Actividad 1: Análisis y Elección de Principios de Diseño para la Plataforma Inteligente de Gestión de Proyectos

Unax Aller

15/03/2025

La calidad y la capacidad de crecer de una aplicación dependen mucho de si se usan bien los principios de diseño. En este texto se explican los principios S.O.L.I.D., DRY, KISS y YAGNI, y se cuenta cómo se van a usar en la arquitectura de nuestra plataforma inteligente de gestión de proyectos. Esta plataforma quiere mejorar cómo se trabaja, predecir si habrá retrasos y dar consejos para mejorar la productividad, así que es importante que tenga un diseño modular y fuerte.

Principios S.O.L.I.D.

El nombre S.O.L.I.D. junta cinco principios que ayudan a hacer un diseño orientado a objetos que sea limpio, flexible y fácil de mantener.

Single Responsibility Principle (SRP)

Cada clase o módulo debe hacer solo una cosa. En nuestra plataforma, por ejemplo, la clase Proyecto solo se encargará de gestionar datos y operaciones que tengan que ver con proyectos, sin mezclar otras cosas como validación o presentación. Esto hace que el código sea más fácil de cambiar y mejorar.

Open/Closed Principle (OCP)

Los módulos tienen que poder ampliarse pero sin tener que ser modificados. Para esto usaremos patrones como Factory o Strategy, lo que permitirá añadir nuevas funciones sin tocar lo que ya está hecho, asegurando que el sistema siga funcionando bien en el futuro.

Liskov Substitution Principle (LSP)

Las subclases tienen que poder usarse en lugar de sus clases base sin que el sistema se rompa. Esto significa que hay que diseñar bien las jerarquías de clases, por ejemplo, para que cualquier tipo de Usuario o Tarea se pueda utilizar sin que haya problemas.

Interface Segregation Principle (ISP)

Es mejor hacer interfaces pequeñas y específicas en vez de una única interfaz gigante. Así, los clientes de la plataforma (como los módulos de notificaciones o reportes) solo usarán los métodos que realmente necesiten, lo que hace que todo sea más ordenado y no haya confusiones.

Dependency Inversion Principle (DIP)

Los módulos importantes no deben depender de cosas concretas, sino de abstracciones. Usando inyección de dependencias, nuestra arquitectura podrá cambiar cosas como el motor de la base de datos o los servicios de IA sin afectar a la lógica principal del sistema.

DRY (Don’t Repeat Yourself)

El principio DRY dice que hay que evitar repetir código y lógica en el sistema.

Aplicación

Se juntarán las funciones comunes (como validaciones, manejo de errores y conversiones de datos) en módulos compartidos. Esto hace que no haya que escribir lo mismo en muchos sitios, lo que facilita que, si hay que cambiar algo, se haga en un solo lugar y sea más fácil de mantener.

KISS (Keep It Simple, Stupid)

El principio KISS dice que es mejor hacer las cosas simples.

Durante el desarrollo, se evitarán soluciones demasiado complicadas o estructuras innecesarias. Se harán las cosas de forma directa y fácil de entender, lo que ayudará a que el sistema sea más fácil de escalar y de mantener en el futuro.

YAGNI (You Aren't Gonna Need It)

Este principio dice que no hay que hacer funcionalidades hasta que realmente hagan falta.

Se pondrán primero los requisitos básicos para que la plataforma funcione bien, sin meter cosas que quizás nunca se usen y que solo compliquen el código. Se irán añadiendo cosas nuevas según lo que pidan los usuarios y lo que realmente haga falta, para que el desarrollo sea más eficiente y ordenado.

Usar los principios S.O.L.I.D., DRY, KISS y YAGNI ayudará a hacer una arquitectura sólida para la plataforma. Siguiendo estos principios, el sistema será modular, flexible y fácil de mantener, permitiendo que se puedan añadir nuevas funciones sin romper nada. Esto no solo hará que el código sea de mejor calidad, sino que también mejorará los procesos de desarrollo, asegurando que la plataforma sea eficiente y pueda crecer sin problemas.