



Introdução a Web Services REST com Spring Framework

Prof. Me. Jorge Luís Gregório
www.jlgregorio.com.br



Agenda

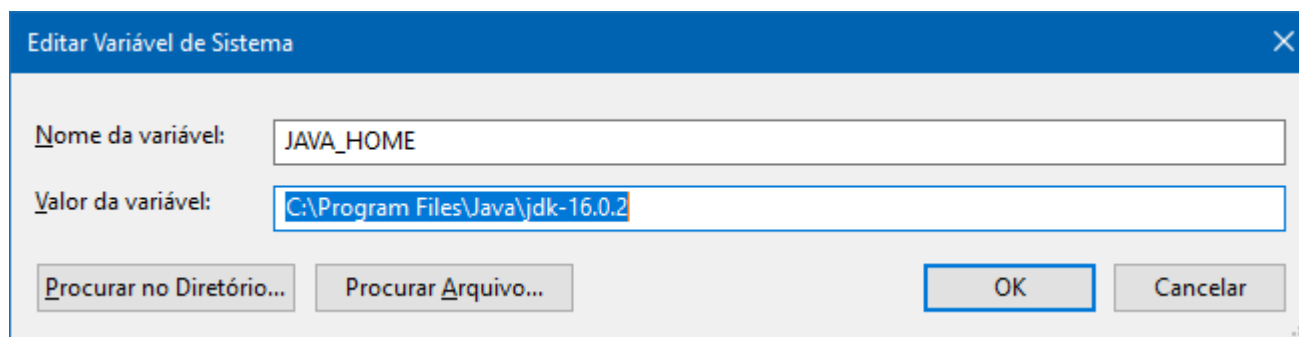
- Preparando o ambiente de desenvolvimento
- Introdução ao Spring Framework
 - Spring Initializr
- Criando um projeto Hello World
 - Criando o *model*
 - Criando o *controller*
 - Entendo *RequestController*, *RequestMapping* e *RequestParam*
- Criando uma calculadora
 - Trabalhando com PathParameters
 - Trabalhando com Exceções

A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the green text box. The leaf is composed of several rounded, overlapping shapes in a light green color, creating a sense of movement and growth.

PREPARANDO O AMBIENTE DE DESENVOLVIMENTO

Baixar e instalar o JDK

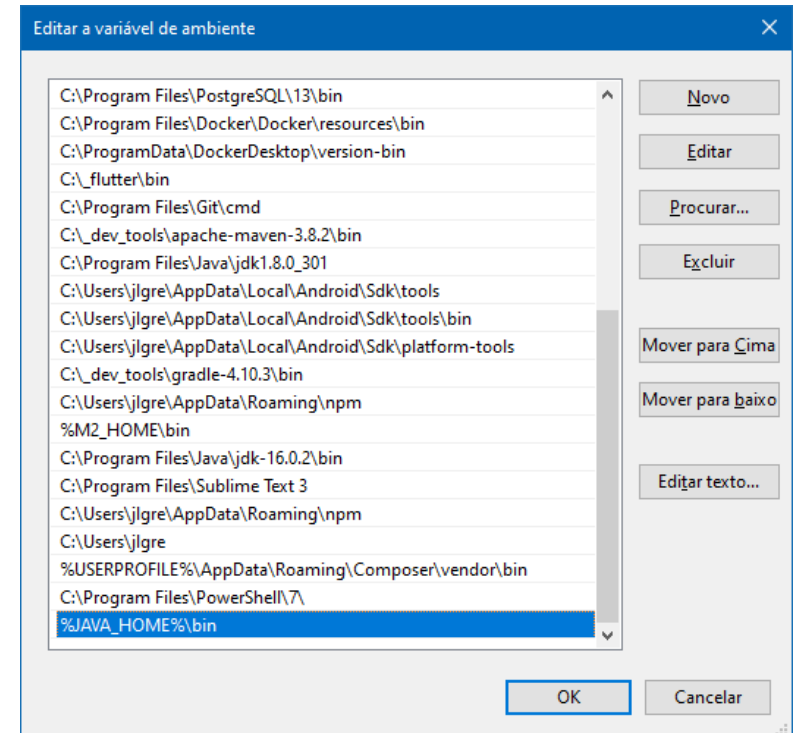
- Baixar e instalar a versão mais recente do Java SE Development Kit (JDK):
<https://www.oracle.com/java/technologies/downloads/>
 - Note que o Java é instalado na pasta *c:\Arquivos de Programas\Java\jdk[**version**]* → onde *[version]* é a versão do JDK instalada;
 - É possível ter várias versões do JDK instaladas em seu computador;
 - Depois de instalado, precisamos criar a variável de ambiente **JAVA_HOME**
 - Para isso, vá em *Configurações Avançadas do Sistema e Variáveis de Ambiente*;
 - Em *Variáveis do Sistema*, clique em Novo e adicione a seguinte entrada:



Note que o valor da variável é o caminho de instalação do JDK.

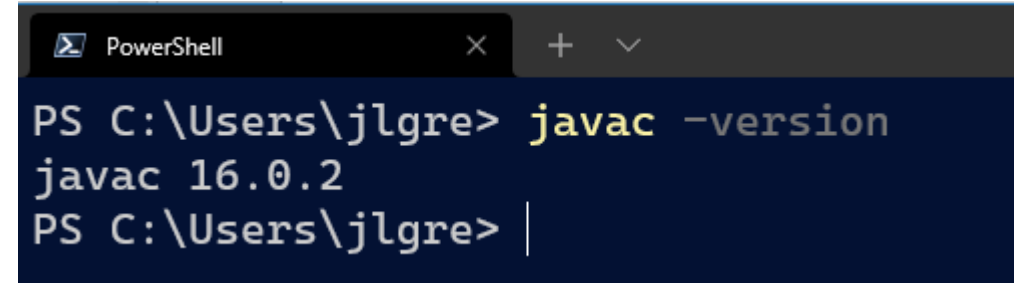
Alterando a variável PATH

- A variável de ambiente ***PATH*** define os caminhos que serão considerados na execução de comandos do *prompt*. Portanto, precisamos adicionar o caminho do executável do Java (java.exe) nessa variável.
- Ainda em *Variáveis de Ambiente* e *Variáveis do Sistema*, abra a variável PATH, clique em Novo e adicione a seguinte entrada:
%JAVA_HOME%\bin



Testando se o JDK foi instalado corretamente

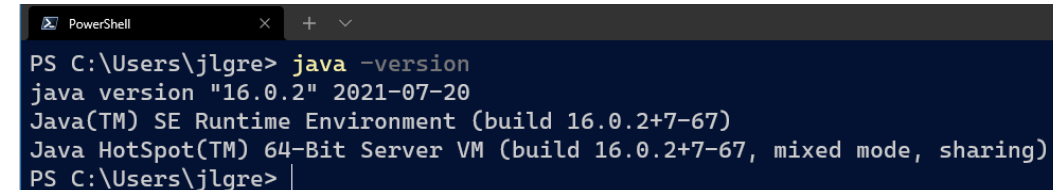
- Abra o *prompt* de comando ou o PowerShell e digite o seguinte comando:
`javac -version`
- Se as informações do compilador Java forem exibidas, então tudo está OK: o JDK e as variáveis de ambiente.



```
PowerShell
PS C:\Users\jlgre> javac -version
javac 16.0.2
PS C:\Users\jlgre> |
```

- Para verificar se o ambiente de execução está OK, digite:

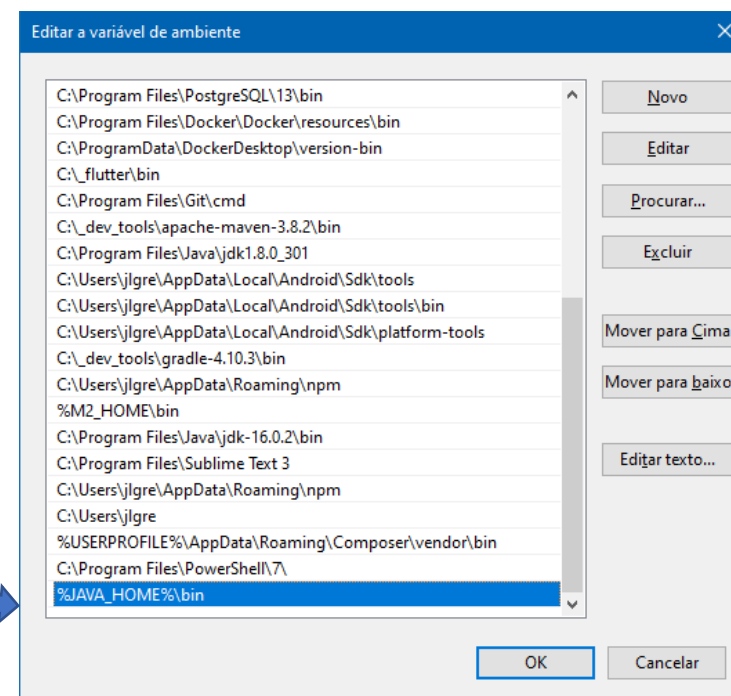
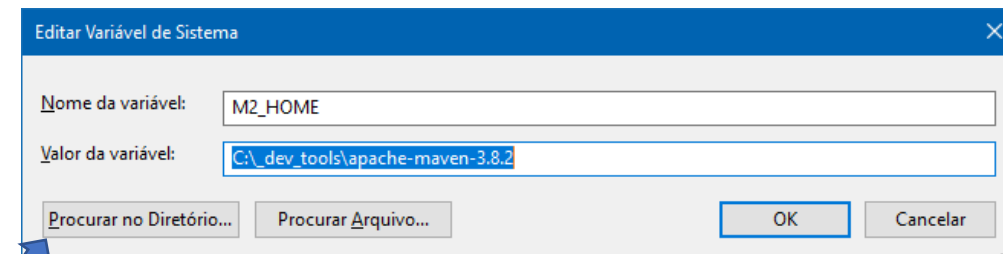
`java --version`



```
PowerShell
PS C:\Users\jlgre> java -version
java version "16.0.2" 2021-07-20
Java(TM) SE Runtime Environment (build 16.0.2+7-67)
Java HotSpot(TM) 64-Bit Server VM (build 16.0.2+7-67, mixed mode, sharing)
PS C:\Users\jlgre> |
```

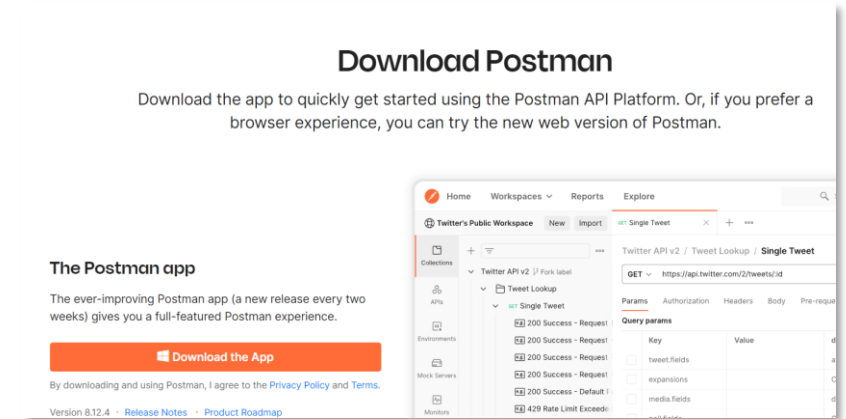
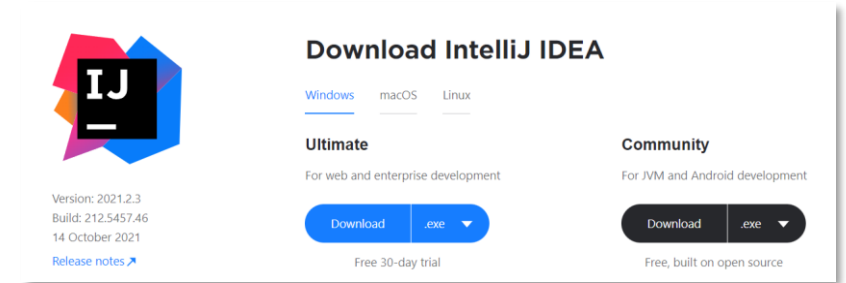
Instalando o Maven

- O **Apache Maven** é um gerenciador de projetos Java;
- Isso significa que ele é usado para construir, gerenciar e publicar aplicações, incluindo o gerenciamento de dependências do seu projeto;
- Baixe a versão mais recente em **formato zip**. Ele não é um instalador, ou seja, descompacte o conteúdo do arquivo baixado em um diretório de sua preferência;
- Depois é necessário criar e configurar **variáveis de ambiente**, igual fizemos na configuração do Java:
 - Em Variáveis de Ambiente, vá em Variáveis do Sistema, clique em novo e adicione a variável **M2_HOME** com o valor sendo o caminho em que o Maven foi descompactado, como mostrado na primeira figura ao lado.
- Depois altere a variável **PATH**, adicionando a seguinte entrada: **%M2_HOME%\bin**, como na segunda figura ao lado:



Instalando as demais ferramentas

- Baixe e instale a versão Community (Gratuita) do **IntelliJ IDEA**, que é a IDE que iremos usar:
<https://www.jetbrains.com/idea/download>
- Instalar o **Postman**, que é um ambiente CLIENT para testar o consumo de serviços:
<https://www.postman.com/downloads/>
 - O Postman possui também um aplicativo Web, mas para usá-lo é necessário criar uma conta;
 - A versão desktop pode ser usada sem a criação de contas.
- Instale e configure o banco de dados de sua preferência;



A large, stylized green leaf graphic is positioned on the left side of the slide, partially overlapping the title box. It has a light green fill and a darker green outline, with a central vein and several smaller veins branching out.

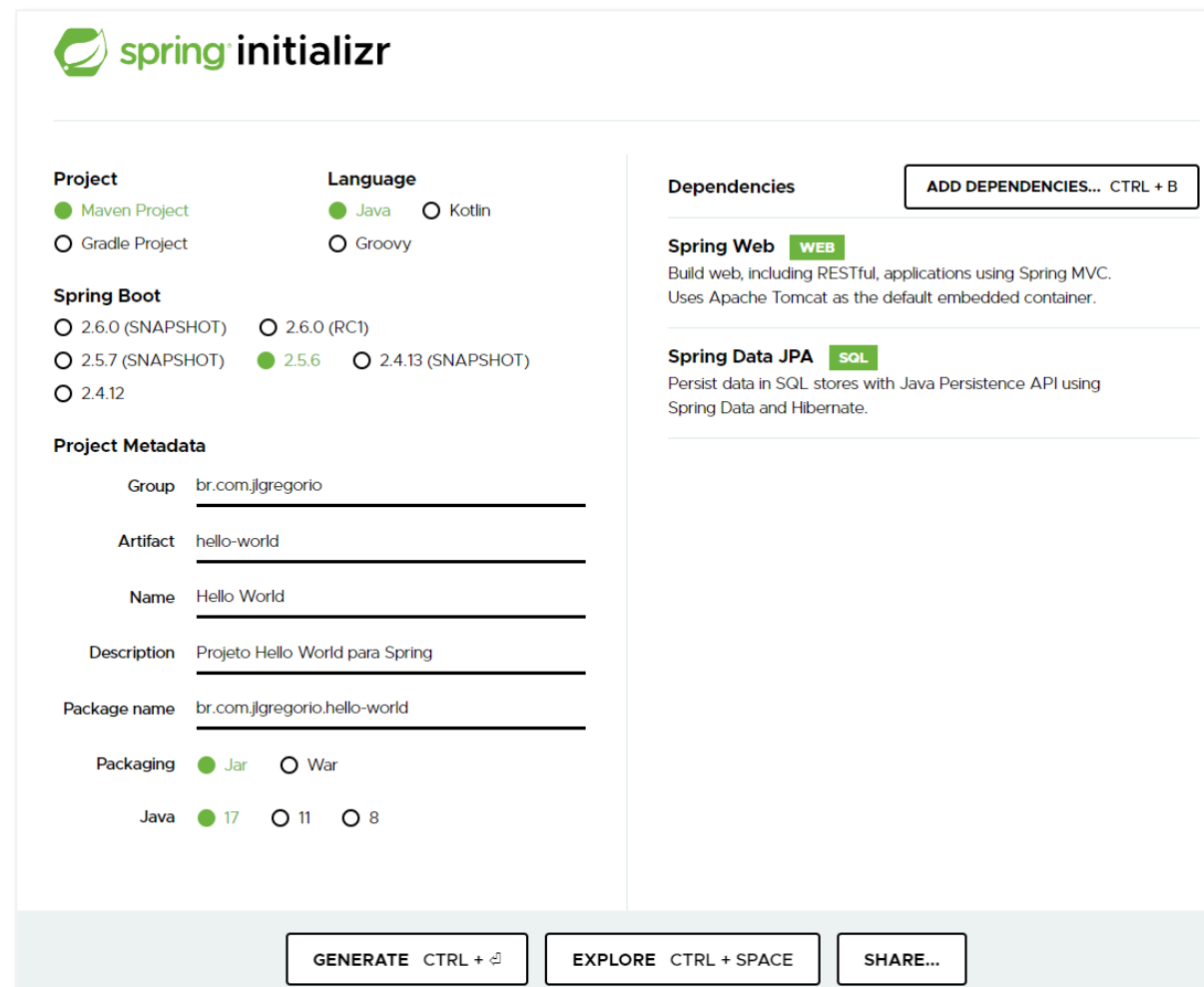
INTRODUÇÃO AO SPRING FRAMEWORK

O que é Spring Framework

- Spring é um framework para o desenvolvimento de aplicações Java;
- Possui diversos módulos para facilitar o desenvolvimento rápido de aplicações Web:
 - **Spring Boot:** auxilia na construção, gerenciamento e publicações de aplicações Java de maneira simples e rápida;
 - **Spring Web MVC:** construção a aplicações Web segundo o padrão de arquitetura MVC.
 - **Spring Data:** provê acesso a diversas fontes de dados;
 - **Spring Cloud:** permite a construção e integração de aplicações Cloud;
 - **Spring Security:** autenticação e controle de acesso;
 - **Spring HATEOAS:** permite a construção de aplicações REST segundo o princípio HATEOAS
- Para mais informações, acesse: <https://spring.io/projects/>

Criando o projeto “HELLO WORLD”

- Para criar um projeto, vamos usar o Spring **Initializr**, que é uma ferramenta em que baixamos o “esqueleto” de um projeto para abrir em nossa IDE.
- Abra o link: <https://start.spring.io/>
- Configure o projeto de acordo com a figura e clique em **Generate** para baixar o projeto.
- Salve o arquivo .zip em uma pasta de sua preferência, depois descompacte-o.



The screenshot shows the Spring Initializr web interface. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.5.6' selected. The 'Project Metadata' section has 'Group' as 'br.com.jlgregorio', 'Artifact' as 'hello-world', 'Name' as 'Hello World', 'Description' as 'Projeto Hello World para Spring', and 'Package name' as 'br.com.jlgregorio.hello-world'. The 'Packaging' section has 'Jar' selected. The 'Java' section has '17' selected. The 'Dependencies' section has 'Spring Web' and 'Spring Data JPA' selected. The 'Generate' button is highlighted.

spring initializr

Project
☒ Maven Project
☐ Gradle Project

Language
☒ Java ☐ Kotlin
☐ Groovy

Spring Boot
☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (RC1)
☐ 2.5.7 (SNAPSHOT) ☒ 2.5.6 ☐ 2.4.13 (SNAPSHOT)
☐ 2.4.12

Project Metadata
Group
Artifact
Name
Description
Package name

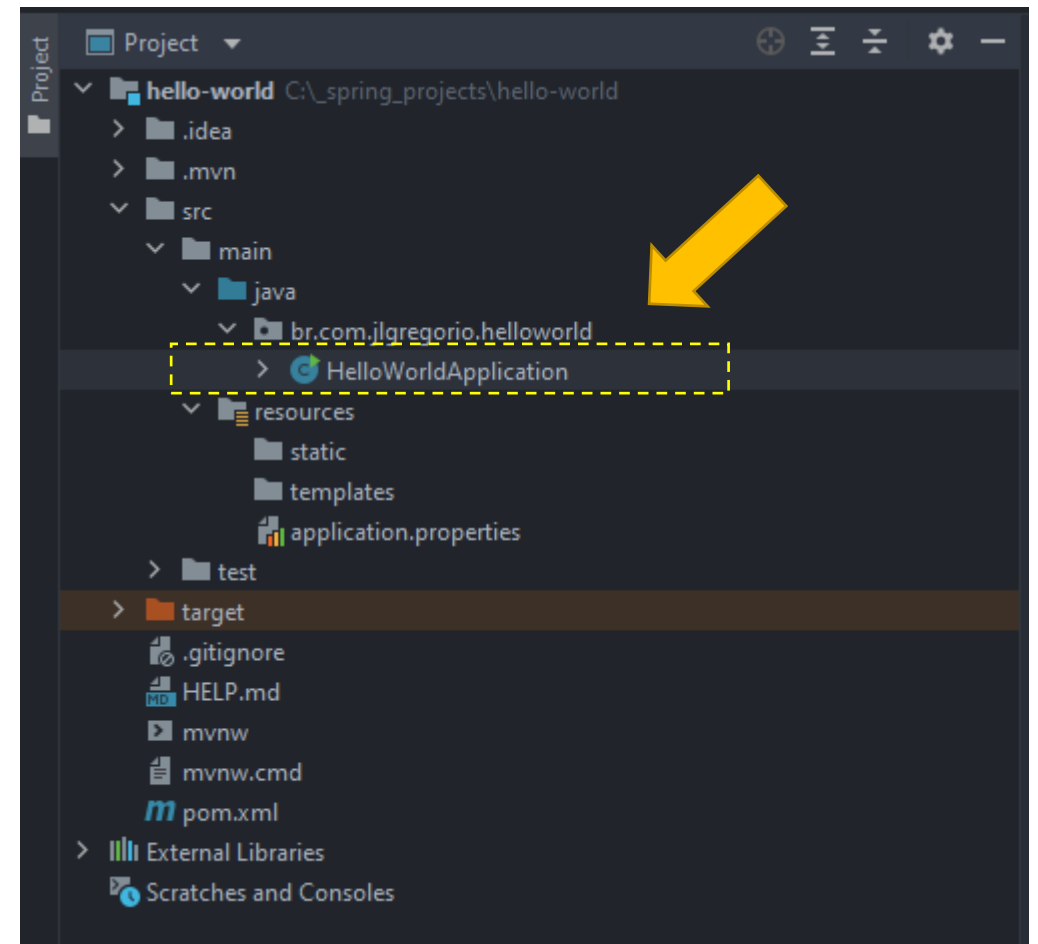
Packaging ☒ Jar ☐ War
Java ☒ 17 ☐ 11 ☐ 8

Dependencies **ADD DEPENDENCIES... CTRL + B**
Spring Web **WEB**
Build web, including RESTful, applications using Spring MVC.
Uses Apache Tomcat as the default embedded container.
Spring Data JPA **SQL**
Persist data in SQL stores with Java Persistence API using
Spring Data and Hibernate.

GENERATE CTRL + G **EXPLORE CTRL + SPACE** **SHARE...**

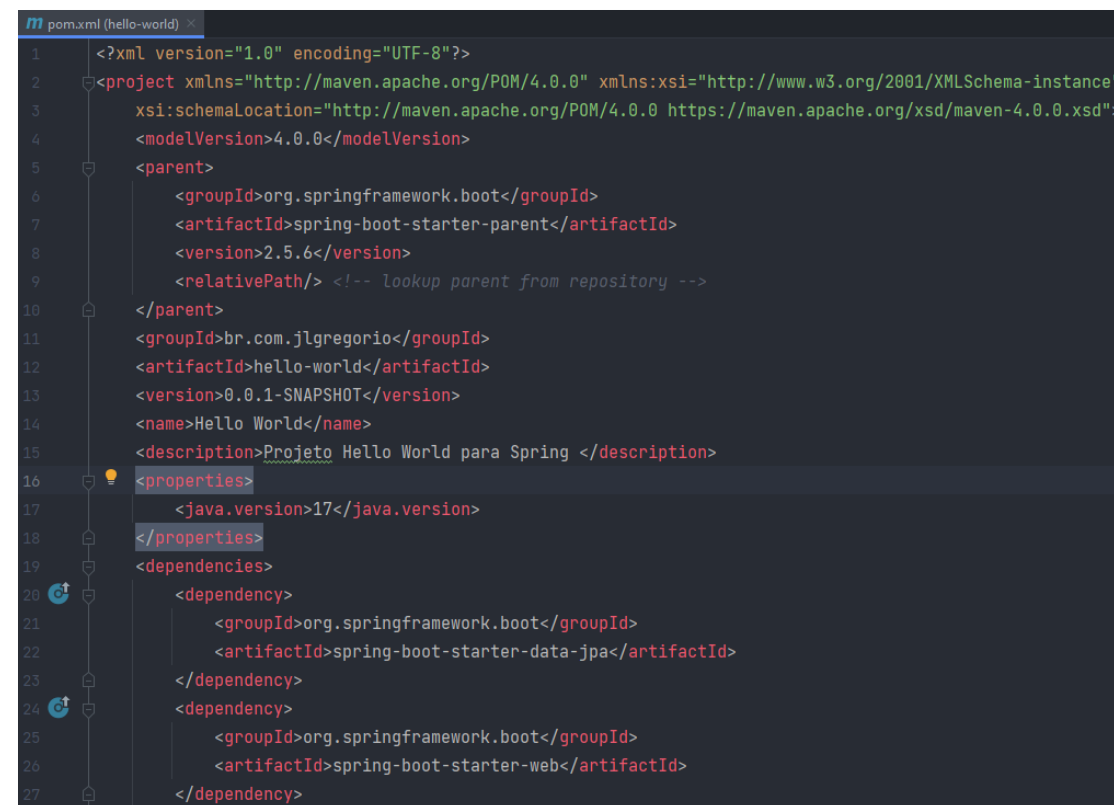
Abrindo o projeto no IntelliJ IDEA

- Abra o IntelliJ IDEA;
- Clique em **Open**, navegue até a **pasta** em que o arquivo **zip** foi descompactado e abra o projeto;
- Na janela projects é possível observar os arquivos do projeto;
- Note que existe um arquivo **HelloWorldApplication** que é o ponto de entrada da aplicação;



O arquivo *pom.xml*

- O arquivo ***pom.xml***, localizado na raiz do projeto, contém as dependências da aplicação;
- É possível adicionar as dependências simplesmente declarando uma nova entrada ***<dependency>*** em ***<dependencies>***, considerando os módulos do Spring.

A screenshot of a code editor showing a Maven pom.xml file. The file is titled 'pom.xml (hello-world)'. The XML content is as follows:

```
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>2.5.6</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>br.com.jlgregorio</groupId>
12  <artifactId>hello-world</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>Hello World</name>
15  <description>Projeto Hello World para Spring</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-data-jpa</artifactId>
23    </dependency>
24    <dependency>
25      <groupId>org.springframework.boot</groupId>
26      <artifactId>spring-boot-starter-web</artifactId>
27    </dependency>
```

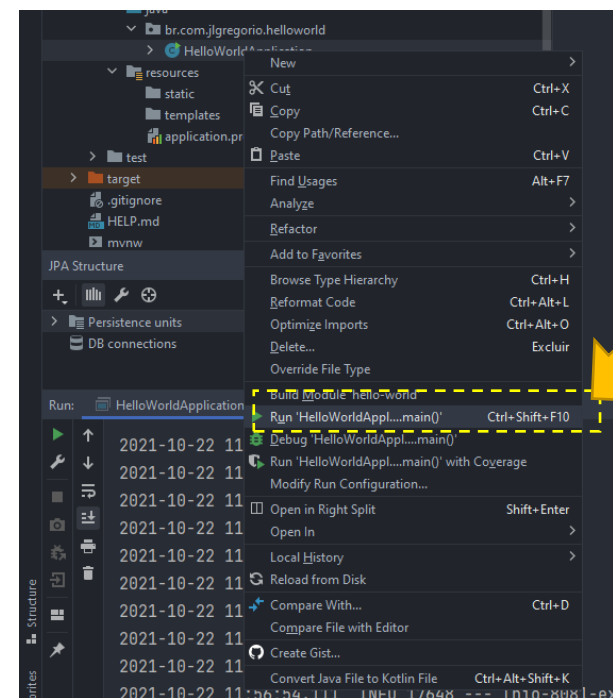
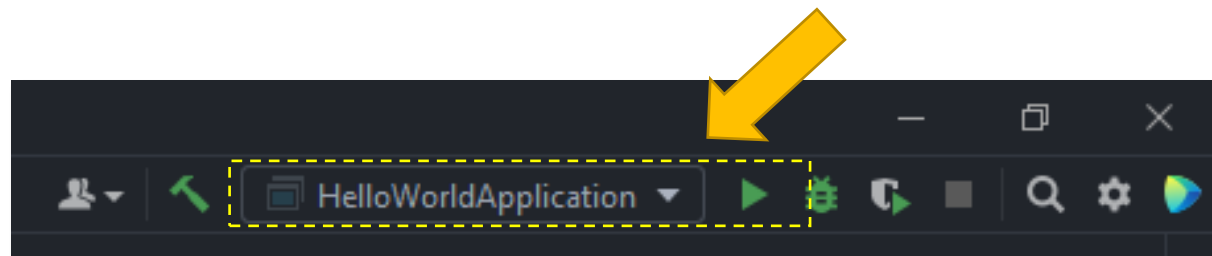
O arquivo *application.properties*

- O arquivo ***src/main/resources/application.properties*** é usado para definir as propriedades da aplicação;
- Nele é possível habilitar ou desabilitar módulos, configurar acesso a banco de dados, entre outras configurações;
- No exemplo a seguir fizemos duas coisas:
 - desativamos o módulo JPA (***Java Persistence API***) → como incluímos a dependência Spring Data JPA, por padrão ela será carregada na inicialização da aplicação. Isso irá gerar um erro, pois não temos banco de dados ainda. Assim, é necessário desativá-la;
 - Mudamos a porta padrão (8080) para 8081

```
spring.autoconfigure.exclude=org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration  
server.port=8081
```

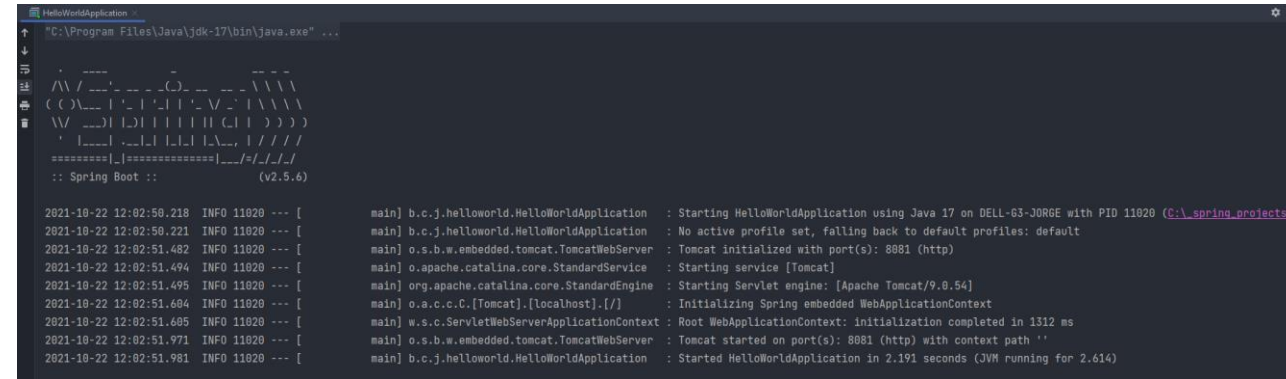
Executando a aplicação

- No painel superior direito, cliquem **Run...**
- Ou clique com o botão direito do mouse no arquivo HelloWorldApplication e **Run...**



Verificando no navegador

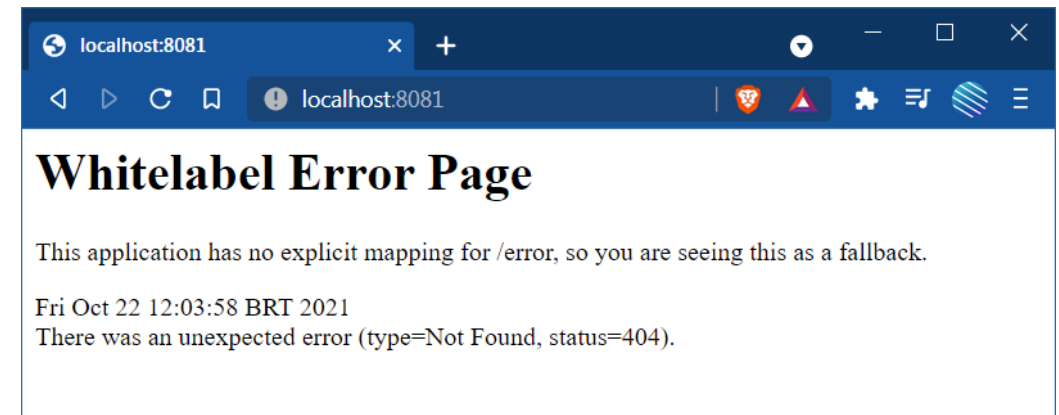
- Ao executar a aplicação, será exibida a mensagem de saída no IntelliJ mostrando que ela estar respondendo na porta 8081.
- Abra o navegador e digite **localhost:8081**. Será exibida uma página de erro padrão, dizendo que não há página de erro padrão mapeada.
- Isso é normal, pois não definimos nenhuma página de visualização e nenhum retorno.
- Enfim, a aplicação está respondendo!



```
"C:\Program Files\Java\jdk-17\bin\java.exe" ...

  ____ _
 / ___ \| | | |
/ /   \| |_| |
\ \   /| | | |
 \___/\_|_|_|_|
:: Spring Boot :: (v2.5.6)

2021-10-22 12:02:50.218 INFO 11020 --- [main] b.c.j.helloworld.HelloWorldApplication : Starting HelloWorldApplication using Java 17 on DELL-63-JORGE with PID 11020 (C:\spring_projects\helloworld\helloworld\bin\java.exe)
2021-10-22 12:02:50.221 INFO 11020 --- [main] b.c.j.helloworld.HelloWorldApplication : No active profile set, falling back to default profiles: default
2021-10-22 12:02:51.482 INFO 11020 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8081 (http)
2021-10-22 12:02:51.494 INFO 11020 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-10-22 12:02:51.495 INFO 11020 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.54]
2021-10-22 12:02:51.604 INFO 11020 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-10-22 12:02:51.605 INFO 11020 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1312 ms
2021-10-22 12:02:51.971 INFO 11020 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8081 (http) with context path ''
2021-10-22 12:02:51.981 INFO 11020 --- [main] b.c.j.helloworld.HelloWorldApplication : Started HelloWorldApplication in 2.191 seconds (JVM running for 2.614)
```



A decorative graphic on the left side of the slide, consisting of several overlapping, stylized green leaves or petals. The leaves are a light green color and have a smooth, curved shape. They are arranged in a way that they appear to be part of a larger plant or flower, with some leaves pointing upwards and others downwards.

CRIANDO UM “HELLO WORLD”

Criando uma *model*

- No pacote principal da aplicação, crie um novo ***pacote*** chamado ***model***, depois uma nova ***classe*** chamada *Welcome*;
- Usando o atalho ***Alt+Insert*** ou clicando com o botão direito do mouse e clicando no comando Generate, crie os construtores e os métodos ***setters*** e ***getters***.

```
Welcome.java x
1 package br.com.jlgregorio.helloworld;
2
3 public class Welcome {
4
5     private long id;
6     private String name;
7     private String message;
8
9 }
```

```
1 package br.com.jlgregorio.helloworld;
2
3 public class Welcome {
4
5     private long id;
6     private String name;
7     private String message;
8
9     public Welcome() {
10
11     }
12
13     public Welcome(long id, String name, String message) {
14         this.id = id;
15         this.name = name;
16         this.message = message;
17     }
18
19     public long getId() {
20         return id;
21     }
22
23     public void setId(long id) {
24         this.id = id;
25     }
26 }
```

Criando um *controller*

- Os controllers são os responsáveis por interceptar as requisições e trata-las de maneira adequada;
- No Spring Framework, é possível criar controladores usando o conceito de anotações.
- No exemplo a seguir criamos um **Rest Controller** usando a anotação *@RestController*, e um end-point para receber requisições com o a anotação *@RequestMapping*
- O retorno é uma classe model serializada, em formato JSON, que é o padrão do retorno de um **Rest Controller**.

```
package br.com.jlgregorio.helloworld.controller;

import br.com.jlgregorio.helloworld.model.Welcome;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import java.util.concurrent.atomic.AtomicLong;

//Defines this class as a Rest Controller
@RestController
public class WelcomeController {

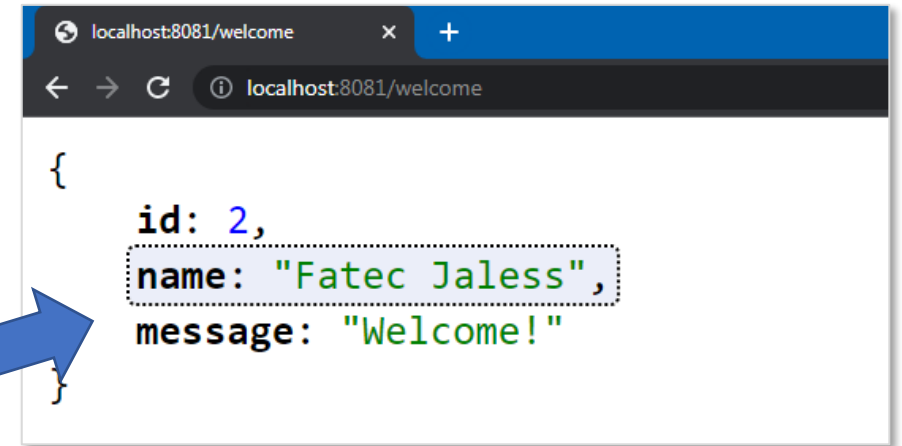
    private final static AtomicLong counter = new AtomicLong();

    //defines this method as a service end-point
    @RequestMapping("/welcome")
    //the RequestParam anotation configures the input parameters from request
    public Welcome welcome( @RequestParam(name = "name", defaultValue = "Fatec Jaless")
String name,
    @RequestParam(name = "msg", defaultValue = "Welcome!") String msg){

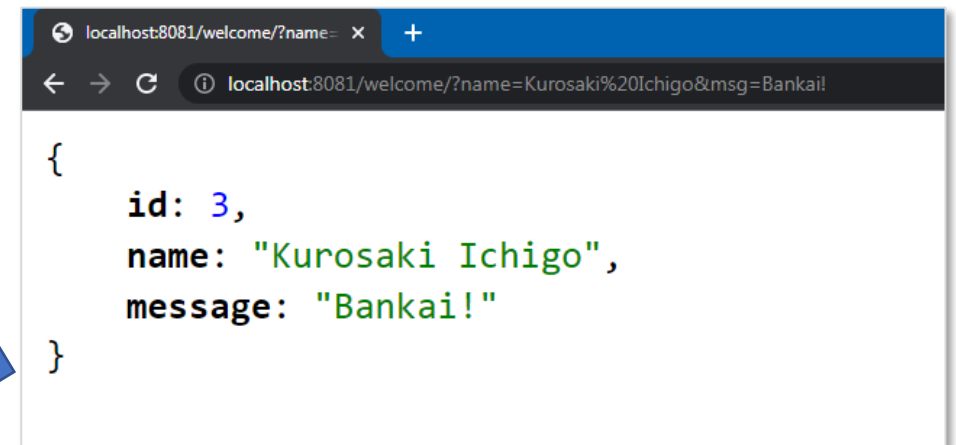
        //return a new Welcome object, already serialized by default
        return new Welcome(counter.incrementAndGet(), name, msg);
    }
}
```

Testando

- Execute a aplicação novamente e faça a requisição via browser digitando a URL: ***localhost:8081/welcome***
- Se não for informado nenhum parâmetro (name e msg), os valores default serão usados para retornar o objeto *Welcome*, como mostrado na figura.
- Agora podemos informar os parâmetros via URL:
- ***localhost:8081/welcome?name=Kurosaki Ichigo&msg=Bankai!***



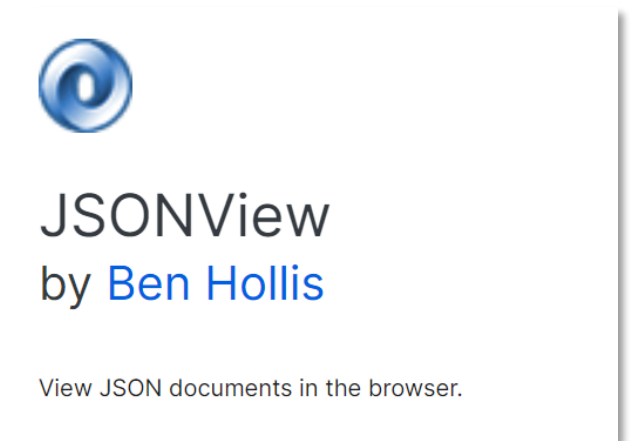
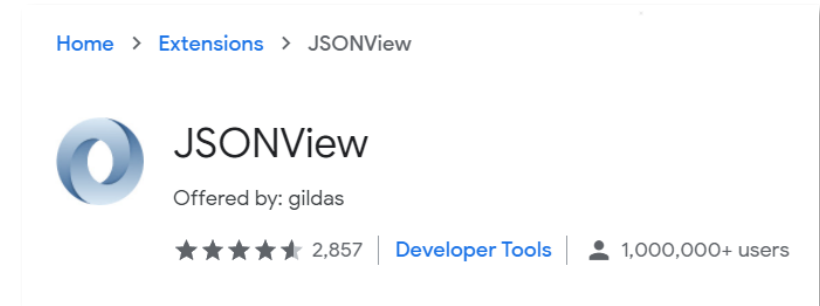
```
{
  id: 2,
  name: "Fatec Jaless",
  message: "Welcome!"
}
```



```
{
  id: 3,
  name: "Kurosaki Ichigo",
  message: "Bankai!"
}
```

Dica

- Instale a extensão JSON View no navegador Chrome ou Firefox para que os arquivos JSON sejam exibidos de maneira mais organizada, como nos exemplo mostrados anteriormente.
- Chrome:
<https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=en>
- Firefox: <https://addons.mozilla.org/en-US/firefox/addon/jsonview/>

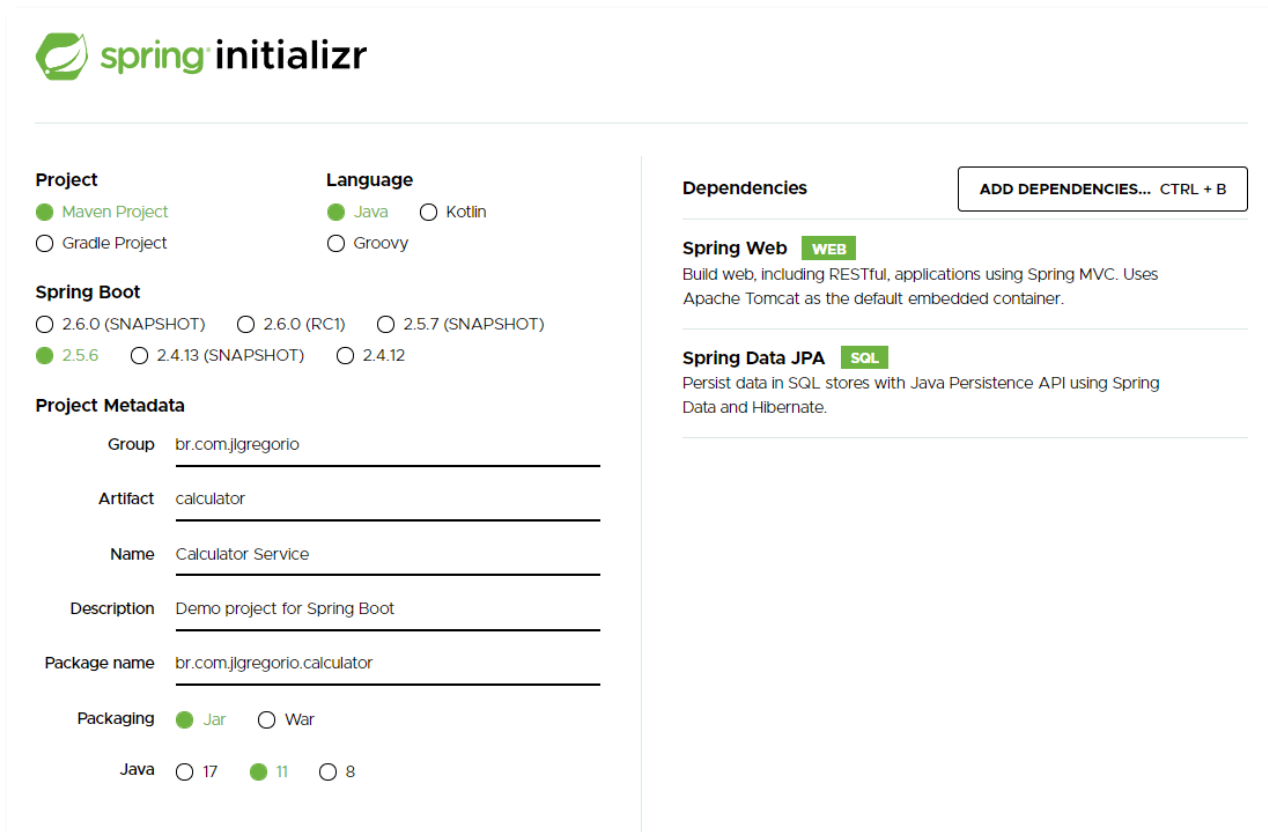


A decorative graphic of a green leaf, partially visible on the left side of the slide, with a light green outline and a darker green fill.

CRIANDO UMA CALCULADORA

Criando o projeto via Spring Initializr

- Abra o Spring Initializr e crie um novo projeto de acordo com as configurações ao lado da figura ao lado:
 - Maven Project
 - Java
 - Spring Boot 2.5.6
 - Packaging: Jar
 - Java: 11
- Vamos implementar uma calculadora simples, ao mesmo tempo em que aprendemos novos conceitos sobre Spring, REST e padrões de projeto. Bora!



The screenshot shows the Spring Initializr web interface. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.5.6' selected. The 'Project Metadata' section has 'Group' as 'br.com.jlgregorio', 'Artifact' as 'calculator', 'Name' as 'Calculator Service', 'Description' as 'Demo project for Spring Boot', and 'Package name' as 'br.com.jlgregorio.calculator'. The 'Packaging' section has 'Jar' selected. The 'Java' section has '11' selected. The 'Dependencies' section has 'Spring Web' and 'Spring Data JPA' selected.

spring initializr

Project
☒ Maven Project
☐ Gradle Project

Language
☒ Java ☐ Kotlin
☐ Groovy

Spring Boot
☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (RC1) ☐ 2.5.7 (SNAPSHOT)
☒ 2.5.6 ☐ 2.4.13 (SNAPSHOT) ☐ 2.4.12

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 17 ☒ 11 ☐ 8

Dependencies ADD DEPENDENCIES... CTRL + B

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

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Criando o model

- Depois de descompactar o arquivo .zip baixado, abra o projeto no IntelliJ IDEA;
- No pacote *br.com.jlgregorio.calculadora*, crie um novo pacote chamado ***model***;
- Agora crie uma classe chamada Calculator, como no exemplo ao lado:

```
package br.com.jlgregorio.calculadora.model;

public class Calculator {

    public static float sum(float n1, float n2){
        return n1 + n2;
    }

}
```


Criando um controller

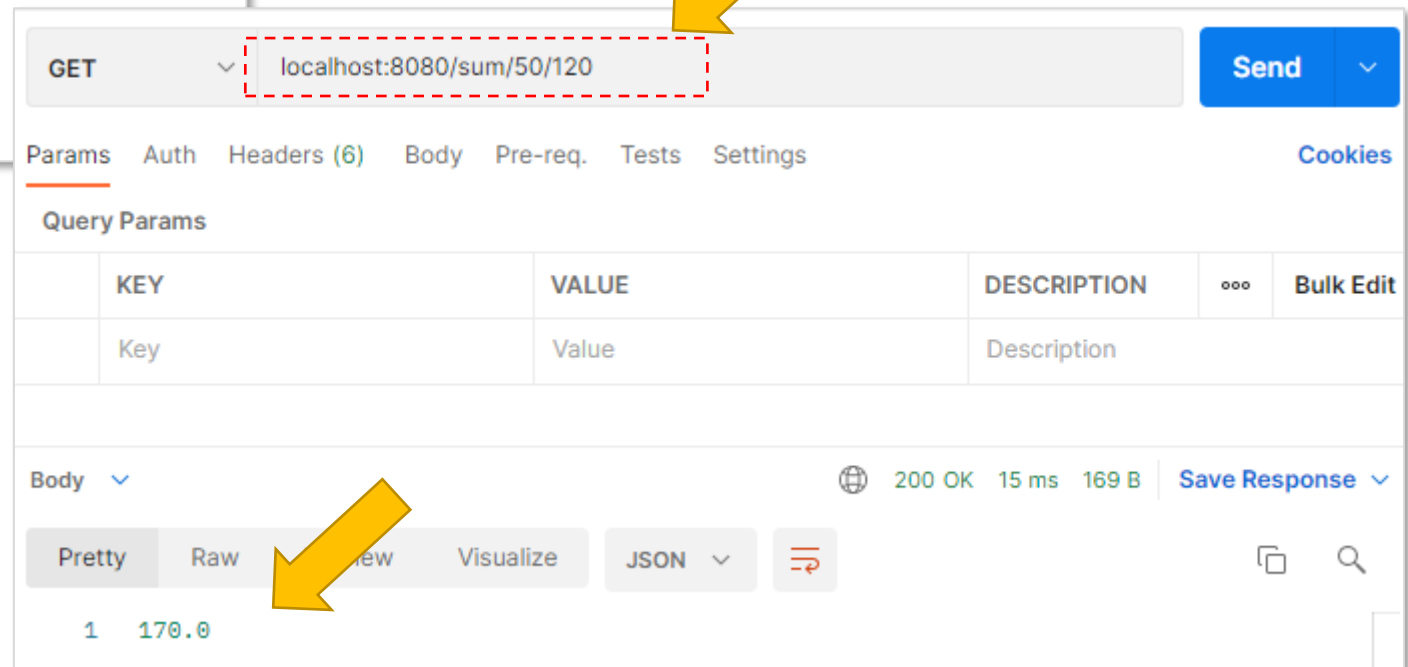
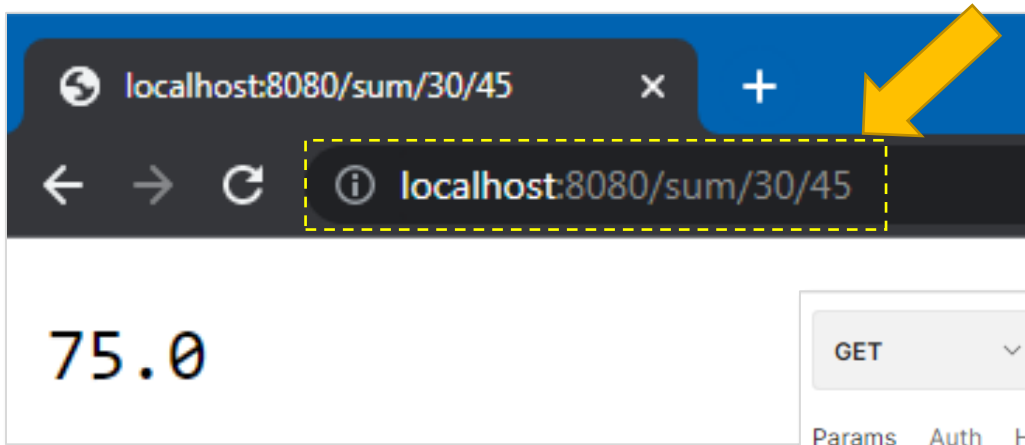
- No pacote *br.com.jlgregorio.calculadora*, crie um novo pacote chamado ***controller***;
- Agora crie uma nova classe Java como nome **CalculatorController**. Observe o código comentado ao lado.

```
package br.com.jlgregorio.calculadora.controller;
import br.com.jlgregorio.calculadora.model.Calculator;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

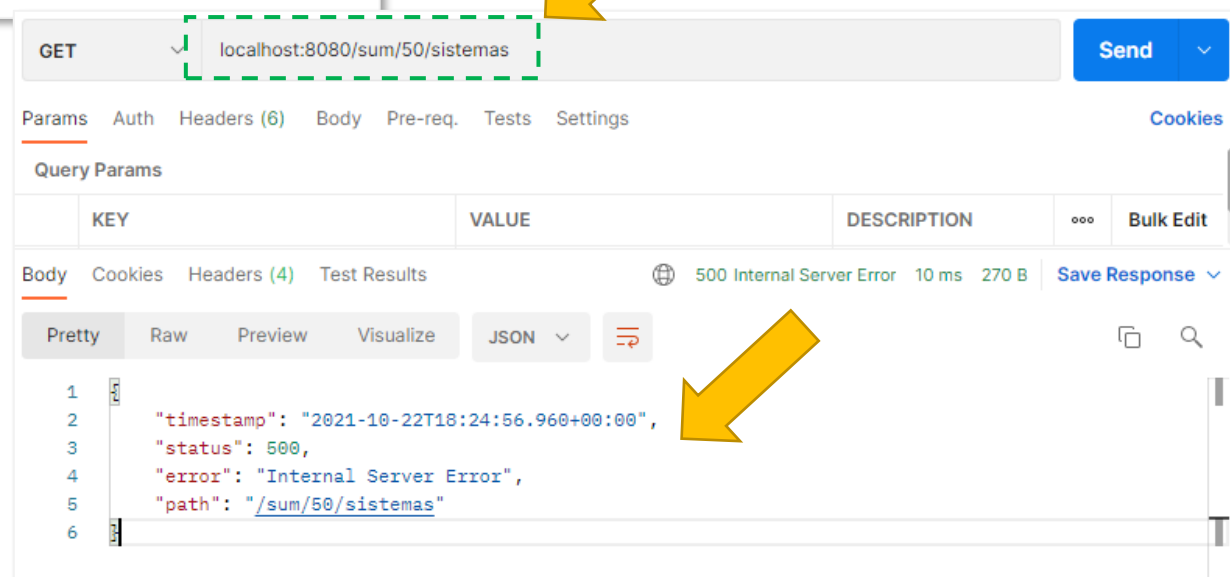
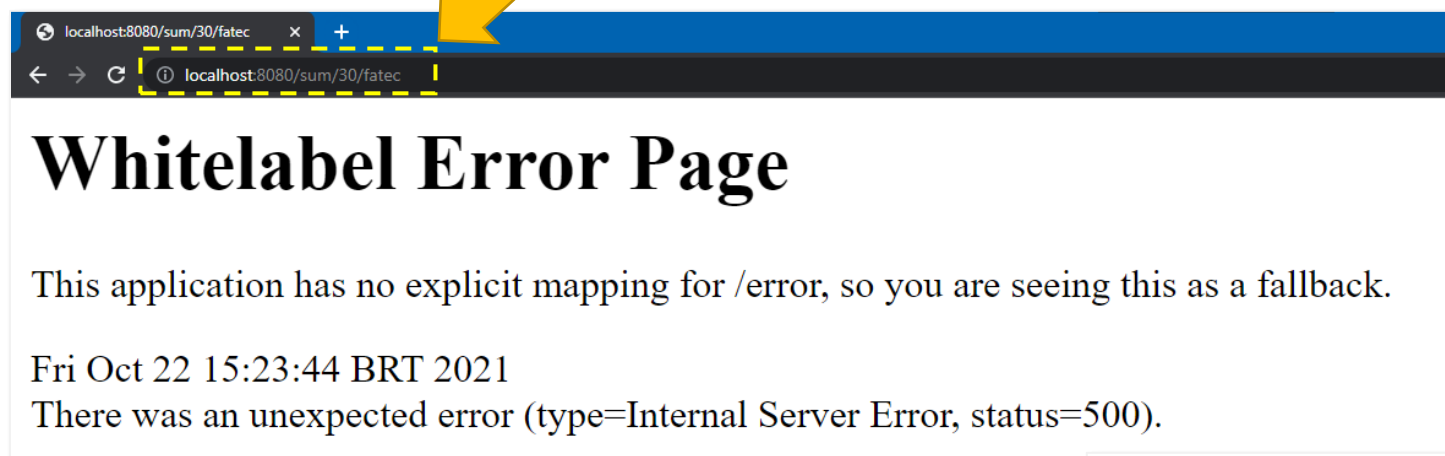
//..define this class as a Rest Controller
@RestController
public class CalculatorController {

    //defines a request mapping, using friendly URL with Path Variables and GET method
    @RequestMapping(value = "/sum/{n1}/{n2}", method = RequestMethod.GET)
    public float sum(@PathVariable("n1") String n1, @PathVariable("n2") String n2 ){
        //return the result
        return Calculator.sum(Float.parseFloat(n1), Float.parseFloat(n2));
    }
}
```

Testando no browser e no Postman



Requisições com erros



Tratando exceções

- Crie um pacote chamado ***exceptions*** no pacote principal;
- Crie uma classe chamada `ExceptionResponse`, como o código ao lado;
- Gere o construtor com parâmetros e os métodos ***setters*** e ***getters***:

```
package br.com.jlgregorio.calculadora.exceptions;

import java.io.Serializable;
import java.util.Date;

public class ExceptionResponse implements Serializable {

    private static final long serialVersionUID = 1L;

    private Date timeStamp;
    private String message;
    private String details;
}
```

Criando uma exceção personalizada

- No pacote exceptions, crie a classe ***InvalidNumberOperationException***, com o código ao lado:
- Essa exceção será usada para tratar erros quando o usuário informar parâmetros não numéricos na requisição;

```
package br.com.jlgregorio.calculadora.exceptions;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

//defines the HTTP status code
@ResponseStatus(HttpStatus.BAD_REQUEST)
public class InvalidNumberOperationException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public InvalidNumberOperationException(String exception) {
        super(exception);
    }
}
```

Criando um manipulador de exceções

- No pacote ***exceptions***, adicione uma nova classe chamada ***CustomizedExceptionHandler***
- O código dela é mostrado ao lado:

```
package br.com.jlgregorio.calculadora.exceptions;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import java.util.Date;
import java.util.concurrent.ExecutionException;

@ControllerAdvice //is a controller advice, to manage exceptions
@RestController //is a Rest Controller
public class CustomizedExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(Exception.class) //catch generic exceptions
    public final ResponseEntity<ExceptionResponse> handleAllExceptions(Exception e, WebRequest request){
        ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(), e.getMessage(), request.getDescription(false));
        return new ResponseEntity<>(exceptionResponse, HttpStatus.INTERNAL_SERVER_ERROR);
    }

    @ExceptionHandler(InvalidNumberOperationException.class) //catch customized exception
    public final ResponseEntity<ExceptionResponse> handleBadRequestExceptions(Exception e, WebRequest request){
        ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(), e.getMessage(), request.getDescription(false));
        return new ResponseEntity<>(exceptionResponse, HttpStatus.BAD_REQUEST);
    }
}
```

Alterando o controlador para capturar exceções

```
package br.com.jlgregorio.calculadora.controller;
import br.com.jlgregorio.calculadora.exceptions.InvalidNumberOperationException;
import br.com.jlgregorio.calculadora.model.Calculator;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
//define this class as a Rest Controller
@RestController
public class CalculatorController {

    //defines a request mapping, using friendly URL with Path Variables and GET method
    @RequestMapping(value = "/sum/{n1}/{n2}", method = RequestMethod.GET)
    public float sum(@PathVariable("n1") String n1, @PathVariable("n2") String n2 ) {
        //return the result
        if(!isNumeric(n1) || !isNumeric(n2)){
            throw new InvalidNumberOperationException("Please, set valid numeric values!");
        }
        return Calculator.sum(Float.parseFloat(n1), Float.parseFloat(n2));
    }

    public boolean isNumeric(String value){
        try{
            Float.parseFloat(value);
            return true;
        } catch (Exception e){
            return false;
        }
    }
}
```

Desafios

- Implemente as demais operações matemáticas: subtração, multiplicação e divisão;
- Como lançar uma exceção se vier números negativos na requisição?
- Crie um novo projeto Spring contendo um **RestController** com apenas uma rota: **/test** – essa rota deverá receber dois **path params**: **text** e **number**. A saída deverá ser a repetição de **text** a quantidade de vezes definidas em **number** durante a requisição.

Sobre mim

JORGE LUÍS GREGÓRIO

- Professor da Faculdade de Tecnologia “Prof. José Camargo” – Fatec Jales, e da Escola Técnica Estadual Dr. José Luiz Viana Coutinho – Etec Jales;
- Articulista do Jornal de Jales – Coluna “Fatecnologia”;
- Apresentador do Tech Trends, podcast oficial da Fatec Jales;
- Bacharel em Sistemas de Informação; Especialista em Desenvolvimento de Software para Web e Mestre em Ciência da Computação.
- Trabalha com tecnologia desde 1998, tendo atuado como analista de suporte; administrador de redes de computadores; desenvolvedor de software, *webdesigner* e professor.
- Site oficial: www.jlgregorio.com.br
- Perfil do LinkedIn: www.linkedin.com/in/jlgregorio81
- Currículo Lattes: <http://lattes.cnpq.br/3776799279256689>

