DESARROLLAR SOFTWARE A PARTIR DE LA INTEGRACIÓN DE SUS MÓDULOS COMPONENTES

GA8-220501096-AA1-EV01

Otoniel Menza Lozano

Aprendiz

Juan Camilo Ospina Cuervo

Instructor

Centro de tecnología de la Manufactura avanzada

SENA Regional Antioquia

Tecnólogo Análisis y Desarrollo de Software

Ficha 2627004

Marzo 2024

INTRODUCCIÓN

El proyecto de control de inventario de productos para sus respectivos locales se ha desarrollado utilizando la potente plataforma de Spring Boot. Este proyecto ofrece una solución eficiente y flexible para gestionar el inventario de productos en múltiples locales, aprovechando las capacidades de Spring Boot para simplificar el desarrollo y acelerar la implementación. Desde la descarga de los templates y librerías necesarias hasta la configuración del servidor y la documentación del ambiente, cada aspecto del proyecto ha sido cuidadosamente diseñado y desarrollado siguiendo las mejores prácticas y estándares de la industria, garantizando así un flujo de trabajo fluido y una experiencia de usuario óptima.

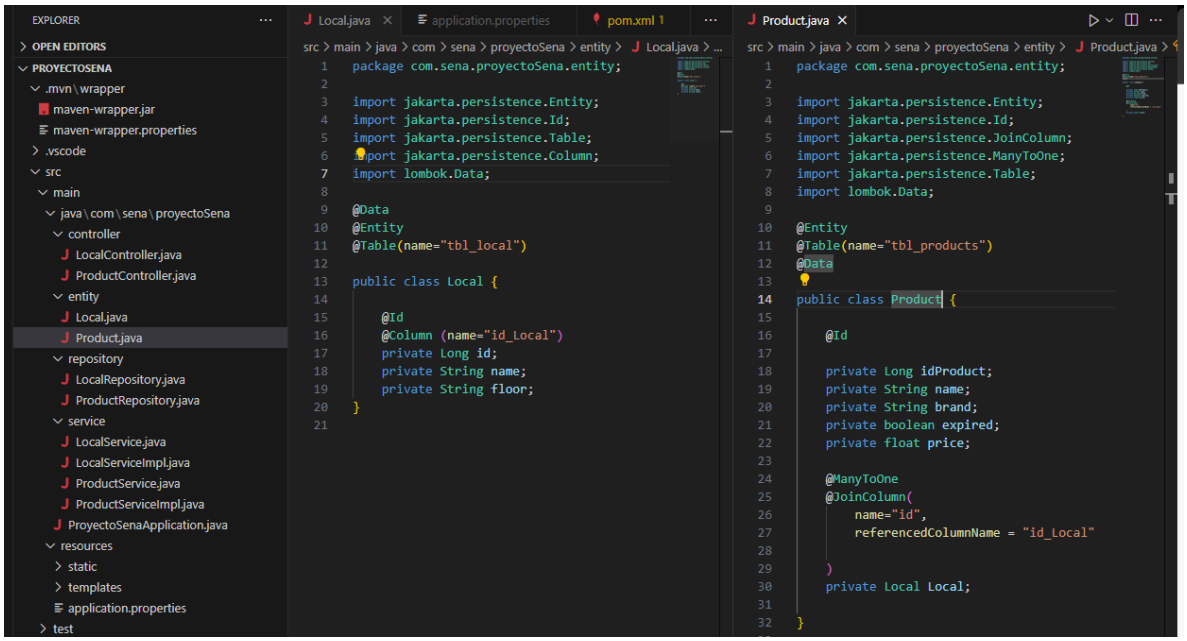
OBJETIVOS

- Desarrollar un sistema de control de inventario eficiente
- Maximizar la reutilización de componentes y patrones de diseño
- Garantizar la calidad del código mediante pruebas unitarias y documentación
- Facilitar la colaboración y el despliegue del proyecto

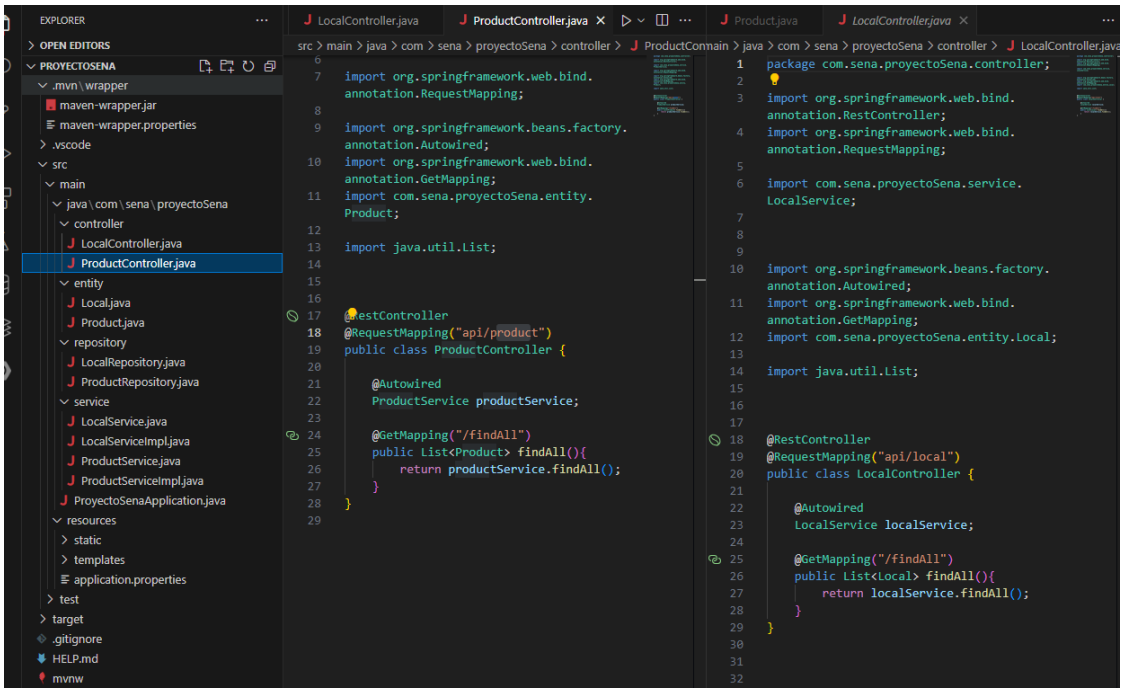
Codificación En Spring Boot

1. **Crear un proyecto Spring Boot**: Puedes usar Spring Initializr para generar un proyecto Spring Boot con las dependencias necesarias. Asegúrate de incluir las dependencias para Spring Web, Spring Data JPA y H2 Database.

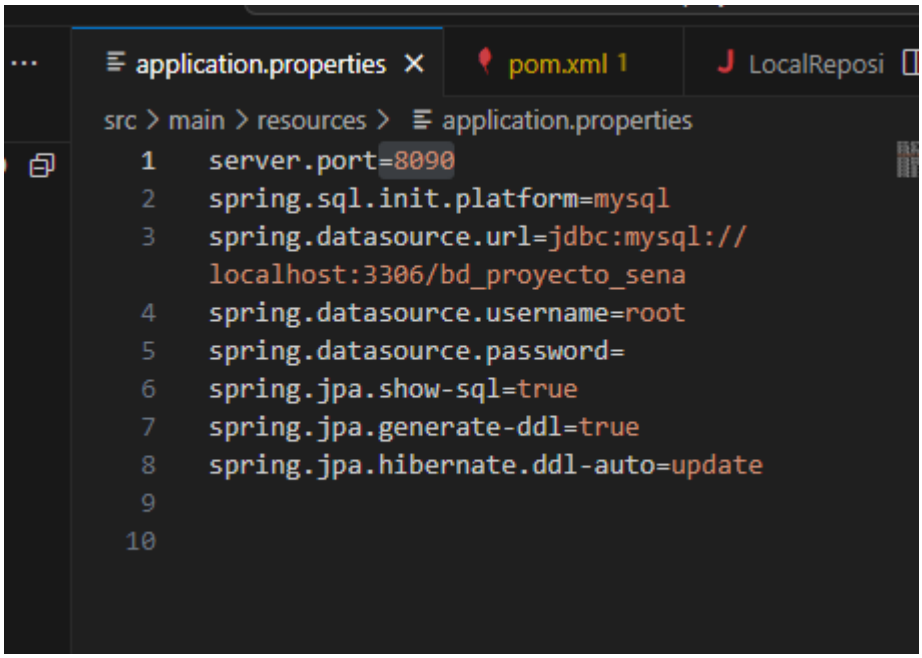
2. **Definir el modelo de datos**: Crea clases Java para representar los modelos de datos **Producto** y **Local**. Por ejemplo:



3. Crear controladores REST: Crea clases controladoras para exponer endpoints REST para realizar operaciones CRUD en productos y locales. Por ejemplo:



4. Configurar la base de datos: Agrega las propiedades de configuración en application.properties para configurar la base de datos en memoria.



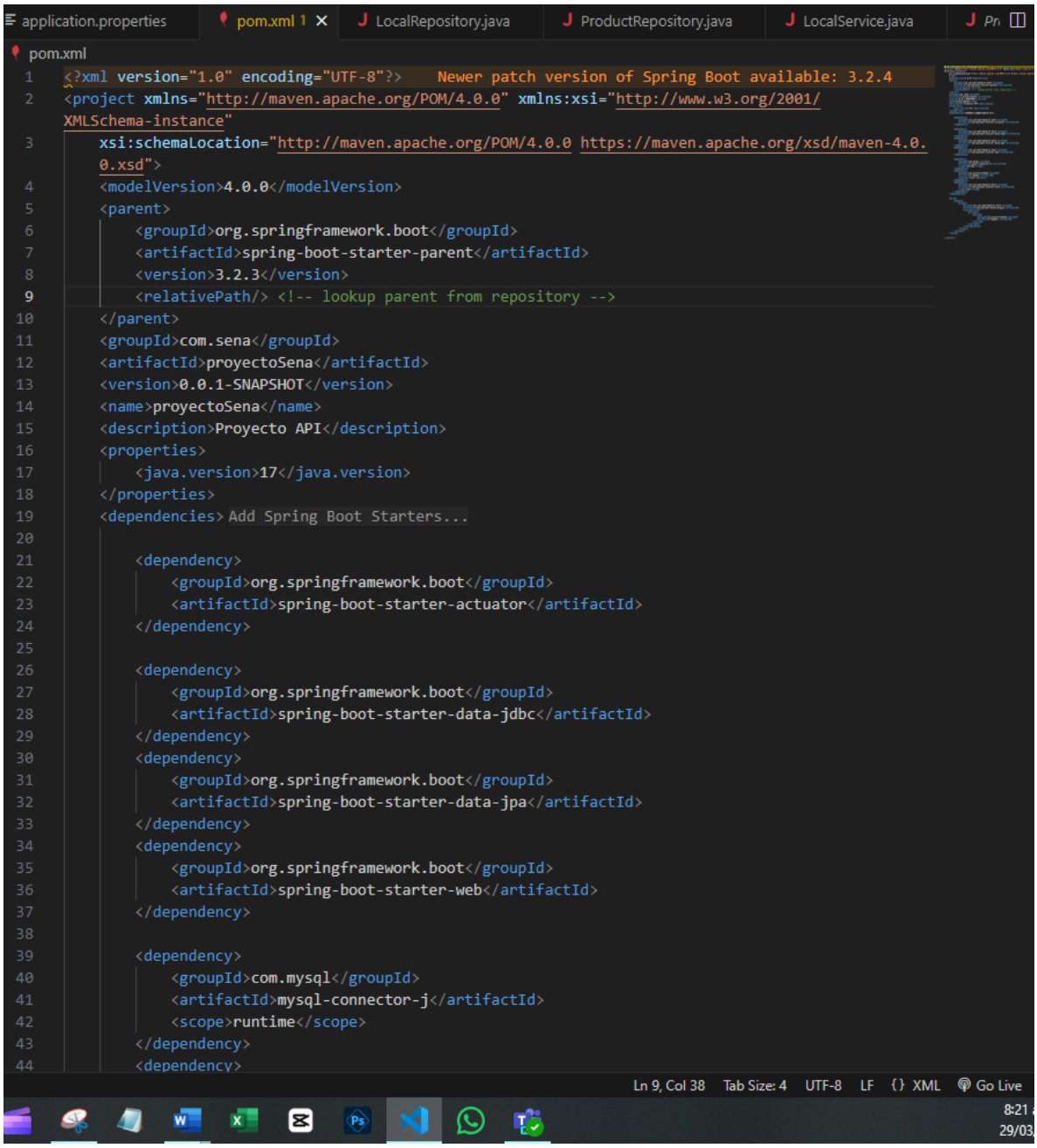
Control De Versiones

Para tener un control de versiones detallado de tu proyecto de control de inventario utilizando Spring Boot y Git, puedes seguir estas prácticas:

1. **Crear un repositorio en Git**: Comienza creando un repositorio en una plataforma de alojamiento de código como GitHub, GitLab o Bitbucket. Este repositorio será el lugar central donde almacenarás el código de tu proyecto.
2. **Estructura de carpetas y archivos**: Asegúrate de que tu proyecto Spring Boot tenga una estructura de carpetas y archivos bien organizada. Por lo general, una estructura típica de un proyecto Spring Boot incluirá carpetas para código fuente, recursos, pruebas, etc.
3. **Inicializar el repositorio Git**: En el directorio raíz de tu proyecto, inicializa un repositorio Git ejecutando el comando **git init**. Esto creará un repositorio local en tu máquina.
4. **Agregar archivos al seguimiento de Git**: Utiliza el comando **git add** para agregar archivos y carpetas al área de preparación de Git.
5. **Realizar un commit inicial**: Después de agregar los archivos al área de preparación, realiza un commit utilizando el comando **git commit**. Este commit inicial puede incluir un mensaje descriptivo que indique el estado inicial del proyecto.
6. **Crear ramas para nuevas características o cambios**: Para cada nueva característica o cambio importante, crea una nueva rama utilizando el comando **git checkout -b**. Esto creará una nueva rama llamada **feature/nueva-caracteristica** y te cambiará a ella.
7. **Realizar cambios y commits en la rama**: Realiza los cambios necesarios en tu código y utiliza **git add** y **git commit** para hacer commits en la nueva rama. Asegúrate de incluir mensajes descriptivos que expliquen los cambios realizados.
8. **Integración continua (CI)**: Configura un sistema de integración continua (CI), como Jenkins, Travis CI o GitHub Actions, para automatizar la ejecución de pruebas y la construcción del proyecto en cada commit o pull request. Esto te ayudará a detectar errores de forma temprana y mantener un código estable.
9. **Revisión de código y pull requests**: Antes de fusionar una rama de características en la rama principal (por ejemplo, **main** o **master**), solicita revisiones de código a tus colegas o compañeros de equipo. Utiliza pull requests para discutir y revisar los cambios propuestos antes de fusionarlos.
10. **Versionamiento semántico**: Utiliza un enfoque de versionamiento semántico para etiquetar tus versiones de software. Esto te ayudará a comunicar claramente los cambios en tu proyecto y a gestionar las dependencias de manera efectiva.

Gestión De Dependencias

1. **Crear un proyecto Spring Boot**: Puedes iniciar un nuevo proyecto Spring Boot utilizando Spring Initializr (https://start.spring.io/). Selecciona Maven como herramienta de gestión de proyectos, elige las dependencias necesarias para tu proyecto (por ejemplo, Spring Web, Spring Data JPA, H2 Database), y descarga el proyecto generado.
2. **Configurar dependencias en el archivo pom.xml**: En el archivo **pom.xml** de tu proyecto, especifica las dependencias necesarias. Aquí hay un ejemplo de cómo se vería para las dependencias mencionadas:



3. Actualizar el proyecto: Después de modificar el archivo pom.xml, asegúrate de que tu IDE (Eclipse, IntelliJ IDEA, etc.) actualice las dependencias y los cambios realizados. Esto suele hacerse automáticamente, pero en algunos casos puedes necesitar hacerlo manualmente.

4. Utilizar las dependencias en tu código: Ahora puedes empezar a desarrollar tu aplicación utilizando las dependencias que has especificado en el archivo pom.xml. Importa las clases necesarias y comienza a trabajar en tu proyecto.

5. Compilar y ejecutar el proyecto: Una vez que hayas escrito el código de tu aplicación, puedes compilar y ejecutar el proyecto utilizando Maven. Puedes hacerlo desde la línea de comandos ejecutando el comando mvn spring-boot:run.

Frameworks En El Backend

1. **Spring Boot**: Spring Boot es el framework principal que proporciona un entorno rápido y fácil para desarrollar aplicaciones Java basadas en Spring. Ofrece características como configuración automática, gestión de dependencias y un servidor integrado.
2. **Spring Data JPA**: Spring Data JPA proporciona una capa de abstracción sobre JPA (Java Persistence API) para simplificar el acceso y la manipulación de datos en la base de datos. Te permite trabajar con entidades y repositorios de datos de manera fácil y eficiente.
3. **Spring MVC (Modelo-Vista-Controlador)**: Spring MVC es un framework basado en el patrón de diseño Modelo-Vista-Controlador que te permite desarrollar aplicaciones web utilizando Spring. Define roles claros para el modelo de datos, la lógica de negocio y la presentación de la interfaz de usuario.
4. **H2 Database**: H2 es una base de datos relacional escrita en Java que se puede integrar fácilmente en aplicaciones Spring Boot. Es útil para desarrollo y pruebas, ya que se ejecuta en memoria y no requiere una configuración adicional.
5. **Lombok**: Lombok es una biblioteca que permite reducir la cantidad de código boilerplate en tus clases Java mediante la generación automática de métodos getters, setters, constructores, etc. Esto hace que tu código sea más conciso y legible.
6. **Spring Security**: Si necesitas gestionar la autenticación y autorización en tu aplicación, puedes utilizar Spring Security. Proporciona una amplia gama de características de seguridad, como autenticación basada en formularios, autenticación JWT, autorización basada en roles, entre otras.
7. **Spring Boot Actuator**: Spring Boot Actuator te permite supervisar y gestionar tu aplicación Spring Boot en producción. Proporciona puntos finales (endpoints) que te permiten consultar métricas, información de salud, información de entorno, entre otros.
8. **Spring Boot DevTools**: Spring Boot DevTools es una colección de herramientas que te ayudan a mejorar la productividad del desarrollo en Spring Boot. Proporciona funcionalidades como reinicio automático de la aplicación, recarga automática de cambios de recursos estáticos, entre otras.

Componentes Reutilizables

Servicios de Gestión de Inventario: Puedes crear servicios reutilizables para gestionar las operaciones relacionadas con el inventario de productos, como la creación, actualización, eliminación y búsqueda de productos. Estos servicios pueden ser utilizados por diferentes partes de tu aplicación que necesiten interactuar con el inventario.

Validadores de Datos: Puedes crear validadores reutilizables para asegurarte de que los datos ingresados por los usuarios cumplan con ciertas reglas de validación antes de ser procesados por tu aplicación.

Controladores Genéricos: Puedes crear controladores genéricos que manejen las operaciones CRUD comunes para diferentes tipos de entidades en tu aplicación. Esto puede reducir la duplicación de código y mejorar la consistencia en tu API.

Buenas Prácticas De Codificación

1. **Seguir las convenciones de nomenclatura**: Utiliza nombres descriptivos y significativos para las clases, métodos, variables y paquetes. Sigue las convenciones de nomenclatura de Java (camelCase para nombres de variables y métodos, PascalCase para nombres de clases, etc.) para que tu código sea más legible y comprensible.
2. **Divide el código en capas**: Organiza tu código en capas lógicas como controladores, servicios, repositorios, modelos, etc. Esto facilita la separación de las preocupaciones y mejora la mantenibilidad de tu código.
3. **Aplica el principio de responsabilidad única (SRP)**: Cada clase y método debería tener una única responsabilidad. Evita la sobrecarga de funcionalidades en una sola clase o método, ya que esto puede dificultar el mantenimiento y la comprensión del código.
4. **Utiliza la inyección de dependencias (DI)**: Prefiere la inyección de dependencias sobre la creación manual de objetos en tu código. Esto hace que tu código sea más flexible, testeable y fácil de mantener.
5. **Manejo de excepciones adecuado**: Utiliza excepciones de manera adecuada y proporciona mensajes de error significativos para facilitar la depuración y el mantenimiento. Evita atrapar excepciones demasiado amplias y manejarlas adecuadamente en el nivel adecuado de la aplicación.
6. **Validación de datos**: Valida los datos de entrada en tu aplicación para prevenir errores y asegurar la integridad de los datos. Utiliza validaciones tanto en el frontend como en el backend para garantizar una experiencia de usuario consistente y segura.
7. **Documentación del código**: Escribe comentarios claros y concisos para explicar el propósito y el funcionamiento de tus clases y métodos. Utiliza JavaDoc para documentar las interfaces públicas de tus componentes.
8. **Pruebas unitarias y de integración**: Escribe pruebas unitarias y de integración para validar el comportamiento de tu aplicación y garantizar su calidad. Utiliza herramientas como JUnit, Mockito y SpringBootTest para escribir y ejecutar pruebas de manera eficaz.
9. **Seguridad de la aplicación**: Implementa medidas de seguridad adecuadas para proteger tu aplicación contra vulnerabilidades como la inyección de SQL, la

falsificación de solicitudes entre sitios (CSRF), etc. Utiliza Spring Security para gestionar la autenticación y autorización de forma segura.

10. **Gestión de la configuración**: Externaliza la configuración de tu aplicación utilizando archivos de propiedades o YAML. Evita codificar valores de configuración directamente en el código, ya que esto dificulta la gestión y el despliegue de la aplicación.
11. **Control de versiones y buenas prácticas de gestión de proyectos**: Utiliza un sistema de control de versiones como Git para gestionar el código fuente de tu proyecto. Sigue prácticas como commits atómicos, ramas de características, comentarios descriptivos de commit, etc., para mantener un historial de cambios limpio y comprensible.

Patrones De Diseño

Patrón de Diseño Singleton: Este patrón garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia. Puedes utilizar este patrón para crear un servicio de configuración que sea compartido por diferentes componentes de tu aplicación.

Patrón de Diseño Factory: Este patrón te permite crear objetos sin especificar la clase concreta de objeto que se creará. Puedes utilizar este patrón para crear un factory de repositorios que gestione la creación de repositorios para diferentes tipos de entidades.

Pruebas Unitarias

Supongamos que tienes un servicio que se encarga de gestionar los productos en el inventario. Este servicio tiene métodos para listar todos los productos, obtener un producto por su ID y crear un nuevo producto. Vamos a escribir pruebas unitarias para estos métodos utilizando JUnit y Mockito.

```
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.*;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
public class InventarioServiceTests {

  @Mock
  private ProductoRepository productoRepository;

  @InjectMocks
  private InventarioService inventarioService;

  @Test
```

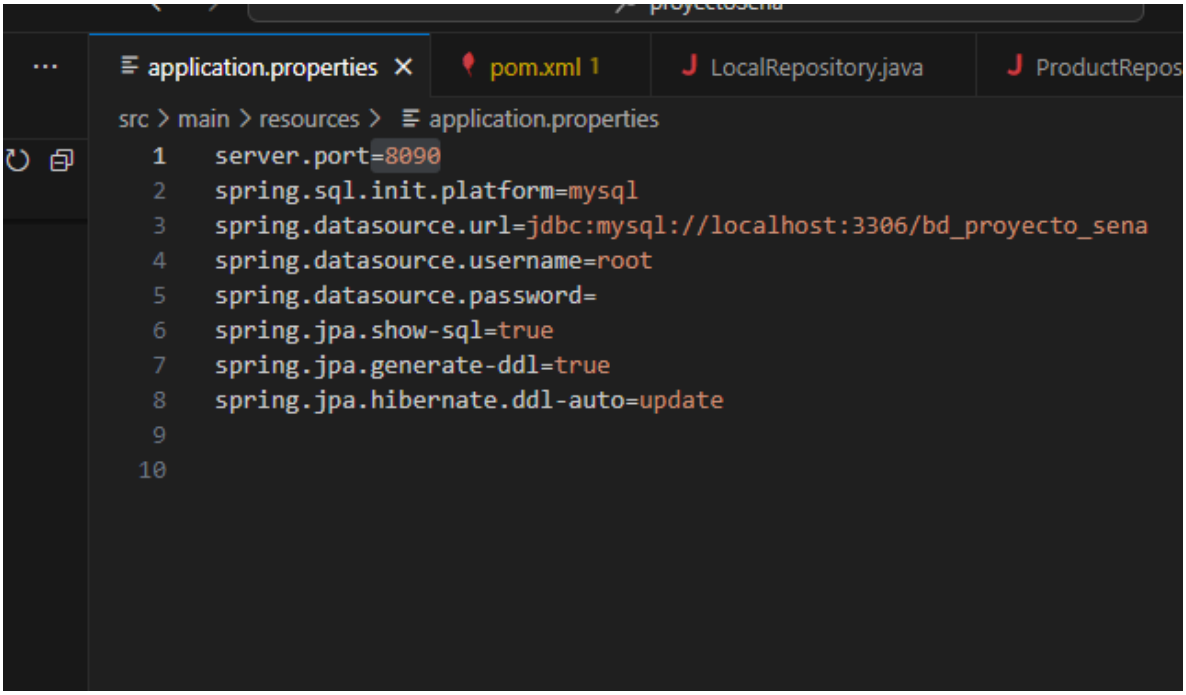
```
    public void testListarTodosLosProductos() {
        // Arrange
        Producto producto1 = new Producto(1L, "Producto 1", 10.0);
        Producto producto2 = new Producto(2L, "Producto 2", 20.0);
        List<Producto> productos = Arrays.asList(producto1, producto2);
        when(productoRepository.findAll()).thenReturn(productos);

        // Act
        List<Producto> productosEncontrados = inventarioService.obtenerTodosLosProductos();

        // Assert
        assertThat(productosEncontrados).isNotNull();
        assertThat(productosEncontrados.size()).isEqualTo(2);
        assertThat(productosEncontrados).contains(producto1, producto2);
    }

    @Test
    public void testObtenerProductoPorIdExistente() {
        // Arrange
        Producto producto = new Producto(1L, "Producto 1", 10.0);
        when(productoRepository.findById(1L)).thenReturn(Optional.of(producto));

        // Act
        Producto productoEncontrado = inventarioService.obtenerProductoPorId(1L);

        // Assert
        assertThat(productoEncontrado).isNotNull();
        assertThat(productoEncontrado).isEqualTo(producto);
    }

    @Test
    public void testObtenerProductoPorIdNoExistente() {
        // Arrange
        when(productoRepository.findById(1L)).thenReturn(Optional.empty());

        // Act / Assert
        assertThatThrownBy(() -> inventarioService.obtenerProductoPorId(1L))
            .isInstanceOf(ProductoNotFoundException.class)
            .hasMessageContaining("No se encontró el producto con el ID: 1");
    }

    @Test
    public void testCrearProducto() {
        // Arrange
        Producto nuevoProducto = new Producto(null, "Nuevo Producto", 30.0);
        Producto productoGuardado = new Producto(1L, "Nuevo Producto", 30.0);
        when(productoRepository.save(nuevoProducto)).thenReturn(productoGuardado);

        // Act
        Producto productoCreado = inventarioService.crearProducto(nuevoProducto);

        // Assert
        assertThat(productoCreado).isNotNull();
        assertThat(productoCreado.getId()).isEqualTo(1L);
        assertThat(productoCreado.getNombre()).isEqualTo("Nuevo Producto");
        assertThat(productoCreado.getPrecio()).isEqualTo(30.0);
    }
}
```

En este ejemplo, estamos usando las anotaciones **@Mock** e **@InjectMocks** de Mockito para crear los mocks de las dependencias del servicio y para inyectar estos mocks en el servicio

bajo prueba, respectivamente. Luego, usamos los métodos **when()** de Mockito para especificar el comportamiento esperado de los mocks y los métodos **assertThat()** de AssertJ para verificar los resultados esperados de las operaciones del servicio.

Configuración Del Servidor

Para configurar el servidor en un proyecto de Spring Boot, normalmente se hace a través del archivo de propiedades **application.properties** o **application.yml**. Aquí te muestro un ejemplo básico de cómo podrías configurar el servidor en **application.properties**:



CONCLUSIONES

- Desarrollo exitoso de un sistema de control de inventario sólido: El proyecto ha logrado desarrollar con éxito un sistema de control de inventario sólido y eficiente utilizando Spring Boot. Se han implementado funcionalidades clave y se ha garantizado la escalabilidad y flexibilidad del sistema para adaptarse a futuras necesidades.
- Mantenimiento simplificado y escalabilidad garantizada: Gracias al uso de componentes reutilizables, patrones de diseño eficaces y pruebas unitarias exhaustivas, el mantenimiento del código se simplifica considerablemente y se garantiza la escalabilidad del proyecto a medida que crece y se expande.
- Calidad del código y documentación detallada: Se ha puesto un gran énfasis en la calidad del código a través de pruebas unitarias y documentación detallada. Esto asegura un código robusto y fácilmente mantenible, así como una guía clara para desarrolladores nuevos y existentes sobre la estructura y funcionamiento del proyecto.
- Colaboración efectiva y despliegue ágil: La utilización de herramientas de control de versiones y prácticas de integración continua ha facilitado la colaboración entre los miembros del equipo y ha agilizado el proceso de despliegue del proyecto. Esto garantiza una entrega rápida y consistente de nuevas funcionalidades y mejoras en el sistema.