

# Programación Frontend y Backend

*BLOQUE SPRING*

Spring Boot



01

## Introducción



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



### Spring Boot

Es una herramienta que nace con la finalidad de simplificar aun más el desarrollo de aplicaciones basadas en el framework de Spring. Spring Boot busca que el desarrollador solo se centre en el desarrollo de la solución, olvidándose por completo de la compleja configuración que actualmente tiene Spring Core para poder funcionar.

### Características

**Configuración:** Spring Boot cuenta con un complejo módulo que autoconfigura todos los aspectos de nuestra aplicación para poder simplemente ejecutar la aplicación, sin tener que definir absolutamente nada.

**Resolución de dependencias:** Con Spring Boot solo hay que determinar que tipo de proyecto estaremos utilizando y el se encarga de resolver todas las librerías/dependencias para que la aplicación funcione.

### Características

**Despliegue:** Spring Boot se puede ejecutar como una aplicación Stand-alone (sin conexión), pero también es posible ejecutar aplicaciones web, ya que es posible desplegar las aplicaciones mediante un servidor web integrado, como es el caso de Tomcat, Jetty o Undertow.

**Métricas:** Por defecto, Spring Boot cuenta con servicios que permite consultar el estado de la aplicación, permitiendo saber si la aplicación está encendida o apagada, memoria utilizada y disponible, número y detalle de los Bean's creados por la aplicación, etc.

### Objetivos:

- 1- Conocer la autoconfiguración de Spring
- 2- Conocer los starters
- 3- Customizar la configuración por defecto
- 4- Gestionar dependencias



02

## Autoconfiguración



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



### Autoconfiguración:

Spring Boot es un marco de trabajo que utiliza la autoconfiguración para simplificar la configuración y el despliegue de aplicaciones de Spring.

Spring Boot utiliza la autoconfiguración para detectar y configurar automáticamente las dependencias necesarias para una aplicación, como la configuración de la base de datos, el servidor web, el sistema de autenticación, entre otros. También proporciona una serie de ajustes y opciones de configuración para personalizar la configuración predeterminada según las necesidades de la aplicación.



### Autoconfiguración:

La autoconfiguración de Spring Boot es muy útil para proyectos pequeños y medianos, ya que reduce significativamente la cantidad de código y configuración requerida. También es fácil de usar para desarrolladores con diferentes niveles de experiencia, ya que **proporciona una configuración rápida y sencilla** para muchos de los componentes y dependencias comunes que se utilizan en la mayoría de las aplicaciones.

La complejidad viene a la hora de sobrecribir alguna preconfiguración ya que hay que entender como funciona, pero la filosofía de springboot es darte algo para que puedas empezar y luego ya puedas perfeccionarlo, en resumen **facilitar el arranque de proyectos.**

### Starters:

Los "starters" en Spring Boot son paquetes de dependencias que proporcionan una forma fácil y rápida de incluir un conjunto específico de componentes y librerías en una aplicación de Spring. Cada "starter" incluye una combinación específica de componentes y librerías que se utilizan juntos con frecuencia para realizar una tarea específica, como el desarrollo de aplicaciones web, el acceso a bases de datos, la seguridad, entre otros.

Al incluir un "starter" en un proyecto, se incluyen automáticamente todas las dependencias necesarias para realizar una tarea específica. Esto significa que no es necesario buscar y agregar manualmente cada dependencia por separado. Además, los "starters" también incluyen configuraciones predeterminadas que simplifican la configuración y el uso de los componentes incluidos.

### Starters: Ejemplo:

Si añadimos el starter **spring-boot-starter-web**, esto es todo lo que nos aporta:

- Spring MVC y las anotaciones de Spring Web. Con esto podremos realizar aplicaciones basadas en servlets
- Jackson para la serialización de JSON
- Un Apache Tomcat embebido
- Spring Boot
- Spring Boot Autoconfiguration
- Spring Boot Logging
- Jakarta annotations
- Spring Framework Core
- SnakeYAML para gestionar propiedades en formato **YAML Ain't Markup Language (YAML)**

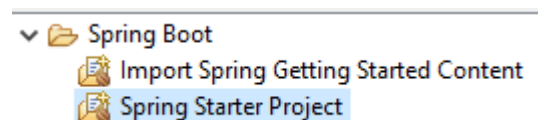
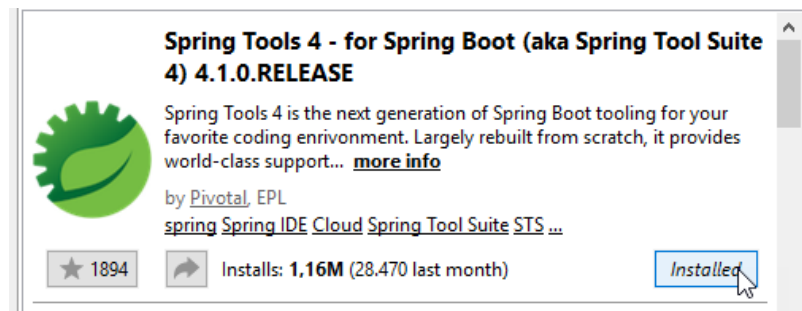
## ¿Cómo arrancamos?

*Existen dos formas de crear un proyecto Spring Boot:*

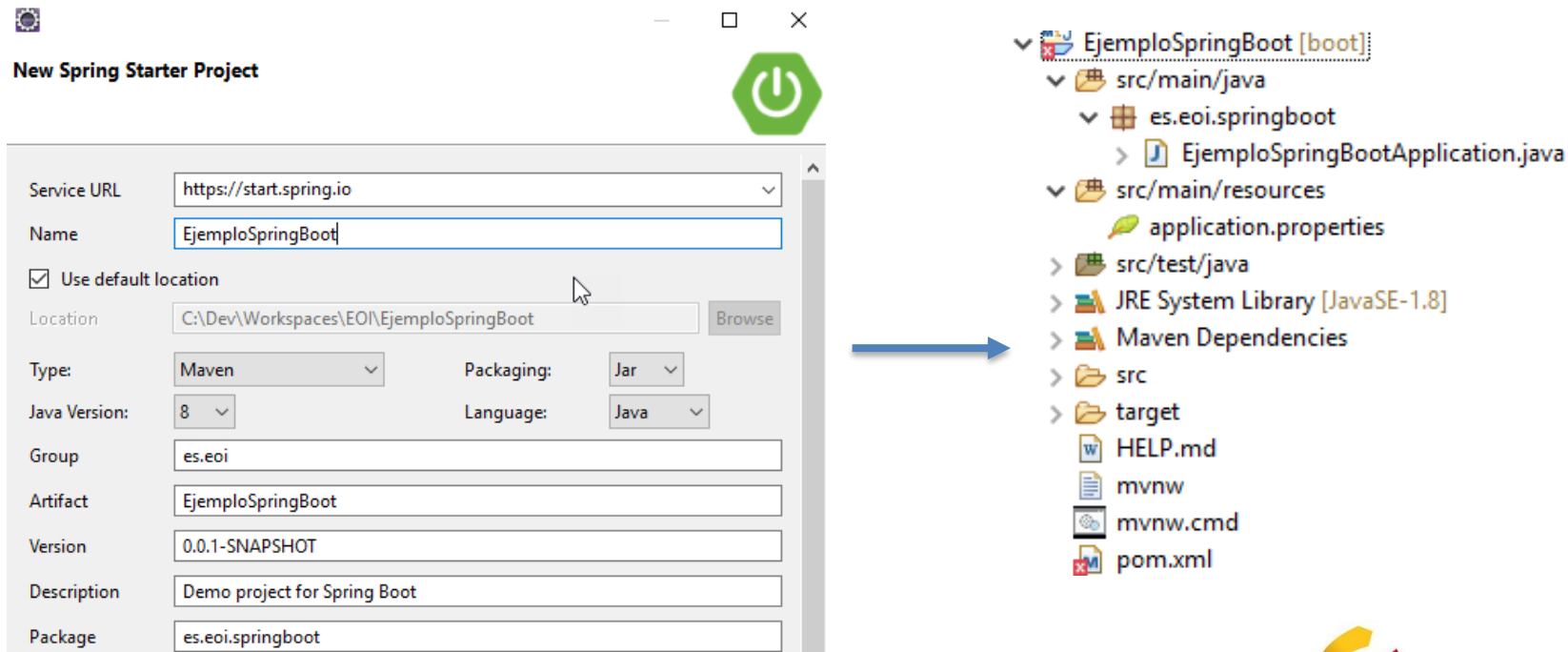
1. Crear el proyecto online:

<https://start.spring.io/>

2. Instalando desde Marketplace (Eclipse) Spring Tools > New Spring Starter Project



### Proyecto Spring Boot



The image shows the 'New Spring Starter Project' dialog box in an IDE. The dialog is titled 'New Spring Starter Project' and has a green power button icon in the top right corner. The fields are filled with the following values:

- Service URL: `https://start.spring.io`
- Name: `EjemploSpringBoot`
- ☒ Use default location
- Location: `C:\Dev\Workspaces\EOI\EjemploSpringBoot` (with a 'Browse' button)
- Type: `Maven`
- Packaging: `Jar`
- Java Version: `8`
- Language: `Java`
- Group: `es.eoi`
- Artifact: `EjemploSpringBoot`
- Version: `0.0.1-SNAPSHOT`
- Description: `Demo project for Spring Boot`
- Package: `es.eoi.springboot`

A blue arrow points from the dialog to a file explorer view of the project structure:

- ▼ EjemploSpringBoot [boot]
  - ▼ src/main/java
    - ▼ es.eoi.springboot
      - ▶ EjemploSpringBootApplication.java
  - ▼ src/main/resources
    - ▶ application.properties
  - ▶ src/test/java
  - ▶ JRE System Library [JavaSE-1.8]
  - ▶ Maven Dependencies
  - ▶ src
  - ▶ target
  - HELP.md
  - mvnw
  - mvnw.cmd
  - pom.xml



03

La clase  
principal



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO

**EQI**

Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



### La clase principal de Springboot

Anotaciones importantes:

**@SpringBootApplication**

**@ComponentScan(basePackages = "edu.es.eoi")**

Y el método main

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

### SUPER IMPORTANTE!!!!

En la clase principal de Springboot si no utilizamos la anotación de `@ComponentScan`, buscará todos los componentes que se encuentren en el package donde este la clase y en todos los packages hijos, pero si no tenemos una estructura jerarquica de paquetes y nuestra clase no esta en el primel nivel puede que **NO ENCUENTRE** los componentes.

### SOLUCIONES:

- 1-Mantener una estructura de paquetes con la clase main en el paquete “padre”
- 2- Definir los paquetes de búsqueda en el `ComponentScan`

## Command Line Runner

Si queremos lanzar una app con springboot por consola en vez de tocar el método main es recomendable hacerlo a través de esta interfaz.

```
public class Example implements CommandLineRunner {...
```

Y sobreescribir el método run

@Override

```
public void run(String... args) throws Exception {...
```



04

## Propiedades



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro





### Propiedades en Spring Boot

Springboot introduce la autoconfiguración de los archivos de propiedades, y una forma sencilla de inyectar parámetros en los beans. Spring boot puede cargar ciertas propiedades con valores por defecto pero podemos sobreescribirlas.

Un ejemplo muy sencillo es cambiar el puerto del tomcat en el que levantamos la app.

Por defecto será el puerto 8080 pero posteriormente veremos como cambiarlo es muy sencillo y es únicamente cambiar el valor en el archivo de propiedades.

### Propiedades en Spring Boot

Vamos a leer propiedades que definamos en el fichero de application.properties

Para ello solo tenemos que anotar con la siguiente anotación @Value donde queremos inyectar la propiedad

```
@Value( "${valorPropiedad}" )
```

```
private String propiedad;
```

### Ejercicio:

Definir una propiedad de saludo:

Message.welcome="Hola Mundo"

1- crear una app que te salude conforme al valor de esta propiedad.

2-cambiar el valor de la propiedad y ver como cambia el comportamiento

### Ejercicio avanzado: Multidioma

Vamos a crear una aplicación que permita cambiar el lenguaje del menú impreso por pantalla utilizando para ello dos ficheros de propiedades distintos.

### Carga condicional en función de propiedades:

Podemos configurar la carga de algunos beans dependiendo del valor de alguna propiedad.

@Bean

@ConditionalOnProperty(prefix="prefix",name="attribute",havingValue="XXX")

Y la propiedad por ejemplo:

prefix.attribute=XXX

Al arrancar la app solo se cargara el servicio si se cumple esa condición.



**Ejercicio: Crear una aplicación que al arrancar cargue el servicio de traducción correspondiente a la propiedad indicada.**

**Crearemos 3 servicios para las zonas de Asia, Eu y USA que implementen la interfaz:**

```
public interface Service {  
    public String doSomething();  
}
```

**Definir los 3 beans a través de @Bean pero poniendo la condición de que solo se cargue el que digamos en el archivo de propiedades → service.zone="USA"**

**Probarlo en un test.**

04

LOGGING



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



### Loggin en springboot:

**Springboot preconfigura el logging en base al fichero de propiedades.  
Utiliza por defecto la implementación de SLF4J.**

### Existen diferentes niveles de traza:

- 1.TRACE: Muestra información detallada para depurar problemas.
- 2.DEBUG: Muestra información adicional que puede ser útil para depurar problemas.
- 3.INFO: Muestra información importante sobre el estado de la aplicación.
- 4.WARN: Muestra una advertencia sobre un problema potencial en la aplicación.
- 5.ERROR: Muestra un error crítico que impide que la aplicación funcione correctamente.

### Loggin en springboot:

En general, es común utilizar los niveles de log INFO y WARN en producción, mientras que se utiliza el nivel de log DEBUG o TRACE en desarrollo y pruebas. Puedes configurar el nivel de traza de log para una clase o paquete específico o para toda la aplicación.

`logging.level.root=WARN` → el por defecto para toda la app

`logging.level.org.springframework.web=DEBUG` → específico para ese paquete

Recuerda que debes reiniciar la aplicación para que los cambios surtan efecto.

### Loggin en springboot:

Preconfigura un logger y para utilizarlo solo tenemos que invocarlo

```
private static Logger LOG = LoggerFactory.getLogger(App.class);
```

A partir de aquí definiremos el nivel de log que queremos

LOG.info(....)

LOG.debug(.....)



### Login en springboot: Ejercicio

**@SpringBootApplication**

**public class MyApplication {**

**private static final Logger LOGGER = LoggerFactory.getLogger(MyApplication.class);**

**public static void main(String[] args) {**

**SpringApplication.run(MyApplication.class, args);**

**LOGGER.debug("Debug level log");**

**LOGGER.info("Info level log");**

**LOGGER.warn("Warn level log");**

**LOGGER.error("Error level log");**

**}**

**}**

05

## Ejercicio Guiado



GOBIERNO  
DE ESPAÑA

MINISTERIO  
DE INDUSTRIA, COMERCIO  
Y TURISMO



Escuela de  
organización  
industrial



**Unión Europea**  
**Fondo Social Europeo**  
**Iniciativa de Empleo Juvenil**  
El FSE invierte en tu futuro



### Ejercicio Guiado

Creamos la siguiente estructura:

es.eoi.springboot.repository

**HelloWorldRepository.java**

**HelloWorldRepositoryImpl.java**

es.eoi.springboot.service

**HelloWorldService.java**

**HelloWorldServiceImpl.java**

## Ejercicio Guiado

Añadiremos en el repository, el siguiente método:

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

A continuación crearemos el flujo correspondiente entre *Service*, *ServiceImpl*, *Repository* y *RepositoryImpl*, utilizaremos las anotaciones de **Spring** para configurar los archivos anteriores: *@Service*, *@Repository*, *@Autowired*, etc.

Por último accederemos a la clase de test que viene creada por defecto:

```
▼ src/test/java  
  ▼ es.eoi.springboot  
    ▼ EjemploSpringBootApplicationTests.java
```

## Ejercicio Guiado

```
package es.eoi.springboot;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;
import es.eoi.springboot.service>HelloWorldService;

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

    @Autowired
    private HelloWorldService service;

    @Test
    public void contextLoads() {
        System.out.println(service.helloWorld());
    }
}
```