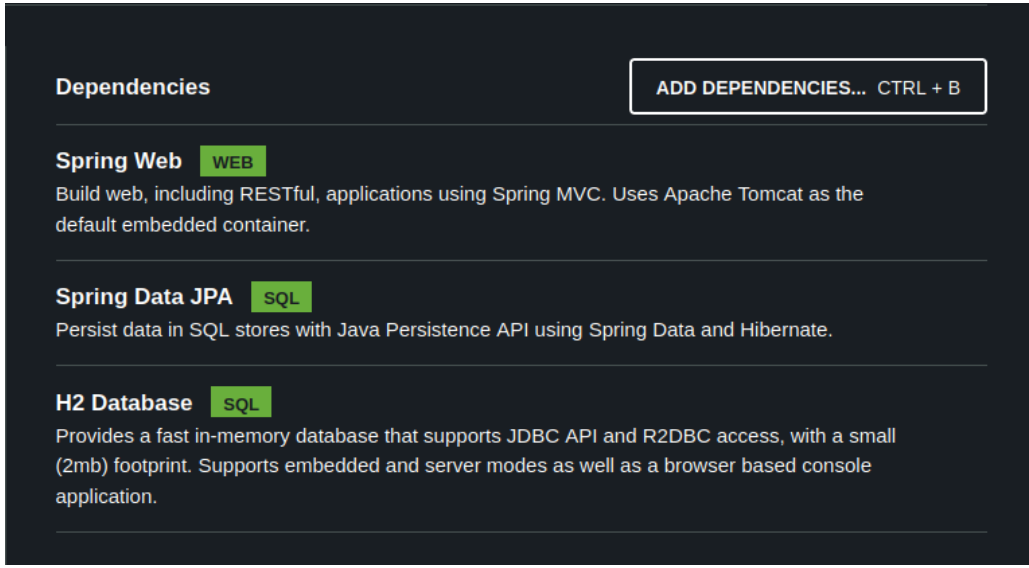
TÓPICO 08 - SPRING WEB

Disciplina de Backend - Professor Ramon Venson - SATC 2026.1

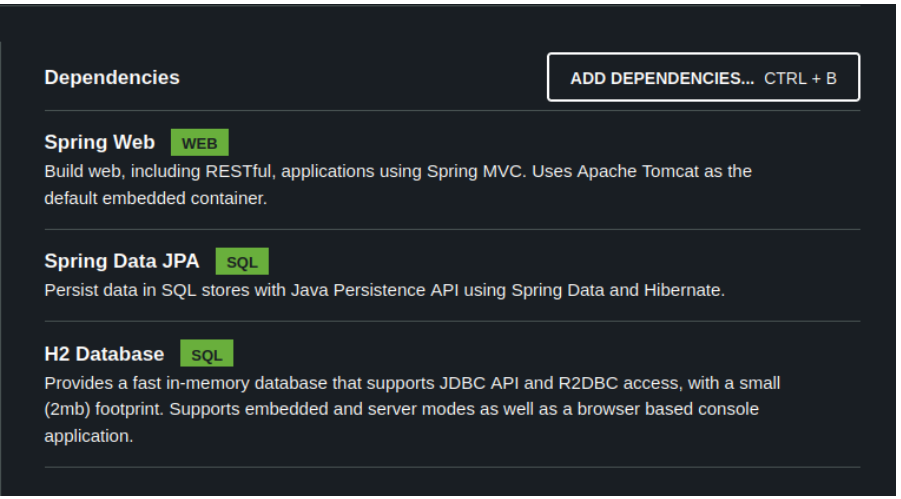
O que é Spring?

- Ecossistema de frameworks para desenvolvimento Java
- Foco em produtividade, organizacao e baixo acoplamento
- Muito usado para APIs REST, sistemas corporativos e microsservicos
- O modulo `Spring Web` facilita a criacao de aplicacoes HTTP



Spring Framework x Spring Boot

- `Spring Framework` fornece os modulos e a base do ecossistema
- `Spring Boot` simplifica a configuracao e a inicializacao do projeto
- Com Spring Boot, o servidor web pode vir embutido
- O objetivo e reduzir configuracoes manuais e acelerar o desenvolvimento



Por que usar Spring Web?

- Criacao de rotas HTTP com anotacoes
- Conversao automatica entre JSON e objetos Java
- Integracao com validacao, injecao de dependencia e tratamento de erros
- Estrutura adequada para separar controller, service e repository

Dependencias comuns

Ao criar um projeto com Spring Boot, algumas dependencias sao muito comuns:

- `spring-boot-starter-web`
- `spring-boot-starter-validation`
- `spring-boot-starter-data-jpa`
- `spring-boot-devtools`

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 configuracoes automaticas do projeto
- `SpringApplication.run(...)` inicializa o contexto e sobe a aplicacao

Controllers

`Controllers` sao classes responsaveis por receber requisicoes HTTP e devolver respostas.

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

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

Mapeando endpoints

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

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

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

Parametros da requisicao

```
@GetMapping("/{id}")
public Produto buscarPorId(@PathVariable Long 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

Respostas HTTP

```java
@DeleteMapping("/{id}")
public ResponseEntity<Void> remover(@PathVariable Long id) {
    service.remover(id);
    return ResponseEntity.noContent().build();
}
```

- `ResponseEntity` permite controlar status, headers e corpo
- Exemplo comum: `200 OK`, `201 Created`, `204 No Content`, `404 Not Found`

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

```java
public class ProdutoDTO {
    @NotBlank
    private String nome;

    @Positive
    private BigDecimal preco;
}
```

```java
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 Long id) {
    return service.detalhar(id);
}
```

Resposta esperada:

```
{
  "id": 10,
  "nome": "Teclado",
  "preco": 199.90
}
```

Tratamento de excecoes

```
@RestControllerAdvice
public class ApiExceptionHandler {
    @ExceptionHandler(ProdutoNaoEncontradoException.class)
    public ResponseEntity<String> tratarNaoEncontrado() {
        return ResponseEntity.status(404).body("Produto nao encontrado");
    }
}
```

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

Configuracoes

Arquivos comuns de configuracao:

- `src/main/resources/application.properties`
- `src/main/resources/application.yml`

Exemplo:

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

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
- Diferenca entre Spring Framework e Spring Boot
- Criacao de rotas com anotacoes
- Uso de JSON, validacao e `ResponseEntity`
- Organizacao em camadas para APIs REST