01 Apresentação

Transcrição

Boas-vindas ao curso de **Spring Boot 3: desenvolva uma API Rest em Java!**
Me chamo Rodrigo Ferreira e serei o seu instrutor ao longo deste curso, em que vamos aprender como usar o Spring Boot na versão `3`.
Rodrigo Ferreira é uma pessoa de pele clara, com olhos castanhos e cabelos castanhos e curto. Veste camiseta preta lisa, tem um microfone de lapela na gola da camiseta, e está sentado em uma cadeira preta. Ao fundo, há uma parede lisa com iluminação azul gradiente.

Objetivos

- Desenvolvimento de uma API Rest
- CRUD (Create, Read, Update e Delete)
- Validações
- Paginação e ordenação

O objetivo neste curso é usarmos o Spring Boot para desenvolvermos uma API Rest, com algumas funcionalidades. A ideia é desenvolver um CRUD, sendo as quatro operações fundamentais das aplicações: **cadastro, listagem, atualização e exclusão de informações**.

Isto é, aprenderemos a desenvolver um CRUD de uma API Rest usando o Spring Boot.

Vamos ver também como aplicar validações das informações que chegam na nossa API, usando o *Bean Validation*. Depois, vamos

aprender a utilizar o conceito de paginação e ordenação das informações que a nossa API vai devolver.

Tecnologias

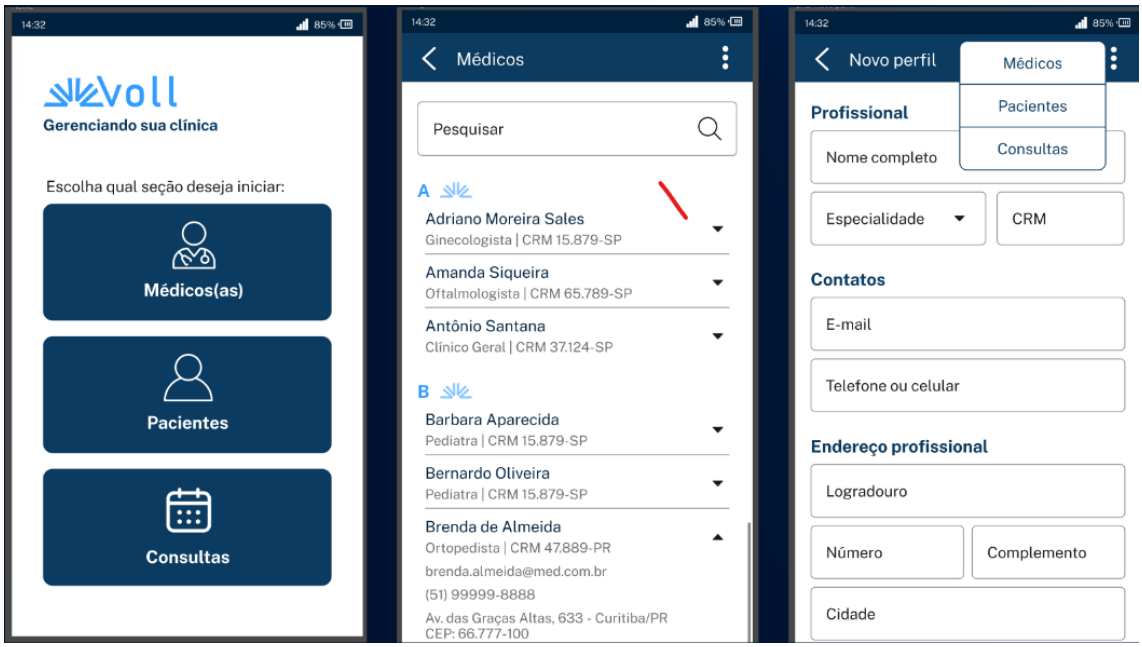
- Spring Boot 3
- Java 17
- Lombok
- MySQL/ Flyway
- JPA/Hibernate
- Maven
- Insomnia

Faremos tudo isso usando algumas tecnologias, como **Spring Boot 3**, sendo a última versão disponibilizada pelo framework. Usaremos, também, o **Java 17** sendo a última versão LTS (*Long-term support*, em português "Suporte de longo prazo") que possui maior tempo de suporte disponível para o Java.
Aprenderemos a usar alguns recursos das últimas versões do Java para deixarmos o nosso código mais simples. Utilizaremos em conjunto com o projeto o **Lombok**, responsável por fazer a geração de códigos repetitivos, como *getters*, *setters*, *toString*, entre outros. Tudo via anotações para o código ficar menos verboso.
Usaremos o banco de dados **MySQL** para armazenar as informações da API e junto com ele utilizaremos a biblioteca **Flyway**. Isso para termos o controle do histórico de evolução do banco de dados, um conceito que chamamos de ***Migration***.
A camada de persistência da nossa aplicação será feita com a **JPA** (*Java Persistence API*), com o **Hibernate** como implementação dessa especificação e usando os módulos do Spring Boot, para tornar esse processo o mais simples possível.

Usaremos o **Maven** para gerenciar as dependências do projeto, e também para gerar o *build* da nossa aplicação. Por último, como focaremos na API Rest (apenas no Back-end), não teremos interface gráfica, como páginas HTML e nem Front-end e aplicativo mobile. Mas para testarmos a API, usaremos o **Insomnia**, sendo uma ferramenta usada para testes em API. Com ela, conseguimos simular a requisição para a API e verificar se as funcionalidades implementadas estão funcionando.

Essas são as tecnologias que usaremos ao longo deste curso.

Qual é o nosso projeto?



Trabalharemos em um projeto de uma clínica médica fictícia. Temos uma empresa chamada **Voll Med**, que possui uma clínica que precisa de um aplicativo para monitorar o cadastro de médicos, pacientes e agendamento de consultas.

Será um aplicativo com algumas opções, em que a pessoa que for usar pode fazer o CRUD, tanto de médicos quanto de pacientes e o agendamento e cancelamento das consultas.

Vamos disponibilizar esse protótipo, mas lembrando que é somente para consultas, para visualizarmos como seria o Front-end. Isso porque o foco deste curso é o Back-end.

A documentação das funcionalidades do projeto ficará em um quadro do Trello com cada uma das funcionalidades. Em cada cartão teremos a descrição de cada funcionalidade, com as regras e validações que vamos implementar ao longo do projeto.

Esse é o nosso objetivo neste curso, aprender a usar o Spring Boot na versão 3 para desenvolvermos o projeto dessa clínica médica, utilizando as tecnologias mencionadas anteriormente.

Vamos lá?

Até a próxima aula!

02 Spring Initializr

Transcrição

O primeiro passo para iniciarmos o nosso projeto é criá-lo, já que neste curso iniciaremos do zero. No caso do Spring Boot, usaremos o Spring Initializr para isso, sendo uma ferramenta disponibilizada pela

equipe do Spring Boot para criarmos o projeto com toda estrutura inicial necessária.
Acessaremos o Spring Initializr pelo site https://start.spring.io/. Nele, será exibido alguns campos para preenchermos sobre o projeto e na parte inferior da tela, temos três botões, sendo o primeiro "Generate" para gerar o projeto.
Como o projeto vai usar o Maven como ferramenta de gestão de dependências e de *build*, deixaremos marcado a opção "Maven Project". Em "Language" deixaremos marcada a opção "Java", que será a linguagem que usaremos.
Na parte "Spring Boot", vamos selecionar a versão do Spring Boot que desejamos gerar o projeto. No momento da gravação deste curso, a mais atual é a versão `2.7.4`, mas temos a versão `3.0.0` que não está liberada ainda, porém, é a que iremos selecionar.

Provavelmente no momento em que estiver assistindo a este curso, essa versão já estará liberada, sem ser a versão beta.

Project

- **Maven Project**: `selecionado`

Language

- **Java**: `selecionado`

Spring Boot

- 3.0.0: `selecionado`

Em "Project Metadata" são solicitadas informações para o Maven configurar o projeto. No campo "Group" colocaremos "med.voll" por ser o nome da empresa, e em "Artifact" e "Name" colocaremos o nome do projeto, "api".

Na descrição, podemos colocar "API Rest da aplicação Voll.med" e em "Package name" (pacote principal da aplicação) ele já pega o *group* e o *artifact*, deixaremos como `med.voll.api`.
No campo "Packaging" é para escolhermos como o projeto será empacotado, que vamos deixar a opção `Jar` selecionada. Usaremos a versão `17` do Java, sendo a última versão *LTS - long term support*, que possui maior tempo de suporte.

Project Metadata

- **Group**: med.voll
- **Artifact**: api
- **Name**: api
- **Description**: API Rest da aplicação Voll.med
- **Package name**: med.voll.api
- **Packaging**: Jar
- **Java**: 17

Agora, à direita da tela temos a seção "Dependencies" e um botão "Add dependencies" (com o atalho "Ctrl + B"). Nela, adicionaremos as dependências do Spring que desejamos incluir no projeto. Para isso, vamos clicar no botão "Add dependencies".

Será aberta uma pop-up com diversas dependências do Spring Boot, e do Spring para selecionarmos. Vamos apertar a tecla "Ctrl" do teclado e clicar em cada uma das dependências que desejamos adicionar, sendo elas:

- Spring Boot DevTools
- Lombok
- Spring Web

O *Spring Boot DevTools* é um módulo do Spring Boot que serve para não precisarmos reiniciar a aplicação a cada alteração feita no código. Isto é, toda vez que salvarmos as modificações feitas no código, ele subirá automaticamente.
Já o Lombok não é do Spring, é uma ferramenta para gerar códigos, como esses códigos verbosos do Java, de `getter` e `setter`, baseado em anotações. Usaremos o Lombok para deixarmos o código mais simples e menos verboso.
A próxima dependência é a Spring Web, dado que vamos trabalhar com uma API Rest e precisamos do módulo web. A princípio deixaremos somente essas três dependências, sem incluir as de banco de dados, de *migration* e de segurança. Mas conforme formos desenvolvendo o projeto, podemos ir adicionando de forma manual.

Após isso, apertaremos a tecla "Esc" para fechar a pop-up. À direita, em "Dependencies", perceba que temos as três listadas.

Depois de preenchermos todas as informações e adicionarmos as dependências, podemos selecionar o botão "Generate" na parte inferior da página.

Dessa forma, vamos gerar o projeto e teremos um arquivo `.zip` com o projeto compactado. Após finalizado o download, clicaremos no arquivo `api.zip` para abrir.
Perceba que ele possui uma pasta chamada `api`, o mesmo nome do projeto que digitamos na tela do Spring Initializr. Clicaremos na pasta `api` e depois no botão "Extract", para extrair. Você pode usar a ferramenta que achar necessária para descompactar o arquivo `.zip`.

Criamos o projeto, baixamos o arquivo zip e o descompactamos. O diretório `api`, é o nosso projeto. Agora, podemos importar na IDE e começar a trabalhar no código.

Na próxima aula, vamos entender como funciona a estrutura de diretórios e como esse projeto já foi criado para nós pelo site do Spring Initializr.

Vamos lá?

03 Para saber mais: Spring e Spring Boot

Spring e Spring Boot não são a mesma coisa com nomes distintos. **Spring** é um **framework** para desenvolvimento de aplicações em Java, criado em meados de 2002 por Rod Johnson, que se tornou bastante popular e adotado ao redor do mundo devido a sua simplicidade e facilidade de integração com outras tecnologias.
O framework foi desenvolvido de maneira **modular**, na qual cada recurso que ele disponibiliza é representado por um módulo, que pode ser adicionado em uma aplicação conforme as necessidades. Com isso, em cada aplicação podemos adicionar apenas os módulos que fizerem sentido, fazendo assim com que ela seja mais leve.
Existem diversos módulos no Spring, cada um com uma finalidade distinta, como por exemplo: o módulo **MVC**, para desenvolvimento de aplicações Web e API's Rest; o módulo **Security**, para lidar com controle de autenticação e autorização da aplicação; e o módulo **Transactions**, para gerenciar o controle transacional.

Entretanto, um dos grandes problemas existentes em aplicações que utilizavam o Spring era a parte de configurações de seus módulos, que era feita toda com arquivos XML, sendo que depois de alguns anos o framework também passou a dar suporte a configurações via classes Java, utilizando, principalmente, anotações. Em ambos os casos, dependendo do tamanho e complexidade da aplicação, e também da quantidade de módulos do Spring utilizados nela, tais configurações eram bastante extensas e de difícil manutenção.

Além disso, iniciar um novo projeto com o Spring era uma tarefa um tanto quanto complicada, devido a necessidade de realizar tais configurações no projeto.

Justamente para resolver tais dificuldades é que foi criado um novo módulo do Spring, chamado de **Boot**, em meados de 2014, com o propósito de agilizar a criação de um projeto que utilize o Spring como framework, bem como simplificar as configurações de seus módulos.

O lançamento do Spring Boot foi um marco para o desenvolvimento de aplicações Java, pois tornou tal tarefa mais simples e ágil de se realizar, facilitando bastante a vida das pessoas que utilizam a linguagem Java para desenvolver suas aplicações.

Ao longo do curso aprenderemos como desenvolver uma aplicação utilizando o Spring Boot, em conjunto com diversos outros módulos do Spring, de maneira simples e produtiva.

04 Para saber mais: Novidades Spring Boot 3

A versão 3 do Spring Boot foi lançada em novembro de 2022, trazendo algumas novidades em relação à versão anterior. Dentre as principais novidades, se destacam:

- Suporte ao Java 17
- Migração das especificações do Java EE para o Jakarta EE
- Suporte a imagens nativas

Você pode ver a lista completa de novidades da versão 3 do Spring Boot no site: Spring Boot 3.0 Release Notes

Atenção! Este curso não terá como foco principal explorar as novidades e recursos da versão 3 do Spring Boot, mas sim o desenvolvimento de uma API Rest utilizando o Spring Boot como framework, sendo que algumas novidades da versão 3 serão utilizadas apenas quando fizerem sentido no projeto.

05 Estrutura do projeto

Transcrição

Após descompactarmos o projeto, no Desktop teremos uma pasta chamada `api`, sendo a pasta do nosso projeto e podemos importá-la na IDE.

Neste curso usaremos o Intellij, o ideal é você também usar essa IDE para não termos nenhum problema de configuração ao longo do caminho.

Com o Intellij aberto, na página inicial temos a mensagem "*Welcome to intellij IDEA*", abaixo três botões na cor azul, sendo eles: *New Project*, *Open* e *Get from VCS*. Clicaremos no segundo botão "Open", para abrirmos o projeto.
Será exibida um pop-up com o título "*Open File or Project*" (em português, "Abrir arquivo ou projeto"), em que vamos até o local que descompactamos o projeto, "Desktop > api". Após selecionar a pasta `api`, basta clicar no botão "Ok", na parte inferior direita.

Com isso, o projeto é importado no Intellij. E a primeira coisa que precisamos fazer ao importar um projeto usando o Maven é verificar se ele baixou as dependências corretamente.

Para fazer essa verificação, na lateral direita do Intellij, escrito da vertical, temos a opção "Maven". Clicaremos nela, será mostrado o projeto `api` com uma seta do lado esquerdo para expandir, vamos selecioná-la.
Temos a pasta `Lifecycle`, mas percebemos que ele ainda está realizando as configurações. Na parte inferior esquerda temos a mensagem: "*Resolving dependencies of api*". Isto é, ele está baixando as dependências do projeto para o computador.
Após aguardar um pouco, no painel do Maven (no canto direito), temos as pastas: `Lifecycle`, `Plugins` e `Dependencies`. Clicaremos na seta à esquerda da pasta `Dependencies`, para expandir.
Perceba que são as dependências que instalamos anteriormente, a do Web, DevTools e Lombox, e também, foi baixado uma `starter-test`.

Esta dependência é o Spring que instala de forma automática, usada para testes automatizados.
Caso uma das dependências não aparece, podemos selecionar o botão "*Reload All Maven Projects*" ("Recarregar todos os projetos Maven"), no ícone do canto superior esquerdo do painel do Maven. Assim, o projeto será recarregado e será feita uma nova tentativa para baixar as dependências.

Podemos minimizar o painel do Maven, clicando no ícone "-" na parte superior direita.

No painel à esquerda do Intellij, temos a estrutura de diretórios do projeto. Como foi o Spring Initializr que criou essa estrutura de diretórios e arquivos de uma aplicação com Spring Boot para nós, vamos entendê-la.

É um projeto que usa Maven, logo está seguinte a estrutura de diretórios do Maven. Note que temos a pasta `src`, com os arquivos `main` e `test` dentro. Na pasta `main`, temos o arquivo `resources` e dentro de `test` temos o `java`. E no diretório raiz, temos o projeto `pom.xml`.
Até agora, nada muito diferente do esperado da estrutura de projetos Maven. Vamos clicar em `pom.xml` para visualizarmos o `xml` do Maven para projetos com Spring Boot.

Perceba que as informações que preenchemos no site constam neste arquivo, a partir da linha 11.

```
pom.xml

//código omitido
```

```
<groupId>med.voll</groupId>
<artifactId>api</artifactId>
<version>0.0.1</version>
<name>api</name>
<description>API Rest da aplicação Voll.med</description>
<properties>
    <java.version>17</java.version>
</properties>

//código omitidoCOPIAR CÓDIGO
```

Neste trecho temos o artefato, o nome, a descrição e a versão do Java. Mais para baixo, temos a tag `<dependencies>` com as dependências que incluímos anteriormente também.

Descendo mais o código, temos a tag `<build>` com um *plugin* do Maven para fazer o `build` do projeto. Em `<exclude>` ele aplica uma configuração devido ao Lombok.

```
//código omitido

<exclude>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</exclude>

//código omitidoCOPIAR CÓDIGO
```

Abaixo, há as tags `repositories` e `pluginRepositories` por estarmos usando uma versão não finalizada, que ainda está em beta.

Porém, onde está o Spring Boot? Ele não está declarado como uma dependência neste arquivo. Essa é a primeira diferença em relação à aplicação com Spring tradicional.

O Spring Boot não vem como uma dependência, dentro da tag `dependencies`. Se subirmos o código do arquivo `pom.xml`, temos uma tag chamada `parent`:

```
//código omitido


<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.0.0-M5</version>
    <relativePath/> <!--lookup parent from repository -->
</parent>


//código omitidoCOPIAR CÓDIGO
```

A tag `parent` é como uma herança da orientação a objetos. É como se o `pom.xml` estivesse herdando de outro `pom.xml` e dentro dessa tag vem de onde ele vai herdar - sendo o `pom.xml` do Spring Boot.
O Spring Boot vem dentro da tag `parent`, em que declaramos para o projeto herdar do arquivo `pom.xml` do Spring Boot. Nela, passamos a versão, o *group Id* e o *artifact Id* do Spring Boot. Isso foi feito de forma automática pelo site do Spring Initializr.
As dependências são os módulos do Spring Boot, ou outras bibliotecas e frameworks que desejarmos usar. Note que nas bibliotecas do Spring Boot não especificamos as versões, colocamos somente o `GroupId` e o `ArtifactId`, como em:

```
//código omitido
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web
</dependency>

//código omitidoCOPIAR CÓDIGO
```

Isso acontece porque ele já sabe qual a versão correta de cada dependência, baseada na versão do Spring Boot. Logo, precisamos especificar somente a versão do Spring Boot e não de cada dependência, é uma facilidade que temos.

Assim que funciona o arquivo `pom.xml` no caso do Maven para um projeto usando o Spring Boot.
Vamos fechar o arquivo `pom.xml` e expandir o menu da pasta `Project`, à esquerda do Intellij. Temos a estrutura de diretórios do Maven, mas em "src > main java", perceba que já foi criado o pacote raiz do projeto, o `med.voll.api`.
Dentro da pasta `med.voll.api`, temos uma classe java chamada `ApiApplication`, selecionaremos ela. Por padrão, ele criou essa classe com o nome `Api` (nome do projeto), `Application`.

`ApiApplication`

```
package med.voll.api;

//código omitido

@SpringBootApplication
public class ApiApplication {
```

```
        public static void main(String[] args) {
            SpringApplication.run(ApiApplication.class, args);
        }

}COPIAR CÓDIGO
```

Note que foi gerada uma classe com o método `main`, logo, essa é a classe que rodará o projeto. Isso foi criado de forma automática para nós.

À esquerda, em "src > main > resources" (diretório do Maven que ficam os arquivos e as configurações do projeto), temos três pastas: `static`, `templates` e `application.properties`.

No arquivo `static` é onde ficam as configurações estáticas da aplicação web, como arquivos de `csv`, JavaScript e imagens. Não usaremos essa pasta, dado que não desenvolveremos uma aplicação web tradicional e sim uma API Rest.

Na pasta `templates`, estão os templates HTML, as páginas do projeto. Não usaremos essa pasta, também, porque essas páginas não ficarão na API Back-end e sim em outra aplicação Front-end.

Por último, na pasta `resources`, temos o arquivo `application.properties`. Clicando nele, note que ele está vazio. Esse é a pasta de configurações do projeto com Spring Boot, usaremos bastante esse arquivo.

Além disso, em "src > test > java", foi criado um pacote com uma classe chamada `ApiApplicationTests`, de exemplos com testes automatizados. Clicando nela, perceba que é um teste que está vazio.

Posteriormente, vamos aprender mais sobre essa parte de testes automatizados e entenderemos como realizá-los em um projeto com Spring Boot.

Essa é a estrutura de diretórios de um projeto com Spring Boot. No caso, estamos usando o Maven e, por isso, a estrutura de diretórios está estruturada dessa forma.

Criando o projeto no site do Spring Boot, o arquivo `pom.xml` é configurado corretamente, com as dependências que escolhemos. Gera as configurações do projeto, como vimos em `resources` e já cria a classe `main ApiApplication`, com a estrutura inicial para rodarmos a nossa aplicação.

O objetivo deste vídeo era importarmos o projeto na IDE e explorar os diretórios e arquivos criados. Agora, podemos rodar esse projeto e inicializá-lo fazendo um "Hello, world".

Veremos como fazer isso no próximo vídeo. Até lá!

06 Hello World

Transcrição

O projeto está importado na IDE e conhecemos um pouco a estrutura dos diretórios e arquivos. Nesta aula, executaremos a aplicação e vamos exibir o "Hello World".
Como mencionado, para executarmos esse projeto com Spring Boot, precisamos rodar a classe `ApiApplication` que possui o método `main`. E essa é uma diferença entre aplicações web tradicionais e usando o Spring Boot.

`ApiApplication`

```
package med.voll.api;
```

```
//código omitido

@SpringBootApplication
public class ApiApplication {

        public static void main(String[] args) {
            SpringApplication.run(ApiApplication.class, args);
        }

} COPIAR CÓDIGO
```

Como funcionava para rodarmos uma aplicação web? Nós adicionávamos um servidor de aplicações, como *TomCat*, *Jetty*, *Glassfish* e *Weblogic*, e colocávamos o projeto dentro do servidor e o inicializavámos. Dessa forma, ele fazia toda configuração e disponibilizava a aplicação.

No Spring Boot, o processo foi invertido. Ao invés de termos um servidor e colocarmos dentro dele a aplicação, é ao contrário: dentro da aplicação é que vai o servidor de aplicação.

Por padrão, o projeto vem com o TomCat como servidor de aplicação e ele já está embutido dentro das dependências do módulo web. Ele não aparece no arquivo `pom.xml` porque está no `xml` do Spring Boot herdado.

Mas já temos um TomCat embutido no projeto, logo não precisamos configurar e inicializar servidor como fazíamos em aplicações web tradicionais, mesmo utilizando o Spring sem o Boot.

Assim, o servidor está embutido no Boot e para rodá-lo entra a classe com o método `main`. Vamos analisar a classe `ApiApplication`:

```
package med.voll.api;


//código omitido


@SpringBootApplication
public class ApiApplication {


        public static void main(String[] args) {
            SpringApplication.run(ApiApplication.class, args);
        }


} COPIAR CÓDIGO
```

É uma classe com uma anotação `@SpringBootApplication`, com um método `main`. Este chama um método estático denominado `run` de uma classe do Spring nomeada `SpringApplication`.
Portanto, para rodar o projeto, basta rodar essa classe com o método `main`, que estamos chamando essa classe do Spring `SpringApplication` com o método `run`. É este método que inicializa o projeto.

Com isso, fica mais simples executarmos a aplicação. Não temos mais as dependências e não precisamos mais configurar os servidores.

Para rodar a aplicação, clicaremos com o botão direito do mouse nela e escolheremos a opção "Run 'ApiApplication.main()'" ou podemos usar o atalho "Ctrl + Shift + F10".

Após rodar, no canto inferior direito da tela é gerado um alerta "*Enable annotation processing*" ("Ativar processamento de anotação"), porque estamos usando o Lombok no projeto, e precisamos habilitar o *annotation processing*. Para habilitar, basta selecionar "Enable annotation processing".
Perceba que ao rodarmos a classe `main`, na aba "Run", foi gerado alguns logs com informações do Spring Boot. Vamos analisar, note que ele exibiu uma *splash* do Spring e a versão do projeto: `v3.0.0-M5`, depois gerou vários logs.

Rolando a barra inferior do terminal para a direita, temos os logs:

Não foram exibidos para os logs

```
"Starting ApiApplication using Java 17.0.4 on alura with PDI 399646"


"No active profile set, falling back to 1 default profile: "default"


DevTools property defaults active! Set 'spring.devtools.add-
properties' to 'false' to disable


Tomcat initialized with port(s): 8080 (http)COPIAR CÓDIGO
```

Ele está informando que a aplicação está sendo inicializada usando Java 17, não há *profile* configurado (aprenderemos sobre profiles mais para frente) e um Tomcat embutido que roda na porta 8080.

No final, ele mostra:

```
Started ApiApplication in 1.388 seconds (process running for 1.709)

COPIAR CÓDIGO
```

Foi bem rápido para ele inicializar o projeto, dado que ele está vazio. Assim, o nosso servidor já está rodando!

Ao rodarmos a classe com o método `main`, o servidor é inicializado e gerado alguns logs. Se tivéssemos algum problema, seríamos interrompidos, ou seja, o servidor seria parado e apareceria uma mensagem de erro.

Deu tudo certo, o projeto está sendo rodado na porta 8080, por padrão. Vamos acessar essa aplicação, para tentarmos disparar uma requisição para a nossa API.

Para isso, abriremos o navegador e acessaremos o seguinte endereço:

```
localhost:8080 COPIAR CÓDIGO
```

Ao apertarmos a tecla "Enter", é exibida a mensagem de erro do Spring: "*Whitelabel Error Page*". Informando que a aplicação foi carregada e que foi recebida a requisição, mas não há nenhum controller, nem endereço mapeado. Assim, retornou o erro 404.

```
Whitelabel Error Page


This application has no explicit mapping for /error, so you are seeing
this as a fallback.


Wed Oct 12 15:03:43 BRT 2022
There was an unexpected error (type=Not Found, status=404).
No message available COPIAR CÓDIGO
```

Essa mensagem era esperada, isso significa que funcionou. Isso porque não mapeamos nenhuma URL no projeto. Agora criaremos o *controller* e fazer o Hello World.

Voltaremos para o Intellij, e no painel à esquerda criaremos uma classe. Em "src > main > java" com a pasta `med.voll.api` selecionada, usaremos o atalho "Alt + Insert".

Será exibido um menu e nele clicaremos a opção "Java Class". No pop-up seguinte, com o título "New Java Class", digitaremos "HelloController", será o nome da nossa classe.

HelloController.java

```
package med.voll.api;


public class HelloController {


} COPIAR CÓDIGO
```

Para não ficarmos com as classes soltas no pacote api, na primeira linha que está sendo definido o pacote, podemos inserir `.controller`. Assim, será gerado um subpacote *controller* em que ficará as classes controller.

```
package med.voll.api.controller;COPIAR CÓDIGO
```

Perceba que apareceu um sublinhado na cor vermelha abaixo, isso significa que ele não encontrou esse pacote. Passando o mouse por cima, clicaremos em "*more actions*", será mostrada duas ações:

- Move to package 'med.voll.api.controller'
- Set package name to 'med.voll.api'

Selecionaremos a primeira opção, `Move to package 'med.voll.api.controller'`, para movermos para o pacote mencionado. No pop-up exibido com o título "Choose Destination Directory" ("Escolha o diretório de destino"), vamos

selecionar `/src/main/java/med/voll/api/controller` e depois clicar no botão "Ok", na parte inferior direita.

Com isso, o pacote controller foi criado e o sublinhado na cor vermelha sumiu.

Na classe `HelloController`, não há nada do Spring Boot e sim o Spring MVC (*Spring Web model-view-controller*). Usaremos funcionalidades do Spring MVC.
Para comunicarmos o Spring MVC que é uma classe controller, acima dela incluiremos a anotação `@Controller`. Mas no caso não estamos trabalhando com aplicação web tradicional, e sim com uma API Rest. Por isso, a anotação será `@RestController`.
Outra anotação que vamos incluir é a `@RequestMapping`. Isso para informarmos qual a URL que esse controller vai responder, que será /hello. Assim, ao chegar uma requisição
para `localhost:8080/hello` vai cair neste controller.

```
package med.voll.api;


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

import org.springframework.web.bind.annotation.RequestMapping;


@RestController

@RequestMapping("/hello")

public class HelloController {


} COPIAR CÓDIGO
```

No controller, é necessário chamarmos algum método. Criaremos o método `public string olaMundo(){}`. Dentro dele, vamos retorna a string `"Hello World!"`.

```
package med.voll.api;


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


@RestController
@RequestMapping("/hello")
public class HelloController {


        public String olaMundo(){
            return "Hello World!";


        }
} COPIAR CÓDIGO
```

Agora, precisamos colocar uma anotação, sendo que o método do protocolo HTTP é para chamar este método.

```
package med.voll.api;


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


@RestController
@RequestMapping("/hello")
public class HelloController {



        @GetMapping
```

```
        public String olaMundo(){
            return "Hello World!";

        }
} COPIAR CÓDIGO
```

Estamos informando para o Spring que essa classe é um controller, com o `RequestMapping("/hello")`. Logo, se chegou uma requisição `/hello` e ela é do tipo `get`, será chamado o método `olaMundo()`.

Podemos salvar essas alterações e voltar para o navegador para acessar o endereço:

```
localhost:8080/hello COPIAR CÓDIGO
```

Note que seguimos com o erro 404.

```
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing
this as a fallback.

Wed Oct 12 15:03:43 BRT 2022
There was an unexpected error (type=Not Found, status=404).
No message available COPIAR CÓDIGO
```

Vamos verificar no IntelliJ o motivo se tornar *not found*, está tudo certo, temos um `@RequestMapping("/hello")`. No caso, não funcionou porque não reiniciamos o projeto, ou seja, alteramos o código, mas não reiniciamos.

Mas não adicionamos o módulo do DevTools, usado justamente para reiniciar de forma automática? Sim, porém, no IntelliJ precisamos configurar o projeto para o DevTools funcionar.

Para incluir essa configuração, usaremos o atalho "Shift + Shift". Será exibida uma tela, que na parte superior temos um ícone de lupa com um campo de busca que digitaremos "settings". Nesta opção informa que podemos usar o atalho "Ctrl + Alt + S" para abrir a tela de configurações.

Na tela aberta, no menu do lado esquerdo, clicaremos na seta da opção "Build, Execution, Deployment" para expandir. Nela, clicaremos na seção "Compiler", e à direita do menu, temos uma *checkbox* ("caixa de seleção") chamada "Build project automatically" ("Construir projeto automaticamente").

Marcaremos a opção "Build project automatically" e depois vamos clicar no botão "Apply", no canto inferior direito da tela. Agora, voltando para o menu do lado esquerdo, clicaremos na opção "Advanced Settings" ("Configurações avançadas").

Na página de configurações avançadas, marcaremos a opção "*Allow auto-make to start even if developed application is currently running*" ("Permitir que a criação automática inicie mesmo se o aplicativo desenvolvido estiver em execução no momento"), dentro da seção "Compiler".

Logo após, podemos clicar no botão "Apply" e depois em "Ok", no canto inferior direito.

No IntelliJ vamos parar o projeto e reiniciar só para garantirmos o pleno funcionamento. Para parar o projeto, clicaremos no botão com o ícone de um quadrado vermelho "■" no canto superior direito. Após isso, vamos selecionar o ícone de play na cor verde "▶", para rodar a aplicação.

Podemos voltar ao navegador e atualizar a página com o endereço `localhost:8080/hello`, clicando no botão "⟳" na parte superior esquerda. Perceba que a mensagem "Hello World!" é exibida na tela.

Será que o DevTools está funcionando? Vamos testar!

Para isso, voltaremos ao IntelliJ e alterar a string que estamos devolvendo no controller.

```
return "Hello World Spring!"; COPIAR CÓDIGO
```

Salvaremos clicando em "Ctrl + S". Com isso, o DevTool deve detectar essa alteração e reiniciar o projeto. Perceba que no terminal os logs estão sendo gerados novamente e a aplicação está sendo reinicializada.

Vamos ao navegador novamente e atualizar a página. Desta vez a mensagem exibida é a "Hello World Spring!". Assim, sabemos que o DevTools está funcionando conforme o esperado.

Aprendemos que há a classe `ApiApplication` com o método `main` e ela que rodamos para inicializar o projeto, que vai carregar o Tomcat e configurar o projeto.

E criamos um *controller*, disparamos uma requisição direto do navegador para testarmos se está tudo funcionando. Essa parte do controller é um Spring MVC, não há nada do Spring Boot, isso já funcionava antes do Boot.

Exibimos o Hello World e o projeto está configurando. Agora, podemos começar a implementar as funcionalidades na aplicação, em que teremos o CRUD dos médicos e pacientes, com agendamentos e cancelamentos das consultas.

Até a próxima aula!

07 Mapeando requisições

Você está trabalhando no desenvolvimento de uma API Rest para uma aplicação de um e-commerce, utilizando o Spring Boot, e cria uma classe `Controller` da seguinte maneira:

```
@RequestMapping("/produtos")
public class ProdutoController {


    @GetMapping
    public String produtosEmEstoque() {
        return "Produtos em estoque...";
    }
} COPIAR CÓDIGO
```

Mas ao executar a aplicação e entrar no endereço `http://localhost:8080/produtos`, você recebe um erro *404 - Not Found*. Por que esse erro ocorreu?

Porque faltou colocar @RestController antes do @RequestMapping para informar que é um controlador e que também é uma api de web service.

DISCUTIR NO FORUM

Você acertou em cheio!

Marcos, sua resposta está correta, parabéns! De fato, para que o Spring Boot reconheça a classe como um controlador REST e possa mapear os endpoints corretamente, é necessário utilizar a anotação @RestController. Essa anotação é essencial para que o Spring saiba que deve tratar a classe como um controlador e que os métodos devem responder a requisições HTTP. Continue assim, prestando atenção aos detalhes das anotações e ao funcionamento do Spring Boot!

08 Faça como eu fiz: criação do projeto

Agora é com você! Faça o mesmo procedimento que eu fiz na aula. Crie o projeto inicial pelo site Spring Initializr, faça o import dele no IntelliJ e por fim crie uma classe Controller como *Hello World*.

Opinião do instrutor

-
-

A classe Controller de exemplo deve ter a seguinte estrutura:

```
@RestController

@RequestMapping("/hello")

public class HelloController {



    @GetMapping

    public String olaMundo() {

        return "Hello World Spring!";

    }

} COPIAR CÓDIGO
```

09 O que aprendemos?

Nessa aula, você aprendeu como:

- Criar um projeto Spring Boot utilizando o site do Spring Initializr;
- Importar o projeto no IntelliJ e executar uma aplicação Spring Boot pela classe contendo o método `main`;
- Criar uma classe Controller e mapear uma URL nela utilizando as anotações `@RestController` e `@RequestMapping`;
- Realizar uma requisição de teste no browser acessando a URL mapeada no Controller.