Spring Boot Lombok

Gustavo García, 22/03/2023.

If in doubt, always refer to the documentation. It takes less than 5 minutes to read the documentation but 5 days to identify and fix a silly bug.

# Lombok: To use or not to use

<https://www.linkedin.com/pulse/lombok-use-omar-ismail/>

Este es un artículo que explica qué es Lombok, para qué sirve y cómo se usa. Vale la pena leerlo.

Dicho brevemente, una ventaja obvia de Lombok es que ayuda a evitar el llamado boiler plate code:

<https://en.wikipedia.org/wiki/Boilerplate_code>

Otra ventaja, no tan obvia pero que a mí me parece más importante, es que en lugar de usar métodos programados específicamente para cada ocasión, se usa código programado por los mejores profesionales, y extensivamente probado. La ventaja de esto es que usamos lo mejor de lo mejor. Una desventaja es que resulta menos flexible, ya que no tenemos el control directo del código.

# Tutorial para el proyecto

<https://www.youtube.com/watch?v=rsr6X5M6-6M&ab_channel=Codeching>

El inglés del autor es claro, aunque tiene un fuerte acento, supongo que húngaro. Sepan disculpar, y quédense con lo bueno, que son las ideas y los conceptos.

Si te ayuda, activá los letreros automáticos (CC).

El proyecto que hice está completo, y tiene comentarios. Eso te ayudará.

El programador necesita el inglés. Podés esquivar el problema hoy, mañana y pasado, una y otra y otra vez. Pero la factura va a llegar inexorablemente, y cuanto más lo retrases, más intereses deberás pagar.

Hoy, ahora, es el momento de arrancar. Mañana, después, es la muletilla de los vencidos.

It's a step by step Java Spring Boot quick start guide. In this tutorial we are going to setup VsCode (Visual Studio Code) to be able to develop Java Spring Boot application in it. We are going to install Java Extension Pack and set up other extensions. We are going to run our Spring Boot maven application at the end of the tutorial from VsCode.

El autor va siguiendo los pasos de Spring Quickstart Guide:

<https://spring.io/quickstart>

[00:00](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=0s) - Intro

[01:05](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=65s) - Generating Spring Boot application

Él usa el asistente web, yo uso la extensión de VS Code.

[02:26](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=146s) - Installing VS Code Java pack extension

Extension pack for Java, de Microsoft.

[03:16](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=196s) - Installing JDK 11

Usa Java 11 oficial, yo también. El OpenJDK es también bastante popular.

[03:56](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=236s) - Setup system environment variables

Como de costumbre. Agrega variables para Maven. Me parece que da por supuesto que uno tiene instalado Maven por fuera de VS Code. No lo explica, pero me parece que se deduce. Para aplicaciones inicializadas para Spring Boot en realidad no hace falta, porque la aplicación incluye un wrapper, y descarga directamente lo que necesita.

[04:56](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=296s) - Setup VS Code setting.json

ctrl + shift + p

Preferences: open Settings (JSON).

Me parece que no hace falta modificar estos settings, si uno ya arregló las variables de entorno. Abrir una terminal con ctrl+shift+ñ y escribir mvn –version. Si muestra la versión correcta, entonces no hace falta.

[06:08](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=368s) - Setup Java project code

[09:19](https://www.youtube.com/watch?v=rsr6X5M6-6M&t=559s) - Setup Longbok

Muy bueno. Explica cómo usar Lombok annotations.

# Generar el proyecto

Generar el proyecto como venimos haciendo siempre.

Ctrl + shift + p

Java: Create Java project

Spring Boot

Maven Project

3.0.5

Java

tup

demo

jar

17

La única dependencia, por ahora, es Spring Web.

# Instalar extensión de Lombok

Instalar en VS Code la extensión para Lombok:

Lombok Annotations Support for VS Code

Esta es una extensión de VS Code. No es la dependencia que se consigna en el pom.xml.

# Para importar Lombok en Maven

<https://projectlombok.org/setup/maven>

Acá está la dependencia que se debe agregar en el pom.xml.

# pom.xml

Editar pom.xml y agregar la siguiente dependencia:

        <dependency>

            <groupId>org.projectlombok</groupId>

            <artifactId>lombok</artifactId>

            <scope>provided</scope>

        </dependency>

Para instalar las dependencias, en la terminal ejecutar:

mvn clean install

# Probar

Ir a la solapa del explorador de proyectos. Buscar el proyecto demo. A la derecha del nombre, hay una flecha con un engranaje. Pulsar ese botón para ejecutar el proyecto.

Aparecerá arriba a la derecha del panel principal la barra de la aplicación, corriendo. Para verla, hay que ir al browser:

<http://localhost:8080/>

Veremos un cartel que no dice que no tenemos una página de errores. Esto significa que la aplicación está realmente corriendo el Tomcat.

# src\main\java\tup\demo\DemoApplication.java

package tup.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

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

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

import tup.demo.models.User;

@SpringBootApplication

/\*

 \* Notar que este proyecto solo tiene dos paquetes: el principal,

 \* donde está esta clase, y models, donde está la clase User.

 \* No tenemos un paquete controllers, en el cual podamos crear

 \* las clases controladoras. Lo que hacemos, en este ejemplo,

 \* es tomar un atajo. Es una solución sucia y poco elegante,

 \* pero es simple y funciona. Entonces esta clase, la clase

 \* principal de la aplicación, tiene que incorporar las

 \* funcionalidades de controlador. Eso lo hacemos con la

 \* anotación @RestController.

 \*

 \* https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/

 \* springframework/web/bind/annotation/RestController.html

 \*

 \*/

@RestController

public class DemoApplication {

    public static void main(String[] args) {

        SpringApplication.run(DemoApplication.class, args);

    }

    /\*

     \* La documentación oficial no es muy clara que digamos. Está en:

     \*

     \* https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/

     \* springframework/web/bind/annotation/GetMapping.html

     \*

     \* Anotación para asignar solicitudes HTTP GET a métodos de controlador

     \* específicos.

     \* Específicamente, @GetMapping es una anotación compuesta que actúa como atajo

     \* para @RequestMapping(method = RequestMethod.GET).

     \* Este artículo de Baeldung es mucho más claro:

     \* https://www.baeldung.com/spring-new-requestmapping-shortcuts

     \*/

    @GetMapping("/")

    /\*

     \* La documentación oficial no es muy clara que digamos. Está en:

     \*

     \* https://docs.spring.io/spring-framework/docs/current/javadoc-api

     \* /org/springframework/web/bind/annotation/RequestParam.html

     \*

     \* Anotación que indica que un parámetro del método debe vincularse a un

     \* parámetro de la solicitud (request) web. En Spring MVC, los "parámetros de

     \* la solicitud" se asignan a parámetros de qyery, datos de formulario y

     \* partes en solicitudes de multipartes. Esto se debe a que la API de servlet

     \* combina los parámetros de query y los datos de formulario en un solo mapa

     \* llamado "parámetros", y eso incluye el análisis automático del cuerpo de la

     \* solicitud.

     \* Este artículo de Baeldung es mucho más claro:

     \* https://www.baeldung.com/spring-request-param

     \*/

    public String hola(@RequestParam(value = "name", defaultValue = "José") String name) {

        /\*

         \* Usamos varios métodos de la clase User, pero no los implementamos nosotros,

         \* sino que fueron generados por la anotación @Data de Lombok.

         \* No estamos hablando de un patrón de inyección de dependencias.

         \* Simplemente, tenemos un objeto de tipo User, y ese objeto

         \* tiene todos los campos y métodos de su clase.

         \*/

        User user = new User();

        user.setName(name);

        return String.format("¡Hola %s!", user.getName());

    }

}

# src\main\java\tup\demo\models\User.java

Crear un paquete models y agregar la siguiente clase:

package tup.demo.models;

import lombok.Data;

/\*

 \* La documentación oficial está en:

 \* https://projectlombok.org/features/Data

 \* y en

 \* https://projectlombok.org/api/lombok/Data.html

 \* Generates getters for all fields, a useful toString method, and hashCode and equals

 \* implementations that check all non-transient fields. Will also generate setters for

 \* all non-final fields, as well as a constructor.

 \* Equivalent to @Getter @Setter @RequiredArgsConstructor @ToString @EqualsAndHashCode.

 \* Los siguientes tutoriales explican cómo se usa:

 \* https://javabydeveloper.com/lombok-data-annotation/

 \* https://www.educative.io/answers/what-is-the-data-annotation-in-lombok

 \*

 \* OJO: no se trata de un patrón de inyección de dependencias.

 \*

 \* NOTAR que solo hemos declarado el campo name. No declaremos explícitamente

 \* ningún método, y por supuesto no los implementamos. Todo eso lo hace

 \* por nosotros automáticamente la anotación @Data. Esto nos evita

 \* tener que escribir boiler plate code, y teminamos usando las implementaciones

 \* de esos métodos que han sido escritas por los mejores programadores. Tiene

 \* la desventaja de que es poco flexible, porque escapa a nuestro control.

 \*/

@Data

public class User {

    private String name;

}

# Probar

Con esto basta para que muestre un mensaje en el browser. Notar que la URL debe ser como sigue:

<http://localhost:8080>

para tomar el valor por defecto, y

<http://localhost:8080/?name=Rubén>

para pasarle en la URL el parámetro que queramos.