Patrón de diseño

Presentado por:

Juan José Martín Vargas – jmartinv@unal.edu.co

Juan Camilo Posso Portilla – jpossop@unal.edu.co

Esteban Prieto Lugo - eprietol@unal.edu.co

Juan Camilo Vergara Tao – juvergarat@unal.edu.co

Profesor:

Oscar Eduardo Alvarez Rodriguez

oalvarezr@unal.edu.co

Universidad Nacional de Colombia

Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación

2025-1S

Patrón de Diseño: Facade

1. Definición y Propósito del Patrón Facade

El patrón de diseño Facade es un patrón estructural cuyo objetivo principal es proporcionar una interfaz unificada y simplificada para acceder a un conjunto de interfaces o componentes de un subsistema complejo. A través de este patrón, se busca reducir la complejidad que puede presentar la interacción directa con los componentes internos, encapsulando la lógica interna del sistema tras una fachada o punto único de acceso.

Este patrón actúa como una "puerta de entrada" que orquesta y coordina llamadas a diversas partes del sistema sin que el cliente tenga que conocer los detalles de bajo nivel de su funcionamiento. De esta manera, permite que los usuarios o módulos que consumen el sistema trabajen con una API clara, coherente y de alto nivel, mientras se oculta la complejidad operativa subyacente.

Propósitos y beneficios de Facade:

- **Reduce el acoplamiento** entre el cliente (por ejemplo, la interfaz de usuario o el frontend) y el sistema interno. El cliente no necesita conocer cómo están implementadas las funcionalidades internas, solo cómo utilizarlas a través de la fachada.

- **Facilita el uso del sistema** proporcionando una API más limpia, clara y centrada en tareas específicas, lo que mejora la experiencia del desarrollador y la mantenibilidad del código.

- **Encapsula las dependencias y detalles de implementación**, de modo que los cambios en los componentes internos no afecten a los consumidores de la interfaz si la fachada se mantiene estable.

- **Mejora la organización del sistema** y servir como punto de control único para realizar validaciones, registros (logs), manejo de errores o tareas de seguridad antes de permitir que las llamadas lleguen a la lógica de negocio.

- **Permite la reutilización** de componentes internos sin exponerlos directamente a todas las capas del sistema, controlando su acceso de forma centralizada.

2. Implementación en el Proyecto

En el desarrollo de esta aplicación basada en Electron, el patrón de diseño Facade se aplicó para simplificar y controlar el acceso desde la capa de presentación (interfaz gráfica) hacia la lógica de negocio y la base de datos del sistema. Esta implementación se centralizó principalmente en el archivo preload.js, el cual actúa como fachada entre el renderer (frontend) y el main process (backend) de Electron.

Flujo general:

1. El usuario interactúa con la interfaz HTML/JavaScript (por ejemplo, al hacer clic en "Registrar").

2. El código del frontend invoca funciones expuestas en window.electronAPI, como registrarUsuario o loginUsuario.

3. Estas funciones son fachadas expuestas por contextBridge.exposeInMainWorld() en preload.js, las cuales internamente llaman a los canales definidos por ipcRenderer.invoke.

4. El proceso principal (main.js) escucha esos canales, llama a los métodos reales que manejan la lógica de negocio (registrarUsuario, loginUsuario en usuarios.js), y devuelve la respuesta al frontend.

3. Justificación del Uso en el Proyecto

El uso del patrón Facade (Fachada) en este proyecto está plenamente justificado por la necesidad de organizar, simplificar y proteger la comunicación entre la interfaz gráfica y la lógica del sistema, en el contexto de una aplicación construida con Electron.

1. Separación de capas y responsabilidades

El patrón Facade permite desacoplar el frontend (renderer) del backend (main process), facilitando que cada parte del sistema se enfoque en su tarea específica:

- El frontend se encarga únicamente de mostrar la interfaz y recoger datos.

- La lógica de negocio y el acceso a la base de datos quedan encapsulados en funciones especializadas, inaccesibles directamente para el usuario.

Esto mejora la mantenibilidad, ya que los cambios internos no afectan la forma en que el frontend consume las funciones.

2. Simplificación de la interfaz

Gracias al objeto electronAPI definido en preload.js, se expone una **interfaz clara y reducida** hacia el renderer.

3. Seguridad

Uno de los principales desafíos en aplicaciones Electron es evitar que el frontend tenga acceso directo a recursos del sistema, como el sistema de archivos o la base de datos. Con contextBridge y el uso de una fachada controlada, se expone **solo lo necesario**, reduciendo el riesgo de acceso no autorizado o ejecución de código malicioso.

4. Escalabilidad

El patrón Facade facilita el crecimiento del sistema. Si en el futuro se agregan más funciones (como edición de perfil, recuperación de contraseña, o conexión a un servidor externo), basta con:

- Implementarlas en la capa de lógica.
- Añadir una entrada en la fachada (electronAPI) que las exponga al renderer.

Esto permite escalar sin comprometer la organización ni la seguridad del sistema.

5. Facilidad de prueba y depuración

Al centralizar el acceso a funcionalidades del sistema en una fachada única, es más fácil:

- Controlar el flujo de datos.
- Registrar actividades y errores en consola.
- Probar funcionalidades específicas sin necesidad de simular toda la interfaz.