



LINGUAGEM DE PROGRAMAÇÃO

AULA 2

Prof. Rafael Veiga de Moraes

CONVERSA INICIAL

O *Spring* é indiscutivelmente o *framework* de desenvolvimento Java mais utilizado no mercado para o desenvolvimento de aplicações corporativas. Ele é composto por um conjunto de projetos que fornecem uma variedade de serviços. Esse *framework* surgiu como uma alternativa a plataforma Java EE no começo dos anos 2000, na época, denominado J2EE devido aos gargalos de desempenho das aplicações por conta da utilização dos EJBs e a sua complexa configuração.

Atualmente, ele fornece uma solução completa para o desenvolvimento de aplicações corporativas, estando presente no nosso dia a dia, seja em aplicações de *streaming*, compras on-line e inúmeras outras soluções inovadoras. Além disso, o *Spring* conta com um conjunto abrangente de extensões e bibliotecas de terceiros que permitem aos desenvolvedores criarem diferentes tipos de aplicações.

Nesta aula, iremos abordar as principais características do *Spring framework* e iniciar o processo de configuração do ambiente de desenvolvimento. Para isso, serão apresentados os softwares necessários para a construção da aplicação e abordados conceitos referentes à arquitetura de uma aplicação *web*, para, futuramente, darmos início ao processo de desenvolvimento.

TEMA 1 – INTRODUÇÃO AO SPRING FRAMEWORK

1.1 HISTÓRIA

Em outubro de 2002, Rod Johnson escreveu o livro *Expert One-on-One J2EE Design and Development*, pela editora Wrox, no qual ele elencou diversas deficiências na estrutura de componentes da plataforma Java EE. Na época, foi denominada J2EE, o que tornavam o processo de desenvolvimento de aplicações corporativas um tanto complexo.

Nesse livro, Rod Johnson propõe uma solução mais simples que se baseia na utilização de classes do tipo POJO (*Plain Old Java Objects*) e a Inversão de Controle (IoC – *Inversion of Control*) por meio da Injeção de Dependência (DI – *Dependency Injection*). Utilizando esses conceitos, Rod Johnson demonstrou em seu livro como implementar uma aplicação de reserva de assentos on-line escalonável e de alta *performance* sem a utilização dos EJBs. O livro fez um enorme sucesso, tanto que, após muitos anos de seu lançamento, ainda apresenta conceitos relevantes no desenvolvimento de aplicações para os dias atuais.

Como boa parte do código-fonte da aplicação era reutilizável, muitos desenvolvedores começaram a utilizá-lo em suas próprias aplicações. Devido a essa enorme aceitação, os desenvolvedores Juergen Hoeller e Yann Caroff entraram em contato com Rod Johnson para que juntos desenvolvessem um projeto *open-source* baseado nos conceitos abordados no livro. Em 2003, deram início ao projeto *Spring*, do inglês *Spring*, que significa “primavera”, nome proposto por Yann, representando um novo começo após o “inverno” da plataforma J2EE.

Em 2003, foi lançada a primeira versão do *Spring Framework*, que foi rapidamente adotada por diversos desenvolvedores. No ano seguinte, Rod Johnson, Juergen Hoeller, Keith Donald e Colin Sampaleanu fundaram a empresa Interface21, cujo foco estava voltado para a consultoria, treinamento e suporte do *Spring*. Desde o lançamento da primeira versão, o *framework* continuou evoluindo e adquirindo novos adeptos, tanto que, em 2006, com a versão 2.0, o *Spring* ultrapassou a marca de 1 milhão de *downloads*.

No ano de 2007, a empresa Interface21 foi rebatizada para *SpringSource*, que, dois anos mais tarde, foi adquirida pela VMWare por 420 milhões de dólares. Atualmente, o *Spring Framework* é gerenciado pela empresa Pivotal, uma *joint venture* criada pelas empresas VMWare e EMC a partir de um investimento da General Electric.

1.2 CARACTERÍSTICAS

Conforme vimos anteriormente, o *Spring* se destacou no mercado por prover uma solução mais simples que a plataforma J2EE, por meio da injeção de dependência e da utilização de classes POJO em vez dos EJBs. Aqui, abordaremos essas duas características do *Spring Framework*.

1.2.1 Injeção de dependência

A inversão de controle é um padrão de desenvolvimento muito utilizado em projetos orientados a objeto, visando diminuir o acoplamento entre as classes da aplicação. Ela consiste basicamente em tirar a responsabilidade de uma classe de instanciar objetos de outras classes das quais ela depende, tendo em vista que as funcionalidades dessas outras classes podem posteriormente vir a ser alteradas.

Isso traz inúmeros benefícios na manutenibilidade do projeto, pois deixa o código-fonte mais legível, facilitando a sua interpretação e melhora a distribuição das responsabilidades das classes que compõem a aplicação. Para entendermos esse padrão de desenvolvimento, vamos analisar a classe *Aplicacao* apresentada no quadro a seguir.

Quadro 1 – Exemplo de forte acoplamento sem IoC

```
public class Aplicacao {  
    private Tarefa tarefa;  
  
    public void executaTarefa() {  
        this.tarefa = new Tarefa();  
        tarefa.executa();  
    }  
}
```

Note que a classe *Aplicacao* possui um vínculo direto com a classe *Tarefa*, o que caracteriza um forte acoplamento, visto que o atributo *tarefa* é instanciado dentro do método *executaTarefa* da classe *Aplicacao*. De acordo com as boas práticas de programação orientada a objetos especificados no princípio SOLID, o forte acoplamento entre classes fere a regra do princípio da responsabilidade única (SRP – *Single Responsibility Principle*).

Uma das soluções possíveis para eliminar o forte acoplamento entre as classes seria, em vez de instanciar o objeto dentro do método *executaTarefa*, criar um *setter* para atribuir um objeto já instanciado ao atributo *tarefa*, tirando essa responsabilidade da classe *Aplicacao*. A solução para isso pode ser observada no Quadro 2, no qual foi inserido o método *setTarefa* de forma a eliminar o forte acoplamento entre as classes.

Quadro 2 – Exemplo de baixo acoplamento com IoC

```

public class Aplicacao {
    private Tarefa tarefa;

    public setTarefa(Tarefa tarefa) {
        this.tarefa = tarefa;
    }

    public void executaTarefa () {
        tarefa.executa();
    }
}

```

O *Spring Framework* se destacou no mercado justamente por automatizar o processo de inversão de controle por meio da injeção de dependência, dessa forma, o desenvolvedor não precisa se preocupar em instanciar um determinado objeto, ele pode deixar essa atribuição a encargo do próprio *framework*. No quadro a seguir, podemos visualizar a injeção de dependência para a classe *Aplicacao* por meio da anotação @Autowired.

Quadro 3 – Exemplo de baixo acoplamento com DI

```

public class Aplicacao {
    @Autowired
    private Tarefa tarefa;

    public void executaTarefa () {
        tarefa.executa();
    }
}

```

Dessa forma, o desenvolvedor não precisa se preocupar em gerenciar o ciclo de vida de uma classe, o próprio *Spring*, por meio do seu contêiner de IoC, faz isso de forma automática, sabendo o exato momento de quando instanciar e destruir o objeto.

1.2.2 Classes POJO

Uma classe do tipo POJO se caracteriza por conter um construtor padrão e possuir apenas os métodos de *getter* e *setter* para acessar os seus atributos. No quadro a seguir, temos um exemplo de uma classe do tipo POJO.

Quadro 4 – Exemplo de uma classe do tipo POJO

```
public class Pessoa {  
    private long id;  
    private String nome;  
    private String cpf;  
  
    public Pessoa() {  
        this.id = 0;  
        this.nome = "";  
        this.cpf = "";  
    }  
  
    public long getId() {  
        return id;  
    }  
  
    public void setId(long id) {  
        this.id = id;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public String getCpf() {  
        return cpf;  
    }  
  
    public void setCpf(String cpf) {  
        this.cpf = cpf;  
    }  
}
```

Um dos grandes problemas da plataforma Java EE no começo dos anos 2000 era o uso dos EJBs, que, embora provessem uma solução robusta, a sua configuração era complexa e exigia muito processamento, tornando-se o principal gargalo de desempenho da aplicação.

Diante disso, o *Spring* surgiu com uma ótima alternativa a plataforma Java EE, já que também provia os recursos necessários para o desenvolvimento de aplicações corporativas Java, porém com um melhor desempenho e uma configuração mais simples, visto que, em vez dos EJBs, foram adotadas classes do tipo POJO para a implementação das regras de negócio da aplicação.

TEMA 2 – CONFIGURANDO O AMBIENTE DE DESENVOLVIMENTO

Antes de codificarmos a nossa aplicação, precisamos previamente baixar e instalar as ferramentas de desenvolvimento pertinentes ao projeto que será desenvolvido. Para isso, iremos

listar os softwares que você deverá instalar na sua máquina para que possamos dar sequência ao curso.

2.1 IDE

O primeiro software a ser instalado é o ambiente integrado de desenvolvimento (IDE – *Integrated Development Environment*), ferramenta de desenvolvimento que fornece funcionalidades que facilitam o processo de codificação e de gerenciamento dos recursos da aplicação. O *Spring Framework* é compatível com diversas IDEs, porém adotaremos o Eclipse como software de desenvolvimento da aplicação a ser desenvolvida, visto que é uma IDE poderosa, *open-source* e largamente utilizada no mercado.

O Eclipse pode ser baixado no site. Ao executar o instalador, serão listadas diversas versões do Eclipse, na qual o usuário deverá selecionar uma delas para dar sequência a instalação. Selecione a opção *Eclipse IDE for Enterprise Java and Web Developers* conforme mostrado na figura a seguir.

Figura 1 – Interface do instalador do Eclipse



Após selecionar a opção citada anteriormente, será exibida a tela de configuração da instalação, devendo o usuário informar o diretório no qual a IDE será instalada, conforme podemos observar na

Figura 2. Configurado o diretório de instalação, basta clicar no botão *Install* para dar início ao processo de instalação do software.

Figura 2 – Tela de instalação do Eclipse



2.2 JDK

Além da instalação da IDE, o desenvolvedor Java também precisa instalar o kit de desenvolvimento Java (JDK – *Java Development Kit*). O JDK é composto por diversos componentes necessários para o desenvolvimento, gerenciamento e monitoramento das aplicações a serem desenvolvidas em linguagem Java. Dentre eles destacam-se a seguir alguns.

- **JRE**: máquina virtual e bibliotecas.
- **Javac**: compilador da linguagem Java.
- **Javadoc**: ferramenta de documentação.
- **Jdb (*Java Debugger*)**: ferramenta de depuração.
- **APIs**: conjunto de serviços da linguagem Java.

2.2.1 JRE

Para que uma aplicação Java possa ser executada em uma determinada máquina, é necessário que nela esteja instalado o ambiente de execução Java (JRE – *Java Runtime Environment*). O JRE é

plug-in composto pela máquina virtual Java (JVM – *Java Virtual Machine*) e a biblioteca padrão da plataforma Java.

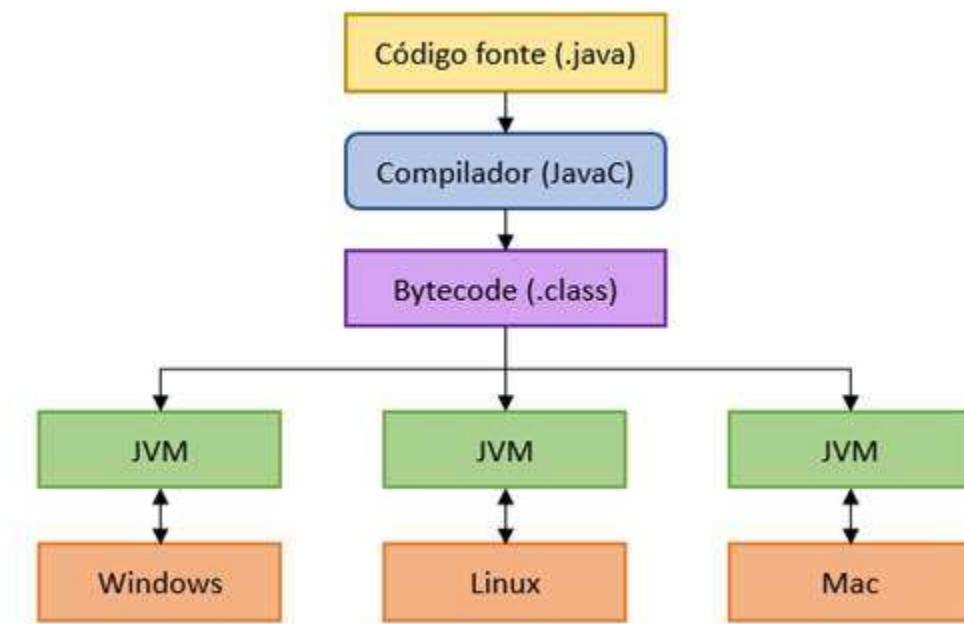
2.2.1.1 Máquina virtual Java

Uma das principais características da linguagem Java é a sua compatibilidade com diferentes sistemas operacionais. O componente que torna possível o desenvolvimento de aplicações Java independentes da plataforma é a máquina virtual Java (JVM – *Java Virtual Machine*).

Diferentemente de outras linguagens de programação, ao compilar o código-fonte de uma aplicação Java, não é gerado um arquivo executável, e sim um arquivo chamado *bytecode*. Ele é um arquivo intermediário gerado pelo compilador da linguagem Java, que será interpretado pela JVM e não contém qualquer vínculo com o sistema operacional do ambiente de desenvolvimento. Isso significa que o *bytecode* é um arquivo independente da plataforma, podendo ser executado em qualquer ambiente que tenha suporte a JVM, por exemplo, os sistemas operacionais Windows, Mac e Linux.

A JVM contém toda infraestrutura necessária para a execução de uma aplicação Java, convertendo o código em *bytecode* em código de máquina, compatível com a plataforma na qual ele está sendo executado, conforme nos mostra a figura a seguir.

Figura 3 – Funcionamento de uma aplicação Java



A JVM possui alguns módulos.

- **Loader:** responsável por carregar o *bytecode* na memória da máquina virtual.
- **Verificador:** analisará se o código em *bytecode* não contém erros.
- **Interpretador:** converterá o *bytecode* em um código de máquina compatível com a plataforma de execução.
- **Coletor de lixo:** fará o gerenciamento de memória da máquina virtual.
- **Compilador JIT (Just In Time):** responsável por otimizar a *performance* da aplicação durante a sua execução.

2.2.1.2 Biblioteca padrão

A biblioteca padrão da linguagem Java contém um conjunto de pacotes que fornecem uma série de recursos necessários para a execução de qualquer aplicação desenvolvida em linguagem Java. Além disso, as bibliotecas auxiliam o programador no processo de desenvolvimento da aplicação, reduzindo a complexidade da implementação de diversas funcionalidades.

Supondo que a aplicação irá armazenar as informações do sistema em um banco de dados relacional, seria necessário que o programador desenvolvesse uma interface de comunicação com o banco de dados para gerenciar os dados da sua aplicação. Porém, isso não é necessário, pois na biblioteca padrão da linguagem Java existe um pacote chamado *java.sql*, que contém a implementação de diversas classes e interfaces de comunicação com o banco de dados. Entre eles destacam-se os que se seguem.

- **Classe DriverManager:** utilizada para definir os parâmetros de acesso ao servidor de banco de dados: IP do servidor, porta de conexão, nome da fonte de dados, entre outros.
- **Interface Connection:** utilizada para estabelecer a conexão com o servidor de banco de dados.
- **Interface ResultSet:** utilizada para obter os resultados de uma *query* (consulta).
- **Interface PreparedStatement:** utilizada para executar um comando SQL.

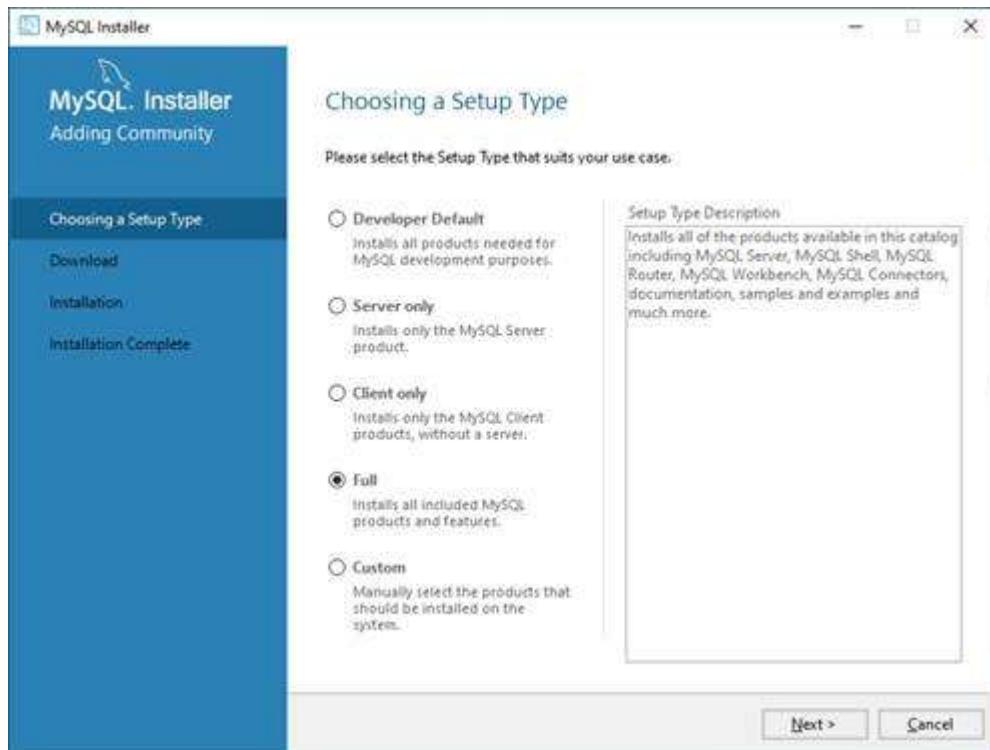
Além da biblioteca padrão, o desenvolvedor também pode adicionar ao projeto bibliotecas de terceiros, bastando para isso importar o arquivo JAR (*Java Archive*) do *framework* que será utilizado.

2.3 SGBD

O SGBD (Sistema Gerenciador de Banco de Dados) é mais uma ferramenta imprescindível para o desenvolvimento da aplicação, visto que necessitamos de um software que permita efetuar o armazenamento de dados do sistema. Entre os vários SGBDs disponíveis no mercado, vamos adotar o MySQL para o desenvolvimento da aplicação, tendo em vista que é uma ferramenta *open-source* robusta e utilizada por grandes corporações.

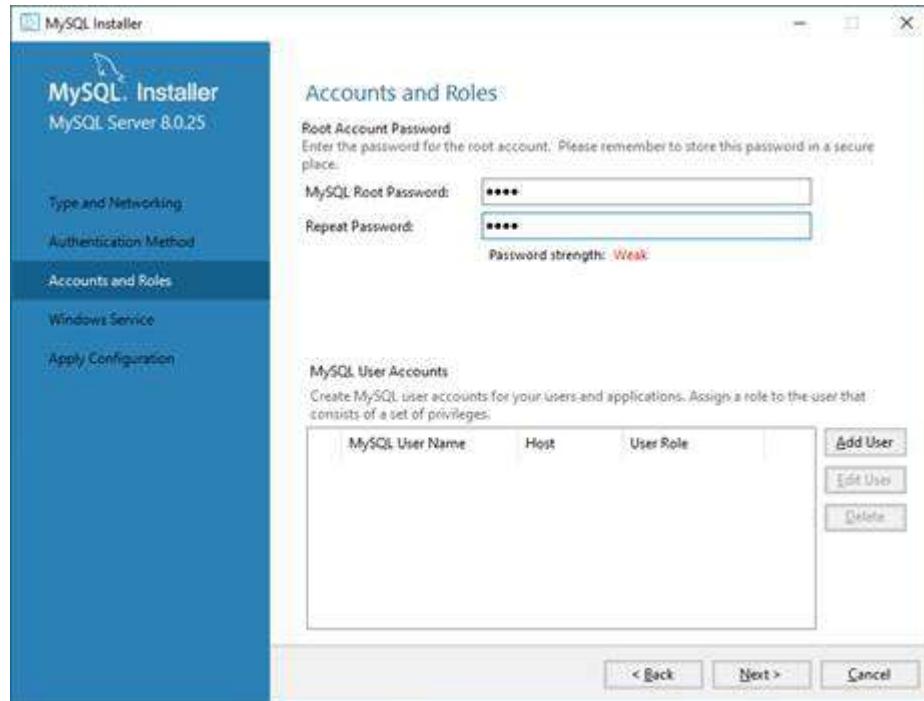
Para realizar a instalação do MySQL, é necessário baixar e instalar o MySQL *Workbench*, software que comprehende o SGBD mais a interface gráfica para executarmos os comandos da Linguagem SQL. O instalador pode ser baixado por meio do site dev.mysql. Ao executar o instalador do MySQL *Workbench*, opte pelo tipo de instalação *Full* conforme a figura a seguir.

Figura 4 – Tela inicial do instalador MySQL *Workbench*



Durante a instalação, serão baixados os arquivos pertinentes ao tipo da instalação selecionada e, caso necessário, será solicitada a instalação de softwares complementares que podem ser baixados manualmente pelo próprio instalador. Boa parte do processo de instalação consiste em ir clicando no botão *Next*, porém duas telas merecem destaque. A primeira é a tela de cadastro *Accounts e Roles*, na qual é definida a senha do usuário administrador do MySQL, conforme podemos verificar na figura a seguir.

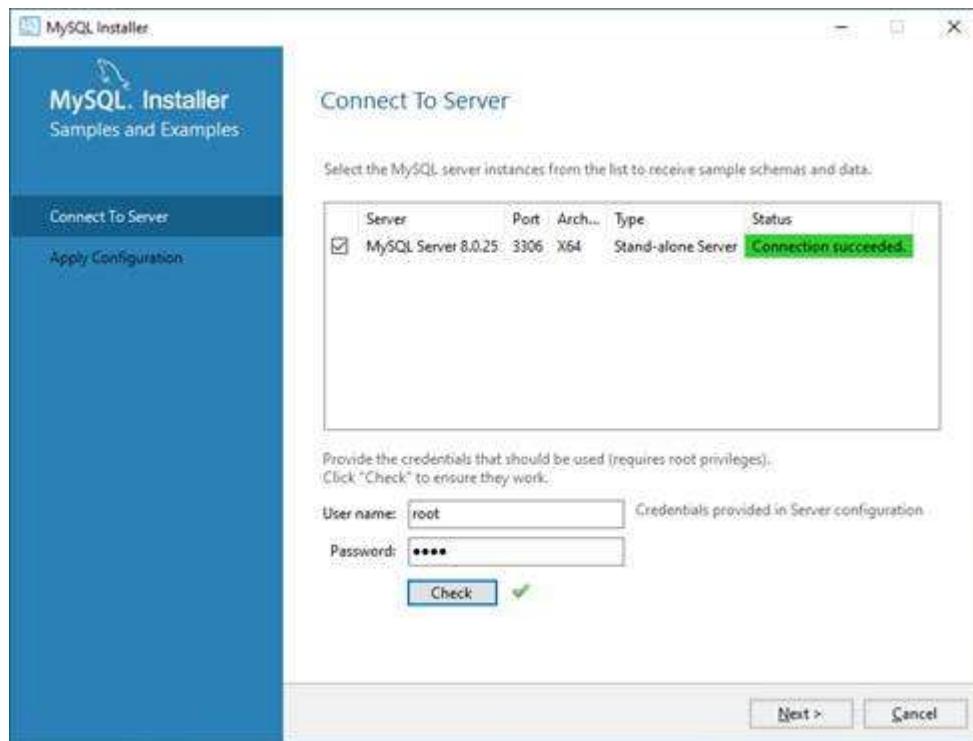
Figura 5 – Tela Accounts and Roles do instalador MySQL Workbench



Cadastre uma senha de fácil recordação, pois, caso você venha a esquecê-la futuramente, terá que reinstalar o SGDB novamente, já que algumas funcionalidades do MySQL precisam de permissão de acesso do usuário administrador.

A segunda tela que merece destaque é a tela *Connect To Server*, nela podemos verificar se a instalação foi realizada com sucesso. Para isso, informe no campo *User name* o usuário administrador "root" sem aspas, no campo *Password*, a senha definida anteriormente na tela *Accounts and Roles* e, em seguida, clique no botão *Check*. Caso a instalação tenha sido realizada com sucesso, será exibida a mensagem *Connection succeeded* com o fundo verde na coluna *Status*, conforme pode ser visto na figura a seguir.

Figura 6 – Tela *Connect To Server* do instalador MySQL Workbench



2.4 POSTMAN

É uma ferramenta que foi desenvolvida para auxiliar os desenvolvedores na criação e teste das APIs. Por meio dele, veremos as APIs que serão desenvolvidas durante o curso, tornando o processo de desenvolvimento mais rápido e objetivo, uma vez que está a todo momento abrindo um *browser* para realizar os testes. O Postman pode ser baixado no site e possui suporte para Windows, Mac ou Linux.

Figura 7 – Interface do *Postman*

The screenshot shows the Twitter API v2 Explore interface. The left sidebar lists various workspace sections like Home, Workspaces, Reports, and Explore. The main area displays a successful API call to the 'Single Tweet' endpoint. The URL is https://api.twitter.com/2/tweets/121. The response body is shown as JSON:

```
{"data": [{"id": "140321812886128420", "text": "Honoree Mitchell went down after a collision with Paul George tonight. end of Game 3. https://t.co/P9L0m0k0dI"}]}
```

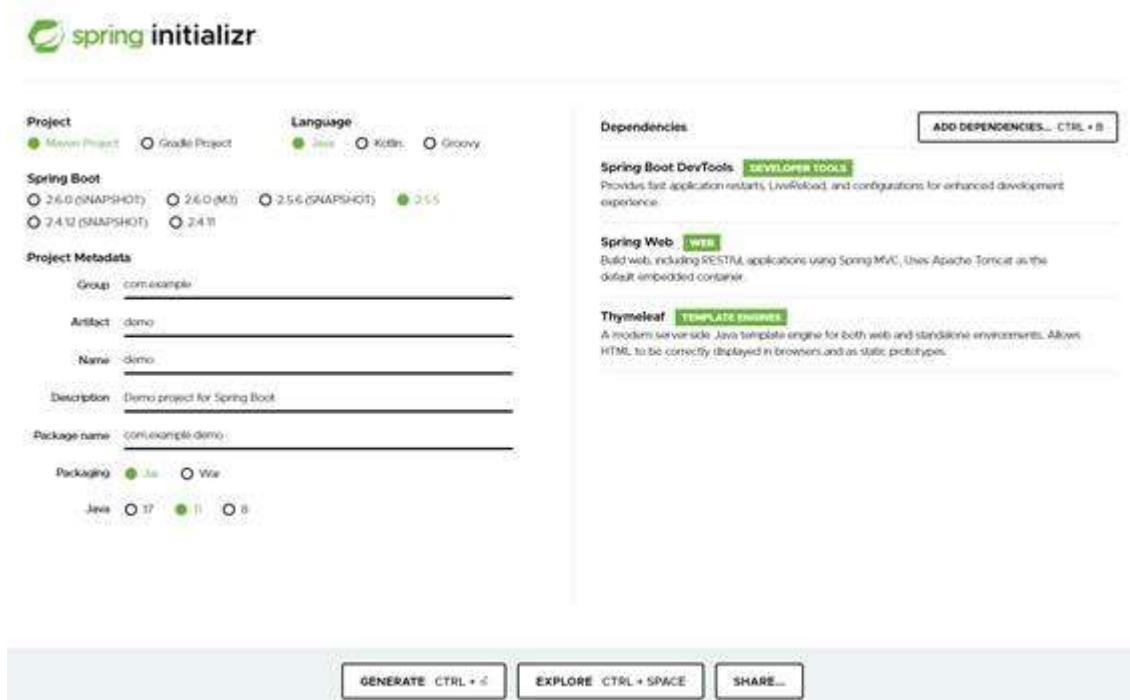
Below the JSON, a note states: "OAuth 2.0 (Token) authentication required if any of the following fields are included in the request: user_fields, user_entities, user_mentions, quoted_tweet_id, referenced_tweets, replies, reply_count, source, author_id."

TEMA 3 – SPRING BOOT

Começar um projeto do zero nem sempre é uma tarefa fácil, pois precisamos configurar diversos recursos da nossa aplicação antes de começarmos a codificação das regras de negócio. Esse processo basicamente consiste em baixar, instalar e configurar as bibliotecas, *frameworks* e serviços que serão utilizados no desenvolvimento da aplicação.

Para agilizar esse processo moroso da configuração inicial de um projeto, o time de desenvolvimento do *Spring* disponibilizou o *Spring Boot*, uma ferramenta que permite ao desenvolvedor criar um projeto *Spring* de forma rápida e fácil, por meio do site.

Figura 8 – Tela de configuração do *Spring Boot*



Ao acessar o site anterior, será exibida a tela de configuração do projeto *Spring*, conforme podemos observar na figura anterior. A configuração do projeto está dividida em cinco grupos.

- **Project:** define a ferramenta de gerenciamento e automação de construção do projeto.
- **Language:** define a linguagem de programação do projeto.
- **Spring Boot:** define a versão do *Spring* que será utilizada no projeto.
- **Project Metadata:** são as informações referentes ao projeto.
- **Dependencies:** lista de dependências que serão utilizadas no projeto.

A seguir, veremos com mais detalhes como esses grupos devem ser configurados.

3.1 PROJECT

No grupo *Project*, deve-se identificar qual será a ferramenta de gerenciamento e automação de construção do projeto, ou de forma mais informal, qual será a ferramenta de *build* (construção) do projeto. Nesse grupo, podemos optar entre duas ferramentas diferentes: Maven e Gradle.

Essas ferramentas serão responsáveis por gerenciar as dependências do projeto, ou seja, cabe a elas baixar, instalar e configurar as bibliotecas, *frameworks* e serviços que serão utilizados no desenvolvimento da aplicação. Além disso, outra responsabilidade delas é a geração do arquivo de *build* do projeto, para que possamos realizar o processo de *deploy* (implantação) da aplicação.

3.2 LANGUAGE

No grupo *Language*, deve-se identificar qual será a linguagem de programação que será utilizada para o desenvolvimento do projeto. Nesse grupo, temos três opções: Java, Kotlin e Groovy. Entre elas, iremos adotar a linguagem Java.

3.3 SPRING BOOT

No grupo *Spring Boot*, deve-se identificar qual será a versão do *Spring* que será utilizada para o desenvolvimento do projeto. Para isso, devemos compreender como se dá a nomenclatura das versões a fim de selecionar a mais adequada para o projeto que virá a ser desenvolvido. Note que algumas versões estão identificadas com M3, *SNAPSHOT* ou apenas pelo número da versão. As versões estão divididas em três grupos.

- **GA** (*General Availability*): são as versões que foram lançadas ao público (versão estável) e nunca sofrerá qualquer alteração.
- **PRE**: são as versões de pré-lançamento e são identificadas pela letra M (acrônimo para marco). As versões de pré-lançamento têm como objetivo disponibilizar os novos recursos do *Spring* para que os desenvolvedores possam testá-los. Como as alterações estão em processo de validação (versão instável), essas versões podem apresentar erros durante a execução do projeto e não devem ser utilizadas para o desenvolvimento de um projeto comercial.

- **SNAPSHOT**: são atualizadas frequentemente e contém as alterações mais recentes do *framework* e, assim como as versões de pré-lançamento, não devem ser utilizadas para o desenvolvimento de um projeto comercial.

Para exemplificar esse processo, o ciclo de vida da versão 1.0.0 seguiria o seguinte processo.

- **1.0.0 SNAPSHOT**: essa versão é atualizada frequentemente com as alterações mais recentes.
- **1.0.0 M1**: sempre que é atingido um marco de desenvolvimento é lançada uma versão de pré-lançamento. A cada marco de desenvolvimento, o identificador do marco é incrementado em uma unidade, dessa forma, o primeiro marco de desenvolvimento é identificado pela sigla M1, o segundo por M2 e assim sucessivamente.
- **1.0.0 GA**: a partir do momento que foi lançado o último marco e a versão instantânea está completa e não apresenta qualquer erro, a versão é disponibilizada para o público. A partir desse momento, qualquer erro que for identificado e novas alterações serão desenvolvidas em uma nova versão.

3.4 PROJECT METADATA

No grupo *Project Metadata*, serão definidas as características do projeto e contém os seguintes campos.

- **Group**: define o nome do grupo.
- **Artifact**: define o nome do artefato.
- **Name**: define o nome do projeto.
- **Description**: descrever o objetivo do projeto.
- **Package name**: define o nome do pacote principal do projeto.
- **Packaging**: define como o projeto será compactado.
- **Java**: define a versão do Java.

3.5 DEPENDENCIES

As dependências do projeto são referentes a bibliotecas, *frameworks* e serviços que serão utilizados no desenvolvimento do projeto. Sempre que quisermos adicionar algum desses recursos, basta adicionarmos a dependência no arquivo pom.xml do projeto. Adicionada a dependência, esta

será baixada e adicionada ao projeto de forma automática pela ferramenta de gerenciamento de construção do projeto (Maven ou Gradle) escolhida no item *Project*.

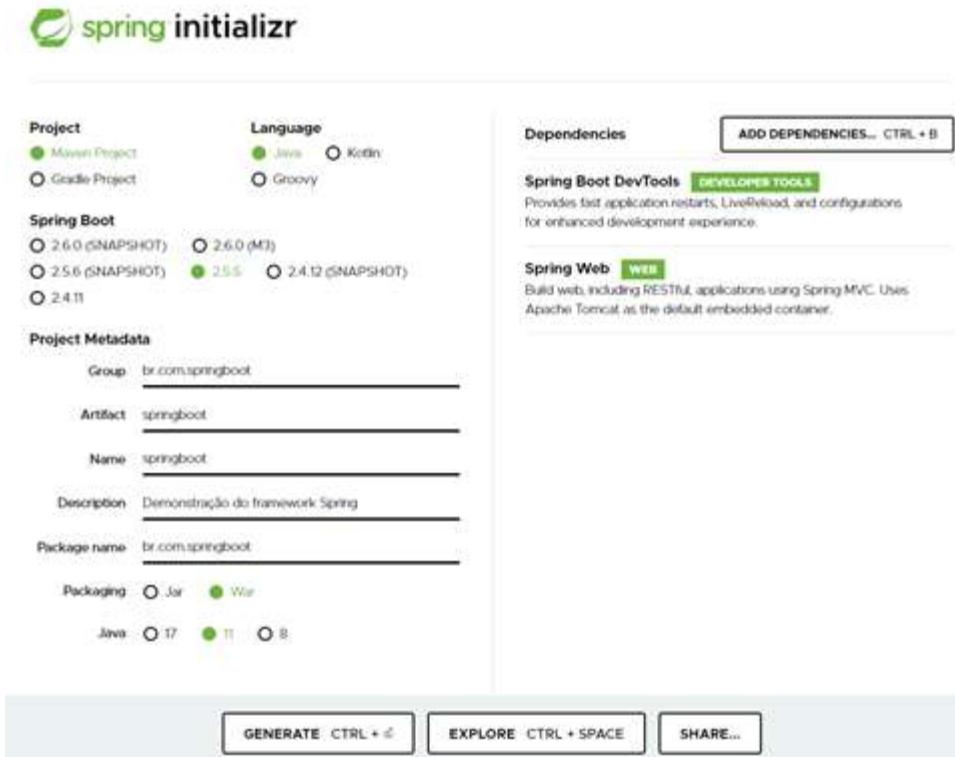
Na tela de configuração do *Spring Boot*, são listadas as dependências mais utilizadas pelos desenvolvedores no item *Dependencies*. Nesse item, já podemos vincular ao projeto algumas dessas dependências, tornando o processo de configuração do projeto bem mais fácil e prático. A seguir, serão listadas algumas das dependências disponíveis para o desenvolvedor.

- **Spring Boot DevTools**: ferramenta que contém configurações para uma experiência de desenvolvimento aprimorada e destaca-se principalmente por reiniciar a aplicação automaticamente a cada alteração no código fonte.
- **Spring Web**: permite a criação de aplicações *web* trazendo por padrão o *servlet container* Apache Tomcat já configurado para prover os serviços da aplicação.
- **Spring Session**: fornece uma API para a implementação e gerenciamento das informações da sessão do usuário.
- **Spring Web Services**: fornece suporte para a criação de *Web Services* do tipo *SOAP*.
- **Thymeleaf**: ferramenta alternativa as especificações JSF e JSP para desenvolvimento das páginas *web*.
- **JDBC API**: API que permite efetuar a conexão com banco de dados relacional.
- **Spring Data JPA**: especificação para realizar a persistência de dados, usando JPA com *Spring Data* e *Hibernate*.
- **Spring Security**: ferramenta para autenticação de usuários altamente personalizável para controle de acessos da aplicação.

TEMA 4 – CRIANDO UM PROJETO COM O SPRING BOOT

Agora que sabemos como configurar um projeto pelo *Spring Boot*, vamos criar o nosso projeto para iniciarmos o processo de implementação, utilizando o *framework Spring*. O primeiro passo é acessarmos o *Spring Boot* pelo *link* apresentado anteriormente.

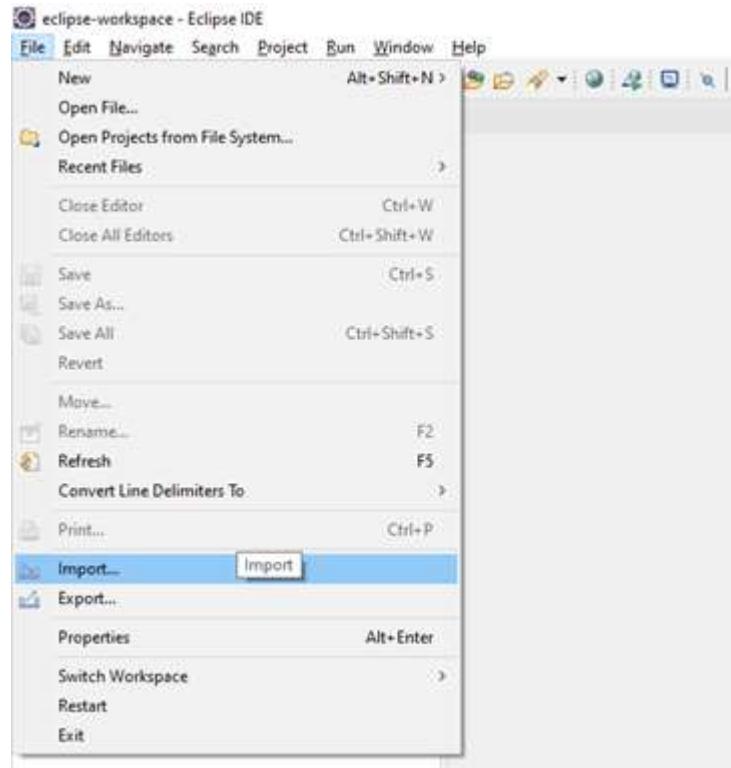
Ao acessar a página de configuração do projeto, adotaremos estas configurações conforme a figura a seguir.

Figura 9 – Configuração do projeto utilizando o *Spring Boot*

Após efetuar a parametrização anterior, basta clicar no botão *Generate* ou pressione a combinação de teclas *Ctrl + Enter* para baixar o arquivo de configuração do projeto. Será gerado um arquivo zip com o nome do projeto, o qual deve ser extraído para posteriormente ser importado na IDE que será adotada para o desenvolvimento da aplicação. Como iremos utilizar o Eclipse como IDE, demonstraremos passo a passo como realizar a importação do projeto por meio dessa ferramenta de desenvolvimento.

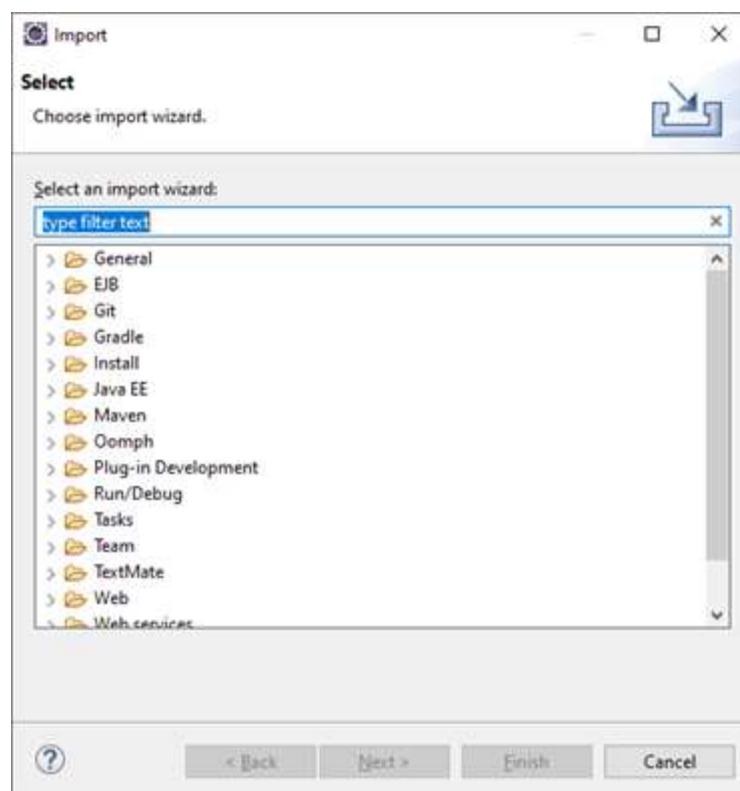
Inicializando o Eclipse, importaremos o arquivo de configuração do projeto utilizando a tela de importação pelo menu "*File > Import*", conforme podemos visualizar na Figura 10.

Figura 10 – Menu para importação de um projeto já existente no Eclipse



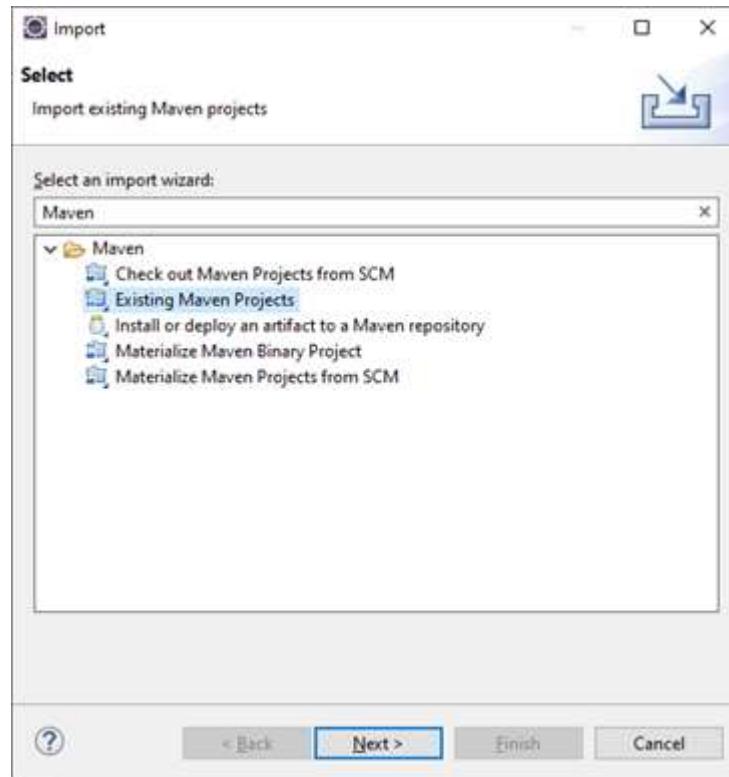
Ao acessar o menu citado anteriormente, será exibida a tela de importação conforme ilustra a Figura 11.

Figura 11 – Tela de importação do Eclipse



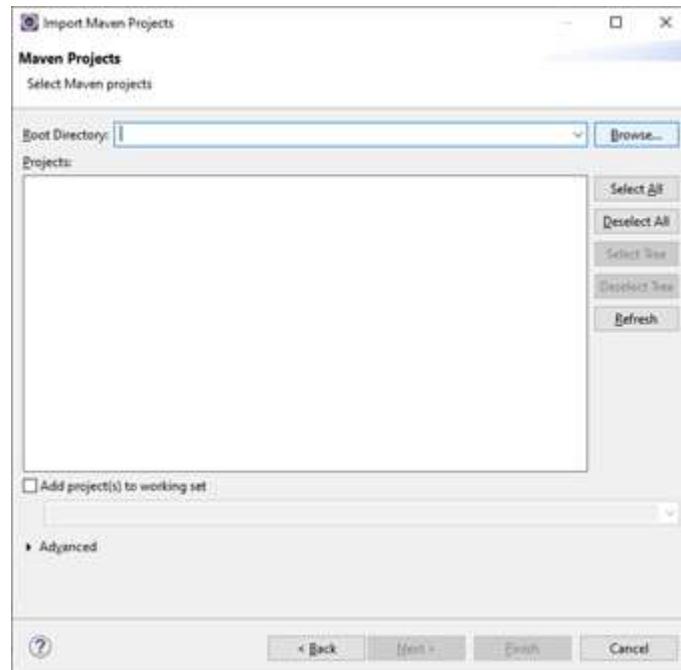
No campo texto digite, "Maven", sem aspas, selecione a opção *Existing Maven Projects* e clique no botão *Next*, conforme a figura a seguir.

Figura 12 – Tela de importação do Eclipse para projetos Maven



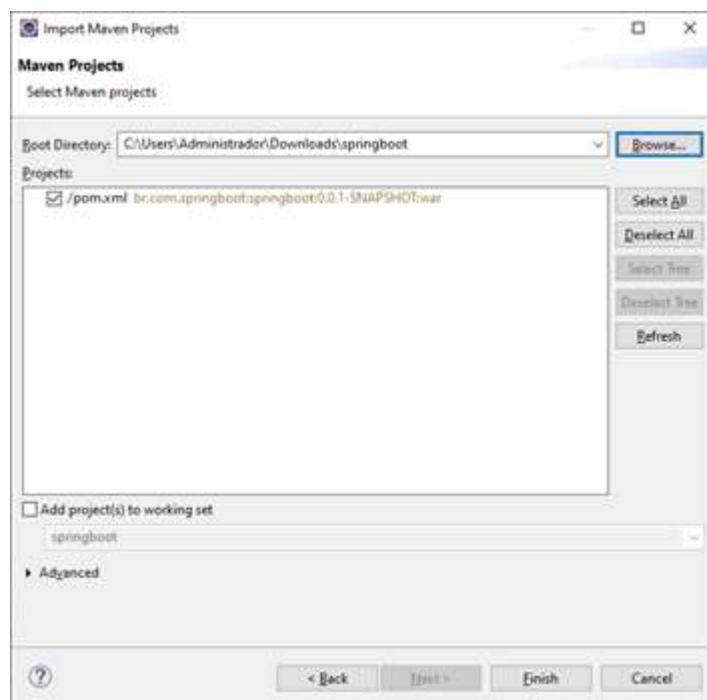
Ao realizar a operação anterior, será exibida a tela de importação de um projeto Maven, conforme podemos observar na Figura 13.

Figura 13 – Importação de um projeto Maven



Nessa tela, deve ser informado o diretório no qual se encontra o arquivo de configuração do projeto, aquele que geramos pelo *Spring Boot*. Utilize o botão *Browse* para realizar tal operação, após selecionado o diretório, clique no botão *Finish*.

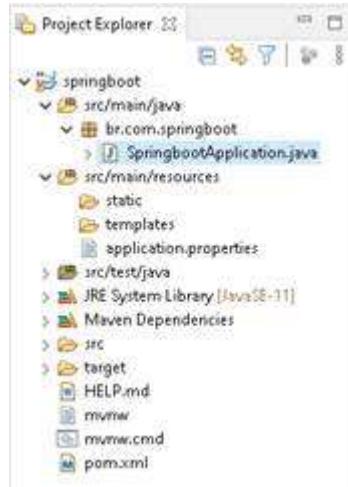
Figura 14 – Importação do projeto *Spring Boot*



Após efetuar a operação citada anteriormente, o Eclipse iniciará o processo de importação do projeto. Realizada a importação do projeto, precisamos compreender como está organizada a

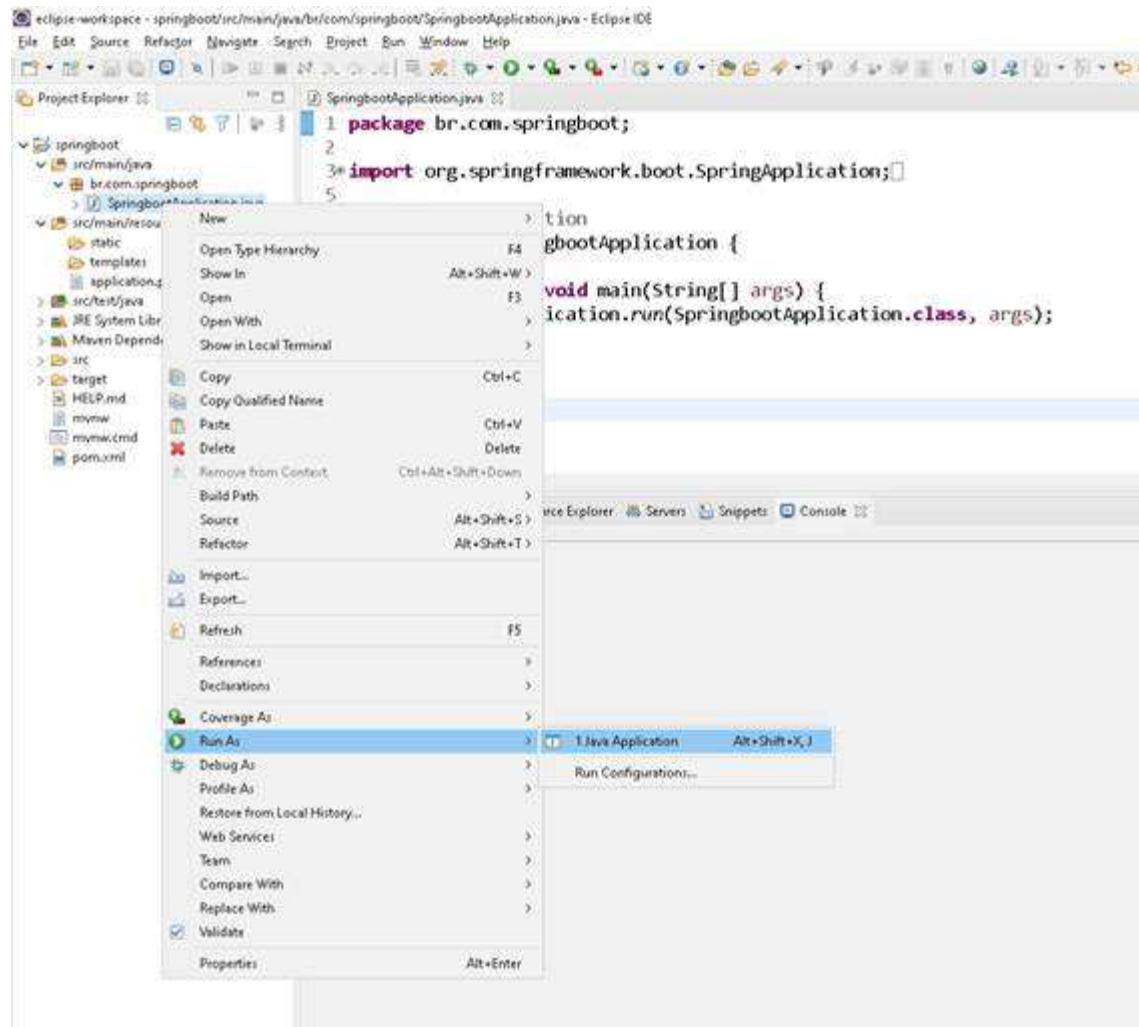
estrutura do projeto. Vamos expandir o projeto clicando duas vezes sobre o nome do projeto na aba *Project Explorer* no canto superior esquerdo, conforme podemos observar na Figura 15.

Figura 15 – Estrutura de um projeto *Spring*



O projeto é composto por diversas pastas e arquivos, localizar a pasta *src/main/java* e vá expandindo os nós até encontrar o arquivo *SpringbootApplication*, classe responsável por iniciar a aplicação. Para verificarmos se o projeto foi importado com sucesso e não contém nenhum erro, vamos executá-lo clicando com o botão direito sobre a classe *SpringbootApplication* e selecionando o menu *Run as > Java Application* conforme a figura a seguir.

Figura 16 – Inicializando a aplicação *Spring*



Como o *Spring Boot* já traz o Tomcat configurado por padrão, esse *Servlet Container* irá iniciar a aplicação, e o *log* referente a esse processo será exibido no console do Eclipse. Caso a inicialização tenha ocorrido com sucesso, no *log*, além de não apresentar qualquer mensagem de erro, informará a porta na qual a aplicação está rodando, indicando que a aplicação já está apta para ser utilizada. Essa informação pode ser encontrada na penúltima linha do *log* do como "Tomcat started on port 8080 (http) with context path".

Figura 17 – Aplicação iniciada com sucesso

```

package br.com.springboot;
import org.springframework.boot.SpringApplication;
public class Springbootapplication {
    public static void main(String[] args) {
        SpringApplication.run(Springbootapplication.class, args);
    }
}

```

The screenshot shows the Eclipse IDE interface with the 'Springbootapplication' class open in the editor. The code is annotated with a decorative illustration of a person sitting at a computer keyboard.

TEMA 5 – CONCEITOS WEB

Neste tema, abordaremos alguns conceitos necessários para que possamos iniciar o processo de codificação de uma aplicação web utilizando o *Spring Framework*. Para isso, é necessário compreender dois conceitos fundamentais, como funciona o protocolo HTTP e o arquitetura MVC.

5.1 PROTOCOLO HTTP

Permite a troca de mensagens entre máquinas distintas por meio de hipertexto, sendo HTTP um acrônimo para *HyperText Transfer Protocol*, ou seja, Protocolo de Transferência de Hipertexto. Esse protocolo atua dentro da camada de aplicação do modelo OSI e é por meio dele que conseguimos navegar pela *web*, acessando os mais variados *sites* disponíveis na rede de internet.

Quando acessamos um determinado site, por exemplo o Google, abrimos o navegador e digitamos o endereço <http://www.google.com.br>. Esse endereço é o que chamamos de URL (*Uniform Resource Locator*) do site. O navegador, por sua vez, comprehende esse conteúdo e faz uma requisição para o servidor <www.google.com.br>, responsável por atender as requisições pertinentes a essa URL. A URL representa um recurso com o qual deseja-se acessar, podendo ser uma imagem, arquivo, vídeo ou página *web*.

Agora que sabemos o que é uma URL, precisamos entender como ela é composta e, para isso, vamos adotar o seguinte endereço: <http://www.gov.br/mec/pt-br>. Essa URL é composta por três elementos.

- **Protocolo:** o prefixo anterior aos "://" da URL identifica o protocolo de comunicação que será adotado para troca de dados entre as máquinas. Nesse caso, estamos utilizando o protocolo HTTP, mas poderíamos utilizar outros protocolos como FTP, TCP etc.
- **Host:** identifica o servidor responsável por fornecer os recursos de um domínio, na URL o *host* está compreendido entre os "://" e a primeira "/" após os "://", sendo o *host* da URL <www.gov.br>. O navegador para acessar o recurso dessa URL irá converter o seu *host* em um endereço de IP por meio do DNS *Lookup*, a fim de identificar o servidor responsável pelo processamento da requisição efetuada pelo usuário.
- **URL Path:** identifica o recurso que será acessado pelo usuário, na URL é identificado logo após o *host*, no endereço adotado como exemplo, a URL Path é /mec/pt-br. Nesse caso, o recurso a ser acessado é uma página *web*, mas poderia ser uma imagem caso a URL Path fosse </images/ico-mec.png/>.

5.1.1 Métodos

Os recursos gerenciados pela aplicação podem ser manipulados de diversas formas por meio de uma URL. Quando uma requisição é efetuada pelo cliente, além da URL, precisamos informar também o método de manipulação do recurso. Na tabela a seguir, são elencados os principais métodos do protocolo HTTP e suas respectivas funcionalidades.

Tabela 1 – Métodos do protocolo HTTP

Método	Funcionalidade
GET	Consultar os dados de um recurso.
POST	Criar um recurso.
PUT	Atualizar os dados de um recurso.
PATCH	Atualizar parcialmente um recurso.
DELETE	Remover os dados de um recurso.
HEAD	Consultar os cabeçalhos da resposta.
OPTIONS	Identificar quais manipulações podem ser efetuadas em um recurso.

Para realizar o CRUD (*Create, Read, Update and Delete*) de uma determinada entidade, ou seja, implementar as operações de inserção, consulta, atualização e remoção de uma entidade no banco de dados, devemos criar as URLs pertinentes a cada uma dessas operações e associá-las a um determinado método. Na tabela a seguir, listamos as URL Path e seus respectivos métodos para exemplificar o *CRUD* dos usuários conforme o padrão *REST*.

Tabela 2 – Modelo *REST*

Método	URL Path	Operação
GET	/usuarios	Retorna todos os usuários do sistema.
GET	/usuarios/id	Retorna os dados do usuário com base no seu id.
POST	/usuarios	Cadastra um usuário.
PUT	/usuarios/id	Atualiza os dados de um usuário com base no seu id.
DELETE	/usuarios/id	Remove um usuário com base no seu id.

Note que, embora tenhamos URL *Paths* repetidas, o que define a operação a ser realizada é o método informado na requisição HTTP.

5.1.2 Porta

Para que a aplicação saiba quais são as requisições que ela deve processar, é necessário que na URL seja informado o número da porta. O Tomcat por padrão atende as requisições da aplicação na porta 8080, portanto, para consultar todos os usuários cadastrados no sistema, utilizariammos a requisição GET 8080/usuarios.

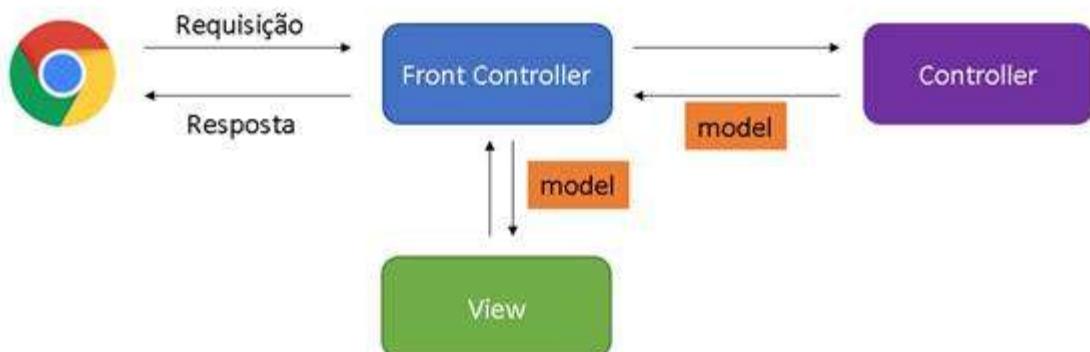
5.1.3 Query String

Permite-nos passar informações adicionais na URL por meio do operador "?" para que a aplicação possa realizar um processamento mais apurado de uma determinada requisição. Supondo que desejamos saber todos os usuários ativos no sistema, poderíamos montar a requisição para GET 8080/usuarios?ativo=true.

5.2 ARQUITETURA MVC

Antes de iniciarmos o processo de codificação da nossa aplicação, precisamos compreender a arquitetura de uma aplicação *web*. Um dos padrões de arquitetura mais utilizados para o desenvolvimento de aplicações *web* é o padrão arquitetural MVC, que é um acrônimo para o termo em inglês *Model* (Modelo), *View* (Visão) e *Controller* (Controlador).

Figura 18 – Padrão MVC do *Spring*



Na figura anterior, temos uma visão do padrão MVC implementado pelo *Spring*, exibindo os três componentes que compõem esse modelo arquitetural (*Model*, *View* e *Controller*) com a adição do *Front Controller*. Por meio de um navegador, será exibida a interface gráfica da aplicação, na qual o usuário realizará uma determinada tarefa. Ao realizar tal tarefa, será enviada ao servidor da aplicação uma requisição HTTP para que esta possa ser processada.

Essa requisição será recebida pelo *Front Controller*, componente responsável por receber as requisições dos clientes e encaminhá-las para os *Controllers* pertinentes, que, por sua vez, realizarão o processamento da requisição efetuada pelo cliente. Caso esse processamento exija algum tipo de interação com o banco de dados, o *Controller* irá realizar essa operação e retornar os dados para o *Front Controller* por meio de um *Model*, que irá conter todas as informações pertinentes à requisição solicitada.

Ao receber o *Model*, o *Front Controller* irá encaminhá-lo a *View*, responsável por realizar o processamento da página *web*, uma vez que, para o seu desenvolvimento, utilizamos componentes (JSP, JSF, entre outros). Portanto, a *View* realizará esse processamento, adicionando os dados providos pelo *Model* ao componente utilizado para o desenvolvimento da página *web*, devolvendo para o *Front Controller* somente o código HTML da página *web* processada.

Assim que a *View* retornar a página *web* para o *Front Controller*, este irá retornar para o navegador uma resposta HTTP que fará o seu processamento, neste caso, exibindo a página *web* com os dados pertinentes à requisição efetuada anteriormente. A partir disso, inicia-se um novo ciclo de requisições por parte do usuário e, para cada nova solicitação, o mesmo fluxo de trabalho será efetuado para resolução das requisições.

FINALIZANDO

Nesta aula, aprendemos a configurar o nosso ambiente de desenvolvimento, instalando as ferramentas necessárias para a construção de uma aplicação utilizando o *Spring Framework*. Após configurado o ambiente de desenvolvimento, por meio do *Spring Boot*, criamos o um projeto *Spring* de forma fácil e rápida sem a necessidade de configurarmos tudo de forma manual.

Além disso, também vimos como importar esse projeto dentro do Eclipse e os conceitos pertinentes para o desenvolvimento de uma aplicação *web*, de como montar uma API REST de requisições HTTP e como é o fluxo de processamento dessa requisição dentro da aplicação *web* baseado no padrão MVC.

REFERÊNCIAS

APACHE. **What is a Maven?**: Maven – Introduction. Disponível em: <apache.org>. Acesso em: 19 dez. 2021.

ECLIPSE **About the Eclipse Foundation**. The Eclipse Foundation, 2021.

MySQL **About MySQL**. MySQL, 2021.

ORACLE. **Java Platform Standard Edition 17 Development Kit**: JDK 17. Disponível em: <oracle.com>. Acesso em: 19 dez. 2021.

POSTMAN. **What is Postman?**. Postman API Platform, 2021.

SNYK. **JVM Ecosystem report 2018**: About our Platform and Application. JVM Ecosystem, 2021.

SPRING. **Initializr**. Spring Initializr, 2021.

SPRING. **Why Spring?**. Spring, 2021.

SPRING. **Web MVC Framework**: 17. Web MVC framework. Disponível em: <spring.io>. Acesso em: 19 dez. 2021.