ESUMER - Alcaldia de Medellín

Ciudadanos digitales

Informe primera entrega

Programación

Juan Esteban García Galvis

18 de octubre de 2024

Índice

1. Introducción

En el desarrollo de software moderno, la utilización de patrones de diseño y arquitecturas adecuadas es esencial para crear sistemas flexibles, mantenibles y escalables. Este informe describe la implementación de una plataforma de comercio electrónico que incorpora una **arquitectura en capas** y los patrones de diseño **Builder**, **Strategy** y **Facade**. Se justifica la elección de cada patrón y se explica cómo se integran en la arquitectura seleccionada.

2. Arquitectura de Software Utilizada

La plataforma sigue una **arquitectura en capas**, separando las responsabilidades en tres niveles principales:

- **Capa de Presentación**: Incluye las vistas y plantillas con las que interactúa el usuario. Maneja la interfaz gráfica y la comunicación con la capa de lógica de negocio.
- **Capa de Lógica de Negocio**: Contiene los servicios y facades que implementan las reglas de negocio, como el cálculo de precios y la gestión de órdenes.
- **Capa de Acceso a Datos**: Gestionada por los modelos y el ORM de Django, permite interactuar con la base de datos de forma eficiente.

Esta arquitectura facilita la **separación de responsabilidades**, mejora la mantenibilidad y permite escalar el sistema de manera más sencilla.

3. Patrones de Diseño Seleccionados

3.1. Patrón de Creación: Builder

El patrón **Builder** se implementó para la creación de productos dentro de la plataforma. Permite construir objetos complejos de manera incremental, lo que es especialmente útil cuando los productos tienen múltiples atributos opcionales o configuraciones específicas.
Justificación:

- **Flexibilidad en la Construcción**: Separa la construcción de un objeto de su representación, permitiendo crear diferentes tipos de productos utilizando el mismo proceso de construcción.
- **Mantenibilidad**: Facilita la adición de nuevos tipos de productos sin alterar el código existente, siguiendo el principio abierto/cerrado.

Integración en la Arquitectura:
El *Builder* se ubica en la capa de lógica de negocio, específicamente en un módulo dedicado a la construcción de productos. Esto permite que la capa de presentación solicite productos personalizados sin conocer los detalles de su construcción.

3.2. Patrón de Comportamiento: Strategy

El patrón **Strategy** se utilizó para manejar las diferentes políticas de precios según el tipo de usuario (regular, premium o mayorista). Encapsula las estrategias de cálculo de precios en clases separadas y permite seleccionar la estrategia adecuada en tiempo de ejecución.
Justificación:

- **Flexibilidad en el Comportamiento**: Permite cambiar las políticas de precios sin modificar el código del cliente, facilitando la adaptación a nuevas reglas de negocio.
- **Reutilización de Código**: Las estrategias pueden ser reutilizadas en diferentes partes del sistema o extendidas para nuevos tipos de usuarios.

Integración en la Arquitectura:
Las estrategias de precio residen en la capa de lógica de negocio. La capa de presentación interactúa con esta capa para obtener los precios ajustados, sin necesidad de conocer cómo se calculan.

3.3. Patrón Estructural: Facade

El patrón **Facade** se aplicó para simplificar la interacción con el sistema de gestión de órdenes. Proporciona una interfaz unificada para un conjunto de interfaces en un subsistema, facilitando su uso.
Justificación:

- **Simplicidad**: Reduce la complejidad al ocultar las operaciones subyacentes de creación y gestión de órdenes.
- **Aislamiento**: Permite aislar la capa de presentación de los detalles internos de la lógica de negocio, promoviendo un bajo acoplamiento.

Integración en la Arquitectura:
La *Facade* se sitúa entre la capa de presentación y la capa de lógica de negocio. La capa de presentación utiliza la fachada para interactuar con el sistema de órdenes, sin necesidad de manejar múltiples clases o métodos complejos.

4. Justificación de la Elección de Patrones

La selección de estos patrones se basó en las necesidades específicas del sistema:

- **Builder**: Necesario para manejar la creación de productos con múltiples configuraciones, mejorando la flexibilidad y la claridad del código.
- **Strategy**: Fundamental para aplicar dinámicamente diferentes políticas de precios, permitiendo una fácil adaptación a cambios en las reglas de negocio.
- **Facade**: Esencial para simplificar la interacción con subsistemas complejos, reduciendo la carga cognitiva y el acoplamiento entre capas.

Estos patrones encajan perfectamente en la arquitectura en capas, ya que cada uno aborda problemas en distintas capas o en la interacción entre ellas, promoviendo un diseño coherente y bien estructurado.

5. Conclusión

La implementación de la arquitectura en capas junto con los patrones de diseño **Builder**, **Strategy** y **Facade** ha resultado en un sistema robusto, flexible y mantenible. La separación clara de responsabilidades y la aplicación estratégica de patrones permiten que el sistema sea escalable y esté preparado para adaptarse a futuros requerimientos.
Esta experiencia demuestra la importancia de elegir la arquitectura y los patrones adecuados para resolver problemas específicos, mejorando la calidad y eficiencia del desarrollo de software.