



## Digital Business Enablement

Prof. Gilberto Alexandre das Neves  
[profgilberto.neves@fiap.com.br](mailto:profgilberto.neves@fiap.com.br)

# Design Patterns

# Design Patterns

Vamos desenvolver uma aplicação utilizando o *framework* **Spring**.

Objetivos:

- Desenvolvimento de uma API Rest
- CRUD (Create, Read, Update e Delete)
- Validações
- Paginação e Ordenações

Tecnologias:

- Spring Boot 3
- Java 17
- Oracle Database
- JPA/Hibernate
- Maven
- Insomnia

Vamos usar algumas tecnologias, como **Spring Boot 3**, sendo a última versão disponibilizada pelo framework.

Usaremos, também, o **Java 17** sendo a última versão **LTS** (Long-term support, em português "Suporte de longo prazo") que possui maior tempo de suporte disponível para o Java.

Usaremos o banco de dados **Oracle** para armazenar as informações da API.

A camada de persistência da nossa aplicação será feita com a **JPA** (Java Persistence API), com o **Hibernate** como implementação dessa especificação e usando os módulos do Spring Boot, para tornar esse processo o mais simples possível.

Usaremos o **Maven** para gerenciar as dependências do projeto, e também para gerar o build da nossa aplicação.

Por último, como focaremos na API Rest (apenas no Back-end), não teremos interface gráfica, como páginas HTML e nem Front-end e aplicativo mobile. Mas para testarmos a API, usaremos o **Insomnia**, sendo uma ferramenta usada para testes em API. Com ela, conseguimos simular a requisição para a API e verificar se as funcionalidades implementadas estão funcionando.

# I Qual é o projeto?

Vamos montar um projeto para conectar prestadores de serviços de reformas em geral com potenciais clientes.

Será um aplicativo com algumas opções, em que a pessoa que for usar pode fazer o CRUD, tanto de prestadores quanto de clientes e o agendamento e cancelamento dos serviços.

Conforme avançamos no desenvolvimento do projeto será apresentado as regras de negócios e validações necessárias de cada funcionalidades/entidades envolvidas.

# Spring Initializr

# Spring Initializr

Para iniciarmos o nosso projeto é criá-lo (iniciaremos do zero) usaremos o **Spring Initializr**, sendo uma ferramenta disponibilizada pela equipe do Spring Boot para criarmos o projeto com toda estrutura inicial necessária.

Acessaremos o **Spring Initializr** pelo site <https://start.spring.io/>.

The screenshot displays the Spring Initializr configuration interface. It is divided into three main sections: Project, Spring Boot, and Project Metadata. In the Project section, 'Maven' is selected under 'Project' and 'Java' is selected under 'Language'. In the Spring Boot section, '3.0.2' is selected. The Project Metadata section contains text input fields for Group (br.com.reformas), Artifact (api), Name (api), Description (API Rest da aplicação de reformas.com.br), and Package name (br.com.reformas.api). Under the Packaging section, 'Jar' is selected. At the bottom, '17' is selected for the Java version.

Project		Language	
<input type="radio"/> Gradle - Groovy	<input type="radio"/> Gradle - Kotlin	<input checked="" type="radio"/> Java	<input type="radio"/> Kotlin
<input checked="" type="radio"/> Maven		<input type="radio"/> Groovy	

Spring Boot			
<input type="radio"/> 3.0.3 (SNAPSHOT)	<input checked="" type="radio"/> 3.0.2	<input type="radio"/> 2.7.9 (SNAPSHOT)	<input type="radio"/> 2.7.8

Project Metadata	
Group	br.com.reformas
Artifact	api
Name	api
Description	API Rest da aplicação de reformas.com.br
Package name	br.com.reformas.api
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 19 <input checked="" type="radio"/> 17 <input type="radio"/> 11 <input type="radio"/> 8

À direita da tela temos a seção "Dependencies" e um botão "Add dependencies". Nela, adicionaremos as dependências do Spring que desejamos incluir no projeto.

Vamos clicar no botão "Add dependencies". Vamos apertar a tecla "Ctrl" do teclado e clicar em cada uma das dependências que desejamos adicionar, sendo elas: **Spring Boot DevTools** e **Spring Web**.

O **Spring Boot DevTools** é um módulo do Spring Boot que serve para não precisarmos reiniciar a aplicação a cada alteração feita no código. Isto é, toda vez que salvarmos as modificações feitas no código, ele subirá automaticamente.

A próxima dependência é a **Spring Web**, dado que vamos trabalhar com uma API Rest e precisamos do módulo web.

Após isso, apertaremos a tecla "Esc" para fechar a pop-up.



**Dependencies**

ADD DEPENDENCIES... CTRL + B

**Spring Boot DevTools** **DEVELOPER TOOLS**  
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

**Spring Web** **WEB**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Depois de preencheremos todas as informações e adicionarmos as dependências, podemos selecionar o botão "Generate" na parte inferior da página.

**GENERATE** CTRL + G

Vamos gerar o projeto e teremos um arquivo .zip com o projeto compactado.

# Spring e Spring Boot

Spring e Spring Boot não são a mesma coisa!

Spring é um framework para desenvolvimento de aplicações em Java, criado em meados de 2002 por Rod Johnson, que se tornou bastante popular e adotado ao redor do mundo devido a sua simplicidade e facilidade de integração com outras tecnologias.

O framework foi desenvolvido de maneira **modular**. Em cada aplicação podemos adicionar apenas os módulos que fizerem sentido.

Existem diversos módulos no Spring, cada um com uma finalidade distinta, como por exemplo:

- o módulo **MVC**, para desenvolvimento de aplicações Web e API's Rest;
- o módulo **Security**, para lidar com controle de autenticação e autorização da aplicação;
- e o módulo **Transactions**, para gerenciar o controle transacional.

# Spring e Spring Boot

Um dos grandes problemas existentes em aplicações que utilizavam o Spring era a parte de configurações de seus módulos, que era feita toda com arquivos **XML**. Depois de alguns anos o framework também passou a dar suporte a configurações via classes Java, utilizando *anotações*.

Dependendo do tamanho e complexidade da aplicação, e também da quantidade de módulos do Spring utilizados nela, tais configurações eram bastante extensas e de difícil manutenção.

Iniciar um novo projeto com o Spring era uma tarefa um tanto quanto complicada, devido a necessidade de realizar tais configurações no projeto.

Para resolver tais dificuldades é que foi criado um novo módulo do Spring, chamado de **Boot**, em meados de 2014, com o propósito de agilizar a criação de um projeto que utilize o Spring como framework, bem como simplificar as configurações de seus módulos.

# Abrindo o projeto

# | Abrindo o projeto

Vamos descompactar o arquivo **.zip** baixado em uma pasta a sua escolha.

Agora abra o **Eclipse IDE for Enterprise Java and Web Developers**.

Vamos no menu **File** e escolha **Open Projects from File System...**

Clique no botão **Directory...** e localize e selecione a pasta descompactada do projeto.

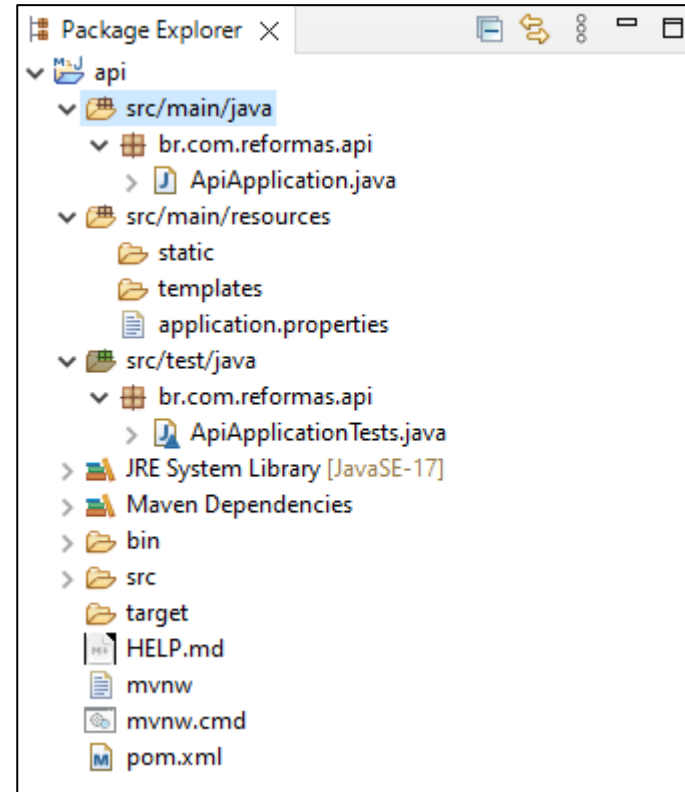
Confirme que ele reconhece como um projeto em **Maven** e clique em **Finish**.

**Importante:** Após a abertura do projeto aguarde alguns minutos, o Eclipse automaticamente vai baixar os arquivos necessários (as dependências que selecionamos). Acompanhe o andamento pela barra de status no canto inferior da janela.

# Estrutura de pastas do projeto

No Package Explorer podemos verificar a estrutura de pastas (pacotes) de nosso projeto.

- **src/main/java** – pacote onde incluimos as Classes, Interfaces, etc da nossa aplicação (*Design Patterns*);
- **ApiApplication.java** – Classe com método main para rodar nossa aplicação;
- **src/main/resources** – pacote para inclusão de outros recursos para a aplicação, como por exemplo, controle de migrações com banco de dados;
- **static** – é onde ficam as configurações estáticas da aplicação web, como arquivos de csv, JavaScript e imagens.
- **templates** – é onde ficam os templates HTML, as páginas do projeto.

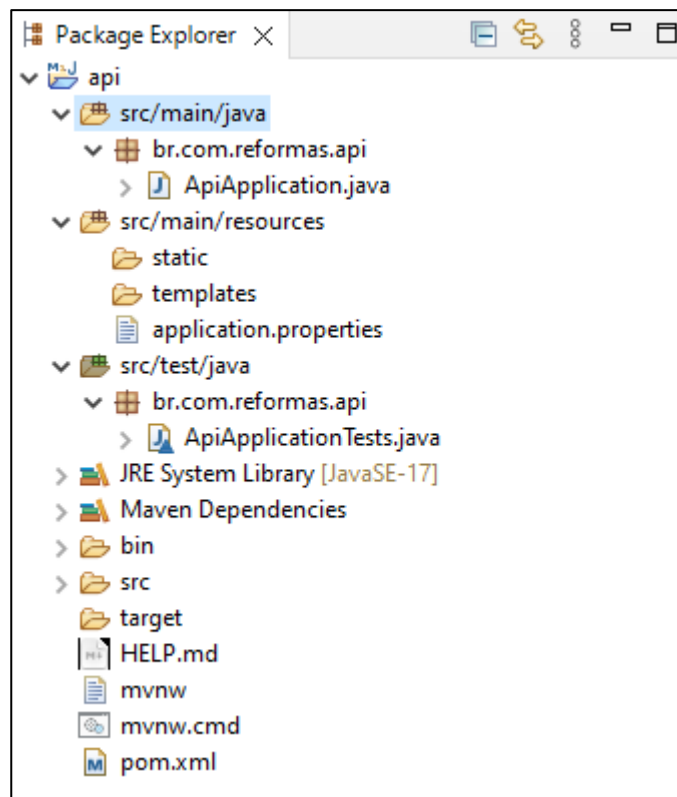


# Estrutura de pastas do projeto

No Package Explorer podemos verificar a estrutura de pastas (pacotes) de nosso projeto.

- **application.properties** – arquivo onde preenchemos as configurações da aplicação, como por exemplo, login e senha para acesso ao banco de dados.
- **src/test/java** – temos a classe `ApiApplicationTest.java` (que atualmente é um teste que está vazio) que serve para realizar teste automatizados.
- **pom.xml** – arquivo XML com, principalmente, as dependências de nossa aplicação.

Essa é a estrutura de diretórios de um projeto com Spring Boot. Como estamos usando o Maven, é por isso que a estrutura de diretórios está organizada dessa forma.





Abra e analise o arquivo **pom.xml**

```
api/pom.xml X
10     </parent>
11     <groupId>br.com.reformas</groupId>
12     <artifactId>api</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>api</name>
15     <description>API Rest da aplicação de reformas.com.br</description>
16     <properties>
17         <java.version>17</java.version>
18     </properties>
19     <dependencies>
20         <dependency>
21             <groupId>org.springframework.boot</groupId>
22             <artifactId>spring-boot-starter-web</artifactId>
23         </dependency>
24
25         <dependency>
26             <groupId>org.springframework.boot</groupId>
27             <artifactId>spring-boot-devtools</artifactId>
28             <scope>runtime</scope>
29             <optional>true</optional>
30         </dependency>
31         <dependency>
32             <groupId>org.springframework.boot</groupId>
33             <artifactId>spring-boot-starter-test</artifactId>
34             <scope>test</scope>
35         </dependency>
36     </dependencies>
37
38     <build>
```

# A Classe ApiApplication

Abra a classe **ApiApplication** e observe que ele possui o método main para execução da aplicação.

```
ApiApplication.java ×
1 package br.com.reformas.api;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class ApiApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(ApiApplication.class, args);
11     }
12
13 }
```

# Executando o projeto

# Executando o projeto

Para executar a aplicação basta rodar a classe **ApiApplication**.

E essa é uma diferença entre aplicações web tradicionais e aplicações usando o Spring Boot.

Nas aplicações web tradicionais, tínhamos que adicionar um servidor de aplicações (como por exemplo, *TomCat*, *Jetty*, *GlassFish*, etc), colocar a aplicação dentro desse servidor para só depois inicializar o servidor e ver a aplicação funcionando.

No Spring Boot o processo foi invertido. Temos o servidor incluído dentro da aplicação (que por padrão, é o *TomCat*)

O projeto vem com o *TomCat* como servidor de aplicação e ele já está embutido dentro das dependências do módulo web. Ele não aparece no arquivo pom.xml porque está no xml do Spring Boot herdado.

# Executando o projeto

Rode a classe **ApiApplication** e observe o que aparece no **Console**.



```
Problems @ Javadoc Console X Servers
ApiApplication (3) [Java Application] C:\Program Files\Java\bin\javaw.exe (4 de fev. de 2023 13:54:44) [pid: 7748]

:: Spring Boot :: (v3.0.2)

2023-02-04T13:54:46.642-03:00 INFO 7748 --- [ restartedMain] br.com.reformas.api.ApiApplication : Starting ApiApplication using Java 17.0.2 with
2023-02-04T13:54:46.646-03:00 INFO 7748 --- [ restartedMain] br.com.reformas.api.ApiApplication : No active profile set, falling back to 1 default profile
2023-02-04T13:54:46.747-03:00 INFO 7748 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.restart.enabled' to 'false' to
2023-02-04T13:54:46.748-03:00 INFO 7748 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the following system properties: 'spring
2023-02-04T13:54:48.327-03:00 INFO 7748 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-02-04T13:54:48.342-03:00 INFO 7748 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-02-04T13:54:48.342-03:00 INFO 7748 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.16]
2023-02-04T13:54:48.432-03:00 INFO 7748 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-02-04T13:54:48.435-03:00 INFO 7748 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2023-02-04T13:54:48.896-03:00 INFO 7748 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2023-02-04T13:54:48.939-03:00 INFO 7748 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with 1 thread
2023-02-04T13:54:48.951-03:00 INFO 7748 --- [ restartedMain] br.com.reformas.api.ApiApplication : Started ApiApplication in 2.763 seconds (process running)
```

Temos várias informações importantes como, por exemplo, a versão do Java utilizado, que o servidor Tomcat foi inicializado e que a aplicação está utilizando a porta 8080 (do protocolo http).

A aplicação vai continuar em execução até que clicamos no botão **stop** (quadrado vermelho) na própria janela do **Console**.

# Executando o projeto

Para testar o funcionamento da aplicação, abra o Navegador e digite na barra de endereços **localhost:8080** e observe a página que é exibida.

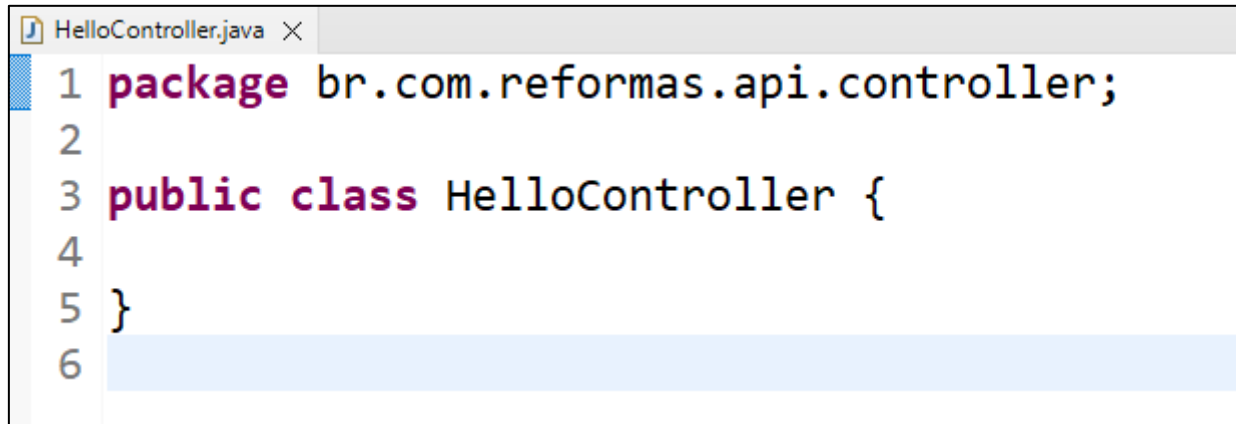


A mensagem informa que a aplicação foi carregada e que foi recebida a requisição, mas não há nenhum controller, nem endereço mapeado. Assim, retornou o erro 404.

Essa mensagem era esperada, isso significa que funcionou. Isso porque não mapeamos nenhuma URL no projeto. Agora criaremos o controller e fazer o Hello World.

# ! Hello World

Dentro do pacote **br.com.reformas.api** vamos criar o pacote **controller** e dentro dele a classe **HelloController**.

A screenshot of a code editor window titled 'HelloController.java'. The code is written in Java and defines a package and a class. The lines are numbered 1 through 6 on the left margin. Line 1: 'package br.com.reformas.api.controller;' Line 2: (empty) Line 3: 'public class HelloController {' Line 4: (empty) Line 5: '}' Line 6: (empty, highlighted with a light blue background).

```
1 package br.com.reformas.api.controller;  
2  
3 public class HelloController {  
4  
5 }  
6
```

Observe que após a criação desta página a aplicação vai reinicializar o servidor (isso vai acontecer a cada modificação que fizermos em nosso projeto).

# Hello World

Na classe **HelloController**, não há nada do *Spring Boot* e sim o **Spring MVC** (*Spring Web model-view-controller*).

Para comunicarmos o **Spring MVC** que é uma classe controller, acima dela incluiremos a anotação **@RestController** (pois estamos trabalhando com uma API Rest).

Outra anotação para incluir é a **@RequestMapping**. Isso informa qual a URL que esse controller vai responder, que será **/hello**. Assim, ao chegar uma requisição para **localhost:8080/hello** vai cair neste controller.

```
HelloController.java X
1 package br.com.reformas.api.controller;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7 @RequestMapping("/hello")
8 public class HelloController {
9
10 }
```



# Hello World

No controller, é necessário chamarmos algum método. Criaremos o método `public String olaMundo(){}.` Dentro dele, vamos retorna a string "Hello World!".

Precisamos colocar uma anotação **@GetMapping** que indique ao protocolo HTTP deve chamar este método. Se chegar uma requisição em **/hello** e ela é do tipo **get**, deverá chamar o método `olaMundo()`.

```
1 package br.com.reformas.api.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 @RequestMapping("/hello")
9 public class HelloController {
10
11     @GetMapping
12     public String olaMundo() {
13         return "Hello World!";
14     }
15 }
```

# Hello World

Digite no navegador na barra de endereços **localhost:8080/hello** e observe o resultado:



Praticando...

1. Crie uma outra classe chamada **UniverseController** e faça o mapeamento para essa classe no endereço **/universe**. Crie dentro dessa classe um método que retorne a *String* **"Hello Universe!"**.
2. Crie uma outra classe chamada **ChuckNorrisController** e faça o mapeamento para essa classe no endereço **/chuck**. Crie dentro dessa classe um método que retorne a *String* **"Hello Chuck Norris!"**.

1. Em relação ao **Spring Boot**, escolha a alternativa correta.
  - a) Spring e Spring Boot são a mesma coisa.
  - b) Existem diversos módulos no Spring Boot como MVC, Security e Transactions.
  - c) Spring Boot foi criado com o propósito de agilizar a criação de um projeto que utilize o Spring como framework.
  - d) Um novo projeto com o Spring Boot é uma tarefa um tanto quanto complicada, devido a necessidade de realizar várias configurações no projeto.
  
2. Qual anotação devemos utilizar para que o Spring reconheça a classe como um Controller quando estamos desenvolvendo uma API Rest?
  - a) @RequestMapping
  - b) @RestController
  - c) @GetMapping
  - d) @PostMapping





Java com Spring, Hibernate e Eclipse. Anil Hemrajani. Pearson, 2013.

Spring MVC: Domine o principal framework web Java. Alberto Souza. Série Caelum.

Site: [alura.com.br](http://alura.com.br)

## Até breve!

1. Em relação ao **Spring Boot**, escolha a alternativa correta.
  - a) Spring e Spring Boot são a mesma coisa.
  - b) Existem diversos módulos no Spring Boot como MVC, Security e Transactions.
  -  c) Spring Boot foi criado com o propósito de agilizar a criação de um projeto que utilize o Spring como framework.
  - d) Um novo projeto com o Spring Boot é uma tarefa um tanto quanto complicada, devido a necessidade de realizar várias configurações no projeto.
  
2. Qual anotação devemos utilizar para que o Spring reconheça a classe como um Controller quando estamos desenvolvendo uma API Rest?
  - a) @RequestMapping
  -  b) @RestController
  - c) @GetMapping
  - d) @PostMapping