PARCIAL 2 Ingeniería de Software 2 – Juan de Dios Rodriguez Perez 2210004

**Caso cotidiano propuesto**: Aplicación de pedidos para una cafetería (**CoffeeApp**)

**CoffeeApp** es una aplicación para una cafetería moderna que permite a los usuarios consultar el menú, personalizar sus bebidas, realizar pedidos y recibir notificaciones cuando su pedido esté listo. El sistema se implementa bajo una arquitectura basada en microservicios, con componentes independientes para menú, pedidos, personalización, notificaciones y usuarios.

Un sistema en el que los usuarios pueden:

* Ver el menú
* Personalizar bebidas
* Hacer pedidos
* Ver el estado del pedido
* Recibir notificaciones cuando el pedido está listo

**Microservicios clave:**

* Servicio de Menú
* Servicio de Pedidos
* Servicio de Personalización
* Servicio de Notificaciones
* Servicio de Usuarios

**Patrones propuestos:**

1. Strategy Pattern (Servicio de Personalización)

¿Por qué? Porque puedes tener diferentes estrategias para preparar una bebida: con leche, sin leche, leche de almendra, más azúcar, sin azúcar, etc. Cada tipo de personalización puede ser una estrategia que se aplica a una bebida base.

2. Observer Pattern (Servicio de Notificaciones)

¿Por qué? Porque los usuarios pueden suscribirse a eventos del sistema (por ejemplo: “tu pedido está listo”) y recibir notificaciones automáticas. Ideal para un patrón publicador-suscriptor.

**Antipatrón propuesto:** God Object

¿Dónde? En el Servicio de Pedidos (mal implementado).

¿Por qué? Porque muchas veces se tiende a meter toda la lógica del pedido (validación, cálculo de precio, notificación, estado, etc.) en una sola clase gigante, lo que lo vuelve incontrolable, difícil de mantener y escalar.

**Arquitectura microservicios**

| **Servicio** | **Puerto** | **Función Principal** |
| --- | --- | --- |
| Menu | 3001 | Muestra las bebidas disponibles |
| Pedido | 3002 | (Pendiente revisar, pero se asume gestiona pedidos) |
| Notificación | 3004 | (Probablemente envía alertas o confirmaciones) |
| Usuario | 3005 | (Gestión de usuarios, no visto aún) |
| Personalización | 4002 | Aplica estrategias de personalización a bebidas |

**Patrón Strategy**

Implementado en el microservicio de personalización (aplicarEstrategias).

Cada tipo de estrategia como leche, azúcar, canela, se puede representar como una estrategia que modifica el comportamiento de la bebida.

Este patrón te permite agregar más estrategias fácilmente sin modificar el código base.

**Patrón Decorator**

Si cada estrategia envuelve a la bebida base para agregar funcionalidad (como en un patrón Decorator clásico), este patrón también se estaría aplicando.

Ejemplo típico: una clase LecheDecorator que extiende el comportamiento de Bebida.

**Antipatrones:**

**God Object**: Si una clase como Cafe hace demasiado (por ejemplo, contiene lógica de precios, estrategias y control de estado).

**Hardcoded Values**: Si defines lógica o decisiones clave con strings directamente en múltiples lugares (if (tipo === 'cafe')).

**Spaghetti Code**: Si el flujo de control y responsabilidad está poco claro.

Uml:

Diagrama

El contenido generado por IA puede ser incorrecto.