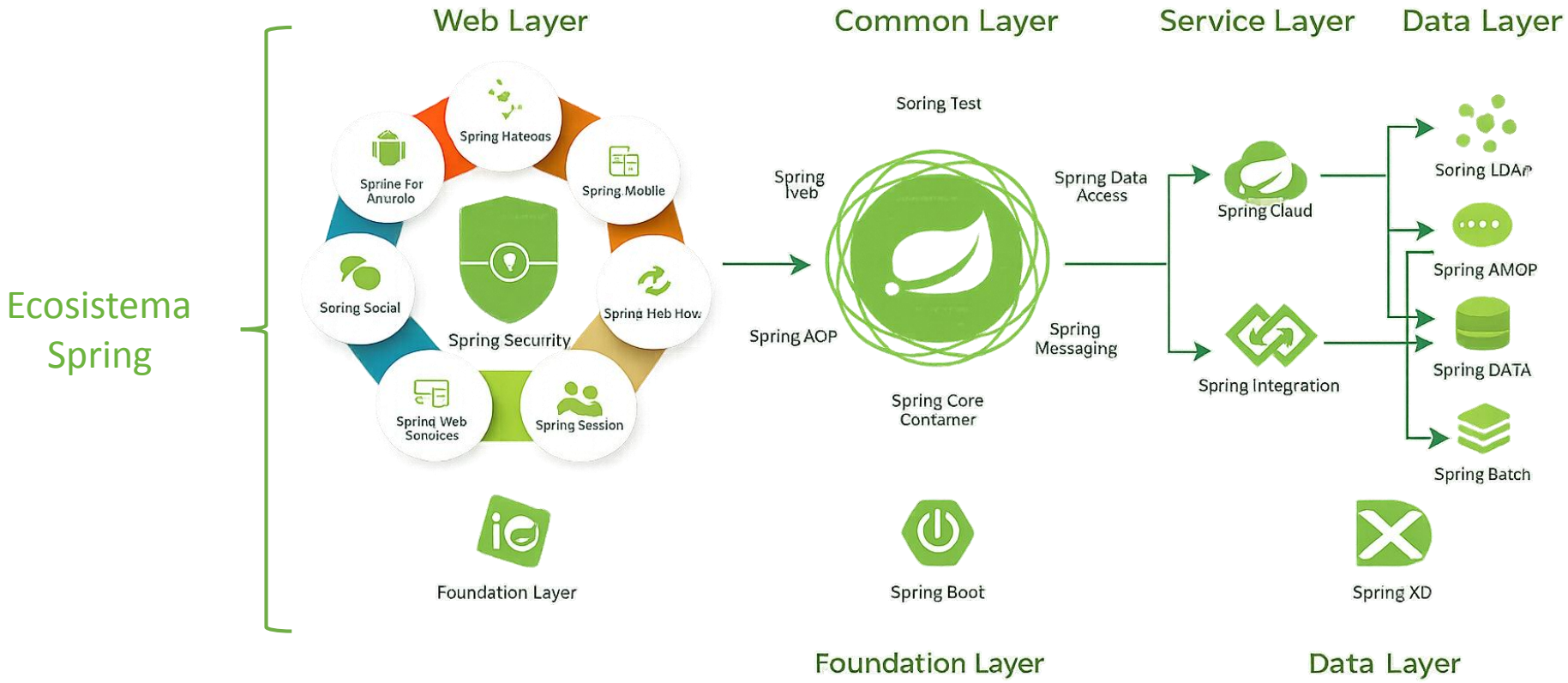
Procesos de Desarrollo de Software

SpringBoot

Sesión Práctica 01

Contenidos

1. ¿Qué es SpringBoot?
2. Elementos y configuración.



¿Qué es spring boot ?

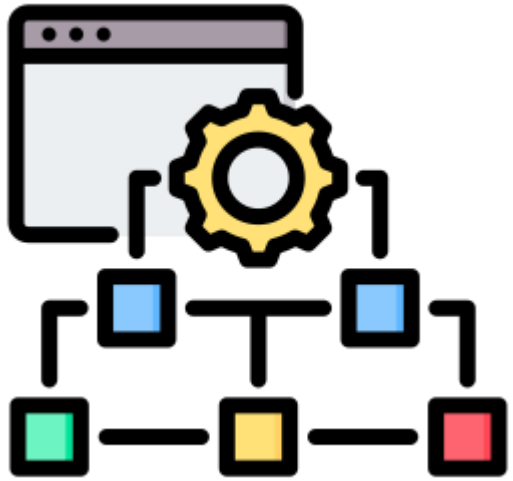
Framework de desarrollo de aplicaciones web en Java

1. SpringBoot - ¿Qué es un framework?

- Llamamos **framework de desarrollo** a:

  "Conjunto de herramientas y reglas que ayudan a crear programas"

- Un framework nos proporciona:
  - Una pila tecnológica.
  - Un conjunto de reglas o funcionalidad preestablecida.
  - Un sistema de configuración.
  - Un ciclo de vida de las acciones de nuestro software.



- Existen muchos frameworks. En concreto para Java:
  - **Jakarta EE**: El estándar de facto en Java.
  - **Spring:** Alternativa más automatizada, con su propio ecosistema.
  - **Quarkus:** Pensado en "la nube", orientado a microservicios.

  ….

1. SpringBoot - ¿Qué es SpringBoot?

- SpringBoot es un Framework de desarrollo web, autoconfigurado y autoextensible basado en Spring.
- ¿Qué significa realmente esto? *Veámoslo con un ejemplo:*
- Supongamos que queremos construir una casa:
  - Java sería el material que usaremos: ladrillos, cemento, pintura…
  - Spring Framework serían los planos.
  - Nosotros, los obreros, somos los encargados de:
    - Unir materiales y fabricar cada elemento (*puertas, ventanas, etc.*)
    - Seguir los planos para encajar todo donde debe.
    - Asegurar cada elemento para que cumpla su función.
    - Controlar todos los imprevistos posibles.
    - Etc. 😢

1. SpringBoot - ¿Qué es SpringBoot?

- Y en este ejemplo ¿Qué sería SpringBoot? ➡ Una casa prefabricada.
- Nosotros elegimos a partir de un catálogo:
  - Las estancias.
  - La disposición.
  - Los elementos de cada estancia.

- Con elementos prefabricados ahorramos:
  - Tiempo en configuración.
  - Testear cada nuevo elemento que se añade.
  - Problemas por conflictos entre componentes.

1. SpringBoot - ¿Qué consigo con SpringBoot?

Desde un punto de vista del desarrollador…

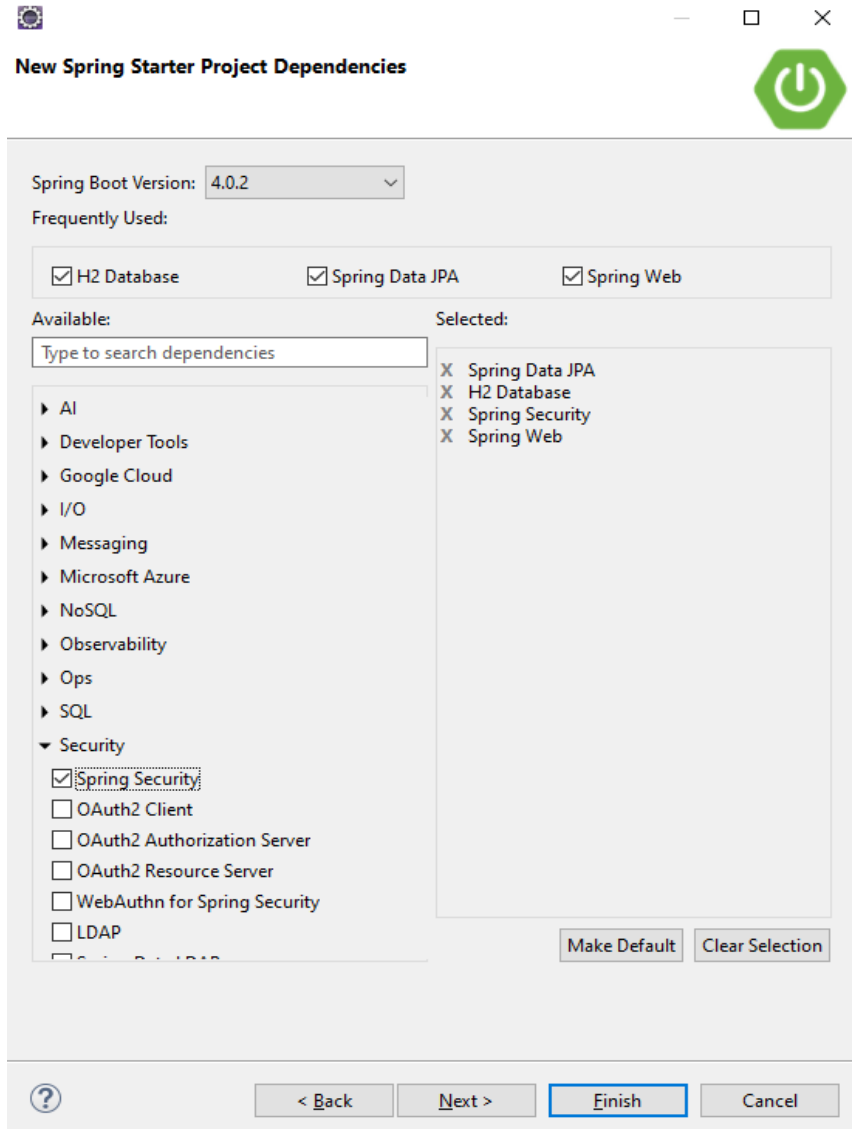
¿Qué tengo que hacer para tener mi aplicación disponible al mundo?

- Instalar y configurar un servidor web.
- Instalar y configurar el lenguaje de programación sobre el que funcionará mi software.
- Instalar y configurar todas las librerías que usará mi software.

1. SpringBoot - ¿Qué consigo con SpringBoot?

¿Qué gano con SpringBoot?

- Un servidor web autoejecutable y configurado.
- Un ecosistema de librerías Maven para:
  - Gestionar aplicaciones web.
  - Crear y configurar bases de datos.
  - JPA para lecturas y escrituras en base de datos.
  - Librería de autenticación y autorización.
  - Sistema de eventos.
  - Sistema de descubrimiento de contenedores.
  - …

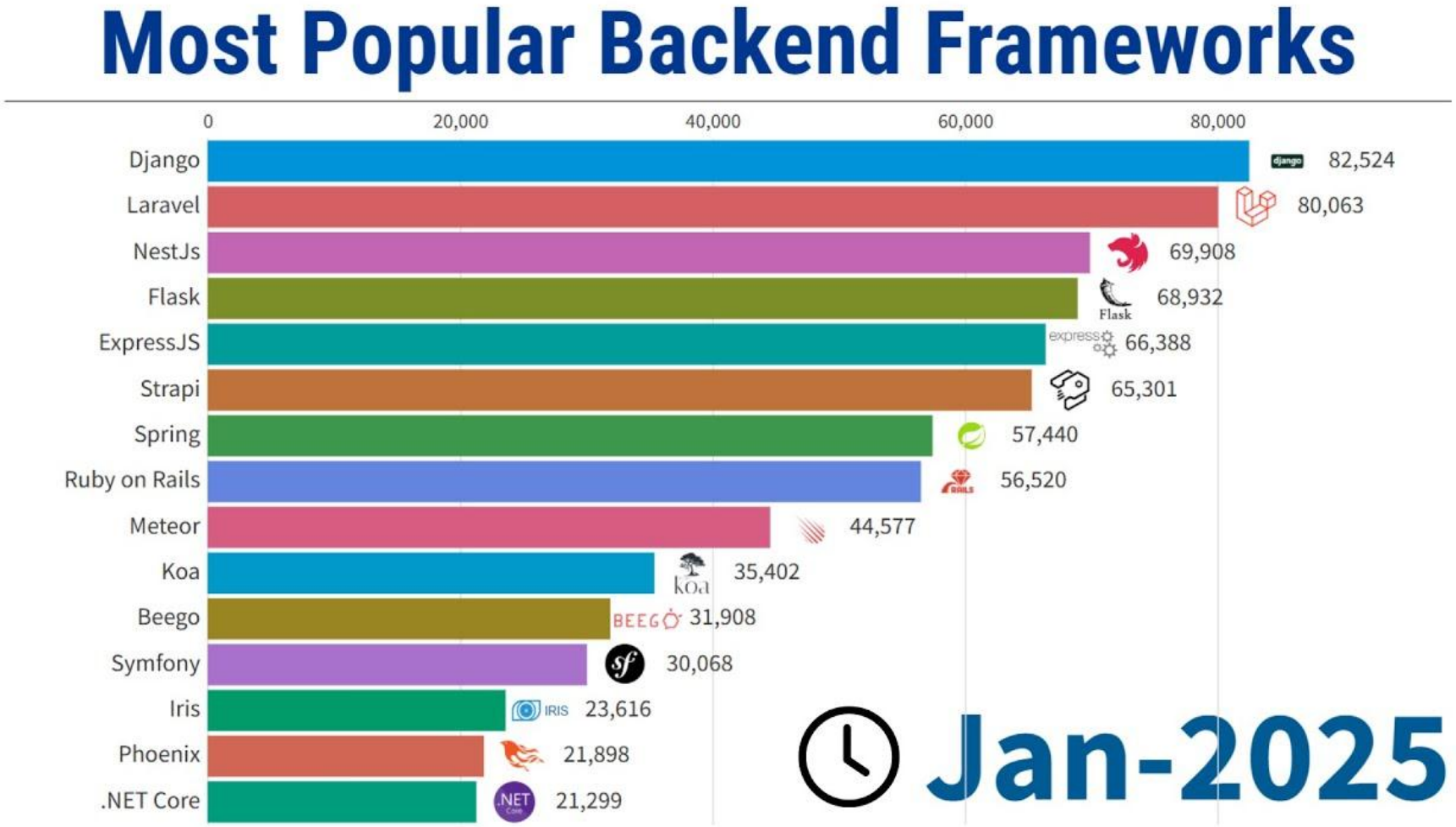


1. SpringBoot - ¿Qué consigo con SpringBoot?

- Un proyecto SpringBoot creado con su arquetipo initializr ya tiene:
  - Un servidor (contenedor) web configurado.
    - Por defecto *Apache Tomcat* (pero puede cambiarse por *Jetty* o *Undertow*)
  - Una aplicación base auto arrancable.

- Para esta práctica nosotros añadiremos:
  - Base de datos en memoria h2.
  - Módulo web.
  - Módulo JPA.
  - Framework de validación.

1. SpringBoot - ¿Por qué usar SpringBoot?



https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2025/

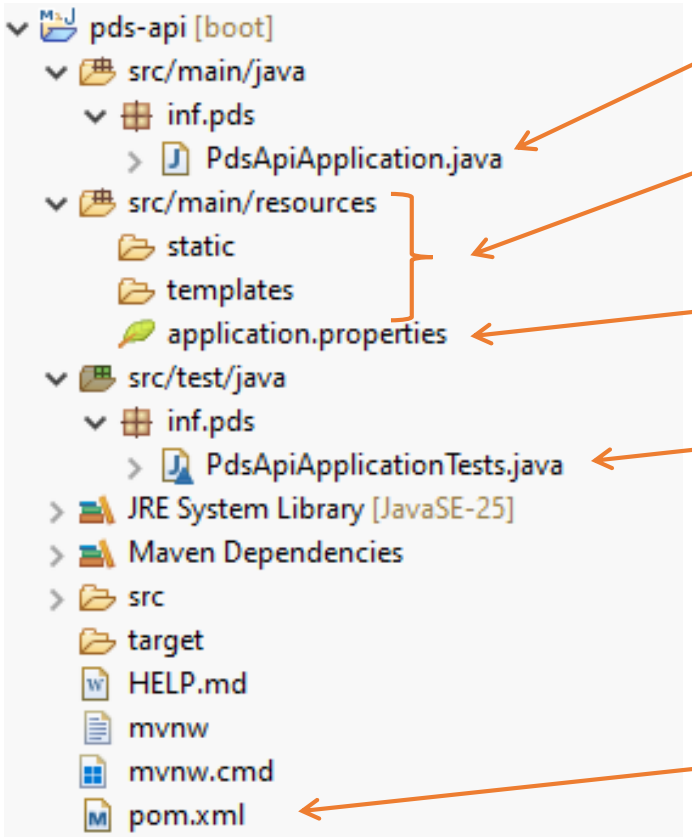
1. SpringBoot - ¿Quién usa SpringBoot?

SpringBoot: Elementos y configuración

Framework de desarrollo de aplicaciones web en Java

2. SpringBoot – Elementos y configuración

Un proyecto SpringBoot es un proyecto Maven o Gradle con un conjunto de librerías.



Clase Java que inicia el proyecto.

Recursos estáticos: css, html, jpg, etc.

Fichero de propiedades.

Clases que ejecutan test.
Ciclo de vida diferente al de la aplicación.

Fichero Maven que define la configuración del proyecto.

2. SpringBoot – Elementos y configuración

- Clase de inicio de SpringBoot.
  - Clase de inicialización.
  - No debemos moverla de paquete ni renombrarla.
  - No debemos meter código nuestro, salvo que sea configuración .

```java
import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class PdsApiApplication {

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

}
```

Anotación de SpringBoot.
Indica que es la clase de inicio.
Activa la configuración automática y el escaneo de componentes.

Arranque de aplicación.

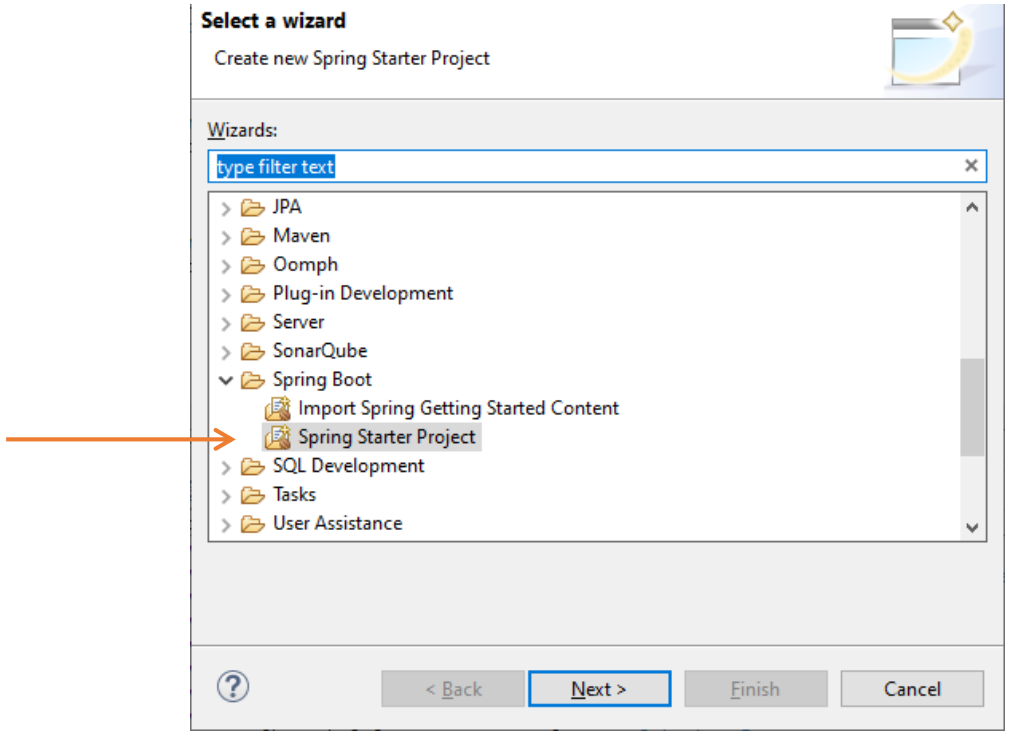
2. SpringBoot – Ficheros de configuración

- Fichero application.properties
  - Par clave-valor para configurar la aplicación
  - Permite tener diferentes ficheros por entorno
  - Sus valores se pueden leer desde Java
    - Para ello usamos la anotación *@Value(${nombre de la clave})*

```
public class SaludoRestAdapter {

    private static final Logger log = LoggerFactory.getLogger(SaludoRestAdapter.class);

    @Value("${pds.saludo}")
    private String saludo;

    public void saluda() {
        log.info("Saludamos: {}", saludo);
    }

}
```

2. SpringBoot – Ejercicio

- Ejercicio:
  - Crea el proyecto SpringBoot a partir del Wizard de Eclipse.
  - Crea la clase `ServicioRestAdapter`.
  - Crea una variable `pds.saludo=Hola Mundo` en el fichero `application.properties`.
  - Invoca el método `saludo`.



2. SpringBoot – Ejercicio

- Ejercicio:
  - Crea el proyecto SpringBoot a partir del Wizard de Eclipse.
  - Crea la clase `ServicioRestAdapter`.
  - Crea una variable `pds.saludo=Hola Mundo` en el fichero `application.properties`.
  - Invoca el método `saludo.`

```java
import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class PdsApiApplication {

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

        SaludoRestAdapter saludoRestAdapter = new SaludoRestAdapter();
        saludoRestAdapter.saluda();
    }

}
```

Llamada a nuestra clase

2. SpringBoot – Ejercicio

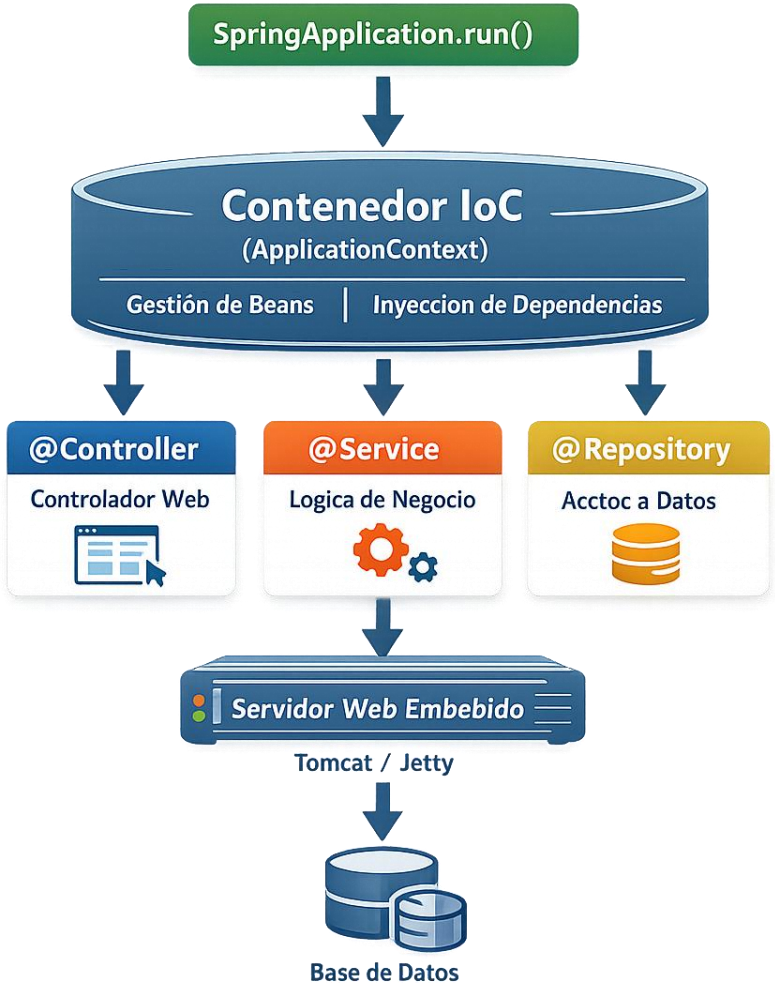
- Ejercicio:
  - Crea el proyecto SpringBoot a partir del Wizard de Eclipse.
  - Crea la clase `ServicioRestAdapter`.
  - Crea una variable `pds.saludo=Hola Mundo` en el fichero `application.properties`.
  - Invoca el método `saludo.`

```
inf.pds.adapters.SaludoRestAdapter        : Saludamos: null
```

En la consola no sale el valor de la variable ¿por qué?

2. SpringBoot – Inyección de dependencias

- SpringBoot gestiona todos sus componentes mediante anotaciones.
  - Ya no haremos **`new XXX`** cuando creemos un servicio o controlador.



2. SpringBoot – Inyección de dependencias

¿Qué es la Inversión de Control?

- Es el contenedor quién crea los objetos.
- Nosotros los solicitamos y los usamos (*Inyección de dependencias*).
- El ciclo de vida y estado lo gestiona el contendor.

- SpringBoot gestiona los componentes mediante anotaciones:
  - `@Component`:
    - Anotación genérica que convierte una clase en componente
  - `@Service`:
    - Hereda de **`@Component`**, convierte una clase en un componente de tipo servicio
  - `@Repository`:
    - Hereda de **`@Component`**, convierte una clase en un componente de tipo controlador JPA
  - `@RestControler`:
    - Hereda de **`@Component`**, convierte una clase en un componente de tipo controlador REST
  - `@Configuration`:
    - Hereda de **`@Component`**, convierte una clase en clase de configuración que se cargará antes de las demás.

2. SpringBoot – Inyección de dependencias

Los componentes de SpringBoot tienen acceso a funcionalidad especial según el tipo de componente defina.

- Para nuestro ejemplo vamos a usar **`@Component`**.

```java
@Component
public class SaludoRestAdapter {

    private static final Logger log = LoggerFactory.getLogger(SaludoRestAdapter.class);

    @Value("${pds.saludo}")
    private String saludo;

    public void saluda() {
        log.info("Saludamos: {}", saludo);
    }

}
```

2. SpringBoot – Inyección de dependencias

Adapto el lanzador para que llame al método **`saluda`**.

Importante: Solo es para este ejemplo. Implemento esta interfaz para tener el método run en esta clase y poder ejecutarlo.

Declaro un constructor y recibo la clase ¿Quién la crea? SpringBoot

```java
@SpringBootApplication
public class PdsApiApplication implements CommandLineRunner{

    private final SaludoRestAdapter saludoRestAdapter;

    // Inyección de dependencia por constructor
    public PdsApiApplication(SaludoRestAdapter saludoRestAdapter) {
        this.saludoRestAdapter = saludoRestAdapter;
    }

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

    @Override
    public void run(String... args) {
        saludoRestAdapter.saluda();
    }
```

Llamo al método

2. SpringBoot – Elementos y configuración

Adapto el lanzador para que llame al método **`saluda`**.

```
inf.pds.adapters.SaludoRestAdapter       : Saludamos: Hola PDS
```

¡Funciona! ¿Qué ha pasado?

- Ahora, gracias a las anotaciones, SpringBoot ha cargado en su gestor de componentes las clases java para gestionarlas.
- Esas clases tienen acceso a la configuración y al resto de componentes.
- Gracias a la inversión de control, es SpringBoot quien gestiona la creación, nosotros sólo debemos encargarnos del uso.
- En los constructores indicamos los componentes que queremos usar.
  - SpringBoot se encarga de asignarlos.

2. SpringBoot – Elementos y configuración

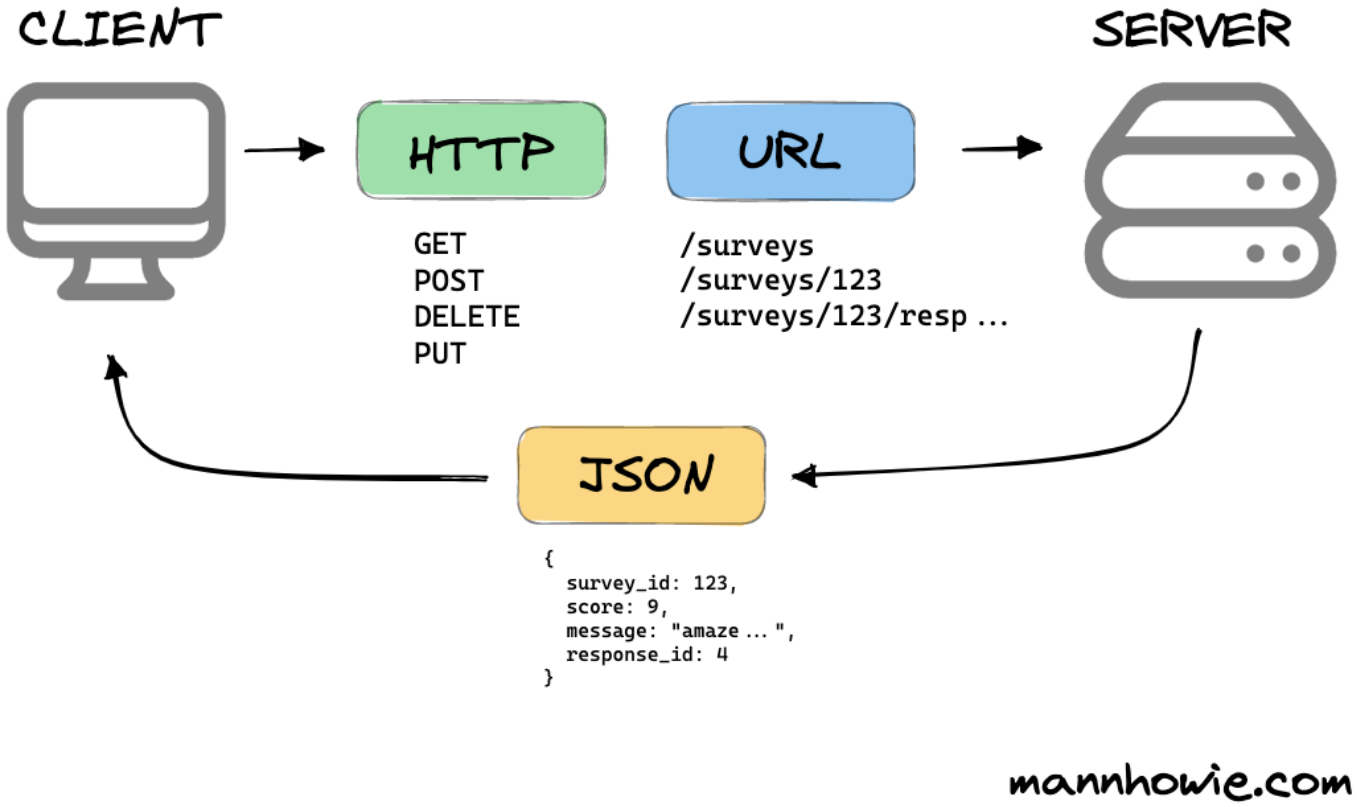
SpringBoot tiene un gran número de anotaciones.

- Cada anotación añade funcionalidad o crea componentes.
- Debemos consultar su manual para, según el tipo de servicio a crear, conocer las anotaciones relevantes.

Main Class	@SpringBootApplication	Spring Boot auto configuration
REST Endpoint	@RestController	Class with REST endpoints
	@RequestMapping	REST endpoint method
	@PathVariable	URI path parameter
	@RequestBody	HTTP request body
Periodic Tasks	@Scheduled	Method to run periodically
	@EnableScheduling	Enable Spring's task scheduling
Beans	@Configuration	A class containing Spring beans
	@Bean	Objects to be used by Spring IoC for dependency injection
Spring Managed Components	@Component	A candidate for dependency injection
	@Service	Like @Component
	@Repository	Like @Component, for data base access
Persistence	@Entity	A class which can be stored in the data base via ORM
	@Id	Primary key
	@GeneratedValue	Generation strategy of primary key
	@EnableJpaRepositories	Triggers the search for classes with @Repository annotation
	@EnableTransactionManagement	Enable Spring's DB transaction management through @Beans objects
Miscellaneous	@Autowired	Force dependency injection
	@ConfigurationProperties	Import settings from properties file
Testing	@SpringBootTest	Spring integration test
	@AutoConfigureMockMvc	Configure MockMvc object to test HTTP queries

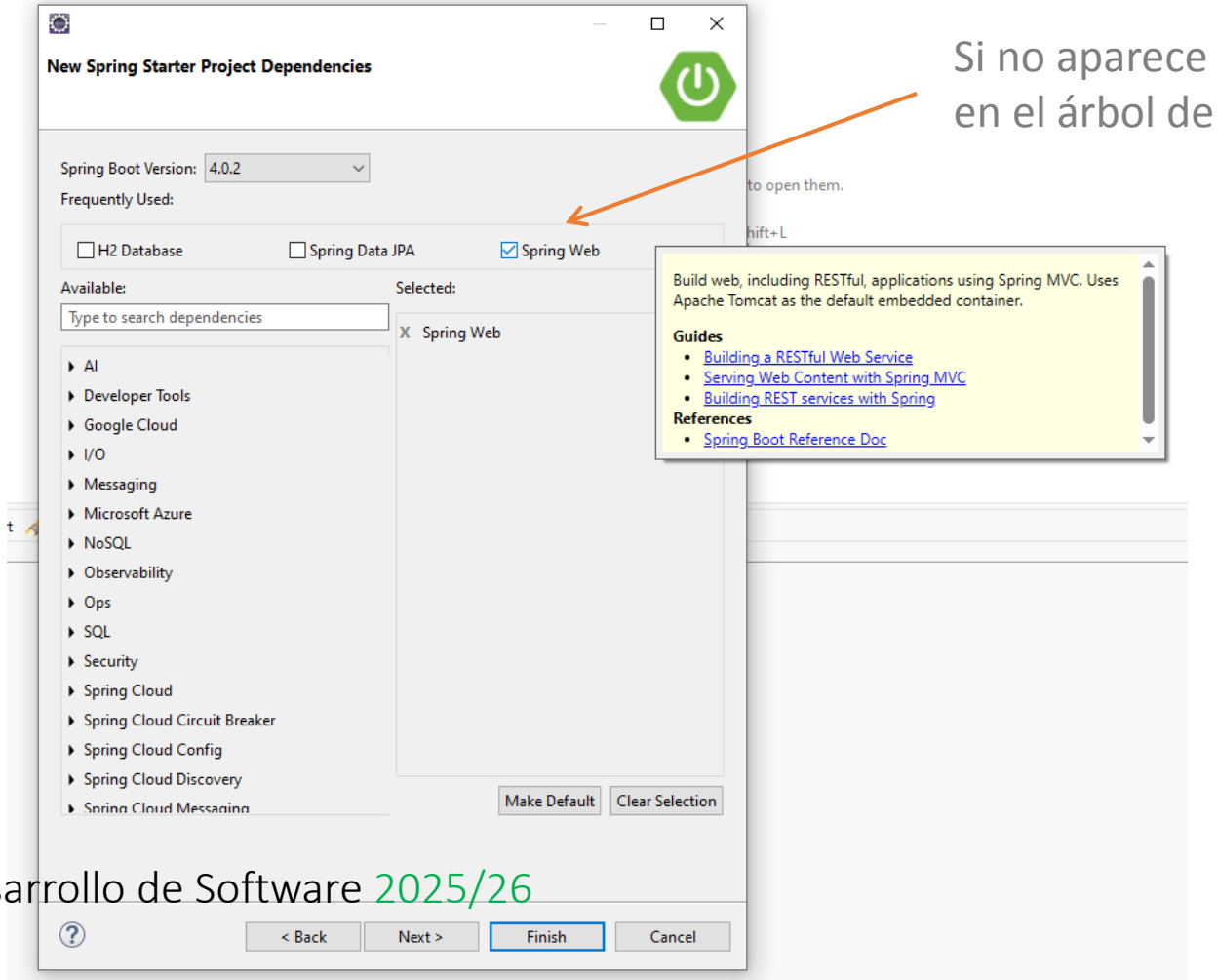
2. SpringBoot – Elementos y configuración

- **Ejercicio:** Endpoint REST que devuelva el mismo texto que antes.



2. SpringBoot – Elementos y configuración

- **Ejercicio:** Endpoint REST que devuelva el mismo texto que antes.
  Creamos un nuevo proyecto añadiendo el módulo **`SpringWeb`**.



Si no aparece lo buscamos en el árbol de componentes

2. SpringBoot – Elementos y configuración

- **Ejercicio:** Endpoint REST que devuelva el mismo texto que antes.
  Añadimos al fichero `application.properties` dos textos.

application.properties

```
1 spring.application.name=pds-api
2 pds.saludo=Hola Mundo
3 pds.despedida=Adios a todos
```

2. SpringBoot – Elementos y configuración

- **Ejercicio:** Endpoint REST que devuelva el mismo texto que antes.
  Creamos nuestro endpoint REST.

Convierte la clase en un componente capaz de escuchar peticiones http

Indicamos que esta clase solo atienda las peticiones que vayan dentro de la ruta /pds

Cargamos las propiedades desde el application.properties

Indico que espero una petición http GET en la ruta /saluda

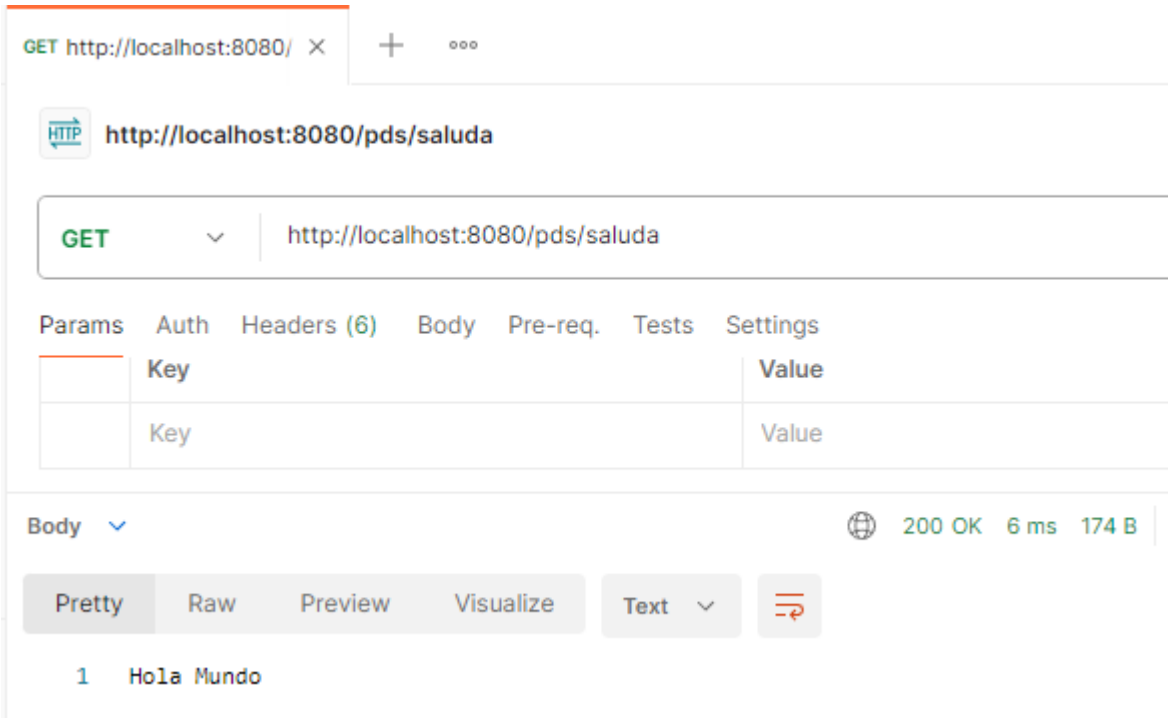
Indico que espero una petición http GET en la ruta /despide

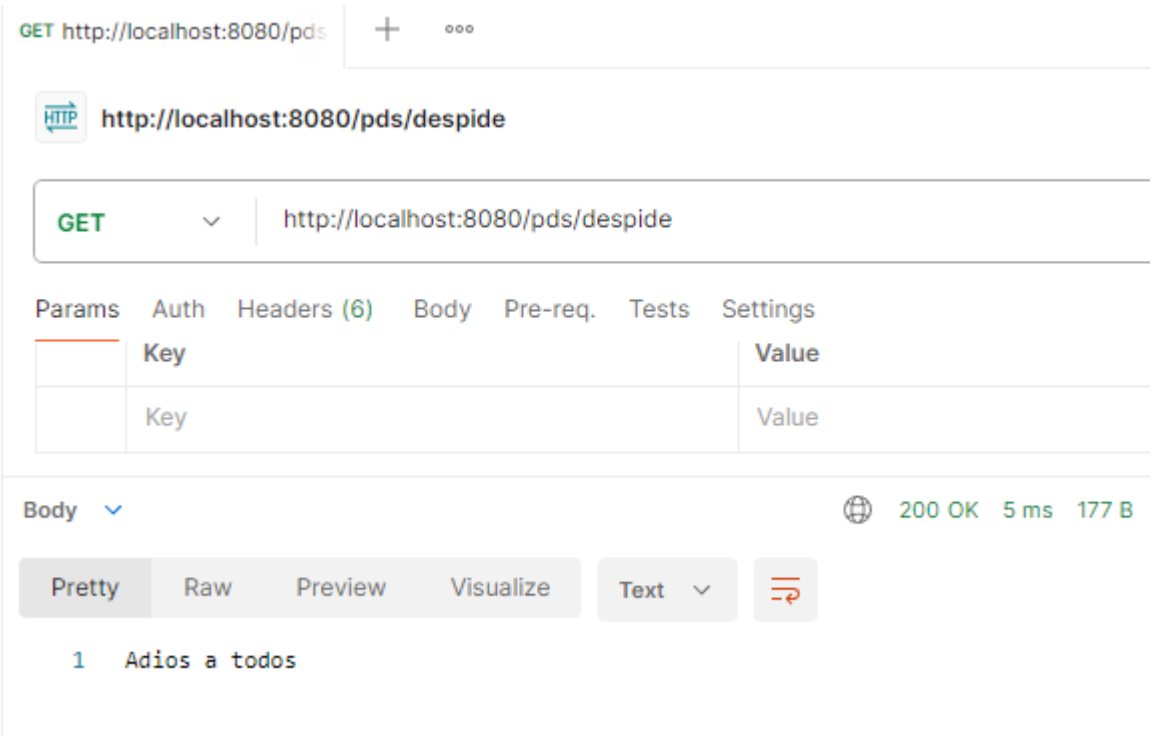
```
@RestController
@RequestMapping("/pds")
public class SaludaRestController {

    @Value("${pds.saludo}")
    private String saludo;

    @Value("${pds.despedida}")
    private String despedida;

    //La url de acceso seria http://localhost:8080/pds/saluda
    @GetMapping("/saluda")
    public String saluda() {
        return saludo;
    }

    //La url de acceso seria http://localhost:8080/pds/despide
    @GetMapping("/despide")
    public String despide() {
        return despedida;
    }

}
```

Importante la ruta empieza en lo indicado por @RequestMapping y cada endpoint añade su path indicado. En este caso, en la anotación @GetMapping.

2. SpringBoot – Elementos y configuración

- **Ejercicio:** Endpoint REST que devuelva el mismo texto que antes.
  Una vez configurado, arrancamos el proyecto y probamos en el navegador o cliente el acceso.





Cliente usado [Postman]