





PORQUE SPRINGBOOT

- Permite una experiencia mucho más accesible y sencilla para todo el desarrollo de Spring.
- Sin una configuración complicada para los desarrolladores
- Provee un 'starter' POMs para simplificar la configuración de Maven
- Usa una herramienta como MAVEN o GRADLE
- Desarrollo rápido en producción
- Servidor web Tomcat embebido, Jetty o Undertow directamente (no se necesita desplegar archivos WAR)
- Se guarda en memoria

Spring Boot está construido sobre Spring Framework. Es una versión más automatizada y simplificada de Spring. Spring Boot facilita la creación de una aplicación en funcionamiento en unos minutos.



ARQUITECTURA DE SPRINGBOOT

PRESENTATION LAYER: Es la primera de las capas. Es responsable por:

- ✓ Autenticación.
- ✓ Convertir datos JSON en objetos (y vice versa).
- ✓ Manejar request (peticiones) HTTP.
- ✓ Transferir la autenticación a la capa de negocios (business layer).

Esta capa es equivalente a una clase de tipo Controller. La clase Controller maneja todas las peticiones API REST(request) entrantes (GET, POST, PUT, DELETE, PATCH) desde el cliente (generalmente un browser).

BUSINESS LAYER: Es responsable por:

- ✓ Realizar la validación.
- ✓ Realizar la autorización.
- ✓ Manejar la lógica de negocios y sus reglas.

Esta capa es la equivalente a la clase de Servicios, donde se maneja la lógica de negocios. La capa de negocios (Business layer) se comunica con la capa de presentación (Presentation layer) y la capa de persistencia (Persistence Layer).

PERSISTENCE LAYER: Es responsable por:

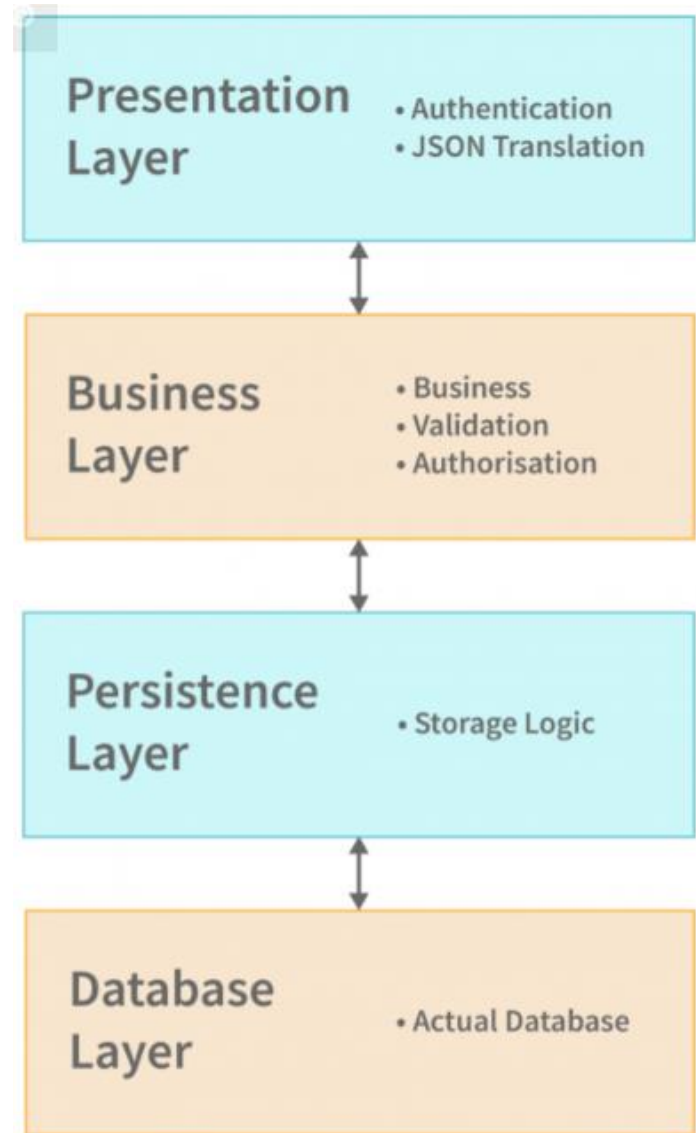
- ✓ Contener la lógica de almacenamiento.
- ✓ Recuperar datos y persistirlos a la BD (y vice versa).

Esta capa es equivalente a una interface de tipo Repository, que es donde escribimos sql queries. La capa de persistencia (Persistence layer) es la única que se comunica con la capa de negocios (Business layer) y la capa de la BD (Database layer).

DATABASE LAYER: Es responsable por:

- ✓ Efectuar operaciones de BD (principalmente operaciones CRUD).

Esta capa es simplemente la BD.



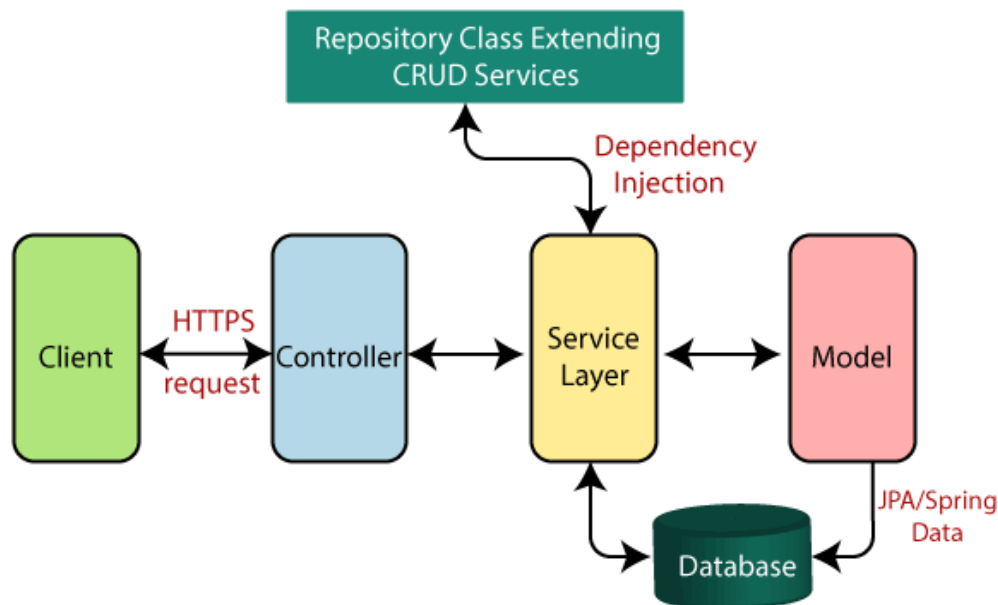


SPRINGBOOT WORKFLOW

El flujo de Spring Boot es así:

- 1.El Cliente hace una petición (request) HTTP.
- 2.La clase Controller recibe la petición HTTP.
- 3.La clase Controller entiende que tipo de petición debe procesar y lo hace.
- 4.Si es necesario, llama a la clase de Servicio.
- 5.La clase Service maneja la lógica de negocio actuando sobre los datos de la BD mapeando los datos vía JPA.
- 6.Si todo está bien retorna la vista (view) una página JSP para renderizar en el browser.

Spring Boot flow architecture





SPRINGBOOT – CONCEPTOS - I

ANOTACIONES (Annotations) *SpringBoot*

Es más fácil trabajar con *SpringBoot* porque configura automáticamente objetos y recursos usando anotaciones (*annotations*), pero son muchas, por lo que probablemente sea una buena idea revisar algunas de las principales para iniciar una aplicación.

@Autowired

Podemos usar *@Autowired* para marcar una dependencia que *SpringBoot* va a resolver e inyectar. Podemos usar esta anotación con un *constructor*, *setter* o inyección de campo.

@SpringBootApplication

Usamos esta anotación para marcar la clase principal de una aplicación *SpringBoot*, encapsula las anotaciones *@Configuration*, *@EnableAutoConfiguration* y *@ComponentScan* con sus atributos predeterminados.



SPRINGBOOT – CONCEPTOS - II

Inyección de Dependencias (Dependency Injection)

Antes de que podamos entender la inyección de dependencias, primero debemos entender qué es la inversión de control (*IoC*) y por qué es importante.

¿Qué es la Inversión de Control?

La inversión de control es un principio de la ingeniería de software mediante el cual el control de objetos o partes de un programa se transfiere a un contenedor o *framework*. Se usa con mayor frecuencia en el contexto de la programación orientada a objetos. Usa el patrón Hollywood (“No me llames, yo te llamo”).

A diferencia de la programación tradicional, en la que nuestro código personalizado realiza llamadas a una biblioteca, *IoC* permite que un *framework* y otro artefacto tome el control del flujo de un programa y realice llamadas a nuestro código personalizado.

Las ventajas de esta arquitectura son:

- a) Desacoplar la ejecución de una tarea de su implementación
- b) facilita el cambio entre diferentes implementaciones
- c) mayor modularidad de un programa
- d) mayor facilidad para probar un programa aislando un componente o dependencias y permitiendo que los componentes se comuniquen a través de contratos



SPRINGBOOT – CONCEPTOS - III

¿Qué es la Inyección de Dependencias?

La inyección de dependencia es un patrón a través del cual implementamos IoC, donde el control que se invierte es la configuración de las dependencias del objeto. El acto de conectar objetos con otros objetos, o "inyectar" objetos en otros objetos, lo realiza un ensamblador en lugar de los objetos mismos.

En pocas palabras, esto permite un acoplamiento flexible de los componentes y transfiere la responsabilidad de administrar los componentes al contenedor.

Existen 3 tipos generales de inyección

Constructor Injection

```
Class Animal {  
    @Autowired  
    public Dog dog() {  
        return new Dog();  
    }  
}
```

Setter Injection

```
Class Animal {  
    private Dog dog;  
  
    @Autowired  
    public void setDog(Dog dog) {  
        this.dog = dog;  
    }  
}
```

Field Injection

```
Class Animal {  
    @Autowired  
    private Dog dog;  
}
```



SPRING INITIALIZR - I

Spring Initializr es una herramienta web proporcionada por Pivotal Web Service, con ella podemos generar fácilmente la estructura de un proyecto básico *SpringBoot*. Ofrece API extensible para crear proyectos basados en JVM. Todo lo que se necesita hacer es escribir el código de la aplicación.


Nos permite seleccionar un proyecto, el lenguaje que queremos usar y agregar dependencias como herramientas de desarrollo, páginas web, etc. Para crear un proyecto, sólo debe hacer “*click*” en el botón *Generar* después de seleccionar las opciones en la pantalla.

El proyecto creado incluye la especificación para armarlo (*build*) en Gradle o con Maven (pom.xml). Incluye también una clase con el método *main()* que arranca la aplicación. La aplicación autoconfigure el contexto de la aplicación (*application context*) y configura (con “vacío”) las propiedades.

Se puede alcanzar en: <https://start.spring.io/>



SPRING INITIALIZR - II

 **spring**initializr

Project
☐ Gradle - Groovy
☐ Gradle - Kotlin
☒ **Maven**

Language
☒ **Java**
☐ Kotlin
☐ Groovy

Spring Boot
☐ 3.1.1 (SNAPSHOT)
☒ **3.1.0**
☐ 3.0.8 (SNAPSHOT)
☐ 3.0.7
☐ 2.7.13 (SNAPSHOT)
☐ 2.7.12

Project Metadata

Group

edu.frp.prog3

Artifact

demo

Name

demo

Description

Proyecto Demo en Spring Boot

Package name

edu.frp.prog3.demo

Packaging

☒ **Jar**
☐ War

Java

☒ **20**
☐ 17
☐ 11
☐ 8

Dependencies

ADD DEPENDENCIES... CTRL + B

No dependency selected

Pongan los nombres que quieran

Únicas opciones

GENERATE CTRL + ↵

EXPLORE CTRL + SPACE

SHARE...

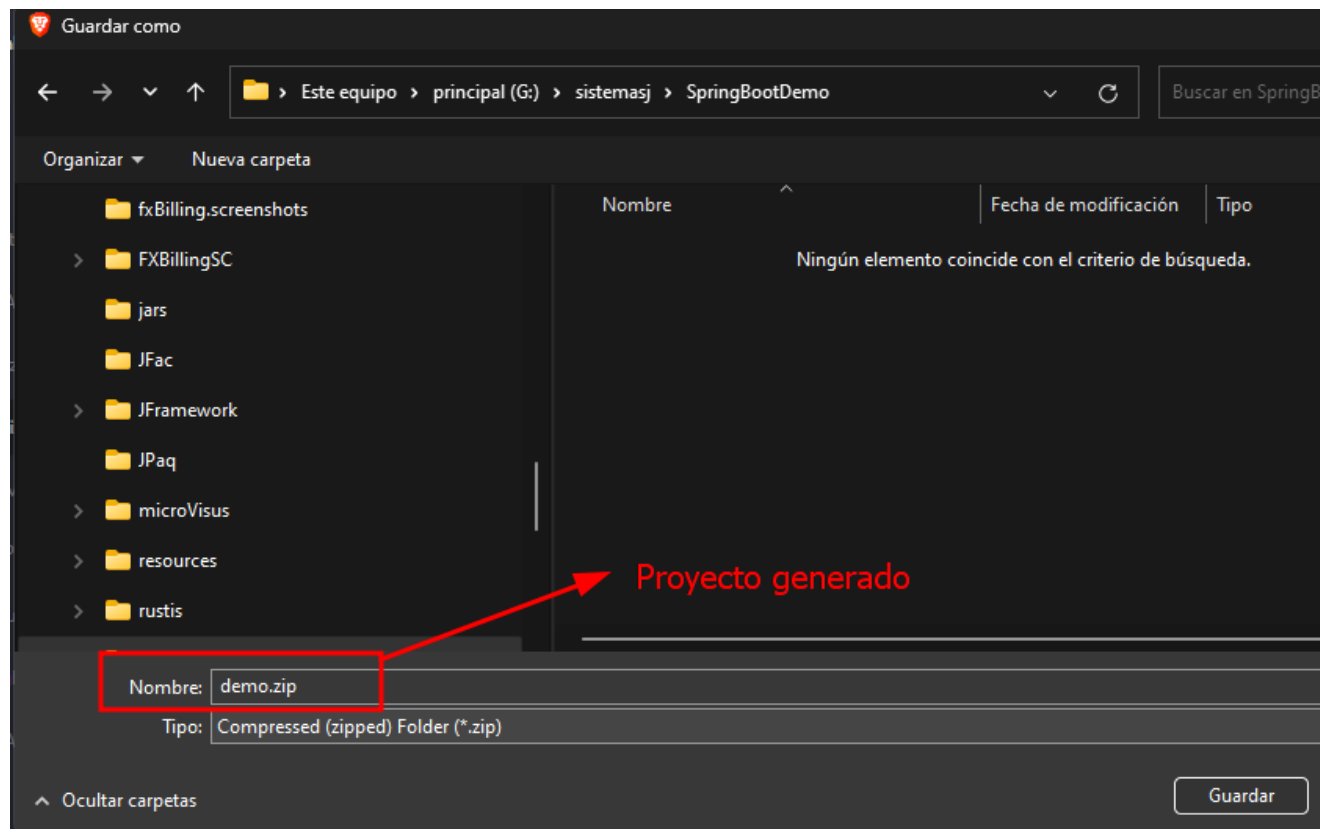


SPRING INITIALIZR - III

Paso 1: Hacemos “*click*” en el botón *Generate*.

Ahora comienza a empaquetar (*packing*) el proyecto en un archivo .zip y permite descargarlo.

Paso 2: Desempaquetamos el archivo ZIP.

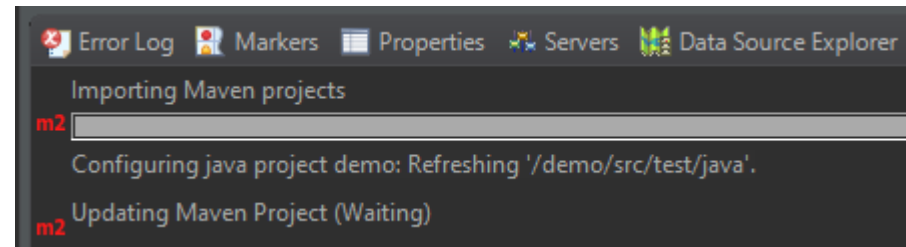
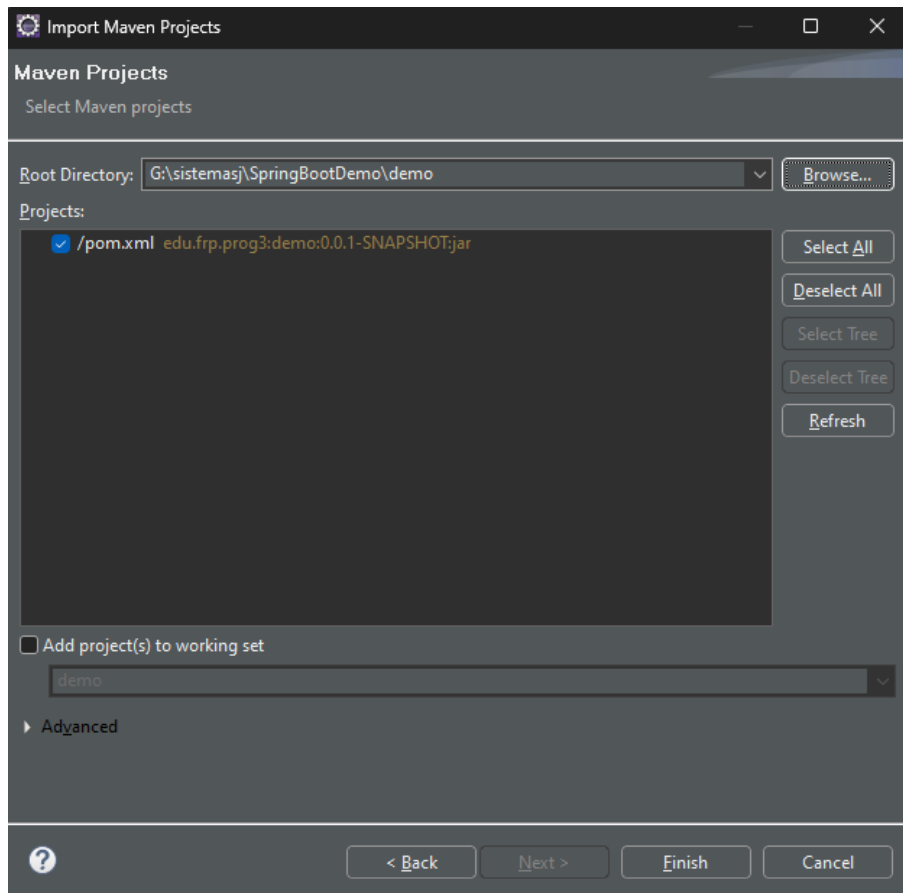




SPRING INITIALIZR - III

Paso 3: Importamos la carpeta creada en Eclipse

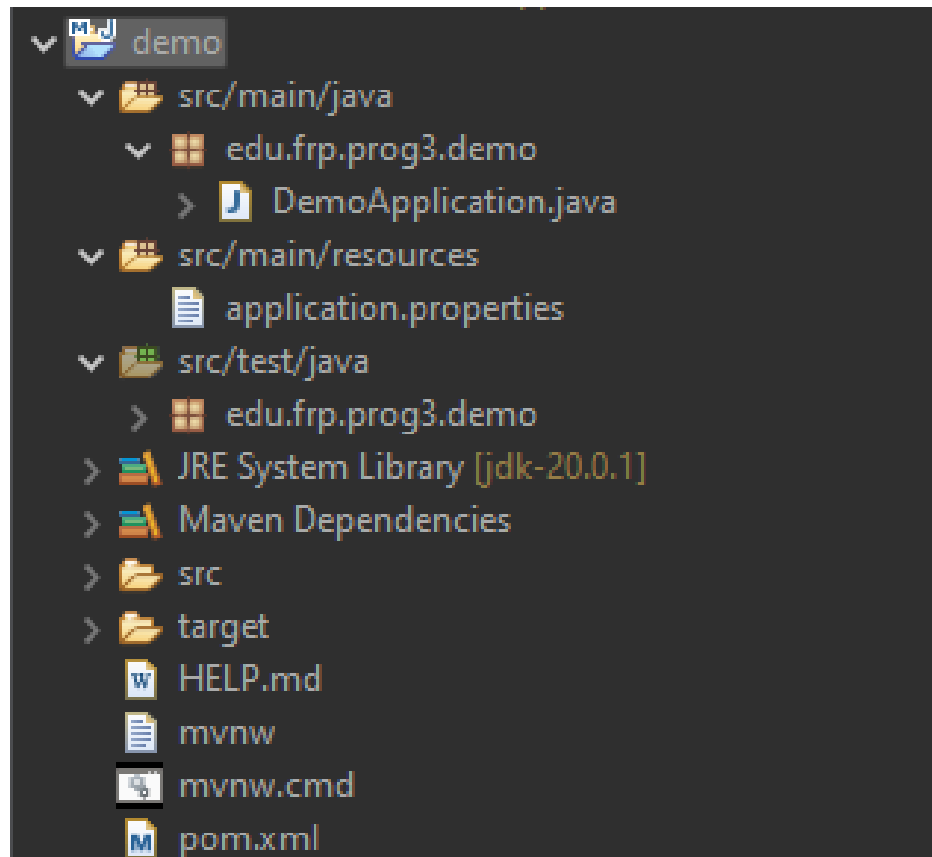
File -> Import -> Existing Maven Project -> Next -> Browse -> Seleccionar la carpeta creada con el proyecto -> Finish





SPRING INITIALIZR - III

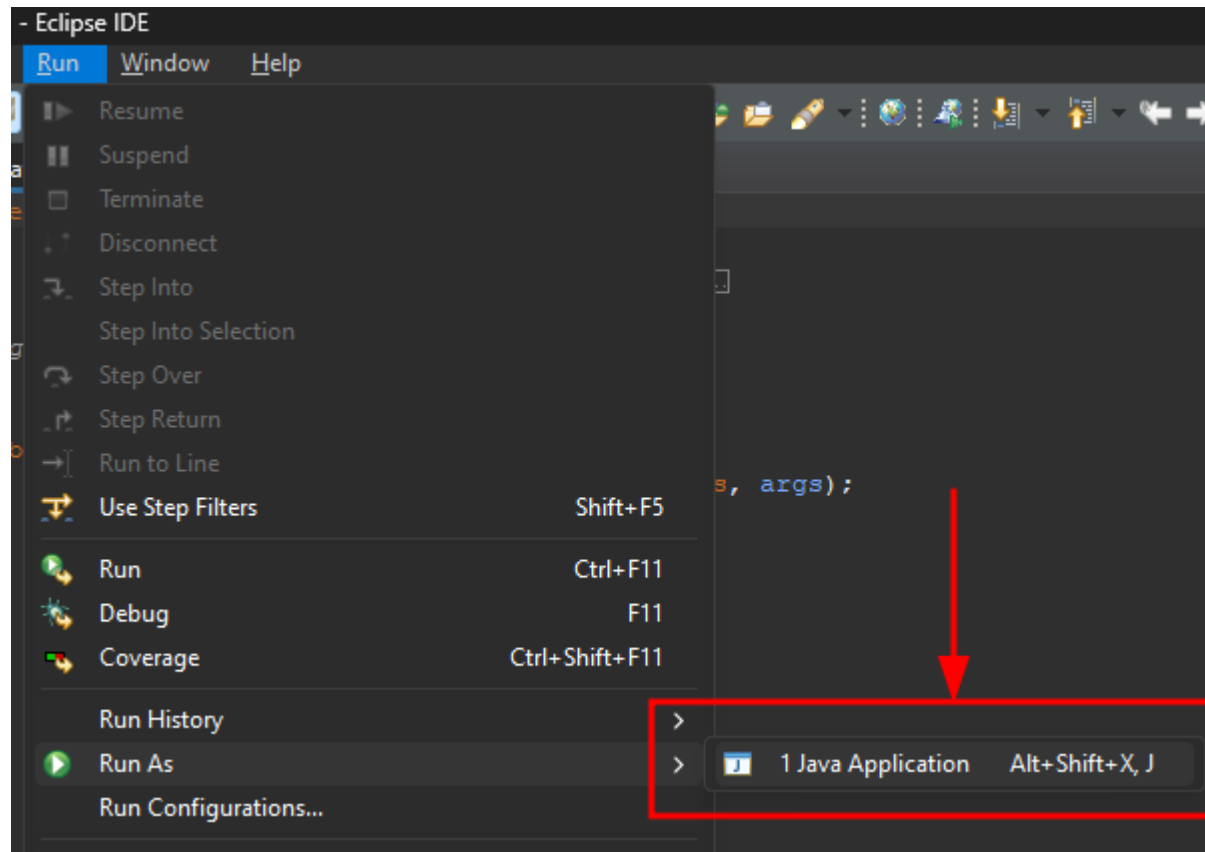
Toma un poco de tiempo en generar el proyecto. Si todo está bien, Podemos verlo en el **Package Explorer**.





SPRING INITIALIZR - IV

Paso 4: Ejecutemos nuestro archivo java.



[illegible]

```
2023-06-16T09:54:12.010-03:00 INFO 10804 --- [          main] edu.frp.prog3.demo.DemoApplication : Starting DemoApplication using Java 20.0.
2023-06-16T09:54:12.012-03:00 INFO 10804 --- [          main] edu.frp.prog3.demo.DemoApplication : No active profile set, falling back to 1
2023-06-16T09:54:12.362-03:00 INFO 10804 --- [          main] edu.frp.prog3.demo.DemoApplication : Started DemoApplication in 0.599 seconds
```



¿Qué nos falta?

- **RESTful:** teoría básica de estos servicios WEB
- **Integrar REST + Spring Boot** sencillo proyecto de Spring Boot con un servicio REST tipo “Hola Mundo”.
- **Armar la BD** una única tabla hecha en PostgreSQL
- **Armar el frontend con Vue.js** siguiendo el modelo de datos de la BD, armaremos las pantallas en Vue.js.
- **Armar el backend en Java** armaremos el modelo y la lógica de negocios en Java.
- **Pruebas finales** compilamos todos y ejecutamos nuestro CRUD.