Proyecto - FortiFile

Presentado por:

Andrés Felipe Alarcón Pulido - *analarconp@unal.edu.co*
Juan Daniel Jossa Soliz - *jjossa@unal.edu.co*
Michel Mauricio Castaneda Braga - *micastanedab@unal.edu.co*
Jaime Darley Angulo Tenorio - *jangulot@unal.edu.co*

Profesor:

Oscar Eduardo Alvarez Rodriguez
oalvarezr@unal.edu.co

14 de Junio

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
2025

FortiFile

Descripción General:

FortiFile es una aplicación de escritorio desarrollada en Python con PyQt5 que permite a los usuarios almacenar, organizar y proteger archivos personales o confidenciales. El sistema emplea técnicas modernas de seguridad (como cifrado, hash y autenticación) junto con una base de datos SQLite para el almacenamiento eficiente de información. Además, la aplicación ha sido diseñada con principios de Clean Code y el uso de patrones de diseño orientados a mejorar la escalabilidad, mantenibilidad y robustez del sistema.

Patrones de Diseño Implementados

A continuación, se detallan los patrones de diseño utilizados o propuestos en el sistema, con su respectiva definición, propósito, implementación y justificación en el contexto de FortiFile.

Factory Method

- **Definición:** El patrón Factory Method define una interfaz para crear un objeto, pero permite que las subclases decidan qué clase instanciar.
- **Propósito:** Permite crear objetos sin acoplar el código cliente a clases concretas.
- **Implementación en FortiFile:** Se usa para crear diferentes tipos de vistas (por ejemplo, menú principal, pantalla de cuenta, login) de forma centralizada, manteniendo la flexibilidad del sistema.
- **Justificación:** Este patrón permite desacoplar la lógica de navegación o cambio de vistas en la aplicación, facilitando que nuevos módulos de interfaz puedan ser integrados en el futuro sin alterar el código existente. Además, mejora la legibilidad y simplifica el mantenimiento.

```python
class ViewFactory:
    def create_view(self, view_type):
        if view_type == "menu":
            return MenuWindow()
        elif view_type == "cuenta":
            return AccountWindow()
        elif view_type == "login":
            return LoginWindow()
        else:
            raise ValueError("Vista no válida")
```

Facade

- **Definición:** El patrón Facade proporciona una interfaz unificada y simplificada para un conjunto de interfaces en un subsistema.
- **Propósito:** Reduce la complejidad de uso de un sistema, encapsulando sus componentes internos.
- **Implementación en FortiFile:** Se implementa una clase FileSecurityManager que agrupa operaciones de cifrado, validación de usuarios y almacenamiento seguro.
- **Justificación:** El patrón Facade simplifica el uso de componentes complejos y mejora la experiencia de desarrollo al proporcionar un único punto de entrada para tareas relacionadas. En FortiFile, esto permite que las operaciones de seguridad sean más intuitivas y menos propensas a errores, además de facilitar pruebas unitarias.

```python
class FileSecurityManager:
    def __init__(self, encryption_strategy, db_connection):
        self.encryptor = Encryptor(encryption_strategy)
        self.db = db_connection

    def save_secure_file(self, user_id, file_data):
        encrypted = self.encryptor.encrypt(file_data)
        self.db.save(user_id, encrypted)

    def validate_and_decrypt(self, user_id, input_pwd, file_id):
        if self.db.validate_user(user_id, input_pwd):
            encrypted = self.db.get_file(file_id)
            return self.encryptor.decrypt(encrypted)
        return None
```

Observer (Intención de Uso)

- **Definición:** El patrón Observer permite que múltiples objetos se suscriban y reciban notificaciones automáticas cuando el estado de otro objeto cambia.
- **Propósito:** Desacopla el emisor del evento de sus receptores.
- **Intención en FortiFile:** Se planea usar Observer para el sistema de notificaciones dentro de la interfaz gráfica (por ejemplo, alertar al usuario cuando se detecten archivos no sincronizados o vencimiento de sesión).
- **Justificación:** Aunque aún no se ha implementado, su aplicación mejoraría la reactividad de la interfaz gráfica y el manejo de eventos asincrónicos de forma limpia, especialmente en sistemas de múltiples vistas o múltiples usuarios.

FortiFile

Strategy (intención de uso)

- **Definición:** El patrón Strategy define una familia de algoritmos, los encapsula y los hace intercambiables. El cliente selecciona cuál usar en tiempo de ejecución.
- **Propósito:** Permite modificar el comportamiento de un algoritmo sin alterar su estructura.
- **Implementación en FortiFile:** Se aplica en el manejo de cifrados. La aplicación puede elegir entre algoritmos de cifrado como AES, Fernet o incluso uno personalizado.
- **Justificación:** El uso del patrón Strategy permite cambiar fácilmente la política de cifrado sin reescribir la lógica que utiliza dicho cifrado. Esta flexibilidad es crucial en aplicaciones que evolucionan, ya que pueden requerirse nuevas políticas de seguridad sin afectar otras partes del sistema. También facilita la realización de pruebas con distintos algoritmos.

```python
class EncryptionStrategy:
    def encrypt(self, data):
        raise NotImplementedError

class FernetEncryption(EncryptionStrategy):
    def encrypt(self, data):
        # implementación usando Fernet

class CustomEncryption(EncryptionStrategy):
    def encrypt(self, data):
        # implementación personalizada

class Encryptor:
    def __init__(self, strategy):
        self._strategy = strategy

    def set_strategy(self, strategy):
        self._strategy = strategy

    def encrypt(self, data):
        return self._strategy.encrypt(data)
```

FortiFile

Conclusión

El uso de estos patrones de diseño ha permitido que FortiFile no solo cumpla con sus objetivos funcionales, sino que también mantenga una arquitectura sólida y flexible. Cada patrón fue elegido estratégicamente para resolver un problema particular de diseño y asegurar la escalabilidad del proyecto. Estas decisiones aportan a la calidad del código y facilitan futuras ampliaciones, mantenimiento y pruebas del sistema.