Diseño de una Plataforma con IA para la Optimización de Proyectos

Iker Arias

Este documento expone de manera integral el proceso de diseño y desarrollo de una plataforma enfocada en la gestión inteligente de proyectos, incorporando algoritmos de inteligencia artificial para anticipar retrasos y brindar recomendaciones a los equipos de trabajo. A lo largo de distintas actividades, se han aplicado principios de ingeniería de software reconocidos y se ha prestado especial atención a la arquitectura, con el fin de obtener un producto estable, escalable y flexible ante futuros cambios.

El informe recoge desde la definición de objetivos hasta las conclusiones finales, pasando por la adopción de patrones de diseño y la implementación de pruebas continuas. Con ello, se demuestra la importancia de establecer buenas prácticas desde el inicio de la concepción de un proyecto, facilitando la mantenibilidad y la evolución de la plataforma en el mediano y largo plazo.

RESUMEN EJECUTIVO La plataforma descrita en estas páginas busca responder a la necesidad, cada vez más frecuente en la industria, de administrar proyectos de forma eficiente y de anticiparse a potenciales conflictos o retrasos. En un entorno donde los equipos pueden trabajar con metodologías ágiles, tradicionales o híbridas, se propuso una arquitectura por capas y la integración de tres patrones de diseño que promueven la cohesión y minimizan el acoplamiento: Factory Method, Facade y Strategy.

Asimismo, el proyecto se cimenta sobre los principios S.O.L.I.D., que permiten desarrollar componentes de software con responsabilidades claras y aisladas. Se han reforzado estos conceptos con otros enfoques clave, como DRY (Don’t Repeat Yourself), KISS (Keep It Simple, Stupid) y YAGNI (You Ain’t Gonna Need It), para evitar la duplicación de código y la creación de funcionalidades innecesarias.

Por otro lado, la adopción de una estrategia de control de calidad compuesta por pruebas unitarias, pruebas de integración y un pipeline de integración continua garantiza que los

cambios se validen de manera oportuna, reduciendo la acumulación de errores e impulsando la calidad a lo largo de todo el ciclo de vida del software. El documento concluye con un análisis comparativo, situando la solución frente a arquitecturas populares en la industria, y exponiendo los beneficios y oportunidades de crecimiento que ofrece.

1. OBJETIVO Y CONTEXTO

1.1 OBJETIVO GENERAL DEL PROYECTO El propósito principal es ofrecer una herramienta digital que, apoyándose en técnicas de inteligencia artificial, ayude a los gestores de proyectos a optimizar la planificación, anticipar retrasos y repartir las tareas de una forma más estratégica. Ello se traduce en una reducción del riesgo de fallos en el cronograma y, en última instancia, en una mejora sustancial de la productividad y la calidad de los entregables.

La plataforma, por tanto, está ideada para cubrir las necesidades de un amplio abanico de sectores que trabajen con proyectos. Aunque en un primer momento se limita a funcionalidades esenciales (gestión de proyectos, asignación de tareas y predicciones básicas), se abre la puerta a ulteriores extensiones, siempre que se mantenga la coherencia con los principios establecidos desde el inicio.

1.2 CONTEXTO DE IMPLEMENTACIÓN En entornos donde conviven diferentes metodologías de desarrollo, la flexibilidad es fundamental. La solución planteada toma en cuenta estas realidades y se organiza en capas, cada una responsable de una parte del sistema: presentación, aplicación, dominio y datos. Esta segmentación permite incorporar nuevas características sin perturbar las secciones ya consolidadas.

Además, se consideró la adopción de principios como SRP (Single Responsibility Principle) para lograr que cada componente se enfoque en un rol específico, minimizando así la complejidad. Conforme el proyecto evolucione, será viable escalar la inteligencia artificial o ampliar las funcionalidades de reportes e integración con otros sistemas, todo ello sin sacrificar la robustez del núcleo.

2. ACTIVIDADES REALIZADAS (ACTIVIDADES 1 Y 2)

2.1 ACTIVIDAD 1: ANÁLISIS Y SELECCIÓN DE PRINCIPIOS DE DISEÑO Para dar a la plataforma una base sólida, se investigaron en detalle los principios S.O.L.I.D., DRY, KISS y YAGNI. Tras analizar ejemplos en diversos contextos, se redactó un documento que demuestra cómo dichos principios se integran en la arquitectura de la plataforma.

- **S.O.L.I.D.**: Provee lineamientos para diseñar clases y módulos con responsabilidades bien definidas, favoreciendo el mantenimiento y la extensibilidad.
- **DRY**: Evita que el mismo código aparezca en distintos puntos del proyecto, promoviendo la creación de funciones y componentes reutilizables.
- **KISS**: Insiste en mantener la solución simple y funcional, sin añadir complicaciones o configuraciones superfluas.
- **YAGNI**: Insta a desarrollar únicamente aquellas características que se requieran en el momento, posponiendo extensiones más avanzadas hasta tener una necesidad demostrada.

El resultado de esta actividad sentó las bases conceptuales para el resto de tareas, asegurando que cualquier decisión posterior se mantuviera alineada con la búsqueda de una arquitectura limpia y fácil de escalar.

2.2 ACTIVIDAD 2: IDENTIFICACIÓN Y JUSTIFICACIÓN DE PATRONES DE DISEÑO
Posteriormente, se escogieron tres patrones de diseño que aportarían soluciones específicas a problemas comunes:

- **Factory Method**: Para crear objetos sin exponer la lógica de construcción, protegiendo al resto del sistema de detalles de implementación y facilitando la introducción de variaciones en los algoritmos.
- **Facade**: Con el objetivo de ofrecer una interfaz simplificada para interactuar con tareas de gestión de proyectos y análisis predictivo, reduciendo el acoplamiento entre la capa de presentación y los servicios internos.
- **Strategy**: Permite configurar distintas reglas de negocio o métodos de priorización sin tener que modificar la clase principal, abriendo la posibilidad de manejar proyectos con metodologías ágiles, tradicionales o mixtas.

El sustento de estos patrones reside en su probada efectividad en entornos de software modular. Cada uno aborda un área de preocupación (creación de objetos, simplicidad de interfaz y variabilidad de comportamiento), reforzando los principios de buena ingeniería establecidos al inicio.

3. ACTIVIDADES REALIZADAS (ACTIVIDADES 3 Y 4)

3.1 ACTIVIDAD 3: ESQUEMA DE LA ARQUITECTURA DEL SOFTWARE Se creó un diagrama UML que evidencia la disposición en capas:

- **Capa de Presentación**: Abarca la interfaz de usuario o la API expuesta a servicios externos.
- **Capa de Aplicación**: Integra los servicios lógicos (TaskService, PredictionService, ReportService) y un punto de entrada unificado a través de Facade.
- **Capa de Dominio**: Contiene las entidades principales (por ejemplo, Project, Task) y los algoritmos de IA, junto con el patrón Factory Method para instanciarlos. A su vez, define la estrategia de priorización a través de Strategy.
- **Capa de Datos**: Maneja la persistencia y recuperación de la información, generalmente mediante un repositorio de base de datos.

Este modelo segmentado permite introducir cambios en un ámbito sin afectar a los demás, fortaleciendo la mantenibilidad y la clara separación de responsabilidades. El uso de interfaces y clases abstractas se hace patente, asegurando que la lógica de negocio esté desacoplada de elementos de infraestructura.

3.2 ACTIVIDAD 4: IMPLEMENTACIÓN PRÁCTICA DE PATRONES DE DISEÑO Para ilustrar el comportamiento real de los patrones, se desarrollaron ejemplos en un lenguaje orientado a objetos, específicamente Java:

- **Factory Method**: Crea diferentes algoritmos de IA de forma centralizada, evitando que la capa de aplicación se vea afectada por la aparición de nuevos métodos de predicción.
- **Facade**: Simplifica el acceso a operaciones de gestión de tareas y reporte de avances, ofreciendo métodos públicos que llaman internamente a los servicios especializados.
- **Strategy**: Permite variar la forma en que se analizan los objetivos y prioridades de un proyecto, sustituyendo fácilmente la implementación concreta sin modificar la lógica que la consume.

Estos ejemplos mostraron que, de surgir nuevas necesidades, se puede alterar o añadir un algoritmo sin modificar las partes del sistema que lo consumen, reflejando la efectividad de los patrones escogidos.

4. ACTIVIDADES REALIZADAS (ACTIVIDADES 5 Y 6)

4.1 ACTIVIDAD 5: ESTRATEGIA DE CONTROL DE CALIDAD Y PRUEBAS UNITARIAS Con el fin de asegurar la fiabilidad del producto, se adoptó un ciclo de validación que abarca:

- **Pruebas Unitarias**: Cada función crítica es evaluada en aislamiento, detectando defectos en etapas tempranas.
- **Pruebas de Integración**: Se verifica la correcta interacción entre servicios, asegurando que tareas, predicciones y reportes colaboren sin errores.
- **Pruebas de Aceptación**: Mediante escenarios de uso, se comprueba que el usuario final perciba un funcionamiento coherente y acorde a los requisitos.
- **Integración Continua (CI)**: Automatiza la ejecución de todas estas pruebas al incorporar cambios en el repositorio, reduciendo la posibilidad de introducir regresiones que pasen desapercibidas.

Asimismo, se optó por el uso de mocks y stubs para simular componentes externos y así evitar complicaciones ligadas a la disponibilidad de servicios de terceros o bases de datos en las pruebas locales. Esta práctica agiliza la detección de anomalías y promueve la cohesión interna, ya que cada módulo depende solo de las interfaces definidas, no de implementaciones particulares.

4.2 ACTIVIDAD 6: PRESENTACIÓN EN VIDEO Para ofrecer una visión más accesible y didáctica de la arquitectura y sus motivaciones, se preparó una presentación en formato de video, con una duración aproximada de cinco minutos. En él, se realiza un recorrido por la estructura en capas, se explican las ventajas de Facade al simplificar el acceso a múltiples servicios, y se demuestra la flexibilidad de Strategy al intercambiar dinámicamente el comportamiento de la plataforma.

El video incluye breves demostraciones del código escrito para cada patrón, destacando cómo la implementación de S.O.L.I.D. elimina dependencias innecesarias y pone en evidencia la facilidad para ampliar o modificar funciones sin arriesgar la estabilidad del proyecto.

5. ACTIVIDAD 7 Y ANÁLISIS COMPARATIVO

5.1 ACTIVIDAD 7: ANÁLISIS COMPARATIVO CON ARQUITECTURAS INDUSTRIALES La comparación con modelos reconocidos en la industria resultó muy útil para dimensionar la pertinencia y solidez de la propuesta:

- **Arquitectura por capas**: La propuesta adoptada coincide con esta tendencia, que es común en grandes corporaciones y proyectos de tamaño medio. Al segmentar la lógica en estratos, se facilita el mantenimiento y la organización.
- **Microservicios**: Aun cuando la plataforma no se basó en microservicios, sí presenta un diseño modular y desacoplado que se podría trasladar a una estructura

distribuida en el futuro, si las exigencias de escalabilidad o disponibilidad así lo requirieran.

- **Arquitectura limpia (hexagonal)**: La idea de separar el núcleo de dominio de los detalles de infraestructura también tiene equivalencias en esta plataforma. Por ejemplo, los algoritmos de IA y las entidades del dominio no dependen de una base de datos o de un framework específico, sino que se encapsulan mediante interfaces.

En definitiva, el análisis refleja una alineación considerable con las prácticas recomendadas en la industria. Se ratifica la conveniencia de la arquitectura elegida y de la adopción de patrones que aseguren la flexibilidad del código.

6. CONCLUSIONES Y APRENDIZAJES OBTENIDOS La experiencia de construcción de esta plataforma demostró cuán relevante es planificar con antelación la estructura de un proyecto de software. Al integrar principios de diseño como S.O.L.I.D. y patrones consolidados en la industria, se ha obtenido un sistema conciso, en el que cada módulo cumple una función clara.

La modularidad adquirida no solo facilita la incorporación de nuevas funcionalidades a futuro, sino que también simplifica la refactorización y la resolución de problemas. Gracias a una arquitectura que separa la lógica de negocio de la infraestructura, es posible adaptar los algoritmos de IA a diferentes metodologías de desarrollo sin reescribir grandes secciones de la aplicación.

En lo que se refiere a la calidad, la adopción de una estrategia de pruebas unitarias, de integración y de aceptación permite detectar incongruencias de manera temprana. Sumado al uso de un entorno de integración continua (CI), se garantiza un flujo constante de validaciones que mitiga la acumulación de errores y reduce costos en fases avanzadas. Dicho enfoque también es un impulso para la cultura de mejora continua en el equipo, fomentando una evolución disciplinada del software.

El análisis comparativo realizado muestra que la plataforma puede encajar sin problemas en entornos corporativos que ya manejan arquitecturas por capas o que, eventualmente, transicionen a modelos de microservicios. La flexibilidad implícita en la solución facilita su despliegue en diversos contextos, siempre atendiendo los lineamientos que hacen de la propuesta un producto escalable y mantenible.

7. DOCUMENTACIÓN DE ACTIVIDADES Y ENLACES

7.1 RESUMEN DE LAS ACTIVIDADES

- **Actividad 1**: Se explicó cómo los principios S.O.L.I.D., DRY, KISS y YAGNI afectan positivamente la claridad y modularidad del software.
- **Actividad 2**: Se fundamentó la elección de los patrones Factory Method, Facade y Strategy, describiendo sus roles en la plataforma.
- **Actividad 3**: Se generó un diagrama UML que detalla la estructura por capas y las dependencias entre módulos.
- **Actividad 4**: Se incluyeron ejemplos concretos de implementación de los patrones, aprovechando la inyección de dependencias y clases base.
- **Actividad 5**: Se definieron las pruebas unitarias y de integración, junto con la configuración de CI, para reforzar la calidad del producto.
- **Actividad 6**: Mediante un video, se ilustró la operación de la plataforma y se demostró el aporte de los patrones al mantenimiento del código.
- **Actividad 7**: Se comparó la solución con arquitecturas como microservicios y hexagonal, hallando similitudes y ventajas de la estructura propuesta.
- **Actividad 8**: Se elabora este informe integrando toda la evidencia generada, con conclusiones y aprendizajes.

7.2 ENLACES RELEVANTES

- **Repositorio GitHub**: https://github.com/IkerArias23/GestionProyectos.git
- **Vídeo del Proyecto: https://drive.google.com/file/d/1mNmPioEkM-t2LWCA94LyczQOeRtVkRHC/view?usp=drive_link**