Análisis y Selección de Principios de Diseños

Para el desarrollo de proyecto voy a seguir los principios de desarrollo S.O.L.I.D., DRY, KISS y YAGNI. Esto es fundamental para asegurarse de que se desarrolle un código limpio, reutilizable, escalable y mantenible.

- **S.O.L.I.D:**

  Este principio promueve la creación de un software robusto y mantenible. Este se basa en las siguientes premisas:
  - **S**ingle Responsibility, se refiere a que cada módulo o clase tendrá una única responsabilidad no se le agregaran tareas extras que no le pertenecen. En el caso del proyecto propuesto por ejemplo un módulo se encargara exclusivamente de la gestión de tareas otro de la IA etc.
  - **O**pen/Cloused, la arquitectura será diseñada para que los complementos estén abiertos a la extensión, pero estos estén cerrados a la modificación para así evitar futuros errores en código que ya funciona. Así en la aplicación a diseñar por ejemplo si tenemos varios tipos de proyectos y queremos agregar otro, se podrá hacer extendiendo una clase base sin modificar el código ya existente.
  - **L**iskov Substitution, este nos indica que las clases derivadas serán sustituibles por sus clases bases sin alterar el comportamiento del sistema. Por ejemplo, si se implementan unos algoritmos de predicción, todos deben cumplir con una interfaz común.
  - **I**nterface Segregation, nos dice que se crearan interfaces específicas para cada funcionalidad, evitando interfaces genéricas que agrupen varias responsabilidades. En el desarrollo del software se implementarán interfaces con solo las funcionalidades necesarias separadas unas de otras y no una general con varias funcionalidades.
  - **D**ependency Inversión, Se aplicará la inyección de dependencias para desacoplar módulos, permitiendo que las clases dependan de abstracciones en lugar de implementaciones concretas. Por ejemplo, la lógica de predicción de retrasos dependerá de una interfaz abstracta para acceder a los datos, permitiendo cambiar esta fuente de datos sin modificar la lógica.

- DRY:

  (Dont Repeat Yourself) Este Principio busca evitar la repetición de código lo que reduce errores y facilita el mantenimiento. En el Proyecto propuesto una implementación de este principio seria:
  La creación de funciones y componentes reutilizables para tareas comunes, como la validación de datos o la generación de informes. Utilización bibliotecas compartidas para funcionalidades repetitivas, como la autenticación de usuarios o el manejo de errores. Por ejemplo, en lugar de implementar la lógica de autenticación en múltiples servicios, se centralizará en un módulo independiente.

- KISS:

  (Keep It Simple, Stupid) Este principio trata sobre mantener el diseño lo mas simple y claro posible. Durante el desarrollo se seguirán las siguientes normas:
  - Evitar la sobreingeniería, priorizando soluciones simples y fáciles de entender.
  - Diseñar módulos con responsabilidades claras y delimitadas.
  - Se utilizarán nombres claros y descriptivos para variables, funciones y clases.
  - Evitar patrones de diseños innecesarios cuando una solución sencilla es suficiente.
  - Se evitarán dependencias innecesarias y se optará por frameworks y herramientas que simplifiquen el desarrollo sin añadir complejidad innecesaria.

- YAGNI:

  (You Aint Gonna Need It) Este principio nos viene a decir que no se incorporen funciones innecesarias en el momento actual permitiendo enfocar los recursos en las funcionalidades esenciales y reducir la deuda técnica. Por lo que seguiremos los siguientes puntos:
  - Se priorizarán las funcionalidades clave, como la gestión de tareas, la predicción de retrasos y las recomendaciones de productividad.
  - No se añadirán características adicionales (como integraciones con herramientas externas) hasta que sean requeridas por los usuarios.

CONCLUSIÓN

La aplicación de los principios S.O.L.I.D., DRY, KISS y YAGNI en el diseño de la arquitectura de la plataforma garantizará un sistema robusto, modular, escalable y fácil de mantener. Estos principios no solo mejorarán la calidad del código, sino que también facilitarán la adaptación a futuros requisitos y cambios en el entorno tecnológico.