

Introducción a los Frameworks en Java

¿Qué es un Framework?

- Un framework es un conjunto de bibliotecas y herramientas que proporciona una estructura genérica y predefinida para desarrollar aplicaciones. Un framework define el esqueleto de una aplicación y establece un flujo de control general, donde tu código se integra dentro de ese flujo.
- Inversión de Control (IoC): Uno de los principios clave en los frameworks es la Inversión de Control, donde el framework controla el flujo de la aplicación y llama al código del desarrollador en los momentos apropiados.

¿Qué es Java Spring Boot?



Java Spring Framework (Spring Framework) es un popular framework empresarial de código abierto que sirve para crear aplicaciones autónomas de producción que se ejecutan en una máquina virtual Java (JVM).

Java Spring Boot (Spring Boot) es una herramienta que acelera y simplifica el desarrollo de microservicios y aplicaciones web con Spring Framework gracias a tres funciones principales:

- Configuración automática
- Un enfoque de configuración obstinado
- La capacidad de crear aplicaciones autónomas

Estas características, combinadas, conforman una herramienta que le permite configurar una aplicación basada en Spring con el mínimo de instalación y configuración.

¿Qué aporta Spring Boot a Spring Framework?



- **Configuración Automática:**

Spring Boot inicializa aplicaciones con configuraciones predefinidas automáticamente, eliminando la necesidad de configurar manualmente dependencias y paquetes. Esto acelera el desarrollo y reduce errores, permitiendo ajustes una vez completada la inicialización.

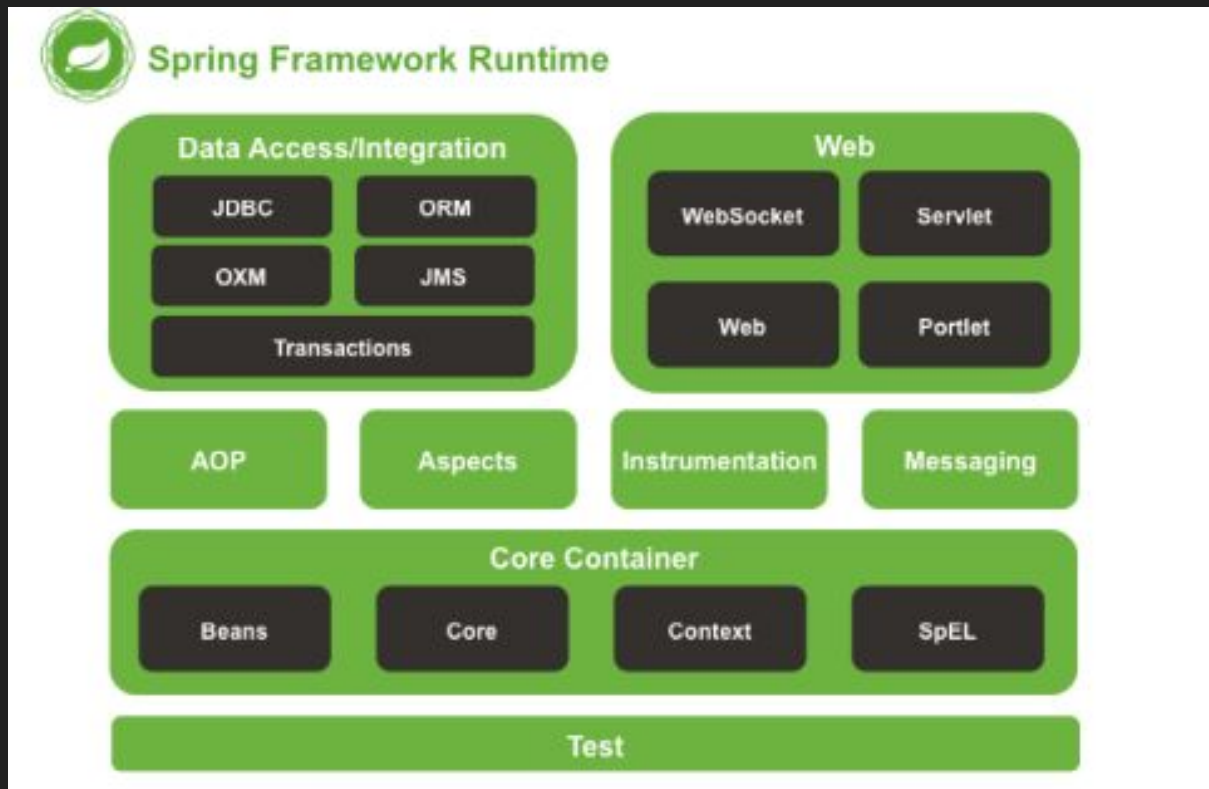
- **Enfoque Obsesivo:**

Spring Boot selecciona y configura automáticamente las dependencias necesarias para el proyecto, usando "Spring Starters" que se eligen según las necesidades del proyecto. Esto facilita la configuración inicial sin necesidad de programar.

- **Aplicaciones Autónomas:**

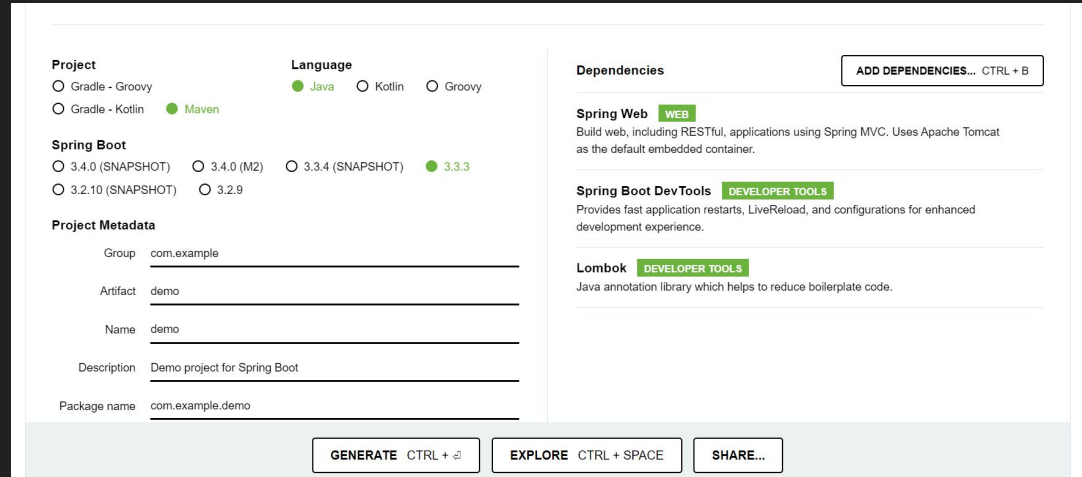
Spring Boot permite crear aplicaciones que se ejecutan por sí solas, integrando un servidor web como Tomcat directamente en la aplicación, lo que elimina la necesidad de un servidor externo. Esto permite ejecutar la aplicación fácilmente en cualquier plataforma.

Módulos de Spring



Spring Initializr


- Sirve para crear aplicaciones Spring Boot de forma rápida y sencilla
- Hay tres formas de utilizarlo
 - A través de la web
 - Plugin instalado en el IDE
 - Spring Boot Cli



The screenshot shows the Spring Initializr web form. It is divided into several sections:
1. **Project**: Radio buttons for **Gradle - Groovy**, **Gradle - Kotlin**, and **Maven** (selected).
2. **Language**: Radio buttons for **Java** (selected), **Kotlin**, and **Groovy**.
3. **Spring Boot**: Radio buttons for versions **3.4.0 (SNAPSHOT)**, **3.4.0 (M2)**, **3.3.4 (SNAPSHOT)**, **3.3.3** (selected), and **3.2.10 (SNAPSHOT)**.
4. **Project Metadata**: Fields for **Group** (com.example), **Artifact** (demo), **Name** (demo), **Description** (Demo project for Spring Boot), and **Package name** (com.example.demo).
5. **Dependencies**: A section with a button **ADD DEPENDENCIES... CTRL + B** and three dependency cards:
 - **Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - **Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
 - **Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.
6. **Bottom Bar**: Three buttons: **GENERATE CTRL + G**, **EXPLORE CTRL + SPACE**, and **SHARE...**

```
>spring initializr
```

Spring Initializr: Create a Maven Project...

recently used 

Spring Initializr: Add Starters...

other commands

Spring Initializr: Create a Gradle Project...

¿Qué es Maven?



Maven es una herramienta de gestión de proyectos basada en POM (Project Object Model). Se encarga de la compilación, documentación, reporte y despliegue de proyectos en Java. Maven facilita la gestión de dependencias, lo que significa que puedes agregar bibliotecas externas a tu proyecto de manera sencilla y automática.

Manejo de Dependencias con Maven

El Archivo pom.xml:

El pom.xml es el archivo de configuración central en Maven. Contiene información del proyecto y detalles de configuración que Maven utiliza para construir el proyecto.

Componentes Clave del pom.xml:

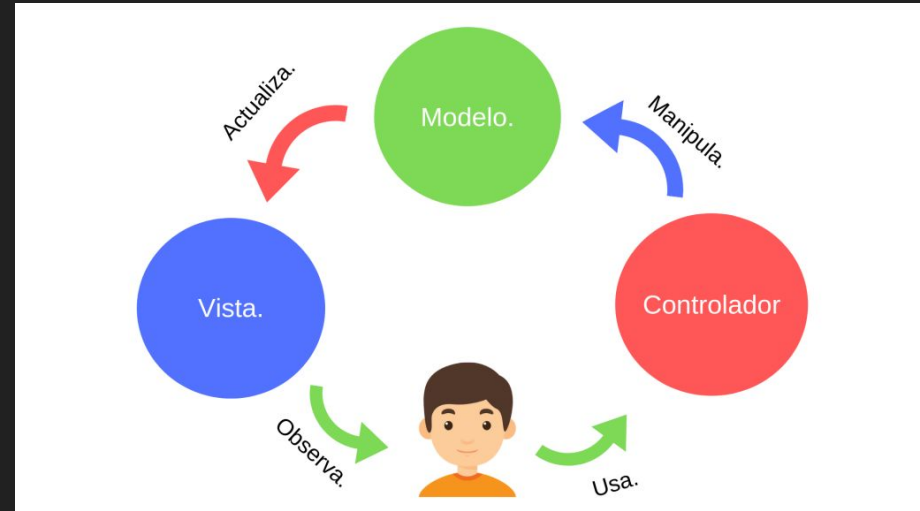
- **groupId:** Define el grupo al que pertenece el proyecto (por ejemplo, un dominio inverso).
- **artifactId:** Nombre del proyecto o módulo.
- **version:** Versión del proyecto.
- **dependencies:** Sección donde se listan las dependencias del proyecto. Cada dependencia tiene un groupId, un artifactId, y una version.
- **build:** Configura cómo se compila y empaqueta el proyecto. Aquí se pueden añadir plugins, como el plugin de Spring Boot.

Patrón de diseño MVC

El patrón MVC es uno de los patrones de diseño más comunes en aplicaciones web. Divide la aplicación en tres componentes principales:

Model (Modelo): Representa los datos y la lógica del negocio. En un proyecto Spring Boot, los modelos suelen ser clases anotadas con `@Entity` para representar tablas en una base de datos.

View (Vista): Se encarga de la presentación de la información al usuario. En Spring Boot, las vistas suelen ser archivos HTML, Thymeleaf o JSP que se renderizan en el navegador del usuario.



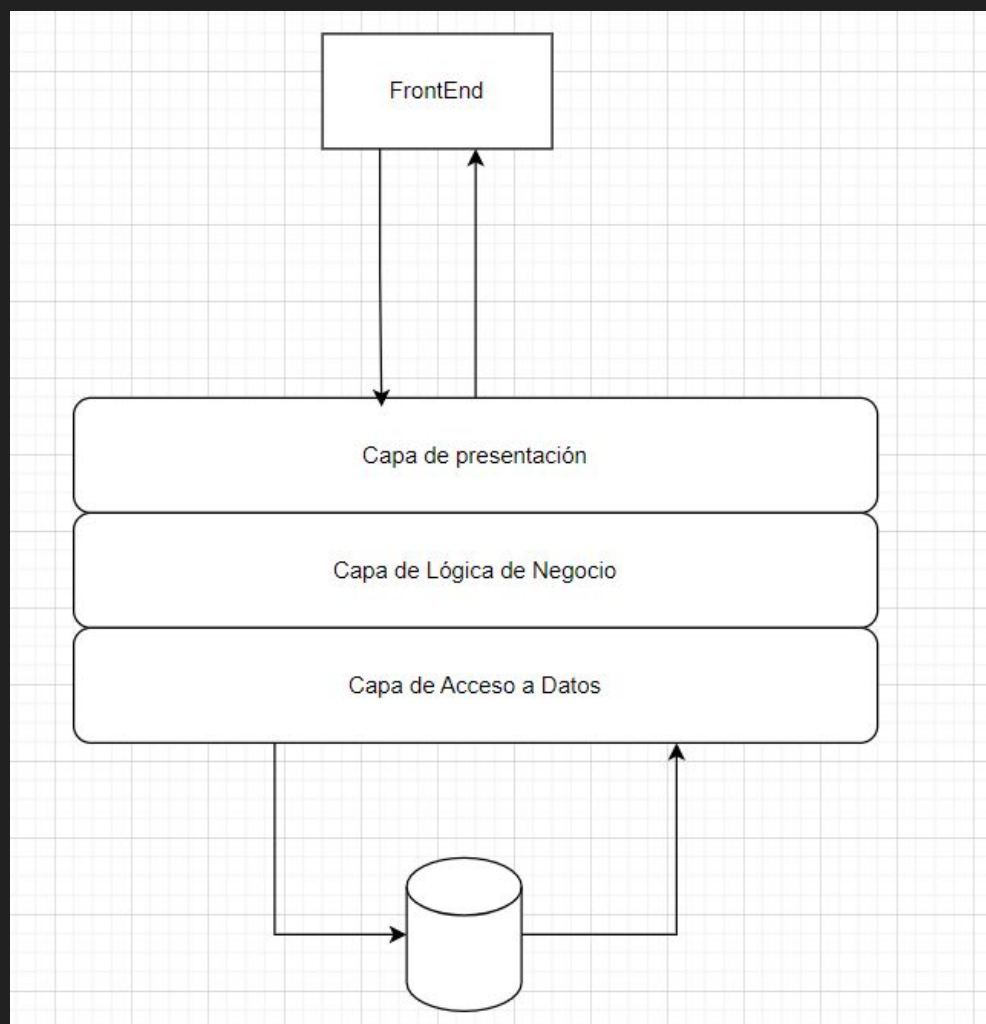
Controller (Controlador): Gestiona las solicitudes del usuario, interactúa con el modelo y selecciona la vista adecuada para devolver una respuesta. En Spring Boot, los controladores se definen con la anotación `@Controller` y contienen métodos anotados con `@RequestMapping`, `@GetMapping`, `@PostMapping`, etc., que manejan las diferentes rutas de la aplicación.

Arquitectura de 3 Capas

La arquitectura de 3 capas es un modelo de diseño que organiza el código en tres capas separadas: Presentación, Lógica de Negocio y Acceso a Datos. Este enfoque mejora la modularidad, facilita el mantenimiento y promueve la reutilización de código.

Capas

- Capa de Presentación: Interactúa con el usuario y presenta los datos. Incluye vistas y controladores que manejan las solicitudes del usuario.
- Capa de Lógica de Negocio: Procesa los datos según las reglas de negocio. Incluye servicios que coordinan las operaciones y aplican la lógica empresarial.
- Capa de Acceso a Datos: Maneja la comunicación con la base de datos. Incluye clases que realizan operaciones CRUD (Crear, Leer, Actualizar, Eliminar).



Controller

El **Controller** es el punto de entrada para las solicitudes HTTP en una aplicación Spring Boot. Los controladores están anotados con `@RestController`, que combina las anotaciones `@Controller` y `@ResponseBody` y establece la entrada y salida de datos en formato JSON. Esto significa que el controlador gestionará las peticiones y respuestas en formato JSON, que es el formato más común en APIs RESTful.

Request Body y Request Header

- La anotación `@RequestBody` nos permite establecer a que formato (tipo de dato) queremos que des-serialice los datos que llegan al endpoint
- `@RequestHeader` nos permite obtener un encabezado de la petición http a partir de especificar su nombre
- Una alternativa a estas dos anotaciones es utilizar `HttpEntity` que nos representa la petición Http completa (body y encabezados)

Construcción de endpoints

- @RequestMapping
- @GetMapping
- @PostMapping
- @PutMapping
- @PatchMapping
- @DeleteMapping
- @PathVariable
- @RequestParam

Service

El Service es la capa de la lógica de negocio. Aquí es donde se ejecutan las reglas de negocio y las interacciones con los repositorios (acceso a la base de datos). Las clases de servicio están anotadas con `@Service`.

Formas de hacer inyección de dependencia

Por constructor

```
@Controller no usages  
public class PruebaController {  
  
    private final PruebaDao preubaDao; 2 usages  
  
    public PruebaController(PruebaDao preubaDao) { no usages  
        this.preubaDao = preubaDao;  
    }  
}
```


Formas de hacer inyección de dependencia

Por setter

```
@Controller no usages
public class PruebaController {

    private PruebaDao preubaDao; 2 usages

    @Autowired no usages
    public void setPruebaDao(PruebaDao preubaDao){
        this.preubaDao = preubaDao;
    }
}
```

Formas de hacer inyección de dependencia

Por anotación

```
@Controller no usages  
public class PruebaController {  
  
    @Autowired no usages  
    private PruebaDao pruebaDao;
```