Spring Platform

- Es un conjunto de **proyectos open source** desarrollados en Java con el objetivo de agilizar el desarrollo de aplicaciones.
- Cuenta con **gran variedad de frameworks, que nos facilitan el trabajo** desde el acceso a datos, infraestructura, creación de aplicaciones web, microservicios, etc.

https://spring.io

https://spring.io/projects

Spring Boot

- Es una extensión de Spring framework que permite la creación **fácil y rápida de aplicaciones web** listas para producción con el concepto de “just run” (solo ejecutar).
- Requiere una **mínima configuración** y se complementa con muchos proyectos de Spring Platform y librerías de terceros.

https://spring.io/projects/spring-boot

Creamos nuestro primer proyecto:

- **Spring Initializr** es una pequeña utilidad Web que permite crear un esqueleto de un proyecto de **Spring Boot** con las opciones que queramos configurar.
- Permite elegir el **proveedor de dependencias** (Maven, Gradle, etc), **el lenguaje a utilizar** (Java, Kotlin, etc), el **tipo de empaquetado** (Jar, War), las **dependencias/librerías** que necesitamos, entre otras configuraciones iniciales.
- Se puede utilizar desde su web oficial, o descargando un complemento para Intellij Idea o el IDE que estemos utilizando.

Estructura del proyecto:

Se crean los siguientes archivos y directorios. Entre los más importantes se encuentran:

- pom.xml
- application.properties
- carpeta target (luego de buildear el proyecto)
- profile (definir entorno de trabajo)

Application.java:

Spring Boot viene con un servidor web embebido que nos permite ejecutar la aplicación directamente desde el archivo main que contiene la anotación **@SpringBootApplication**, desde Intellij IDEA presionando el botón.

Creando una API:

- Dentro del paquete **helloworld** creamos el archivo HelloRestController.java y en el mismo creamos el el método **sayHello()** que va a devolver un String (esto será el formato de respuesta de la API).
- **@RestController:** Esta anotación informa a Spring que la clase va a ser utilizada para generar una API REST.
- **@GetMapping:**Esta anotación configura el método sayHello() para que sea un **punto de entrada HTTP GET** en**“/”**.

MVC

- El **Modelo-Vista-Controlador** es un patrón de arquitectura de software que separa la lógica de negocio, de la lógica de la vista en una aplicación.
- **Modelo:** Se encarga de los datos, generalmente (pero no obligatoriamente) consultando alguna base de datos.
- **Controlador:** Se encarga de “controlar”; recibe las órdenes del usuario, solicita los datos al modelo y se los comunica a la vista.
- **Vista:**Es la representación visual de los datos

Inyección de dependencias e IoC

Inversión de Control:

- Es un principio de diseño de software en el que el flujo de ejecución de un programa se invierte respecto a los métodos de programación tradicionales.
- Generalmente, es un framework el que toma el control, el que define el flujo de actuación o el ciclo de vida de una petición.
- Es un concepto también conocido como principio de Hollywood: *“No nos llames, nosotros te llamamos”*.
- En la Inversión de control (IoC en inglés), el objeto cede el control a alguien más (puede ser un propio framework u otros objetos).

Inyección de Dependencias:

- La inyección de dependencias (DI en inglés), es un Patrón de Diseño orientado a objetos en el que se le suministran los objetos a una clase en lugar de que sea ella misma quien los cree.
- Su objetivo es tener un código lo suficientemente desacoplado como para que permita un fácil mantenimiento y escalabilidad. Esta técnica permite que a un objeto se le provean las dependencias que necesite.
- Es un proceso mediante el cual los objetos definen sus dependencias (es decir, los otros objetos con los que van a trabajar)

Formas de inyectar dependencias:

Existen distintas formas principales mediante las cuales se puede implementar la inyección de dependencias:

- Mediante un constructor
- Mediante un método Set
- Mediante la anotación **@Autowired**

Diccionario/Resumen de Anotaciones de esta clase

- **@SpringBootApplication:** Nos permite especificar que trabajamos sobre una aplicación de Spring Boot. Habilita 3 características, la autoconfiguración del proyecto (@EnableAutoConfiguration), la búsqueda de componentes/paquetes de la aplicación (@ComponentScan) y la posibilidad de realizar configuraciones extras (@Configuration).

- **@RestController:** Anotación para identificar el controlador de un servicio REST

- **@GetMapping:** Anotación para "mapear" las peticiones mediante el método GET dentro de nuestra aplicación.