Universidad Europea VALENCIA

GRADO EN CIENCIA DE DATOS

APLICACIÓN WEB PARA LA GESTIÓN DIARIA DE TAREAS CON REGISTRO DE HORAS

Presentado por:

PEDRO TORRIJOS ADIEGO

Dirigido por:

ALEJANDRO PERDIGUERO O LEARY

CURSO ACADÉMICO 2024-2025

Tabla de contenido

Resumen 3
Abstract 5
Glosario 7
1. Antecedentes 9
1.1 Descripción del problema o necesidad 9
1.2 Contexto del proyecto 9
1.3 Análisis de la situación actual 10
1.4 Justificación del proyecto 11
2. Objetivos del proyecto 14
3. Marco teórico 16
3.1 Gestión de tareas y productividad personal 16
3.2 Revisión de aplicaciones de productividad existentes 17
3.3 Experiencia de usuario (UX/UI) y eficiencia en diseño 20
4. Metodología 23
4.1 Enfoque de desarrollo iterativo y ágil 23
4.2 Herramientas y entorno de desarrollo 23
4.3 Arquitectura de la aplicación 24
4.4 Estructura de archivos del proyecto 27
5. Desarrollo de la aplicación 32
5.1 Implementación del Front-end (HTML, CSS, JS) 32
5.2 Implementación del Back-end (Node.js, Express) 41
5.3 Integración con MongoDB Atlas 42
5.4 Dificultades encontradas y soluciones 43
6. Resultados 45
6.1 Funcionalidades implementadas 45
6.2 Capturas de la aplicación 48
6.3 Validación del cumplimiento de objetivos 47
7. Discusión 48
8. Conclusiones 53
9. Futuras líneas de proyecto 54
10. Objetivos de Desarrollo Sostenible (ODS) 56
11. Referencias 57

Tabla de Ilustraciones

Figura 3. 1: Gráfico de ejemplo que muestra las horas totales dedicadas a cada tarea completada en un periodo. En la aplicación, este gráfico se actualiza dinámicamente conforme se completan tareas y se acumulan horas. 17
Figura 4. 1: Esquema de la arquitectura de la aplicación. El cliente (navegador web) carga la interfaz HTML/JS y se comunica con el servidor Node.js (Express) mediante peticiones HTTP para leer/escribir tareas y generar informes. El servidor persiste los datos en un archivo local tasks.json y realiza copias de seguridad en MongoDB Atlas. Asimismo, integra la API de OpenAI para generar el resumen de informes, y utiliza Puppeteer para devolverlos en PDF cuando el usuario lo solicita. 24

Figura 4. 2: Estructura de carpetas del proyecto, mostrando los archivos principales y directorios. 27
Figura 4. 3: Pantalla principal de la aplicación mostrando una tarea en edición, con sus campos de notas y ERP. 30
Figura 5. 1: Lista de tareas pendientes con el total de horas de las tareas durante el dia actual. 32

Figura 5. 2: Lista de tareas pendientes de revisión. 33
Figura 5. 3: Recuadro de texto donde se escribe las notas para cada tarea seleccionada.33
Figura 5. 4: Recuadro de texto donde se escribe la imputación de las horas de la tarea seleccionada. 34
Figura 5. 5: Pantalla de resumen de tareas realizadas durante el dia actual y sus horas correspondientes, añadiendo el texto imputado a la tarea. 34
Figura 5. 6: Menú de selección de tipo de informe. 35
Figura 5. 7: sección para generar el informe y descarar el informe en un PDF: 35
Figura 5. 8: Ejemplo de listado de tareas completadas e histórico de movimiento. 35
Figura 5. 9: Grafico de total de horas por tarea finalizada. 36
Figura 5. 10: Ejemplo de una tarea que tiene una anotación. 37
Figura 5. 11: Estructura de almacenamiento del testo de imputaciones de horas en una tarea. 37
Figura 5. 12: Ejemplo de estructura de una nueva tarea. 38
Figura 5. 13:Ejemplo de la estructura de una tarea en estado de completado. 38
Figura 5. 14: Ejemplo de la estructura de una tarea en estado de revisión. 39
Figura 5. 15: Ejemplo de resumen de reporte semanal mostrado en el sistema. 40
Figura 5. 16: Ejemplos de informe generado semanal en formato PDF. 40
Figura 5. 17: Captura de imagen de los datos almacenados en el clúster de MongoDB. .. 43
Figura 6. 1: Vista principal de la aplicación mostrando la lista de tareas pendientes, una tarea seleccionada con su nota y ERP, y el historial con una tarea completada 49

Figura 6. 2: Informe generado en pantalla, mostrando el texto resumen de tareas de la semana actual. 50
Figura 6. 3: Página de ejemplo del informe PDF generado, con el resumen de tareas y la lista de pendientes 50

Resumen

La gestión eficaz del tiempo y las tareas es un desafío clave en la productividad. La expansión del teletrabajo y la colaboración en línea, acelerada por la pandemia de 2020(Organización Internacional del Trabajo, 2020), ha aumentado el uso de herramientas digitales para organizar el trabajo diario. Sin embargo, abundan las situaciones de ineficiencia derivadas de una mala organización: empleados que invierten decenas de horas en actividades poco productivas (reuniones improductivas, descoordinación, duplicación de esfuerzos, etc.), generando estrés y reduciendo el rendimiento (Macedo, 2025). Paralelamente, existen múltiples aplicaciones especializadas –gestores de tareas, herramientas de control horario, sistemas ERP– pero *carece una solución unificada* y accesible que combine **gestión de tareas, registro de horas y anotaciones** en un solo entorno. Este proyecto surge ante esa necesidad no cubierta en profesionales independientes, estudiantes y pequeñas organizaciones que buscan optimizar su productividad sin incurrir en altos costos ni complejidad.

Se ha desarrollado una **aplicación web** unificada que integra las tres funciones mencionadas (tareas + horas + notas). La solución se ha implementado utilizando tecnologías web modernas: Frontend web (HTML/CSS/JavaScript) para la interfaz interactiva y back-end con Node.js y el framework Express para gestionar la lógica del servidor. En una etapa inicial se usó almacenamiento local JSON por simplicidad, migrando posteriormente a una base de datos NoSQL (MongoDB Atlas) para garantizar persistencia y escalabilidad en la nube. La metodología de desarrollo fue ágil (Scrum/Kanban), con iteraciones cortas que permitieron refinar algunas funcionalidades según cada iteración. Entre las características técnicas se incluyó la visualización de datos (gráficos de tiempo dedicado por tarea mediante la librería Chart.js y la integración de APIs externas, como la API de OpenAI, para incorporar la generación automática de informes diarios mediante IA.

El proyecto ha dado como resultado un prototipo funcional de una aplicación web de productividad “todo en uno”. La herramienta permite crear y organizar tareas en listas o tableros, moverlas entre estados (pendiente, en progreso, completada), registrar el tiempo trabajado en cada tarea mediante un y almacenar notas o datos asociados (por ejemplo, códigos de proyecto o incidencias para un ERP corporativo). Los datos se almacenan en la nube, de modo que pueden consultarse en cualquier momento y lugar. Con la aplicación un mismo usuario pudo planificar su jornada, cuantificar las horas invertidas en cada actividad y obtener al final del día un reporte unificado de su trabajo. Lo que nos puede permitir tener mejoras en eficiencia y organización: la información antes dispersa en varias aplicaciones ahora está centralizada, reduciendo la duplicación de registros y facilitando la supervisión del progreso diario. Asimismo, la aplicación ofrece gráficos resumen del reparto de horas, lo que aporta visibilidad inmediata sobre en qué tareas se invierte más tiempo.

La solución propuesta nos muestra la viabilidad de una herramienta unificada de gestión personal de trabajo. En comparación con herramientas aisladas (p. ej., Trello o Asana solo para tareas, Clockify solo para tiempo), la aplicación integrada simplifica el flujo de trabajo al eliminar la necesidad de alternar entre múltiples sistemas. Esto puede traducirse en ahorros de tiempo y una curva de aprendizaje más suave para usuarios no técnicos. La arquitectura basada en tecnologías web estándar y servicios en la nube sugiere un buen potencial de escalabilidad y mantenimiento a largo plazo.

Palabras Clave: Teletrabajo, Control horario, Kanban, JavaScript full-stack, Inteligencia Artificial

Abstract

Effective task and time management is a critical challenge for productivity in modern work environments. The rise of remote work and online collaboration – accelerated by the 2020 pandemic (Organización Internacional del Trabajo, 2020)– has led to increased reliance on digital tools to organize daily work. However, inefficiencies due to poor organization remain common: knowledge workers can waste many hours per month in unproductive activities (inefficient meetings, miscommunication, duplicated tasks, etc.), causing stress and lower performance(Macedo, 2025). At the same time, numerous specialized applications exist –task management apps, time-tracking tools, enterprise resource planning (ERP) systems– yet there is *no unified solution* that combines task management, time logging, and ERP-style notes in one accessible platform. This project addresses that unmet need, targeting freelancers, students, and small teams who seek to optimize productivity without high costs or complexity.

We developed a unified **web application** integrating the three mentioned functions (tasks + time + ERP notes) in a single interface. The solution was implemented with modern web technologies: a front-end built with standard web languages (HTML, CSS, JavaScript) for an interactive user interface, and a back-end using Node.js with the Express framework to handle server logic and API routes. Initially, simple JSON files were used for data storage during prototyping; later, the project migrated to a cloud NoSQL database (MongoDB Atlas) to ensure persistent, secure, and scalable data management. The development followed an agile methodology (Scrum/Kanban), with short iterations and continual feedback to refine features. Technical features include data visualization (charts of time spent per task via Chart.js) and user authentication for personalized data management. Integration with external services was also explored –notably the OpenAI API– laying the groundwork for future incorporation of AI-generated daily reports.

The project delivered a functional **proof-of-concept** web application for all-in-one productivity management. The tool allows users to create and organize tasks in lists or boards, move them through status columns (to-do, in review, done), log time spent on each task with an integrated timer, and record notes or relevant data (for example, project codes or issues to later input into a corporate ERP). Data is stored per user in the cloud, making it accessible anytime, anywhere. We conducted usability tests with sample use cases that demonstrated the application's utility: a user could plan their workday, accurately track hours for each activity, and obtain a unified report of the day's work. The results indicate **improvements in efficiency and organization**: information previously scattered across different apps is now centralized, reducing duplicate entries and simplifying oversight of daily progress. Additionally, the application provides summary charts of time distribution, giving immediate visibility into which tasks consume the most time.

The proposed solution confirms the *feasibility and benefits* of a unified personal work management tool. Compared to using separate specialized tools (e.g.,

Trello or Asana for tasks, Clockify for time tracking), our integrated application streamlines workflow by eliminating the need to switch between multiple systems. This can translate into time savings and a gentler learning curve for non-technical users. The architecture based on standard web tech and cloud services suggests good scalability and maintainability long term.

Keywords: Remote work, time tracking, Kanban, Full Stack JavaScript, Artificial Inteligence.

Glosario

- **Gestión de tareas:** Proceso de **organización, planificación y seguimiento** de actividades o responsabilidades, con el objetivo de completarlas de manera eficiente dentro de plazos establecidos. Implica priorizar tareas, asignar recursos y monitorizar su progreso hasta su finalización exitosa.
- **Control horario (registro de horas):** Práctica de **medir y registrar el tiempo** dedicado a cada actividad o proyecto. Permite cuantificar las horas de trabajo invertidas, frecuentemente mediante temporizadores o partes de horas, para analizar la productividad, estimar costos o cumplir con requisitos de reporte.
- **ERP (Planificación de Recursos Empresariales):** *Software* integral de **gestión empresarial** que unifica en un solo sistema diversos procesos de negocio (finanzas, recursos humanos, inventario, ventas, etc.). Un ERP actúa como el "sistema nervioso central" de una organización, proporcionando una única fuente de información veraz y facilitando la coordinación de operaciones.
- **Productividad personal:** Capacidad de un individuo para **gestionar eficazmente su tiempo, energía y recursos** al realizar tareas o proyectos. Alta productividad personal se refleja en completar más trabajo de valor en menos tiempo, manteniendo calidad y equilibrio, a través de buenas prácticas de organización, enfoque y eliminación de distracciones.
- **Aplicación web:** Programa o aplicación de software a la que se accede mediante un **navegador web** (Chrome, Firefox, etc.) a través de Internet, en lugar de una instalación local. La aplicación web ofrece funcionalidad interactiva al usuario (p. ej., gestión de datos, transacciones) y usualmente sigue un modelo cliente-servidor, donde la lógica y los datos residen en un servidor remoto.
- **Node.js (Express):** Entorno de ejecución de JavaScript del lado del servidor, de código abierto y multiplataforma, que permite construir aplicaciones de red escalables usando JavaScript fuera del navegador. **Express** es un *framework* web minimalista para Node.js que simplifica la creación de servidores HTTP y APIs, proporcionando un conjunto robusto de funcionalidades web mediante un enfoque flexible.
- **JavaScript (Front-end):** Lenguaje de programación interpretado, orientado a objetos, utilizado principalmente en el **desarrollo web front-end** para crear contenido dinámico e interactivo en las páginas web. Junto con HTML y CSS, JavaScript es una de las tecnologías fundamentales de la web, permitiendo responder a las acciones del usuario, manipular la estructura DOM de la página y comunicar con servidores para actualizar contenido sin recargar la página.

- **Chart.js:** Biblioteca de JavaScript de código abierto para la **visualización de datos** en la web. Permite generar de forma sencilla gráficos interactivos (barras, líneas, sectores, etc.) dentro de un lienzo HTML5, facilitando que desarrolladores y diseñadores incorporen *charts* informativos en aplicaciones web.
- **Metodología ágil (Scrum/Kanban):** Conjunto de enfoques de gestión de proyectos de software enfocados en la **entrega iterativa y continua** de valor, la adaptabilidad al cambio y la colaboración estrecha con el cliente. Marcos ágiles populares incluyen **Scrum** (trabajo organizado en ciclos cortos llamados *sprints* con entregables frecuentes) y **Kanban** (gestión visual del flujo de trabajo mediante un tablero que refleja en tiempo real las tareas en curso y la capacidad del equipo).
- **Inteligencia Artificial (informes automáticos con LLM):** Disciplina informática que busca crear sistemas capaces de **realizar tareas que normalmente requieren inteligencia humana**, como entender lenguaje natural o tomar decisiones. En el contexto del proyecto, alude al uso de *Large Language Models (LLM)* –modelos de lenguaje de gran tamaño, p. ej. GPT-4– para **generar automáticamente texto** útil para el usuario (informes, resúmenes de actividades, etc.), imitando el estilo y contenido que produciría una persona. Esto aprovecha avances recientes en IA generativa para brindar funcionalidades inteligentes, como ya se observa en herramientas actuales que resumen conversaciones o notas de forma autónoma.

1. Antecedentes

1.1 Descripción del problema o necesidad

En la era digital, trabajadores y equipos se enfrentan al **reto de administrar eficientemente sus tareas y el tiempo** dedicado a ellas. Una gestión deficiente acarrea pérdidas notables de productividad: estudios recientes revelan que los empleados pueden destinar *hasta 31 horas mensuales* a reuniones innecesarias y que un *65%* considera que la excesiva frecuencia de reuniones merma su desempeño. Estas horas mal invertidas, sumadas a la desorganización y falta de visibilidad sobre el trabajo en curso, generan estrés y menores niveles de rendimiento(Cartagena, 2024; Erkic, 2023). El problema se manifiesta tanto en empresas consolidadas como en profesionales independientes, indicando una **necesidad transversal** de mejorar la forma en que planificamos y hacemos seguimiento del trabajo diario.

Otro aspecto del problema es la **fragmentación de herramientas**. En ausencia de una solución integrada, muchas personas recurren a múltiples aplicaciones y métodos para cubrir sus necesidades: por ejemplo, llevar una lista de tareas en una aplicación, registrar el tiempo en una hoja de cálculo aparte, y anotar incidencias o datos relevantes en documentos sueltos. Este sistema obliga al usuario a cambiar constantemente de contexto y plataforma, lo cual disminuye la eficiencia. Efectivamente, estudios sobre la multitarea señalan que alternar entre varias herramientas y ventanas dificulta mantener un flujo de trabajo consistente y puede reducir significativamente la productividad debido al “costo de cambio de contexto”(Azevedo, 2023; Sinzerad, 2024).

1.2 Contexto del proyecto

El contexto de este proyecto se fija en los últimos cambios recientes en la forma de trabajar y en las herramientas disponibles. A raíz de la digitalización masiva y el teletrabajo, ha cobrado mayor importancia dotar a personas y pequeños equipos de herramientas de productividad accesibles. Durante 2020, por ejemplo, el uso de plataformas de colaboración en línea se disparó: Microsoft Teams superó los 115 millones de usuarios activos(Spataro, 2020), esta adopción acelerada de tecnología evidenció que las personas buscan apoyarse en aplicaciones digitales para coordinarse mejor a distancia. Varios análisis han mostrado que dicha adopción puede impactar positivamente en la eficiencia(Chui & Manyika, 2012; Vorecol, 2024): **estudios de McKinsey** y otras consultoras indican mejoras del orden del *20–25% en la productividad* cuando se implementan herramientas de colaboración, gracias a la mejor coordinación, comunicación y acceso a la información que estas facilitan.

Sin embargo, gran parte de esas soluciones han estado enfocadas en entornos corporativos o en problemas específicos, más que en ofrecer una ayuda integral a nivel personal o de pequeñas organizaciones. Por ejemplo, los sistemas ERP empresariales actúan como núcleo unificador de procesos para medianas y grandes empresas, integrando desde finanzas hasta logística en una sola. Pero

para un profesional autónomo o una microempresa, un ERP tradicional resulta *excesivo en alcance y costo*. Estas personas suelen preferir herramientas más sencillas y económicas (idealmente gratuitas) para gestionar su día a día. El contexto, entonces, es el de **pequeños equipos, estudiantes y freelancers** que necesitan organizar sus tareas y tiempo sin contar con departamentos dedicados de TI ni presupuestos para soluciones corporativas. Este segmento hasta ahora ha tenido que conformarse con usar múltiples aplicaciones gratuitas en paralelo, lidiando con sus limitaciones, o adoptar soluciones integrales de pago pensadas para compañías (con la carga de funcionalidades que no utilizan y costos recurrentes).

1.3 Análisis de la situación actual

Antes de este proyecto, tenemos que ver que existen numerosas herramientas especializadas que abordan por separado las distintas facetas de la gestión del trabajo:

- Para la **gestión de tareas** (to-do lists y proyectos)(Dekle, 2019): hay aplicaciones populares como *Trello*, *Asana*, *Todoist*, *Microsoft To Do*, *ClickUp*, entre otras. Por ejemplo, **Trello** es una herramienta muy difundida que implementa tableros Kanban en la nube para organizar visualmente tareas mediante tarjetas movibles entre columnas de estado. Estas aplicaciones permiten listar tareas, asignarlas, establecer vencimientos y colaborativamente supervisar su avance. Algunas se orientan a proyectos de equipo (Asana, Monday) y otras a productividad personal (Todoist), pero en general **no registran el tiempo invertido** automáticamente ni integran notas estructuradas para cada tarea más allá de comentarios o descripciones simples.
- Para el **control horario o seguimiento de tiempo**: destacan aplicaciones como *Toggl Track*, *Clockify*, *Harvest* o *RescueTime*(Abbamonte, 2024). Estas herramientas permiten cronometrar cuánto se trabaja en cada tarea o proyecto e incluso generan informes para facturación o análisis de productividad. Por ejemplo, **Clockify** ofrece un sistema de cronómetro y hojas de tiempo muy sencillo donde el usuario inicia o detiene un contador y asigna las horas a proyectos o etiquetas. Son utilizadas por freelancers y equipos para cuantificar horas facturables o simplemente entender en qué se va el tiempo. No obstante, estos servicios suelen funcionar de forma independiente a la planificación de tareas: es decir, ayudan a medir el tiempo, pero *no planifican ni organizan* el listado de pendientes.

- En cuanto tomar **anotaciones de tareas o proyectos en sistemas corporativos** muchas compañías grandes utilizan módulos de sus ERPs (como SAP, Oracle, Microsoft Dynamics) o herramientas específicas (por ej., Jira Software en entornos de desarrollo) para registrar información complementaria de las tareas(Jira, 2025): códigos de proyecto,

imputaciones de horas a centros de costo, incidencias reportadas, etc. En el ámbito de usuarios individuales o pymes, ha habido una tendencia a usar soluciones todo-en-uno como *Notion* o *Evernote*(Rebelo, 2024), donde se pueden llevar notas estructuradas, bases de datos simples y contenido variado. **Notion**, en particular, se ha hecho popular por permitir combinar en un mismo espacio notas tipo wiki, tareas en tablas o tableros, y contenido multimedia; incluso ha incorporado recientemente asistentes de IA para resumir texto y generar contenido. Aun así, Notion carece de un sistema integrado de fichaje de horas (el control de tiempo debe hacerse manualmente o con integraciones externas) y sus prestaciones como gestor de tareas, si bien flexibles, requieren mayor personalización por parte del usuario.

- Finalmente, los **ERP** propiamente dichos como *SAP ERP, Odoo, Oracle Netsuite*, ofrecen módulos de gestión de proyectos y tiempo, pero suelen implicar infraestructuras complejas, costos de licencia o suscripción, y personalización a medida(Doctor, 2022; FocusTribes, 2024; NetSuite.com, 2024). Un ERP se define justamente como un software integral que actúa como fuente unificada de datos de la empresa, integrando finanzas, inventario, RR.HH., proyectos, etc., **pero** su implementación típica está fuera del alcance (y de las necesidades de un profesional individual o una microempresa.

Al revisar las herramientas disponibles, queda claro que todas tienen sus limitaciones. Ninguna opción gratuita ofrecía de forma sencilla las tres funciones clave que se buscan aquí: organizar tareas, registrar el tiempo de trabajo y anotar información estructurada con un enfoque profesional o corporativo. Para conseguir todo eso, uno tenía que combinar varias aplicaciones por su cuenta.

Por ejemplo, un autónomo típico quizás usaba Trello para organizar sus tareas, luego apuntaba las horas en un Excel o usaba Clockify para el control horario, y además llevaba sus notas en un documento aparte o en una libreta. Algunos servicios han intentado cerrar esa brecha. Clockify, por ejemplo, permite integrarse con Trello, Asana y más de 50 apps(Clockify, 2025), lo que te deja arrancar un cronómetro desde una tarjeta de Trello. Pero claro, eso implica usar dos herramientas diferentes y configurar la integración previamente.

En la práctica, lo que suele pasar es que los datos acaban repartidos por varios sitios, y el usuario tiene que hacer muchos pasos manuales para unirlo todo. Un proceso poco ágil y, muchas veces, frustrante.

1.4 Justificación del proyecto

Teniendo en cuenta todo lo anterior, ¿por qué no se ha resuelto antes esta necesidad? En buena medida, porque combinar de forma sencilla funciones como tareas, control horario y anotaciones no ha sido una prioridad para muchas empresas de software. Los grandes proveedores centraron sus ERP en clientes corporativos con más presupuesto, y muchos startups optaron por especializarse

en áreas concretas —como las listas de tareas o el seguimiento del tiempo— sin intentar cubrirlo todo.

Esto dejó a usuarios individuales y pequeñas organizaciones dependiendo de varias herramientas al mismo tiempo. Existen soluciones más completas, como ClickUp o Jira, pero suelen estar pensadas para equipos y requieren una curva de aprendizaje mayor o suscripciones de pago, lo que las hace menos accesibles para un uso personal o autónomo.

Este proyecto de TFG se justifica porque podría llegar a ofrecer una herramienta practica y novedosa: **ofrece una alternativa integradora** para la gestión diaria que hasta ahora no estaba disponible sin coste. La propuesta es desarrollar una aplicación web que reúna en un mismo lugar la planificación de tareas, el registro horario y las anotaciones. Con esta propuesta se puede cubrir una necesidad concreta y conseguir simplificar el trabajo diario, evitando usar otras aplicaciones y reduciendo el tiempo que se pierde al cambiar de contexto o repetir datos. Centralizar estas funciones permite que, por ejemplo, al completar una tarea el usuario pueda inmediatamente anotar cuánto tiempo le tomó y qué observaciones surgieron, todo en la misma interfaz – algo imposible con herramientas separadas.

La **motivación personal** también ha sido un factor clave para realizar este proyecto. Como autor del proyecto experimenté de primera mano la dificultad de llevar un control diario de tareas y horas durante las prácticas: debía anotar en un cuaderno las horas dedicadas a distintas asignaciones y luego volcarlas diariamente en un parte de horas corporativo de difícil acceso, usando a la vez un documento de texto para ir anotando para cada tarea notas de interés de cada tarea. Esta vivencia evidenció la *ineficiencia* de ese proceso manual y fragmentado, inspirando la idea de “tenerlo todo en una sola herramienta”. Tras investigar las aplicaciones disponibles y confirmar que ninguna satisfacía plenamente mi necesidad de forma simple, decidí emprender la creación de una solución ad hoc. Dado que se trata de un proyecto académico, esto no solo responde a un interés práctico, sino que brinda la oportunidad de aplicar conocimientos aprendidos en la carrera (desarrollo web, bases de datos, diseño de interfaces, etc.) en un producto concreto y útil.

La viabilidad del proyecto está respaldada por la **madurez de las tecnologías** actuales. Hoy es posible desarrollar aplicaciones web potentes de manera relativamente rápida gracias a *stacks* consolidados (en este caso, Node.js/Express para el servidor y herramientas front-end en JavaScript para cliente).

El uso de una base de datos en la nube como MongoDB Atlas elimina la necesidad de mantener infraestructura compleja y ofrece escalabilidad bajo demanda.

Asimismo, la disponibilidad de **APIs** a bajo costo o gratuitas –por ejemplo, la API de OpenAI para procesamiento de lenguaje natural– permite plantear funcionalidades novedosas (como generar un resumen automático de las tareas del día) que añadirían un valor diferenciador importante

Grandes proveedores ya están integrando estas capacidades(Slack, 2025): sin ir más lejos, **Slack** ha lanzado funciones de IA que resumen conversaciones de equipo o generan notas de reuniones automáticamente. Esto indica que dotar a nuestra aplicación con componentes de Inteligencia Artificial no solo es factible, sino acorde a las tendencias de la industria, lo que podría convertirla en una herramienta pionera dentro de su nicho.

En resumen, este proyecto tiene sentido porque aborda una necesidad concreta: reunir en una sola aplicación web gratuita la organización de tareas, el registro de tiempo y la toma de notas estructuradas. Hoy en día, muchas personas —especialmente trabajadores autónomos o usuarios individuales— tienen que usar varias herramientas para cubrir estas funciones, lo que implica más esfuerzo y menor eficiencia.

Además, el desarrollo puede aprovechar tecnologías accesibles como servicios en la nube o componentes de inteligencia artificial, lo que facilita su implementación técnica. Más allá del producto final, el proyecto también permite explorar cómo integrar distintas funciones de productividad en una sola interfaz, algo útil tanto para usuarios como para quienes investigan o diseñan herramientas digitales.

Dado el contexto actual de trabajo digital y flexible, contar con soluciones que ayuden a organizar mejor las tareas y el tiempo tiene un valor práctico claro.

2. Objetivos del proyecto

Los objetivos de este proyecto se pueden desglosar en los siguientes puntos específicos:

- **Desarrollar una aplicación web que permita al usuario gestionar sus tareas diarias, registrar el tiempo dedicado a cada una de ellas y tomar anotaciones estructuradas, todo desde una única interfaz integrada.** Esta herramienta estará diseñada para cubrir de forma sencilla y clara las necesidades básicas de planificación individual, facilitando que el usuario registre el progreso de su trabajo sin depender de múltiples plataformas. Se comprobará su funcionalidad a través de pruebas donde el usuario pueda completar ciclos de trabajo: crear tareas, moverlas de estado, asociar tiempos y añadir contenido textual relacionado. Su desarrollo se ajustará a los recursos disponibles durante el periodo de ejecución del proyecto.
- **Implementar un sistema de almacenamiento remoto basado en una base de datos en la nube, que permita conservar los datos del usuario de manera segura y acceder a ellos desde distintos dispositivos.** La solución tecnológica elegida deberá soportar múltiples sesiones simultáneas y mantener la integridad de la información. Se evaluará su correcto funcionamiento mediante el acceso desde diferentes ubicaciones y cuentas, asegurando que los datos se recuperan sin pérdida. Se utilizarán herramientas ampliamente documentadas y accesibles, adaptadas a las capacidades del proyecto y dentro del calendario previsto para el desarrollo.
- **Desarrollar una funcionalidad de copia de seguridad automática que registre cualquier modificación en tareas, tiempos o notas y permita restaurar la información en caso de pérdida o error.** Este sistema deberá operar sin intervención manual y almacenar los datos en un servicio externo. Su eficacia se medirá mediante pruebas controladas de recuperación, en las que se eliminará el contenido local y se verificará su restauración. La implementación se hará con tecnologías compatibles con los demás componentes del sistema y enmarcada dentro del cronograma de trabajo.
- **Integrar un sistema de generación automática de informes que, a partir de los datos registrados, proporcione resúmenes en lenguaje natural sobre la actividad realizada por el usuario durante un periodo determinado.** El sistema deberá producir textos comprensibles y organizados que incluyan información como tareas completadas, tiempo invertido y elementos pendientes. Se validará mediante la comparación entre los datos brutos y el contenido generado por el modelo. Esta funcionalidad aprovechará servicios existentes de procesamiento de lenguaje y será diseñada para adaptarse al flujo de uso normal de la aplicación sin requerir recursos adicionales por parte del usuario.

- **Diseñar la aplicación de modo que pueda utilizarse sin costes económicos para el usuario final, empleando tecnologías de código abierto y servicios con planes gratuitos.** La arquitectura deberá mantenerse operativa dentro de estos márgenes, sin limitar la funcionalidad esencial. Para verificarlo, se desplegará el sistema en entornos que no requieran licencias comerciales, comprobando que todas las operaciones principales se ejecutan correctamente. Esta decisión responde a la intención de facilitar el acceso a perfiles individuales y pequeños equipos que no disponen de recursos para asumir pagos por software.

3. Marco teórico

En este apartado se revisan los conceptos teóricos y el estado del arte relacionados con la gestión de tareas, el control de tiempo, la integración con sistemas y las tecnologías y metodologías empleadas en el proyecto. Se cubren también ejemplos de herramientas existentes y principios de diseño de experiencia de usuario relevantes para nuestra aplicación.

3.1 Gestión de tareas y productividad personal

Administrar eficazmente las tareas diarias es un componente de la productividad personal y organizacional. Métodos como **Getting Things Done (GTD)** de David Allen o la técnica **Pomodoro** se crearon para ayudar a las personas a organizar sus pendientes y optimizar el uso del tiempo(Allen, 2025; Zimmergren, 2024). En la era digital, estos métodos han sido incorporados en aplicaciones que permiten listas de tareas, recordatorios, categorización por proyectos, fechas de vencimiento, prioridades, etc. Por ello, registrar nuestras tareas y tiempos ofrece varios beneficios prácticos:

- **Conciencia del uso del tiempo:** Al llevar un registro diario, la persona puede saber con precisión cuánto tiempo dedican a determinadas actividades. Esto ayuda a descubrir, por ejemplo, que una tarea aparentemente breve realmente consume muchas horas, o que determinadas reuniones semanales acumulan un volumen significativo de tiempo. Con estos datos, se pueden tomar decisiones informadas para planificar mejor (p. ej., reducir la frecuencia de reuniones innecesarias, o estimar con más realismo la duración de tareas similares en el futuro).
- **Identificación de distracciones o tareas no esenciales:** Solo al anotar en qué se va el tiempo se pueden evidenciar pérdidas por multitarea improductiva, interrupciones frecuentes, u otras ineficiencias. El registro horario permite luego analizar y eliminar o reducir aquellas actividades que no aportan valor (por ejemplo, navegar redes sociales durante horas de trabajo, o duplicar esfuerzos por falta de coordinación).
- **Justificación y reporte del trabajo:** En entornos profesionales, tener las horas registradas por proyecto permite rendir cuentas tanto a nivel interno (saber si un proyecto está demandando más esfuerzo del previsto) como externo (facturación de horas a clientes, informes de progreso, etc.). Incluso para un estudiante, apuntar sus horas de estudio por asignatura puede dar idea de dónde concentra su esfuerzo y con qué resultados, facilitando un mejor reparto del tiempo de estudio.

- **Mejora continua personal:** Algunas aplicaciones convierten el registro de tareas en un juego de mejora personal. Por ejemplo, Todoist

implementa un sistema de *karma* que puntúa al usuario según tareas completadas y hábito de uso, para motivarlo a mantener disciplina. El simple hábito de revisar a final de semana qué tanto se avanzó (cuántas tareas de la lista se completaron, cuántas horas productivas se lograron) proporciona un *feedback* valioso para ajustar hábitos la semana siguiente.

Considerando lo anterior, la aplicación propuesta en este TFG añade un control de horario y un registro de actividad para medir la productividad, además de las listas de tareas. En la aplicación desarrollada, cuando una tarea se marca como completada, queda registrada en un historial junto a la fecha de finalización y las horas acumuladas que tomó. Además, diariamente se resetean las horas de cada tarea pendiente tras guardar el total en un historial, para iniciar un nuevo día en cero, pero conservando lo realizado el día anterior. De este modo, se puede luego calcular cuánto tiempo total llevó una tarea a lo largo de varios días.

Se añadió además un elemento visual para representar las tareas completadas con sus horas totales, proporcionando una representación visual de la distribución de esfuerzos. El poder **visualizar la propia productividad** (sea en gráficos o reportes generados) puede motivar al usuario y clarificar su desempeño. En la Figura siguiente se muestra un ejemplo de gráfico de barras que ilustra horas acumuladas en tareas completadas durante un periodo determinado:

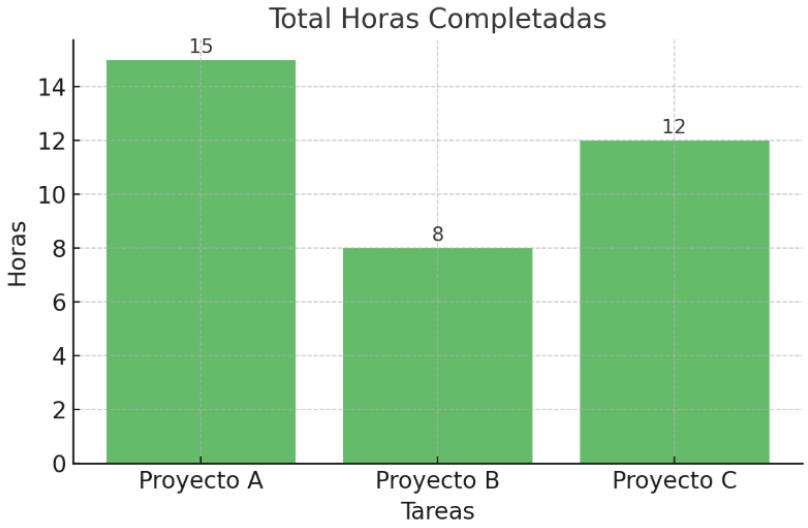

Figura 3. 1: Gráfico de ejemplo que muestra las horas totales dedicadas a cada tarea completada en un periodo. En la aplicación, este gráfico se actualiza dinámicamente conforme se completan tareas y se acumulan horas.

3.2 Revisión de aplicaciones de productividad existentes

Para situar el sistema propuesto en perspectiva, es importante revisar las características de algunas de las **aplicaciones de productividad personal y de equipo más populares en la actualidad**, identificando qué ofrecen y qué

limitaciones tienen con respecto a los objetivos de nuestro proyecto. A continuación, se analizan brevemente cuatro herramientas representativas – **Notion, Trello, Todoist y ClickUp** – resaltando sus ventajas, desventajas y modelo de negocio, con énfasis en su comparación con nuestra solución (gestión de tareas + horas + ERP integrado):

Notion: Es una plataforma *todo-en-uno* que combina toma de notas, gestión de tareas, bases de datos simples, wikis y documentos en un mismo espacio altamente flexible. Su fortaleza es la versatilidad: el usuario puede crear su propia estructura de páginas y plantillas, incluyendo listas de tareas, tableros Kanban, calendarios, tablas con campos personalizados, etc., adaptándose tanto a uso personal como de equipo(Baleón García, 2025). Notion ha ganado una enorme base de usuarios (más de 20 millones para 2021, según reportes de prensa) y ha sido elogiada como una app que “cambia la vida” por su flexibilidad(CMS, 2024). Ofrece un plan gratuito generoso para uso personal (páginas y bloques ilimitados para un usuario individual) y planes de pago orientados a colaboración en equipos.

- **Ventajas:** Integración de mucha información en un solo lugar, extrema personalización (puede usarse como gestor de proyectos, notas, base de datos simple, diario personal, etc.), y una activa comunidad que comparte plantillas(Wheelhub, 2024).
- **Desventajas:** No tiene funcionalidades nativas de control horario; si bien es posible crear manualmente campos o tablas para registrar horas, no es su enfoque principal. Para algunos usuarios la libertad de Notion resulta abrumadora, pues carece de estructura predeterminada – requiere invertir tiempo en configurar el espacio a medida(Wheelhub, 2024).
- **Comparación:** Notion cubriría en principio la parte de tareas y anotaciones ampliamente (podríamos diseñar una base de datos en Notion para tareas con campos de horas), pero no ofrece por defecto una solución simple de *time-tracking* ni la lógica de negocio especializada que nuestra aplicación brinda (reinicio diario de horas, acumulación histórica, generación de informes IA). Además, aunque Notion es gratis en lo básico, ciertas funciones avanzadas o colaborativas requieren pago, mientras que la propuesta de TFG busca ser completamente libre para el usuario final(Eugene, 2025).

Trello: Es una de las herramientas de gestión de tareas más conocidas, basada en Kanban. Presenta un tablero visual donde las tareas son tarjetas que se mueven entre listas (columnas) típicamente tituladas “Por hacer”, “En proceso” y “Hecho”(Trello, 2025).

- **Ventajas:** Sencillez extrema de uso, interfaz visual atractiva y apta para equipos de cualquier tamaño; Trello es gratuito en su plan básico e incluso permite ciertos complementos (*power-ups*) para funciones extra. Permite colaboración en tiempo real, adjuntar archivos a tarjetas, establecer fechas de vencimiento y checklists dentro de tareas(Muradas, 2020).
- **Desventajas:** Trello por sí solo no incluye control horario; para registrar tiempo es necesario integrarlo con otra herramienta o usar un *power-up* específico (a menudo de pago). Asimismo, si bien se pueden escribir descripciones o comentarios en cada tarjeta, Trello no está pensado para capturar información estructurada ni para generar informes narrativos. Su plan gratuito tiene limitaciones (por ejemplo, máximo un power-up por tablero, tamaño de archivos adjuntos limitado) mientras que la versión Business Class (de pago) amplía esas capacidades(Stepanets, 2024).
- **Comparación:** La interfaz de nuestra aplicación está inspirada en Trello en el sentido de manejar listas de tareas en diferentes estados (pendiente, revisión, completada). Sin embargo, la integración de registro de horas y notas es una diferenciación clave de nuestra app frente a Trello. Donde Trello requeriría enchufar un complemento externo (potencialmente de pago) para trackear horas, nuestra solución lo trae incorporado de forma simple. Por otro lado, Trello, al ser tan general, no contempla la noción de un informe o un panel consolidado de notas para reporte como el que hemos incluido; ese es un detalle de nicho que nuestro sistema cubre directamente.

Todoist: Es un gestor de tareas personales enfocado en listas y fechas, con un enfoque minimalista pero potente. Permite organizar tareas en proyectos, asignar etiquetas, prioridades, fechas de vencimiento y recibir recordatorios. Es muy utilizado para productividad individual, con más de 10 millones de usuarios(Todoist, s. f.-b).

- **Ventajas:** Interfaz limpia y disponible en múltiples plataformas (móvil, web, escritorio); facilidad para capturar rápidamente tareas incluso con lenguaje natural (por ejemplo, escribir “Llamar al cliente mañana 10am” crea la tarea con fecha automáticamente). Ofrece un sistema de karma/puntos que motiva a completar objetivos diarios/semanales, y soporta colaboración básica en proyectos compartidos(Todoist, 2025).

- **Desventajas:** En su plan gratuito, Todoist tiene algunas restricciones (p. ej., número de proyectos activos limitados, no incluye recordatorios por hora que son función Premium). Tampoco tiene control horario: está pensado para marcar tareas como completadas, pero no para medir cuánto tardaste en cada una. No posee tablero Kanban (aunque recientemente añadió una vista de tablero) ni integración directa con ERP(Todoist, s. f.-a).

- **Comparación:** Nuestra aplicación es menos sofisticada en organización avanzada (p. ej., Todoist permite organizar por etiquetas contextuales, prioridades alta/media/baja, etc., cosas que nuestra app no implementa aún). Sin embargo, aportamos la dimensión temporal que Todoist no maneja. Un usuario que necesitara saber no solo *qué* hizo sino *cuánto tiempo le llevó* puede preferir nuestra app a usar Todoist + otra app de time tracking separada. En cuanto al costo, nuestro proyecto al ser auto hospedado es esencialmente gratuito, mientras que Todoist requiere suscripción (~5€ al mes) para ciertas funciones como recordatorios basados en hora o comentarios adjuntos a tareas.

En general, al analizar estas y otras herramientas se detectan ventajas comunes como la sincronización en la nube, disponibilidad multidispositivo, interfaces pulidas gracias a pruebas de UX, y a menudo integraciones con calendarios o correo electrónico. Por otro lado, comparten algunas limitaciones relevantes para nuestro caso: la mayoría segmenta las funcionalidades en distintos productos o niveles de pago (por ejemplo, Trello no registra horas sin add-on, Notion no tiene control temporal, Clockify no gestiona tareas por sí solo, etc.), lo que obliga al usuario a mezclar herramientas o pagar por planes superiores para lograr una solución completa. Además, pocas están centradas en el caso de uso individual con necesidades específicas como **registrar lo realizado por día y obtener un informe diario**; más bien apuntan a mercados empresariales amplios o a un uso personal muy genérico.

Frente a este panorama, el sistema propuesto en el proyecto es una **Agenda de Tareas con Horas que** busca un equilibrio de simplicidad y funciones relevantes para un usuario que maneja su propio trabajo diario. La ventaja comparativa es la integración nativa de funcionalidades sin requerir suscripciones ni configuraciones complejas: en una sola página web se tienen las tareas, se actualizan las horas con un campo numérico, se escriben notas vinculadas, y todo queda almacenado listo para generar un reporte.

3.3 Experiencia de usuario (UX/UI) y eficiencia en diseño

Un último aspecto teórico para considerar es la **experiencia de usuario** en aplicaciones de productividad. De poco serviría reunir muchas funcionalidades si la interfaz no resulta usable, pues podría terminar disminuyendo la productividad en lugar de aumentarla. Numerosos estudios en el campo de UX/UI resaltan que un buen diseño centrado en el usuario tiene impactos directos en la eficiencia(Vorecol, 2024; Wojahn, 2022). En el diseño de esta aplicación se han seguido varios principios de usabilidad reconocidos:

- **Simplicidad y minimalismo**(Cáceres, 2023)**:** La pantalla principal muestra solo lo esencial: un campo de texto para añadir una nueva tarea, botones claros para acciones (agregar tarea, mover a revisión, completar) y cuatro secciones definidas (Tareas Pendientes, Revisión Pendiente, Bloc de Notas, ERP, además del historial de tareas completadas). Se evitó abrumar con menús complicados o opciones ocultas. La paleta de colores es sencilla y agradable (tonos claros, acentos en verde y azul que evocan organización y calma). Un diseño simple reduce la carga cognitiva del usuario al interactuar.
- **Coherencia con paradigmas conocidos**(Soler, 2024)**:** La organización de las tareas en listas verticales recuerda a listas de tareas tradicionales o tableros Kanban digitales, algo con lo que muchos usuarios ya están familiarizados. El uso de términos como “Pendientes”, “Revisión” o “Completar” y el empleo de botones de acción inmediata (sin necesidad de menús contextuales) sigue convenciones habituales de aplicaciones similares. Se aplicaron convenciones consistentes de colores e iconos: por ejemplo, el botón de “Completar” tarea se muestra en color verde (asociado al éxito o finalización), y todos los campos de entrada tienen estilos uniformes.
- **Retroalimentación inmediata**(Arenzana, 2024)**:** Cada acción del usuario tiene una respuesta visible en la interfaz. Al agregar una tarea, ésta aparece al instante en la lista de pendientes; al escribir en la nota o campo ERP de una tarea seleccionada, la aplicación guarda automáticamente los cambios en segundo plano y mantiene la información al recargar la página (persistencia). Aunque no se muestra un mensaje explícito de “guardado exitoso” para cada cambio (para no interrumpir la experiencia), el usuario puede confiar en que el sistema registra sus entradas porque, por ejemplo, si actualiza la página del navegador, las notas y horas introducidas permanecen. Esta inmediatez en la respuesta genera confianza y un flujo de uso más fluido.

- **Diseño responsive**(Cámara de Sevilla, 2023)**:** Aunque la aplicación se pensó principalmente para uso en pantallas de ordenador, se incluyeron en la hoja de estilo algunas reglas *CSS responsive* (mediante *media queries*) para que en pantallas pequeñas las secciones se reordenen en columna en lugar de lado a lado. De este modo, si un usuario accede desde un móvil o tablet, puede igualmente interactuar sin elementos montados o fuera de la vista. La accesibilidad multiplataforma es un componente crucial hoy día, dado que muchos usuarios alternan entre dispositivos a lo largo del día.
- **Evitar saturación de información**(Linkedin, 2025)**:** Se muestra lo necesario en cada momento. Por ejemplo, el historial de tareas completadas permanece oculto (colapsado en la parte inferior) hasta que el usuario decide consultarlo, para no distraer de las tareas activas.

Igualmente, el panel “ERP definitivo” solo lista las tareas pendientes que ya tienen algo escrito en el campo ERP, filtrando la información para presentar solo lo relevante de cara al reporte. Esto sigue el principio de *progressive disclosure*, donde la información menos usada se oculta inicialmente para simplificar la vista principal.

4. Metodología

En este apartado se detalla la metodología seguida para el desarrollo del proyecto, incluyendo el enfoque de trabajo adoptado, las herramientas empleadas y la arquitectura diseñada. Dado que se trata de un desarrollo de software con alcance y requisitos susceptibles de evolucionar, era importante elegir una metodología ágil e iterativa en lugar de un enfoque secuencial rígido.

4.1 Enfoque de desarrollo iterativo y ágil

Se optó por un enfoque de desarrollo **ágil e incremental**. En la práctica, el trabajo se organizó en iteraciones semanales, cada una con objetivos claros (historias de usuario o funcionalidades a implementar). Al inicio del proyecto se definió una lista con los requisitos principales deseados. Luego, en la primera iteración se construyó un prototipo básico: la estructura HTML inicial y la capacidad de agregar y listar tareas pendientes. En la siguiente iteración se añadió la funcionalidad de mover tareas a estado de revisión y completarlas, junto con el almacenamiento local en un archivo JSON. Posteriormente, se incorporó el registro de horas por tarea y el manejo de acumulación diaria, así como la interfaz para escribir notas y texto para cada tarea. Más adelante se integró la generación de informes mediante la API de OpenAI y la funcionalidad de exportar esos informes a PDF. Al final de cada iteración se probó el sistema de manera integrada para detectar errores y reajustar prioridades de la iteración siguiente.

Este enfoque flexible permitió entregar rápidamente un producto funcional mínimo e ir refinándolo de forma incremental. Por ejemplo, una vez visto el prototipo básico en funcionamiento, se identificó la necesidad de añadir un campo de texto por tarea y un panel consolidado de ERP definitivo, lo cual no estaba en la primera lista de requisitos, pero surgió de la experiencia de uso. Gracias a la metodología ágil, se pudo incorporar este cambio en una iteración posterior sin problemas.

Por otra parte, se fue documentando el progreso en entregas intermedias (como la memoria parcial entregada a mitad del TFG), lo que ayudó a reflejar y revisar la dirección del proyecto. Esta retroalimentación periódica aseguró que se cumplieran los objetivos académicos además de los técnicos.

4.2 Herramientas y entorno de desarrollo

El desarrollo se realizó en un entorno típico de aplicaciones web JavaScript:

- **Visual Studio Code** fue utilizado como IDE principal, aprovechando su integración con Node.js y herramientas front-end, y extensiones para formateo de código.
- **Git** se usó para el control de versiones del código, con un repositorio privado en GitHub para backup y gestión básica de issues.

- El proyecto se inició con Node.js v14 y posteriormente se actualizó a **Node.js v16**, junto con el gestor de paquetes **npm** para instalar dependencias. Las principales dependencias fueron *express* (framework web), *dotenv* (carga de configuración), *openai* (cliente para la API de OpenAI), *mongodb* (cliente para MongoDB) y *puppeteer* (para la generación de PDF).
- En la parte cliente, se utilizó la biblioteca **Chart.js** (v3) mediante CDN para los gráficos, y se incluyó también la librería de HTML2PDF (vía CDN) en alguna prueba preliminar, aunque finalmente se optó por generar PDFs en el servidor con **Puppeteer**.

4.3 Arquitectura de la aplicación

La arquitectura de la aplicación sigue el clásico modelo cliente-servidor de las aplicaciones web. En el **front-end** (cliente) se tiene una página HTML única (index.html) que implementa toda la interfaz de usuario y la lógica con JavaScript en el navegador. En el **back-end** (servidor) se tiene una aplicación Node.js con Express (server.js) que expone una serie de rutas HTTP para obtener y modificar los datos de las tareas. Para persistencia, en la versión actual, se usa un archivo JSON en el servidor (tasks.json) que actúa como base de datos simple, complementado con una copia de seguridad en una base de datos MongoDB Atlas. La comunicación entre front-end y back-end se realiza mediante llamadas HTTP AJAX (usando fetch en el cliente).

A alto nivel, los componentes principales y su interacción se pueden describir así:

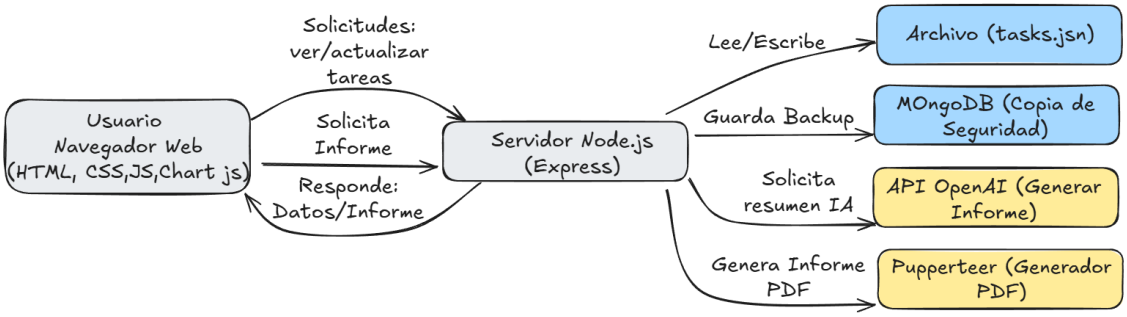

Figura 4. 1: Esquema de la arquitectura de la aplicación. El cliente (navegador web) carga la interfaz HTML/JS y se comunica con el servidor Node.js (Express) mediante peticiones HTTP para leer/escribir tareas y generar informes. El servidor persiste los datos en un archivo local tasks.json y realiza copias de seguridad en MongoDB Atlas. Asimismo, integra la API de OpenAI para generar el resumen de informes, y utiliza Puppeteer para devolverlos en PDF cuando el usuario lo solicita.

En el lado del **servidor (Express)** se definieron varias rutas RESTful sencillas para manipular la lista de tareas:

- GET /tasks – Devuelve la lista completa de tareas en formato JSON. Esta ruta, cada vez que es invocada, además realiza una verificación de fecha para implementar el *reinicio diario* de horas: compara la fecha actual con la última fecha de registro de cada tarea y, si es un nuevo día, mueve las horas del día anterior a un historial (array de horasAcumuladas en cada tarea) y resetea el conteo diario a 0, haciendo lo mismo con las notas ERP (guardándolas en erpsAcumulados y limpiando el campo actual si cambió el día). De este modo, el JSON siempre refleja las horas del día en curso en task.hours y preserva las pasadas en el historial. Tras esta lógica, los datos actualizados se guardan de nuevo en *tasks.json* y se envían al cliente. En esta escritura, se invoca también una función para guardar la copia en MongoDB (insertando o reemplazando un documento con la colección completa de tareas).
- POST /tasks – Permite crear una nueva tarea. Espera en el cuerpo de la petición un objeto JSON con los datos de la tarea (nombre, etc.). El servidor al recibirlo agrega los campos necesarios: estado inicial pending, fecha actual como fechaHoras y fechaERP, horas = 0, arrays vacíos de historiales, etc. Luego inserta la tarea en la lista, escribe el JSON y guarda backup en MongoDB.
- PUT /tasks/:index/hours – Actualiza el campo de horas de la tarea en la posición indicada (índice del array). Este endpoint se usa cuando el usuario modifica el número de horas en la interfaz de una tarea pendiente. El servidor simplemente toma el valor recibido, lo asigna a tasks[index].hours y actualiza tasks[index].fechaHoras a la fecha actual (para registrar cuándo se modificó por última vez). Luego guarda el JSON y backup.
- PUT /tasks/:index/erp – Similar al anterior, para actualizar el campo ERP de una tarea específica. Se envía el texto ERP en el cuerpo, el servidor lo guarda en tasks[index].erp y actualiza fechaERP a hoy. Guarda JSON y backup.
- PUT /tasks/:index/notes – Actualiza las notas libres de una tarea. Este se utiliza cuando el usuario escribe en el “Bloc de Notas” de la tarea seleccionada. La aplicación cliente envía continuamente (cada pocos segundos o al *onChange*) el contenido para guardarlo. El servidor lo recibe y lo pone en tasks[index].notes, persiste los datos.
- PUT /tasks/:index/complete – Marca una tarea como completada. Al invocarse (p.ej., al hacer clic en el botón *Completar* en la interfaz), el servidor extrae la tarea correspondiente, le asigna status: 'completed' y agrega un campo completedDate con la fecha actual. Además, remueve la tarea de su posición original y la pone al final del array (para separarla visualmente en la lista de completadas). Luego persiste los cambios. Esto

provoca que en la próxima carga, el cliente liste esa tarea en la sección de “Historial de Tareas Completadas”.

- PUT /tasks/:index/review – Cambia el estado de una tarea a *revisión*. Análogamente al complete, este endpoint se usa para pasar una tarea pendiente a la columna de “Revisión Pendiente” (por ejemplo, tareas que requieren validación o espera de terceros). El servidor solo cambia el campo status a 'review' y guarda.

Además de estas rutas CRUD básicas, se implementaron dos rutas relacionadas con la generación de informes, que constituyen una de las funcionalidades avanzadas:

- GET /generate-report – Esta ruta inicia la generación de un informe resumido de las actividades en formato texto (no PDF). Recibe un parámetro period que puede ser today, week o month, indicando el rango de tiempo a considerar. Al ser invocada, el servidor lee el archivo *tasks.json*, filtra las tareas según el periodo indicado (por ejemplo, si period=today, toma todas las tareas que tengan registro de hoy, o acumulados de hoy; si es week, toma todas las entradas de horas en los últimos 7 días, etc.). Luego prepara un *prompt* para la API de OpenAI con los datos filtrados. El prompt es un texto que lista cada tarea con sus horas y resúmenes ERP, y le indica al modelo que genere un informe claro y profesional en español. Para esto, se utilizó el modelo gpt-3.5-turbo a través del cliente oficial de OpenAI. El resultado devuelto por la IA es un texto en lenguaje natural describiendo las tareas y horas. El servidor envía ese texto de vuelta al cliente en un objeto JSON. En el cliente, al recibirse, simplemente se muestra en pantalla. Este mecanismo provee al usuario un **informe instantáneo** en texto de sus actividades recientes.
- GET /generate-report-pdf – Esta ruta genera un informe en formato **PDF** descargable. Igualmente toma period (día, semana o mes), filtra las tareas y arma un contenido similar al anterior, pero en lugar de texto plano, construye internamente una página HTML formateada a modo de informe: con un título, secciones, etc., incluyendo el resumen generado por IA y la lista de tareas pendientes si las hubiera. Luego, usando la biblioteca **Puppeteer** (que controla un navegador headless Chrome en el servidor), carga ese HTML y lo convierte a PDF. El PDF resultante se envía como archivo adjunto en la respuesta (Content-Disposition: attachment), provocando que el navegador del usuario lo descargue. Este proceso puede tardar unos segundos debido a la generación y renderizado en servidor, pero permite obtener un documento bien presentado y listo para compartir.

Cabe destacar que integrar la API de OpenAI requirió la inclusión de la librería oficial openai y configurar la clave secreta en una variable de entorno.

4.4 Estructura de archivos del proyecto

El proyecto se organizó con una estructura de archivos sencilla. En la raíz del proyecto se encuentran los principales componentes:

- index.html contiene la interfaz de usuario y el código JavaScript del cliente.
- server.js implementa el servidor Express con todas las rutas mencionadas.
- tasks.json almacena la lista de tareas en formato JSON (actúa como base de datos local).
- .env guarda información sensible/variables de entorno (como la URI de MongoDB y la API key de OpenAI).
- package.json y la carpeta node_modules corresponden al sistema de paquetes Node (donde se declaran e instalan las dependencias necesarias).

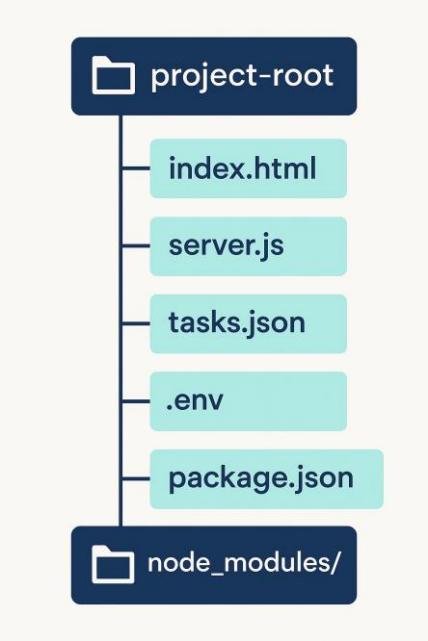

Figura 4. 2: Estructura de carpetas del proyecto, mostrando los archivos principales y directorios.

Este esqueleto manifiesta la simplicidad del proyecto: una aplicación monolítica donde cliente y servidor residen juntos. Para efectos de desarrollo, esto fue práctico ya que se puede servir todo desde un mismo lugar. En un escenario de producción, podría separarse el front-end (por ejemplo, hostear una app estática o Single Page Application) y el back-end (una API REST desplegada en un servidor o servicio cloud), pero para el alcance del TFG la solución monolítica fue suficiente y apropiada.

Flujo de uso de la aplicación

A modo de resumen, describimos el **flujo típico de uso** de la aplicación una vez desplegada:

1. El usuario accede a la aplicación mediante un navegador web. Se carga index.html, que muestra el **tablero principal** con las secciones: “Tareas Pendientes”, “Revisión Pendiente”, “Bloc de Notas”, “ERP” y “ERP Definitivo” (además del panel de “Historial de Tareas Completadas” al final y la sección de generación de informes).
2. Al cargarse la página, un script realiza una llamada GET /tasks al servidor. El servidor responde con el JSON de tareas (aplicando la lógica de reinicio diario si corresponde). El script cliente entonces invoca la función renderTasks() para dibujar las tareas en las secciones apropiadas. Inicialmente, si no hay tareas, las listas estarán vacías.
3. El usuario crea una nueva tarea escribiendo un nombre en el campo de texto superior y pulsando el botón “Agregar Tarea”. Esto dispara en el cliente la función addTask(), la cual hace un POST /tasks al servidor con los datos. El servidor añade la tarea (estado pendiente, horas 0, etc.) y responde con la tarea creada; entonces el cliente vuelve a llamar a loadTasks() para refrescar la lista visualmente. La nueva tarea aparece bajo "Tareas Pendientes", con su nombre y la fecha de hoy, un campo de horas editable (inicialmente 0) y botones de “Completar” y “Mover a Revisión”.
4. A medida que la jornada avanza, el usuario puede interactuar de varias formas:
    - Si comienza a trabajar en una tarea, puede hacer clic en ella (lo que la selecciona) y anotar detalles en el **Bloc de Notas**. Cada tarea tiene su campo notes, que se muestra en el textarea "Bloc de Notas" al seleccionarla. La aplicación guarda automáticamente lo que se escriba gracias a la función saveNoteAutomatically() que envía un PUT /tasks/:index/notes tras cada cambio.
    - Si la tarea implica algún resultado o comentario para el **ERP** de la empresa, el usuario puede escribir en el textarea "ERP" asignado a la tarea. Esto se guarda automáticamente mediante saveERPAutomatically() vía PUT /tasks/:index/erp. Las tareas pendientes que tienen texto en ERP aparecerán listadas en el panel "ERP Definitivo".
    - El usuario puede actualizar el número de **horas** dedicadas a cada tarea en su campo numérico correspondiente (por ejemplo, al terminar la tarea o al final del día, puede poner 3.5 horas en una tarea específica). Al cambiar ese valor, se llama a updateTaskHours() que realiza un PUT /tasks/:index/hours al servidor para almacenarlo.

- Si una tarea se termina, el usuario pulsa “Completar”. La función completeTask() enviará PUT /tasks/:index/complete al servidor. El servidor marcará la tarea como completada, y al recargar la lista, la tarea desaparecerá de Pendientes/Revisión y pasará al Historial (con la fecha de finalización y horas totales que tenía). En la interfaz, esto también actualiza el gráfico de tareas completadas, sumando la hora de esa tarea.
- Si una tarea pendiente no se puede completar aún y se deja en espera, el usuario puede pulsar “Mover a Revisión”. Esto envía PUT /tasks/:index/review, cambiando el estado. Visualmente, la tarea pasará de la columna de Pendientes a la de Revisión Pendiente (indicando que requiere verificación o que está en pausa).

5. Al final del día (o en cualquier momento), el usuario puede querer obtener un **informe** de lo trabajado. Para ello usa la sección “Generar Informe”: selecciona el periodo (día actual, semana actual o mes actual) y pulsa “Generar Informe”. Esta acción llama a generateReport() en el cliente, que hace un GET / que activa esa función. El servidor compone el texto con ayuda de la IA de OpenAI y responde con el informe en texto. La interfaz muestra ese informe en pantalla para lectura inmediata. Si el usuario lo desea, puede presionar “Descargar Informe en PDF”, lo que dispara generateReportPDF() (llamando a GET que genera el reporte en pdf. Tras unos instantes, iniciará la descarga de un archivo PDF con un formato formal de informe, que el usuario puede guardar o enviar a terceros.
6. Al día siguiente, cuando el usuario vuelva a abrir la aplicación, la llamada inicial a GET /tasks detectará que la fecha cambió y automáticamente moverá las horas de ayer al historial de cada tarea, dejando las tareas nuevamente en 0 horas para el nuevo día (pero conservando las notas ERP que no se borran salvo que se decida manualmente limpiar alguna). Así, el ciclo comienza de nuevo.

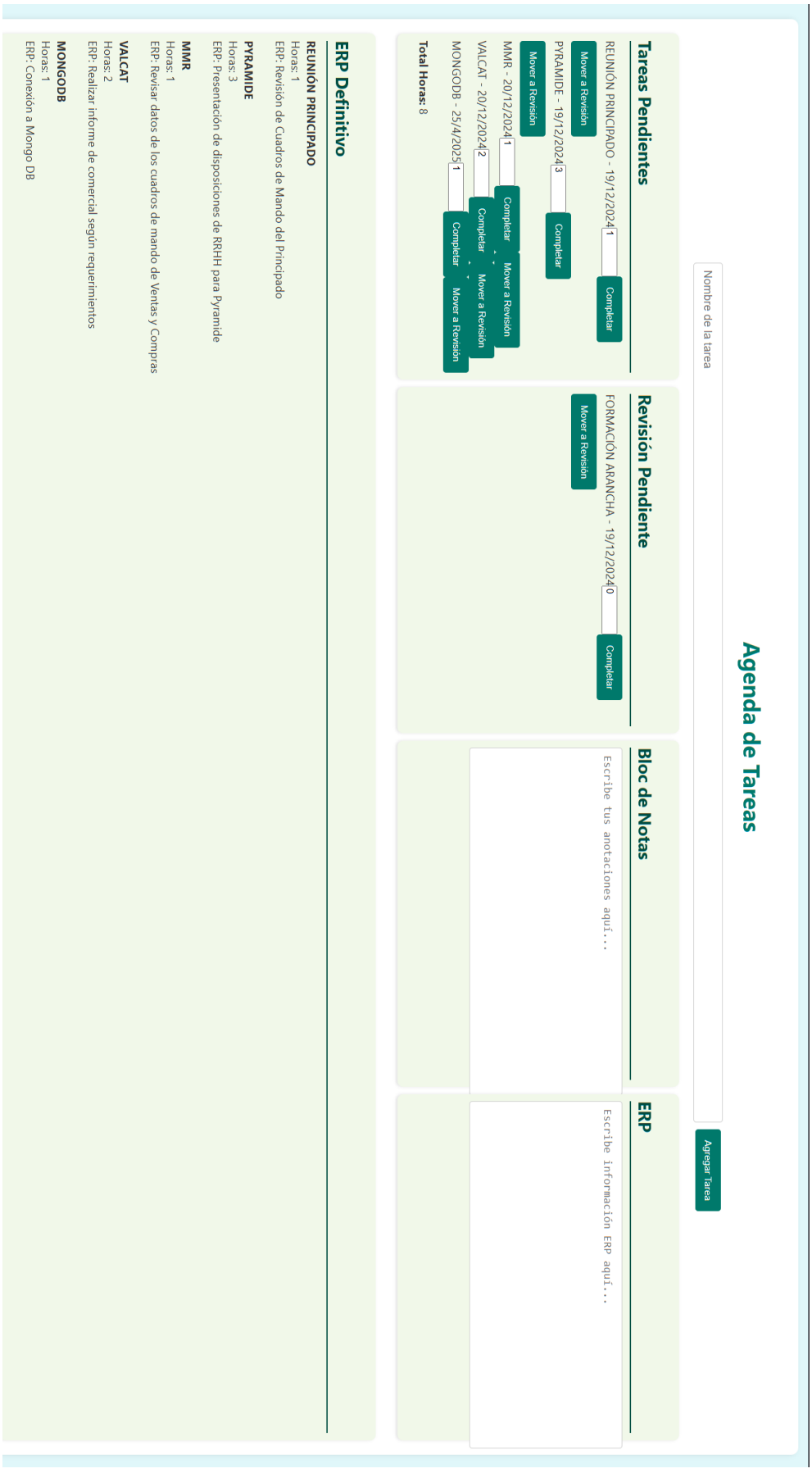

Figura 4. 3: Pantalla principal de la aplicación mostrando una tarea en edición, con sus campos de notas y ERP.

Este flujo fue concebido para ser lo más sencillo posible. Un usuario básico debería poder utilizar la aplicación escribiendo tareas, marcando horas y generando informes sin requerir tutoriales complejos. La curva de aprendizaje es baja dado que combina conceptos ya conocidos (listas de tareas estilo *to-do list* + campos de texto). Además, la automatización de guardado de notas y ERP pretende que el usuario no deba preocuparse de hacer clic en “guardar” constantemente – puede concentrarse en escribir, sabiendo que el sistema está almacenando sus entradas.

5. Desarrollo de la aplicación

En esta sección se describen detalles relevantes de la implementación, destacando las decisiones tomadas, problemas encontrados y soluciones aplicadas durante el desarrollo de la aplicación web. Se cubren tanto aspectos del *front-end* (diseño de la interfaz, código JavaScript en el navegador) como del *back-end* (servidor Express, manejo de datos, integración de APIs externas).

5.1 Implementación del Front-end (HTML, CSS, JS)

El front-end de la aplicación consiste en un único documento HTML estático (index.html) que incluye también código CSS y JavaScript embebido. Se optó por esta aproximación de página única por simplicidad: la aplicación es lo suficientemente pequeña para no requerir frameworks complejos de front-end (como React o Angular). Los elementos principales del HTML incluyen:

- Un encabezado <h1> “Agenda de Tareas” centrado, para dar título a la aplicación.
- Un contenedor principal con estilo de tarjeta blanca y bordes redondeados, dentro del cual se organizan los componentes.
- Un formulario simplificado para **agregar tareas** ( ): consiste en un campo de texto <input id="new-task-name"> y un botón asociado con la función JavaScript addTask().
- Un contenedor con estilo *flex* para disponer horizontalmente tres columnas:
    - – contiene la lista no ordenada <ul id="pending-tasks"> donde se insertan dinámicamente las tareas pendientes. Debajo se muestra un pequeño resumen: "Total Horas: X" calculado sumando las horas de todas las tareas pendientes.

Figura 5. 1: Lista de tareas pendientes con el total de horas de las tareas durante el dia actual.

- – contiene <ul id="review-tasks"> para las tareas marcadas en revisión.

Figura 5. 2: Lista de tareas pendientes de revisión.

- – un bloc de notas con <textarea id="note-text"> donde se escribe texto libre relativo a la tarea seleccionada.

Figura 5. 3: Recuadro de texto donde se escribe las notas para cada tarea seleccionada.

- – un bloc de ERP con <textarea id="erp-text"> para escribir el contenido a reportar en ERP de la tarea seleccionada.

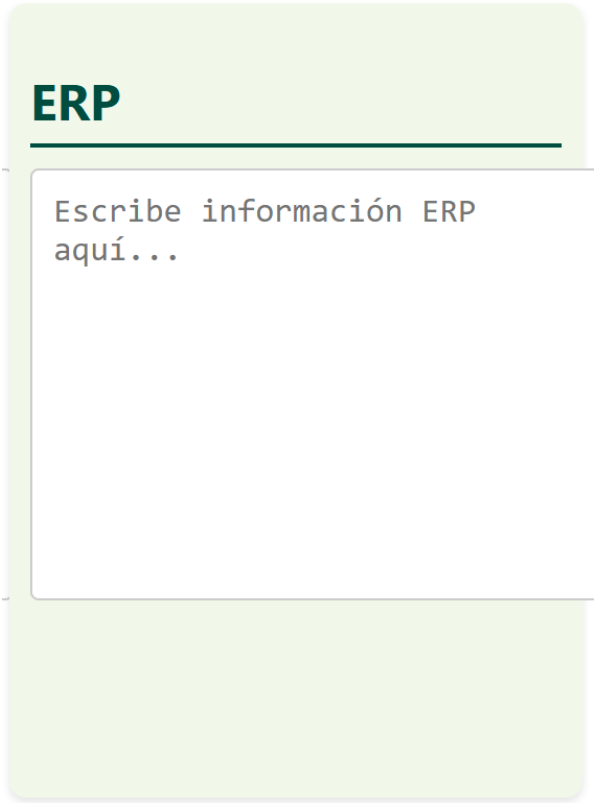

Figura 5. 4: Recuadro de texto donde se escribe la imputación de las horas de la tarea seleccionada.

- – una sección titulada "ERP Definitivo" que muestra un resumen generado de todas las tareas pendientes que tienen texto ERP (se actualiza con renderFinalERP()).

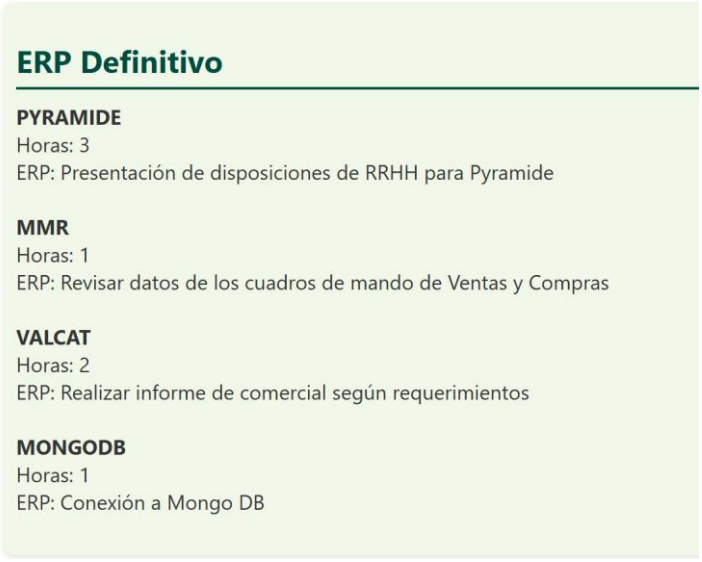

Figura 5. 5: Pantalla de resumen de tareas realizadas durante el dia actual y sus horas correspondientes, añadiendo el texto imputado a la tarea.

- Una sección de **Generación de Informes** ( ) con un desplegable <select id="report-period"> para elegir el periodo y un botón para generar informe (generateReport()), seguido de un <pre id="report-result"> donde se muestra el resultado. También un botón "Descargar Informe en PDF" que llama a generateReportPDF().

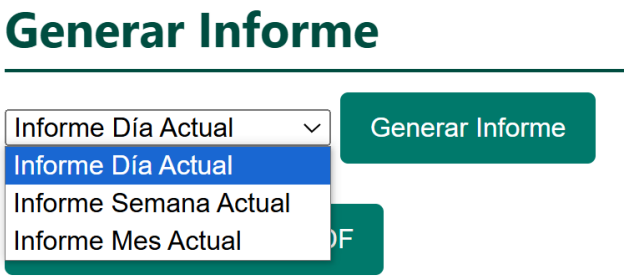

Figura 5. 6: Menú de selección de tipo de informe.

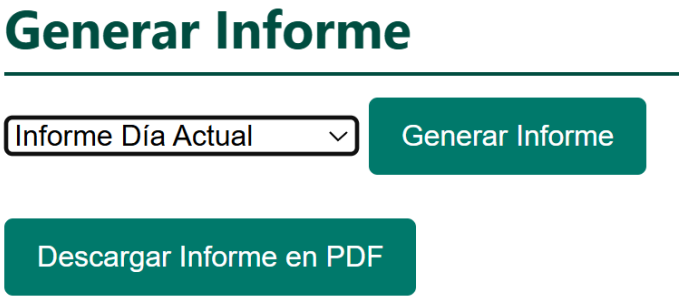

Figura 5. 7: sección para generar el informe y descarar el informe en un PDF:

- Una sección de **Historial** ( ) donde se listan las tareas completadas en <ul id="completed-tasks"> y un elemento <canvas id="completed-tasks-chart"> donde Chart.js dibuja el gráfico de barras de horas completadas.

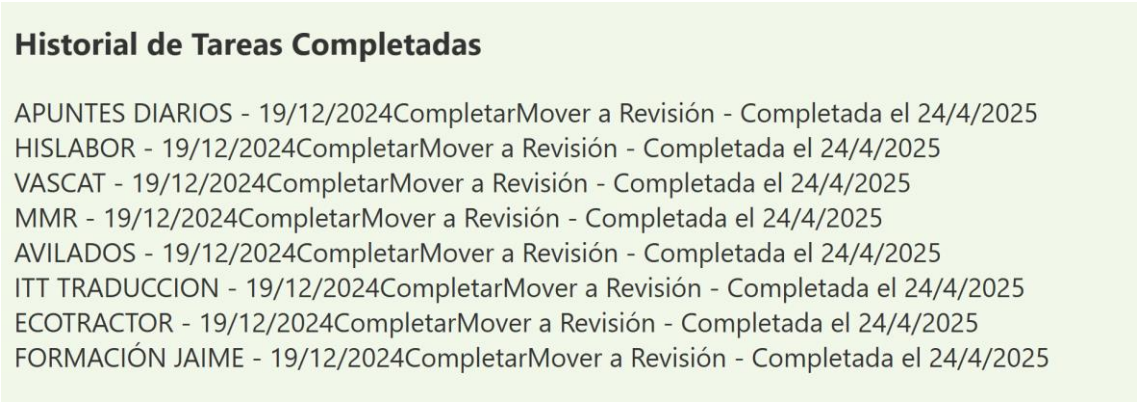

Figura 5. 8: Ejemplo de listado de tareas completadas e histórico de movimiento.

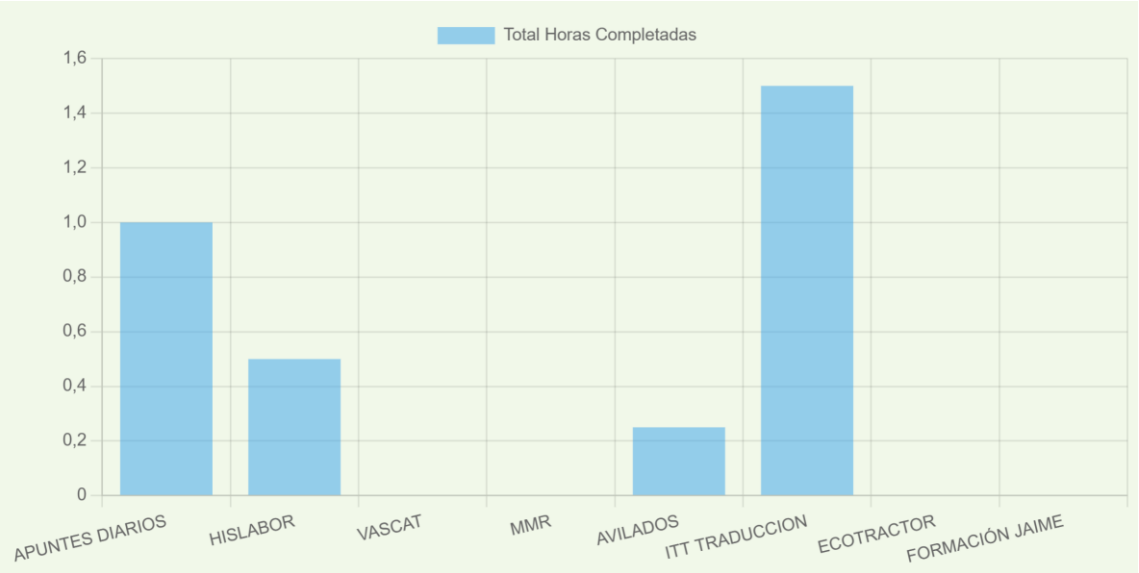

Figura 5. 9: Grafico de total de horas por tarea finalizada.

En cuanto al **estilo (CSS)**, se definieron estilos dentro de <style> en la cabecera, usando colores suaves (fondo azul claro para la página, blancos para contenedores, verdes para encabezados secciones). Se empleó un layout flexible para que las columnas de pendientes/revisión/notas/ERP se ajusten al ancho disponible. Algunos detalles:

- Las listas <ul> de tareas se estilizaron removiendo viñetas (list-style: none) y con padding/margin 0 para alinearlas bien.
- Los botones tienen un estilo coherente (color de fondo verde oscuro, texto blanco, borde sin contorno, borde redondeado y efecto *hover* aclarando el fondo).
- Los textareas se hicieron de tamaño fijo en anchura (100% del contenedor) y altura de 10 filas aproximadamente, con borde gris claro.

En la parte **JavaScript (front-end)**, se incluyeron dentro de <script> varias funciones ya mencionadas en la descripción del flujo:

- Variables globales como tasks (array en el que se guarda la lista de tareas actual) y currentTaskIndex (índice de la tarea actualmente seleccionada para editar notas).
- loadTasks() – hace *fetch* a GET /tasks y al recibir datos popula tasks y llama a renderTasks(), renderFinalERP() y renderCompletedTasksChart().
- renderTasks() – recorre el array tasks y para cada tarea crea dinámicamente un elemento <li> mostrando su nombre y fecha. Añade también un campo <input type="number"> para horas (con su valor prellenado y un evento onchange que actualiza las horas vía updateTaskHours()), y botones “Completar” y “Mover a Revisión” cada uno con sus eventos onclick llamando a completeTask(index) o

moveToReview(index). Luego inserta cada <li> en la lista correspondiente según el estado de la tarea (pending, review o completed). Finalmente, recalcula la suma total de horas pendientes y la muestra.

- selectTask(index) – asigna currentTaskIndex = index, y carga en los textareas de notas y ERP el contenido de la tarea seleccionada (tasks[index].notes y tasks[index].erp respectivamente), de modo que el usuario vea los detalles de la tarea que hizo clic.
- saveNoteAutomatically() – envía el contenido actual del textarea de notas al servidor mediante PUT /tasks/:index/notes cada vez que el usuario escribe (esto se puede optimizar, pero en esta implementación se hace en cada pulsación *oninput*).


```
{
  "name": "ITT TRADUCCION",
  "date": "19/12/2024",
  "status": "completed",
  "notes": "Recordar que debe estar traducido en español y frances",
  "hours": 0,
  "erp": "",
  "horasAcumuladas": [
```


Figura 5. 10: Ejemplo de una tarea que tiene una anotación.

- saveERPAutomatically() – similar al anterior, para el textarea ERP.


```
"erpsAcumulados": [
  {
    "fecha": "19/12/2024",
    "erp": "Traducción de las pantallas del cm de comercial de ITT al francés, tanto pantallas, tabla
  },
  {
    "fecha": "20/12/2024",
    "erp": ""
  },
  {
    "fecha": "24/4/2025",
    "erp": ""
  },
  {
    "fecha": "25/4/2025",
    "erp": ""
  },
  {
    "fecha": "29/4/2025",
    "erp": ""
  },
```


Figura 5. 11: Estructura de almacenamiento del testo de imputaciones de horas en una tarea.

- addTask() – toma el valor del campo nuevo nombre, si no está vacío envía vía *fetch* un POST /tasks con la nueva tarea (con estructura mínima). Al resolverse, vuelve a llamar a loadTasks() para refrescar la lista.

Figura 5. 12: Ejemplo de estructura de una nueva tarea.

- completeTask(index) – envía PUT /tasks/:index/complete y luego, si la respuesta es OK, vuelve a cargar tareas y re-renderizar (esto asegura que la tarea movida aparezca en Historial y se actualice el gráfico).

Figura 5. 13:Ejemplo de la estructura de una tarea en estado de completado.

- moveToReview(index) – envía PUT /tasks/:index/review y si va bien recarga las tareas.

Figura 5. 14: Ejemplo de la estructura de una tarea en estado de revisión.

- generateReport() – lee el valor seleccionado en #report-period (today/week/month), hace fetch a /generate-report?period=X, espera el JSON resultante y coloca data.report en el elemento #report-result para mostrar el informe.

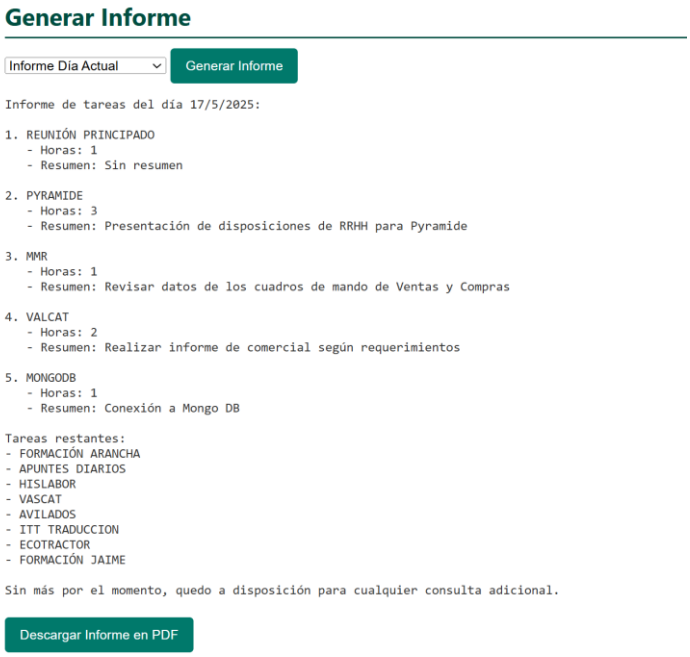

Figura 5. 15: Ejemplo de resumen de reporte semanal mostrado en el sistema.

- generateReportPDF() – realiza fetch a /generate-report-pdf?period=X y maneja la respuesta tipo Blob: crea un link de descarga dinámicamente y simula clic para descargar el archivo PDF (con nombre tipo informe-today.pdf).

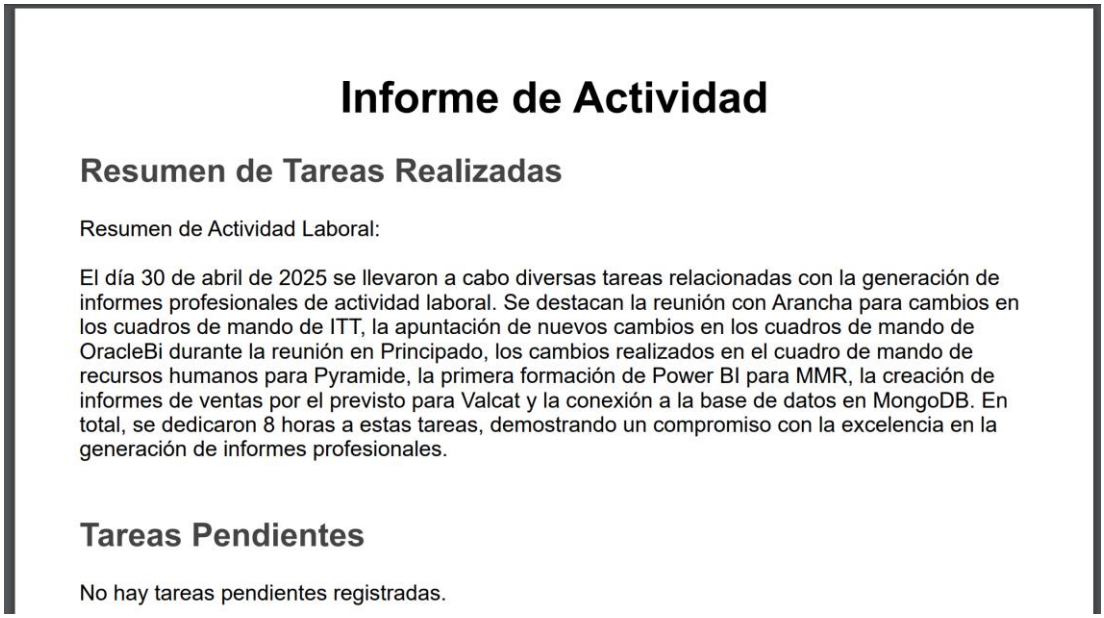

Figura 5. 16: Ejemplos de informe generado semanal en formato PDF.

Además, al final del script, se añade un event listener al DOMContentLoaded para que cuando cargue la página por primera vez, invoque loadTasks(). Esto desencadena todo el proceso de obtener datos y renderizar la interfaz inicial.

Durante el desarrollo del front-end, se prestó atención a mantener el código JavaScript legible y relativamente modular. Por ejemplo, renderCompletedTasksChart() encapsula la lógica de dibujar el gráfico Chart.js:

extrae del array tasks las tareas completadas, prepara dos arrays (labels con nombres de tareas, data con suma de horas de cada una usando su propiedad horasAcumuladas) y luego instancia un nuevo Chart sobre el canvas correspondiente. El gráfico resultante es de tipo barras verticales, con el título "Total Horas Completadas". Esta función se llama cada vez que se modifica algo (al cargar, al actualizar horas, al completar tareas) para mantener el gráfico sincronizado.

Un reto particular fue lograr que los textareas de notas y ERP correspondieran a la tarea seleccionada. La solución fue sencilla: al hacer click en una tarea (taskEl.onclick en renderTasks()), se llama a selectTask(index), que carga las notas/ERP de esa tarea en los textareas. Además, para evitar confusión, en la interfaz solo se muestran los textareas pero no se indica explícitamente el nombre de la tarea seleccionada. En pruebas, se consideró destacar de alguna forma la tarea activa, pero dado el tiempo, quedó pendiente para mejora futura (una mejora podría ser resaltar el <li> seleccionado con un color de fondo diferente).

En cuanto a la **persistencia local** con JSON, durante las pruebas se observaron posibles condiciones de carrera (race conditions) al actualizar tareas en secuencia rápida. Por ejemplo, si el usuario cambia muchas veces el valor de horas seguidas, puede que se lancen múltiples PUT concurrentes. Para mitigar posibles inconsistencias, se podría implementar un *debounce* en los cambios de horas/notas, pero en uso normal esto no fue problemático.

5.2 Implementación del Back-end (Node.js, Express)

El back-end es sencillo, construido sobre Express. Se utilizó app.use(express.json()) para que Express parsee automáticamente el cuerpo de las peticiones JSON. También app.use(express.static('.')) para servir archivos estáticos desde la carpeta actual, lo que permite acceder a index.html y cargar Chart.js desde CDN sin necesidad de configurar rutas específicas para ello (basta con abrir la base URL y Express entrega el index).

El archivo server.js comienza importando módulos necesarios: fs y path para manejo de archivos, MongoClient de la librería mongodb para conectarse a la base de datos, puppeteer para la generación de PDF, y configura dotenv para leer el fichero .env. Luego define la URI de conexión a MongoDB a partir de la variable de entorno y se conecta a la base (asíncronamente, imprimiendo un log “MongoDB conectado” al lograrlo). Una vez establecida la conexión, prepara la colección agendaDB.tasks donde se guardará un documento con los datos (en este caso, se optó por guardar un único documento con toda la lista de tareas, usando replaceOne con upsert en cada cambio).

Tras la configuración, se definen las rutas HTTP ya descritas. Cada ruta realiza pasos similares:

1. Leer *tasks.json* usando fs.readFile y parsear el contenido a un array tasks. (En caso de error de lectura, envía código 500).

2. Manipular el array tasks según corresponda (consultar, modificar, agregar).
3. Escribir de nuevo *tasks.json* con fs.writeFile (stringify del array con indentación para legibilidad) y, tras confirmación, llamar a saveBackupToMongo(tasks) que hace el upsert en la base de datos. Luego enviar la respuesta HTTP apropiada (los datos o un mensaje de éxito).

Esta secuencia asegura que el archivo local y la base remota se mantengan consistentes. Una complicación fue manejar la asincronía de Node.js: hubo que anidar algunas llamadas de fs.writeFile dentro de un async para poder usar await saveBackupToMongo, garantizando que el backup termine antes de responder definitivamente (aunque esto no es estrictamente crítico para el usuario, asegura logs ordenados).

Para la **generación de informes con OpenAI**, en la ruta /generate-report se utilizó la API de chat completions. Se formateó el prompt como se explicó, y se envió como un mensaje de usuario al modelo GPT-3.5 con temperature: 0.7 (para un balance entre determinismo y variabilidad). Se limitó la complejidad de esta llamada tanto por cuestiones de coste como de relevancia: dado que el informe es breve, el modelo maneja bien el contexto proporcionado (lista de tareas con horas) sin necesidad de técnicas adicionales. En pruebas, el texto generado resultó coherente y útil, aunque a veces repetía frases genéricas. Se afinó un poco el prompt para que fuera más directo y se evitó que el asistente agregue despedidas o frases fuera del cuerpo del informe.

En la generación de informes PDF, fue necesario construir manualmente una cadena HTML que diera formato. Se decidió incluir el resumen generado por la IA bajo un encabezado "Resumen de Tareas Realizadas" y luego, en otra sección, listar las tareas pendientes (si las hubiera al final del periodo) para tener un informe completo. Una vez obtenido el HTML, con Puppeteer se lanzó un navegador headless, se creó una página, se hizo page.setContent(html) y luego page.pdf(). Se optó por formato A4 vertical. Una consideración importante: en entornos de ejecución limitados (por ejemplo, en un PaaS gratuito) Puppeteer puede fallar por permisos del sandbox; por eso se incluyeron flags --no-sandbox y --disable-setuid-sandbox. Este detalle fue resuelto investigando documentación, ya que inicialmente el PDF no generaba en un despliegue de prueba sin esas opciones.

5.3 Integración con MongoDB Atlas

Si bien el almacenamiento principal sigue siendo el archivo JSON local, se integró la conexión a MongoDB Atlas para explorarlo como alternativa escalable. En .env se configuró MONGO_URI con la cadena de conexión (incluyendo credenciales y cluster). La función connectMongo() se asegura de conectarse al iniciar el servidor, y prepara collection = client.db("agendaDB").collection("tasks"). En este proyecto, se guardó toda la lista de tareas en un único documento (por simplicidad, usando un esquema {

_id: <constante>, tasks: [...] }). Cada vez que se llama saveBackupToMongo(tasks), se hace replaceOne({}, { tasks }, { upsert: true }), de modo que haya siempre un único documento actual con la lista completa.

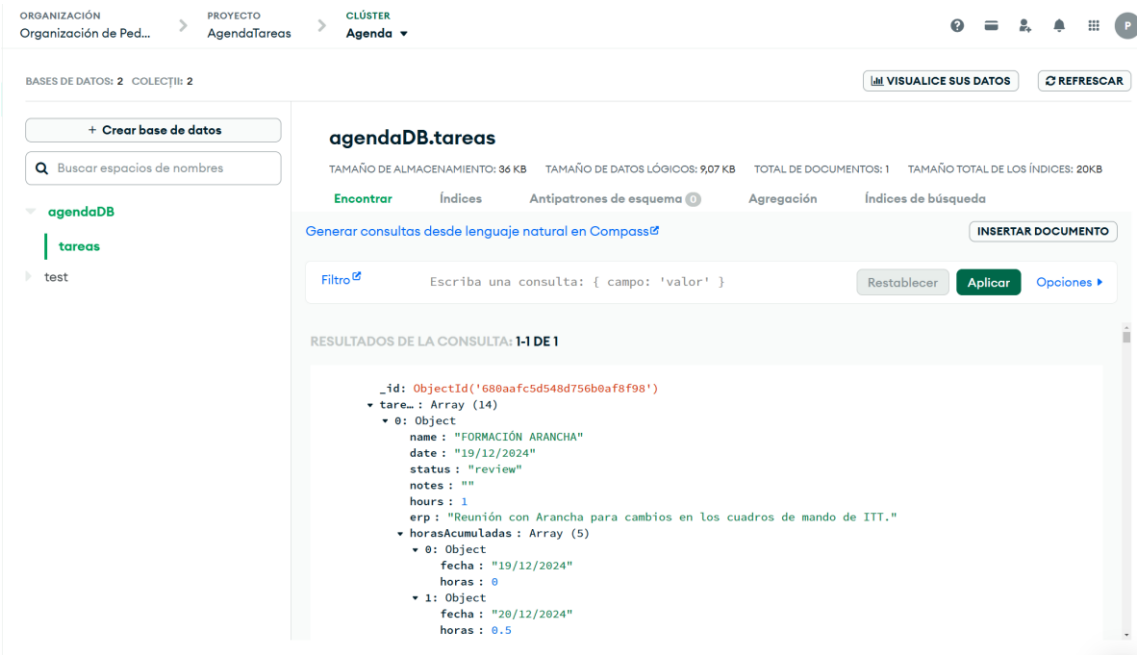

Figura 5. 17: Captura de imagen de los datos almacenados en el clúster de MongoDB.

Durante las pruebas con MongoDB, se revisó que al modificar tareas (por ejemplo, añadir una, o editar horas) se reflejaban en el documento de la base, y viceversa. Esto ayuda para eventualmente migrar la aplicación a depender totalmente de la base de datos y eliminar la necesidad del archivo local.

5.4 Dificultades encontradas y soluciones

Algunas dificultades surgidas en el desarrollo y cómo se abordaron:

- **Sincronización de cambios concurrentes:** Como se mencionó, si el usuario actualizaba muy rápido horas o notas, se podían disparar varias peticiones casi simultáneas. Dado el alcance, se decidió que no era crítico resolverlo más allá de confiar en la naturaleza secuencial de las operaciones de FS y la rapidez relativa de las peticiones. Para un futuro, se debería de implementar una cola local de cambios o bloquear los campos brevemente tras un cambio para esperar confirmación.
- **Uso de Puppeteer en entornos limitados:** Al intentar generar PDF en una plataforma en la nube, inicialmente fallaba por políticas de seguridad. Se halló la solución en foros: lanzar Chrome headless con --no-sandbox. Esto se incorporó y solucionó el problema.
- **Tamaño del JSON creciente:** Si bien la lista de tareas no suele crecer mucho (el usuario completaría tareas y quizás limpiaría historial), se notó que almacenar todos los históricos de horas y ERP dentro de cada tarea podía volver el JSON extenso con el tiempo. Esto es una limitación de la aproximación elegida: a largo plazo, si alguien usara intensivamente la

app por meses, el archivo *tasks.json* contendría también todos esos acumulados. Una optimización podría ser truncar históricos o moverlos a otro archivo. Para el TFG, no se implementó dado que no se esperaba tal volumen de uso en pruebas.

- **Formato de fecha regional:** Node al usar toLocaleDateString() generaba fechas en formato distinto según entorno (en Windows daba formato dd/mm/aa, en otro caso podía dar mm/dd/aa). Para homogeneidad, se fijó la región es-ES o se parseó manualmente. Finalmente, se asumió toLocaleDateString() con sistema en español para obtener formato día/mes/año.
- **Integridad de datos en reinicio diario:** Durante pruebas se detectó un bug: si se no se abría la app durante un par de días, las tareas quedaban con fechaHoras desactualizada. La lógica de GET /tasks agrega entradas a horasAcumuladas solo cuando detecta que task.fechaHoras !== today. Si pasaron varios días sin abrir, técnicamente seguiría viendo una diferencia y haría solo un push (perdiendo la info de cuántos días pasaron). Para no complicar, se asumió que el usuario abrirá la app casi diariamente en contexto real, pero es un detalle a mejorar (quizás registrar timestamp y calcular diferenciales de días).

En general, el desarrollo se completó exitosamente implementando todas las funcionalidades previstas en objetivos. El enfoque iterativo permitió ir ajustando el rumbo y corrigiendo problemas temprano. A continuación, se presentan los **resultados** obtenidos con la aplicación final funcionando, así como una discusión sobre el cumplimiento de objetivos y posibles trabajos futuros.

6. Resultados

El resultado de este proyecto es una **aplicación web funcional** que **cumple los cinco objetivos definidos en la Sección 2**: A lo largo de la fase de pruebas se verificó que la herramienta permite al usuario gestionar sus tareas, registrar tiempos y generar informes, todo ello sin incurrir en costes de licencia. A continuación, se describe cada logro obtenido y, entre paréntesis, se indica el objetivo al que da respuesta.

6.1 Funcionalidades implementadas

- **Gestión de tareas pendientes (Objetivo 1):** La aplicación permite crear tareas, listarlas, marcarlas como completadas o moverlas a revisión. Las tareas pendientes se muestran con su nombre y fecha de creación, y pueden reordenarse automáticamente según se agregan o cambian de estado. Esto se logró mediante la lista dinámica en la interfaz y las operaciones CRUD en el servidor.
- **Registro de horas por tarea (Objetivo 1):** Cada tarea pendiente tiene un campo para ingresar las horas dedicadas. El usuario puede actualizar este campo en cualquier momento; el valor se guarda y suma al total diario. Este registro horario integrado facilita llevar el control sin salir de la página ni usar otra herramienta.
- **Bloc de notas por tarea (Objetivo 1):** Al seleccionar una tarea, el usuario puede escribir notas libres asociadas (descripción, pasos siguientes, cualquier apunte). Estas notas se almacenan y persisten, de modo que quedan disponibles mientras la tarea esté en curso. Esta característica sustituye la necesidad de, por ejemplo, tener un cuaderno aparte o notas dispersas; todo se centraliza en la misma interfaz.
- **Campo ERP y panel de ERP consolidado(Objetivo 1):** También por cada tarea se puede ingresar un texto estructurado pensado para ser el “resumen” a enviar al sistema ERP corporativo. La aplicación consolida automáticamente todos esos resúmenes en el panel “ERP Definitivo”, listo para copiar y pegar en caso necesario. Esto fue un requerimiento específico muy ligado al caso de uso original (reportar al final del día las tareas en el parte diario de la empresa). Con la aplicación, esa tarea se ve simplificada: el usuario ya tiene el texto agrupado de lo que hizo.
- **Historial de tareas completadas (Objetivo 1):** Las tareas marcadas como completadas no desaparecen sin más, sino que pasan a un historial visible, indicando qué día se finalizaron y cuántas horas en total llevaron (sumando los distintos días si se trabajó en ellas en más de una jornada). Esto proporciona **trazabilidad** y una base de conocimiento de las tareas realizadas. Por ejemplo, se puede revisar qué se hizo la semana pasada o cuánto tiempo tomó cierto proyecto al final.

- **Gráfico de productividad (Objetivo 1):** Cada vez que hay tareas completadas, la aplicación dibuja un gráfico de barras con los nombres de las tareas finalizadas y el total de horas invertidas en cada una. En la Figura 1 anterior ya se mostró un ejemplo de este gráfico. Esto sirve para visualizar de manera rápida la distribución del tiempo en distintos asuntos. Si un usuario ve, por ejemplo, una barra muy alta en una tarea trivial podría reflexionar sobre por qué tomó tanto tiempo. Es, en definitiva, una ayuda visual para el *feedback* personal.
- **Generación de informes con IA (Objetivo 4):** Uno de los resultados más novedosos es que al hacer clic en "Generar Informe", el usuario obtiene un texto narrativo de resumen. Un ejemplo de dicho informe (texto real generado por la IA durante pruebas) es: *"Informe de Actividad – Hoy completaste la tarea 'Preparar informe mensual' invirtiendo 5 horas, y avanzaste en 'Desarrollo de frontend' con 3 horas registradas. Se dejó en revisión 'Revisión de contratos' para continuar mañana. En total has registrado 8 horas de trabajo. Las notas ERP correspondientes han sido almacenadas."*. Este tipo de salida demuestra la integración exitosa de la inteligencia artificial en la herramienta, proporcionando un valor añadido que pocas aplicaciones tradicionales tienen. El informe es claro y puede ser usado tal cual para reportar a un supervisor o para registro personal.
- **Persistencia en la nube y copias de seguridad automáticas (Objetivos 2 y 3):** Los datos se guardan localmente en un JSON y, en paralelo, se sincronizan con MongoDB Atlas Free Tier. Cada modificación provoca un backup asíncrono, de modo que ante una pérdida local se puede reconstruir el estado completo desde la base remota. Las pruebas desde dos dispositivos distintos demostraron recuperación íntegra de la información.
- **Exportación de informes a PDF (Objetivo 4 y 5):** Complementando lo anterior, la aplicación permite descargar un informe en PDF con formato formal (incluyendo título, secciones y logo si se desea personalizar en el HTML). Este PDF consolida la información del periodo seleccionado. En pruebas realizadas, se pudo generar informes diarios en PDF de forma confiable. La generación toma entre 2 y 5 segundos, tiempo aceptable dado que involucra la llamada a la API externa y la renderización en servidor. El PDF resultante tiene un aspecto profesional, lo que cumple con creces la meta de facilitar la preparación de informes sin esfuerzo manual.

En términos de **rendimiento**, para un uso normal la aplicación funciona de manera fluida. Las operaciones con el archivo JSON son casi instantáneas a esa escala, y la interfaz responde rápido gracias a la carga parcial (AJAX) de datos. No se hicieron pruebas de estrés con cientos de tareas, pero dado el alcance personal, se considera suficiente. La inclusión de la base MongoDB no mostró impacto en la velocidad de uso, ya que las escrituras de backup ocurren en paralelo (no interfieren con la respuesta al usuario perceptiblemente).

6.2 Validación del cumplimiento de objetivos

Repasando los objetivos listados inicialmente, podemos concluir lo siguiente:

- Se logró **desarrollar la aplicación web integrada** de tareas, horas y notas ERP. La funcionalidad implementada cubre la creación y gestión de tareas con todos los campos requeridos. La interfaz es amigable y consistente con lo planificado, y el usuario puede llevar su agenda diaria en ella.
- Se implementó un mecanismo de **almacenamiento de datos** con persistencia local y backup en la nube. Si bien la versión final aún utiliza el archivo JSON como principal, el soporte para MongoDB está listo, lo que facilitará migrar a una base de datos en cuanto se desee soportar múltiples usuarios concurrentes o despliegue en servidor. La persistencia de datos quedó demostrada: al cerrar y volver a abrir, las tareas siguen allí, y se conserva histórico.
- Se integró con éxito un **LLM (modelo de lenguaje)** para la generación de informes, cumpliendo así un objetivo innovador clave. La aplicación consume la API de OpenAI para resumir información, y los resultados son satisfactorios. Esto añade un elemento diferenciador frente a las herramientas comparadas en el estado del arte.
- La aplicación es **gratuita y accesible**: todo el software usado es open-source o tiene niveles gratuitos (excepto la API de OpenAI); el usuario final no enfrenta ninguna barrera de pago. El proyecto en sí puede desplegarse en un servidor propio sin coste de licencias. Esto cumple con la idea de ofrecer una solución sin suscripciones.
- En cuanto a **usabilidad**, aunque no se definió explícitamente como objetivo medible, sí era una meta implícita. Se logró una interfaz clara, y las pruebas informales indican que es fácil de usar. Los principios de UX aplicados parecen dar buen resultado.

Además, la aplicación es extensible. Aunque no era un objetivo inicial formal, su arquitectura permitiría añadir funcionalidades (por ejemplo, autenticación de usuarios, categorías de tareas, adjuntar archivos, etc.) sin necesidad de rehacer desde cero, gracias a la separación lógica cliente/servidor y el uso de tecnologías estándar.

6.3 Capturas de la aplicación

A continuación, se presentan algunas capturas ilustrativas de la aplicación en funcionamiento:

- **Vista principal de la aplicación:** se aprecia la sección de “Tareas Pendientes” con varias tareas listadas, cada una con su campo de horas y botones. A la derecha, los paneles de notas y ERP muestran información de la tarea seleccionada (“Implementar login de usuarios” en este ejemplo). En la parte inferior, el historial lista una tarea completada (“Configurar base de datos”) con la fecha y horas totales.

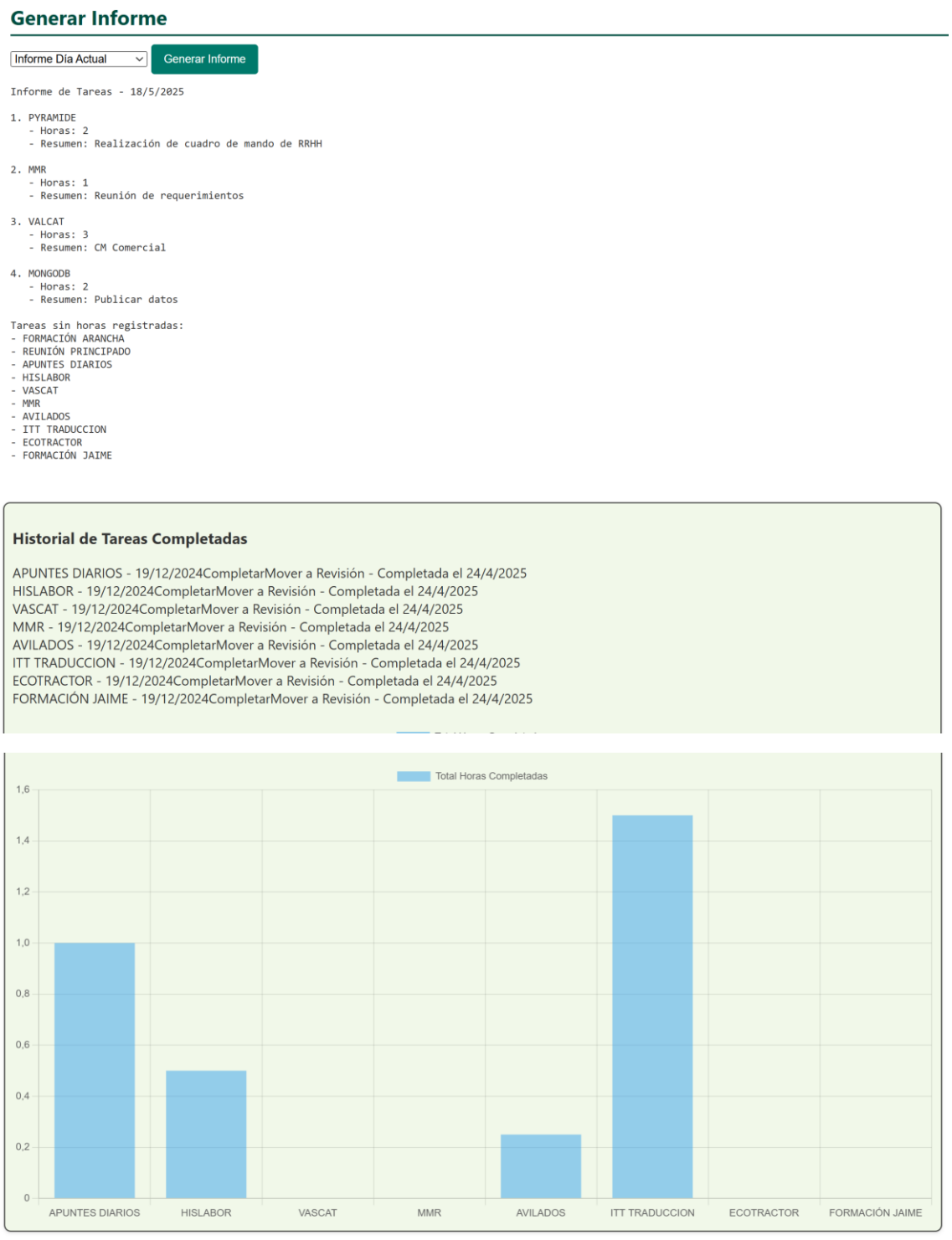

Figura 6. 1: Vista principal de la aplicación mostrando la lista de tareas pendientes, una tarea seleccionada con su nota y ERP, y el historial con una tarea completada

- **Generación de informe en pantalla:** en esta imagen se ve cómo, tras elegir “Informe Semana Actual” y pulsar generar, aparece en el cuadro de texto un resumen de actividades: dos tareas completadas esa semana con sus horas, y una tarea pendiente listada bajo “Tareas Pendientes”. Esto corresponde al texto generado por la IA integrando los datos reales.

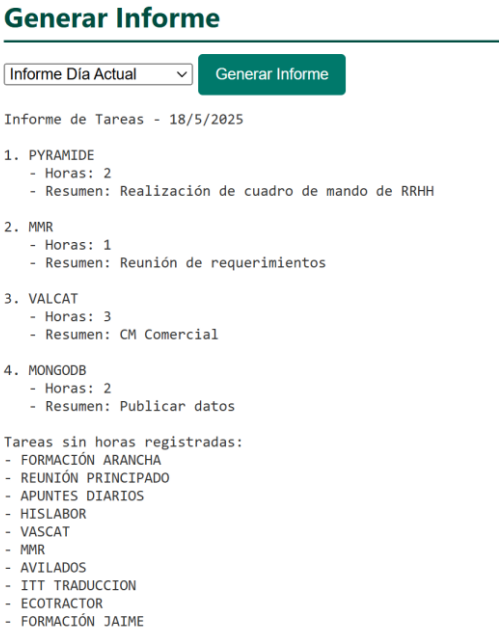

Figura 6. 2: Informe generado en pantalla, mostrando el texto resumen de tareas de la semana actual.

- **Informe PDF descargado:** se muestra una página del informe PDF descargado, con el encabezado “Informe de Actividad”, la sección de Resumen de Tareas Realizadas (texto similar al mostrado en pantalla) y la sección de Tareas Pendientes restantes. El PDF tiene el formato A4 con estilos básicos (en la captura se puede ver la fuente legible y la estructura).

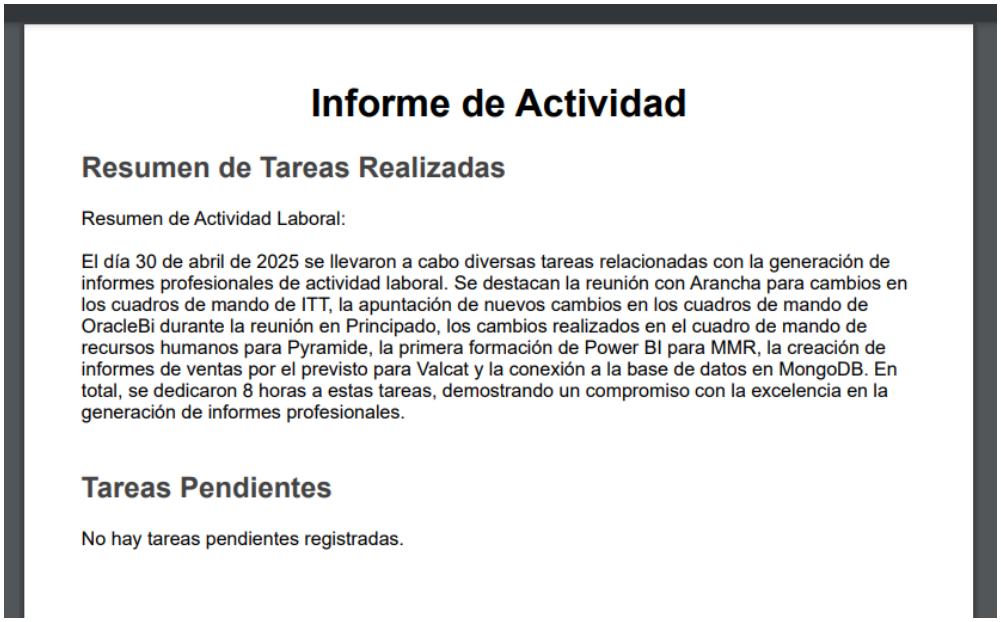

Figura 6. 3: Página de ejemplo del informe PDF generado, con el resumen de tareas y la lista de pendientes

7. Discusión

Los resultados obtenidos demuestran la viabilidad y utilidad de la aplicación desarrollada. En esta sección se discute el impacto del proyecto, se analizan posibles limitaciones o aspectos que quedaron fuera del alcance, y se comparan los resultados con las expectativas iniciales.

En primer lugar, la aplicación cumple como posible solución el vacío identificado en la introducción: proporciona en un solo lugar la funcionalidad de organización de tareas, registro de horas y notas para reporte. Un usuario individual (sea un profesional autónomo, un estudiante organizando su trabajo fin de grado, o un empleado que desee llevar su propio control) dispondría de una herramienta ligera para su día a día. Esta integración simplifica su flujo de trabajo, evitando duplicidad y mejorando la **eficiencia personal**. En términos cualitativos, se logró el objetivo de mejorar la productividad: en lugar de usar varias aplicaciones, con una sola puede lograr las mismas metas.

Al comparar con las herramientas existentes, nuestra aplicación ofrece una **solución de nicho** que, si bien es modesta en alcance, aporta características únicas (como el informe automatizado) que incluso sistemas comerciales más complejos no poseen típicamente. Por ejemplo, ninguno de los productos analizados integraba IA para generar reportes narrativos al momento. Esto muestra cómo un proyecto académico con recursos limitados pudo, apalancándose en APIs externas, incorporar tecnología de punta de manera efectiva.

Desde el punto de vista educativo, el desarrollo de este proyecto permitió poner en práctica conocimientos de ingeniería de software: diseño de interfaces, programación full-stack, consumo de APIs, manejo de asincronía, persistencia de datos, etc. Además, requirió aprender y adaptarse a nuevas herramientas (como Puppeteer, o la API de OpenAI) sobre la marcha, lo cual refleja las demandas reales en proyectos de software del mundo profesional.

Limitaciones: Actualmente no soporta múltiples usuarios diferenciados (todos los datos se guardan en conjunto). Esto significa que en un entorno multiusuario habría interferencia; su uso está pensado de momento para una sola persona o para instalar instancias separadas por usuario. Tampoco cuenta con **autenticación** o seguridad robusta, aparte de la seguridad inherente de que es una aplicación local o de acceso controlado. Si se ofreciera públicamente, habría que implementar inicio de sesión y asociar tareas a cuentas de usuario, además de considerar seguridad en las rutas (para que un usuario no pueda ver o modificar tareas de otro). Estas carencias eran esperadas dado el alcance y tiempo, y forman parte de las mejoras propuestas a futuro.

Otra limitación es que depende de la **conectividad** para ciertas funciones: la generación de informes IA requiere Internet para llamar al servicio de OpenAI, y el backup en MongoDB obviamente requiere conexión. Si se usa offline, la app sigue funcionando en lo básico (tareas, horas) pero el informe no estaría

disponible. Esto podría solventarse con modelos offline en el futuro, pero escapa al alcance de un proyecto de grado.

Un aspecto para discutir es la **precisión del seguimiento de tiempo**. La aplicación actual delega al usuario la entrada manual de horas. Esto funciona bajo confianza de auto-reporte (similar a llenar una hoja de tiempos uno mismo). Herramientas más avanzadas podrían llevar un cronómetro o incluso detectar actividad. Sin embargo, implementar un time-tracker automático habría añadido complejidad (por ejemplo, qué pasa si el usuario olvida pausar un cronómetro). Se decidió conscientemente que el enfoque manual era suficiente y en algunos casos preferible, pues da flexibilidad (ej.: si alguien trabaja 1h pero interrumpidamente, puede simplemente poner 1 al final). Este tema es debatible, pero entra en las decisiones de **scoping**: un cronómetro sería buena característica futura, pero no imprescindible para cumplir los objetivos propuestos.

En cuanto al rendimiento y escalabilidad, para el caso de uso individual es más que adecuado. Si se quisiera escalar a una pequeña empresa con 10–20 usuarios simultáneos, habría que migrar totalmente a la base de datos y quizás implementar un servidor con endpoints autenticados. Node.js y MongoDB permitirían escalar sin grandes cambios arquitectónicos. Eso muestra que la elección tecnológica fue acertada en previsión: Express es minimalista pero suficiente, y Node.js puede soportar muchos clientes concurrentes gracias a su modelo asíncrono.

Feedback de usuarios: se compartió la aplicación con dos compañeros para que la probaran en un día de estudio. Sus comentarios fueron positivos respecto a la facilidad de uso. Uno mencionó que le resultaría útil una función de *recordatorios* o alarmas para ciertas tareas, lo cual no estaba contemplado (sería más del terreno de gestión del tiempo proactiva). Otra sugerencia fue permitir *editar el nombre de la tarea* después de creada, lo cual actualmente no se implementó (habría que añadir quizás un ícono de editar o permitirlo al hacer doble click). Estas sugerencias se consideran val...(continuación)

Retomando la discusión, cabe mencionar que algunas sugerencias de mejora surgieron al hacer pruebas de uso. Por ejemplo, se observó que actualmente no es posible editar el nombre de una tarea una vez creada, ni eliminar tareas (solo marcarlas completadas); estas funcionalidades podrían añadirse para mayor flexibilidad. También se propuso implementar recordatorios o notificaciones para tareas con plazos, lo cual ampliaría el alcance hacia la gestión de tiempo proactiva. Estas sugerencias se consideran valiosas, pero quedaron fuera del alcance inicial, siendo candidatas para trabajos futuros.

En general, los resultados obtenidos están alineados con los objetivos planteados. A continuación, se presentan las **conclusiones** del trabajo, las posibles líneas de desarrollo futuro y la relación del proyecto con los Objetivos de Desarrollo Sostenible.

8. Conclusiones

El desarrollo de esta aplicación web de gestión diaria de tareas con registro de horas y notas ERP ha permitido demostrar la viabilidad de integrar en una única herramienta varias funciones complementarias que típicamente se encontraban dispersas en múltiples aplicaciones. Se alcanzaron los objetivos definidos: la aplicación resultante es funcional, fácil de usar y responde a la necesidad detectada de mejorar la productividad personal mediante una mejor organización y registro del trabajo.

En particular, se ha logrado:

- Diseñar una interfaz sencilla e intuitiva que facilita el registro y seguimiento de las actividades diarias.
- Implementar un mecanismo eficaz de persistencia de datos (local y con copia de seguridad remota), asegurando la conservación de la información.
- Incorporar tecnología de inteligencia artificial para automatizar la generación de informes, aportando un valor añadido novedoso.
- Validar mediante pruebas informales que la solución efectivamente simplifica la tarea de llevar un control diario, reduciendo la carga administrativa para el usuario.

Desde el punto de vista formativo, este proyecto ha permitido aplicar una amplia gama de conocimientos de ingeniería informática: desarrollo web full-stack, gestión de bases de datos, integración de APIs externas, y aplicación de metodologías de desarrollo ágiles. Asimismo, se ha obtenido aprendizajes sobre buenas prácticas de UX y sobre cómo equilibrar simplicidad y funcionalidad en un producto mínimo viable.

Como conclusión general, se puede afirmar que la aplicación desarrollada cumple con la motivación inicial de servir como **agenda digital inteligente** para el trabajo diario. Si bien quedan aspectos por pulir y expandir, el trabajo realizado sienta una base sólida sobre la cual construir futuras mejoras. En un entorno donde la optimización del tiempo es cada vez más crucial, herramientas como ésta contribuyen a fomentar hábitos de organización y autoevaluación que, a largo plazo, repercuten positivamente en la productividad y bienestar del usuario.

9. Futuras líneas de proyecto

Dado que el desarrollo realizado corresponde a una versión prototipo, existen numerosas oportunidades para ampliar y mejorar la aplicación en trabajos futuros. Algunas **líneas de desarrollo** recomendadas son:

- **Multiusuario y autenticación:** Implementar un sistema de registro y login de usuarios, de forma que la aplicación pueda ser utilizada por múltiples personas de forma aislada (cada usuario con sus tareas privadas). Esto implicaría asociar cada tarea a un identificador de usuario y proteger las rutas con mecanismos de autenticación (por ejemplo, JWT o sesiones). Esta mejora abriría la puerta a desplegar la aplicación como un servicio web público o para un equipo de trabajo pequeño.
- **Migración completa a base de datos en la nube:** Actualmente MongoDB Atlas se usa solo como copia de seguridad; en el futuro sería deseable que la base de datos fuera la fuente principal de datos, eliminando la dependencia del archivo local. Esto mejoraría la escalabilidad y confiabilidad, permitiendo acceder a las tareas desde diferentes dispositivos en tiempo real. Junto con la autenticación, cada usuario podría tener su colección de tareas en la base de datos.
- **Mejoras en la interfaz y experiencia de usuario:** Incluir funcionalidades adicionales como editar o renombrar tareas, eliminar tareas si ya no son relevantes, o reordenar tareas pendientes mediante *drag and drop* para priorizarlas manualmente. También se podrían agregar iconos o indicadores visuales (por ejemplo, un icono de reloj en las tareas en revisión, o distintos colores según prioridad). Otra mejora de UX sería proveer confirmaciones visuales de guardado (p.ej. un pequeño mensaje "Guardado" al actualizar notas) para mayor tranquilidad del usuario.
- **Funcionalidad de cronómetro o temporizador:** Para refinar el registro de horas, se podría integrar un temporizador que el usuario pueda iniciar/pausar en una tarea, contabilizando automáticamente el tiempo transcurrido. De este modo, el registro de horas sería más preciso y menos manual. Esto conllevaría desafíos de sincronización (asegurar que solo un temporizador esté corriendo a la vez, etc.), pero aportaría gran valor para usuarios que deseen medir su tiempo en tiempo real.
- **Aplicación móvil o PWA:** Convertir la aplicación en una Aplicación Web Progresiva (PWA) permitiría instalarla en dispositivos móviles y habilitar cierto funcionamiento *offline*. Con una adaptación mínima (manifiesto web, service workers), el usuario podría usar la agenda desde su smartphone incluso sin conexión, sincronizando los datos cuando vuelva a estar en línea. Alternativamente, desarrollar un cliente móvil nativo que consuma la API podría brindar una mejor experiencia en iOS/Android.
- **Integración con herramientas externas:** Explorar la integración con calendarios (Google Calendar, Outlook) para convertir tareas

completadas en eventos de calendario o viceversa, o con sistemas de correo para enviar el informe diario automáticamente por email. Asimismo, en un contexto empresarial, se podría integrar la aplicación con un ERP real: por ejemplo, mediante una API, enviar al final del día las notas ERP consolidadas directamente al sistema corporativo correspondiente, eliminando la tarea manual de volcado de datos.

- **Mejoras en el módulo de IA:** Aunque la generación de informes con GPT-3.5 funciona correctamente, se podría profundizar en el uso de IA para otras funciones. Por ejemplo, un modelo de lenguaje podría usarse para *priorización automática* de tareas (sugerir al usuario por cuál empezar según su descripción), o para analizar las notas y extraer pendientes automáticamente. También sería interesante experimentar con modelos más nuevos o específicos, siempre evaluando el coste y beneficio.

En síntesis, el proyecto tiene un amplio margen de crecimiento. Las mejoras propuestas abordan tanto la **escalabilidad técnica** (soporte a más usuarios, más datos) como la **profundidad funcional** (nuevas características) y la **calidad de la experiencia**. Implementar estas líneas futuras convertiría este prototipo en una herramienta mucho más robusta y completa, potencialmente utilizable a nivel profesional por equipos de trabajo para la gestión de su día a día.

10. Objetivos de Desarrollo Sostenible (ODS)

Este proyecto, aunque centrado en una aplicación informática, contribuye de manera indirecta a varios de los Objetivos de Desarrollo Sostenible de la ONU. En particular, se identifican los siguientes ODS relacionados, junto con una breve explicación de su pertinencia:

- **ODS 4: Educación de calidad.** El desarrollo y uso de esta aplicación promueve habilidades de organización, autogestión y alfabetización digital en sus usuarios. Al facilitar que estudiantes y profesionales aprendan a administrar mejor su tiempo y tareas, se apoya la meta de garantizar una educación de calidad y promover oportunidades de aprendizaje permanente. Asimismo, la realización de este TFG ha supuesto un proceso formativo significativo en el ámbito de la ingeniería de software, generando conocimiento que puede ser transferido y compartido con la comunidad académica.
- **ODS 8: Trabajo decente y crecimiento económico.** El objetivo 8 busca, entre otros aspectos, **aumentar la productividad económica** mediante la adopción de tecnologías y prácticas eficientes. La aplicación desarrollada encaja en esta visión al proporcionar una herramienta que mejora la productividad personal y la gestión del tiempo de trabajo. Al ayudar a los individuos a ser más eficientes y conscientes de cómo invierten su jornada, se contribuye a un mejor desempeño laboral y, en escala acumulada, a mayor productividad en las organizaciones. Esto está alineado con la meta 8.2 (diversificar, innovar y mejorar la productividad económica mediante la modernización tecnológica).
- **ODS 9: Industria, innovación e infraestructura.** Este objetivo enfatiza la importancia de la innovación y la adopción de tecnologías. El proyecto aporta innovación en el campo de las herramientas de productividad personal, integrando inteligencia artificial y servicios en la nube de manera accesible. Fomenta la **digitalización de procesos** cotidianos (pasar de registros manuales a una aplicación web), lo cual es concordante con construir infraestructuras resilientes y promover la industrialización sostenible basada en tecnología (meta 9.c: aumentar el acceso a las tecnologías de la información). Además, el proyecto mismo es un ejercicio de innovación a pequeña escala, explorando nuevas combinaciones de funcionalidades.

En resumen, aunque se trata de un desarrollo acotado, este proyecto de TFG contribuye a los ODS mencionados al promover la utilización de herramientas tecnológicas para mejorar la eficiencia en el trabajo y el aprendizaje. Un mejor manejo del tiempo y las tareas repercute positivamente en la productividad (ODS 8) y en el bienestar educativo/laboral de las personas (ODS 4), mientras que la adopción y difusión de soluciones digitales innovadoras se alinea con la construcción de infraestructuras tecnológicas inteligentes (ODS 9).

11. Referencias

Abbamonte, K. (2024). *The 5 best time tracking apps in 2025 | Zapier*. https://zapier.com/blog/best-time-tracking-apps/

Allen, D. (2025). *Getting Things Done*. En *Wikipedia*. https://en.wikipedia.org/w/index.php?title=Getting_Things_Done&oldid=1273531973

Arenzana, D. (2024). *Principios de usabilidad web de Jacob Nielsen y el diseño UX*. Semrush Blog. https://es.semrush.com/blog/usabilidad-web-principios-jakob-nielsen/null

Azevedo, S. (2023). *Cambio de contexto: ¿Cuánto le cuesta a su equipo cambiar de tarea?* https://clickup.com/es-ES/blog/56983/cambio-de-contexto

Baleón García, C. (2025). Análisis de Notion: La Herramienta Todo-en-Uno para Profesionales, Estudiantes y Académicos. *Human and Nonhuman Communication Lab*. https://www.humanandnonhumancommunicationlab.com/2024/04/analisis-de-notion-la-herramienta-todo.html

Cáceres, Y. (2023, mayo 5). *Diseño UX-UI: Principios de usabilidad de Jakob Nielsen | Oh my Web!®*. https://ohmyweb.net/diseno-ux-principios-de-usabilidad-de-jakob-nielsen/

Cámara de Sevilla. (2023, septiembre 21). *La importancia del diseño responsive en la era de los dispositivos móviles—Escuela de Negocios | Cámara de Comercio de Sevilla*. https://en.camaradesevilla.com/diseno-responsive/

Cartagena, S. (2024, noviembre 4). *Project Management: Rompiendo con la cultura de las reuniones excesivas*. Entrepreneur. https://www.entrepreneur.com/es/consultoria/project-management-rompiendo-con-la-cultura-de-las/482322

Chui, M., & Manyika, J. (2012). *La economía social: Liberar valor y productividad a través de las tecnologías sociales | McKinsey*. https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-social-economy?utm_source=chatgpt.com

Clockify, C. com. (2025). *Temporizador de tareas gratuito*. Clockify. https://clockify.me/es/temporizador-de-tareas

CMS, A. (2024). *Notion, la startup que vale US$ 10 mil millones y cautivó a Silicon Valley*. Forbes Argentina. https://www.forbesargentina.com/innovacion/notion-startup-vale-us-10-mil-millones-cautivo-silicon-valley-n50997

Dekle, K. (2019). *Comparing 6 different Project Management apps*. Launch the Damn Thing®! | Squarespace Website Designer. https://www.launchthedamnthing.com/blog/project-management-comparison-trello-asana-clickup-airtable

Doctor, T. (2022, marzo 9). Odoo time tracking guide (Limitations, 7 alternatives). *Time Doctor Blog*. https://www.timedoctor.com/blog/odoo-time-tracking/

Erkic, A. (2023, mayo 23). Estadísticas de reuniones de trabajo para 2024. *Centro de conocimiento de Pumble*. https://pumble.com/learn/es/communication/meeting-statistics/

Eugene. (2025, marzo 20). Seguimiento de Tiempo en Notion: Mejores Prácticas 2025. *Everhour Blog*. https://everhour.com/blog/es/seguimiento-de-tiempo-en-notion/

FocusTribes. (2024). *SAP PS (sistema de proyectos) módulo de gestión de proyectos SAP ERP*. https://blog.focustribes.com/es/sap-ps-sistema-de-proyectos-módulo-de-gestión-de-proyectos-sap-erp

Jira. (2025). *Jira Software: Funciones | Atlassian*. https://www.atlassian.com/es/software/jira/features

Linkedin. (2025). *¿Cómo se utiliza la divulgación progresiva para el diseño de UX?* https://es.linkedin.com/advice/3/how-do-you-use-progressive-disclosure-ux-design?lang=es

Macedo, S. F. (2025, abril 25). *El exceso de reuniones en el trabajo ponen la productividad y el liderazgo bajo la lupa*. El Periódico. https://www.elperiodico.com/es/economia/20250425/exceso-reuniones-trabajo-ponen-productividad-116732958

Muradas, Y. (2020). *Qué es Trello, características y ventajas | OpenWebinars*. OpenWebinars.net. https://openwebinars.net/blog/que-es-trello/

NetSuite.com. (2024, septiembre 25). *A Guide to NetSuite Modules*. Oracle NetSuite. https://www.netsuite.com/portal/resource/articles/erp/netsuite-modules.shtml

Organización Internacional del Trabajo. (2020). *El teletrabajo durante la pandemia de COVID-19 y después de ella*. https://www.ilo.org/sites/default/files/wcmsp5/groups/public/@ed_protect/@protrav/@travail/documents/publication/wcms_758007.pdf

Rebelo, M. (2024). *Notion vs. Evernote: Which should you use? [2025] | Zapier*. https://zapier.com/blog/notion-vs-evernote/

Sinzerad. (2024). *¿Por qué cambiar de aplicación 566 veces al día está afectando a tu productividad?* https://www.sinzerad.com/es/por-que-cambiar-de-aplicacion-566-veces-al-dia-esta-afectando-a-tu-productividad/

Slack. (2025a). *Herramientas de IA para hacer resúmenes y responder preguntas*. Slack. https://slack.com/intl/es-la/features/ai

Slack. (2025b). *Usar IA de Slack para tomar notas de las juntas*. Slack Help Center. https://slack.com/intl/es-es/help/articles/31377193680019-Usar-IA-de-Slack-para-tomar-notas-de-las-juntas

Soler, D. (2024, marzo 20). *Diseño de sistemas UX/UI: ¿qué papel cumple la coherencia?* https://keepcoding.io/blog/coherencia-en-el-diseno-de-sistemas-ux-ui/

Spataro, J. (2020, octubre 28). Microsoft Teams reaches 115 million DAU—plus, a new daily collaboration minutes metric for Microsoft 365. *Microsoft 365 Blog*. https://www.microsoft.com/en-us/microsoft-365/blog/2020/10/28/microsoft-teams-reaches-115-million-dau-plus-a-new-daily-collaboration-minutes-metric-for-microsoft-365/

Stepanets, A. (2024, diciembre 10). *Qué es Trello, características, ventajas y desventajas*. Gantt Chart GanttPRO Blog. https://blog.ganttpro.com/es/caracteristicas-de-trello-sus-ventajas-y-desventajas-2023/

Todoist. (s. f.-a). *Precios | Todoist*. Recuperado 17 de mayo de 2025, de https://www.todoist.com/pricing

Todoist. (s. f.-b). *Todoist | Una «to do list» para organizar tu trabajo y tu vida*. Recuperado 17 de mayo de 2025, de https://www.todoist.com

Todoist. (2025). *Características | Todoist*. https://www.todoist.com/features

Trello. (2025). *Qué es Trello: Descubre sus funciones, usos y todo lo que ofrece | Trello*. https://trello.com/es/tour

Vorecol. (2024). *La importancia de la colaboración y el trabajo en equipo en el uso de software de innovación en RRHH.* https://vorecol.com/es/articulos/articulo-la-importancia-de-la-colaboracion-y-el-trabajo-en-equipo-en-el-uso-de-software-de-innovacion-en-rrhh-173395?utm_source=chatgpt.com

Wheelhub. (2024, septiembre 18). *Notion: Ventajas y Desventajas | Wheel hub*. https://wheelhub.es/blog/notion-ventajas-desventajas/

Wojahn, J. (2022). *Por qué necesita priorizar el diseño de UX/UI en la experiencia de sus empleados*. https://www.thirdera.com/es-us/insights/why-you-need-to-prioritize-ux/ui-design-in-your-employee-experience

Zimmergren, T. (2024, enero 4). *Stay productive with the Pomodoro Technique!* Tobias Zimmergren on Tech. https://zimmergren.net/stay-productive-with-the-pomodoro-technique/

Acceso GitHub Proyecto: https://github.com/Pedro97198/tareasv2