UNIVERSIDAD DEL BÍO-BÍO
FACULTAD DE CIENCIAS EMPRESARIALES
SISTEMAS DE INFORMACIÓN

Implementar conexión Spring Boot con MySQL, patrones de diseño y testings con Junit.

INTEGRANTE: Benjamín Jiménez Chandía

Docentes: Roberto Anabalon - Gastón Márquez - César Aguilera

Asignatura: Ingeniería de software

Fecha: 13/11/2025

Chillán - Chile

Introducción

Este informe detalla el desarrollo de un sistema de back-end para una Muebleria. El objetivo principal de esta evaluación es aplicar el stack tecnológico y los conocimientos adquiridos en clases para construir un software robusto y funcional.

El proyecto consiste en una API REST construida con el framework Spring Boot (versión 3.5.7) y Java 21, conectada a una base de datos MySQL local. El sistema implementa la lógica de negocio clave de la mueblería, permitiendo la gestión de un catálogo (CRUD), el registro de variaciones de productos que afectan el precio, y un módulo transaccional para procesar cotizaciones y ventas, asegurando la consistencia del stock.

Finalmente, el proyecto pone énfasis en la calidad del software mediante la implementación de patrones de diseño específicos para resolver problemas de arquitectura y la validación de la lógica de negocio a través de pruebas unitarias con JUnit.

Repositorio en github: https://github.com/benja0320/Eval2-Ing-Soft-Muebleria.git

Funcionamiento del repositorio

El proyecto es una API REST construida sobre Spring Boot, la cual se inicia mediante la clase principal MuebleriaApplication. La arquitectura sigue un patrón de tres capas:

Capa de Controladores: Clases como MuebleController y CotizacionController exponen las URL de la API. Reciben peticiones HTTP y DTOs (como MuebleDTO) validados para interactuar con el sistema.

Capa de Servicios: Aquí reside la lógica de negocio. Clases como VentaService implementan las reglas clave, como la validación de stock, usando @Transactional para asegurar la consistencia de los datos.

Capa de Repositorios: Interfaces como MuebleRepository usan Spring Data JPA para comunicarse con la base de datos MySQL y gestionar las entidades (ej. Mueble).

Adicionalmente, el sistema centraliza el manejo de errores (como StockInsuficienteException) mediante la clase GlobalExceptionHandler. También se

incluye un MenuRunner que provee una interfaz de línea de comandos (para operar el sistema directamente desde la terminal.

Para resolver directrices, se seleccionaron patrones de diseño específicamente enfocados en resolver problemas específicos a la lógica de negocio y arquitectura, en este caso se optó por implementar el patrón Repositorio y el patrón Decorador

Patron Decorador

Debido a que el problema pedía explícitamente registrar variaciones que modifiquen o aumenten el precio de un producto se obito por implementar una forma flexible de calcular el precio final de un mueble, que permitiera que un mueble tenga 0, 1 o muchas variaciones.

Este componente se implemento manualmente en el paquete patterns.decorator y sus componentes relevantes son: CalcularPrecio, corresponde a una interfaz base con método para calcular el precio; Mueble, la cual devuelve el precio base del mueble; VariacionDecorator decora un mueble utilizando la entidad variación, el metodo calcularPrecio llama al precio del objeto envuelto y le suma el precio de la variación con aumentarPrecio.

Patron Repository

Este patron esta enfocado en separa la logica de negocios de la logica dedicada al acceso a datos. El propósito de este patron (y de spring-boot) es evitar acoplar fuertemente la apicasionde MySQL, evitando escribir explicaciones instrucciones en SQL.

Este patron es provisto por spring Data JPA de forma nativa, culla dependencia desde el inicio en el pom.xml, y se imprenta creando interfaces en la carpeta Repository como MuebleRepository, CotizacionRepository y VariacionRepository. Destacar que Spring boot automáticamente provee los métodos DRUD para las entidades al extender JpaRepository .

Las clases de servicio como MuebleService o VentaService solo inyectan las interfaces y las utilizan manteniendo la logica de negocios limpia e independiente de la base de datos subyacentes.

Instrucciones de ejecución

Para ejecutar el proyecto se recomienda colocarlo en su carpeta de preferencia usando "git clone https://github.com/benja0320/Eval2-Ing-Soft-Muebleria.git ". Una vez clonado el repositorio debemos preparar el terreno para ejecutarlo.

Primero debe abrir la carpeta db, ahí encontrará el archivo .sql de la base de datos. Como se desea facilitar el ingreso de dato, se ha proporcionado una base de datos con elementos ya ingresados. Lo que usted debe hacer es encender los servicios de xampp, Posteriormente ingresar a "http://localhost/phpmyadmin/", en la parte superior seleccionar "importar", seleccionar el archivo "muebleria_db". De esta manera ya habra cargado la BDD.

Para ejecutar el proyecto, debe dirigirse al directorio "src/main/java/com/example/muebleria/" y buscar el archivo MenuRunner.java o MuebleriaApplication.java (con cualquiera de los dos se puede ejecutar). Abrir la clase y en su editor de codigo (De preferencia vsCode o intelij IDEA) seleccionar el triángulo superior derecho y dale a RUN con java. Si hizo todo lo anterior correctamente el programa debería arrancar sin problemas.

Para ejecutar los test, se recomienda utilizar Intelij IDEA puesto que, el vsCode no reconoce la clase VariacionServiceTest.java como una clase test, también se recomienda realizar cada test (o clase completa de test) por separado, esto para enviar errores imprevistos.

Pruebas y Resultados

Las siguientes imágenes muestran resultados obtenidos después de las interacciones realizadas:

Creación de un mueble:

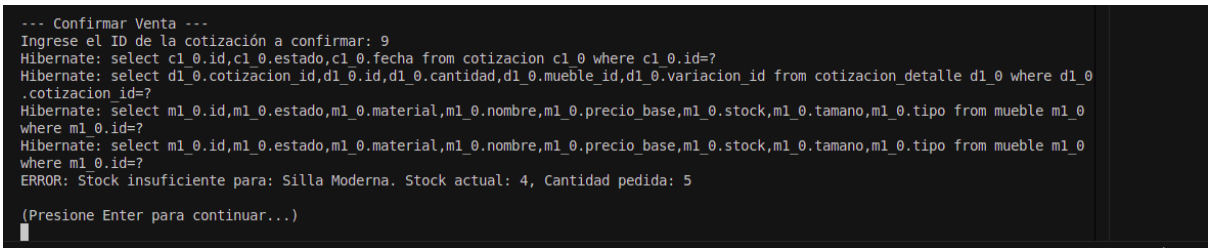
```
(base) benjamin@trillem:/media/benjamin/Data/tareas/Inge Software/muebleria$  cd /media/benjamin/Data/tareas/Inge\ Software/muebler
ia ; /usr/bin/env /usr/lib/jvm/jdk-21.0.8-oracle-x64/bin/java @/tmp/cp_1du5p1mgglfdk2lim5eca1lv1.argfile com.example.muebleria.Mueb
leriaApplication
=====================================================
   BIENVENIDO A MUEBLERÍA FALSA 123. S.A.
=====================================================
INFO: La API REST está corriendo en http://localhost:8080

--- MENÚ PRINCIPAL ---
1. Listar muebles activos
2. Crear nuevo mueble
3. Crear cotización
4. Confirmar venta (por ID de cotización)
5. Gestionar Variaciones
6. Salir
Seleccione una opción: 1

--- Listado de Muebles Activos ---
Hibernate: select m1_0.id,m1_0.estado,m1_0.material,m1_0.nombre,m1_0.precio_base,m1_0.stock,m1_0.tamano,m1_0.tipo from mueble m1_0
where m1_0.estado=?
ID: 1 | Nombre: Silla Clásica | Precio: $50000 | Stock: 18
ID: 2 | Nombre: Mesa de Comedor | Precio: $200000 | Stock: 6
ID: 3 | Nombre: Sillón Relax | Precio: $150000 | Stock: 4
ID: 4 | Nombre: Estante Biblioteca | Precio: $80000 | Stock: 15
ID: 6 | Nombre: Silla de Oficina | Precio: $75000 | Stock: 25
ID: 7 | Nombre: Silla Moderna | Precio: $6000 | Stock: 4
ID: 8 | Nombre: Armario | Precio: $4500 | Stock: 22

(Presione Enter para continuar...)


--- MENÚ PRINCIPAL ---
1. Listar muebles activos
2. Crear nuevo mueble
3. Crear cotización
4. Confirmar venta (por ID de cotización)
5. Gestionar Variaciones
6. Salir
Seleccione una opción: 2

--- Crear Nuevo Mueble ---
Nombre: Armario Pequeño
Tipo (ej. Silla, Mesa): Armario
Precio Base (ej. 50000): 2990
Stock inicial: 11
Hibernate: insert into mueble (estado,material,nombre,precio_base,stock,tamano,tipo) values (?,?,?,?,?,?,?)
¡Mueble creado con éxito! ID: 9

--- MENÚ PRINCIPAL ---
1. Listar muebles activos
2. Crear nuevo mueble
3. Crear cotización
4. Confirmar venta (por ID de cotización)
5. Gestionar Variaciones
6. Salir
```

Listar los muebles y ver el nuevo mueble creado

```
¡Mueble creado con éxito! ID: 9

--- MENÚ PRINCIPAL ---
1. Listar muebles activos
2. Crear nuevo mueble
3. Crear cotización
4. Confirmar venta (por ID de cotización)
5. Gestionar Variaciones
6. Salir
Seleccione una opción: 1

--- Listado de Muebles Activos ---
Hibernate: select m1_0.id,m1_0.estado,m1_0.material,m1_0.nombre,m1_0.precio_base,m1_0.stock,m1_0.tamano,m1_0.tipo from mueble m1_0
where m1_0.estado=?
ID: 1 | Nombre: Silla Clásica | Precio: $50000 | Stock: 18
ID: 2 | Nombre: Mesa de Comedor | Precio: $200000 | Stock: 6
ID: 3 | Nombre: Sillón Relax | Precio: $150000 | Stock: 4
ID: 4 | Nombre: Estante Biblioteca | Precio: $80000 | Stock: 15
ID: 6 | Nombre: Silla de Oficina | Precio: $75000 | Stock: 25
ID: 7 | Nombre: Silla Moderna | Precio: $6000 | Stock: 4
ID: 8 | Nombre: Armario | Precio: $4500 | Stock: 22
ID: 9 | Nombre: Armario Pequeño | Precio: $2990 | Stock: 11

(Presione Enter para continuar...)
```

Cotizar un mueble y agregar una variación (barniz premium)

intentar vender una silla con stock insuficiente

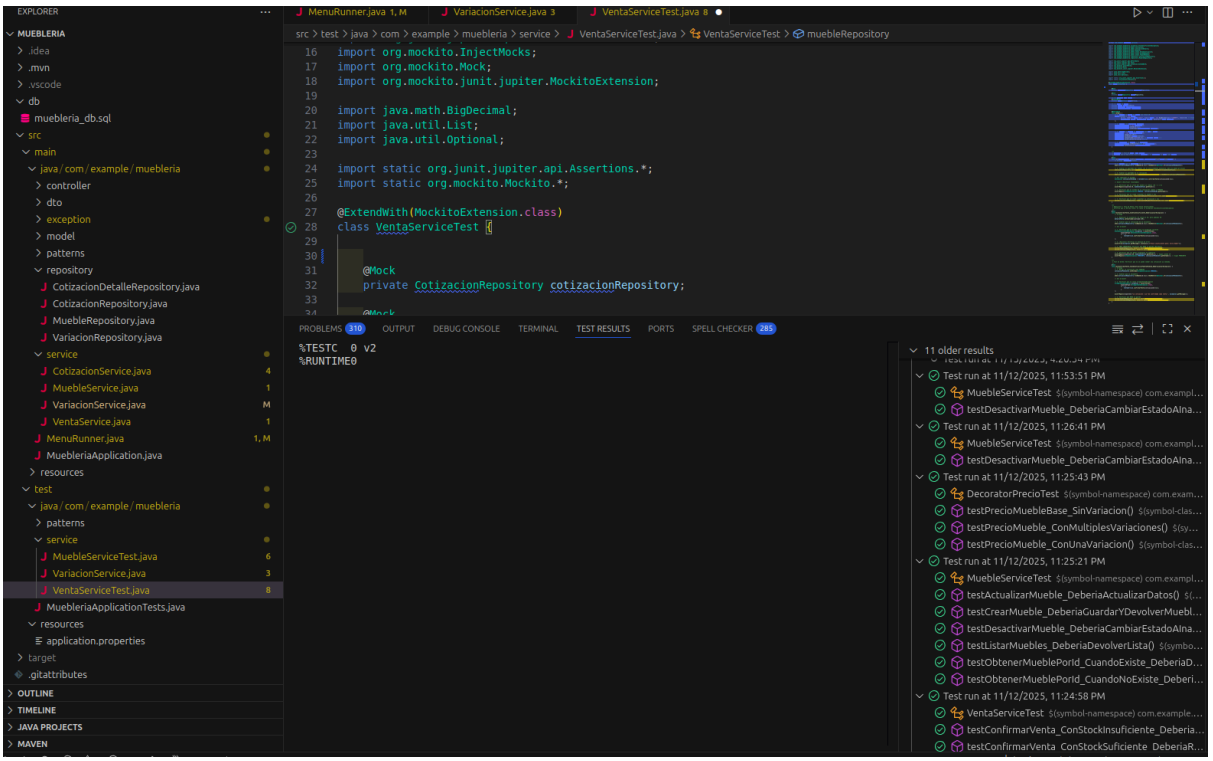


gestionar variaciones

```
INFO: La API REST está corriendo en http://localhost:8080

--- MENÚ PRINCIPAL ---
1. Listar muebles activos
2. Crear nuevo mueble
3. Crear cotización
4. Confirmar venta (por ID de cotización)
5. Gestionar Variaciones
6. Salir
Seleccione una opción: 5

--- Gestión de Variaciones ---
1. Listar todas las variaciones
2. Registrar nueva variación
3. Volver al menú principal
Seleccione una opción: 1
Hibernate: select v1_0.id,v1_0.aumento_precio,v1_0.nombre from variacion v1_0
ID: 1 | Nombre: Barniz Premium | Aumento de Precio: $15000
ID: 2 | Nombre: Cojines de Seda | Aumento de Precio: $25000
ID: 3 | Nombre: Ruedas de Goma | Aumento de Precio: $5000
ID: 4 | Nombre: Madera de Roble | Aumento de Precio: $9000

(Presione Enter para continuar...)


--- Gestión de Variaciones ---
1. Listar todas las variaciones
2. Registrar nueva variación
3. Volver al menú principal
Seleccione una opción: 2

--- Registrar Nueva Variación ---
Nombre (ej. Barniz Premium): Madera de Acasio
Aumento de Precio (ej. 15000): 900
Hibernate: insert into variacion (aumento_precio,nombre) values (?,?)
¡Variación registrada con éxito! ID: 5

--- Gestión de Variaciones ---
1. Listar todas las variaciones
2. Registrar nueva variación
3. Volver al menú principal
Seleccione una opción: 1
Hibernate: select v1_0.id,v1_0.aumento_precio,v1_0.nombre from variacion v1_0
ID: 1 | Nombre: Barniz Premium | Aumento de Precio: $15000
ID: 2 | Nombre: Cojines de Seda | Aumento de Precio: $25000
ID: 3 | Nombre: Ruedas de Goma | Aumento de Precio: $5000
ID: 4 | Nombre: Madera de Roble | Aumento de Precio: $9000
ID: 5 | Nombre: Madera de Acasio  | Aumento de Precio: $900

(Presione Enter para continuar...)
```

Test

Historial general de test ejecutados



Conclusión

El desarrollo de este proyecto cumplió exitosamente con todos los objetivos, entregando una API REST funcional con Spring Boot y MySQL. La lógica de negocio crítica, como la gestión de stock, se resolvió de forma robusta usando @Transactional en el VentaService para garantizar la consistencia de los datos.

Los patrones de diseño fueron claves: el patrón Repositorio (vía Spring Data JPA) abstrajo el acceso a datos, mientras que el patrón Decorador solucionó elegantemente el requisito de precios dinámicos para las variaciones. Finalmente, las pruebas unitarias con JUnit (VentaServiceTest y DecoratorPrecioTest) fueron esenciales para validar la correcta implementación de toda la lógica de negocio y sus casos de error.