Ecommerce con Spring Boot

Creacion de la BD: La base de datos se creara en MySQL Workbench y se llamara Ecommerce Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Creacion del Proyecto : la configuración inicial del proyecto Java para un sistema de ecommerce utilizando Spring Initializr, una herramienta oficial para generar proyectos Spring Boot preconfigurados.

**Project**

* **Maven**: Se ha seleccionado *Maven* como herramienta de construcción. Esta permite gestionar dependencias, compilar el proyecto y empaquetarlo.

**Language**

* **Java**: Se ha elegido Java como lenguaje principal del proyecto, en lugar de Kotlin o Groovy.

**Spring Boot Version**

* **3.5.4**: Es la versión estable seleccionada de Spring Boot, compatible con Java 17 y con mejoras respecto a versiones anteriores.

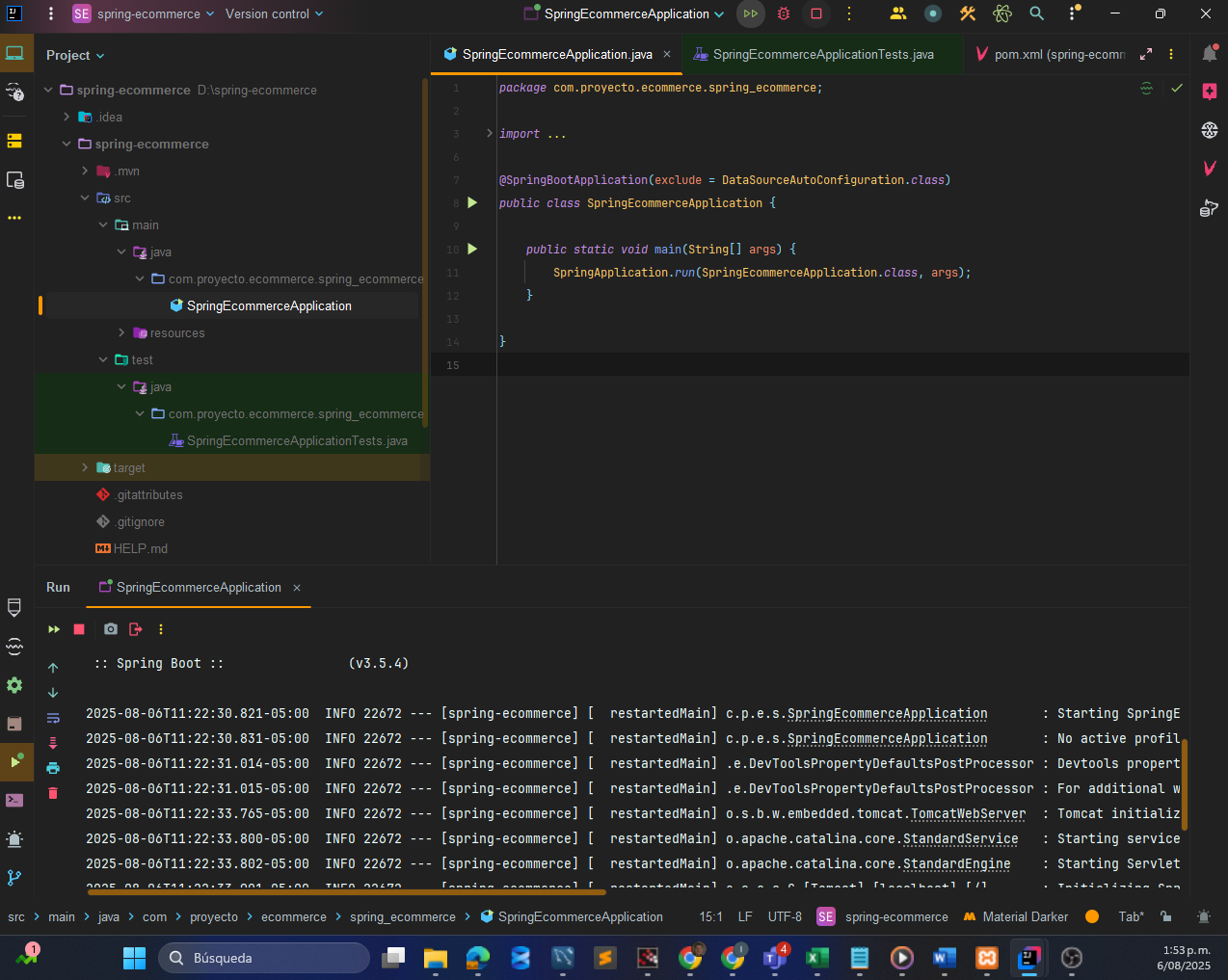
**Dependencias Incluidas**

Las siguientes librerías han sido seleccionadas al momento de generar el proyecto:

1. **Lombok (Developer Tools)**  
   Reduce el código repetitivo (boilerplate) mediante anotaciones como @Getter, @Setter, @Builder, etc.
2. **Spring Web (Web)**  
   Habilita la creación de aplicaciones web, especialmente RESTful APIs, usando Spring MVC y Tomcat como servidor embebido.
3. **Thymeleaf (Template Engines)**  
   Motor de plantillas que permite generar contenido HTML dinámico desde el servidor.
4. **Spring Data JPA (SQL)**  
   Facilita el acceso a bases de datos mediante Java Persistence API (JPA) y Hibernate.
5. **MySQL Driver (SQL)**  
   Controlador JDBC necesario para conectar la aplicación con bases de datos MySQL.
6. **Spring Boot DevTools (Developer Tools)**  
   Herramientas de desarrollo que permiten reinicios automáticos, recarga en caliente (LiveReload) y una experiencia de desarrollo más ágil.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.



**Estructurar el proyecto y crear las clases para el modelo 02**

Para la estructura del proyecto se necesita una **estructura típica de paquetes (carpetas) organizada por capas según la arquitectura por capas o arquitectura de tipo MVC (Modelo - Vista - Controlador).**

**Paquetes que aparecen:**

**1. controller**

**Responsabilidad:** manejar las peticiones HTTP que vienen del cliente (navegador, Postman, etc.).

* Contiene las clases con anotaciones como @RestController o @Controller.
* Cada método se encarga de atender una URL y delega la lógica al servicio.

**2. model**

**Responsabilidad:** definir las entidades del dominio (por ejemplo, productos, usuarios, órdenes, etc.).

* Contiene clases anotadas con @Entity si usas JPA.
* Representan las tablas de tu base de datos.

**3. repository**

**Responsabilidad:** acceder a la base de datos.

* Contiene interfaces que extienden JpaRepository, CrudRepository, etc.
* Spring genera la implementación automáticamente.

**4. service**

**Responsabilidad:** contener la lógica de negocio.

* Aquí se hace el procesamiento de datos, validaciones, llamadas a los repositorios, etc.
* Usualmente están anotadas con @Service.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

Dentro de la carpeta model creamos 4 clases de entidad que representan las tablas en la base de datos estas clases son Usuario, Producto. Orden, DetalleOrden y contienen la siguiente estructura

**Anotaciones de Lombok**

Estas anotaciones son parte de la librería Project Lombok que ayuda a reducir código repetitivo:

| **Anotación** | **¿Qué hace?** |
| --- | --- |
| @Getter | Genera automáticamente los métodos get para todos los atributos. |
| @Setter | Genera automáticamente los métodos set. |
| @AllArgsConstructor | Genera un constructor con todos los campos como parámetros. |
| @NoArgsConstructor | Genera un constructor vacío (sin argumentos). |

**Atributos de la clase**

La clase tiene los siguientes atributos:

Esto representa un usuario típico con:

* id: Identificador único.
* nombre: Nombre real.
* username: Nombre de usuario.
* mail: Correo electrónico.
* direccion: Dirección física.
* telefono: Número de contacto.
* tipo: Posiblemente el rol del usuario (ej: admin, cliente...).
* password: Contraseña (💡 no deberías mostrarla en toString por seguridad).

**🔁 Método toString()**

Este método sobrescrito (@Override) sirve para obtener una representación en texto del objeto, útil para logs o debugging.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.Texto

El contenido generado por IA puede ser incorrecto.Texto

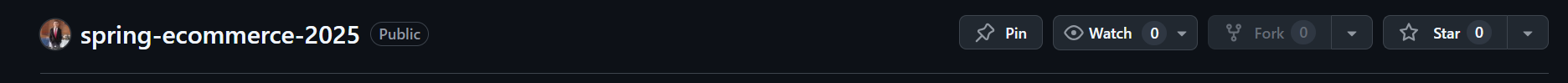
El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**Cómo versionar el proyecto usando Git y GitHub**

Lo Primero que haremos será crear el repositorio donde guardaremos todo el código que vamos a realizar para esto ingresamos a <https://github.com/Ivan-Cast> y creamos un repositorio en mi caso lo llame [**spring-ecommerce-2025**](https://github.com/Ivan-Cast/spring-ecommerce-2025)



Luego de esto desde gitbash o la terminal dentro del proyecto seleccionamos los siguientes comandos

cd /ruta/de/tu/proyecto

Inicializar Git (si aún no está inicializado)

git init

**Vincular el repositorio remoto que ya tienes en GitHub**

Copia la URL de tu repo de GitHub (por ejemplo: https://github.com/usuario/nombre-repo.git) y pon:

git remote add origin https://github.com/usuario/nombre-repo.git

Agregar los archivos al staging

git add .

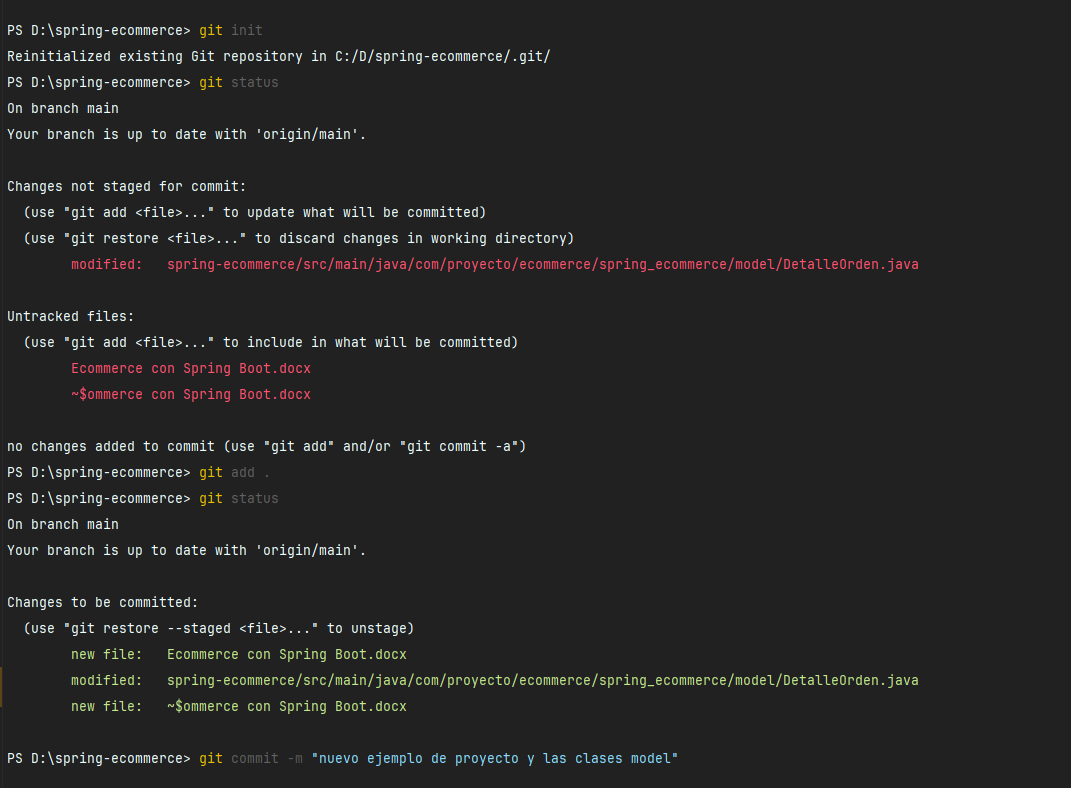
**Hacer el primer commit**

git commit -m "Subiendo proyecto inicial"

**Subirlo al repositorio remoto**

Si tu repositorio remoto ya tiene contenido y quieres fusionar con lo que tienes:

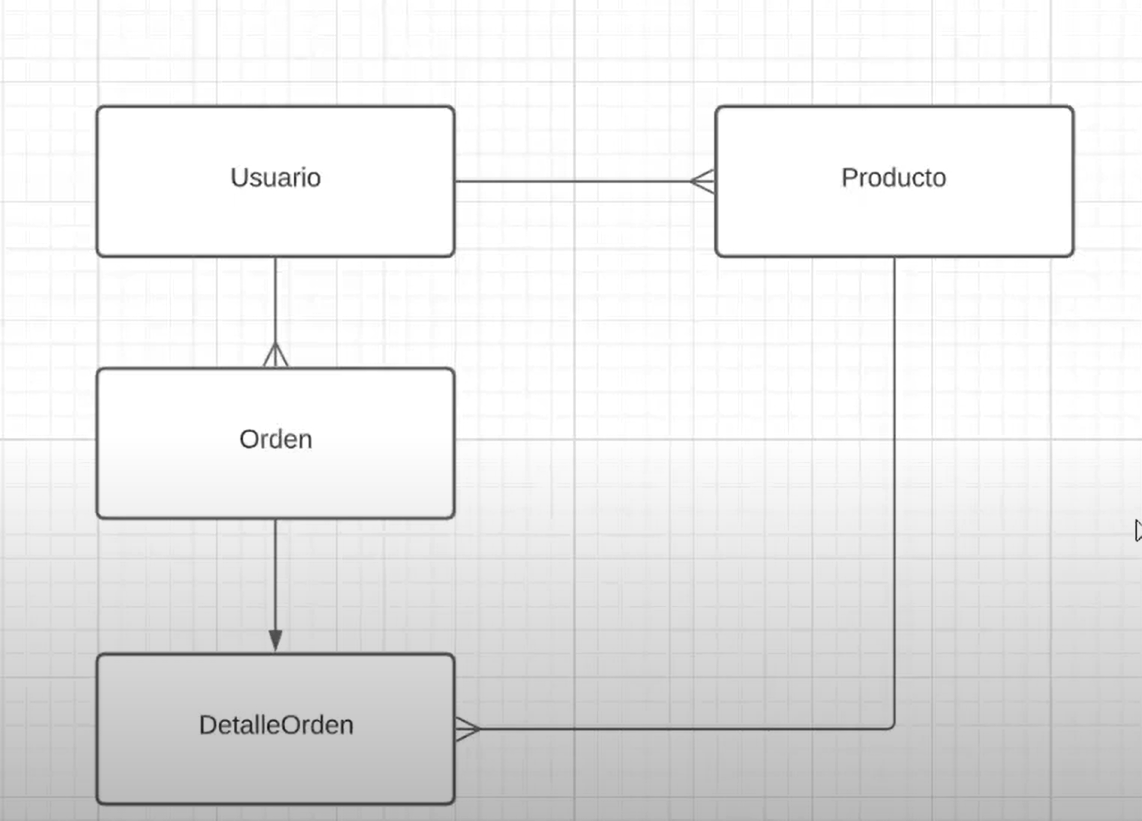
git pull origin main --rebase

****

**Texto

El contenido generado por IA puede ser incorrecto.**

**Mapeo de clases a tablas y relaciones JPA**

****

En este diagrama de entidad relación se muestra las relaciones que van a tener las diferentes tablas de nuestra Base de Datos

Un usuario puede tener de uno a varios productos, un usuario como administrador puede subir a la plataforma uno o varios productos

Un usuario puede tener varias ordenes por que puede realizar varias compras

Una orden puede tener únicamente un detalleOrden

En un detalleOrden puede haber uno o varios productos

**Tabla:** Usuarios

**Relaciones:**

* **Uno a muchos con Producto** → un usuario puede publicar varios productos.
* **Uno a muchos con Orden** → un usuario puede hacer varias órdenes.

**Claves foráneas:** las FK estarán en las tablas Productos y Ordenes apuntando al id de Usuarios.

Texto

El contenido generado por IA puede ser incorrecto.

**Tabla:** Productos

**Relaciones:**

* **Muchos a uno con Usuario** → muchos productos pueden pertenecer al mismo usuario.

**Clave foránea:** usuario\_id en Productos apuntando a Usuarios.id.

Texto

El contenido generado por IA puede ser incorrecto.

* **Tabla:** Ordenes
* **Relaciones:**
  + **Muchos a uno con Usuario** → muchas órdenes pueden ser hechas por un usuario.
  + **Uno a uno con DetalleOrden** → aquí asumes que cada orden tiene un solo detalle (en la vida real podría ser @OneToMany).
* **Clave foránea:** usuario\_id en Ordenes apuntando a Usuarios.id.

Texto

El contenido generado por IA puede ser incorrecto.

* **Tabla:** Detalles
* **Relaciones:**
  + **Uno a uno con Orden** → cada detalle pertenece a una sola orden.
  + **Uno a uno con Producto** → cada detalle hace referencia a un producto específico.
* **Claves foráneas:**
  + orden\_id en Detalles apuntando a Ordenes.id
  + producto\_id en Detalles apuntando a Productos.id

Texto

El contenido generado por IA puede ser incorrecto.

**Nombre de la aplicación**

spring.application.name=spring-ecommerce

* Define el nombre de tu aplicación en Spring Boot.
* Se usa internamente para logs, tracing o cuando usas servicios como Spring Cloud.

**Configuración de la conexión a MySQL**

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

* Indica el driver JDBC de MySQL que Spring Boot usará para conectarse.
* com.mysql.cj.jdbc.Driver es el driver moderno de MySQL (compatible con MySQL 8 y posteriores).

spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce

* Es la URL de conexión a la base de datos.
* localhost → el servidor MySQL está en tu PC.
* 3306 → el puerto por defecto de MySQL.
* ecommerce → el nombre de la base de datos.

spring.datasource.username=root

spring.datasource.password=

* Usuario y contraseña de MySQL.
* Aquí el usuario es root y no hay contraseñ

**Configuración de Hibernate / JPA**

spring.jpa.hibernate.ddl-auto=update

* Indica cómo Hibernate debe manejar el esquema de la base de datos:
  + update → actualiza tablas y columnas según tus entidades, sin borrar datos.
  + Otras opciones: create, create-drop, validate, none.

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

* Le dice a Hibernate qué "dialecto" SQL usar (en este caso, el de MySQL).
* Esto permite que Hibernate genere SQL compatible con MySQL.

spring.jpa.show-sql=true

* Muestra en consola las consultas SQL que ejecuta Hibernate.

spring.jpa.properties.hibernate.format\_sql=true

* Formatea las consultas SQL para que se vean más legibles en la consola.

**Estrategia de nombres de tablas y columnas**

spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl

* Define cómo Hibernate va a convertir nombres de atributos/clases a nombres de columnas/tablas en la base de datos.
* PhysicalNamingStrategyStandardImpl significa que usará exactamente los nombres que pongas en las anotaciones (@Table, @Column), sin modificar a snake\_case.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Conceptos**

@SpringBootApplication(exclude = DataSourceAutoConfiguration.*class*):

**¿Qué hace @SpringBootApplication?**

Esa anotación es un **atajo** que combina varias anotaciones de Spring Boot:

@Configuration

@EnableAutoConfiguration

@ComponentScan

Es decir: le dice a Spring Boot:

“Configura automáticamente mi aplicación, escanea los componentes y usa esta clase como configuración principal.”

**¿Y qué significa exclude = DataSourceAutoConfiguration.class?**

Spring Boot tiene una característica llamada **"auto-configuración"**, donde **detecta automáticamente** que tienes una base de datos y configura la conexión por ti.

Pero si no tienes una base de datos configurada (como es tu caso), **esa auto-configuración falla**.

Al hacer esto:

@SpringBootApplication(exclude = DataSourceAutoConfiguration.class)

Le estás diciendo a Spring Boot:

“❌ No intentes configurar una base de datos automáticamente.”

@Entity

**Qué hace:**  
Indica que la clase es una entidad JPA, es decir, que se va a mapear a una tabla en la base de datos.

**Dónde se usa:**  
Encima de la clase.

**Efecto:**  
Hibernate/JPA creará una tabla (o usará una existente) para almacenar objetos de esta clase.

@Table(name = "...")

**Qué hace:**  
Especifica el nombre de la tabla en la base de datos que representa a la entidad.

**Dónde se usa:**  
Encima de la clase, junto con @Entity.

**Efecto:**  
La tabla se llamará Usuarios en la base de datos, en lugar de usar el nombre por defecto (usuario).

@Id

**Qué hace:**  
Marca el campo que es la clave primaria de la tabla.

**Dónde se usa:**  
En un atributo de la entidad.

**Efecto:**  
Ese campo será la columna de clave primaria.

@GeneratedValue(strategy = ...)

**Qué hace:**  
Indica que el valor de la clave primaria se genera automáticamente.

**Estrategias comunes:**

* + IDENTITY: Usa autoincremento de la base de datos (MySQL, SQL Server).
  + SEQUENCE: Usa una secuencia (PostgreSQL, Oracle).
  + AUTO: Hibernate elige la estrategia según la base de datos.

@OneToMany(mappedBy = "...")

**Qué hace:**  
Define una relación **uno a muchos**.  
mappedBy indica el **nombre del atributo** en la otra entidad que posee la relación (el lado propietario).

**Dónde se usa:**  
En colecciones (List, Set).

**Efecto:**  
Un usuario puede tener muchos productos, pero el campo usuario en Producto es quien guarda la clave foránea.

@ManyToOne

**Qué hace:**  
Define una relación **muchos a uno** (muchos registros de esta entidad pertenecen a un registro de otra entidad).

**Dónde se usa:**  
En el lado que tiene la clave foránea.

**Efecto:**  
En la tabla de esta entidad, habrá una columna usuario\_id que apunta a la tabla Usuarios.

@OneToOne

**Qué hace:**  
Define una relación **uno a uno** entre dos entidades.

**Dónde se usa:**  
En ambos lados de la relación, pero uno es propietario y otro usa mappedBy.

**Efecto:**  
Una fila en una tabla se relaciona con exactamente una fila en otra tabla.

mappedBy

**Qué hace:**  
Indica que la relación **es inversa** y que el campo mencionado es el que realmente guarda la relación en la base de datos.