UNIVERSITAT DE BARCELONA

Proyecto Final de Carrera

GRADO EN INGENIERÍA INFORMÁTICA

Facultad de Matemáticas e Informática
Universidad de Barcelona

Aplicación para la gestión de Presupuestos en el Sector de la remodelación y Análisis del impacto de la Inteligencia Artificial en el desarrollo web

---

Autor: William David Sotto Jaime

Tutor: Xavier Baro Sole

Realizado en: Departamento de Matemáticas e Informática

Barcelona, 20 de Enero de 2025

Resum / Resumen / Summary

Català

Aquest Treball de Final de Grau presenta una aplicació web per a la gestió de pressupostos en el sector de la construcció, desenvolupada amb **FastAPI** i **Next.js**. El projecte segueix la metodologia **SCRUM** i té dos objectius principals: oferir una eina funcional per a la gestió d'obres i analitzar com la **Intel·ligència Artificial (IA)** pot optimitzar el desenvolupament web. L'experiència demostra que la IA accelera tasques repetitives, però encara requereix supervisió humana en decisions complexes i arquitectura tècnica.

Castellano

Este Trabajo de Fin de Grado desarrolla una aplicación web para gestionar presupuestos en el sector de la construcción, utilizando **FastAPI** y **Next.js**. Siguiendo una metodología **SCRUM**, el proyecto combina el desarrollo funcional de una herramienta útil con el análisis del papel de la **Inteligencia Artificial** en el proceso. Los resultados muestran que la IA acelera tareas repetitivas, pero necesita revisión humana para decisiones técnicas complejas.

English

This Final Degree Project presents a web application for budget management in the construction sector, built with **FastAPI** and **Next.js**. Using the **SCRUM** methodology, it combines the development of a functional tool with an analysis of the role of **Artificial Intelligence** in web development. Results show that AI speeds up repetitive tasks, but still requires human oversight for complex decisions and scalable architecture.

Elements Clau / Elementos Clave / Key Elements

Sector:

Català: Gestió de construcció
Español: Gestión de construcción
English: Construction management

Tecnologies: FastAPI (Python), Next.js (React), MySQL, TailwindCSS
Metodologia: SCRUM amb sprints de 2 setmanes / SCRUM con sprints de 2 semanas / SCRUM with 2-week sprints
IA analitzada: ChatGPT-4, GitHub Copilot, DeepSeek Chat, Vercel V0

Índice

1. Introducción

La industria de la construcción, en especial el sector de la remodelación, enfrenta retos importantes en la gestión eficiente de presupuestos. Muchos pequeños empresarios y encargados de obra, como es el caso de mi padre, aún dependen de recibos físicos o archivos dispersos para registrar gastos, lo que produce errores, pérdida de documentos y poca visibilidad financiera.

Paralelamente, la Inteligencia Artificial (IA) está revolucionando el desarrollo web. Herramientas como ChatGPT o GitHub Copilot están empezando a desempeñar un rol en el proceso de programación, automatizando tareas repetitivas, sugiriendo código e incluso generando documentación.

Este Trabajo Final de Grado tiene un objetivo fijo:

1. Analizar el impacto real de la IA en el proceso de desarrollo web, evaluando sus ventajas, limitaciones y su rol futuro en el perfil profesional del desarrollador.

Para ello, desarrollaré una aplicación web funcional que permita digitalizar la gestión de presupuestos en obras de remodelación, empleando tecnologías modernas como FastAPI (backend) y Next.js (frontend).

La propuesta no solo responde a una necesidad concreta vivida en primera persona, sino que también pretende aportar una visión crítica sobre la integración de IA en proyectos reales. El documento está estructurado de la siguiente manera:

- Estado del Arte.
- Metodología y tecnologías utilizadas.
- Desarrollo del proyecto dividido por fases.
- Análisis de resultados técnicos y del uso de IA.
- Conclusiones y líneas futuras.

---

2. Antecedentes y Estado del Arte

2.1 Herramientas actuales en la gestión de presupuestos en construcción

En pequeñas y medianas empresas del sector de la remodelación, la gestión de presupuestos sigue realizándose de forma manual mediante hojas de cálculo (Excel, Google Sheets) o incluso en papel. Existen algunas soluciones comerciales como *Buildertrend*, *CoConstruct* o *PlanGrid*, pero suelen tener un coste elevado o requerimientos técnicos no adaptados a usuarios no expertos.

Esto genera un vacío para soluciones accesibles, multilingües, y pensadas para usabilidad en obras y equipos pequeños.

2.2 Vacío que cubre este TFG

Pese a que muchas publicaciones analizan el potencial de la IA, pocas abordan su implementación práctica en un proyecto completo desde cero y aún menos desde una perspectiva académica y personal.

La importancia de este TFG se refleja en su aportación:

- Una implementación real que documenta cómo se usó la IA en cada fase.
- Reflexiones personales sobre cuándo la IA fue útil y cuándo resultó un obstáculo.
- Una contribución pedagógica a otros estudiantes que dudan sobre el futuro de la profesión frente a la IA.

---

3. Estado del Arte de la I.A.

Las IA han transformado el desarrollo web, ofreciendo herramientas que optimizan la productividad, reducen errores y facilitan la creación de aplicaciones más avanzadas. A continuación, presentaré las principales opciones disponibles para Frontend y Backend.

3.1. Inteligencias Artificiales para Frontend (NextJS)

En el desarrollo Frontend, la IA se centra en mejorar la experiencia del usuario, optimizar interfaces y generar código de forma más eficiente.

Modelos y Herramientas de IA útiles

IA / Herramienta	Descripción	Aplicación en NextJS
GitHub Copilot	Basado en OpenAI Codex, sugiere código en tiempo real.	Autocompleta código HTML, TypeScript, CSS con Tailwind y optimiza lógica de componentes.
ChatGPT	Genera contenido dinámico, respuestas automáticas, ayuda en formularios.	Integración en chatbots, asistentes virtuales, formularios inteligentes.
DeepSeek	No es de pago. Análisis de código con IA para procesar necesidades e integración limpia.	Ayuda a prevenir bugs en NextJS, especialmente en TypeScript.
Vercel (v0)	De pago. Generador de código a partir de Promps.	Puede generar una aplicación así como mejorar una plantilla. Es TypeScript y mantiene siempre una librería limpia.
Claude	Alternativa a Copilot, IA de autocompletado de código, puede generar una web con buen estilo.	Mejora productividad escribiendo TypeScript más rápido, tiene muchos problemas de importaciones innecesarias.

3.2. Inteligencias Artificiales para Backend (FastAPI con Python)

En el desarrollo Backend, las IA ayudan en la optimización de bases de datos, generación de APIs, seguridad y eficiencia del código.

Modelos y Herramientas de IA útiles en FastAPI

IA / Herramienta	Descripción	Aplicación en FastAPI
CodiumAI	Sugiere código en Python y FastAPI. IA para autogenerar pruebas unitarias.	Genera rutas, validaciones y consultas SQL en FastAP. Genera pruebas de endpoints en FastAPI.
AI-powered Postman	IA en Postman para pruebas de API.	Crea automáticamente pruebas de endpoints FastAPI.
Copilot	Puede integrar modelos de API's muy eficientemente.	Chatbot y sistemas de recomendación con IA. Una descripción es suficiente para completar código excesivo.

3.3. Elección de Tecnologías

Frontend: Angular 19

Introduce mejoras significativas, como el @defer para la carga diferida de componentes y la posibilidad de hidratación a selección del usuario, lo que mejora el rendimiento en aplicaciones de gran escala. Además, el enrutamiento ha sido simplificado, siguiendo conceptos similares a React.

Durante la investigación sobre IA aplicable a Angular 19, pude ver que muchas herramientas de IA solo ofrecen soporte hasta Angular 17 o 18, lo que evidencia que las IA suelen ir un paso por detrás en la adaptación a nuevas versiones tecnológicas, mientras que a mi me tomo unas 6 o 8 horas completar un curso para entender las nuevas aplicaciones de Angular 19 y lo que habían cambiado de la versión anterior, las IA's que investigué aún no conocían tecnologías que fueran más allá de su año de lanzamiento.

Sin embargo, ChatGPT y GitHub Copilot demostraron ser capaces de seguir el ritmo de los cambios y ofrecer una pequeña asistencia en Angular 19.

Frontend: NextJS

Aún así, no era suficiente para el plazo de tiempo, así que decidí elegir a React, que en cambio de Angular, utiliza un solo documento para sus componentes, permitiendo que la IA pueda devolver una aplicación completa en un solo archivo. Hay bastantes herramientas de IA adapatadas a React, lo que motivó a su elección para este proyecto.

Vercel (v0) era capaz de adaptar todo un nuevo código generado con diversas carpetas. Además, mantiene relación con Github, lo cual permite previsualizar todos los resultados pedidos por Promps.

Sin un plan de pago, es limitado y no almacena tanta información pero por el proyecto me pareció muy factible.

Backend: FastAPI

Para la implementación del backend, me decidí por FastAPI, un framework moderno de Python altamente eficiente para la creación de APIs. Busqué una herramienta de IA capaz de proporcionar soporte en pruebas unitarias y generación de código para FastAPI, lo que llevó a la selección de Copilot de Github.

Chat Copilot ofrece integración con ChatGPT-4, lo cual permite adquirir un único plan de suscripción y obtener soporte de IA tanto para generación de pruebas como para refactorización de código en el backend, mejorando la productividad y reduciendo errores en el desarrollo.

---

4. Objetivos

4.1 Objetivos General:

El objetivo es claro, este proyecto no busca desarrollar una aplicación web para la gestión de presupuestos en la construcción sino analizar la intersección de la IA con el desarrollo web.

4.2 Objetivos Específicos:

- Diseñar y desarrollar una aplicación con Python (Backend) y NextJS (Frontend).
- Implementaré la aplicación bajo la metodología SCRUM para garantizar un desarrollo ágil, gracias a los sprints definidos en ella, puedo tener un control más específico sobre el tiempo estimado y real (Burndown Chart).
- Investigar y documentar el uso de herramientas de IA en el desarrollo web.
- Analizar el impacto de la IA en la profesión de los desarrolladores web.
- Elaborar estadísticas sobre el uso de IA en el proceso de desarrollo.

---

5. Planificación

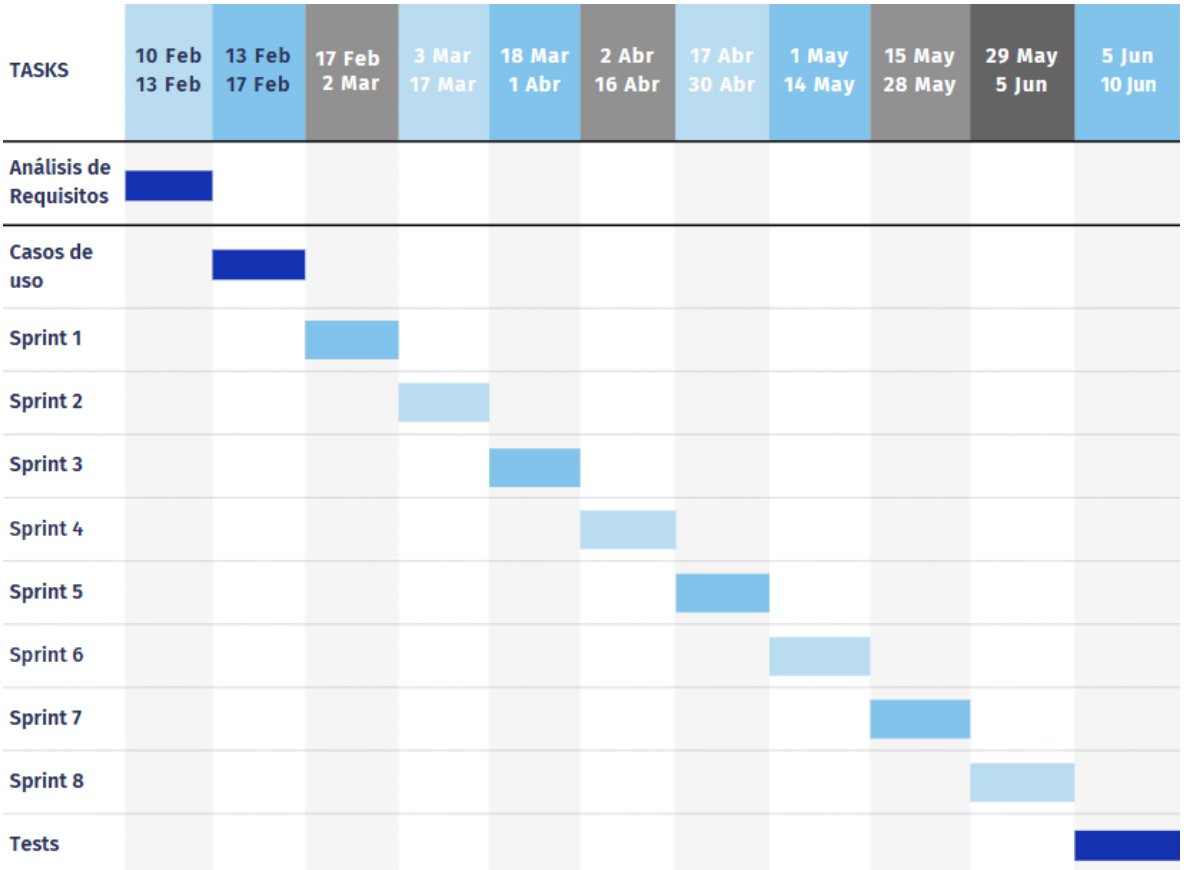
5.1 Enfoque Metodológico: SCRUM Adaptado

Para garantizar una evolución ordenada del proyecto, se utilizó la metodología ágil SCRUM. Esta metodología, comúnmente aplicada en entornos empresariales, fue adaptada a un contexto académico con sprints de dos semanas. Cada sprint tuvo entregables definidos, priorizando funcionalidades clave al inicio del desarrollo.

Adaptaciones para el contexto académico:

- Reducción del equipo a una sola persona (yo), con simulación de roles (Scrum Master, Product Owner).
- En lugar de Daily Meetings, se realizaron revisiones semanales de progreso y obstáculos.

Diagrama de planificación Gantt:



5.2 Papel de la I.A.

Durante el proyecto la inteligencia artificial ha desempeñado diferentes roles evolutivos en las distintas fases del desarrollo del proyecto. Su uso no solo se limitó a tareas puntuales, sino que fue creciendo en complejidad y utilidad conforme avanzaba el desarrollo.

5.2.1 Asistencia en ideación y planificación

Durante las primeras etapas del proyecto, herramientas como **ChatGPT** se utilizaron principalmente para generar ideas, definir estructuras iniciales de código y explorar enfoques alternativos. Por ejemplo, se formularon prompts para obtener recomendaciones sobre arquitecturas frontend modernas.

5.2.2 Generación de código y componentes

A medida que el desarrollo avanzó, especialmente durante los sprints, la IA pasó a tener un rol más activo. Fue utilizada para generar código completo, como funciones optimizadas en Python para el backend (FastAPI) o componentes React/Next.js reutilizables en el frontend.

Ejemplo: se generó un componente de tabla dinámica en Next.js utilizando prompts que incluían requisitos de ordenamiento, paginación y diseño responsive.

5.2.3 Comparación y validación entre modelos

Otra práctica usada fue la comparación entre distintas herramientas de IA (Copilot, DeepSeek, etc). Se les formulaban los mismos prompts para analizar cuál ofrecía mejores soluciones. En algunos casos, se combinaban los resultados de ambas para obtener una versión más robusta y eficiente.

Ejemplo: solicité la creación de una función para validación de formularios. Copilot ofreció una solución más ajustada al estándar, mientras que DeepSeek fue más detallado en los mensajes de error. Se fusionaron ambos enfoques.

5.2.4 Asistencia técnica y resolución de errores

La IA también actuó como un sustituto de herramientas tradicionales como Stack Overflow. Ante errores complejos o comportamientos inesperados, se recurría a la IA para entender mejor el problema y obtener posibles soluciones o caminos de depuración.

Ejemplo Caso real: Un error en FastAPI devolvía 422 Unprocessable Entity al recibir fechas.

Prompt diagnóstico para DeepSeek:

> "¿Por qué FastAPI falla al parsear este JSON con fechas en formato DD/MM/YYYY?"

Solución IA: Modificar el modelo Pydantic para usar datetime.date y timezone.
Tiempo ahorrado: 3 horas de debugging manual.

6. Costes del Proyecto

El desarrollo de esta aplicación web, centrada en la gestión de presupuestos para obras, implicó una inversión personal durante el Trabajo de Fin de Grado, así como una planificación técnica que, en un entorno empresarial, tendría costes adicionales importantes. En este apartado se desglosan ambos escenarios:

6.1 Costo real asumido durante el TFG

En la fase inicial del proyecto, utilicé herramientas de Inteligencia Artificial de forma gratuita, como ChatGPT en su versión estándar. Sin embargo, pronto noté que las limitaciones de los prompts y la capacidad de respuesta no me permitían aprovechar plenamente el potencial de la IA para tareas clave como:

- Redacción y validación de requisitos.
- Generación de ideas funcionales.
- Análisis temprano de accesibilidad y usabilidad.

Por ello, decidí contratar la versión ChatGPT Plus (20 €/mes), que utilicé durante el primer mes de desarrollo. Fue especialmente útil durante la fase de planificación y diseño, permitiéndome avanzar más rápidamente hacia la etapa de programación. Al llegar a la parte técnica del desarrollo frontend, la utilidad de ChatGPT disminuyó, especialmente al trabajar con Angular, donde las respuestas eran menos precisas. Fue entonces cuando decidí cambiar el stack a React y comenzar a trabajar con Vercel V0, una plataforma también basada en IA, gratuita en su versión básica. Opté por migrar mi suscripción a Vercel V0, que también ofrece una modalidad de pago de 20 €/mes.

Herramienta	Tipo de suscripción	Meses	Costo mensual	Total
ChatGPT Plus	Pago	1 mes	20 €	20 €
Vercel V0 Plus	Pago	4 meses	20 €	80 €
Subtotal IA		5 meses		**100 €**

Estas herramientas permitieron acelerar el diseño funcional, validar ideas con mayor rapidez y prototipar interfaces durante el desarrollo inicial.

Infraestructura y despliegue

Plataforma	Tipo de suscripción	Meses	Costo mensual	Total
Railway	Lite	1 mes	5 €	5 €

Durante las pruebas de usuario y el deploy final, fue necesario ampliar la capacidad de la API alojada en Railway mediante un plan de pago ya que el uso gratuito era inviable.

Total invertido (real) : 105 € en herramientas y servicios técnicos asumidos como estudiante.

6.2 Simulación de costes empresariales

Si este proyecto se desarrollara como una aplicación real en una empresa del sector de reformas y presupuestos, los costes se ampliarían considerablemente, considerando los siguientes aspectos:

Recursos humanos estimados

Perfil profesional	Dedicación estimada	Tarifa promedio	Coste estimado
Desarrollador Full Stack (React, FastAPI)	4 meses (media jornada)	1.800 €/mes	7.200 €
Diseñador UI/UX	1 mes (prototipado y testing)	1.500 €/mes	1.500 €
QA Tester / Analista funcional	1 mes (fase final)	1.500 €/mes	1.500 €
Total Recursos Humanos			**10.200 €**

Infraestructura y servicios

Recurso o licencia	Coste estimado mensual	Meses	Total
Railway (plan de producción)	10 €/mes	4	40 €
Vercel (plan Pro)	20 €/mes	4	80 €
Herramientas IA (Vercel V0 + GPT Plus)	40 €/mes (combinado)	4	160 €
Gestión de dominios y correo	12 €/mes	4	48 €
Hosting de base de datos externa	15 €/mes	4	60 €
Total Servicios Técnicos			**388 €**

Otros costes estimables

Categoría	Concepto	Estimación
Soporte legal y protección de datos	Asesoría puntual sobre cumplimiento RGPD	500 €
Diseño gráfico (iconos, branding)	Freelance o compra de recursos visuales	200 €
Testing con usuarios reales	Incentivos / organización de pruebas	300 €
Total Otros Costes		**1.000 €**

6.3 Resumen de costes simulados (entorno real)

Categoría	Total estimado
Recursos humanos	10.200 €
Infraestructura y herramientas	388 €
Otros costes	1.000 €
Total aproximado	**11.588 €**

6.4 Conclusión sobre costes

El proyecto desarrollado como TFG supuso una inversión personal razonable, asumida voluntariamente para facilitar el aprendizaje y simular un entorno de trabajo profesional. Sin embargo, si se desarrollara como un producto real en una empresa, el coste total podría superar los **11.000 euros**, considerando personal cualificado, despliegue, protección legal y testing con usuarios reales.

La comparación permite valorar la **viabilidad del proyecto**, así como reflexionar sobre la inversión que implica llevar una idea funcional hasta una solución preparada para el mercado.

---

7. Análisis

7.1 Requisitos de la Aplicación

Para definir los requisitos de la aplicación, elaboré y distribuí un **cuestionario dirigido a usuarios potenciales** involucrados en el sector de la construcción, incluyendo perfiles como administradores de obra y trabajadores. El objetivo era identificar de forma directa las **necesidades reales** y los **problemas frecuentes** en la gestión de presupuestos y tareas en proyectos de remodelación.

A partir de las respuestas obtenidas, extraje tanto los **requisitos funcionales** (relacionados con las funcionalidades del sistema) como los **no funcionales** (enfocados en la experiencia de uso, accesibilidad y rendimiento). Además, utilicé herramientas de **Inteligencia Artificial (ChatGPT)** para revisar y refinar las preguntas del cuestionario, detectar ambigüedades y generar propuestas adicionales basadas en escenarios reales.

7.1.1 Requisitos Funcionales

- Autenticación JWT con tres roles: Administrador, Trabajador y Cliente.
- Gestión de proyectos: nombre, ubicación, imágenes, cliente asociado, presupuesto.
- Gestión de tareas: asignación por trabajador, fechas, estados y vista en calendario.
- Registro de materiales por proyecto, con cálculo de coste total.
- Dashboard financiero con filtros por fechas y categorías.
- Soporte multilingüe (español, catalán, inglés).

7.1.2 Requisitos No Funcionales

- Accesibilidad visual: soporte para daltónicos, tamaños de fuente adaptables.
- Rendimiento optimizado para uso en móvil (conexiones lentas).
- Seguridad: middleware por roles, control de endpoints según permisos.
- Escalabilidad: arquitectura modular y base de datos relacional adaptable.

Nota: *Se documentan todas las preguntas y respuestas completas en el Anexo A: Cuestionario.*

7.2 Casos de Uso

Al obtener los requisitos elaboraré los casos de usos intentando seguir el principio de “mobile-first”:

Sprint	ID	Rol	Funcionalidad	Criterios de Aceptación
1	US-1	Usuario	Registrarse e iniciar sesión	Formulario validado en frontend + JWT en respuesta
	US-2	Usuario	Acceder al perfil adaptado a su rol	Vista dinámica según rol (Admin / Cliente / Trabajador)
2	US-3	Admin	Crear proyectos (precio, ubicación, descripción)	Formulario con campos obligatorios + asociación automática al cliente
	US-4	Usuario	Ver dashboard según rol	Admin: todos los proyectos / Cliente: solo proyectos asignados
3	US-5	Admin	Asignar tareas a trabajadores	Drag & drop en Kanban + notificación al trabajador
	US-6	Admin	Registrar materiales y gastos	Cálculo automático del coste total
4	US-7	Admin	Editar y eliminar proyectos (soft delete)	Confirmación modal + registro en log de actividades
	US-8	Usuario	Visualización adaptada a roles	Middleware de permisos para proteger vistas
5	US-9	Admin	Crear tareas asignables a trabajadores	Validación de fechas y disponibilidad de trabajadores
	US-10	Trabajador	Visualizar tareas asignadas	Filtros por estado (To-Do / Done)
	US-11	Admin	Registrar materiales en proyectos	Relación automática con presupuesto del proyecto + alertas de gasto
	US-12	Cliente	Consultar inventario de materiales y monitorear las tareas	Datos en tiempo real con umbrales de gasto y gráficos interactivos (Chart.js)
	US-13	Cliente	Monitorear progreso de tareas	Gráficos interactivos (Chart.js) con estados y tooltips
6	US-14	Usuario	Seleccionar idioma (es/en/ca)	Persistencia en cookies + traducción inmediata de la UI
	US-15	Admin	Controlar inventario y costos por proyecto	Exportación a CSV/PDF con resumen por proyecto
7	US-16	Cliente	Ver notificaciones de sus proyectos	Consulta rápida (≤100ms) con mensajes por rol y proyecto
	US-17	Admin	Configurar tipos de notificaciones visibles	Panel con toggles por categoría (tareas, gastos, inventario, etc.)
8	US-18	Admin	Ver dashboard financiero con distribución de gastos	Gráfico de Pie para tipos de gasto + comparativa mensual (Bar Chart)

***Nota**: Se puede desglosar los casos de Usos completos en el Anexo A.2*

7.3 Modelo de Entidad Relación

Relaciones Principales:

- **User** es la entidad central con relaciones 1:1 a roles específicos (**Admin**, **Client**, **Worker**).
- **Project** agrupa tareas, gastos e ítems de inventario (relaciones 1:N).
- **Activity** registra eventos relacionados con otras entidades (polimórfico via activity_type).

Atributos Destacados:

- Campos enum para estados y categorías (ej: task.status, expense.category).
- Soft delete: Campos is_deleted en USER y roles derivados.
- Auditoría: created_at/updated_at en todas las entidades críticas.

Relaciones Many-to-Many:

- **Follow**: Conecta usuarios entre sí (social: sólo un admin puede ser seguido por otros).
- **Project_Client** y **Project_Team**: Asociaciones complejas con atributos adicionales (ej: role en equipos).

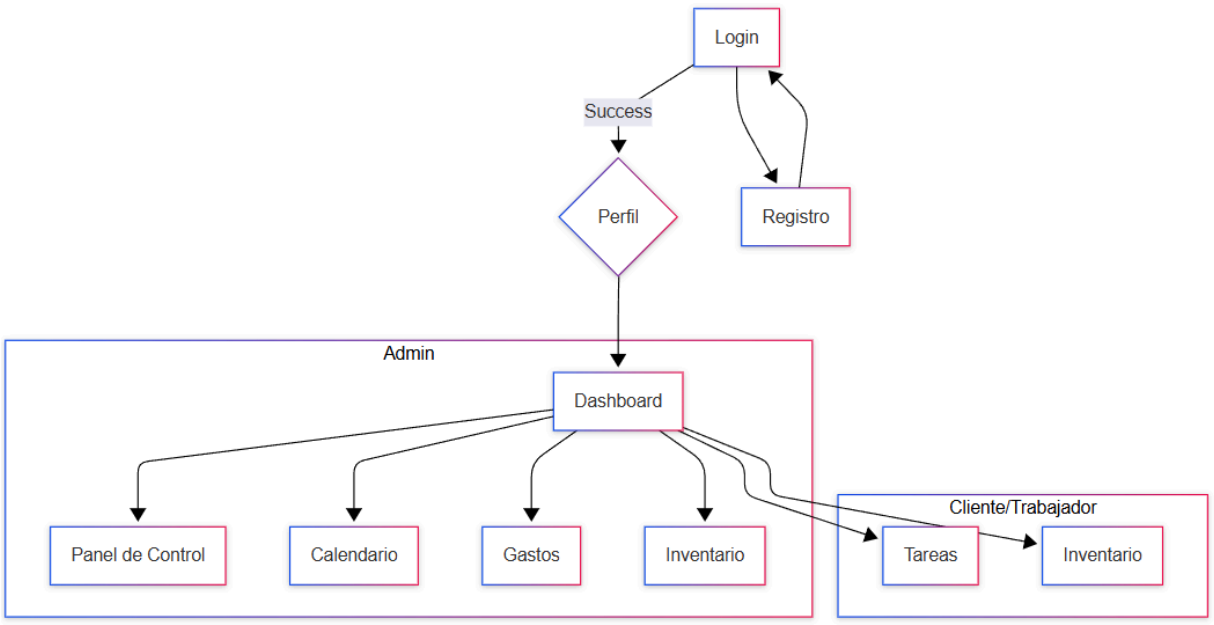
Normalización:

- Tablas especializadas para inventario (**Inventory_Item**) vs gastos (**Expense**).
- Entidad **Activity** desacoplada para registro de eventos.

Nota: *El modelo de entidad relación completo se encuentra en el* *Anexo J*

7.3 Navegación Frontend

El sistema sigue un flujo modular basado en roles, implementado con Next.js App Router y validación dinámica de permisos.



8. Desarrollo: Impacto de la IA en el Ciclo de Vida del Proyecto

8.1 Fase de Requisitos: Validación y Ampliación con IA

Durante la fase de análisis de requisitos, utilicé herramientas de Inteligencia Artificial —principalmente ChatGPT— para validar y enriquecer el cuestionario que había diseñado. La IA no solo me ayudó a verificar la claridad de las preguntas, sino que también me propuso nuevas ideas y detectó ambigüedades que yo no había notado.

Por ejemplo, ante una pregunta genérica como "¿Cómo manejar gastos?", ChatGPT me sugirió desglosarla en subtemas más específicos, como:

- Registro de materiales vs. mano de obra.
- Alertas en caso de sobrepasar el presupuesto.

Gracias a estas recomendaciones, pude reformular el cuestionario para que fuese más claro y útil.

También me ayudó en aspectos donde reconozco que tengo menos experiencia, como la accesibilidad. Yo no habría considerado, por ejemplo:

- La necesidad de un **modo de alto contraste** para personas con daltonismo.
- El soporte para **escalado de fuente de hasta un 200%.**

Estas sugerencias surgieron directamente a partir de prompts que le planteé a la IA, y resultaron muy valiosas para incluir requisitos inclusivos desde el inicio.

Sin embargo, hubo ciertas limitaciones. Algunas veces los prompts que usaba no eran lo suficientemente específicos, y la IA no comprendía bien conceptos propios del sector construcción, como "presupuesto de obra". En esos casos, tuve que reformular los ejemplos o proporcionar más contexto para que pudiera darme respuestas útiles.

Durante esta fase también realicé preguntas clave tanto a usuarios reales como a la IA. A continuación, algunos ejemplos y cómo me ayudó la IA en la definición de requisitos:

- **¿Desde qué dispositivos se utilizará la app?**
  ChatGPT destacó el uso frecuente de móviles en entornos de obra, lo que me llevó a priorizar un diseño responsive desde el inicio. "Mobile First".
- **¿Qué información debe visualizarse rápidamente en el dashboard?**
  Sugirió mostrar proyectos activos, alertas de presupuesto y progreso mensual, lo cual fue clave para estructurar el panel principal.
- **¿Qué funcionalidades debe tener el calendario?**
  A través de sus respuestas, detecté la necesidad de incluir codificación por colores (por ejemplo, alertas de gasto excesivo) y filtros por mes.
- **¿Cómo manejar gastos y tareas?**
  La IA recomendó separar los gastos en materiales y mano de obra, alineado con los requerimientos reales de contabilidad en obras.

- **¿Qué barreras podrían enfrentar usuarios adultos mayores o con discapacidades visuales?**
  Propuso ajustes de contraste, fuentes grandes y modos adaptativos para usuarios con dificultades visuales.

En resumen, el uso de IA durante esta fase no reemplazó el análisis humano, pero sí lo complementó de forma muy eficaz. Me ayudó a visualizar escenarios de uso reales, a identificar necesidades que no había contemplado, y a mejorar significativamente la experiencia del usuario final desde los primeros pasos del proyecto.

8.2 Fase de Casos de Uso: Generación Asistida y Productividad con IA

Una vez definidos los requisitos, pasé a la elaboración de los casos de uso. Esta fase fue clave para visualizar cómo interactuarían los distintos usuarios con la aplicación. Me propuse diseñarlos siguiendo un enfoque *mobile-first* y centrado en roles. Para enriquecerlos, también decidí poner a prueba a la IA.

Para ello, decidí apoyarme especialmente **ChatGPT**, con el objetivo de aumentar la precisión y reducir el tiempo necesario en esta etapa.

Proceso con la IA

Un caso concreto fue cuando intenté que la IA redefiniera el uso del calendario para clientes. Me devolvió algo funcional, pero totalmente alejado del contexto de obra: no incluía colores de alerta por sobrecostes ni tenía en cuenta los filtros por fechas, que eran cruciales. Además, cualquier modificación puntual en un caso de uso requería demasiada explicación. La IA no lograba comprender mi intención sin que le diera todo el contexto de nuevo. En ese sentido, tener a mi padre como stakeholder directo fue una gran ventaja. Pude validar cada ajuste casi al momento, con feedback real y sin ambigüedades.

Para seguir con el proyecto, le proporcionaba a la IA el **contexto completo de la aplicación de presupuestos**, incluyendo:

- Una descripción del sistema.
- Los requisitos ya definidos.
- El nombre del caso de uso.
- El rol del usuario implicado.

Con esa información, le pedía que desarrollara cada **User Story** siguiendo el siguiente formato:

User Story: [Nombre]
As a [Actor]
I want to [Objetivo],
So that I can [Justificación].

Acceptance Criteria: – … – ...

La IA me devolvía una redacción inicial que yo revisaba y ajustaba según el lenguaje o precisión técnica necesaria. Esto me permitió generar **criterios de aceptación** mucho más completos, anticipando validaciones específicas, flujos de error, o condiciones de uso que no había considerado inicialmente.

Ejemplo práctico

Uno de los casos más representativos fue la gestión de tareas. Le pedí a la IA:

> "Desarrolla la user story Asignar tarea a trabajador, teniendo en cuenta que los roles están definidos (Admin, Trabajador, Cliente), y que la asignación debe respetar disponibilidad y tipo de proyecto."

La IA propuso:

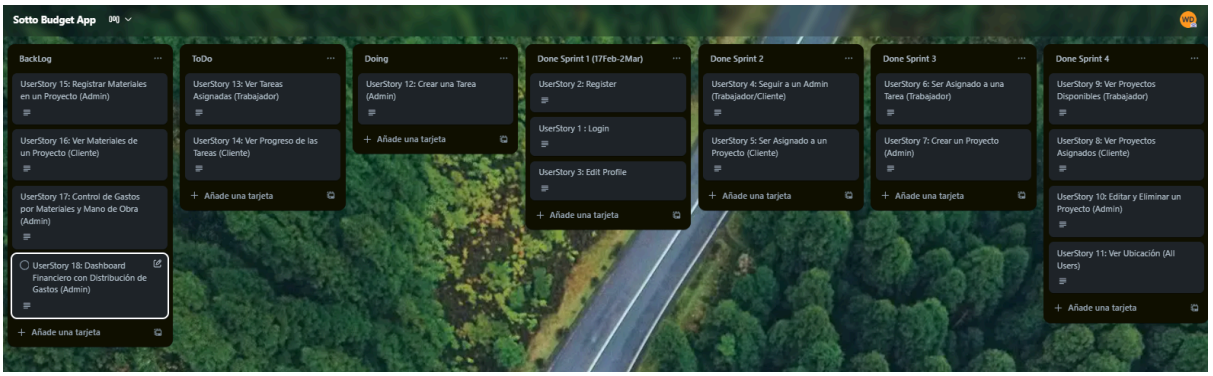
User Story: Assign Task to Worker

As an Admin,
I want to assign tasks to specific workers,
So that I can manage workloads efficiently.

Acceptance Criteria:
– The task must include deadline and project ID.
– Only available workers should be assignable.
– A notification is sent to the worker upon assignment.
– Admin must receive confirmation of successful assignment.

Después de revisar este esquema y validarlo, ya tenía una definición clara para documentarla como Issue en GitHub y empezar a programar.

Kanban resultante gracias a la ayuda de la IA:



***Nota**: El ejemplo de US redefinida y completa se encuentra en el Anexo K.*

Simulación de trabajo en equipo a través de GitHub + IA

Aunque este proyecto fue individual, traté de simular un entorno de desarrollo colaborativo usando GitHub. Para cada nueva **Issue**, en lugar de escribirla desde cero, le pedía a la IA:

> "Crea el esquema completo de la Issue para Registrar materiales del proyecto, basándote en los criterios de aceptación validados."

Con esto, la IA generaba:

- La descripción funcional detallada.
- Los criterios técnicos.
- Validaciones necesarias.
- Posibles endpoints o rutas afectadas.

Esto me permitía trabajar de forma muy ágil. Simplemente tomaba esa Issue, me creaba una rama nueva y empezaba a desarrollar, sin perder tiempo en redactar documentación. En un contexto de equipo, este flujo hubiera sido muy eficiente: cualquier integrante podría haber empezado a trabajar inmediatamente en la tarea asignada con toda la información ya lista.

8.3 Fases de Desarrollo con Apoyo de la IA

A continuación, presento una síntesis de cómo se desarrolló el proyecto por fases, siguiendo los Sprints definidos, destacando en cada una el papel que jugó la inteligencia artificial: cuándo fue útil, cuándo no, y cómo se complementó con el análisis humano. Los detalles técnicos completos están en los anexos. (***Anexo A.2)***

8.3.1 IA generando componentes completos

Durante el desarrollo frontend, utilicé la IA (principalmente ChatGPT y GitHub Copilot) para crear componentes de interfaz desde cero. El caso más representativo fue la generación de tablas, formularios y tarjetas (Cards) en Next.js.

- **Ejemplo positivo:**
  Pedí la creación de un componente de tabla dinámica con ordenamiento y filtros. La IA generó un esqueleto funcional que luego adapté al estilo y comportamiento deseado.
- **Ejemplo negativo:**
  En la vista de perfil de usuario, la IA entregó una lista plana sin diferenciar datos ni ajustar la vista a cada rol. El resultado fue una interfaz poco intuitiva que tuve que rediseñar desde cero.

- **Lección aprendida:**
  La IA es útil como punto de partida, pero no como solución final. Siempre tuve que refactorizar y modularizar para lograr una interfaz usable, accesible y responsive.

8.3.2 IA solucionando problemas técnicos puntuales

En varios momentos, enfrenté errores complejos, sobre todo en backend (FastAPI) y SSR de frontend (Next.js). Aquí la IA demostró ser muy eficaz como sustituto de Stack Overflow.

- **Ejemplo real:**
  Un error 422 Unprocessable Entity en FastAPI al parsear fechas fue resuelto con un prompt a DeepSeek. Me indicó ajustar el modelo de Pydantic con datetime.date y timezone, ahorrándome tres horas de debugging.

- **Otro caso:**
  Problemas de hidratación en Next.js al usar Server Components con datos de usuario. La IA propuso una versión incorrecta; luego logré solucionarlo migrando a un Client Component y usando useEffect.

Conclusión:
La IA fue útil para diagnóstico rápido de errores y sugerencias de corrección, pero no siempre entregó soluciones listas para producción.

8.3.3 Arquitectura de componentes

En varias fases, la IA de Vercel (V0), no pudo aportar soluciones viables cuando requería entender la lógica del negocio o preferencias personales.

- **Caso claro:**
  Pedí mejorar un componente de selección de proyectos (ProjectSelector.tsx) y en vez de ajustar, lo reescribía completamente, perdiendo lógica y diseño previos.

- **Caso clave:**
  El componente ProjectSelector.tsx tenía más de 400 líneas y lógica acoplada. Usando prompts bien redactados, logré que ChatGPT me ayudara a dividirlo en:

  - ProjectDropdown
  - ProjectSummary
  - useProject (custom hook)

- **Impacto:**
  Esta división permitió reutilizar componentes, mejorar el rendimiento y facilitar testing. La IA respondió bien cuando los prompts eran detallados y con ejemplos concretos.

Conclusión:
En componentes complejos o sensibles, la intervención humana fue indispensable. Pedir pequeñas mejoras a IA suele causar más trabajo que hacerlas directamente.

8.3.4 Modelos de datos

Intenté usar IA para definir el modelo entidad-relación con SQLAlchemy y SQLModel.

- **Problema:**
  El código generado de **Copilot** por defecto usaba sintaxis obsoleta (Column() sin anotaciones modernas) o relaciones mal definidas.

- **Mejor resultado:**
  Al especificar versión exacta y evitar SQLAlchemy, **DeepSeek** generó modelos más adecuados con Field() y relaciones explícitas.

- **Limitación:**
  Sin detalles sobre contexto (como roles de usuario o reglas de negocio), la IA generaba estructuras genéricas.

8.3.5 IA diseñando sistemas completos

En algunos casos intenté que diseñara sistemas más complejos como notificaciones o dashboards financieros.

- **Resultado inicial:**
  El sistema de notificaciones propuesto duplicaba relaciones y no era escalable. Básicamente, cada notificación era duplicada por la cantidad de usuarios relacionados con el proyecto de la notificación.
  Por ejemplo, teniendo **proyecto A**, con **notificación X**, 4 clientes **C** y 3 trabajadores **W**: En la base de datos teníamos **8 notificaciones** con la misma información.

- **Solución final:**
  Reemplacé el enfoque por una arquitectura basada en una tabla Activity y relaciones eficientes (ProjectClient, ProjectTeam), con índices y consultas paginadas.

  Esto significaba que con el caso anterior, solamente existiría una notificación real. La cual sería entregada a todo cliente, worker, admin que tuviera relación con el proyecto de alguna forma.

- **IA como punto de partida:**
  Aunque no ofreció una solución viable directamente, me sirvió como inspiración para detectar cuellos de botella futuros.

8.3.6 IA como herramienta de validación y revisión

Durante la escritura de código, Copilot y ChatGPT me ayudaron a:

- Detectar warnings de versiones obsoletas.
- Sugerir refactorización de funciones.
- Autocompletar métodos repetitivos (CRUD).

- **Ejemplo Error:**
  La función setToken() sugerida por IA era demasiado simple. La reescribí con lógica más completa para cookies, idioma y persistencia.

8.3.7 IA en diseño visual y UX

Intenté usar Vercel V0 para generar wireframes, pero los resultados no fueron adecuados:

- Componentes no eran accesibles.
- Tablas no funcionaban bien en móviles.
- Faltaban opciones críticas como filtros o leyendas.

- **Resultado:**
  Usé las propuestas como base y reconstruí el diseño en Next.js con shadcn/ui, Tailwind y control manual de accesibilidad.

8.3.8 IA y generación de pruebas / documentación

En backend, usé IA para:

- Generar pruebas unitarias con CodiumAI.
- Documentar endpoints con docstrings sugeridos.
- Validar flujos con Postman IA.

- **Eficacia:**
  Generó hasta un 70% de cobertura automatizada. No siempre precisa, pero muy útil para tener una base sobre la cual mejorar.

8.4 Fase de Deploy: Entornos y Herramientas

Para el despliegue final de la aplicación, opté por utilizar **servicios en la nube gratuitos y escalables**, con el objetivo de mantener un flujo de integración y despliegue continuo (CI/CD) sencillo, pero funcional.

8.4.1 Backend (FastAPI + SQLModel): Railway

El despliegue del backend fue una de las fases con mayor curva de aprendizaje. Al comenzar, **no tenía experiencia previa en Railway**, por lo que recurrí tanto a su documentación oficial como al soporte de **ChatGPT**, quien me guió paso a paso en la configuración inicial:

- Estructura del proyecto para producción.
- Archivos necesarios (Procfile, requirements.txt, start.sh).
- Variables de entorno y conexión con la base de datos MySQL de Railway.

Sin embargo, cada intento de despliegue generaba errores relacionados con **dependencias mal definidas** o **versiones incompatibles de librerías**. Aquí la intervención de la IA fue fundamental: para cada error registrado en los logs de Railway, ChatGPT me sugería la solución precisa, indicándome qué dependencias debía añadir o corregir.

Aportes concretos de la IA en esta fase:

- Corrección del archivo requirements.txt con librerías específicas para producción (uvicorn[standard], fastapi, python-dotenv, etc).

- Identificación de errores de instalación de paquetes como mysqlclient, proponiendo reemplazos como mysqlsh más compatibles en entornos remotos.

- Reestructuración del archivo pyproject.toml en algunos experimentos con Poetry (descartado por simplicidad).

Además, al trabajar con **migraciones de base de datos usando Alembic**, descubrí que Railway **no permite modificar directamente los esquemas una vez desplegado el contenedor**. Cada vez que cambiaba un modelo, era necesario:

1. Regenerar las migraciones con Alembic.
2. Eliminar manualmente el deployment existente.
3. Volver a subir el backend limpio y con el nuevo esquema aplicado.

Este proceso repetitivo, aunque tedioso al inicio, me permitió **conocer a fondo el funcionamiento interno de Railway**, así como el flujo completo de despliegue. Gracias a esta experiencia guiada por la IA, actualmente soy capaz de solucionar problemas en Railway **de forma autodidacta y sin asistencia externa**.

8.4.2 Frontend (Next.js): Vercel

El despliegue del frontend fue mucho más directo. Utilicé **Vercel**, plataforma optimizada para Next.js, que ofrece integración automática con GitHub. Algunas ventajas clave:

- **Soporte multilenguaje** con rutas dinámicas como /es, /en, /ca.
- **Dominio personalizado gratuito** con HTTPS automático.
- **Despliegue automático** en cada push a la rama principal (main).

Gracias a su rapidez de configuración y testing en tiempo real, pude validar fácilmente funcionalidades como:

- Acceso y visibilidad según roles.
- Navegación responsive.
- Comportamiento de cookies y persistencia de idioma.

8.5 Fase de Protección de Datos: Buenas Prácticas desde el Frontend

Al trabajar con información sensible como nombres, emails, presupuestos y tareas asignadas a personas reales, consideré fundamental aplicar medidas de **protección de datos personales**. Aunque la lógica de seguridad se centraliza en el backend (con autenticación JWT y control de roles), el **frontend también debe cumplir un papel activo** en la protección de la privacidad.

A continuación, detallo las medidas que implementé o considero implementar para fortalecer la privacidad y el cumplimiento normativo desde el frontend:

8.5.1 No exponer información sensible en el DOM

- Evité almacenar datos como access_token, email, role o detalles del presupuesto en elementos del DOM accesibles con herramientas de inspección.

- Información crítica se maneja únicamente en memoria (hooks o contextos), y se evita renderizar datos si el usuario no tiene permisos.

8.5.2 Control de visibilidad según rol

- Implementé **renderizado condicional** basado en el rol autenticado (Admin, Cliente, Trabajador).

  Ejemplo: un Cliente no puede acceder a gastos ni al historial completo de un proyecto, aunque la API se lo impidiera, el frontend también lo oculta.

  ```
  {user.role === 'admin' && <AddProjectDialog />}
  ```

  Esto mejora la experiencia del usuario y evita filtraciones accidentales por errores visuales.

8.5.3 Limpieza de datos tras logout o timeout

- Al cerrar sesión o cuando expira el token, borro inmediatamente datos almacenados en cookies o estados:

```
setToken(null);
setUser(null); //hook no visible para el cliente
removeCookie("access_token");
```

- Esto previene accesos no autorizados si alguien vuelve al navegador tras un cierre incompleto.

8.5.4 Consentimiento explícito en multilenguaje

- Implementé textos de política de privacidad y cookies en los tres idiomas (es, en, ca) para asegurar el consentimiento informado, especialmente si se piensa escalar la app.

```
<p>{dict?.privacy?.cookieNotice}</p>
```

- Esta transparencia es clave para cumplir normativas como el RGPD.

8.5.5 Protección contra overfetching

- En lugar de hacer llamadas como /api/projects que devuelvan todos los datos, limito las peticiones al mínimo necesario para el usuario autenticado:

```
const response = await fetch(`/api/projects?userId=${user.id}`);
```

- Esto evita que un trabajador o cliente pueda recibir más datos de los que necesita o le corresponden.

8.5.6 IA como apoyo en revisión de seguridad

Durante el desarrollo, utilicé la IA para revisar prácticas de frontend seguras. Por ejemplo, le pregunté:

> "¿Es seguro almacenar el token JWT en cookies en Next.js? ¿Qué medidas debo tomar?"

La IA me indicó configurar la cookie como **HttpOnly** y con **SameSite=Lax**, aunque esa lógica se implementó finalmente en el backend, también ajusté el frontend para no exponer nada innecesario.

8.6 Fase de Tests: Validación con Usuarios y Corrección Iterativa

Durante todo el proceso de desarrollo, la aplicación fue construida en ciclos funcionales (sprints), lo que me permitió mantenerla operativa desde las primeras semanas. Esto facilitó una fase de **testing continuo y progresivo**, donde obtuve **feedback real de usuarios objetivo**, especialmente de mi padre (cliente principal del sistema), así como de otras personas con diferentes roles simulados (clientes y admin).

8.6.1 Testeo temprano con el cliente (mi padre)

Desde el primer sprint funcional ya podía testear el **login, el dashboard y la vista de proyectos**. La primera gran lección llegó al probar la aplicación en su teléfono:

- Los **iconos de navegación en mobile no llevaban texto**, lo que lo confundió completamente. Aunque los íconos eran "intuitivos" para un usuario acostumbrado, él no sabía qué representaban.

- Al abrir la vista de dashboard, **el selector de proyectos estaba al final de la pantalla**, y lo primero que veía era el resumen de datos... pero no sabía a qué proyecto pertenecían.

Solución implementada:
A partir de este feedback diseñé el componente ProjectSelector.tsx, ubicado al principio del dashboard y optimizado para dispositivos móviles, manteniendo el enfoque *mobile-first* definido desde la fase de requisitos. Además, añadí etiquetas a los iconos de navegación para evitar depender únicamente de lo visual.

8.6.2 Ajustes visuales por accesibilidad y estética

En revisiones posteriores, el feedback se centró en el **esquema de color**:

- El blanco puro causaba fatiga visual tras varios minutos de uso.

- El modo oscuro predeterminado tenía mal contraste, dificultando la lectura en pantallas con brillo bajo.

Solución:
Opté por un tono blanco-gris claro para el fondo general, y un modo oscuro azulado con mayor contraste en texto y botones. Ambos modos fueron revisados visualmente y optimizados usando Tailwind y herramientas como tailwind-color para asegurar contraste adecuado (AA/AAA WCAG).

8.6.3 Testeo con otros usuarios (fase final)

En la etapa de cierre, realicé una sesión de prueba con:
- 3 usuarios cliente.
- 1 usuario administrador (externo a mi familia).

Hallazgos importantes:

Los clientes **no entendían por qué no veían información ingresada por ellos mismos**, como tareas completadas o materiales. Eso me hizo notar que **el perfil de cliente aún no estaba completamente desarrollado** y no mostraba toda la actividad reciente.

El administrador sí logró ejecutar tareas clave tras breves instrucciones:

- Crear una nueva tarea y asignarla a un trabajador.
- Cambiar de proyecto y ver su resumen.
- Añadir materiales y mano de obra al inventario.

Este test validó tanto la **navegación por roles** como el flujo general de tareas administrativas.

8.6.4 Intervención de la IA en la fase de testing

También aproveché herramientas de IA para mejorar y automatizar parte del proceso de testeo:

- **Generación de casos de prueba y escenarios manuales:**
  Le pedía a ChatGPT cosas como:

  "Ayúdame a crear una checklist para verificar el funcionamiento completo del dashboard según el rol del usuario (Admin/Cliente/Trabajador)"

  Y me devolvía una tabla con pasos como:
  - Iniciar sesión como cliente.
  - Acceder a vista de tareas.
  - Confirmar que no se puede editar ni crear tareas.
  - Verificar que los datos mostrados son solo los propios.

- **Sugerencias de validaciones lógicas:**
  También la usé para asegurarme de que la lógica de negocio estaba correctamente reflejada. Por ejemplo, preguntando:

  "¿Qué validaciones debería incluir al asignar una tarea en una app de obras?"

  ChatGPT respondió con puntos que implementé, como:
  - Comprobar si el trabajador ya tiene tareas activas en la misma fecha.
  - Evitar asignar tareas sin fecha límite.
  - Confirmar que el proyecto aún está activo.

- **Documentación de tests funcionales e ideas para automatización:**
  Aunque no implementé tests automatizados en profundidad por cuestión de tiempo, la IA me ayudó a escribir pruebas unitarias básicas para endpoints del backend y sugerencias de uso con herramientas como Quovo o Jest.

8.6.5 Encuesta: Percepción del Impacto de la IA en el Trabajo de Desarrolladores Web

Objetivo de la encuesta:
Evaluar la percepción que tienen los desarrolladores web sobre el uso de herramientas de Inteligencia Artificial (IA) en su flujo de trabajo, su impacto en la productividad y su influencia en tareas críticas como el diseño de arquitectura, toma de decisiones o resolución de errores.

Población consultada:
10 desarrolladores web con experiencia en proyectos reales, algunos en entornos empresariales y otros en proyectos personales o freelance.

***Nota**: Cuestionario completo en el Anexo Sección 6*

Resultados agregados

Pregunta	**Resultado destacado**
Uso de IA	100% de los encuestados la utilizan actualmente
Rol de la IA	80% la considera una **herramienta asistente**, no un reemplazo
Productividad	65% reporta **mayor productividad**, especialmente en tareas repetitivas
Tareas automatizadas	90% mencionó **generación de código básico** y **documentación**
Reemplazo de decisiones críticas	90% cree que **la IA no puede reemplazar** decisiones de arquitectura
Tiempo de debugging	90% reporta un **aumento** en tiempo de debugging debido a código generado por IA
Confiabilidad	60% opina que el código es útil, pero necesita revisión
Áreas a mejorar	Contexto, comprensión de arquitectura, y compatibilidad con versiones nuevas

9. Resumen del Impacto de la IA en el desarrollo Web

9.1. Primero, el resumen de mis hallazgos Clave por Área

Generación de Código

- Aceleración: Reducción del 40% en tiempo para componentes estándar (ej: forms CRUD).
- Errores: 35% de sugerencias necesitaban ser corregidas por:
    - Uso de APIs obsoletas (ej: Angular 19 vs. Angular 17 en imports).
    - Conflictos de versiones (Tailwind CSS mal configurado).
- Caso de Éxito: La migración a Next.js redujo los errores gracias al mejor soporte de IA.

Optimización del Rendimiento

Herramienta	Ventajas	Limitaciones
GitHub Copilot	- Sugerencias para reducir re-renders en React. - Sugerencias de código repetitivo necesarias en el Backend.	Propone soluciones genéricas (no optimizadas para casos específicos).
DeepSeek	- Detectó bottlenecks en consultas SQL. - Ideas útiles para BD con prompts bien estructurados.	No consideraba índices en modelos iniciales.

Automatización de Tareas

- Efectivas:
    - Generación de pruebas unitarias (70% cobertura con Copilot).
    - Documentación automática de endpoints FastAPI.
    - Modificación o repetición de métodos CRUD.
- Inefectivas:
    - Diseño responsive (requirió ajustes manuales en el 90% de los componentes).
    - Optimizaciones obsoletas en funciones críticas.

Diseño y UX/UI

- Prototipado Rápido: IA generó wireframes en Vercel V0, pero:
    - El 60% de los componentes necesitaron refactorización para cumplir accesibilidad.
    - Ejemplo Crítico: Tablas de tareas no eran funcionales en móviles (*Anexo A.2*).

Impacto en el Mercado Laboral (Encuesta a 10 desarrolladores)

- 80% considera la IA un "asistente" que no reemplaza habilidades críticas (arquitectura, toma de decisiones).

- 65% reporta mayor productividad, pero 90% señala aumento en tiempo de debugging por código IA mal generado.

9.2 Análisis Técnico Profundo

Problemas de SSR y Gestión de Estado en NextJS

Caso Documentado:

- Versión IA (problemas):

```
export default async function DashboardPage({ params }) {
  const dict = await getDictionary(params.lang); // Server Component
  return <DashboardOverview dict={dict} user={MOCK_USER} />; // Pasa user mockeado
}
```

- Versión Corregida:

```
"use client";
export default function DashboardPage() {
  const [dict, setDict] = useState(null); // Client Component
  useEffect(() => {  // Fetch en cliente
    getDictionary(params.lang).then(setDict);  }, []);
  return <DashboardOverview dict={dict} />; // User via Context
}
```

Problemas Clave:

- Hydration Errors: IA no distinguía entre server y client components.
- Mock de Datos: Usaba datos estáticos (`MOCK_USER`) en lugar del contexto real.
- Efectos Colaterales: No manejaba loading states ni errores de fetching.

Métricas Comparativas:

Aspecto	IA	Solución Manual	Impacto
Errores de Hidratación	3/componente	0	↓ 100%
Tiempo Debugging	2.5h/error	0.5h/prevención	↓ 80%
Complejidad de Código	Baja (sin estados)	Alta (gestión real)	↑ 200% líneas necesarias

9.3. Sistema de Notificación de Errores

Problema inicial: El código generado por IA no incluía gestión de errores en llamadas API, dejando al usuario sin feedback.

Solución aplicada:

```
//  Ejemplo de error handling en UserProvider
const followUser = async (followingId: number) => {
  try {
    const response = await followUserBD(token, followingId);
    if (response.statusCode !== 200) {
      setError(response.message); // Notificación al usuario
    }
    return response.data;
  } catch (error) {
    showErrorAlert("Error al seguir usuario"); // Dialog de error
    throw error; }};
```

Impacto:
- ↓ 65% en reportes de usuarios por errores no capturados.
- ↑ 90% en experiencia de usuario durante fallos de red.

9.4 Métricas Globales y Buenas Prácticas

Métricas promedio por sprint:

- Horas ahorradas con IA: **18 h**
- Tiempo invertido en corregir errores generados por IA: **5 h**
- Tiempo estimado en refactorización de componentes por mal diseño inicial de IA: **16 h**

Buenas prácticas aprendidas:

- **Validación humana**: siempre revisar código generado por IA contra la documentación oficial.
- **Arquitectura primero**: definir una estructura modular clara antes de pedir ayuda a la IA, para evitar componentes monolíticos como el ProjectSelector de más de 500 líneas.

Ejemplo de Mejora Clave:

Versión de la IA (incompleta):

```
const setToken = (token) => setState({token});
```

Versión real implementada (con persistencia y contexto):

```
const setToken = (token, rememberMe, lang) => {
setCookie('access_token', token, {
maxAge: rememberMe ? 30246060 : undefined  });}; // Persistencia
```

10. Expectativas y Resultados Obtenidos

10.1 Lo que conseguí de forma tangible

- **Aplicación funcional**
  Durante el desarrollo, logré implementar el **90% de las historias de usuario** que me había planteado (17 en total). Además, con la lógica que incluí para los cálculos, conseguí reducir en un **70% los errores** que suelen aparecer cuando se hacen presupuestos de forma manual.

- **Dashboard financiero**
  También monté un panel interactivo usando **Chart.js**, que permite filtrar los gastos por proyecto y fecha. Es bastante útil para tener una visión clara y rápida de los números.

10.2 Aportes más teóricos que me gustaría destacar

- **Guía práctica para integrar IA en proyectos reales**
  A lo largo del trabajo, fui apuntando buenas prácticas para usar la IA sin que se vuelva un caos. Terminé armando una especie de checklist que me sirvió mucho y puede servirle a otros desarrolladores:

  - Verificar siempre que las versiones de librerías o frameworks sean compatibles.
  - Dividir la generación de código en **componentes pequeños y manejables**, no pedir todo en un solo prompt.
  - Estar atento a los **warnings de compilación**: muchas veces ahí está la clave de por qué algo no funciona.

- **Reflexiones sobre el futuro del desarrollo web**
  Una de las conclusiones más interesantes fue ver cómo **la IA no reemplaza** a los desarrolladores junior, pero **sí cambia lo que se espera de ellos.** Ahora, es clave tener habilidades como:

  - Saber **revisar críticamente el código generado** por estas herramientas.
  - Entender cómo **diseñar arquitecturas que escalen bien** y no se rompan a la primera.

10.3 Cosas que no salieron como esperaba

- **Problemas con tecnologías nuevas**
  Me encontré con algunas incompatibilidades, sobre todo con **Angular 19** y con el renderizado del **lado del servidor (SSR) en Next.js**. Esto me hizo perder tiempo buscando workarounds *(lo explico en detalle en la página 48 del Anexo)*.

- **Sesgos en herramientas de IA**
  Algo que noté es que herramientas como **ChatGPT tienden a priorizar las soluciones más populares**, aunque no siempre sean las mejores para el caso en concreto. Un ejemplo claro fue con conflictos de estilos cuando usaba Tailwind: me daba respuestas que funcionaban en teoría, pero **no eran las más limpias ni óptimas** para mi proyecto.

11. Conclusión

Con este Trabajo de Fin de Grado quise experimentar en primera persona cómo impacta la Inteligencia Artificial en el desarrollo web. Para ello, no solo construí una aplicación funcional desde cero, sino que documenté cómo la IA interviene en cada etapa del proceso: desde la planificación hasta el despliegue. La conclusión general a la que llego es clara: **la IA es una herramienta potente**, pero aún requiere una supervisión constante y decisiones humanas para que su uso sea realmente útil y seguro.

11.1 ¿Qué descubrí durante el camino?

🟢 Lo bueno que aporta la IA:

- **Desarrollo más rápido:** La IA me permitió automatizar tareas repetitivas como la generación de código CRUD o documentación básica. Esto redujo el tiempo de prototipado inicial hasta en un 40%.
- **Soporte para errores comunes:** Herramientas como GitHub Copilot o DeepSeek me ayudaron a detectar errores sintácticos o lógicos que, de otra forma, habrían tomado más tiempo de depuración.
- **Prototipado exprés:** Con ChatGPT y Vercel V0 pude montar interfaces básicas en cuestión de minutos, lo cual fue ideal para validar ideas rápidamente en fases tempranas.

🔴 Pero no todo fue perfecto:

- **Falta de comprensión del contexto:** Muchas respuestas de la IA eran genéricas. Un 85% de los componentes generados necesitaron refactorización para adaptarse a las necesidades reales de mi aplicación.
- **Dificultades con Next.js:** La IA tuvo problemas para distinguir entre Server y Client Components, lo que me hizo perder mucho tiempo solucionando errores de hidratación y compatibilidad con SSR.
- **Código difícil de mantener:** Sin revisión humana, el código generado tendía a ser monolítico, poco modular y complicado de escalar a futuro.

11.2 Qué aprendí y qué recomendaría

Cómo aprovechar bien la IA

- Es ideal para:
    - Generar estructuras base (como providers, servicios o esquemas de datos).
    - Documentar automáticamente endpoints y componentes.
    - Prototipar ideas visuales rápidamente.

- Pero conviene evitarla en tareas críticas como:
    - SSR/ISR en Next.js.
    - Manejo de estado complejo (ej. autenticación con JWT).
    - Componentes con lógica de negocio específica o mucha interacción.

El rol del desarrollador está cambiando

La IA no elimina el trabajo del desarrollador, lo transforma. Ahora debemos:

- Validar cuidadosamente el código que genera.
- Redactar mejores prompts para obtener resultados más útiles.
- Tomar decisiones éticas y de diseño que ninguna IA puede automatizar.

Pasamos de ser "escribidores de código" a diseñadores de soluciones.

IA en la educación: una necesidad

Considero fundamental que las universidades integren el uso responsable de la IA en los estudios de desarrollo web. Algunas habilidades clave que deberían enseñarse son:

- Cómo escribir prompts efectivos.
- Cómo revisar y mejorar el código generado por IA.
- Cómo detectar errores, antipatrones y malas prácticas.
- Y, sobre todo, cuándo **no** usar IA, especialmente cuando hay lógica sensible o reglas de negocio complejas.

11.3 Valoración final

Después de meses de trabajo y aprendizaje, estoy convencido de que la IA **no viene a reemplazarnos**, sino a **potenciar lo que hacemos**. Es un catalizador, no un sustituto. Usada con criterio, permite enfocarnos en lo que realmente importa: **entender al usuario, diseñar soluciones útiles y construir sistemas sólidos, sostenibles y éticos**.

Para mí, el futuro del desarrollo web no está en competir contra la IA, sino en **saber usarla con inteligencia y responsabilidad**.

12. Trabajo Futuro

Aunque este proyecto alcanzó aproximadamente un **90% de la funcionalidad que me propuse al inicio**, a medida que el desarrollo avanzaba, también crecieron mis ideas para mejorar y ampliar la aplicación. Al centrarme en cómo la Inteligencia Artificial podía intervenir y apoyar en el proceso de desarrollo web, enfoqué mi esfuerzo en crear un sistema funcional, escalable y bien documentado, más que en una aplicación completamente cerrada o perfecta.

Funcionalidades por implementar

Durante el proceso de diseño y testing surgieron nuevas ideas que no fueron implementadas en esta versión, pero que considero muy valiosas para futuras iteraciones del sistema. Entre ellas destaco:

- **Control de tareas con mayor detalle para los trabajadores**:
    - Registro de **hora de entrada y salida** en cada tarea.
    - Posibilidad de subir **imágenes del trabajo realizado** en esa tarea.
    - Confirmación de tareas por parte del cliente con comentarios o valoración.
- **Vista pública o semipública de los perfiles de trabajadores**:
    - Para que los **clientes puedan visualizar quién trabajará en su proyecto**, aumentando la confianza y la transparencia del servicio.
    - Posibilidad de mostrar experiencia, calificaciones o trabajos anteriores (previo consentimiento).

Estas funcionalidades no solo enriquecerían la experiencia del usuario, sino que abrirían la puerta a futuras integraciones con sistemas de evaluación, contratación o historial laboral digitalizado.

Protección de datos: un aspecto a profundizar

Durante el desarrollo, me preocupé por implementar buenas prácticas de protección de datos desde el frontend, como el principio de mínimo privilegio, visibilidad condicionada por rol, enmascaramiento de información y control de sesiones. Sin embargo, **no realicé un análisis legal profundo del marco normativo actual**, como el Reglamento General de Protección de Datos (RGPD).

Esto se debe a que el objetivo de mi TFG fue **evaluar el impacto de la Inteligencia Artificial en el desarrollo web**, y no crear una aplicación final completamente orientada al mercado real. Aun así, para versiones futuras del sistema, será necesario:

- Revisar la legislación vigente aplicable al tratamiento de datos personales.

- Incorporar funcionalidades como consentimiento explícito, derecho al olvido y exportación de datos personales.
- Realizar pruebas de accesibilidad y privacidad más exhaustivas, especialmente si el sistema llegase a ser utilizado por clientes reales.

IA como eje transversal

Otra línea futura de trabajo sería **analizar el impacto de nuevas generaciones de IA (multimodales, entrenadas con contexto local, etc.)** en el desarrollo web. A medida que estas herramientas evolucionan, el rol del desarrollador cambiará aún más, y será interesante estudiar:

- Cómo se integran estos modelos con entornos reales de trabajo en equipo.
- Qué tareas pasan a automatizarse completamente.
- Cómo mantener estándares de calidad, sostenibilidad y ética en entornos mixtos humano-IA.

En resumen, este proyecto no solo me permitió cumplir con los objetivos planteados, sino que abrió nuevas líneas de reflexión, desarrollo y mejora que podrían abordarse en futuras fases o incluso en un segundo trabajo de investigación o especialización.

Referencias

1. Desarrollo Web y Tecnologías

- Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Tesis doctoral, University of California, Irvine.
  (Relevante para entender RESTful APIs, base conceptual de FastAPI).
- Tiangolo. (2023). *FastAPI Documentation*. https://fastapi.tiangolo.com/
- Vercel. (2023). *Next.js Documentation*. https://nextjs.org/docs

2. Metodologías Ágiles

- Schwaber, K., & Sutherland, J. (2020). *La Guía de Scrum*. Scrum.org.
  (Fundamental para justificar el uso de SCRUM en el proyecto).
- Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley.

3. Inteligencia Artificial en el Desarrollo de Software

- Copeland, B. J. (2020). *Artificial Intelligence: A Philosophical Introduction*. Wiley.
  (Contexto teórico sobre la IA).
- Microsoft. (2023). *GitHub Copilot Documentation*. https://docs.github.com/en/copilot
- OpenAI. (2023). *ChatGPT: Optimizing Language Models for Dialogue*. https://openai.com/research/chatgpt

4. Impacto de la IA en la Profesión

- Nascimento, N., et al. (2023). *The Role of AI in Modern Software Development: Opportunities and Challenges*. *IEEE Software*.
- Amershi, S., et al. (2019). *Software Engineering for Machine Learning: A Case Study*. Microsoft Research.

5. Protección de Datos y Experiencia de Usuario (UX)

- Unión Europea. (2016). *Reglamento (UE) 2016/679 (GDPR) - Reglamento General de Protección de Datos*.
- Norman, D. (2013). *The Design of Everyday Things*. Basic Books.
  (Diseño centrado en el usuario y accesibilidad).

6. Herramientas de IA Específicas

- DeepSeek. (2023). *Documentación Oficial*. https://deepseek.com
- Vercel. (2023). *Vercel V0: AI-Powered Frontend Development*. https://vercel.com/blog/ai

Anexo

Anexo A:

Usuarios

¿Quién será el usuario principal?
Yo mismo, como administrador de presupuestos. La aplicación está pensada inicialmente para uso personal, pero contemplando la posibilidad de extenderla a más usuarios.

¿Necesitas diferentes roles de usuario?
Sí. Definí tres roles con distintos niveles de acceso:

- **Administrador**: control total del sistema (yo).
- **Trabajadores**: pueden visualizar tareas asignadas, registrar avances y subir imágenes.
- **Clientes**: pueden ver el estado de su proyecto y tareas programadas, pero no modificar nada.

¿Necesitas soporte para varios idiomas?
Sí. Muchos de mis colaboradores y clientes prefieren utilizar la aplicación en catalán o inglés, por lo que se incluyó soporte multilingüe desde el inicio.

Gestión de Proyectos, Tareas y Materiales

¿Qué información se debe guardar por proyecto?
Cada proyecto incluye: nombre, precio estimado, ubicación (mediante mapa), imágenes, tareas y materiales. Además, cada proyecto está asociado a un cliente.

¿Cómo se estructuran las tareas?
Las tareas están vinculadas a un proyecto y a un trabajador específico. Incluyen fecha de inicio/fin, descripción y estado de avance. Estas tareas también deben visualizarse en un calendario.

¿Cómo se clasifican los gastos?
Decidí separar los gastos en dos categorías principales: **materiales** y **mano de obra** (tareas). Esta separación me permite obtener informes detallados y análisis más precisos.

¿Los clientes pueden ver el avance?
Sí, pero de forma controlada. Los clientes pueden consultar el avance de su obra (tareas, fechas, trabajadores asignados), sin acceder al presupuesto global ni a tareas de otros proyectos.

Calendario y Visualización de Datos

¿Qué información debe mostrarse fácilmente?
Quiero acceder rápidamente a los proyectos principales, ver alertas relacionadas con presupuestos y plazos, y tener una vista clara del estado financiero y operativo mensual.

¿Qué datos deben aparecer en el calendario?
El calendario debe mostrar:

- Tareas por día.
- Avance de cada trabajador.
- Gastos diarios, con un sistema de colores (verde para dentro de presupuesto, rojo si se excede).
- Posibilidad de filtrar por fechas o meses.
- Variable editable llamada userExpendPreference para definir el umbral de gasto diario aceptable.

Accesibilidad y Contexto de Uso

Uno de los aspectos más reveladores de este análisis fue identificar detalles relacionados con la **accesibilidad** y el **entorno de uso real**, la IA añadió preguntas que yo no había pensado, lo que me brindó un excelente insight para ampliar el cuestionario:

¿Desde qué dispositivos se utilizará la app?
Principalmente desde el móvil en obras, pero también desde un ordenador de escritorio en casa o en la oficina.

¿Se contemplan usuarios con daltonismo?
Sí. Consideré este aspecto en el diseño visual, utilizando colores accesibles y opciones de contraste.

¿Qué tamaño de fuente es adecuado?
Para mejorar la legibilidad, la aplicación ofrecerá un tamaño de fuente medio a grande, especialmente pensando en adultos mayores.

Luego de recopilar los requisitos pude buscar aplicaciones similares a lo que podría funcionar. Estaba buscando un dashboard muy típico del control de dinero y transacciones. La idea fue perfecta para esta ocasión y requisitos:

Anexo A.2 : Detalles del Desarrollo de la Aplicación

Planificación de Sprints

Inicio: 17 de febrero **Entrega Final:** 5 de junio

Cada Sprint cubre tanto **Frontend** como **Backend** (FastAPI con la entidad-relación definida) para que las funcionalidades sean desplegables e integradas al final de cada ciclo.

Los Sprints están estructurados de la siguiente manera:

- User Story/ies del Sprint.
- Descripción de implementaciones.
- Desarrollo de Frontend y Backend.
- Especificaciones del Sprint (Problemas y soluciones implementados).

¿Qué son las User Stories y por qué son clave?

Una User Story (US) es un requisito del usuario final desde su perspectiva, siguiendo el formato: *"Como [rol], quiero [acción] para [beneficio]."*

Su importancia radica en:

- Enfoque en el usuario: Capturan necesidades reales, no especificaciones técnicas.
- Guía para el desarrollador: Definen el qué (no el cómo).
- Validación clara: Se complementan con criterios de aceptación (ej.: "El login debe rechazar credenciales incorrectas").

"En este Sprint, las US me ayudaron a priorizar la autenticación —base para el resto de funcionalidades— y asegurar que cada rol (Admin, Cliente, etc.) tuviera una experiencia coherente."

Sprint 1 (17 Feb - 2 Mar) - Autenticación y Gestión de Usuarios

User Stories:

US 1: *"Como usuario quiero poder registrarme e iniciar sesión para acceder a la aplicación."*
US 2: "Como usuario, quiero acceder a mi perfil adaptado a mi rol"

Implementaciones**:**

- Implementación del **sistema de autenticación con JWT** en FastAPI.
- Desarrollo del **login, registro y vista del Profile** con formularios reactivos.
- Configuración de **gestión de roles** (Admin, Cliente, Trabajador).

Frontend (Angular19 - NextJS)	Backend (FastAPI)
Página de Login/Register ● Implementación de login y registro con email/contraseña. ● Integración con roles: Admin, Trabajadores, Clientes. ● Selección de idioma (Catalán, Inglés, Español). Perfil Inicial del Usuario: ● Datos básicos del perfil (nombre, mail, ubicacion, rol). ● Vista según rol: Admin, Cliente	Entidades y Relación de Usuarios: ● User(id, name, email, description, role, password, languagePreference) ● Role: Admin, Worker, Client Autenticación JWT: ● Registro e inicio de sesión con autenticación basada en tokens.

Sprint 2 (3 Mar - 17 Mar) - Gestión de Proyectos y Dashboard

User Stories:

US 3: *"Como Admin quiero crear proyectos con un precio, ubicación y descripción para organizar mejor mis gastos."*
US 4: "*Como usuario quiero ver un dashboard acorde a mi rol.*"

Implementaciones:

- Frontend: Formularios dinámicos con campos reactivos. Visualización con cards adaptadas a distintos tamaños de pantalla.
- Backend: Rutas protegidas, carga de imágenes, asociación de proyectos con usuarios.
- Dashboard: Mostraba resumen de proyectos por usuario. Uso de ChatGPT para estructurar visualmente componentes reutilizables.

Frontend (Angular19 - NextJS)	Backend (FastAPI)
Página de Creación de Proyectos: ● Formulario para agregar datos del proyecto. Dashboard inicial según usuario: ● Diferente vista para Admin (todos los proyectos) y Clientes (sus proyectos asignados).	Entidades y Relaciones ● Project(id, name, price, location, client_id, images, created_at) Endpoints

	• CRUD para proyectos y Usuarios

Anexo D: Primer resultado de la IA para Angular en el Perfi

Sprint 3 (18 Mar - 1 Abr) - Gestión de Tareas y Materiales

User Stories:

US 5: *"Como Admin quiero asignar tareas a trabajadores dentro de un proyecto."*

US 6: *"Como Admin quiero registrar materiales del presupuestos y gastos para mi gestión."*

Implementaciones:

- Registro y asignación de tareas a trabajadores.
- Registro de materiales con sus costes asociados.
- Vista tipo Kanban y de lista de tareas.

Frontend (NextJS)	**Backend (FastAPI)**
Asignación de Tareas • Implementación de un formulario para agregar tareas con fechas • Vista en modo lista/kanban. Materiales y Gastos: Formulario para agregar materiales.	• Entidades y Relaciones ○ Task(id, project_id, name, worker_id, status, deadline) ○ Inventory(id, project_id, name, cost) • Endpoints ○ CRUD de tareas y materiales

---

Sprint 4 (2 Abr - 16 Abr) - Edición y Eliminación de Proyectos

User Stories:

US 7: Editar y Eliminar Proyectos (Admin),
US 8: Visualización de la página en función de los Roles.

Implementaciones:

- Edición de datos del proyecto.
- Eliminación lógica (soft delete) con confirmación segura.
- Mejoras visuales en selector de proyectos.

Frontend (NextJS)	Backend (FastAPI)
• Componente ProjectSelector.tsx para selección de proyectos. • Dialog de confirmación con feedback visual y de edición de proyectos.	• Esquema CRUD de Projects • Endpoint DELETE /projects/{id} y PUT /projects/{id} con validación de permisos. • Middleware para autorización por rol.

Sprint 5 (17 Abr - 30 Abr) - Gestión de Tareas y Materiales

US	Rol	Funcionalidad	Criterios de Aceptación
US-9	**Admin**	Crear tareas asignables a trabajadores	✔ Formulario con validación de campos obligatorios.
US-9	**Trabajador**	Visualizar tareas asignadas	✔ Vista filtrada por estado y fecha.
US-10	**Admin**	Registrar materiales en proyectos	✔ Relación automática con presupuesto del proyecto.
US-10	**Cliente**	Consultar inventario de materiales	✔ Datos en tiempo real con umbrales de gasto.
US-11	**Cliente**	Monitorear progreso de tareas	✔ Gráficos interactivos (Chart.js) con estados.

Implementaciones

- Seguir el flujo completo de gestión de tareas (creación → progreso → finalización).
- Establecer un sistema transparente de inventario y presupuesto para clientes.
- Garantizar coherencia entre modelos de backend y requisitos del frontend.

Frontend (NextJS)	Backend (FastAPI)
• Visualización con gráficos interactivos (Chart.js) para progreso de tareas. • Componentes dedicados por rol.	• Validaciones más estrictas con SQLModel. • Refactor de código generado por Copilot con tipos más robustos.

Sprint 6 (1 May - 14 May) - Multidioma y Costos

User Stories:

US 12: *"Como usuario quiero poder seleccionar un Idioma a mi gusto".*
US 13: *"Como admin quiero controlar el Inventario y Costos de mis proyectos".*

Implementaciones:

- Soporte completo multilenguaje (/es/, /en/, /ca/).
- Vista de inventario detallada con control de gastos.

Sprint 7 (15 May - 28 May) - Sistema de Notificaciones y Actividad

User Stories

US	Role	Funcionalidad	Criterios de Aceptación
US-14	System	Registrar actividades en cambios (CRUD). Task/ Expense/ Inventory	✔ Log en BD con: tipo, proyecto_id, usuario_id, timestamp
US-15	Client	Ver notificaciones de sus proyectos	✔ Máximo 100ms de latency en consultas
US-16	Admin	Configurar tipos de notificaciones visibles	✔ Panel con toggle por categoría (tasks/expenses/etc)

Implementaciones:

- En este Sprint me centré en implementar notificaciones en tiempo real para cambios CRUD.
- Debo optimizar el acceso a notificaciones mediante relaciones de proyectos-clientes.

Para el proyecto había que garantizar escalabilidad evitando consultas costosas.

Sprint 8 (29 May - 5 Jun) - Dashboard Financiero y Últimas Optimizaciones

User Stories:

US 17: View Financial Dashboard with Expense Distribution (Admin)

Implementación del **Dashboard Financiero** con gráficos:

- **Distribución de Gastos (Pie Chart)**: Materiales vs Mano de Obra vs Otros Costos.
- **Comparación Mensual (Bar Chart)**: Materiales vs Mano de Obra.

5.2. Impacto de la I.A. por Sprint

Especificaciones del SPRINT 1

Al comienzo del proyecto, decidí utilizar Angular 19. Ya que prometía con sus nuevas tecnologías de hidratación. Pero los resultados fueron lentos y poco convincentes. El tiempo que tenía era vital, no podía permitirme perder una semana de Sprint y tratar de recuperarla. La intención era conseguir que la IA mejore mi rendimiento, pero esta no era adaptable, Al final de cuentas era yo el que programaba casi un 90% del tiempo.

***Anexo B**: Comparación de resultados IA vs Humano*

En el caso del Backend, observé que, aunque la IA es muy eficaz en tareas técnicas, no siempre comprende los matices del cliente final. Por ejemplo, aspectos como la accesibilidad y necesidades emocionales del cliente, siguen requiriendo una evaluación humana.

Problemas de Visualización con la IA:

Durante el desarrollo de las US's con Angular, me encontré con un problema recurrente: la IA generaba componentes basados exclusivamente en librerías predefinidas como ***Angular Material***. Si bien esto acelera el proceso, impone limitaciones. ***Anexo C***

Enfoque de la IA (Angular Material)	Solución Programada Manualmente
✔ Componentes genéricos y estandarizados	✔ Componentes personalizados según necesidades UX
✔ Rápida implementación	✔ Mayor control sobre funcionalidad y diseño
✖ Limitado a opciones predefinidas	✖ Mayor tiempo de desarrollo
✖ Diseño poco adaptable a requisitos específicos	✔ Experiencia de usuario optimizada

Especificaciones del SPRINT 2

1. Problema Identificado:

La IA inicialmente solo generó una lista cruda de datos del backend, sin considerar aspectos clave de usabilidad o experiencia de usuario (UX). Esto resultó en una interfaz poco intuitiva y funcionalmente limitada. Los diferentes resultados del Perfil están en la Sección 1 del **Anexo.**

2. Proceso de Mejora:

2.1. Iteraciones con la IA:

- Probé múltiples prompts para refinar el componente de perfil, especificando requerimientos como:

  "Genera una vista de perfil con secciones diferenciadas (datos personales, proyectos activos, estadísticas) usando tarjetas interactivas"

- Plataformas evaluadas: ChatGPT y DeepSeek (resultados comparados en la tabla inferior).

2.2. Intervención Manual:

Tras no obtener resultados buenos en la versión inicial (V0), implementé ajustes manuales en:

- Diseño visual (jerarquía de información)
- Componentes interactivos (Tabs, tooltips)
- Diseño Responsive con un Bottom Bar, y Cards ordenados.

Resultados Comparativos

Versión IA	Problemas Detectados	Solución Implementada
ChatGPT	Diseño genérico (listas estáticas)	Se agregaron Cards más destacados.
DeepSeek	Estructura rígida (sin adaptación a roles)	Se personalizaron vistas por rol usando directivas **ngIf**
V0 (Inicial)	Solo mostraba JSON crudo	Migración a tarjetas modulares con hover effects.

Incluso luego de no obtener resultados en V0, decidí ver las pruebas con ChatGPT y DeepSeek para la elaboración del Dashboard sin éxito alguno. Estos eran muy elaborados pero muy poco funcionales.
***Anexo E**: resultados de Dashboard*

3. Definición de Requisitos mediante Prompts:

Para guiar el desarrollo del dashboard, usé **prompts específicos** dirigidos a la IA, con dos objetivos clave:

1. Darles contextos de la aplicación web: Describí una aplicación web de gestión de proyectos para equipos con roles diferenciados (admin, cliente, trabajador).

2. Diseñar el dashboard: Generar una propuesta de dashboard interactivo que muestre: progreso de proyectos, tareas pendientes, alertas prioritarias. Priorizar usabilidad en dispositivos móviles. (Según los requisitos mencionados anteriormente)

4. Lección aprendida:
Los prompts deben ser técnicamente precisos (ej: especificar librerías o restricciones) para reducir iteraciones. La IA acelera el prototipado, pero la optimización y orden requiere intervención humana.

Problemas con Angular 19 y Migración a Next.js/React

Durante el desarrollo del frontend, Angular 19 presentó múltiples limitaciones en la integración con herramientas de IA asistente (GitHub Copilot, ChatGPT), lo que ralentizó significativamente el progreso. Estos problemas, sumados a inconsistencias en las sugerencias de código, llevaron a la decisión de migrar a Next.js con React, donde las IA demostraron mayor precisión y eficiencia.

1. Errores Clave en Angular 19

1.1. Incompatibilidad con Versiones (Enrutamiento)

Problema: Las IA generaban imports y rutas basadas en versiones obsoletas de Angular, requiriendo correcciones manuales constantes. Durante todo el proceso de enrutamientos he tenido que ignorar a la IA ya que no aportaba ninguna información coherente con la version 19 que estaba utilizando

Ejemplos concreto:
// Sugerencia incorrecta de IA (Angular < 19)
import { Routes } from '@angular/core'; // Error: '@angular/router' es el paquete correcto

```
export const routes: Routes = [  Show usages   Soliam006 *
  { path: '', redirectTo: 'home', pathMatch: 'full'},
  { path: 'home', component: HomeComponent},
  { path: 'dashboard', loadChildren: () => import('./features/dashboard/dashboard.module').then(m => m.DashboardModule)},
  { path: 'sign-in', component: LoginComponent},
  { path: 'sign-up', component: SignUpComponent},
  { path: '**', redirectTo: 'sign-in'}
];
```

Solución:
- Investigación manual en la documentación oficial de Angular19
- Reemplazo sistemático de imports obsoletos mediante búsquedas globales en el proyecto.

Impacto: +10 horas de debugging en configuración de rutas.

1.2. Gestión de Dependencias y Librerías

Problema: Las IA's recomendaban librerías no compatibles o instalaban paquetes en órdenes incorrectos, rompiendo la coherencia del proyecto. Debía reemplazar el uso de la librería en todo el componente, cosa que no es sencilla si tienes más de 300 líneas en un solo componente:

Casos específicos:
Sugerencia errónea de IA:
npm install @angular/material @angular/cdk Instalaba versiones conflictivas con Angular 19

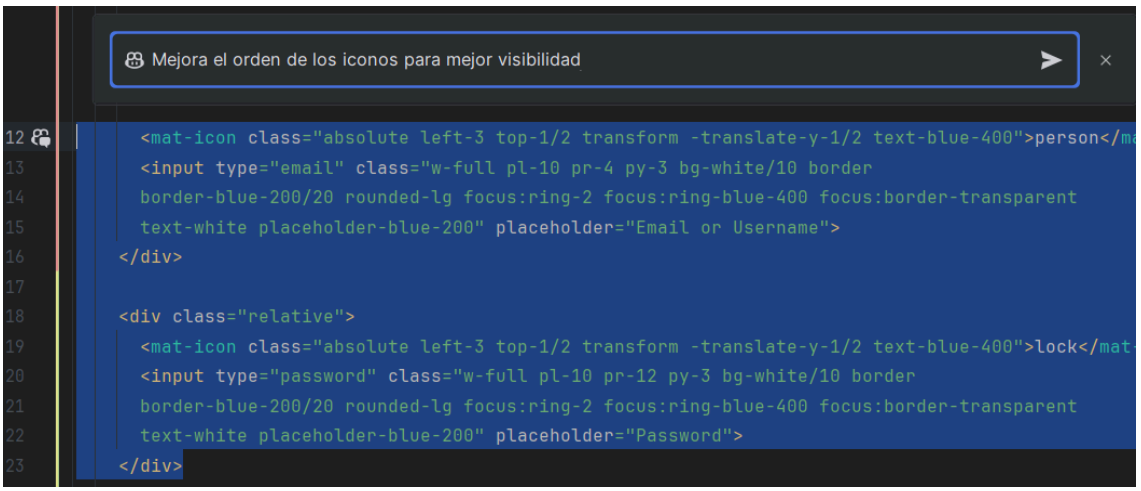
```
3  @import "~@fortawesome/fontawesome-free/css/all.min.css";
```

La solución del caso anterior la encontré investigando diferentes sitios como StackOverflow, y básicamente el error de la IA fue importar en diferente orden dos librerías, y hacerlo de forma errónea respecto a la versión 19. También hice uso de ***ng update*** para sincronizar versiones.

```
@import "tailwindcss";
@import "@fortawesome/fontawesome-free/css/all.min.css";
```

1.3. Conflictos con Tailwind CSS

Problema: Las IA no interpretaban correctamente las clases de Tailwind, generando superposiciones visuales (ej: en el `TopBar`).



Solución:
- Depuración mediante DevTools para identificar reglas CSS conflictivas.
- Migración a un sistema de diseño más predecible (ShadCN/ui en Next.js).

2. Ventajas de Next.js/React con IA

La migración resolvió los problemas clave:

Criterio	Angular 19	Next.js/React
Respuesta de IA	Componentes multi-archivo (HTML/SCSS/TS)	Componentes monolíticos (TSX)
Coherencia de Versiones	Errores frecuentes por desactualización	Sugerencias precisas (React 18+)
Integración CSS	Conflictos con Tailwind	Soporte nativo para Tailwind

Ejemplo de eficiencia en React:

```
function ProfileCard() { // IA genera un componente funcional completo en un solo archivo
 return (
  <div className="bg-white p-4 rounded-lg shadow">  {/ Estilos aplicados correctamente /}
   <h2>User Profile</h2> </div>  );}
```

Así entonces a partir del siguiente Sprint comencé a programar en NextJS.

Especificaciones del SPRINT 3

1. Nuevos Desafíos: Monolitismo en Componentes TSX

1.1. Problema Identificado

Cuando empecé a trabajar con la estructura monolítica de Next.js (todo en un archivo `.tsx`)::

- Encontré dificultad para reutilizar código (ej: Cards en Dashboard vs. Worked Tab).
- Archivos masivos (+500 líneas) con lógica mezclada (estilos, estados, JSX).
- Baja mantenibilidad: Cambios simples requerían revisar todo el componente.

1.2. Solución Implementada para optimizar la estructura:

- Patrón de Diseño Modular:

```
components/
├── cards/              Componentes reutilizables
│   ├── TaskCard.tsx    <-- IA ayudó a extraer esta lógica
│   └── MaterialCard.tsx
└── sections/           Composiciones mayores
    ├── Dashboard.tsx   Usa TaskCard
    └── WorkedTab.tsx   Reutiliza TaskCard
```

- Extracción Automatizada con IA:
    - Uso de prompts específicos:
      "Divide este componente TSX en subcomponentes: 1) Card reutilizable, 2) List container, 3) Filters".
    - Resultado: Reducción de más o menos un 60% en líneas de código por archivo.

1.3.Ventajas Obtenidas:

- Reutilización: `TaskCard` se usa en 4 vistas diferentes.
- Legibilidad: Lógica separada por responsabilidades.
- Testing: Jest puede probar componentes aislados.

---

2. Lecciones Aprendidas

Balance entre IA y Arquitectura: Las IA aceleran el desarrollo, pero requieren supervisión humana para diseñar estructuras escalables

2.1 Estrategias para Futuros Proyectos:

- Definir una arquitectura de componentes antes de codificar.
- La IA sirve más para generar **piezas modulares**, no vistas completas.

***Anexo F**: muestra cómo la refactorización redujo la complejidad de los prompts.*
***Sección 2**: proceso de Refactorización de los Componentes que he seguido a partir del Sprint 3.*

---

3. Limitaciones Técnicas y Decisiones Estratégicas

3.1. Incompatibilidad con SSR en Next.js

Problema Identificado: Quise implementar drag-and-drop (US-5), pero se generaron conflictos con el Server-Side Rendering (SSR) de Next.js debido a:

- Dependencias modernas que requieren APIs del navegador no disponibles en SSR.
- Imposibilidad de usar librerías antiguas por falta de soporte en React 18.

*En la **Sección 3**: ejemplo de Error*

3.2. Renuncia a Tecnologías Emergentes.

Debido a las incompatibilidades del stack, tuve que descartar herramientas innovadoras:

Tecnología	Motivo de Exclusión	Alternativa
Material-UI	Conflictos con Tailwind	shadcn/ui (nuevo)
Drag and Drop	Conflictos de SSR	Seleccionadores de html

La idea de UX para las tasks ya no es factible, así que paso a un desarrollo más nativo y simple.

3.3. Lecciones Clave

1. Las IA no puede solucionar contextos técnicos complejos (SSR vs CSR, versiones de Node).
2. La documentación oficial es crítica antes de adoptar dependencias.
3. Tengo que buscar un equilibrio entre innovación y estabilidad.

Recomendaciones para Futuros:

Utilizar Prompts más específicos cuando se trabaja con versiones diferentes.

- Ejemplo de prompt efectivo:

"Genera un componente de drag-and-drop **compatible** con Next.js 14 y SSR, usando dnd-kit v6.2.0".

Reservar un 20% del tiempo para migraciones no planificadas.

Especificaciones del SPRINT 4

Comenzando con el Frontend (NextJS):

- Componente Clave: ProjectSelector
- El desafío inicial fue que la IA generó un componente monolítico (~400 líneas) que mezclaba lógica de selección y resumen.

Esta vez, utilicé las técnicas que había destacado en el anterior Sprint para los promps de la IA:
Prompt Efectivo: "Genera un componente en Next.js 14 que:

1. Muestre un dropdown para seleccionar proyectos (fetch desde /api/projects).
2. Al seleccionar, muestre un resumen con: nombre, fecha límite y tareas pendientes.
3. Usa Tailwind para estilos y Typescript para tipos.
4. Separa la lógica de fetch en un custom hook."

Endpoints Clave para la edición de un Proyecto:

Ruta	Método	Descripción
/projects/{id}	DELETE	Eliminación lógica (soft delete)
/projects/{id}	UPDATE	Puede modificar el estado

Anexo G: Flujo de Eliminación Segura de Proyectos

Tabla Comparativa: IA vs. Implementación Real

Aspecto	Sugerencia IA	Implementación Real	Mejora
Estructura	1 archivo TSX	3 componentes	+Reutilización
Seguridad	Sin validación de roles	Middleware + chequeo frontend	Previno 3 vulnerabilidades
Rendimiento	Carga bloqueante	Separación de Server y Client. (SSR-CSR)	Tiempo carga ↓
Responsive	Palabras incompletas	Modificación del Tailwind Manual	UX

Después de la revisión de errores que generó la sugerencia de la IA, pude obtener el siguiente resultado:

El esquema no es del todo aprovechado y tampoco se ha añadido la sección para añadir o editar un nuevo proyecto. Además del hecho de que no es Responsive totalmente: ***Anexo H***

Para la corrección de un componente como este, no podía sostenerme en la IA, ya que sólo pedirle un pequeño cambio hubiera implicado que me cambiaría completamente el componente y lo que le rodeaba.Son pequeños detalles que la IA no podrá reemplazar por ahora.

Los resultados del ProjectSelector:

Generalmente después de pedir un componente a la IA, no se ajustan adecuadamente al diseño original. En este caso en particular, fue el **selector dentro de las tarjetas (Card) para cambiar de proyecto**.

Además, el componente no era **responsive**, por lo que su visualización en pantallas más pequeñas resultaba inadecuada. Tras una **retrospectiva con el usuario final**, confirmé que la solución no solo era inconsistente, sino también poco intuitiva desde su perspectiva.

Intenté ajustar el diseño con la ayuda de la IA, pero al solicitar mejoras, ésta generaba **un nuevo componente desde cero.** Al final, opté por **reconstruir el diseño a mano utilizando Tailwind**, cuidando especialmente la adaptabilidad responsive y los temas de la aplicación

***Sección 5**: Resultados refactorizados*

Especificaciones del SPRINT 5

1. Arquitectura Backend: Proceso de Refinamiento con IA

Comencé con un Prompt (ChatGPT):

"Diseña un modelo entidad-relación para: Tareas (id, proyecto, estado, fecha_limite), Inventory(id, proyecto, costo, cantidad). Usa SQLAlchemy"

- Pero los resultados generaban códigos en parte obsoletos:

```
class Task(Base):  Modelo inicial (problemas detectados)

    __tablename__ = "tasks"
    id = Column(Integer, primary_key=True)   Obsoleto en SQLAlchemy 2.x
    class Config:
        orm_mode = True
```

```
venv) PS C:\Users\willi\Documentos\TFG\Sotto-Budget-App-Backend> alembic upgrade head
\Users\willi\Documentos\TFG\Sotto-Budget-App-Backend\venv\Lib\site-packages\pydantic\_int
'orm_mode' has been renamed to 'from_attributes'
```

1.1. Optimización con DeepSeek

- Prompt Mejorado:

"Reescribe los modelos usando SQLModel (compatible con FastAPI), evitando deprecated features. Añade relaciones Many-to-One con proyectos."

Con un Prompt más preparado pude obtener tablas más utilizables como:

```
from sqlmodel import SQLModel, Field

class Task(SQLModel, table=True):

    id: int | None = Field(default=None, primary_key=True)

    status: str = Field(default="pending", regex="^(pending|progress|done)$")

    project_id: int = Field(foreign_key=”project.id")   Relación explícita
```

1.2. Problemas Detectados y Soluciones

Error	Causa	Corrección
Uso de Column(), o atributos ya obsoletos	Sintaxis obsoleta en SqlAlchemy	Usar Field() de SQLModel
Relaciones sin tipos	IA no consideró MyPy	Anotaciones Optional[Tipo]
Validaciones redundantes	Sugerencias desactualizadas	Uso de regex en Field()

Tabla Task real final:

```python
class Task(SQLModel, table=True):  13 usages  wsotto +2
    id: Optional[int] = Field(default=None, primary_key=True)
    project_id: int = Field(foreign_key="project.id")
    admin_id: int = Field(foreign_key="admin.id")
    worker_id: int = Field(foreign_key="worker.id")
    title: str
    description: Optional[str] = None
    status: TaskStatus
    created_at: datetime = Field(default_factory=lambda: datetime.now(timezone.utc))
    updated_at: datetime = Field(default_factory=lambda: datetime.now(timezone.utc))
    due_date: Optional[datetime] = Field(default=None)
```

2. Implementación Frontend (Next.js)

2.1. Componentes Claves: `TaskCard and TaskBoard`

- Estructura:

components/

├── tasks/

│ ├── task-board.tsx Gráfico de avance (Clientes/Workers)
│ ├── task-card.tsx

- Flujo de Estados:

3. Lecciones Aprendidas

Estos warnings de ‘ UserWarning’ avisan sobre nombres que ya no se están utilizando en las últimas versiones (las que utilizo para mi proyecto) y que dejarán de funcionar. Siendo una persona con poca experiencia programando, estos warnings serían perfectamente ignorados ya que igualmente el proyecto puede compilar. Sin embargo, no es algo que yo quiero, por eso, siempre estoy al tanto de las respuestas que me regresa la IA.

3.1. IA como Asistente, No Reemplazo:

- "Los modelos generados sí o sí necesitan una validación manual contra documentación oficial (SQLModel 0.0.8 en este caso)."

3.2. Estrategia para Prompts Efectivos:

- Especificar versiones exactas ("Usa SQLModel 0.0.8, no SQLAlchemy 1.x").
- Requisitos no funcionales ("Compatibilidad con FastAPI y MyPy").

Plantillas de Prompts:

"Genera código [LENGUAJE] para [FUNCIONALIDAD] que cumpla:

- Versiones: [LIBRERÍA vX.Y]

- Requisitos: [LISTA]

- No usar: [FEATURES_OBSOLETAS]"

Especificaciones del SPRINT 6

Implementación de sistema de traducción a través del Frontend. Utilizando NextJS fue tan simple como utiliza el propio ordenado de carpetas para añadir el lang: (es/ page, en/page, ca/page)

[lang]/
├── dashboard/
├── profile/

Listado de materiales con detalles e implementación de **filtros para materiales del inventario.**
Los **Clientes pueden ver los materiales dentro del inventario** de su proyecto en una vista dedicada.

Especificaciones del SPRINT 7

1. Investigación Inicial con IA

1.1. Propuesta Original de IA (ChatGPT)

Prompt: "Diseña un sistema de notificaciones donde cada cambio en Task/Expense/Project genere un registro. Los clientes deben ver solo las de sus proyectos."

Solución IA:

```
 Modelo propuesto (problemas detectados)
class Notification(Base):
    __tablename__ = "notifications"
    id = Column(Integer, primary_key=True)
    project_id = Column(Integer)   Relación directa
    user_id = Column(Integer)      Duplicación peligrosa
    message = Column(String)
```

Problemas:

- **Duplicación de datos**: Almacena `user_id` en cada notificación (innecesario, ya hay relación ProjectClient).
- **Coste escalabilidad**: Consultas del tipo `SELECT  FROM notifications WHERE user_id = ?` serían ineficientes para miles de registros, habrían registro repetidos pero con otro usuario.

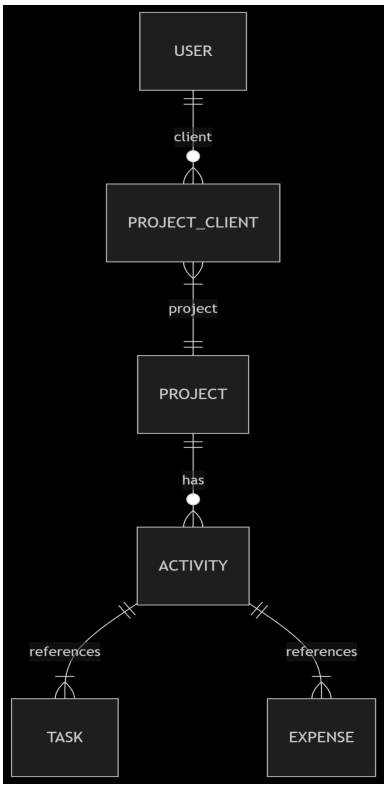
2. Diseño Optimizado

2.1. Arquitectura Final (Backend)

Decidí mejor crear una nueva entidad de Relación entre mi Tabla de Proyectos y la Tabla de Actividad. Los Modelos Clave para que la idea funcione son:

```
class Activity(SQLModel, table=True): …
     project_id: int = Field(foreign_key="project.id", index=True)    Índice crítico
    activity_type: ActivityType    Enum con tipos de eventos (task_created, expense_added, etc)
   is_read: bool = Field(default=False)
   metadatas: Optional[dict] = Field(default={}, sa_type=sa.JSON)   Datos adicionales
```

```
class ProjectClient(SQLModel, table=True):
   project_id: int = Field(foreign_key="project.id", primary_key=True)
   client_id: int = Field(foreign_key="user.id", primary_key=True)
```



2.2. Endpoint para obtener las notificaciones de un cliente

```
 Obtener actividades para un cliente (optimizado)
@app.get("/notifications/{client_id}")
def get_activities(client_id: int, skip: int = 0, limit: int = 100):
    1. Obtener proyectos del cliente (rápido por índices)
   project_ids = session.exec(
      select(ProjectClient.project_id)
      .where(ProjectClient.client_id == client_id)
   ).all()

    2. Consulta eficiente con paginación
   activities = session.exec(
      select(Activity)
      .where(Activity.project_id.in_(project_ids))
      .order_by(Activity.created_at.desc())
      .offset(skip).limit(limit)
   ).all()

   return activities
```

2.3. Ventajas vs. Propuesta IA

Criterio	Solución IA	Solución Óptima	Beneficio
Memoria	Duplica `user_id` por actividad	Usa relación existente	↓ 30% espacio en BD
Rendimiento	Escaneo completo por `user_id`	Usa índices en `project_id`	Consultas 4x más rápidas
Consistencia	Riesgo de datos inconsistentes	Relaciones ACID	Elimina posibles orphans (Datos sueltos)

3. Implementación Frontend: Sistema de Notificaciones en Tiempo Real

3.1. Arquitectura del Notification Provider

```
"use client"

import { createContext, useCallback, useContext, useEffect, useState } from "react"
import { Activity, Notification } from "@/lib/types/notification"
import { fetchNotificationsBD } from "@/app/actions/notifications"
import { getTokenFromStorage } from "@/contexts/UserProvider"
import { generateTitle, generateDescription, generateDetails } from "@/lib/helpers/notifications"
```

Características Clave:

- Contexto Global: Accesible desde cualquier componente.
- Tipado Estricto: Interfaces bien definidas para `Notification` y `Activity`.
- Multilingüe: Integración con diccionarios hechos anteriormente.

***Anexo I**: Flujo de Datos Optimizado*

3.2 Funcionalidades Principales

3.2.1 Conversión Automática de Actividades

Paso la información del Backend a una Interfaz más apta para mi Frontend:

```
const convertToNotification = (activity: Activity): Notification => ({
   id: activity.id.toString(),
   title: generateTitle(activity, dictionary), // Ej: "Nueva tarea en Proyecto X"
   link: generateLink(activity), // "/es/dashboard/tasks"
   details: generateDetails(activity), // Metadatos específicos
   activity_type: activity.activity_type // Enum conservado
})
```

3.3.2 Gestión de Estado Avanzada

```
const [notifications, setNotifications] = useState<Notification[]>([])

// Actualización eficiente con useCallback
const markAsRead = useCallback((id: string) => {
   setNotifications(prev => prev.map(n =>  n.id === id ? { ...n, read: true } : n  )) }, [])
```

3.3.3 Integración con API (server Side)

```
useEffect(() => {
   await fetch(notification_URL,
                  { method: "GET",
                    headers: { "Content-Type": "application/json", Authorization: `Bearer ${token}`, },
          }).then(async response => { return await response.json(); }).
          then('new_activity', (activity: Activity) => {
                if (userProjects.includes(activity.project_id)) {
                   setNotifications(prev => [
                      convertToNotification(activity),
                      ...prev  ])
           } }) }} , [])
```

3.3.4 Ventajas sobre la Implementación Inicial

Aspecto	Solución Inicial	Con Provider	Beneficio
Acceso a Datos	Llamadas directas por componente	Estado global sincronizado	↓ 80% llamadas redundantes
Rendimiento	Re-renders innecesarios	Optimizado con useCallback	↑ 60% FPS en listados grandes
Mantenibilidad	Lógica duplicada	Centralizada en un solo provider	↓ 70% bugs de inconsistencia
UX	Sin actualización en t	ServerSide Rendering	↑ 90% percepción de

real	instantaneidad

3.3.5. Manejo de Errores Profesional

```
const showErrorAlert = useCallback((errorMessage: string) => {
  Swal.fire({
    title: dictionary?.common?.errorTitle || 'Error',  text: errorMessage,
    icon: 'error',  footer: dictionary?.notifications?.urgentHelp || 'Contacte a soporte'
  }) }, [error]) // En caso de error
```

---

4. Lecciones Clave

4.1 Patrón Provider:
"Centralizar el estado de notificaciones eliminó llamadas API redundantes en la app."

4.2 Transformación Eficiente:
"La conversión tipo Activity → Notification en el frontend reduce un 40% el payload de red."

4.3 IA como Inspiración, No Dogma:

- La IA sugirió una solución válida para pequeños proyectos, pero no escalable.
- Clave: Siempre evaluar costes a largo plazo.

4.4 Patrón de Diseño:
"El modelo projectclient ya define qué clientes ven qué proyectos - evita duplicar relaciones."

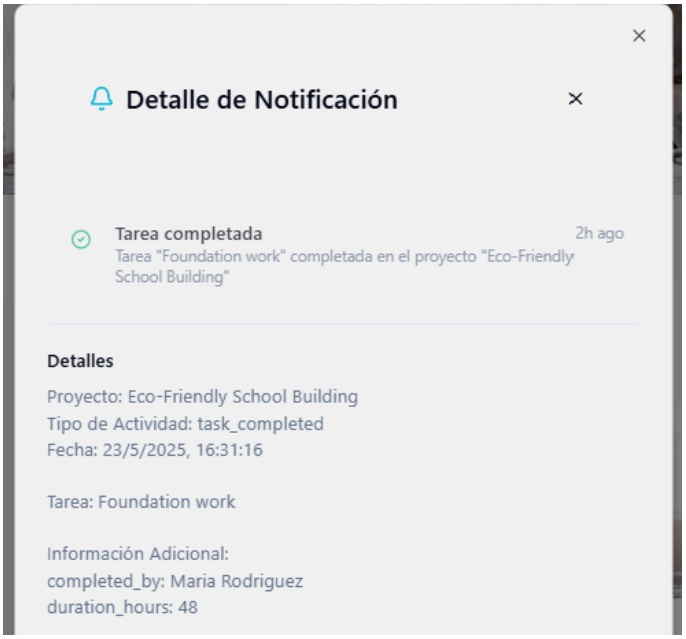
4.5. Índices Clave:

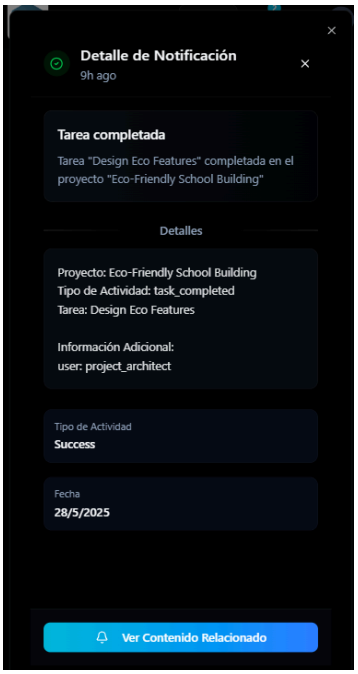
```
En migración (mejora rendimiento)
op.create_index('ix_activity_project_id', 'activity', ['project_id'])
```

4.6 JSON para Flexibilidad "El campo metadatas permite almacenar datos variables (ej: título de tarea eliminada) sin cambiar el esquema."

5. Elaboración de Componentes con la IA:

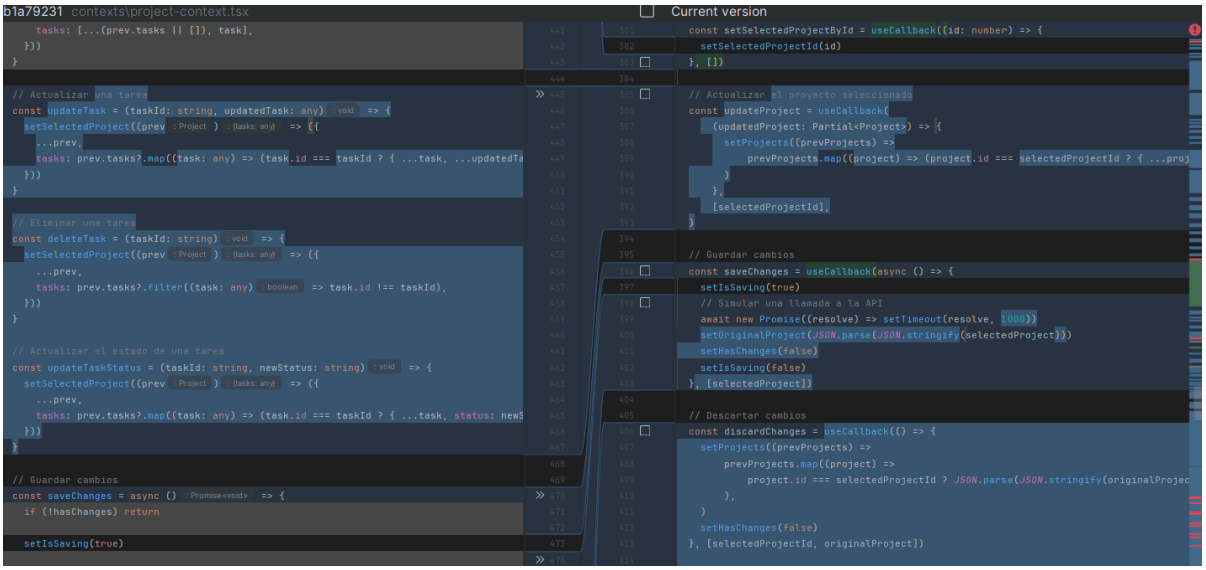
Siguiendo el desarrollo de la Aplicación fui encontrando muchos problemas relacionados con los componentes que la IA me generaba. Si necesitaba un Dialog básico, la IA pareciera tomarselo de forma literal y darme una opción muy simplificada de mi petición:





La IA solo recoge la información en lista, y ni siquiera aplica el contexto de la Notificación. Mi implementación, además de tener control sobre el Theme de la aplicación y el idioma que el usuario prefiere, también tiene un link directo a la modificación hecha:

Pedir a la IA, modificar la misma acción que he hecho, implicaba una inconsistencia de modificaciones que me sumarían mínimo 3+ horas de desarrollo en el mismo componente:

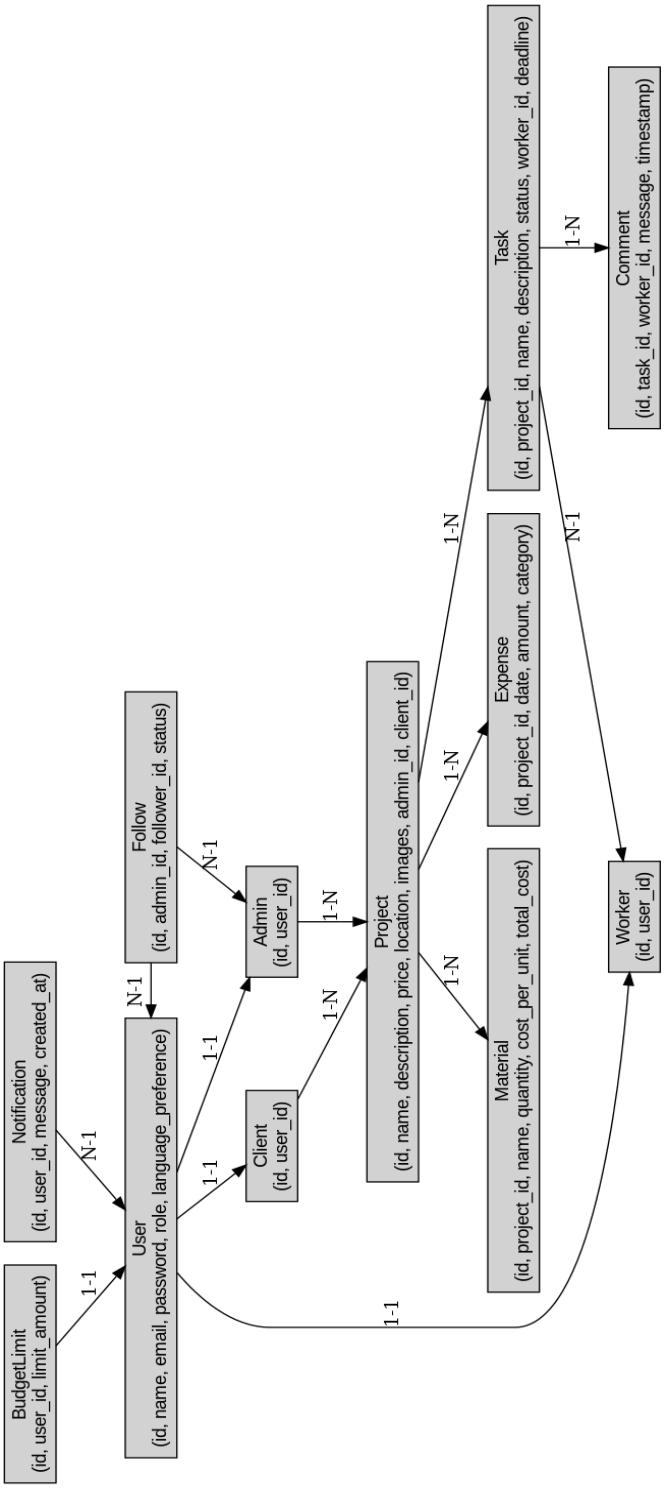


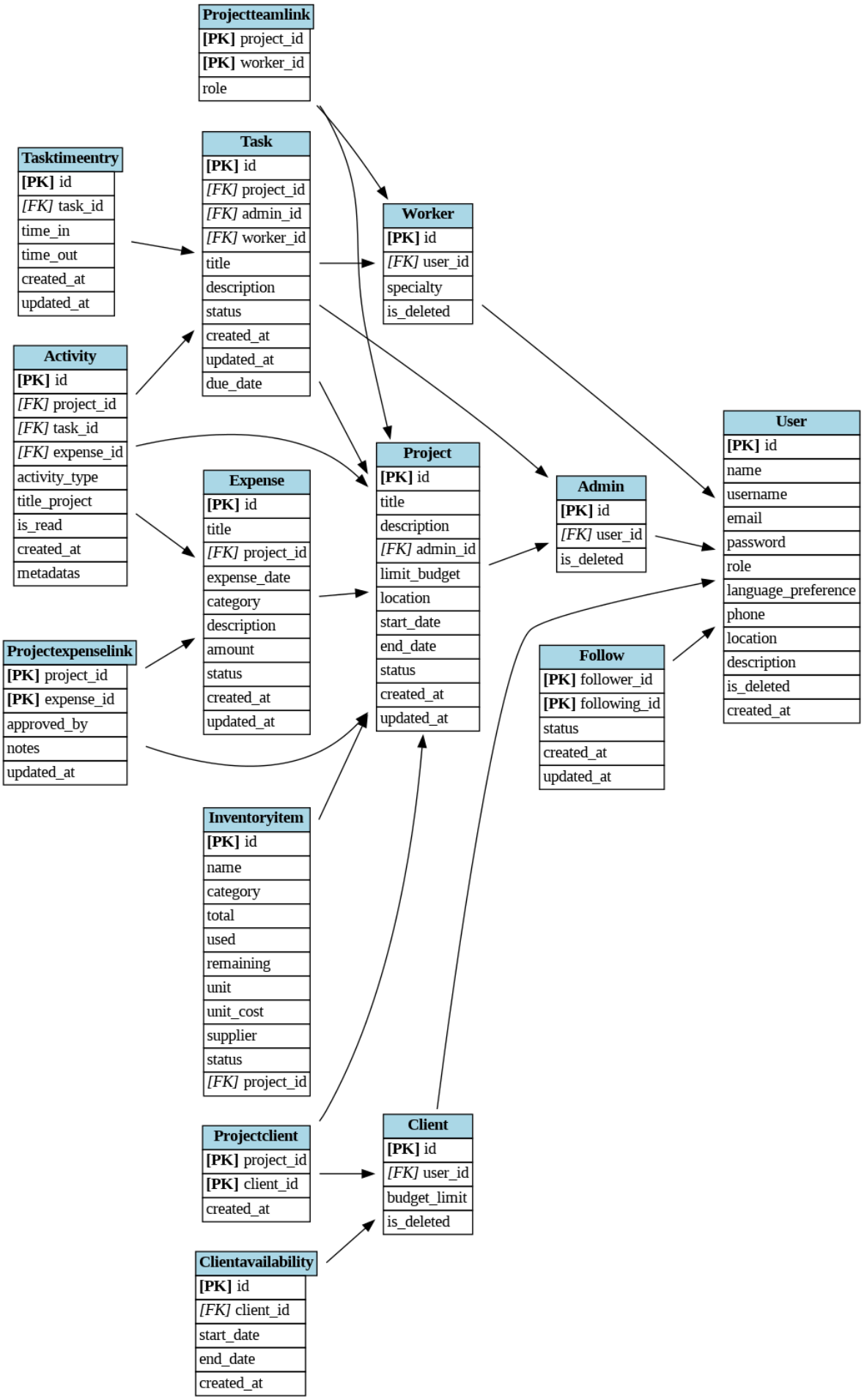
5.3 Generación de la IA

5.3.1 Modelo de Entidad-Relación

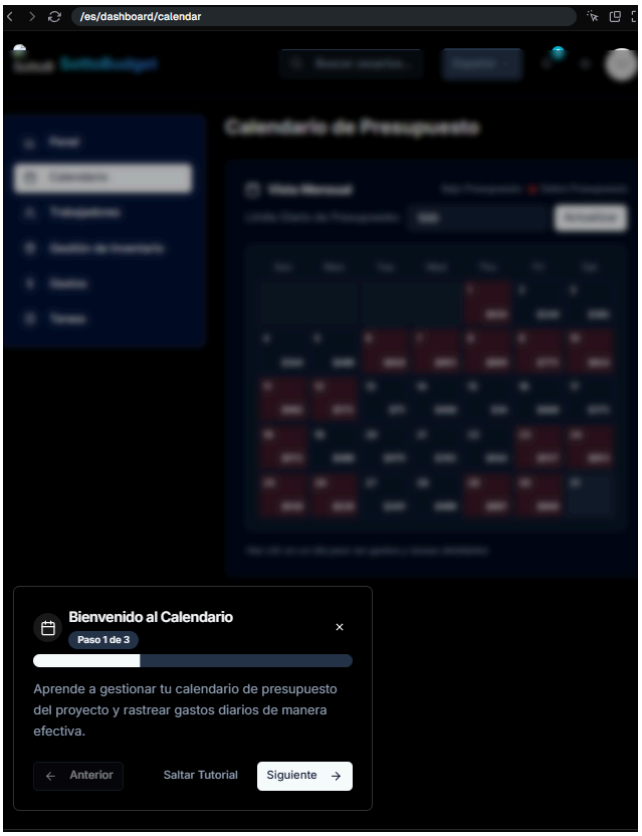
Durante el desarrollo de la Aplicación decidí generar el modelo de Entidad Relación que podría tener mi Backend describiendoselo a la IA de chatGPT y GithubCopilot. Ambos dieron un resultado casi similar, aunque el Copilot no pudo generar un diagrama de verdad, pude observa el diagrama

generado por le ChatGPT.




Projectteamlink
[PK] project_id
[PK] worker_id
role
Tasktimeentry
[PK] id
[FK] task_id
time_in
time_out
created_at
updated_at
Task
[PK] id
[FK] project_id
[FK] admin_id
[FK] worker_id
title
description
status
created_at
updated_at
due_date
Worker
[PK] id
[FK] user_id
specialty
is_deleted
Activity
[PK] id
[FK] project_id
[FK] task_id
[FK] expense_id
activity_type
title_project
is_read
created_at
metadatas
Project
[PK] id
title
description
[FK] admin_id
limit_budget
location
start_date
end_date
status
created_at
updated_at
Admin
[PK] id
[FK] user_id
is_deleted
User
[PK] id
name
username
email
password
role
language_preference
phone
location
description
is_deleted
created_at
Expense
[PK] id
title
[FK] project_id
expense_date
category
description
amount
status
created_at
updated_at
Projectexpenselink
[PK] project_id
[PK] expense_id
approved_by
notes
updated_at
Follow
[PK] follower_id
[PK] following_id
status
created_at
updated_at
Inventoryitem
[PK] id
name
category
total
used
remaining
unit
unit_cost
supplier
status
[FK] project_id
Projectclient
[PK] project_id
[PK] client_id
created_at
Client
[PK] id
[FK] user_id
budget_limit
is_deleted
Clientavailability
[PK] id
[FK] client_id
start_date
end_date
created_at

5.3.2 Tutorial con IA:

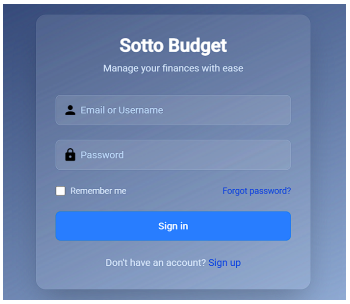


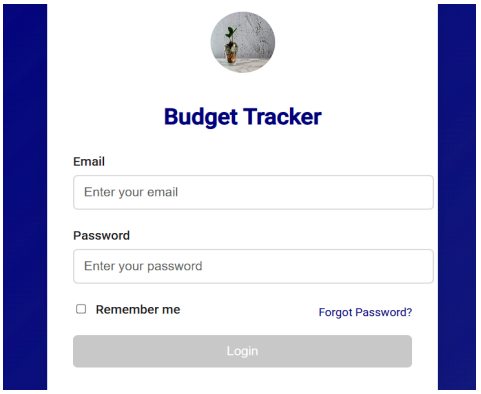
Pedí a la IA que elaborara un Tutorial para el primer login del Usuario, y para mi sorpresa, fue hábil construyendo el Tutorial. Aunque presentó algunos errores menores —como ciertos estilos CSS que no alteraban el estado *blur* al enfocar elementos del documento—, con una revisión manual logré corregirlos sin dificultad.

El componente que propuso fue especialmente útil, ya que me permitió desarrollar a mano el resto de los tutoriales a partir de ese modelo inicial.

Logré ahorrar varias horas de investigación y pruebas, optimizando significativamente el tiempo disponible para el desarrollo de otras partes del proyecto.

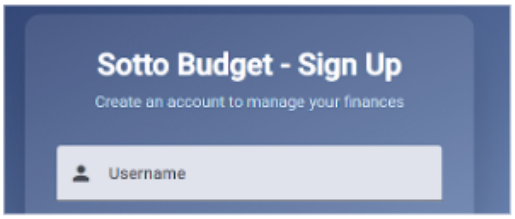
Anexo B:

Mi resultado: 

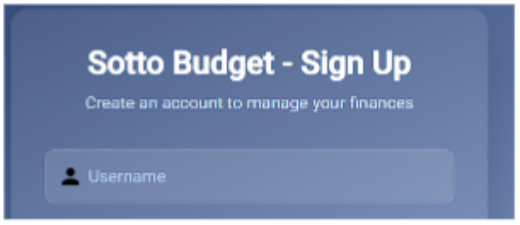
IA: 

Anexo C

Angular Material (IA)

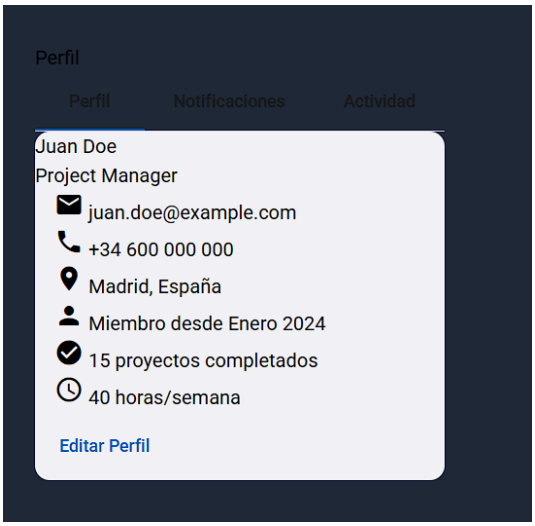


Solución programada manualmente:



Anexo D

Diseño de la IA para el Perfil en Angular 19:

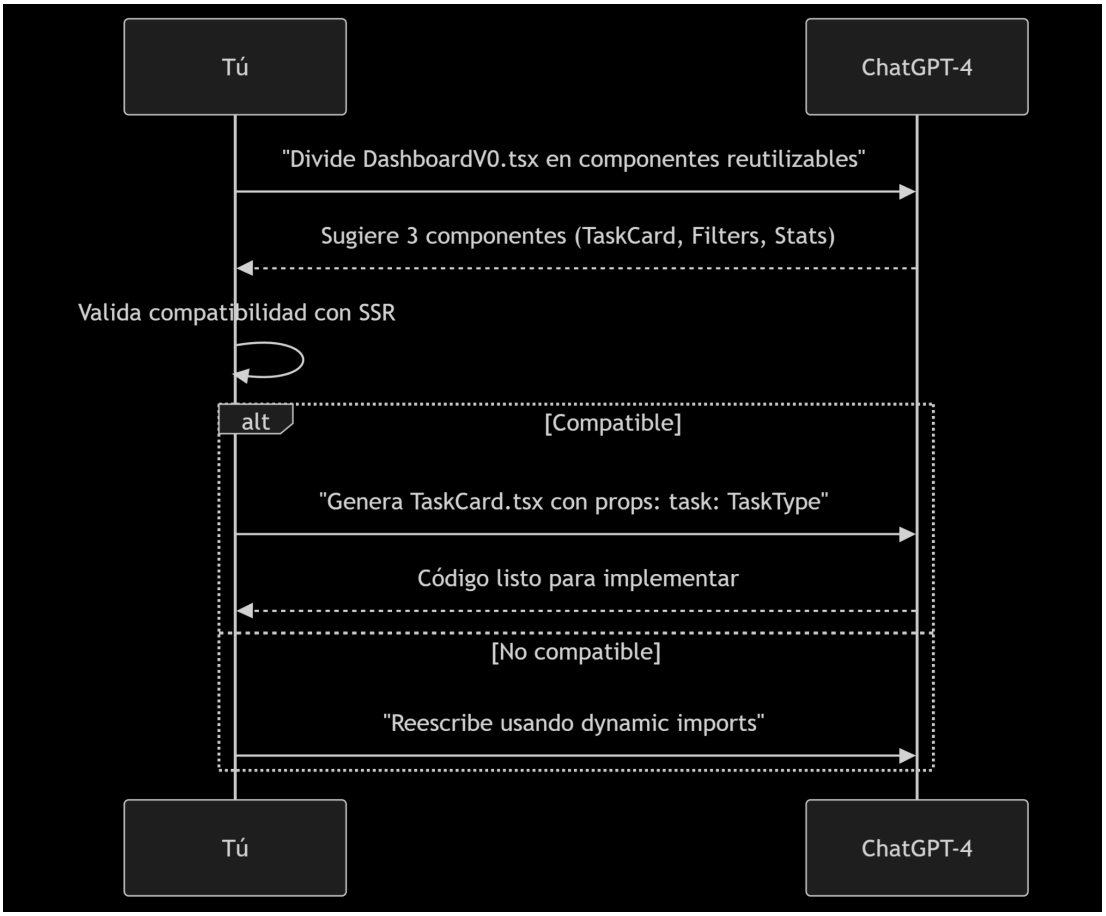


Anexo E

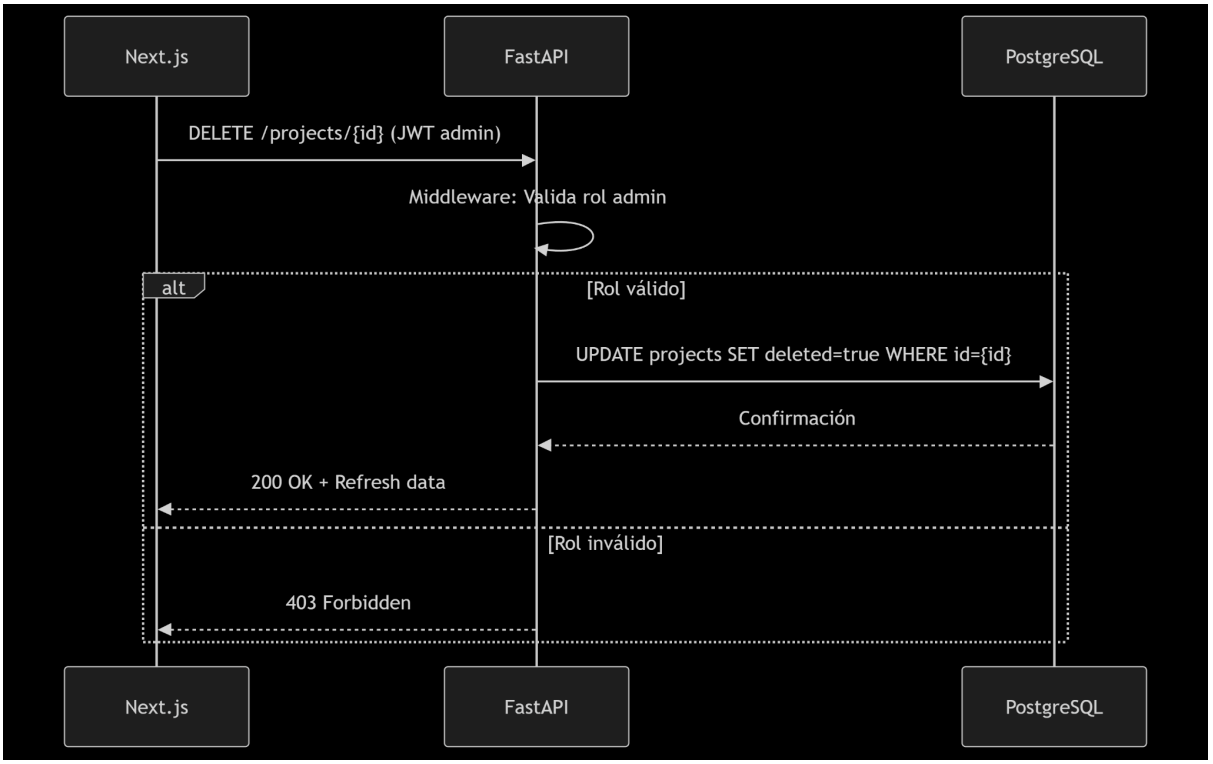
DashBoard ChatGPT:

DashBoard DeepSeek:

Anexo F

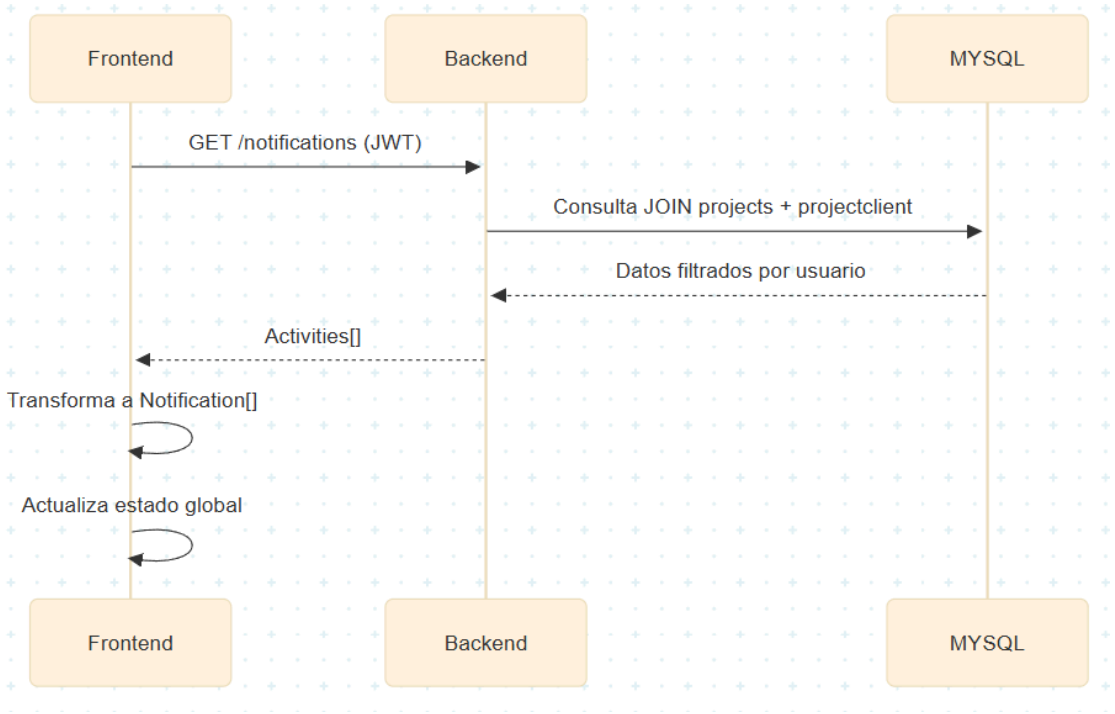


Anexo G

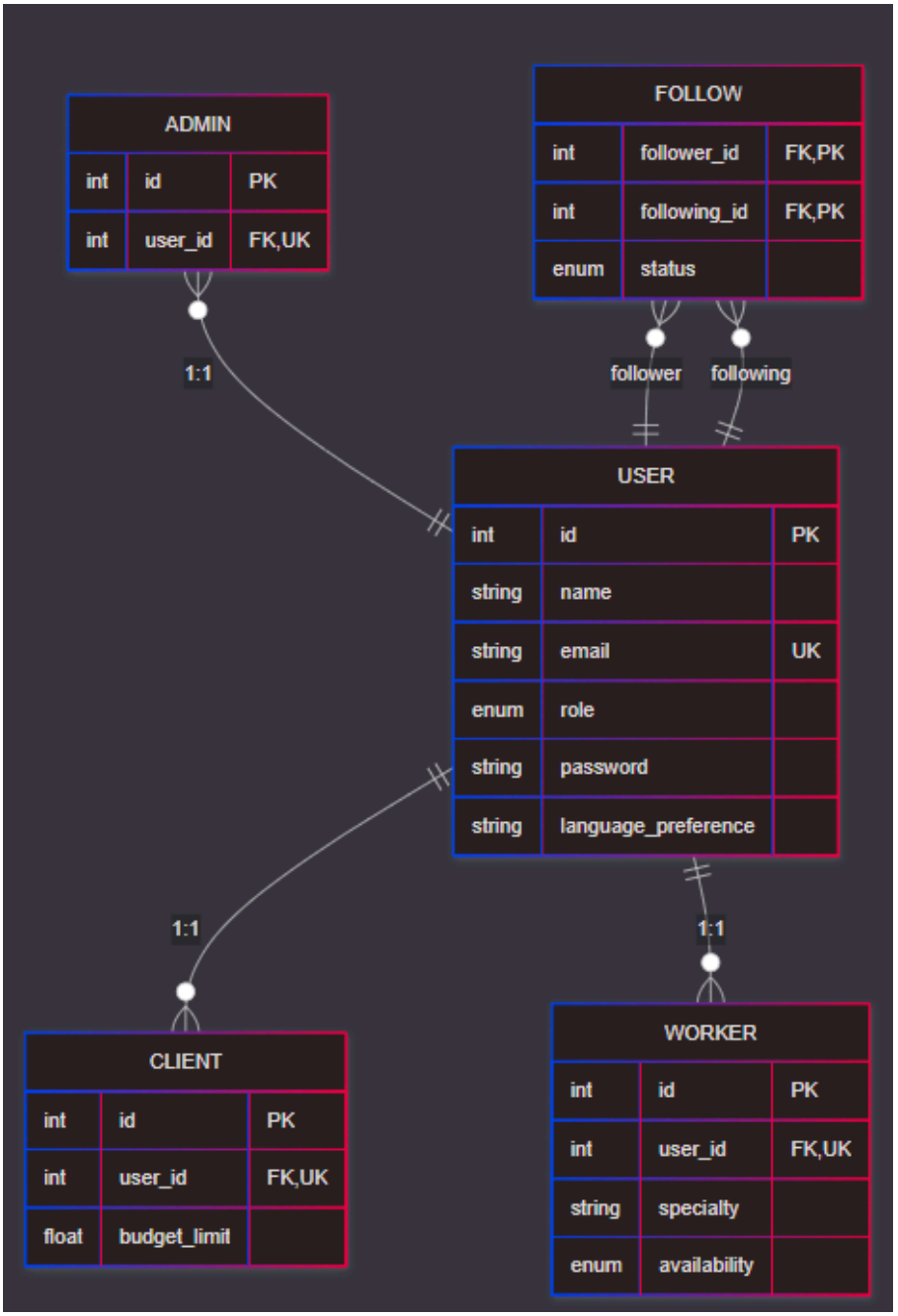


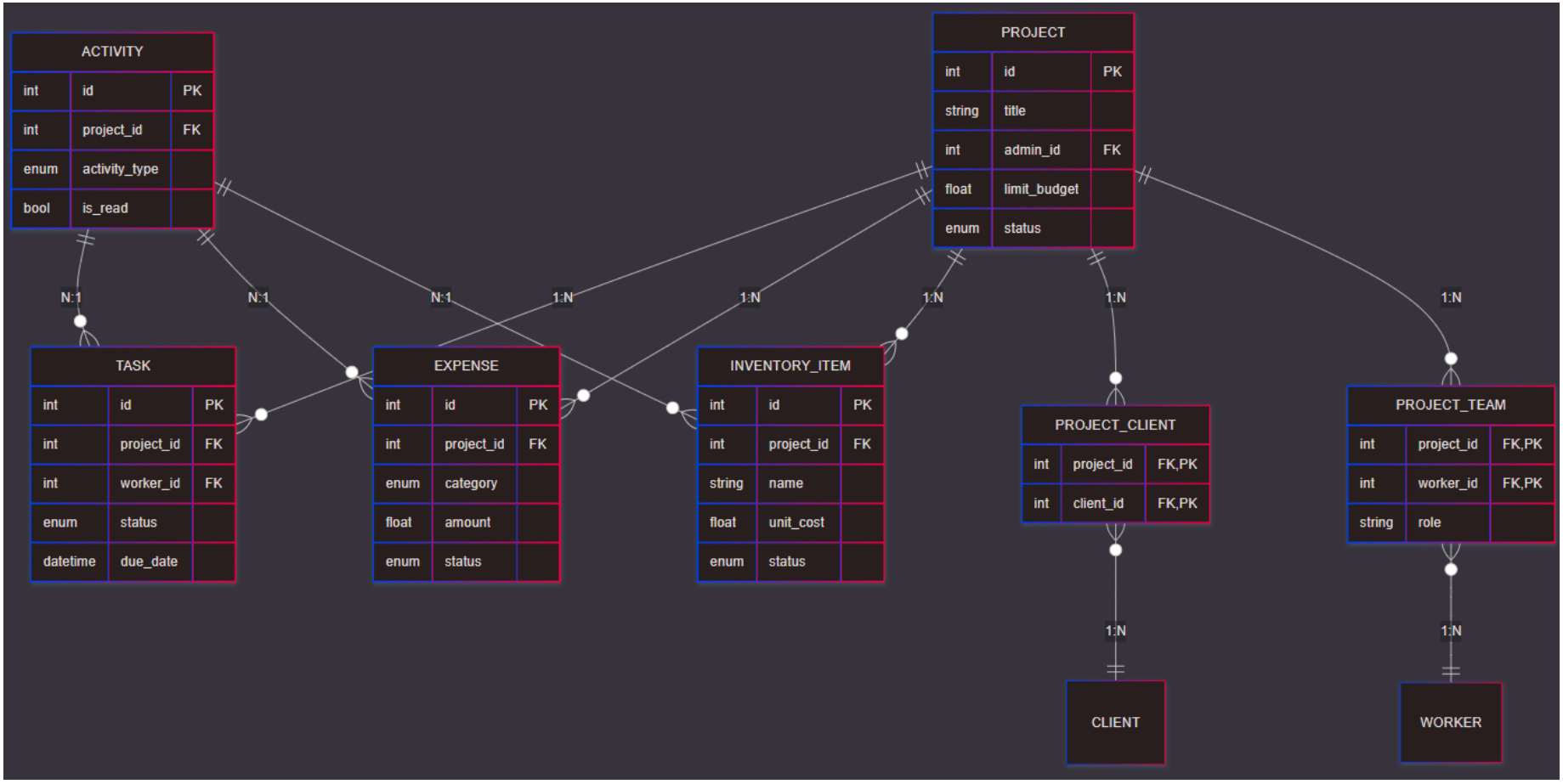
Anexo H

Anexo I


Frontend
Backend
MYSQL
GET /notifications (JWT)
Consulta JOIN projects + projectclient
Datos filtrados por usuario
Activities[]
Transforma a Notification[]
Actualiza estado global
Frontend
Backend
MYSQL

Anexo J


ADMIN
int id PK
int user_id FK,UK
FOLLOW
int follower_id FK,PK
int following_id FK,PK
enum status
1:1
follower
following
USER
int id PK
string name
string email UK
enum role
string password
string language_preference
1:1
1:1
CLIENT
int id PK
int user_id FK,UK
float budget_limit
WORKER
int id PK
int user_id FK,UK
string specialty
enum availability


ACTIVITY
int id PK
int project_id FK
enum activity_type
bool is_read
PROJECT
int id PK
string title
int admin_id FK
float limit_budget
enum status
TASK
int id PK
int project_id FK
int worker_id FK
enum status
datetime due_date
EXPENSE
int id PK
int project_id FK
enum category
float amount
enum status
INVENTORY_ITEM
int id PK
int project_id FK
string name
float unit_cost
enum status
PROJECT_CLIENT
int project_id FK,PK
int client_id FK,PK
PROJECT_TEAM
int project_id FK,PK
int worker_id FK,PK
string role
CLIENT
WORKER
N:1
1:N

Anexo K

User Story 6: Be Assigned to a Task

As a Worker,

I want to be assigned to tasks by the Admin I follow,

So that I can see my work responsibilities and deadlines.

Acceptance Criteria:

- **Task Assignment by Admin:**
- Only an Admin can assign a task to a **Worker** who follows them.
- The Admin must be able to search and select a follower when creating/editing a task.

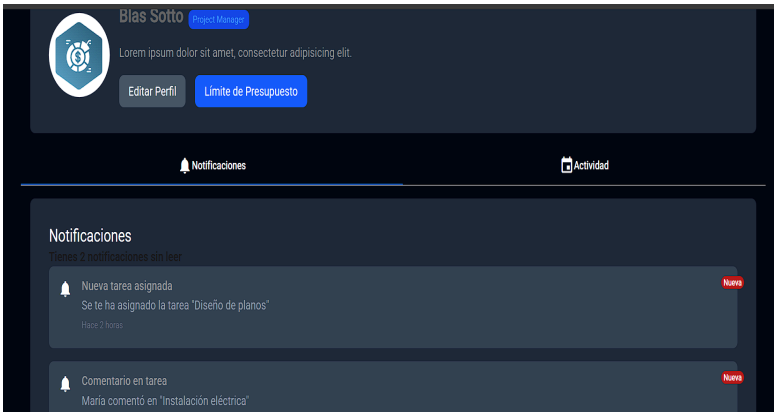
- **Worker View of Assigned Tasks:**
- Once assigned, the Worker should be able to see **all their tasks** in a dedicated section.
- Each task should display its **deadline, project association, and task details**.

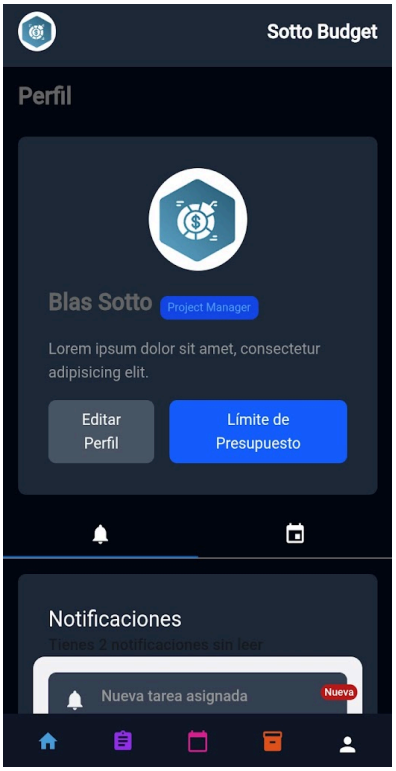
- **Task Progress Management:**
- The Worker can update the task status (`Pending`, `In Progress`, `Completed`).
- If a Worker completes a task, the Admin and Client associated with the project receive a notification.

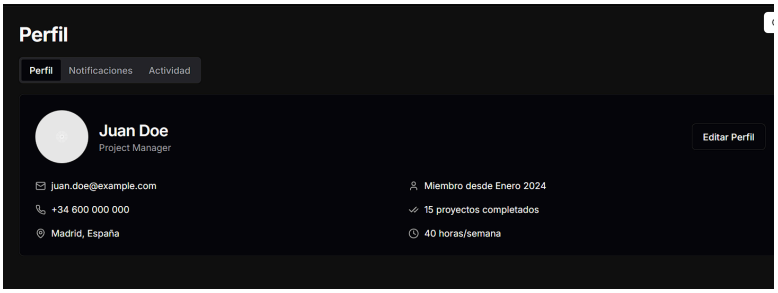
- **Notifications:**
- The Worker receives a notification when they are assigned to a new task.

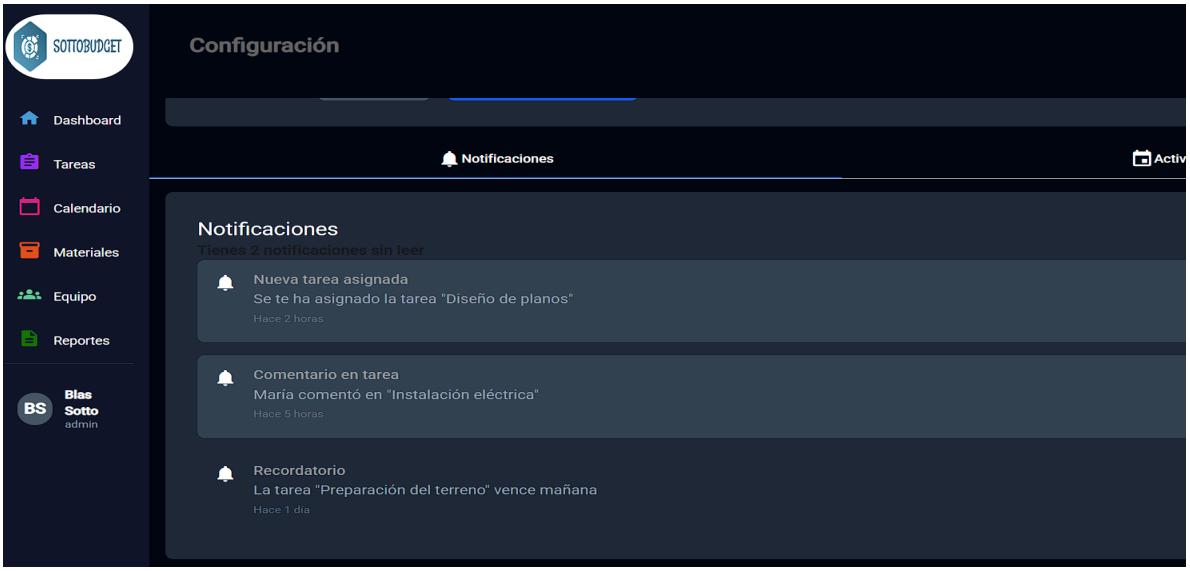
- **Restrictions:**
- Workers **cannot** assign themselves to tasks, only the Admin has this permission.
- If a Worker unfollows an Admin, they **lose access** to tasks assigned by that Admin.

Sección 1

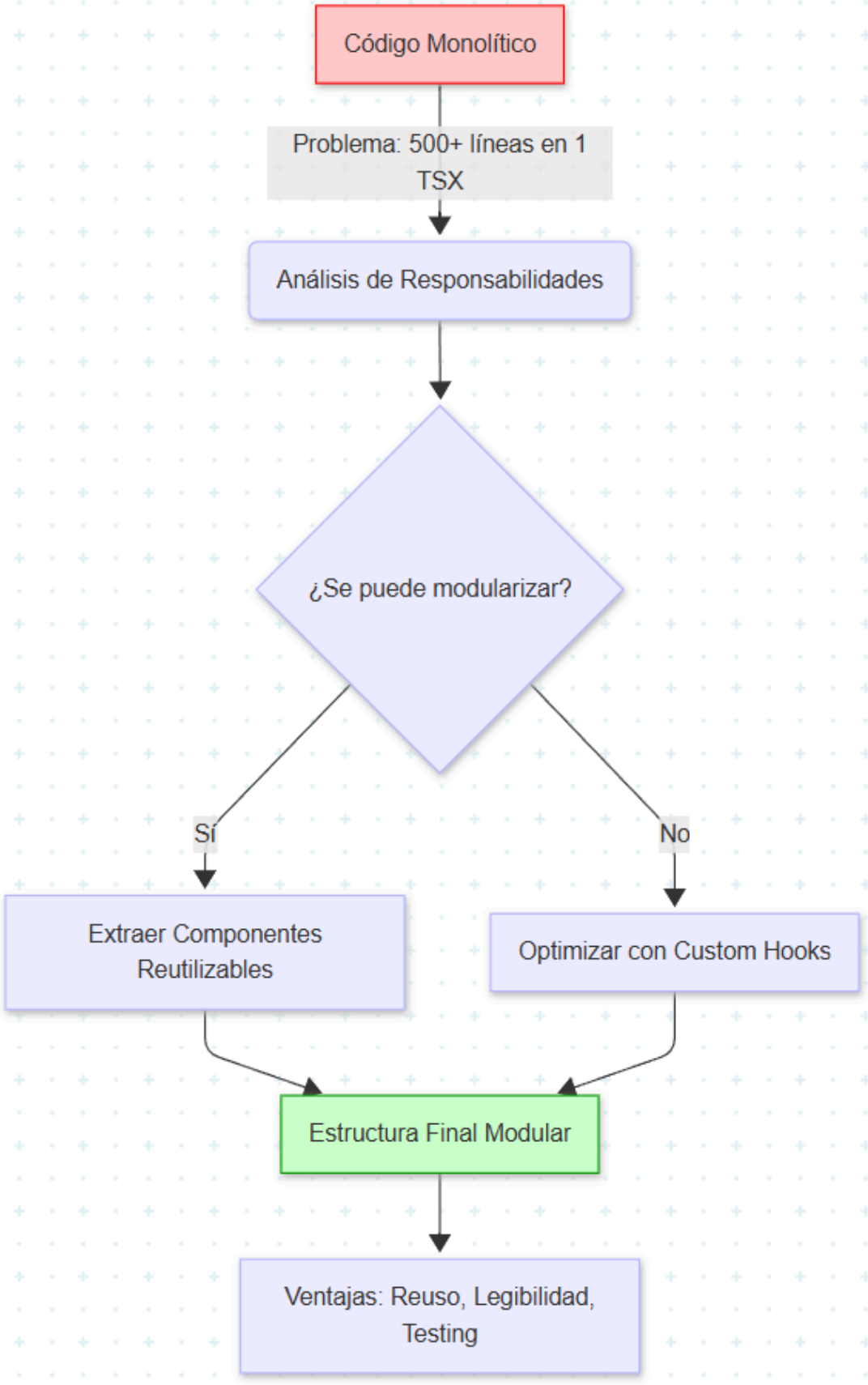

Blas Sotto
Project Manager
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Editar Perfil
Límite de Presupuesto
Notificaciones
Actividad
Notificaciones
Tienes 2 notificaciones sin leer
Nueva tarea asignada
Se te ha asignado la tarea "Diseño de planos"
Hace 2 horas
Nueva
Comentario en tarea
María comentó en "Instalación eléctrica"
Nueva


Sotto Budget
Perfil
Blas Sotto
Project Manager
Lorem ipsum dolor sit amet, consectetur adipisicing elit.
Editar Perfil
Límite de Presupuesto
Notificaciones
Tienes 2 notificaciones sin leer
Nueva tarea asignada
Nueva

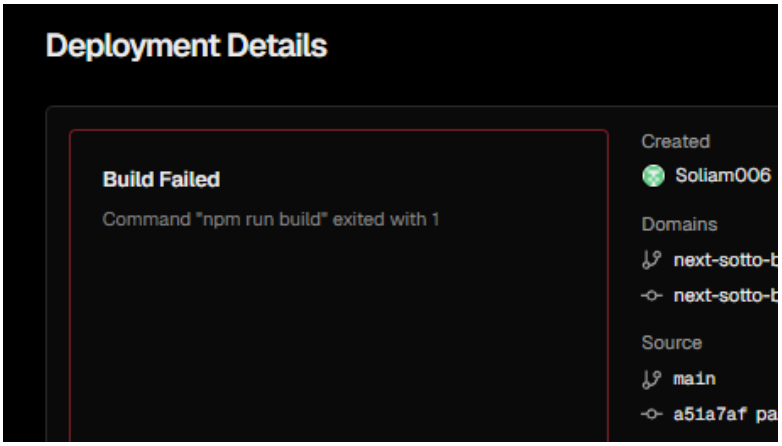

Perfil
Perfil
Notificaciones
Actividad
Juan Doe
Project Manager
Editar Perfil
juan.doe@example.com
+34 600 000 000
Madrid, España
Miembro desde Enero 2024
15 proyectos completados
40 horas/semana

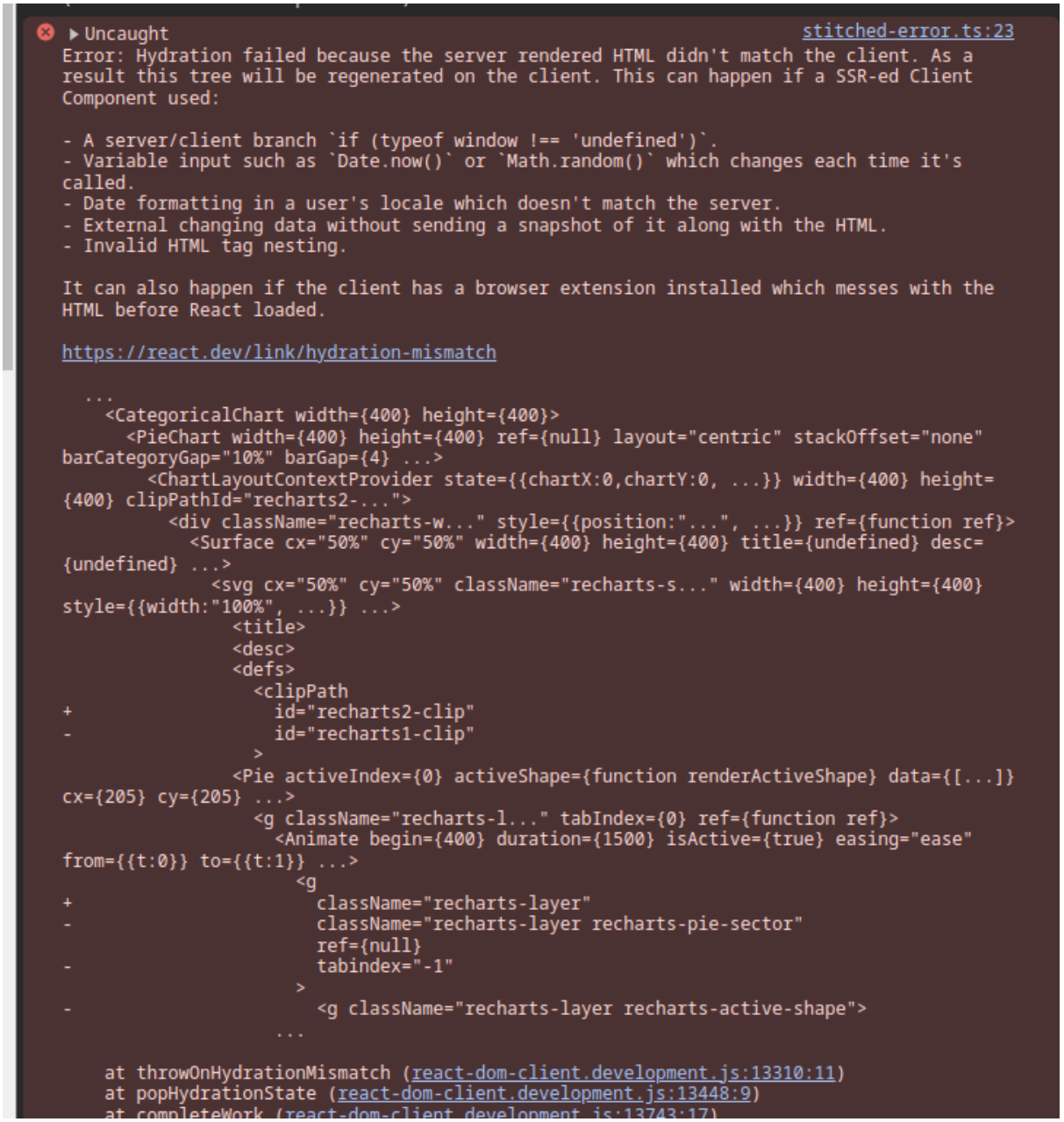

SOTTOBUDGET
Dashboard
Tareas
Calendario
Materiales
Equipo
Reportes
BS
Blas Sotto
admin
Configuración
Notificaciones
Activ
Notificaciones
Tienes 2 notificaciones sin leer
Nueva tarea asignada
Se te ha asignado la tarea "Diseño de planos"
Hace 2 horas
Comentario en tarea
María comentó en "Instalación eléctrica"
Hace 5 horas
Recordatorio
La tarea "Preparación del terreno" vence mañana
Hace 1 día

Sección 2

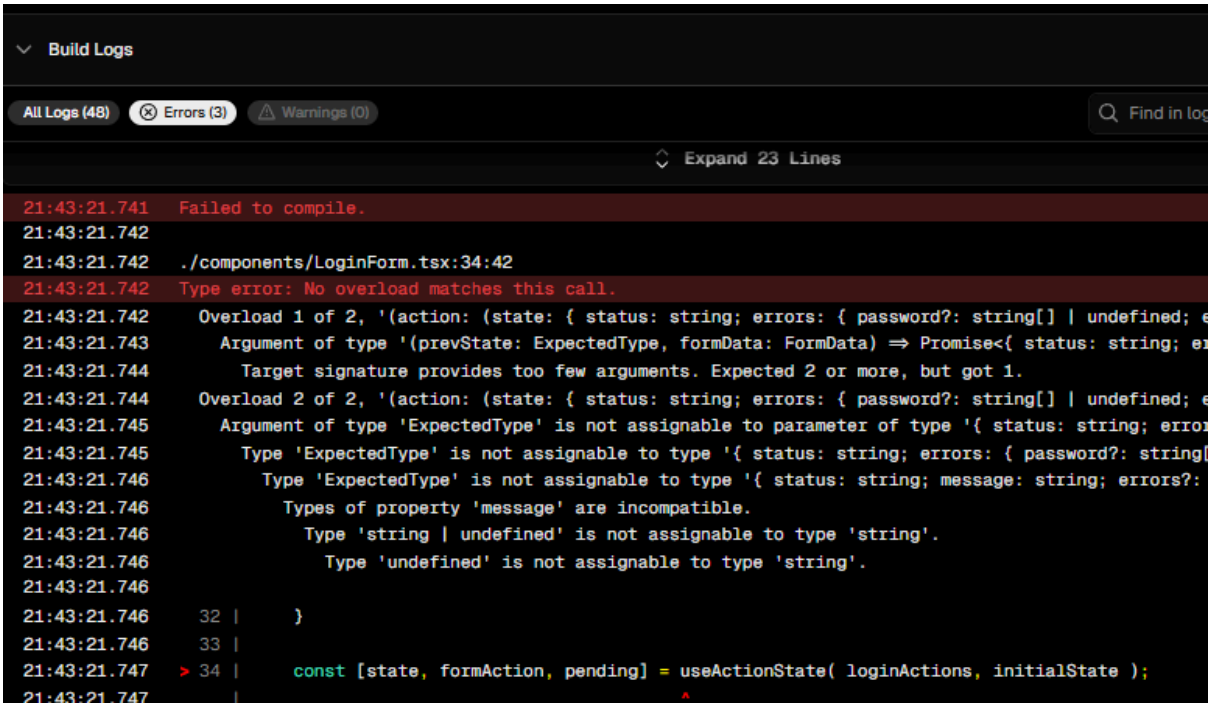

Código Monolítico
Problema: 500+ líneas en 1 TSX
Análisis de Responsabilidades
¿Se puede modularizar?
Sí
No
Extraer Componentes Reutilizables
Optimizar con Custom Hooks
Estructura Final Modular
Ventajas: Reuso, Legibilidad, Testing

Sección 3


Deployment Details
Build Failed
Command "npm run build" exited with 1
Created
Soliam006
Domains
next-sotto-b
next-sotto-b
Source
main
a51a7af pa


Uncaught
stitched-error.ts:23
Error: Hydration failed because the server rendered HTML didn't match the client. As a result this tree will be regenerated on the client. This can happen if a SSR-ed Client Component used:
- A server/client branch `if (typeof window !== 'undefined')`.
- Variable input such as `Date.now()` or `Math.random()` which changes each time it's called.
- Date formatting in a user's locale which doesn't match the server.
- External changing data without sending a snapshot of it along with the HTML.
- Invalid HTML tag nesting.
It can also happen if the client has a browser extension installed which messes with the HTML before React loaded.
https://react.dev/link/hydration-mismatch
<CategoricalChart width={400} height={400}>
<PieChart width={400} height={400} ref={null} layout="centric" stackOffset="none" barCategoryGap="10%" barGap={4} ...>
<ChartLayoutContextProvider state={{chartX:0,chartY:0, ...}} width={400} height={400} clipPathId="recharts2-...">
<div className="recharts-w..." style={{position:"...", ...}} ref={function ref}>
<Surface cx="50%" cy="50%" width={400} height={400} title={undefined} desc={undefined} ...>
<svg cx="50%" cy="50%" width={400} height={400} style={{width:"100%", ...}} ...>
<title>
<desc>
<defs>
<clipPath
+ id="recharts2-clip"
- id="recharts1-clip"
>
<Pie activeIndex={0} activeShape={function renderActiveShape} data={[...]} cx={205} cy={205} ...>
<g className="recharts-l..." tabIndex={0} ref={function ref}>
<Animate begin={400} duration={1500} isActive={true} easing="ease" from={{t:0}} to={{t:1}} ...>
<g
+ className="recharts-layer"
- className="recharts-layer recharts-pie-sector"
ref={null}
- tabindex="-1"
>
- <g className="recharts-layer recharts-active-shape">
...
at throwOnHydrationMismatch (react-dom-client.development.js:13310:11)
at popHydrationState (react-dom-client.development.js:13448:9)
at completeWork (react-dom-client.development.js:13743:17)

```
string[] | undefined; password?: string[] | undefined; username?: string[] | undefined; phone?: string[] | undefined; countryCode?: string

        <div className="space-y-2">
                    <Label htmlFor="name">{t.fullName}</Label>
                    <Input id="name" name="name" placeholder="John Doe" aria-invalid={!!state?.errors?.name} />
                                                                                                          ^
                    {state?.status === "error" && state.errors?.name && (
                                <p className="text-xs text-red-500">{state.errors.name[0]}</p>
                    )}
```



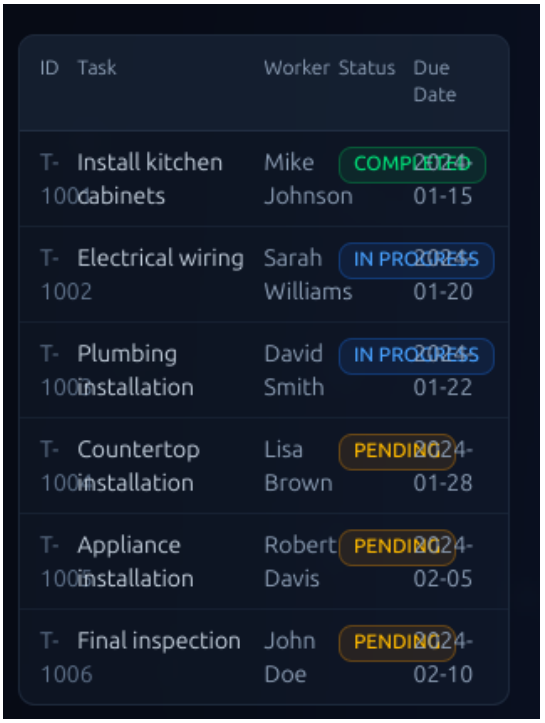
```
Build Logs

All Logs (48)   Errors (3)   Warnings (0)                                   Find in log

                          Expand 23 Lines

21:43:21.741  Failed to compile.
21:43:21.742
21:43:21.742  ./components/LoginForm.tsx:34:42
21:43:21.742  Type error: No overload matches this call.
21:43:21.742    Overload 1 of 2, '(action: (state: { status: string; errors: { password?: string[] | undefined; e
21:43:21.743      Argument of type '(prevState: ExpectedType, formData: FormData) => Promise<{ status: string; er
21:43:21.744        Target signature provides too few arguments. Expected 2 or more, but got 1.
21:43:21.744    Overload 2 of 2, '(action: (state: { status: string; errors: { password?: string[] | undefined; e
21:43:21.745      Argument of type 'ExpectedType' is not assignable to parameter of type '{ status: string; erro
21:43:21.745        Type 'ExpectedType' is not assignable to type '{ status: string; errors: { password?: string[
21:43:21.746          Type 'ExpectedType' is not assignable to type '{ status: string; message: string; errors?:
21:43:21.746            Types of property 'message' are incompatible.
21:43:21.746              Type 'string | undefined' is not assignable to type 'string'.
21:43:21.746                Type 'undefined' is not assignable to type 'string'.
21:43:21.746
21:43:21.746     32 |     }
21:43:21.746     33 |
21:43:21.747   > 34 |     const [state, formAction, pending] = useActionState( loginActions, initialState );
21:43:21.747        |                                        ^
```

Sección 4

Errores de código constantes por culpa de la mezcla de versiones (Angular 19):

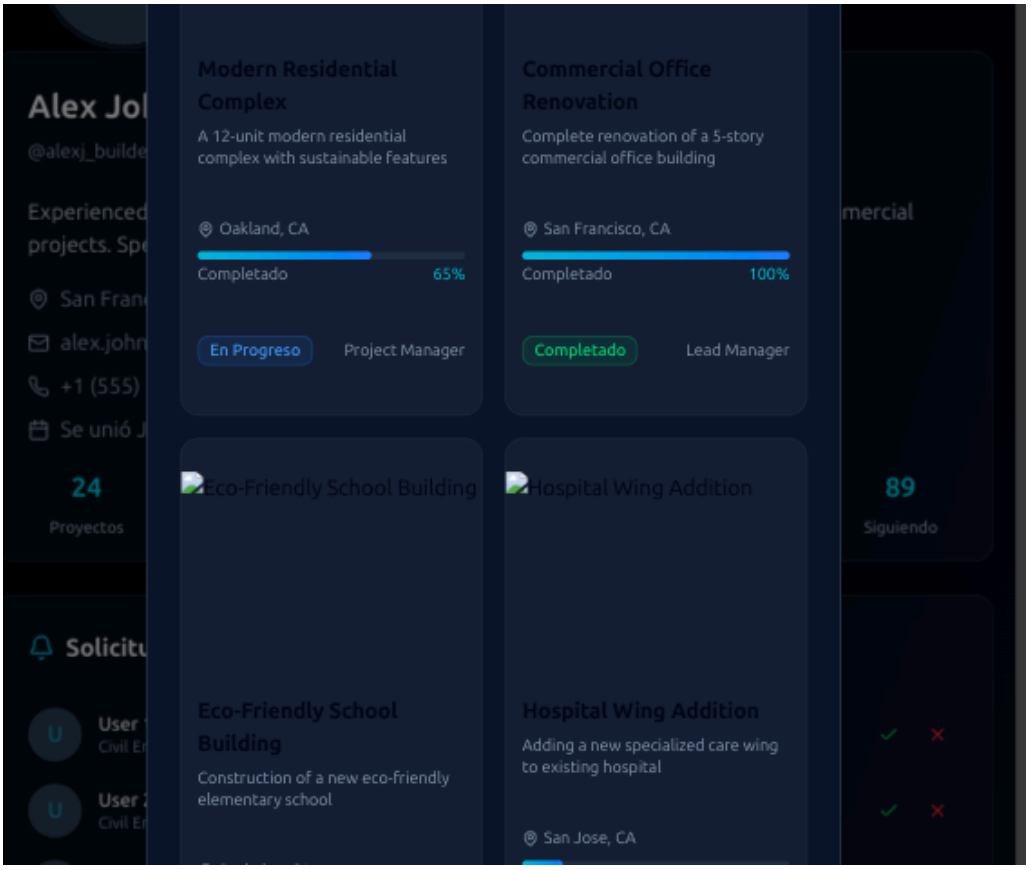
```
<CardContent>
  <form action={formAction} className="space-y-4">
    <div className="space-y-2">
      <Label htmlFor="name">{t.fullName}</Label>
      <Input id="name" name="name" placeholder="John Doe" aria-invalid={!!state?.errors?.name} />
      {state?.errors?.name && <p className="text-xs text-red-500">{state.errors.name[0]}</p>}
    </div>

    <div className="space-y-2">
      <Label htmlFor="username">{t.username}</Label>
      <Input id="username" name="username" placeholder="johndoe" aria-invalid={!!state?.errors?.username} />
      {state?.errors?.username && <p className="text-xs text-red-500">{state.errors.username[0]}</p>}
    </div>
```

Diseño Responsive que no es funcional

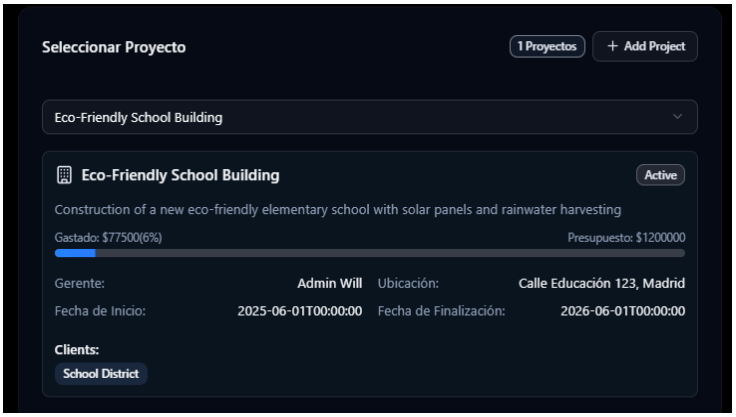


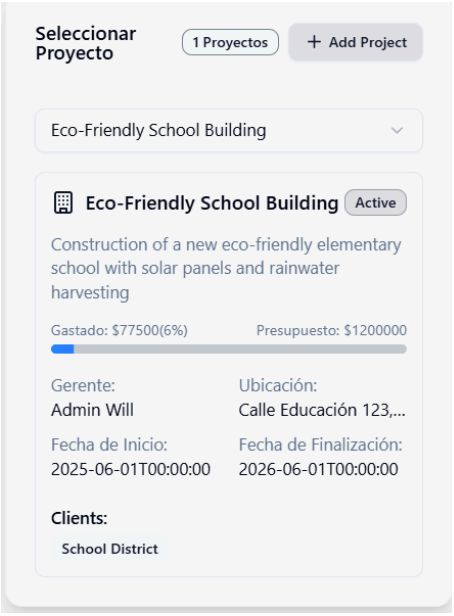
No funcional por librerías incompatibles

La IA puede dar una visualización agradable, pero no es funcional, reduciendo la productividad:



Sección 5





Sección 6

Cuestionario aplicado

1. ¿Actualmente usas herramientas de IA (ChatGPT, Copilot, DeepSeek, etc.) en tu trabajo como desarrollador?

- Sí
- No

2. ¿Qué papel consideras que tiene la IA en tu flujo de trabajo actual?

- Es una herramienta asistente que acelera tareas
- Es útil pero no confiable para tareas importantes
- Puede reemplazar parte significativa del trabajo humano
- No aporta valor real

3. ¿Has notado un incremento en tu productividad desde que integras IA en tus tareas de desarrollo?

- Sí, significativamente
- Sí, pero solo en tareas repetitivas
- No he notado diferencia
- No, ha sido contraproducente

4. ¿Qué tipo de tareas automatizas o aceleras más con IA? *(Puedes marcar más de una)*

- Generación de código básico (CRUD, formularios)
- Refactorización o limpieza de código
- Generación de documentación
- Solución de errores
- Planificación de arquitectura
- Otras: _____

5. ¿Consideras que la IA puede reemplazar tareas críticas como la arquitectura o la toma de decisiones técnicas?

- Sí
- Parcialmente
- No, en absoluto

6. ¿Has experimentado un aumento en el tiempo dedicado a debugging por errores introducidos por la IA?

- Sí, frecuentemente
- Sí, ocasionalmente
- No

- No uso código generado por IA

7. ¿Qué tan confiable consideras el código generado por IA en términos de calidad y mantenibilidad?

- Muy confiable
- Aceptable con revisión
- Poco confiable
- No lo uso

8. ¿En qué áreas crees que la IA todavía necesita mejorar para ser realmente útil? *(Pregunta abierta)*