UNIVERSIDAD DEL BÍO-BÍO

Facultad de Ciencias Empresariales

Departamento de Ciencias de la Computación y Tecnologías de Información

Ingeniería Civil Informática.

Ingeniería de Software

Evaluación 2

Integrante: Carlos Vásquez Rodríguez

Docente: Gastón Márquez

Roberto Anabalón Suazo

Asignatura: Ingeniería de Software

Carrera: Ingeniería Civil en Informática

Fecha: 13/11/2025

Introducción

Se desarrolló un programa en el contexto de una mueblería cuyo objetivo principal es implementar un backend para la empresa ficticia “Mueblería los muebles hermanos S.A.”. El backend desarrollado permite gestionar un catálogo de muebles y sus variaciones, así como generar cotizaciones y confirmar ventas, actualizando el stock de manera consistente.

A continuación se deja el link del repositorio en el cual se encuentra el proyecto.

Repositorio github: https://github.com/CarlosVsq/Evaluacion2IngSoftware

Dependencias del proyecto

La solución se implementó utilizando las siguientes dependencias de Spring boot:

- **Spring web:** Proporciona los componentes necesarios para crear controladores REST, manejar solicitudes HTTP y levantar un servidor embebido (Tomcat) de forma automática.
- **Spring Data JPA:** Facilita el acceso a datos mediante JPA y Hibernate, permitiendo mapear entidades y usar repositorios para interactuar con la base de datos sin escribir SQL manual.
- **Spring Boot Starter Test:** Incluye herramientas para realizar pruebas unitarias y de integración, como JUnit y Mockito, integradas con el ecosistema de Spring Boot.
- **Spring Boot DevTools:** Proporciona herramientas para acelerar el desarrollo, como reinicios automáticos al modificar código, recarga en caliente y mejoras en el tiempo de compilación, facilitando un ciclo de desarrollo más rápido y eficiente.

Estructura por capas del proyecto

La aplicación se estructura siguiendo una arquitectura clásica en capas, alineadas con el patrón Model-View-Controller y una capa de servicios:

- **Capa de presentación (Controladores)**
    - MuebleController
    - VariacionController
    - CotizacionController

Estos controladores exponen los endpoints necesarios para gestionar el catálogo, las variaciones y cotizaciones/ventas.

- **Capa de acceso a datos (Repositorios)**
    - MuebleRepository
    - VariacionRepository
    - CotizacionRepository

Estas clases se encargan de la persistencia de las entidades mediante Spring Data JPA.

- **Modelo de domino (Entidades JPA)**
- **Objetos de transferencia (DTO)**
- **Capa de servicio**
    - MuebleService

Contiene la lógica de negocio principal relacionada con muebles, variaciones y cotizaciones.

Patrones de diseño implementados

- **Fachada (Facade) en MuebleService**

La clase MuebleService actúa como una fachada hacia la lógica de negocio relacionada con muebles, variaciones y cotizaciones. Desde el punto de vista de los controladores, el servicio ofrece una interfaz unificada y de alto nivel para:

➔ Gestionar el catálogo de muebles (crear, listar, actualizar, desactivar).
➔ Gestionar las variaciones asociadas a un mueble.
➔ Calcular precios finales considerando variaciones.
➔ Crear y confirmar cotizaciones, incluyendo la actualización de stock.

Internamente, la fachada coordina múltiples repositorios (MuebleRepository, VariacionRepository, CotizacionRepository) y encapsula reglas como la creación de una variación básica o la validación de stock suficiente.

- **DTO (Data Transfer Object)**

La clase CotizacionRequest funciona como un Data Transfer Object utilizado para recibir la información necesaria al crear una cotización (por ejemplo, una lista de identificadores de muebles y cantidades). Esto desacopla el contrato de la API REST de las entidades internas de persistencia, facilitando la evolución del modelo sin romper el cliente.

- **Repository**

Las interfaces MuebleRepository, VariacionRepository y CotizacionRepository implementan el patrón Repository, muy habitual en aplicaciones con Spring Data JPA. Este patrón proporciona una abstracción sobre el acceso a datos, de manera que la lógica de negocio no se ve contaminada por detalles de persistencia.

En la práctica, se exponen métodos como:

➔ Búsqueda de muebles activos.
➔ Búsqueda de variaciones por identificador de mueble.
➔ Validaciones específicas (existencia de variaciones, conteos, etc.)

Ejecución del proyecto

Requisitos previos

- Instalar java 17 (o posterior)
- Instalar XAMPP (se puede tener otra aplicación similar que cuente con MySQL)
- Instalar Maven

Base de datos

Primero debemos configurar la base de datos, ya que sin esta no se podrá realizar nada. Se levanta la base de datos mediante XAMPP o alguna otra herramienta que utilice MySQL.

Debemos crear una base de datos llamada “**gestion_mueblería”**, ya que así está en el código. De todas formas se puede modificar en el archivo llamado application.properties, ubicado en la carpeta de src/main/resources del proyecto.

Luego no es necesario crear las tablas, ya que hibernate las generará al ejecutar el código.

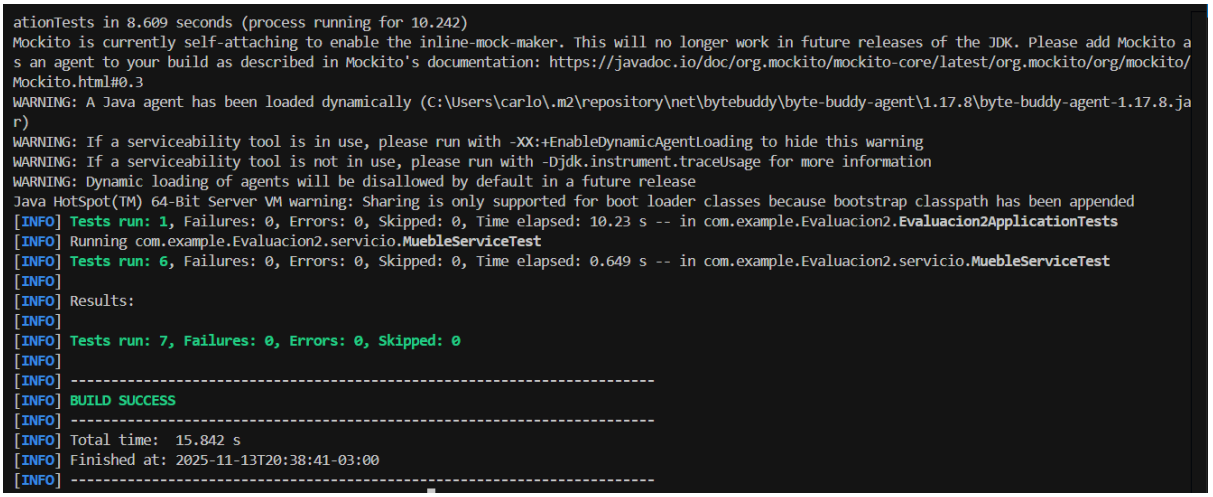
Testing

Para la ejecución de las pruebas unitarias, lo que debemos tener es la base de datos levantada y con el nombre correcto (gestion_muebleria).

Luego en el IDE que estemos utilizando, debemos estar dentro de la carpeta Evaluacion2 y ejecutar el comando “**mvn test”**. Este comando iniciará todos los 6 test que se programaron.

Para ver las pruebas unitarias de forma individual. Tenemos que ir al directorio Evaluacion2/src/test/java/com/example/Evaluacion2/servicio y aquí encontraremos el archivo MuebleServiceTest.java, este se encarga de todos los test.

Con esto deberíamos de ver una salida similar a la que se muestra a continuación.



Postman

Para probar el proyecto en runtime podemos utilizar postman y utilizar los endpoints disponibles.

Primero debemos de tener la base de datos levantada con el nombre correcto, sino al intentar ejecutar el proyecto, este se caerá.

Con la base de datos levantada, ahora debemos compilar y ejecutar el proyecto, esto se puede hacer con Maven o desde el mismo IDE que estemos utilizando, para esto debemos runnear el archivo **Evaluacion2Application.java**.

Ahora con el programa listo, podemos pasar a postman y probar las funcionalidades. A continuación se deja una serie de endpoints disponibles para probar, con su url y json correspondiente si es necesario.

1) **Crea un mueble**

POST http://localhost:8080/api/muebles

{

"nombre": "Silla",

"tipo": "comedor",

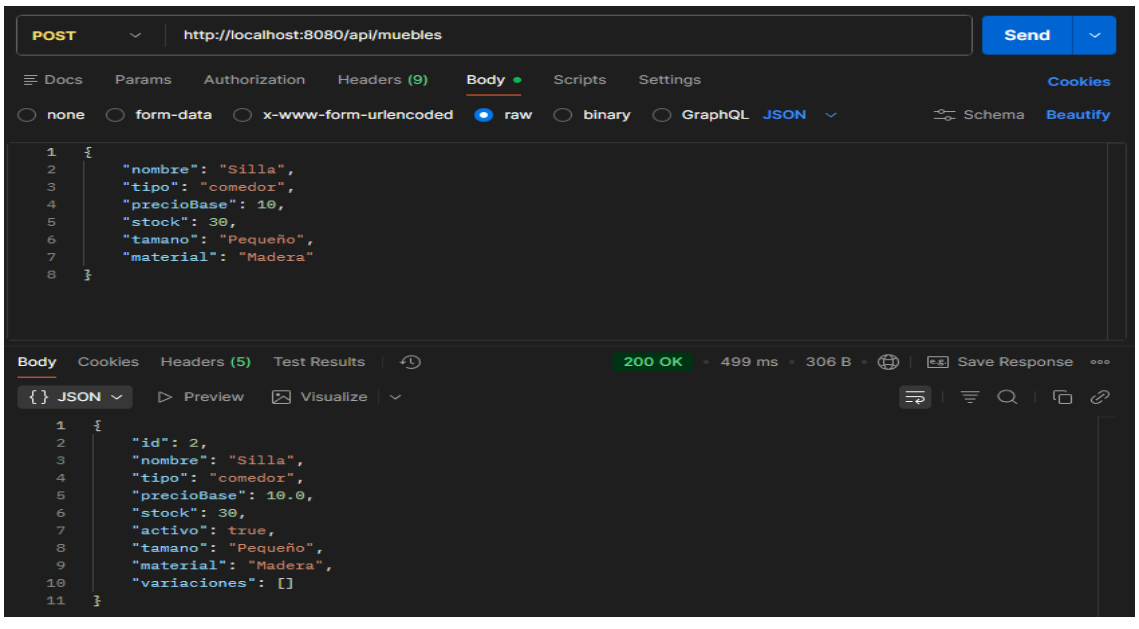
"precioBase": 10,

"stock": 30,

"tamano": "Pequeño",

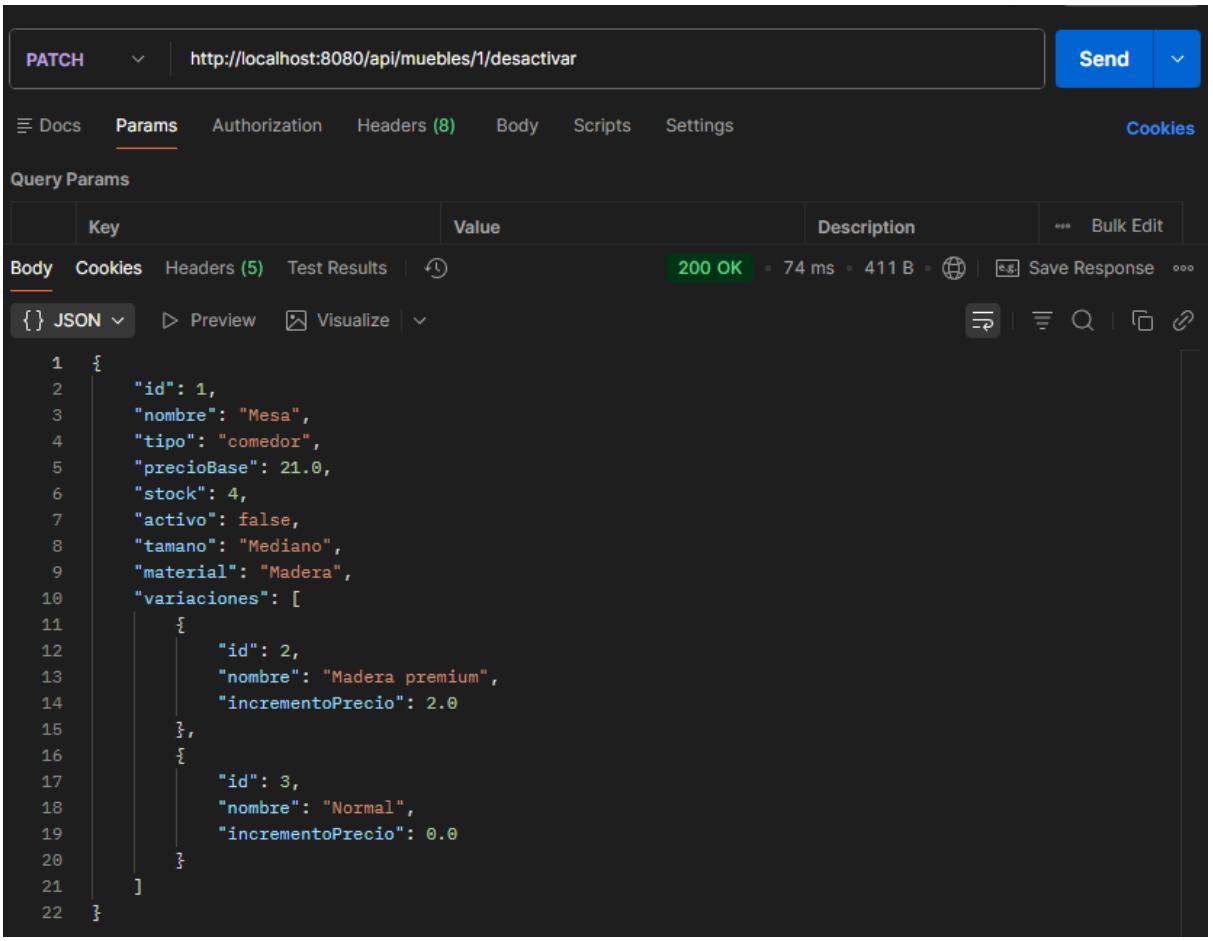
"material": "Madera"

}



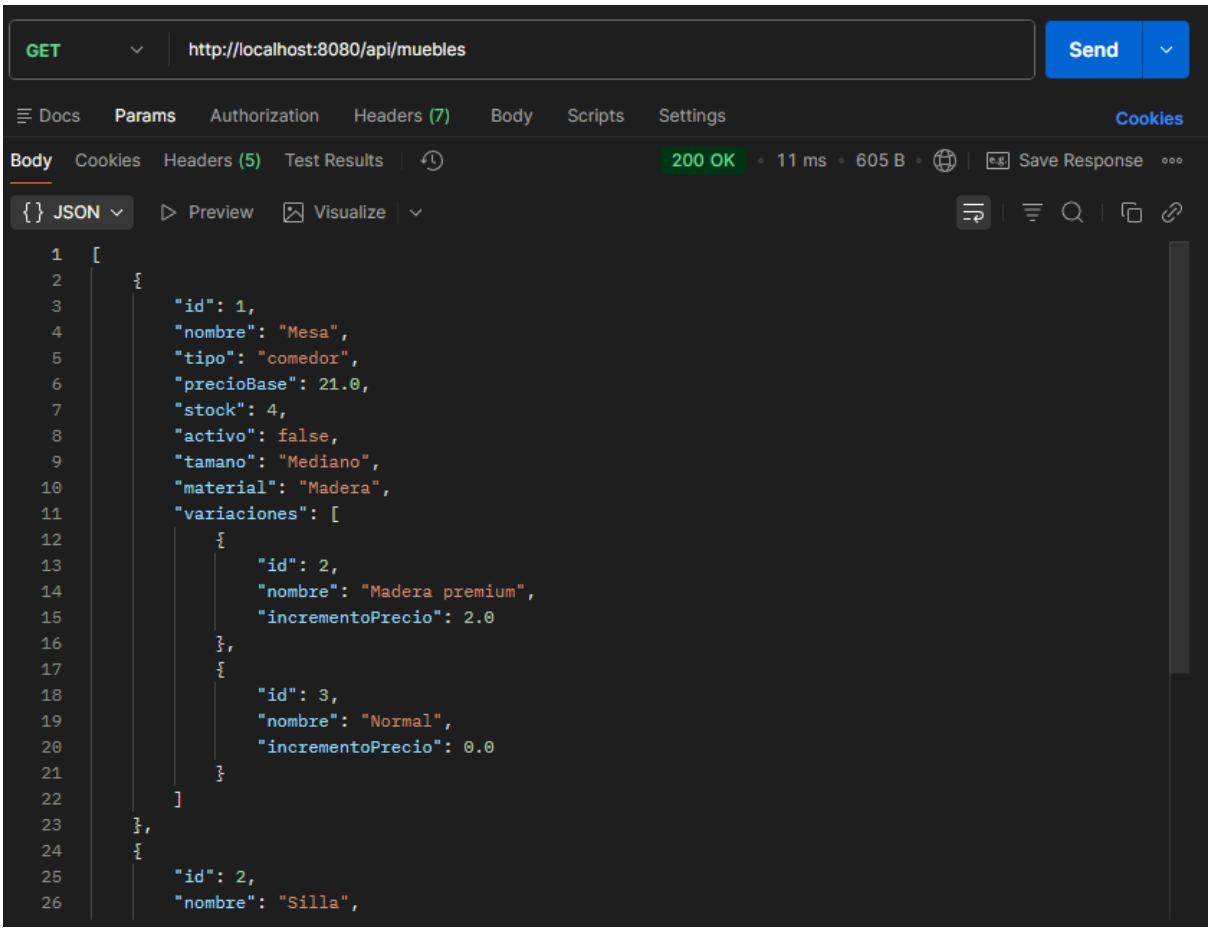
2) Desactivar mueble

PATCH http://localhost:8080/api/muebles/{idMueble}/desactivar



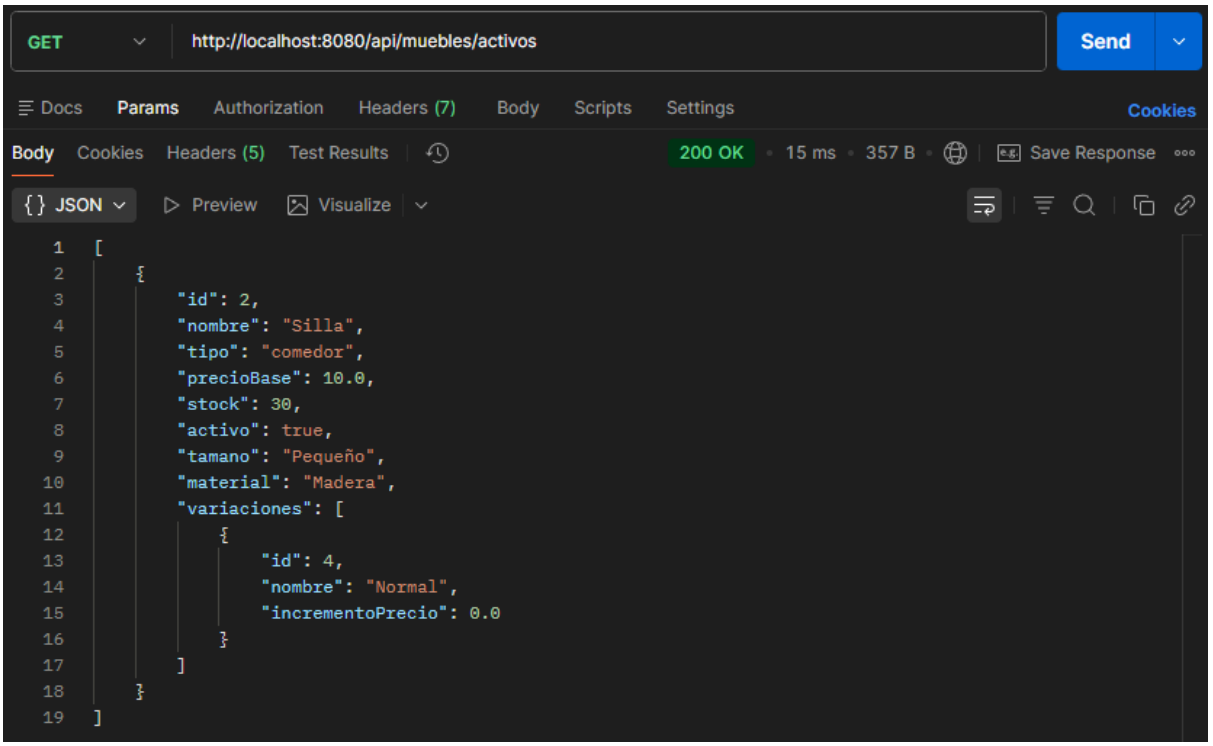
3) Obtener todos los muebles existentes

GET http://localhost:8080/api/muebles



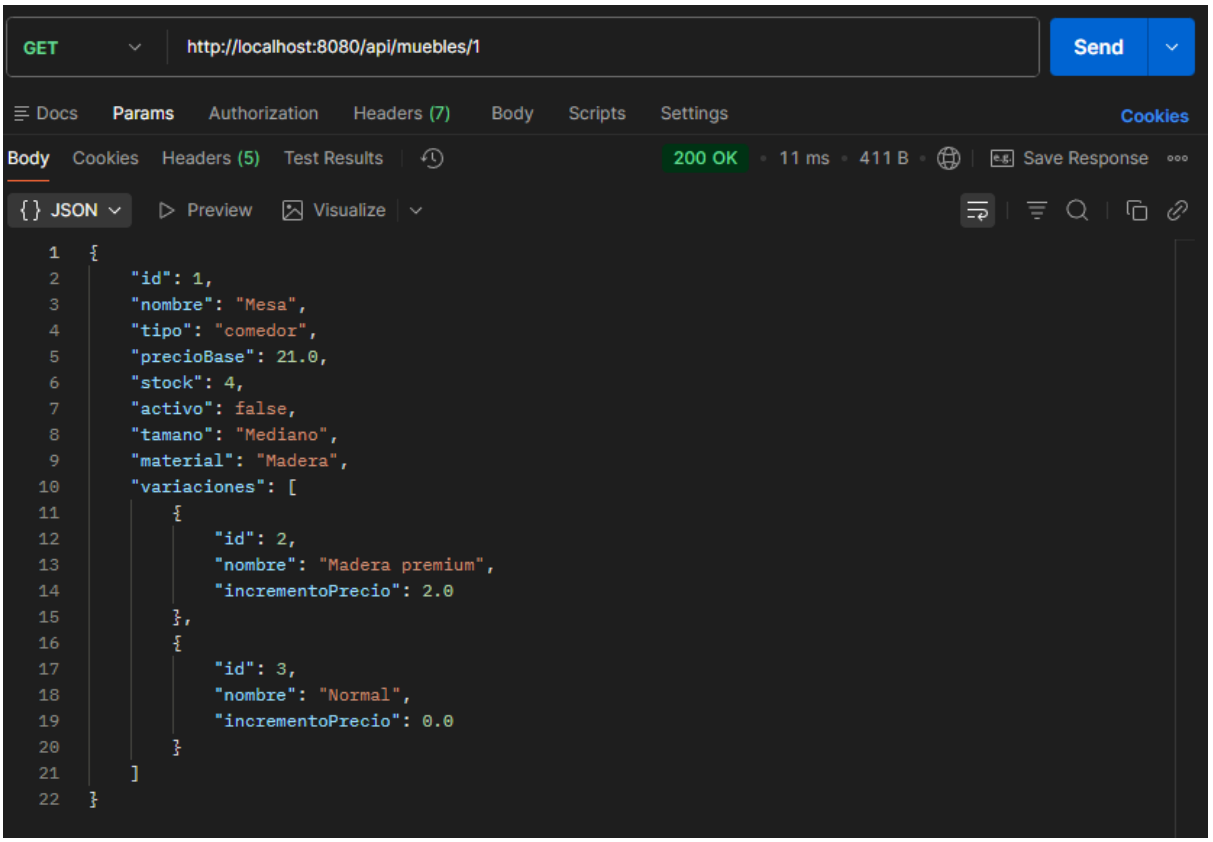
4) Obtener todos los muebles activos

GET http://localhost:8080/api/muebles/activos



5) Obtener un mueble específico

GET http://localhost:8080/api/muebles/{idMueble}

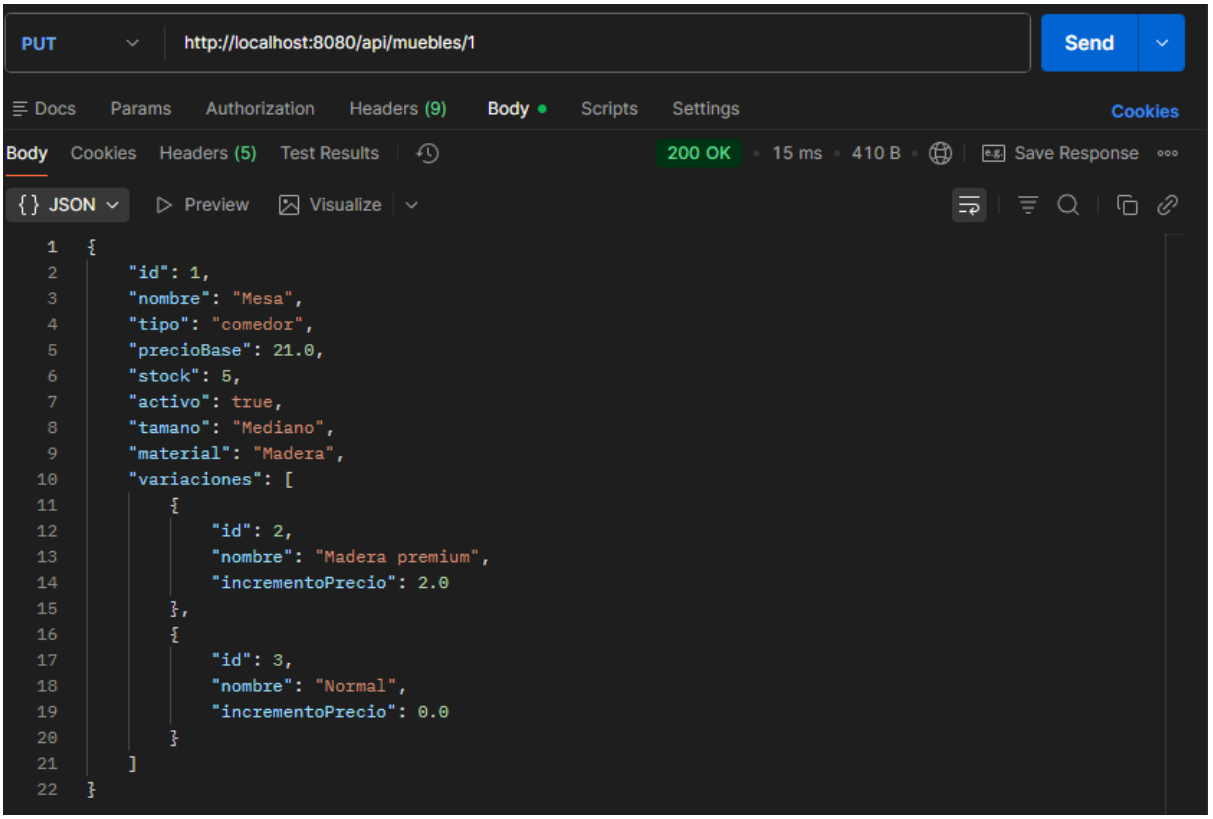


6) Actualizar mueble

PUT http://localhost:8080/api/muebles/1

Json body:

```
{
    "nombre": "Mesa",
    "tipo": "comedor",
    "precioBase": 21,
    "stock": 5,
    "activo" : true,
    "tamano": "Mediano",
    "material": "Madera"
}
```

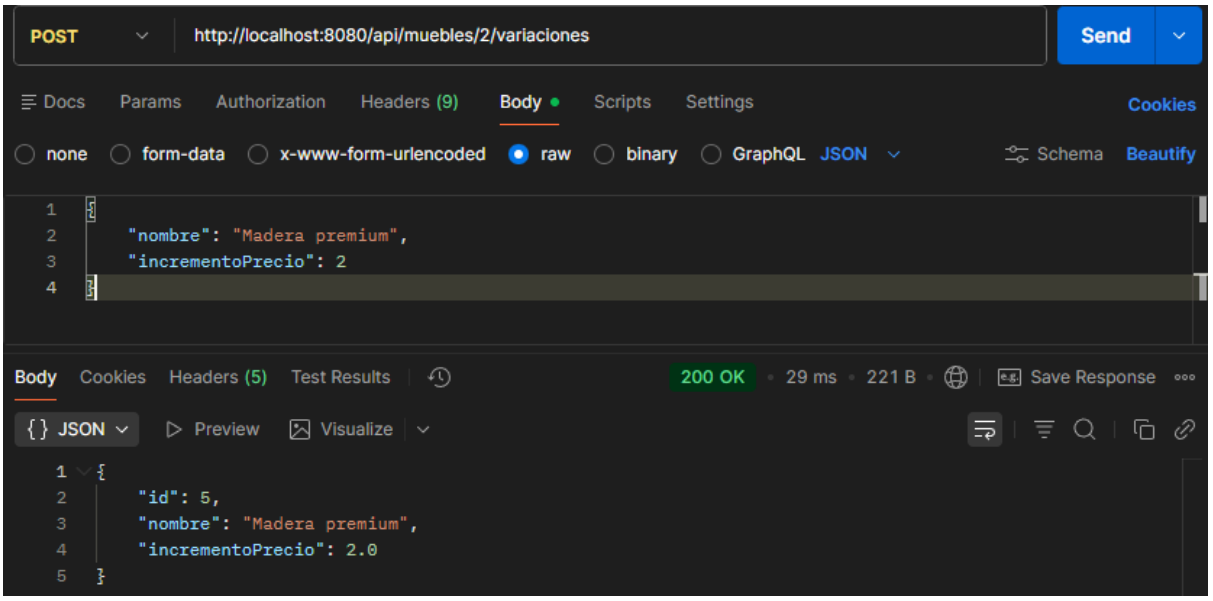


7) Crea variación en un mueble

POST http://localhost:8080/api/muebles/{idMueble}/variaciones

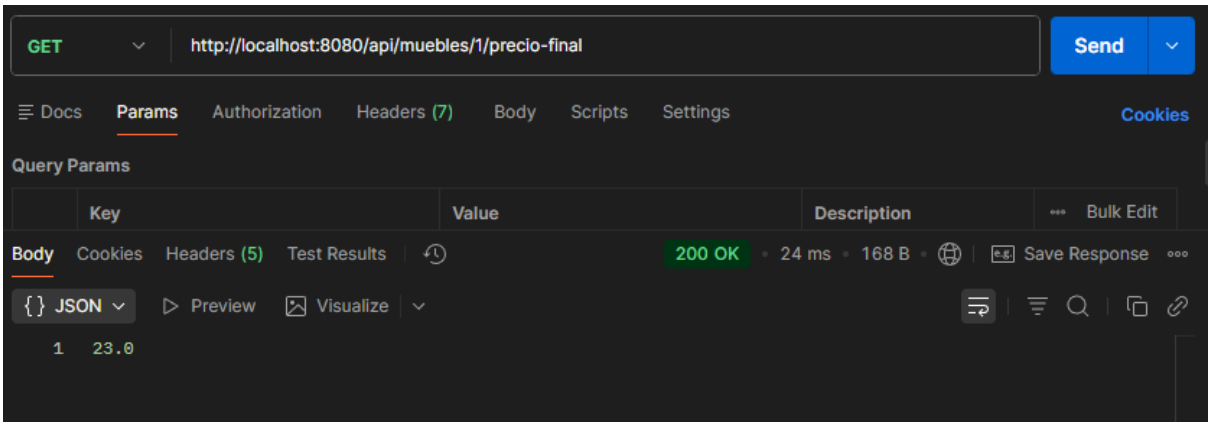
Json body:

```
{
   "nombre": "Madera premium",
   "incrementoPrecio": 2
}
```



8) Obtener precio final

GET http://localhost:8080/api/muebles/{idMueble}/precio-final



9) Crea cotización

POST http://localhost:8080/api/cotizaciones

Json body:

{

"muebleIds": [1,2]

}

nota: debe de ser un array de ids de mueble, es decir que deben estar entre [] aunque sea un solo número, sino dará un error.