Restaurant Management 🍝

Coach - Sherilyn Maestre

El objetivo de este proyecto es desarrollar un API RESTful que permita gestionar las operaciones de un restaurante, cubriendo aspectos clave como la administración de menús, reservas, pedidos, y clientes. El API debe implementar todos los métodos HTTP principales (GET, POST, PUT, DELETE) y cumplir con los principios de diseño RESTful.

Requisitos Generales:

1. **Solución implementada al 100%**: El proyecto debe tener todas las funcionalidades terminadas al 100%.
2. **Buenas prácticas y baja complejidad**: Ningún método puede tener una complejidad mayor a 8 y ninguna clase puede tener una complejidad mayor a 10, del mismo modo, no se deben tener code smells.
3. **Lenguaje y Frameworks:** El API debe implementarse utilizando Java 17, Gradle y Spring Boot.
4. **Base de Datos**: El sistema debe conectarse a una base de datos relacional SQL para almacenar y gestionar la información.
5. **Manejo de Datos**: Se debe utilizar JPA para el manejo de las entidades y consultas a la base de datos.
6. **Patrones de Diseño**: Aplicar mínimo 3 patrones de diseño donde se considere relevante dentro del proyecto (El singleton no puede ser uno de ellos).

Requisitos Específicos:

1. **Recursos del API**:
    - **Clientes**: Crear, listar, actualizar, eliminar y operaciones adicionales que requiera para llevar a cabo la lógica solicitada.

- **Menús**: Crear, listar, actualizar, eliminar y operaciones adicionales que requiera para llevar a cabo la lógica solicitada.
- **Platos**: Crear, listar, actualizar, eliminar y operaciones adicionales que requiera para llevar a cabo la lógica solicitada.
- **Pedidos**: Crear, listar, actualizar, eliminar y operaciones adicionales que requiera para llevar a cabo la lógica solicitada.

2. A parte de las operaciones para administrar las entidades anteriormente especificadas, el dueño del restaurante necesita aplicar unas **reglas de negocio** dentro de la aplicación para darle beneficios a los clientes que más compran y para sacarle más ganancia a los platos más populares.

   Cuando un cliente haya realizado 10 pedidos, debe pasar de ser un usuario común a ser un usuario frecuente (Esto debe guardarse junto a la información del cliente). Adicionalmente, los usuarios frecuentes obtienen 2.38% de descuento sobre todos los pedidos que realice. Para conocer los pedidos que ha realizado una persona **NO SE PUEDE** manejar como una atributo, ese valor se debe obtener de una consulta a la base de datos.

   Del mismo modo, cuando un plato haya sido comprado más de 100 veces, debe pasar de ser un plato común a ser un plato popular (Esto debe guardarse junto a la información del plato). Adicionalmente, para beneficio del restaurante, los platos populares deberán costar un 5.73% más que los platos comunes. Para conocer la cantidad de veces que se ha comprado un plato **NO SE PUEDE** manejar como un atributo, ese valor se debe obtener de una consulta a la base de datos.

   Para la implementación del algoritmo que solucione este problema se recomienda usar el patrón Observer y el patrón Chain of Responsability, sin embargo, queda a su disposición los patrones de diseño que quiera implementar.

3. **Operaciones Permitidas**:
    - **GET**: Recuperar recursos o listas de recursos (e.g., listar menús, obtener detalles de un cliente).
    - **POST**: Crear nuevos recursos (e.g., registrar un cliente, agregar un nuevo plato).
    - **PUT**: Actualizar recursos existentes (e.g., modificar información de un menú o cliente).
    - **DELETE**: Eliminar recursos (e.g., borrar un pedido o una reserva).

NOTA: Ten preparada tu colección en Insomnia o Postman antes del espacio de evaluación.

☆*: .。. o(≧▽≦)o .。.:*☆