02

Hello World

Transcrição

[00:00] Vamos começar então nosso treinamento de construção de APIs REST usando Spring Boot. A primeira coisa que precisamos fazer é criar nosso projeto. Tendo o Eclipse aberto, para criar o projeto (Como se trata de um projeto utilizando o Maven) poderíamos ir ao menu "File" e selecionaríamos "new", depois "Maven Project" e abriríamos uma tela branca para digitar as informações do Maven. Depois teríamos que abrir o arquivo `.xml` e declarar as dependências do Spring Boot. Faríamos isso tudo manualmente.

[00:37] Como é um pouco chato e é sempre igual, o próprio pessoal do Spring criou um site que é um gerador de projetos com Spring Boot, chamado Spring initializr. No caso, vamos utilizar esse site, porque é muito mais simples, rápido e prático.

[01:00] Acessaremos o site pela URL https://start.spring.io/ (https://start.spring.io/). Na tela inicial do Spring Initializr, você vai precisar passar algumas informações sobre o seu projeto. Por exemplo, ele te pergunta, no campo "*Project*", se você quer criar um projeto usando Maven ou Gradle. Algumas pessoas preferem o Gradle. No campo "*Language*", pergunta qual a linguagem de programação. O Spring Boot suporta as linguagens Java, Katlin ou Groovy.No campo "*Spring Boot*", pergunta a versão do Spring Boot e outras informações do projeto.

[01:27] Na data de criação do treinamento, a versão estável é a 2.1.4. No curso, é essa que vamos utilizar. O recomendado é que vocês utilizem o download do projeto que vamos disponibilizar e que já vem com essa versão, para garantir que não haverá nenhum problema.

[01:50] Ainda na página do Spring Initializr, mais abaixo, temos que preencher o campo "*Project Metadata*" com as informações a respeito do projeto que o Maven precisa: primeiro, quem é o group id; segundo, o artifact id. No nosso caso, o primeiro vai ser "br.com.alura" e o segundo será "forum". Descendo um pouco mais, encontraremos o botão "*More Options*", "Opções a mais", para descrição e pacote, mas, a princípio, vamos deixar as opções padrão. Por fim, temos o campo "*Dependencies*" para adicionar as dependências do Spring Boot que você quer utilizar no seu projeto. Para fins didáticos, vamos deixar, por enquanto, só o módulo web. Mas, no mundo real, você vai usar o módulo de segurança (o da JPA, por exemplo) e já vai adicionando ele na hora de gerar seu projeto. Aqui no exemplo, vamos usar apenas o módulo Web.

[02:45] No fim da página, temos o botão verde "*Generate Project*" ou "Gerar o projeto". Quando você pressionar esse botão, ele vai fazer o download de um arquivo zip (no nosso exemplo, ele se chama "forum" que é o nome do projeto. Tudo fica: `forum.zip`). Abrindo esse arquivo (e, dentro dele, a pasta "forum"), perceberemos que esse zip nada mais é do que um projeto com a estrutura do Maven. A ideia é descompactar ele em algum diretório. Para isso, selecionaremos a pasta "forum". Em seguida, na parte de cima da página, canto esquerdo, pressionaremos "Extract To" e ele mostrará a tela "Extraction path and options", onde escolheremos um diretório do computador. Tendo terminado, voltaremos ao Eclipse (nossa IDE) e pressionaremos "File > Import". Na tela do Import, selecionaremos a subopção “existing maven projects” e apertaremos "Next". Na próxima tela, no campo browser, escolheremos a pasta onde foi feita o download, selecionando "Downloads > forum". Em seguida, pressionaremos "Selecionar pasta", depois, "finish", e ele vai importar o projeto no Eclipse.

[03:30] Na primeira vez que você fizer isso, ele vai baixar as dependências do projeto do Spring Boot a partir da internet. Dependendo da sua conexão pode demorar. Assim que ele importou, você vai observar que nada mais é do que um projeto na estrutura do Maven. Tem os diretórios "src/main/java", "src/main/resources" e "src/test/java", tudo certo.

[04:00] Vamos abrir o `pom.xml` (que corresponde ao `forum/pom.xml`) um arquivo importante do nosso projeto. Ele tem as informações que digitamos no site.

```
<groupId>br.com.alura</groupId>
    <artifactId>forum</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

COPIAR CÓDIGO

Repare que temos também uma tag (que é a primeira tag que ele coloca), a tag `<parent>`.

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
```

COPIAR CÓDIGO

ELe usa essa tag para "herdar" do `pom.xml` do Spring Boot. Ou seja, esse é um jeito de dizer que você quer usar o Spring Boot no projeto. Você "herda", ou, declara na tag `<parent>` o `pom.xml` do Spring Boot. Se não estivéssemos usando o Srpring Initializr, teríamos que digitar manualmente.

[04:30] Depois tem a versão do Java, que por padrão é o 1.8.

```
<properties>
    <java.version>1.8</java.version>
</properties>
```

COPIAR CÓDIGO

E depois as dependências que adicionamos no último campo.

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

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

COPIAR CÓDIGO

No caso, eu só escolhi a dependência web e veio uma de testes automatizados por padrão. Isso acontece porque o Spring considera que você vai fazer testes automatizados no seu projeto.

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

```
            <scope>test</scope>
        </dependency>
    </dependencies>
```

COPIAR CÓDIGO

Por último, o plugin para fazer o build do projeto.

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

COPIAR CÓDIGO

[04:55] O `pom.xml` é basicamente isso. O que vamos mexer no arquivo futuramente é: ir à tag `<dependency>` e adicionar as outras dependências que utilizaremos posteriormente.

[05:05] Além disso, se pressionarmos o "src/main/Java", perceberemos que ele criou um pacote "br.com.alura.forum", de acordo com a estrutura que nós definimos, e a classe chamada `fórumApplication.java` , que é o nome do projeto.

```
package br.com.alura.forum;
```

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class ForumApplicationTests {

    @Test
    public void contextLoads() {
    }

}
```

COPIAR CÓDIGO

Se trata de uma classe que tem um método `main()` e chama o método `run()` estático da classe `SpringApplication`. Essa classe tem a anotação `@springbootapplication`. Essa é a classe que vamos utilizar para rodar nosso projeto. Nele, nós não adicionamos o servidor separado (Tomcat) e incluímos o projeto nele. Para rodar o projeto, rodamos a classe `main` e ele tem um Tomcat embutido que roda para subir nossa aplicação.

[05:45] Vou rodar para fazer uma demonstração. Você vê que ele vai imprimir algumas coisas e gerar o log:

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

COPIAR CÓDIGO

Então, ele tem um Tomcat embutido e, por padrão, roda na porta 8080. Isso é configurável também.

[06:03] Vamos fazer o teste para ver se está funcionando. Vou acessar a URL http://localhost:8080/ (http://localhost:8080/). Deu uma tela de erro, que é a tela de erro do Spring Boot. Deu erro justamente porque o endereço não está mapeado. Mas se apareceu essa tela, significa que o Spring Boot está funcionando.

[06:27] Só para garantir que está tudo ok, vamos fazer nosso famoso Hello World. Vou criar um controller, mapear para o endereço barra, e ver se o Spring vai chamar o endereço.

[06:38] Para criar um controller, vou criar uma nova classe pressionando "br.com.alura.forum" e selecionando o comando "Ctrl + N". Na próxima tela, vou selecionar "Class". Depois, vou trocar o "*Package*", "pacote", para "controller" (Package: br.com.alura.forum.controller). O nome da classe vai ser "HelloController". Em seguida, apertamos "Finish". No Spring MVC, você anota a classe com a `@controller`, para o Spring conseguir encontrar a classe, e ele que vai fazer o gerenciamento da classe. Depois, nós só temos que fazer o import da classe (da anotação que vem do pacote do Spring). E vou criar um método `public`, que devolve uma `string` e que vou chamar o método de `hello()`. Tudo fica: `public string hello() {`. Agora, vou apenas devolver uma `string` para ver se vai funcionar e inserir `"Hello World!";`.

```
package br.com.alura.forum.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
public class HelloController {
```

```
    @RequestMapping("/")
    @ResponseBody
    public String hello() {
        return "Hello World!";
    }

}
```

COPIAR CÓDIGO

[07:24] Tenho que colocar, em cima do método `public string hello() {` , o `@RequestMapping("/")` só para dizer qual é a URL quando o Spring chamar esse método. Note que, no exemplo, eu coloquei uma barra, "/". Se eu chamar a URL http://localhost:8080/users (http://localhost:8080/users), ele tem que cair nesse método e devolver essa string:

"Hello World!"

[07:41] Como eu não quero navegar para uma página ("*JSP*", *Java Server Pages*, ou *Thymeleaf*) tenho que colocar mais uma anotação, que é o `@ResponseBody` . Senão, o Spring vai considerar que o retorno (a string "Hello World") é uma página e ele vai procurar essa página no nosso projeto. Colocando o `@ResponseBody` , ele devolve a string direto para o navegador.

[08:01] Vou salvar. Tenho que parar o projeto, abrir a classe `main` e rodar de novo. Acessando a URL http://localhost:8080/ (http://localhost:8080/) Vou dar um refresh na página. Perceba que apareceu o "Hello World!", então funcionou. Assim, temos nosso projeto criado, nosso "Hello World", tudo funcionando. Podemos começar a desenvolver nossa API de fato.