

An aerial photograph of a university campus. The campus features several large, modern buildings with white and red roofs, interspersed with lush green lawns and trees. A winding river or canal flows through the campus, with a bridge crossing it. In the foreground, there's a large, curved building with a red roof. The overall scene is bright and sunny, with clear skies.

# WEB ACADEMY

## Frameworks Back-end

Daniel Augusto Nunes da Silva

# **Apresentação**

# Ementa

- Frameworks Back-end. **Spring Framework**. Injeção de dependência. **Spring Boot**. Persistência de dados com **JPA**, Hibernate e Mapeamento Objeto-Relacional (ORM). Spring Data. **Arquitetura REST e APIs**. Mapeamento de requisições HTTP. Segurança.



# Objetivos

- **Geral:** Habilitar o aluno na utilização de **frameworks para desenvolvimento de aplicações WEB voltadas para o back-end**, apoiadas nas ferramentas dos projetos que fazem parte do Spring.
- **Específicos:**
  - Compreender o papel dos frameworks no contexto do desenvolvimento web.
  - Apresentar os principais recursos da família de projetos Spring com ênfase na construção de projetos Spring Boot.
  - Demonstrar como o conjunto de ferramentas do Spring podem otimizar a persistência de dados.
  - Capacitar o aluno na construção de uma API REST baseada em um projeto Spring Boot.

# Conteúdo programático

## Introdução

- Programação server-side;
- Frameworks web (back-end);
- Spring Framework;
- Inversão de controle e injeção de dependência.

## Spring Boot

- Introdução ao Spring Boot;
- Criação de projetos Spring Boot;
- Anotações e meta-anotações;
- Execução da aplicação e deploy no servidor de produção.

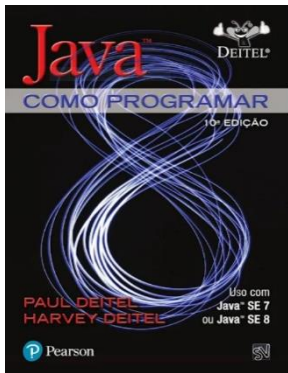
## Persistência de dados

- Introdução ao JPA, Hibernate e ORM;
- Estratégias para geração de chaves primárias;
- Relacionamento entre entidades;
- Spring Data.

## API

- Introdução à arquitetura REST e construção de APIs.
- Camadas de uma API REST.
- Endpoints e mapeamento de requisições HTTP.
- Segurança: autenticação de usuários e CORS.

# Bibliografia



## Java: Como Programar.

Paul Deitel e Harvey Deitel  
10ª Edição – 2016  
Editora Pearson  
ISBN 9788543004792



## Spring in Action

Craig Walls  
6ª Edição – 2021  
Editora Manning  
ISBN 9781617297571



## Engenharia de Software Moderna

Marco Tulio Valente  
<https://engsoftmoderna.info/>



# Sites de referência

- Spring Boot Reference Documentation
  - <https://docs.spring.io/spring-boot/docs/current/reference/html/index.html>
- Spring Getting Started Guides
  - <https://spring.io/guides#getting-started-guides>
- Apostila Java e Orientação a Objetos (Caelum/Alura)
  - <https://www.alura.com.br/apostila-java-orientacao-objetos>
- Java Tutorial (VS Code)
  - <https://code.visualstudio.com/docs/java/java-tutorial>

# Ferramentas

- **Visual Studio Code:** <https://code.visualstudio.com/Download>
- **Extension Pack for Java (Extensão do VS Code):**  
<https://marketplace.visualstudio.com/items?vscjava.vscode-java-pack>
- **Spring Boot Extension Pack (Extensão do VS Code):**  
<https://marketplace.visualstudio.com/items?itemName=pivotal.vscode-boot-dev-pack>
- **Thunder Client (Extensão do VS Code):**  
<https://marketplace.visualstudio.com/items?itemName=rangav.vscode-thunder-client>
- **XML (Extensão do VS Code):**  
<https://marketplace.visualstudio.com/items?itemName=redhat.vscode-xml>



# Ferramentas: JDK e Maven

- **JDK 11**

- <https://www.oracle.com/br/java/technologies/javase/jdk11-archive-downloads.html>
- Criar a variável de ambiente JAVA\_HOME configurada para o diretório de instalação do JDK. Exemplo: “C:\Program Files\Java\jdk-11.0.13”.
- Adicionar “%JAVA\_HOME%\bin” na variável de ambiente PATH.
- Tutorial de configuração: [https://mkyong.com/java/how-to-set-java\\_home-on-windows-10/](https://mkyong.com/java/how-to-set-java_home-on-windows-10/)

- **Maven**

- <https://maven.apache.org/download.cgi>
- Adicionar o diretório de instalação do Maven na variável de ambiente PATH. Exemplo: “C:\apache-maven\bin”.
- Tutorial de instalação: <https://mkyong.com/maven/how-to-install-maven-in-windows/>

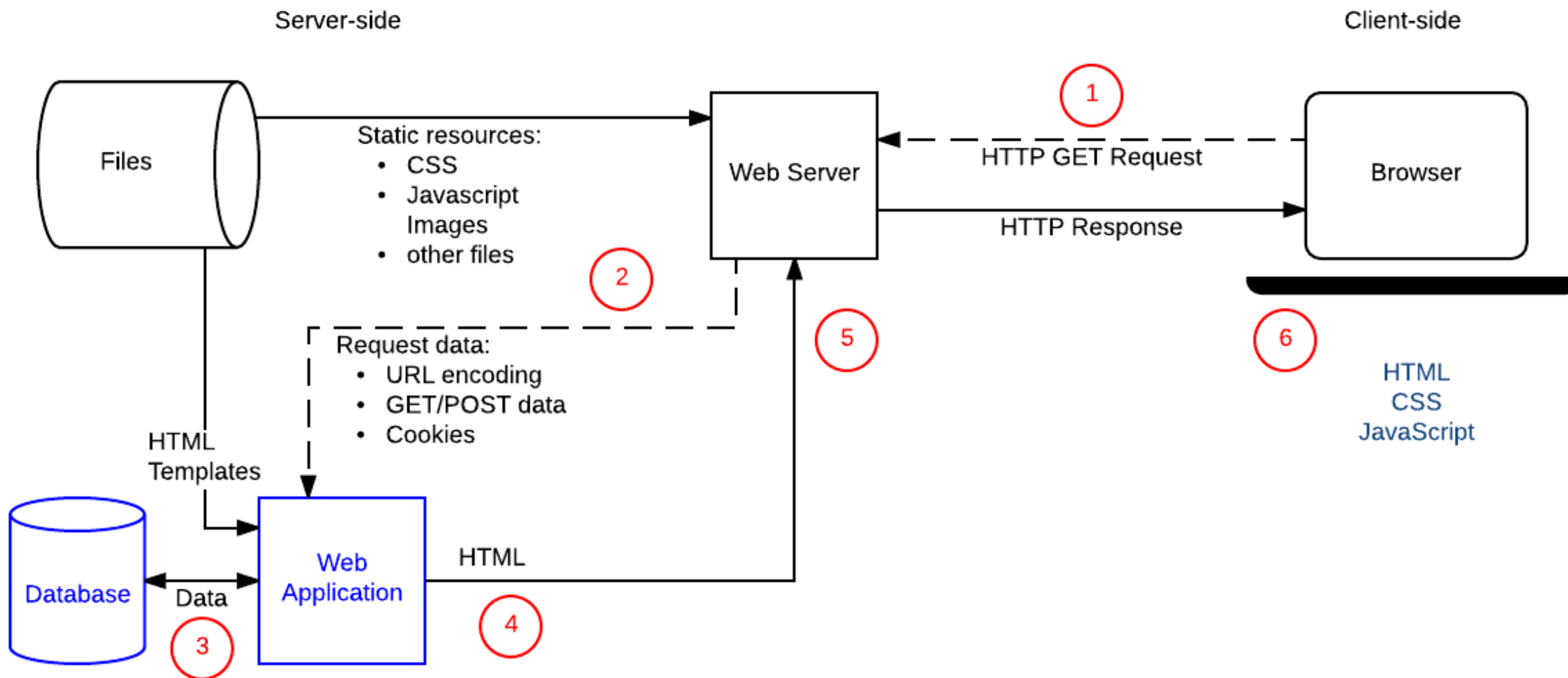
# Contato



<https://linkme.bio/danielnsilva/>

# Introdução

# Programação server-side



Fonte: [https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First\\_steps/Introduction](https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/First_steps/Introduction)

# Frameworks web (back-end)

- Fornecem ferramentas que **simplificam as operações comuns de desenvolvimento**.
- **Não precisamos de um framework**, mas facilitam muito o trabalho de desenvolvimento.
- Vantagens: produtividade, padronização, reusabilidade, segurança.
- Desvantagens: dependência, segurança (vulnerabilidades), performance.
- Exemplos: Django e Flask (Python), Laravel (PHP), Spring (Java).



# Spring


- Originalmente denominado **Spring Framework**.
- Pretendia tornar o desenvolvimento de aplicações J2EE mais fácil.
- O foco do framework não é apenas aplicações web.
- Os recursos para desenvolvimento de aplicações web são baseados em **servlets**.
- Conceitos importantes: **inversão de controle** e **injeção de dependência**.



# Inversão de controle e injeção de dependência

- **Inversão de controle permite mudar o fluxo de controle de um programa**, transferindo para um componente externo a responsabilidade de quando executar determinado procedimento.
- A **injeção de dependência** é uma **forma de aplicar a inversão de controle**.
- A dependência não é criada internamente (nova instância de um objeto), mas “injetada” por uma classe externa.

**Criação de dependência**



```
1. public class Controller {
2.     private PessoaDao dao;
3.     public Controller() {
4.         this.dao = new PessoaDao("mysql");
5.     }
6.     public Pessoa getById(int id) {
7.         return dao.getById(id);
8.     }
9. }
10. Controller c = new Controller();
11. Pessoa pessoa = c.getById(1);
```

# Inversão de controle e injeção de dependência

- **Inversão de controle permite mudar o fluxo de controle de um programa**, transferindo para um componente externo a responsabilidade de quando executar determinado procedimento.
- A **injeção de dependência** é uma **forma de aplicar a inversão de controle**.
- A dependência não é criada internamente (nova instância de um objeto), mas “injetada” por uma classe externa.

```
1. public class Controller {  
2.     private PessoaDao dao;  
3.     public Controller(PessoaDao dao) {  
4.         this.dao = dao;  
5.     }  
6.     public Pessoa getById(int id) {  
7.         return dao.getById(id);  
8.     }  
9. }
```

```
10. PessoaDao dao = new PessoaDao("mysql");  
11. Controller c = new Controller(dao);  
12. Pessoa pessoa = c.getById(1);
```

**Injeção de dependência**



**Escopo externo**

# Inversão de controle e injeção de dependência

- **Inversão de controle permite mudar o fluxo de controle de um programa**, transferindo para um componente externo a responsabilidade de quando executar determinado procedimento.
- A **injeção de dependência** é uma **forma de aplicar a inversão de controle**.
- A dependência não é criada internamente (nova instância de um objeto), mas “injetada” por uma classe externa.

```
1. public class Controller {  
2.     private IDao dao;  
3.     public Controller(IDao dao) {  
4.         this.dao = dao;  
5.     }  
6.     public Pessoa getById(int id) {  
7.         return dao.getById(id);  
8.     }  
9. }
```

```
10. IDao dao = new AlunoDao("mysql");  
11. Controller c = new Controller(dao);  
12. Aluno aluno = c.getById(1);
```

**Injeção de dependência**

**Escopo externo**

# Inversão de controle e injeção de dependência

- Para saber mais sobre o assunto:
  - <https://engsoftmoderna.info/artigos/injecao-dependencia.html>
  - <https://engsoftmoderna.info/cap6.html#template-method>
  - <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans>



# Spring

- O framework ganhou muitos recursos e foi desmembrado em vários projetos, entre eles:
  - **Spring Framework**: fornece os recursos “básicos”.
  - **Spring Data**: facilita a integração com vários tipos de tecnologias de gerenciamento de dados.
  - **Spring Security**: autenticação e controle de acesso.
  - **Spring Boot**: abstrai a complexidade de configuração de servidores de aplicação.

# Spring Boot

# Introdução ao Spring Boot

- Facilita o processo de configuração e implantação das aplicações.
  - **Servidor de aplicação embutido.**
  - Gerenciamento de dependências e configurações por meio dos **starters**.
- Responsável por impulsionar a plataforma Spring.



# Criando projetos Spring Boot

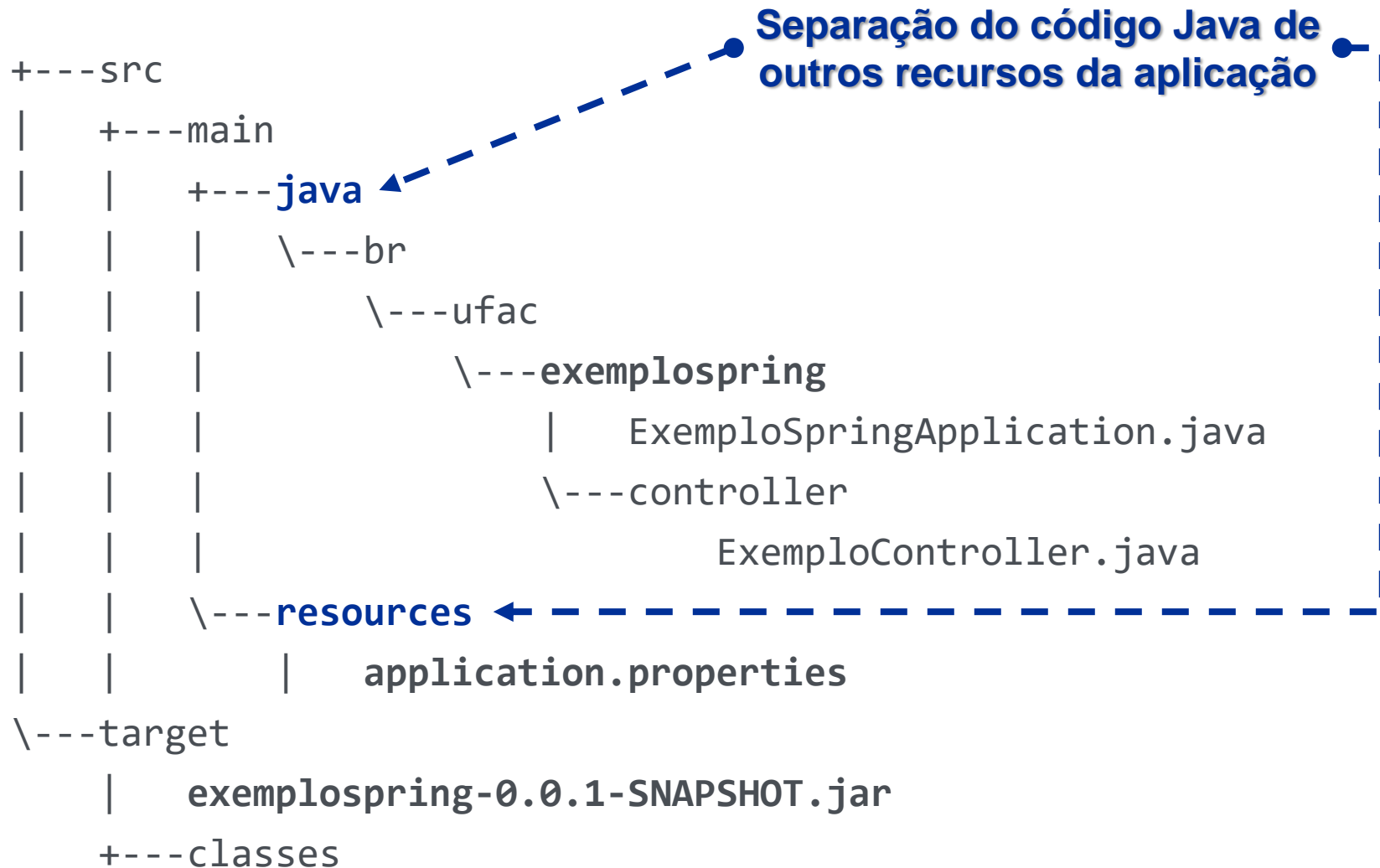
- É necessário um gerenciador de projetos como o **Maven**.
- A ferramenta **Spring Initializr** (<https://start.spring.io/>) ajuda a criar o projeto com as dependências necessárias.
- O **VS Code também pode fornecer um recurso semelhante** por meio de extensões.
- É um projeto Maven como qualquer outro, exceto pelos **starters** adicionados como dependências ao projeto.
  - Starters: <https://docs.spring.io/spring-boot/docs/current/reference/html/using.html#using.build-systems.starters>
  - Maven: <https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/>

# Estrutura do projeto

```
+---src
|   +---main
|   |   +---java
|   |   |   \---br
|   |   |       \---ufac
|   |   |           \---exemplospring
|   |   |               |   ExemploSpringApplication.java
|   |   |               \---controller
|   |   |                   ExemploController.java
|   |   \---resources
|   |       |   application.properties
\---target
    |   exemplospring-0.0.1-SNAPSHOT.jar
    +---classes
```



# Estrutura do projeto



# Estrutura do projeto

```
+---src
|   +---main
|   |   +---java
|   |   |   \---br
|   |   |       \---ufac
|   |   |           \---exemplospring ← - - - - -
|   |   |               |   ExemploSpringApplication.java
|   |   |               \---controller
|   |   |                   ExemploController.java
|   |   \---resources
|   |       |   application.properties
\---target
    |   exemplospring-0.0.1-SNAPSHOT.jar
    +---classes
```

A classe que contém o  
método main() deve ficar  
na raiz do pacote principal.

# Estrutura do projeto


```
+---src
|   +---main
|   |   +---java
|   |   |   \---br
|   |   |       \---ufac
|   |   |           \---exemplospring
|   |   |               |   ExemploSpringApplication.java
|   |   |               \---controller
|   |   |                   ExemploController.java
|   |   \---resources
|   |       |   application.properties  ◀-----
\---target
    |   exemplospring-0.0.1-SNAPSHOT.jar
+---classes
```

**Define propriedades da aplicação, como conexão com banco de dados, segurança, porta TCP, etc.**

# Estrutura do projeto

```
+---src
|   +---main
|   |   +---java
|   |   |   \---br
|   |   |       \---ufac
|   |   |           \---exemplospring
|   |   |               |   ExemploSpringApplication.java
|   |   |               \---controller
|   |   |                   ExemploController.java
|   |   \---resources
|   |       |   application.properties
\---target
    |   exemplospring-0.0.1-SNAPSHOT.jar
    +---classes
```

**Executável JAR contendo a aplicação completa.**



# Anotações

- Em Java, **uma anotação descreve um componente** (classe, método ou atributo), adicionando metadados ao código.
  - **@SpringBootApplication** identifica a classe principal da aplicação.
- Anotações representam uma **alternativa aos arquivos de configuração XML**.
- Uma parte significativa do funcionamento do **Spring Boot depende de anotações**.
  - <https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-annotation-config>



# Anotações

```
1. @SpringBootApplication
2. public class Application {
3.     public static void main(String[] args) {
4.         Application.run(Application.class, args);
5.     }
6. }
```

# Meta-anotações

- Muitas anotações são na verdade meta-anotações (anotações que encapsulam outras anotações).
- **@SpringBootApplication** é uma meta-anotação para:
  - **@Configuration**, que permite registrar *beans* no contexto ou importar classes de configuração adicionais;
  - **@EnableAutoConfiguration**, que habilita a configuração automática do Spring Boot para aplicar configurações baseadas nas dependências que foram adicionadas.
  - **@ComponentScan**, que faz uma busca por outras classes anotadas com @Component.

# Executando a aplicação

- A aplicação pode ser inicializada de três formas:
  - **Spring Dashboard.**
  - **Maven:**
    - > mvn spring-boot:run
  - **Executando o pacote (JAR):**
    - > mvn package
    - > java -jar target\exemplo.jar
- **Deploy:** o arquivo JAR pode ser executado no servidor de produção.

# Persistência de dados

# Introdução ao JPA, Hibernate e ORM

- **Java Persistence API (JPA)**, atualmente *Jakarta Persistence*, fornece uma interface comum para persistência de dados.
- JPA define uma forma de representar as entidades de banco de dados relacionais através de classes, utilizando a técnica do **mapeamento objeto-relacional** (ORM, *object-relational mapping*).
- **JPA é apenas uma especificação**, não faz ORM.
- Frameworks ORM, como o **Hibernate**, implementam JPA, gerando as chamadas SQL automaticamente.

# Introdução ao JPA, Hibernate e ORM

```
1. @Entity
2. public class Especialidade implements Serializable {
3.     @Id // Chave primária
4.     @GeneratedValue(strategy = GenerationType.IDENTITY)
5.     @Column(nullable = false, updatable = false)
6.     private Long id;
7.     @Column(nullable = false, unique = true)
8.     private String nome;
9. }
```

# Estratégias para geração de chaves primárias

- GenerationType.**IDENTITY**: no MySQL é o mesmo que utilizar AUTO\_INCREMENT, mas pode mudar para diferentes SGBD.
- GenerationType.**SEQUENCE**: um *sequence* é um recurso do SGBD para gerar chaves únicas para um grupo (*sequence*), podendo existir vários no banco de dados, mas nem todo SGBD suporta esta funcionalidade.
- GenerationType.**TABLE**: utiliza uma tabela para gerenciar as chaves geradas, sendo uma estratégia compatível com qualquer SGBD, mas que pode afetar o desempenho.
- GenerationType.**AUTO**: o framework ORM (Hibernate) escolhe a estratégia de acordo com o SGBD.

# Dependência (pom.xml)

1. `<dependency>`
2.     `<groupId>org.springframework.boot</groupId>`
3.     `<artifactId>spring-boot-starter-data-jpa</artifactId>`
4. `</dependency>`
5. `<dependency>`
6.     `<groupId>mysql</groupId>`
7.     `<artifactId>mysql-connector-java</artifactId>`
8.     `<scope>runtime</scope>`
9. `</dependency>`



# Configurações de conexão (application.properties)

- **Fonte de dados:**

1. `spring.datasource.url=jdbc:mysql://localhost:3306/sgcm?createDatabaseIfNotExist=true`
2. `spring.datasource.username=root`
3. `spring.datasource.password=root`

# Configurações de conexão (application.properties)

- **JPA/Hibernate/ORM:**

1. `spring.jpa.show-sql=true`
2. `spring.jpa.hibernate.ddl-auto=update`
3. `spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect`

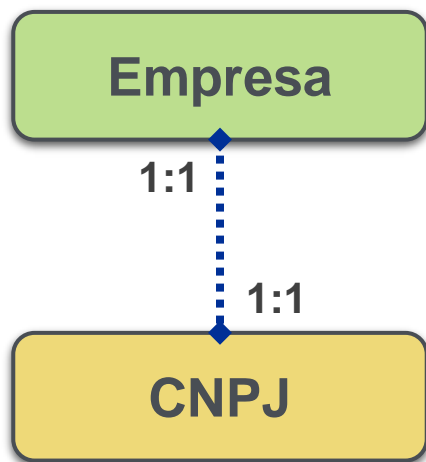
# Configurações de conexão (application.properties)

- **Inicialização do banco de dados com scripts SQL:**
  1. `spring.jpa.defer-datasource-initialization=true`
  2. `spring.sql.init.mode=always`
  3. `spring.sql.init.continue-on-error=true`
- <https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto.data-initialization.using-basic-sql-scripts>

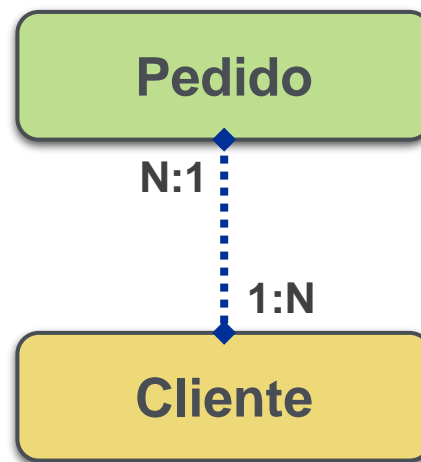
# Relacionamento entre entidades

- O **Hibernate facilita o mapeamento de entidades relacionadas**, por meio do ORM, utilizando anotações definidas no JPA.

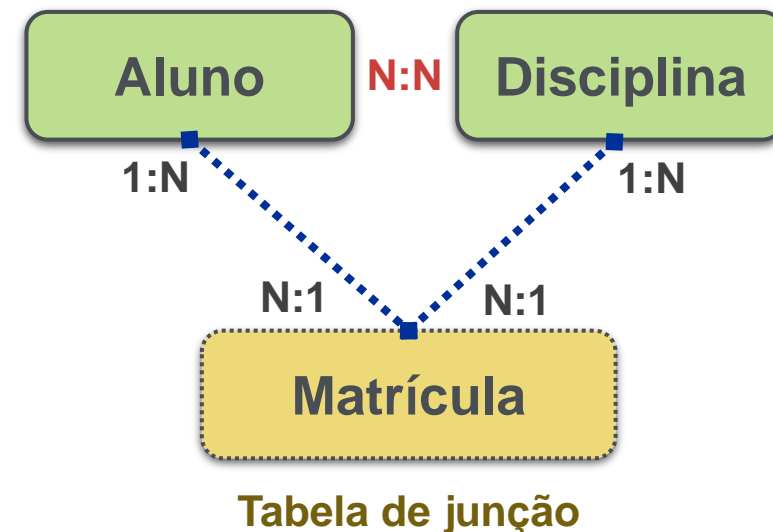
## @OneToOne



## @ManyToOne @OneToMany

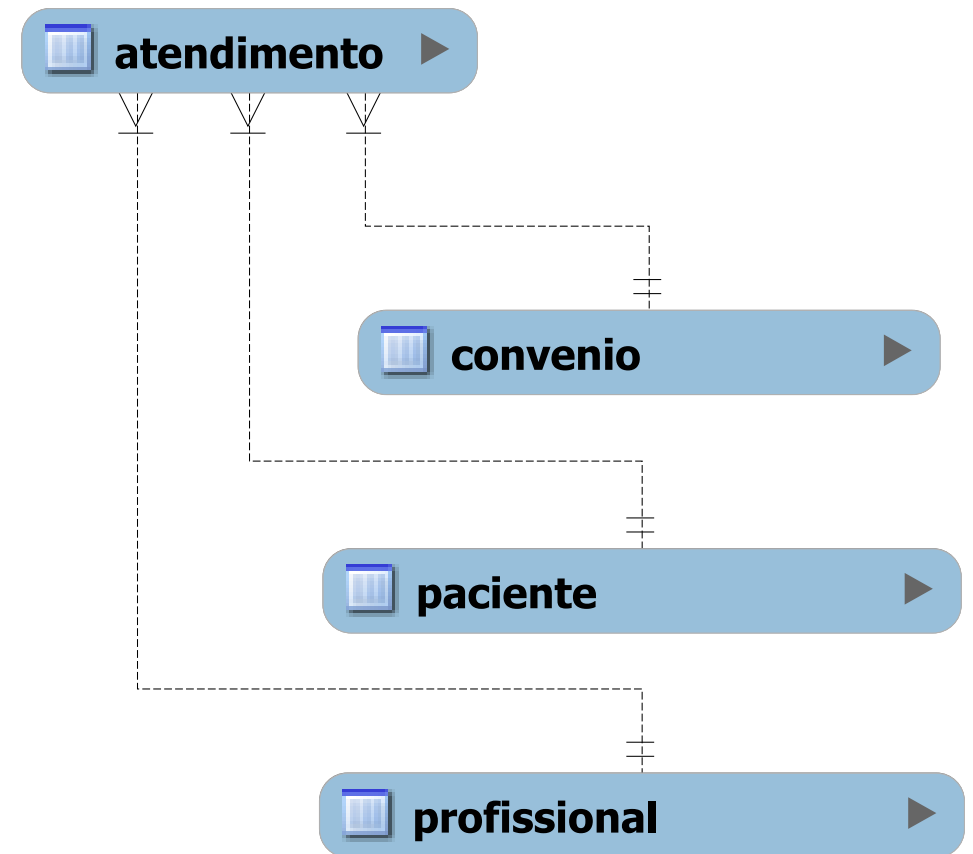


## @ManyToMany



# Relacionamento entre entidades

```
1. @Entity
2. public class Atendimento {
3.     @ManyToOne(optional = false)
4.     private Profissional profissional;
5.     @ManyToOne
6.     private Convenio convenio;
7.     @ManyToOne(optional = false)
8.     private Paciente paciente;
9. }
```



# Spring Data

- Spring Data fornece um **mecanismo de acesso a dados** de vários tipos diferentes de banco de dados, incluindo relacionais (**JPA**), orientado a documento (MongoDB), grafos (Neo4j) e outros.
- **Spring Data JPA facilita a implementação de repositórios de acesso a dados baseados em JPA**, por meio de uma interface que fornece desde recursos básicos para operações CRUD até funcionalidades avançadas de paginação, consultas customizadas, dentre outros.
- **Dispensa a criação de DAOs** e implementação de métodos específicos para acessos ao banco de dados.

# Repositórios e métodos de consulta

```
1. public interface UnidadeRepository extends JpaRepository<Unidade, Long> {  
2.     @Query("SELECT u FROM Unidade u WHERE u.nome LIKE %?1%" +  
3.         " OR u.endereco LIKE %?1%")  
4.     List<Unidade> findByAll(String termoBusca);  
5.     List<Unidade> findByNome(String nome);  
6.     List<Unidade> findByEndereco(String endereco);  
7.     List<Unidade> findByNomeAndEndereco(String nome, String endereco);  
8. }
```

**Métodos de consulta:** <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details>

**Palavras-chave:** <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repository-query-keywords>

# Injeção de dependência no Spring

```
1. @Controller
2. public class ExemploController {
3.     private final AtendimentoRepository repo;
4.     @Autowired
5.     public ExemploController(AtendimentoRepository repo) {
6.         this.repo = repo;
7.     }
8. }
```



**API**

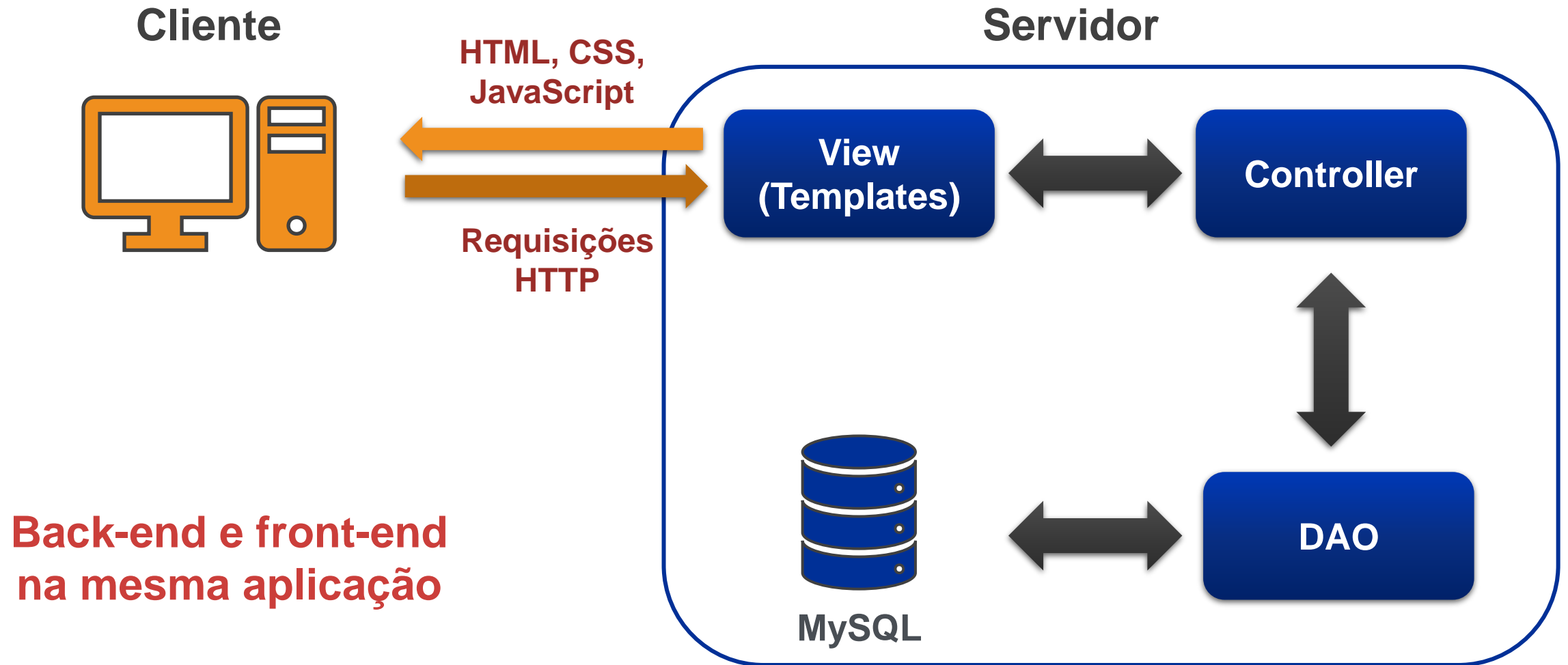
# Introdução à arquitetura REST e APIs

- A arquitetura **REST** (***RE**presentational **State** **T**ransfer*) **define um conjunto de restrições** para a criação serviços web.
- Diferente de uma aplicação baseada em RPC (*Remote Procedure Call*), REST não define acesso a métodos/procedimentos, mas sim à recursos (objetos, JSON, XML, etc.), por meio de protocolos como o HTTP e identificadores (URLs) .

# Introdução à arquitetura REST e APIs

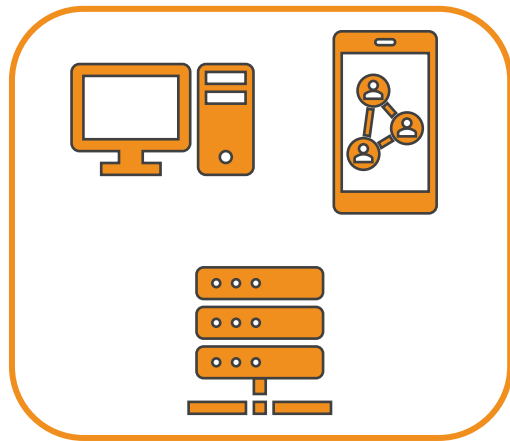
- Uma **API** (*A*pplication *P*rogramming *I*nterface) é um **conjunto de definições e protocolos** para construção e integração de aplicações, e poder ser baseada na arquitetura REST.
- Por meio de uma API é possível trocar informações com outros softwares **sem precisar saber como eles foram implementados**.
- Recursos de uma API podem ser acessados por meio dos **endpoints** (URLs).
  - Exemplo: <http://localhost:8080/atendimento/> fornece acesso a lista de atendimentos.

# Arquitetura de uma aplicação web

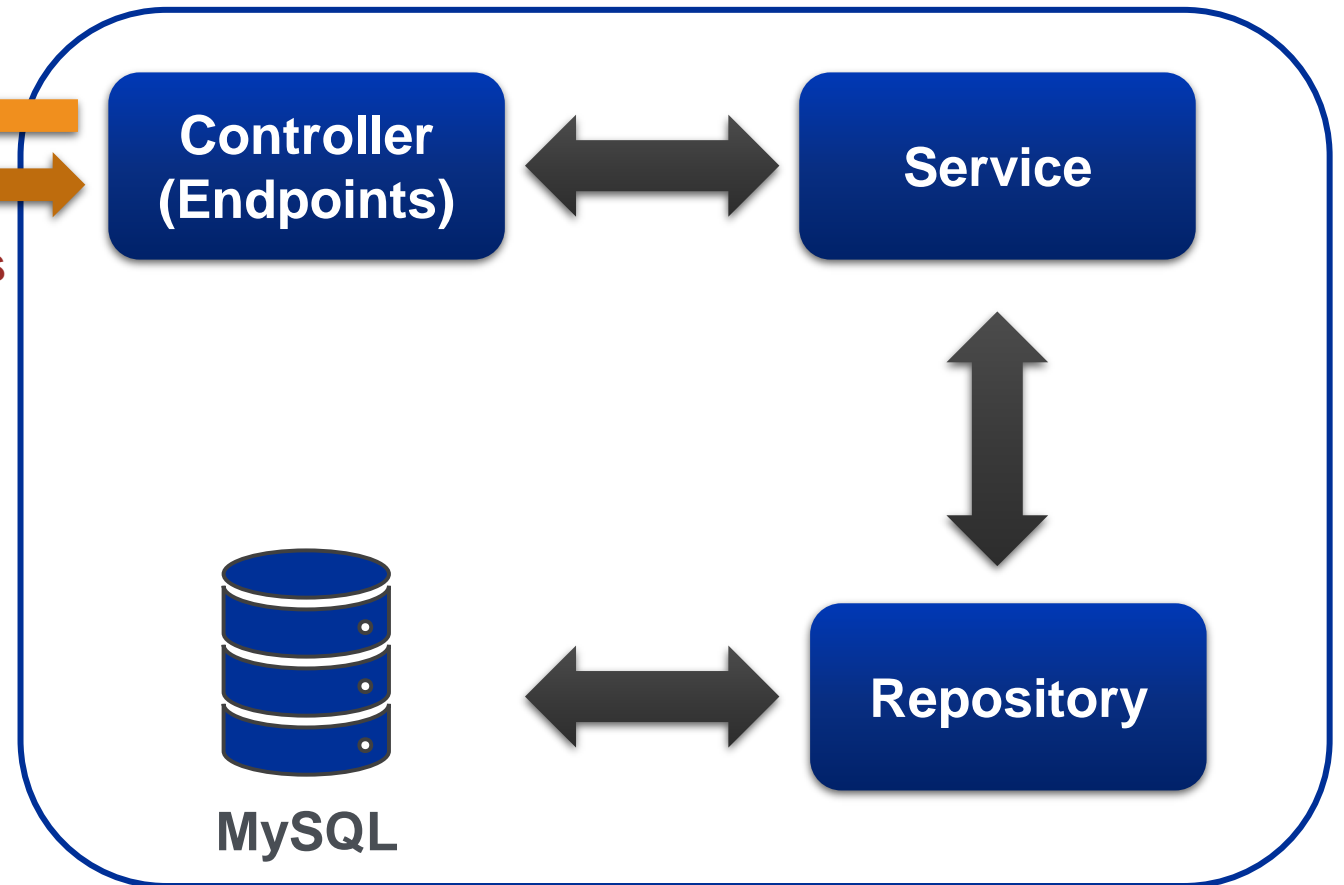


# Arquitetura de uma API

Cliente (Front-end)



Servidor (Back-end)



JSON

Requisições  
HTTP

Back-end e front-end  
separados

# Camada de serviço é necessária?

- Nem sempre é necessária, especialmente em aplicações simples.
- Separação de responsabilidades:
  - **Controller**: expõe os **endpoints**.
  - **Service**: lógica de negócio.
  - **Repository**: persistência e acesso aos dados.
- **Lógica de negócios** pode começar simples (operações CRUD), mas **pode ficar mais complexa**.

# Mapeamento de requisições HTTP

- No Spring, o **@RequestMapping** é utilizado para **mapear requisições HTTP** feitas para URLs específicas, atribuindo a um método ou classe a tarefa de manipular estas requisições.

```
1. @RestController
2. @RequestMapping("/atendimento")
3. public class AtendimentoController implements ICrudController<Atendimento> {
4.     @RequestMapping(value =("/{id}", method = RequestMethod.GET)
5.     public ResponseEntity<Atendimento> getById(@PathVariable("id") Long id) {
6.         Atendimento registro = servico.getById(id);
7.         return new ResponseEntity<>(registro, HttpStatus.OK);
8.     }
9. }
```

# Mapeamento de requisições HTTP

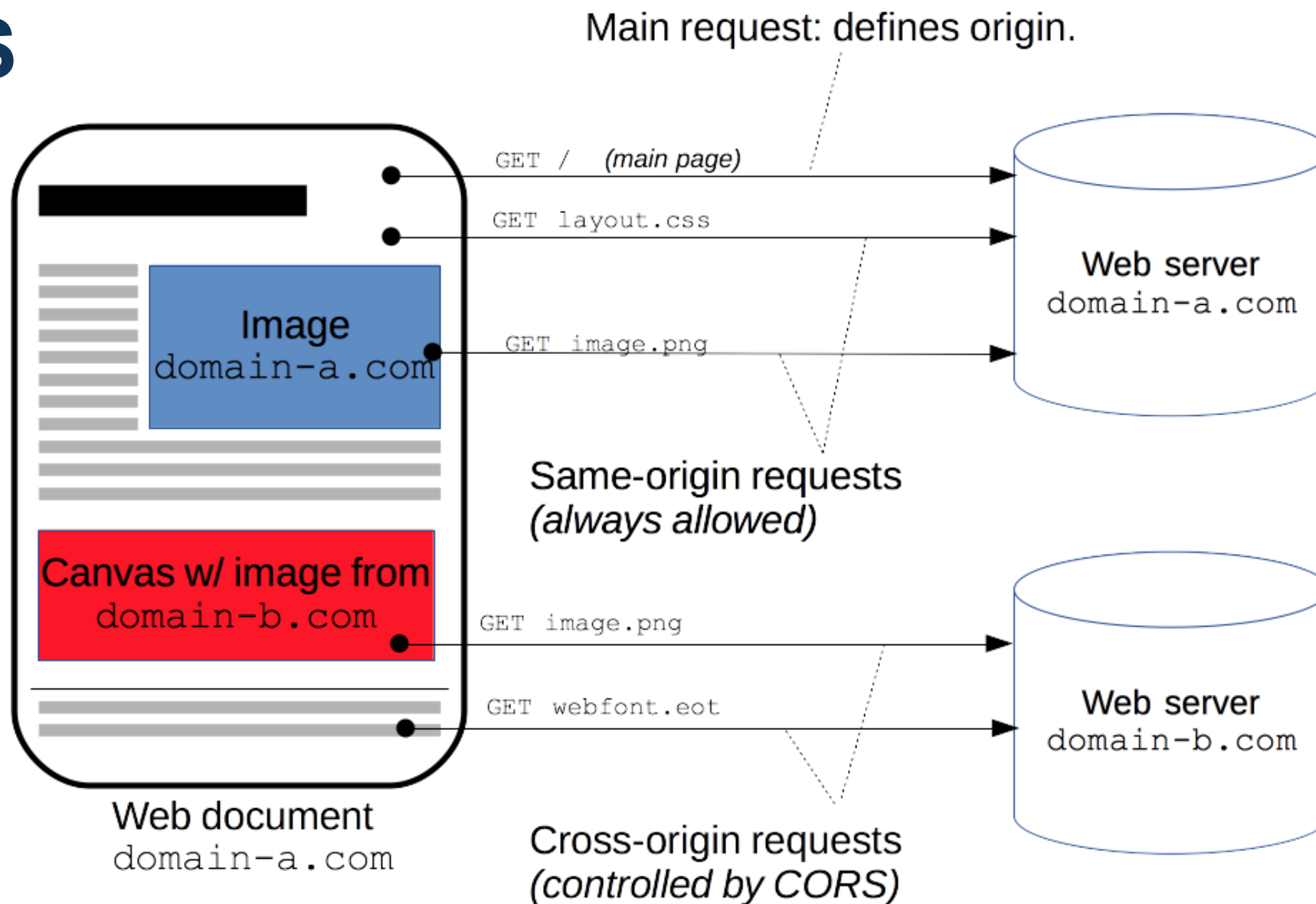
Anotação	CRUD	Atalho para...
@GetMapping	READ	@RequestMapping(method = RequestMethod.GET)
@PostMapping	CREATE	@RequestMapping(method = RequestMethod.POST)
@PutMapping	UPDATE (Completo)	@RequestMapping(method = RequestMethod.PUT)
@DeleteMapping	DELETE	@RequestMapping(method = RequestMethod.DELETE)
@PatchMapping	UPDATE (Parcial)	@RequestMapping(method = RequestMethod.PATCH)



# CORS

- **CORS** (**C**ross-**O**rigin **R**esource **S**haring) é um **mecanismo de segurança** que gerencia requisições entre domínios, **impedindo que scripts executem códigos maliciosos**.
- Uma requisição entre domínios é uma solicitação HTTP feita pelo navegador do **dominio-a.com** para o **dominio-b.com** por meio requisições assíncronas (AJAX).
- **Origem** é a combinação do **protocolo + porta + domínio da solicitação**.
  - **http://dominio-a.com:9000/** é diferente de **https://dominio-a.com:9000/**
- CORS é um **padrão em todos os navegadores modernos**.

# CORS



Fonte: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS>

# Requisições simples

[https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS#requisi%C3%A7%C3%B5es\\_simples](https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS#requisi%C3%A7%C3%B5es_simples)

Browser (<https://www.site.com>)

## Request:

GET <https://www.api.com?q=test>  
origin: <https://www.site.com>

## Response:

HTTP/1.1 200 OK  
access-control-allow-origin: <https://www.site.com>

Server (<https://www.api.com>)

Fonte: <https://www.baeldung.com/cs/cors-preflight-requests>

# Requisições com pré-envio

[https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS#requisi%C3%A7%C3%B5es\\_com\\_pr%C3%A9-envio](https://developer.mozilla.org/pt-BR/docs/Web/HTTP/CORS#requisi%C3%A7%C3%B5es_com_pr%C3%A9-envio)

Browser (<https://www.site.com>)

## Pre-flight Request:

OPTIONS <https://www.api.com?q=test>  
access-control-request-method: GET  
access-control-request-headers: custom-header, ...  
origin: <https://www.site.com>

## Pre-flight Response:

HTTP/1.1 204 No Content  
  
access-control-allow-origin: <https://www.site.com>  
access-control-allow-methods: GET  
access-control-allow-headers: custom-header, accept, ...  
access-control-max-age: 6000

## Request:

GET <https://www.api.com?q=test>  
  
origin: <https://www.site.com>  
custom-header: test

## Response:

HTTP/1.1 200 OK  
  
access-control-allow-origin: <https://www.site.com>

Server (<https://www.api.com>)

Fonte: <https://www.baeldung.com/cs/cors-preflight-requests>

# CORS

```
1.  @Bean
2.  public CorsFilter corsFilter() {
3.      CorsConfiguration corsConfig = new CorsConfiguration();
4.      corsConfig.setAllowedOrigins(Arrays.asList("http://localhost:5500"));
5.      corsConfig.setAllowedMethods(Arrays.asList("*"));
6.      corsConfig.setAllowedHeaders(Arrays.asList("*"));
7.      UrlBasedCorsConfigurationSource configSource = new UrlBasedCorsConfigurationSource();
8.      configSource.registerCorsConfiguration("/**", corsConfig);
9.      return new CorsFilter(configSource);
10. }
```

# Habilitar SSL no Spring Boot

- **Criar certificado**

```
keytool -genkeypair -keyalg RSA -keysize 2048 -storetype PKCS12 -keystore  
cert.p12 -validity 3650
```

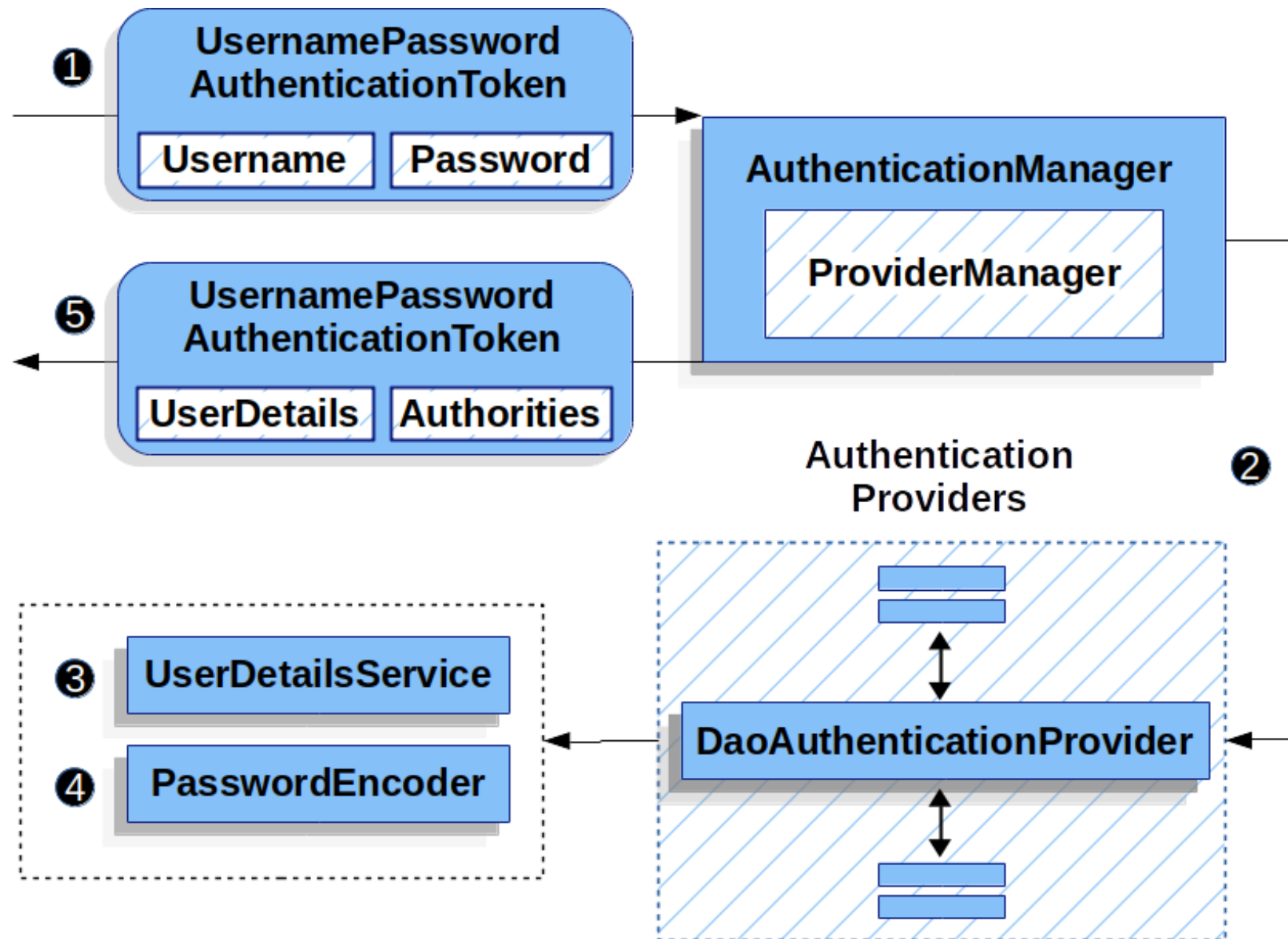
- O arquivo **cert.p12** deve ser colocado no diretório **src/main/resources/** do projeto.

- **application.properties**

- `server.ssl.key-store=classpath:cert.p12`
- `server.ssl.key-store-password=webacademy`
- `server.ssl.key-store-type=PKCS12`

# Autenticação de usuários

**Spring Security:** fornece recursos para **autenticação**, **autorização de acesso** e proteção contra ataques comuns.



Fonte: <https://docs.spring.io/spring-security/site/docs/5.5.x-SNAPSHOT/reference/html5/>

**Fim!**





# Referências

- DEITEL, Paul; DEITEL, Harvey. **Java: Como Programar**. 10. ed. São Paulo: Pearson, 2016. 968 p. MOZILLA (ed.). **MDN Web Docs: Aprendendo desenvolvimento web**. [S. l.], 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn>.
- MARCO TULIO VALENTE. **Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade**, 2020. Disponível em: <https://engsoftmoderna.info/>
- MOZILLA (ed.). **MDN Web Docs: Aprendendo desenvolvimento web**. [S. l.], 2022. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn>.
- SPRING (ed.). **Spring Boot Reference Documentation**. [S. l.], 2022. Disponível em: <https://docs.spring.io/spring-boot/docs/current/reference/html/index.html>.
- WALLS, Craig. **Spring in Action**. 6. ed. Shelter Island: Manning, 2021. 520 p.