Documentación de Diseño de OpenMarket

Introducción

En este documento se documentarán las decisiones de diseño, tácticas y patrones de diseño aplicados en la implementación del proyecto OpenMarket. Se detallarán las diferentes características del sistema y cómo se abordaron desde el punto de vista del diseño.

Decisiones de Diseño

1. **Arquitectura Cliente-Servidor**: Se optó por una arquitectura cliente-servidor para permitir una separación clara de responsabilidades y facilitar el escalado del sistema.
2. **Capas**: Se dividió la aplicación en capas (presentación, lógica de negocio, acceso a datos) para promover la modularidad y facilitar el mantenimiento.
3. **Uso de patrones de diseño**: Se decidió utilizar varios patrones de diseño para abordar problemas específicos y mejorar la estructura del código.

Patrones de Diseño Aplicados

1. **Factory**: Se utilizó el patrón Factory para la creación de objetos de productos en el sistema. Esto permitió encapsular la lógica de creación y abstracción de la implementación concreta.
2. **Singleton**: El patrón Singleton se implementó en el módulo de gestión de stock de productos para garantizar que solo existiera una instancia de dicho módulo en todo el sistema.
3. **Adapter**: Se utilizó el patrón Adapter para integrar el sistema con el sistema de pagos externo. Esto permitió adaptar la interfaz del sistema de pagos a la interfaz esperada por el sistema OpenMarket.
4. **Mediator**: El patrón Mediator se aplicó en la comunicación entre el módulo de gestión de productos y el módulo de compras. Esto facilitó la interacción entre estos dos módulos sin generar acoplamiento excesivo.
5. **Observer**: Se empleó el patrón Observer para permitir la notificación de cambios en el stock de productos a los usuarios interesados. De esta manera, los usuarios podían recibir actualizaciones en tiempo real.
6. **Chain of Responsibility**: El patrón Chain of Responsibility se utilizó en el manejo de búsquedas de productos. Se estableció una cadena de procesamiento en la cual cada componente podía manejar la búsqueda o pasarla al siguiente componente de la cadena.

Tácticas de Diseño Aplicadas

1. **Separación de responsabilidades**: Se asignaron responsabilidades claras a cada componente del sistema para evitar la sobrecarga de funcionalidades en un solo lugar y permitir una mayor modularidad.

2. **Inyección de dependencias**: Se utilizó la técnica de inyección de dependencias para desacoplar los componentes del sistema y facilitar las pruebas unitarias.
3. **Validación de datos:** Se aplicó una estricta validación de datos para garantizar la integridad y la consistencia de la información almacenada en el sistema. Se implementaron mecanismos de validación en diferentes capas del sistema para asegurar que los datos ingresados cumplieran con los requisitos y restricciones establecidos.

Arquitectura del Sistema

El sistema OpenMarket se diseñó utilizando una arquitectura cliente-servidor de tres capas. Cada capa tiene responsabilidades claras y se comunican entre sí para brindar una funcionalidad completa y coherente. A continuación, se describe la arquitectura general del sistema:

1. **Capa de Presentación**: Esta capa se encarga de la interfaz de usuario y la interacción con los usuarios. Aquí se encuentra la lógica de presentación, como la renderización de vistas, la gestión de formularios y la interacción con el usuario.
2. **Capa de Lógica de Negocio**: En esta capa se encuentra la lógica y las reglas del sistema OpenMarket. Aquí se implementan los diferentes roles de usuarios (vendedor, usuario anónimo, usuario registrado) y sus funcionalidades correspondientes, como la gestión de productos, la búsqueda de productos y los procesos de compra y venta.
3. **Capa de Acceso a Datos**: Esta capa se encarga de interactuar con la base de datos para el almacenamiento y la recuperación de información.

Implementación

Durante la implementación del proyecto OpenMarket, se crearon clases y métodos específicos para cada rol y funcionalidad del sistema. Se establecieron interfaces para cada componente y se utilizaron clases concretas que implementaban estas interfaces. Se aplicaron los patrones de diseño mencionados anteriormente para estructurar y organizar el código de manera clara y coherente.

La implementación del proyecto se realizó utilizando un lenguaje de programación orientado a objetos, como Java, y se utilizaron frameworks y bibliotecas adicionales para facilitar el desarrollo y mejorar la eficiencia. Se siguieron las mejores prácticas de desarrollo de software, como el uso de estándares de codificación, pruebas unitarias y revisiones de código para garantizar la calidad del software final.

Conclusiones

La aplicación de patrones de diseño y tácticas de diseño en el proyecto OpenMarket ha permitido mejorar la estructura, la modularidad y la mantenibilidad del sistema. Estas decisiones de diseño han facilitado el desarrollo, las pruebas y las futuras modificaciones del sistema. La arquitectura cliente-servidor de tres capas, junto con la implementación de patrones de diseño, ha permitido crear un sistema flexible, escalable y fácil de mantener.

Además, las tácticas de diseño aplicadas, como la separación de responsabilidades, la inyección de dependencias, han contribuido a mejorar la eficiencia y la calidad del sistema OpenMarket.

En resumen, el diseño y la implementación del proyecto OpenMarket se han realizado teniendo en cuenta las mejores prácticas de desarrollo de software y utilizando patrones de diseño y tácticas de diseño adecuadas para garantizar un sistema de alta calidad y fácil mantenimiento.