Informe Final

Resumen Ejecutivo:

Este informe detalla el proceso de diseño y desarrollo de una plataforma inteligente de gestión de proyectos potenciada por inteligencia artificial.
A lo largo de las actividades realizadas, se han aplicado principios de diseño como S.O.L.I.D., DRY, KISS y YAGNI, junto con patrones de diseño como Factory Method, Adapter y Observer. Estas decisiones técnicas y arquitectónicas han permitido crear una solución modular, escalable y fácil de mantener.
Además, se ha implementado una estrategia de pruebas unitarias para garantizar la calidad del software. También se ha realizado un análisis comparativo con soluciones industriales como Jira y Trello para ver nuestras fortalezas y en que podemos mejorar.

El informe concluye con los aprendizajes obtenidos y las áreas de mejora identificadas.

Justificación de Decisiones Técnicas y Arquitectónicas:

- Principios de Diseños Aplicados:

  1) Principios S.O.L.I.D.

  Los principios S.O.L.I.D. son un conjunto de buenas prácticas que promueven la creación de software robusto y mantenible. A continuación, se describe su aplicación en el proyecto:

  - S (Single Responsibility Principle - SRP):

    Descripción: Cada módulo o clase debe tener una única responsabilidad.
    Aplicación en el Proyecto: En la plataforma, cada módulo tiene una responsabilidad clara. Por ejemplo, el Módulo de Proyectos se encarga exclusivamente de la gestión de proyectos, mientras que el Módulo de Tareas maneja las tareas asociadas a esos proyectos. Esto facilita la comprensión, el mantenimiento y la escalabilidad del código.

- O (Open/Closed Principle - OCP):

  Descripción: Las entidades de software deben estar abiertas para la extensión, pero cerradas para la modificación.
  Aplicación en el Proyecto: La arquitectura se diseñó para que los componentes estén abiertos a la extensión, pero cerrados a la modificación. Por ejemplo, si se desea agregar un nuevo tipo de proyecto, se puede hacer extendiendo una clase base sin modificar el código existente.

- L (Liskov Substitution Principle - LSP):

  Descripción: Las clases derivadas deben ser sustituibles por sus clases base sin alterar el comportamiento del sistema.
  Aplicación en el Proyecto: Las clases derivadas de Project (como SmallProject y BigProject) son sustituibles por su clase base sin alterar el comportamiento del sistema. Esto garantiza que cualquier proyecto, ya sea pequeño o grande, pueda ser manejado de la misma manera.

- I (Interface Segregation Principle - ISP):

  Descripción: Las interfaces deben ser específicas para cada funcionalidad, evitando interfaces genéricas.
  Aplicación en el Proyecto: Se crearon interfaces específicas para cada funcionalidad. Por ejemplo, hay una interfaz para la gestión de usuarios y otra para la generación de informes, en lugar de una única interfaz que agrupe ambas responsabilidades.

- D (Dependency Inversion Principle - DIP):

  Descripción: Los módulos de alto nivel no deben depender de módulos de bajo nivel, sino de abstracciones.
  Aplicación en el Proyecto: La lógica de negocio (módulos de alto nivel) no depende directamente de la implementación de la base de datos (módulos de bajo nivel), sino de abstracciones. Esto permite cambiar la fuente de datos sin modificar la lógica de negocio.

2) Principio DRY (Don’t Repeat Yourself)

Evita la duplicación de código, lo que reduce errores y facilita el mantenimiento.

Aplicación en el Proyecto: Se crearon funciones y componentes reutilizables para tareas comunes, como la validación de datos o la generación de informes. Por ejemplo, en lugar de implementar la lógica de autenticación en múltiples servicios, se centralizó en un módulo independiente.

3) Principio KISS (Keep It Simple, Stupid)

Promueve la simplicidad en el diseño y la implementación.

Aplicación en el Proyecto: Se evitó la sobreingeniería, priorizando soluciones simples y fáciles de entender. Se utilizaron nombres claros y descriptivos para variables, funciones y clases, y se evitaron dependencias innecesarias.

4) Principio YAGNI (You Aren't Gonna Need It)

Descripción: No implementar funcionalidades que no sean necesarias en el momento actual.

Aplicación en el Proyecto: Se priorizaron las funcionalidades clave, como la gestión de tareas, la predicción de retrasos y las recomendaciones de productividad. No se añadieron características adicionales (como integraciones con herramientas externas) hasta que sean requeridas por los usuarios.

- Patrones de Diseño:

1) Factory Method:

El patrón Factory Method define una interfaz para crear objetos, pero permite a las subclases decidir qué clase instanciar. Este patrón promueve la flexibilidad y la extensibilidad al desacoplar la creación de objetos de su uso.

Aplicación en el Proyecto: Se utilizó para crear diferentes tipos de proyectos (SmallProject y BigProject). Esto permite añadir nuevos tipos de proyectos en el futuro sin modificar el código existente, cumpliendo con el principio Open/Closed (OCP).

Ejemplo Práctico:

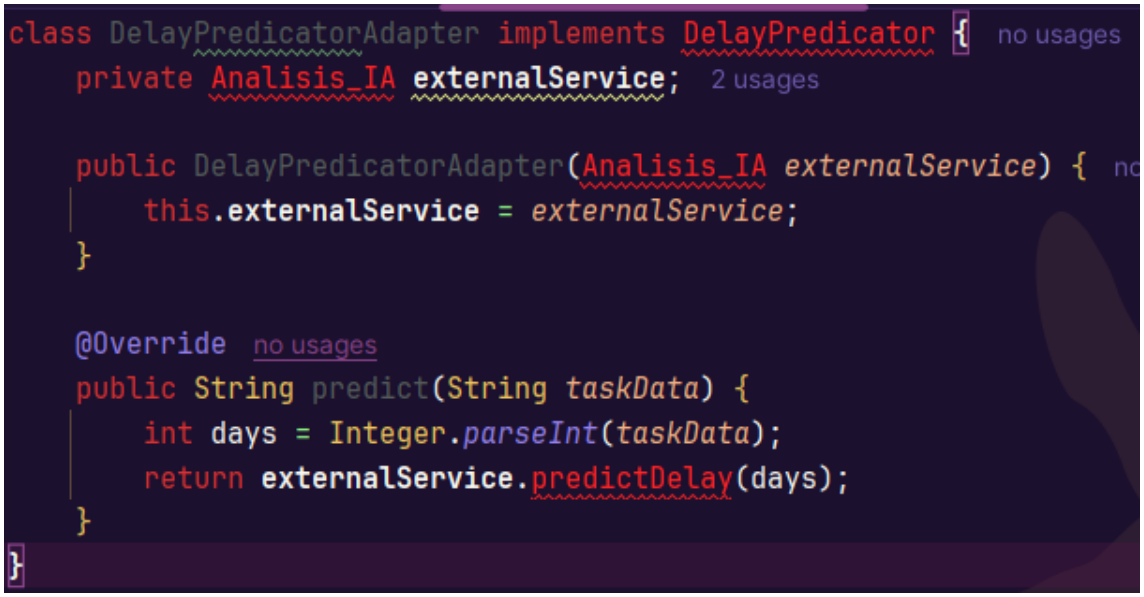
```
class ProjectFactory {  no usages
    public Project createProject(String projectType) {  no usages
        if (projectType.equalsIgnoreCase( anotherString: "small")) {
            return new SmallProject();
        } else if (projectType.equalsIgnoreCase( anotherString: "big")) {
            return new BigProject();
        } else {
            throw new IllegalArgumentException("Tipo de proyecto no válido");
        }
    }
}
```

2) Adapter:

El patrón Adapter permite que clases con interfaces incompatibles trabajen juntas. Actúa como un puente entre dos interfaces, convirtiendo la interfaz de una clase en otra que el cliente espera.

Aplicación en el Proyecto: Se utilizó para integrar un servicio externo de análisis de IA (Analisis_IA) con la plataforma. El adaptador convierte los datos del servicio externo en un formato compatible con nuestro sistema.

Ejemplo Práctico:



3) Observer:

El patrón Observer define una dependencia uno-a-muchos entre objetos, de modo que cuando un objeto cambia de estado, todos sus dependientes son notificados y actualizados automáticamente.

Aplicación en el Proyecto: Se implementó para notificar automáticamente a los usuarios cuando cambia el estado de un proyecto. Esto mejora la colaboración y la productividad del equipo.

Ejemplo Práctico:

```
class Project {  no usages
    private List<Observer> observers = new ArrayList<>();  2 usages
    private String status;  1 usage

    public void setStatus(String status) {  no usages
        this.status = status;
        notifyObservers("El proyecto ahora está " + status);
    }

    public void attach(Observer observer) {  no usages
        observers.add(observer);
    }

    private void notifyObservers(String message) {  1 usage
        for (Observer observer : observers) {
            observer.update(message);
        }
    }
}
```

- Estrategias de Pruebas Unitarias:
    - Definición de Pruebas Unitarias:

      Las pruebas unitarias son pruebas automatizadas que verifican el correcto funcionamiento de unidades individuales de código (por ejemplo, métodos o clases) de forma aislada.

    - Herramientas y Frameworks:

      -JUnit: Framework de pruebas unitarias para Java.
      -Mockito: Biblioteca para crear objetos simulados (mocks) y aislar las unidades de código bajo prueba.
      -Cobertura de Código: Herramientas como JaCoCo para medir la cobertura de las pruebas.

- Estrategia de Implementación:

  -Identificación de Unidades Para Probar: Enfocarse en los módulos críticos del sistema, como la lógica de negocio, la gestión de proyectos y la integración con IA.

  -Creación de Casos de Prueba: Verificar el comportamiento esperado del código con entradas válidas, entradas límite y situaciones excepcionales.

  -Uso de Mocks y Stubs: Utilizar Mockito para simular dependencias externas y probar las unidades de código de forma aislada.

  -Automatización de Pruebas: Integrar las pruebas unitarias en un pipeline de CI (por ejemplo, con Jenkins o GitHub Actions) para ejecutarlas automáticamente en cada cambio de código.

  -Cobertura de Código: Establecer un objetivo mínimo de cobertura de código (por ejemplo, 80%) y monitorearlo con herramientas como JaCoCo.

- Análisis Comparativo:

  En esta sección, se compara la arquitectura diseñada para la plataforma de gestión de proyectos con soluciones reales utilizadas en la industria, como Jira (de Atlassian) y Trello. El objetivo es identificar las fortalezas de nuestra arquitectura, así como las áreas en las que podemos mejorar para alcanzar un nivel competitivo en el mercado.

  1) Comparación con Jira (Atlassian)

  Características Principales de Jira:

  -Modularidad: Jira está diseñado como una plataforma altamente modular, con plugins y extensiones que permiten personalizar su funcionalidad.

  -Escalabilidad: Soporta desde equipos pequeños hasta grandes organizaciones, con integraciones avanzadas con otras herramientas como Confluence y Bitbucket.

  -Gestión de Dependencias: Ofrece funcionalidades avanzadas para gestionar dependencias entre tareas y proyectos.

2) Comparación con Trello

Características Principales de Trello:

-Simplicidad: Trello se centra en la simplicidad y usabilidad, utilizando un enfoque basado en tableros y tarjetas.

-Colaboración en Tiempo Real: Ofrece funcionalidades avanzadas de colaboración en tiempo real, como comentarios y notificaciones instantáneas.

-Integraciones: Se integra fácilmente con otras herramientas populares como Slack, Google Drive y GitHub.

Análisis General

- Fortalezas de Nuestra Arquitectura:

  Integración con IA: La incorporación de inteligencia artificial para predecir retrasos y generar recomendaciones es un diferenciador clave que no ofrecen muchas soluciones existentes.

  Modularidad: Hemos aplicado principios de diseño como S.O.L.I.D. y patrones como Factory Method y Adapter, lo que hace que la arquitectura sea modular y fácil de extender.

  Simplicidad: La arquitectura es simple y fácil de entender, lo que facilita su mantenimiento y adaptación a nuevos requisitos.

- Áreas de Mejora:

  Ecosistema de Plugins: Podríamos desarrollar una arquitectura más abierta que permita la integración de plugins desarrollados por terceros, similar a Jira.

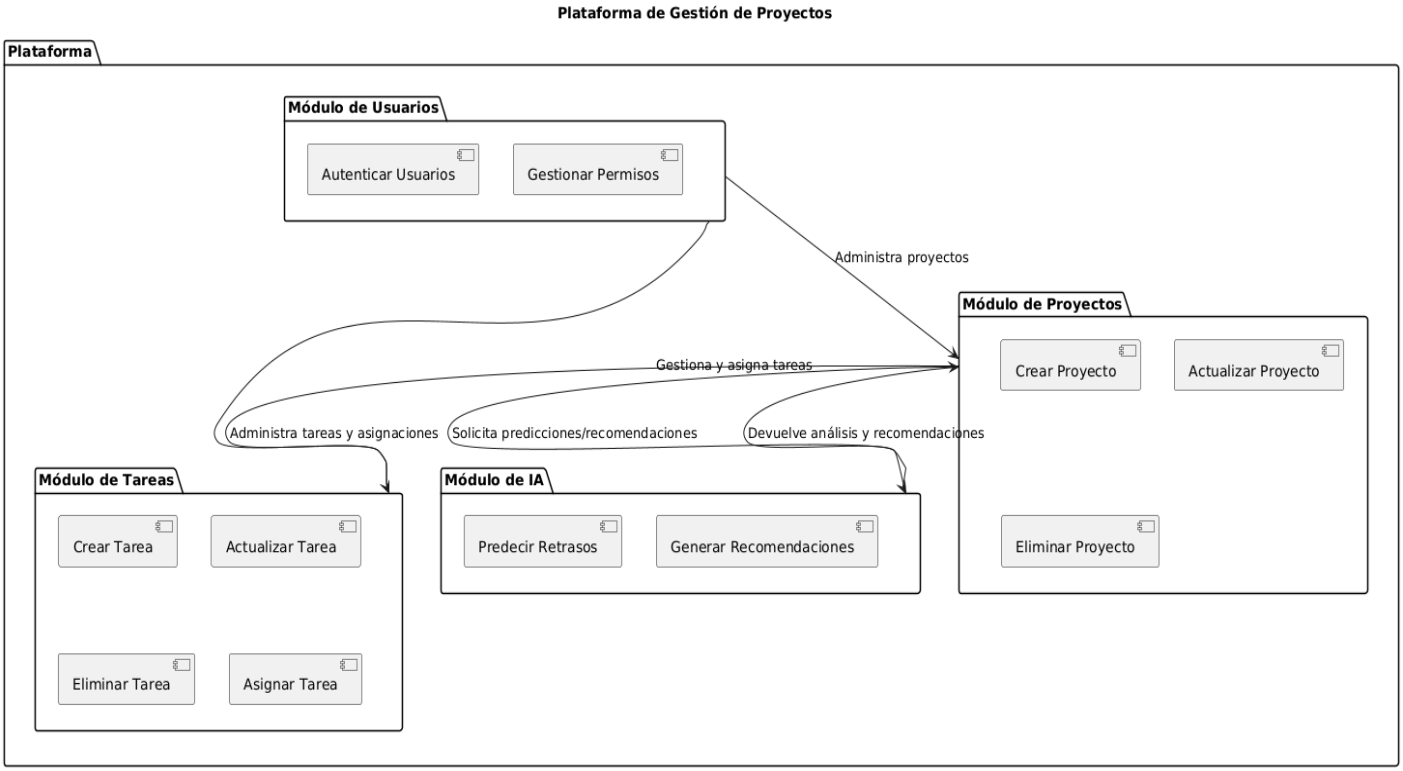
  Escalabilidad: Deberíamos incluir más consideraciones sobre escalabilidad horizontal y gestión de grandes volúmenes de datos para soportar organizaciones más grandes.

  Interfaz de Usuario: Mejorar la interfaz de usuario para hacerla más intuitiva y fácil de usar, siguiendo el ejemplo de Trello.

  Colaboración en Tiempo Real: Añadir más funcionalidades de colaboración en tiempo real, como comentarios y edición simultánea, para mejorar la colaboración entre los miembros del equipo.

Documentación:

- Diagrama:



- Enlace Repositorio:
  https://github.com/MarcosAlonso05/Gestion-de-Proyectos

- Páginas de Interés:
  https://refactoring.guru/design-patterns
  https://blog.ahierro.es/principios-kiss-dry-y-yagni/

Conclusiones:

En conclusión, la aplicación de principios de diseño y patrones de diseño ha permitido crear una plataforma modular, escalable y fácil de mantener. La estrategia de pruebas unitarias ha sido clave para garantizar la calidad del software y reducir la deuda técnica. Por ultimo el análisis comparativo con soluciones industriales ha permitido identificar áreas de mejora, como la escalabilidad y la interfaz de usuario.

Aprendizajes:

Importancia de la Modularidad: Aplicar principios de diseño como S.O.L.I.D. ha sido fundamental para garantizar un código limpio y extensible. Esto nos ha permitido crear una arquitectura que es fácil de mantener y extender.

Utilidad de los Patrones de Diseño: Los patrones de diseño como Factory Method, Adapter y Observer han sido clave para resolver problemas comunes en el desarrollo de software, como la creación de objetos, la integración de servicios externos y la notificación de cambios.

Necesidad de Pruebas Unitarias: Implementar pruebas unitarias y automatizar su ejecución ha sido esencial para garantizar la calidad del código y reducir la deuda técnica. Esto nos ha permitido detectar y corregir errores de manera temprana.

Valor de la Inteligencia Artificial: La integración de IA ha añadido un valor único a nuestra plataforma, permitiendo a los usuarios obtener predicciones y recomendaciones automatizadas. Esto es un diferenciador clave frente a soluciones existentes.