

TÓPICO 09 - SPRING WEB

Disciplina de Backend - Professor Ramon Venson - SATC 2026.2



O que é Spring?

O spring é um ecossistema de **frameworks para desenvolvimento** Java, com foco em produtividade, organização e baixo acoplamento.

Pode ser usado para criar aplicações desktop, web, mobile, microserviços, **APIs REST**, integração com banco de dados e muito mais.

Spring Framework x Spring Boot

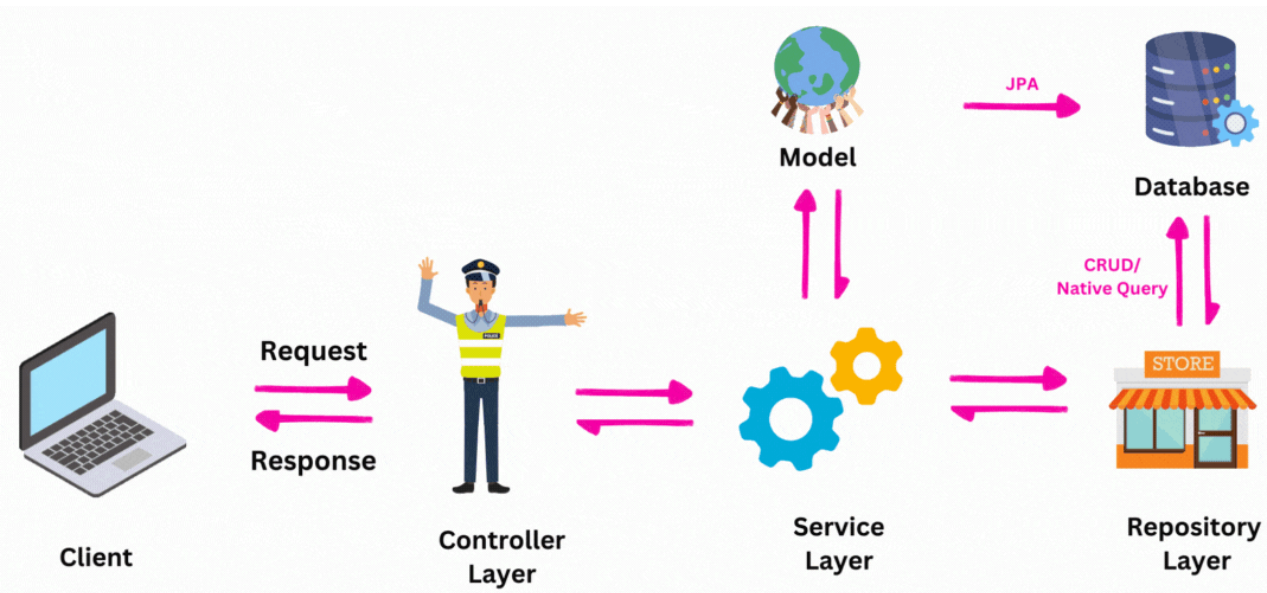
- Spring Framework fornece os módulos e a base do ecossistema
- Spring Boot simplifica a configuração e a inicialização do projeto
- Com Spring Boot, o servidor web pode vir embutido
- O objetivo é reduzir configurações manuais e acelerar o desenvolvimento



Por que usar Spring Web?

O Spring Web é um módulo do Spring Framework que facilita a criação de **APIs REST** e aplicações web.

Ele abstrai as complexidades do **protocolo HTTP**, permitindo que você se concentre na lógica de negócio.



Spring Initializr

O **Spring Initializr** é uma ferramenta online para gerar projetos Spring Boot.

Para começar um novo projeto, basta selecionar as dependências desejadas e baixar o arquivo zip.

The current version of Spring Boot changes regularly. Just choose the latest release (but not snapshot).

Click on 'Add dependencies', type 'Web' in the search box, then click on the dependency 'Spring Web' to select it.

The screenshot shows the Spring Initializr web application interface. The header includes the Spring logo and the text 'spring initializr'. The main content area is divided into several sections: 'Project' with radio buttons for 'Maven Project' (selected) and 'Gradle Project'; 'Language' with radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'; 'Spring Boot' with radio buttons for versions '2.3.0 M4', '2.3.0 (SNAPSHOT)', '2.2.7 (SNAPSHOT)', '2.2.6' (selected), '2.1.14 (SNAPSHOT)', and '2.1.13'; 'Project Metadata' with input fields for 'Group' (com.example), 'Artifact' (demo), 'Name' (demo), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.demo); 'Packaging' with radio buttons for 'Jar' (selected) and 'War'; and 'Java' with radio buttons for versions '14', '11', and '8' (selected). On the right, the 'Dependencies' section has an 'ADD DEPENDENCIES...' button and a search box. Below the search box, the 'Spring Web' dependency is listed with a 'WEB' tag and a description: 'Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.' At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'. A Twitter icon is visible in the bottom left corner.

Recomendação de Configuração

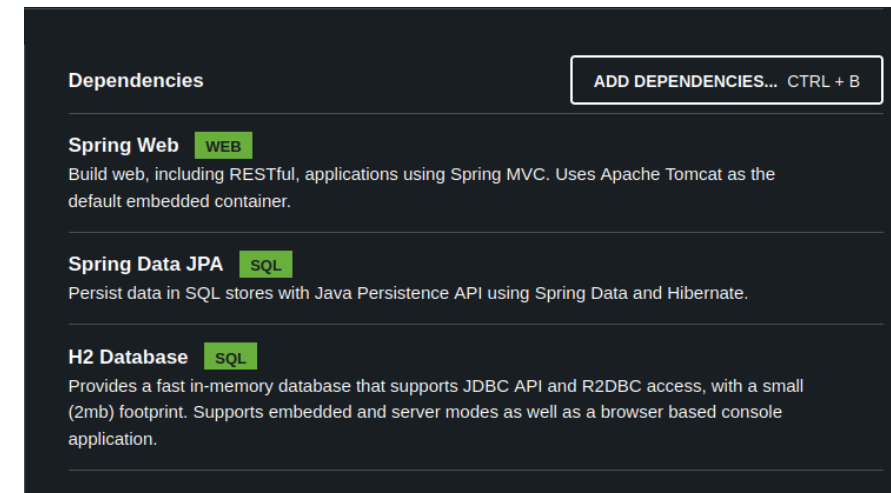
As configurações recomendadas para o Spring Initializr nessa disciplina são:

- Project : Maven
- Language : Java
- Spring Boot : versão mais estável e atual
- Project Metadata
 - Group : nome de domínio reverso (ex.: com.exemplo)
 - Artifact : nome do projeto (ex: loja-api)
 - Packaging : Jar
 - Configuration : YAML
 - Java : versão instalada na máquina

Dependências

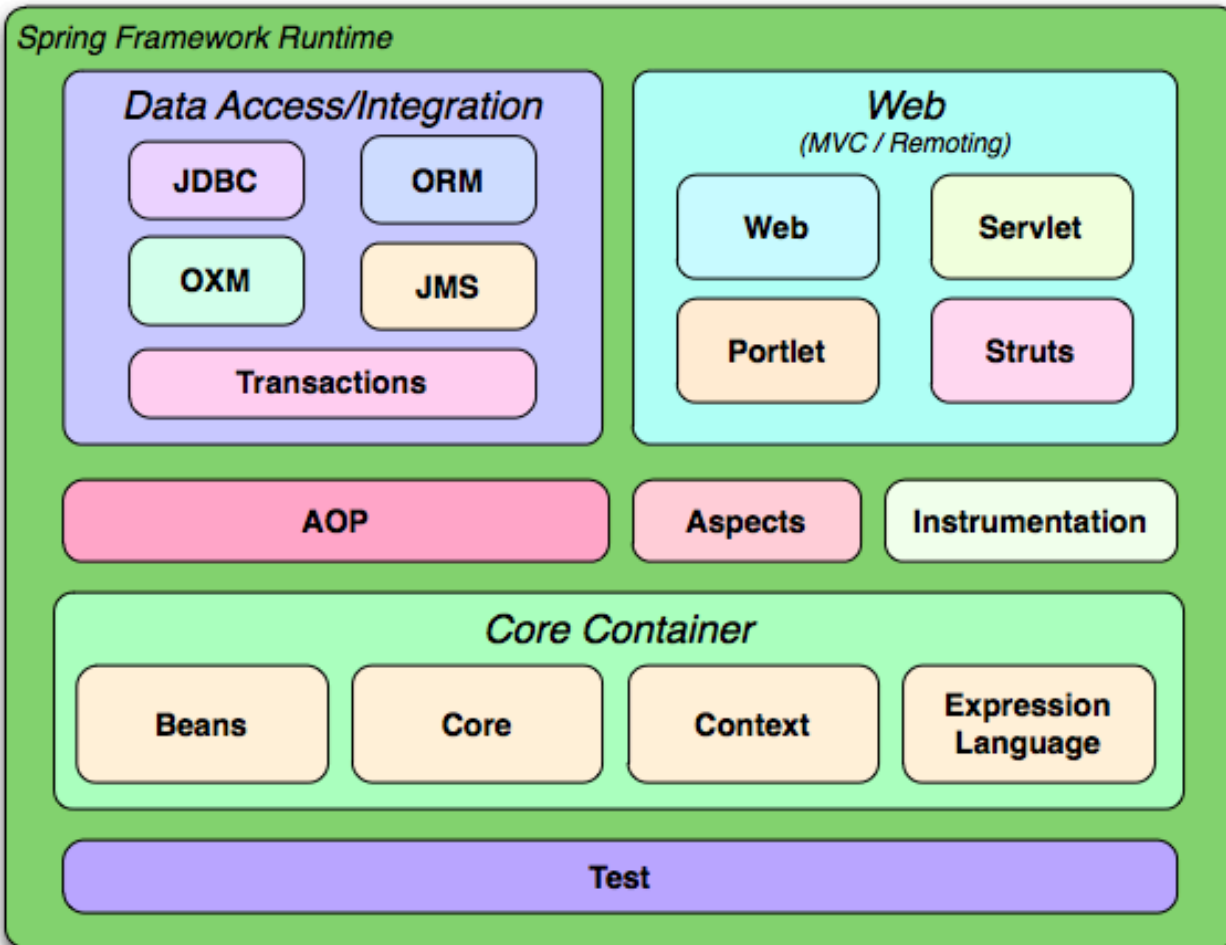
No processo de criação, podemos selecionar algumas dependências que serão utilizadas no projeto.

- `spring-boot-starter-web` : para criar APIs REST
- `spring-boot-starter-validation` : para validacao de dados
- `spring-boot-starter-data-jpa` : para persistência de dados
- `spring-boot-devtools` : para reinicio automatico
- `h2` : banco de dados em memória para testes



Outras dependências opcionais incluem:

- `spring-boot-starter-test` : para testes unitários e de integração
- `lombok` : para reduzir boilerplate de código
- `spring-boot-starter-security` : para autenticação e autorização
- `thymeleaf` : para renderização de páginas HTML
- `flyway` ou `liquibase` : para versionamento de banco de dados
- `spring-boot-starter-amqp` : para integração com RabbitMQ
- `spring-ai-starter-model-ollama` : para integração com modelos de linguagem da Ollama



Após a criação do projeto, você pode fazer o download do arquivo zip, descompactá-lo e abrir a pasta no seu editor de código favorito (como VS Code ou IntelliJ IDEA).

Estrutura basica de projeto

```
src/main/java/com/exemplo/app/  
    controller/  
    service/  
    repository/  
    model/  
    dto/  
Application.java
```

- Cada camada possui uma responsabilidade bem definida
- Isso ajuda na manutencao, teste e evolucao da aplicacao

Classe principal

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

- `@SpringBootApplication` ativa configurações automáticas do projeto
- `SpringApplication.run(...)` inicializa o contexto e sobe a aplicação

Controllers

`Controllers` são classes responsáveis por receber requisições HTTP e devolver respostas.

```
@RestController  
@RequestMapping("/produtos")  
public class ProdutoController {  
}
```

- `@RestController` indica uma API REST
- `@RequestMapping` define o caminho base da rota

Mapeando endpoints

```
@GetMapping
public List<Produto> listar() {
    return service.listar();
}

@PostMapping
public Produto criar(@RequestBody Produto produto) {
    return service.criar(produto);
}
```

- `@GetMapping` atende requisições GET
- `@PostMapping` atende requisições POST
- `@RequestBody` converte JSON em objeto Java

Parametros da requisicao

```
@GetMapping("/{id}")
public Produto buscarPorId(@PathVariable UUID id) {
    return service.buscarPorId(id);
}

@GetMapping("/buscar")
public List<Produto> buscarPorNome(@RequestParam String nome) {
    return service.buscarPorNome(nome);
}
```

- `@PathVariable` le partes da URL
- `@RequestParam` le parametros da query string

Retorno padrão

Por padrão, basta retornar o objeto (ou a lista) e o Spring monta a resposta:

```
@GetMapping
public List<Produto> listar() {
    return service.listar();
}

@GetMapping("/{id}")
public Produto buscarPorId(@PathVariable UUID id) {
    return service.buscarPorId(id);
}

@DeleteMapping("/{id}")
public void remover(@PathVariable UUID id) {
    service.remover(id);
}
```

Injecao de dependencia

No Spring, objetos podem ser gerenciados pelo container e injetados automaticamente.

```
@Service
public class ProdutoService {
    private final ProdutoRepository repository;

    public ProdutoService(ProdutoRepository repository) {
        this.repository = repository;
    }
}
```

- `@Service` marca uma classe de regra de negocio
- O Spring fornece a dependencia pelo construtor

Camadas da aplicacao

- `Controller` : recebe a requisicao e monta a resposta
- `Service` : concentra regras de negocio
- `Repository` : acessa banco de dados
- `Model` ou `Entity` : representa os dados da aplicacao
- `DTO` : estrutura dados de entrada e saida

Validacao de dados

```
public class ProdutoDTO {  
    @NotBlank  
    private String nome;  
  
    @Positive  
    private BigDecimal preco;  
}
```

```
public Produto criar(@Valid @RequestBody ProdutoDTO dto) {  
    return service.criar(dto);  
}
```

- `@Valid` ativa a validacao do objeto recebido
- Anotacoes como `@NotBlank` e `@Positive` evitam dados invalidos

JSON automaticamente

Quando um metodo retorna um objeto Java em um `@RestController`, o Spring normalmente serializa a resposta em JSON.

```
@GetMapping("/{id}")  
public ProdutoDTO detalhar(@PathVariable UUID id) {  
    return service.detalhar(id);  
}
```

Resposta esperada:

```
{  
  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  
  "nome": "Teclado",  
  "preco": 199.90  
}
```

Tratamento de excecoes

O Spring permite centralizar o tratamento de erros com `@RestControllerAdvice` :

```
@RestControllerAdvice
public class ApiExceptionHandler {
    @ExceptionHandler(ProdutoNaoEncontradoException.class)
    public String tratarNaoEncontrado() {
        return "Produto nao encontrado";
    }
}
```

- `@RestControllerAdvice` centraliza tratamento de erros
- Evita repetir blocos `try-catch` em todos os controllers

O controle do **status** e dos **cabeçalhos** da resposta de erro será aprofundado no

Tópico 11.

Configurações

O arquivo de configuração de uma aplicação Spring Boot pode ser feito em `application.properties` ou `application.yml`.

Esse arquivo é importante para definir propriedades como porta do servidor, conexão com banco de dados, logging e outras configurações.

Exemplo do `application.properties` :

```
spring.application.name=loja-api  
server.port=8080  
spring.datasource.url=jdbc:h2:mem:teste
```

Exemplo do `application.yml` :

```
spring:  
  application:  
    name: loja-api  
  datasource:  
    url: jdbc:h2:mem:teste  
server:  
  port: 8080
```

Fluxo de uma requisicao

1. Cliente envia uma requisicao HTTP
2. O `Controller` recebe a rota correspondente
3. O `Service` aplica as regras de negocio
4. O `Repository` consulta ou persiste dados
5. A resposta volta como JSON com um status HTTP

Exemplo de rota REST

```
POST /produtos HTTP/1.1  
Content-Type: application/json
```

```
{  
  "nome": "Mouse",  
  "preco": 89.90  
}
```

Resposta:

```
HTTP/1.1 201 Created  
Content-Type: application/json
```

Boas Práticas

- Separar responsabilidade entre camadas
- Evitar regra de negocio diretamente no controller
- Usar DTOs em vez de expor entidades diretamente
- Validar entradas recebidas pela API
- Retornar codigos HTTP coerentes com o resultado

O que aprendemos hoje

- O papel do Spring Web no ecossistema Java
- Diferença entre Spring Framework e Spring Boot
- Criação de rotas com anotações
- Uso de JSON, validação e retorno de objetos
- Organização em camadas para APIs REST