



# Implementação da Arquitetura MVC

Prof. Sérgio Assunção Monteiro

## Descrição

Apresentação dos aspectos essenciais para implementação da arquitetura MVC.

## Propósito

O conhecimento teórico e prático da arquitetura MVC é essencial para profissionais de tecnologia que pretendem trabalhar no desenvolvimento de serviços disponibilizados na web e que possam ser consumidos por outras aplicações tanto web, como de dispositivos móveis.

## Preparação

Para rodar os exemplos, você vai precisar ter os seguintes programas instalados:

- JSDK versão 10 ou superior;
- Maven 3.8.4;
- Eclipse;
- Postman;

- Talend API Tester (instalar o plugin no navegador).

De modo geral, o mundo da computação é colaborativo. É comum, quando encontramos erros nos programas e nas bibliotecas utilizadas, copiarmos o log de erro e procurarmos em qualquer motor de busca. Provavelmente, alguma resposta será retornada em algum fórum de discussão, como StackOverflow, Github, Reddit, entre outros. Não só isso é uma prática comum na comunidade de desenvolvimento de software e computação, como também nos possibilita aprender cada vez mais.

## Objetivos

---

### Módulo 1

## Compreendendo e utilizando o gerenciador de dependências Maven

Reconhecer o gerenciador de dependências Maven.

---

### Módulo 2

## Criação de um projeto Spring Boot

Analisar a criação de um projeto Spring Boot.

---

### Módulo 3

## Criação dos pacotes segundo o modelo MVC

Identificar os aspectos para a criação dos pacotes segundo o modelo MVC.

---

# Introdução

Vivemos em uma época com demandas em todas as áreas: econômica, social, entretenimento, notícias, política, educação, ciências, entre muitas outras. Em especial, no ambiente web, essas demandas são atendidas por aplicações que seguem um modelo de computação distribuída no qual as diversas partes envolvidas se comunicam por meio de protocolos, mas são independentes umas das outras.

Nesse cenário, a própria construção das aplicações exigiu compartimentar as unidades dos sistemas de modo que pudessem ter ciclos evolutivos diferentes que não prejudicassem o objetivo principal deles. A arquitetura MVC trata exatamente da separação dos módulos de uma aplicação como modelo, visão e controle. Além disso, o uso de ferramentas como o Maven e de frameworks como o Spring Boot auxilia e padroniza o processo de desenvolvimento. Ao longo deste conteúdo, vamos abordar esses assuntos e implementaremos algumas aplicações práticas que nos ajudarão a entender como usar esses recursos no dia a dia.



# 1 - Compreendendo e utilizando o gerenciador de dependências Maven

Ao final deste módulo, você será capaz de reconhecer o gerenciador de dependências Maven.

## Introdução ao gerenciador de dependências Maven

### Aspectos básicos da arquitetura MVC

O padrão de arquitetura **MVC** — abreviatura de *Model-View-Controller* — faz a separação de uma aplicação em três componentes principais (SPÄTH, 2021):

#### Modelo



É a parte responsável pelo domínio de dados, ou seja, pelas operações de recuperação e armazenamento de dados de um banco de dados. De forma mais objetiva, é nessa camada que fazemos o mapeamento das tabelas do banco de dados e de suas respectivas colunas, com classes que serão gerenciadas pela aplicação.

#### Visão



É o componente responsável pela interface do usuário com a aplicação. Exemplos de aplicações da camada de visão são:

- Telas para operações de cadastro, consulta e edição de tabelas.
- Exibição de relatórios.

#### Controlador



É o componente que trata da interação do usuário com o modelo. Isso significa que o controlador é quem faz tratamentos de dados, como conversões, consultas e quaisquer procedimentos associados à manipulação lógica dos dados, de modo que a camada de visão receba esses resultados em um formato que possa ser exibido sem a necessidade de outras operações.

O padrão MVC nos ajuda a segmentar as diferentes partes de uma aplicação em lógica de acesso a dados, lógica de negócios e lógica da interface do usuário, e nos dá ao mesmo tempo um mecanismo para realizar um acoplamento fraco entre esses elementos. Essas características são muito úteis para desenvolvimento de grandes projetos em equipe. Mas o MVC é apenas uma arquitetura. Para utilizá-la na prática, precisamos de ferramentas que nos auxiliem nesse processo. Aqui, vamos estudar o Maven.

## O gerenciador de dependências Maven

O Apache Maven é uma ferramenta de automação para construção e gerenciamento de projetos baseados em Java. Entre as aplicações do Maven, podemos destacar:

- Disponibilidade de muitas dependências que podem ser adicionadas para o projeto;
- Padronização da configuração do projeto;
- Gerenciamento de dependências com atualização automática;
- Compatível com versões anteriores;
- Disponibilização de relatório de erros;
- Garantia do uso consistente em todos os projetos;
- Facilidade de integração do projeto com o sistema de controle de versões.

Agora que tivemos uma visão geral do Maven, vamos entender seus aspectos fundamentais e como fazer a sua instalação e configuração.

## Conceitos básicos

O Maven é formado pelos seguintes componentes:

## Arquivos POM

São arquivos XML que contêm informações sobre o projeto e informações de configuração, como dependências, por exemplo, que são usadas pelo Maven para construir o projeto.

## Dependências e repositórios

As dependências são bibliotecas externas necessárias para o projeto; já os repositórios são diretórios de arquivos JAR empacotados.

## Ciclos de vida de construção, fases e metas

Um ciclo de vida de construção é uma sequência de fases, sendo que uma fase consiste em uma sequência de metas. Cada fase é responsável por uma tarefa específica. A seguir estão algumas das fases mais importantes no ciclo de vida de compilação padrão:

### **validate**

Verifica se todas as informações necessárias para a construção estão disponíveis.

### **compile**

Compila o código-fonte.

### **test-compile**

Compila o código-fonte do teste.

**test**

Executa testes unitários.

**package**

Empacota o código-fonte compilado nos formatos JAR ou WAR.

**integration-test**

Executa testes de integração separadamente dos testes unitários.

**install**

Instala o pacote em um repositório local.

**deploy**

Copia o pacote para o repositório remoto.

As fases são compostas por uma sequência de metas. As principais metas são:

**compiler**

Está vinculado à fase **compile** responsável pela compilação.

### compiler: testCompile

Está vinculado à fase **test-compile**.

### install: install

Está vinculado à fase **install**.

### jar:jar e war:war

Estão vinculados à fase **package**.

## Perfis

No arquivo POM (que conheceremos logo adiante), um perfil é definido pelas tags `profile`. O objetivo é construir perfis por meio de um conjunto de valores de configuração. Por exemplo, pode construir um perfil para desenvolvimento e teste.

## Plugins

São usados para atingir perfis com objetivos específicos. Por exemplo, podemos utilizar o plugin do [Apach Tomcat](#) para executar um contêiner Apache Tomcat para desenvolvimento de aplicações web.

Ainda sobre os conceitos básicos do Maven, temos os artefatos e arquétipos. Veja a diferença entre eles:



## Apache Tomcat

O servidor Apache Tomcat é um contêiner web de código-fonte aberto baseado em Java que foi criado para executar aplicações web que utilizam tecnologias Servlets e JSPs.

### Artefato

É um recurso gerado ou usado por um projeto, como, por exemplo, os arquivos JAR e WAR.



### Arquétipo

É um padrão do qual todas as outras coisas do mesmo tipo são feitas.

O nosso próximo passo é verificar os pré-requisitos para instalar o Maven.

## Criação de um projeto Maven

### Pré-requisitos de instalação

Antes de instalar o Maven, precisamos ter o JSDK instalado na máquina. Além disso, a variável de sistema JAVA\_HOME deve estar apontando para a instalação do JDK. Podemos obter detalhes da instalação do Java acessando o site da Oracle.com.

### Download do Maven

Agora, devemos baixar o Maven para máquina local. Para isso, precisamos acessar o site do Maven por meio de um buscador.

Na página de downloads, devemos ir para a seção “Files” e baixar a última versão estável. Os exemplos que veremos adiante foram feitos na versão: **apache-maven-3.8.4-bin.zip**

# Instalação

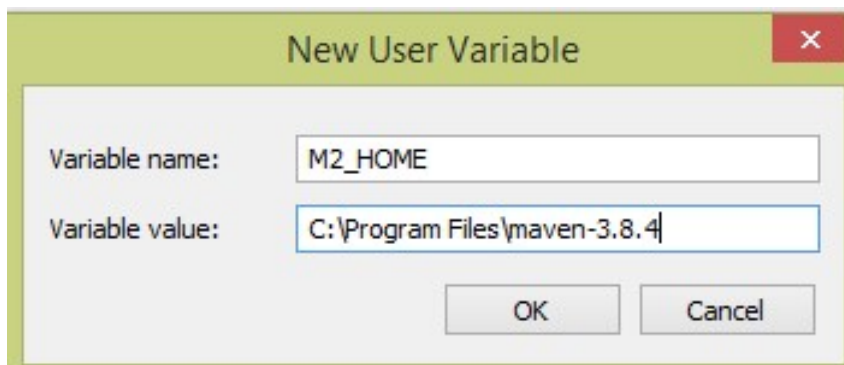
Depois de baixar o Maven, o próximo passo é descompactar o arquivo e adicioná-lo à pasta bin com o comando mvn ao PATH. As etapas detalhadas são:

## Passo 1

Renomear a pasta para **"maven-3.8.4"** e copiá-la para o endereço **"C:\Program Files"**.

## Passo 2

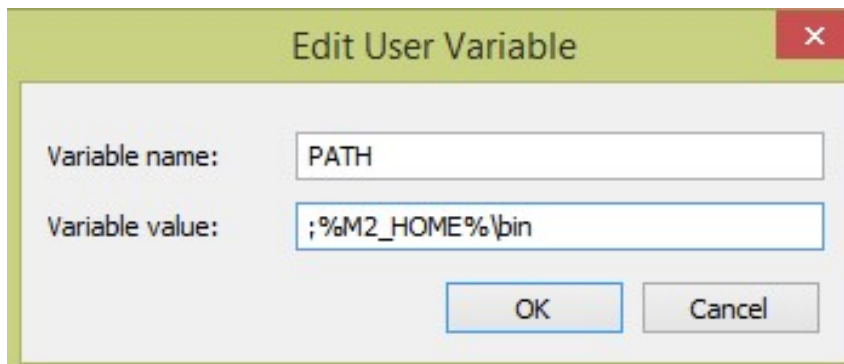
Criar a variável de ambiente **"M2\_HOME"** e apontá-la para o endereço da pasta do Maven, conforme a imagem a seguir:



Criação da variável M2\_HOME.

## Passo 3

Apontar a variável de ambiente **"PATH"** para a pasta **"bin"** do Maven, conforme a seguir:



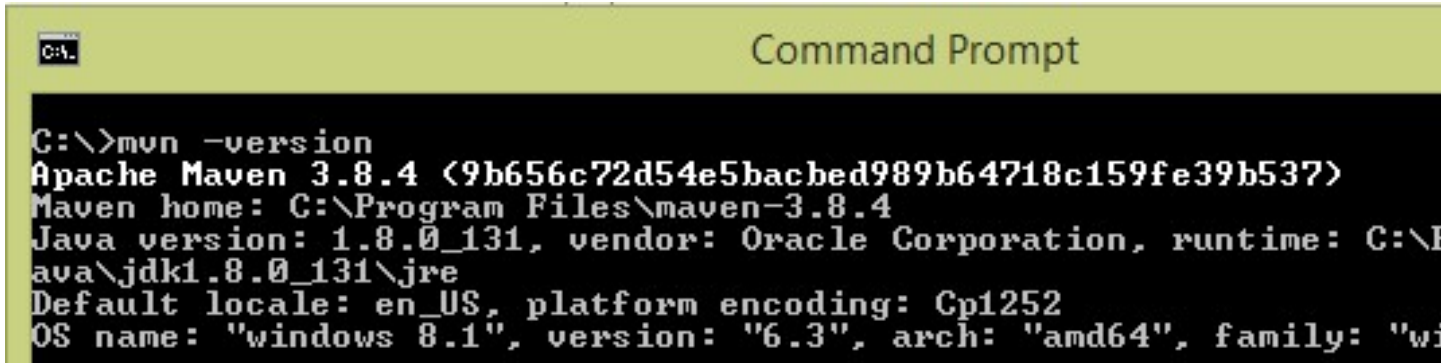
Edição da variável PATH.

## Passo 4

Agora podemos testar a nossa instalação. Basta abrir um terminal de linha de comando e digitar:

**mvn -version**

Na imagem a seguir, apresentamos o resultado:



```
C:\>mvn -version
Apache Maven 3.8.4 (9b656c72d54e5baced989b64718c159fe39b537)
Maven home: C:\Program Files\maven-3.8.4
Java version: 1.8.0_131, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_131\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 8.1", version: "6.3", arch: "amd64", family: "wi
```

Execução de linha de comando no Prompt.

Pronto! O Maven está funcionando corretamente. Agora vamos, finalmente, desenvolver o nosso primeiro exemplo.

## Criação de um projeto simples

O nosso objetivo é desenvolver um projeto simples no Maven. Como boa prática de desenvolvimento, precisamos criar uma pasta na qual vamos conceber o projeto. No nosso caso, criamos a pasta **"teste\_maven"**. O endereço completo é: **C:\teste\_maven**

Em seguida, devemos abrir o Prompt de Comando e executar a seguinte linha de código:

Prompt De Comando



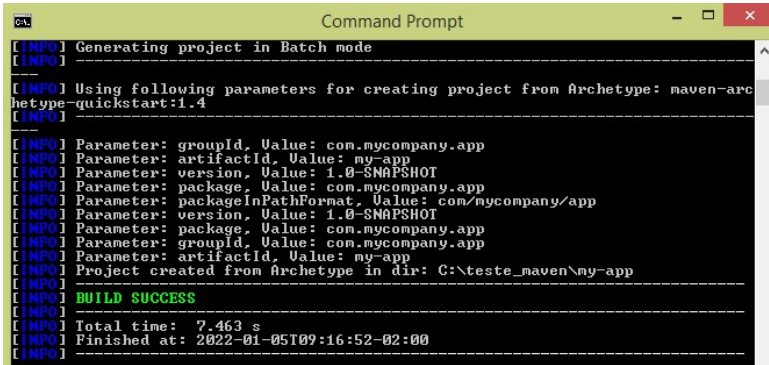
```
1 mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId
```

O comando que acabamos de executar é o arquétipo **"generate"**, que gera um projeto baseado no arquétipo **"quickstart"**. Na primeira vez que executamos o comando **"mvn"** ocorre um pouco de demora, pois o Maven está baixando os arquivos jars de plugin e outros arquivos para o repositório local.

## Atenção!

É possível que haja problemas nessa execução, pois o servidor remoto pode atingir o tempo limite antes que os downloads sejam concluídos. Caso isso ocorra, basta executar o comando novamente.

Ao final, veremos uma tela, conforme a imagem a seguir:



```
CL Command Prompt
[INFO] Generating project in Batch mode
[INFO]
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4
[INFO]
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: packageInPathFormat, Value: com/mycompany/app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: my-app
[INFO] Project created from Archetype in dir: C:\teste_maven\my-app
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 7.463 s
[INFO] Finished at: 2022-01-05T09:16:52-02:00
[INFO]
```

Resultado da execução do mvn.

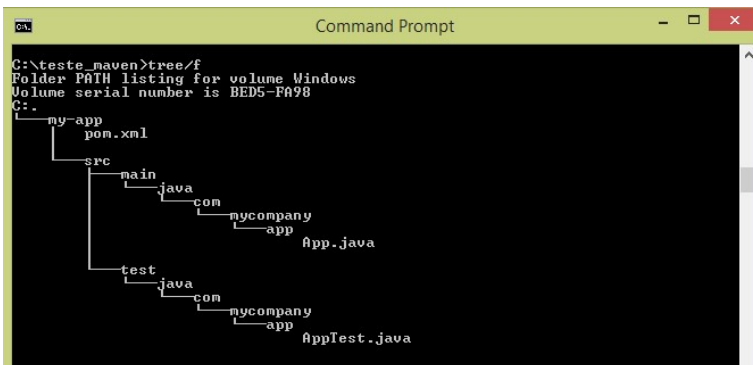
Então, vamos explorar um pouco o projeto que acabamos de criar. Para isso, precisamos digitar na linha de comando:

### Prompt De Comando



```
1 tree /f
```

Teremos como resultado a árvore de diretórios e os respectivos arquivos associados, como na imagem a seguir.



```
CL Command Prompt
C:\teste_maven>tree /f
Folder PATH listing for volume Windows
Volume serial number is BED5-FA98
C:.
|-- my-app
|   |-- pom.xml
|   |-- src
|   |   |-- main
|   |   |   |-- java
|   |   |   |   |-- com
|   |   |   |   |   |-- mycompany
|   |   |   |   |   |   |-- app
|   |   |   |   |   |   |-- App.java
|   |   |-- test
|   |   |   |-- java
|   |   |   |   |-- com
|   |   |   |   |   |-- mycompany
|   |   |   |   |   |   |-- app
|   |   |   |   |   |   |-- AppTest.java
```

Resultado do comando tree/f.

Como podemos notar na imagem anterior, a pasta **src/main/java** contém o código-fonte do projeto. Já a pasta **src/test/java** contém o código-fonte de teste e o arquivo **pom.xml**. O nosso próximo passo é analisar o arquivo pom.xml.

# Arquivo POM

O arquivo pom.xml é o modelo de objeto do projeto. Ele é a parte principal para configurar um projeto no Maven.

## Comentário

Normalmente, referimo-nos ao arquivo pom.xml apenas como POM.

O POM, de um modo geral, é um arquivo que contém muitas configurações.

O POM do nosso projeto, embora seja bem simples, possui mais de 70 linhas. Vamos destacar apenas as partes principais:

## XML



```
1 <project xmlns='http://maven.apache.org/POM/4.0.0'
2   xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
3   xsi:schemaLocation='http://maven.apache.org/POM/4.0.0
4   http://maven.apache.org/xsd/maven-4.0.0.xsd'>
5   <groupId>com.mycompany.app</groupId>
6   <artifactId>my-app</artifactId>
7   <version>1.0-SNAPSHOT</version>
8   <name>my-app</name>
9   <url>http://www.example.com</url>
10  <dependencies>
11    <dependency>
12      <groupId>junit</groupId>
13      <artifactId>junit</artifactId>
14      <version>4.11</version>
```

Nesse arquivo, podemos destacar as seguintes tags:

- **< project >** : corresponde à raiz do projeto;
- **< groupId >** : é o id do grupo do projeto;
- **< artifactId >** : é o id do artefato (projeto);
- **< version >** : é a versão do artefato no grupo especificado.

Precisamos dar uma atenção especial para as tags “groupId”, “artifactId” e “version” do POM, pois a composição delas forma o nome do artefato totalmente qualificado do projeto, que é dado por: **< groupId > : < artifactId > : < version >**.

No caso do nosso exemplo, o nome do artefato totalmente qualificado é: **"com.mycompany.app:my-app:1"**

Outra tag que vamos destacar é a **“dependencies”**. Ela é responsável pela definição das dependências utilizadas no projeto. No caso do nosso exemplo, utilizamos a dependência para trabalhar com o artefato **“junit”**, que é necessário para fazer testes.

A seguir, vamos desenvolver um projeto do Maven usando o ambiente de desenvolvimento Eclipse.

## Utilização do Maven

### Fazendo testes com o Java

Para fazer testes com o Java, vamos entrar na pasta **“my-app”** e abrir o arquivo **“App.java”**. Ele está no caminho: **C:\teste\_maven\my-app\src\main\java\com\mycompany\app**

O arquivo App.java é dado conforme o código a seguir:

Java



```
1  /**
2   * Hello world!
3   */
4  public class App
5  {
6      public static void main( String[] args )
7      {
8          System.out.println( 'Hello World' );
9      }
10 }
```

Esse arquivo foi gerado automaticamente quando executamos o comando da seção anterior, **mvn archetype:generate**, conforme as informações que passamos para o Maven, que foram:

- groupId=com.mycompany.app
- artifactId=my-app
- archetypeArtifactId=maven-archetype-quickstart
- archetypeVersion=1.4
- interactiveMode=false

Então, com o objetivo de explorarmos melhor o exercício, vamos modificar a frase do comando **System.out.println** de “Hello World” para “**Primeiro projeto correto!**”.

Feita essa modificação, devemos salvar o arquivo. Em seguida, vamos para pasta “my-app” e executar a seguinte linha de comando:

Cmd



```
1 mvn package
```

Como resultado, obtemos a mensagem que tudo funcionou corretamente, conforme podemos ver a seguir:



```
Command Prompt

Downloaded from central: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.5
/xz-1.5.jar (100 kB at 153 kB/s)
[INFO] Building jar: C:\teste_maven\my-app\target\my-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 27.042 s
[INFO] Finished at: 2023-01-05T10:24:38-02:00
[INFO] -----
C:\teste_maven\my-app>mvn package
```

Resultado da execução do mvn package.

O comando que acabamos de executar é chamado de fase. Quando uma **fase** é fornecida para o Maven, ele executará todas as fases da sequência até a fase que foi fornecida.

## Testando o arquivo JAR

Depois que executamos o “**mvn package**”, podemos observar que o Maven criou uma nova pasta chamada de “**target**” dentro da pasta “**my-app**”. Explorando a pasta “**target**”, podemos ver o arquivo: **my-app-1.0-SNAPSHOT.jar**

Trata-se do projeto compilado e empacotado em um arquivo JAR. Para testarmos, basta executar o comando a seguir no caminho da pasta “**my-app**”:

Cmd



```
1 java -cp target/my-app-1.0-SNAPSHOT.jar com.mycompany.app.App
```

Como resultado, obtemos a saída: **Primeiro projeto correto!**

## Criando um projeto Maven no Eclipse

O exemplo que fizemos foi útil para nos familiarizarmos com o Maven, mas não é viável no cotidiano de desenvolver projetos. Precisamos, então, de uma ferramenta que facilite nosso processo de programação. Uma dessas ferramentas muito popular é o Eclipse. Veja, a seguir, como utilizá-lo.

### Passo 1

Acesse o site **eclipse.org** e faça o download do Eclipse.

### Passo 2

Copie o Eclipse para uma pasta local.

### Passo 3

Entre na pasta do Eclipse e procure pelo arquivo “**eclipse.exe**”.

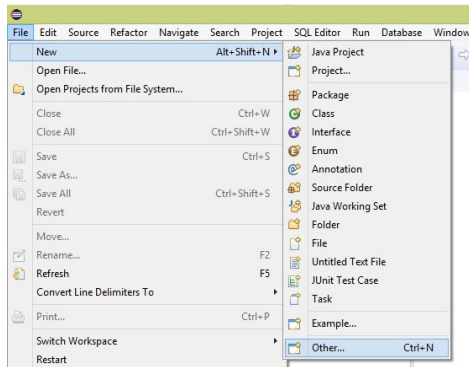


## Passo 4

Execute o arquivo e comece a utilizar o Eclipse.

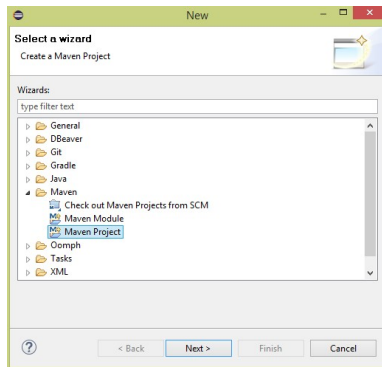
O nosso objetivo é criar um projeto Maven no Eclipse. Para isso, devemos clicar no Menu e selecionar esta sequência de passos: **File -> New -> Other**

Na imagem a seguir, essa sequência está demonstrada:



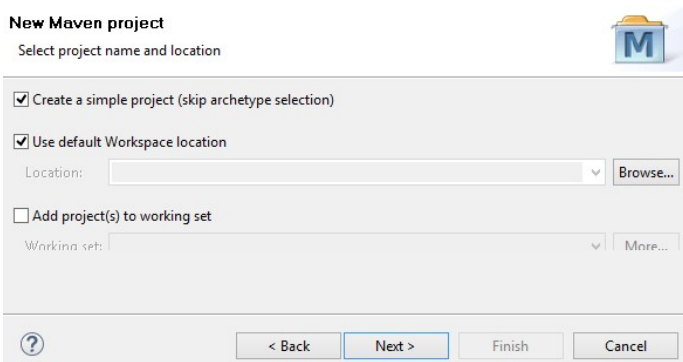
Criação de projeto no Eclipse.

Em seguida, selecionamos a opção para criar um projeto Maven, conforme podemos ver na imagem a seguir:



Criação de um projeto Maven.

Então, pressionamos o botão **“Next”**. No formulário para criar um novo projeto Maven, selecionamos a opção **“Create a simple project”** e pressionamos o botão **“Next”**. Podemos ver essa etapa na imagem:

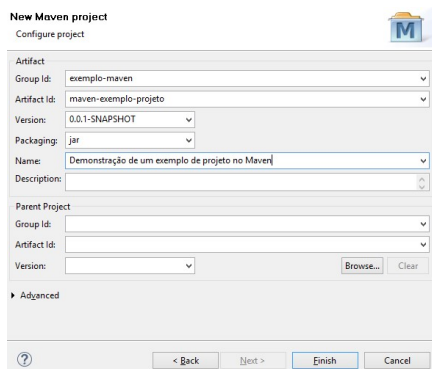


Seleção de opções para criar o projeto.

Em seguida, preenchemos os campos do formulário de acordo com as configurações a seguir:

- Group Id: exemplo-maven
- Artifact Id: maven-exemplo-projeto
- Version: 0.0.1-SNAPSHOT
- Packing: jar
- Name: Demonstração de um exemplo de projeto no Maven

Na imagem abaixo, podemos ver o formulário resultante:



Criação de projeto Maven.

Por fim, pressionamos o botão “Finish” e o projeto já está pronto.

## Conhecendo o projeto

Agora que criamos o projeto Maven no Eclipse, vamos conhecê-lo melhor. Primeiro, vamos abrir o arquivo “pom.xml” que é dado por:

Xml



```
1 <project xmlns='http://maven.apache.org/POM/4.0.0' xmlns:xsi='http://www.w3.org/2001/XMLSchema
2     <modelVersion>4.0.0</modelVersion>
3     <groupId>exemplo-maven</groupId>
4     <artifactId>maven-exemplo-projeto</artifactId>
5     <version>0.0.1-SNAPSHOT</version>
```

```
6      <name>Demonstração de um exemplo de projeto no Maven</name>
7      <dependencies>
8          <dependency>
9              <groupId>junit</groupId>
10             <artifactId>junit</artifactId>
11             <version>4.11</version>
12             <scope>test</scope>
13         </dependency>
14     </dependencies>
```

No cabeçalho do arquivo, estão as configurações que fornecemos em uma das etapas da criação do projeto. Temos também uma dependência **“junit”**, que foi colocada automaticamente, além do plugin **“maven-compiler”**.

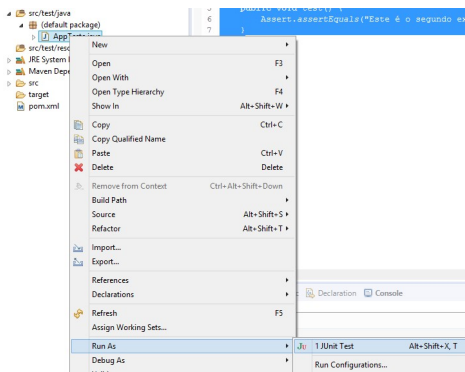
Em seguida, vamos modificar o arquivo **“AppTest.java”**, para que ele fique conforme o código a seguir:

Java



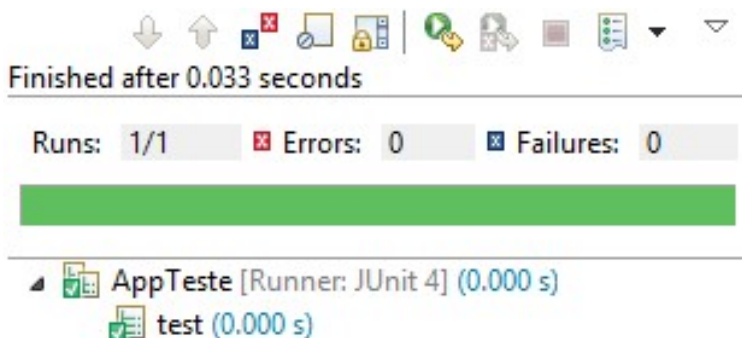
```
1  import org.junit.Assert;
2  import org.junit.Test;
3  public class AppTeste {
4      @Test
5      public void test() {
6          Assert.assertEquals('Este é o segundo exemplo!', new String('Este é o segundo exemp
7      })
8  }
```

O nosso objetivo é mostrar como executar o programa e entender melhor alguns comandos. Feita a modificação, vamos à execução. Para isso, devemos clicar com o botão direito do mouse sobre a classe **“AppTeste”**, selecionar a opção **“Run As”** e, por fim, pressionar a opção **“JUnit Test”**, como demonstrado na imagem:



Execução da JUnit.

Então, obtemos como resultado informações de que o programa funcionou corretamente, conforme podemos ver a seguir:



Resultado da execução.

Embora seja bem simples, essa etapa é fundamental em qualquer projeto de desenvolvimento, pois são os testes unitários que são usados para fazer validações. Com esse resultado, conseguimos rodar um exemplo real de um projeto Maven no Eclipse.



## Compreensão e utilização do gerenciador de dependências Maven

Veja agora os principais conceitos sobre a utilização do gerenciador de dependências Maven.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



## Falta pouco para atingir seus objetivos.

### Vamos praticar alguns conceitos?

#### Questão 1

A arquitetura MVC aumenta a robustez de um sistema. No entanto, o programador precisa ficar atento a diversos detalhes para construir cada classe conforme o papel que vai desempenhar. Nesse sentido, selecione a alternativa correta a respeito dos aspectos básicos da arquitetura MVC.

- A A utilização do gerenciador de dependências Maven oferece recursos que auxiliam na separação das camadas de um projeto.
- B Ao aplicar a arquitetura MVC, o gerenciador de dependências Maven obriga o programador a aplicar a separação de camadas.
- C Uma das dificuldades relacionadas à arquitetura MVC é o excesso de passos para o desenvolvimento de um projeto, o que limita bastante sua utilização prática.
- D O arquivo POM é a concretização prática da aplicação da arquitetura MVC.

- E O Maven e o MVC são tecnologias equivalentes para separação de responsabilidades das classes de um programa orientado a objetos.

**Parabéns! A alternativa A está correta.**

A arquitetura MVC é baseada na separação das camadas de modelo, visão e controle, no entanto, é responsabilidade do desenvolvedor aplicar essa separação. Por isso, é essencial utilizar ferramentas que ajudem nesse processo, como é o caso do gerenciador de dependências Maven.

## Questão 2

O Maven facilita bastante o desenvolvimento de projetos MVC. Na prática, o tempo e os recursos para desenvolver projetos de software são escassos, então, dispor de um gerenciador de dependências é muito importante nesse contexto. Nesse sentido, selecione a alternativa correta a respeito da utilização do Maven.

- A A limitação de recursos computacionais, como espaço disponível em memória, torna primordial a utilização de arquivos JAR fornecidos pelo Maven.
- B A utilização de dependências permite desenvolver testes unitários que são fundamentais para garantir a qualidade de um projeto.
- C Os artefatos do arquivo POM são as dependências que o projeto é obrigado a implementar.
- D Uma das vantagens de utilizar o Maven em um projeto é que os testes unitários já estão disponíveis automaticamente.
- E Para utilizar o Maven em um projeto Java, é necessário utilizar uma ferramenta de desenvolvimento como o Eclipse, por exemplo.

Parabéns! A alternativa B está correta.

O Maven é um gerenciador de dependências. Na prática, ao utilizá-lo, temos acesso a diversos recursos que facilitam o desenvolvimento, como testes, gerenciamento de banco de dados e serviços Rest, por exemplo.



## 2 - Criação de um projeto Spring Boot

Ao final deste módulo, você será capaz de analisar a criação de um projeto Spring Boot.

## Introdução à criação de um projeto Spring Boot

### O que é o Spring Boot

O **Spring Boot** é um framework de código aberto. Ele foi desenvolvido pela empresa Pivotal e é usado para criar microsserviços, que são basicamente funcionalidades de uma aplicação com as seguintes características:

- Acoplamento fraco;
- Possibilidade de implantação de forma independente.

Por causa dessas características, utilizar microsserviços se encaixa bem em um processo rápido de desenvolvimento.

## Comentário

O Spring Boot facilita o desenvolvimento de aplicações Java, pois já traz muitas dependências que podem ser conectadas ao projeto.

Um exemplo de dependência que podemos — e vamos — usar com o Spring Boot é a **Spring Web Services**, que é usada exatamente para desenvolver serviços para Web.

Outra característica vantajosa do Spring Boot é que o próprio framework auxilia no processo de programação. Além disso, mais um ponto marcante do Spring Boot é o uso de anotações, que será o nosso próximo passo.

## Anotações do Spring Boot

As anotações do Spring Boot são usadas para fornecer informações sobre um programa. Elas permitem direcionar o framework para assumir o controle e aplicar seus padrões na aplicação, quando necessário.

**Existem muitas anotações e, realmente, precisamos conhecê-las para utilizar o Spring Boot, pois é responsabilidade do programador colocá-las no código.**

A seguir, apresentamos algumas das principais anotações (WALLS, 2016):

### @Bean

É usada na marcação no nível dos métodos e indica que um método produz um [bean](#) que deve ser gerenciado pelo contêiner Spring.

### ean

Um objeto que é criado, gerenciado e destruído pelo contêiner do Spring.



## @Service

É usada no nível de classe. Ele mostra que a classe anotada é uma classe de serviço e chama APIs externas.

## @Repository

É o objeto de acesso de dados — *Data Access Object* (DAO) — que acessa o banco de dados diretamente. Indica que a classe anotada é um repositório.

## @Configuration

É uma anotação de nível de classe, sendo usada como uma fonte de definições de bean.

## @Controller

É usada para indicar que a classe é um manipulador de requisições da web.

## @RequestMapping

É usada para mapear a solicitação HTTP. É aplicada com classes e com métodos.

## @Autowired

É usada para conectar automaticamente o spring bean em métodos “**setter**”, construtor e variáveis de instância. Ele injeta dependência de objeto de modo implícito.

## @Component

É uma anotação de nível de classe que transforma a classe em um bean.

## @SpringBootApplication

Consiste nas anotações @Configuration, @ComponentScan e @EnableAutoConfiguration.

## @EnableAutoConfiguration

É colocada na classe principal do aplicativo e tem como função instruir o Spring Boot a começar a adicionar beans.

## @ComponentScan

É usada com a anotação @Configuration para permitir que o Spring conheça os pacotes a serem verificados para componentes anotados.

## @Required

É aplicada aos métodos do bean "setter". Indica que a propriedade requerida deve ser preenchida no momento da configuração no bean alvo.

## @Qualifier

É usada junto com a anotação @Autowired, quando é necessário mais controle sobre o processo de injeção de dependência.

## @RestController

É usada para criar serviços da web RESTful usando Spring MVC. O Spring RestController cuida do mapeamento de dados de solicitação para o método de solicitação. Quando o corpo da resposta é gerado, ele o converte em resposta JSON ou XML.

Concluída essa introdução ao Spring Boot, vamos estudar os aspectos fundamentais para podermos criar um projeto.

# Aspectos fundamentais para criação de um projeto Spring Boot

## Criando o projeto “Olá, Mundo!”

O nosso objetivo é criar um programa bem simples que exiba a mensagem “Olá, Mundo!”. Então, para criar um projeto no Spring Boot, precisamos acessar o site **start.spring.io**. Nesse endereço, vamos encontrar um formulário com vários campos que precisam ser preenchidos. Em seguida, vamos preencher os campos com os seguintes valores:

- **Project:** Maven Project
- **Language:** Java
- **Spring Boot:** 2.6.2
- **Group:** com.exemplo
- **Artifact:** OlaMundo
- **Name:** OlaMundo
- **Description:** Este é o meu primeiro programa com o Spring Boot
- **Package name:** com.exemplo.OlaMundo
- **Packaging:** Jar
- **Java:** 11

Além disso, vamos utilizar a dependência “**Spring Web**”. Na imagem a seguir, mostramos como fica o preenchimento:

The image shows a screenshot of the Spring Boot start.spring.io form. The 'Project' section has 'Maven Project' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '2.6.2' selected. The 'Project Metadata' section has 'Group' set to 'com.exemplo', 'Artifact' set to 'demo', 'Name' set to 'demo', 'Description' set to 'Demo project for Spring Boot', and 'Package name' set to 'com.exemplo.demo'. The 'Packaging' section has 'Jar' selected. The 'Java' section has '11' selected.

Spring Initializr.

Depois de preencher os campos, pressionamos o botão **"Generate"**. Ele vai gerar o arquivo **"OlaMundo.zip"**. Precisamos baixá-lo para uma pasta local e descompactá-lo. Então, vamos obter uma pasta **"OlaMundo"**. Podemos ver a estrutura dessa pasta executando o comando abaixo:

Cmd



1 tree /f

Como resultado do comando, obteremos a seguinte estrutura:

[illegible]

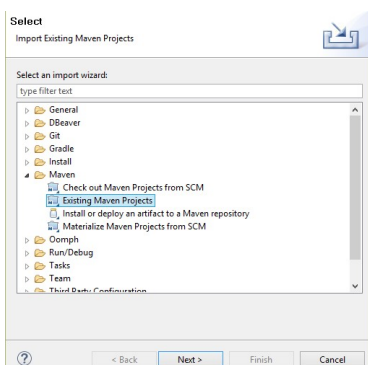
Resultado da execução do comando `tree/f`.

Dessa forma, acabamos de gerar a base do nosso projeto com Spring Boot.

## Importando o projeto "Olá, Mundo!" para o Eclipse

Agora, vamos importar o nosso projeto para o Eclipse. No menu do Eclipse, devemos selecionar a seguinte sequência: **File -> Import -> Maven -> Existing Maven Projects**

Na imagem a seguir, mostramos como funciona esse processo.

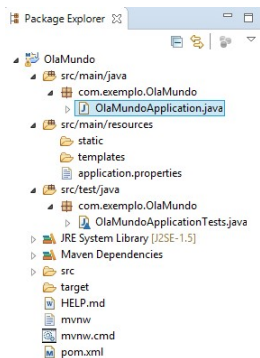


Importação de Projeto Maven.

Em seguida, pressionamos o botão **“Next”**. A próxima janela vai nos solicitar o endereço do projeto. Basta apontar para a pasta **“OlaMundo”** e pressionar o botão **“Finish”**.

## Explorando o projeto “Olá, Mundo!”

Depois de importar o projeto, podemos trabalhar com seus artefatos. Basta acessar a aba **“Package Explorer”**, conforme a imagem a seguir:



Estrutura do projeto.

O código da classe **“OlaMundoApplication”** é dado por:

Java



```
1 package com.exemplo.OlaMundo;
2 import org.springframework.boot.SpringApplication;
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 @SpringBootApplication
5 public class OlaMundoApplication {
6     public static void main(String[] args) {
7         SpringApplication.run(OlaMundoApplication.class, args);
8     }
9 }
```

Na primeira linha, vemos o nome do pacote “**com.exemplo.OlaMundo**”.

Nas duas linhas seguintes, importamos os pacotes básicos relacionados ao Spring Boot. Na quarta linha, vemos a seguinte anotação: **@SpringBootApplication**

Como já vimos, essa anotação ativa três recursos: configuração automática, varredura de componentes e o registro de configurações extras, ou seja, é uma combinação de três anotações: **@EnableAutoConfiguration**, **@ComponentScan** e **@Configuration**.

Por fim, a linha principal do nosso programa é: **SpringApplication.run(OlaMundoApplication.class, args);**

Essa é a linha responsável por iniciar a execução da aplicação.

## O POM do projeto

Agora, vamos explorar o arquivo pom.xml do projeto. Ele é dado por:

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
3      xsi:schemaLocation='http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/ma
4      <modelVersion>4.0.0</modelVersion>
5      <parent>
6          <groupId>org.springframework.boot</groupId>
7          <artifactId>spring-boot-starter-parent</artifactId>
8          <version>2.6.2</version>
9          <relativePath/> <!-- lookup parent from repository -->
10     </parent>
11     <groupId>com.exemplo</groupId>
12     <artifactId>OlaMundo</artifactId>
13     <version>0.0.1-SNAPSHOT</version>
14     <name>OlaMundo</name>
15     <description>Este é o meu primeiro programa com o Spring Boot</description>
16     <properties>
17         <java.version>11</java.version>
18     </properties>
19     <dependencies>
20         <dependency>
21             <groupId>org.springframework.boot</groupId>
```

```
22         <artifactId>spring-boot-starter-web</artifactId>
23     </dependency>
24     <dependency>
25         <groupId>org.springframework.boot</groupId>
26         <artifactId>spring-boot-starter-test</artifactId>
27         <scope>test</scope>
28     </dependency>
29 </dependencies>
30 <build>
```

Logo no cabeçalho do arquivo, podemos ver as informações que fornecemos para o Spring Initializr. Outro ponto para destacarmos é a dependência “**spring-boot-starter-web**” que será necessária para nossos serviços serem acessíveis na web.

## Configurando o arquivo application.properties

Antes de executar o projeto, precisamos configurar o arquivo “application.properties”, que se encontra no caminho: **/OlaMundo/src/main/resources/application.properties**

Então, devemos escrever o código a seguir:

Java



```
1 spring.main.web-application-type=none
```

Essa linha de código tem por objetivo desabilitar o ambiente da web. Isso é necessário para o primeiro teste que faremos. Depois, mudaremos essa configuração. O arquivo “**application.properties**” é importante para fazermos diversas configurações, por exemplo, a porta que vamos utilizar para disponibilizar os serviços e detalhes sobre o banco de dados.

## Primeira execução do projeto





```
7  @SpringBootApplication
8  public class OlaMundoApplication {
9      public static void main(String[] args) {
10         SpringApplication.run(OlaMundoApplication.class, args);
11     }
12     @RequestMapping('/')
13     String getMensagem() {
14
```

Perceba que incluímos as anotações **@RestController** e **@RequestMapping("/")**. Além disso, também criamos o método **getMensagem()**, que retorna a mensagem **“Olá, mundo!”**.

## Atualização do application.properties

Depois de modificar a classe **“OlaMundoApplication”**, precisamos fazer mais algumas modificações no arquivo **“application.properties”**, que vai ficar assim:

Java



```
1  spring.main.web-application-type=servlet
2  server.port = 8085
```

A seguir, explicamos os significados dessas propriedades:

### Propriedade “spring.main.web-application-type”



É uma enumeração dos possíveis tipos de aplicativos da web. Existem três valores possíveis:

- **none:** o aplicativo não deve ser executado como um aplicativo da web e não deve iniciar um servidor da web incorporado;
- **reactive:** o aplicativo deve ser executado como um aplicativo da web reativo e deve iniciar um servidor da web reativo incorporado;
- **servlet:** o aplicativo deve ser executado como um aplicativo da web baseado em servlet e deve iniciar um servidor da web de servlet embutido.

### Propriedade "server.port"



É a propriedade que se refere à porta do servidor. Por padrão, essa porta é a 8080.

É importante conhecer essas propriedades, pois os valores que elas devem assumir dependem das configurações fornecidas pela infraestrutura do ambiente em que o projeto vai operar.

## Execução do projeto "Olá, Mundo!"

Agora, vamos executar nosso projeto. Novamente, pressionamos o botão direito do mouse e selecionamos a opção: **Run As -> Java Application**

O programa vai executar e exibir a seguinte saída:

```
INFO 3256 --- [main] o.s.b.w.e
INFO 3256 --- [main] c.exemplo
INFO 3256 --- [nio-8085-exec-1] o.a.c.c.C
INFO 3256 --- [nio-8085-exec-1] o.s.web.s
INFO 3256 --- [nio-8085-exec-1] o.s.web.s
```

Resultado da execução da aplicação.

Agora, abrimos um navegador no endereço: **http://localhost:8085/**

E obtemos o resultado exibido nesta imagem:

# Olá, mundo!

Execução do serviço no navegador.

Esse resultado demonstra que nosso projeto “**Olá, mundo!**” já está rodando no ambiente web.

Na próxima seção, vamos conhecer algumas ferramentas que nos ajudam a fazer testes de serviços.

# Ferramentas para testar serviços RESTful

## Serviços RESTful

**RESTful** é uma abreviação para *Representational State Transfer* (REST), que em português é traduzido para transferência representacional de estado.

**RESTful é uma arquitetura que especifica restrições, como a interface uniforme, e é aplicada para serviços da web.**

No estilo arquitetural REST, dados e funcionalidades são considerados recursos e são acessados usando identificadores de recursos uniformes — do inglês *Uniform Resource Identifiers* (URIs).

Para utilizar um serviço REST, precisamos informar o tipo de método, o endereço e o recurso.

Os métodos REST podem ser:

### POST

É o que tem o objetivo de criar e é utilizado para operações de inserção. Os códigos de acertos e erros dos métodos desse tipo são:

- **Acerto:** 201, que significa “criado”;
- **Erro:** 404, que significa “não encontrado”, e 409, que significa “conflito”.

### GET

É usado para operações de leitura. Os códigos de acertos e erros dos métodos desse tipo são:

- **Acerto:** 200, que significa “OK”;
- **Erro:** 404, que significa “não encontrado”.

## PUT

É usado para operações de atualização. Os respectivos códigos de acertos e erros são:

- **Acerto:** 200, que significa “OK”, e 204, que significa “nenhum conteúdo”;
- **Erro:** 404, que significa “não encontrado”, e 405, que significa “método não autorizado”.

## PATCH

É similar ao tipo PUT, mas deve usado para mudanças parciais, enquanto o PUT é utilizado para atualizar todo recurso.

## DELETE

É usado para operações de exclusão. Os respectivos códigos de acertos e erros são:

- **Acerto:** 200, que significa “OK”;
- **Erro:** 404, que significa “não encontrado”, e 405, que significa “método não autorizado”.

Vamos utilizar agora duas ferramentas que vão nos auxiliar a testar o nosso projeto Spring Boot: Postman e Talend API Tester.

## Postman

O **Postman** é uma ferramenta muito utilizada para fazer testes de serviços REST. Veja a seguir os passos para sua utilização.

A instalação dele é bem simples. Depois de instalado, precisamos rodar a nossa aplicação “Olá, Mundo!” novamente. Em seguida, executamos o Postman e preenchemos os campos da seguinte forma:

## Passo 1

Faça o download no site **Postman.com**.

A instalação dele é bem simples.

## Passo 2

Depois de instalado, rode a aplicação **“Olá, Mundo!”** novamente.

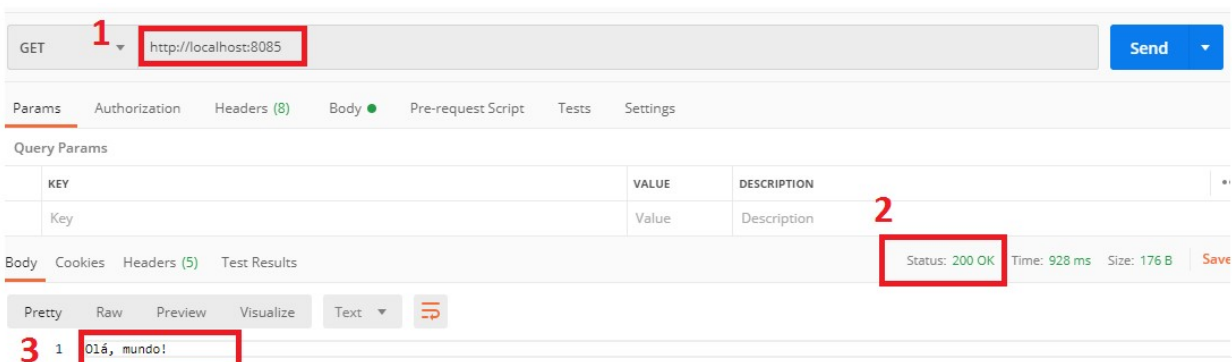
## Passo 3

Execute o Postman.

Após a execução, preencha os campos da seguinte forma:

- **Tipo de método:** GET
- **Endereço:** http://localhost:8085

Em seguida, pressionamos o botão **“Send”**. Então receberemos a mensagem **“Olá, mundo!”** e o status **“200”**, indicando que tudo funcionou corretamente. Na imagem a seguir, mostramos o resultado do teste no Postman.



Execução de serviço no Postman.

## Talend API Tester

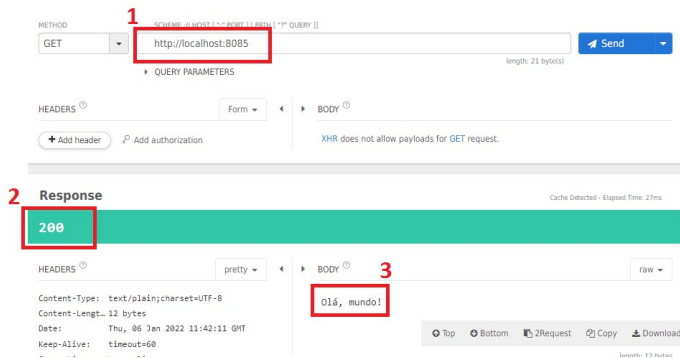
Nesta seção, mostraremos como fazer o teste da nossa aplicação com o Talend API Tester, ou, simplesmente, **API Tester**. Essa é uma ferramenta voltada para teste de serviços REST muito simples de ser utilizada. A primeira facilidade é sobre a instalação: podemos utilizá-la como um plugin do navegador. A

outra facilidade que ela oferece é a dos seus recursos visuais que ajudam bastante o trabalho de teste. Isso realmente faz a diferença nesse tipo de aplicação.

## Comentário

A navegação do API Tester é semelhante à do Postman. O primeiro passo do teste é garantir que a aplicação esteja em execução. Em seguida, preenchemos os campos da API Tester do mesmo modo que fizemos no Postman.

Veja na imagem a seguir como funciona esse processo:



Execução de serviço no API Tester.

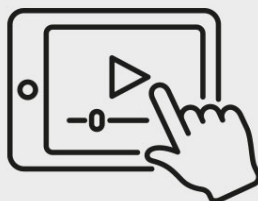
Podemos ver, portanto, que a nossa aplicação Spring Boot funcionou corretamente. Porém, os aspectos visuais do API Tester destacam com mais detalhes o resultado do teste do que o Postman, embora o objetivo de ambos seja o mesmo.



## Criação de um projeto Spring Boot

Veja agora os principais conceitos para criação de um projeto Spring Boot.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



# Falta pouco para atingir seus objetivos.

## Vamos praticar alguns conceitos?

### Questão 1

O Spring Boot é um framework que facilita a publicação de projetos na web. Desse modo, é mais simples segmentar as responsabilidades dos componentes de uma equipe nas tarefas de desenvolvimento. Nesse sentido, selecione a alternativa correta a respeito dos aspectos essenciais do desenvolvimento de um projeto com o Spring Boot:

- A Qualquer projeto no Spring Boot utiliza anotações e, em especial, a anotação `@RestController`.
- B Um projeto Spring Boot é obrigado a ter uma classe principal que inicia o ciclo de vida da aplicação.
- C Os projetos que utilizam Spring Boot são acessados apenas na porta 8080.
- D Para executar uma aplicação Spring Boot, é necessário utilizar o navegador de páginas web apontando para o endereço localhost.
- E O arquivo POM de um projeto Spring Boot contém informações sobre as configurações das propriedades da aplicação.

Parabéns! A alternativa B está correta.

O Spring Boot é um framework de código aberto voltado para o desenvolvimento de aplicações que possam realizar operações de leitura e escrita na web. Um projeto que utiliza Spring Boot pode ter diversas classes que seguem a arquitetura MVC cujo desenvolvimento é obrigação do desenvolvedor, mas qualquer projeto deve ter uma classe principal com o método “main” que inicia o ciclo de vida da aplicação.

## Questão 2

Aplicações para web são fundamentais em diversos cenários. Basicamente, algumas aplicações fazem requisições de operações de consultas e gravações que são tratadas por serviços. O Spring Boot se encaixa exatamente para atender essas necessidades, uma vez que oferece diversos recursos que facilitam o processo de desenvolvimento. Nesse sentido, selecione a alternativa correta a respeito do desenvolvimento de um projeto web com Spring Boot:

- A Para que um projeto com Spring Boot possa executar na web, é necessário configurar apenas a propriedade “spring.main.web-application-type” no arquivo POM.
- B Só é possível testar uma aplicação Spring Boot no ambiente web por meio de ferramentas como o Postman, por exemplo.
- C Para testar um serviço de aplicação Spring Boot na web com o Postman, basta apenas colocar o endereço dele.
- D O Postman é uma extensão do Spring Boot no sentido de facilitar os testes de uma aplicação web.
- E Para disponibilizar um serviço na web, um projeto com Spring Boot precisa utilizar a anotação @RequestMapping.

**Parabéns! A alternativa E está correta.**



O Spring Boot é um framework que facilita o desenvolvimento de aplicações com serviço de publicação na web. No entanto, é responsabilidade do desenvolvedor explicitar isso por meio do uso de anotações, tais como `@RestController` e `@RequestMapping`.



### 3 - Criação dos pacotes segundo o modelo MVC

Ao final deste módulo, você será capaz de identificar os aspectos para a criação dos pacotes segundo o modelo MVC.

## Aspectos fundamentais da criação de pacotes segundo o modelo MVC

### Arquitetura MVC com Spring Boot

Até aqui, vimos que a arquitetura MVC é um modelo que organiza os elementos estruturais de uma aplicação nas camadas de modelo, visão e controle. Essa organização tem grandes impactos práticos no processo de desenvolvimento, pois torna natural a separação de responsabilidades da equipe de

desenvolvimento. Quem trabalha com operações de banco de dados é responsável pela camada de modelo; os responsáveis pelos aspectos visuais da aplicação trabalham com a camada de visão; por fim, quem trabalha com a lógica de processamento dos dados e com a construção de serviços é responsável pela camada de controle.

## Comentário

Na prática, existem diversos outros aspectos de um projeto que precisam ser tratados, como controle de versões, testes unitários e de carga e, claro, os processos de integração e deploy contínuos. Entretanto, o nosso foco é o desenvolvimento do software e, nesse sentido, o MVC é uma arquitetura importante para a construção de um projeto que possa sofrer evoluções e manutenções em algumas de suas camadas com baixo impacto para as demais.

O Spring Boot é um framework que ajuda a implementar o MVC na aplicação (WALLS, 2016). Basicamente, vamos precisar criar cinco pacotes, que são:

## Entity

Nesse pacote, vamos implementar as classes da camada modelo que vão mapear as tabelas do banco de dados.

## Repository

Aqui estarão as classes **DAO** — do inglês *Data Access Object*. Essas classes são interfaces — não é a interface gráfica — que abstraem e encapsulam os mecanismos de acesso aos dados. Na prática, a interface vai herdar da classe `JpaRepository`.

## Service

Nesse pacote, estarão as classes interface, que mapeiam os serviços que serão disponibilizados pela aplicação.

## ServiceImpl

Nesse pacote, estarão as classes que implementam os métodos das classes do tipo interface do pacote Service. Além disso, utiliza as classes do pacote Repository.

## Controller

Ele possui as classes com os serviços que serão disponibilizados pela aplicação. Além disso, utiliza as classes do pacote ServiceImpl.

Na próxima seção, vamos desenvolver uma aplicação no Spring Boot aplicando MVC.

## Configuração inicial

O nosso objetivo é desenvolver uma aplicação na qual possamos gerenciar os livros que usamos nos estudos. Basicamente, ela terá as seguintes funcionalidades:

Listar todos os livros.

Consultar um livro específico.

Cadastrar um livro por vez.

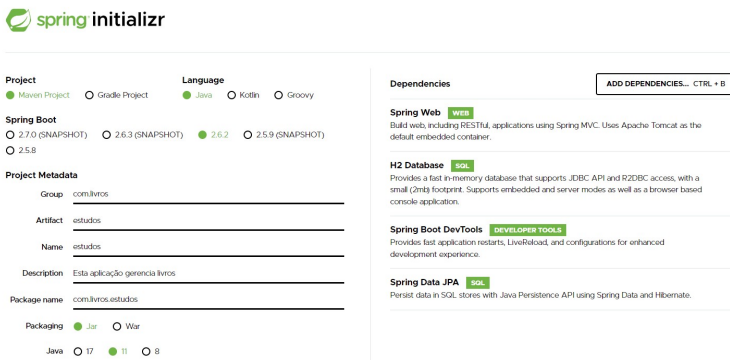
Editar um livro por vez.

Excluir um livro por vez.

O primeiro passo para criar o projeto é acessar o **Spring Initializr** e configurá-lo da seguinte forma:

- **Project:** Maven Project
- **Language:** Java
- **Spring Boot:** 2.6.2
- **Group:** com.exemplo
- **Artifact:** estudos
- **Name:** estudos
- **Description:** Esta aplicação gerencia livros
- **Package name:** com.livros.estudos
- **Packaging:** Jar
- **Java:** 11

Além disso, vamos utilizar as dependências **“Spring Web”**, **“H2 Database”**, **“Spring Boot DevTools”** e **“Spring Data JPA”**. Na imagem a seguir, é possível ver como ficou a configuração que fizemos:



The screenshot shows the Spring Initializr web application interface. On the left, under 'Project', 'Maven Project' is selected. Under 'Language', 'Java' is selected. Under 'Spring Boot', version '2.6.2' is selected. The 'Project Metadata' section shows: Group: com.livros, Artifact: estudos, Name: estudos, Description: Esta aplicação gerencia livros, Package name: com.livros.estudos, Packaging: Jar, and Java version: 11. On the right, the 'Dependencies' section lists four selected dependencies: Spring Web (Web), H2 Database (SQL), Spring Boot DevTools (Developer Tools), and Spring Data JPA (SQL). A button 'ADD DEPENDENCIES... CTRL + B' is visible at the top right of the dependencies section.

Configuração do Spring Initializr.

Depois disso, executaremos os seguintes passos:

## Passo 1

Pressionar o botão **“generate”**.

## Passo 2

Baixar o arquivo para o repositório local.

## Passo 3

Descompactá-lo.

## Passo 4

Importar o projeto no Eclipse como um “**projeto Maven**” (veja necessidade de executar “File/Restart”).

A seguir, vamos estruturar o nosso projeto para que fique em conformidade com o MVC.

## Estrutura do Projeto

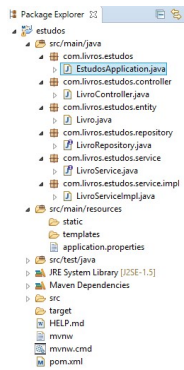
Agora, vamos criar os pacotes e as respectivas classes, para que fique em conformidade com o MVC. No quadro que segue, mostramos os pacotes que devemos criar e suas respectivas classes.

Pacote	Classe
com.livros.estudos.entity	Livro
com.livros.estudos.repository	LivroRepository
com.livros.estudos.service	LivroService
com.livros.estudos.service.impl	LivroServiceImpl

Pacote	Classe
com.livros.estudos.controller	LivroController

Quadro: Pacotes e classes da aplicação.  
Elaborado por: Sérgio Assunção Monteiro.

A seguir, podemos ver o resultado da estrutura de pacotes e classes que criamos:



Estrutura do projeto.

# Arquivo application.properties

Criados os pacotes e as classes, vamos configurar o arquivo “**application.properties**”. O conteúdo do arquivo é dado a seguir:

Java 

```
1 spring.main.web-application-type=servlet
2 server.port = 8085
3 spring.datasource.driver-class-name=org.h2.Driver
4 spring.datasource.url=jdbc:h2:mem:bootapp;DB_CLOSE_DELAY=-1
5 spring.datasource.username=sa
6 spring.datasource.password=
```

A propriedade “**web-application-type**” é um servlet e estamos trabalhando com a porta 8085. Além disso, estamos utilizando o servidor de banco de dados “**h2**”, que é muito útil para realizar testes sem a

necessidade de criar um banco de dados local. Para acessarmos o servidor de banco de dados, configuramos o usuário e a senha.

O nosso próximo passo é desenvolver cada uma das classes do projeto.

# Implementação das classes

## As classes da aplicação

Agora que criamos a estrutura do projeto, precisamos desenvolver as classes. As classes do nosso projeto são:

### EstudosApplication

É a classe principal do projeto, é ela que inicia a aplicação.

### Livro

É a classe que mapeia a tabela livro do banco de dados. No nosso caso, ela vai ter apenas dois campos, que são o identificador e o título do livro.

### LivroRepository

Essa classe estende a classe JpaRepository com os parâmetros das classes Livro e Long.

### LivroService

É uma interface com as assinaturas dos métodos que vão manipular os dados.

## LivroServiceImpl

É a classe que implementa os métodos da interface LivroService.

## LivroController

É a classe controladora que contém os serviços que a aplicação disponibiliza.

Em seguida, vamos estudar detalhadamente cada uma dessas classes.

## Classe EstudosApplication

A classe **EstudosApplication** foi criada ainda no Spring Initializr. Ela é responsável por iniciar o ciclo de vida da aplicação. A seguir, podemos ver o código da classe:

Java



```
1 package com.livros.estudos;
2 import org.springframework.boot.SpringApplication;
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 @SpringBootApplication
5 public class EstudosApplication {
6     public static void main(String[] args) {
7         SpringApplication.run(EstudosApplication.class, args);
8     }
9 }
```

Precisamos destacar alguns pontos dessa classe:



# @SpringBootApplication

É a anotação utilizada, uma vez que ela é a principal classe do projeto.

## SpringApplication.run

É o método responsável pelo início do ciclo de vida da aplicação.

### Classe Livro

A próxima classe que veremos é a Livro. Ela é a classe que mapeia a tabela. No nosso caso, é uma tabela bem simples, mas podemos ver muitos pontos importantes nela. A seguir, podemos ver seu código:

Java



```
1 package com.livros.estudos;
2 import org.springframework.boot.SpringApplication;
3 import org.springframework.boot.autoconfigure.SpringBootApplication;
4 @SpringBootApplication
5 import java.io.Serializable;
6 import javax.persistence.Column;
7 import javax.persistence.Entity;
8 import javax.persistence.GeneratedValue;
9 import javax.persistence.GenerationType;
10 import javax.persistence.Id;
11 import javax.persistence.Table;
12 import com.sun.istack.NotNull;
13 @Entity
14 @Table(name='TB_LIVRO')
15 public class Livro implements Serializable{
16     private static final long serialVersionUID = 1L;
17     @Id
18     @GeneratedValue(strategy=GenerationType.AUTO)
19     private Long id;
20     @NotNull
21     @Column(name = 'titulo')
22     private String titulo;
23     public Long getId() {
24         return id;
```

```
25     }  
26     public void setId(Long id) {  
27         this.id = id;  
28     }  
29     public String getTitulo() {  
30         return titulo;  
31     }
```

Alguns pontos devem ser destacados aqui:

## @Entity

Essa anotação indica que essa classe mapeia uma entidade do banco de dados.

## @Table

A anotação especifica qual é a tabela que a classe Livro mapeia.

## Classe Livro

Essa classe precisa ser “serializável”, pois os dados vão transitar na web.

## @Id e @GeneratedValue

Elas indicam, respectivamente, que o campo id é o identificador da classe e que ele é autoincrementado.

## @NotNull e @Column

Elas especificam que o campo “titulo” não pode receber dados nulos e que ele está mapeado com uma coluna “titulo” na tabela “TB\_LIVRO”.

## Interface LivroRepository

Analisaremos nesta seção a interface **LivroRepository**. Basicamente, vamos estender a classe **JpaRepository**. A seguir, podemos ver o código dessa interface:

```
1 package com.livros.estudos.repository;
2 import org.springframework.data.jpa.repository.JpaRepository;
3 import com.livros.estudos.entity.Livro;
4 public interface LivroRepository extends JpaRepository<Livro, Long>{
5     Livro findById(long id);
6 }
```

Quando escrevemos a `JpaRepository`, passamos como parâmetro para ela o nome da classe **Livro** e a classe **Long**. O que estamos fazendo aqui é informando ao Jpa qual a classe entidade do nosso projeto, para que possamos usar o método “**findById**”, que retorna um objeto “**Livro**” a partir de um “**id**”, ou seja, é uma consulta por identificador.

## Interface LivroService

A interface **LivroService** contém as assinaturas para os métodos que vão manipular os dados. A seguir, podemos ver o código dela:

Java



```
1 package com.livros.estudos.service;
2 import java.util.List;
3 import org.springframework.validation.annotation.Validated;
4 import org.springframework.web.bind.annotation.PathVariable;
5 import org.springframework.web.bind.annotation.RequestBody;
6 import com.livros.estudos.entity.Livro;
7 public interface LivroService {
8     public List<Livro> listaLivros();
9     public Livro listaLivroUnico(@PathVariable(value='id') long id);
10    public Livro salvaLivro(@RequestBody @Validated Livro livro);
11    public void excluirLivro(Long id);
12    public Livro atualizarLivro(Long id, Livro livro);
13 }
```

Aqui, vamos destacar as seguintes anotações:

## @PathVariable

É usada para atribuir o “id” da URI como argumento do método “listaLivroUnico”.

## @RequestBody

É a que indica ser necessário enviar um “Livro” no corpo da requisição para o método “salvaLivro”.

## @Validated

É usada para validar o parâmetro passado para o método “salvaLivro”.

## Classe LivroServiceImpl

A classe LivroServiceImpl implementa os métodos da interface LivroService. Nessa classe, estão as regras de negócio para a manipulação de dados. Veja a seguir o código dessa classe:

Java



```
1 package com.livros.estudos.service.impl;
2 import java.util.List;
3 import org.springframework.stereotype.Service;
4 import com.livros.estudos.entity.Livro;
5 import com.livros.estudos.repository.LivroRepository;
6 import com.livros.estudos.service.LivroService;
7 @Service
8 public class LivroServiceImpl implements LivroService{
9     private LivroRepository livroRepository;
10     public LivroServiceImpl(LivroRepository livroRepository) {
11         super();
12         this.livroRepository = livroRepository;
13     }
14     public List<Livro> listaLivros(){
15         return livroRepository.findAll();
16     }
17     public Livro listaLivroUnico(long id){
18         return livroRepository.findById(id);
19     }
20     public Livro salvaLivro(Livro livro) {
21         return livroRepository.save(livro);
22     }
```

```
23     public Livro atualizarLivro(Long id, Livro livroAtualizado) {  
24         Livro livro = this.listaLivroUnico(id);  
25         livro.setTitulo(livroAtualizado.getTitulo());  
26         return livroRepository.save(livro);  
27     }  
28     public void excluirLivro(Long id) {  
29         livroRepository.deleteById(id);  
30     }  
31 }
```

Vamos destacar alguns pontos dessa classe:

## @Service

Essa anotação indica que é uma classe de serviço, ou seja, que vai manipular os dados da classe modelo mediante a implementação das regras de negócio.

## livroRepository

A utilização desse objeto herda o comportamento da classe “JpaRepository” e, assim, nos poupa de implementar todos os métodos CRUD.

## atualizarLivro

Esse método tem uma estrutura um pouco diferente dos demais métodos da classe, pois primeiro faz uma busca do objeto por um id e, em seguida, faz a atualização desse objeto.

## Classe LivroController

Chegamos à última classe do nosso projeto, LivroController, que implementa os serviços que serão disponibilizados pela aplicação. Ela não deve fazer a manipulação dos dados, pois isso é responsabilidade da classe LivroServiceImpl. Podemos ver a seguir o código da classe:

Java



```
1 package com.livros.estudos.controller;  
2 import java.util.List;  
3 import org.springframework.beans.factory.annotation.Autowired;
```

```
4  import org.springframework.validation.annotation.Validated;
5  import org.springframework.web.bind.annotation.DeleteMapping;
6  import org.springframework.web.bind.annotation.GetMapping;
7  import org.springframework.web.bind.annotation.PathVariable;
8  import org.springframework.web.bind.annotation.PostMapping;
9  import org.springframework.web.bind.annotation.PutMapping;
10 import org.springframework.web.bind.annotation.RequestBody;
11 import org.springframework.web.bind.annotation.RestController;
12 import com.livros.estudos.entity.Livro;
13 import com.livros.estudos.service.LivroService;
14 @RestController
15 public class LivroController {
16     @Autowired
17     LivroService livroService;
18     public LivroController(LivroService livroService) {
19         super();
20         this.livroService = livroService;
21     }
22     @GetMapping('/livros')
23     public List<Livro> listaLivros(){
24         return livroService.listaLivros();
25     }
26     @GetMapping('/livro/{id}')
27     public Livro listaLivroUnico(@PathVariable(value='id') long id){
28         return livroService.listaLivroUnico(id);
29     }
30     @PostMapping('/livro')
```

Vamos destacar os principais aspectos dessa classe:

## @RestController

Indica que essa classe é a controladora dos serviços REST.

## @Autowired

Faz a injeção do objeto “livroService” na classe LivroController.

## @GetMapping

Indica que os métodos “listaLivroUnico” e “listaLivros” são do tipo GET - retornam valores para as requisições de consultas dos usuários.

## @PostMapping

Indica que o método “salvaLivro” é do tipo POST, ou seja, faz a persistência dos dados.

## @PutMapping

Indica que o método “atualizarLivro” é do tipo PUT, ou seja, faz a edição dos dados.

## @DeleteMapping

Indica que o método “excluirLivro” é do tipo DELETE, ou seja, faz a exclusão dos dados.

Com isso, concluímos a construção do nosso projeto. O próximo passo é vê-lo em ação. É exatamente isso que faremos na próxima seção.

# Programa em operação

## Executar o programa

Para executar o programa, precisamos clicar com o botão direito do mouse sobre a classe “EstudosApplication” e selecionar: Run As -> Application. Em seguida, veremos o log de execução no console do Eclipse, conforme a imagem a seguir:

```
INFO 7452 --- [ main] o.h.e.t.j.p.i.JtaPlatf
INFO 7452 --- [ main] j.LocalContainerEntity
WARN 7452 --- [ main] JpaBaseConfiguration$
INFO 7452 --- [ main] o.s.b.w.embedded.tomca
INFO 7452 --- [ main] com.livros.estudos.Est
```

Resultado da execução da aplicação.

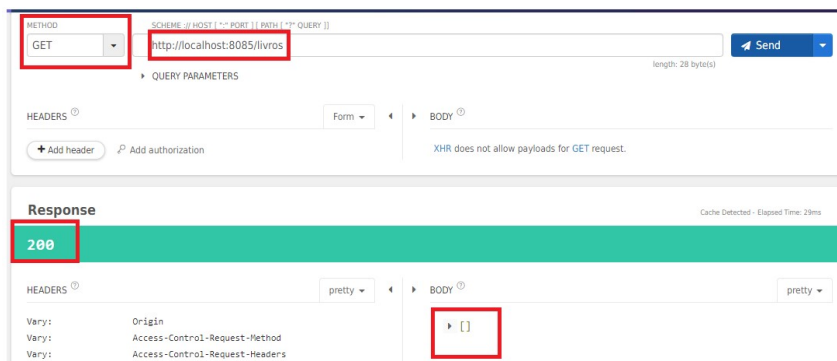
Pronto! O programa já está em operação e podemos começar a realizar os testes. Para isso, vamos utilizar o API Tester.

## Consultar livros

O primeiro teste é o de consulta de livros. Como não cadastramos nenhum livro, o programa deve retornar uma lista vazia. Para realizar o teste, devemos preencher o API Tester da seguinte forma:

- **Método:** GET
- **Endereço:** http://localhost:8085/livros

Na imagem a seguir, podemos ver os detalhes da execução.



Resultado da consulta de livros.

O resultado da execução foi conforme esperávamos, ou seja, nenhum livro cadastrado.

## Cadastrar livros

Agora vamos cadastrar dois livros. Para cadastrar um livro, devemos preencher o API Tester da seguinte forma:

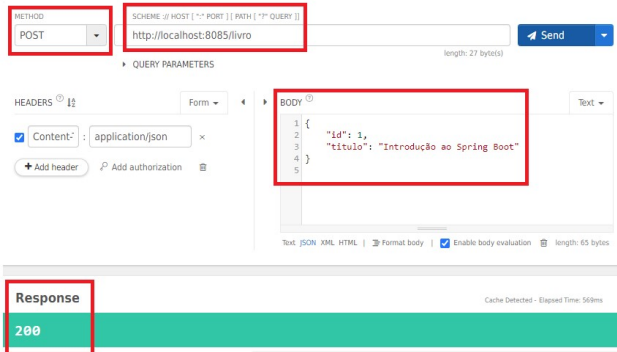
- **Método:** POST
- **Endereço:** http://localhost:8085/livro



- **BODY:**

```
{  
  "id": 1,  
  "titulo": "Introdução ao Spring Boot"  
}
```

Veja na imagem os detalhes da execução.



Resultado da consulta de livros.

Como vimos, tudo funcionou corretamente. Agora, vamos cadastrar mais um livro. Precisamos modificar apenas o campo BODY para:

#### Java



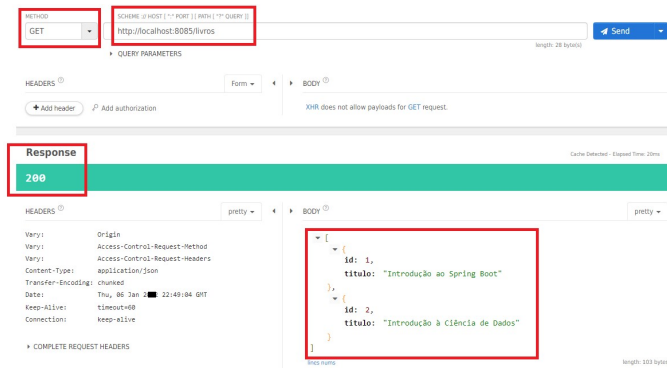
```
1 {  
2   'id': 2,  
3   'titulo': 'Introdução à Ciência de Dados'  
4 }
```

## Consultar livros novamente

Vamos então consultar os livros cadastrados no nosso sistema. Novamente, precisamos preencher o API Tester com os seguintes parâmetros:

- **Método:** GET
- **Endereço:** http://localhost:8085/livros

Os detalhes da execução constam na imagem que segue:



Resultado da consulta de todos os livros.

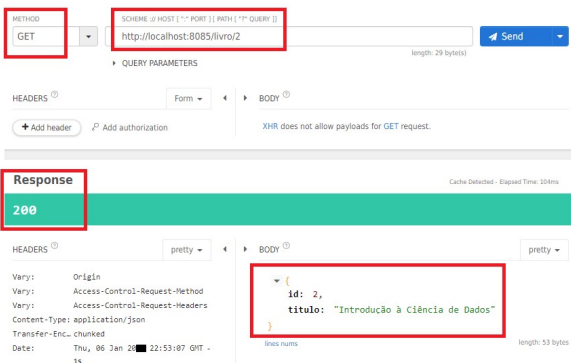
Veja que o resultado apresenta os dois livros que cadastramos.

## Consultar um livro específico

O próximo teste é a consulta de um livro específico. Vamos pesquisar pelo segundo livro. Para isso, precisamos preencher o API Tester com os seguintes parâmetros:

- **Método:** GET
- **Endereço:** http://localhost:8085/livros/2

Na imagem a seguir, podemos ver os detalhes da execução.



Consulta de um livro específico.

Como se vê na imagem, a nossa consulta trouxe apenas o livro específico que queríamos.

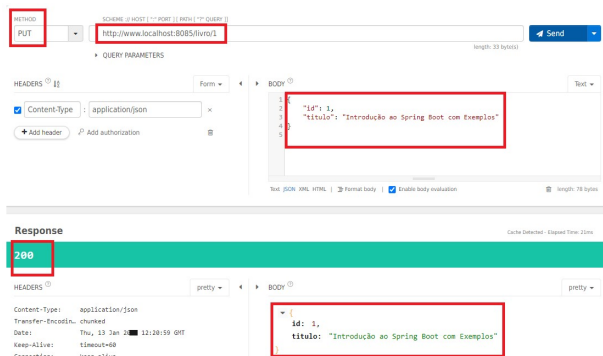
## Editar um livro específico

Dessa vez, o teste é modificar um livro específico. No caso, vamos modificar o primeiro livro que cadastramos. Para alterar um livro, devemos preencher o API Tester da seguinte forma:

- **Método:** PUT
- **Endereço:** `http://localhost:8085/livro/1`
- **BODY:**

```
{  
  "id": 1,  
  "titulo": "Introdução ao Spring Boot com Exemplos"  
}
```

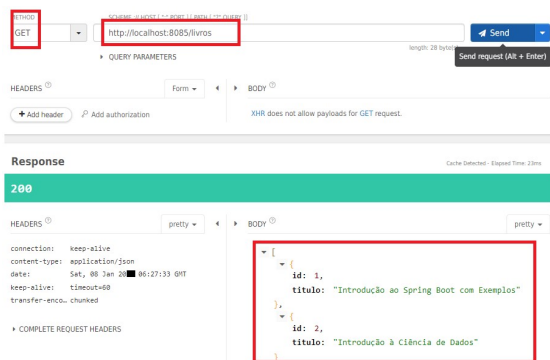
A execução detalhada está visível nesta imagem:



Alteração de um livro específico.

Fica demonstrado, assim, que a alteração foi realizada com sucesso.

Agora, vamos listar todos os livros novamente. Na imagem a seguir, vemos o resultado da consulta.



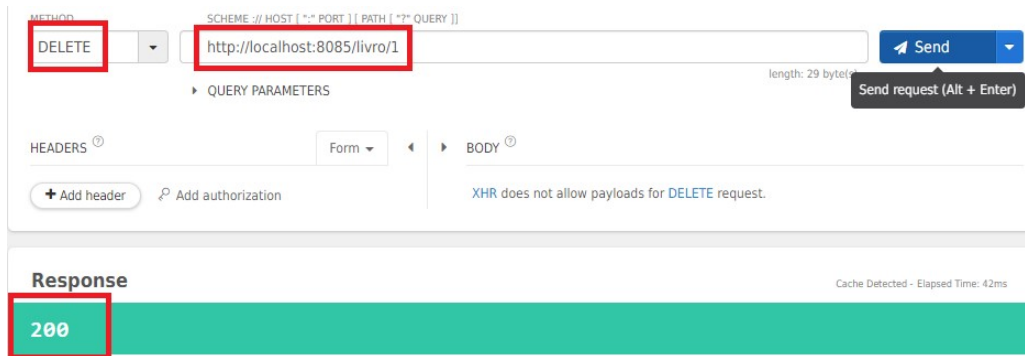
Consulta depois da alteração.

## Excluir um livro específico

O nosso próximo teste é excluir um livro específico. No caso, vamos excluir o primeiro livro que cadastramos. Para excluir um livro, devemos preencher o API Tester com os seguintes parâmetros:

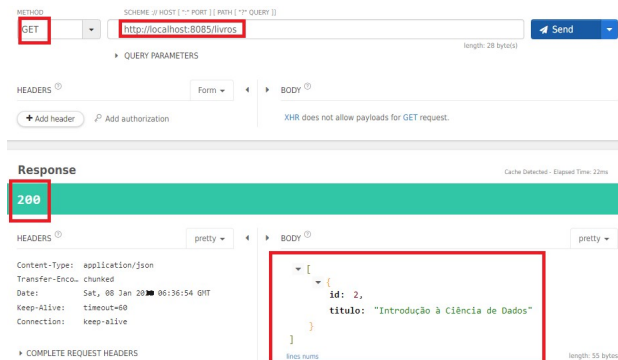
- **Método:** DELETE
- **Endereço:** http://localhost:8085/livros/1

Os detalhes da execução constam na imagem que segue:



Exclusão de um livro.

Na imagem anterior, podemos ver que a exclusão foi realizada com sucesso. Então, vamos fazer uma nova consulta, cujos detalhes da execução seguem na próxima imagem.



Consulta depois da exclusão.

Com a conclusão de todos esses testes, concluímos também a execução do nosso projeto. Bons estudos!



## Criação dos pacotes segundo o modelo MVC

Veja agora os principais conceitos sobre a criação dos pacotes segundo o modelo MVC.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



## Falta pouco para atingir seus objetivos.

### Vamos praticar alguns conceitos?

#### Questão 1

O Spring Boot fornece importantes recursos para a implementação de um projeto MVC. Isso pode ser facilmente verificado por meio de anotações que são adequadas para determinados contextos de classes com objetivos bem específicos. Nesse sentido, selecione a alternativa correta a respeito dos aspectos fundamentais da criação de pacotes segundo o modelo do MVC:

- A É responsabilidade da classe controladora o mapeamento dos campos de uma tabela e o controle da integridade dos dados.
- B As requisições de serviço são gerenciadas pelas classes do pacote “entity” que verificam se as tabelas são capazes de atender às solicitações.
- C Os serviços de uma classe correspondem aos controles internos de gerenciamento de dados.
- D Para que um projeto Spring Boot funcione, é necessário implementar as funções de inserção, consulta, modificação e exclusão de dados.

- E A interface do pacote “service” estabelece um contrato com uma classe do pacote “service.impl” que deve implementar os serviços que serão utilizados pela classe “controladora”.

Parabéns! A alternativa E está correta.

A arquitetura MVC tem muitas vantagens em termos de separação de responsabilidades das diversas partes que constituem um sistema. Isso facilita bastante a manutenção de um projeto e o trabalho em equipe. A estrutura básica de separação de responsabilidades de um projeto deve ter uma classe principal que inicia o ciclo de vida e uma interface que estabelece os serviços que devem ser implementados e utilizados por uma classe controladora.

## Questão 2

A utilização de ferramentas de teste, como o Postman, auxilia a garantir a qualidade de um projeto desenvolvido com o Spring Boot. Na realização de um teste bem-sucedido com o Postman, o analista usou a seguinte configuração:

- A As informações fornecidas na questão não são suficientes para fazer afirmações a respeito das operações que foram executadas.
- B É possível afirmar que a operação realizada foi de inserção.
- C Foi realizada uma operação de escrita que pode ser uma inserção ou uma atualização de um registro.
- D A porta utilizada pelo analista foi a 8087, caso ela seja modificada para 8080, é possível afirmar que a operação realizada foi de atualização do registro.
- E O Spring Boot determina automaticamente qual a operação que deve ser executada mesmo que elas possuam endereço igual sem a intervenção do programador.

Parabéns! A alternativa C está correta.

O método Post é associado aos serviços REST que fazem operações de escrita. Como o teste foi bem-sucedido, é possível afirmar que se trata ou de uma operação de inserção ou de atualização.

## Considerações finais

Como vimos ao longo deste conteúdo, a implementação da arquitetura MVC envolve os conhecimentos teórico e prático. O conhecimento teórico é fundamental, pois não se trata de decorar comandos. Por sua vez, o conhecimento prático é importante para implementar um sistema, fazer as diversas configurações e ser capaz de identificar soluções a partir da interpretação das mensagens de erros.

Ao utilizar o Maven como ferramenta de automação do desenvolvimento e o Spring Boot como framework, fomos capazes de implementar programas práticos que disponibilizam serviços na web. Vimos ainda ferramentas que nos auxiliam a testar esses serviços, como o Postman e o Talend API Tester.

Desenvolver um sistema com o uso do Maven e do Spring Boot é uma importante decisão para garantir padronização e velocidade no processo evolutivo do ciclo de vida de uma aplicação. Além disso, são ferramentas bastante valorizadas pelo mercado e, portanto, o domínio delas é um diferencial para o mercado de trabalho.



## Podcast

Ouçá agora um resumo dos pontos abordados ao longo do seu estudo até aqui.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.



## Explore +

Para complementar seu aprendizado, acesse o site Spring e aprenda mais sobre os recursos desse framework por meio de muitos exemplos práticos. Em especial, procure pelo termo “Microservices with Spring Boot” e aprofunde seus conhecimentos sobre o Spring Boot.

Aprofunde seus conceitos sobre fases e objetivos acessando o site oficial do Maven. Você terá acesso a muitos exemplos práticos que contribuirão para o seu estudo.

## Referências

SPÄTH, P. **Beginning Java MVC 1.0**: model view controller development to build web, cloud, and microservices applications. [S. l.]: Apress, 2021.

WALLS, C. **Spring Boot in action**. [S. l.]: Manning Publications, 2016.

### Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.



Download material

O que você achou do conteúdo?



Relatar problema