

Spring

Conceitos básicos para o desenvolvimento de APIs





BACK-END



FRONT-END



APIs

Ecosystem 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;

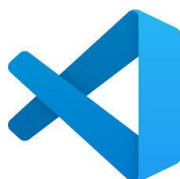
Spring Data JPA: Fornece suporte ao **acesso a dados relacionais (OO x Tabelas)** usando a API Java Persistence (JPA). Ele simplifica o desenvolvimento de camadas de acesso a dados, abstraindo grande parte do código repetitivo necessário para realizar operações comuns de persistência. Oferece recursos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



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/>)



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

Artifact

Name

Description

Package name

Packaging ☐ Jar ☒ War

Java ☐ 24 ☒ 21 ☐ 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.

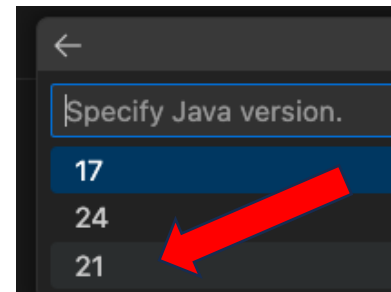
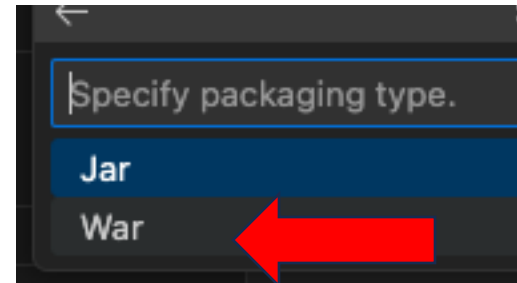
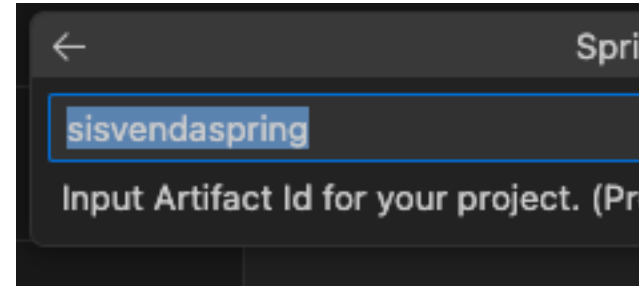
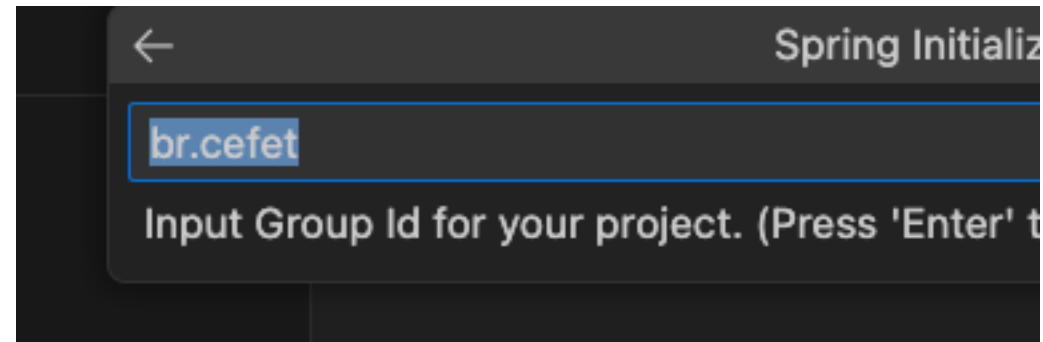
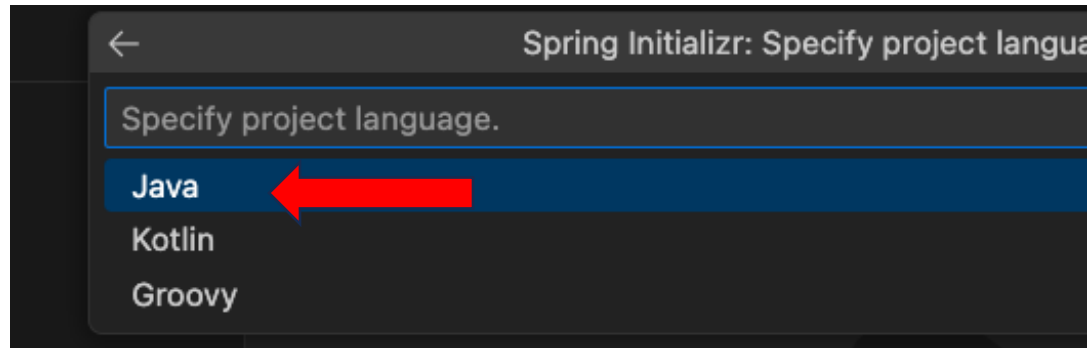
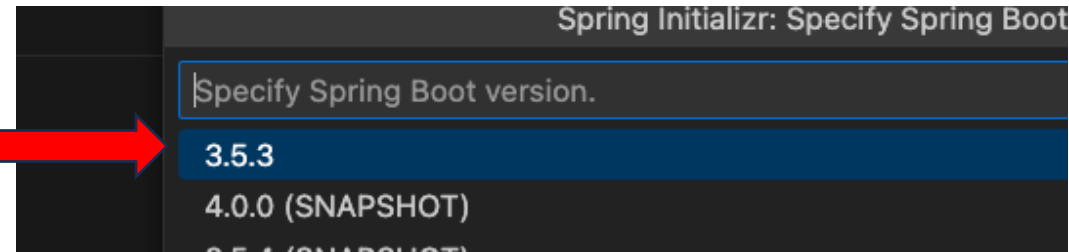
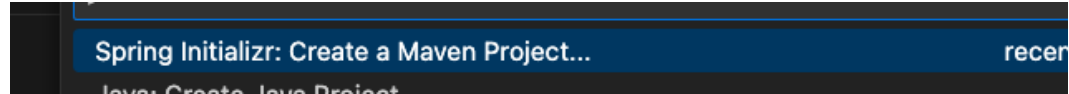
GENERATE ⌘ + ↵

EXPLORE CTRL + SPACE

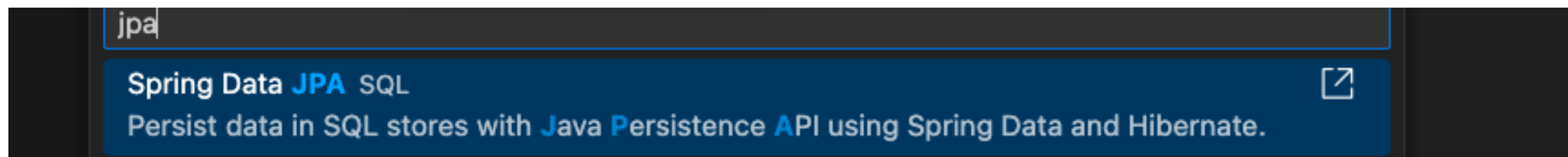
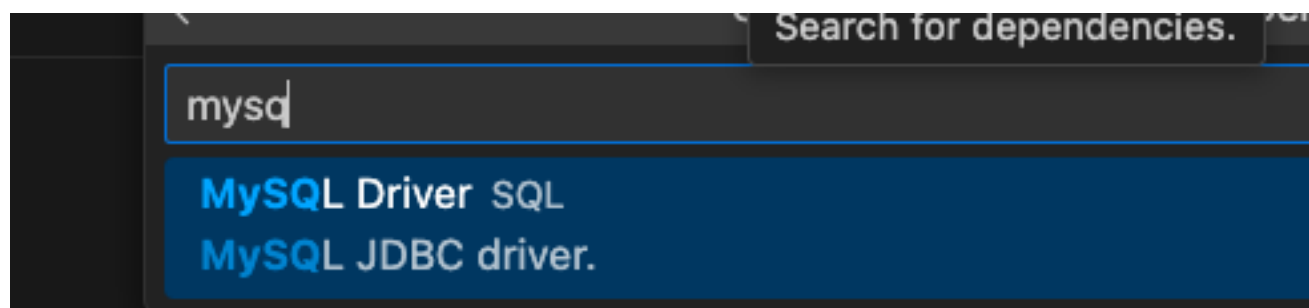
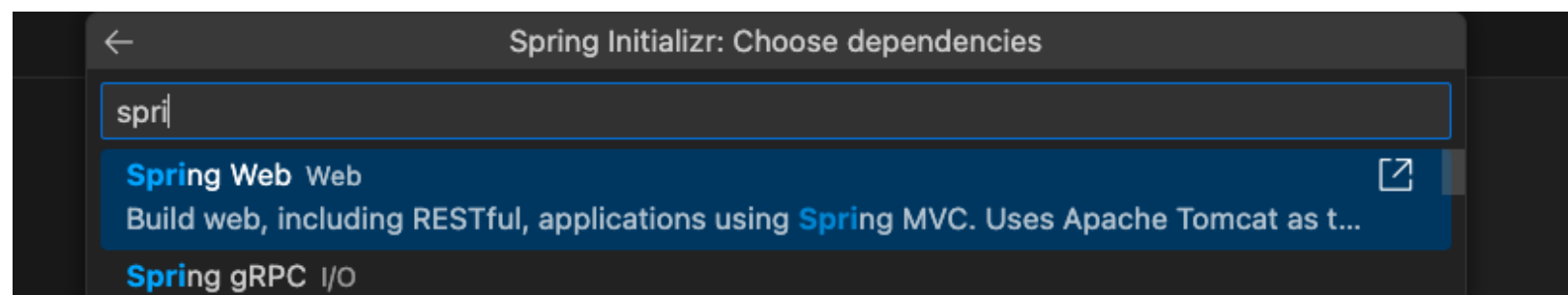
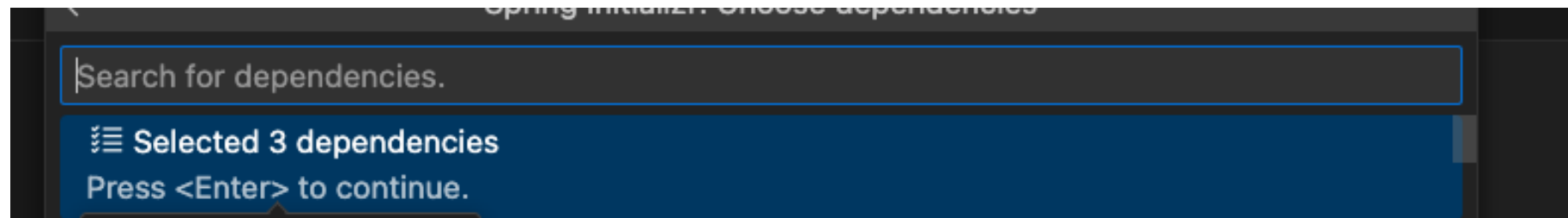
...

Novo Projeto

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



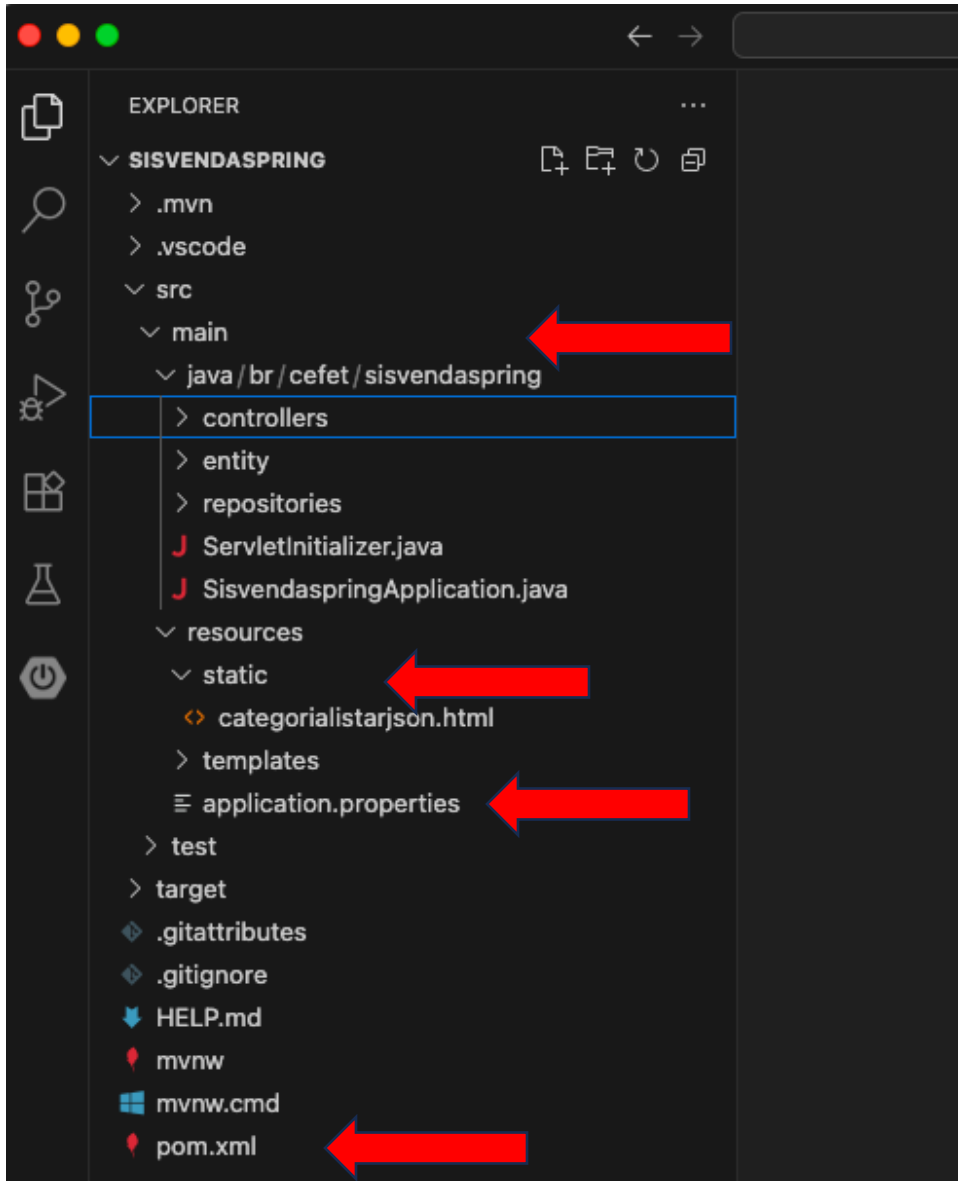
Novo Projeto



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

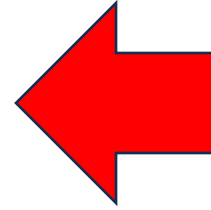
Arquivo pom.xml (Exemplo)

```
<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>  
</dependencies>
```


Maven

Arquivo pom.xml (Exemplo)

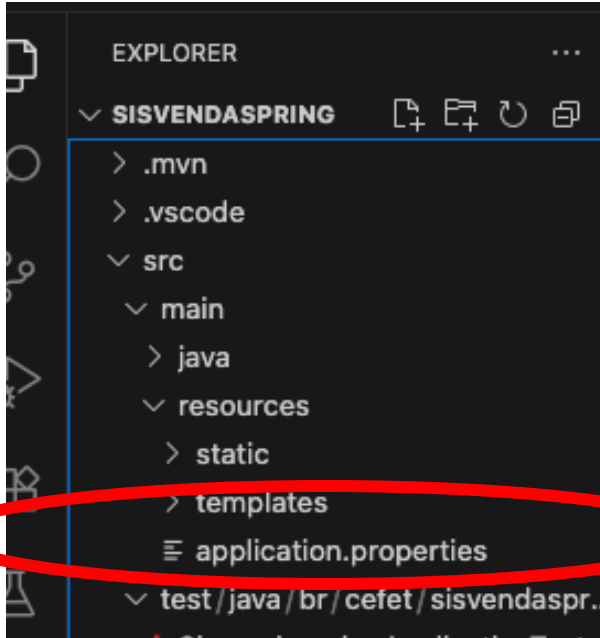
```
<build>  
  <finalName>sisvendaspring</finalName>  
  <plugins>  
    <plugin>  
      <groupId>org.springframework.boot</groupId>  
      <artifactId>spring-boot-maven-plugin</artifactId>  
    </plugin>  
  </plugins>  
</build>
```



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:**

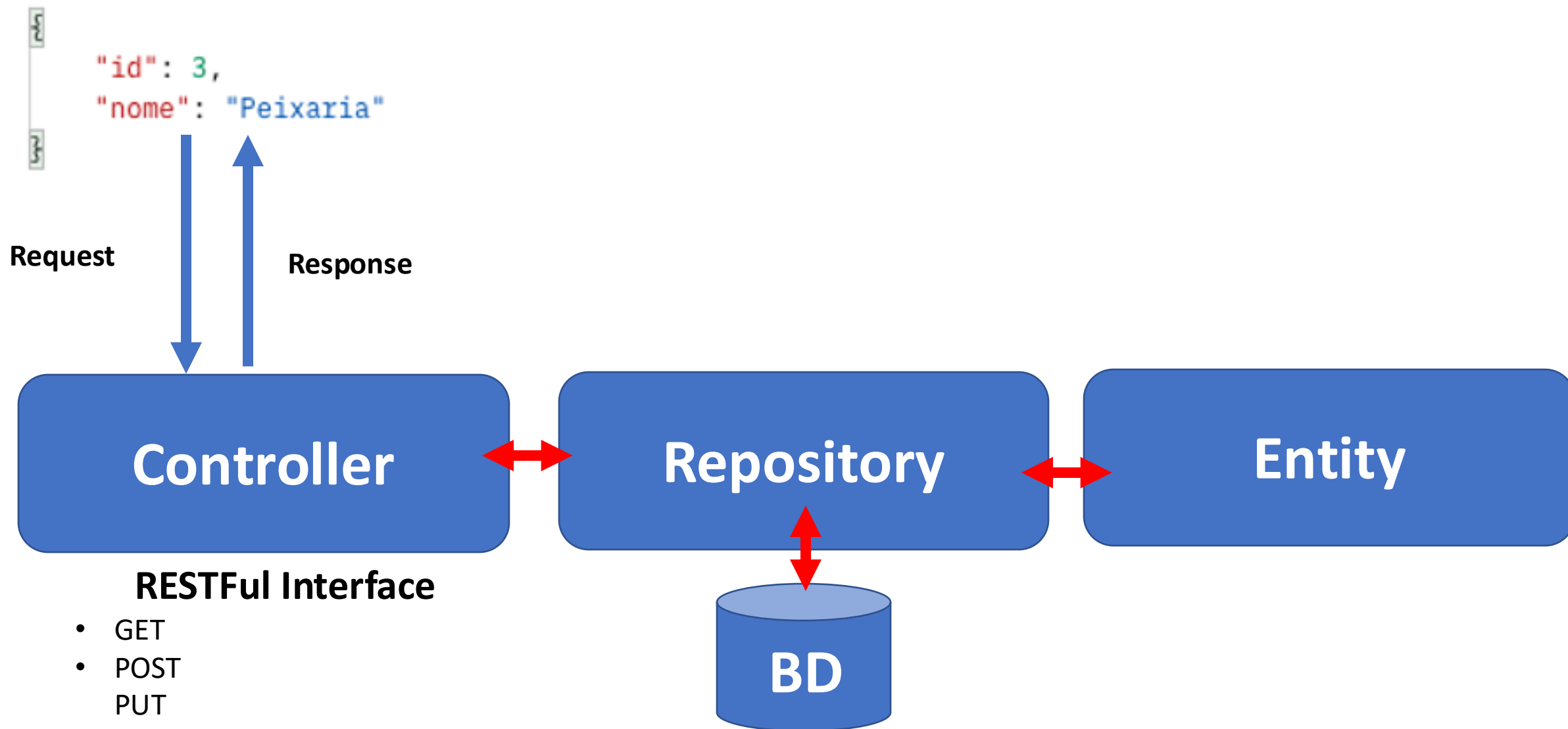
- **ddl-auto:** cria /altera a estrutura de tabelas automaticamente no banco;
- **dbspring** é o database
- **show-sql:** mostra no terminal o sql executado;

- **Na interface do MySQL, execute, apenas:**

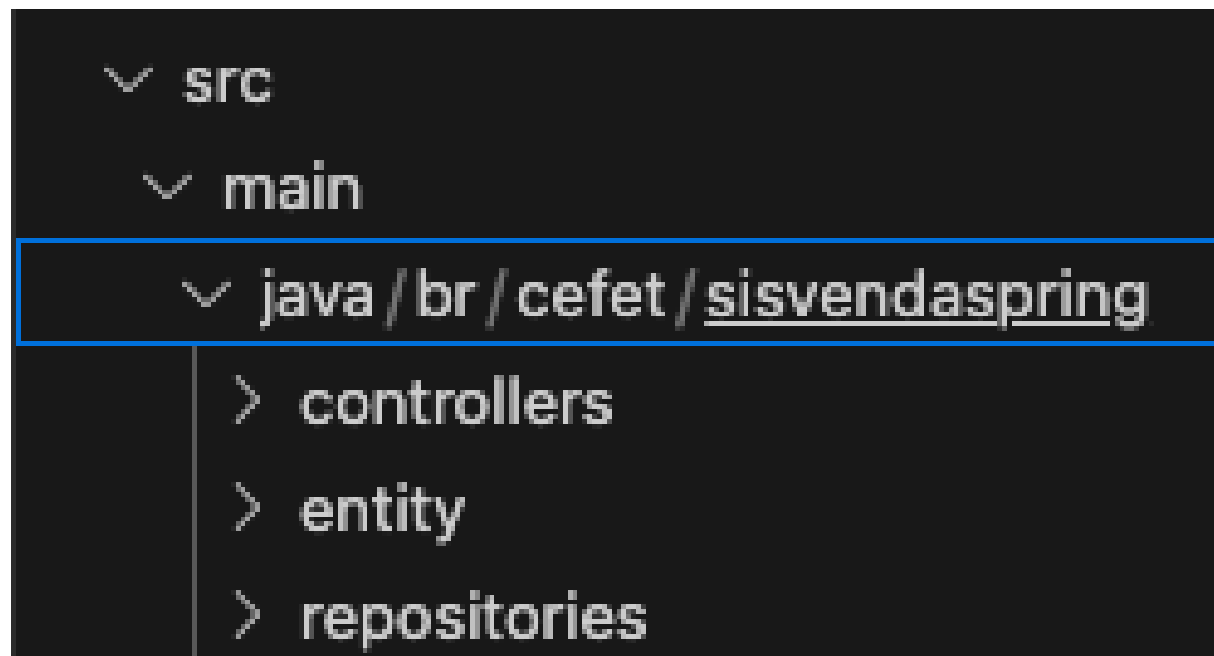
```
CREATE DATABASE dbspring
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```



Arquitetura da API (MVC)



VSCoide – Criar os pacotes



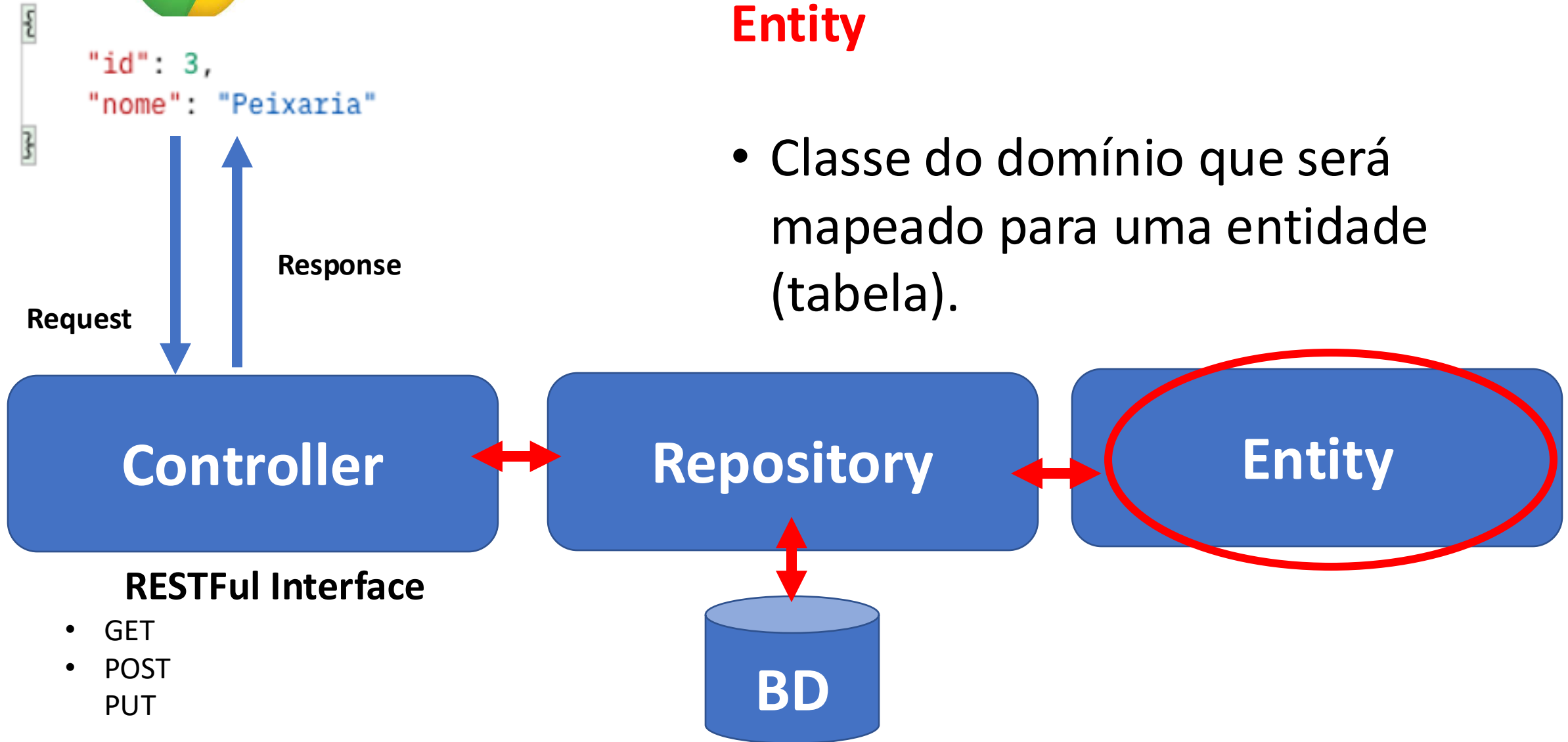
- **Entity**
 - Receberá as classes de modelo;
- **Repository**
 - Contém as interfaces que implementam (herdam) os recursos JPA;
 - Um REPOSITORY inclui todos os métodos, como classificação, paginação de dados e operações CRUD.
 - A notação @Repository é usada para realizar a especificação;
- **Controller**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



Arquitetura da API (MVC)

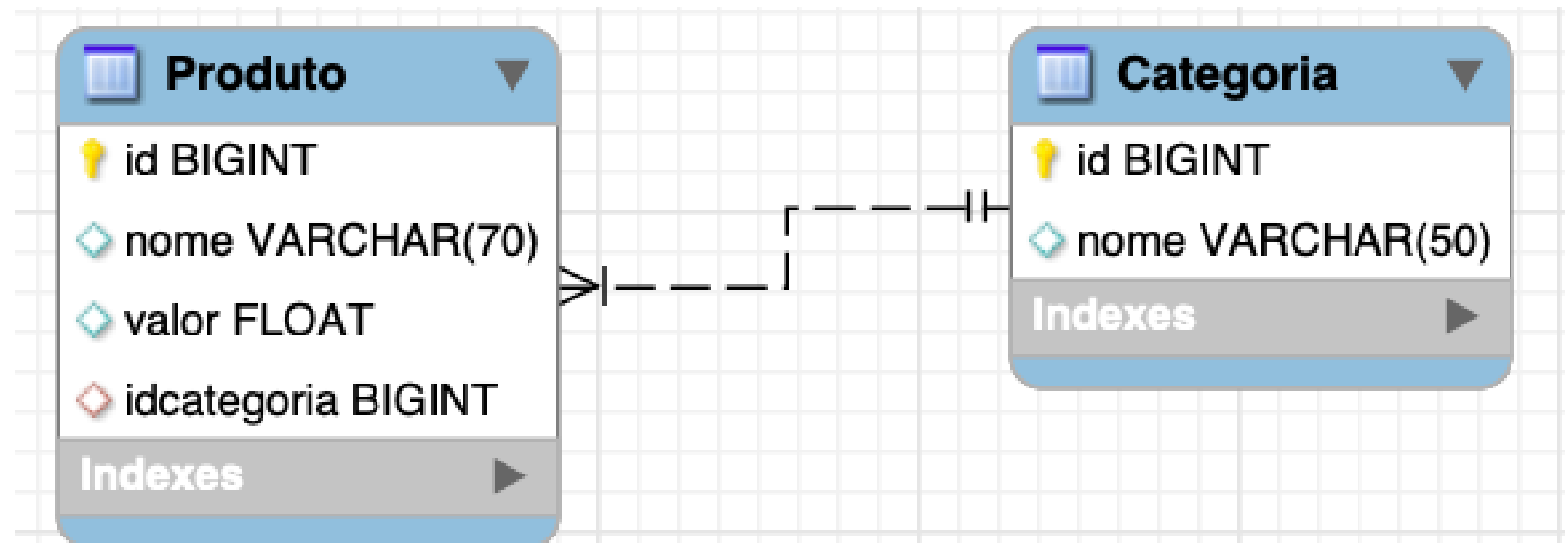
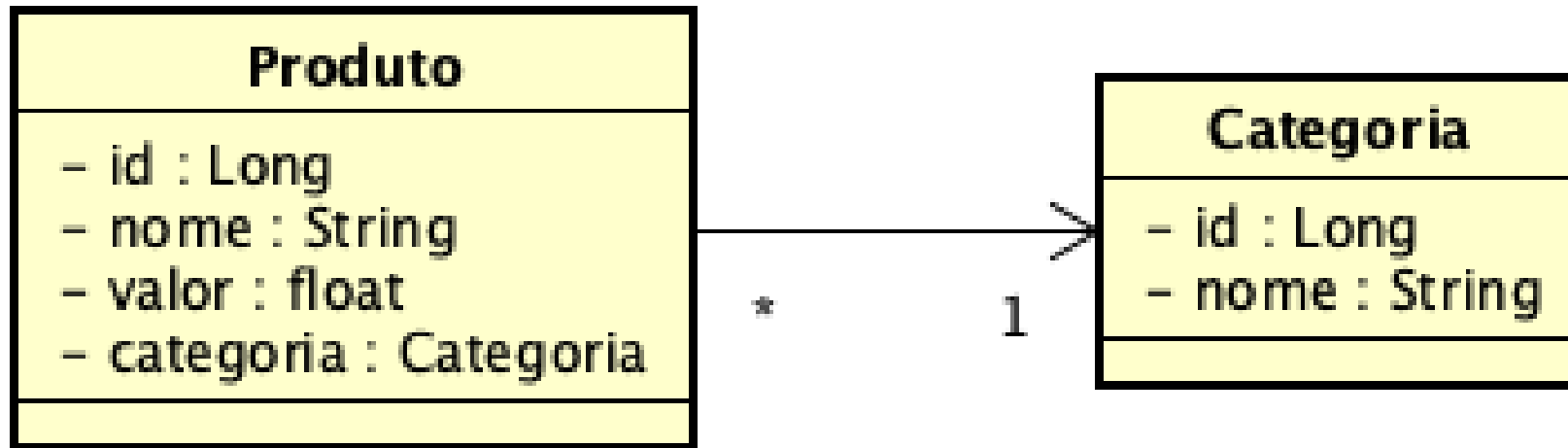
Entity

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



- 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

```
@Entity
```

```
@Table(name = "usu") \\ Quando o nome da tabela for diferente
```

```
public class Usuario {
```

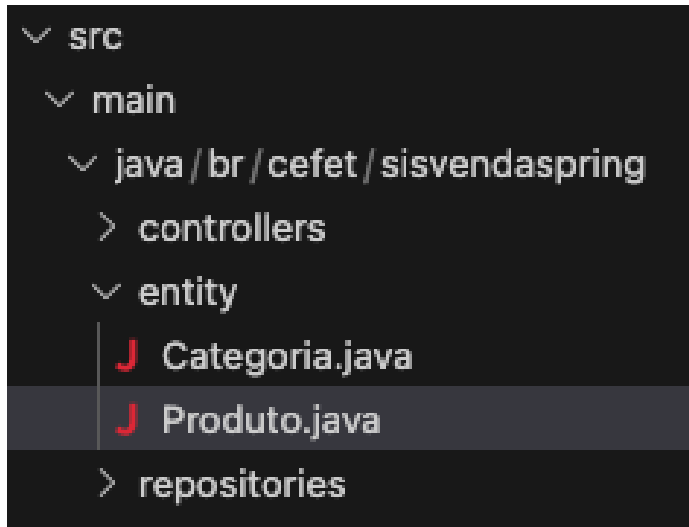
```
\\ Quando o nome do campo for diferente do atributo
```

```
@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 {
```

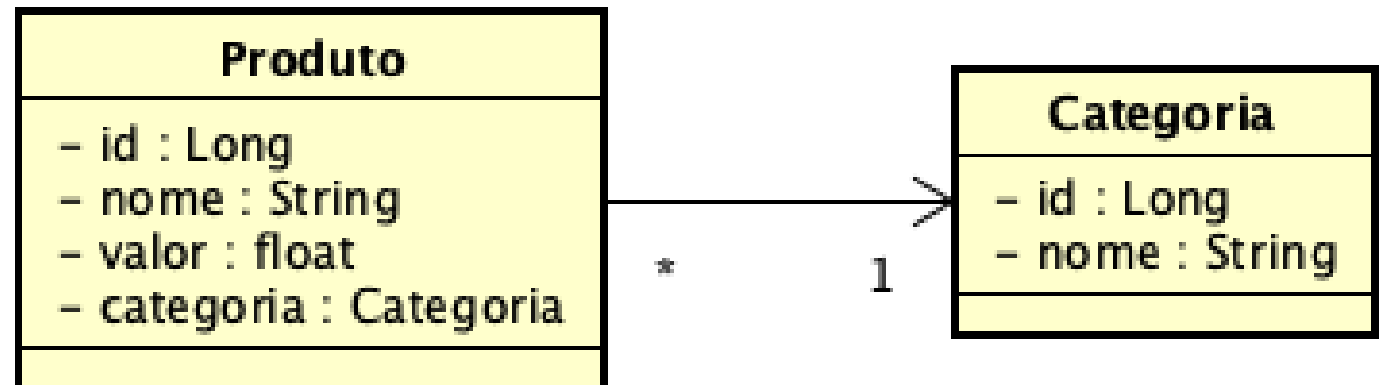
```
    private Long id;
```

```
    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

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  PORTS  [x] zsh +

vendaspringApplication for test class br.cefet.sisvendaspring.SisvendaspringApplicationTests

:: Spring Boot ::                (v3.5.3)

2025-06-29T14:42:16.854-03:00 INFO 6242 --- [sisvenda] [main] b.c.s.SisvendaspringApplicationTests : Starting Sisvendaspring
sing Java 21.0.4 with PID 6242 (started by eliezerdutra in /Users/eliezerdutra/Documents/CEFET/vcCode2/springboot/sisvendaspring/sisvendas
2025-06-29T14:42:16.857-03:00 INFO 6242 --- [sisvenda] [main] b.c.s.SisvendaspringApplicationTests : No active profile set,
default profile: "default"
2025-06-29T14:42:18.183-03:00 INFO 6242 --- [sisvenda] [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring D
s in DEFAULT mode.
2025-06-29T14:42:18.307-03:00 INFO 6242 --- [sisvenda] [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data
in 106 ms. Found 1 JPA repository interface.
2025-06-29T14:42:18.974-03:00 INFO 6242 --- [sisvenda] [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing
[name: default]
2025-06-29T14:42:19.043-03:00 INFO 6242 --- [sisvenda] [main] org.hibernate.Version : HHH000412: Hibernate C
.18.Final
2025-06-29T14:42:19.086-03:00 INFO 6242 --- [sisvenda] [main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level
2025-06-29T14:42:19.492-03:00 INFO 6242 --- [sisvenda] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setu
ss transformer
2025-06-29T14:42:19.538-03:00 INFO 6242 --- [sisvenda] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Startin
2025-06-29T14:42:20.089-03:00 INFO 6242 --- [sisvenda] [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added c
.cj.jdbc.ConnectionImpl@2e4ecd8
2025-06-29T14:42:20.091-03:00 INFO 6242 --- [sisvenda] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start c
2025-06-29T14:42:20.209-03:00 INFO 6242 --- [sisvenda] [main] org.hibernate.orm.connections.pooling : HHH10001005: Database
Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Database driver: undefined/unknown
Database version: 8.0.30
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-06-29T14:42:22.044-03:00 INFO 6242 --- [sisvenda] [main] o.h.e.j.p.i.JtaPlatformInitiator : HHH000489: No JTA plat
hibernate.transaction.jta.platform' to enable JTA platform integration)
Hibernate: create table categoria (id bigint not null auto_increment, nome varchar(255), primary key (id)) engine=InnoDB
Hibernate: create table produto (id bigint not null auto_increment, nome varchar(255), valor float(23) not null, idcategoria bigint, prima
logues
```

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

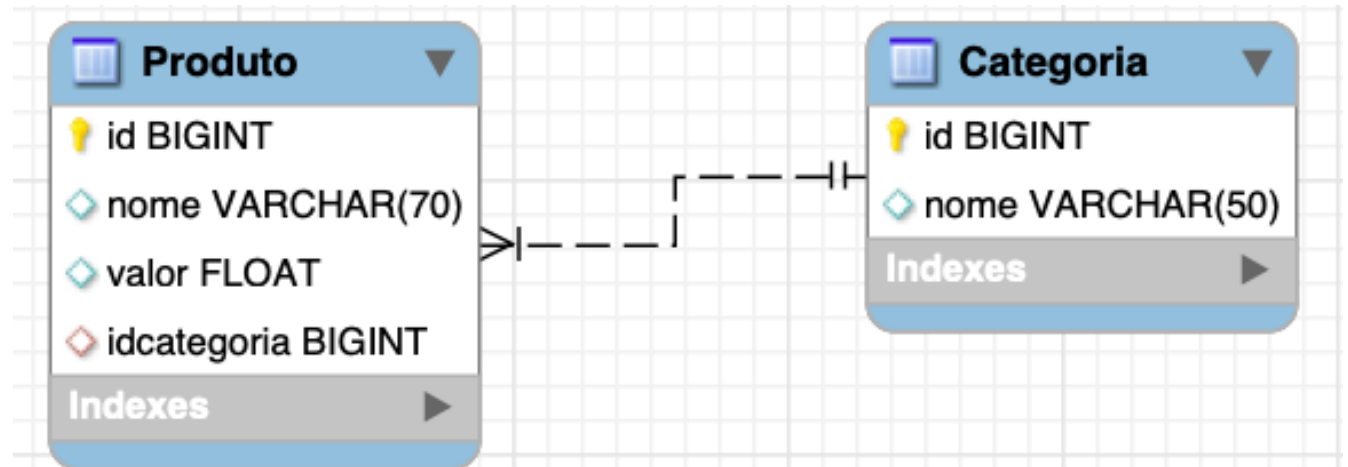
```
CREATE DATABASE dbspring CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
use dbspring;
```

```
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);
```

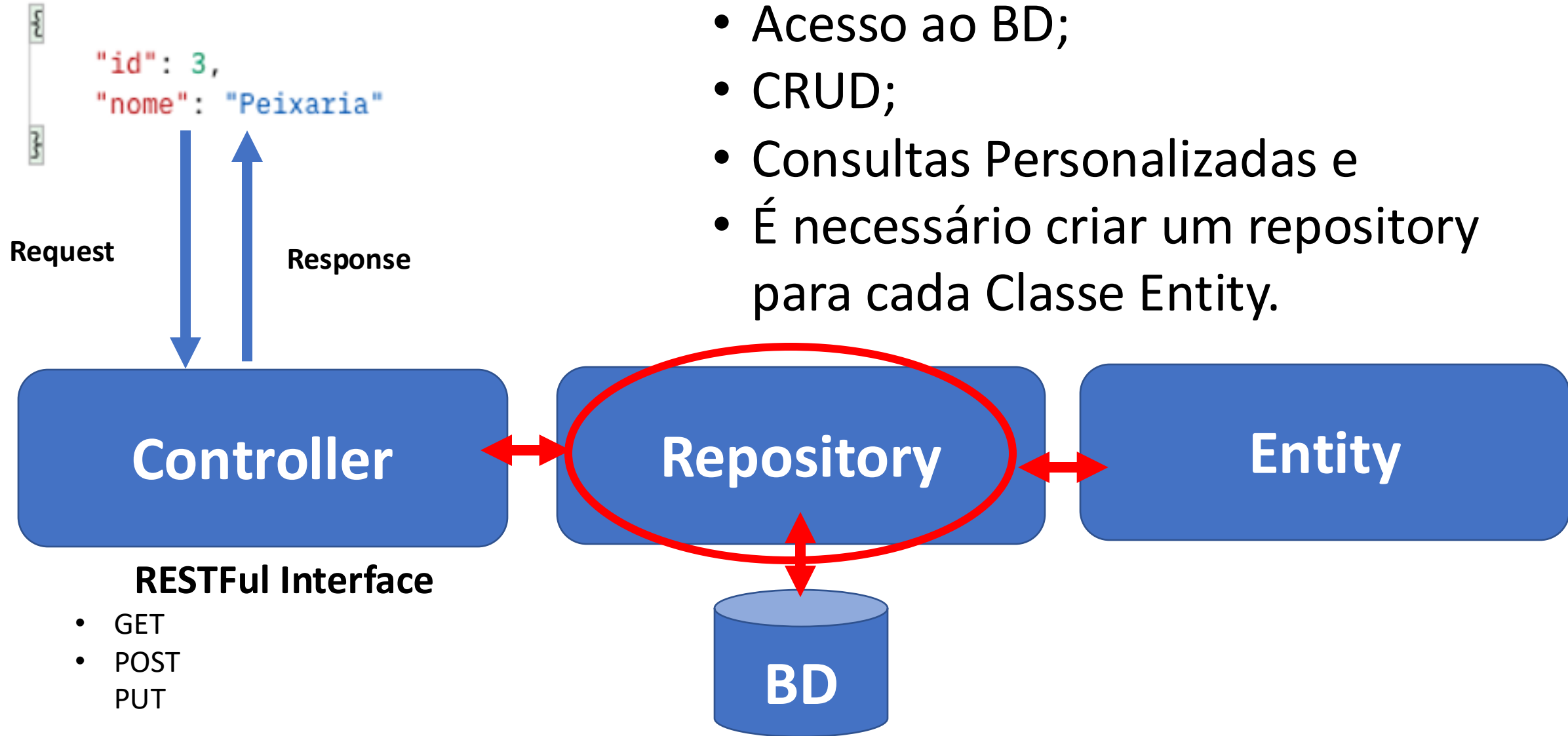




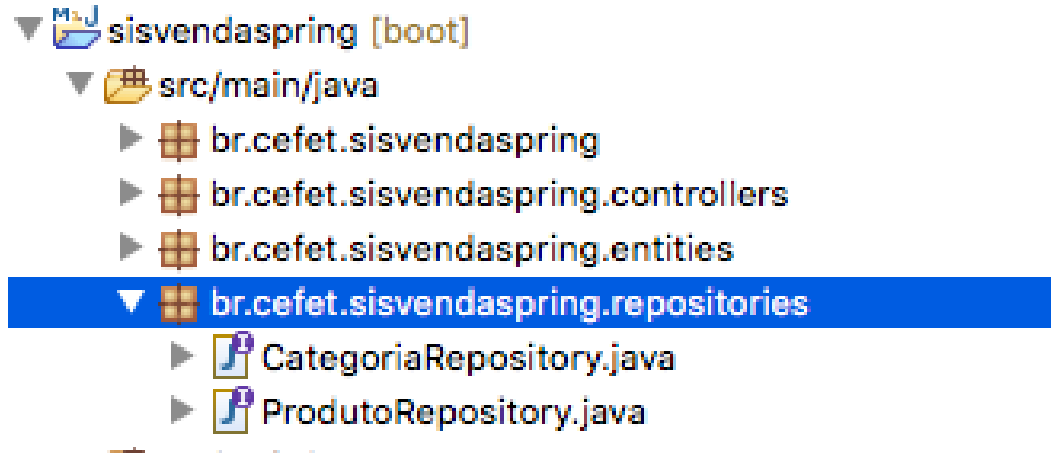
Arquitetura da API (MVC)

Repository (“DAO”)

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



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> {  
  
}  

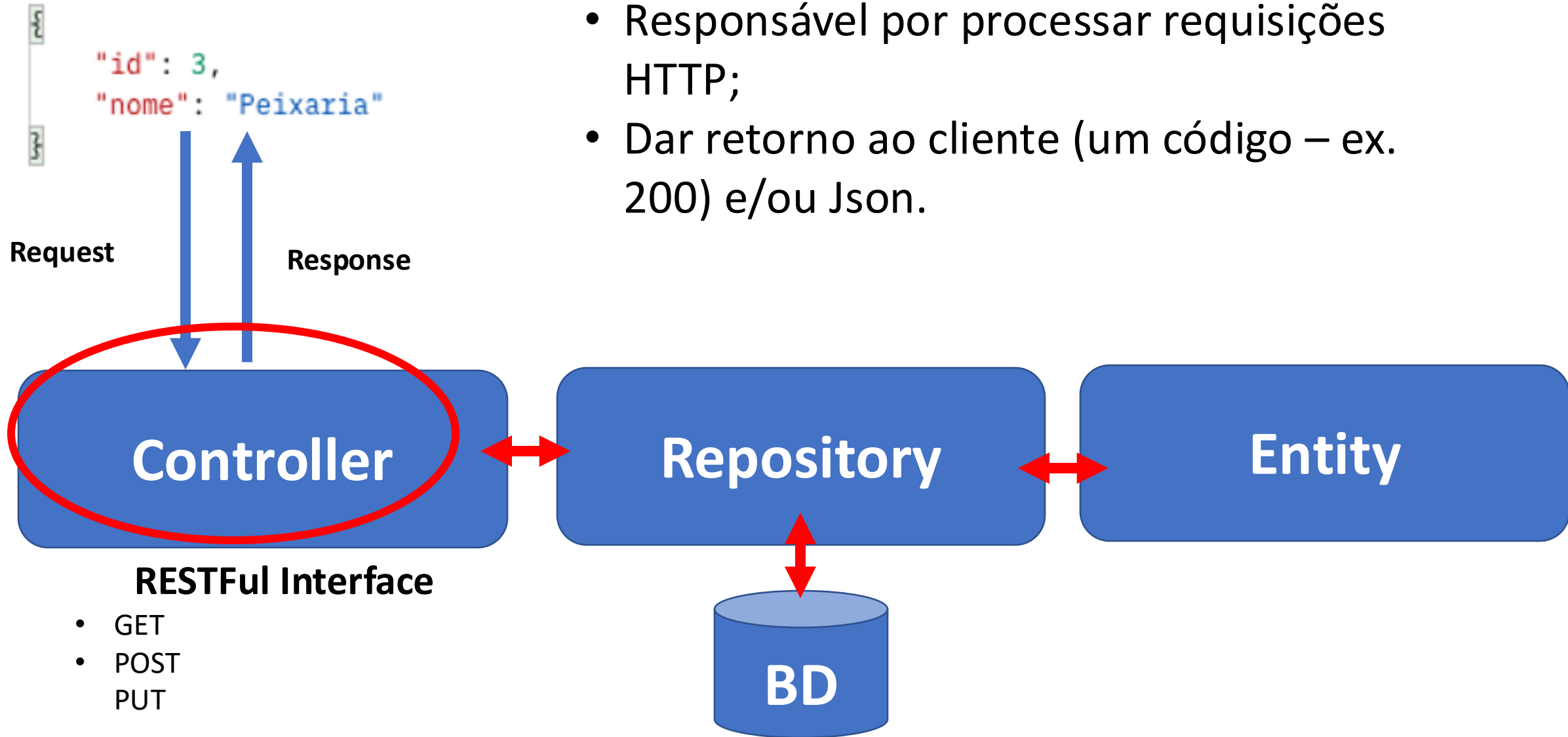
```




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

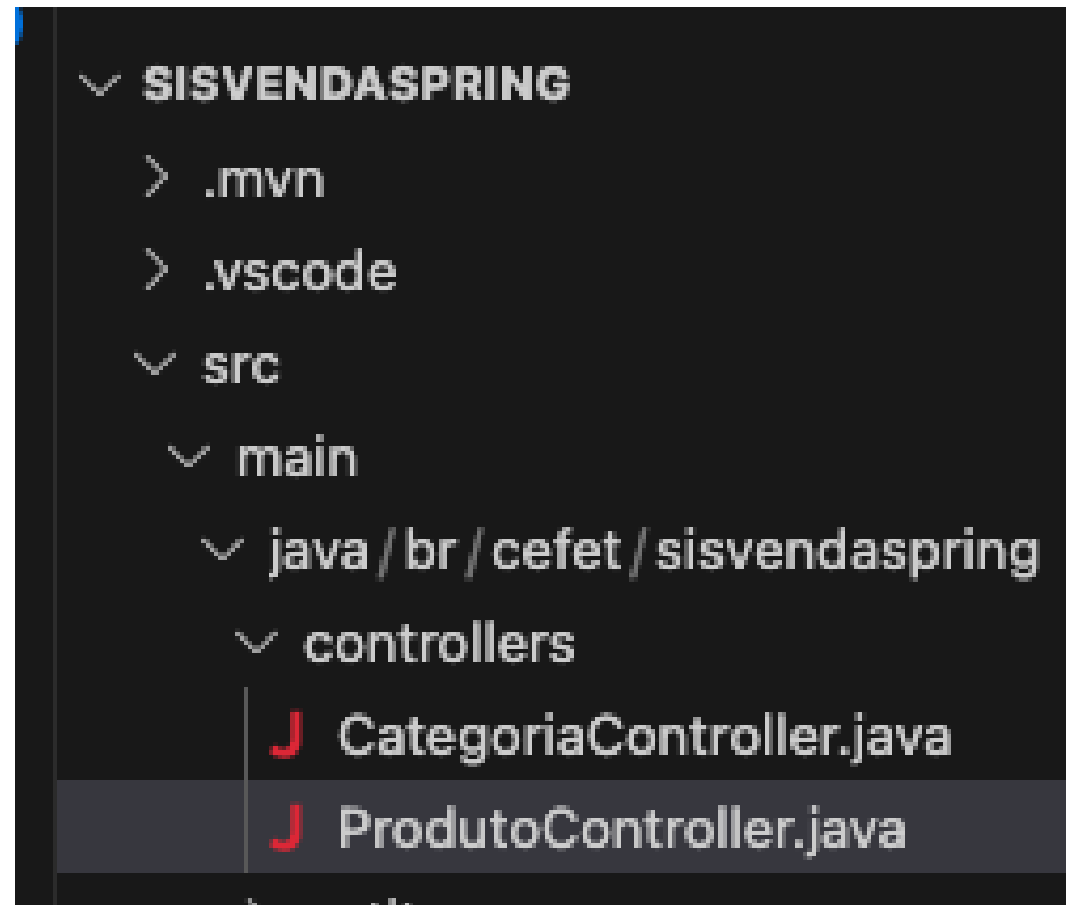
VERBO HTTP	URL (RESTful)	OPERAÇÃO
GET	sisvendaspring/categorias	Lista todas as categorias
GET	sisvendaspring/categorias/3	Retorna uma categoria
DELETE	sisvendaspring/categorias/3	Exclui uma categoria
POST	sisvendaspring/categorias/	Insere uma categoria
PUT	sisvendaspring/categorias/	Atualiza uma categoria

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

```
{
  "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();}

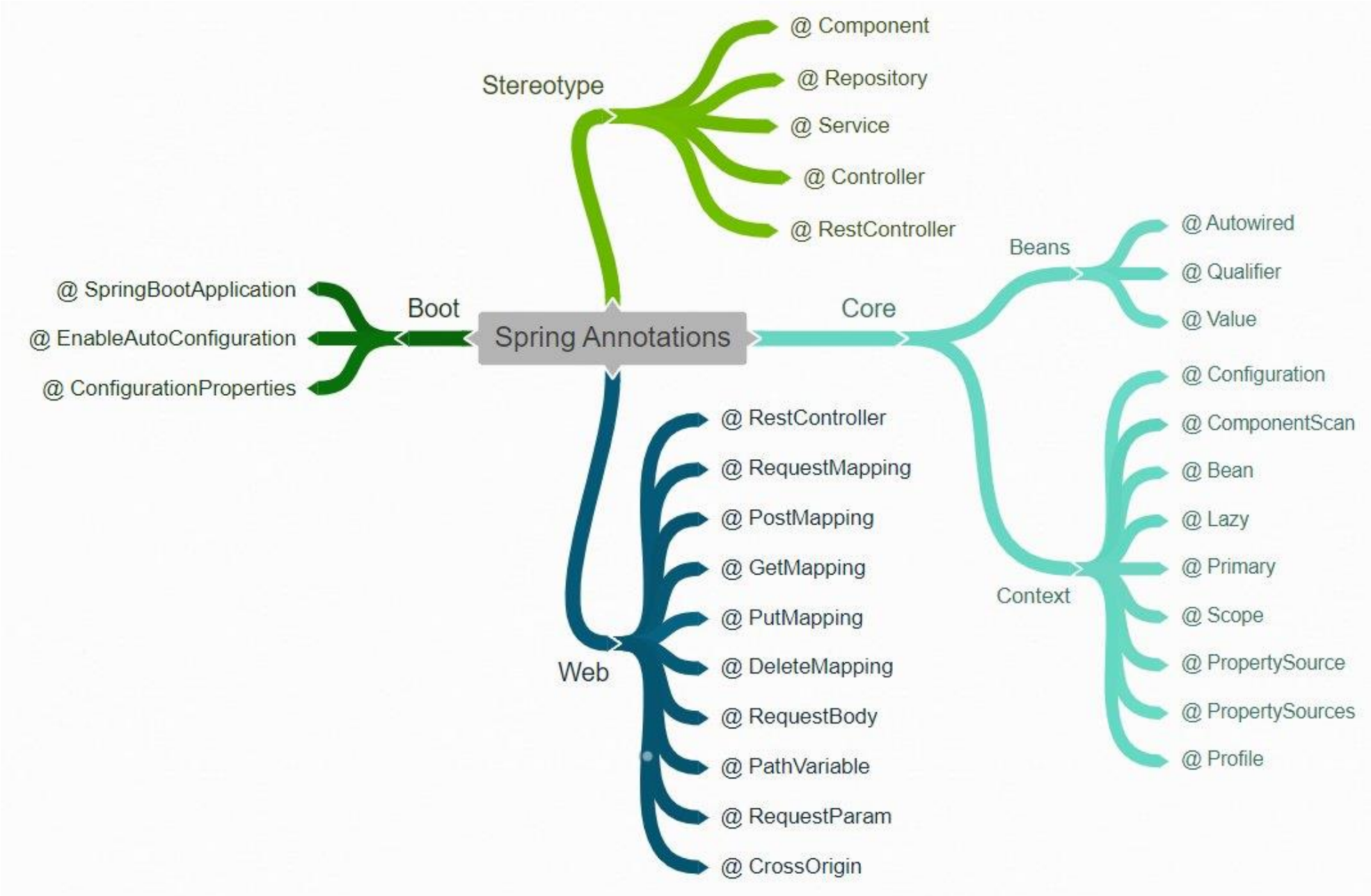
    @GetMapping(value =("/{id}")
    public Categoria findById(@PathVariable Long id) {
        return repository.findById(id).get();
    }

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

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

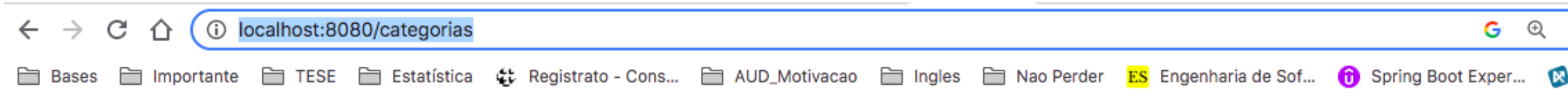
    @PutMapping("/")
    public void update(@RequestBody Categoria categoria) {
        repository.save(categoria);}}
```

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

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

The screenshot shows a REST client interface with the following details:

- Environment:** Sisvenda - Spring
- Method:** GET
- URL:** http://localhost:8080/categorias
- Params:** Query Params table with columns KEY, VALUE, and DESCRIPTION.
- Response:** Status: 200 OK (circled in red), Time: 22 ms, Size: 244 B.
- Body:** JSON array of categories.

KEY	VALUE	DESCRIPTION
Key	Value	Description

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

Testar – GET {ID}

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

http://localhost:8080/categorias/3

Save

DELETE

http://localhost:8080/categorias/3

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

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

Body

Cookies

Headers (4)

Test Results

Status: 200 OK

Time: 390 ms

Size: 123 B

Save Response

Pretty

Raw

Preview

Visualize

Text

1

Testar - Inserir

http://localhost:8080/categorias/

POST http://localhost:8080/categorias/ Send

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

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

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

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

Body Cookies Headers (5) Test Results Status: 200 OK Time: 189 ms Size: 191 B Save Response

Pretty Raw Preview Visualize JSON

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

Testar - Atualizar

The screenshot shows a REST client interface with the following elements:

- Method:** PUT (circled in red)
- URL:** http://localhost:8080/categorias/ (circled in red)
- Body Tab:** Selected and circled in red.
- Body Type:** JSON (circled in red)
- Body Content:** A JSON object:

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

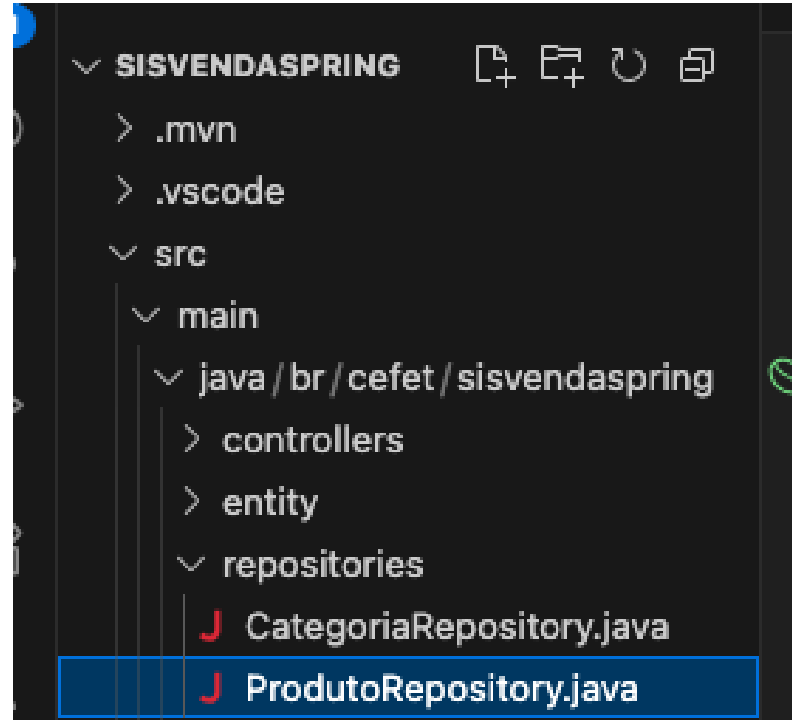
 (The entire body content is circled in red)
- Buttons:** Send, Cookies, Beautify

Below the interface, the full URL is displayed: `http://localhost:8080/sisvendaspring/categorias/`

The screenshot shows the response view of the REST client with the following elements:

- Response Tab:** Selected.
- Response Status:** Status: 200 OK, Time: 65 ms, Size: 123 B
- Response Action:** Save Response
- Response Format:** Pretty, Raw, Preview, Visualize, Text
- Response Content:** 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> {  
}
```

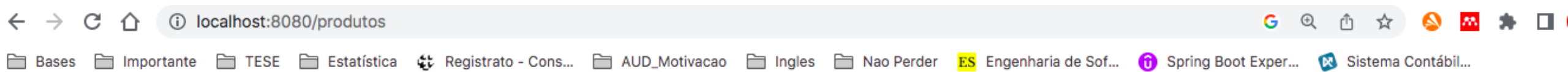
Controller -Produto

```
@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 findById(@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

Status: 200 OK

Time: 30 ms

Size: 238 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

```
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 displays a REST client interface with the following components:

- URL Bar:** `http://localhost:8080/produtos/`
- Method and URL:** **POST** `http://localhost:8080/produtos/`
- Body Tab:** Selected, showing a JSON payload:

```
1 {
2   "id": 0,
3   "nome": "Agua Sanitária",
4   "valor": 4.99,
5   "categoria": {
6     "id": 1,
7     "nome": "Limpeza"
8   }
9 }
```
- Format Selection:** **JSON** (selected), with other options: none, form-data, x-www-form-urlencoded, raw, binary, GraphQL.
- Buttons:** Save, Send, Beautify.
- Response Section:**
 - Body Tab:** Selected, showing the response JSON:

```
1 {
2   "id": 5,
3   "nome": "Agua Sanitária",
4   "valor": 4.99,
5   "categoria": {
6     "id": 1,
7     "nome": "Limpeza"
8   }
9 }
```
 - Status:** 200 OK
 - Time:** 77 ms
 - Size:** 248 B
 - Buttons:** Save Response, Copy, Search.

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 rotas 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

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	<div>Response body</div> <pre>[{ "id": 2, "nome": "Cloro", "valor": 4.99, "categoria": { "id": 1, "nome": "Limpeza" } }, { "id": 3,</pre>

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	<div><p>Response body</p><pre>{ "id": 5, "nome": "Açougue" }</pre><div> Download</div></div> <div><p>Response headers</p><pre>connection: keep-alive content-type: application/json date: Mon,30 Jun 2025 09:19:47 GMT keep-alive: timeout=20 transfer-encoding: chunked</pre></div>