# Aula sobre Spring Boot

## 1. Parte teórica – O que é Spring Boot? (20–25 min)

### 1.1. Problema que o Spring Boot resolve

- Antes do Spring Boot, para criar uma aplicação Java web você precisava:
    - Configurar servidor (Tomcat, WildFly, etc.)
    - Criar WAR, configurar XML, datasources, bibliotecas, etc.
    - Muito “trabalho chato” antes mesmo de escrever a regra de negócio.
- Isso tornava o **início do projeto lento**, complexo, difícil para iniciantes.

O Spring Boot veio para:

- **Simplificar a configuração**
- **Acelerar o início de novos projetos**
- **Padronizar projeto Java moderno** para web/api.

Use essa frase em aula:

> "Com Spring Boot, você deixa de perder tempo configurando infraestrutura e passa a focar na regra de negócio."

### 1.2. O que é o Spring Boot (conceito)

Pontos-chave para você explicar:

- É um **framework** em cima do **Spring Framework**.
- Facilita a criação de:
    - APIs REST
    - Aplicações web
    - Sistemas com banco de dados, segurança, etc.
- Principais características:
    - **Autoconfiguração** (auto-configuration)
    - **Opinionated defaults** (padrões já pensados)

- **Embedded server** (servidor embarcado, ex: Tomcat já dentro do JAR)
- **Starter dependencies** (pacotes prontos de dependências)

## 1.3. Onde usar Spring Boot (casos de uso)

Dê exemplos práticos:

- Criar **API REST** para um frontend Angular/React/Vue.
- Backend para **aplicativos mobile** (Android/iOS).
- Sistemas internos da empresa (ERP, CRM, etc.).
- Microsserviços que se comunicam entre si.
- Integração com outros sistemas (gateways, filas, etc.).

## 1.4. Vantagens e benefícios

1. **Produtividade alta**
   - Start muito rápido (projeto sobe em minutos).
   - Muitas coisas já prontas: logs, erros padrão, estrutura de pastas.
2. **Menos configuração manual**
   - Menos XML.
   - Muitas configurações feitas automaticamente baseadas nas dependências.
3. **Servidor embarcado**
   - Você gera um **.jar executável**.
   - Para subir o sistema: `java -jar minha-app.jar`.
   - Não precisa instalar/administrar um Tomcat externo.
4. **Integração fácil com banco de dados**
   - Uso comum com Spring Data JPA, Hibernate, etc.
   - Mapeamento de entidades de forma simples.
5. **Grande comunidade e mercado**
   - Muito material, tutoriais, cursos, vagas de emprego.

    - Muitas empresas já padronizaram em Spring Boot.
6. **Escalável e moderno**
    - Suporta arquitetura de microsserviços.
    - Integra com Docker, Kubernetes, Cloud, etc.

## 1.5. Conceitos básicos que o aluno precisa saber

- **Projeto Spring Boot** é basicamente:
    - Uma aplicação Java com uma classe principal (método `main`)
    - Anotações para configurar comportamento.
- Conceitos que vão aparecer:
    - `@SpringBootApplication`
    - `@RestController`
    - `@GetMapping`, `@PostMapping`, etc.
    - `application.properties` ou `application.yml` para configurações.

## 2.1. Criando o projeto (Spring Initializr)

Explique o passo a passo (você pode mostrar na tela):

1. Acessar o site do Spring Initializr:
    - https://start.spring.io (você pode comentar, não precisa citar versão específica).
2. Configurações principais:
    - **Project**: Maven
    - **Language**: Java
    - **Spring Boot**: usar a versão estável sugerida pelo site.
    - **Project Metadata**:
        - Group: `com.seu.nome` (ex: `com.alex.dev`)
        - Artifact: `primeiro-springboot`
        - Name: `primeiro-springboot`

        - Package name: deixar o padrão (`com.alex.dev.primeiro_springboot` ou similar).
        - Packaging: `Jar`
        - Java: versão compatível com sua JDK (11, 17, etc.)
    - Dependências (botão “Add Dependencies”):
        - `Spring Web`
3. Baixar o arquivo `.zip` e extrair.
4. Importar no IDE:
    - No IntelliJ: `File > Open` e selecione a pasta.
    - No Eclipse/STS: `Import > Existing Maven Project`.

## 2.2. Estrutura do projeto (explicar pastas principais)

Mostre a estrutura típica:

- `src/main/java`
    - `com.exemplo.primeirospringboot`
        - `PrimeiroSpringbootApplication.java` (classe principal)
- `src/main/resources`
    - `application.properties`
- `pom.xml`
- `pom.xml`: arquivo do Maven com as dependências.
- `application.properties`: arquivo de configurações da aplicação.

- Pacote principal: onde fica a classe com `@SpringBootApplication`.

## 2.3. Classe principal – ponto de entrada

```
java

package com.exemplo.primeirospringboot;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PrimeiroSpringbootApplication {

    public static void main(String[] args) {
        SpringApplication.run(PrimeiroSpringbootApplication.class,
args);
    }

}
```

- `@SpringBootApplication`:
  - Marca essa classe como o ponto de partida da aplicação.
  - Habilita:
    - **Auto-configuration**
    - **Component scan** (varre o pacote em busca de componentes Spring)
- `main`:
  - Ponto de entrada padrão Java.
  - `SpringApplication.run(...)`:
    - Sobe o contexto do Spring.
    - Sobe o servidor embarcado (Tomcat por padrão).
    - Inicializa toda a aplicação.

## 2.4. Primeiro endpoint REST – “Hello, Spring Boot”

Vamos criar um controlador simples.

Crie um novo pacote:

- `com.exemplo.primeirospringboot.controller`

E dentro, uma classe:

```
java

package com.exemplo.primeirospringboot.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/hello")
    public String hello() {
        return "Olá, Spring Boot!";
    }
}
```

**Explanação do código:**

- `@RestController`:
  - Diz que essa classe é um **controller REST**.
  - Combina `@Controller` + `@ResponseBody` (ou seja, o retorno vai direto no corpo da resposta HTTP).
- `@GetMapping("/hello")`:
  - Indica que o método `hello()` responde à requisição HTTP GET no caminho `/hello`.
- `public String hello()`:
  - Retorna uma String.
  - Quando o usuário acessar `http://localhost:8080/hello`, verá `Olá, Spring Boot!`.

Você pode comentar:

> "Esse já é o nosso primeiro endpoint. Se você rodar a aplicação agora e acessar /hello, já tem um serviço web funcionando."

### 2.5. Executando o projeto

Explique como rodar:

- Pelo IDE:
    - Clique com botão direito na classe `PrimeiroSpringbootApplication`.
    - `Run As > Java Application` (Eclipse) ou `Run 'PrimeiroSpringbootApplication'` (IntelliJ).
- Pelo Maven (se quiser mostrar):
    - `mvn spring-boot:run`

Quando subir, mostrar no console algo como:

- Mensagem de que o Tomcat iniciou na porta 8080.
- Mensagem `Started PrimeiroSpringbootApplication in X seconds`.

Depois:

- Abra o navegador:
    - `http://localhost:8080/hello`

Mostre o resultado na tela.

## 2.6. Segundo exemplo – Retornando um objeto JSON

1. Crie um pacote `model`:

```
java

package com.exemplo.primeirospringboot.model;

public class Cliente {

    private Long id;
    private String nome;
    private String email;

    public Cliente(Long id, String nome, String email) {
        this.id = id;
        this.nome = nome;
        this.email = email;
    }

    public Long getId() {
        return id;
    }

    public String getNome() {
        return nome;
    }

    public String getEmail() {
        return email;
    }
}
```

2. Altere o controller para retornar um `Cliente`:

```
java

package com.exemplo.primeirospringboot.controller;

import com.exemplo.primeirospringboot.model.Cliente;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ClienteController {
```

```
    @GetMapping("/cliente-exemplo")
    public Cliente obterClienteExemplo() {
        return new Cliente(1L, "João da Silva",
"joao.silva@example.com");
    }
}
```

**Explicação:**

- Mesmo sem configurar nada extra, o Spring Boot:
  - Converte automaticamente o objeto Java (`Cliente`) em **JSON**.
  - Isso porque o starter `spring-boot-starter-web` já traz o Jackson na bagagem (lib de JSON).
- Acesso:
  - `http://localhost:8080/cliente-exemplo`
- Resultado esperado (exemplo de JSON):

```
json

{
  "id": 1,
  "nome": "João da Silva",
  "email": "joao.silva@example.com"
}
```

Você explica:

> "Agora não estamos mais devolvendo texto simples. Estamos devolvendo um JSON, que é o formato padrão de comunicação entre backend e frontend."

## 2.7. Terceiro exemplo – Endpoint com parâmetro (path variable)

Mostre como receber parâmetros via URL:

```java
package com.exemplo.primeirospringboot.controller;

import com.exemplo.primeirospringboot.model.Cliente;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ClienteController {

    @GetMapping("/cliente/{id}")
    public Cliente obterClientePorId(@PathVariable Long id) {
        // Em um projeto real, isso viria do banco de dados.
        // Aqui vamos devolver um cliente "fake" usando o ID
passado.
        return new Cliente(id, "Cliente " + id, "cliente" + id +
"@example.com");
    }
}
```

**Explicação:**

- `@GetMapping("/cliente/{id}")`:
  - Indica que esse método responde a URLs como `/cliente/1`, `/cliente/2`, etc.
- `@PathVariable Long id`:
  - Faz o Spring pegar o valor `{id}` da URL e jogar no parâmetro `id` do método.
- Retorno:
  - Cria um `Cliente` usando o ID recebido, só para demonstrar a dinâmica.

Isso já mostra:

- Como receber dados pela URL.

- Como retornar JSON dinâmico.

## 2.8. Quarto exemplo – Endpoint POST com corpo (request body)

```
java

package com.exemplo.primeirospringboot.controller;

import com.exemplo.primeirospringboot.model.Cliente;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ClienteController {

    @PostMapping("/cliente")
    public Cliente criarCliente(@RequestBody Cliente cliente) {
        // Aqui em um projeto real você salvaria o cliente no banco.
        // Vamos apenas retornar o mesmo cliente como se tivesse
sido "salvo".
        return cliente;
    }
}
```

**Explicação:**

- `@PostMapping("/cliente")`:
  - Endpoint que responde a HTTP POST.
- `@RequestBody Cliente cliente`:
  - O Spring converte automaticamente o JSON enviado no corpo da requisição para o objeto `Cliente`.
- Na prática:
  - O aluno pode usar Postman ou Insomnia para enviar um POST com JSON:

```
json

{
  "id": 10,
  "nome": "Maria",
  "email": "maria@example.com"
}
```

- O serviço devolve o mesmo JSON na resposta.

Frase para fixar:

> "Com poucas linhas de código, conseguimos criar uma API que recebe e devolve JSON, pronta para se conectar com qualquer frontend."

## 2.9. Comentando rapidamente o `application.properties`

Mostre o arquivo `src/main/resources/application.properties` e explique:

- Podemos configurar:
  - Porta do servidor:
    `server.port=8081`
  - Nome da aplicação:
    `spring.application.name=primeiro-springboot`

Exemplo:

properties

```
spring.application.name=primeiro-springboot
server.port=8081
```

-