

Curso

Aplicações JAVA com SPRING BOOT



Prof. Msc. Antonio B. C. Sampaio Jr
engenheiro de software & professor

@abctreinamentos
@amazoncodebr

www.abctreinamentos.com.br
www.amazoncode.com.br



AULA DE HOJE



- UNIDADE 1 – INTRODUÇÃO
- UNIDADE 2 – FUNDAMENTOS DO SPRING BOOT
- UNIDADE 3 – PERSISTÊNCIA DE DADOS NO SPRING BOOT
- UNIDADE 4 – PROJETO WEB NO SPRING BOOT
- UNIDADE 5 – PROJETO REST API NO SPRING BOOT
- UNIDADE 6 – PROJETO REST API NO SPRING BOOT COM REACTJS

PROJETOS DE HOJE



- 1º Projeto Spring Boot – Impressão de Mensagens
- 2º Projeto Spring Boot – Impressão de Mensagens na WEB
- 3º Projeto Spring Boot – Aplicação Servidor Público
- 4º Projeto Spring Boot – Aplicação Servidor Público na WEB
- 5º Projeto Spring Boot – Aplicação Servidor Público no SGBD MYSQL
- 6º Projeto Spring Boot – Aplicação Servidor Público no MONGO DB

UNIDADE 2

FUNDAMENTOS DO SPRING BOOT

DOCUMENTAÇÃO SPRING BOOT

ESTRUTURA DE UM PROJETO SPRING BOOT

PADRÕES DE PROJETO

ANOTAÇÕES MAIS COMUNS DO SPRING BOOT

PROFILES SPRING BOOT

ARQUITETURA SPRING BOOT

PROJETOS PRÁTICOS

3º Projeto Spring Boot – Aplicação Servidor
Público

4º Projeto Spring Boot – Aplicação Servidor
Público WEB

DOCUMENTAÇÃO SPRING BOOT


DOCUMENTAÇÃO SPRING BOOT


- **Para que Serve?**
 - A documentação oficial do Spring Boot é uma referência completa para desenvolvedores que desejam utilizar essa ferramenta de desenvolvimento em seus projetos. Ela contém informações detalhadas sobre como usar as funcionalidades do Spring Boot, desde a configuração inicial do projeto até a implementação de recursos avançados.
 - A documentação do Spring Boot é útil para aprender como usar o framework, para solucionar problemas, para entender o funcionamento interno de determinada funcionalidade e para encontrar informações técnicas específicas, como APIs e classes.

DOCUMENTAÇÃO SPRING BOOT

- **Para que Serve?**
 - A documentação é atualizada regularmente, conforme novas versões do Spring Boot são lançadas. Ela é uma fonte confiável e precisa de informações, e pode ajudar a evitar erros e bugs no desenvolvimento de aplicações Spring Boot. Além disso, a documentação oficial também contém exemplos de código e tutoriais, que podem ser úteis para aprender e entender melhor como usar as funcionalidades do framework.

DOCUMENTAÇÃO SPRING BOOT





- 1. Legal
- 2. Getting Help
- 3. Documentation Overview
- 4. Getting Started
- 5. Upgrading Spring Boot
- 6. Developing with Spring Boot
- 7. Core Features
- 8. Web
- 9. Data
- 10. Messaging
- 11. IO
- 12. Container Images
- 13. Production-ready Features
- 14. Deploying Spring Boot Applications
- 15. GraalVM Native Image Support
- 16. Spring Boot CLI
- 17. Build Tool Plugins
- 18. "How-to" Guides
- Appendices

Spring Boot Reference Documentation

Phillip Webb • Dave Syer • Josh Long • Stéphane Nicoll • Rob Winch • Andy Wilkinson • Marcel Overdijk • Christian Dupuis • Sébastien Deleuze • Michael Simons • Vedran Pavić • Jay Bryant • Madhura Bhawe • Eddú Meléndez • Scott Frederick • Moritz Halbritter

Version 3.0.5

This document is also available as [multiple HTML pages](#) and as [a PDF](#).

1. Legal

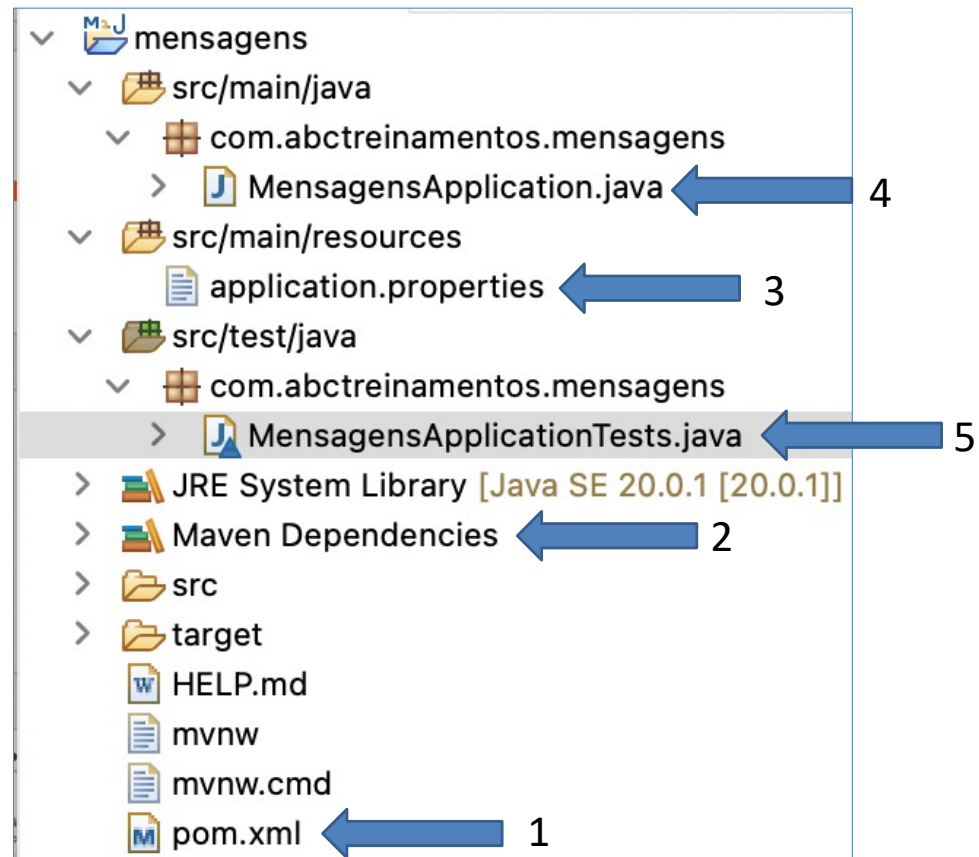
Copyright © 2012-2023

Copies of this document may be made for your own use and for distribution to others, provided that you do not charge any fee for such copies and further provided that each copy contains this Copyright Notice, whether distributed in print or electronically.

<https://docs.spring.io/spring-boot/docs/3.0.5/reference/htmlsingle/#legal>

ESTRUTURA DE UM PROJETO SPRING BOOT

ESTRUTURA DE UM PROJETO SPRING BOOT



(1) ARQUIVO POM.XML

- O **Maven** é uma ferramenta desenvolvida pela Apache para gerenciar projetos Java, organizando as suas dependências e automatizando os seus builds.
- O arquivo **pom.xml** (*Project Object Model*) define todas as configurações do Maven.
- Vamos dividir o arquivo **pom.xml** em 05 partes, para facilitar o seu entendimento:
- 1ª. Parte: Define todas as configurações básicas e dependências que serão herdadas em cada projeto Spring Boot

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.0.5</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

(1) ARQUIVO POM.XML

- 2ª. Parte: Identificação do Projeto

```
<groupId>com.abctreinamentos</groupId>  
<artifactId>mensagensWeb</artifactId>  
<version>0.0.1-SNAPSHOT</version>  
<name>mensagensWeb</name>  
<description>Envio de mensagens - Spring Boot</description>
```

- 3ª. Parte: Propriedades do Projeto

```
<properties>  
  <java.version>20</java.version>  
</properties>
```

(1) ARQUIVO POM.XML

- 4ª. Parte: Dependências do Projeto

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

- 5ª. Parte: Plugins necessários para a compilação do Projeto

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

(2) SPRING BOOT MAVEN PLUG-IN

- O **Spring Boot Maven Plugin** é um plugin Maven que fornece suporte para compilar e empacotar aplicativos Spring Boot em um único arquivo executável. Ele automatiza muitas tarefas comuns, como a criação de um arquivo JAR executável, a adição de dependências necessárias ao arquivo JAR e a execução do aplicativo.
- O plugin fornece vários objetivos do Maven que permitem realizar tarefas relacionadas ao Spring Boot, como a execução do aplicativo Spring Boot, a execução de testes de integração do Spring Boot e a execução de um servidor Spring Boot embutido.

(2) SPRING BOOT MAVEN PLUG-IN



- Faz o empacotamento dos arquivos .JAR e .WAR para serem executáveis



- Roda Aplicações Spring Boot

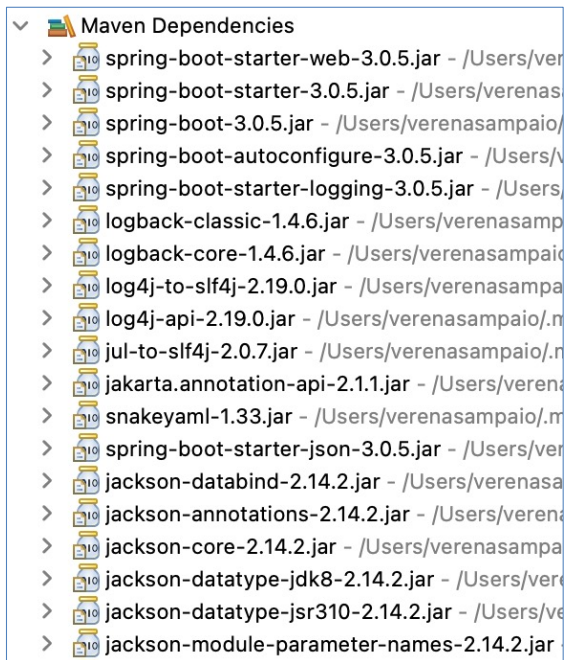


- Fornece os *Spring Boot Starters* - que são um conjunto de dependências pré-configuradas para casos de uso comuns, como aplicativos da Web, acesso ao banco de dados, segurança, etc.



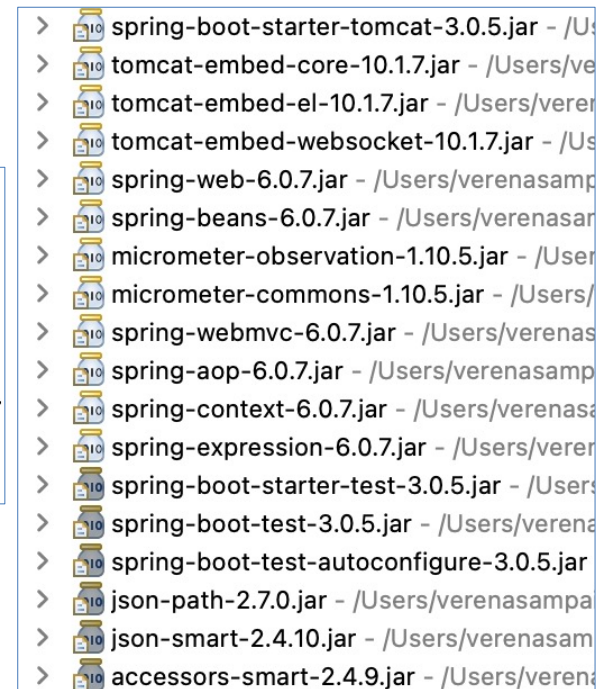
- Gerencia o Ciclo de Vida de uma Aplicação Spring Boot

(2) DEPENDÊNCIAS MAVEN



```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```



(3) ARQUIVO **APPLICATION.PROPERTIES**

- O arquivo **application.properties** é um arquivo de configuração utilizado pelo Spring Boot para configurar diferentes aspectos da aplicação, como conexão de banco de dados, configurações de servidor web, propriedades do ambiente, entre outros.
- Ao executar uma aplicação Spring Boot, o arquivo **application.properties** é lido automaticamente pelo framework e as configurações definidas nele são aplicadas na aplicação.

.A.1. Core Properties

- .A.2. Cache Properties
- .A.3. Mail Properties
- .A.4. JSON Properties
- .A.5. Data Properties
- .A.6. Transaction Properties
- .A.7. Data Migration Properties
- .A.8. Integration Properties

.A.9. Web Properties

- .A.10. Templating Properties
- .A.11. Server Properties
- .A.12. Security Properties
- .A.13. RSocket Properties
- .A.14. Actuator Properties
- .A.15. Devtools Properties
- .A.16. Testing Properties

(4) CLASSE MENSAGENSAPPLICATION.JAVA

- É a Classe Principal do 1º Projeto Spring Boot.

```
@SpringBootApplication
public class MensagensApplication {

    public static void main(String[] args) {
        System.setProperty("java.awt.headless", "false");
        SpringApplication.run(MensagensApplication.class, args);
        JOptionPane.showMessageDialog(null, "Primeiro Projeto Spring Boot");
        System.out.println("Primeiro Projeto Spring Boot");
        create(); read(); update(); delete();
    }

    public static void create() {
        System.out.println("Criação de um Registro");
    }

    public static void read() {
        System.out.println("Leitura de um Registro");
    }

    public static void update() {
        System.out.println("Atualização de um Registro");
    }

    public static void delete() {
        System.out.println("Exclusão de um Registro");
    }
}
```

(5) CLASSE MENSAGENSAPPLICATIONTESTS.JAVA

- É a Classe Principal de Testes do 1º Projeto Spring Boot.

```
@SpringBootTest
class MensagensApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

PADRÕES DE PROJETO

PADRÕES DE PROJETO

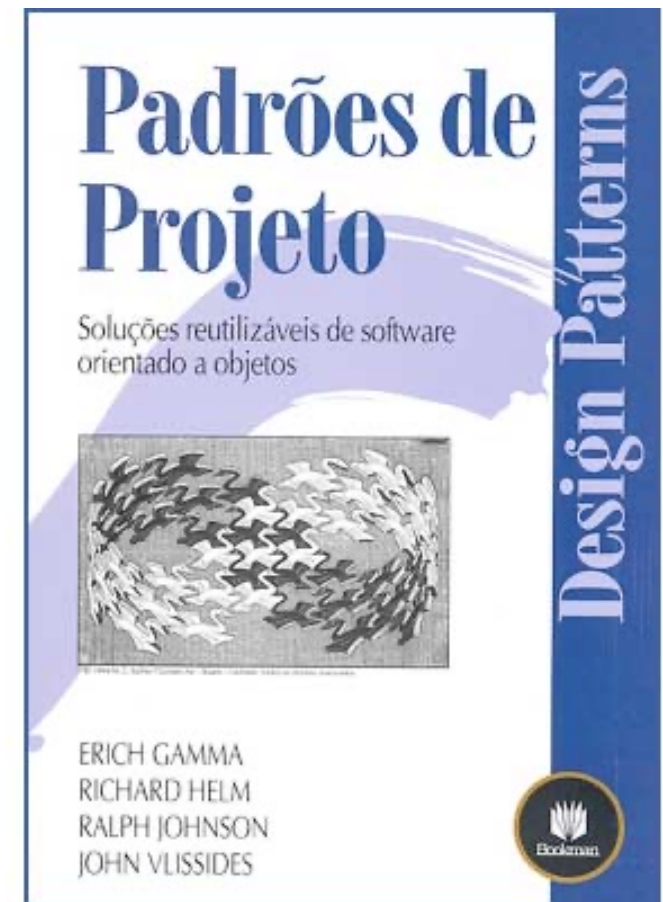
- **Definição**

- Padrões de projeto de software, também conhecidos como *Design Patterns*, são soluções comuns e testadas para problemas recorrentes no desenvolvimento de software. Eles são uma forma de capturar a experiência e o conhecimento coletivo de desenvolvedores que enfrentaram e resolveram problemas semelhantes no passado.
- Os padrões de projeto fornecem uma linguagem comum para desenvolvedores, permitindo que eles comuniquem de forma eficaz soluções e estratégias de projeto. Eles ajudam a evitar reinventar a roda e podem aumentar a eficiência e a qualidade do desenvolvimento de software.

PADRÕES DE PROJETO

- **Definição**

- Existem vários tipos de padrões de projeto, como padrões de criação, padrões estruturais e padrões comportamentais. Cada tipo aborda um conjunto diferente de problemas no desenvolvimento de software. Alguns exemplos de padrões de projeto famosos incluem Singleton, Factory Method, MVC, Observer, Strategy, Adapter, Injeção de Dependência, Inversão de Controle, entre outros.
- O livro ao lado é a referência no estudo de padrões de projeto na área de engenharia de software.





INVERSÃO DE CONTROLE

- **Definição**

- Inversão de Controle (*Inversion of Control* - IoC) é um princípio de design de software que permite criar sistemas mais flexíveis e com menor acoplamento entre os componentes. A ideia é que, em vez de cada componente do sistema ser responsável por instanciar e gerenciar suas próprias dependências, essa responsabilidade é transferida para um componente externo, chamado de container de IoC.
- O container de IoC é responsável por gerenciar as dependências dos componentes do sistema e, assim, controlar o fluxo de execução da aplicação. Ele provê aos componentes as dependências necessárias e pode substituí-las ou modificar seu comportamento de acordo com a configuração da aplicação.



INVERSÃO DE CONTROLE

- **Definição**

- Dessa forma, os componentes podem ser desenvolvidos de forma mais modular e reutilizável, facilitando a manutenção e evolução do sistema como um todo.
- A inversão de controle é frequentemente associada ao padrão de projeto Injeção de Dependência (*Dependency Injection* - DI), que é uma técnica utilizada para implementar a IoC.
- Com a DI, as dependências de um componente são "injetadas" nele pelo container de IoC, em vez de serem criadas internamente pelo componente. Isso torna o componente mais flexível, pois ele não precisa saber como suas dependências são criadas, apenas como utilizá-las.

INJEÇÃO DE DEPENDÊNCIA

- **Definição**

- É um padrão de projeto usado para evitar o alto nível de acoplamento de código dentro de uma aplicação. Sistemas com baixo acoplamento de código são melhores pelos seguintes motivos: aumento na facilidade de manutenção/implementação de novas funcionalidades e também facilita a realização de testes unitários.

```
public class Pedido {  
  
    public static void main(String[] args) {  
        List<Produto> produto = new ArrayList<>();  
        Produto roupa = new Produto();  
        produto.add(roupa);  
    }  
}
```

- A Classe Pedido está altamente “acoplada” com a Classe Produto. Se houver uma alteração no construtor de Produto, a classe Pedido também deverá ser alterada.



INJEÇÃO DE DEPENDÊNCIA

- A Injeção de Dependência (DI) é uma das formas de se implementar a IoC. A outra forma é utilizar o padrão de projeto *Service Locator*.
- Existem quatro maneiras de implementar a DI:
 - Construtor
 - Setter
 - Interface
 - Anotações



INJEÇÃO DE DEPENDÊNCIA

- **Construtor**

- A dependência é fornecida como parâmetro do construtor da classe. É a forma mais comum de realizar a injeção de dependência.

```
public class Pedido {  
    private List<Produto> produto;  
    public Pedido (List<Produto> produto)  
    {  
        this.produto = produto;  
    }  
}
```

INJEÇÃO DE DEPENDÊNCIA

- **Construtor**
 - E se houver mais de um construtor (sobrecarga), como o Spring saberá qual será o ponto de injeção?

```
public class Pedido {  
  
    private List<Produto> produto;  
  
    // Esse é o ponto de injeção  
    @Autowired  
    public Pedido (List<Produto> produto)  
    {  
        this.produto = produto;  
    }  
  
    public Pedido (String nomeProduto)  
    {  
        // sobrecarga  
    }  
}
```



INJEÇÃO DE DEPENDÊNCIA

- **Método Setter**

- A dependência é definida por meio de um método setter. Essa técnica permite que as dependências sejam alteradas a qualquer momento durante a execução do programa.

```
public class Pedido {  
  
    private List<Produto> produto;  
  
    // Esse é o ponto de injeção  
    @Autowired  
    public void setProduto(List<Produto> produto)  
    {  
        this.produto = produto;  
    }  
}
```

INJEÇÃO DE DEPENDÊNCIA

- **Interface**
 - Uma interface é definida para a dependência, e a classe que precisa dessa dependência é projetada para implementar essa interface.

```
public class Pedido {  
    private List<IProduto> produto;  
    public Pedido (List<IProduto> produto)  
    {  
        this.produto = produto;  
    }  
}
```

INJEÇÃO DE DEPENDÊNCIA

- **Anotações**

- Anotações são usadas para identificar as dependências que precisam ser injetadas. O container de injeção de dependência examina as anotações e injeta as dependências correspondentes automaticamente.

```
public class Pedido {  
  
    // Esse é o ponto de injeção  
    @Autowired  
    private List<Produto> produto;  
  
    public List<Produto> produtos()  
    {  
        return produto;  
    }  
}
```

ANOTAÇÕES MAIS COMUNS DO SPRING BOOT



ANOTAÇÕES SPRING BOOT

- **Anotações**

- O Spring Boot é construído em cima do Spring Framework e, portanto, herda muitas das anotações do Spring. Algumas das anotações mais comuns do Spring Boot incluem:
 - **@SpringBootApplication**: é uma combinação de três anotações do Spring: **@Configuration**, **@EnableAutoConfiguration** e **@ComponentScan**. Ela é usada para marcar uma classe como sendo a principal de uma aplicação Spring Boot.
 - **@RestController**: é usada para marcar uma classe como controladora do Spring MVC e permite criar endpoints da web que retornam dados no formato **JSON**.



FORMATO JSON

- JSON (*JavaScript Object Notation*) é um formato compacto, de padrão aberto independente, de troca de dados simples e rápida entre sistemas, especificado por Douglas Crockford em 2000, que utiliza texto legível a humanos, no formato atributo-valor.
- É ideal para enviar e receber informações pela Internet.

```
{  
  "id": 123,  
  "nome": "JSON T-Shirt",  
  "preco": 99.99,  
  "estoque": {  
    "deposito": 300,  
    "loja": 20  
  }  
}
```



ANOTAÇÕES SPRING BOOT

- **Anotações**

- **@Autowired**: é usada para injetar dependências automaticamente em uma classe. O Spring Boot gerencia essas dependências para você.
- **@Value**: é usada para injetar valores de propriedades definidas em um arquivo **application.properties** ou **application.yml**.
- **@GetMapping, @PostMapping, @PutMapping, @DeleteMapping**: são usadas para mapear endpoints HTTP com os métodos GET, POST, PUT e DELETE, respectivamente.
- **@PathVariable**: é usada para injetar variáveis de caminho em um método de controlador.



ANOTAÇÕES SPRING BOOT

- **Anotações**

- **@RequestBody**: é usada para injetar o corpo de uma solicitação HTTP em um objeto.
- **@Bean** é usada para indicar que um método de um componente Spring (como uma classe anotada com **@Configuration**, por exemplo) deve ser responsável por criar e configurar um objeto gerenciado pelo container Spring, também conhecido como **bean**.
- **@PostConstruct** é usada para indicar que um método deve ser executado logo após a inicialização do bean.



ANOTAÇÕES SPRING BOOT

- **Anotações**

- **@Document** é usada no Spring Data MongoDB para marcar uma classe de domínio como representante de um documento do MongoDB.
- **@Configuration** é utilizada para marcar uma classe como uma classe de configuração do Spring, onde são definidos beans e outras configurações do container. Além disso, classes marcadas com **@Configuration** também podem conter métodos marcados com **@Bean**, que definem **beans específicos**.
- Essas são apenas algumas das anotações mais comuns do Spring Boot. Existem muitas outras anotações que podem ser usadas para personalizar e configurar um aplicativo Spring Boot.



- **Definição**

- Um **bean** é simplesmente um objeto que é gerenciado pelo container Spring. Esses objetos são criados, montados e gerenciados pelo próprio Spring, e podem ser instanciados através da configuração do container, como por exemplo, através de anotações como **@Component**, **@Service**, **@Repository**, **@Controller**, entre outras.
- O uso de **beans** permite uma maior flexibilidade e modularidade em um sistema, pois permite que objetos sejam criados de forma desacoplada e possam ser injetados em outras classes que dependem deles. Isso torna a aplicação mais fácil de ser testada, mantida e escalada, já que a configuração dos **beans** pode ser ajustada de acordo com as necessidades de cada ambiente ou cenário de uso.



TIPOS DE BEANS

- São 04 Os Tipos de Beans:

- @Component
- @Service
- @Repository
- @Controller



TIPOS DE BEANS

- **@Component**: é a anotação mais genérica e pode ser utilizada para qualquer classe que seja um componente do sistema.

```
@Component
public class Cliente {
    ...
}
```

- **@Service**: é uma especialização de **@Component**, utilizada para classes que possuem regras de negócio.

```
@Service
public class ClienteService
{
    public void service()
    {
        //regras de negócios
    }
}
```




TIPOS DE BEANS

- **@Repository**: também é uma especialização de **@Component**, utilizada para classes que realizam operações de acesso a dados, como leitura e gravação em um banco de dados.

```
@Repository
public class ClienteDAO
{
    public void delete()
    {
        //código
    }
}
```

- **@Controller**: é uma especialização de **@Component**, utilizada para classes que são responsáveis por receber e tratar requisições HTTP em um aplicativo web.



TIPOS DE BEANS

- **@Controller / @RestController**: anotação utilizada em classes que realizarão a função de **Controller** de uma aplicação Web MVC.

```
@SpringBootApplication
@RestController
public class MensagensWebApplication {

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

    @GetMapping("/")
    public String init() {
        return("<a href=\"/create\">Create</a> <br/>");
    }
}
```

- A anotação **@Controller** representa uma aplicação WEB tradicional que precisa de uma camada **View** para apresentar o conteúdo HTML + CSS + JS.



TIPOS DE BEANS

- **@RestController**: anotação utilizada para implementar uma API REST, isto é, uma aplicação que retorna dados no padrão JSON/HTTP.
- **@RestController = @Controller + @ResponseBody**

TESTAR!

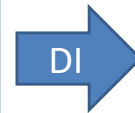


TIPOS DE BEANS

- Além desses 04 Tipos de Beans, é possível usar o **@Bean** em um método para tornar a instância retornada por ele como um objeto gerenciado pelo Spring.

```
@Configuration
public class Config {

    @Bean
    public ClientServ clientServ() {
        ClientServ client = new ClientServ();
        return client;
    }
}
```



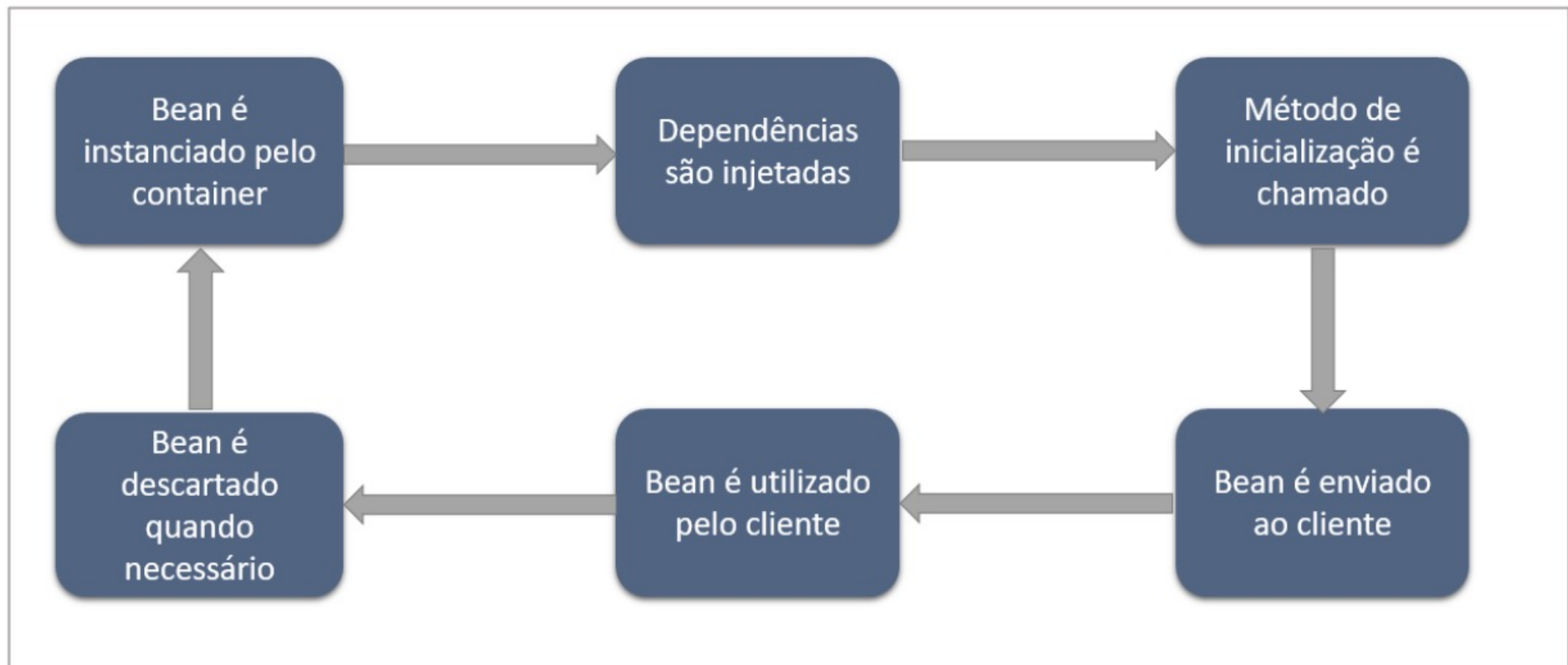
```
@SpringBootApplication
public class Spring App {

    @Autowired
    private ClientServ clientServ;
}
```

- Por padrão, **beans** que usam **@Bean** são criados dentro de classes do tipo **@Configuration**.



CICLO DE VIDA DOS BEANS



PROFILES
SPRING BOOT



PROFILES SPRING BOOT

- **Definição**

- Profiles são uma maneira de configurar e personalizar o comportamento da aplicação com base em diferentes ambientes ou cenários, como desenvolvimento, teste, produção, etc.
- Cada profile representa um conjunto de propriedades de configuração que são ativadas quando o perfil correspondente é selecionado. Isso permite que seja alterada dinamicamente o comportamento da aplicação com base no ambiente em que ela está sendo executada.



PROFILES SPRING BOOT

- **Definição**

- Qualquer bean @Component ou @Configuration pode ser “marcado” com @Profile para limitar quando ele será executado:

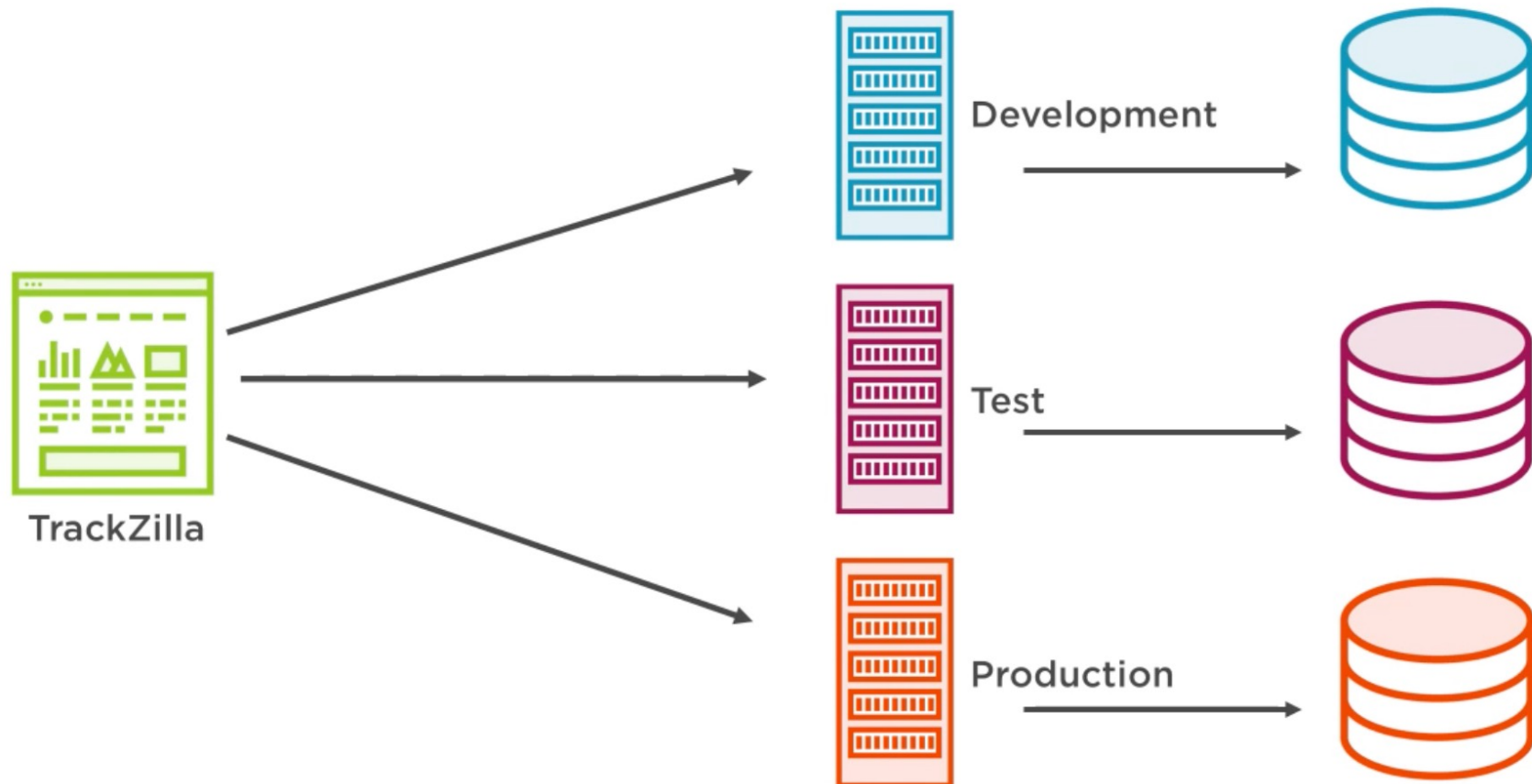
```
@Configuration  
@Profile("dev")  
public class DevConfiguration {  
    // ...  
}
```

- No arquivo `application.properties`, a seguinte propriedade deverá ser definida:

```
spring.profiles.active=dev
```




PROFILES SPRING BOOT





PROFILES SPRING BOOT

- Os profiles podem ser configurados em arquivos de propriedades, como por exemplo: application.properties.
- As informações abaixo deverão ser especificadas no arquivo **application.properties**.

```
spring.profiles.active=dev
```

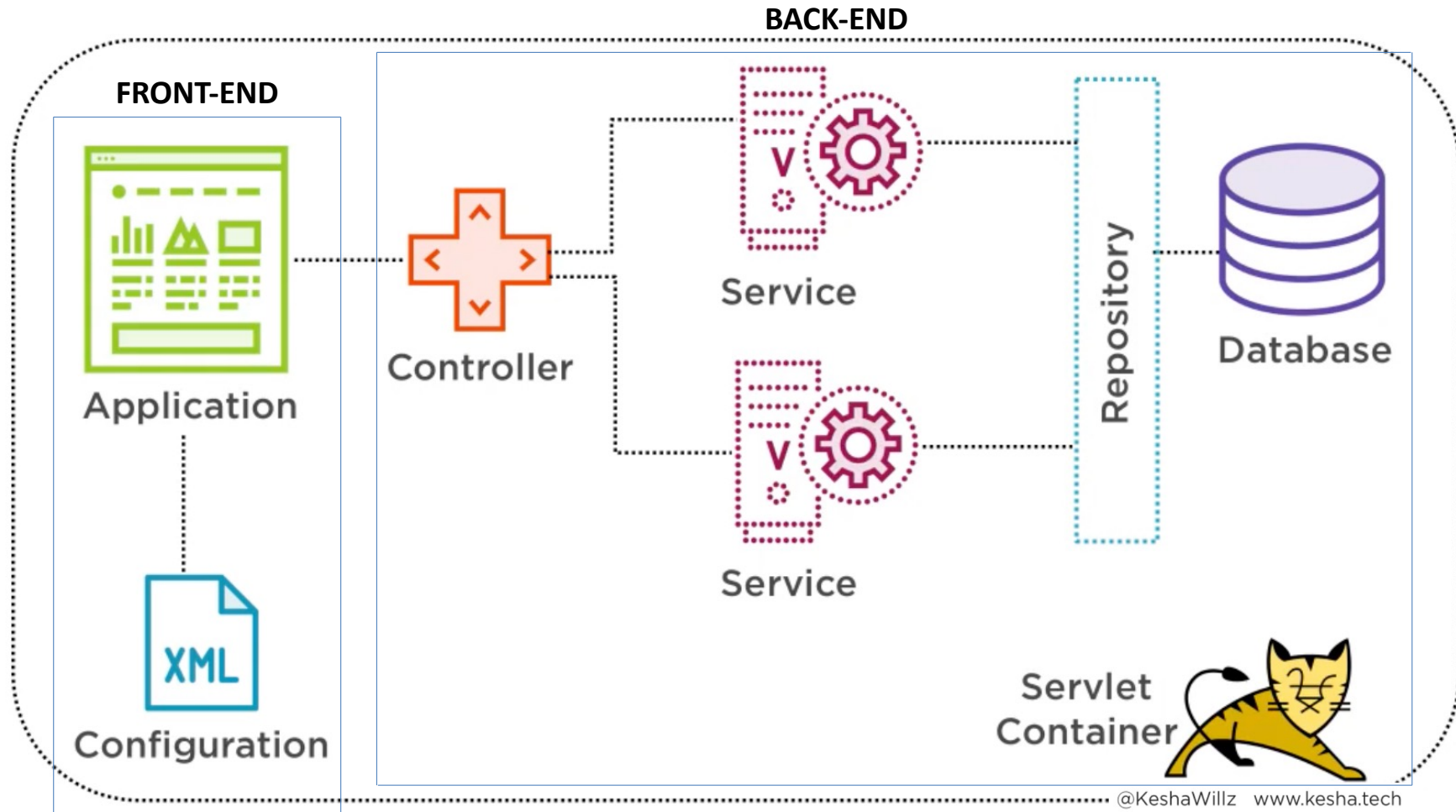
- E é possível criar um arquivo de configuração específico para cada tipo de profile.

```
applications-{profile}.properties  
  
applications-dev.properties  
applications-test.properties  
applications-prod.properties
```

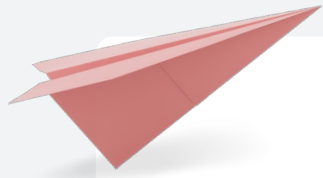
ARQUITETURA
SPRING BOOT



ARQUITETURA SPRING BOOT

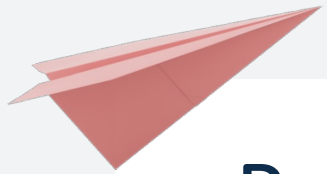


PROJETOS PRÁTICOS



3º Projeto Spring Boot – Aplicação Servidor Público





Passos:

- Criar o repositório de dados (servidorPublico.json)
- Criar os componentes Spring Boot
- Criar os métodos CRUD





TELA PRINCIPAL

Foto	Nome	Cargo	Órgão	Lotação	Email	Cursos	Ações
	Isabela Sampaio	Auditor	ANVISA	Brasília	assoftbel@gmail.com	Não Tem Cursos	Detalhar Alterar Excluir
	Heila Ghassan	Estagiário	STN	Brasília	heila@gmail.com	Não Tem Cursos	Detalhar Alterar Excluir
	Maria Fontenele	Analista	ENAP	Brasília	mariafontenele@enap.br	Cursos Matriculados	Detalhar Alterar Excluir
	Caio Santos	Analista Tributário	RFB	Rio de Janeiro	caiosantos@rfb.gov.br	Cursos Matriculados	Detalhar Alterar Excluir





TELA DO SERVIDOR

Isabela Sampaio

Informações Funcionais



Matrícula: 6

Órgão: ANVISA

Cargo: Auditor

Data de Admissão: 2022-11-21T12:55:40.496Z

Lotação: Brasília

Exercício: SUPEN

Vínculo: Estatutário

Informações Pessoais

Email: assoftbel@gmail.com

Telefone: (91)991124552

Celular: (91)981144444

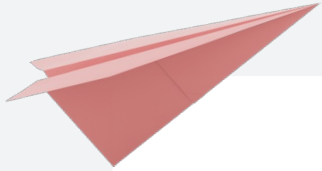
CPF: 00000000000

Data de Nascimento: 2022-08-03T12:55:40.496Z

Naturalidade: Belém

Voltar





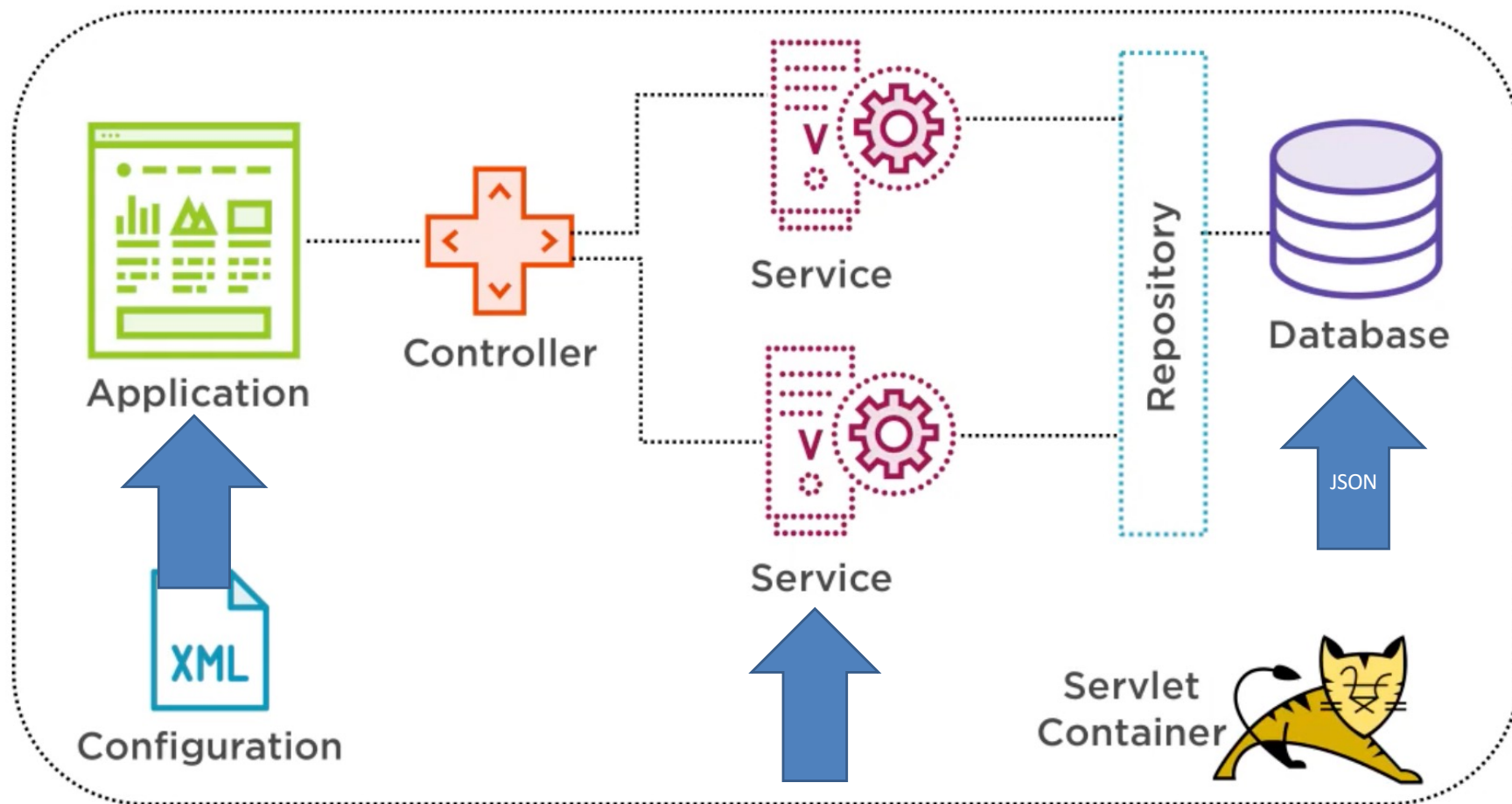
SERVIDORPUBLICO.JSON

```
[
  {
    "matricula": 3,
    "nome": "Maria Fontenele",
    "foto": "https://abctreinamentos.com.br/imgs/maria.png",
    "orgao": "ENAP",
    "vinculo": "Estatutário",
    "cargo": "Analista",
    "lotacao": "Brasília",
    "exercicio": "Departamento de Treinamento",
    "email": "mariafontenele@enap.br",
    "telefone": "(61) 3255-6010",
    "celular": "(61) 99910-5722",
    "cpf": "123.4567.789-01",
    "naturalidade": "Recife",
  }
]
```



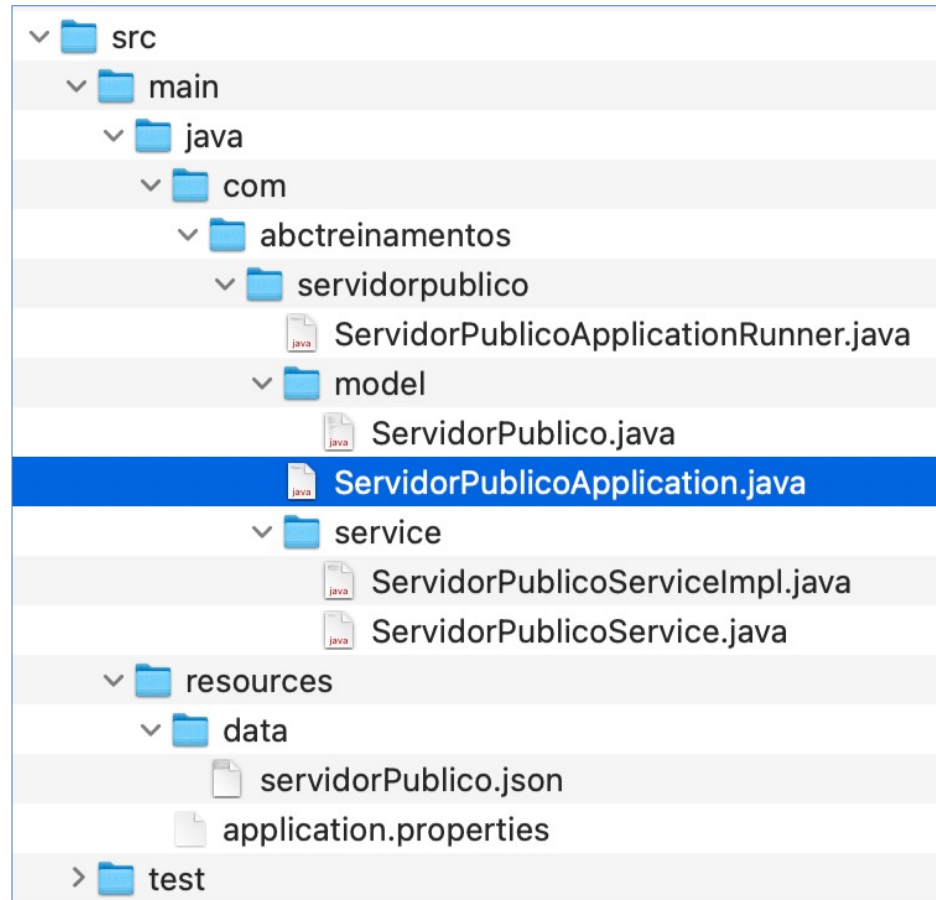


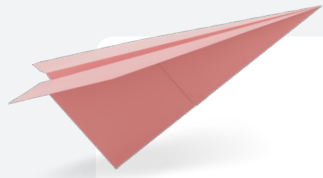
ARQUITETURA SPRING BOOT





ARQUITETURA DO PROJETO





4º Projeto Spring Boot – Aplicação Servidor Público WEB





ARQUITETURA DO PROJETO

