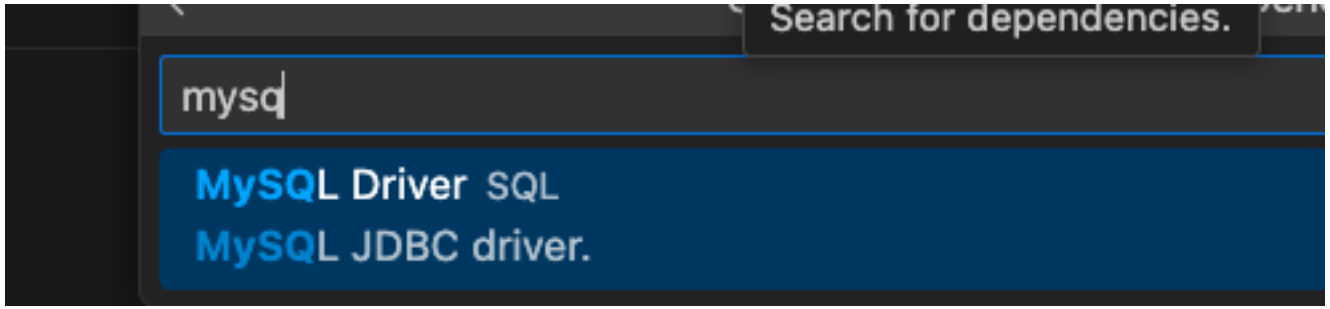


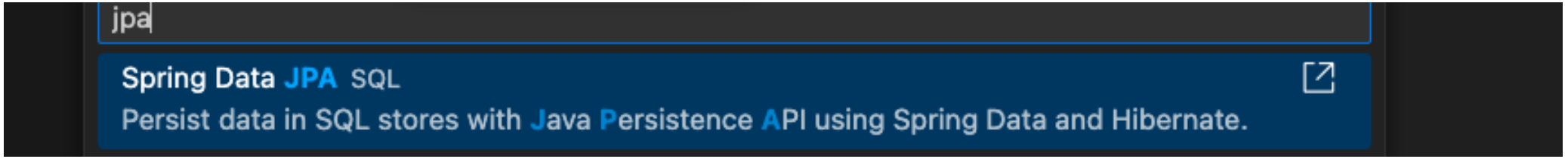


Novo Projeto

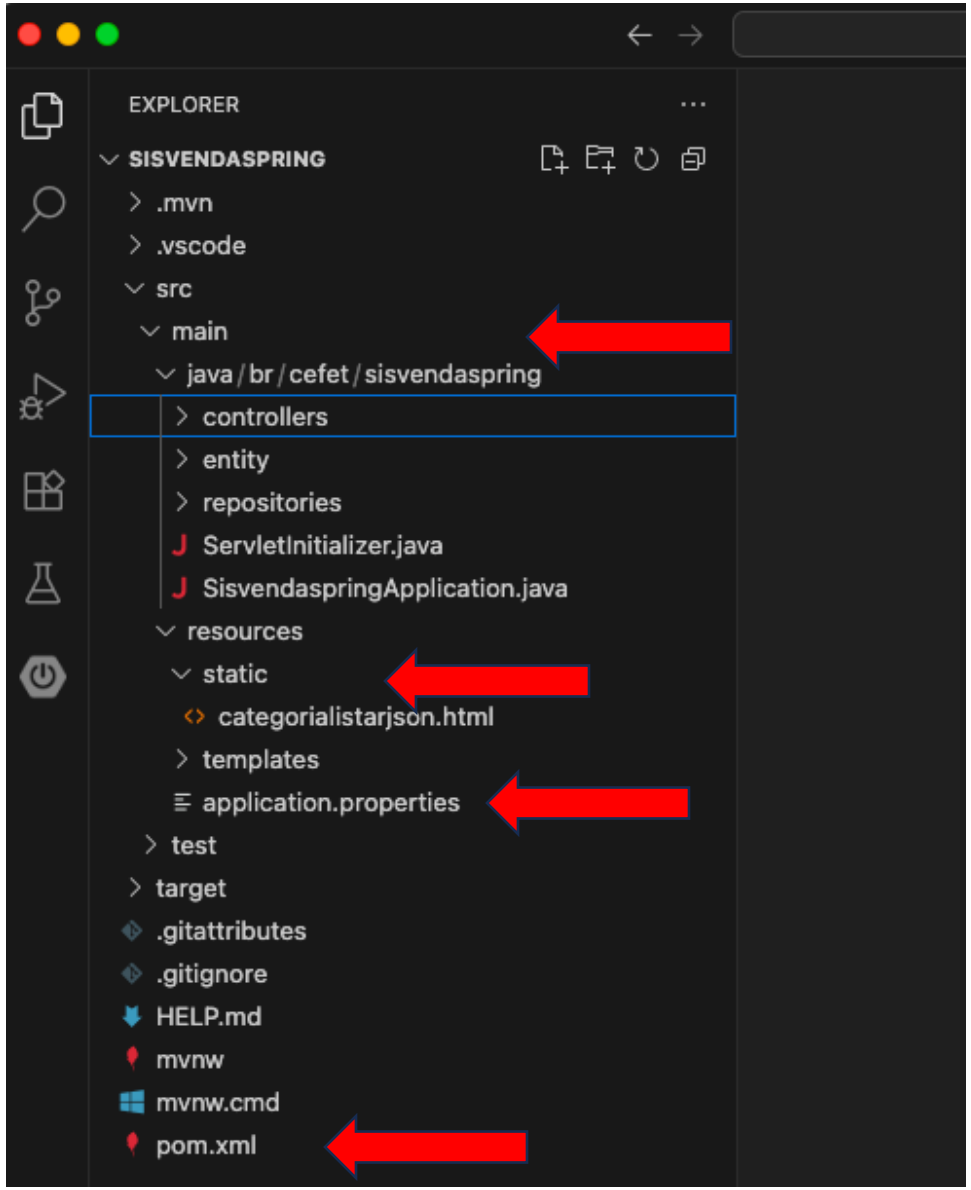








Ambiente - VSCode



Principais Subdiretórios e arquivos

- **main:** Código Java
  JavaBeans
  Controllers

- **Resource / Static:**
  Páginas WEB (html+css+js)

- **Resource / aplication.Properties**
  Arquivo de configuração ao Banco de dados

- **Pom.xmml**
  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 contêm todas as dependências utilizadas no projeto.

Maven

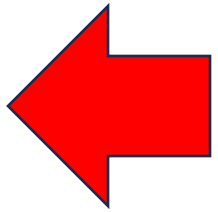
Arquivo pom.xml (Exemplo)

```
<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)

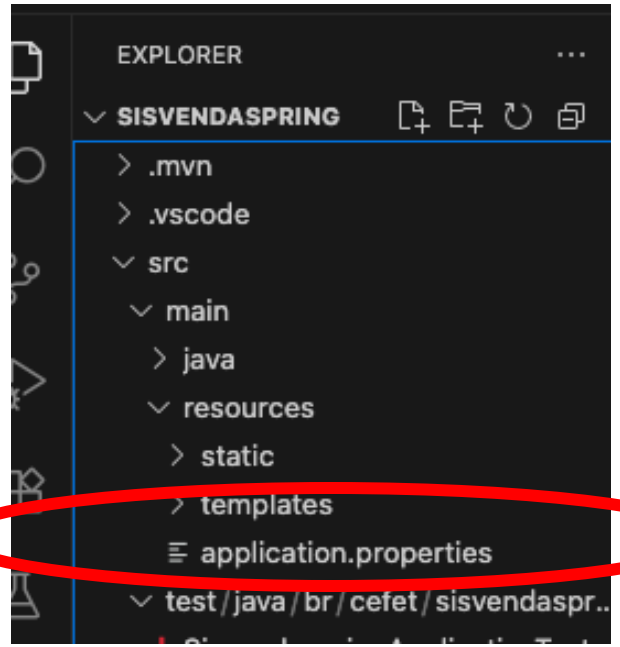
```
<build>
    <finalName>sisvendaspring</finalName>
    <plugins>
        <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
    </build>
```



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

Configurando acesso ao BD

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

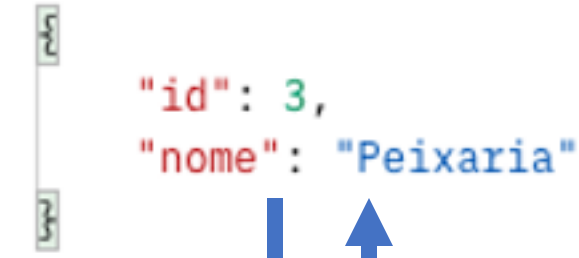



```
spring.application.name=sisvendaspring
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/dbspring
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.open-in-view:false
spring.jpa.show-sql: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

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

  CREATE DATABASE dbspring
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;

Arquitetura da API (MVC)



Request

Response

Controller

Repository

Entity

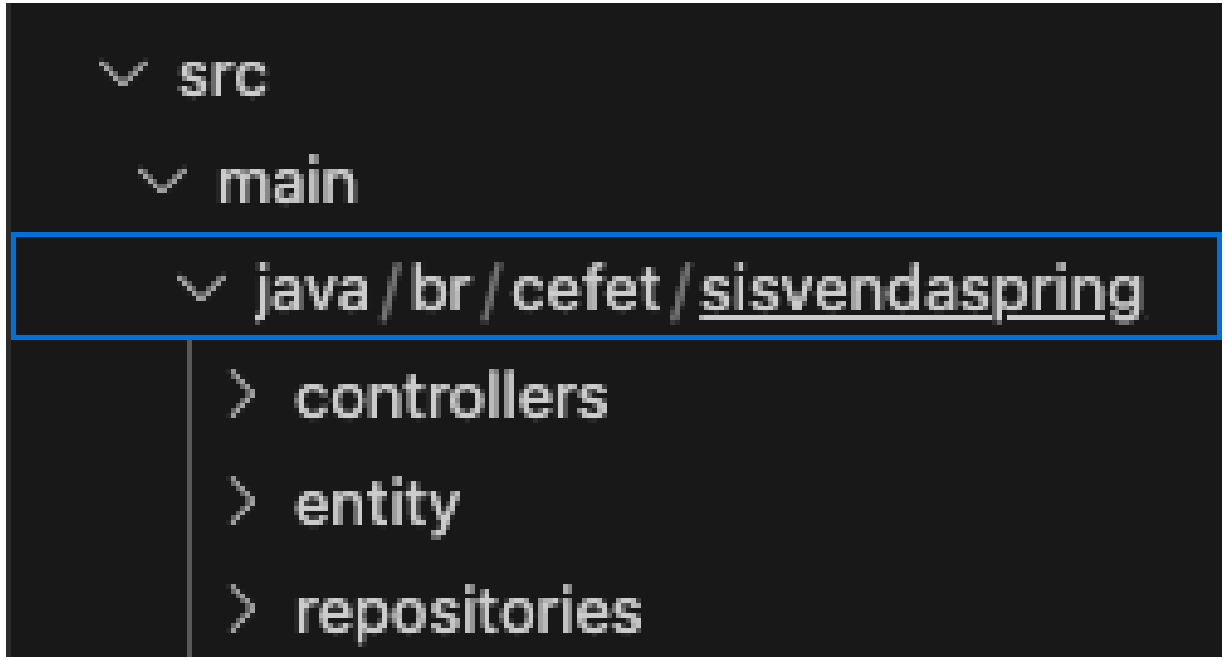
BD

RESTFul Interface

- GET
- POST

PUT

VSCode – Criar os pacotes



- **Entity**
  - Receberá as classes de modelo;
- **Respository**
  - 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)

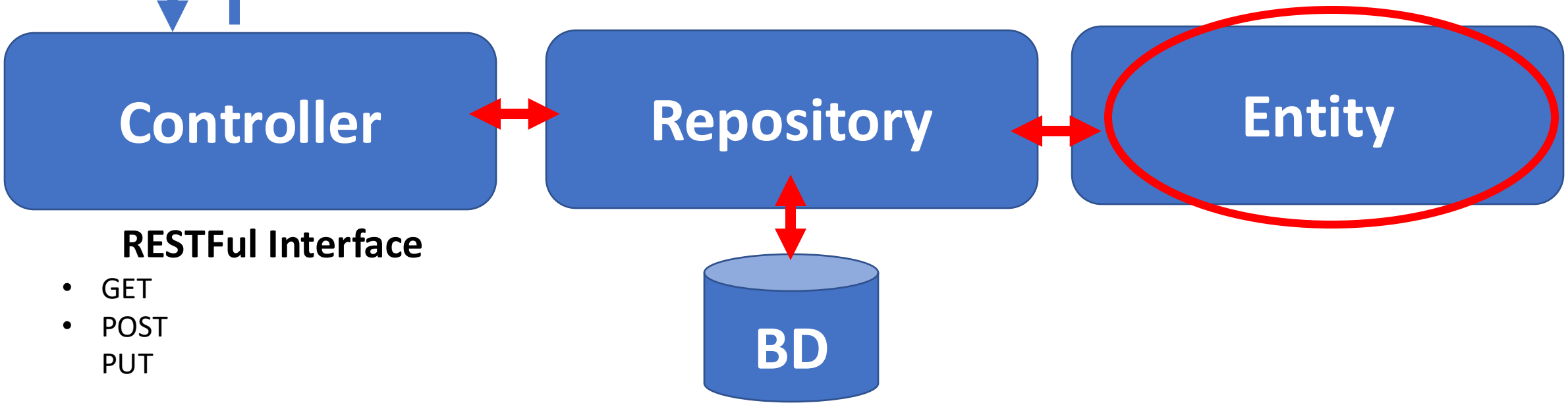
```
"id": 3,
"nome": "Peixaria"
```

Request

Response

Entity

- Classe do domínio que será mapeado para uma entidade (tabela).



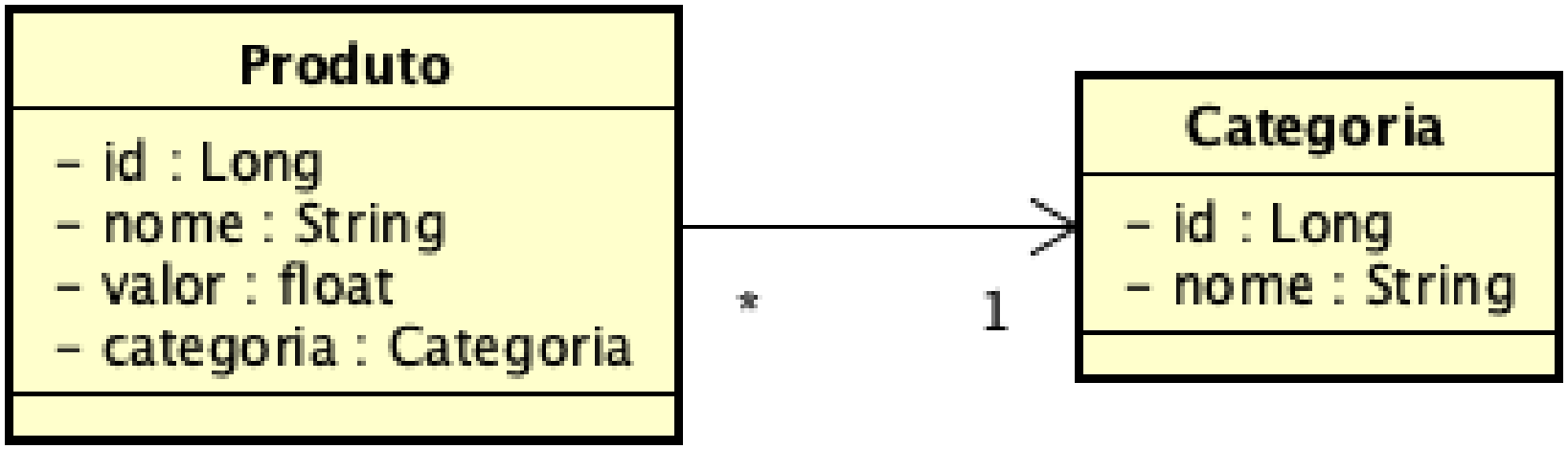
RESTFul Interface

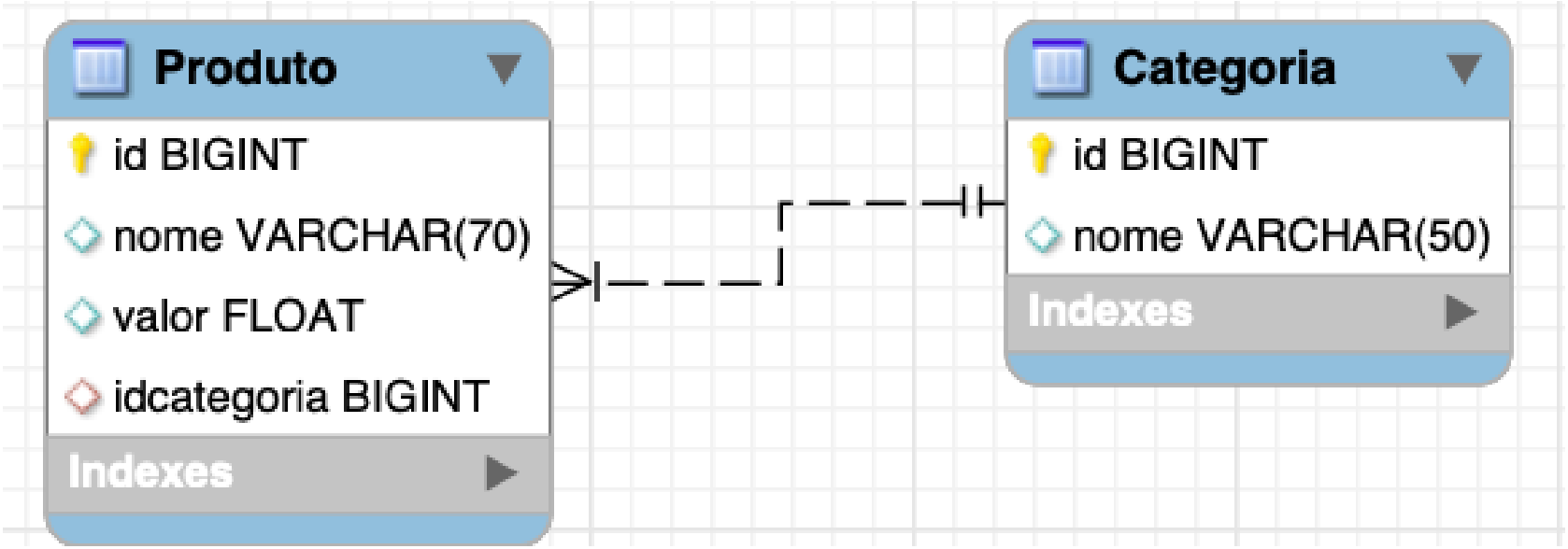
- GET
- POST
  PUT

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

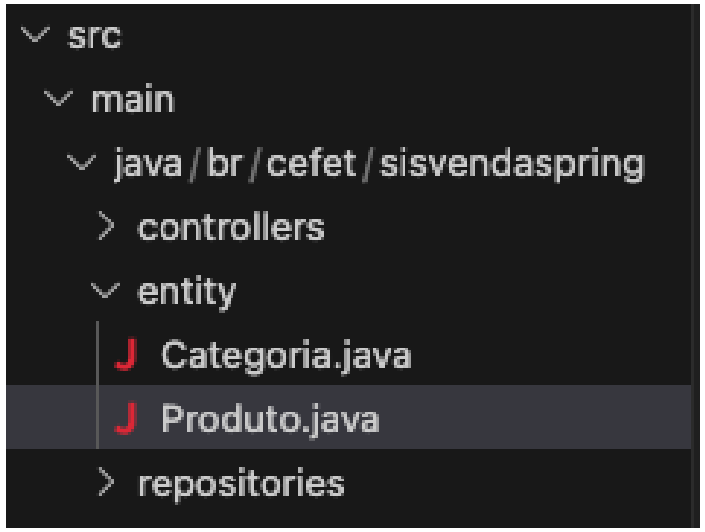
```
@Entity
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String nome;
    private float valor;

    \\Indica o relacionamento entre entidades
    @ManyToOne
    @JoinColumn(name = "idcategoria")
    private Categoria categoria;
```

@OneToOne
@ManyToOne
@OneToMany
@ManyToMany

Spring Data JPA - Exemplo

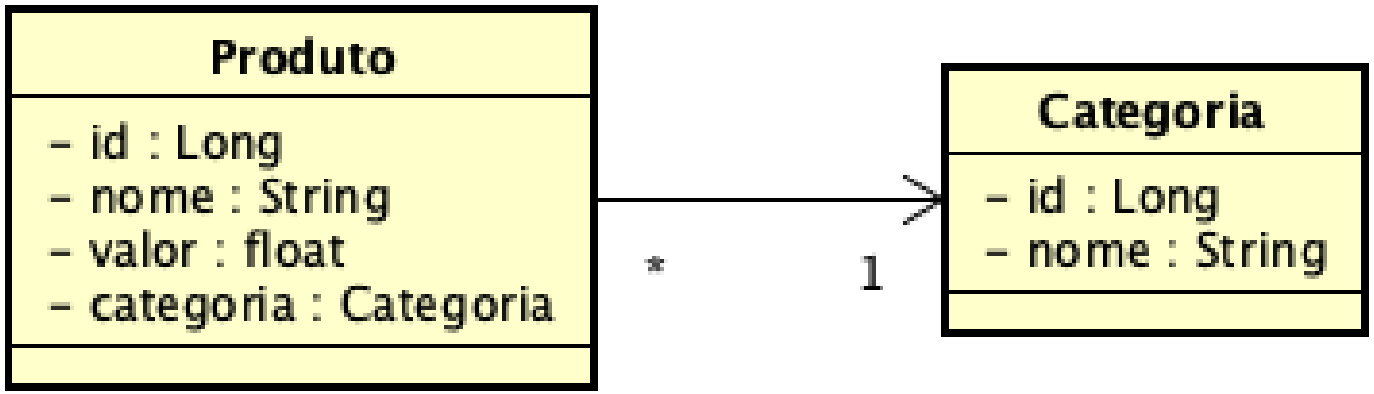
```
@Entity
@Table(name = ”usu") \\ Quando o nome da tabela for diferente
public class Usuario {

\\ 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




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

public class Categoria {

public Long id;
public String nome;
.....
```

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

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



SpringToolSuite – Mapeamento Objeto Relacional - Categoria

```
package br.cefet.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

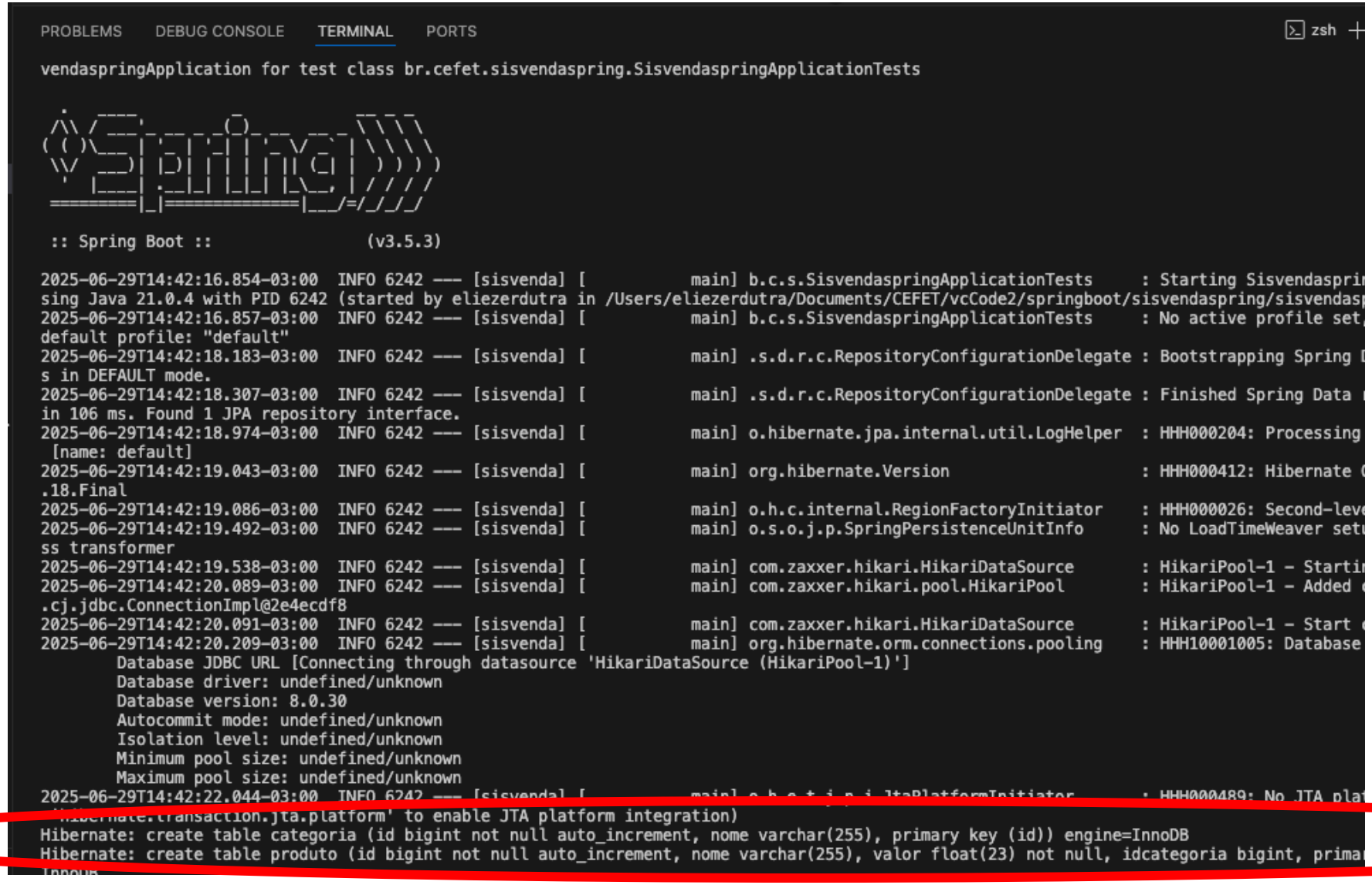
No terminal,

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



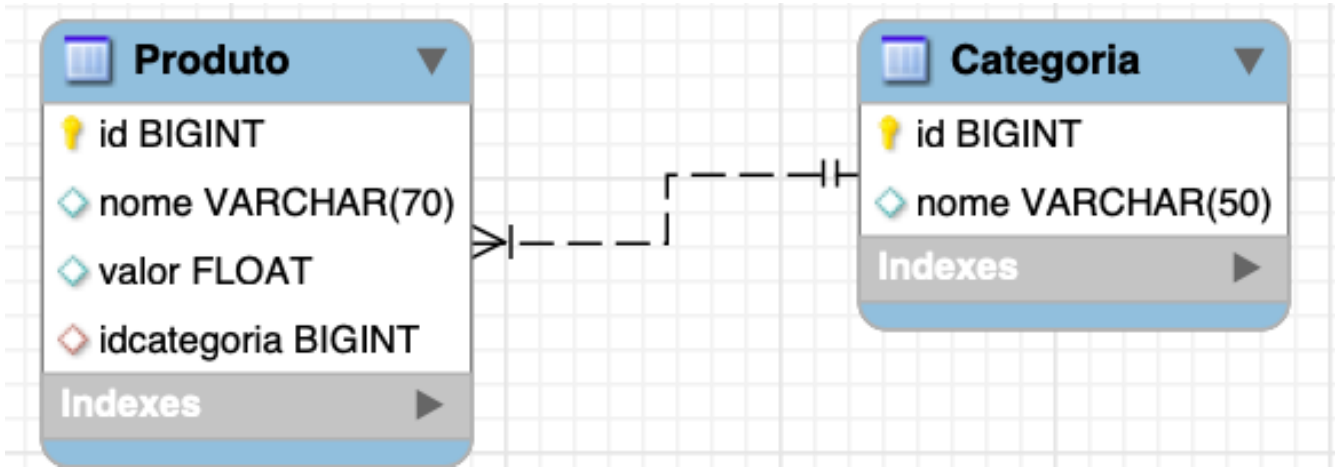
Compile o projeto

mvn package

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

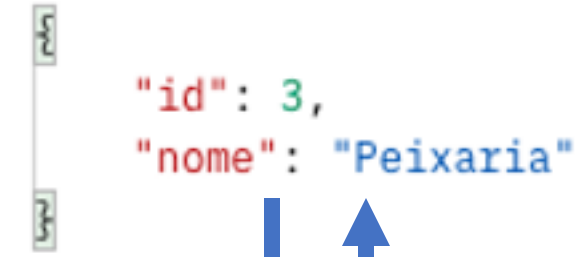
MySQL – Criar o banco de dados – Código completo

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)

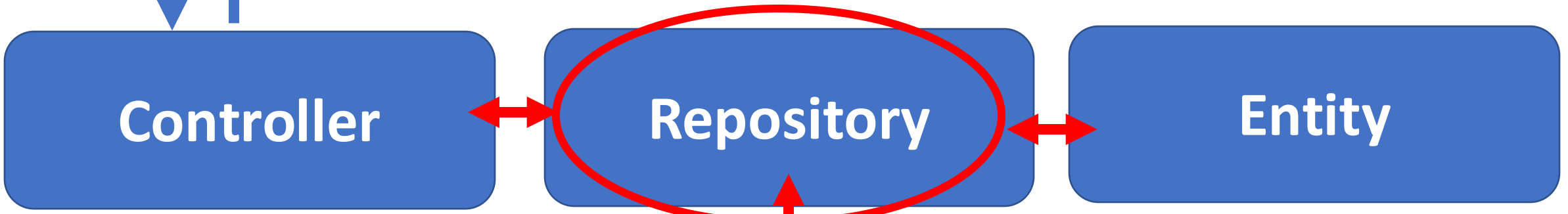


Request

Response

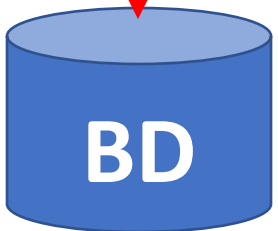
Repository (*"DAO"*)

- Acesso ao BD;
- CRUD;
- Consultas Personalizadas e
- É necessário criar um repository para cada Classe Entity.

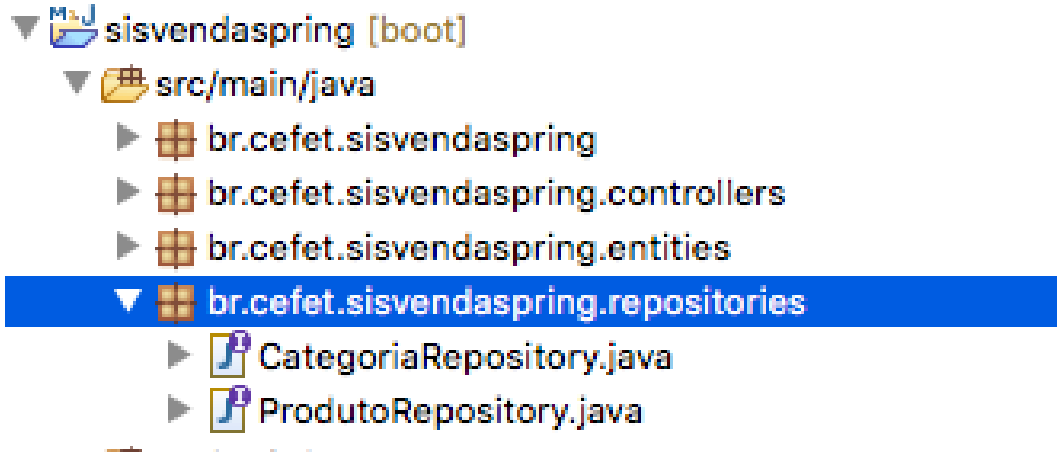


RESTFul Interface

- GET
- POST
  PUT

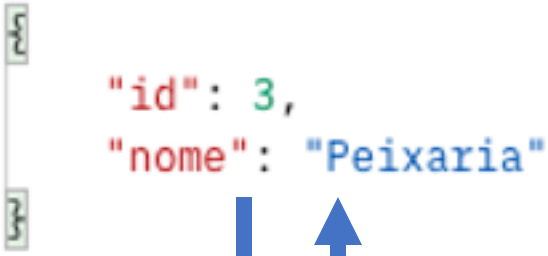


Repository - Categoria



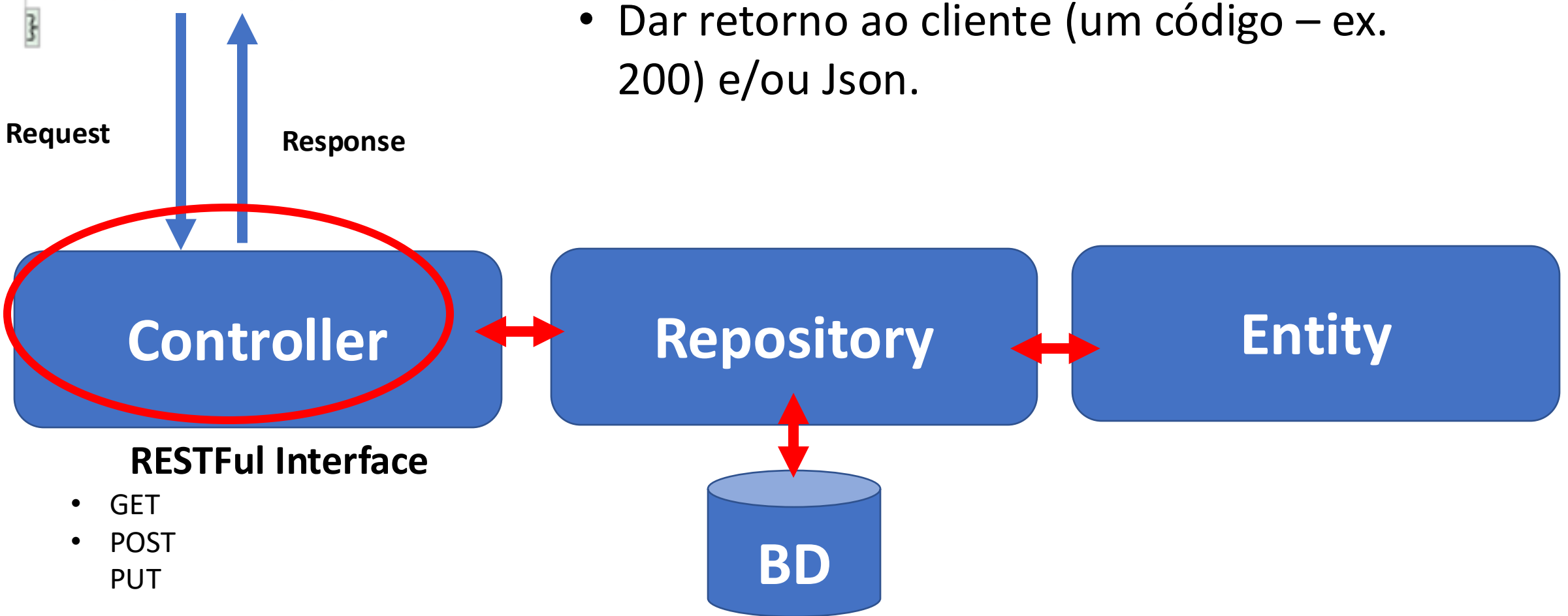

```
package br.cefet.sisvendaspring.repositories;
import org.springframework.data.jpa.repository.JpaRepository;
import br.cefet.sisvendaspring.entities.Categoria;


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

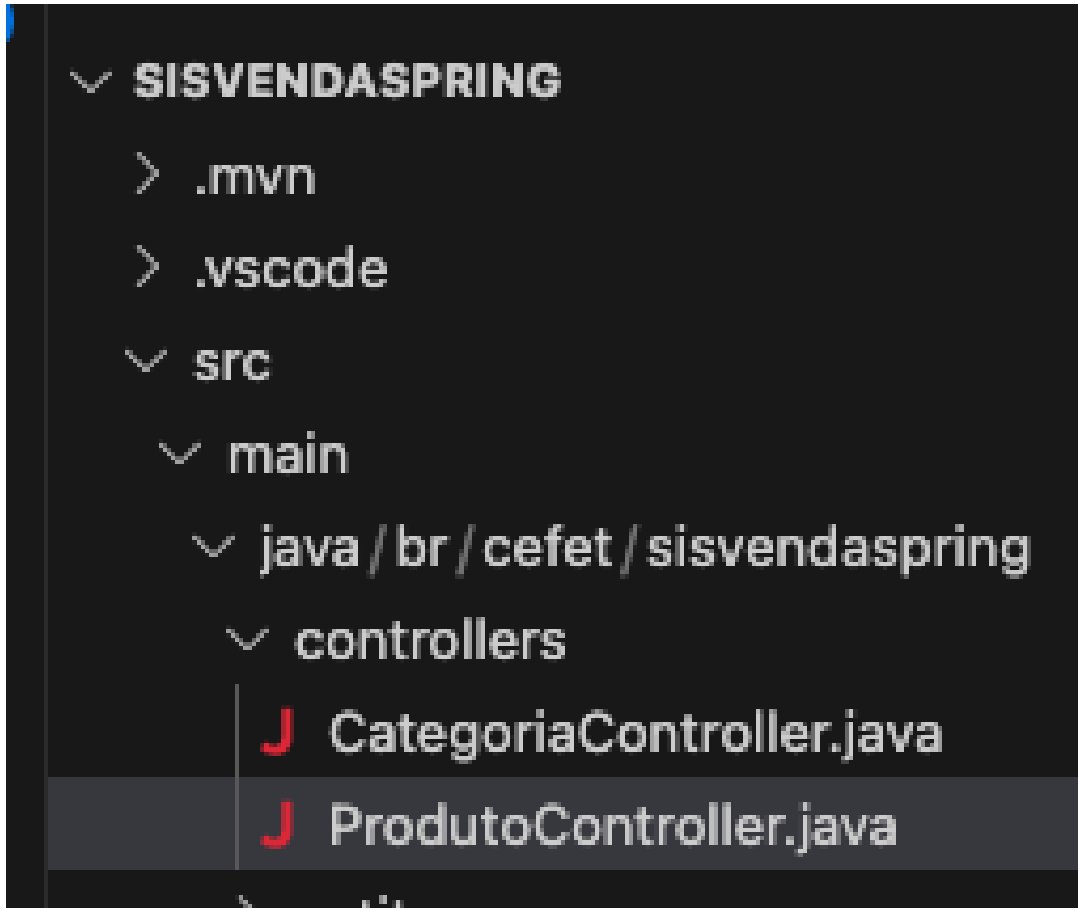
VERBO HTTP	URL (RESTFul)	OPERAÇÃO
GET	sisvendaspring/categorias	Lista todas as categorias
GET	sisvendaspring/categorias/3	Retorna uma categoria
DELETE	sisvendaspring/categorias/3	Exclui uma categoria
POST	sisvendaspring/categorias/	Insere uma categoria
PUT	sisvendaspring/categorias/	Atualiza uma categoria


```
[
    {
        "id": 1,
        "nome": "Limpeza"
    },
    {
        "id": 2,
        "nome": "Padaria"
    },
    {
        "id": 4,
        "nome": "Brinquedo de Adulto"
    }
]
```

```
{
    "id": 4,
    "nome": "Brinquedo de Adulto"
}
```


Exemplos JSON

Controller


SISVENDASPRING
.mvn
.vscode
src
main
java / br / cefet / sisvendaspring
controllers
CategoriaController.java
ProdutoController.java

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();
}
```

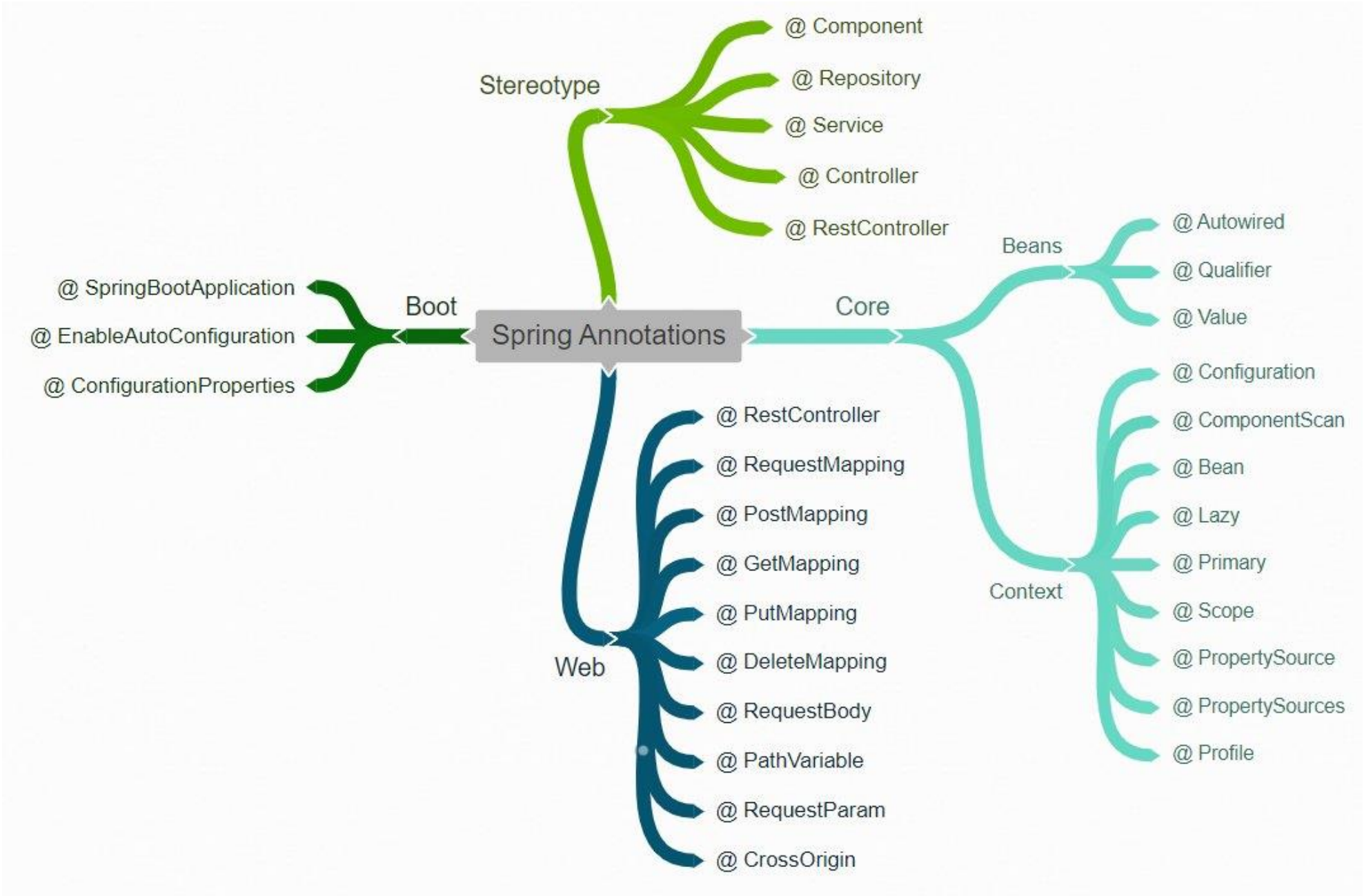
Controller - Categorias

```
@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

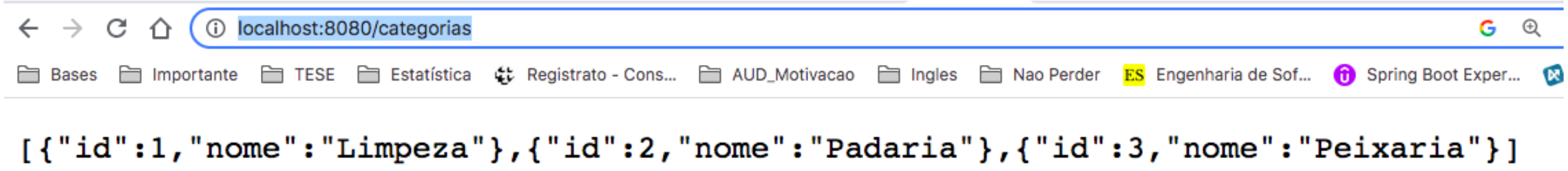
```
@RestController
@RequestMapping(value = "/categorias")
public class CategoriaController {
	@Autowired
	private CategoriaRepository repository;

	@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

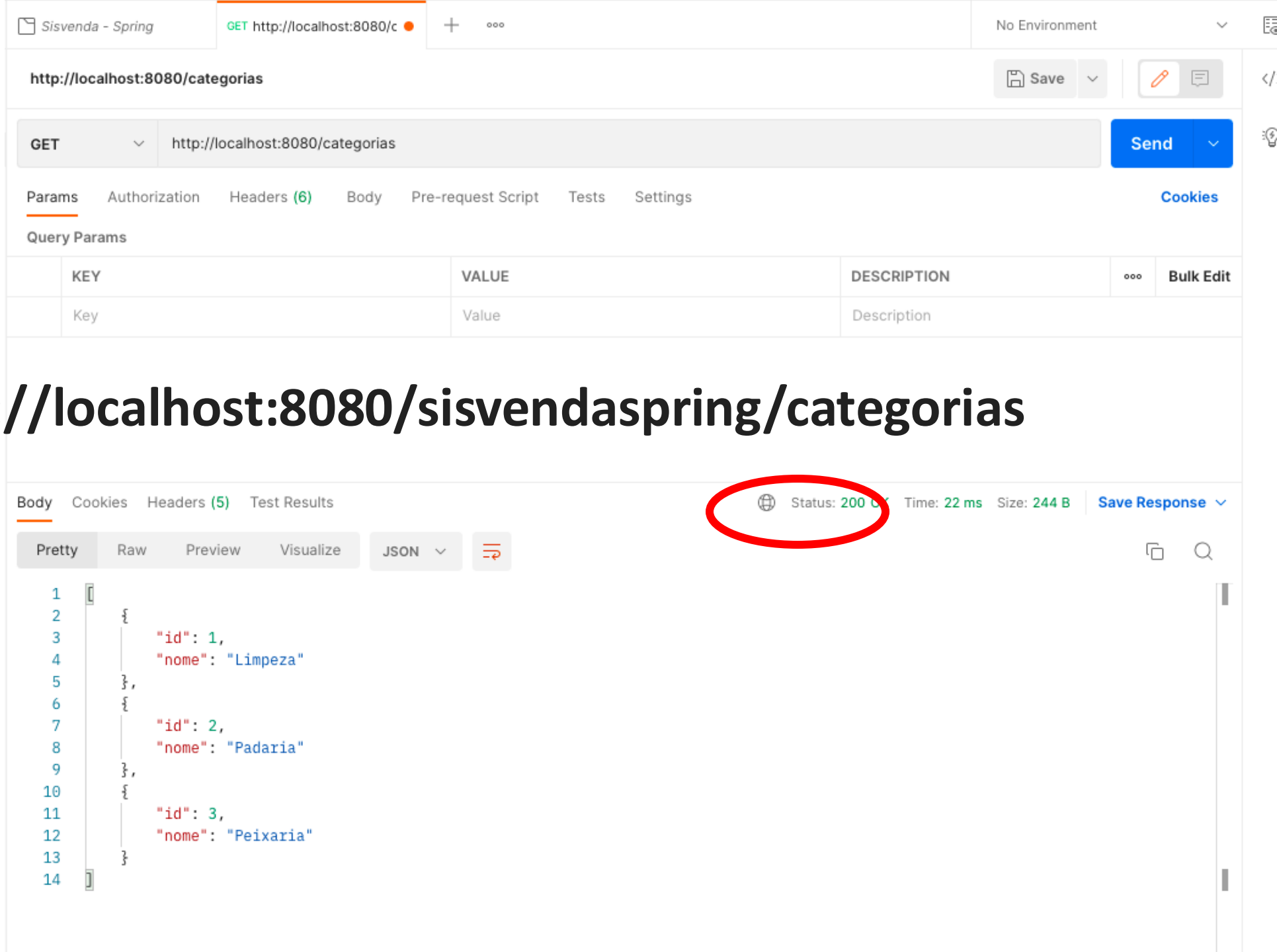


Testar - CATEGORIA

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

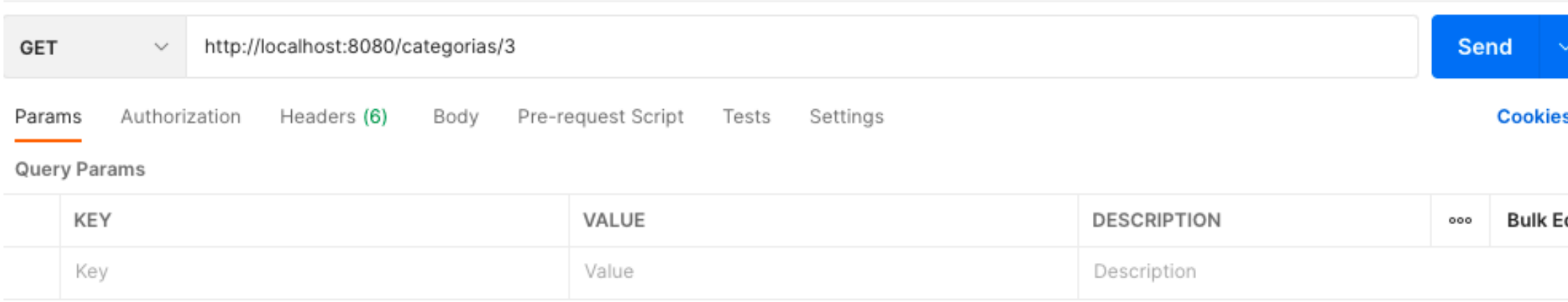


Testar
Listar Todas

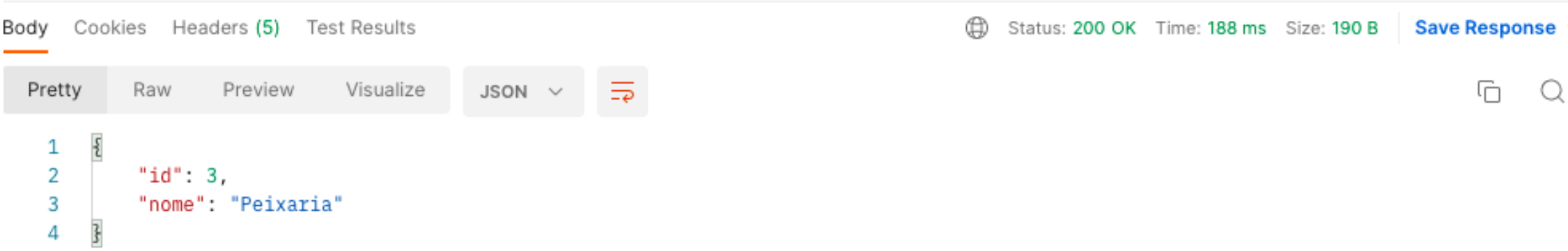


GET: http://localhost:8080/sisvendaspring/categorias

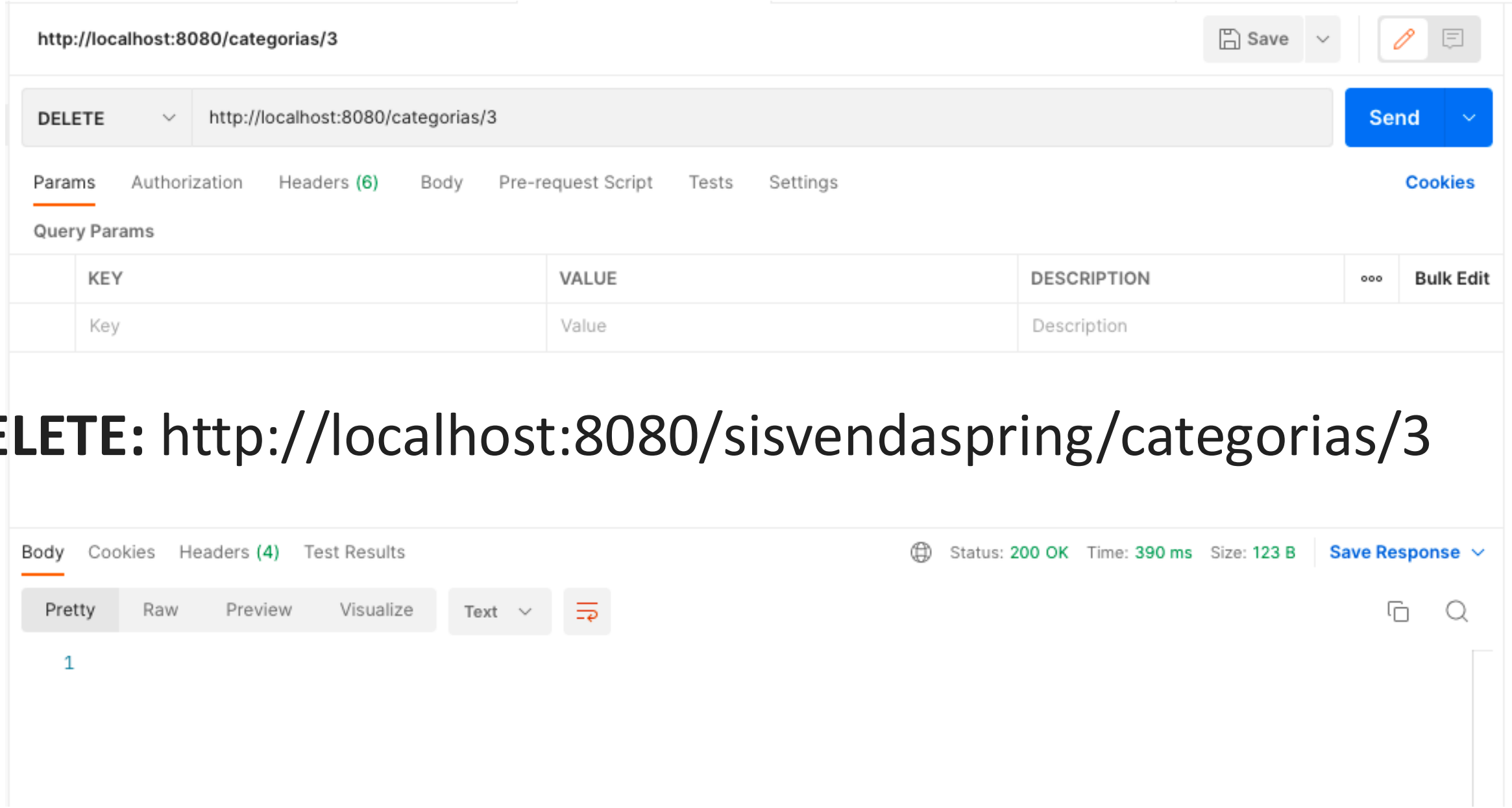
Testar – GET {ID}



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

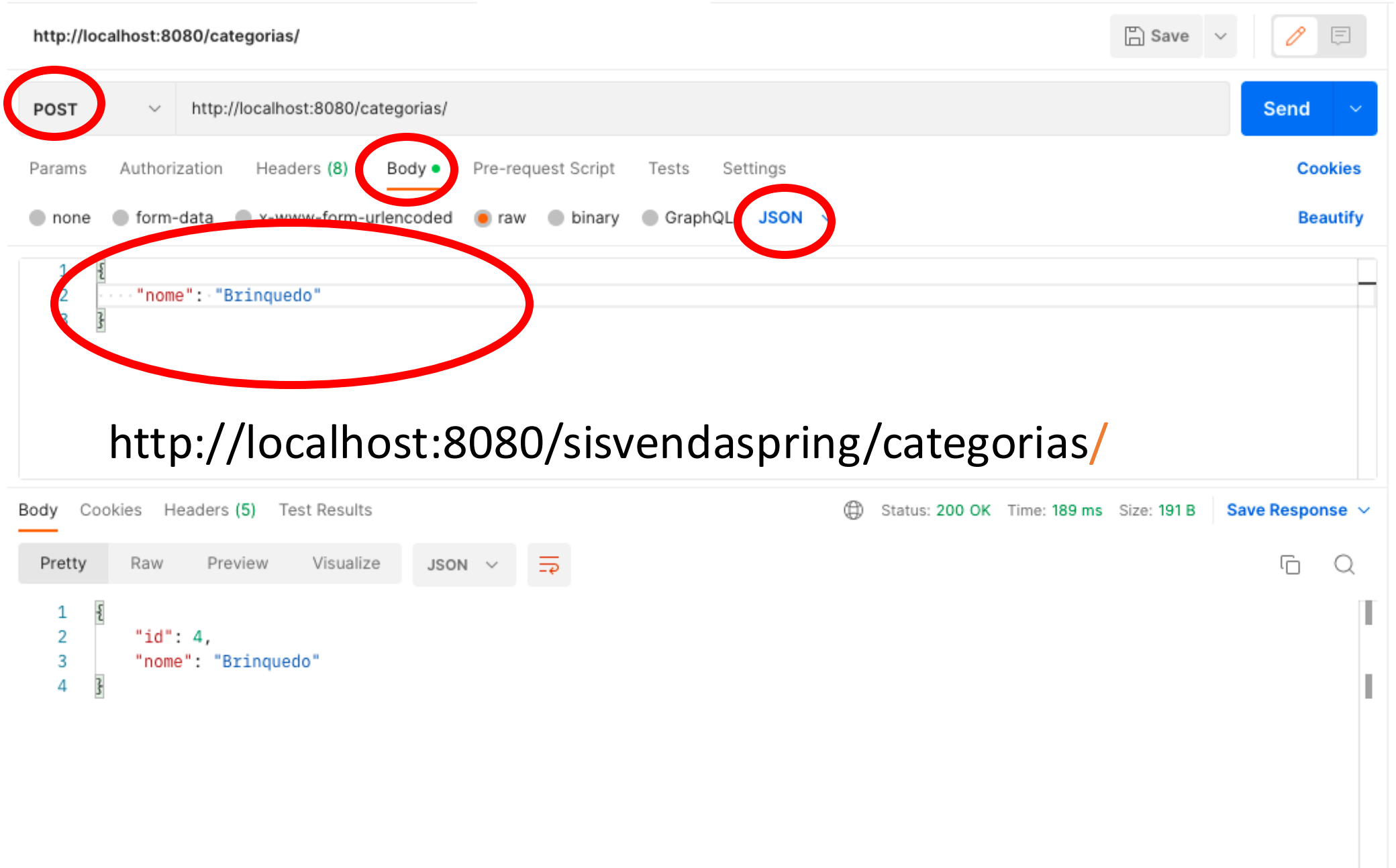


Testar - DELETE

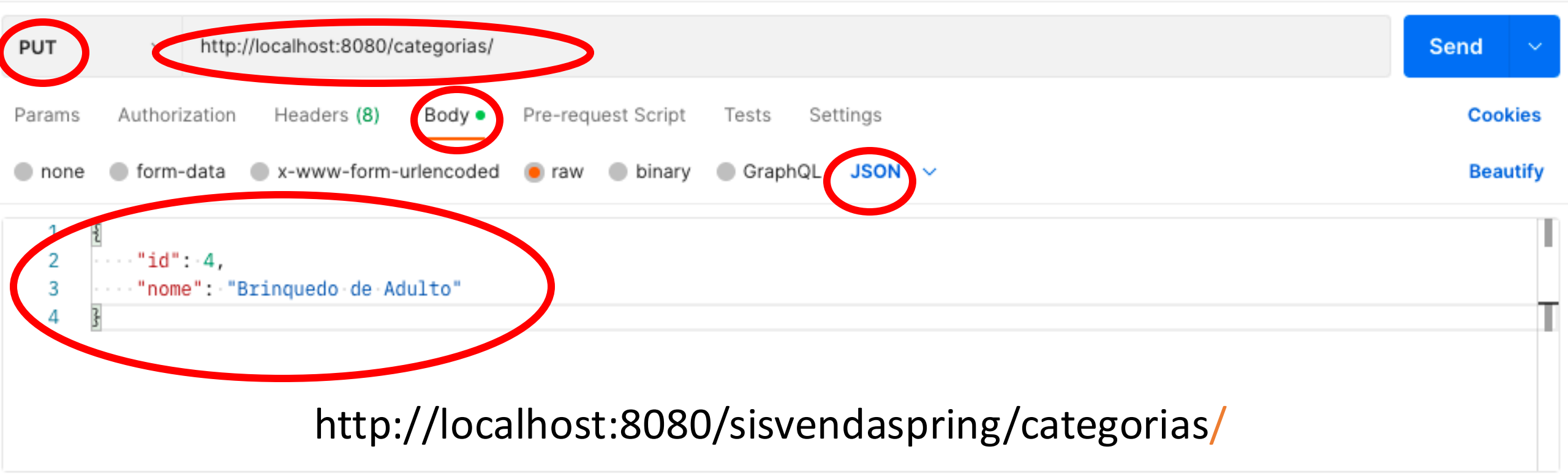


DELETE: http://localhost:8080/sisvendaspring/categorias/3

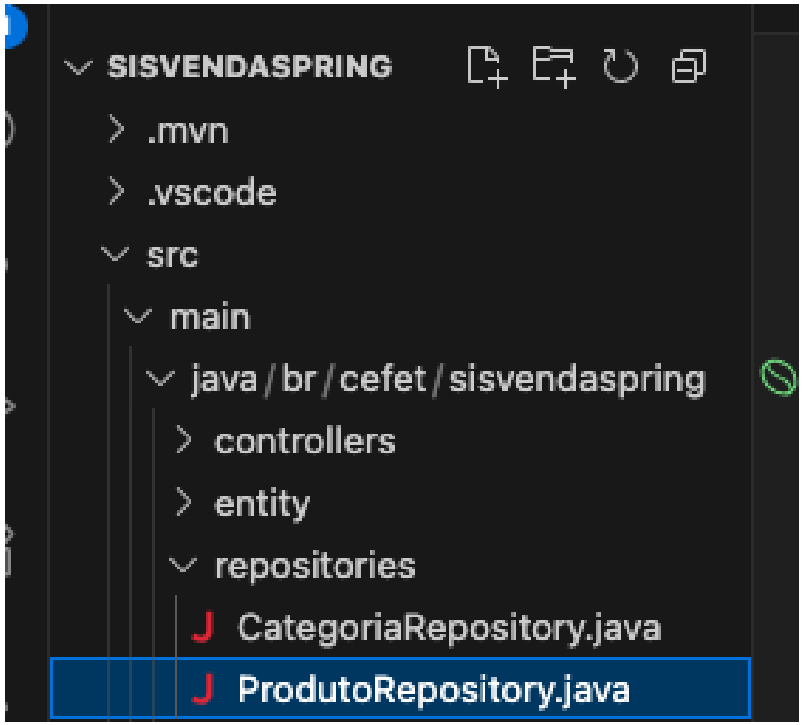
Testar - Inserir



Testar - Atualizar



Repository - Produto



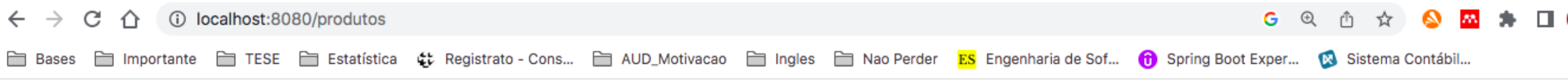

```
package br.cefet.sisvendaspring.repositories;
import br.cefet.sisvendaspring.entity.Produto;
import org.springframework.data.jpa.repository.JpaRepository;

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

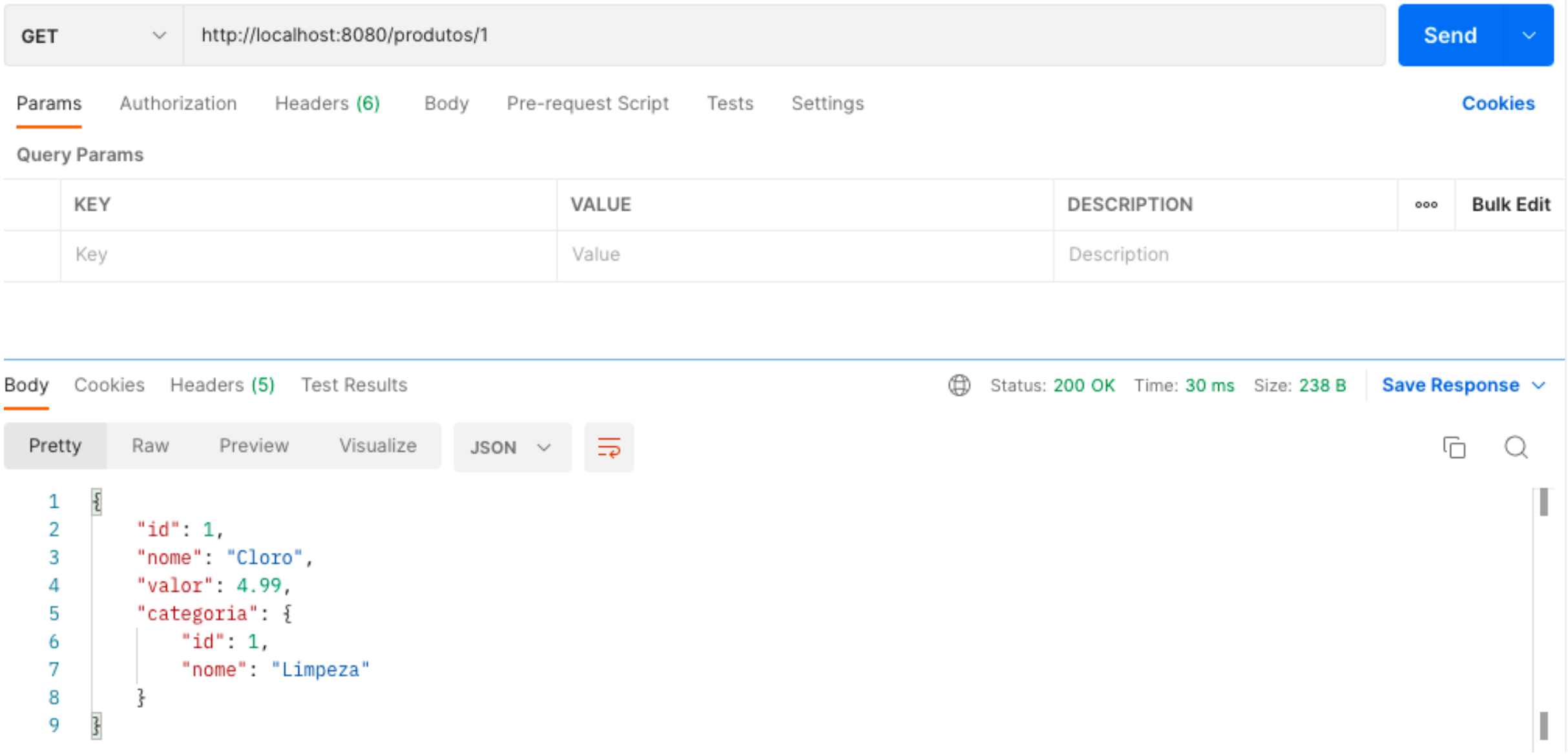
GET: http://localhost:8080/sisvendaspring/produtos




```
[{"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"}}]
```

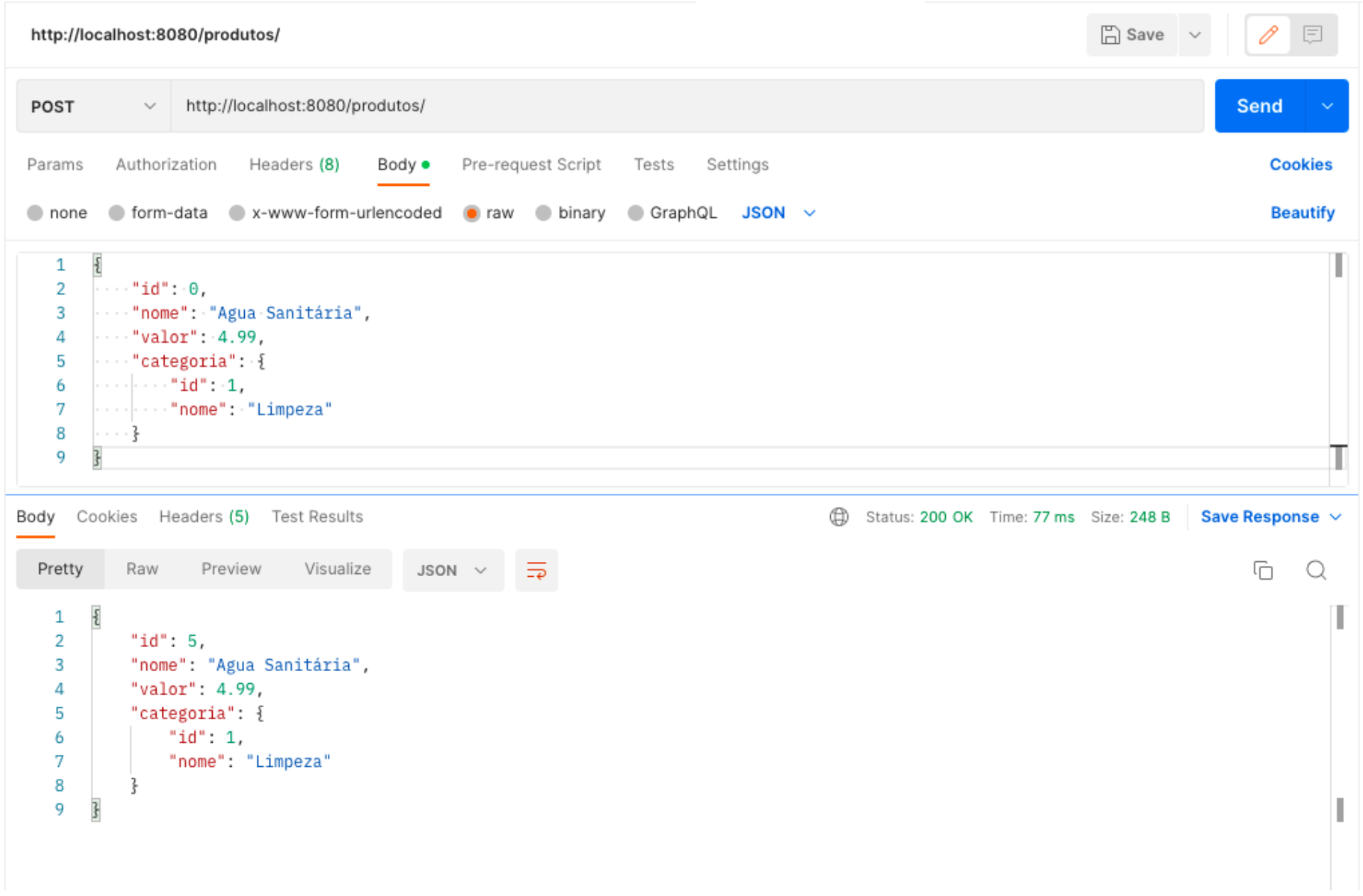
Testar – Buscar UM

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



Testar – Inserir

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



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 | Cancel

No parameters

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,
```

POST /categorias

Parameters

Cancel Reset

No parameters

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
```

Responses