


Documento Técnico – Patrones de Diseño

 Sistema de Gestión de Consumos y Arrendamientos Residenciales
Curso: Ingeniería de Software I (2016701)
Universidad Nacional de Colombia – Sede Bogotá
Julio de 2025

1. Introducción

En este documento se explican los patrones de diseño aplicados en el sistema desarrollado. Estos patrones fueron seleccionados con base en su capacidad para mejorar la **modularidad**, **testabilidad** y **mantenibilidad** del código fuente. Su uso también está alineado con los objetivos del curso, fomentando buenas prácticas de ingeniería de software y arquitectura limpia.

El sistema implementa dos patrones clave: **Modelo–Vista–Presentador (MVP)** y **Fachada (Facade)**. 

2. Modelo–Vista–Presentador (MVP)

Definición

MVP es un patrón arquitectónico que separa la interfaz de usuario de la lógica de negocio. Se compone de tres elementos:





- **Modelo:** contiene los datos y reglas del negocio (por ejemplo, **Apartamento**, **Recibo**).
 - **Vista:** muestra la información al usuario y captura eventos (por ejemplo, **ApartamentosView** en PyQt5).
 - **Presentador:** actúa como intermediario, recibe acciones desde la Vista, interactúa con el Modelo o Casos de Uso, y actualiza la Vista según los resultados.
-

Implementación en el proyecto

Cada ventana o formulario de la interfaz (`MainWindow`, `PagosView`, `ReportesView`) tiene un **presentador asociado** (`MainPresenter`, `PagosPresenter`, etc.). La Vista se encarga únicamente de capturar acciones del usuario y mostrar los resultados, sin tener conocimiento de la lógica interna.

Por ejemplo, cuando el usuario desea generar un recibo, la Vista comunica ese evento al Presentador, que coordina los servicios necesarios, obtiene los datos, y finalmente los muestra en pantalla o los exporta como PDF.

✓ Justificación de uso

-  Separa completamente lógica de negocio y lógica de presentación.
 -  Facilita las pruebas unitarias de la lógica (Presentadores y Servicios).
 -  Mejora la organización de carpetas, rutas y responsabilidades.
 -  Permite modificar la interfaz sin tocar la lógica interna del sistema.
-

3. Fachada (Facade)

Definición

El patrón **Facade** proporciona una interfaz unificada y simplificada a un conjunto complejo de subsistemas. Su propósito es reducir el acoplamiento entre las capas externas (como la UI) y la lógica interna, encapsulando la complejidad del sistema.

Implementación en el proyecto

Creamos una clase llamada `SistemaResidencialFacade`, ubicada en la capa de casos de uso (`src/usecases/servicios`). Esta clase agrupa las operaciones más comunes del sistema, tales como:

- Cargar la lista de apartamentos.
- Registrar pagos y consumos.
- Generar reportes y recibos.

Los Presentadores no interactúan directamente con varios servicios, sino que delegan sus operaciones a esta fachada. Así se reduce la complejidad del código en la capa de presentación.

✓ Justificación de uso

- 💡 Simplifica la interacción entre UI y lógica del sistema.
 - 🔒 Oculta detalles de implementación interna.
 - 🧱 Reduce el acoplamiento y mejora la cohesión.
 - 📈 Escalable: se pueden agregar más servicios sin afectar a los presentadores.
-

4. Conclusión

Los patrones MVP y Facade han sido implementados para ofrecer una arquitectura limpia, modular y flexible. Esto no solo facilita el mantenimiento del sistema, sino que también lo prepara para futuras ampliaciones y adaptaciones.

Gracias a estos patrones:

- 👤 Las responsabilidades están claramente divididas.
- 🧪 Se pueden hacer pruebas sin depender de la UI.
- 🛠️ Se pueden modificar funcionalidades sin afectar otras capas.
- 🚀 El sistema está preparado para escalar con calidad.

Estos patrones reflejan una aplicación real de los principios SOLID y demuestran el compromiso del equipo con la calidad técnica del proyecto. 🧠✨