



Módulo 2.- Desarrollo Back-End con Spring Boot

- Arquitectura de microservicios: conceptos y diseño.
- Introducción a Spring Boot y sus componentes.
- Controladores REST y manejo de rutas.
- Inyección de dependencias con Spring Framework.
- Creación de microservicios orientados al dominio del MSGG.

Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

🎯 ¿Qué es la arquitectura de microservicios?

Es un estilo de arquitectura que **descompone una aplicación monolítica en servicios pequeños, independientes y desplegables por separado**, donde cada uno representa una **unidad funcional autónoma** dentro del sistema.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

"En lugar de construir una aplicación grande y única (monolito), la dividimos en muchas piezas pequeñas (microservicios), como si fueran legos. Cada pieza hace una cosa específica, y todas se comunican entre sí para formar el sistema completo."



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Ejemplo en el mundo real:

Imagina una app de e-commerce:

- Servicio de Usuarios
- Servicio de Productos
- Servicio de Carrito
- Servicio de Pagos

Cada uno puede vivir en un servidor distinto, ser creado por diferentes equipos y escalar de forma autónoma.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Principios clave

- Descomposición por dominio
Cada microservicio representa una función del negocio (ej: Usuarios, Productos, Pagos).
- Independencia de despliegue
Los servicios pueden ser desarrollados, testeados y desplegados sin afectar a los demás.
- Base de datos por servicio (opcional)
Cada servicio gestiona su propia base de datos, evitando el acoplamiento a nivel de datos.
- Comunicación a través de APIs
La interacción entre microservicios ocurre típicamente a través de REST (HTTP) o mensajería (RabbitMQ, Kafka).



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

- **Descomposición por dominio**

Cada microservicio representa una **unidad funcional del negocio**. En lugar de tener una sola aplicación que maneja todo (usuarios, productos, pagos, reportes, etc.), se divide la lógica en componentes autónomos **basados en el modelo de negocio**.



Ejemplo:

- En un sistema de ventas:
- `UsuarioService` → gestiona autenticación y datos del cliente
 - `ProductoService` → administra catálogo y stock
 - `PagoService` → procesa transacciones
 - `PedidoService` → maneja órdenes y entregas

Cada uno de estos refleja un **subdominio del negocio** (esto se llama "Domain-Driven Design", DDD).



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

- **Independencia de despliegue**

Cada microservicio puede ser **desarrollado, probado y desplegado por separado**. No necesitas detener o reconstruir toda la aplicación para actualizar un solo componente.

Beneficios:

- Puedes actualizar PagoService sin reiniciar todo el sistema.
- Si ProductoService falla, el resto puede seguir funcionando.
- Cada equipo puede trabajar en su servicio sin interferir con otros.

Ejemplo:

Una nueva funcionalidad de descuento se implementa solo en PedidoService y se despliega en minutos, sin impactar a los servicios de usuarios o pagos.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

- **Base de datos por servicio (opcional)**

Cada microservicio puede tener su propia base de datos, aislada del resto. Esto refuerza el principio de encapsulamiento: el servicio controla sus datos y nadie más accede directamente.

¿Por qué es opcional?

Aunque es lo ideal en microservicios, en entornos más simples o en fase inicial, algunos servicios pueden compartir la base. Pero a medida que el sistema crece, lo óptimo es que cada uno tenga su propia persistencia.

Ejemplo:

UsuarioService usa una base de datos PostgreSQL.

ProductoService usa MongoDB.

PagoService usa una base relacional con alto rendimiento (como Oracle o MySQL).

Esto permite elegir la tecnología de almacenamiento óptima para cada dominio



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

- **Comunicación a través de APIs**

Como cada servicio es independiente, **no se comunican por llamadas internas**, sino a través de **APIs públicas o sistemas de mensajería**.

Formas comunes de comunicación:

- **REST API (HTTP)**: PedidoService consulta productos a través de GET /productos/{id}
- **Mensajería asincrónica**: PagoService publica un evento “pago realizado” en RabbitMQ o Kafka, y PedidoService lo escucha.

REST vs Mensajería:

REST (sincrónico)	Mensajería (asincrónico)
Llamada directa	Se publica en una cola
Respuesta inmediata	Procesamiento desacoplado
Usa HTTP	Usa brokers como RabbitMQ/Kafka
Más simple de implementar	Más escalable y resiliente



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Conclusión

Cada uno de estos principios **fomenta modularidad, autonomía, escalabilidad y robustez** en sistemas complejos. La clave es diseñar los microservicios de forma que reflejen el negocio real y puedan evolucionar sin bloquearse entre sí.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Arquitectura de microservicios: conceptos y diseño.

E Comparación: Monolito vs Microservicios

Característica	Monolito	Microservicios
Estructura	Una sola aplicación	Múltiples servicios pequeños
Despliegue	Todo junto	Independiente por servicio
Escalabilidad	Vertical	Horizontal (por componente)
Dependencias	Muy acoplado	Bajo acoplamiento
Mantenimiento	Costoso a largo plazo	Mayor complejidad inicial, pero modular
Tecnologías	Un único stack	Puede variar por servicio



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Diseño de un sistema basado en microservicios

1. Identificación de dominios

Antes de comenzar a codificar, se debe **entender el negocio y dividirlo en contextos lógicos independientes**, llamados *Bounded Contexts*, usando el enfoque de **Domain-Driven Design (DDD)**.

¿Cómo se aplica?

- Se analizan los procesos y reglas del negocio.
- Se define qué hace cada parte del sistema y dónde están los límites naturales.

Ejemplo para e-commerce:

- AuthService: autenticación y gestión de usuarios
- ProductService: catálogo de productos
- OrderService: gestión de pedidos
- PaymentService: procesamiento de pagos

Tip: Cada uno de estos es un microservicio con lógica y base de datos propias.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Diseño de un sistema basado en microservicios

2. Contratos y APIs bien definidos

Cada microservicio debe tener una API clara, estable y documentada que defina cómo otros servicios pueden comunicarse con él.

¿Cómo se logra?

- Se expone una API REST con controladores @RestController
- Se documenta la API usando Swagger o OpenAPI, lo cual permite:
 - Saber qué endpoints existen
 - Qué datos esperan
 - Qué respuestas devuelven

Herramientas:

- Springdoc OpenAPI
- Swagger UI

Esto facilita el desarrollo en paralelo entre equipos.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Diseño de un sistema basado en microservicios

3. Gestión de comunicación

Tipos de comunicación:

a) Directa (sincrónica):

- Usando **HTTP REST**
- El cliente espera la respuesta del servidor
- Útil para operaciones simples como GET /productos

b) Indirecta (asincrónica):

- Usando **mensajería** (RabbitMQ, Kafka)
- El emisor **publica un mensaje** que otros servicios escuchan
- Ideal para flujos desacoplados y alta disponibilidad (ej: “pago recibido” → se despacha el pedido)

Elegir bien el tipo de comunicación es **clave para la escalabilidad y resiliencia**.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Diseño de un sistema basado en microservicios

4. Resiliencia y tolerancia a fallos

¿Por qué es importante?

En un sistema distribuido, **los fallos son inevitables** (un microservicio puede estar caído, lento o sobrecargado). Hay que diseñar con eso en mente.

Patrones comunes:

- **Circuit Breaker**: si un servicio falla varias veces, se “abre el circuito” y deja de llamarlo por un tiempo.
- **Retry**: intenta varias veces antes de fallar.
- **Timeout**: limita cuánto tiempo espera una respuesta.

Herramientas:

- Spring Cloud Resilience4J
- Netflix Hystrix (deprecated, pero aún usado en algunos lugares)

Esto mejora la **experiencia del usuario y la estabilidad del sistema**.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Diseño de un sistema basado en microservicios

5. Observabilidad

Es la capacidad de **entender qué está ocurriendo dentro del sistema**, incluso cuando tiene múltiples servicios en ejecución.

Incluye:

- **Logs centralizados** (Ej: ELK Stack – Elasticsearch, Logstash, Kibana)
- **Monitoreo de métricas** (Ej: Prometheus + Grafana)
- **Trazabilidad distribuida**: seguir una solicitud que pasa por varios microservicios
Ej: usando **Zipkin** o **Spring Cloud Sleuth**

Permite detectar cuellos de botella, errores y medir desempeño.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Diseño de un sistema basado en microservicios

En un sistema bien diseñado:

- Cada servicio tiene su dominio, API y base de datos
- Se comunican por REST o eventos
- Están preparados para fallar y recuperarse
- Y son completamente monitoreables



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

- Mantén tus servicios pequeños, enfocados y cohesionados.
- Evita la duplicación de lógica compartida (crear bibliotecas comunes si es necesario).
- Aplica pruebas automatizadas y despliegue continuo por servicio.
- Usa un **API Gateway** para centralizar entrada/salida.
- Desacopla lo más posible, pero mantén coherencia transaccional donde sea necesario (eventualmente con **eventos de dominio**).



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

1. Mantén tus servicios pequeños, enfocados y cohesionados

Cada microservicio debe encargarse de **una única responsabilidad de negocio**. No debe hacer “de todo un poco”, ni mezclarse con otras lógicas.

Beneficio:

- Facilita el entendimiento, mantenimiento y escalabilidad
- Permite que los equipos se especialicen por dominio

Ejemplo mal diseñado:

Un OrderService que también registra usuarios o procesa pagos.

Ejemplo bien diseñado:

- OrderService: solo maneja pedidos
- UserService: solo usuarios
- PaymentService: solo pagos



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

2. Evita la duplicación de lógica compartida

Cuando múltiples servicios necesitan la **misma lógica** (por ejemplo, validaciones de RUT, formato de fechas, reglas tributarias), duplicar esa lógica puede provocar **inconsistencias y errores difíciles de rastrear**.

Solución:

- Extraer esa lógica común en **bibliotecas compartidas** (Java libs en Maven repos)
 - Importar esa dependencia en los servicios que la necesiten
- 📌 Así evitas duplicación y mejoras la mantenibilidad del código.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

3. Aplica pruebas automatizadas y despliegue continuo por servicio

Cada microservicio debe tener su propio flujo de integración y despliegue continuo (CI/CD), incluyendo:

- Pruebas unitarias y de integración
- Construcción del artefacto (JAR/Docker)
- Despliegue automatizado (por ejemplo, con GitHub Actions, GitLab CI, Jenkins)

Beneficio:

- Reduce el riesgo de errores humanos
- Permite lanzamientos rápidos y seguros por cada servicio

Los servicios pueden evolucionar de forma independiente y con confianza.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

4. Usa un API Gateway para centralizar entrada/salida

Es una **puerta de entrada única** al sistema, que expone las APIs al mundo externo y enruta las solicitudes al microservicio correspondiente.

Beneficios del API Gateway:

- Control de seguridad (autenticación/autorización)
- Enrutamiento inteligente
- Limitación de tasa (rate limiting)
- Monitoreo y logging

Ejemplos:

- Spring Cloud Gateway
- Kong
- NGINX
- Zuul (legacy)

El cliente (app móvil o navegador) nunca se comunica directamente con los microservicios.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

5. Desacopla lo más posible, pero mantén coherencia transaccional cuando sea necesario

Idealmente, los servicios deben estar lo más desacoplados posible para lograr independencia. Sin embargo, algunas operaciones críticas (como registrar una orden y su pago) requieren coherencia de datos.

Solución:

Usar eventos de dominio para comunicar cambios entre servicios (eventual consistency)

O aplicar patrones como Sagas si la transacción abarca múltiples servicios

El desafío está en encontrar el equilibrio entre desacoplamiento y la necesidad de consistencia.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Arquitectura de microservicios: conceptos y diseño.

Buenas prácticas en microservicios

Cierre:

Diseñar microservicios no es solo dividir en piezas, sino hacerlo de forma estratégica: cada servicio debe ser pequeño, autónomo, testeado, observable y con reglas claras de comunicación.



Módulo 2.- Desarrollo Back-End con Spring Boot

0 Introducción a Spring Boot y sus componentes.

¿Qué es Spring Boot?

Spring Boot es un **framework de código abierto** basado en **Spring Framework(1)**, diseñado para facilitar y acelerar la creación de aplicaciones web y microservicios en Java.

Permite crear aplicaciones **autónomas, listas para producción**, que se ejecutan directamente sobre la **Máquina Virtual de Java (JVM)**, sin necesidad de configuraciones complejas ni servidores externos.

Gracias a su enfoque de "**convención sobre configuración**", Spring Boot reduce la cantidad de código repetitivo y facilita el desarrollo con herramientas ya preconfiguradas.

(1) **Spring Framework** es un **framework de desarrollo en Java** que proporciona una estructura completa para crear aplicaciones empresariales de forma flexible, modular y mantenible.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Introducción a Spring Boot y sus componentes.

¿Para qué sirve Spring Boot?

- Para desarrollar un proyecto en menos tiempo
- Crear un código más limpio y consistente
- Reutilizar herramientas o módulos
- Automatizar tareas

“Spring Boot es ideal para construir microservicios porque te permite levantar servicios independientes, ligeros, con REST APIs, en minutos. Además, puedes agregar módulos como Spring Cloud, JWT, etc., según el crecimiento del proyecto.”



Módulo 2.- Desarrollo Back-End con Spring Boot



Introducción a Spring Boot y sus componentes.

Documentación

- <https://docs.spring.io/spring-boot/>
- <https://spring.io/projects/spring-boot#learn>



Módulo 2.- Desarrollo Back-End con Spring Boot

O Introducción a Spring Boot y sus componentes.

Maven

¿Para qué sirve?

Simplifica y automatiza la **construcción** y **gestión** de tus proyectos de software.

¿Cómo funciona?

Se basa en un archivo de configuración clave: el **pom.xml** (Project Object Model).

Beneficios Clave

- **Gestión de Dependencias:** Descarga y organiza las librerías que tu proyecto necesita.
¡Olvídate de buscarlas manualmente!
- **Construcción Estándar:** Compila, prueba y empaqueta tu código de forma automática y consistente.
- **Ciclo de Vida Estandarizado:** Define un proceso de construcción predecible y uniforme (compilación, pruebas, etc.).
- **Menos Configuración:** Promueve "convención sobre configuración", reduciendo el trabajo manual.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Introducción a Spring Boot y sus componentes.

¿Qué hace Spring Boot “mágicamente”?

Spring Boot aplica el principio de "**convención sobre configuración**", lo cual:

- Auto-configura:
 - Servidor embebido Tomcat
 - Jackson para convertir a JSON
 - Controladores REST con `@RestController`
- Escanea automáticamente beans usando `@ComponentScan`
- Crea el `ApplicationContext` con solo poner `@SpringBootApplication`.

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

`@SpringBootApplication`:

- Equivale a `@Configuration`, `@EnableAutoConfiguration`, y `@ComponentScan`.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Introducción a Spring Boot y sus componentes.

¿Qué incluye automáticamente Spring Boot?

Starter	¿Qué trae?
spring-boot-starter-web	Spring MVC + Tomcat embebido + Jackson
spring-boot-starter-data-jpa	JPA + Hibernate + validaciones
spring-boot-starter-security	Spring Security preconfigurado
spring-boot-devtools	Recarga automática de cambios
spring-boot-starter-test	JUnit + Mockito + Spring Test



Módulo 2.- Desarrollo Back-End con Spring Boot

O Introducción a Spring Boot y sus componentes.

Los “starters” más usados

Starter	¿Qué incluye?	Para qué sirve
spring-boot-starter-web	Spring MVC, Tomcat, Jackson	APIs REST
spring-boot-starter-data-jpa	JPA, Hibernate	Persistencia
spring-boot-starter-test	JUnit, Mockito	Pruebas
spring-boot-devtools	Recarga automática en desarrollo	Desarrollo



Módulo 2.- Desarrollo Back-End con Spring Boot

O Introducción a Spring Boot y sus componentes.

¿Cómo se diferencia de usar Spring tradicional?

Aspecto	Spring tradicional	Spring Boot
Configuración	Manual, XML o JavaConfig	Automática
Servidor	Externo (Tomcat, Jetty)	Embebido
Estructura	Fragmentada	Unificada por starters
Inversión de tiempo inicial	Alto	Bajo

“Con Spring tradicional, tardabas horas solo en configurar. Con Spring Boot, estás desarrollando tu primera API en minutos.”



Estructura típica de un proyecto Spring Boot

```
com.ejemplo.miapp
└── controller
    └── ProductoController.java
└── service
    └── ProductoService.java
└── repository
    └── ProductoRepository.java
└── model
    └── Producto.java
└── MiAppApplication.java
```

Spring escanea desde donde está la clase `@SpringBootApplication` hacia abajo en el árbol de paquetes.

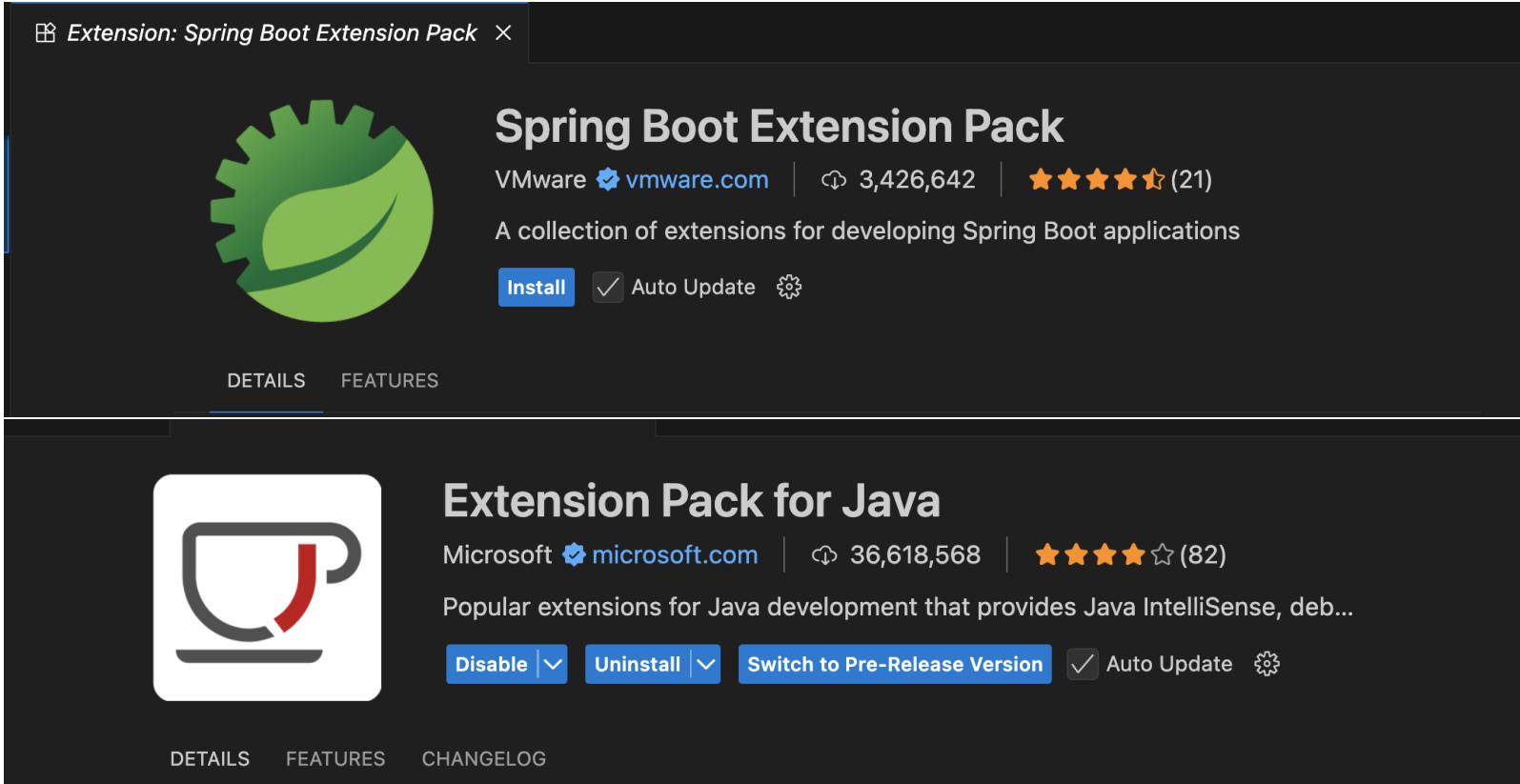
Diagrama



Ejemplo: Primer controlador

Módulo 2.- Desarrollo Back-End con Spring Boot

Introducción a Spring Boot y sus componentes.



The screenshot shows the 'Spring Boot Extension Pack' page in the VMware Marketplace. The page features a large green gear icon with a leaf-like pattern. The title 'Spring Boot Extension Pack' is displayed prominently. Below it, the developer information 'VMware' and 'vmware.com' is shown, along with the download count '3,426,642' and a rating of '★★★★★ (21)'. A descriptive text states 'A collection of extensions for developing Spring Boot applications'. At the bottom, there are 'Install' and 'Auto Update' buttons. Below the main card, there is another card for the 'Extension Pack for Java' by Microsoft, featuring a coffee cup icon, a download count of '36,618,568', a rating of '★★★★★ (82)', and options to 'Disable', 'Uninstall', 'Switch to Pre-Release Version', and 'Auto Update'.



Welcome X

EXPLORER ...

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or create a new Java project by clicking the button below.

Create Java Project

Visual Studio Code

Editing evolved

Start

- New File...
- Open...
- Clone Git Repository...
- Connect to...
- New Workspace with Copilot...

Recent

- CursoJavaAvanzado ~/Relatorias
- acs-clean ~/Proyectos
- src ~/Relatorias/CursoJavaAvanzado
- irsFrontend ~/sbs
- Proyectos ~/Documents/GitHub
- More...

Walkthroughs

- Get started with VS Code**
Customize your editor, learn the basics, and start coding
- Getting Started with Spring Boot in V...** New
- Get Started with Java Development New
- Learn the Fundamentals
- GitHub Copilot Updated

Search ...

OUTLINE



Seleccionar

The screenshot shows the Visual Studio Code (VS Code) interface with the "Walkthrough: Spring Boot Extension Pack" extension active. The left sidebar displays the "EXPLORER" view with a message: "NO FOLDER OPENED" and "You have not yet opened a folder." It includes buttons for "Open Folder", "Clone Repository", and "Create Java Project". Below these are instructions for using Git and source control, and links to Java project creation options. The main content area features a large title "Getting Started with Spring Boot..." and a subtitle "An Overview of the Spring Tools in VS Code to get started and to work with...". A callout box highlights the "Start with a Spring Boot project" section, which describes using the Spring Initializr integration to create new projects. A yellow arrow points from this section to the "Create New Project" button. To the right, a modal window titled "Spring Initializr: Choose dependencies" shows two selected dependencies: "Spring Web" and "Spring Boot DevTools".

EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, [add a folder](#) instead.

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or create a new Java project by clicking the button below.

Create Java Project

Get Started with Spring Boot...

An Overview of the Spring Tools in VS Code to get started and to work with...

Start with a Spring Boot project

The easiest way to create new Spring Boot projects in VS Code is to use the Spring Initializr integration. Open the command palette, search for *Spring Initializr* and create a new project from there.

Create New Project

You can also start with a sample project to try the full features. Below button helps you try the *Spring PetClinic* sample project with one-click.

Open Sample Project

Explore your projects

Spring Initializr: Choose dependencies

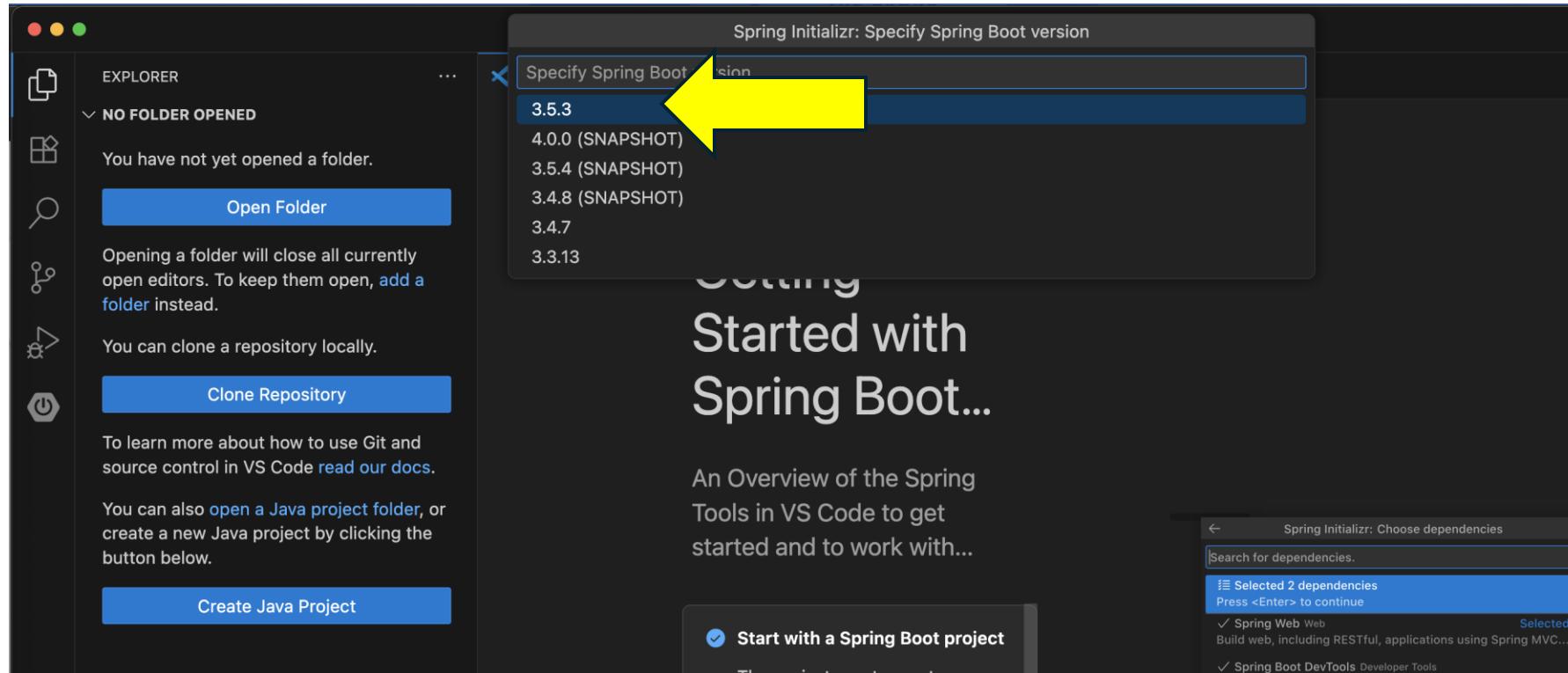
Selected 2 dependencies
Press <Enter> to continue

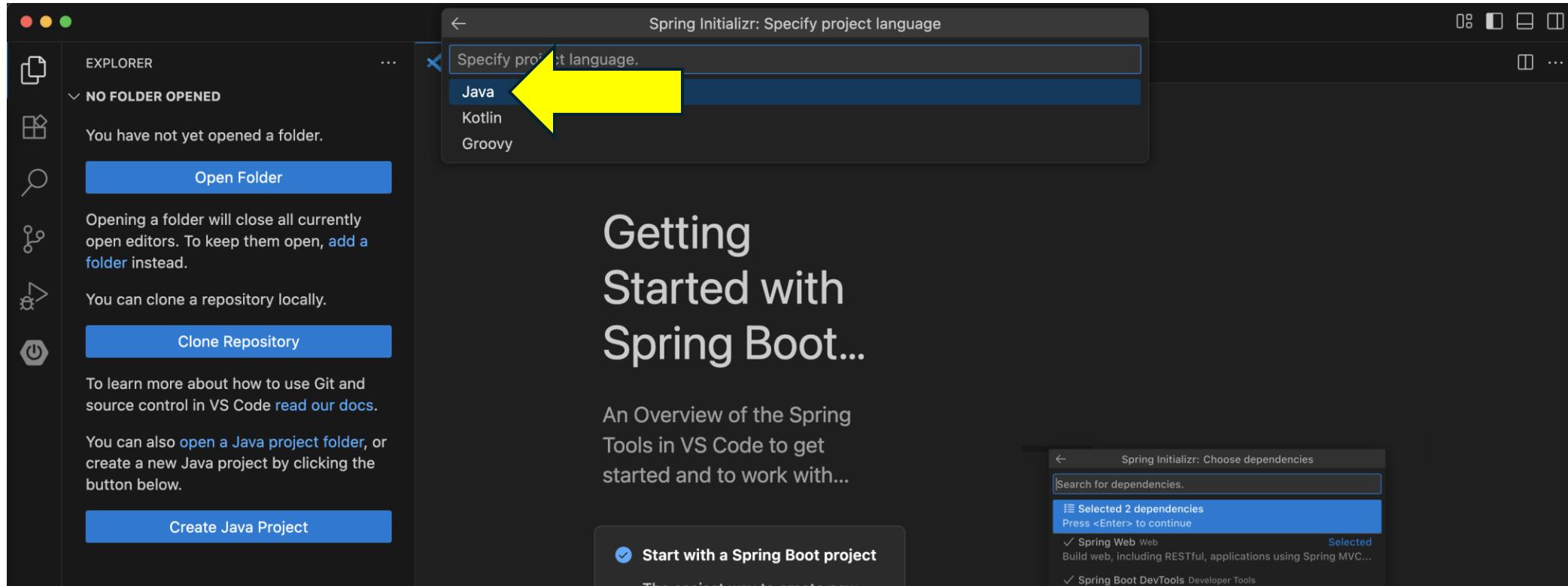
✓ Spring Web Web Selected
Build web, including RESTful, applications using Spring MVC...

✓ Spring Boot DevTools Developer Tools

Lombok Developer Tools







EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, [add a folder instead](#).

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or create a new Java project by clicking the button below.

Create Java Project

Spring Initializr: Input Group Id

com.example

Input Group Id for your project. (Press 'Enter' to confirm or 'Escape' to cancel)

GO BACK

Getting Started with Spring Boot...

An Overview of the Spring Tools in VS Code to get started and to work with...

Spring Initializr: Choose dependencies

Search for dependencies.

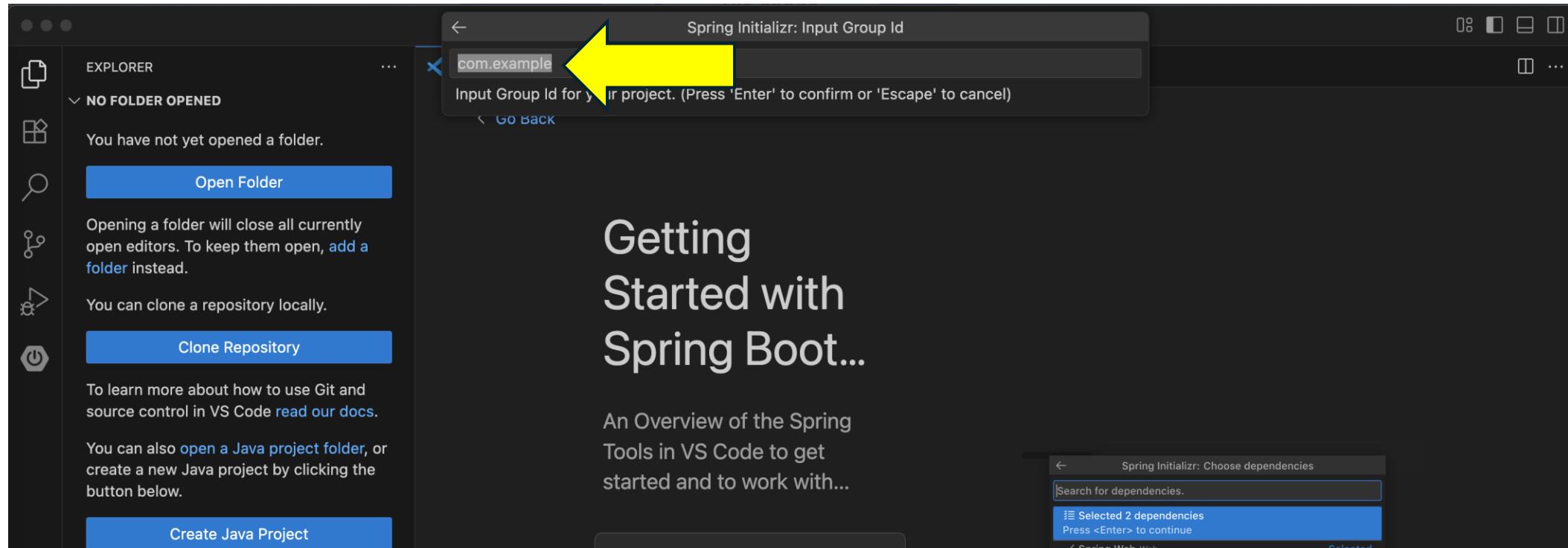
Selected 2 dependencies

Press <Enter> to continue

✓ Spring Web web Selected

Id Group es el administrador de dependencias de proyectos.
Punto importante cuando toque empaquetar





The screenshot shows the Spring Initializr interface within the VS Code extension. A yellow arrow points to the input field where 'com.example' has been typed. The interface includes sections for 'EXPLORER', 'NO FOLDER OPENED', 'Clone Repository', 'Create Java Project', and 'Getting Started with Spring Boot...'.

Spring Initializr: Input Group Id

Input Group Id for your project. (Press 'Enter' to confirm or 'Escape' to cancel)

com.example

← Go Back

EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, add a folder instead.

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or create a new Java project by clicking the button below.

Create Java Project

Getting Started with Spring Boot...

An Overview of the Spring Tools in VS Code to get started and to work with...

Spring Initializr: Choose dependencies

Search for dependencies.

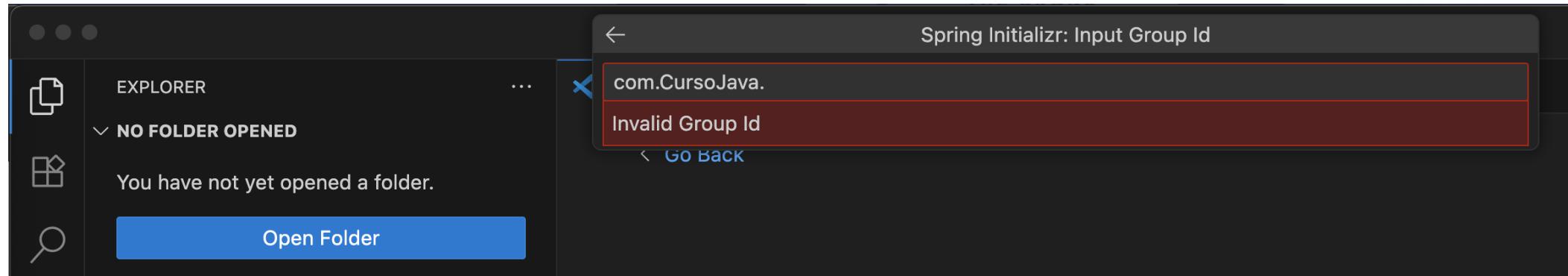
Selected 2 dependencies

Press <Enter> to continue

✓ Spring Web web Selected

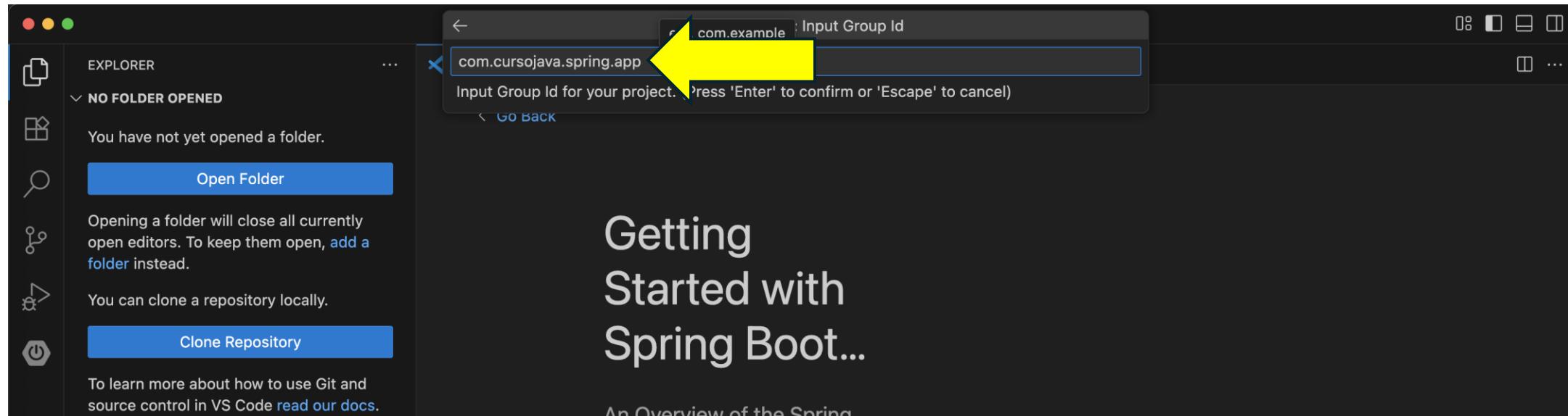
Id Group es el administrador de dependencias de proyectos.
Punto importante cuando toque empaquetar.
Recordando que debemos seguir un estilo de estructura de directorio





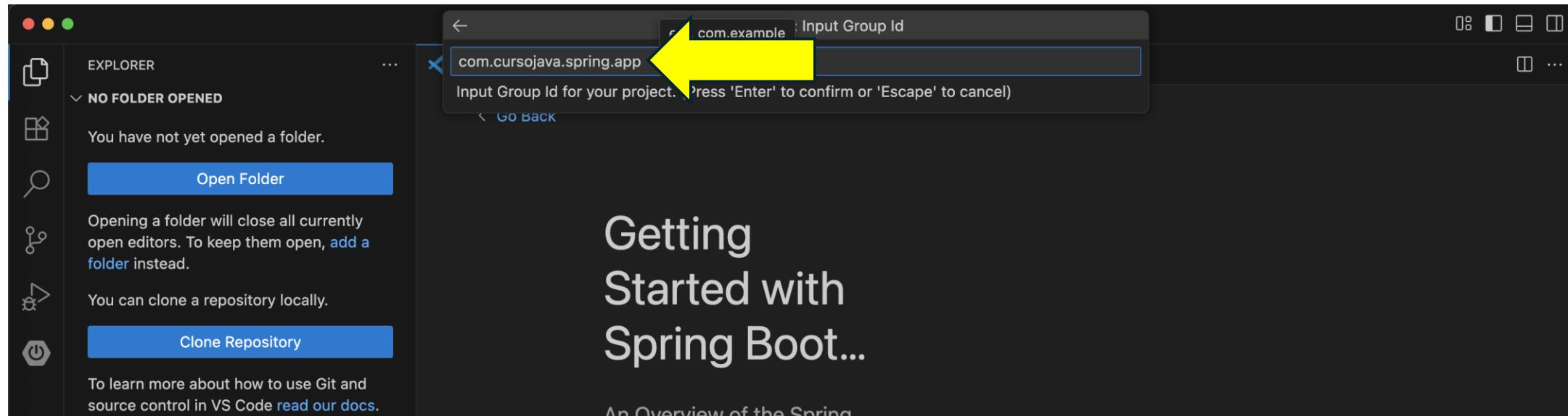
No se permite el uso de mayúsculas





Agregar nombre de la aplicación





Agregar nombre de la aplicación
Luego "Enter"

Getting Started with Spring Boot...

An Overview of the Spring



The screenshot shows the Spring Initializr interface. At the top, there is a search bar with placeholder text "e.g. demo" and a tooltip "Input Artifact Id". Below the search bar, the text "springboot-applications" is entered. A message below the input field says "Input Artifact Id for your project. (Press 'Enter' to confirm or 'Escape' to cancel)". In the center, the title "Getting Started with Spring Boot i..." is displayed. Below it, a subtitle reads "An Overview of the Spring Tools in VS Code to get started and to work with existing...". At the bottom, there is a blue button labeled "Create New Project".

← e.g. demo r: Input Artifact Id

springboot-applications

Input Artifact Id for your project. (Press 'Enter' to confirm or 'Escape' to cancel)

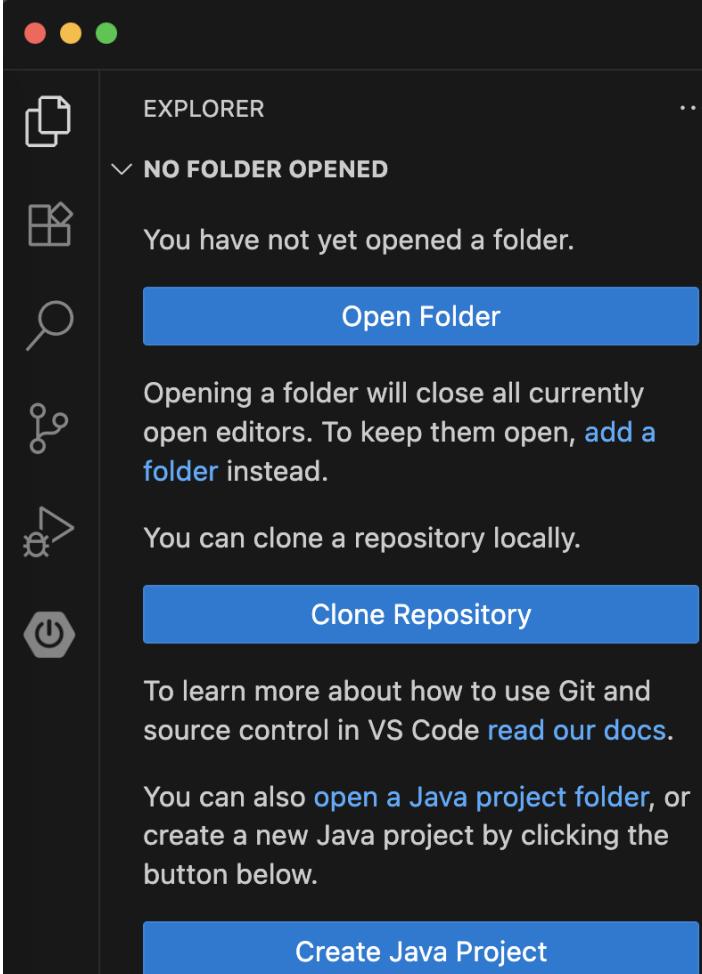
Getting Started with Spring Boot i...

An Overview of the Spring Tools in VS Code to get started and to work with existing...

Create New Project

Nombre del proyecto para que se identifique dentro de una carpeta de proyectos





Para tipo de empaquetado

Specify packaging type. | specify packaging type

Specify packaging type.

Jar

War

EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, [add a folder instead](#).

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or create a new Java project by clicking the button below.

Create Java Project

Getting Started with Spring Boot...

An Overview of the Spring Tools in VS Code to get started and to work with...

Spring In

Search for dependency

Selected 2 depen
Press <Enter> to co

Spring Web Web



EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, [add a folder](#) instead.

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or create a new Java project by clicking the button below.

Create Java Project

Spring Initializr: Specify Java version

Back (^-)

Java version.

17

24

21

Getting Started with Spring Boot...

An Overview of the Spring Tools in VS Code to get started and to work with...

Start with a Spring Boot project

Spring Initializr: Choose dependencies

Search for dependencies.

Selected 2 dependencies

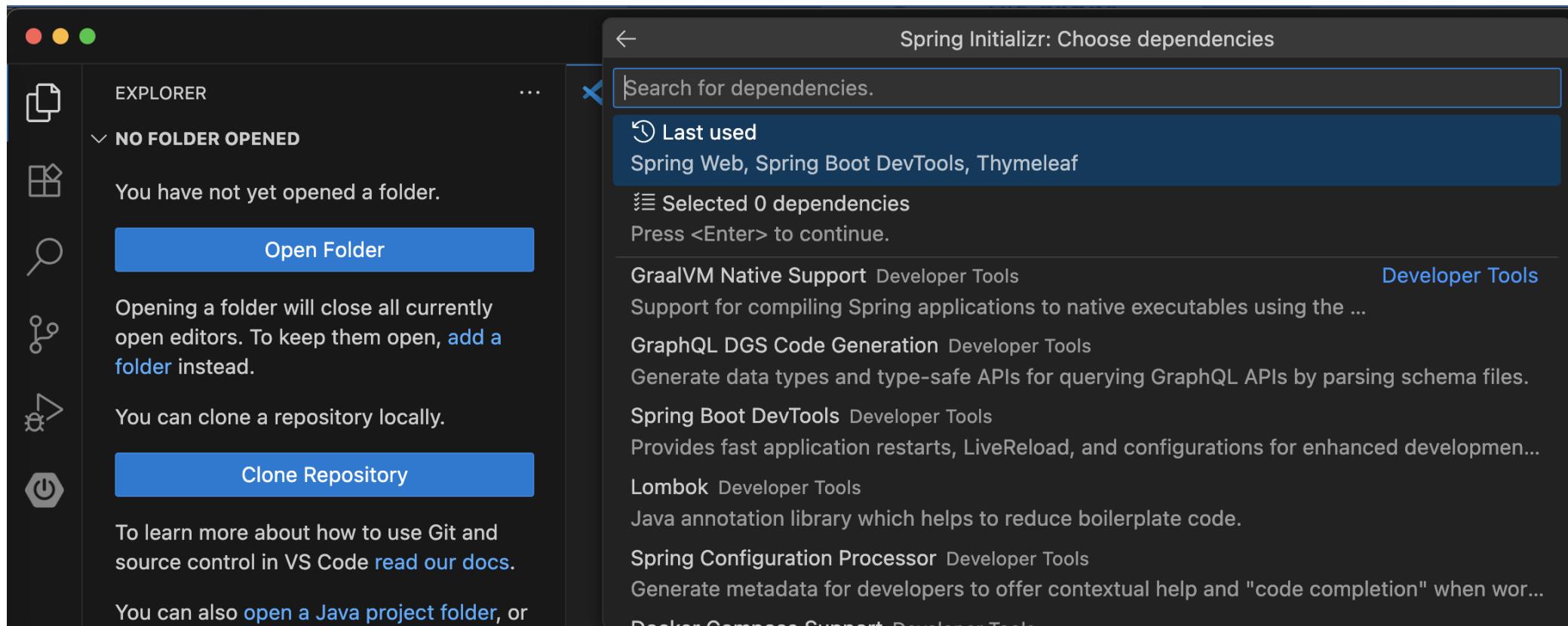
Press <Enter> to continue

✓ Spring Web Web Selected
Build web, including RESTful, applications using Spring MVC...

✓ Spring Boot DevTools Developer Tools

Especificamos una versión de Java





EXPLORER

NO FOLDER OPENED

You have not yet opened a folder.

Open Folder

Opening a folder will close all currently open editors. To keep them open, [add a folder instead](#).

You can clone a repository locally.

Clone Repository

To learn more about how to use Git and source control in VS Code [read our docs](#).

You can also [open a Java project folder](#), or

Spring Initializr: Choose dependencies

Last used

Spring Web, Spring Boot DevTools, Thymeleaf

Selected 0 dependencies

Press <Enter> to continue.

GraalVM Native Support Developer Tools

Support for compiling Spring applications to native executables using the ...

GraphQL DGS Code Generation Developer Tools

Generate data types and type-safe APIs for querying GraphQL APIs by parsing schema files.

Spring Boot DevTools Developer Tools

Provides fast application restarts, LiveReload, and configurations for enhanced development...

Lombok Developer Tools

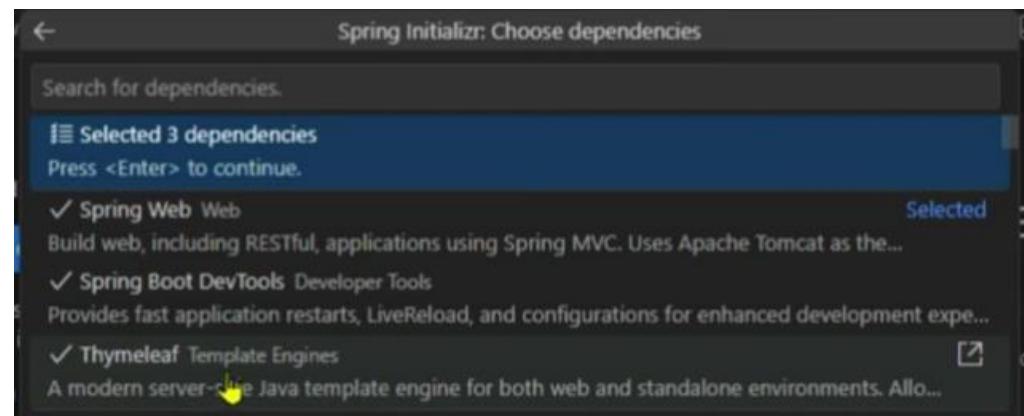
Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor Developer Tools

Generate metadata for developers to offer contextual help and "code completion" when wor...

Docker Compose Support Developer Tools

Luego nos solicitará elegir las dependencias.



Spring Initializr: Choose dependencies

Search for dependencies.

Selected 3 dependencies

Press <Enter> to continue.

✓ Spring Web Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the...

✓ Spring Boot DevTools Developer Tools

Provides fast application restarts, LiveReload, and configurations for enhanced development exp...

✓ Thymeleaf Template Engines

A modern server-side Java template engine for both web and standalone environments. All...



← Spring Initializr: Choose dependencies

spring web

Spring Web Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as t...

Spring Web Services Web

Facilitates contract-first SOAP development. Allows for the creation of flexible web services...

Spring Reactive Web Web

Build reactive web applications with **Spring WebFlux** and Netty.

Last used

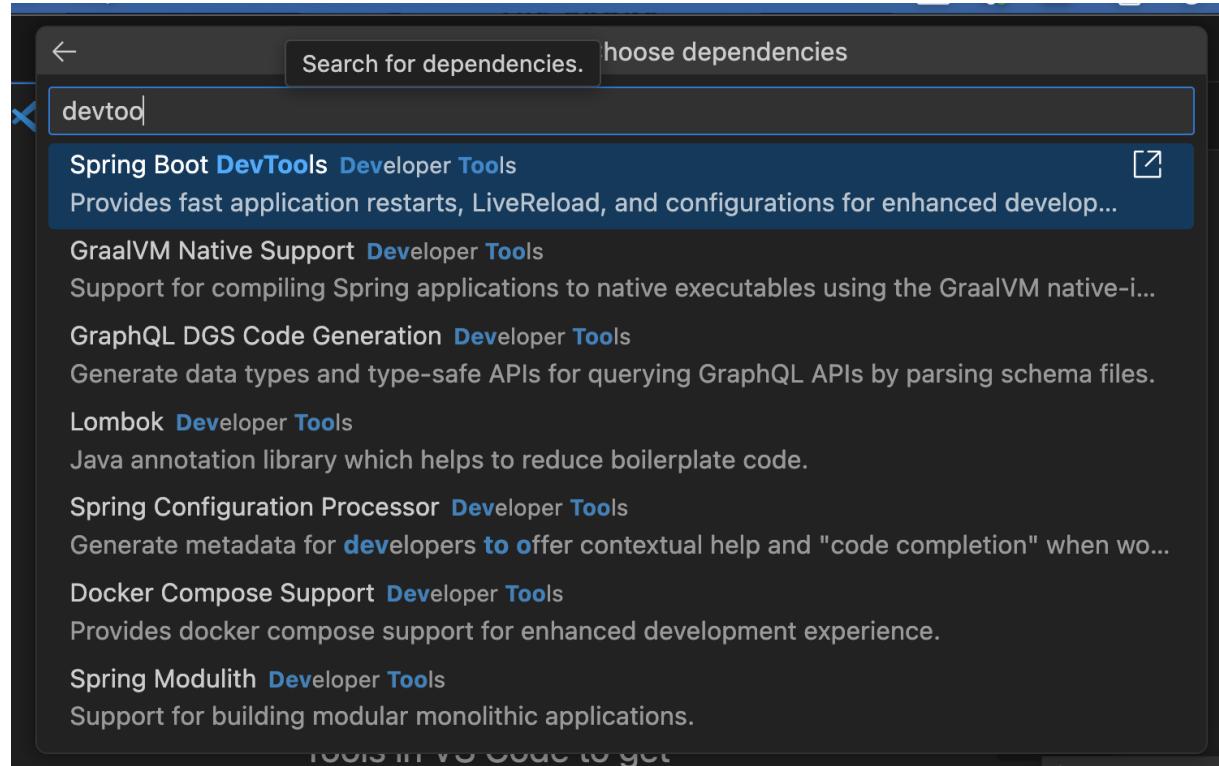
Spring Web, Spring Boot DevTools, Thymeleaf

Esta es la dependencia que necesitas para crear **APIs REST y aplicaciones web tradicionales** con Spring MVC.

Internamente incluye:

- `spring-web`
- `spring-webmvc`
- Servidor **Tomcat embebido**
- Soporte para `@RestController`, `@RequestMapping`, etc.
- Conversión automática de objetos a JSON (usando Jackson)





spring-boot-devtools es una dependencia opcional que incluye herramientas para **mejorar la experiencia del desarrollador** durante el desarrollo de una aplicación Spring Boot.



¿Para qué sirve?

Funcionalidad	Explicación clara
 Reinicio automático	Cuando cambias un archivo Java o un template HTML, la aplicación se reinicia sola (hot reload)
 LiveReload	Si estás trabajando con HTML o frontend, se recarga automáticamente el navegador (si usas un plugin compatible)
 Desactiva cachés de templates	Evita tener que limpiar caché al cambiar archivos HTML, Thymeleaf, etc.
 Cambia el comportamiento en modo dev	Por ejemplo, desactiva la seguridad o el logging detallado si lo configuras
 Detecta errores más rápido	Al estar más reactiva, te da feedback casi en tiempo real



←

Search for dependencies. ↗ dependencies

thy

Thymeleaf Template Engines 

A modern server-side Java template engine for both web and standalone environments....

Spring Boot Actuator Ops

Supports built in (or custom) endpoints **that** let **you** monitor and manage your application - ...

Consul Configuration Spring Cloud Config

Enable and configure **the** common patterns inside **your** application and build large distribut...

Thymeleaf es un **motor de plantillas HTML** para aplicaciones Java del lado del servidor. Se integra perfectamente con Spring Boot y permite crear páginas HTML dinámicas, donde puedes insertar datos desde tu backend usando sintaxis muy parecida a HTML puro.



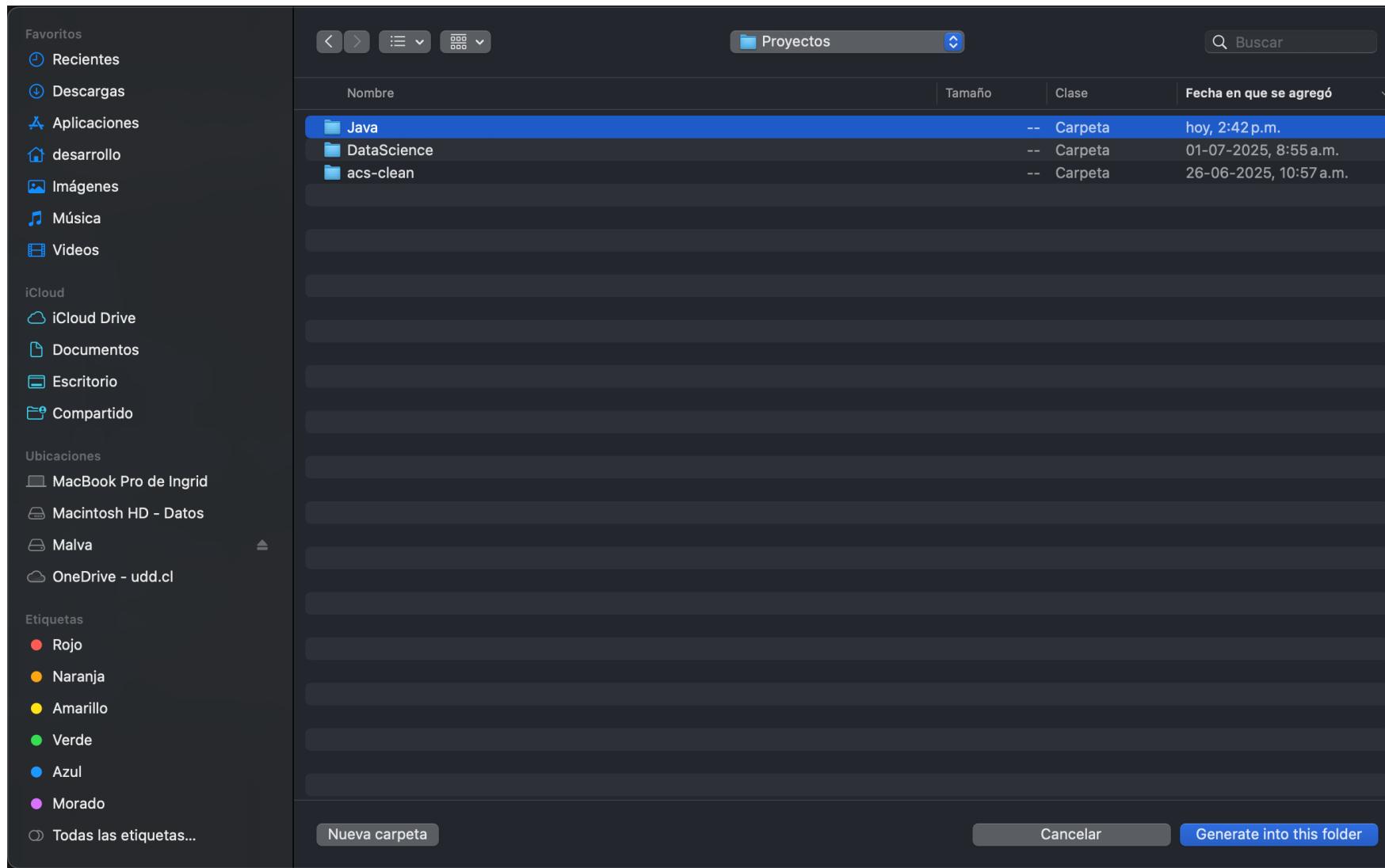
← Spring Initializr: Choose dependencies

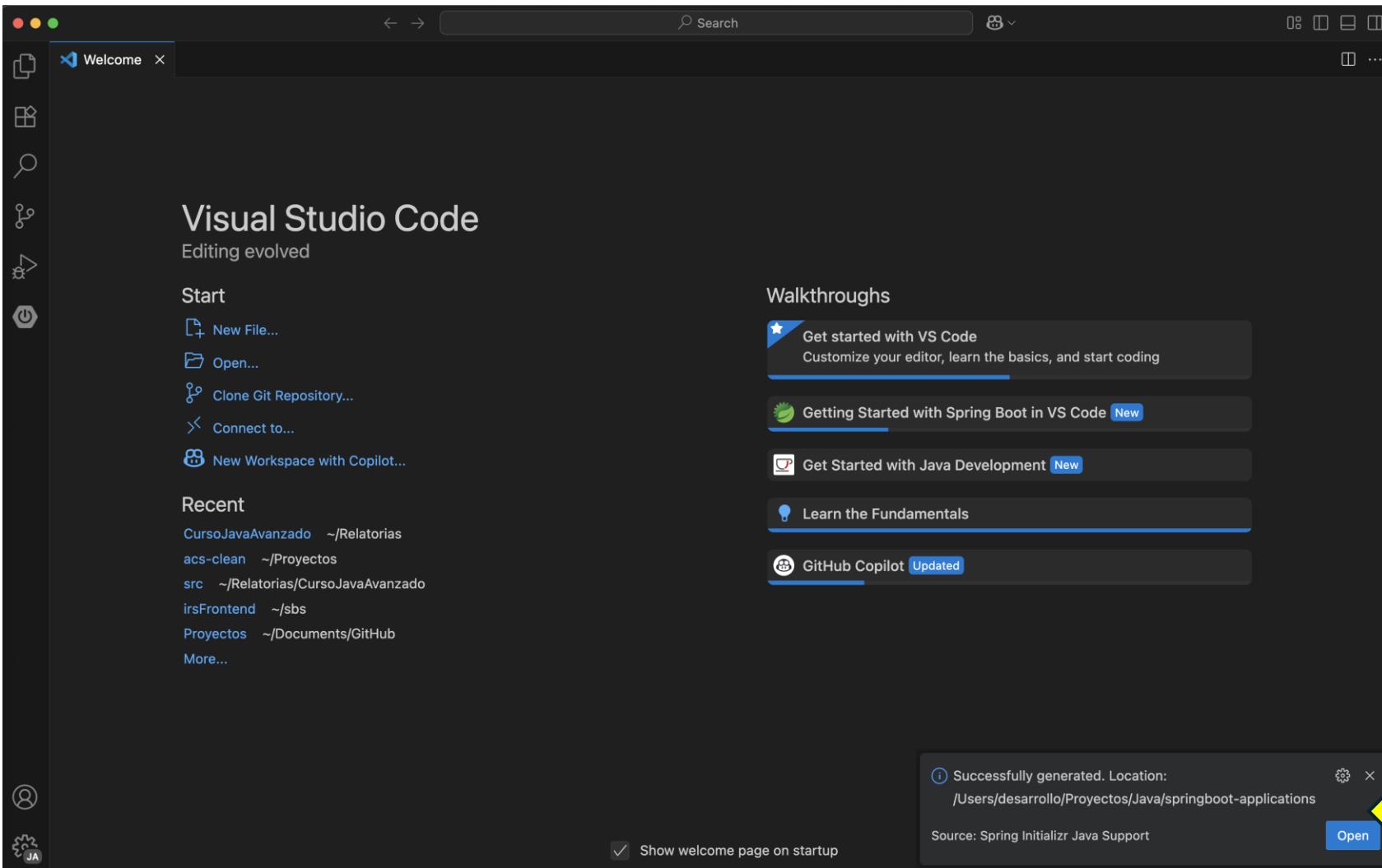
Search for dependencies.

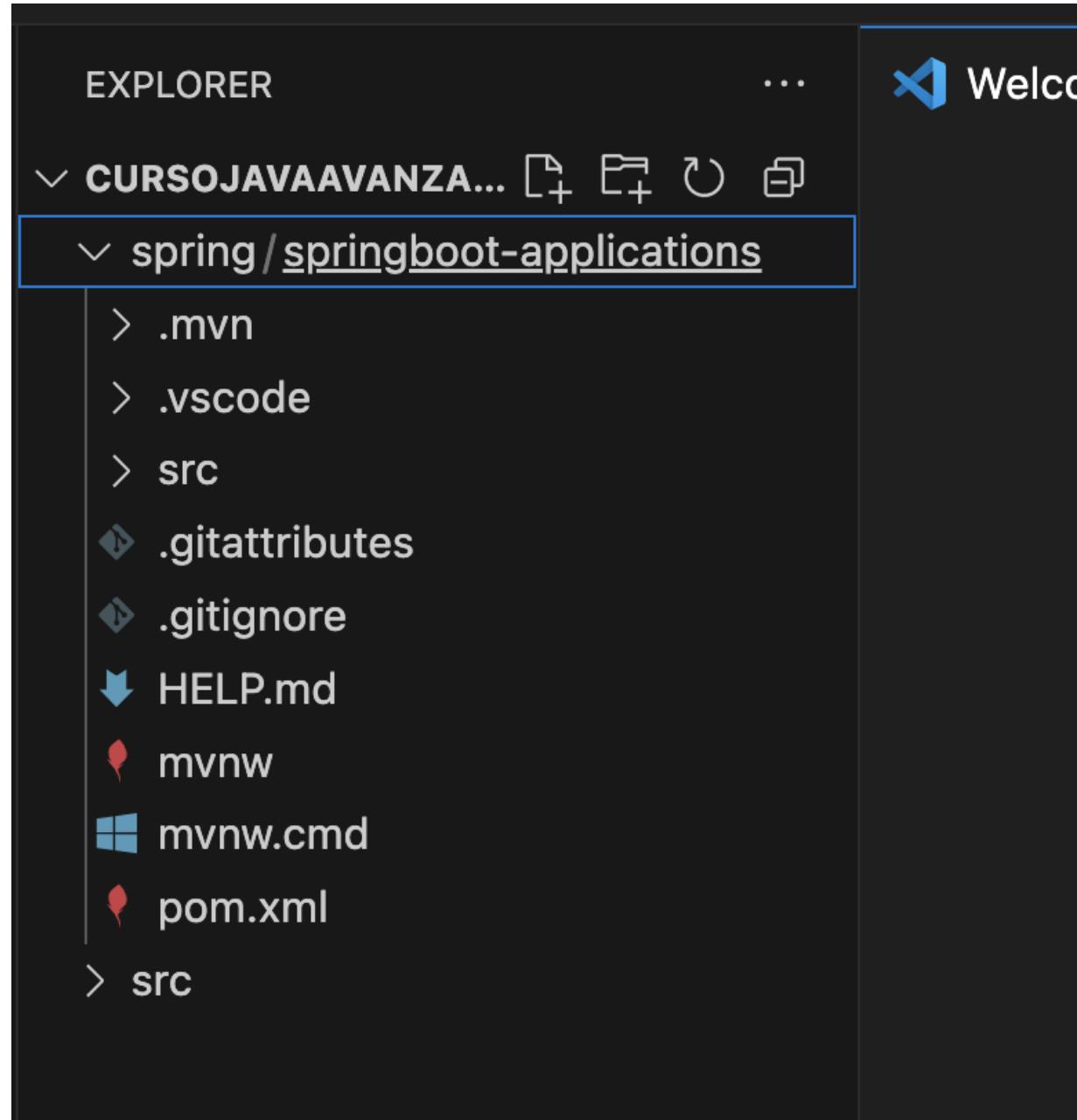
Selected 3 dependencies
Press <Enter> to continue.

- ✓ Spring Web Web Selected
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomca...
- ✓ Spring Boot DevTools Developer Tools
Provides fast application restarts, LiveReload, and configurations for enhanced developmen...
- ✓ Thymeleaf Template Engines
A modern server-side Java template engine for both web and standalone environments. Allo...

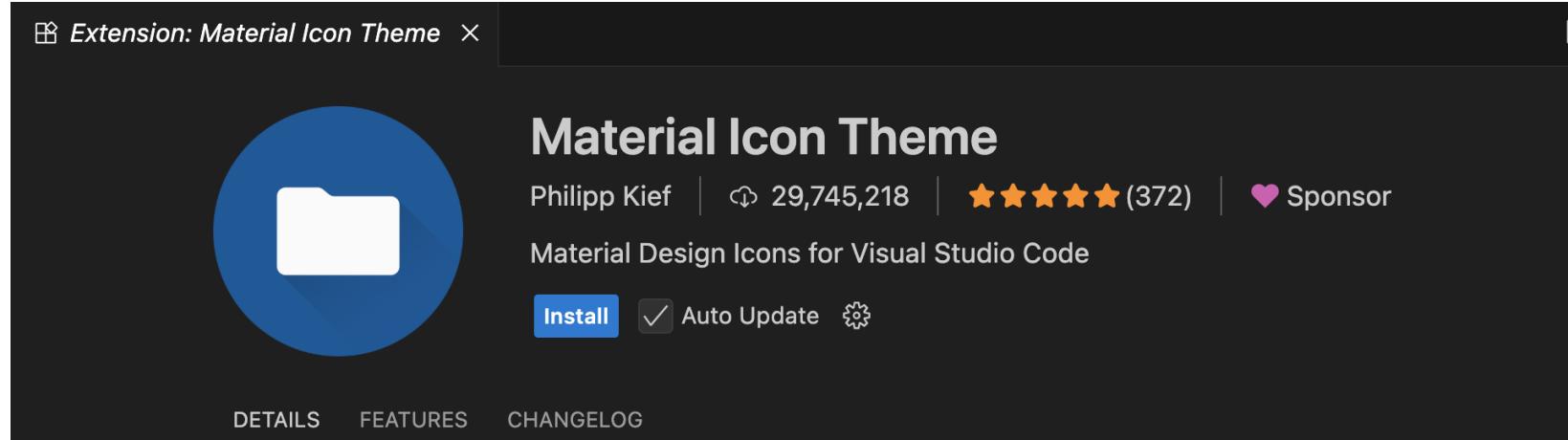








Extension: Material Icon Theme X



The image shows a screenshot of a Visual Studio Code extension page for 'Material Icon Theme'. The page has a dark background. On the left is a large blue circular icon containing a white folder icon. To the right of the icon, the extension name 'Material Icon Theme' is displayed in a large white font. Below the name, the author 'Philipp Kief' is listed, along with the download count '29,745,218' and a 5-star rating '(372)'. There is also a 'Sponsor' button with a heart icon. A descriptive text 'Material Design Icons for Visual Studio Code' follows. At the bottom, there are three buttons: 'Install' (in blue), 'Auto Update' (with a checked checkbox), and a gear icon for settings. Below these buttons are three navigation links: 'DETAILS', 'FEATURES', and 'CHANGELOG'.

Material Icon Theme

Philipp Kief | 29,745,218 | ★★★★★(372) | Sponsor

Material Design Icons for Visual Studio Code

Install Auto Update 

DETAILS FEATURES CHANGELOG



The screenshot shows a dark-themed instance of the Visual Studio Code (VS Code) code editor. The left side features the Explorer sidebar with a tree view of the project structure. The main workspace contains a search bar at the top and two open files: a properties file and a Java class.

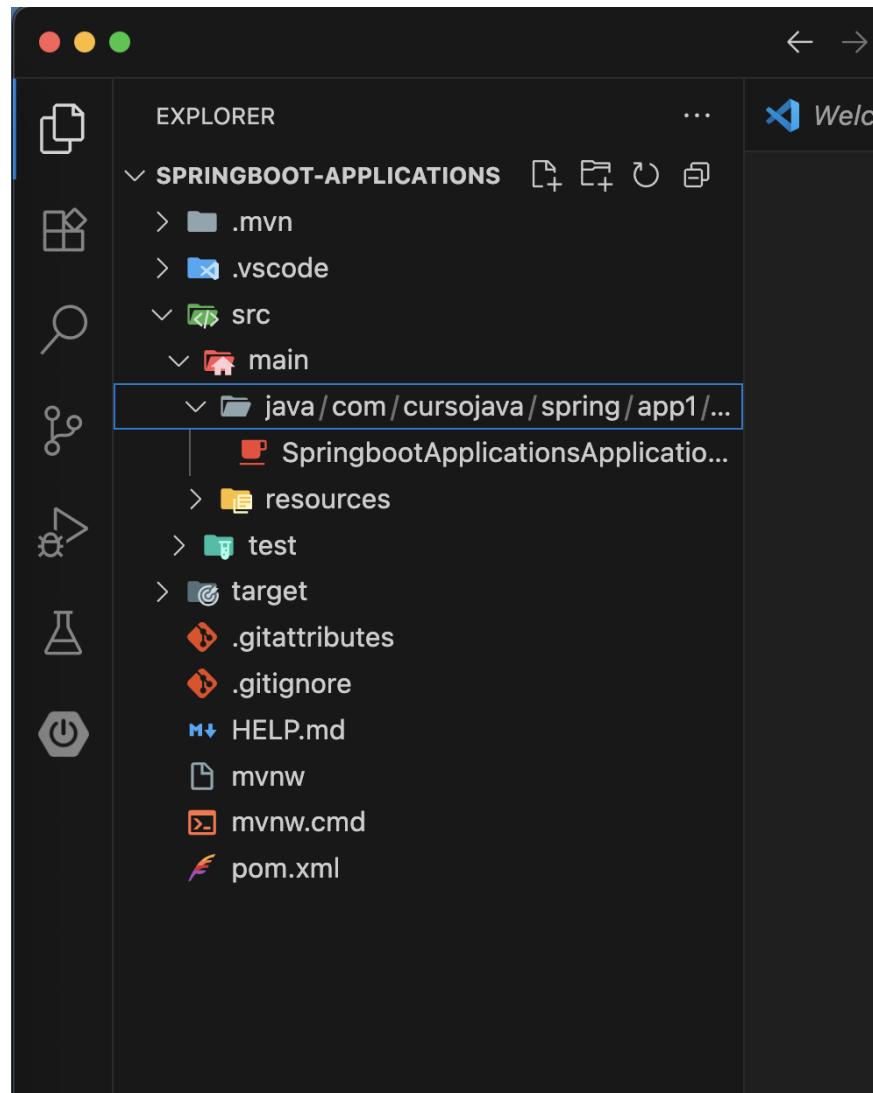
EXPLORER

- CURSOJAVAAVANZADO
 - .vscode
 - spring/springboot-applications
 - .mvn/wrapper/maven-wrapper.properties
 - .vscode
 - src
 - main
 - java/com/cursojava/spring/app1/springboot_applications/ SpringbootApplicationsApplication.java
 - resources/static
 - test/java/com/cursojava/spring/app1/springboot_applications
 - target
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml
 - src
 - Main.class
 - Main.java

application.properties

```
spring > springboot-applications > src > main > resources >
1  spring.application.name=springboot-application
2
```



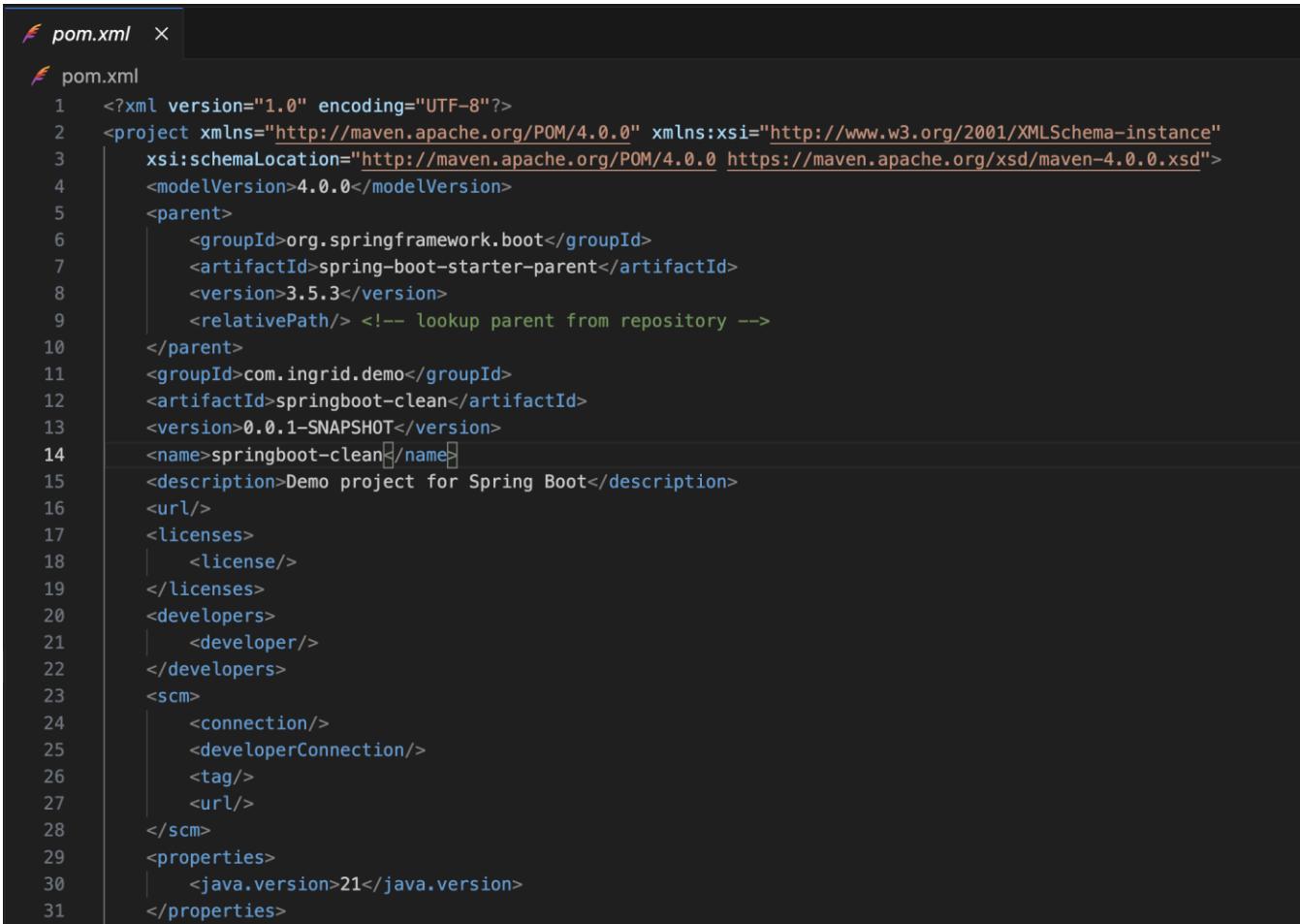


¿Qué puedes hacer ahora?

Explora el proyecto

- Abre `src/main/java` → allí encontrarás el paquete base, por ejemplo:
`com.cursojava.springbootapplications`

¿Qué es pom.xml?

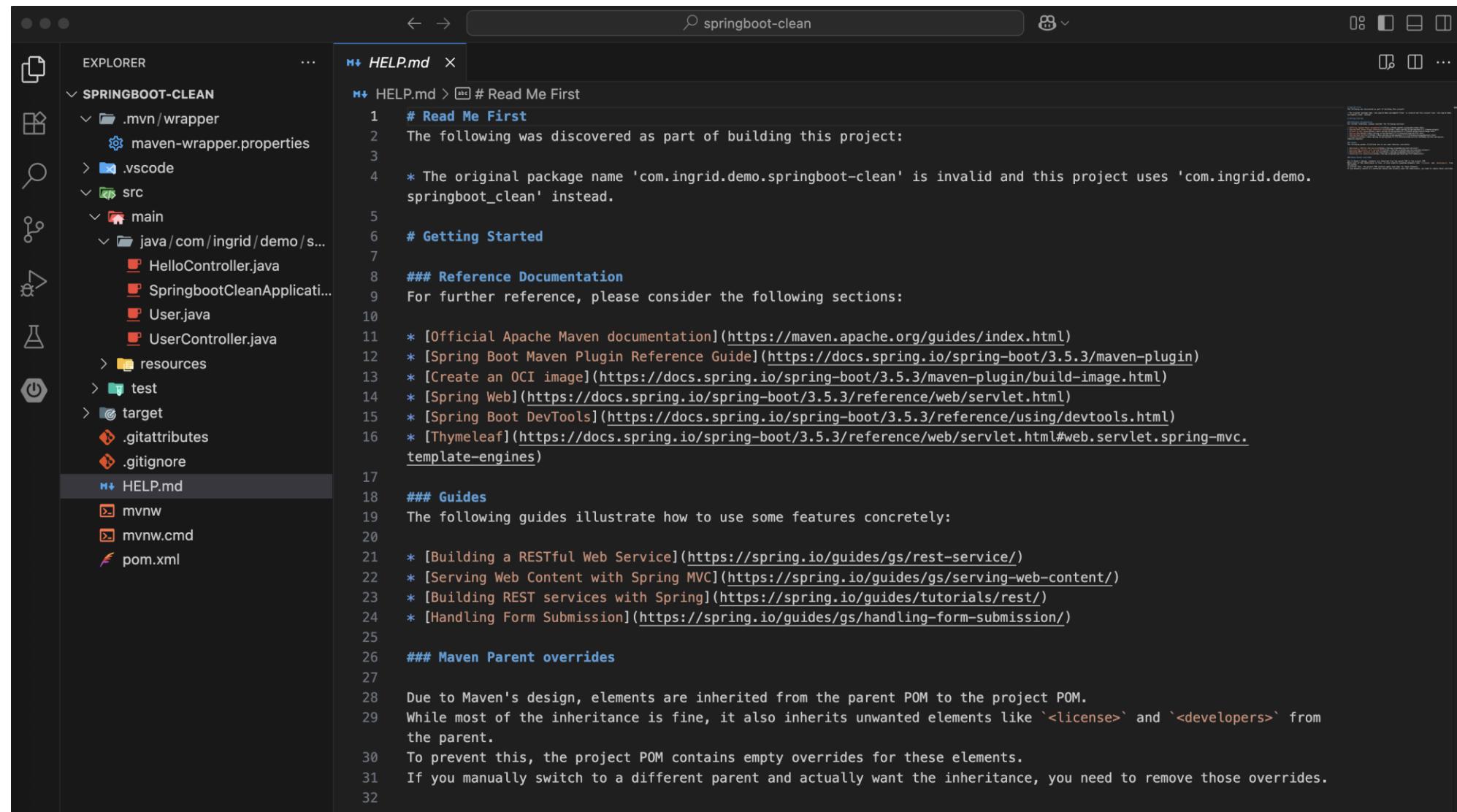


The screenshot shows a code editor with a dark theme. On the left, there's a sidebar with two tabs: 'pom.xml' (which is active) and 'pom.xml'. The main area displays the XML content of the pom.xml file. The code is color-coded: blue for tags like <project>, <parent>, <groupId>, etc.; green for attributes like version and encoding; and red for URLs in namespaces and xsi:schemaLocation. The XML structure includes details about the project's dependencies, parent project, version, name, description, licenses, developers, SCM, and properties (specifically java.version).

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>3.5.3</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.ingrid.demo</groupId>
  <artifactId>springboot-clean</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>springboot-clean</name>
  <description>Demo project for Spring Boot</description>
  <url/>
  <licenses>
    <license/>
  </licenses>
  <developers>
    <developer/>
  </developers>
  <scm>
    <connection/>
    <developerConnection/>
    <tag/>
    <url/>
  </scm>
  <properties>
    <java.version>21</java.version>
  </properties>
```

Significa **Project Object Model**, y el archivo **pom.xml** es un documento en formato XML que le dice a Maven:

- **Qué dependencias** (librerías) necesita el proyecto.
- **Qué versión de Java** estás usando.
- **Qué plugins** usar para compilar, testear o empaquetar.
- **Qué nombre y versión** tiene el proyecto.
- **Qué configuración usar para ejecutar Spring Boot**, entre otras cosas.



The screenshot shows the Visual Studio Code interface with the following details:

- Explorer View:** Shows the project structure under "SPRINGBOOT-CLEAN".
- Search Bar:** Contains the text "springboot-clean".
- Code Editor:** Displays the content of the "HELP.md" file.

The content of the "HELP.md" file is as follows:

```
1 # Read Me First
2 The following was discovered as part of building this project:
3
4 * The original package name 'com.ingrid.demo.springboot-clean' is invalid and this project uses 'com.ingrid.demo.springboot_clean' instead.
5
6 # Getting Started
7
8 ### Reference Documentation
9 For further reference, please consider the following sections:
10
11 * [Official Apache Maven documentation](https://maven.apache.org/guides/index.html)
12 * [Spring Boot Maven Plugin Reference Guide](https://docs.spring.io/spring-boot/3.5.3/maven-plugin)
13 * [Create an OCI image](https://docs.spring.io/spring-boot/3.5.3/maven-plugin/build-image.html)
14 * [Spring Web](https://docs.spring.io/spring-boot/3.5.3/reference/web/servlet.html)
15 * [Spring Boot DevTools](https://docs.spring.io/spring-boot/3.5.3/reference/using/devtools.html)
16 * [Thymeleaf](https://docs.spring.io/spring-boot/3.5.3/reference/web/servlet.html#web.servlet.spring-mvc.template-engines)
17
18 ### Guides
19 The following guides illustrate how to use some features concretely:
20
21 * [Building a RESTful Web Service](https://spring.io/guides/gs/rest-service/)
22 * [Serving Web Content with Spring MVC](https://spring.io/guides/gs/serving-web-content/)
23 * [Building REST services with Spring](https://spring.io/guides/tutorials/rest/)
24 * [Handling Form Submission](https://spring.io/guides/gs/handling-form-submission/)
25
26 ### Maven Parent overrides
27
28 Due to Maven's design, elements are inherited from the parent POM to the project POM.
29 While most of the inheritance is fine, it also inherits unwanted elements like `<license>` and `<developers>` from
the parent.
30 To prevent this, the project POM contains empty overrides for these elements.
31 If you manually switch to a different parent and actually want the inheritance, you need to remove those overrides.
32
33
```

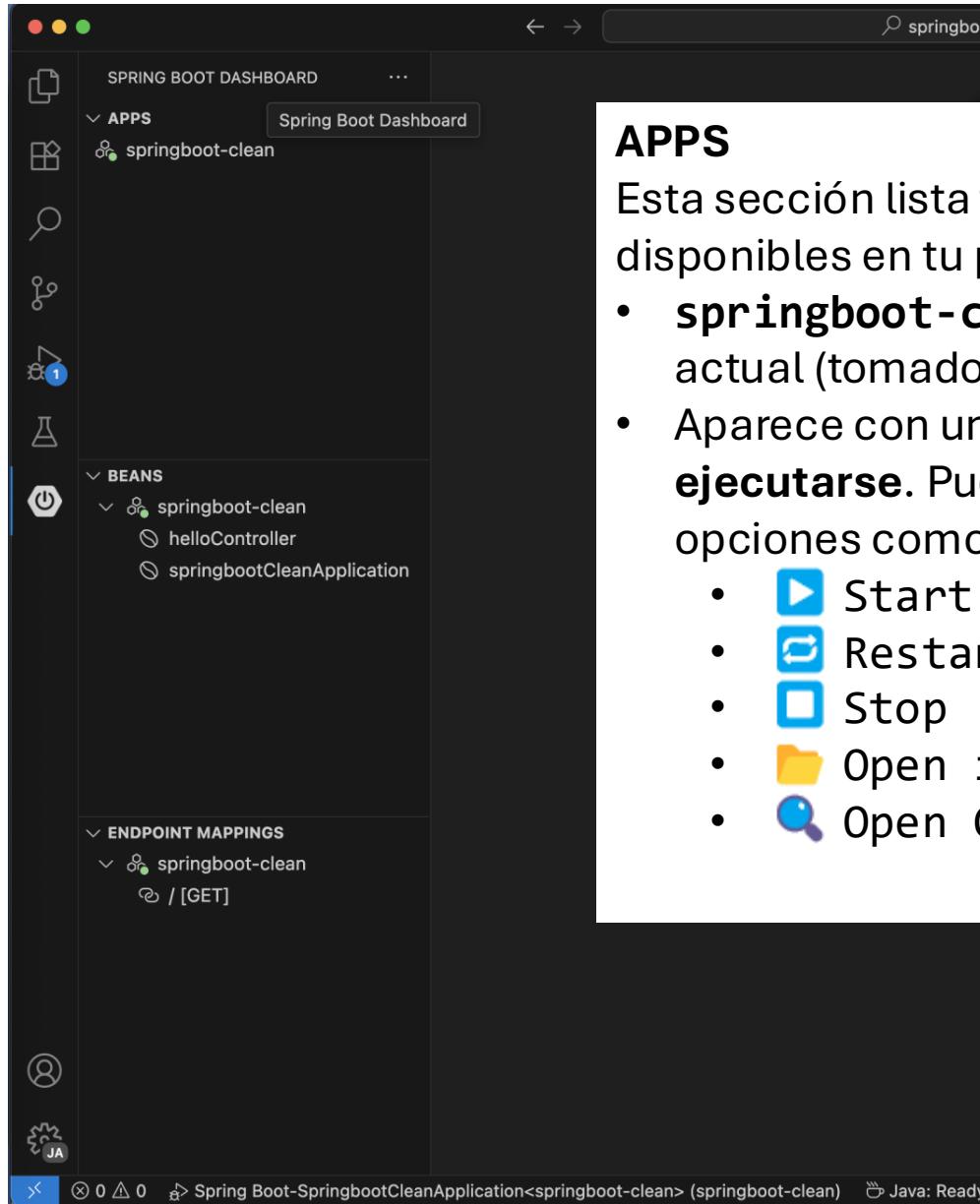
Help: Para toda la documentación del proyecto.

The screenshot shows the Visual Studio Code interface with a dark theme. The left sidebar contains icons for Explorer, Search, Problems, and others. The Explorer view shows a project structure for 'SPRINGBOOT-CLEAN' with files like '.mvn/wrapper/maven-wrapper.properties', '.vscode', 'src/main/java/com/ingrid/demo/springboot_clean/HelloController.java', 'SpringbootCleanApplication.java' (which is currently selected), 'User.java', 'UserController.java', 'resources/application.properties', 'static', 'templates', 'test/java/com/ingrid/demo/SpringbootCleanApplication.java', and 'target' directory. The main editor area displays the code for 'SpringbootCleanApplication.java':

```
src > main > java > com > ingrid > demo > springboot_clean > SpringbootCleanApplication.java > Language Support for Java(T)
1 package com.ingrid.demo.springboot_clean;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class SpringbootCleanApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringbootCleanApplication.class, args);
11     }
12 }
13
14 }
```

Centro de nuestro proyecto, donde se configura todo.

Panel Spring Boot Dashboard

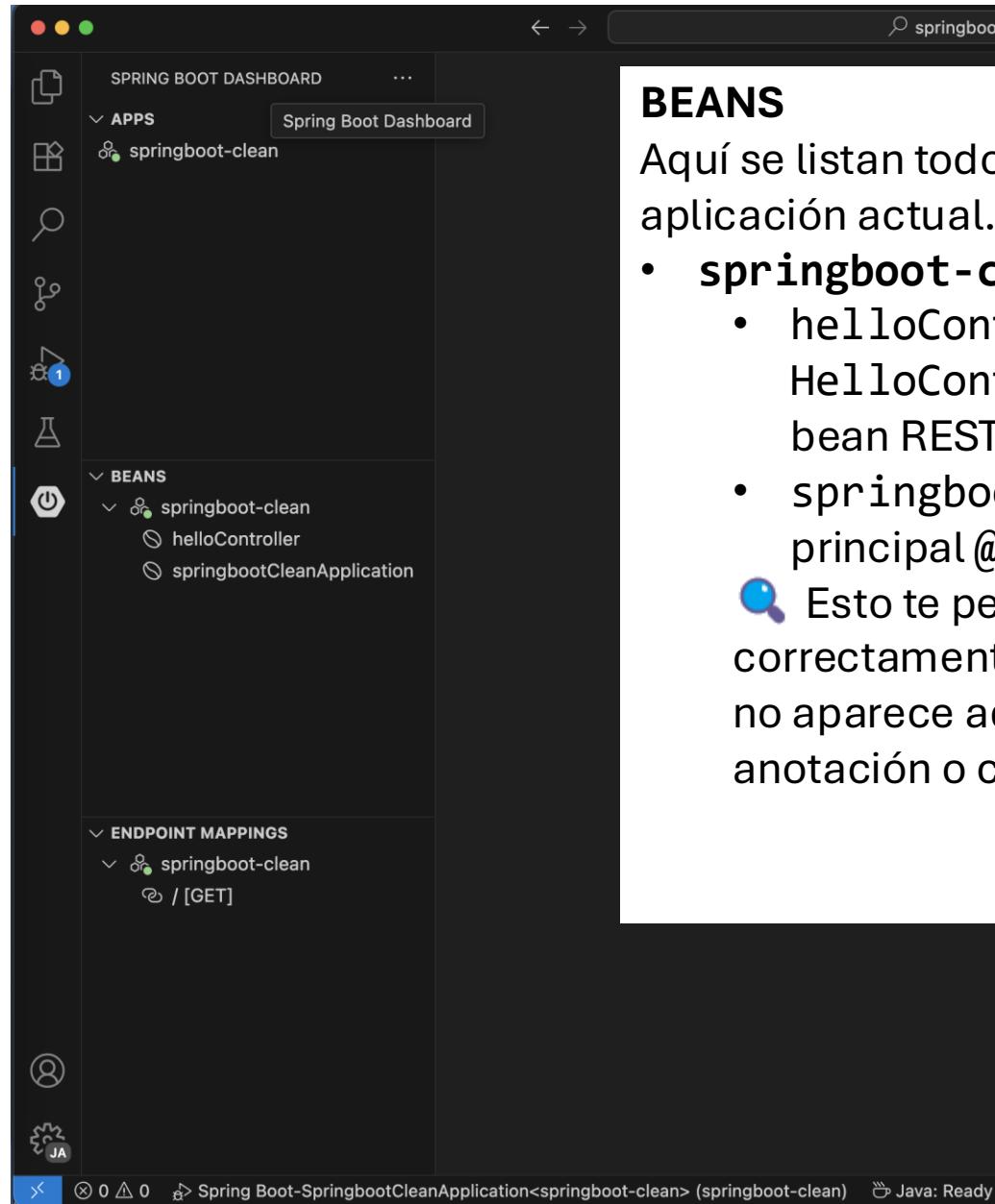


APPS

Esta sección lista todas las aplicaciones Spring Boot disponibles en tu proyecto.

- **springboot-clean**: Es el nombre de tu aplicación actual (tomado del <artifactId> del pom.xml).
- Aparece con un **ícono verde** porque está **lista para ejecutarse**. Puedes hacer clic derecho sobre ella y elegir opciones como:
 - Start
 - Restart
 - Stop
 - Open in Terminal
 - Open Config

Panel Spring Boot Dashboard

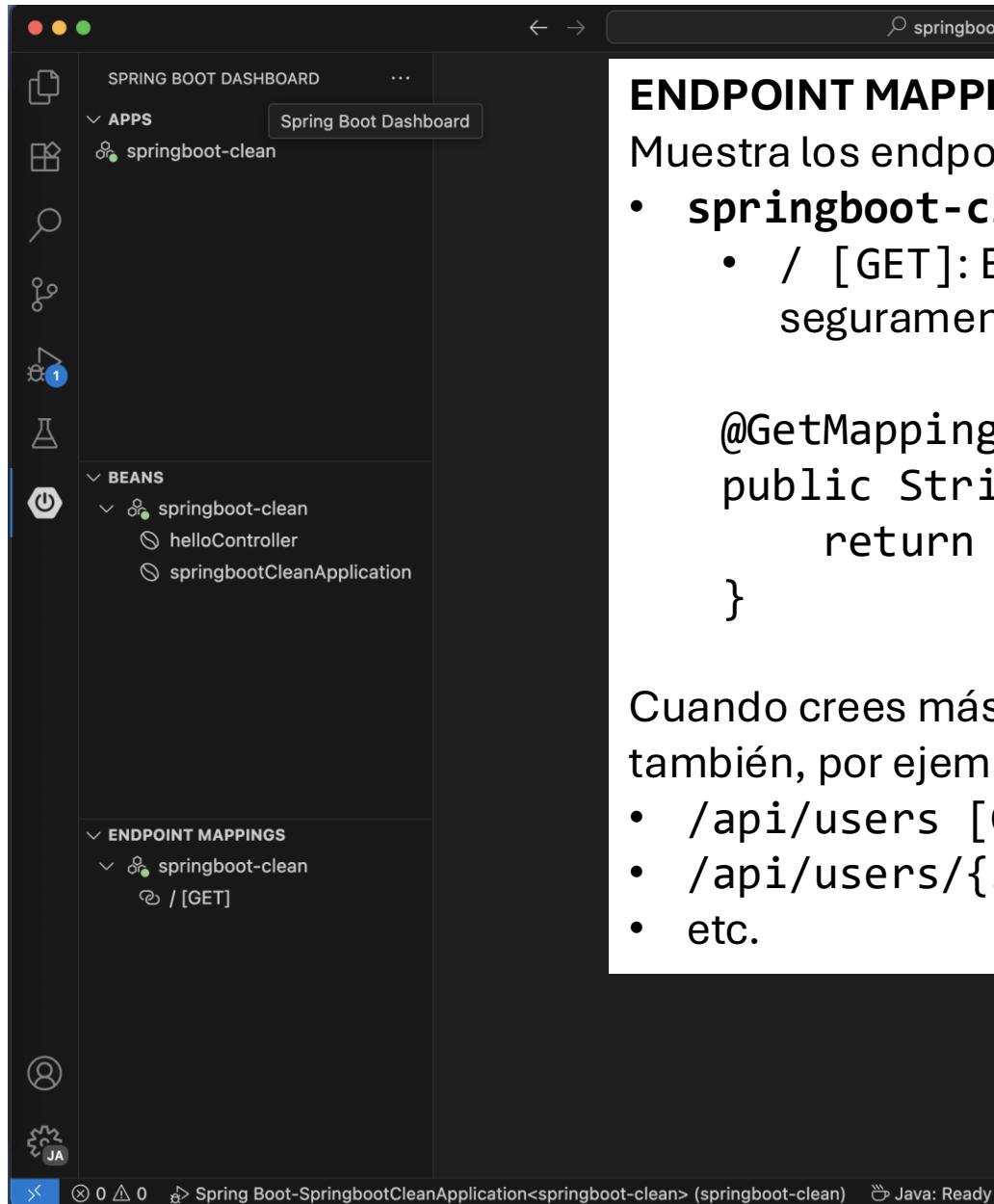


BEANS

Aquí se listan todos los **Beans registrados por Spring** en la aplicación actual.

- **springboot-clean**: Es el contexto de tu app.
 - **helloController**: Tu clase `HelloController.java` fue detectada como un bean REST.
 - **springbootCleanApplication**: Es la clase principal `@SpringBootApplication`.
 - 🔍 Esto te permite ver que Spring está cargando correctamente tus componentes. Si algún controlador no aparece aquí, es señal de que algo falta en su anotación o configuración.

Panel Spring Boot Dashboard



ENDPOINT MAPPINGS

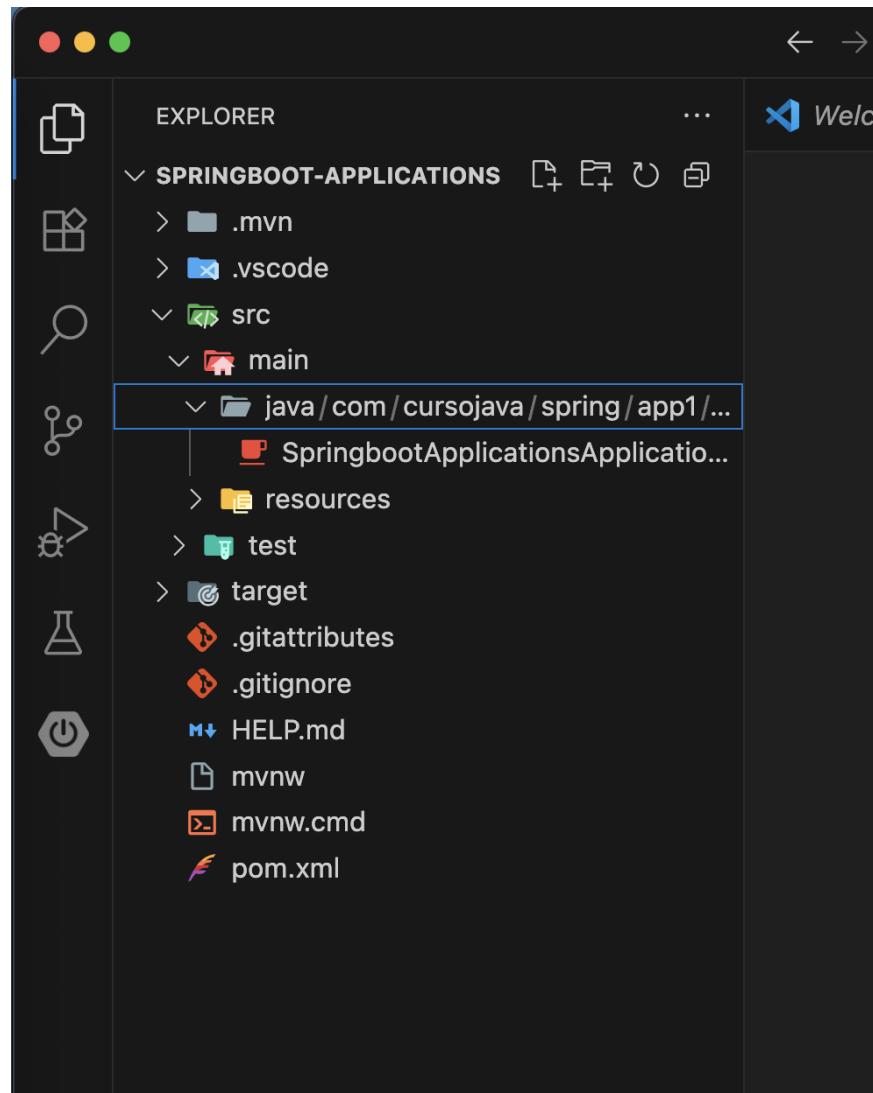
Muestra los endpoints REST registrados por tus controladores.

- **springboot-clean** (nuevamente el nombre de tu app)
 - / [GET]: Este endpoint es detectado desde tu HelloController, seguramente desde un método como:

```
@GetMapping("/")
public String hello() {
    return "Hola, mundo!";
}
```

Cuando crees más endpoints (como los de UserController), aparecerán aquí también, por ejemplo:

- /api/users [GET]
- /api/users/{id} [DELETE]
- etc.



Explora el proyecto

- Abre la clase `SpringbootApplicationsApplication.java` (la principal)

```
src > main > java > com > cursojava > spring > app1 > springboot_applications > SpringbootApplicationsApplication.java
1 package com.cursojava.spring.app1.springboot_applications;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6@SpringBootApplication
7 public class SpringbootApplicationsApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringbootApplicationsApplication.class, args);
11     }
12 }
13
14 }
```

`@SpringBootApplication`

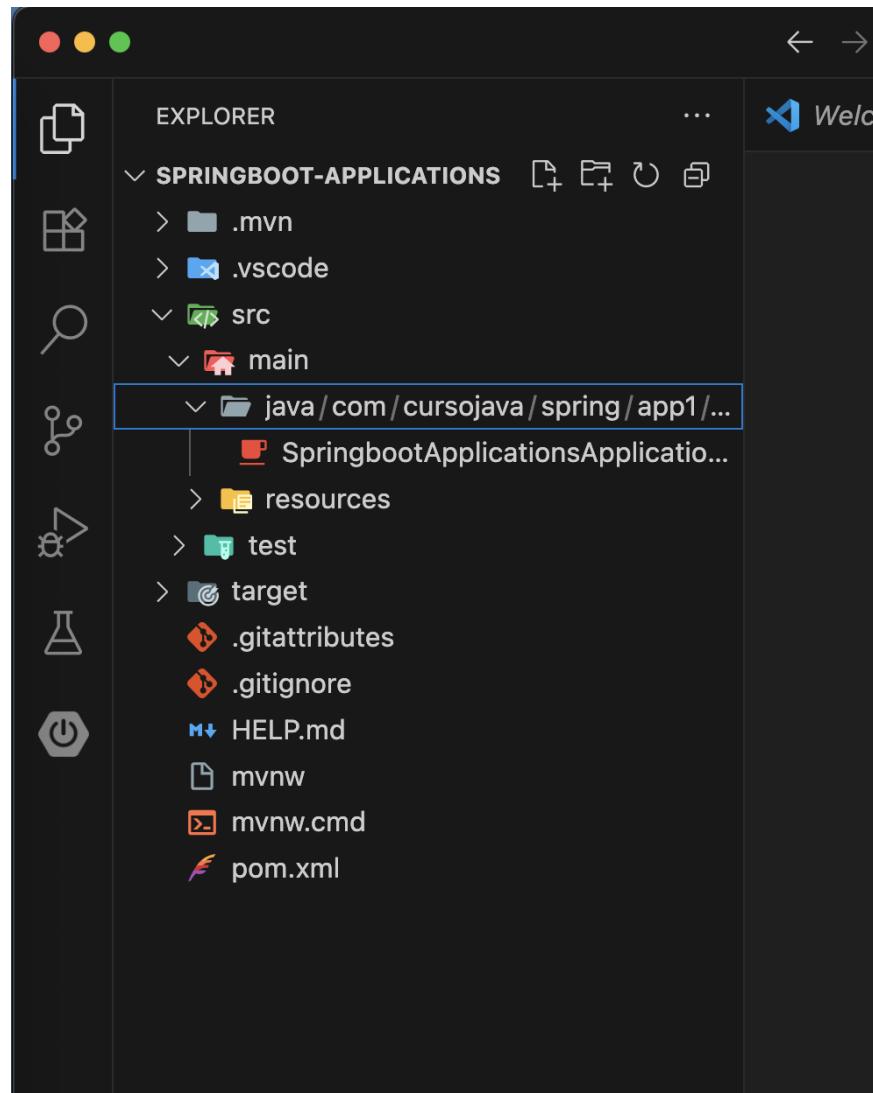
// Equivale a:

`@Configuration`

`@EnableAutoConfiguration`

`@ComponentScan`

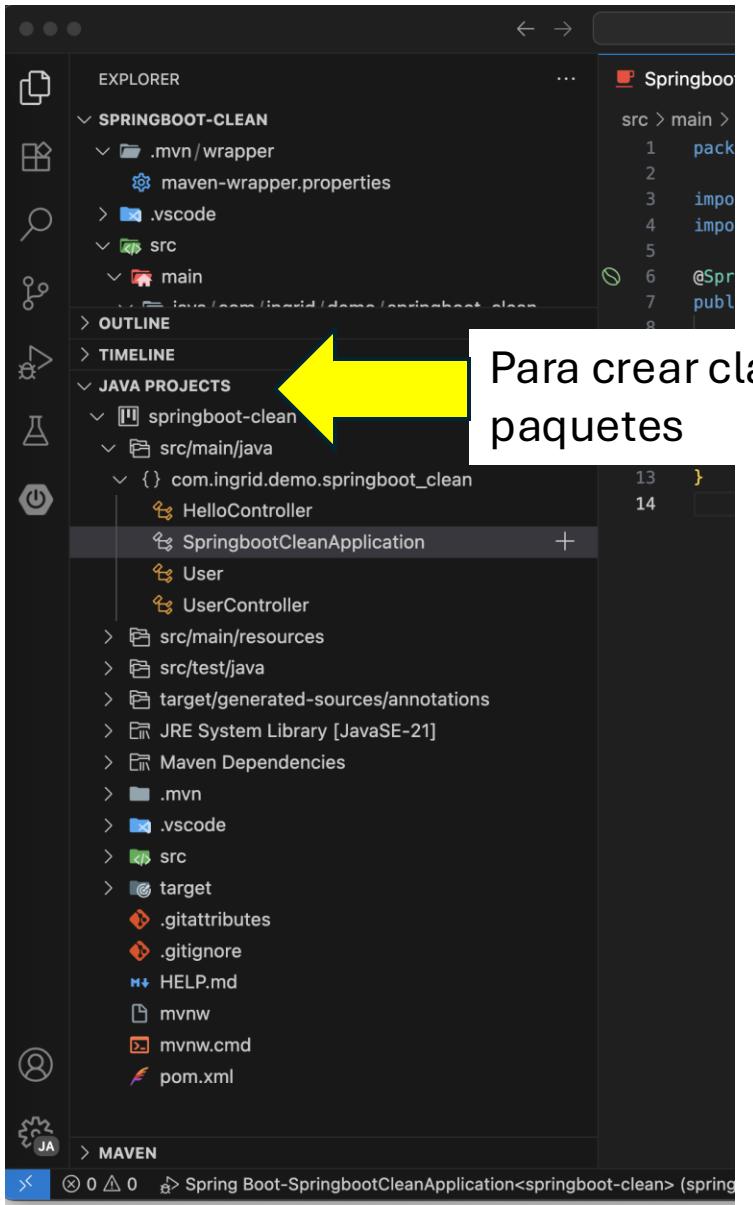
Esto permite levantar toda la app sin configuraciones adicionales.



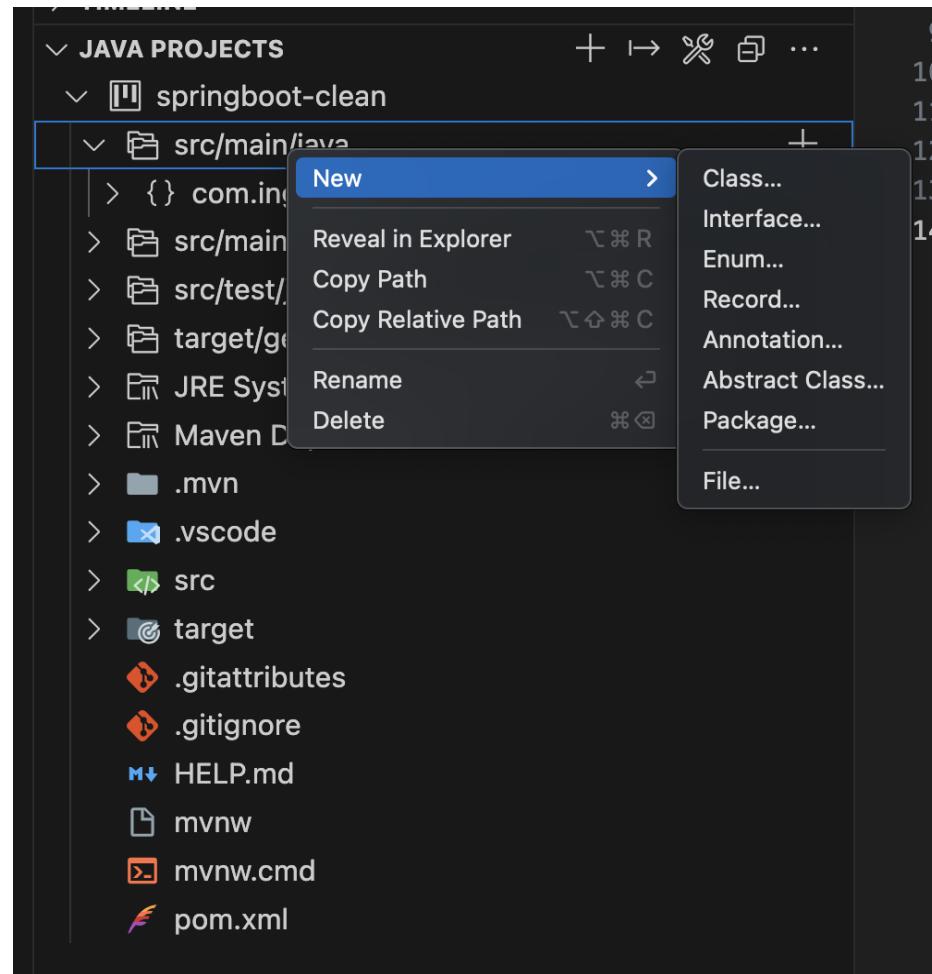
Ejecuta el proyecto

Haz clic derecho sobre esa clase y selecciona **“Run Java”** o usa el **Spring Boot Dashboard** si ya está visible en la barra lateral.

```
Preview HELP.md SpringbootApplicationsApplication.java
src > main > java > com > cursojava > spring > app1 > springboot_applications > SpringbootApplicationsApplication.java
1 package com.cursojava.spring.app1.springboot_applications;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6@SpringBootApplication
7 public class SpringbootApplicationsApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringbootApplicationsApplication.class, args);
11     }
12 }
13
14 }
```



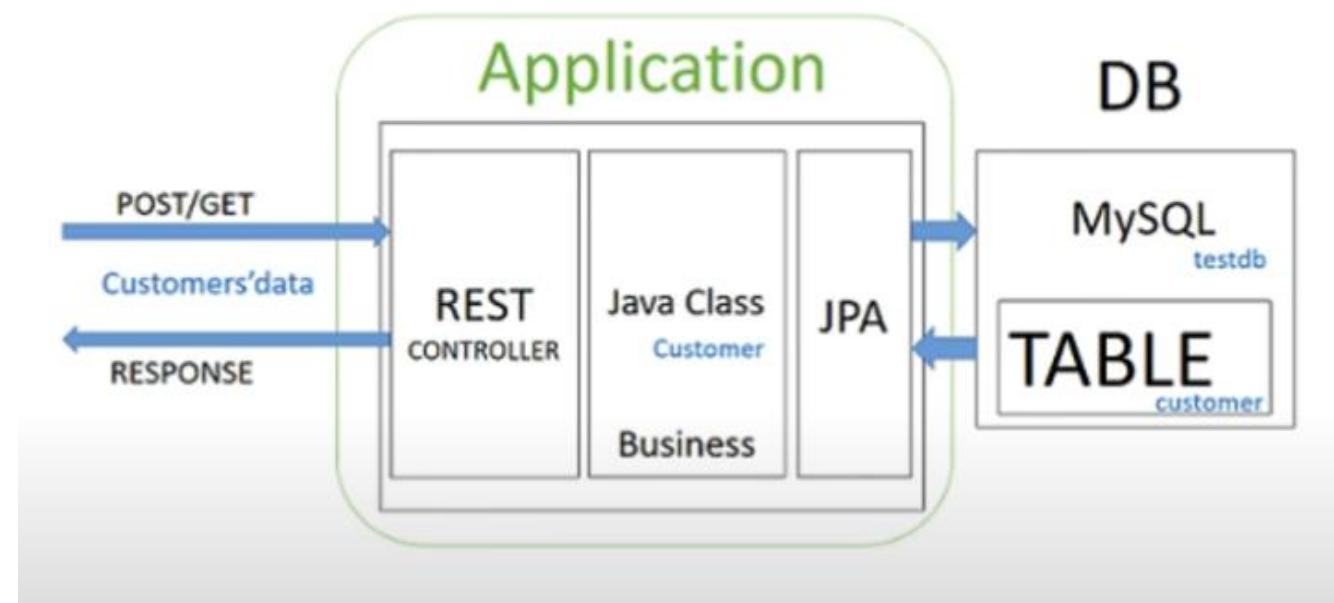
Para crear clases y paquetes



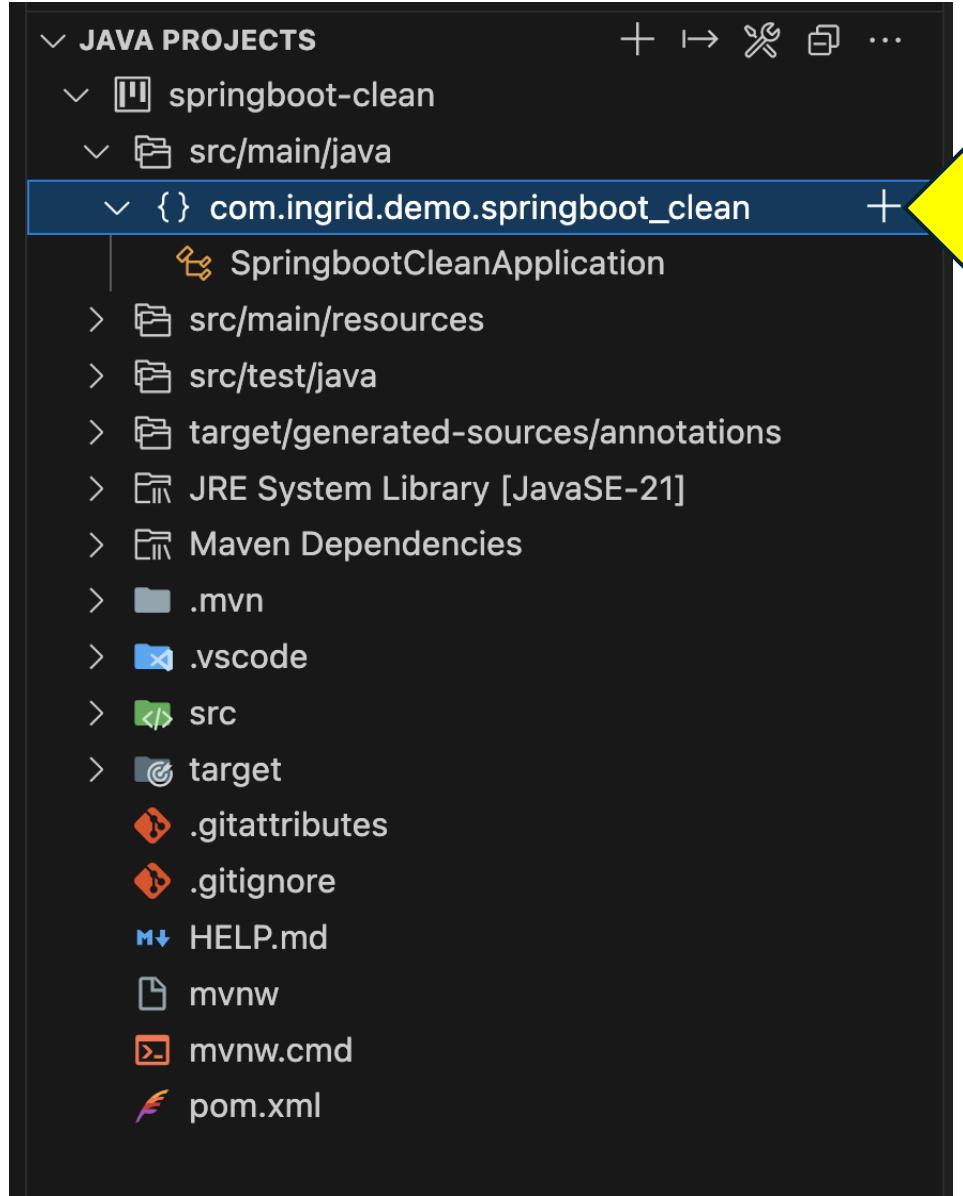
Creando un controlador

Un **controlador en Spring Boot** es una clase Java que se encarga de **manejar las peticiones HTTP** que llegan a tu aplicación web. Es una parte central del patrón **MVC (Modelo - Vista - Controlador)** y permite que tu aplicación responda a solicitudes del tipo:

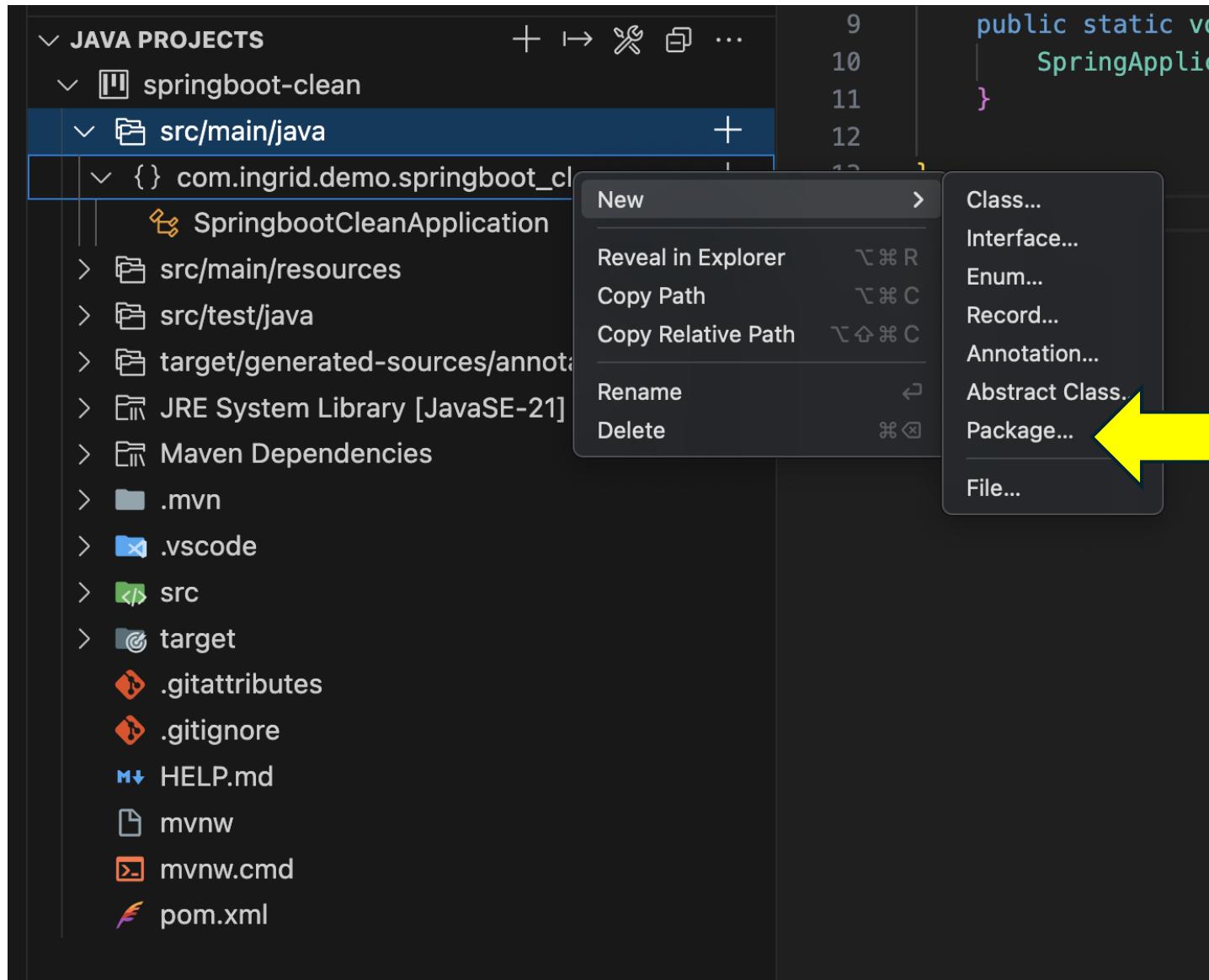
- GET /api/usuarios
- POST /api/usuarios
- DELETE /api/usuarios/1
- etc.



Actividad: Crear un controlador



Crearíamos una clase
dentro de nuestro
paquete



Pero... botón derecho:

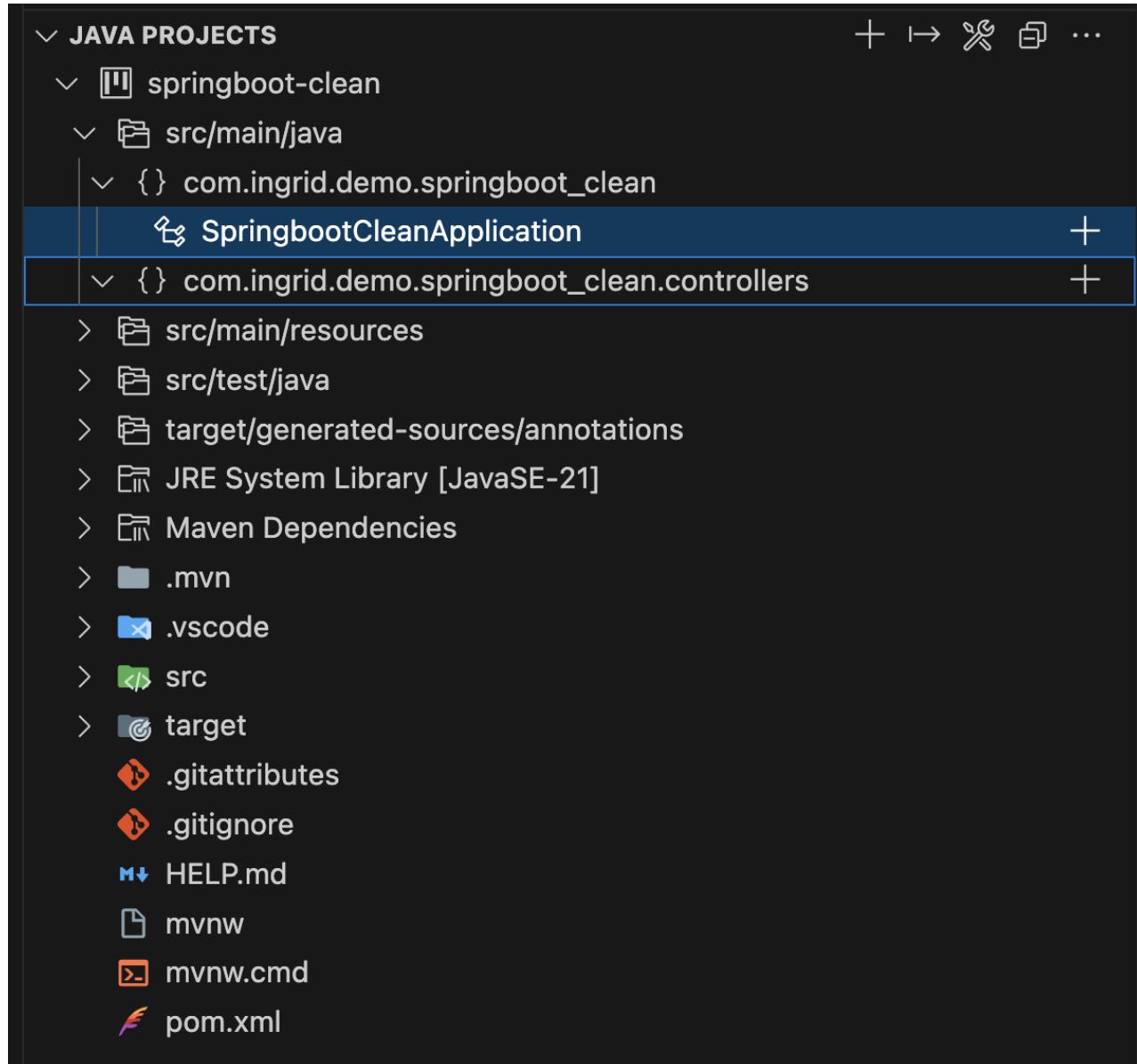
Acá crearíamos un paquete

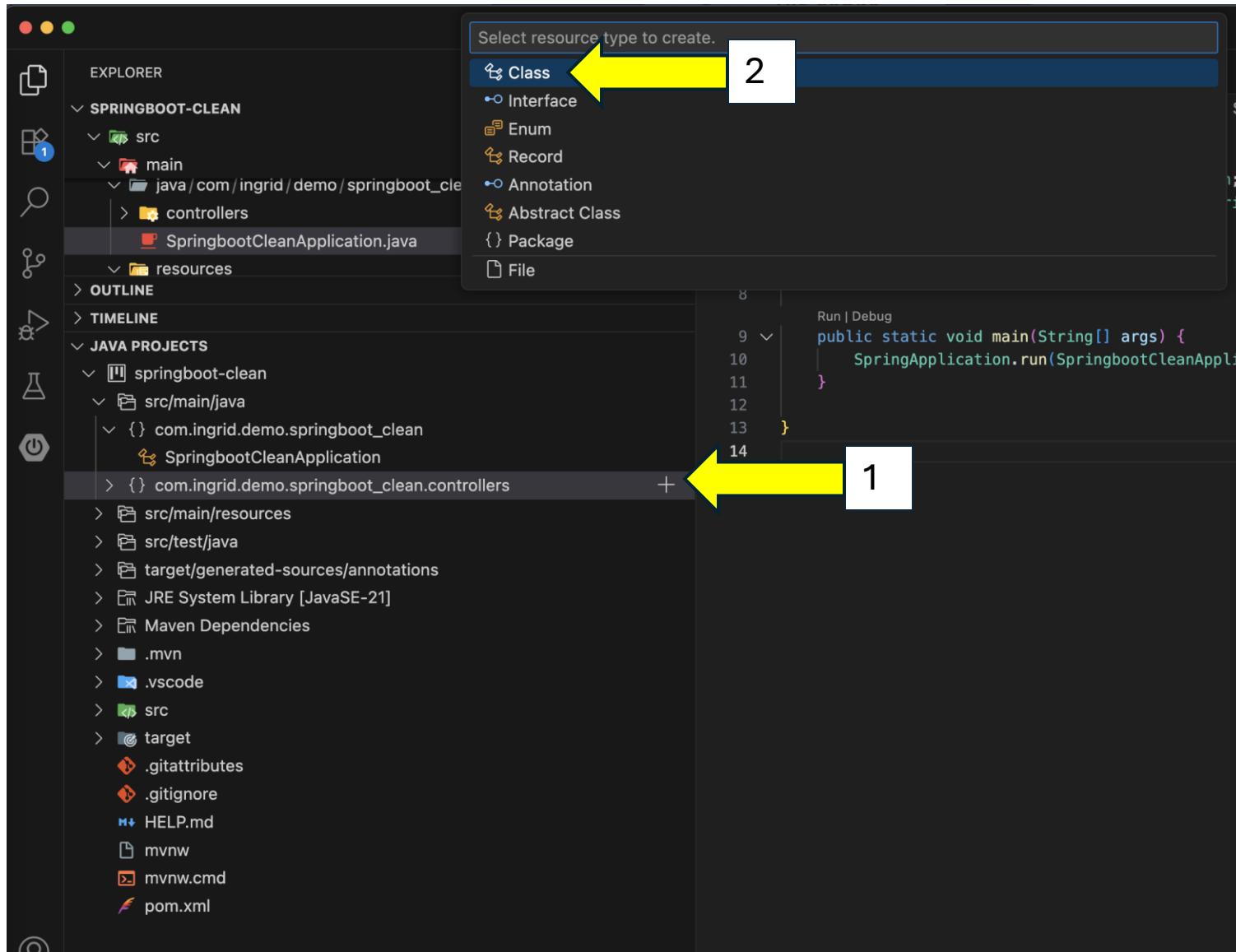
The screenshot shows a Java IDE interface with a dark theme. In the top right corner, there is a code completion dialog box with a blue border. Inside the dialog, the text "com.ingrid.demo.springboot_clean.|" is displayed, with the last character being a cursor. Below this, a message says "Press 'Enter' to confirm your input or 'Escape' to cancel". To the left of the dialog, the project explorer shows a tree structure under the heading "SPRINGBOOT-CLEAN". It contains a "src" folder, a "main" folder, and a "java/com/ingrid/demo/springboot_clean" folder. The "java/com/ingrid/demo/springboot_clean" folder is currently selected. On the right side of the interface, a code editor window displays the beginning of a Java file:

```
src > main > java > com > ingrid > demo > springboot_clean > SpringbootCleanApplication.java
1 package com.ingrid.demo.springboot_clean;
2
3 import org.springframework.boot.SpringApplication;
```

This screenshot shows the same Java IDE interface as the first one, but with a different focus. The code completion dialog now displays "com.ingrid.demo.springboot_clean.controllers|", with the last character being a cursor. The message "Press 'Enter' to confirm your input or 'Escape' to cancel" is still present. The project explorer on the left shows the same "SPRINGBOOT-CLEAN" structure. In the code editor on the right, the "SpringbootCleanApplication.java" file is shown with its full content:

```
src > main > java > com > ingrid > demo > springboot_clean > SpringbootCleanApplication.java > ...
1 package com.ingrid.demo.springboot_clean;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
```

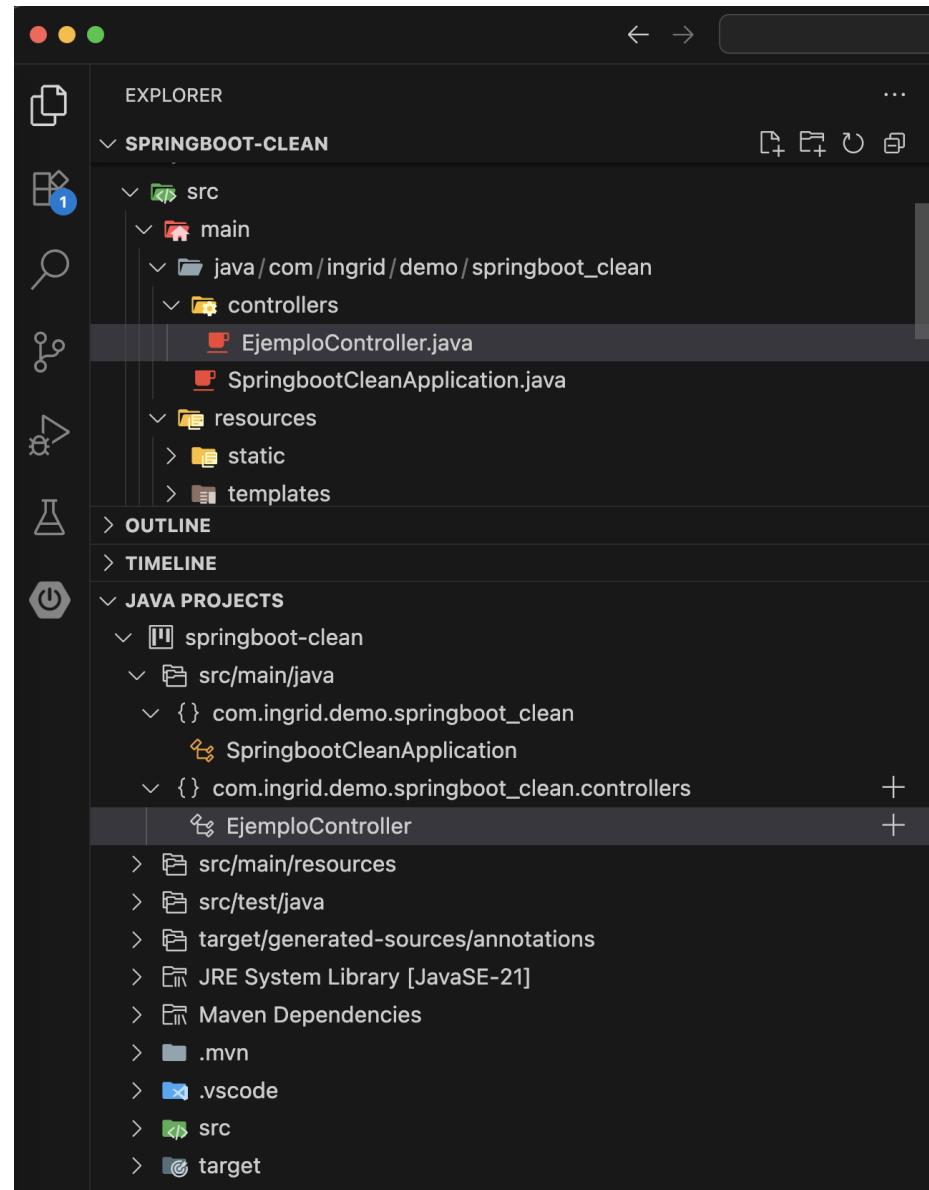




The screenshot shows a Java IDE interface with the following details:

- EXPLORER** view on the left, showing the project structure under **SPRINGBOOT-CLEAN**. It includes a file named **SpringbootCleanApplication.java** and a package named **com.ingrid.demo.springboot_clean.controllers**.
- Code Editor** view on the right, displaying the **SpringbootCleanApplication.java** file. The cursor is at the end of the class definition.
- A **Code Completion** dialog is open at the top right, with the text **EjemploController** entered. A tooltip says: "Press 'Enter' to confirm your input or 'Escape' to cancel".
- The code editor shows the following code:

```
src > main > java > com > ingrid > demo > springboot_clean > SpringbootCleanApplication.java
1 package com.ingrid.demo.springboot_clean;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class SpringbootCleanApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(SpringbootCleanApplication.class, args);
11     }
12 }
13
14 }
```



The screenshot shows the Visual Studio Code (VS Code) interface with a dark theme. The top bar includes standard window controls (red, yellow, green circles), a back/forward button, a search bar containing "springboot-clean", and a file icon.

The left sidebar (Explorer) shows the project structure:

- SPRINGBOOT-CLEAN**
 - src**
 - main**
 - java/com/ingrid/demo/springboot_clean**
 - controllers**
 - EjemploController.java**
 - SpringbootCleanApplication.java**
 - resources**
 - static**
 - templates**
 - OUTLINE**
 - TIMELINE**
 - JAVA PROJECTS**
 - springboot-clean**
 - src/main/java**
 - { } com.ingrid.demo.springboot_clean**
 - SpringbootCleanApplication**
 - { } com.ingrid.demo.springboot_clean.controllers**
 - EjemploController**
 - src/main/resources**
 - src/test/java**
 - target/generated-sources/annotations**
 - JRE System Library [JavaSE-21]**
 - Maven Dependencies**

```
SpringbootCleanApplication.java EjemploController.java
> ingrid > demo > springboot_clean > controllers > EjemploController.java > Language Support for Java
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class EjemploController {
7     // Aquí puedes agregar métodos para manejar las solicitudes HTTP
8     // Ejemplo: @GetMapping("/ejemplo")
9     // public String ejemplo() {
10     //     return "vistaEjemplo";
11     // }
12 }
```

SpringbootCleanApplication.java

EjemploController.java

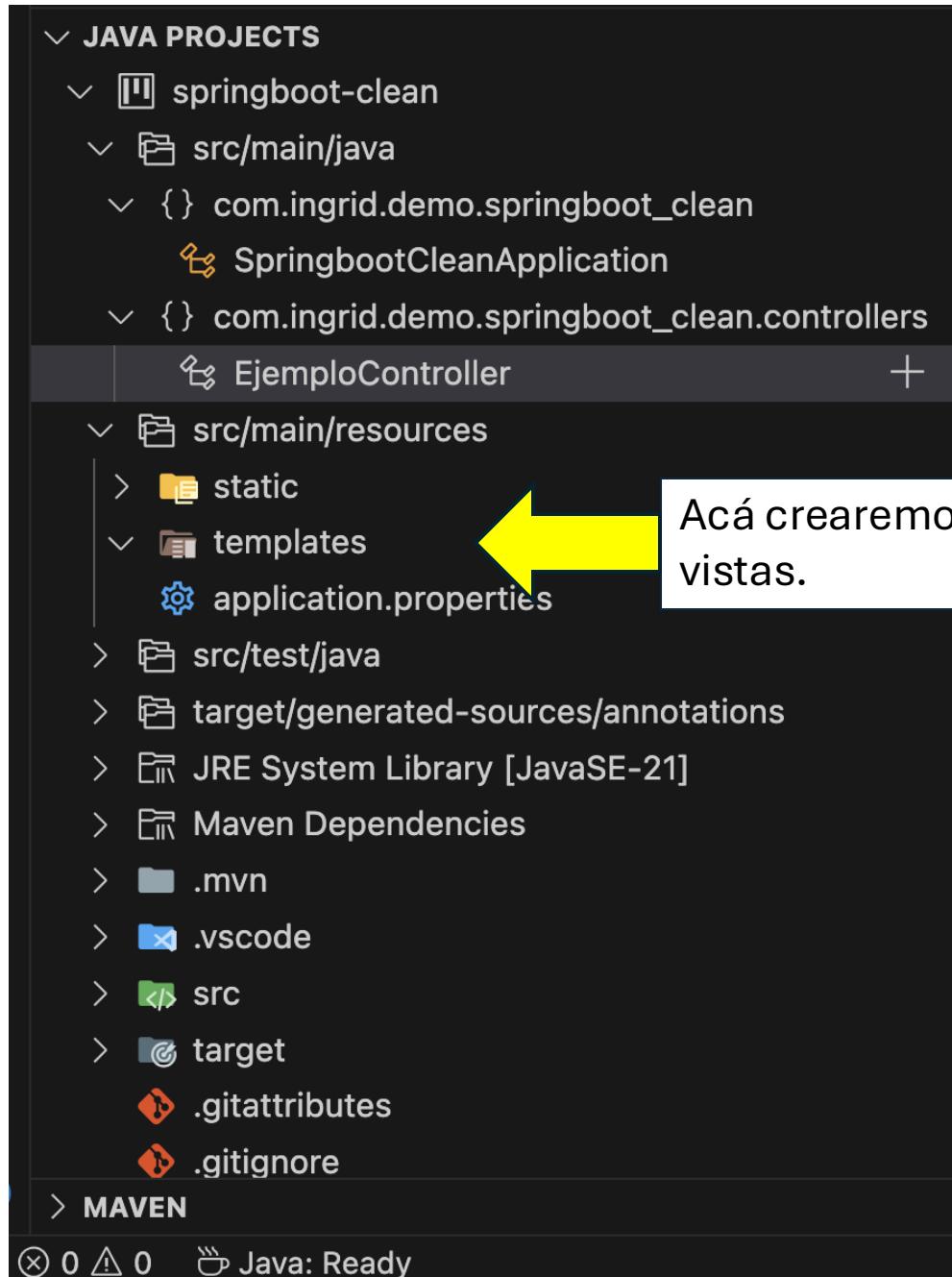
```
_clean > controllers > EjemploController.java > Language Support for Java(TM) by Red Hat > EjemploController.java
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestParam;
6
7
8 @Controller
9 public class EjemploController {
10
11     @GetMapping("path/to/method")
12     public String getMethodName(@RequestParam String param) {
13         return new String();
14     }
15
16     public String info() {
17         return "detalle_info";
18     }
19
20 }
21
```

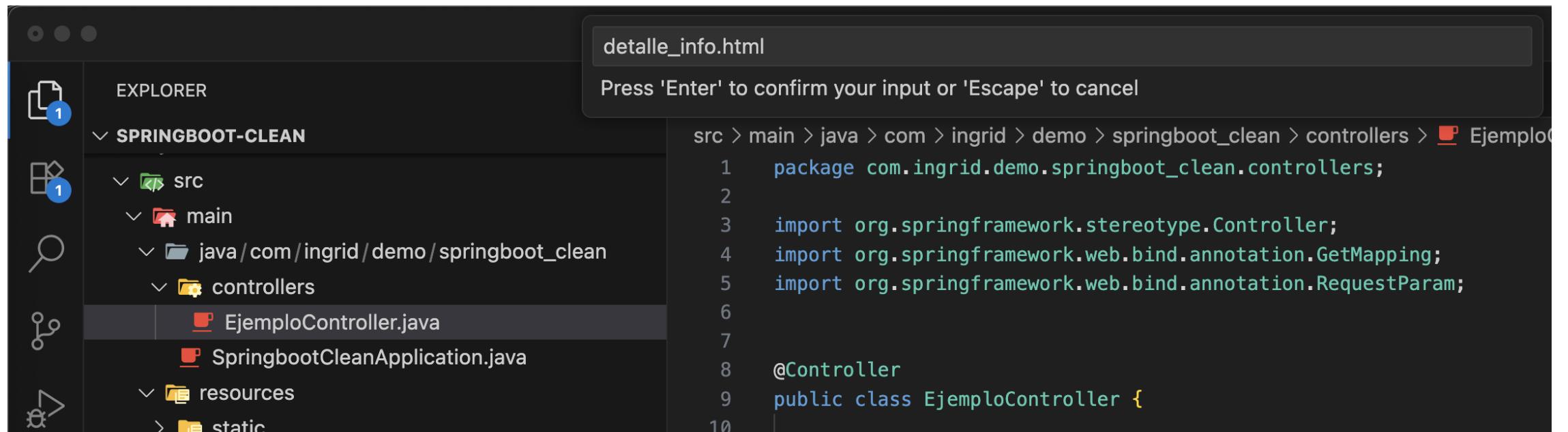
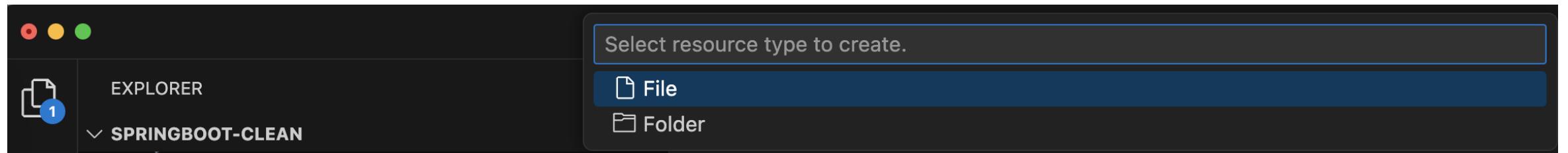
SpringbootCleanApplication.java

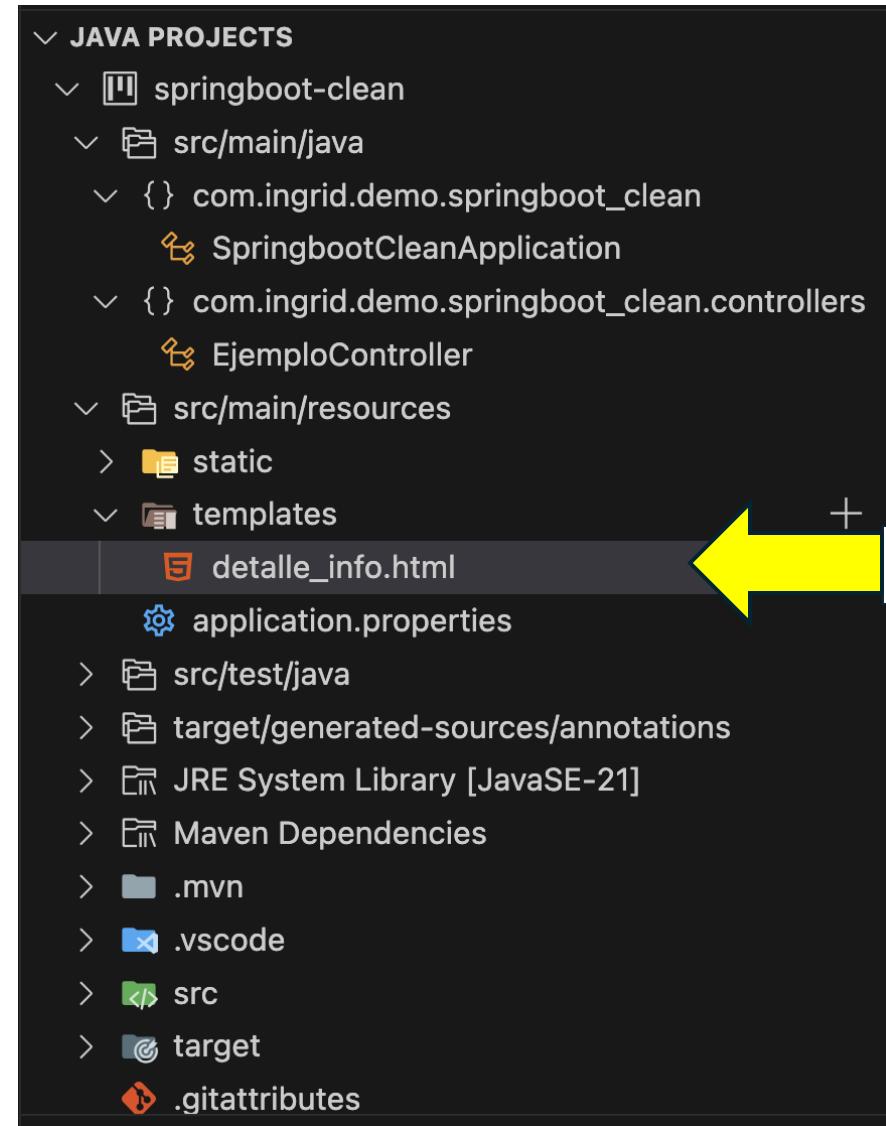
EjemploController.java

src > main > java > com > ingrid > demo > springboot_clean > controllers > EjemploController.java > ...

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestParam;
6
7
8 @Controller
9 public class EjemploController {
10
11     @GetMapping("/detalle_info")
12     public String getMethodName(@RequestParam String param) {
13         return new String();
14     }
15
16     public String info() {
17         return "detalle_info";
18     }
19
20 }
```







Creado nuestro archivo

SpringbootCleanApplication.java

EjemploController.java

detalle_info.html

src > main > resources > templates > detalle_info.html > html

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Document</title>
7  </head>
8  <body>
9
10 </body>
11 </html>
```

```
src > main > resources > templates > 5 detalle_info.html > html
1   <!DOCTYPE html>
2   <html lang="en">
3   <head>
4       <meta charset="UTF-8">
5       <meta name="viewport" content="width=device-width, initial-scale=1.0">
6       <title>Pagina Spring Boot</title>
7   </head>
8   <body>
9       <h1>Servidor en linea</h1>
10
11  </body>
12  </html>
```

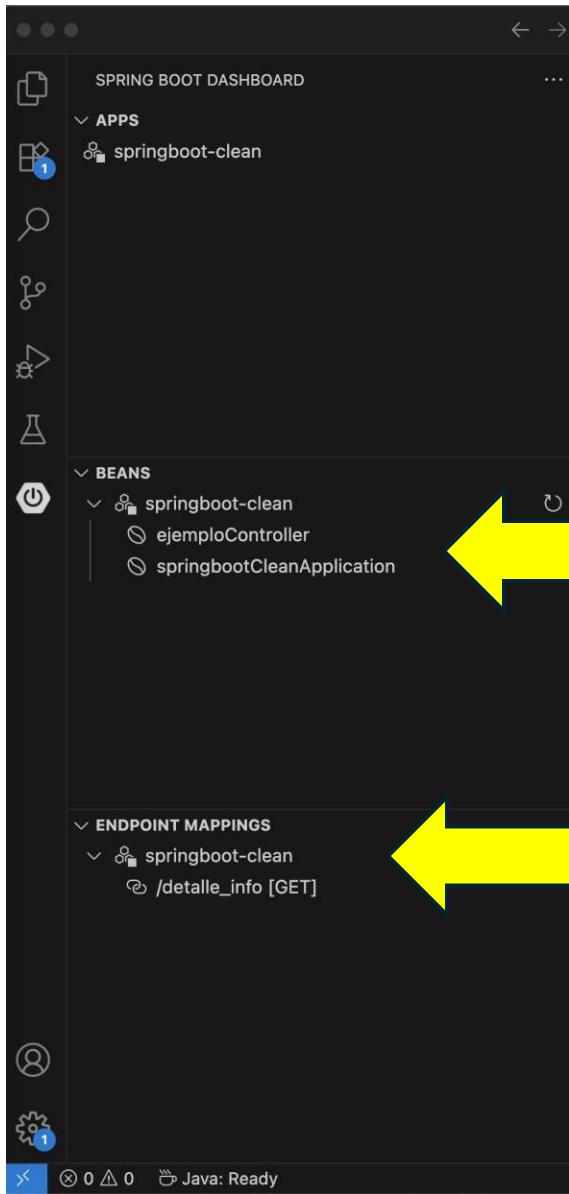
SpringbootCleanApplication.java

EjemploController.java

detalle_info.html

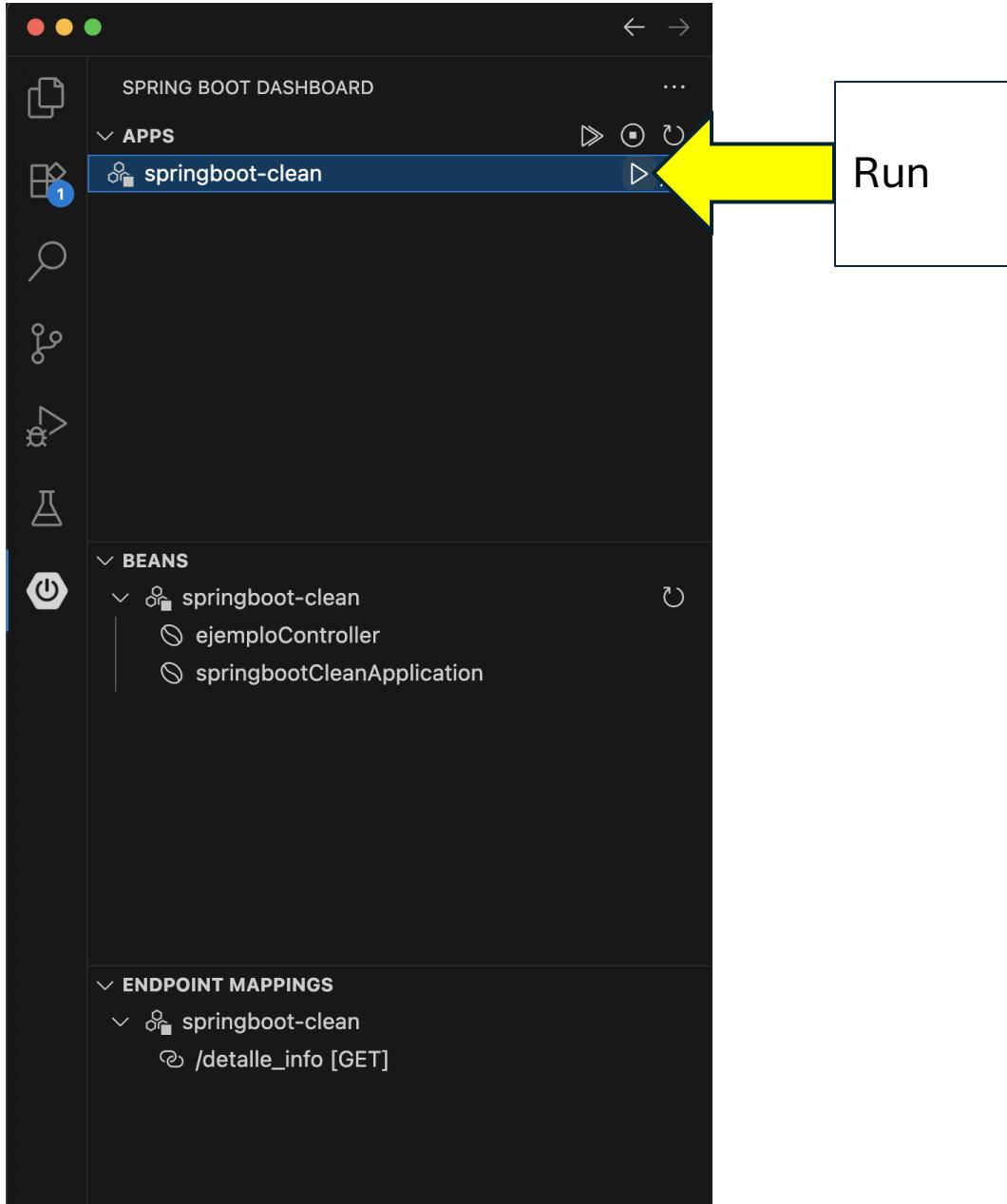
src > main > java > com > ingrid > demo > springboot_clean > controllers > EjemploController.java > ...

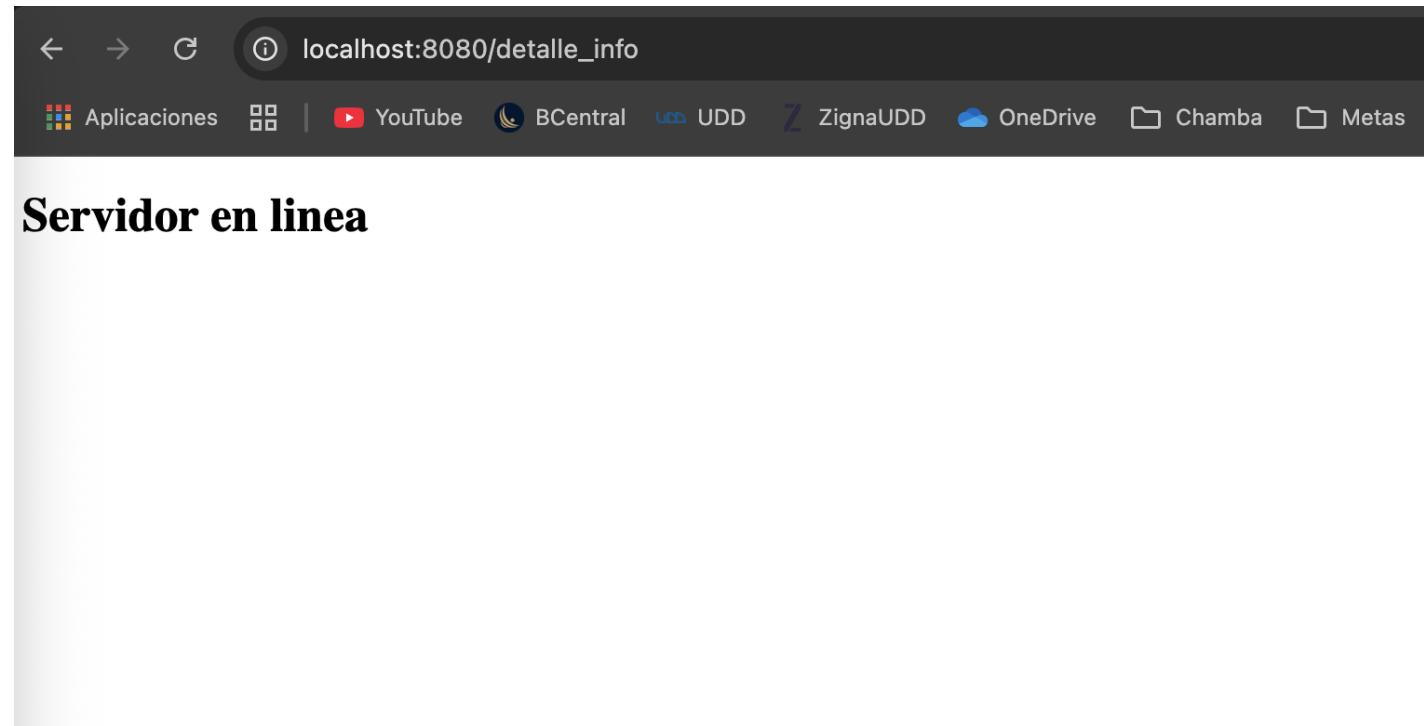
```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5
6
7 @Controller
8 public class EjemploController {
9
10    @GetMapping("/detalle_info")
11    public String info() {
12        return "detalle_info";
13    }
14
15 }
16
```



2. beans: componentes manejados por framework

Metodos





Primer levantamiento de ejecución de proyecto.

The screenshot shows a code editor interface with a dark theme. The top navigation bar includes File, Edit, Selection, View, Go, Run, and a search bar containing "springboot-app1". Below the navigation is the Explorer sidebar, which lists the project structure under "SPRINGBOOT-APP1": src, main, java, com, informaticoconfig, spring, app1, resources, static, templates, detalles_info.html, application.properties, test, and target. The "application.properties" file is currently selected and open in the main editor area. The file contains the following configuration:

```
server.port = 9525
spring.application.name=springboot-app1
```

En caso de tener problemas con el puerto.

The screenshot shows a Java IDE interface with the following tabs at the top:

- SpringbootCleanApplication.java
- EjemploController.java (selected tab)
- detalle_info.html
- application.properties

The file path in the header bar is: src > main > java > com > ingrid > demo > springboot_clean > controllers > EjemploController.java > ...

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6
7
8 @Controller
9 public class EjemploController {
10
11     @GetMapping("/detalle_info")
12     public String info(Model model) {
13         model.addAttribute("Titulo", "Sevidor en linea");
14         model.addAttribute("Servidor", "Este es un ejemplo de controlador en Spring Boot");
15         model.addAttribute("Autor", "Ingrid");
16         //Patron de inyeccion de dependencias
17         return "detalle_info";
18     }
19
20 }
21
```

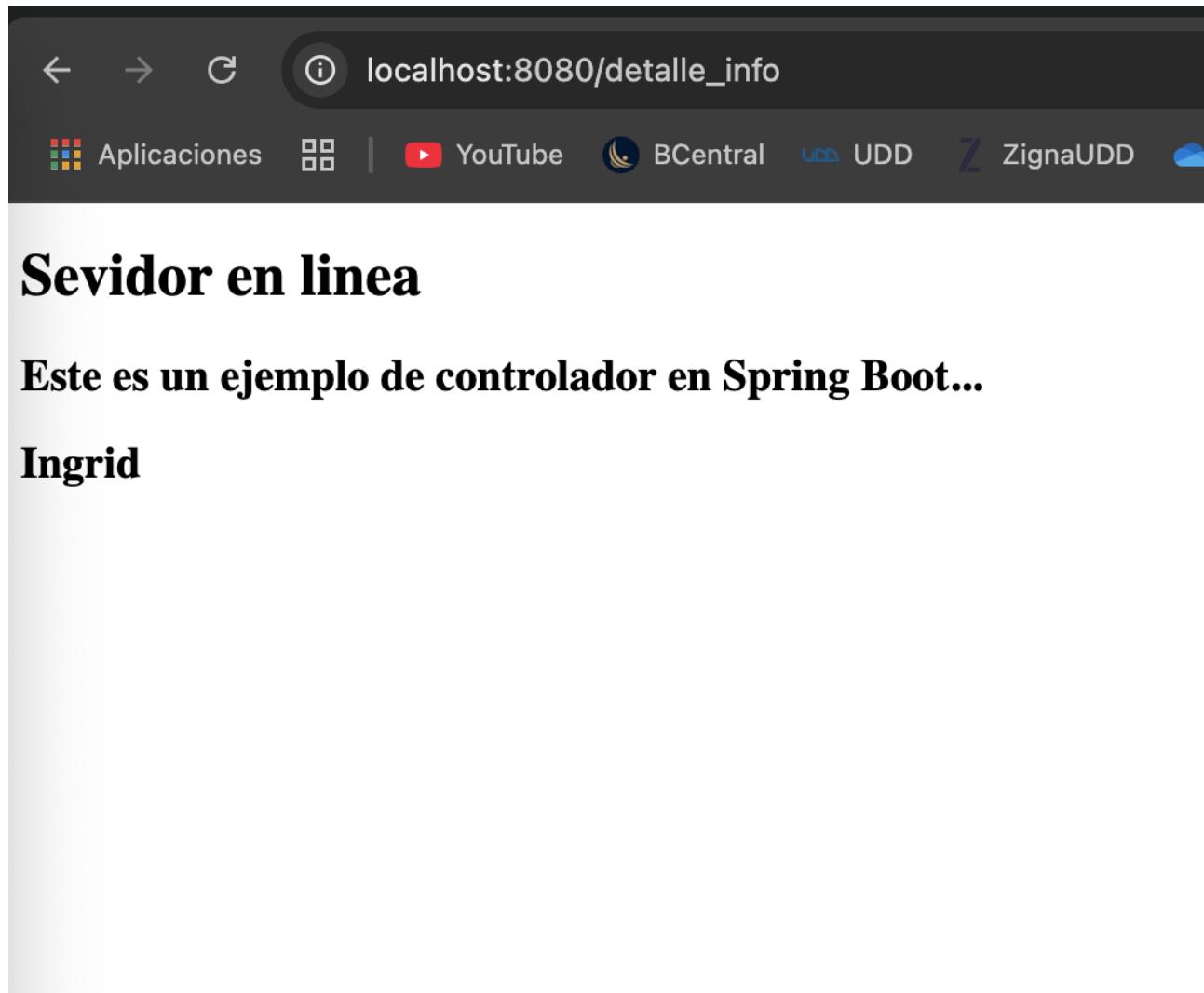
The screenshot shows a code editor interface with three tabs at the top: 'SpringbootCleanApplication.java', 'EjemploController.java', and 'detalle_info.html'. The 'detalle_info.html' tab is currently active, indicated by a blue border and a white dot in the tab bar. Below the tabs, a breadcrumb navigation bar shows the file path: 'src > main > resources > templates > detalle_info.html > html'. The main content area displays the following HTML code:

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Pagina Spring Boot</title>
7  </head>
8  <body>
9      <h1>Servidor en linea</h1>
10
11 </body>
12 </html>
```

Thymeleaf => Motor de plantillas en java para la generación de nuestras vistas dinámicas con nuestras aplicaciones de Spring

The screenshot shows a code editor interface with three tabs at the top: 'SpringbootCleanApplication.java', 'EjemploController.java', and 'detalle_info.html'. The 'detalle_info.html' tab is active, displaying a Thymeleaf template file. The file path 'src > main > resources > templates > detalle_info.html' is shown above the code area. The code itself is a simple HTML document with Thymeleaf directives for styling and content:

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title th:text="${Titulo}"> Pagina Spring Boot</title>
7  </head>
8  <body>
9      <h1 th:text="${Titulo}"></h1>
10     <h2 th:text="${Servidor}"></h2>
11     <h2 th:text="${Autor}"></h2>
12
13 </body>
14 </html>
```

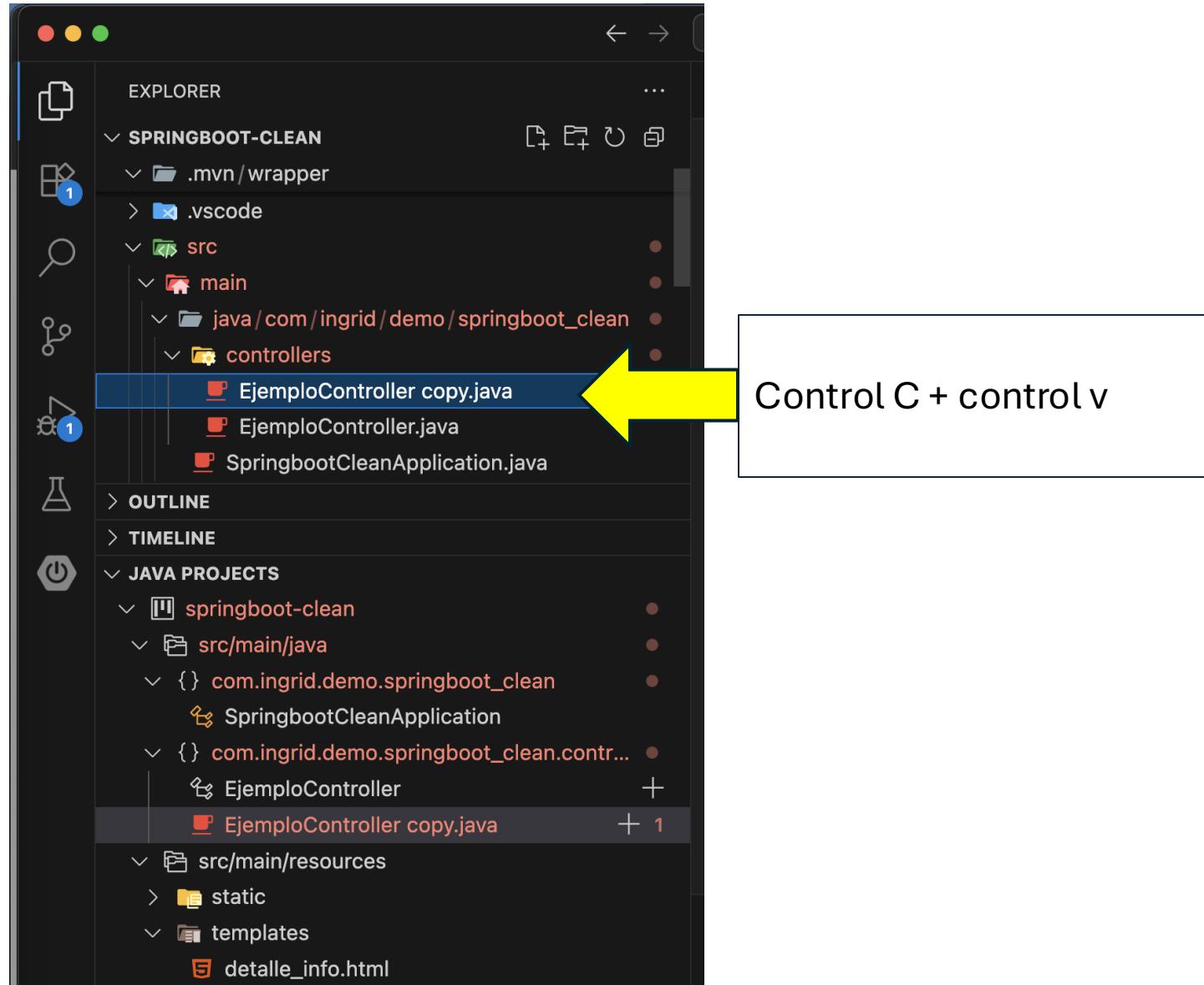


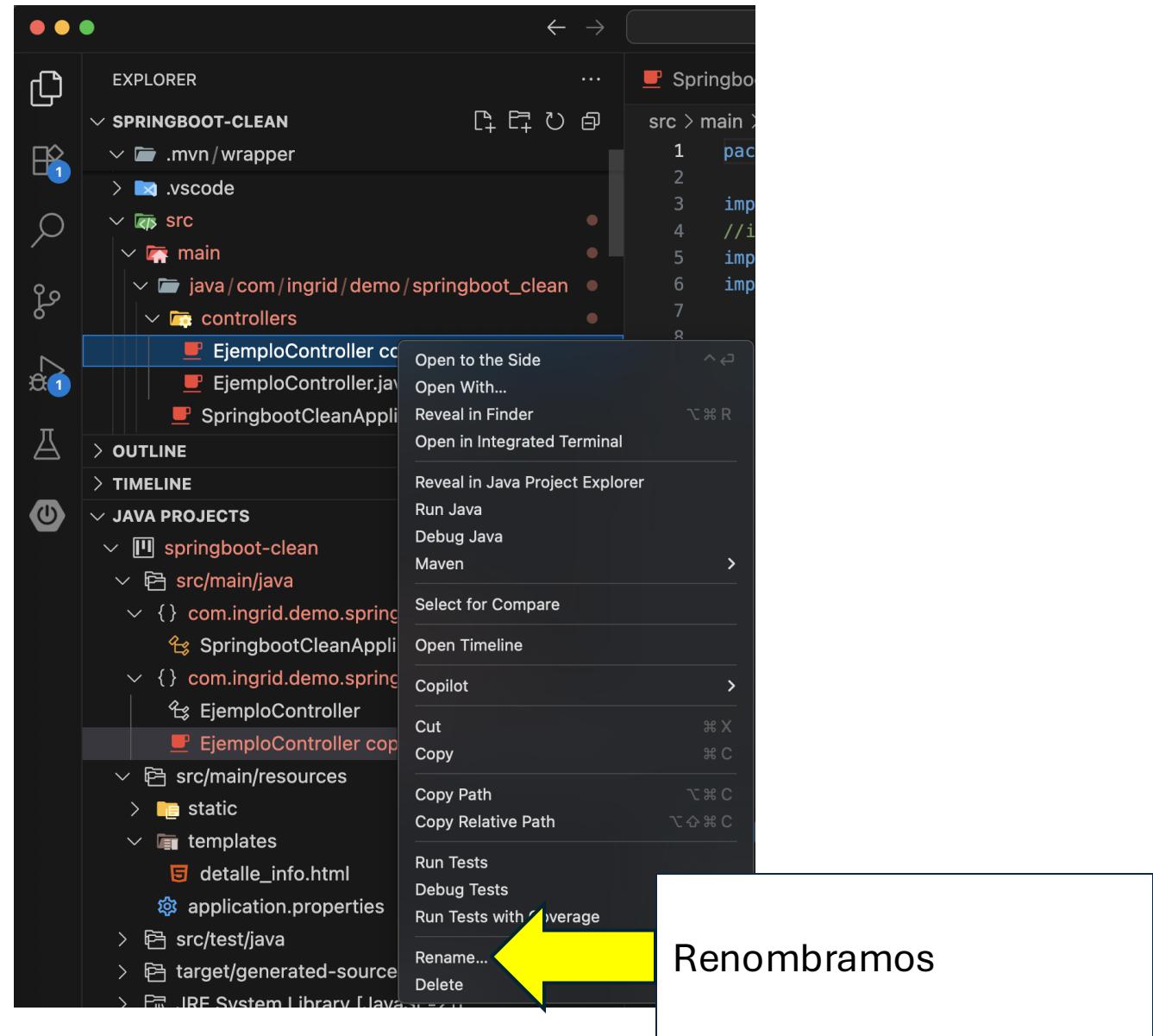
Pasar datos a una vista
generada por un
controlador

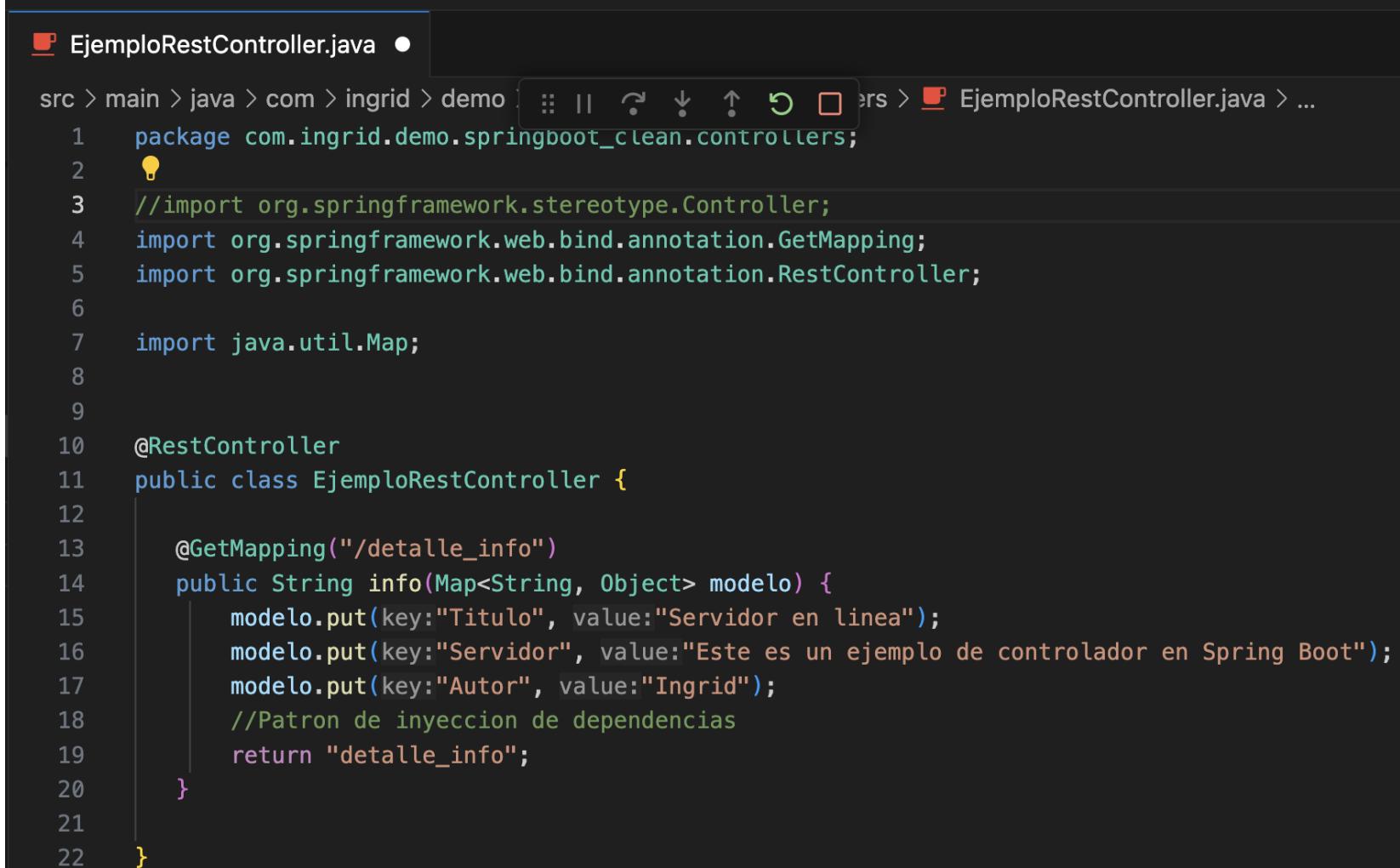
src > main > java > com > ingrid > demo > springboot_clean > Controller.java > ...

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 //import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import java.util.Map;
7
8
9 @Controller
10 public class EjemploController {
11
12     @GetMapping("/detalle_info")
13     public String info(Map<String, Object> modelo) {
14         modelo.put(key:"Titulo", value:"Servidor en linea");
15         modelo.put(key:"Servidor", value:"Este es un ejemplo de controlador en Spring Boot...");
16         modelo.put(key:"Autor", value:"Ingrid");
17         //Patron de inyeccion de dependencias
18         return "detalle_info";
19     }
20
21 }
22 }
```

RestController -> Una notación para definir un controlador que maneja solicitudes http y que produce respuestas directamente en formato json / xml







```
EjemploRestController.java ●
src > main > java > com > ingrid > demo > EjemploRestController.java > ...
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 //import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 import java.util.Map;
8
9
10 @RestController
11 public class EjemploRestController {
12
13     @GetMapping("/detalle_info")
14     public String info(Map<String, Object> modelo) {
15         modelo.put(key:"Titulo", value:"Servidor en linea");
16         modelo.put(key:"Servidor", value:"Este es un ejemplo de controlador en Spring Boot");
17         modelo.put(key:"Autor", value:"Ingrid");
18         //Patron de inyeccion de dependencias
19         return "detalle_info";
20     }
21
22 }
```

Nuestro método
Hadler se convierte a
un método Handler
rest

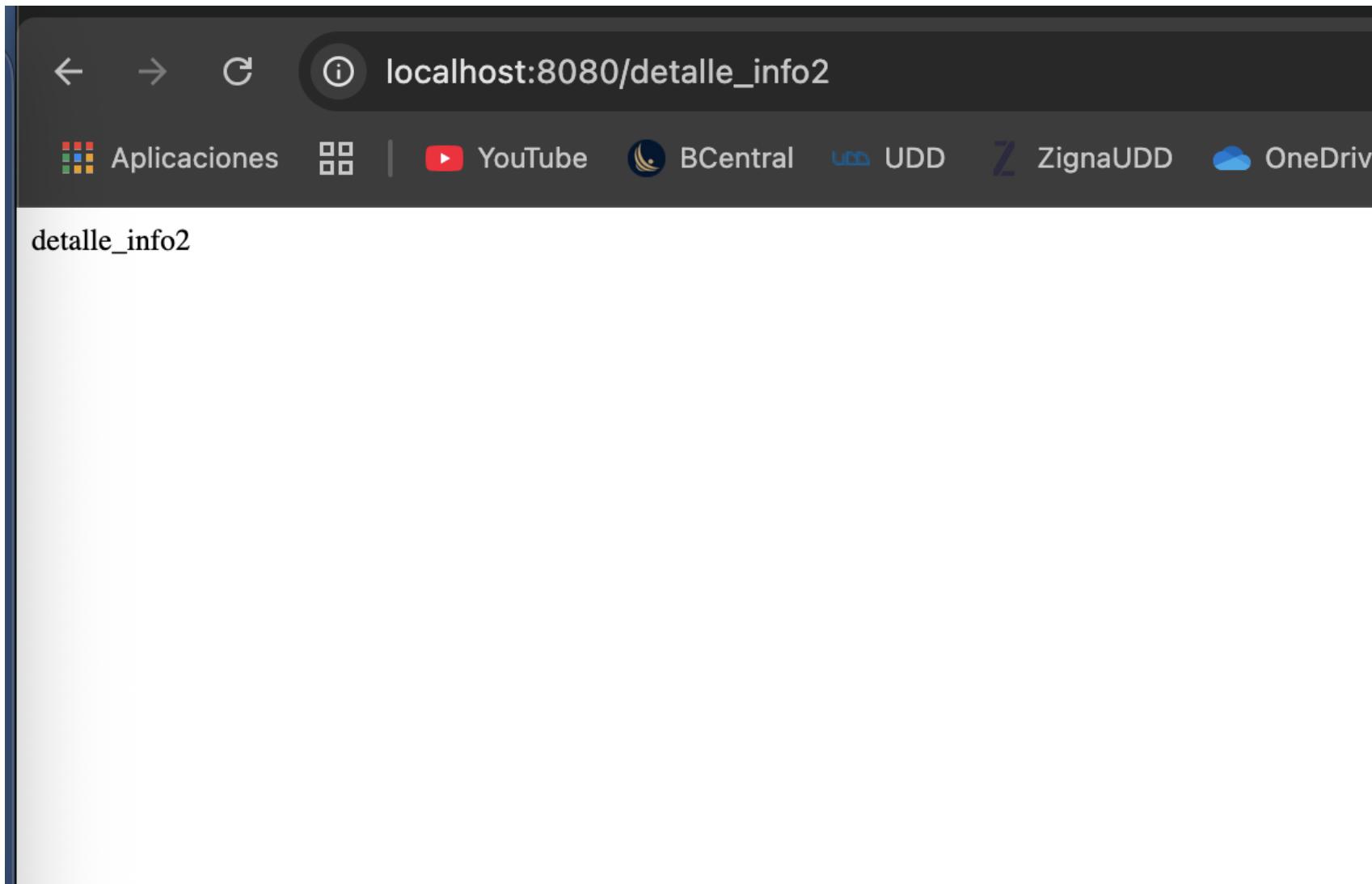
Estamos convirtiendo este nuevo controlador en un Api Rest

Api Rest un estilo de construcción de interfaces de programación de aplicaciones que permite la comunicación entre sistemas a través de solicitudes http.

EjemploRestController.java

Springboot_clean > controllers > EjemploRestController.java Support for Java(TM) by Red Hat > EjemploRestController.java

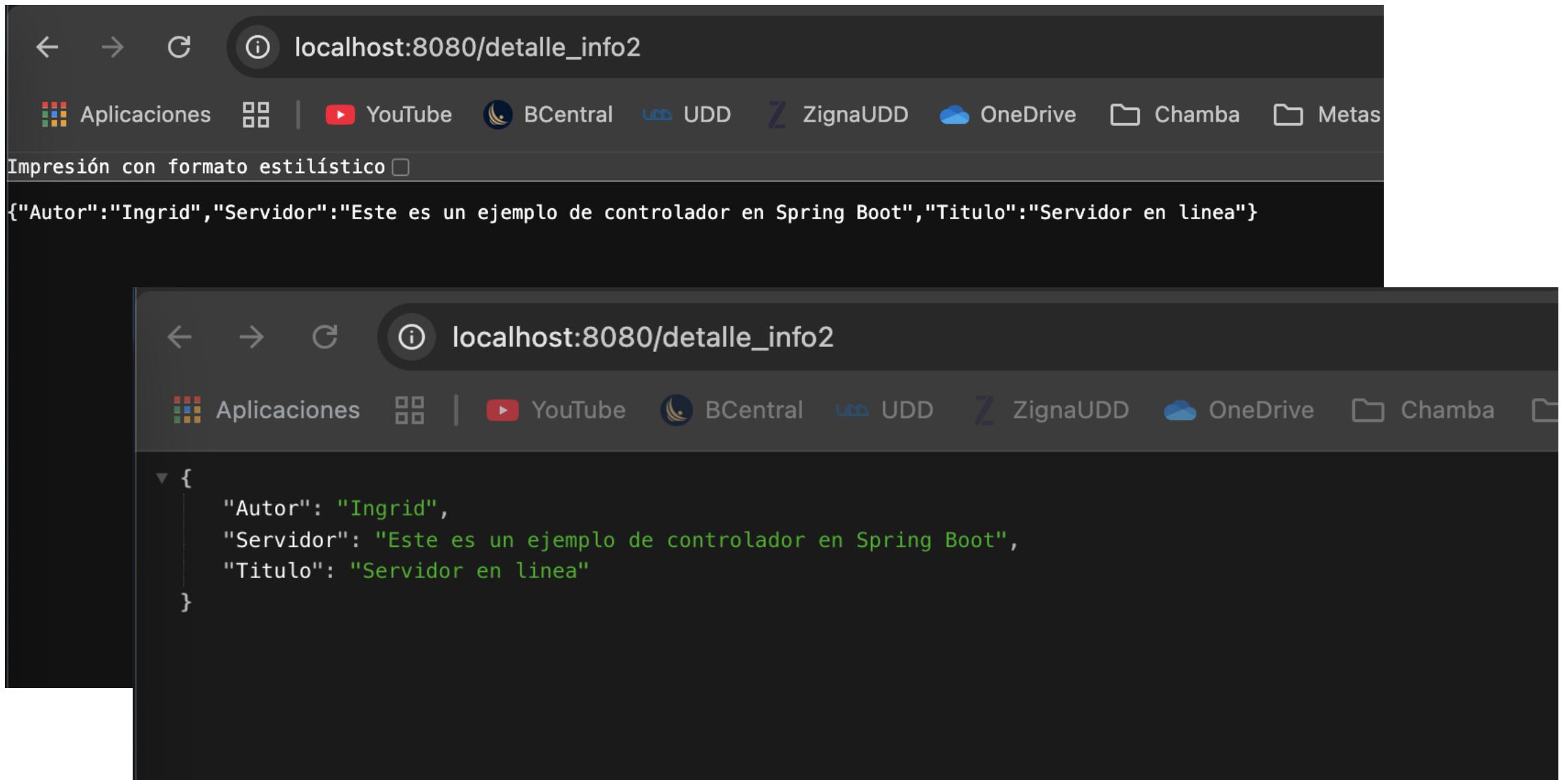
```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 //import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 import java.util.Map;
8 import java.util.HashMap;
9
10
11 @RestController
12 public class EjemploRestController {
13     @GetMapping("/detalle_info2")
14     public String info() {
15         Map<String, Object> respuesta = new HashMap<>();
16         respuesta.put(key:"Titulo", value:"Servidor en linea");
17         respuesta.put(key:"Servidor", value:"Este es un ejemplo de controlador en Spring Boot");
18         respuesta.put(key:"Autor", value:"Ingrid");
19         //Patron de inyeccion de dependencias
20         return "detalle_info2";
21     }
22 }
23
24 }
```



EjemploRestController.java X

for Java(TM) by Red Hat > EjemploRestController > detalles_info2

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 //import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 import java.util.Map;
8 import java.util.HashMap;
9
10
11 @RestController
12 public class EjemploRestController {
13
14     @GetMapping("/detalle_info2")
15     public Map<String, Object> detalles_info2() {
16         Map<String, Object> respuesta = new HashMap<>();
17         respuesta.put(key:"Titulo", value:"Servidor en linea");
18         respuesta.put(key:"Servidor", value:"Este es un ejemplo de controlador en Spring Boot");
19         respuesta.put(key:"Autor", value:"Ingrid");
20         //Patron de inyeccion de dependencias
21         return respuesta;
22     }
23
24 }
25
```



The image shows two screenshots of a web browser window in dark mode, displaying a JSON response from a Spring Boot application.

The top screenshot shows the raw JSON output:

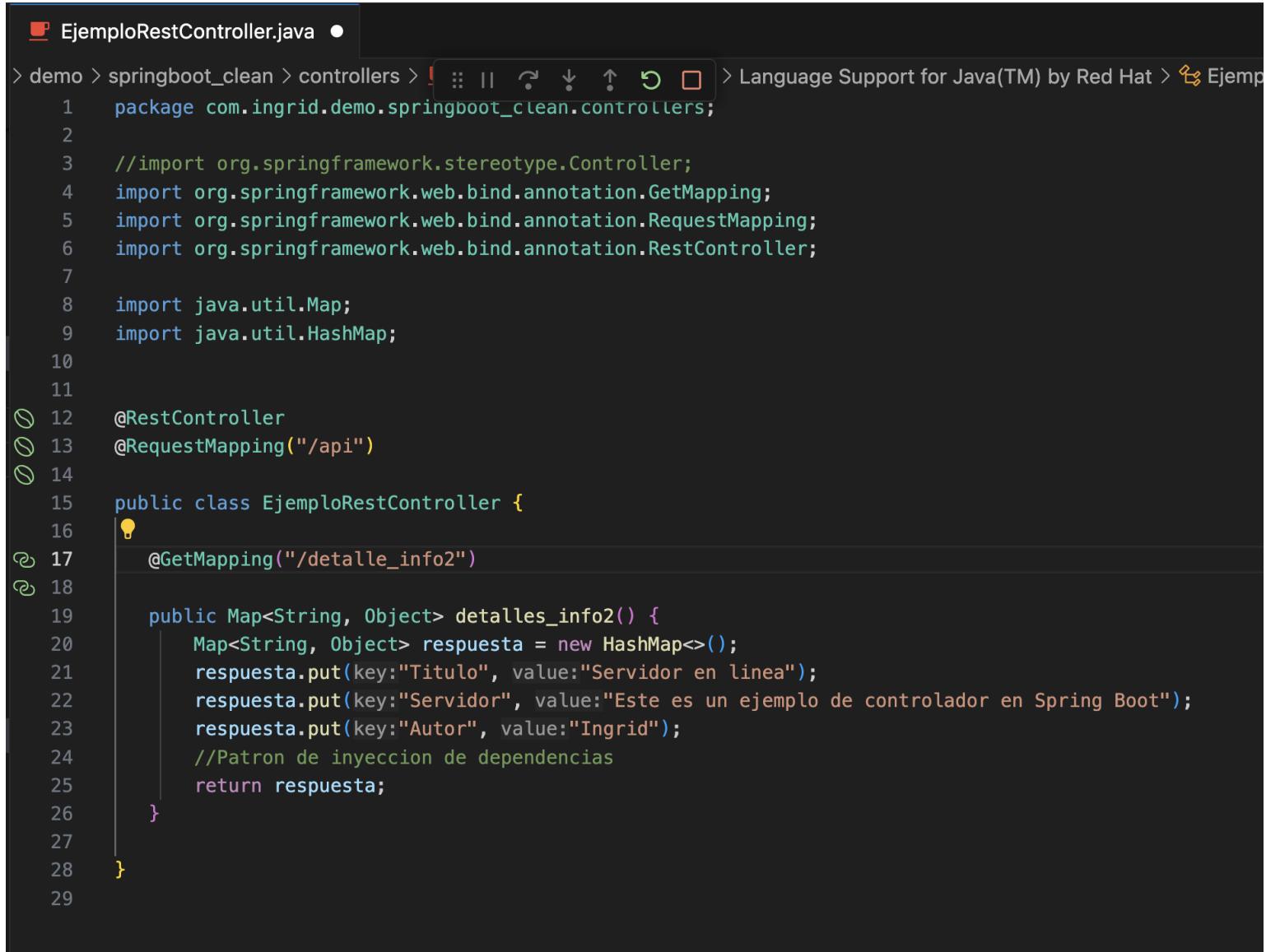
```
{"Autor": "Ingrid", "Servidor": "Este es un ejemplo de controlador en Spring Boot", "Titulo": "Servidor en linea"}
```

The bottom screenshot shows the JSON output with syntax highlighting and collapsible sections:

```
▼ {  
    "Autor": "Ingrid",  
    "Servidor": "Este es un ejemplo de controlador en Spring Boot",  
    "Titulo": "Servidor en linea"  
}
```

@RequestMapping: Anotación que se utiliza para mapear solicitudes http a métodos específicos en una clase controlada.

Vamos a poder definir solicitudes entrantes que son basadas en criterios como los métodos get o métodos post, put o delete.



```
> demo > springboot_clean > controllers > EjemploRestController.java ●
> Language Support for Java(TM) by Red Hat > EjemploRestController.java

 1 package com.ingrid.demo.springboot_clean.controllers;
 2
 3 //import org.springframework.stereotype.Controller;
 4 import org.springframework.web.bind.annotation.GetMapping;
 5 import org.springframework.web.bind.annotation.RequestMapping;
 6 import org.springframework.web.bind.annotation.RestController;
 7
 8 import java.util.Map;
 9 import java.util.HashMap;
10
11
12 @RestController
13 @RequestMapping("/api")
14
15 public class EjemploRestController {
16
17     @GetMapping("/detalle_info2")
18
19     public Map<String, Object> detalles_info2() {
20         Map<String, Object> respuesta = new HashMap<>();
21         respuesta.put(key:"Titulo", value:"Servidor en linea");
22         respuesta.put(key:"Servidor", value:"Este es un ejemplo de controlador en Spring Boot");
23         respuesta.put(key:"Autor", value:"Ingrid");
24         //Patron de inyeccion de dependencias
25         return respuesta;
26     }
27
28 }
29
```

Y con esto ya nuestro detalles_info2, se convierte en un api rest

Creamos una ruta de primer nivel, pues esta encima del controlador

← → C



localhost:8080/api/detalle_info2

Aplicaciones



YouTube



BCentral



UDD



ZignaUDD



On

▼ {

```
"Autor": "Ingrid",
"Servidor": "Este es un ejemplo de controlador en Spring Boot",
"Titulo": "Servidor en linea"
```

}

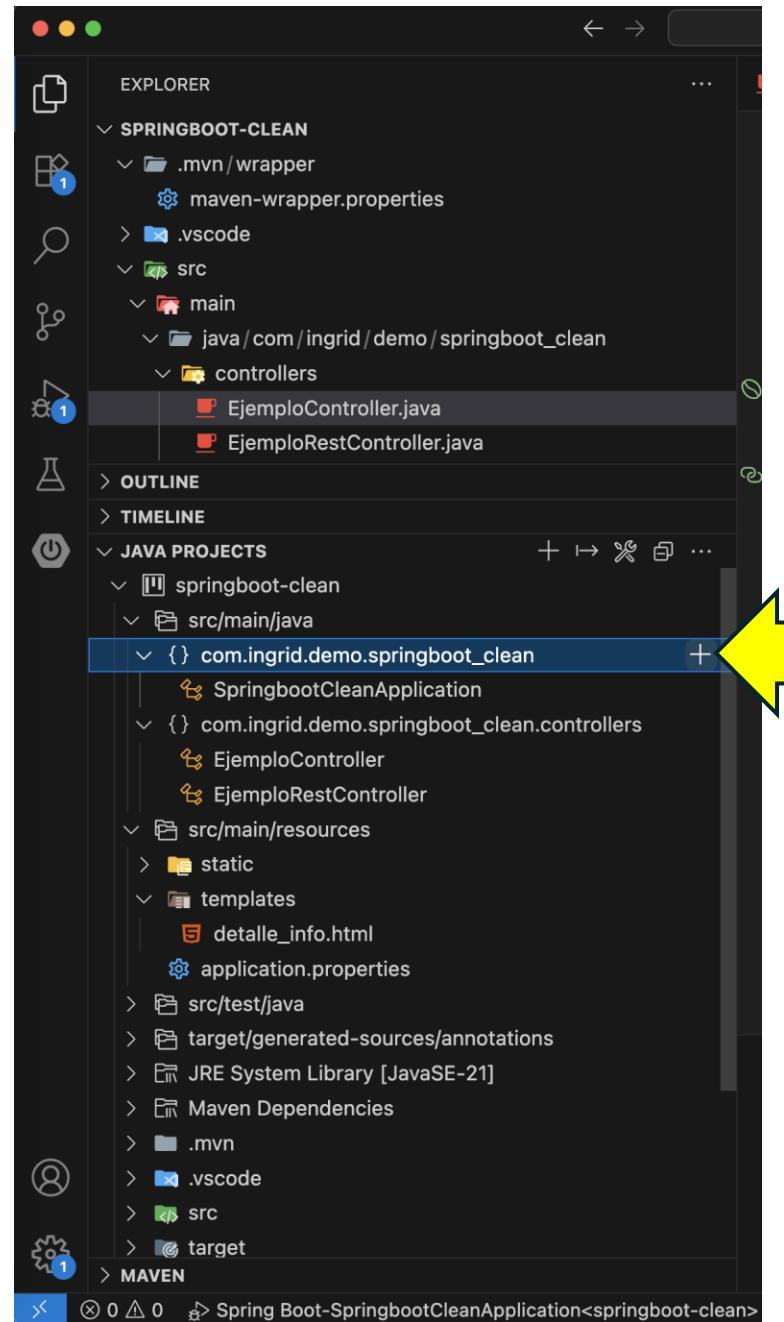
El @RequestMapping trabaja con solicitudes del tipo GET

The diagram illustrates the MVC (Model-View-Controller) pattern. It shows three components: Controlador (Controller), Vista (View), and Modelo (Model). The Controller is represented by a yellow arrow pointing to the '@Controller' annotation in the code. The View is represented by a yellow arrow pointing to the '@GetMapping' annotation. The Model is represented by a yellow arrow pointing to the 'Map<String, Object>' parameter in the 'info' method. The code itself is a Java file named 'EjemploController.java'.

```
src > main > java > com > ingrid > demo > EjemploController.java > ...
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 //import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import java.util.Map;
7
8
9 @Controller
10 public class EjemploController {
11
12     @GetMapping("/detalle_info")
13     public String info(Map<String, Object> modelo) {
14         modelo.put(key:"Titulo", value:"Servidor en linea");
15         modelo.put(key:"Servidor", value:"Este es un ejemplo");
16         modelo.put(key:"Autor", value:"Ingrid");
17         //Patron de inyeccion de dependencias
18         return "detalle_info";
19     }
20
21 }
22
```

Objeto Model es un parámetro que se usa en los métodos de los controladores.

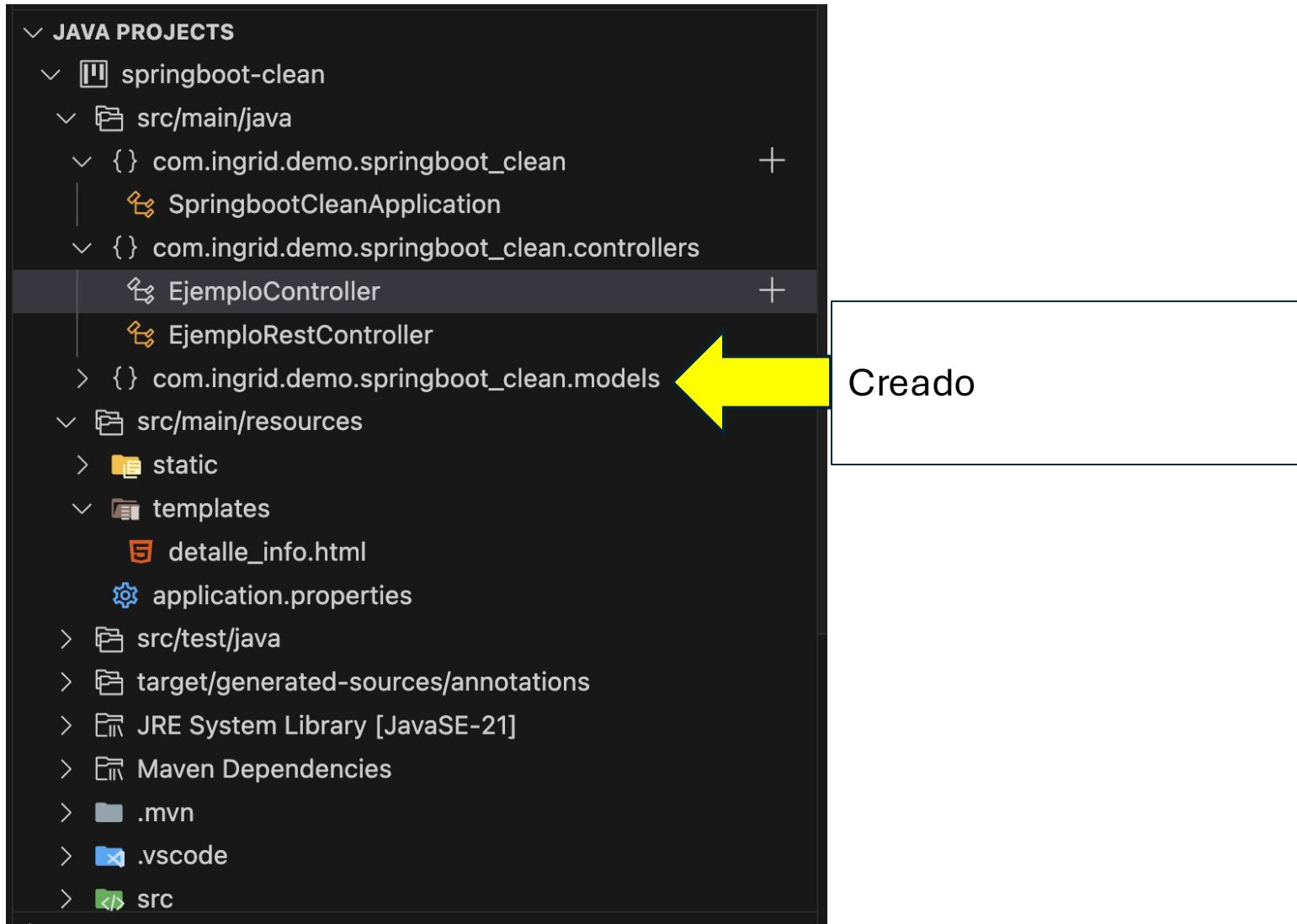
Spring lo que hace es que inyecta automáticamente el objeto model cuando se define en la declaración de un método vamos a ver un ejemplo.

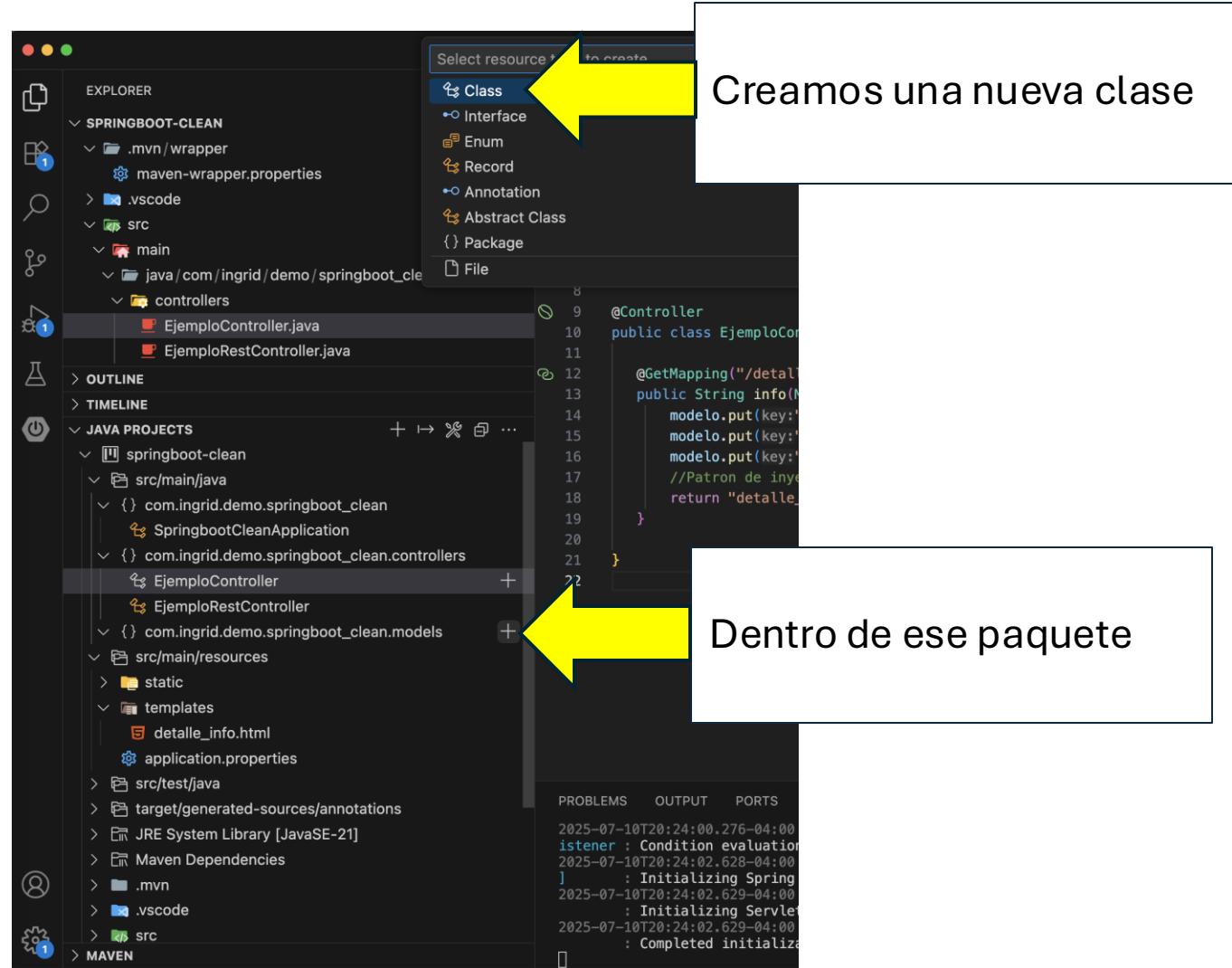


Crear un nuevo paquete

The screenshot shows the Visual Studio Code interface. On the left is the Explorer sidebar with project files like '.mvn/wrapper', '.vscode', 'src', 'main', and 'java/com/ingrid/demo/springboot_clean/controllers'. A floating search bar at the top has the text 'com.ingrid.demo.springboot_clean.models' and a placeholder 'Input the package name'. The main editor area displays Java code:

```
src > main > java > com > ingrid >
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 //import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import java.util.Map;
7
8
```





EjemploRestController.java

EjemploController.java

Empleados.java 7 ●

src > main > java > com > ingrid > demo > springboot_clean > models > Empleados.java > ...

```
1 package com.ingrid.demo.springboot_clean.models;
2
3 public class Empleados {
4
5     private String nombre, apellido, direccion, puesto;
6     private int edad, telefono, id;
7
8 }
9
```

The screenshot shows a dark-themed code editor in VS Code. The current file is `Empleados.java`, which contains the following code:

```
src > main > java > com > ingrid > demo > springboot_clean > models > Empleados.java > Empleados.java 7 ●
1 package com.ingrid.demo.springboot_clean.models;
2
3 public class Empleados {
4
5     private String nombre, apellido, direccion, puesto;
6     private int edad, telefono, id;
7
8 }
9
10 }
```

A context menu is open over the line containing the variable declarations (lines 5-6). The menu includes the following items:

- Go to Definition F12
- Go to Declaration
- Go to Type Definition
- Go to Implementations ⌘ F12
- Go to References ⌘ F12
- Go to Super Implementation
- Go to Test
- Peek >
- Find All References ⌘ ⌄ F12
- Find All Implementations
- Show Call Hierarchy ⌘ ⌄ H
- Show Type Hierarchy

Below the context menu, there are additional options:

- Copilot >
- Rename Symbol F2
- Change All Occurrences ⌘ F2
- Format Document ⌘ ⌄ F
- Format Document With... ⌘ ⌄ R
- Refactor... ⌘ ⌄ R
- Source Action...

The bottom left of the interface shows the **PROBLEMS** tab with 7 issues and the **OUTPUT** tab. The **OUTPUT** tab displays log messages from a Spring Boot application:

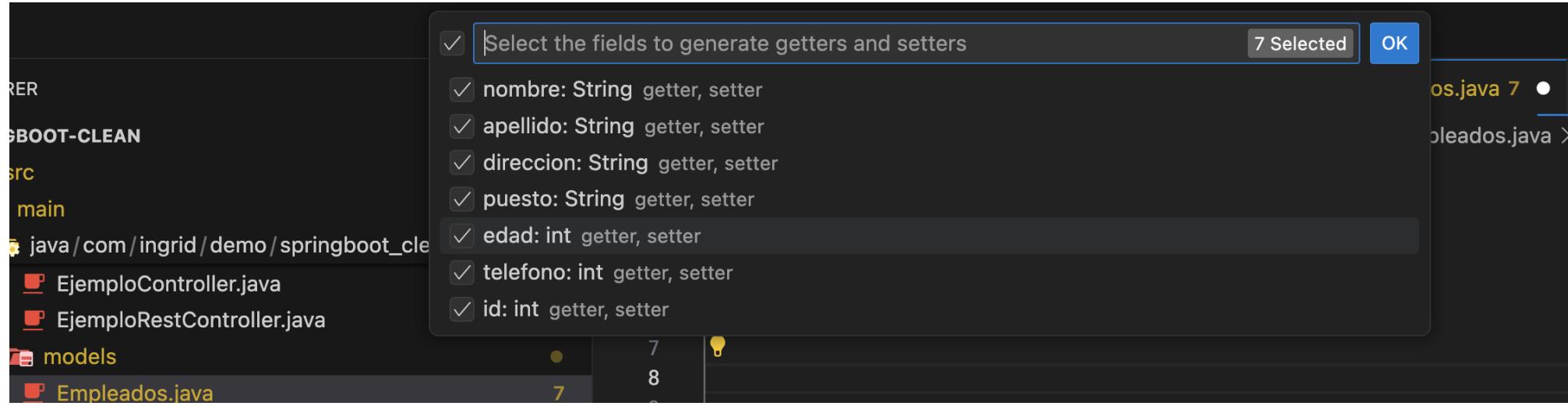
```
2025-07-10T21:00:14.162Z [main] INFO o.s.boot.SpringApplication - Application 'SpringbootCleanApp' started successfully (started in 0.111 seconds)
```

The bottom right shows the status bar with the message "Ln 8, Col 1 Spaces: 4".

EjemploRestController.java EjemploController.java Empleados.java 7 ●

src > main > java > com > ingrid > demo > springboot_clean > models > Empleados.java > Empleados

```
1 package com.ingrid.demo.springboot_clean.models;
2
3 public class Empleados {
4
5     private String nombre, apellido, direccion, puesto;
6     private int edad, telefono, id;
7
8
9     Source Action
10
11     ⌂ Generate Tests...
12     ⌂ Organize imports ⌂
13     ⌂ Generate Getters and Setters... Enter to Apply, ⌂ Enter to Preview
14     ⌂ Generate Getters...
15     ⌂ Generate Setters...
16     ⌂ Generate Constructors...
17     ⌂ Generate hashCode() and equals()...
18     ⌂ Generate toString()...
19     ⌂ Override/Implement Methods...
20     ⌂ Generate Delegate Methods...
```



The screenshot shows a Java code editor with the file `Empleados.java` open. The code defines a class `Empleados` with private fields `nombre`, `apellido`, `direccion`, `puesto`, `edad`, `telefono`, and `id`. It includes getters and setters for each field. A yellow lightbulb icon is visible near the start of the `setNombre` method, indicating a potential issue or suggestion.

```
src > main > java > com > ingrid > demo > springboot_clean > models > Empleados.java > Empleados.java
1 package com.ingrid.demo.springboot_clean.models;
2
3 public class Empleados {
4
5     private String nombre, apellido, direccion, puesto;
6     private int edad, telefono, id;
7
8     public String getNombre() {
9         return nombre;
10    }
11    public void setNombre(String nombre) {
12        this.nombre = nombre;
13    }
14    public String getApellido() {
15        return apellido;
16    }
17    public void setApellido(String apellido) {
18        this.apellido = apellido;
19    }
20    public String getDireccion() {
21        return direccion;
22    }
23    public void setDireccion(String direccion) {
24        this.direccion = direccion;
25    }
26    public String getPuesto() {
27        return puesto;
28    }
29    public void setPuesto(String puesto) {
30        this.puesto = puesto;
31    }
32    public int getEdad() {
33        return edad;
34    }
35    public void setEdad(int edad) {
36        this.edad = edad;
37    }
38}
```

The screenshot shows a Java code editor with three tabs at the top: EjemploController.java, EjemploRestController.java, and Empleados.java. The Empleados.java tab is active. The code is a simple POJO with a constructor and getters/setters for nombre, apellido, direccion, and puesto. A yellow lightbulb icon is visible on line 17, indicating a potential issue or suggestion.

```
c > main > java > com > ingrid > demo > springboot_clean > models > Empleados.java > Empleados > Empleados()
1 package com.ingrid.demo.springboot_clean.models;
2
3 public class Empleados {
4
5     private String nombre, apellido, direccion, puesto;
6     private int edad, telefono, id;
7
8     //Constructor
9     public Empleados(String nombre, String apellido, String direccion, String puesto,
10                    int edad, int telefono, int id) {
11         this.nombre = nombre;
12         this.apellido = apellido;
13         this.direccion = direccion;
14         this.puesto = puesto;
15         this.edad = edad;
16         this.telefono = telefono;
17         this.id = id;
18     }
19
20     public String getNombre() {
21         return nombre;
22     }
23     public void setNombre(String nombre) {
24         this.nombre = nombre;
25     }
26     public String getApellido() {
27         return apellido;
28     }
29     public void setApellido(String apellido) {
30         this.apellido = apellido;
31     }
32     public String getDireccion() {
33         return direccion;
34     }
35     public void setDireccion(String direccion) {
36         this.direccion = direccion;
37     }
38 }
```

Agregamos el constructor e inicializamos

localhost:8080/api/detalle_info2

```
Aplicaciones | YouTube | BCentral | UDD | Zignal
```

```
▼ {  
  ▼ "Empleado": {  
    "nombre": "Juan",  
    "apellido": "Perez",  
    "direccion": "Calle Falsa 123",  
    "puesto": "Desarrollador",  
    "edad": 30,  
    "telefono": 123456789,  
    "id": 1  
  }  
}
```

Bonus

The screenshot shows a Java IDE interface with multiple tabs open. The active tab is 'EjemploController.java'. The code in the editor is as follows:

```
src > main > java > com > ingrid > demo > springboot_clean > controllers > EjemploController.java > Language
```

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 //import org.springframework.ui.Model;
6 import org.springframework.web.bind.annotation.GetMapping;
7
8 import com.ingrid.demo.springboot_clean.models.Empleados;
9
10 //import java.util.Map;
11
12
13 @Controller
14 public class EjemploController {
15
16     @GetMapping("/detalle_info")
17     public String info(Model modelo) {
18         Empleados empleado1 = new Empleados(nombre:"Juan", apellido:"Perez", direccion:"Calle Falsa 123",
19                                             edad:30, telefono:123456789, id:1);
20         modelo.addAttribute(attributeName:"Empleado", empleado1);
21
22         //Patron de inyeccion de dependencias
23         return "detalle_info";
24     }
25
26 }
27
```

The screenshot shows a code editor interface with four tabs at the top: 'EjemploController.java', 'detalle_info.html', 'EjemploRestController.java', and 'Empleados.java'. The 'detalle_info.html' tab is active, displaying a Thymeleaf template file. The file structure is as follows:

```
src > main > resources > templates > detalle_info.html > html > body > ul > li > span
```

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns:th="http://www.thymeleaf.org">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title th:text="${Titulo}"> Pagina Spring Boot</title>
7  </head>
8  <body>
9      <h1>Información de empleado</h1>
10     <ul>
11         <li><strong>Nombre</strong><span th:text="${Empleado.nombre}"></span></li>
12         <li><strong>Apellido</strong><span th:text="${Empleado.apellido}"></span></li>
13         <li><strong>Direccion</strong><span th:text="${Empleado.direccion}"></span></li>
14         <li><strong>Puesto</strong><span th:text="${Empleado.puesto}"></span></li>
15         <li><strong>Edad</strong><span th:text="${Empleado.edad}"></span></li>
16         <li><strong>Telefono</strong><span th:text="${Empleado.telefono}"></span></li>
17         <li><strong>ID</strong><span th:text="${Empleado.id}"></span></li>
18     </ul>
19
20 </body>
21 </html>
```

← → ⌂



localhost:8080/detalle_info



Aplicaciones



| YouTube



BCentral



UDD



Zignal

Información de empleado

- **Nombre**Juan
- **Apellido**Perez
- **Direccion**Calle Falsa 123
- **Puesto**Desarrollador
- **Edad**30
- **Telefono**123456789
- **ID**1

`@ModelAttribute` -> se utiliza para **vincular datos de un formulario HTML**

EjemploController.java

detalle_info.html X

EjemploRestController.java

Empleados.java

```
src > main > resources > templates > detalle_info.html > html > body > table
  2   <html lang="en" xmlns:th="http://www.thymeleaf.org">
  8     <body>
10       <ul>
18     </ul>
19
20   <h1>Lista de Empleados</h1>
21   <table border="1">
22     <thead>
23       <tr>
24         <th>Nombre</th>
25         <th>Apellido</th>
26         <th>Dirección</th>
27         <th>Puesto</th>
28         <th>Edad</th>
29         <th>Teléfono</th>
30         <th>Departamento</th>
31       </tr>
32     </thead>
33     <tbody>
34       <tr th:each="Empleado : ${Empleados}">
35         <td th:text="${Empleado.nombre}"></td>
36         <td th:text="${Empleado.apellido}"></td>
37         <td th:text="${Empleado.direccion}"></td>
38         <td th:text="${Empleado.puesto}"></td>
39         <td th:text="${Empleado.edad}"></td>
40         <td th:text="${Empleado.telefono}"></td>
41         <td th:text="${Empleado.id}"></td>
42       </tr>
43     </tbody>
44   </table>
45
46   </body>
47 </html>
```

← → ⌂



localhost:8080/detalle_info

Aplicaciones



YouTube



BCentral



UDD



ZignaUDD



OneDrive



Chamba

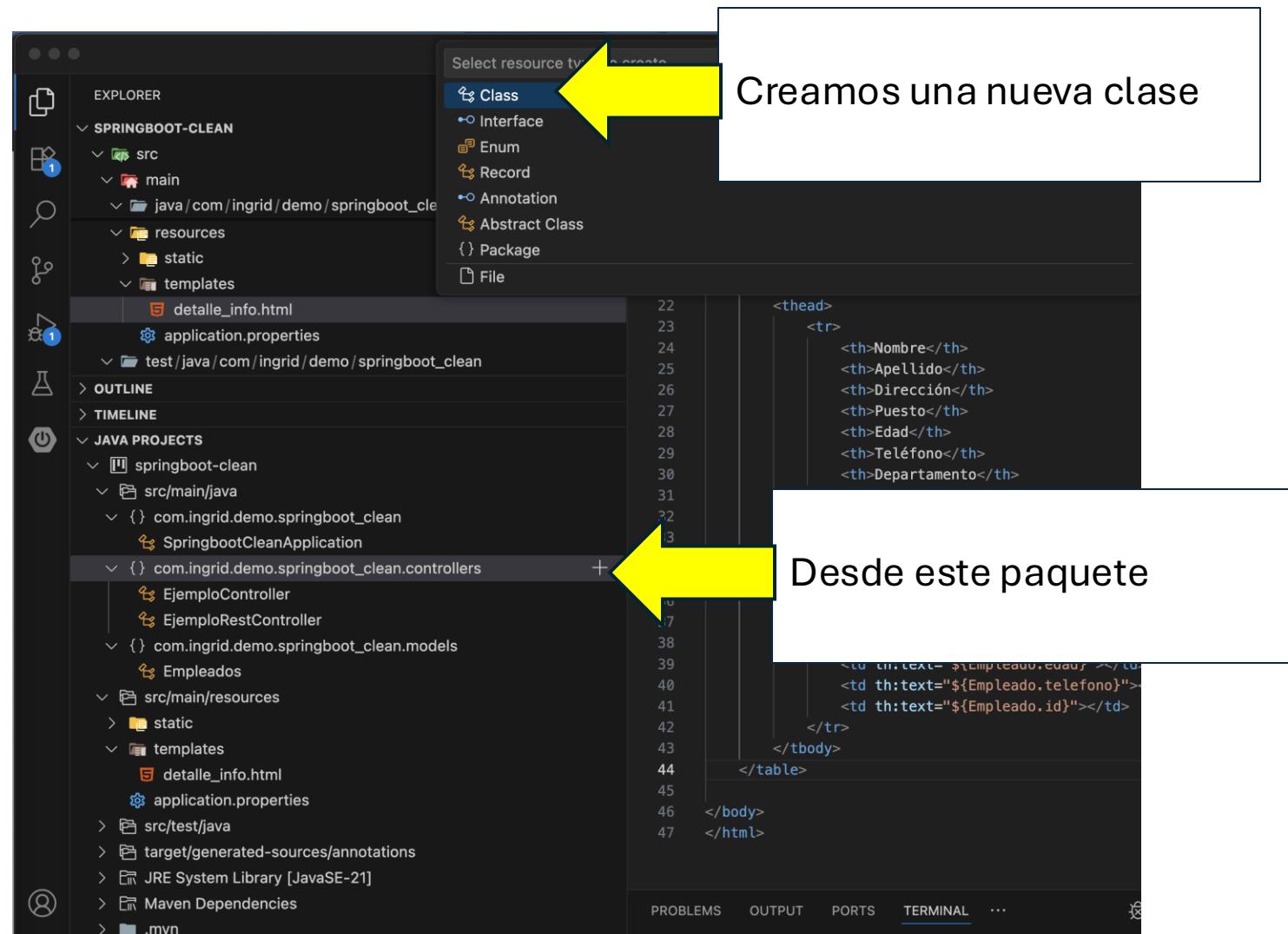
Información de empleado

- **Nombre**Juan
- **Apellido**Perez
- **Direccion**Calle Falsa 123
- **Puesto**Desarrollador
- **Edad**30
- **Telefono**123456789
- **ID**1

Lista de Empleados

Nombre	Apellido	Dirección	Puesto	Edad	Teléfono	Departamento
Ana	Gomez	Avenida Siempre Viva 456	Gerente	35	987654321	2
Luis	Martinez	Calle del Sol 789	Analista	28	456123789	3

@RequestParam -> se usa para obtener parámetros directamente de la URL (query string) en una solicitud HTTP, típicamente en métodos GET



The screenshot shows a Java Spring Boot project named "SPRINGBOOT-CLEAN" in an IDE. The project structure is as follows:

- src/main/java/com/ingrid/demo/springboot_clean/controllers**:
 - EjemploController.java
 - EjemploRestController.java
 - RequestParamControllers.java (selected)
- src/main/java/com/ingrid/demo/springboot_clean/models**:
 - Empleados.java
- src/main/resources/templates**:
 - detalle_info.html

The **RequestParamControllers.java** file is open in the editor, showing the following code:

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 public class RequestParamControllers {
4
5 }
```

The **Outline** pane shows the class structure of **RequestParamControllers**:

- com.ingrid.demo.springboot_clean.controllers.RequestParamControllers
- + EjemploController
- + EjemploRestController
- + RequestParamControllers
- + Empleados

EjemploController.java

detalle_info.html

RequestParamControllers.java

parametroDTO.java

```
src > main > java > com > ingrid > demo > springboot_clean > controllers > parametroDTO.java > pa :: || ⌂ ⌄ ⌅
1  package com.ingrid.demo.springboot_clean.controllers;
2
3  public class parametroDTO {
4      private String param;
5
6      public String getParam() {
7          return param;
8      }
9
10     public void setParam(String param) {
11         this.param = param;
12     }
13
14
15
16 }
17
```

EjemploController.java detalle_info.html RequestParamControllers.java parametroDTO.java

```
src > main > java > com > ingrid > demo > springboot_clean > controllers > RequestParamControllers.java ::
```

```
1 package com.ingrid.demo.springboot_clean.controllers;
2
3 import org.springframework.web.bind.annotation.RequestMapping;
4 import org.springframework.web.bind.annotation.RestController;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RequestParam;
7
8
9 @RestController
10 @RequestMapping("/api/parametros")
11 public class RequestParamControllers {
12
13     @GetMapping("/detalle")
14     public parametroDTO detalle(@RequestParam String param) {
15         parametroDTO parametro1 = new parametroDTO();
16         parametro1.setParam(param);
17         return parametro1; // Replace with actual implementation
18     }
19
20 }
21
```

A screenshot of a web browser window. The address bar shows the URL `localhost:8080/api/parametros/detalle?param=Saludos`. Below the address bar is a navigation bar with icons for back, forward, and search. The main content area displays a JSON object:

```
▼ {  
  "param": "Saludos"  
}
```

Módulo 2.- Desarrollo Back-End con Spring Boot

Controladores REST y manejo de rutas.

◆ Anotaciones clave

Anotación	Significado
@RestController	Declara una clase como controlador REST
@RequestMapping	Define una ruta base común para todos los métodos de la clase
@GetMapping	Maneja solicitudes GET
@PostMapping	Maneja solicitudes POST
@PutMapping	Maneja actualizaciones (PUT)
@DeleteMapping	Maneja eliminaciones (DELETE)



Módulo 2.- Desarrollo Back-End con Spring Boot

O Controladores REST y manejo de rutas.

Buenas prácticas para controladores REST

- Mantén los controladores **delgados**: solo deben coordinar, no contener lógica.
- Usa **constructor injection** para los servicios.
- Usa nombres de ruta **semánticos y coherentes** (/productos, /usuarios, etc.).
- Agrupa los endpoints REST por dominio o contexto.
- **No mezcles controladores REST con vistas (Model, Thymeleaf).**



Módulo 2.- Desarrollo Back-End con Spring Boot

O Controladores REST y manejo de rutas.

Visualizar la separación de capas (Controller → Service → Repository)

- Cliente → Controlador → Servicio → Repositorio → Base de Datos

```
@RestController
public class ProductoController {
    private final ProductoService service;

    public ProductoController(ProductoService service) {
        this.service = service;
    }

    @GetMapping("/productos")
    public List<Producto> listar() {
        return service.obtenerTodos();
    }
}
```



Módulo 2.- Desarrollo Back-End con Spring Boot

O Inyección de dependencias.

¿Qué es la inyección de dependencias?

- Es un patrón donde un objeto recibe sus dependencias desde **afuera**, en lugar de crearlas por sí mismo.
- Spring lo implementa a través de su contenedor de IoC (ApplicationContext), que crea, configura y conecta objetos (beans).

```
public class Servicio {  
    private Repositorio repo;  
  
    public Servicio(Repositorio repo) {  
        this.repo = repo;  
    }  
}
```



Módulo 2.- Desarrollo Back-End con Spring Boot

O Inyección de dependencias.

Diferencias entre **@Component**, **@Service**, **@Repository**

Anotación	Uso	Observaciones
@Component	Clase genérica gestionada por Spring	Base para las demás
@Service	Lógica de negocio	Semántica para servicios
@Repository	Acceso a datos	Integra manejo de excepciones con Spring Data

Todas son detectadas por Spring en el escaneo de componentes, pero su uso semántico ayuda a separar responsabilidades.”



Módulo 2.- Desarrollo Back-End con Spring Boot

O Inyección de dependencias.

Comparación visual o explicativa entre constructor injection y field injection

```
// ❌ Field injection (NO recomendado)
@Autowired
private ProductoRepository repo;

// ✅ Constructor injection (RECOMENDADO)
@Service
public class ProductoService {
    private final ProductoRepository repo;

    public ProductoService(ProductoRepository repo) {
        this.repo = repo;
    }
}
```



- Beneficios de constructor injection:
- Facilita testeo
 - Hace obligatorias las dependencias
 - Inmutable y limpio

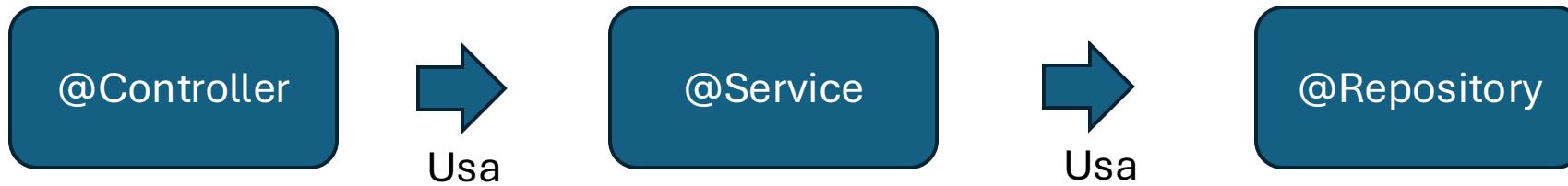


Módulo 2.- Desarrollo Back-End con Spring Boot



Inyección de dependencias.

Cómo Spring “arma” las dependencias



Con flechas que muestran cómo funciona el escaneo y registro de beans (@ComponentScan, @SpringBootApplication, etc.).



Módulo 2.- Desarrollo Back-End con Spring Boot

O Inyección de dependencias.

¿Qué es el contenedor IoC?

Imagina una **fábrica automática**:

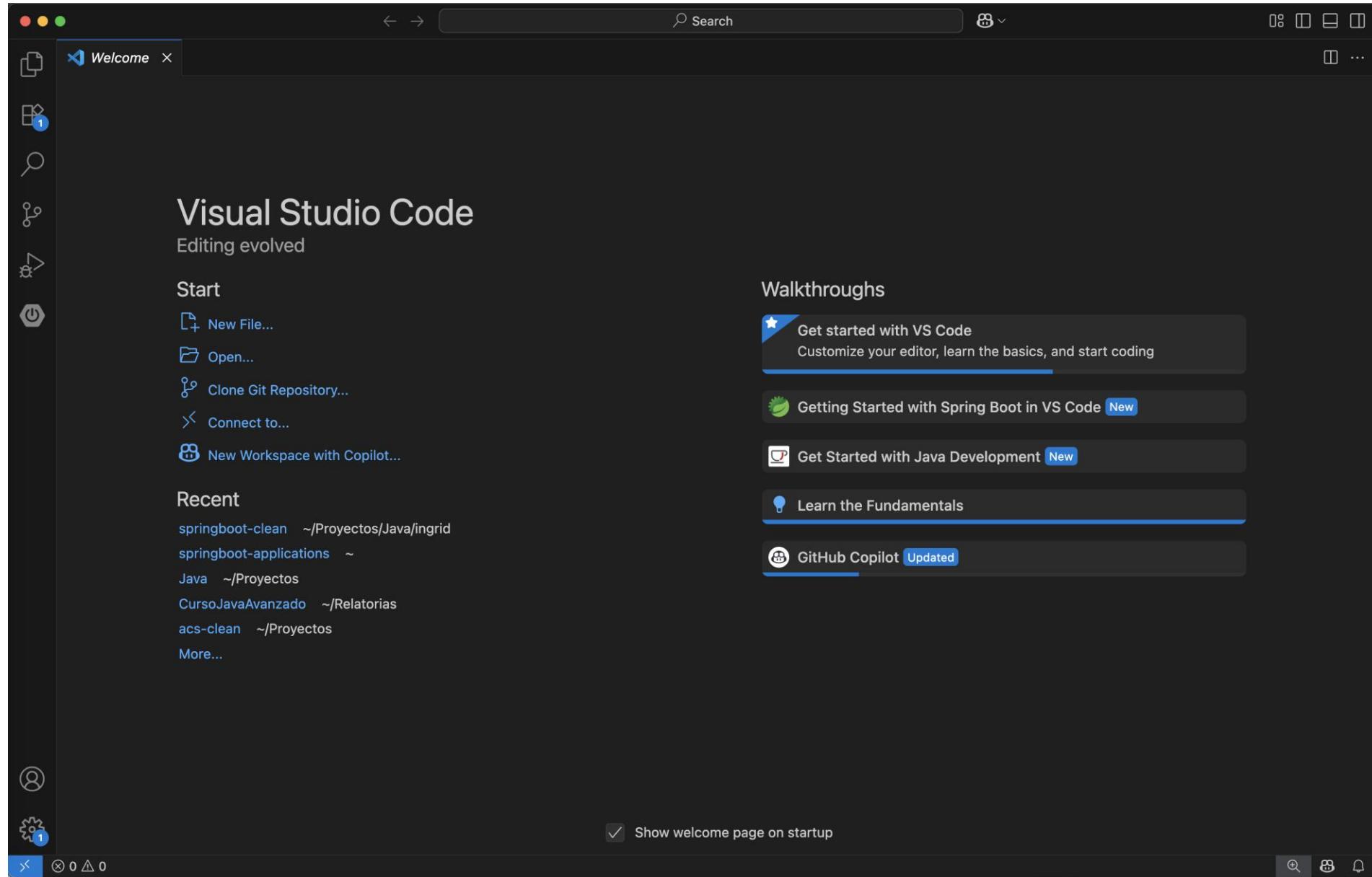
- Tú defines los planos de los objetos (@Component, @Service, @Repository)
- Spring **construye, configura e inyecta** esos objetos por ti
- Tú solo dices qué necesitas y Spring lo entrega

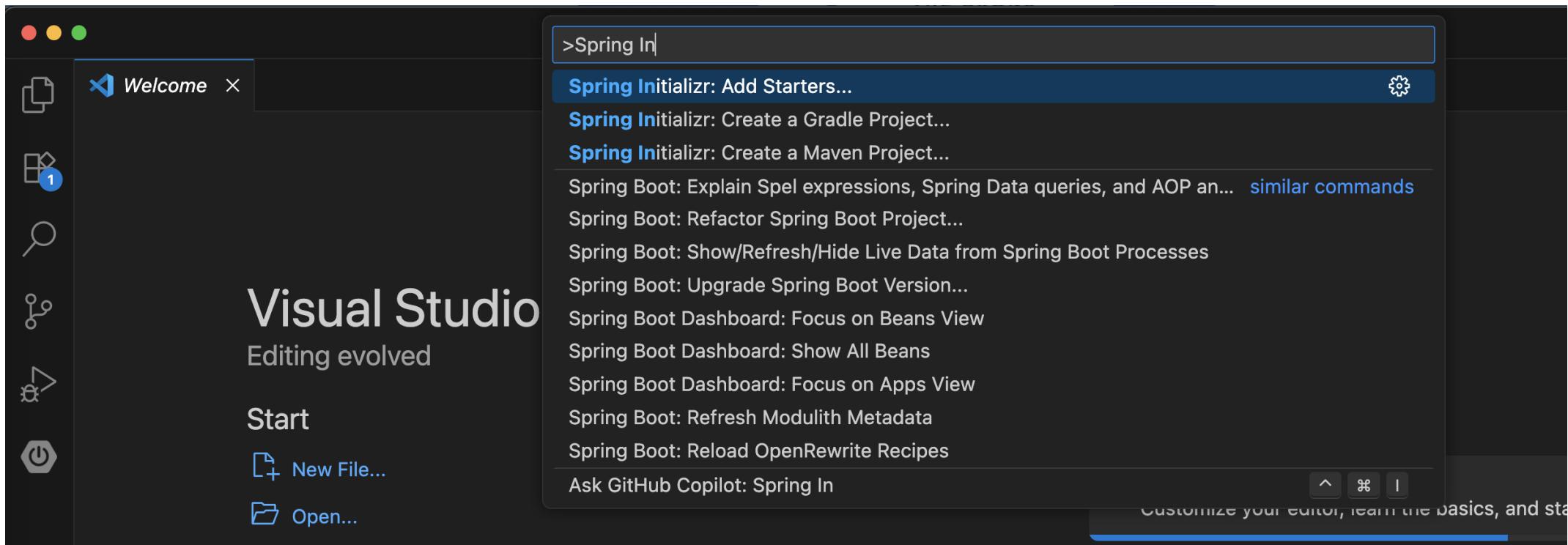
Nombre real de esa fábrica en Spring: ApplicationContext

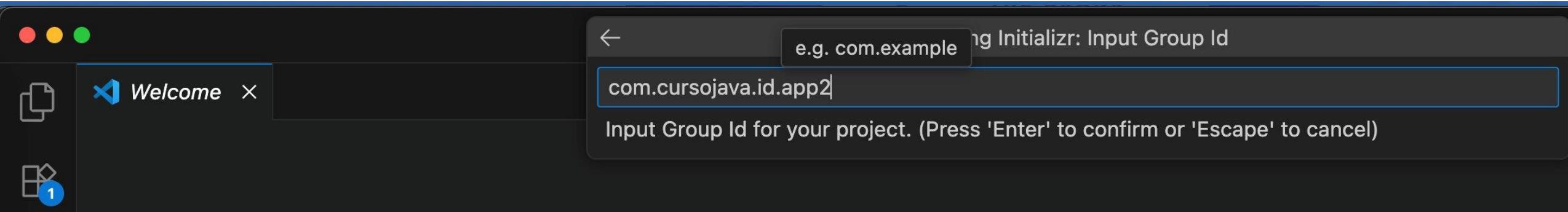
Es el núcleo que gestiona el **ciclo de vida de los beans** (objetos gestionados por Spring).

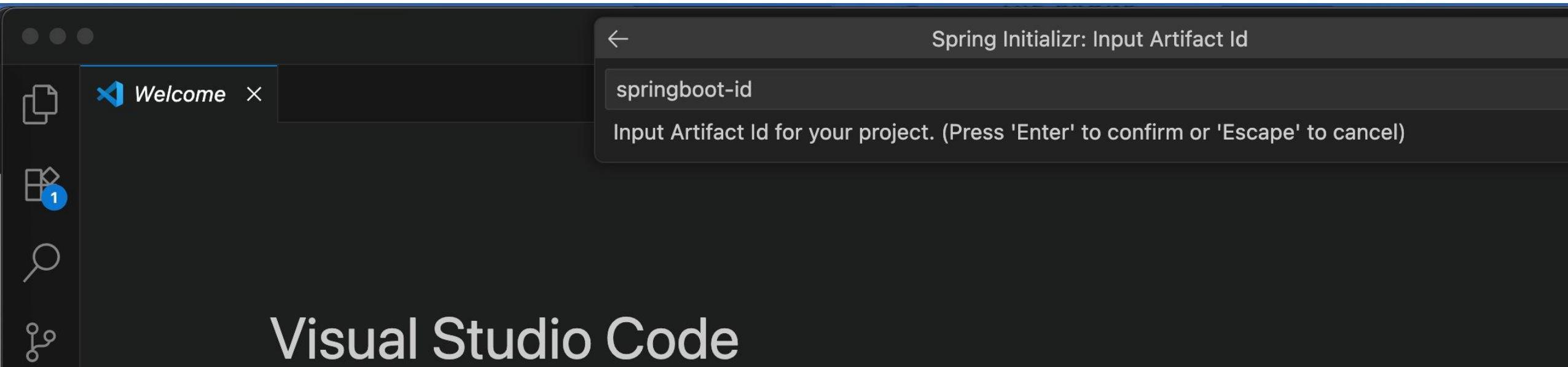


Ejemplo: Inyección de dependencias









The screenshot shows the Visual Studio Code interface with the Spring Initializr extension open. The sidebar on the left includes icons for file operations, a welcome screen, a folder containing one item, a search function, and developer tools. The main area displays the Spring Initializr interface with the title "Spring Initializr: Choose dependencies". A search bar at the top says "Search for dependencies." Below it, a message states "Selected 2 dependencies" and "Press <Enter> to continue." Two items are listed: "Spring Web" (selected) and "Spring Boot DevTools". The "Spring Web" entry is described as building web applications using Spring MVC and Apache Tomcat. The "Spring Boot DevTools" entry is described as providing fast application restarts, LiveReload, and configurations for enhanced development. Other visible items include "GraalVM Native Support", "GraphQL DGS Code Generation", "Lombok", "Spring Configuration Processor", and "Docker Compose Support".

Visual Studio
Editing evolved

Start

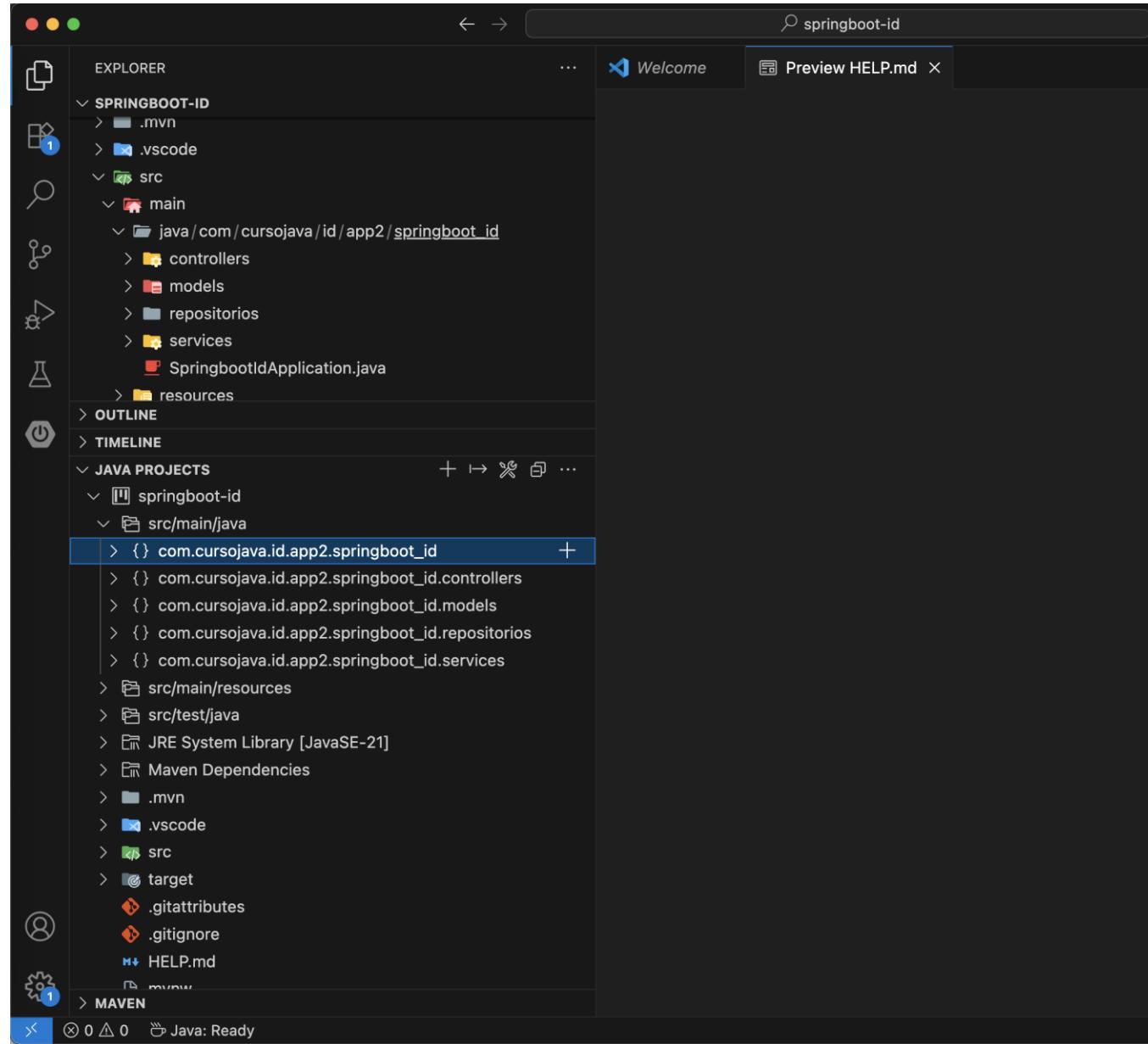
- New File...
- Open...
- Clone Git Repository...

Spring Initializr: Choose dependencies

Search for dependencies.

Selected 2 dependencies
Press <Enter> to continue.

- ✓ Spring Web Web Selected
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomca...
- ✓ Spring Boot DevTools Developer Tools
Provides fast application restarts, LiveReload, and configurations for enhanced developmen...
- GraalVM Native Support Developer Tools Developer Tools
Support for compiling Spring applications to native executables using the ...
- GraphQL DGS Code Generation Developer Tools
Generate data types and type-safe APIs for querying GraphQL APIs by parsing schema files.
- Lombok Developer Tools
Java annotation library which helps to reduce boilerplate code.
- Spring Configuration Processor Developer Tools
Generate metadata for developers to offer contextual help and "code completion" when wor...
- Docker Compose Support Developer Tools



Los servicios interactúan con los repositorios, y los repositorios son los que acceden a los datos. Van a las bases de datos, hacen consultas, select, actualizaciones, etc.

Y los servicios interactúan con estos objetos para la lógica de negocio con eso vamos a poder nosotros manipular la información.

Clase Model

Es la clase que representa los modelos de datos de nuestra aplicación. Esta clase se diseña para reflejar la estructura de las entidades o datos que se manejan en nuestra aplicación.

Cada clase que se cree dentro del paquete model va a representar una entidad. Es decir, un objeto que puede ser usuario, producto, etc.

The screenshot shows the VS Code interface with a dark theme. The Explorer sidebar on the left displays a project structure for a Spring Boot application named 'SPRINGBOOT-ID'. The 'src/main/java/com/cursojava/id/app/springboot_id' folder contains 'controllers', 'models', and 'repository' packages. A context menu is open over the 'repository' package, with 'New Java File...' selected. A sub-menu for 'Class...' is open, showing options like 'New Java File...', 'New Java Package...', 'New Java Project...', 'Maven', and 'Find in Folder...'. The main editor area on the right shows a Java code snippet for a 'Productos' class:

```
src > main > java > com > cursojava > id > app > springboot_id
1 package com.cursojava.id.app.springboot_id;
2
3 public class Productos {
4     private Long idProducto;
5     private String nombre;
6     private Double precio;
7
8     public Productos() {
9         this.idProducto;
10        this.nombre;
11        this.precio;
12    }
13
14    public Long getIdProducto() {
15        return idProducto;
16    }
17
18    public void setIdProducto(Long idProducto) {
19        this.idProducto = idProducto;
20    }
21
22    public String getNombre() {
23        return nombre;
24    }
25
26    public void setNombre(String nombre) {
27        this.nombre = nombre;
28    }
29
30    public Double getPrecio() {
31        return precio;
32    }
33
34    public void setPrecio(Double precio) {
35        this.precio = precio;
36    }
}
```

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists the project structure for a Spring Boot application named "SPRINGBOOT-ID". The "src/main/java/com.cursojava/id/app/springboot_id/models" folder contains a file named "Productos.java". The code editor on the right displays the following Java code:

```
1 package com.cursojava.id.app.springboot_id.models;
2
3 public class Productos {
4
5 }
6
```

Se crea la clase Productos

Productos.java X

Repo_Productos.java

src > main > java > com > cursojava > id > app > springboot_id > models > Productos.java > Productos > setPrecio(int)

```
1 package com.cursojava.id.app.springboot_id.models;
2
3 public class Productos {
4     private Long idProducto;
5     private String nombre;
6     private int precio;
7
8     public Productos(Long idProducto, String nombre, int precio) {
9         this.idProducto = idProducto;
10        this.nombre = nombre;
11        this.precio = precio;
12    }
13
14    public Long getIdProducto() {
15        return idProducto;
16    }
17    public void setIdProducto(Long idProducto) {
18        this.idProducto = idProducto;
19    }
20    public String getNombre() {
21        return nombre;
22    }
23    public void setNombre(String nombre) {
24        this.nombre = nombre;
25    }
26    public int getPrecio() {
27        return precio;
28    }
29    public void setPrecio(int precio) {
30        this.precio = precio;
31    }
32
33
34 }
```

Detalle clase producto

```
src > main > java > com > cursojava > id > app > springboot_id > repositorios > Repo_Productos.java > Repo_P
 1 package com.cursojava.id.app.springboot_id.repositorios;
 2 import java.util.ArrayList;
 3 import java.util.Arrays;
 4 import java.util.List;
 5 import com.cursojava.id.app.springboot_id.models.Productos;
 6
 7 public class Repo_Productos {
 8
 9     List<Productos> datos;
10
11
12     public Repo_Productos() {
13         this.datos = new ArrayList<>(Arrays.asList(
14             new Productos(idProducto:1L, nombre:"Disco duro 110", precio:500),
15             new Productos(idProducto:2L, nombre:"USB 20GB", precio:400),
16             new Productos(idProducto:3L, nombre:"Mouse", precio:50),
17             new Productos(idProducto:4L, nombre:"Teclado", precio:60),
18             new Productos(idProducto:5L, nombre:"Monitor 20", precio:2000),
19             new Productos(idProducto:6L, nombre:"Monitor 24", precio:2500),
20             new Productos(idProducto:7L, nombre:"Monitor 27", precio:3000),
21             new Productos(idProducto:8L, nombre:"Laptop 15", precio:5000),
22             new Productos(idProducto:9L, nombre:"Laptop 17", precio:6000),
23             new Productos(idProducto:10L, nombre:"Laptop 19", precio:7000)
24         ));
25     }
26
27
28     public List<Productos> findAll() {
29         return datos;
30     }
31
32     public Productos buscaById(Long id) {
33         return datos.stream()
34             .filter(producto -> producto.getIdProducto().equals(id))
35             .findFirst()
36             .orElse(other:null);
37     }
38
39 }
40
```

Detalle clase Repo_productos

The screenshot shows the Visual Studio Code (VS Code) interface. The Explorer sidebar on the left displays the project structure under the 'SPRINGBOOT-ID' folder. The 'src/main/java/com/cursojava/id/app/springboot_id/repo...' folder contains 'Repo_Productos.java'. Below it is a 'services' folder which contains 'Productos_Services.java'. The 'OUTLINE' and 'TIMELINE' sections are collapsed. The 'JAVA PROJECTS' section shows the 'springboot-id' project with its subfolders: 'src/main/java', 'src/main/resources', 'src/test/java', 'JRE System Library [JavaSE-21]', 'Maven Dependencies', and '.mvn'. The 'Editor' tab at the top has tabs for 'Productos.java', 'Repo_Productos.java', and 'Productos_Services.java'. The 'Productos_Services.java' tab is active, showing the following code:

```
1 package com.cursojava.id.app.springboot_id.services;
2
3 public class Productos_Services {
4
5 }
6
```

Creamos la clase Productos_Services en el paquete services.

The screenshot shows a Java code editor with the following tabs:

- Productos.java U
- Repo_Productos.java U
- Productos_Services.java U X
- ProductosController.java U

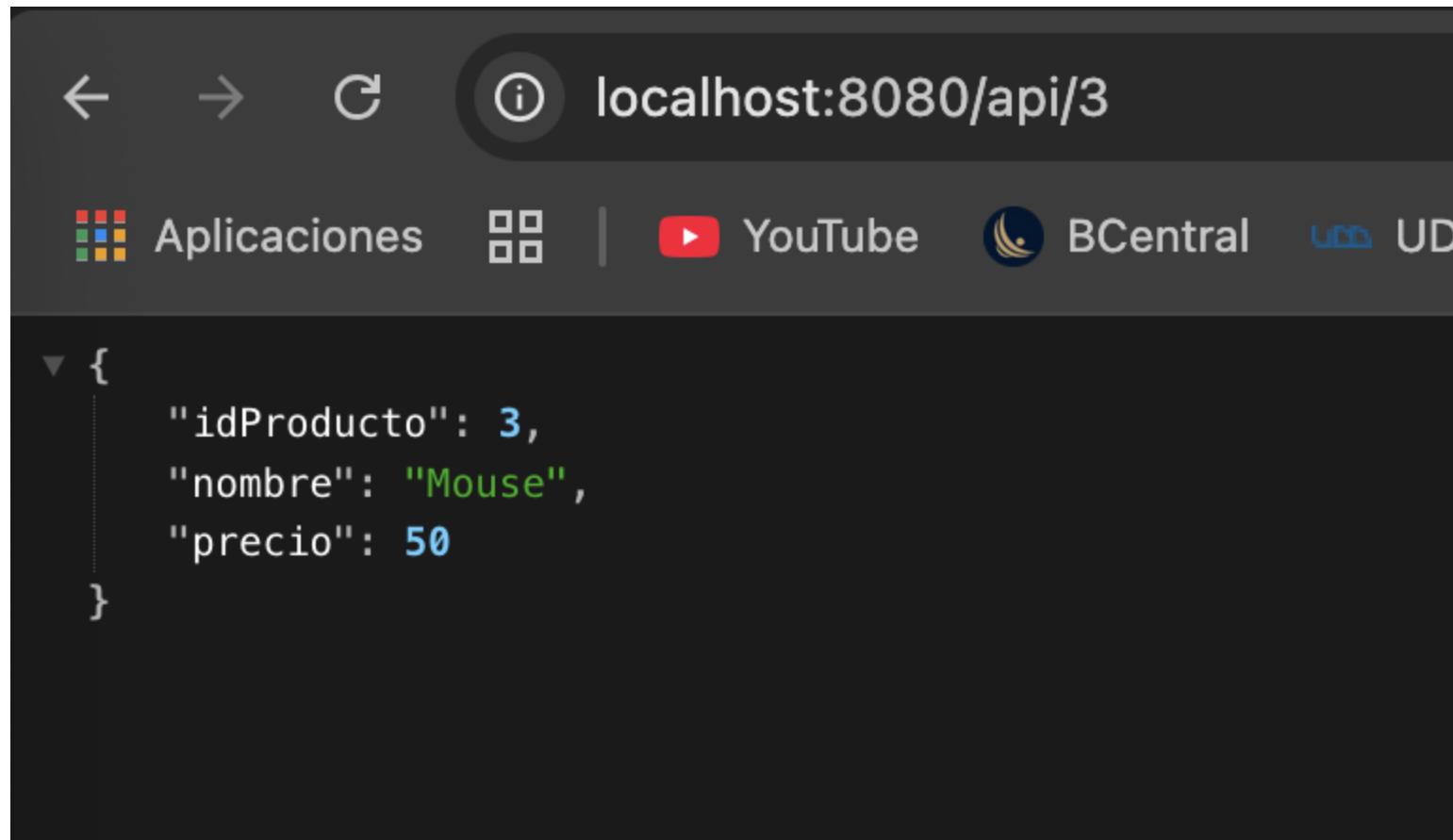
The current file is `Productos_Services.java`. The code implements a service layer for products using functional programming techniques with Java Streams. It includes methods to find all products with a price increase and to find a product by its ID.

```
src > main > java > com > cursojava > id > app2 > springboot_id > services > Productos_Services.java > Productos_Services > findAll()
1 package com.cursojava.id.app2.springboot_id.services;
2
3 import java.util.List;
4 import java.util.stream.Collectors;
5
6 import com.cursojava.id.app2.springboot_id.repositorios.Repo_Productos;
7 import com.cursojava.id.app2.springboot_id.models.Productos;
8
9 public class Productos_Services {
10
11     // Instancia del repositorio para acceder a los datos de los productos
12     private Repo_Productos repositorio = new Repo_Productos();
13
14     // Método para obtener todos los productos, aplicando un aumento de precio
15     public List<Productos> findAll() {
16         // Obtiene todos los productos del repositorio
17         return repositorio.findAll()
18             .stream() // Convierte la lista a un Stream para operaciones funcionales
19             .map(p -> { // Mapea cada producto a una nueva versión
20                 double precioTotal = p.getPrecio(); |
21                 p.setPrecio((int) precioTotal); // Actualiza el precio del producto (se hace un cast a int)
22                 return p; // Devuelve el producto modificado
23             })
24             .collect(Collectors.toList()); // Recolecta los productos modificados de nuevo en una lista
25     }
26
27     // Método para buscar un producto por su ID
28     public Productos buscaId(Long idProducto) {
29         // Delega la búsqueda al repositorio
30         return repositorio.buscaById(idProducto);
31     }
32 }
33
```

```
Productos.java U Repo_Productos.java U Productos_Services.java U ProductosController.java U ●
src > main > java > com > cursojava > id > app2 > springboot_id > controllers > ProductosController.java > Language Support for Java(TM) by Red Hat > Products :::
 3   import java.util.List;
 4
 5   import org.springframework.web.bind.annotation.GetMapping;
 6   import org.springframework.web.bind.annotation.PathVariable;
 7   import org.springframework.web.bind.annotation.RequestMapping;
 8   import org.springframework.web.bind.annotation.RestController;
 9
10  import com.cursojava.id.app2.springboot_id.services.Productos_Services;
11  import com.cursojava.id.app2.springboot_id.models.Productos;
12
13 @RestController // Indica que esta clase es un controlador REST y los métodos devolverán datos directamente (ej. JSON)
14 @RequestMapping("/api") // Mapea todas las URLs de esta clase para que comiencen con /api
15 public class ProductosController {
16
17     // Inyecta el servicio de productos. Spring Boot creará e inyectará una instancia.
18     // Esto se llama Inyección de Dependencias.
19     private final Productos_Services productosServices;
20
21     // Constructor para la inyección de dependencias del servicio.
22     // Spring Boot lo detectará automáticamente.
23     public ProductosController() {
24         this.productosServices = new Productos_Services();
25     }
26
27
28     // Endpoint para obtener todos los productos (ej. GET /api/productos)
29     @GetMapping("/productos")
30     public List<Productos> obtenerTodosLosProductos() {
31         return productosServices.findAll();
32     }
33
34     // Endpoint para obtener un producto por su ID (ej. GET /api/productos/1)
35     @GetMapping("/productos/{id}")
36     public Productos obtenerProductoPorId(@PathVariable Long id) {
37         Productos producto = productosServices.buscaId(id);
38         // Podrías añadir lógica aquí para manejar si el producto es null (no encontrado),
39         // por ejemplo, lanzar una excepción HttpNotFound o devolver un ResponseEntity.
40         return producto;
41     }
42 }
43
```



```
[{"idProducto": 1, "nombre": "Disco duro 110", "precio": 750}, {"idProducto": 2, "nombre": "USB 20GB", "precio": 600}, {"idProducto": 3, "nombre": "Mouse", "precio": 75}, {"idProducto": 4, "nombre": "Teclado", "precio": 90}, {"idProducto": 5, "nombre": "Monitor 20", "precio": 3000}, {"idProducto": 6, "nombre": "Monitor 24", "precio": 3750}, {"idProducto": 7, "nombre": "Monitor 27", "precio": 4500}, {"idProducto": 8, "nombre": "Laptop 15", "precio": 7500}, {"idProducto": 9, "nombre": "Laptop 17", "precio": 9000}, {"idProducto": 10, "nombre": "Laptop 19", "precio": 10500}]
```



A screenshot of a mobile browser displaying a JSON object. The URL in the address bar is `localhost:8080/api/3`. The JSON content is:

```
{  
    "idProducto": 3,  
    "nombre": "Mouse",  
    "precio": 50  
}
```

Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

Diseñar microservicios **orientados al dominio** significa estructurar tu sistema en base a **las reglas y conceptos del negocio** (no en base a la tecnología).

En lugar de tener servicios genéricos como:

/api/data
/api/utils

Tienes microservicios centrados en
contextos del negocio:

/productos
/ordenes
/usuarios

Cada uno:

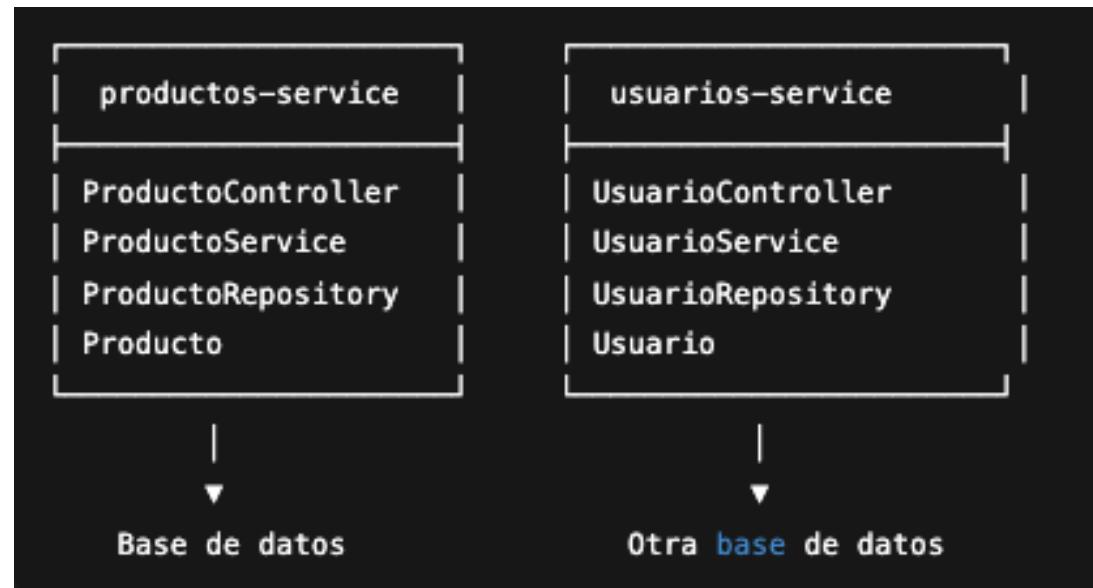
- Modela su dominio con sus propias entidades
- Contiene su lógica de negocio
- Gestiona su propia base de datos (idealmente)



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

Diagrama conceptual



Cada microservicio es **independiente, cohesivo** y modelado según un **subdominio del negocio**.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

Ejemplo simple de microservicio de productos orientado al dominio.

Crear un **microservicio RESTful** que permita:

- Listar productos
- Agregar un nuevo producto

El microservicio sigue un **diseño en capas** y está **centrado en el dominio** “Producto”.

Link: <https://github.com/Joselota/Relatorias/tree/main/productos-service>



Módulo 2.- Desarrollo Back-End con Spring Boot

Creación de microservicios orientados al dominio del MSGG.

Ejemplo simple de microservicio de productos orientado al dominio.

1. Modelo del dominio: Producto.java

```
public class Producto {  
    private Long id;  
    private String nombre;  
    private Double precio;  
    private Integer stock;  
  
    // Getters y setters  
}
```

Esta clase hace:

- Representa el **modelo del dominio**: un producto en tu sistema.
- No contiene lógica, solo datos.
- Es usada por todas las demás capas (controlador, servicio, repositorio).
- Puede anotarse con `@Entity` si conectas a una base de datos relacional.



Módulo 2.- Desarrollo Back-End con Spring Boot

Creación de microservicios orientados al dominio del MSGG.

Ejemplo simple de microservicio de productos orientado al dominio.

2. Lógica de negocio: ProductoService.java

```
@Service
public class ProductoService {
    private final ProductoRepository repo;

    public ProductoService(ProductoRepository repo) {
        this.repo = repo;
    }

    public List<Producto> listarProductos() {
        return repo.findAll();
    }

    public Producto guardar(Producto p) {
        // aquí podrías validar que el precio > 0, etc.
        return repo.save(p);
    }
}
```

Esta clase hace:

- Es la **capa de servicio**, anotada con `@Service` para que Spring la detecte.
- Contiene la **lógica de negocio**: reglas, validaciones, decisiones del dominio.
- No expone rutas ni se comunica con HTTP.
- Usa el repositorio para acceder a los datos.



Módulo 2.- Desarrollo Back-End con Spring Boot

Creación de microservicios orientados al dominio del MSGG.

Ejemplo simple de microservicio de productos orientado al dominio.

3. Exposición REST: ProductoController.java

```
@RestController
@RequestMapping("/productos")
public class ProductoController {
    private final ProductoService service;

    public ProductoController(ProductoService service) {
        this.service = service;
    }

    @GetMapping
    public List<Producto> listar() {
        return service.listarProductos();
    }

    @PostMapping
    public Producto crear(@RequestBody Producto p) {
        return service.guardar(p);
    }
}
```

Esta clase hace:

- Es el **punto de entrada del microservicio** para clientes (Postman, frontend, otro servicio).
- Está anotada con `@RestController`, que combina:
 - `@Controller`
 - `@ResponseBody` (para retornar JSON directamente)
- Define dos endpoints:
 - GET `/productos` → lista todos los productos
 - POST `/productos` → crea un nuevo producto desde un JSON enviado en el cuerpo de la solicitud



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

Ejemplo simple de microservicio de productos orientado al dominio.

4. Acceso a datos: ProductoRepository.java

```
@Repository  
public interface ProductoRepository extends JpaRepository<Producto, Long> {}
```

Esta clase hace:

- Define la **capa de persistencia**, usando Spring Data JPA.
- JpaRepository le da métodos listos como findAll(), save(), deleteById(), etc.
- Spring la detecta por @Repository y la inyecta donde se necesite.
- **No escribes SQL**, lo maneja Spring automáticamente.



Módulo 2.- Desarrollo Back-End con Spring Boot

O Creación de microservicios orientados al dominio del MSGG.

¿Cómo probarlo?

- Levanta el microservicio con Spring Boot (VS Code, IntelliJ, terminal).
- Accede a <http://localhost:8080/productos> con método GET para ver la lista.
- Envía un POST con JSON a esa misma ruta para crear un producto nuevo.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

Es un **microservicio RESTful** llamado productos-service que permite:

- Ver un listado de productos (GET)
- Agregar nuevos productos (POST)

Está desarrollado con **Spring Boot** y sigue una **estructura en capas**, lo que lo hace escalable, mantenable y alineado con buenas prácticas de desarrollo backend.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

¿Qué es este ejercicio?

Es un **microservicio RESTful** llamado `productos-service` que permite:

- Ver un listado de productos (GET)
- Agregar nuevos productos (POST)

Está desarrollado con **Spring Boot** y sigue una **estructura en capas**, lo que lo hace escalable, mantenible y alineado con buenas prácticas de desarrollo backend.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

¿Qué componentes tiene y qué hace cada uno?

1. Producto.java

Capa: Modelo de dominio

Rol: Representa un producto del negocio (nombre, precio, stock).

Es una clase Java simple (POJO), sin lógica, solo datos.

2. ProductoService.java

Capa: Servicio (lógica de negocio)

Rol: Administra la lógica de productos.

- Contiene una lista interna (ArrayList) donde se almacenan los productos.

- Define métodos para:

- **listar productos**

- **guardar un producto nuevo**

Aquí podrías agregar validaciones de negocio, como verificar que el precio sea mayor que cero, etc.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

¿Qué componentes tiene y qué hace cada uno?

3. ProductoController.java

Capa: Controlador REST

Rol: Expone los métodos del servicio a través de HTTP.

Define dos endpoints:

- GET /productos: retorna la lista actual de productos (como JSON)
- POST /productos: permite enviar un nuevo producto en formato JSON para agregarlo a la lista

Este controlador usa anotaciones de Spring:

- `@RestController`: lo convierte en un controlador REST
- `@RequestMapping`: define la URL base /productos
- `@GetMapping, @PostMapping`: definen los métodos HTTP



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

¿Qué componentes tiene y qué hace cada uno?

4. ProductosApplication.java

Capa: Aplicación principal

Rol: Es el punto de entrada del microservicio.

Contiene el método `main()` con `SpringApplication.run(...)` que:

- Levanta el servidor embebido (Tomcat)
- Carga todas las clases anotadas con `@RestController`, `@Service`, etc.
- Inicia la aplicación escuchando en el puerto 8080



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

¿Qué componentes tiene y qué hace cada uno?

5. application.properties

Rol: Configura parámetros de la aplicación.

Por ejemplo:

server.port=8080

Esto indica que el servicio se expone por el puerto 8080.

6. pom.xml

Rol: Define las dependencias necesarias para que Spring Boot funcione.

Incluye:

- **spring-boot-starter-web:** para crear APIs REST con Tomcat y Jackson (JSON)
- **spring-boot-starter-test:** para pruebas unitarias (no usado por ahora)

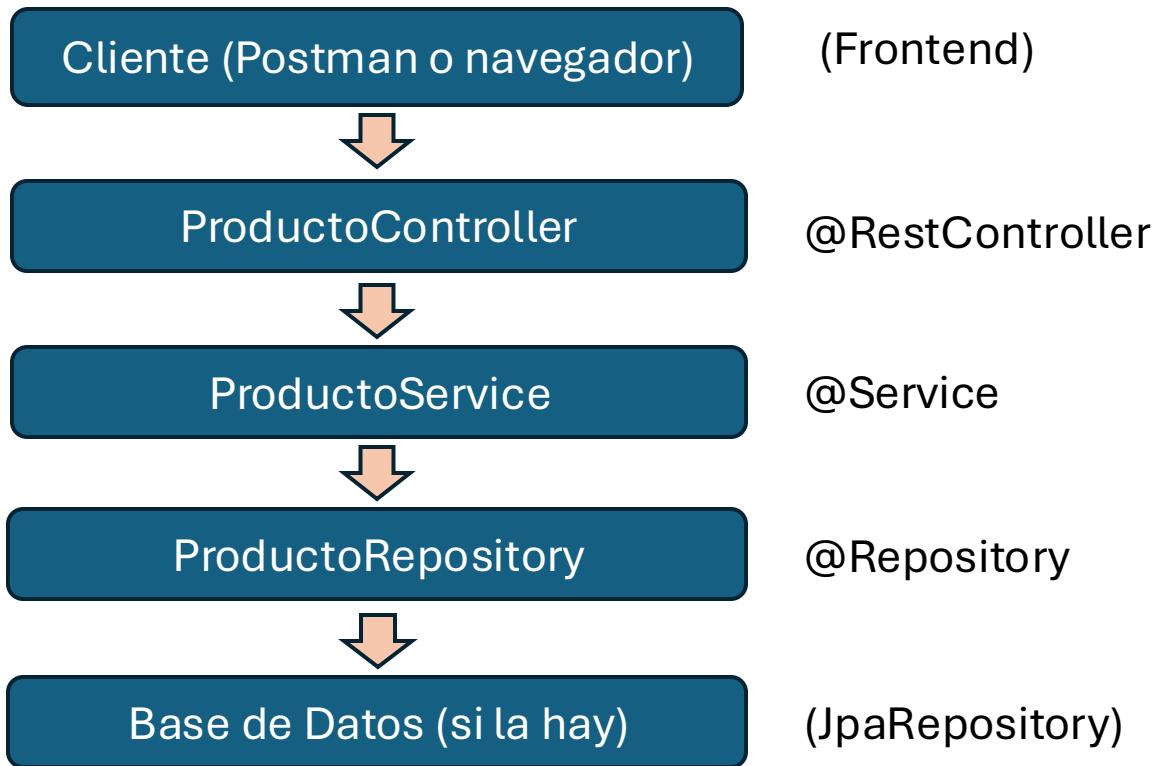


Módulo 2.- Desarrollo Back-End con Spring Boot

Creación de microservicios orientados al dominio del MSGG.

Ejemplo simple de microservicio de productos orientado al dominio.

Flujo completo



Módulo 2.- Desarrollo Back-End con Spring Boot

O Creación de microservicios orientados al dominio del MSGG.

Características del microservicio orientado al dominio

- Las clases están **nombradas y organizadas según el negocio** (Producto, no "DataHelper" o "Manager").
- Cada clase tiene **una responsabilidad clara**.
- Puedes extraer este microservicio como un módulo independiente fácilmente.



Módulo 2.- Desarrollo Back-End con Spring Boot

○ Creación de microservicios orientados al dominio del MSGG.

Resumen de buenas prácticas

- Diseña servicios centrados en el dominio
- Usa `@Service` para lógica, no en controladores
- Valida reglas del negocio
- Maneja errores correctamente
- Separa DTO de entidades (opcional)
- Versiona tu API REST (v1, v2...)

