

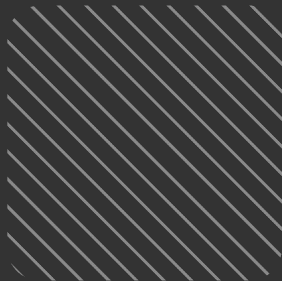


Spring Boot

//Configurações e primeiros passos

IT BOARDING

BOOTCAMP



Índice



01 Introdução a MVC

02 Criação de uma API

03 @GetMapping +
@PathVariable

IT BOARDING

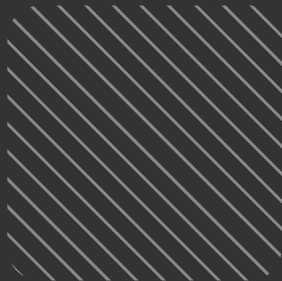
BOOTCAMP

Padrão MVC

//Conceitos básicos

IT BOARDING

BOOTCAMP



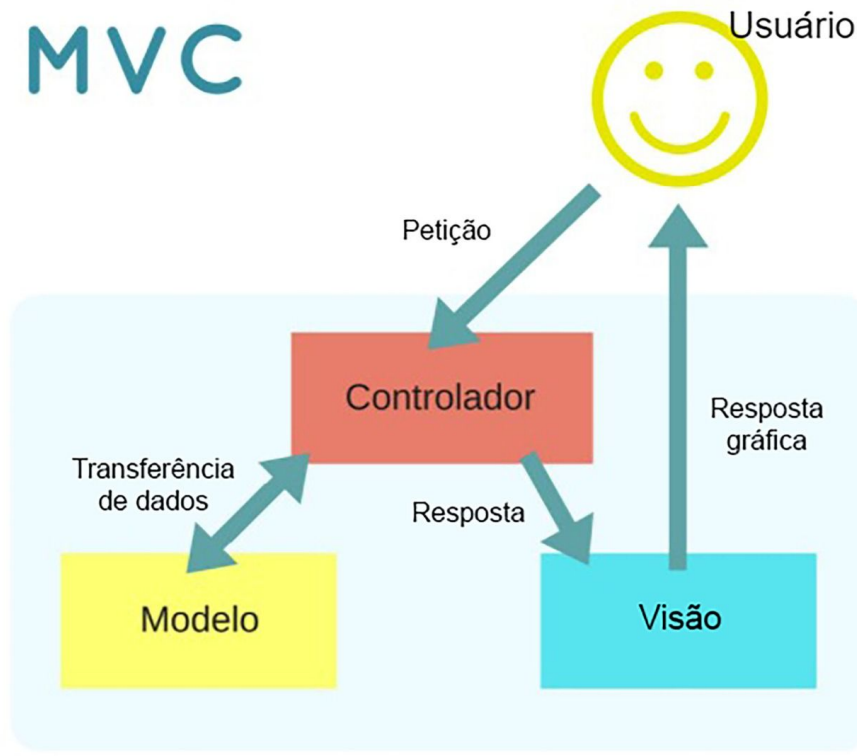


// MVC

- O **Model-View-Controller** é um padrão de arquitetura de software que separa a **lógica de negócios**, da **lógica de visualização** em uma aplicação.
- **Modelo:** Se encarrega dos dados, geralmente (mas não necessariamente) consultando um banco de dados.
- **Controller:** É responsável por "controlar"; recebe os pedidos do usuário, solicita os dados do modelo e os comunica à vista de todos.
- **View:** É a representação visual dos dados



MVC





Como o MVC é aplicado em um aplicativo da web?

- **Modelo:**

- É composto pelo **conjunto de classes** que utilizamos para realizar a modelagem correspondente (lógica de negócios e banco de dados, etc.) de nossa aplicação.

- **View:**

- É representado pelo Frontend, onde podemos ter um arquivo HTML ou um JSP (Java Server Pages) que serve de interface gráfica para que o usuário obtenha suas solicitações.

- **Controller:**

- O controlador é o **ponto de união entre o modelo e a vista** e como o seu nome indica é responsável por controlar a vista que deve mostrar, os dados que nela serão incluídos e recolher os dados enviados da vista para aja de acordo (salve-os, faça uma consulta, etc.).
- Para criar um **controlador com Spring Boot** simplesmente temos que adicionar a anotação **@RestController** à classe Java que determinamos para isso.



Criação de uma API

//com Spring Boot

IT BOARDING

BOOTCAMP





// Criando uma API

- Vamos criar uma nova classe Java chamada `HelloRestController` e vamos atribuí-la como nosso controlador usando o Annotation `@RestController`.
- Dentro da classe, vamos criar o método `sayHello()` que retornará uma String "Hello World".
- Vamos marcar esse método com a anotação `@GetMapping`, que nos diz que quando nossa API recebe uma solicitação GET por meio do protocolo HTTP, ela deve retornar o resultado do método `sayHello()`.

```
package com.ejemplo.sayhello;

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

@RestController
public class HolaRestController {

    @GetMapping
    public String sayHello() {

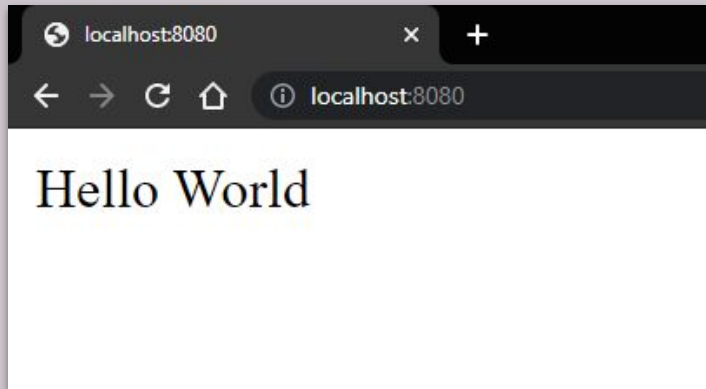
        return "Hello World";
    }
}
```





// Criando uma API

- Feito isso, vamos executar nosso aplicativo e testar em nosso navegador para chamar a URL de nosso aplicativo para verificar se recebemos a resposta de **sayHello()**.
- Geralmente, o URL padrão é **localhost:8080**.
- É **importante** verificar em qual porta o Spring Boot levanta nossa aplicação. Por padrão, sempre será 8080.



```
.e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web'
o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
o.apache.catalina.core.StandardService   : Starting service [Tomcat]
org.apache.catalina.core.StandardEngine   : Starting Servlet engine: [Apache Tomcat/9.0.46]
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 756 ms
o.s.b.d.a.OptionalLiveReloadServer        : LiveReload server is running on port 35729
o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port(s): 8080 (http) with context path ''
c.ejemplo.sayhello.SayHelloApplication    : Started SayHelloApplication in 1.424 seconds (JVM running for 1.795)
o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet         : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet         : Completed initialization in 0 ms
```


Parâmetros

//com **Spring Boot**


IT BOARDING

BOOTCAMP





// Parâmetros

- Um **@GetMapping** pode receber diferentes parâmetros que podemos usar nas funções que declaramos.
- Para fazer isso, vamos modificar nosso **@GetMapping** para indicar como iremos receber o parâmetro. 
- Em seguida, adicionaremos em sayHello() a recepção do parâmetro mencionado. Lembre-se: *o nome da variável que receberá sayHello deve ser igual ao do parâmetro.*

```
@GetMapping("/{name}")  
public String sayHello(@PathVariable String name) {  
    return "Hello World";  
}
```



// Parâmetros

- Agora completamos nosso "Hello World" adicionando o nome que receberemos como parâmetro do navegador.



- Em seguida, rodamos nossa aplicação e do navegador passamos um nome como parâmetro para ver se funciona corretamente.



Com isso, "ensinamos" o Spring a interpretar a URL e como queremos que os parâmetros sejam usados.

```
@GetMapping("/{name}")  
public String sayHello(@PathVariable String name) {  
    return "Hello World: " + name;  
}
```



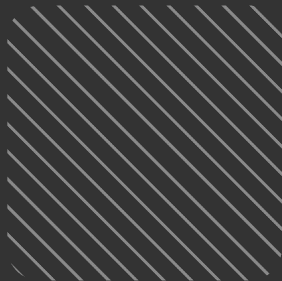


Dicionário

//Anotações de **Spring Boot**

IT BOARDING

BOOTCAMP





// Dicionário / Resumo das Anotações desta aula

- **@SpringBootApplication**: Ele nos permite especificar que trabalhamos em um aplicativo Spring Boot. Permite 3 funcionalidades, a autoconfiguração do projeto (@EnableAutoConfiguration), a procura de componentes / pacotes de aplicações (@ComponentScan) e a possibilidade de efetuar configurações extras (@Configuration).
- **@RestController**: Anotação para identificar o controlador de um serviço do tipo REST.
- **@GetMapping**: Anotação para “mapear” as solicitações por meio do método GET em nosso aplicativo.
- **@GetMapping + @PathVariable**: Anotação para indicar o parâmetro que vamos receber em nosso método.



IT BOARDING

BOOTCAMP



Obrigado



IT BOARDING

BOOTCAMP

