



*Serviço Nacional de Aprendizagem Industrial*

**PELO FUTURO DO TRABALHO**

# Introdução ao Spring Boot

## Desenvolvimento de Sistemas

Prof. Me. Reneilson Santos

Março/2024

# Agenda

- IDEs
- Build de Sistemas
- Spring Boot
- Criando API Rest com SpringBoot



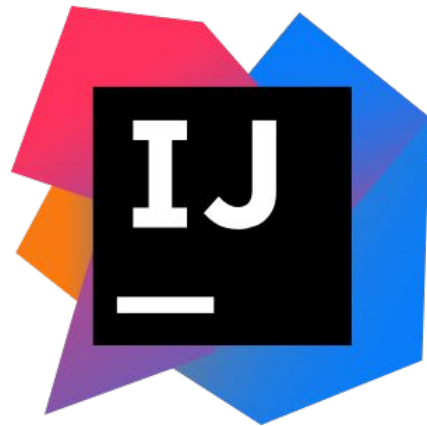
**IDEs**

# IDEs

São ambientes de desenvolvimento integrados que facilitam a implementação do nosso código, já permitindo:

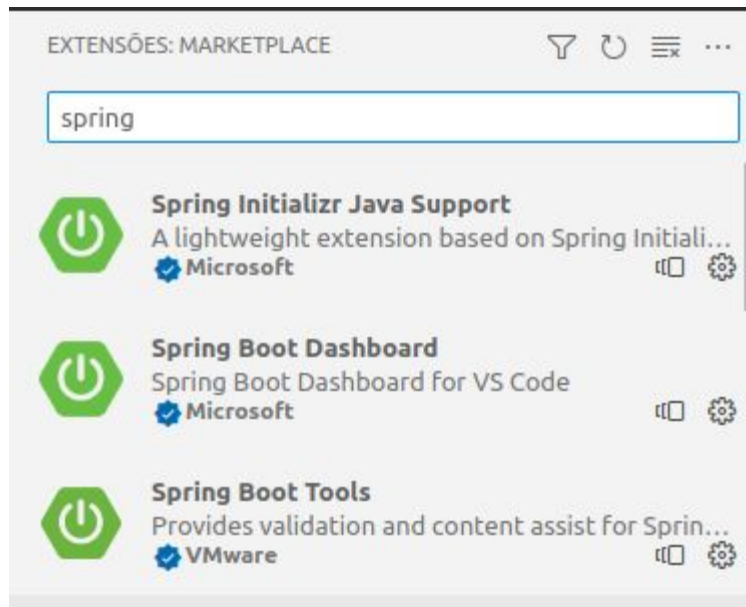
- **Editar** o código com destaques para palavras chaves da linguagem utilizada;
- **Compilar** o código ou executá-lo em tempo real;
- **Depurar** o código utilizando debuggers conectados ou não a navegadores (dependendo do tipo de código implementado)
- **Modelar os dados**, permitindo a criação de classes, métodos, interfaces, de maneira dinâmica;
- **Testar** o nosso código com a integração com algum ambiente de testes, produzindo, por exemplo, relatórios de testes automatizados;

# IDEs



# Extensão Spring Boot para VSCode

# Extensões para o Spring Boot no VSCode



- **Spring Initializr Java Support:** semelhante ao site, para auxiliar na inicialização de um projeto Spring;
- **Spring Boot Dashboard:** auxilia na visualização de classes do projeto;
- **Spring Boot Tools:** traz ferramenta que nos permite inicializar o projeto, validar o código, etc.



# Build de Sistemas

# Build de Sistemas

- Checagem de Dependências
- Validação do Código
- Coleta de Métricas
- Execução de Testes
- Compilação
- Empacotamento



# Spring Boot

# Spring Boot



- O problema de inicialização de projetos e configuração de frameworks, como visto, é justamente uma das maiores dificuldades para uso de Frameworks.
- O Spring Boot surgiu para **facilitar o processo de inicialização e configuração de projetos** de desenvolvimento web Java, já entregando ao desenvolvedor toda a configuração necessária para o bom desenvolvimento do software, além de "esconder" ou "mesclar" configurações, sempre com intuito de deixar o desenvolvimento mais *focado na lógica do negócio e não em problemas de configuração.*

# Spring Boot



O Spring Boot usa a configuração baseada em convenções, o que significa que ele **assume uma configuração padrão para a maioria dos aplicativos, eliminando a necessidade de configurar cada aspecto do aplicativo manualmente.**

Ele também é altamente modular e permite que os desenvolvedores adicionem facilmente recursos adicionais, como **bancos de dados e segurança, com o mínimo de configuração.**

# Spring Boot





Outra vantagem do Spring Boot é sua capacidade de **integrar-se perfeitamente com outras tecnologias**, como o Spring Data, o Spring Security e o Spring Cloud.

Ele também **inclui um servidor web incorporado**, o que significa que os desenvolvedores podem criar e executar aplicativos web *sem precisar de um servidor web externo*.

O Spring Boot é amplamente adotado pela comunidade Java e é amplamente utilizado em aplicativos empresariais em todo o mundo. Ele oferece muitos recursos e benefícios para os desenvolvedores, tornando o processo de desenvolvimento de aplicativos mais eficiente e fácil.

# Criando API Rest com SpringBoot

# Spring Boot Start (<https://start.spring.io>)



**Project**

- ☒ Gradle - Groovy
- ☐ Gradle - Kotlin
- ☐ Maven

**Language**

- ☒ Java ☐ Kotlin
- ☐ Groovy

**Spring Boot**

- ☐ 3.2.0 (SNAPSHOT) ☐ 3.2.0 (M2) ☐ 3.1.4 (SNAPSHOT)
- ☒ 3.1.3 ☐ 3.0.11 (SNAPSHOT) ☐ 3.0.10
- ☐ 2.7.16 (SNAPSHOT) ☐ 2.7.15

**Project Metadata**



Group

Artifact

**Dependencies**

ADD DEPENDENCIES... CTRL + B



No dependency selected



GENERATE CTRL + ↵

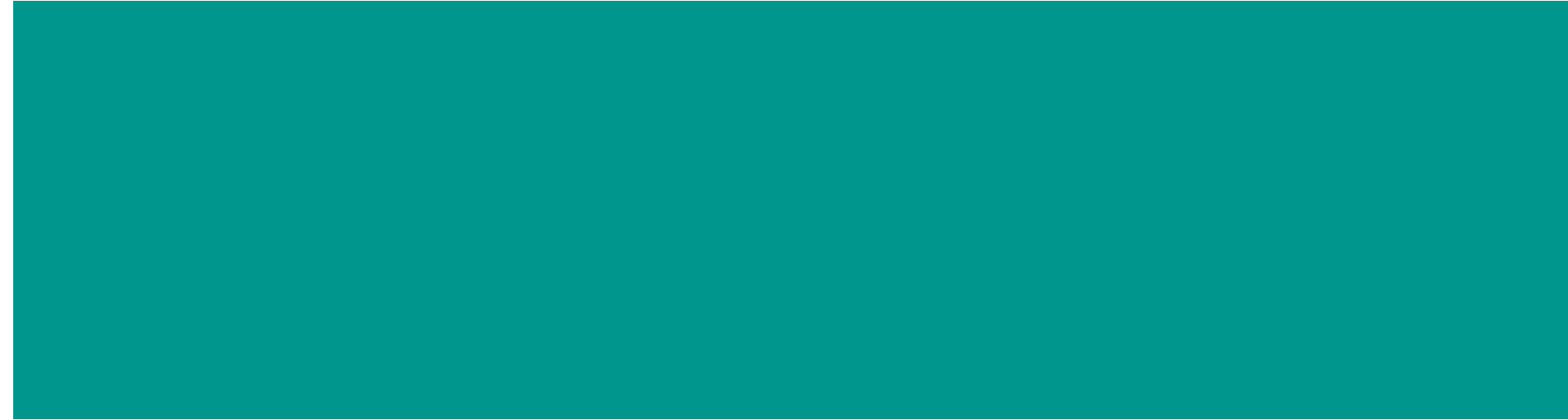
EXPLORE CTRL + SPACE

SHARE...





# Entendendo o Projeto



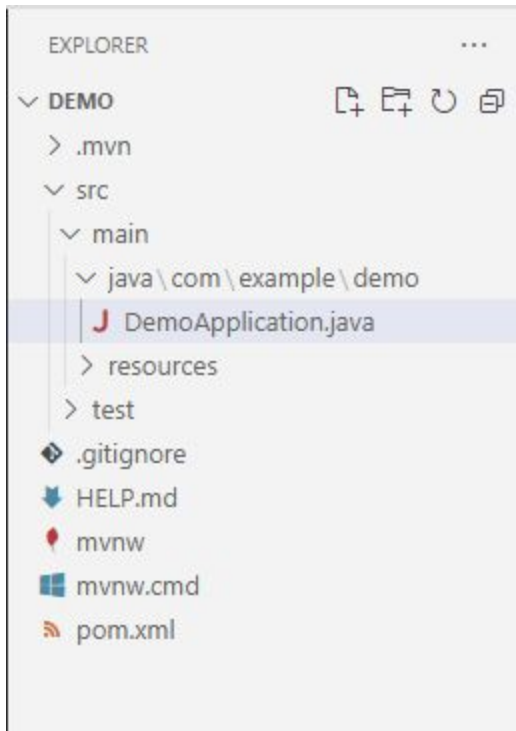
# Classe Principal e Método *main*

O método *main* é, como o próprio nome diz, o *principal* método de um código JAVA. É através dele que o programa/software é iniciado.

Ou seja, este é o primeiro método que será executado quando a aplicação for inicializada, então ela deve conter a chamada para demais funções do seu código ou permitir "ouvir" entradas do usuário para que algo seja realizado dentro do software.

No caso do Spring, a função *main* pode simplesmente executar a aplicação Spring usando o comando:

***SpringApplication.run([Nome\_da\_Classe].class, args);***



DemoApplication.java ×

src > main > java > com > example > demo > DemoApplication.java

```
1  package com.example.demo;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6  @SpringBootApplication
7  public class DemoApplication {
8
9      public static void main(String[] args) {
10         SpringApplication.run(DemoApplication.class, args);
11     }
12
13 }
14
```

# Annotation @SpringApplication

Em uma aplicação Spring Boot, deve-se utilizar a diretiva

## @SpringApplication

na classe principal, ou seja, a classe que conterá o método *main*.

Essa *annotation* diz para o sistema que aquela classe:

- É uma classe de configuração;
- Permite escanear os diferentes componentes que existam na sua aplicação;
- Realiza a auto configuração da aplicação Spring Boot.

Logo, a classe *principal* terá dois propósitos, inicialização e configuração.

# Maven - Arquivo pom.xml

O Maven é um gerenciador de build e dependências baseado no conceito de *project object model* (POM). Traduzindo, ele permite configurar as dependências dos projetos apontando para os identificadores das mesmas num arquivo chamado pom.xml.

Dessa forma, não faz-se necessário procurar os arquivos jars das bibliotecas para adicioná-los ao classpath do projeto. O próprio Maven encarrega-se disso, só faz-se necessário informar o identificador daquela dependência para que ele saiba quem baixar.

Além disso, ele gerencia os procedimentos que devem ser executados durante o build da sua aplicação.

EXPLORER

...

DEMO

> .idea

> .mvn

> src

> main

> java\com\example\demo

DemoApplication.java

> resources

> test

.gitignore

HELP.md

mvnw

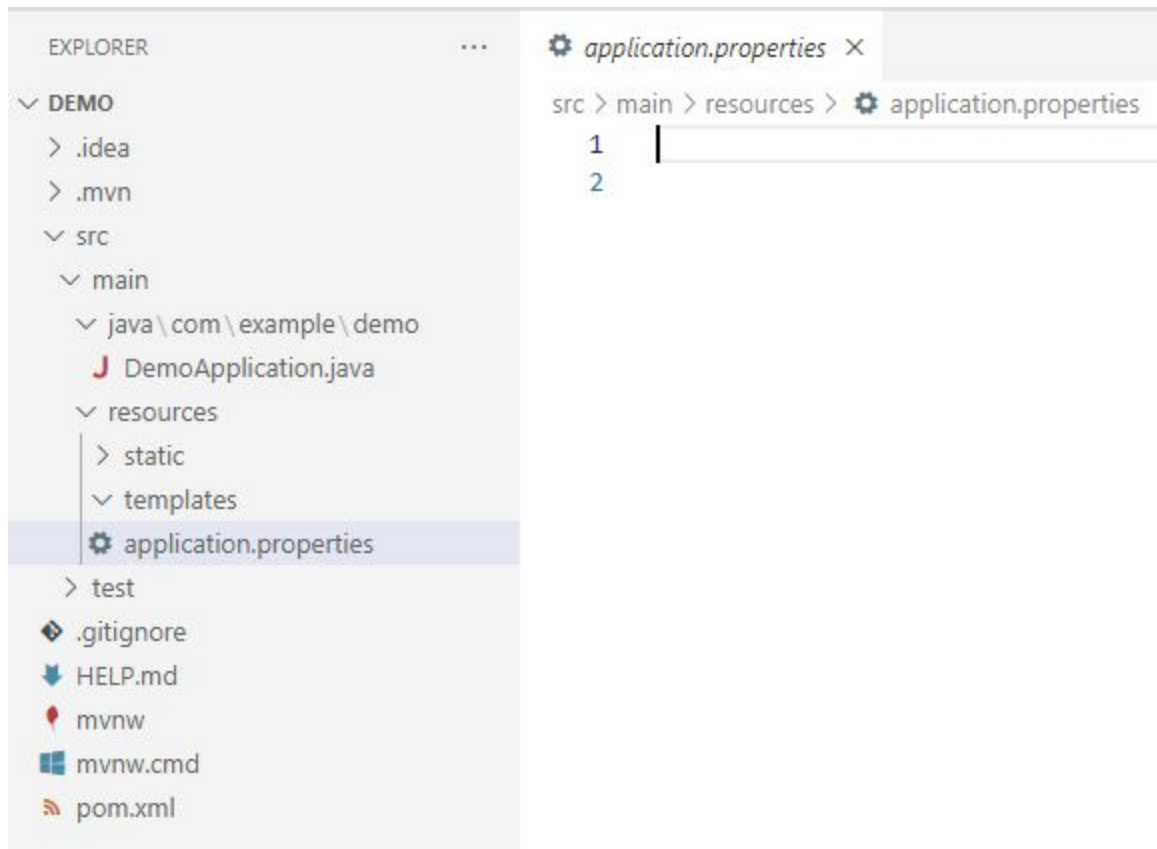
mvnw.cmd

pom.xml

pom.xml

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.0.4</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>com.example</groupId>
12  <artifactId>demo</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>demo</name>
15  <description>Demo project for Spring Boot</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>
```



Outro arquivo que será bastante utilizado no nosso projeto Spring Boot é o arquivo `application.properties`, localizado na pasta `resources`.

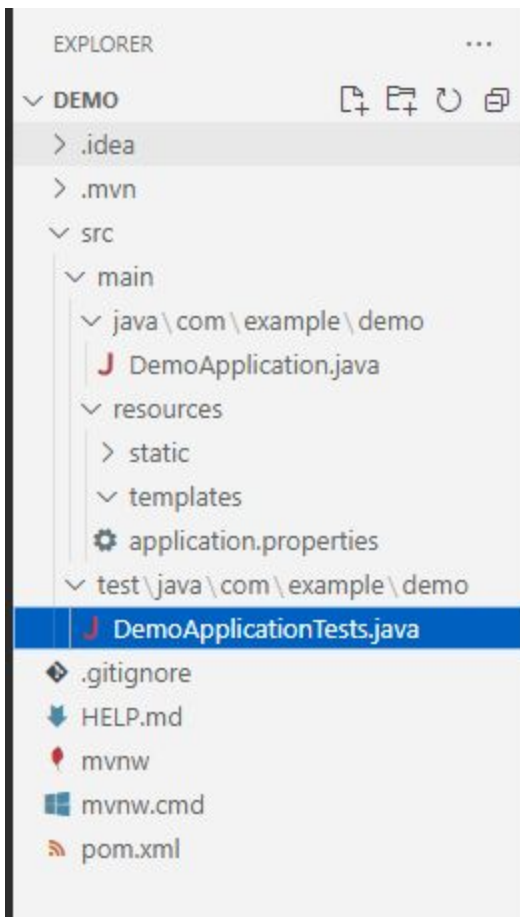
Esse arquivo é responsável pela configuração do nosso app (acesso a banco de dados, dados de segurança ou ainda variáveis “globais”).

Por padrão esse arquivo vem vazio.

# Arquivos de Testes

Por padrão, um projeto criado pelo Initializer do SpringBoot cria para nós um arquivo de testes padrão também, ou seja, algumas bibliotecas de testes já estão incluídas como padrão no nosso projeto e pode ser utilizada para criação de testes unitários.





DemoApplicationTests.java

src > test > java > com > example > demo > DemoApplicationTests.java

```
1 package com.example.demo;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.boot.test.context.SpringBootTest;
5
6 @SpringBootTest
7 class DemoApplicationTests {
8
9     @Test
10     void contextLoads() {
11     }
12
13 }
14
```

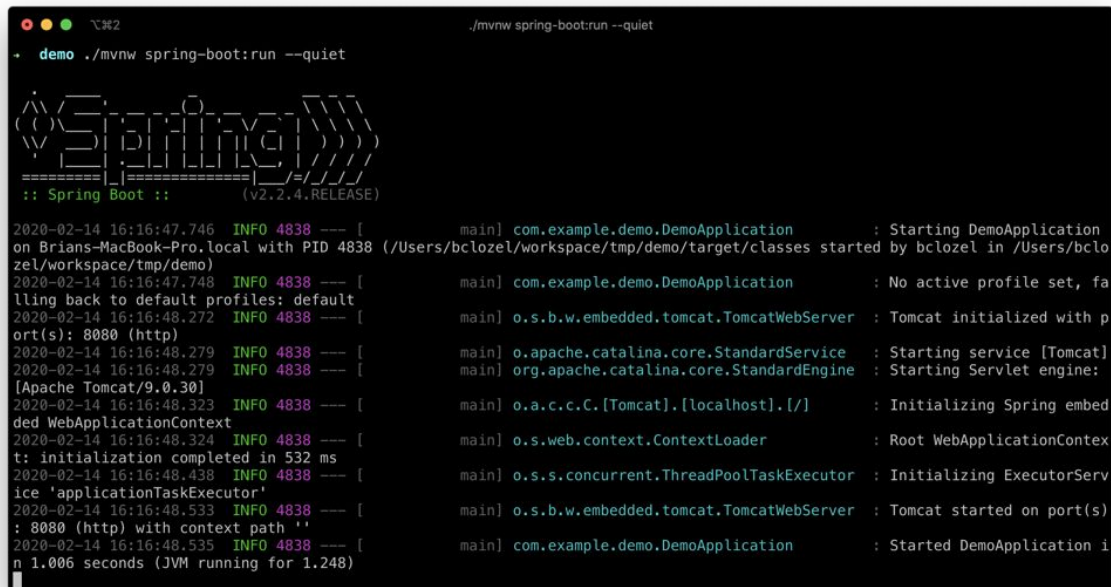
# Start Backend - Maven

`./mvnw spring-boot:run`

- *Linux/Mac*

`mvnw spring-boot:run`

- *Windows*



```
demo ./mvnw spring-boot:run --quiet

  ____ _
 / ___ \| | | |
/ /   \| |_| |
\ \   /| | | |
 \___/\_|_|_|_|
:: Spring Boot :: (v2.2.4.RELEASE)

2020-02-14 16:16:47.746 INFO 4838 --- [main] com.example.demo.DemoApplication : Starting DemoApplication
on Brians-MacBook-Pro.local with PID 4838 (/Users/bclozel/workspace/tmp/demo/target/classes started by bclozel in /Users/bclozel/workspace/tmp/demo)
2020-02-14 16:16:47.748 INFO 4838 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to default profiles: default
2020-02-14 16:16:48.272 INFO 4838 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-02-14 16:16:48.279 INFO 4838 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-02-14 16:16:48.279 INFO 4838 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.30]
2020-02-14 16:16:48.323 INFO 4838 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2020-02-14 16:16:48.324 INFO 4838 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 532 ms
2020-02-14 16:16:48.438 INFO 4838 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2020-02-14 16:16:48.533 INFO 4838 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s) 8080 (http) with context path ''
2020-02-14 16:16:48.535 INFO 4838 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 1.006 seconds (JVM running for 1.248)
```

# Start Backend - Gradle

```
./gradlew bootRun
```

- *Linux/Mac*

gradlew bootRun

- *Windows*

```
reneilson@reneilson-Lenovo-ideapad-330-15IKB:~/Downloads/primeiro_projeto$ ./gradlew bootRun
> Task :bootRun

  ____ _
 / ___ \| | | |
/ /___ \| |_| |
\___)___|_____|
:: Spring Boot ::                (v2.4.2)

2021-02-10 20:39:11.901 INFO 17576 --- [ restartedMain] c.e.p.PrimeiroProjetoApplication : Starting PrimeiroProjetoAppl
ication using Java 11.0.9.1 on reneilson-Lenovo-ideapad-330-15IKB with PID 17576 (/home/reneilson/Downloads/primeiro_projeto/bui
d/classes/java/main started by reneilson in /home/reneilson/Downloads/primeiro_projeto)
2021-02-10 20:39:11.905 INFO 17576 --- [ restartedMain] c.e.p.PrimeiroProjetoApplication : No active profile set, falli
ng back to default profiles: default
2021-02-10 20:39:11.977 INFO 17576 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults a
ctive! Set 'spring.devtools.add-properties' to 'false' to disable
2021-02-10 20:39:11.977 INFO 17576 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related l
ogging consider setting the 'logging.level.web' property to 'DEBUG'
2021-02-10 20:39:12.715 INFO 17576 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port
(s): 8080 (http)
2021-02-10 20:39:12.734 INFO 17576 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
```

## Execução

Windows PowerShell

Copyright (C) Microsoft Corporation. Todos os direitos reservados.

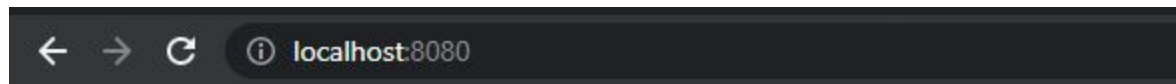
Experimente a nova plataforma cruzada PowerShell <https://aka.ms/pscore6>

```
PS C:\Users\Aluno\Downloads\demo\demo> & 'C:\Users\Aluno\.vscode\extensions\redhat.java-1.15.0-win32-x64\jre\17.0.6-win32-x86_64\bin\java.exe' '@C:\Users\Aluno\AppData\Local\Temp\cp_7llwgudmbtth5mrwm0kym5w1n.argfile' 'com.example.demo.DemoApplication'
```

[illegible]

```
2023-03-09T17:25:00.874-03:00 INFO 5392 --- [ restartedMain] com.example.demo.DemoApplication : Starting DemoApplication using Java 17.0.6 with PID 5392
(C:\Users\Aluno\Downloads\demo\demo\target\classes started by Aluno in C:\Users\Aluno\Downloads\demo\demo)
2023-03-09T17:25:00.878-03:00 INFO 5392 --- [ restartedMain] com.example.demo.DemoApplication : No active profile set, falling back to 1 default profile:
"default"
2023-03-09T17:25:00.943-03:00 INFO 5392 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.a
dd-properties' to 'false' to disable
```

# Execução



## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Thu Mar 09 17:25:10 BRT 2023

There was an unexpected error (type=Not Found, status=404).

No message available

# Hello World



# Controlador

Para trabalhar com uma API Rest, ou seja, uma API que permita utilizar os métodos padrões do HTTP:

**GET, POST, PATCH/PUT, DELETE**

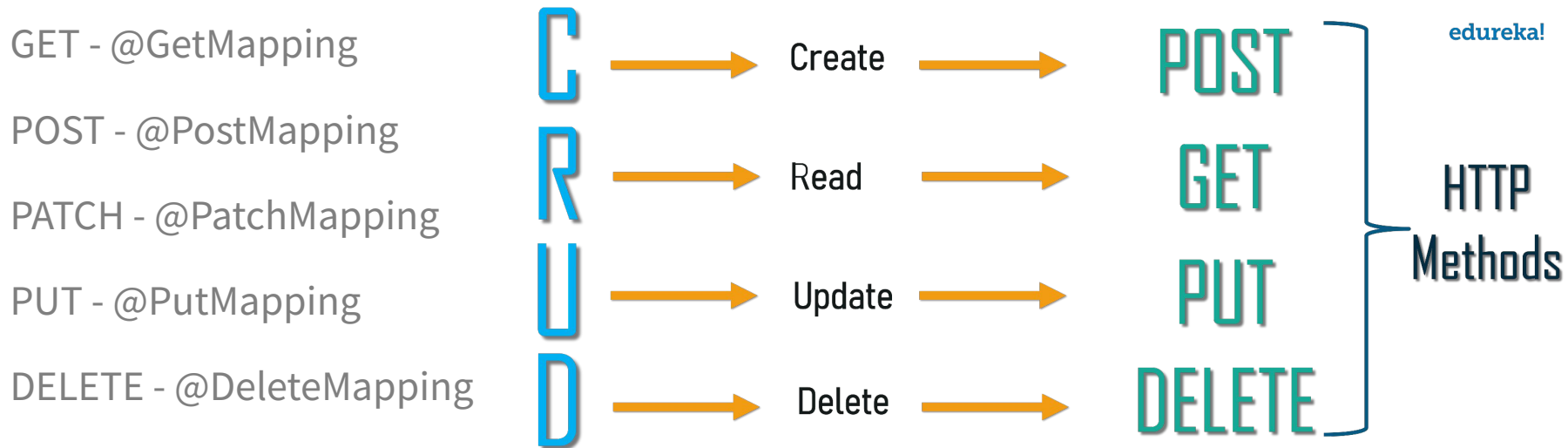
É preciso criar classes que sejam controladores utilizando a *annotation*

**@RestController**

Essa *annotation* fará com que todos os métodos implementados possam retornar um **ResponseBody**, ou seja, retornar uma resposta HTTP.

# Annotations para métodos HTTP

Para cada um dos métodos HTTP existe uma *annotation* equivalente no Spring:





# Caminho

Para indicar o "caminho" ou seja, a URL que deve ser utilizada para acessar os métodos HTTP do controlador, adiciona-se a *annotation* `@RequestMapping` com o endereço:

**`@RequestMapping("/hello")`**

Ou pode-se aplicar o caminho diretamente em cada método da mesma forma mas na *annotation* do método:

**`@GetMapping("/hello")`**

```
@RestController
@RequestMapping("/hello")
public class Hello {
    @GetMapping
    public String hello() {
        return "Hello World";
    }
}
```

# Parâmetros Através da URL

Dentro dos métodos que definimos como GET podemos passar parâmetros utilizando a diretiva:

**@RequestParam(value="param", defaultValue="Meu valor padrão", required=false)**

Antes do nome da variável do nosso método, dessa forma, quando usado dentro do método essa variável terá o valor que recebeu a partir da URL ou seu valor padrão passado pelo *defaultValue*.

<http://localhost:8080/home?param=Meu novo valor>

**@GetMapping("home")**

**public String metodo(@RequestParam(value="param") String parametro){ ... }**

```
@RestController
@RequestMapping ("/hello")
public class Hello {
    @GetMapping
    public String hello (@RequestParam (value="name", defaultValue="") String name ) {
        return "Hello World. <br> Welcome " + name;
    }
}
```

# Valor inválido

Pode-se definir parâmetros opcionais dentro dos métodos de retorno no código, neste caso, utiliza-se o parâmetro **required** e passa o valor *false* para ele dentro da *annotation* `@RequestParam`.

Dentro do método pode-se verificar se o valor recebido é nulo e fazer alguma operação distinta.

Exemplo:

```
(@RequestParam(value = "nome", required=false) String nome) ...
```

```
if (nome == null){ ... } else { ... }
```

## Exercício

**Faça um método GET que ao receber M-matutino, V-Vespertino ou N- Noturno imprima a mensagem "Bom Dia!", "Boa Tarde!" ou "Boa Noite!" ou "Valor Inválido!", conforme o caso.**

---