ACTIVIDAD 8: INFORME FINAL INTEGRADO

TÍTULO DEL PROYECTO

Plataforma Inteligente de Gestión de Proyectos Potenciada por IA

NOMBRE:

Mario Blanco Herranz

ASIGNATURA

Técnicas de Programación

FECHA DE ENTREGA

16/03/2025

INTRODUCCIÓN

La presente documentación integra todo el trabajo realizado para diseñar una plataforma que, combinando inteligencia artificial y buenas prácticas de ingeniería de software, permita a los equipos de desarrollo y gestión optimizar sus flujos de trabajo. La necesidad de predicciones de retrasos, recomendaciones inteligentes y flexibilidad en la asignación de tareas hace imperativa una arquitectura sólida y sostenible. Este informe resume las actividades realizadas y los resultados obtenidos a lo largo de todo el proyecto.

RESUMEN EJECUTIVO:

La plataforma inteligente de gestión de proyectos presentada en este informe se concibió con el propósito de mejorar la experiencia y eficiencia de equipos de trabajo de diferentes sectores. Con la capacidad de predecir retrasos y proponer estrategias de asignación de tareas basadas en inteligencia artificial, el sistema pretende elevar los estándares de productividad.

Para alcanzar este objetivo, se estableció una arquitectura en capas, donde cada capa asume responsabilidades claramente delimitadas. La adopción de patrones de diseño como Factory Method, Facade y Strategy resultó esencial para garantizar que las distintas partes del sistema permanezcan bien organizadas y sean fáciles de mantener. El uso de estos patrones, además de cumplir la función de resolver problemas de acoplamiento y flexibilidad, propicia la incorporación ordenada de nuevas funcionalidades.

En el apartado de calidad y pruebas, se contemplan estrategias unitarias y de integración continua para asegurar que las mejoras o correcciones no introduzcan defectos en el sistema. Con ello, se consigue una retroalimentación permanente y se reduce de manera sustancial el riesgo de acumulación de deuda técnica. Al mismo tiempo, se facilita la evolución progresiva del proyecto, haciendo viable escalar su alcance sin sacrificar la estabilidad.

Por último, se realiza una comparación de la propuesta con modelos arquitectónicos ampliamente reconocidos en la industria, como microservicios o arquitecturas limpias, constatando que la solución se sitúa en un punto intermedio que combina la sencillez de la arquitectura por capas con la posibilidad de evolucionar hacia sistemas más distribuidos cuando el contexto de uso lo demande. A lo largo de este informe, se explica en detalle la forma en que cada decisión técnica encamina la plataforma hacia un estado de alta mantenibilidad y competitividad.

1. OBJETIVO Y CONTEXTO

1.1 Objetivo

El objetivo principal fue diseñar la arquitectura de una plataforma que combine la gestión de proyectos con funciones de inteligencia artificial, brindando a los equipos la capacidad de:

Predecir posibles retrasos.

Recibir recomendaciones para optimizar la carga de trabajo.

Revisar informes y estadísticas de desempeño en tiempo real.

1.2 Contexto del Proyecto

Se concibe la plataforma dentro de un escenario donde las metodologías ágiles, tradicionales o mixtas conviven en el día a día de las organizaciones. Para tener un producto con buena acogida en el mercado, se requiere un equilibrio entre versatilidad y mantenibilidad. Desde un inicio, se asumió la importancia de respetar los principios S.O.L.I.D. y de aplicar patrones de diseño que aseguren la escalabilidad del sistema.

Además, se definió un alcance inicial que cubra las funcionalidades básicas (gestión de usuarios y proyectos, asignación de tareas, integración con un módulo de IA para predicciones). Se dejó abierta la posibilidad de añadir a futuro integraciones con otras herramientas de seguimiento y colaboración en línea (como Slack o GitHub), siguiendo el principio YAGNI para no sobrecargar la versión inicial.

2.1 ACTIVIDAD 1: ANÁLISIS Y SELECCIÓN DE PRINCIPIOS DE DISEÑO

La primera labor consistió en desmenuzar los principios S.O.L.I.D., DRY, KISS y YAGNI, estableciendo un mapa de cómo cada uno incidiría en la estructura del software. El Single Responsibility Principle promueve que cada módulo se encargue de una sola tarea, lo que reduce drásticamente la complejidad en la base de código. El principio Open/Closed resguarda la posibilidad de añadir funcionalidades sin necesidad de alterar las clases existentes, un aspecto fundamental cuando se trabaja con algoritmos de IA.

Bajo el Liskov Substitution Principle, la plataforma puede contar con múltiples subclases para los algoritmos de IA, siempre y cuando cumplan el contrato acordado en la superclase. Esto resulta vital para intercambiar métodos ágiles y tradicionales de forma fluida. Con Interface Segregation, los servicios de gestión de tareas y aquellos de reportes se mantienen separados, evitando exponer métodos inútiles a módulos que no los necesitan. Finalmente, el Dependency Inversion Principle fomenta la inyección de dependencias para desacoplar partes críticas de la aplicación.

A nivel macro, DRY (Don’t Repeat Yourself) se adopta para no replicar código en diferentes lugares, centralizando la lógica de la IA en servicios dedicados. KISS (Keep It Simple, Stupid) recuerda la necesidad de no incurrir en complejidades innecesarias, mientras que YAGNI (You Ain’t Gonna Need It) previene la introducción de características adelantadas a su tiempo. Estos principios han guiado la planificación de cada módulo, asegurando un desarrollo coherente y mantenible.

2.2 ACTIVIDAD 2: IDENTIFICACIÓN Y JUSTIFICACIÓN DE PATRONES DE DISEÑO

Como siguiente paso, se seleccionaron tres patrones concretos:

Factory Method: Útil para la creación de objetos relacionados con la lógica de IA. Se evitan constructores de uso manual y se facilita la inclusión de nuevos algoritmos si el proyecto evoluciona.

Facade: Unifica múltiples operaciones internas (creación de tareas, predicciones de retrasos, generación de reportes) bajo una interfaz central. De esta forma, la capa de presentación o cualquier integración externa ve un único punto de entrada, que oculta complejidades.

Strategy: Permite cambiar los métodos de priorización de tareas o incluso la política de asignación de recursos, dependiendo del tipo de proyecto (ágil, tradicional o híbrido) sin reescribir la lógica del servicio.

Estos patrones suponen ventajas claras en la reducción del acoplamiento y la mejora de la organización. Además, sus beneficios se alinean con los principios S.O.L.I.D., pues Factory Method favorece el cumplimiento de DIP, Facade mantiene la interfaz sencilla para los clientes y Strategy garantiza la flexibilidad de LSP y OCP.

3.1 ACTIVIDAD 3: ESQUEMA DE LA ARQUITECTURA DEL SOFTWARE

En esta actividad se desarrolló un diagrama UML que muestra una arquitectura por capas:

Capa de Presentación: Se encarga de recibir peticiones y mostrar resultados a los usuarios o sistemas externos que consuman los servicios.

Capa de Aplicación: Contiene servicios especializados (TaskService, PredictionService, ReportService), así como una clase Facade que centraliza la interacción, asegurando que la capa de presentación no deba conocer los detalles internos.

Capa de Dominio: Constituye el núcleo de la lógica de negocio e IA. Aquí se localizan los algoritmos y el Factory Method que los crea. También se ubican las diferentes estrategias (Strategy) para cambiar la priorización y asignación de tareas.

Capa de Datos: Se ocupa de la persistencia, proveyendo métodos para leer y escribir información relacionada con proyectos, tareas y otros recursos.

Mediante esta estructura, se establece una comunicación ordenada y segmentada. Las dependencias fluyen desde la capa de presentación hacia la de datos, pasando por la aplicación y el dominio, lo que minimiza el acoplamiento y facilita las pruebas, así como la posible migración de un componente a otro entorno o tecnología en el futuro.

3.2. IMPLEMENTACIÓN PRÁCTICA DE PATRONES DE DISEÑO

Para evidenciar la utilidad de los patrones, se desarrollaron ejemplos en Java:

El Factory Method encapsula la creación de algoritmos de IA, diferenciando la instanciación según se trate de un proyecto ágil o tradicional, pero manteniendo una interfaz común.

La Facade provee métodos simplificados para la gestión de tareas y la ejecución de predicciones, unificando la lógica del sistema en un solo punto de acceso.

El Strategy da margen para intercambiar distintos enfoques de priorización, adaptables según las necesidades de cada proyecto.

Estos ejemplos confirman la facilidad con la que se pueden incorporar nuevos algoritmos o nuevas políticas de priorización sin afectar la integridad del resto del sistema. Asimismo, facilitan la sustitución o refactorización de un módulo específico.

4.1: ESTRATEGIA DE CONTROL DE CALIDAD Y PRUEBAS UNITARIA

Para garantizar un producto estable, se diseñó un plan de pruebas basado en:

Pruebas Unitarias: Cada clase o método relevante se evalúa en aislamiento, permitiendo detectar fallos e manera inmediata.

Pruebas de Integración: Verifican la coherencia al unir módulos como TaskService y PredictionService. También validan la comunicación con la capa de datos.

Pruebas de Aceptación: Proponen escenarios reales donde el usuario final interactúa con la interfaz (o con la fachada), confirmando que la plataforma cumpla los requisitos.

Integración Continua (CI): Automatiza la ejecución de las pruebas cada vez que se hace un commit al repositorio. Esto emite alertas tempranas de inconsistencias, manteniendo la calidad de cada versión.

En el marco de esta estrategia, se toman en cuenta herramientas de mocking para simular respuestas de servicios externos o bases de datos. De esta forma, no se depende de la disponibilidad de terceros para comprobar la validez del software. Este enfoque reduce costos y acelera la frecuencia de pruebas, fomentando la práctica de entregas continuas y la detección temprana de bugs.

4.2: PRESENTACIÓN EN VIDEO

Para exponer la arquitectura, las decisiones de diseño y los resultados alcanzados, se elaboró un video de aproximadamente cinco minutos de duración. En él, se explica cómo los principios S.O.L.I.D. y los patrones de diseño se traducen a un código limpio

y manejable. También se enfatiza la forma en que la IA, acoplada mediante un Factory Method, posibilita la escalabilidad de algoritmos de predicción.

Durante la presentación, se exponen ejemplos prácticos de la funcionalidad de la Facade y del cambio de estrategias de priorización, resaltando la adaptabilidad del sistema cuando los requerimientos varían. Este recurso multimedia complementa el presente documento, ayudando a quienes revisen el proyecto a comprender visualmente los aportes y el impacto de cada decisión arquitectónica.

5.1: ANÁLISIS COMPARATIVO CON ARQUITECTURAS INDUSTRIALES

En esta sección, se llevó a cabo un ejercicio de comparación entre la plataforma propuesta y varios enfoques reconocidos:

Arquitectura por capas: La solución encaja en este estilo, con una clara división entre presentación, aplicación, dominio y datos. Esto va acorde con la práctica habitual de grandes organizaciones que prefieren separar la lógica de interfaz, negocio y persistencia.

Microservicios: Si bien el proyecto no se compone de servicios independientes, su modularidad habilita la posibilidad de extraer ciertos componentes a microservicios en un futuro, si el volumen de usuarios o las necesidades de escalabilidad lo aconsejan. De momento, el nivel de complejidad no justifica esa fragmentación.

Arquitectura hexagonal o limpia: El concepto de aislar el dominio y forzar la dependencia de la infraestructura hacia él también se observa en la propuesta. Por ejemplo, la capa de dominio se mantiene libre de referencias directas a frameworks o a detalles de la capa de datos, lo que incrementa la mantenibilidad.

Este análisis permite constatar que el proyecto está en sintonía con modelos reconocidos y bien establecidos, sin comprometer la claridad. Su principal fortaleza radica en la capacidad para adaptarse a metodologías de desarrollo modernas y evolucionar hacia niveles superiores de escalabilidad cuando sea necesario.

CONCLUSIONES Y APRENDIZAJES OBTENIDOS

La elaboración de la plataforma y la aplicación de los principios de diseño han supuesto un aprendizaje significativo en materia de ingeniería de software. Se corroboró que la adopción temprana de principios S.O.L.I.D. y las prácticas de DRY, KISS y YAGNI tienen un impacto directo en la mantenibilidad y evolutividad del código. Gracias a esta base, las modificaciones realizadas no rompen la estructura, y el equipo puede ampliar las funcionalidades de forma controlada.

Otro de los puntos clave es la capacidad de la plataforma para integrarse con futuras herramientas o metodologías. Al emplear patrones como Factory Method, Facade y Strategy, se logra un diseño flexible, en el que cada parte del sistema conserva su independencia. Esto facilita la inserción de nuevos algoritmos de IA, la adaptación de las estrategias de priorización de tareas o la alteración de servicios sin comprometer el resto de la aplicación.

En lo referente a la calidad, la implementación de pruebas unitarias, de integración y de aceptación, junto con la integración continua, brinda un contexto de seguridad para los desarrolladores. De esta forma, se promueve una cultura de refactorización continua, sin temor a colapsar la solución completa. El resultado es una plataforma fiable, con menor deuda técnica y con una lógica de negocio claramente definible.

Por último, el análisis comparativo con arquitecturas industriales revela que la propuesta no se queda atrás frente a modelos más complejos. Con pequeños ajustes, puede evolucionar hacia microservicios o adoptar una visión hexagonal más estricta. En definitiva, se demuestra que una arquitectura robusta, bien concebida desde la raíz, es fundamental para un software que responda a las demandas y retos cambiantes de la industria.

7.1 RESUMEN DE LAS ACTIVIDADES

Actividad 1: Se exploraron los principios S.O.L.I.D., DRY, KISS, YAGNI, y se elaboró un documento exponiendo cómo aplicarlos al proyecto.

Actividad 2: Se seleccionaron tres patrones de diseño (Factory Method, Facade, Strategy) y se justificó su pertinencia para resolver problemas de creación, estructura y comportamiento.

Actividad 3: Se diseñó un diagrama UML para ilustrar la arquitectura del software, mostrando las capas y sus dependencias.

Actividad 4: Se implementaron ejemplos de código en Java para cada patrón, con comentarios que aclaran la utilidad y la forma de integrarlos.

Actividad 5: Se definió una estrategia de control de calidad basada en pruebas unitarias, de integración, aceptación e integración continua, con el fin de reducir la deuda técnica.

Actividad 6: Se grabó un video de presentación en el que se resume el enfoque arquitectónico y se demuestran las principales características de la plataforma.

Actividad 7: Se efectuó un análisis que compara la plataforma con arquitecturas industriales, validando su viabilidad y fortalezas.

Actividad 8: Finalmente, se redactó este informe final que integra los resultados, conclusiones y aprendizajes.

7.2 ENLACES RELEVANTES

Repositorio GitHub:
https://github.com/m4ri0bl/Plataforma_Inteligente_Gestion_Proyectos.git

Video Explicativo: