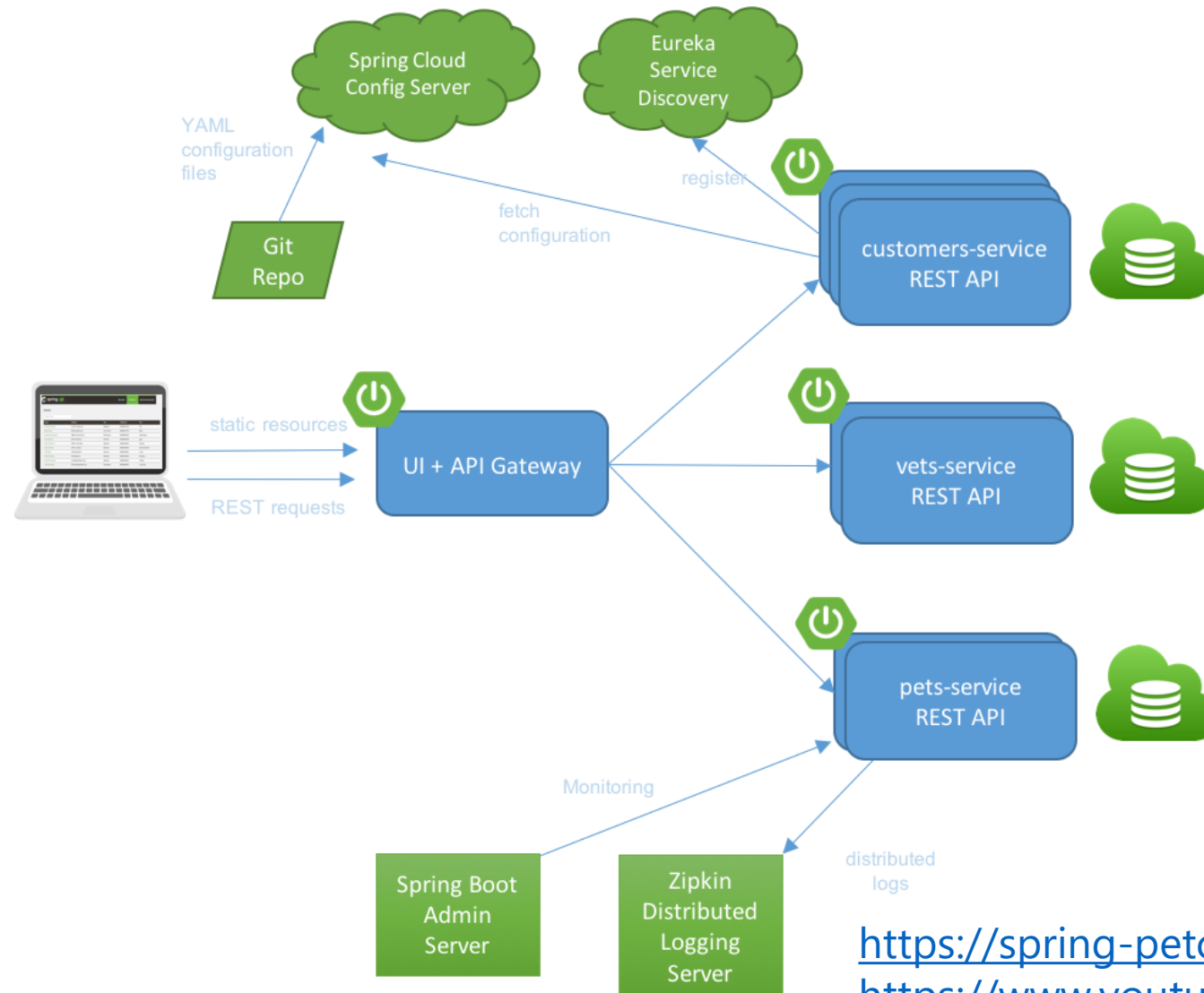


# Tópicos Especiais em Ciência da Computação – TE29S

Professor Dr. Evandro Miguel Kuszera  
[evandrokuszera@utfpr.edu.br](mailto:evandrokuszera@utfpr.edu.br)

# Microserviços

# Microserviços



<https://spring-petclinic.github.io/>

[https://www.youtube.com/watch?v=1PuEKs\\_WAjY](https://www.youtube.com/watch?v=1PuEKs_WAjY)

# Microserviços

---

Microserviços são uma forma de estruturar uma aplicação como um conjunto de serviços **pequenos**, **independentes** e **especializados**, que trabalham **juntos** para realizar funções mais amplas.

Cada serviço é responsável por um conjunto específico de funcionalidades do sistema (gestão de pedidos, gestão de usuários, faturamento, etc).

Geralmente os microserviços se comunicam por meio de uma API (HTTP/REST, gRPC ou mensageria).

# Monólitos vs Microserviços

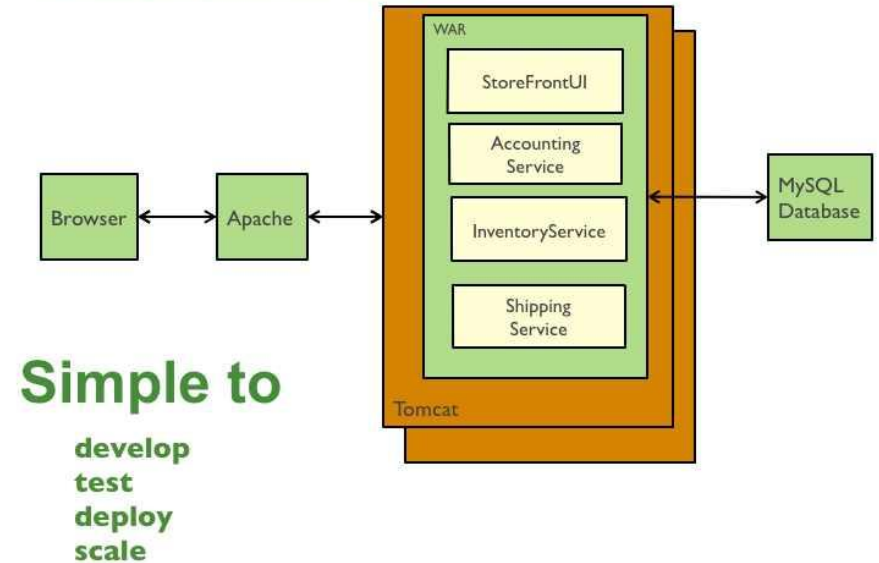
## Arquitetura monolítica:

- Construção do sistema como uma única aplicação coesa.
- Todos os módulos estão interconectados, compartilhando um banco de dados e um servidor.
- Atualizações de pequenas partes da aplicação podem requerer a reimplantação do sistema inteiro, aumentando risco de erros e ocasionando mudanças lentas.

## Microserviços:

- Isolamento de falhas.
- Escalar serviços individualmente.
- Adoção de diferentes tecnologias.
- Modularidade.
- Integração entre sistemas.
- Desenvolvimento distribuído.

Traditional web application architecture



# Desafios dos Microserviços

---

## **Complexidade operacional:**

- Com o aumento do número de serviços, a complexidade de gerenciar a infraestrutura aumenta.
- Cada microserviço pode exigir sua própria infraestrutura, necessitando de orquestração eficiente, como o uso de plataformas de containers como Docker e Kubernetes.

# Desafios dos Microserviços

---

## **Comunicação interserviços:**

- A comunicação entre microserviços pode adicionar latência e complexidade.
- Protocolos de comunicação, como HTTP ou mensageria assíncrona, precisam ser bem projetados para garantir a confiabilidade e a eficiência do sistema como um todo.

# Desafios dos Microserviços

---

## **Gerenciamento de dados distribuídos:**

- Em uma arquitetura de microserviços, os dados frequentemente estão distribuídos entre diferentes serviços, cada um com seu próprio banco de dados.
- Isso pode gerar dificuldades na manutenção da consistência dos dados, especialmente em operações que envolvem múltiplos serviços.



# Desafios dos Microserviços

---

## **Monitoramento e rastreamento:**

- Manter visibilidade e rastreamento de transações em um ambiente distribuído pode ser complicado.
- Ferramentas de monitoramento e logging precisam ser configuradas para garantir que as falhas possam ser diagnosticadas rapidamente.

# Linguagens para Microserviços

---

Some languages that can be used for microservices include: [↗](#)



## Java

A good choice for microservices due to its easy-to-understand annotations syntax. Spring Boot is a popular framework for building microservices in Java. [↗](#)



## Python

An interpreted language that allows developers to write clean, indented code. It also provides access to build APIs. [↗](#)



## Golang

Also known as Go Language, this language is considered to be ideal for microservice-based development. [↗](#)



## Nodejs

An open-source JavaScript runtime environment that is popular for microservices due to its speed, scalability, and ease of deployment. [↗](#)



## NET

A cross-platform language that is supported and maintained by Microsoft. It has built-in Docker containers to help develop microservices. [↗](#)



## C++

A programming language with object-oriented and imperative features that help developers write portable, fast programs. [↗](#)



## Ruby

A scripting language that can be used for microservices, business applications, and more. [↗](#)

# Spring Framework

# Spring Framework

---

**É um framework para desenvolvimento de aplicações Java.**

- Implementa os conceitos de Inversão de Controle (IoC) e Injeção de Dependências (DI).

## **Inversion of Control (IoC)**

- Em vez do programador criar código para instanciar objetos e suas dependências, essa tarefa é passada para um *container* gerenciar.

## **Dependency Injection (DI)**

- Um objeto recebe os objetos que ele depende, assim o comportamento pode ser injetado em outros objetos.

**Spring Framework é o projeto base para diversos outros projetos Spring.**

- <https://spring.io/projects>
- Spring Data, Spring Cloud, Spring Security, etc.

# Inversion of Control (IoC) e Dependency Injection (DI)

---

Em vez da classe ser responsável por criar instâncias de objetos de outras classes de que precisa, essa responsabilidade é atribuída para o container/framework usado.

## Exemplo simples sem IoC

Classe Motorista precisa de um Carro para realizar suas funções.

Classe Motorista controla a criação do objeto Carro, criando **forte acoplamento**.

## Problema:

Se quisermos modificar o tipo de Carro?

CarroHibrido

CarroEletrico

```
public class Carro {  
    public void dirigir() {  
        System.out.println("O carro está em movimento.");  
    }  
}  
  
public class Motorista {  
    private Carro carro;  
  
    public Motorista() {  
        // Instanciação direta do objeto Carro (acoplamento forte)  
        this.carro = new Carro();  
    }  
  
    public void dirigirCarro() {  
        carro.dirigir();  
    }  
}
```

# Inversion of Control (IoC) e Dependency Injection (DI)

---

Com IoC, o Spring é responsável por criar e fornecer instâncias de Carro para a classe Motorista.

Classe deve ser anotada com @Component ou equivalente.

Spring usa Injeção de Dependência (DI) para fornecer um objeto Carro.

A dependência (Carro) é injetada automaticamente.

Tipos de DI: por construtor, por Setter, por campo.

```
import org.springframework.stereotype.Component;

@Component
public class Carro {
    public void dirigir() {
        System.out.println("O carro está em movimento.");
    }
}
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Motorista {

    private Carro carro;

    // Injeção de dependência via construtor
    @Autowired
    public Motorista(Carro carro) {
        this.carro = carro;
    }

    public void dirigirCarro() {
        carro.dirigir();
    }
}
```

# Inversion of Control (IoC) e Dependency Injection (DI)

---

## Usando IoC e DI

Crie as classes Carro e Motorista.

Recupere o contexto do Spring.

Recuperar instância de Motorista via Spring.

Neste ponto, um objeto Carro foi injetado em Motorista.

```
DemoApplication.java x Carro.java Motorista.java
1 package br.edu.utfpr.demo;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @SpringBootApplication
8 @RestController
9 public class DemoApplication {
10
11     public static void main(String[] args) {
12         // Inicializa o Spring Boot e cria o ApplicationContext
13         var context = SpringApplication.run(DemoApplication.class, args);
14
15         // Recuperando um bean gerenciado pelo Spring a partir do contexto
16         Motorista motorista = context.getBean(Motorista.class);
17         motorista.dirigir();
18     }
19
20 }
```

# Spring Boot





# Spring Boot

---

## **De acordo com a documentação:**

- Permite a criação de aplicações “stand-alone”, gerando um único JAR.
- Fornece servidores de aplicação embutidos, como Tomcat, Jetty, etc.
- Fornece dependências 'iniciais' opinativas para simplificar sua configuração de compilação.
- Configura automaticamente as bibliotecas do Spring para trabalhar em conjunto.
- Fornece recursos prontos para produção, como métricas, verificações de integridade e configuração externalizada.
- Nenhuma geração de código e nenhum requisito para configuração XML.

# Criando um projeto Spring Boot

## Requisitos

- Mínimo Java 17 (<https://jdk.java.net/>)
- IntelliJ (<https://www.jetbrains.com/>)

## Acesse Spring Initializr

- <https://start.spring.io/>
- Configure seu projeto e faça o download.



**Project**

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy

☒ Maven

**Spring Boot**

☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M3) ☐ 3.3.5 (SNAPSHOT) ☒ 3.3.4

☐ 3.2.11 (SNAPSHOT) ☐ 3.2.10

**Project Metadata**

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 23 ☒ 21 ☐ 17

# Criando um projeto Spring Boot

## Group

- É usado para identificar de maneira única uma organização ou projeto.
- Ele segue a convenção de pacotes Java (nomes de domínio invertidos).

## Artifact

- Identifica um artefato (biblioteca, jar, war, etc.) específico dentro do grupo.
- Este será o nome do arquivo gerado: demo-version.jar

## Name

- É uma descrição amigável do projeto.
- Em geral, serve para propósitos descritivos e pode aparecer em arquivos de configuração ou relatórios de build.

## Adicionar dependências

- Spring Web: para construir aplicações Web, incluindo RESTful APIs.

## Depois:

- Clique em GENERATE para download.
- Descompacte em um diretório de sua escolha.
- Abra o projeto usando o IntelliJ.



### Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Maven

### Language

☒ Java ☐ Kotlin ☐ Groovy

### Spring Boot

☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M3) ☐ 3.3.5 (SNAPSHOT) ☒ 3.3.4  
☐ 3.2.11 (SNAPSHOT) ☐ 3.2.10

### Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 23 ☐ 21 ☒ 17

### Dependencies

ADD ...

#### Spring Web WEB

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



GENERATE

EXPLORE

SHARE...

# Criando um projeto Spring Boot

Estrutura do Projeto - IntelliJ

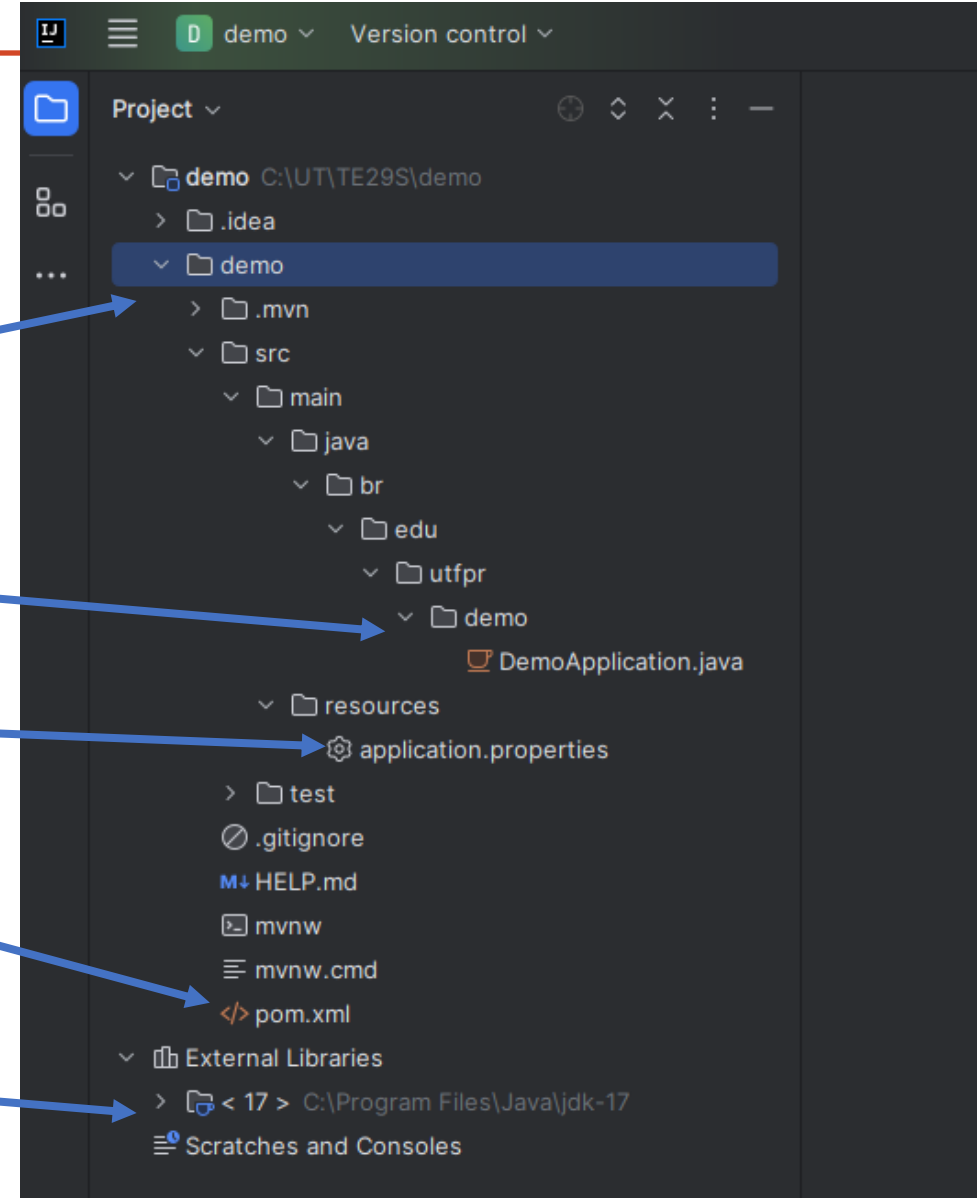
Estrutura de pacotes

Código-fonte Java

Configurações do Projeto

Project Object Model (pom.xml)

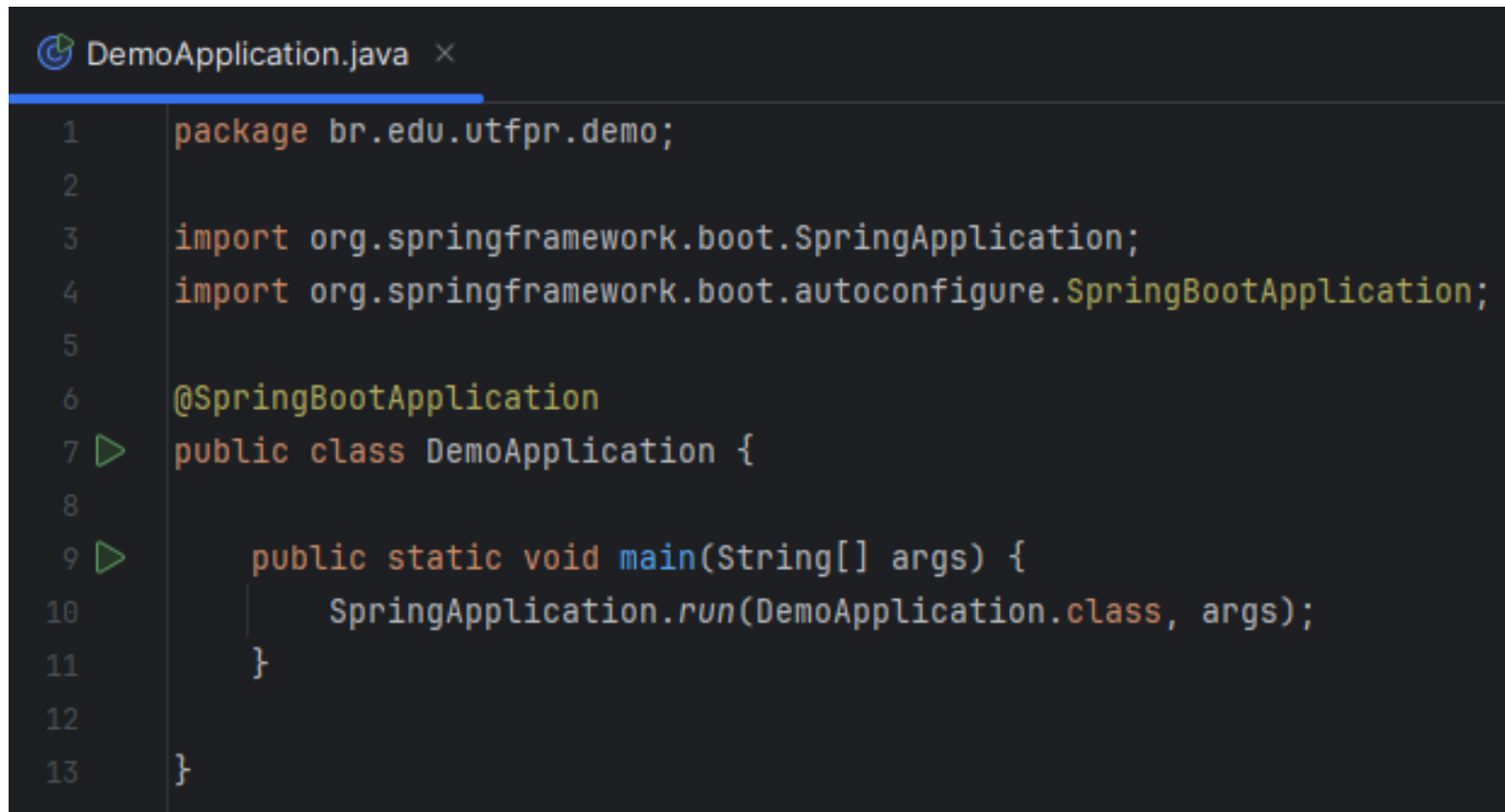
Bibliotecas



# Criando um projeto Spring Boot

---

Classe principal de uma aplicação Spring Boot (ponto de entrada)



```
1 package br.edu.utfpr.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 }
```

# Criando um projeto Spring Boot

---

## Criando um web service

### Endpoint

localhost:8080/hello

Anotação @RestController

Informa ao Spring que o código descreve um endpoint que deve ser disponibilizado para web.

```
DemoApplication.java x
1  package br.edu.utfpr.demo;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RestController;
7
8  @SpringBootApplication
9  @RestController
10 public class DemoApplication {
11
12     public static void main(String[] args) {
13         SpringApplication.run(DemoApplication.class, args);
14     }
15
16     @GetMapping("/hello") no usages
17     public String hello(){
18         return "Hello";
19     }
20 }
```

# Criando um projeto Spring Boot

---

## Endpoint

localhost:8080/hello

ou

localhost:8080/hello?name=Fulano

Anotação @RequestParam

Informa o Spring para esperar um parâmetro chamado "name".

Valor padrão: "World"

```
DemoApplication.java x
1  package br.edu.utfpr.demo;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RequestParam;
7  import org.springframework.web.bind.annotation.RestController;
8
9  @SpringBootApplication
10 @RestController
11 ▶ public class DemoApplication {
12
13 ▶     public static void main(String[] args) {
14         SpringApplication.run(DemoApplication.class, args);
15     }
16
17     @GetMapping("/hello") no usages
18     public String hello(@RequestParam(value="name", defaultValue = "World") String name){
19         return String.format("Hello %s!", name);
20     }
21 }
```

# Tópicos Especiais em Ciência da Computação – TE29S

Professor Dr. Evandro Miguel Kuszera  
[evandrokuszera@utfpr.edu.br](mailto:evandrokuszera@utfpr.edu.br)