



Universidad Politécnica
de Madrid



**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

**Desarrollo de un sistema software
usando Copilot**

Autor: Diego Ojeda Higuera

Tutor: Oscar Dieste Tubio

Tutora: Francy Diomar Rodríguez

Madrid, Enero 2026

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Desarrollo de un sistema software usando Copilot

Enero 2026

Autor: Diego Ojeda Higuera

Tutores:

Oscar Dieste Tubio

ETSI Informáticos

Universidad Politécnica de Madrid

Francy Diomar Rodríguez

ETSI Informáticos

Universidad Politécnica de Madrid

Agradecimientos

Quiero expresar mi más sincero agradecimiento a mis tutores de Trabajo Fin de Grado, Óscar Dieste y Francy Rodríguez, por el apoyo continuo, la cercanía y el conocimiento compartido durante todo el desarrollo de este proyecto. Su gran implicación, disponibilidad y dedicación han sido clave no solo para la correcta realización del trabajo, sino también para ayudarme a superar dificultades, tomar mejores decisiones técnicas y crecer como desarrollador y como estudiante a lo largo de este proceso.

Finalmente, deseo agradecer a mis amigos Jaime y Asier, así como a Eli, por el apoyo constante, la comprensión y el ánimo que me han brindado durante estos meses. Su cercanía y apoyo personal han sido fundamentales para mantener la motivación y afrontar con mayor confianza el desarrollo de este trabajo.

Resumen

La adopción de herramientas de inteligencia artificial generativa ha supuesto una transformación significativa en los procesos actuales de desarrollo de software. Asistentes como GitHub Copilot y los modelos de lenguaje de gran tamaño (LLMs) han introducido nuevas formas de escribir código, documentar sistemas, depurar errores y acelerar tareas de desarrollo. No obstante, su uso generalizado también ha generado preocupaciones relacionadas con la calidad del software, la dependencia excesiva del desarrollador, la pérdida de comprensión del sistema y la aparición de malas prácticas. En este contexto, resulta necesario realizar un análisis crítico y basado en la experiencia práctica de su impacto real.

Este Trabajo Fin de Grado analiza el uso de herramientas de inteligencia artificial generativa como apoyo al desarrollo de software, con el objetivo principal de identificar buenas y malas prácticas en el desarrollo asistido por IA. El estudio se fundamenta en el desarrollo práctico de un sistema software real: una API web para la gestión de una biblioteca, implementada utilizando FastAPI, SQLAlchemy y Pytest, junto con una interfaz web y un sistema de control de acceso basado en roles. El proceso de desarrollo siguió una metodología estructurada con ciclos iterativos e incrementales, lo que permitió evaluar de forma continua la interacción entre el desarrollador y estas herramientas a lo largo de todas las fases del proyecto, incluyendo la toma de requisitos, el diseño de la arquitectura, la implementación, la depuración y las pruebas.

Los sistemas de IA generativa se utilizaron como apoyo en la extracción de requisitos, la generación de código, la identificación y resolución de errores, la refactorización y el diseño de pruebas. A través de múltiples iteraciones, el trabajo pone de manifiesto tanto los escenarios en los que estas herramientas mejoran de forma efectiva la productividad y facilitan la toma de decisiones, como aquellos en los que un uso inadecuado conduce a un aumento de la complejidad, a errores difíciles de detectar o a una menor comprensión del sistema desarrollado. Se presta especial atención a las diferencias en el aprovechamiento de estas herramientas según el nivel de experiencia del desarrollador, destacando la importancia del control humano y la revisión crítica.

Los resultados obtenidos permiten establecer un conjunto de recomendaciones prácticas para integrar la inteligencia artificial generativa en los flujos de trabajo de desarrollo de software de manera responsable y eficiente. Las conclusiones subrayan que, aunque estas herramientas pueden aumentar significativamente la productividad, deben utilizarse como apoyo al desarrollador y no como sustitutos del razonamiento técnico, garantizando así la calidad, mantenibilidad y sostenibilidad del software desarrollado.

Abstract

The rapid adoption of generative artificial intelligence tools has significantly transformed modern software development processes. Assistants such as GitHub Copilot and large language models (LLMs) have introduced new ways of writing code, documenting systems, debugging errors, and accelerating development tasks. However, their widespread use has also raised concerns regarding code quality, developer dependency, loss of system understanding, and the emergence of poor development practices. In this context, a critical and experience-based analysis of their impact is necessary.

This Final Degree Project analyzes the use of generative AI tools as support in the development of a real software system, with the main objective of identifying good and bad practices in AI-assisted software engineering. The study is grounded in the practical development of a web-based library management system implemented as a RESTful API using FastAPI, SQLAlchemy, and Pytest, with a frontend interface and role-based access control. The development process followed a structured methodology with iterative and incremental cycles, allowing continuous evaluation of the interaction between the developer and AI tools throughout all phases of the project, including requirements elicitation, architecture definition, implementation, debugging, and testing.

Generative AI was employed as an assistant for requirements documentation, code generation, error diagnosis, refactoring, and test design. Through multiple development iterations, the project highlights scenarios where AI tools effectively improve productivity and support decision-making, as well as situations where their misuse leads to increased complexity, hidden errors, or reduced developer understanding. Particular attention is paid to differences in how AI tools should be used depending on the developer's experience level, emphasizing the importance of critical review and human control.

The results of this work provide a set of practical recommendations for integrating generative AI into software development workflows in a responsible and effective manner. The conclusions emphasize that, while AI tools can significantly enhance productivity, they must be used as supportive instruments rather than autonomous solution generators, ensuring that software quality, maintainability, and developer expertise are preserved.

Contenido

1	Introducción	1
1.1	Contexto y motivación del trabajo	1
1.2	Objetivos del estudio	1
1.3	Alcance y limitaciones	2
1.4	Estructura del documento	3
2	Marco teórico	4
2.1.1	Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data	4
2.1.2	Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence	4
2.1.3	Don't Expect Juniors to Teach Senior Professionals to Use Generative AI	5
2.2	Tecnologías usadas	5
2.2.1	Copilot	5
2.2.2	Chat GPT	5
2.2.3	FastApi	5
2.2.4	Pytest	5
3	Metodología de trabajo	6
3.1	Plan de trabajo	6
3.1.1	Toma de requisitos	6
3.1.2	Arquitectura	6
3.1.3	Implementación	6
3.1.4	Depuración	6
3.1.5	Pruebas	7
4	Desarrollo	8
4.1	Extracción de Requisitos	8
4.1.1	Primera iteración: primera generación de documentos	8
4.1.2	Segunda iteración: mejora wireframes y casos de uso	10
4.2	Arquitectura	12
4.3	Diseño	13
4.4	Implementación	13
4.4.1	Iteración 0: primer intento	13
4.4.2	Primera iteración: estructura general y despliegue de la API	14
4.4.3	Segunda iteración: front, sistema login y diferenciación de vistas	16
4.4.4	Tercera iteración: creación catálogo, copias y división js	18
4.4.5	Cuarta iteración: nuevo diseño	20
4.4.6	Quinta iteración: mejora funciones existentes	21
4.4.7	Sexta iteración: creación pestaña prestamos	23

4.4.8	Séptima iteración: creación Alembic	25
4.4.9	Cambio de conversación	25
4.4.10	Octava iteración: adminSanciones.....	26
4.4.11	Novena iteración: adminReservas	26
4.5	Depuración	28
4.5.1	Iteración 0: intento generación código completo.....	28
4.5.2	Primera iteración	28
4.5.3	Segunda iteración.....	28
4.5.4	Tercera iteración.....	29
4.5.5	Cuarta iteración.....	30
4.5.6	Quinta iteración.....	30
4.5.7	Sexta iteración.....	31
4.5.8	Séptima iteración.....	31
4.5.9	Octava iteración.....	32
4.5.10	Novena iteración.....	33
4.6	Pruebas.....	33
5	Resultados	35
6	Lecciones aprendidas.....	38
6.1	Requisitos	38
6.2	Arquitectura.....	41
6.3	Implementación	41
6.4	Depuración	44
6.5	Pruebas.....	46
7	Conclusiones	48
7.1	Resumen de hallazgos	48
7.1.1	Recomendaciones prácticas para futuros desarrolladores	48
7.1.2	Limitaciones del uso de IAs.....	49
7.2	Riesgos y desafíos	49
7.3	Equilibrio entre automatización y control humano	50
7.4	Sostenibilidad, objetivos del milenio: industria, innovación e infraestructura	50
8	Bibliografía	51
9	Anexos.....	52

1 Introducción

Este capítulo introduce el trabajo de fin de grado, contextualizando el impacto de las inteligencias artificiales generativas, como GitHub Copilot, en el desarrollo de software. Se exploran las motivaciones detrás del estudio, que busca analizar cómo estas herramientas afectan la calidad del código y la formación de los desarrolladores. Además, se establecen los objetivos principales del trabajo, que incluyen la identificación de buenas y malas prácticas al utilizar IA en el desarrollo de software, así como las implicaciones de su integración en flujos de trabajo profesionales. Finalmente, se delimitan el alcance y las limitaciones del estudio, destacando su enfoque en asistentes de programación y la metodología adoptada para el desarrollo de una API web de gestión de biblioteca.

1.1 Contexto y motivación del trabajo

En los últimos años, la irrupción de las inteligencias artificiales generativas ha supuesto un punto de inflexión en numerosos ámbitos profesionales, y de manera especialmente significativa en el desarrollo de sistemas software. Herramientas como GitHub Copilot, basadas en modelos de lenguaje avanzados, han transformado la forma en la que los desarrolladores escriben código, documentan sistemas y resuelven problemas técnicos. Este cambio ha generado un intenso debate en la comunidad académica y profesional acerca de sus buenas y malas prácticas, así como sobre su impacto real en la calidad del software y en la formación de los desarrolladores. En este contexto, resulta especialmente relevante analizar cómo estas herramientas influyen en los profesionales del desarrollo informático.

Inicialmente, una de las premisas más extendidas fue que este tipo de asistentes inteligentes supondrían un gran apoyo para los desarrolladores junior [1], al facilitarles la escritura de código, acelerar el aprendizaje y reducir la barrera de entrada a tecnologías complejas. Sin embargo, con el uso continuado de estas herramientas, han comenzado a surgir dudas sobre esta afirmación. Diversos estudios y experiencias prácticas sugieren que, sin una base sólida de conocimientos, el uso indiscriminado de IA puede fomentar la dependencia, dificultar la comprensión profunda del código generado y derivar en malas prácticas como la aceptación de soluciones incorrectas, inseguras o ineficientes.

Por el contrario, se está observando que los desarrolladores senior son quienes logran extraer un mayor beneficio de herramientas como Copilot. Gracias a su experiencia previa, son capaces de evaluar críticamente las sugerencias de la IA, adaptarlas al contexto del proyecto, detectar errores potenciales y utilizarlas como un acelerador de productividad sin comprometer la calidad del software. Conocer cómo usan la IA los juniors y los seniors ayudaría a entender mejor los puntos fuertes y débiles de la IA en cada tipo de desarrollador, permitiendo por ejemplo idear mejores procesos de formación para estudiantes de informática.

1.2 Objetivos del estudio

El presente Trabajo de Fin de Grado tiene como finalidad analizar de forma crítica el impacto del uso de inteligencias artificiales generativas en el desarrollo de sistemas software, prestando especial atención a su aplicación práctica y a las implicaciones que conlleva para la calidad del producto.

El objetivo, que constituye el eje central del trabajo, es el análisis y descubrimiento de buenas y malas prácticas en el desarrollo de software asistido por IAs generativas. Este objetivo abarca la identificación de estrategias efectivas de uso, incluyendo la formulación adecuada de preguntas (prompts),

los enfoques que realmente agilizan el desarrollo y aquellos que, por el contrario, no aportan mejoras significativas o incluso lo ralentizan. Asimismo, se pretende establecer criterios y recomendaciones que permitan integrar estas herramientas de forma responsable, eficiente y crítica dentro de los flujos de trabajo profesionales.

1.3 Alcance y limitaciones

Este trabajo tiene como objetivo analizar el uso de herramientas de inteligencia artificial, concretamente asistentes de programación como Copilot, como apoyo al desarrollo de software. El alcance del estudio incluye la observación y análisis de distintos aspectos clave del proceso de desarrollo, tales como la productividad percibida durante la implementación, la calidad del código generado, la comprensión progresiva del sistema por parte del desarrollador y la integración de estas herramientas dentro de flujos de trabajo reales.

El estudio se apoya en experiencias prácticas, ejemplos controlados y un análisis crítico del proceso de desarrollo, con el propósito de extraer conclusiones aplicables tanto al ámbito académico como al profesional. No obstante, este proyecto no contempla el desarrollo, entrenamiento o modificación de modelos de inteligencia artificial, ni pretende realizar una evaluación exhaustiva de todas las herramientas existentes en el mercado. El análisis se limita a aquellas soluciones más representativas y relevantes para el contexto del estudio y los objetivos del trabajo.

Metodología de trabajo

La metodología seguida combina un enfoque estructurado con un desarrollo iterativo. A nivel global, el proyecto se organiza siguiendo un modelo en cascada entre las distintas fases principales: análisis de requisitos, diseño de la arquitectura, implementación, depuración y pruebas. Sin embargo, dentro de las fases de requisitos, implementación y depuración, se adopta un enfoque iterativo e incremental, en el que cada funcionalidad se desarrolla, valida y corrige antes de avanzar a la siguiente.

Este enfoque ha permitido integrar de manera efectiva el uso de la IA como herramienta de apoyo, facilitando la resolución de problemas concretos y la mejora progresiva del sistema sin comprometer su estabilidad.

Tipo de sistema desarrollado

El sistema desarrollado en este trabajo es una API web para la gestión de una biblioteca, diseñada para ofrecer funcionalidades de administración y consulta a distintos tipos de usuarios. La API permite gestionar entidades como usuarios, libros, préstamos, reservas y sanciones, incorporando mecanismos de autenticación, control de acceso por roles y administración avanzada mediante paneles específicos.

El sistema se ha concebido como una solución modular y extensible, lo que ha facilitado tanto su desarrollo iterativo como su depuración progresiva.

Productos obtenidos

Como resultado del desarrollo del proyecto se han obtenido los siguientes productos principales, que se corresponden con las distintas iteraciones descritas a lo largo del trabajo:

- Una API REST funcional para la gestión de una biblioteca.
- Módulos administrativos para la gestión de usuarios, préstamos, reservas y sanciones.

- Sistema de control de acceso basado en roles.
- Gestión de la evolución del esquema de la base de datos mediante Alembic.
- Documentación implícita del proceso de desarrollo iterativo y de depuración apoyado en el uso de la IA.

Estos productos reflejan un sistema completo y operativo, desarrollado y refinado a través de múltiples iteraciones de implementación y depuración.

Código fuente y anexos

El código fuente completo del proyecto, junto con los recursos adicionales y anexos necesarios para su ejecución y evaluación, se encuentra disponible en el siguiente repositorio público:

<https://github.com/DiegoOjeda/TFG.git>

La estructura del GIT se explica en el **ANEXO [A]**

Este repositorio permite reproducir el desarrollo realizado, analizar las decisiones de diseño adoptadas y evaluar de forma transparente los resultados obtenidos durante el proyecto.

1.4 Estructura del documento

El documento se estructura en varios capítulos que abordan de forma progresiva el desarrollo del proyecto y el análisis del uso de herramientas de inteligencia artificial como apoyo al desarrollo de software. En primer lugar, se presenta una introducción en la que se contextualiza el trabajo, se definen los objetivos y se justifica su motivación, seguida de un marco teórico que recoge los conceptos fundamentales relacionados con el desarrollo de APIs web y la aplicación de la inteligencia artificial en el ámbito del desarrollo de software. A continuación, se expone el alcance y las limitaciones del trabajo, delimitando los aspectos abordados y describiendo la metodología empleada y el tipo de sistema desarrollado. El núcleo del documento lo constituye el desarrollo práctico del proyecto, donde se detalla la implementación de la API de forma iterativa, el uso de la inteligencia artificial como herramienta de apoyo, así como el proceso de depuración y pruebas. Posteriormente, se incluye un capítulo dedicado a buenas prácticas y limitaciones, en el que se analizan las decisiones técnicas adoptadas y las restricciones encontradas durante el desarrollo. Finalmente, el documento concluye con un capítulo de conclusiones, en el que se valoran los resultados obtenidos y se proponen posibles líneas de trabajo futuro, completándose con un apartado de anexos que incluye información adicional y el acceso al repositorio de código fuente.

2 Marco teórico

Este capítulo expone las principales tecnologías utilizadas en el desarrollo del proyecto, incluyendo GitHub Copilot, ChatGPT, FastAPI y pytest. Se analiza cómo estas herramientas fueron empleadas para optimizar el proceso de desarrollo, desde la generación de código hasta las pruebas del sistema. Además, se revisan estudios previos que abordan el uso de la inteligencia artificial en el desarrollo de software, destacando cómo su impacto varía entre desarrolladores junior y senior. Se profundiza en cómo los seniors, por su experiencia, son capaces de aprovechar mejor las herramientas de IA, mientras que los juniors pueden enfrentarse a riesgos como la dependencia excesiva o la falta de comprensión profunda del código. También se discute el avance general de la inteligencia artificial en el ámbito laboral, mostrando cómo estas tecnologías están transformando la forma en que se desarrollan y gestionan los proyectos de software.

2.1.1 Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data

Este artículo analiza si la adopción de inteligencia artificial generativa constituye un cambio tecnológico sesgado por antigüedad, afectando de forma desproporcionada a trabajadores junior frente a perfiles senior. Mediante datos de más de 62 millones de trabajadores y 285.000 empresas estadounidenses (2015-2025) y la identificación de empresas “adoptantes” mediante textos en ofertas de empleo, los autores observan que los perfiles junior experimentan descensos notables en empleo tras la adopción de GenAI, mientras que los perfiles senior se mantienen prácticamente estables. El ajuste ocurre a través de una reducción en nuevas contrataciones y no mediante despidos ni ascensos.

El estudio aporta evidencia empírica temprana sobre cómo la IA generativa altera la demanda relativa de trabajo en función de la experiencia, y complementa literatura previa sobre “skill-biased technological change” mostrando que el sesgo puede no ser solo en habilidades, sino también en antigüedad. Para un marco teórico, el artículo permite posicionar la IA como tecnología sustitutiva de trabajo rutinario y de bajo grado de autonomía, reforzando patrones de desplazamiento laboral en tareas replicables. [1]

2.1.2 Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence

Este artículo presenta seis hechos estilizados sobre los efectos recientes de la IA generativa en el mercado laboral estadounidense utilizando datos administrativos de una gran plataforma de nóminas. Se observa que los trabajadores de 22-25 años en ocupaciones más expuestas presentan una caída relativa del empleo del 13%, mientras que los trabajadores senior o con menor exposición se mantienen estables. Además, los ajustes se producen vía empleo y no vía salarios, reforzando la hipótesis de sustitución en etapas iniciales de carrera profesional.

Los autores interpretan estos resultados como una señal temprana del impacto desigual de la IA según nivel de experiencia y tipo de tarea. Aporta un marco empírico útil que sugiere que la IA automatiza preferentemente actividad junior y tareas repetitivas, mientras que para perfiles senior tiene efectos más complementarios. [2]

2.1.3 Don't Expect Juniors to Teach Senior Professionals to Use Generative AI

Este trabajo adopta un enfoque organizacional y cualitativo para analizar cómo trabajadores junior y senior incorporan la IA generativa en entornos profesionales. Los autores muestran que no es realista asumir que los juniors pueden actuar como “puentes de adopción tecnológica” hacia mandos senior, especialmente en tecnologías emergentes. Identifican diferentes tácticas y limitaciones de los juniors en la mitigación de riesgos asociados a la IA (exactitud, contextualización), resaltando que estos tienden a proponer soluciones únicamente a nivel de proyecto en lugar de a nivel organizacional o sistémico.

Este artículo no se centra en efectos laborales agregados, sino en dinámicas de adopción dentro de empresas, por lo que es menos cercano a efectos de sustitución laboral y más relacionado con gestión del cambio, alfabetización tecnológica y gobernanza de IA. [3]

2.2 Tecnologías usadas

2.2.1 Copilot

Copilot integrado en Microsoft Teams permite generar y resumir contenido a partir de conversaciones y reuniones, creando automáticamente notas, acciones y documentos estructurados. En este TFG se ha usado para apoyar la toma de requisitos y la elaboración inicial de documentos de casos de uso, transformando discusiones con el cliente en artefactos textuales que luego se han revisado y depurado manualmente. [4]

2.2.2 Chat GPT

ChatGPT 5.0, basado en el modelo GPT-5, es un asistente de IA orientado a tareas avanzadas de generación y análisis de texto, incluyendo programación y razonamiento técnico. En el proyecto se ha empleado para apoyar la programación, la depuración de errores y el diseño de pruebas, proponiendo fragmentos de código, explicaciones y ejemplos de tests que han sido posteriormente validados e integrados por la persona desarrolladora. [5]

2.2.3 FastApi

FastAPI es un framework moderno de Python para construir APIs de alto rendimiento, con soporte nativo de tipado, validación de datos mediante modelos y generación automática de documentación OpenAPI. En este TFG se ha utilizado como tecnología principal de backend para definir los endpoints del sistema, los modelos de datos y las respuestas de la API, facilitando tanto la implementación como las pruebas a través de su documentación interactiva. [6]

2.2.4 Pytest

Pytest es un framework de pruebas para Python centrado en la escritura sencilla y escalable de tests unitarios y de integración, con descubrimiento automático de pruebas y un sistema flexible de fixtures. En el proyecto se ha usado para definir y ejecutar pruebas automatizadas sobre la lógica de negocio y sobre los endpoints de la API, comprobando códigos de estado, respuestas y comportamientos esperados en distintos escenarios. [7]

3 Metodología de trabajo

La metodología empleada en el desarrollo de este proyecto combina un enfoque estructurado con un desarrollo iterativo. A nivel general, el trabajo se ha organizado siguiendo un modelo en cascada, en el que las distintas fases del proyecto se han abordado de manera secuencial, permitiendo establecer una base sólida antes de avanzar a la siguiente etapa. Sin embargo, dentro de algunas de estas fases se ha adoptado un enfoque iterativo e incremental, en el que las funcionalidades se han desarrollado, evaluado y refinado de forma progresiva. Este planteamiento ha permitido detectar y corregir errores de manera temprana, integrar mejoras de forma continua y adaptar el desarrollo a las necesidades reales del proyecto, facilitando además la incorporación de herramientas de inteligencia artificial como apoyo efectivo durante todo el proceso.

Los requisitos están basados en el [ANEXO \[B\]](#)

3.1 Plan de trabajo

El trabajo se ha dividido en 5 fases, en 2 de ellas hemos hecho uso de la metodología iterativa incremental para conseguir los mejores resultados, estas fases han sido la toma de requisitos y la implementación, la forma de aplicar la metodología fue un poco distinta en cada caso.

3.1.1 Toma de requisitos

En esta fase empezamos a trabajar con una grabación de una entrevista imitando una reunión desarrollador/cliente, en la cual se comentaban los distintos aspectos que iba a tener la aplicación, con esta entrevista se hicieron unos primeros productos (Brief, requisitos funcionales, casos de uso y wireframes), con estos productos, y principalmente con los wireframes, volvimos a repetir la entrevista, en este caso para ver los casos de uso que se podían haber pasado por alto en la primera entrevista, con esta nueva entrevista se volvieron a hacer todos los productos para ya hacer unos semi definitivos con los que comprobar que la visión del cliente y la mía era la misma.

3.1.2 Arquitectura

La fase de arquitectura se abordó de manera ágil, ya que el tipo de sistema a desarrollar estaba bien definido desde el inicio y no requería un diseño especialmente complejo. Por este motivo, no se aplicó una metodología formal específica más allá de establecer una estructura clara y coherente para la API, definiendo los principales módulos, responsabilidades y relaciones entre ellos. Esta aproximación permitió avanzar rápidamente hacia la fase de implementación sin introducir sobrecarga innecesaria en el proceso de diseño.

3.1.3 Implementación

En esta fase el uso de la metodología fue muy importante para avanzar con el desarrollo ayudado por IA a la vez que el no perdía el contexto ni se acumulaban los errores.

Empecé con una estructura de trabajo vacía para comprender como quería ChatGPT realizar el proyecto, cada iteración añadió funcionalidades acotadas y probaba en cada ciclo antes de añadir cualquier otra funcionalidad, esto me permitía tener un avance controlado, mayor facilidad de localizar los errores y tener siempre un versión reciente y estable

3.1.4 Depuración

La depuración se llevó a cabo siguiendo un enfoque iterativo e incremental, desarrollándose en paralelo con la fase de implementación. Cada nueva

funcionalidad añadida era verificada y depurada antes de avanzar a la siguiente, corrigiendo errores tanto recientes como heredados de iteraciones anteriores. Este proceso continuo permitió mantener un código funcional y estable en todo momento, facilitando además la identificación de problemas y la mejora progresiva del sistema con el apoyo de herramientas de inteligencia artificial.

3.1.5 Pruebas

La fase de pruebas se desarrolló de forma similar a la arquitectura, sin seguir una metodología de pruebas formal o exhaustiva. Las comprobaciones realizadas se centraron en validar el correcto funcionamiento de las funcionalidades implementadas y su integración dentro del sistema. Este enfoque permitió detectar errores de manera temprana y asegurar que cada cambio introducido cumpliera con los requisitos definidos, manteniendo la estabilidad de la API a lo largo del desarrollo.

4 Desarrollo

Este capítulo detalla el proceso de desarrollo a través de las distintas fases del proyecto, comenzando con la extracción de requisitos, donde se definieron las funcionalidades clave de la API web para la gestión de la biblioteca, como la administración de usuarios, libros y préstamos. En la fase de diseño de la arquitectura, se estableció una estructura modular y extensible, permitiendo una integración eficiente de la inteligencia artificial en el flujo de trabajo. La implementación se abordó de manera iterativa, desarrollando y refinando cada módulo de la API, con el apoyo de herramientas como GitHub Copilot para acelerar la generación de código y mejorar la productividad. Durante la fase de depuración, se utilizó un enfoque progresivo para identificar y corregir errores, asegurando la estabilidad y calidad del sistema. Finalmente, en la fase de pruebas, se aplicaron pruebas unitarias y de integración utilizando pytest, evaluando el rendimiento y la funcionalidad de la API, y garantizando que cumpliera con los requisitos establecidos al inicio del proyecto. A lo largo de cada fase, el uso de la inteligencia artificial se combinó con una revisión crítica para optimizar los resultados sin comprometer la calidad del software.

4.1 Extracción de Requisitos

La extracción de requisitos constituye una etapa fundamental en el desarrollo de software, y en este proyecto se ha potenciado mediante el uso de inteligencia artificial. Además, para comprobar la mejor forma de extraer requisitos utilizando la IA llevamos 2 vías de trabajo para darle la información una por transcripciones de las reuniones, y otra dándole la información de mis apuntes de las reuniones, ambas individualizadas en cada una de las iteraciones

4.1.1 Primera iteración: primera generación de documentos

Esta iteración comenzó con una reunión presencial con mis tutores haciendo el papel de cliente y yo el papel de desarrollador. En la reunión se hablaron de los requisitos que el cliente quería que tuviera la aplicación, de los que se tomó nota y se grabó la conversación. Dicha grabación fue posteriormente transcrita para maximizar la precisión y el detalle en la recopilación de requisitos. Con la transcripción y un documento de notas estructuradas en Word, se utilizaron las capacidades de Copilot para transformar estos archivos en un brief (un documento breve que resume los objetivos, requisitos y contexto principal de un proyecto), como se puede ver en la [CONVERSACIÓN 1](#), una lista de requisitos funcionales, una lista de casos de uso completos. Este enfoque permitió

automatizar la identificación y organización de información relevante, logrando mayor coherencia y trazabilidad en la documentación final.



Conversación 1: Generación de brief a partir de transcripción

Con la primera generación de documentos, se procedió a crear wireframes mediante Copilot. En las primeras iteraciones, los wireframes ofrecieron una visión preliminar del sistema, aunque aún resultaban poco definidos como se puede ver en el DIAGRAMA 1.



Diagrama 1: Primeros wireframes

El uso de wireframes como soporte visual facilitó el diálogo, impulsando la colaboración entre el cliente y el equipo de desarrollo y logrando que las

especificaciones finales estuvieran completamente alineadas con las expectativas y requisitos del usuario.

Con estos primeros wireframes se realizó otra reunión en la cual con apoyo de los wireframes se harían modificaciones de los documentos generados anteriormente.

4.1.2 Segunda iteración: mejora wireframes y casos de uso

Con estos primeros wireframes se pasó a hacer una segunda iteración de toma de requisitos que apoyados por los wireframes se vio si faltaba algún caso de uso por reflejar, esta reunión se grabó para su posterior transcripción con lo cual se volverían a sacar los archivos de Brief, requisitos funcionales y casos de uso, como se puede ver en la CONVERSACIÓN 2, también con los cambios especificados se crearon los wireframes definitivos, los cuales fueron más lentos de realizar debido a alto nivel de fiabilidad que se quería reflejar en ellos. Este proceso de refinamiento progresivo facilitado por la inteligencia artificial permitió una validación temprana de las decisiones de diseño y aceleró el prototipado, asegurando que el producto respondiera a las necesidades del usuario final.



The screenshot shows a chat window with a user message: "Transcripcion_grabacion1.docxA partir de estas 2 transcripciones, quiero que me generes un documento de brief del proyecto". Below this, the Copilot response reads: "He revisado ambas transcripciones y puedo generar un **brief del proyecto** que sintetice los puntos clave. Aquí tienes una propuesta estructurada:". The generated brief is titled "Brief del Proyecto: Sistema de Gestión Bibliotecaria" and includes a "Contexto" section: "La universidad actualmente utiliza un sistema manual para la gestión de préstamos y reservas de libros. Se busca desarrollar un sistema digital que permita automatizar estas operaciones, mejorar la experiencia del usuario y optimizar la administración de la biblioteca."

Conversación 2: Segunda iteración

Para garantizar una cobertura completa de las necesidades del cliente, se realizó una tercera reunión dedicada a revisar los wireframes de alto nivel junto con los casos de uso. Este encuentro sirvió para validar la propuesta funcional, detectar posibles carencias y ajustar los detalles finales de la estructura y funcionalidades del sistema.

Para entender mejor todo el proceso de extracción de requisitos se ha elaborado el DIAGRAMA 2

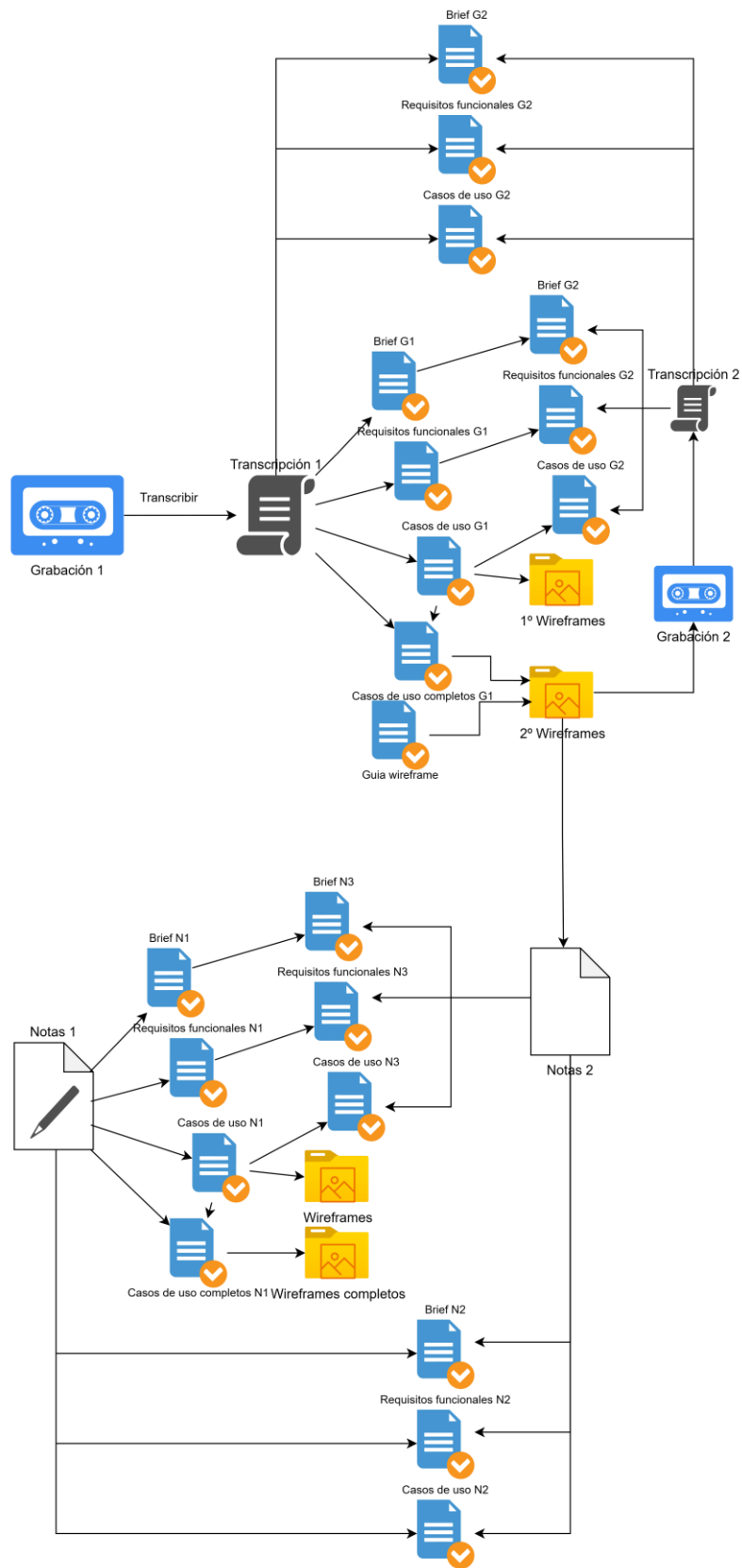


Diagrama 2: Flujo extracción requisitos

4.2 Arquitectura

Durante el diseño arquitectónico, se consultó a Copilot en busca de recomendaciones sobre posibles estructuras alternativas o mejoras. Sin embargo, las sugerencias corroboraron que esta arquitectura, basada en front-end, API y dos bases de datos diferenciadas, era la opción más adecuada para cumplir con los requisitos de eficiencia, escalabilidad y seguridad establecidos en el proyecto. La arquitectura del sistema se basa en tres componentes principales: el front-end, una API y dos bases de datos. El front-end se encarga de la interacción directa con el usuario, mientras que la API actúa como intermediaria para gestionar la lógica de negocio y la comunicación con las bases de datos. En este caso, se utilizan dos bases de datos distintas para cubrir diferentes aspectos del sistema: una base de datos externa perteneciente a la universidad que contiene la información de los usuarios y con la que nuestro sistema interactúa, aunque no sea de nuestra propiedad, y una base de datos propia dedicada al almacenamiento y gestión de la información específica de la biblioteca. A partir de la estructura propuesta hecha por chatGPT se realizó el DIAGRAMA 3 para representarlo

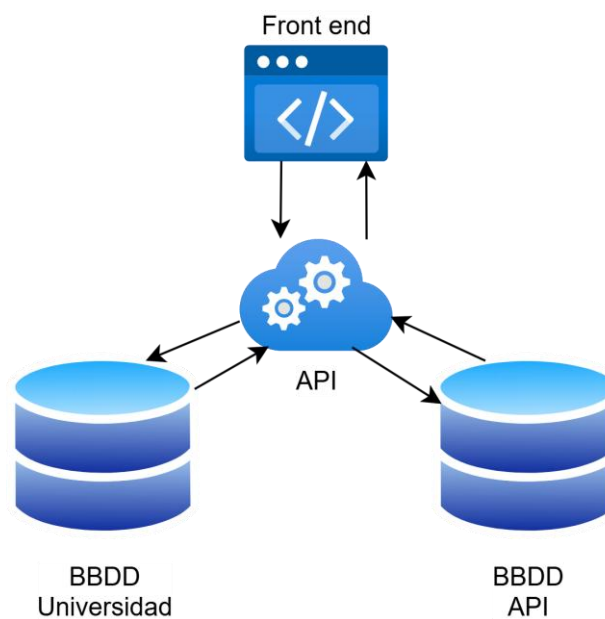


Diagrama 3: Esquema arquitectura

Esta arquitectura con múltiples bases de datos permite mantener un control adecuado de la información sensible y diferenciada según su origen y propósito, facilitando a la vez la integración y seguridad mediante la API como capa de mediación.

Así, la doble base de datos garantiza que la interacción con los datos de usuarios se mantenga segura y controlada, mientras que la base específica de la biblioteca facilite la gestión ágil y eficiente de la información propia del sistema, consolidando una arquitectura sólida y escalable.

4.3 Diseño

En esta app, el diseño consistiría esencialmente en repartir responsabilidades entre el front y el back. No vimos necesario hacer ya que el diseño es una parte más de conceptualización para plantearse como vas a programar, y como la idea era que lo máximo posible lo programase la IA generativa, decidí confiar en la IA para la generación conjunta del front y back

4.4 Implementación

La implementación del código ha sido otra de las partes del trabajo donde más tiempo he dedicado, y donde más importante ha sido el uso de IA

4.4.1 Iteración 0: primer intento

A la hora de hacer la implementación pase por un primer intento que fue pedirle al copilot integrado en VSCode que me realizase todo el trabajo a partir de los casos de uso y los requisitos. Eso acabo con un espacio de trabajo muy completo, y a primera vista muy bien planteado, pero a la hora de levantar el servicio API daban demasiados errores como se puede ver en la imagen CÓDIGO 1, los cuales no era capaz de corregir, o con la corrección de unos se generaban otros. Además que de esta forma tenía poco contexto del funcionamiento del sistema por lo que dificultaba la realización de ajustes en el sistema sin la ayuda de chatGPT, lo cual generaba dependencia operacional respecto a la herramienta para cualquier arreglo

```
(.venv) PS C:\Users\Diego\vscode Workspace\Library-system-api> uvicorn app.main:app
  rv = self.invoke(ctx)
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\click\core.py", line 1269, in invoke
  return ctx.invoke(self.callback, **ctx.params)
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\click\core.py", line 824, in invoke
  return callback(*args, **kwargs)
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\main.py", line 423, in main
  run(
  ~~~^
  app,
  ~~~^
  ...<46 lines>...
  h11_max_incomplete_event_size=h11_max_incomplete_event_size,
  ~~~~~^
  )
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\main.py", line 593, in run
  server.run()
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\server.py", line 67, in run
  return asyncio_run(self.serve(sockets=sockets), loop_factory=self.config.get_loop_factory())
File "C:\Python314\Lib\asyncio\runners.py", line 204, in run
  return runner.run(main)
File "C:\Python314\Lib\asyncio\runners.py", line 127, in run
  return self._loop.run_until_complete(task)
File "C:\Python314\Lib\asyncio\base_events.py", line 719, in run_until_complete
  return future.result()
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\server.py", line 71, in serve
  await self._serve(sockets)
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\server.py", line 78, in _serve
  config.load()
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\config.py", line 439, in load
  self.loaded_app = import_from_string(self.app)
File "c:\Users\Diego\vscode Workspace\Library\venv\Lib\site-packages\uvicorn\importer.py", line 19, in import_from_string
  module = importlib.import_module(module_str)
File "C:\Python314\Lib\importlib\_init_.py", line 88, in import_module
  return _bootstrap.gcd_import(name[level:], package, level)
File "<frozen importlib._bootstrap>", line 1398, in gcd_import
File "<frozen importlib._bootstrap>", line 1371, in find_and_load
File "<frozen importlib._bootstrap>", line 1342, in find_and_load_unlocked
File "<frozen importlib._bootstrap>", line 938, in load_unlocked
File "<frozen importlib._bootstrap_external>", line 762, in exec_module
File "<frozen importlib._bootstrap>", line 491, in call_with_frames_removed
File "C:\Users\Diego\vscode Workspace\Library-system-api\app\main.py", line 3, in <module>
```

Código 1: Errores con el primer intento

4.4.2 Primera iteración: estructura general y despliegue de la API

Viendo que no resultó viable generar el proyecto completo en una única iteración, decidí empezar pidiéndole un sistema sencillo el cual pudiera levantar y, progresivamente, irle añadiendo funciones. De esta forma tenía todo el contexto de la aplicación y al ir añadiendo funciones por fases, los errores eran más fáciles de localizar.

Empecé pidiendo una estructura de proyecto sencilla que pudiera manejar libros que se puede ver en la imagen CONVERSACIÓN 3, usuarios, préstamos y reservas. De esta forma tenía ya el proyecto estructurado, aunque con archivos vacíos.

Dame una estructura de archivos para hacer una api con FastApi que tenga libros usuarios prestamos y reservas

Te propongo una estructura limpia y escalable, pensada para:

- Entidades: libros, usuarios, préstamos, reservas
- Separar: routers, modelos de BD, schemas (Pydantic), servicios/lógica y configuración

text

```
library-api/  
├─ src/  
│ └─ app/  
│   │ └─ main.py  
│   │ └─ core/  
│   │   │ └─ config.py          # Configuración general (settings, variables entorno)  
│   │   │ └─ database.py      # Conexión DB, SessionLocal, Base de SQLAlchemy  
│   │   └─ models/  
│   │     │ └─ user_model.py   # Modelo SQLAlchemy Usuario  
│   │     │ └─ book_model.py  # Modelo SQLAlchemy Libro  
│   │     │ └─ loan_model.py  # Modelo SQLAlchemy Préstamo  
│   │     └─ reservation_model.py # Modelo SQLAlchemy Reserva  
│   └─ schemas/
```

Conversación 3: Estructura inicial del proyecto

Una vez tenía esta estructura lo siguiente fue pedirle que me hiciera los modelos, esquemas, routers y servicios uno de esos modelos se puede ver en la imagen CÓDIGO 2 basados en el documento de casos de uso que salió de las iteraciones de la extracción de requisitos, ninguno de estos archivos sería el definitivo pero ya era una base con la que levantar la API y ver los distintos endpoints.

```

1  from sqlalchemy import Column, Integer, String, Boolean, Enum, ForeignKey, DateTime
2  from sqlalchemy.orm import relationship
3  from sqlalchemy.sql import func
4  from app.core.database import Base
5  import enum
6
7
8  class AccessLevel(str, enum.Enum):
9      GENERAL = "general" # Prestables normalmente
10     REFERENCE = "reference" # Material de referencia (UC-10)
11
12
13  class CopyStatus(str, enum.Enum):
14     AVAILABLE = "available"
15     LOANED = "loaned"
16     RESERVED = "reserved"
17     NOT_LOANABLE = "not_loanable" # Por política
18     LOST = "lost"
19     DAMAGED = "damaged"
20
21
22  class Book(Base):
23     """
24     Título (UC-01, UC-02, UC-12)
25     """
26     __tablename__ = "books"
27
28     id = Column(Integer, primary_key=True, index=True)
29     title = Column(String, nullable=False, index=True)
30     authors = Column(String, nullable=False) # "Autor1; Autor2"
31     isbn = Column(String, index=True, nullable=True)
32     edition = Column(String, nullable=True)
33     subject = Column(String, nullable=True)
34
35     access_level = Column(Enum(AccessLevel), nullable=False, default=AccessLevel.GENERAL)
36
37     created_at = Column(DateTime(timezone=True), server_default=func.now())
38
39     copies = relationship("BookCopy", back_populates="book", cascade="all, delete-orphan")
40     reservations = relationship("Reservation", back_populates="book", cascade="all, delete-orphan")
41
42
43  class BookCopy(Base):
44     """
45     Ejemplar físico con código único (RB-02, BR-1.2)
46     """
47     __tablename__ = "book_copies"
48
49     id = Column(Integer, primary_key=True, index=True)
50     book_id = Column(Integer, ForeignKey("books.id", ondelete="CASCADE"), nullable=False)
51
52     barcode = Column(String, unique=True, index=True, nullable=False) # identificador ejemplar
53     is_reference = Column(Boolean, default=False) # si es material de referencia
54
55     status = Column(Enum(CopyStatus), nullable=False, default=CopyStatus.AVAILABLE)
56
57     created_at = Column(DateTime(timezone=True), server_default=func.now())
58
59     book = relationship("Book", back_populates="copies")
60     loans = relationship("Loan", back_populates="copy")

```

Código 2: Book_model Inicial

Aparte también pedí que me creara el documento requirements.txt para poder instalar todas las librerías necesarias para que funcione la implementación que está realizando, aparte de pedirle la forma de hacer el despliegue de la API y como buscar el swagger con el que manejar y utilizar los distintos endpoints.

4.4.3 Segunda iteración: front, sistema login y diferenciación de vistas

En esta segunda iteración del proyecto se da un paso fundamental en la evolución del sistema de biblioteca: pasar de un conjunto de endpoints funcionales a una aplicación usable desde el punto de vista del usuario final. Hasta este momento, el backend ya contaba con endpoints básicos para la gestión de usuarios, libros, préstamos y reservas, lo que permitía realizar operaciones CRUD sobre cada entidad, pero sin una experiencia de uso integrada ni control de acceso.

La conversación con la IA refleja este cambio de enfoque: se deja atrás el uso de llamadas directas a funciones del frontend, como la carga de préstamos, y se introduce la necesidad de estructurar la aplicación en torno a un sistema de autenticación, roles y vistas diferenciadas.

Punto de partida: frontend sin control de acceso

La existencia de llamadas directas como `get prestamos` indica que el frontend podía acceder a información del sistema sin ningún tipo de restricción previa. En este estado inicial:

- No existía un mecanismo de autenticación.
- El sistema no distinguía entre tipos de usuarios.
- Las vistas no estaban asociadas a permisos concretos.

Aunque este enfoque es habitual en fases tempranas de desarrollo, resulta insuficiente para un sistema de biblioteca real, donde el acceso a la información debe estar condicionado a la identidad del usuario y a su rol dentro del sistema.

Incorporación del sistema de login

Ante esta situación, el siguiente paso natural fue la creación de un sistema de login que actuara como punto de entrada a la aplicación. Esta decisión no solo afecta a la interfaz, sino que introduce un nuevo eje estructural en el frontend.

El sistema de login se diseñó con los siguientes objetivos:

- Identificar al usuario que accede al sistema.
- Asociar cada sesión a un usuario concreto existente en la base de datos.
- Permitir que el frontend conozca el rol del usuario autenticado.

Dado que el backend ya disponía de endpoints básicos, el login se apoya en estos para validar credenciales y devolver la información mínima necesaria (identificador y rol), sin introducir aún lógica avanzada de autorización.

Introducción del rol bibliotecario

Uno de los cambios más relevantes de esta iteración es la introducción de un nuevo rol: el bibliotecario. A diferencia del usuario estándar, este rol representa a la persona encargada de la gestión de la biblioteca y requiere una visión más amplia del sistema.

Desde el punto de vista funcional, el bibliotecario debe poder:

- Consultar información de otros usuarios.
- Crear préstamos y reservas en nombre de terceros.
- Gestionar sanciones asociadas a los usuarios.

Aunque en esta fase del proyecto dichas operaciones se apoyan únicamente en endpoints básicos, la definición del rol resulta clave, ya que condiciona toda la estructura del frontend y sienta las bases para futuras ampliaciones del sistema.

Login como pantalla inicial del sistema

Otro requisito importante que surge en esta iteración es que la primera pantalla visible al acceder a la aplicación sea siempre el login. De este modo:

- Ningún usuario puede acceder directamente a listados de libros, préstamos o reservas sin autenticarse.
- El sistema garantiza que toda interacción parte de una identidad conocida.

A nivel de frontend, esto implica que la aplicación comprueba el estado de autenticación nada más iniciarse y redirige automáticamente al formulario de login cuando no existe una sesión válida.

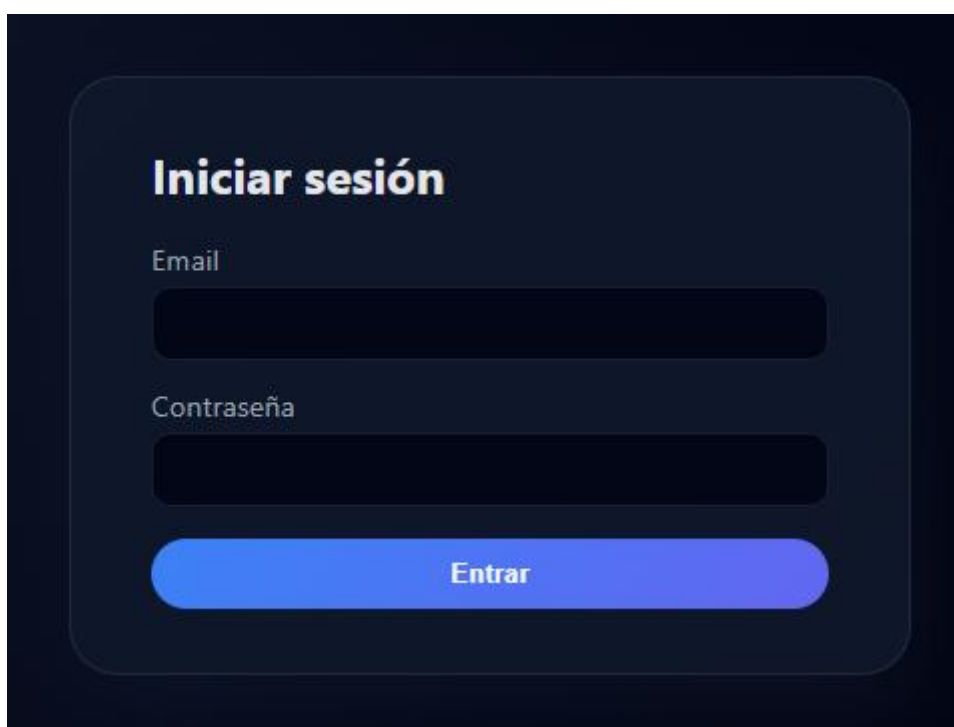


Ilustración 1: Vista del Login

Diferenciación de vistas según el rol del usuario

Una vez autenticado, el sistema redirige al usuario a una vista principal diferente en función de su rol. Esta diferenciación no se limita a mostrar u ocultar botones, sino que define dos flujos de uso claramente separados:

Usuarios normales: acceden a un dashboard orientado a la consulta del catálogo, la visualización de sus propios préstamos y reservas, y la gestión básica de su cuenta.

Bibliotecarios: acceden a un dashboard de administración, desde el cual pueden consultar datos globales del sistema y realizar operaciones sobre usuarios, préstamos y reservas.

Esta separación permite que el frontend refleje de forma clara las responsabilidades de cada rol, incluso cuando la lógica del backend sigue siendo simple.

4.4.4 Tercera iteración: creación catálogo, copias y división js

En la tercera iteración el foco del proyecto se desplaza desde “tener login y dashboards por rol” a construir una funcionalidad central de un sistema de biblioteca: el catálogo. En una biblioteca real, el catálogo no es solo un listado de libros; es el punto desde el que se gestionan altas, bajas, modificaciones, búsquedas y, especialmente, el enlace con los flujos de reservas y préstamos.

Por eso, esta iteración introduce tres mejoras que se retroalimentan:

Un módulo de catálogo completo accesible desde el dashboard del bibliotecario.

Una refactorización del frontend para evitar el crecimiento descontrolado del código (división por pestañas / archivos).

La incorporación de un concepto que faltaba para que el sistema sea realista: copias/ejemplares de un libro.

Aunque el backend sigue basado en endpoints básicos, esta iteración ya obliga a modelar mejor el dominio (qué es un libro vs qué es una copia) y a estructurar el frontend como un sistema mantenible.

Catálogo en el dashboard del bibliotecario

Aquí aparecen tres necesidades funcionales que definen el alcance del catálogo para bibliotecarios:

- Gestión de libros: crear, editar y eliminar.
- Búsqueda: localizar libros rápidamente (por título, autor, ISBN, etc.).
- Operaciones en nombre de terceros: desde el catálogo poder iniciar una reserva para otro usuario.

Esto es importante en memoria de TFG porque refleja cómo el rol bibliotecario deja de ser un “usuario con más permisos” y pasa a tener un flujo propio: gestiona el catálogo y realiza acciones administrativas.

Diseño de la navegación: “click en Catálogo” abre un módulo con subopciones

A nivel de desarrollo frontend, este paso suele implicar convertir el dashboard de bibliotecario en algo más parecido a un “panel”:

- Menú lateral o superior con secciones (Catálogo, Usuarios, Préstamos, Reservas...).
- Al seleccionar Catálogo, se carga una vista con:
 - Listado de libros + buscador
 - Botón “Crear libro”
 - Acciones por libro: “Editar”, “Eliminar”, “Reservar para usuario”

Esto da coherencia a la aplicación: las funciones ya no están “sueltas”, sino agrupadas por propósito.

Formulario de creación de libros: primer flujo CRUD completo en UI

Aunque ya existían endpoints básicos, implementar el formulario obliga a resolver varios detalles prácticos:

- Campos mínimos: título, autor, ISBN, edición, materia y Código de barras
- Validaciones básicas en frontend:
 - obligatorios
 - formato (ISBN si lo incluyes)

- longitudes máximas razonables
- Lógica de estado:
 - al crear, refrescar el listado del catálogo
 - mostrar feedback (éxito/error)

Reservas para otra gente desde el catálogo: unir módulos

Este punto es especialmente relevante: ya no se trata solo de “gestionar libros”, sino de iniciar un flujo:

1. Bibliotecario busca un libro
2. Selecciona “Reservar”
3. Elige un usuario (por id, email o buscador)
4. Crea la reserva “en nombre de” ese usuario

Aunque el backend sea básico, el frontend aquí ya implementa un patrón “admin action” típico:

UI orientada a seleccionar entidades (Libro + Usuario) y crear un recurso (Reserva).

Esto también expone una necesidad que normalmente aparece justo aquí: ¿la reserva es sobre el libro o sobre la copia?

Y esa pregunta enlaza directamente con el siguiente bloque (copias).

División del frontend por pestañas (un .js por pestaña): refactorización por mantenibilidad

Por defecto, la IA tiende a generar soluciones concentradas en un único archivo de gran tamaño, ya que este enfoque resulta más sencillo de producir a partir de un prompt único. Sin embargo, en un proyecto de software real esta estrategia provoca rápidamente una degradación de la calidad del código, afectando negativamente a su legibilidad, dificultando la escalabilidad del sistema, complicando la realización de pruebas y aumentando el esfuerzo necesario para localizar y corregir errores.

En este proyecto, la iteración anterior ya había introducido dashboards diferenciados y gestión de roles, lo que incrementó notablemente la complejidad del frontend. Al añadir en esta tercera iteración el módulo de catálogo, con formularios de creación y edición, buscadores y múltiples acciones asociadas a los libros, el archivo principal comenzó a crecer de forma descontrolada, volviéndose difícil de mantener y evolucionar. La división del código en varias pestañas, con un archivo JavaScript independiente para cada sección, surge así como una respuesta directa a este problema, reduciendo el acoplamiento entre funcionalidades y permitiendo que cada módulo concentre una responsabilidad clara y bien definida.

Creación de copias: corrección del modelo del dominio (Libro ≠ Ejemplar)

Este hallazgo constituye el aspecto más relevante de la iteración desde el punto de vista conceptual. En los sistemas de biblioteca, especialmente en fases tempranas de desarrollo, es habitual cometer el error de modelar únicamente la entidad “Libro”, tratándola como si fuera directamente prestable. Sin embargo, en un contexto real, un libro representa la obra en sí misma —definida por atributos como el título, el autor o el ISBN—, mientras que las copias o ejemplares corresponden a las unidades físicas concretas asociadas a dicha obra. Son estas copias las que realmente se prestan o reservan, o, al menos, las

que determinan la disponibilidad efectiva de un libro. Ignorar esta distinción conduce a modelos incompletos que no reflejan correctamente el funcionamiento real de una biblioteca, lo que hace imprescindible introducir el concepto de copias para garantizar la coherencia del sistema.

Resultado de la tercera iteración

Al finalizar esta tercera iteración, el sistema experimenta un avance significativo tanto a nivel funcional como estructural. El rol de bibliotecario dispone ya de un módulo de Catálogo plenamente operativo, accesible desde su dashboard, que permite gestionar los libros del sistema mediante operaciones de creación, modificación y eliminación directamente desde la interfaz de usuario. Este módulo incorpora además un buscador que facilita la localización de títulos dentro del catálogo y habilita acciones administrativas propias de este rol, como la creación de reservas en nombre de otros usuarios.

Paralelamente, el frontend gana en mantenibilidad y claridad gracias a la división del código en diferentes pestañas y archivos JavaScript, evitando la concentración de toda la lógica en un único fichero y facilitando así la evolución futura del sistema.

Desde el punto de vista del modelado del dominio, se corrige una carencia fundamental al introducir el concepto de copias o ejemplares. Esta mejora permite sentar las bases para un control realista de la disponibilidad de los libros, garantizar la coherencia de los préstamos y dotar de sentido funcional a las reservas dentro de un contexto de biblioteca real.

En conjunto, esta iteración ejemplifica de manera clara el enfoque iterativo seguido en el proyecto: se parte de una base funcional simple, la implementación de la interfaz de usuario revela limitaciones del modelo inicial y, finalmente, se ajusta dicho modelo para lograr un sistema más consistente y alineado con la realidad del dominio.

4.4.5 Cuarta iteración: nuevo diseño

En esta cuarta iteración el objetivo ya no es añadir nuevas entidades o endpoints, sino mejorar de forma deliberada la calidad de la interfaz. Esto es especialmente relevante en un sistema software que ya funciona a nivel básico: cuando el backend y los flujos principales (login, dashboards por rol, catálogo, copias) están operativos, el siguiente cuello de botella suele ser la usabilidad y la percepción de producto. En este caso, el estado anterior era una interfaz “funcional pero pobre”: estilo anticuado, casi sin CSS, con cajas negras sobre fondo claro. Es decir, una UI que cumple, pero que no transmite estructura, jerarquía visual ni comodidad de uso.

Cambios que habría que hacer para mejorar el estilo

Dado el estado anterior (casi sin CSS), el salto a un estilo moderno suele implicar introducir varios elementos que antes no existían:

Un sistema de diseño básico (aunque no sea formal)

En una UI “antigua” típica, cada caja se estiliza como se puede y se repiten decisiones inconsistentes (tamaños, márgenes, fuentes). Para modernizar, se necesita definir reglas mínimas:

- tipografías y tamaños consistentes
- espaciado uniforme (padding/margin)
- colores y jerarquías (fondo, texto, acentos)

- bordes redondeados, sombras suaves, estados hover/focus
- componentes reutilizables (tarjetas, botones, inputs, tablas)

Esto no significa crear un “design system” complejo, sino establecer una base para que el proyecto pueda crecer sin volver al caos.

Rehacer componentes visuales clave

- tarjetas (“cards”) para secciones de dashboard
- tablas legibles para listados (libros, usuarios, préstamos)
- formularios con alineación, etiquetas claras, validación visible
- modales o paneles para acciones (editar, confirmar borrado)

4.4.6 Quinta iteración: mejora funciones existentes

En esta quinta iteración dejé de centrarme en “añadir pantallas nuevas” para enfocarme en algo que, en un sistema de biblioteca real, marca la diferencia entre una demo y una herramienta utilizable: mejorar las funciones existentes para que el flujo de trabajo sea natural y eficiente. Hasta este momento el sistema ya permitía gestionar usuarios, libros, copias, préstamos y reservas con endpoints básicos y una interfaz funcional, pero empecé a notar fricciones claras en el uso diario, sobre todo desde el rol de bibliotecario. En concreto, había operaciones que eran técnicamente posibles, pero poco prácticas porque dependían de datos que un bibliotecario no suele conocer o manejar (como IDs internos), o porque obligaban a procesos manuales innecesarios (como inferir el siguiente código de barras sin ayuda del sistema).

Por eso, esta iteración la orienté a pulir cuatro aspectos: crear préstamos usando correo en lugar de ID, facilitar la creación de copias mostrando el último código de barras, permitir la eliminación de copias y añadir un mecanismo reutilizable para recuperar un usuario a partir del correo.

Cambiar el flujo de creación de préstamos: de “user_id” a “correo”

El primer problema que atacué fue el flujo de creación de préstamos. En una implementación temprana, era normal que el endpoint o la UI pidieran un `user_id` porque es el identificador natural en base de datos. Sin embargo, en la práctica esto es incómodo: el bibliotecario no conoce el ID del usuario, y obligarle a buscarlo previamente rompe el flujo de trabajo.

Lo que hice fue rediseñar el proceso para que el bibliotecario pudiera crear un préstamo usando un dato que sí tiene sentido a nivel operativo: el correo del usuario.

A nivel conceptual, el cambio fue convertir un flujo de dos pasos en uno solo:

- Antes: buscar usuario → copiar id → crear préstamo con `user_id`.
- Ahora: escribir correo → el sistema resuelve el usuario → crear préstamo.

En frontend esto me obligó a cambiar el formulario: donde antes había un campo para el ID, lo sustituí por un input de correo (con validación básica de formato). En backend o en la lógica de integración, necesitaba un punto fiable para convertir correo en usuario. Y ahí apareció el siguiente requisito: recuperar usuario a partir del correo como función transversal.

Recuperar usuario a partir del correo: convertir una necesidad puntual en una utilidad reutilizable

Cuando implementé la creación de préstamos con correo, me di cuenta de que no era un caso aislado. En un sistema de biblioteca, muchas acciones administrativas pueden partir de datos “humanos” (correo, DNI, nombre,

número de socio) y no de IDs internos. Así que en lugar de resolverlo “ad hoc” solo para préstamos, lo diseñé como un mecanismo reutilizable: una función capaz de recuperar un usuario a partir del correo.

En la práctica, esto lo planteé como una pieza común que puedo usar en cualquier operación que necesite un usuario pero donde el bibliotecario introduzca un correo. Es decir, en vez de duplicar lógica, centralicé el comportamiento en un único punto:

1. Validar que el correo tiene un formato aceptable.
2. Consultar al backend el usuario correspondiente.
3. Manejar el caso de usuario inexistente con un error claro para la UI.
4. Devolver el usuario para completar la operación principal.

Este enfoque me ayudó a mantener el sistema coherente: la UI siempre hace lo mismo cuando el bibliotecario trabaja con correos, y los mensajes de error son consistentes. Además, de cara a futuras mejoras, esta función me deja el terreno preparado para extender el sistema a búsquedas por otros atributos sin reescribir el flujo entero.

Mostrar el último código de barras al crear copias: reducir errores y acelerar el proceso manual

El siguiente ajuste nació de un problema realista: al introducir copias, el sistema ya distinguía entre “libro” y “ejemplar”, pero el proceso de alta de copias todavía era poco amigable si el código de barras debía escribirse manualmente.

En bibliotecas, los códigos de barras suelen seguir una secuencia interna. Si el bibliotecario crea una nueva copia, normalmente necesita saber cuál fue el último código asignado para poder escribir el siguiente. Antes, esto obligaba a buscar el último ejemplar manualmente o a inferirlo sin confirmación, lo que aumenta la probabilidad de errores y duplicados.

Para resolverlo, modifiqué el flujo de creación de copias para que, cuando el bibliotecario crea copias de un libro, el sistema le muestre explícitamente el código de barras del último ejemplar existente de ese libro. Así, el bibliotecario tiene una referencia inmediata y puede continuar la secuencia de forma segura.

Este cambio es interesante porque no implica “lógica compleja”, pero mejora mucho la calidad del sistema. No estoy automatizando la generación completa del código (aunque podría ser una iteración futura), pero sí estoy eliminando incertidumbre y reduciendo fricción. En la interfaz, esto se traduce en algo tan simple como mostrar un texto de apoyo del tipo: “Último código de barras registrado: X”, justo al lado del formulario de creación.

Eliminación de copias: completar el ciclo de vida del ejemplar

Una vez que añadí la creación de copias, apareció el siguiente hueco lógico: si un ejemplar se pierde, se da de baja o se registró incorrectamente, el bibliotecario debe poder eliminarlo. Sin esta función, el catálogo se degrada con el tiempo porque el stock queda inflado o incoherente.

Por eso implementé la eliminación de copias como parte de esta iteración. A nivel de UI, esto implica que, dentro de la gestión de copias asociadas a un libro, cada ejemplar tenga una acción de “Eliminar” con confirmación, para evitar borrados accidentales.

A nivel conceptual, aquí también tuve que pensar en coherencia con el dominio: eliminar una copia no debería ser posible si está asociada a un préstamo activo

o a una reserva vigente (aunque en tu estado actual comentas que los endpoints son básicos y quizás esa validación aún no está). Aun así, dejé el flujo preparado para que, si el backend devuelve un error de integridad o conflicto, la UI lo represente de forma clara y el bibliotecario entienda por qué no se puede eliminar.

De cara al TFG, este punto es clave porque muestra una evolución típica: cuando se introduce una entidad nueva (copias), no basta con “crearla”, hay que implementar su ciclo completo: creación, consulta y baja.

Resultado de la quinta iteración: mejoras orientadas a uso real

Al final de esta iteración, el sistema no cambió tanto en “cantidad de funcionalidades” como en calidad operativa. Lo que conseguí fue que el bibliotecario pudiera trabajar de una forma más natural:

- Ya no depende de IDs internos para crear préstamos, sino que puede usar el correo.
- El sistema incluye una utilidad reutilizable para recuperar usuarios por correo, lista para aplicarse en otras funciones administrativas.
- La creación de copias se volvió más segura y rápida gracias a la referencia del último código de barras registrado.
- La gestión de copias quedó completa al incorporar la eliminación de ejemplares, evitando inconsistencias en el stock.

Esta iteración encaja muy bien con un enfoque de desarrollo asistido por IA porque refleja un patrón real: una vez que la IA ayuda a levantar la estructura del sistema y las pantallas principales, el trabajo de mayor valor está en detectar fricciones y transformarlas en mejoras concretas que hacen que el sistema sea realmente utilizable.

4.4.7 Sexta iteración: creación pestaña préstamos

En la sexta iteración decidí centrarme específicamente en la gestión de los préstamos desde el punto de vista del bibliotecario. Aunque el sistema ya permitía crear préstamos y realizar devoluciones, faltaba una vista centralizada que permitiera supervisarlos de forma global y operar sobre ellos de manera ágil. En una biblioteca real, los préstamos no se gestionan de forma aislada, sino como un conjunto que requiere priorización, seguimiento y toma de decisiones continuas.

Por ello, el objetivo de esta iteración fue crear una pantalla administrativa dedicada exclusivamente a los préstamos y dotarla de las herramientas necesarias para que el bibliotecario pudiera trabajar de forma eficiente en su día a día.

Creación de la pantalla de administración de préstamos

El primer paso de esta iteración fue la creación de un nuevo archivo de frontend, `adminPrestamos.js`, que actúa como pantalla principal para la gestión de préstamos. Hasta este momento, los préstamos existían dentro del sistema, pero no contaban con un espacio propio desde el que gestionarlos de forma centralizada.

Esta nueva pantalla se diseñó con un enfoque claramente administrativo. Su finalidad no es solo mostrar información, sino servir como punto de control desde el cual el bibliotecario puede consultar el estado de los préstamos y actuar

directamente sobre ellos. De esta forma, los préstamos pasan a tratarse como una entidad principal dentro de la aplicación, al mismo nivel que libros o usuarios.

Listado de préstamos ordenado por fecha de finalización

Uno de los requisitos clave de esta pantalla fue que los préstamos se mostraran ordenados por los que terminan antes. Esta decisión responde a una necesidad operativa muy concreta: en el trabajo diario de una biblioteca, los préstamos próximos a vencer son los que requieren mayor atención.

Al presentar primero los préstamos con fechas de finalización más cercanas, el sistema ayuda al bibliotecario a priorizar tareas como devoluciones, renovaciones o posibles sanciones. Este criterio convierte el listado de préstamos en una herramienta activa de trabajo y no simplemente en una tabla informativa.

Búsqueda de préstamos por correo del usuario

Otro aspecto importante de esta iteración fue la incorporación de un sistema de búsqueda por correo electrónico del usuario. En lugar de obligar al bibliotecario a localizar manualmente un préstamo dentro de una lista extensa, se le permite introducir directamente el email del usuario y filtrar los préstamos asociados.

Este enfoque refuerza una idea clave que ya se había trabajado en iteraciones anteriores: el sistema debe adaptarse a la forma real de trabajar del bibliotecario, que identifica a los usuarios por datos comprensibles y cotidianos, como el correo electrónico, y no por identificadores internos del sistema.

La búsqueda por correo mejora significativamente la rapidez de atención y reduce la fricción en situaciones habituales, como consultas en mostrador o gestiones telefónicas.

Acciones disponibles sobre cada préstamo: devolver, renovar y sancionar

Una vez definido el listado y los mecanismos de búsqueda, el siguiente paso fue dotar a cada préstamo de acciones claras y accesibles. En esta iteración se definieron tres acciones principales, visibles directamente en la lista de préstamos:

Devolver préstamo, que permite cerrar el préstamo de forma inmediata cuando el ejemplar es entregado

Renovar préstamo, pensada para los casos en los que el usuario solicita una ampliación del plazo.

Sancionar préstamo, que introduce la posibilidad de gestionar situaciones problemáticas, como retrasos o incumplimientos de las normas.

Estas acciones completan el ciclo de vida del préstamo desde el punto de vista del bibliotecario y evitan la necesidad de navegar a otras pantallas o realizar procesos indirectos. El préstamo deja de ser un elemento pasivo y se convierte en una entidad sobre la que se puede actuar directamente.

Resultado de la sexta iteración: una gestión de préstamos orientada a uso real

Al finalizar esta sexta iteración, la gestión de préstamos del sistema quedó notablemente reforzada:

Se creó una pantalla administrativa específica (adminPrestamos.js) para centralizar la gestión de préstamos.

Los préstamos se muestran ordenados por fecha de finalización, facilitando la priorización del trabajo diario.

Se incorporó la búsqueda por correo electrónico del usuario, alineando el sistema con el uso real en una biblioteca.

Cada préstamo incluye acciones claras para devolver, renovar o sancionar, cubriendo los casos más habituales.

Esta iteración refleja una evolución natural del proyecto: una vez que las funcionalidades básicas están implementadas, el valor está en organizarlas, hacerlas visibles y adaptarlas a un flujo de trabajo realista. Además, encaja especialmente bien en el contexto de un TFG sobre desarrollo asistido por IA, ya que muestra cómo la IA facilita la creación rápida de nuevas pantallas, mientras que el diseño del flujo y las decisiones funcionales siguen recayendo en el desarrollador.

4.4.8 Séptima iteración: creación Alembic

Para gestionar los cambios en el esquema de la base de datos a lo largo del desarrollo, decidí incorporar un sistema de migraciones. Para ello utilicé la IA como apoyo en la configuración inicial de Alembic, herramienta estándar para el control de versiones del esquema en proyectos basados en SQLAlchemy.

En primer lugar, con ayuda de la IA identifiqué Alembic como la solución más adecuada para mantener sincronizados los modelos de la aplicación y la estructura real de la base de datos. A continuación, inicialicé Alembic dentro del proyecto, lo que generó la estructura básica de directorios y archivos de configuración necesarios para trabajar con migraciones.

Posteriormente, configuré el archivo alembic.ini para establecer la conexión con la base de datos del proyecto. En paralelo, adapté el archivo env.py para integrar Alembic con los modelos definidos mediante SQLAlchemy, de forma que la herramienta pudiera detectar automáticamente los cambios en las entidades y generar migraciones a partir de ellos.

Una vez completada esta configuración, establecí un flujo de trabajo basado en la generación de migraciones cada vez que se producía una modificación en los modelos (creación de tablas, cambios en columnas, relaciones, etc.). Con ayuda de la IA, confirmé los comandos necesarios para crear nuevas revisiones y aplicar dichas migraciones a la base de datos de forma incremental, garantizando así la coherencia del esquema entre las distintas versiones del proyecto.

Este enfoque me permitió evolucionar la base de datos de manera controlada durante el desarrollo iterativo, reduciendo errores y facilitando la trazabilidad de los cambios estructurales, al mismo tiempo que la IA actuó como apoyo puntual para resolver dudas de configuración y buenas prácticas.

4.4.9 Cambio de conversación

Tras completar esta iteración de la implementación, fue necesario iniciar una nueva conversación con la IA. La conversación anterior había acumulado un volumen elevado de mensajes y contexto, lo que provocó que las respuestas comenzaran a ser más lentas y menos ágiles. Para evitar que esto afectara al

ritmo de desarrollo, opté por reiniciar la interacción manteniendo únicamente la información relevante.

Con el fin de que la IA recuperara rápidamente el contexto general del proyecto, copié y pegué el árbol de archivos completo de la aplicación y proporcioné una descripción detallada de la finalidad del proyecto, su arquitectura y las principales tecnologías utilizadas. De este modo, la IA pudo reconstruir una visión global del sistema sin necesidad de arrastrar todo el historial previo de la conversación.

Este procedimiento me permitió seguir utilizando la IA de forma eficiente como herramienta de apoyo durante la implementación, asegurando que las respuestas se adaptaran correctamente a la estructura real del proyecto y a las decisiones de diseño ya tomadas, sin incurrir en los problemas derivados de una ventana de contexto excesivamente cargada.

4.4.10 Octava iteración: adminSanciones

Antes de iniciar la implementación, revisé el modelo de datos de sanciones existente para asegurar que incluía todos los campos necesarios: identificador de usuario, correo electrónico, motivo, fechas y estado de la sanción. Con apoyo de la IA, definí los criterios funcionales del módulo, centrados en mostrar todas las sanciones registradas y permitir su filtrado. También determiné que el acceso a esta funcionalidad debía estar restringido a usuarios con rol administrativo.

Diseño del listado de sanciones en el panel de administración

Diseñé la vista de listado de sanciones pensando en la claridad y la usabilidad. Cada sanción se muestra con la información más relevante y la estructura permite una rápida revisión por parte del administrador. Preparé la vista de manera que pudiera ampliarse en futuras iteraciones con nuevas columnas o funcionalidades adicionales, manteniendo coherencia con el resto del panel de administración.

Implementación de la funcionalidad de búsqueda

Para mejorar la experiencia del usuario, añadí filtros de búsqueda que permiten localizar sanciones de forma eficiente. Se implementaron búsquedas por correo electrónico y por nombre de usuario, que pueden utilizarse de forma independiente o combinada. Esto facilita encontrar rápidamente los registros relevantes sin necesidad de recorrer manualmente todo el listado de sanciones.

- Búsqueda por correo electrónico del usuario.
- Búsqueda por nombre de usuario.

Desarrollo de la función listSanctions

Para centralizar la lógica de recuperación de datos, desarrollé la función listSanctions. Esta función acepta parámetros opcionales para los filtros y construye la consulta de forma dinámica, aplicando únicamente los filtros que se proporcionan. La IA me ayudó a validar la lógica de filtrado y a mantener la función clara, legible y fácilmente mantenible.

4.4.11 Novena iteración: adminReservas

En la novena iteración me centré en la creación del módulo de administración de reservas, con el objetivo de que los administradores pudieran visualizar todas las reservas registradas y gestionar nuevas sanciones relacionadas con usuarios.

Antes de comenzar la implementación, analicé la estructura de datos de reservas y definí los requisitos funcionales principales:

- Visualización completa de todas las reservas registradas en el sistema.
- Capacidad de búsqueda por correo electrónico del usuario asociado a la reserva.
- Funcionalidad para crear nuevas sanciones a partir de reservas problemáticas o incumplidas.

Con la ayuda de la IA, revisé la coherencia de los modelos existentes y cómo integrarlos de forma eficiente con el módulo de sanciones que ya había desarrollado en la iteración anterior.

Diseño de la vista de administración de reservas

Para la interfaz de administración, diseñé una vista que permite a los administradores consultar todas las reservas de manera clara y estructurada. Cada reserva muestra información clave como usuario, correo electrónico, fecha de la reserva y estado actual. Preparé la vista de forma que fuera fácil de ampliar en el futuro, por ejemplo, para añadir filtros adicionales o exportación de datos.

La IA me ayudó a definir cómo enlazar la vista de reservas con la creación de sanciones, asegurando que el flujo fuera intuitivo y consistente con la lógica del sistema.

Implementación de la búsqueda por correo electrónico

Implementé un sistema de filtrado que permite localizar reservas asociadas a un usuario específico mediante su correo electrónico. Esta funcionalidad mejora la eficiencia administrativa y evita la necesidad de revisar manualmente todas las reservas registradas.

Búsqueda por correo electrónico: el filtro se aplica solo si se proporciona un valor, evitando afectar la consulta en su totalidad.

La lógica de filtrado se integró con el backend de manera que puede reutilizarse en otros módulos administrativos si es necesario.

Creación del endpoint postSanctions

Para poder gestionar sanciones derivadas de reservas, desarrollé un endpoint llamado postSanctions que permite crear nuevas sanciones asociadas a usuarios que incumplen normas o condiciones del sistema. Con la ayuda de la IA, estructuré la función para:

1. Validar los datos de entrada y asegurar que se cumplieran las restricciones de integridad del sistema.
2. Asociar correctamente la sanción con el usuario correspondiente y, opcionalmente, con la reserva que la motivó.
3. Registrar la sanción en la base de datos y devolver una respuesta clara que confirme su creación.

Esto garantiza que la administración de reservas y la gestión de sanciones estén completamente integradas, evitando inconsistencias y simplificando la labor del administrador.

4.5 Depuración

Con cada una de las iteraciones de la implementación, antes de seguir implementando pasaba por una fase de depuración en la que comprobaba que las funciones cumplieran no solo con lo que tenían que hacer, sino que buscaba mejorar la forma en que chatgpt lo había implementado.

4.5.1 Iteración 0: intento generación código completo

Antes de iniciar el desarrollo iterativo propiamente dicho, realicé una primera prueba que denominé Iteración 0, en la que intenté que la IA (Copilot) generara todas las funcionalidades de la aplicación de forma automática. El objetivo era evaluar si era posible acelerar significativamente el desarrollo inicial mediante la generación completa de código.

Sin embargo, este enfoque no funcionó como esperaba. La IA produjo un volumen de código muy extenso, pero al ejecutarlo surgieron numerosos errores que dificultaban la compilación y la integración de los distintos módulos. La acumulación de fallos hizo que ni siquiera pudiera desplegar la API correctamente.

4.5.2 Primera iteración

Durante la depuración de esta iteración detecté un error crítico al ejecutar la API: FastAPI intentaba utilizar el módulo `dependencies` completo como si fuera una función, lo que provocaba fallos al inyectar dependencias en los endpoints. Al revisar el código, me di cuenta de que el problema se debía a la forma en que estaba realizada la importación: estaba importando el módulo completo en lugar de la función específica o el objeto concreto que debía ser utilizado para gestionar la base de datos.

Con ayuda de la IA, revisé la documentación y recibí sugerencias sobre cómo corregir la importación. La solución consistió en:

1. Importar explícitamente la función que devuelve la sesión de la base de datos, en lugar del módulo completo.
2. Actualizar todos los endpoints y lugares donde se inyectaba la dependencia para que utilizaran la función correcta.
3. Comprobar que la sesión de la base de datos se abría y cerraba correctamente en cada petición, evitando errores de conexión o de alcance de la sesión.

Tras aplicar estos cambios, la API dejó de lanzar errores relacionados con la inyección de dependencias y la conexión con la base de datos funcionó de manera estable. La IA me ayudó a identificar la causa raíz del problema y a estructurar la importación de forma adecuada, asegurando consistencia y buena práctica en el manejo de dependencias de FastAPI.

4.5.3 Segunda iteración

Durante esta iteración me encontré con varios errores que impedían el correcto funcionamiento de la aplicación:

Error en la importación de `require_roles`

Al intentar proteger ciertos endpoints con roles de usuario, FastAPI lanzaba un error indicando que `require_roles` no existía en el módulo `dependencies`. Al revisar el código, detecté que el problema se debía a que `require_roles` estaba definido en el módulo `security` y no en `dependencies`, por lo que la importación era incorrecta.

- Con la ayuda de la IA, realicé las siguientes acciones para solucionarlo:
- Localicé correctamente la función `require_roles` dentro del módulo `security`.
- Actualicé todas las importaciones en los endpoints y tests que utilizaban esta dependencia para apuntar al módulo correcto.

Verifiqué que la protección de roles funcionara como se esperaba, asegurando que solo los usuarios con permisos adecuados podían acceder a las rutas correspondientes.

Error al crear nuevos usuarios

Además, no era posible crear nuevos usuarios en la base de datos. Tras analizar los logs y la estructura del proyecto, comprobé que los cambios recientes en los esquemas y modelos habían provocado inconsistencias entre la base de datos existente y los nuevos modelos de SQLAlchemy.

Para solucionarlo, con ayuda de la IA, realicé los siguientes pasos:

- Eliminé la base de datos antigua para eliminar conflictos de esquema.
- Reapliqué la creación de la base de datos desde cero, asegurando que reflejara los cambios recientes en los modelos.
- Validé que la creación de usuarios funcionara correctamente, y que los datos se almacenaran de manera coherente con el nuevo esquema.

Gracias a estas correcciones, tanto la gestión de roles como la creación de usuarios funcionaron correctamente, permitiendo continuar con las siguientes iteraciones de implementación y depuración de forma estable.

4.5.4 Tercera iteración

En esta iteración me centré en la creación del catálogo y en la reorganización de los archivos JavaScript, dividiendo el código en módulos más pequeños y creando copias para facilitar el mantenimiento y la escalabilidad.

Durante la depuración surgió un problema importante: la función de cerrar sesión en el panel de administración dejó de funcionar. Esto ocurrió porque al dividir el JS, la función `logout` quedó en un archivo que no estaba correctamente vinculado con la plantilla de administración, por lo que el botón de cerrar sesión ya no podía invocarla.

Para resolver este error con ayuda de la IA, llevé a cabo los siguientes pasos:

1. Localicé el archivo JS donde estaba definida la función `logout` y analicé cómo se importaban los scripts en la plantilla HTML del panel de administración.
2. Reubiqué la función en un archivo común que se cargara en todas las vistas del panel, asegurando su disponibilidad global.
3. Actualicé la referencia en el botón de cerrar sesión para que llamara a la función correcta.
4. Verifiqué que la función funcionara correctamente en todos los navegadores y que no se produjeran errores de consola.

Gracias a estas correcciones, la funcionalidad de cerrar sesión volvió a estar operativa, garantizando que los administradores pudieran salir de la sesión de manera segura. Además, la división del código JS permitió un mantenimiento más ordenado y sentó las bases para futuras iteraciones del proyecto.

4.5.5 Cuarta iteración

En esta iteración trabajé en la implementación del nuevo diseño de la interfaz de usuario, buscando actualizar estilos, reorganizar elementos y mejorar la experiencia visual. Al aplicar los primeros cambios detecté que los estilos CSS no se aplicaban correctamente, y que el side-bar no mostraba la estructura ni el display esperado. Esto provocaba que la apariencia de la página fuera inconsistente y que algunos elementos no fueran interactivos como debía.

Con ayuda de la IA, identifiqué que los problemas se debían principalmente a la forma en que se cargaban los archivos CSS y a conflictos entre selectores en distintas hojas de estilo. Para resolverlos, realicé las siguientes acciones:

1. Revisé y corregí la ruta de los archivos CSS en todas las plantillas para asegurar que se cargaran correctamente.
2. Ajusté los selectores de CSS conflictivos y corregí el display del side-bar para que se mostrara y se comportara adecuadamente en todas las vistas.
3. Comprobé que los cambios de estilo se reflejaran en tiempo real y que no existieran conflictos con otros componentes o estilos globales.
4. Validé la compatibilidad visual en distintas resoluciones y navegadores para asegurar una experiencia uniforme.

Tras estas correcciones, los cambios de diseño se aplicaron correctamente, mejorando la consistencia visual de la interfaz y dejando la base lista para nuevas mejoras estéticas o funcionales.

4.5.6 Quinta iteración

Durante esta iteración me centré en optimizar y corregir las funciones existentes, en particular la función de creación de préstamos. Al probar los cambios, surgieron varios problemas que impedían su correcto funcionamiento:

- Creación de préstamos a partir del correo electrónico: la función debía permitir crear préstamos usando el correo del usuario en lugar de su ID, pero ninguna función previa recuperaba al usuario a partir del correo. Esto provocaba errores al intentar asociar el préstamo a un usuario inexistente en el contexto del backend.
 - Con la ayuda de la IA, implementé una función que buscaba en la base de datos al usuario correspondiente según su correo.
 - Modifiqué la función de creación de préstamos para que utilizara esta búsqueda y asociara correctamente al usuario con el préstamo.
 - Validé que la nueva lógica funcionara correctamente con distintos correos existentes y que no generara errores si el correo no coincidía con ningún usuario.
- Inconsistencia en el número de parámetros: la función de préstamo esperaba recibir 2 variables, pero el frontend estaba enviando 3.
 - Revisé el flujo de datos desde el frontend y ajusté la función para que procesara correctamente el parámetro adicional cuando era relevante, o lo ignorara en caso contrario.
 - Probé diferentes combinaciones de datos para asegurar que la función respondiera adecuadamente ante distintas configuraciones de entrada.

Gracias a estas correcciones, la función de creación de préstamos pasó a funcionar de forma consistente, permitiendo tanto la asociación correcta de

usuarios mediante correo electrónico como el manejo adecuado de los parámetros enviados desde el frontend. Esto permitió avanzar en la integración de la funcionalidad de préstamos sin introducir inconsistencias en los datos.

4.5.7 Sexta iteración

En esta iteración implementé la pestaña de préstamos en la interfaz, permitiendo a los administradores visualizar y gestionar los préstamos registrados en el sistema. Durante las pruebas, surgieron varios problemas que impedían que la funcionalidad se mostrara correctamente en la web:

- Cambios de código no reflejados en la interfaz: al modificar los archivos, la web seguía mostrando la versión anterior debido a la caché del navegador.
 - Solución: realicé una recarga completa de la página usando Ctrl + F5, asegurando que se cargaran los archivos JavaScript y CSS actualizados.
- Falta de endpoint `getPrestamos`: la pestaña no podía mostrar los datos porque no existía un endpoint que devolviera los préstamos.
 - Con ayuda de la IA, implementé el endpoint `getPrestamos` en el backend, asegurando que devolviera la información de todos los préstamos de manera estructurada y filtrable según los criterios necesarios.
- Información incompleta de libros y usuarios: el modelo `Loan` solo almacenaba los IDs de los usuarios y libros, por lo que la pestaña no podía mostrar datos como el nombre del usuario o el título del libro.
 - Añadí consultas en el backend para recuperar los datos completos a partir de los IDs.
 - Modifiqué la respuesta del endpoint para incluir toda la información relevante, asegurando que la interfaz pudiera mostrar los datos completos.

Tras aplicar estas correcciones, la pestaña de préstamos comenzó a funcionar correctamente. Los administradores podían ver todos los préstamos con información completa de usuarios y libros, y los cambios en el código se reflejaban correctamente en la interfaz sin problemas de caché. Esto permitió avanzar con la integración de nuevas funcionalidades de gestión de préstamos de manera estable.

4.5.8 Séptima iteración

Durante esta iteración trabajé en la configuración de Alembic para gestionar las migraciones de la base de datos y mantener sincronizados los modelos de SQLAlchemy con la estructura real de la base de datos. Al probar la generación y aplicación de migraciones, detecté un problema crítico: Alembic no apuntaba a la misma base de datos que los modelos del proyecto, lo que provocaba que los cambios realizados en los modelos no se reflejaran correctamente al aplicar migraciones.

Para solucionar este problema con la ayuda de la IA, realicé los siguientes pasos:

1. Revisión de la configuración de Alembic: inspeccioné el archivo `alembic.ini` y verifiqué la cadena de conexión, asegurándome de que coincidiera con la base de datos utilizada por SQLAlchemy.

2. Actualización del env.py: ajusté el archivo env.py de Alembic para que importara correctamente los modelos del proyecto y pudiera detectar los cambios realizados en ellos.
3. Generación de migraciones iniciales: creé una migración inicial que alineara la base de datos con la estructura de los modelos actuales.
4. Pruebas de migración: ejecuté comandos de upgrade y downgrade para asegurarme de que las migraciones funcionaban correctamente y que se podían revertir sin errores.
5. Validación de coherencia: verifiqué que la base de datos resultante estuviera completamente sincronizada con los modelos y que las futuras migraciones pudieran aplicarse de manera segura.

Gracias a estas correcciones, Alembic quedó correctamente configurado y listo para gestionar las migraciones de manera confiable, asegurando la coherencia entre los modelos del proyecto y la base de datos en cada iteración futura.

4.5.9 Octava iteración

Durante la implementación del módulo Admin Sanciones, surgieron varios errores que impedían la funcionalidad completa y estable del sistema:

- Falta de endpoints necesarios: no existían los endpoints getSanctions ni getSanctionsByUserId, lo que impedía recuperar las sanciones de manera general o filtradas por usuario.
- Información incompleta del usuario: el modelo Sanction solo almacenaba los IDs de los usuarios y no permitía recuperar datos como nombre o correo electrónico.
- Errores en las rutas: la nomenclatura de la ruta de sanctions estaba incorrecta, y esto generaba conflictos en las llamadas a la API.
- Duplicación de endpoints: se habían creado endpoints duplicados debido a que el prefix de las rutas estaba definido dos veces.

Para resolver estos problemas, con ayuda de la IA realicé las siguientes acciones:

- Creación de los endpoints faltantes: implementé getSanctions y getSanctionsByUserId para permitir consultas completas y filtradas por usuario.
- Recuperación completa de datos de usuario: modifiqué las consultas en el backend para unir los datos de la sanción con los datos del usuario correspondiente, permitiendo mostrar información completa en la interfaz administrativa.
- Corrección de rutas y prefijos: ajusté los nombres y eliminé la duplicación de los prefijos en la configuración de FastAPI, evitando conflictos y llamadas incorrectas.
- Pruebas exhaustivas: verifiqué que todos los endpoints devolvieran datos correctos, incluyendo pruebas con múltiples usuarios y sanciones, asegurando la consistencia y la estabilidad del módulo.

Gracias a estas correcciones, el módulo Admin Sanciones quedó totalmente operativo: se podían visualizar sanciones completas, filtrarlas por usuario y gestionar toda la información de manera estable desde el panel administrativo. Esto consolidó la funcionalidad administrativa y permitió continuar con las siguientes iteraciones sin problemas de integración.

4.5.10 Novena iteración

En esta iteración me centré en el módulo AdminReservas, que permite a los administradores visualizar y gestionar las reservas realizadas por los usuarios. Durante las pruebas no se presentaron errores significativos. La API y los endpoints ya estaban estables y completamente funcionales, lo que indicaba que las iteraciones anteriores de depuración habían consolidado la base del sistema.

Pude verificar que:

- La visualización de reservas funcionaba correctamente, mostrando todos los datos relevantes de cada usuario y reserva.
- Los filtros de búsqueda por correo electrónico o nombre de usuario devolvían resultados precisos y consistentes.
- La integración con los módulos de sanciones y préstamos era correcta, y no se producían conflictos ni duplicaciones de endpoints.
- La interfaz administrativa respondía de manera estable y coherente, reflejando todos los cambios realizados en el backend.

Esta estabilidad permitió que la API alcanzara un estado robusto y confiable, listo para futuras ampliaciones o ajustes, sin necesidad de intervención adicional en esta iteración de depuración.

4.6 Pruebas

Durante la fase de verificación y validación del sistema, se llevó a cabo un proceso completo de pruebas del código con el objetivo de garantizar el correcto funcionamiento de la API, el cumplimiento de las reglas de negocio y la seguridad en el acceso a los distintos recursos. Para ello, se definió una estrategia de testeo automatizado basada en `pytest`, que permitió ejecutar todas las pruebas desde un único comando y de forma reproducible, sin depender de la base de datos real del sistema como se puede ver en la imagen CÓDIGO 3.

En primer lugar, se desarrollaron tests de integración y funcionales sobre los distintos endpoints de la API. Estos tests verifican el comportamiento esperado de cada recurso (usuarios, libros, préstamos, reservas y sanciones), comprobando tanto los códigos de respuesta HTTP como los datos devueltos. Para asegurar el aislamiento del entorno de pruebas, se utilizó una base de datos SQLite específica para testing, que se crea y destruye automáticamente durante la ejecución de los tests. Además, se definieron *fixtures* para generar usuarios de prueba con distintos roles, crear datos de dominio y obtener tokens de autenticación reales mediante el endpoint de login, permitiendo validar

correctamente los mecanismos de autenticación y autorización.

```

===== tests coverage =====
coverage: platform win32, python 3.14.0-final-0
-----
Name                               Stmts  Miss  Cover   Missing
-----
app\__init__.py                     0      0  100%
app\core\__init__.py                0      0  100%
app\core\config.py                  15      0  100%
app\core\database.py                 6      0  100%
app\dependencies.py                  8      4   50%    8-12
app\main.py                          24      0  100%
app\models\__init__.copy.py         0      0  100%
app\models\__init__.py              6      0  100%
app\models\book_model.py            38      0  100%
app\models\loan_model.py            20      0  100%
app\models\reservation_model.py     29      0  100%
app\models\sanction_model.py        22      0  100%
app\models\user_model.py            26      0  100%
app\routers\__init__.py             0      0  100%
app\routers\auth_router.py          20      0  100%
app\routers\book_router.py          34      8   76%    32, 37, 47, 56-57, 66, 75-76
app\routers\loan_router.py          25      4   84%    27, 38, 43, 48
app\routers\reservation_router.py   31      7   77%    28, 41, 54, 57, 69-72
app\routers\sanction_router.py      20      2   90%    38, 47
app\routers\user_router.py          47     17   64%    32, 43, 61, 66-73, 77-82
app\schemas\__init__.py            0      0  100%
app\schemas\book_schema.py         28      0  100%
app\schemas\loan_schema.py         17      0  100%
app\schemas\reservation_schema.py  13      0  100%
app\schemas\sanction_schema.py     11      0  100%
app\schemas\user_schema.py         16      0  100%
app\services\__init__.py            0      0  100%
app\services\book_service.py        66     42   36%    11, 41-51, 56-85, 93-96, 100-110, 114-119, 123-124, 138-147
app\services\loan_service.py        88     57   35%    27-35, 42, 45, 78-127, 131-190, 194
app\services\policies_service.py    11      6   45%    34, 38-43
app\services\reservation_service.py 53     12   77%    15, 19, 24, 27, 31, 47, 55, 85, 88, 97, 119-120
app\services\sanction_service.py    24      3   88%    15, 36, 46
app\services\user_service.py         8      4   50%    8-14
app\utils\__init__.py               0      0  100%
app\utils\security.py               59      9   85%    23-27, 63, 87-89, 93
-----
TOTAL                               765    175   77%
27 passed, 25 warnings in 1.18s

```

Código 3: resultado test de cobertura

Posteriormente, se incorporaron tests de cobertura con la herramienta `pytest-cov`, con el fin de medir qué partes del código eran ejecutadas durante las pruebas. Esta etapa permitió identificar posibles zonas no cubiertas y reforzar los tests en aquellos puntos críticos de la lógica de negocio y de la API. De este modo, la cobertura se utilizó como una métrica de calidad que ayuda a reducir errores y a prevenir regresiones futuras en el desarrollo del sistema.

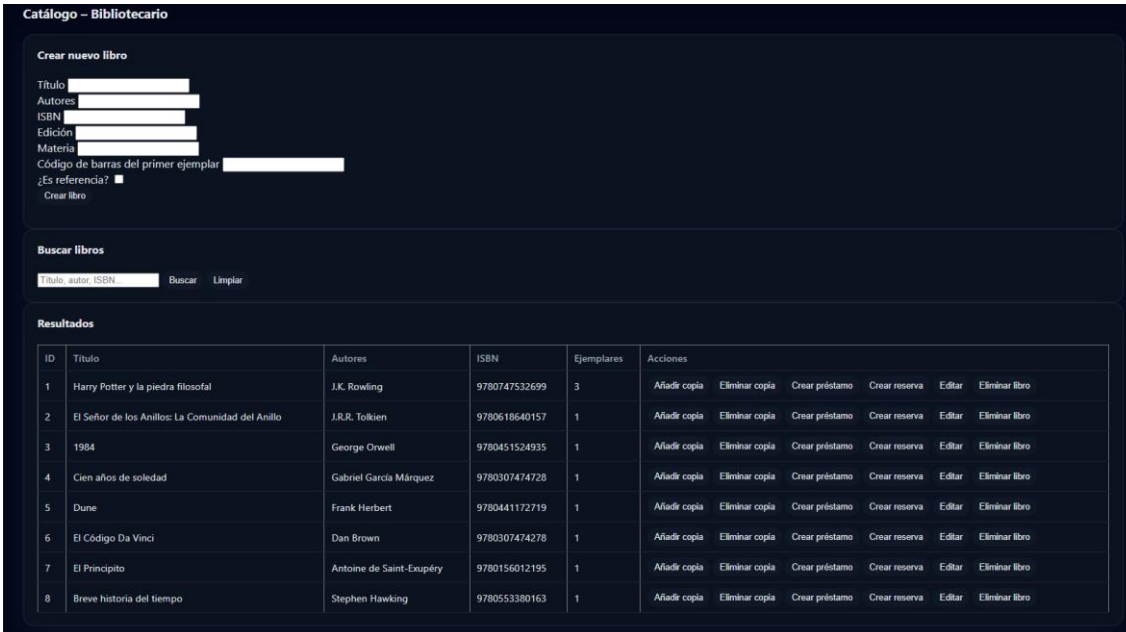
Finalmente, se realizaron tests de tipo end-to-end (E2E), en los que se simula el uso real de la API mediante peticiones HTTP completas. Estos tests validan flujos completos del sistema, como el inicio de sesión, el acceso a endpoints protegidos mediante token, la creación y consulta de recursos, así como la aplicación de restricciones de permisos según el rol del usuario. Adicionalmente, se comprobó el contrato de la API a través del esquema OpenAPI, asegurando que todas las rutas están correctamente registradas y accesibles.

En conjunto, este enfoque de testeo automatizado proporciona una validación integral del sistema, asegurando la coherencia entre la lógica de negocio, la seguridad y el diseño de la API. La combinación de tests funcionales, cobertura de código y pruebas end-to-end permite consolidar una base de software robusta, mantenible y preparada para su evolución futura.

5 Resultados

En este apartado se presentan los resultados obtenidos tras el proceso de implementación iterativa de la aplicación. Como resultado final, se ha desarrollado una interfaz gráfica funcional que cubre los principales casos de uso definidos para la gestión bibliotecaria: consulta del catálogo, gestión de préstamos, reservas y sanciones. A continuación, se describen las distintas ventanas del sistema, apoyándose en capturas de pantalla representativas.

En primer lugar, la ventana de Catálogo, mostrada en la ILUSTRACIÓN 1, permite visualizar el conjunto de obras disponibles en el sistema. Como se observa en la imagen del catálogo, la interfaz presenta la información de forma estructurada, facilitando la consulta de los ejemplares y el acceso a las operaciones asociadas. Esta ventana constituye el punto de entrada principal para la interacción con los fondos bibliográficos.



Catálogo - Bibliotecario

Crear nuevo libro

Título

Autores

ISBN

Edición

Materia

Código de barras del primer ejemplar

¿Es referencia?

Crear libro

Buscar libros

Título, autor, ISBN Buscar Limpiar

Resultados

ID	Título	Autores	ISBN	Ejemplares	Acciones
1	Harry Potter y la piedra filosofal	J.K. Rowling	9780747532699	3	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
2	El Señor de los Anillos: La Comunidad del Anillo	J.R.R. Tolkien	9780618640157	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
3	1984	George Orwell	9780451524935	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
4	Cien años de soledad	Gabriel García Márquez	9780307474728	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
5	Dune	Frank Herbert	9780441172719	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
6	El Código Da Vinci	Dan Brown	9780307474278	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
7	El Principito	Antoine de Saint-Exupéry	9780156012195	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro
8	Breve historia del tiempo	Stephen Hawking	9780553380163	1	Añadir copia Eliminar copia Crear préstamo Crear reserva Editar Eliminar libro

Ilustración 2: Catalogo en la web

La ventana de Préstamos, ilustrada en la ILUSTRACION 2, centraliza la gestión de los préstamos activos. Tal y como se aprecia en la imagen de préstamos, el usuario puede consultar los préstamos en curso y acceder a la funcionalidad necesaria para registrar nuevos préstamos, lo que permite un control efectivo del estado de los ejemplares prestados.



Panel de bibliotecario

Bienvenido, Diego

Gestión de préstamos

Buscar préstamos por email de usuario

Buscar Limpiar

Préstamos (ordenados por fecha de fin)

ID	Usuario	Email	Libro	Ejemplar	Fecha fin	Estado	Acciones
2	Pedro	user@alumnos.upm.es	Harry Potter y la piedra filosofal	HP-002	6/1/2026	returned	Devolver Prolongar Sancionar
1	Pedro	user@alumnos.upm.es	Harry Potter y la piedra filosofal	HP-003	21/1/2026	returned	Devolver Prolongar Sancionar

Ilustración 3: Prestamos en la web

Por su parte, la ventana de Reservas, mostrada en la ILUSTRACIÓN 3, permite gestionar las reservas realizadas por los usuarios. Como se ve en la imagen de reservas, esta pantalla ofrece una visión clara de las reservas existentes, facilitando su seguimiento y administración dentro del sistema.

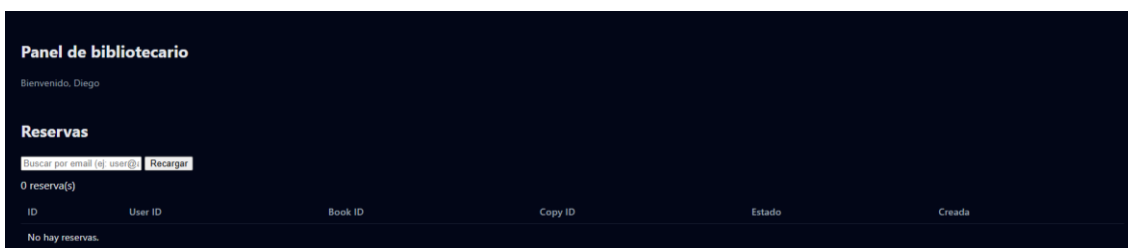


Ilustración 4: Reservas en la web

La ventana de Sanciones, presentada en la ILUSTRACION 4, recoge la información relativa a las penalizaciones aplicadas a los usuarios. En la imagen de sanciones se observa cómo el sistema muestra de forma ordenada las sanciones registradas, contribuyendo a una gestión transparente y coherente de las mismas.

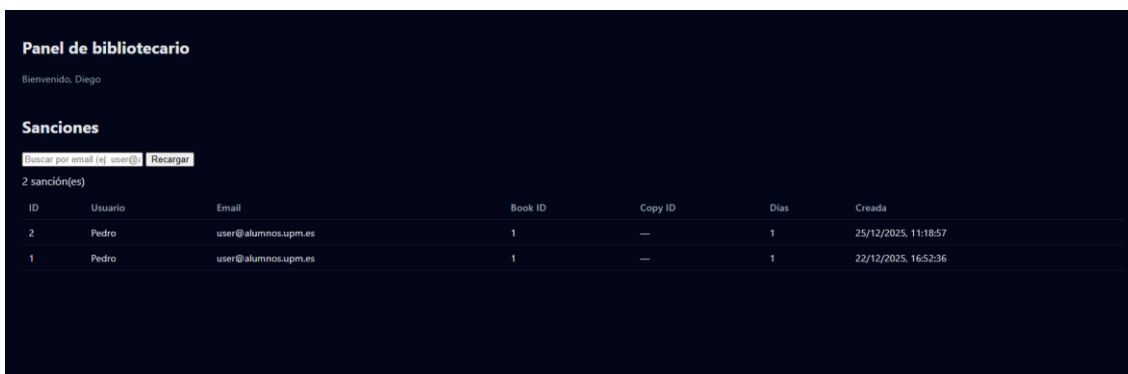


Ilustración 5: Sanciones en la web

Finalmente, la aplicación hace uso de diversos cuadros de diálogo para guiar al usuario en la realización de las distintas acciones disponibles en el sistema, como la gestión de préstamos, reservas u otras operaciones relacionadas. Estos cuadros de diálogo permiten introducir información adicional o confirmar decisiones antes de completar cada acción. A modo de ejemplo, en la ILUSTRACIÓN 5 se muestra uno de estos cuadros de diálogo, correspondiente a una de las operaciones de préstamo, en el que el sistema solicita la selección del ejemplar sobre el que se va a realizar la acción

127.0.0.1:8000 dice

Selecciona el ejemplar (copia) que quieres prestar (solo disponibles):

1. HP-001 [available]
2. HP-002 [available]
3. HP-003 [available]

Introduce el número de la copia a prestar:

Aceptar Cancelar

Ilustración 6: Cuadro dialogo de prestamos

En conjunto, las imágenes presentadas evidencian que la aplicación resultante cumple con los requisitos funcionales definidos, ofreciendo una interfaz clara y coherente. El enfoque de desarrollo iterativo, apoyado en el uso de Copilot para la implementación de nuevas funcionalidades, ha permitido obtener un sistema plenamente operativo y alineado con los objetivos planteados al inicio del trabajo.

6 Lecciones aprendidas

Este capítulo detalla el proceso de desarrollo a través de las distintas fases del proyecto, comenzando con la extracción de requisitos, en la que se definieron las funcionalidades clave de la API web para la gestión de la biblioteca, como la administración de usuarios, libros y préstamos. En la fase de diseño de la arquitectura, se estableció una estructura modular y extensible que permitió integrar de manera eficiente la inteligencia artificial en el flujo de trabajo. Esta arquitectura facilitó la implementación de nuevas características de manera incremental y permitió un control detallado sobre la evolución del sistema.

En la fase de implementación, el desarrollo se abordó de manera iterativa, utilizando herramientas como GitHub Copilot para acelerar la generación de código y mejorar la productividad. No obstante, a lo largo de esta fase también se encontraron limitaciones, como la generación de código que, en ocasiones, no estaba completamente optimizado o requería ajustes manuales. Esto evidenció la necesidad de un enfoque crítico al utilizar la inteligencia artificial, especialmente en aspectos que requerían una adaptación específica al contexto del proyecto. Además, surgieron algunas malas prácticas, como la dependencia excesiva de las sugerencias automáticas de la IA sin una revisión detallada, lo que en ocasiones derivó en soluciones subóptimas.

Finalmente, en la fase de depuración y pruebas, se adoptó un enfoque progresivo para identificar y corregir errores, asegurando la estabilidad y la calidad del sistema. Las pruebas unitarias y de integración se realizaron utilizando pytest, evaluando el rendimiento y la funcionalidad de la API de acuerdo con los requisitos establecidos. Aunque el uso de la inteligencia artificial facilitó el proceso, se destacó la importancia de mantener un control manual constante para evitar que las limitaciones y malas prácticas asociadas a estas herramientas afectaran negativamente el desarrollo. A lo largo de todas las fases, se evidenció la necesidad de un equilibrio entre la automatización proporcionada por la IA y la supervisión crítica del desarrollador para garantizar un producto final de alta calidad.

6.1 Requisitos

A lo largo del desarrollo del proyecto se han extraído diversas lecciones relacionadas con el uso de herramientas de inteligencia artificial durante la toma de requisitos y el trabajo iterativo. Una de las principales conclusiones es que la IA resulta especialmente útil en fases tempranas y conceptuales del proyecto, siempre que se utilice con el nivel de detalle adecuado. En la toma de requisitos, por ejemplo, el uso de la IA para la creación de wireframes de baja fidelidad ha demostrado ser una práctica eficaz como se ve en el [DIAGRAMA 4](#). Este tipo de wireframes permite validar rápidamente que la visión del cliente y la del desarrollador están alineadas, ofreciendo una representación general del diseño, el número de páginas y la ubicación de las funcionalidades sin

necesidad de un alto nivel de detalle. La rapidez con la que la IA genera estos bocetos facilita la comunicación inicial y acelera la validación de ideas.

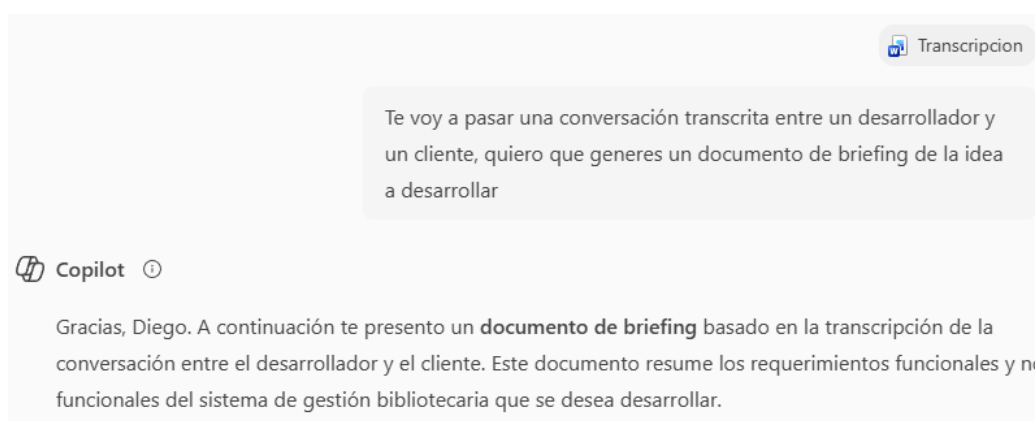


Diagrama 4: Wireframe bajo nivel

Sin embargo, esta misma experiencia pone de manifiesto que el uso de la IA para generar wireframes de alta fidelidad no resulta recomendable. Cuando se requiere un nivel elevado de precisión visual, la IA tiende a introducir pequeños errores en textos, colores o formas, lo que obliga a realizar múltiples iteraciones para corregirlos. Además, estos wireframes son poco flexibles ante cambios pequeños, ya que cualquier ajuste suele implicar rehacer partes importantes del diseño. Esto convierte a la IA en una herramienta poco eficiente para este tipo de tareas, siendo más adecuado recurrir a herramientas de diseño específicas y reservar la IA para fases más conceptuales.

Otra lección relevante se ha extraído del uso de la IA en la toma de requisitos mediante transcripciones automáticas de reuniones como se puede ver en la CONVERSACIÓN 4. Sustituir la toma de notas manual por transcripciones generadas por IA ha permitido recoger de forma literal todo lo expresado por el cliente, reduciendo interpretaciones subjetivas y minimizando el riesgo de malentendidos. Además, esta práctica ha mejorado la calidad de las reuniones, ya que el desarrollador puede centrarse plenamente en la conversación y en formular preguntas relevantes. Posteriormente, la información puede ser

analizada, resumida y estructurada con ayuda de la IA, aumentando la precisión y eficiencia en esta fase inicial del proyecto.



Conversación 4: Generación de documentos usando una transcripción

En cuanto al trabajo iterativo, una lección clave ha sido la importancia de proporcionar a la IA los resultados consolidados de iteraciones anteriores. Cuando la IA dispone de versiones ya validadas como referencia, es capaz de mantener la coherencia de estilo, estructura y contenido, centrándose únicamente en aplicar las modificaciones solicitadas como se ve en la ILUSTRACIÓN 6. Esto reduce la introducción de errores innecesarios y mejora la eficiencia del proceso de mejora continua. Por el contrario, basarse únicamente en grabaciones o notas de iteraciones previas provoca que la IA pierda el contexto global, generando versiones inconsistentes que requieren correcciones adicionales para recuperar la coherencia del trabajo.

Brief del Proyecto: Sistema de Gestión de Biblioteca

1. Objetivo

El sistema busca gestionar libros, préstamos, reservas y usuarios de forma eficiente.

2. A alcance

Cubre registro de libros, gestión de usuarios, préstamos, devoluciones y reservas.

3. Público objetivo

Alumnos, profesores y personal administrativo de la universidad.

4. Beneficios

Optimización del tiempo, control de inventario y mejora en la experiencia del usuario.

Notas 1 y Notas 2

Briefing del Proyecto: Sistema de Gestión de Biblioteca

1. **Objetivo del Proyecto**

Desarrollar un sistema digital para la gestión integral de una biblioteca universitaria. Este sistema permitirá registrar libros, gestionar préstamos, reservas y devoluciones, así como administrar distintos tipos de usuarios con funcionalidades específicas según su rol.

2. **Módulos Principales del Sistema**

Gestión de Libros

- Registro de libros con los siguientes datos:
 - Nombre
 - ID único
- Estado de los libros:
 - Prestado
 - Reservado
 - Devuelto
- Acceso restringido:
 - Algunos libros o juegos estarán disponibles solo para profesores.

Gestión de Usuarios

Tipos de usuarios:

- **Usuarios generales (alumnos):**
 - Correo electrónico
 - Matrícula
 - DNI
 - Nombre
- **Personal universitario (PAS y profesores):**
 - Nombre
 - DNI
 - Correo electrónico

Brief 1 y Notas 2

Ilustración 7: Comparativa entre briefs

6.2 Arquitectura

Durante el desarrollo del proyecto se extrajeron diversas lecciones relacionadas con el diseño y la gestión de la arquitectura del sistema. Al tratarse de una API web con una estructura relativamente estándar, la arquitectura inicial pudo definirse de forma rápida y sin la necesidad de aplicar una metodología arquitectónica formal. No obstante, el trabajo práctico permitió identificar aspectos clave que influyeron directamente en la mantenibilidad y evolución del sistema.

Una de las principales lecciones aprendidas fue la importancia de definir una arquitectura clara desde las primeras fases del desarrollo, incluso cuando el sistema no presenta una complejidad elevada. Contar con una separación bien delimitada entre capas —como rutas, lógica de negocio, modelos y dependencias— facilitó la incorporación progresiva de nuevas funcionalidades y redujo el impacto de los cambios realizados durante las iteraciones de implementación y depuración. Esta estructura permitió además que el uso de herramientas de inteligencia artificial fuera más efectivo, ya que el código seguía patrones coherentes y fácilmente reconocibles.

Al mismo tiempo, la experiencia mostró que una arquitectura excesivamente rígida en fases tempranas puede resultar contraproducente. En varias iteraciones fue necesario ajustar la organización de módulos, mover funciones entre archivos o redefinir responsabilidades para adaptarse a nuevas necesidades funcionales. Este proceso evidenció que una arquitectura flexible, capaz de evolucionar junto con el sistema, resulta más adecuada en proyectos desarrollados de forma iterativa e incremental.

Otra lección relevante estuvo relacionada con la coherencia entre la arquitectura del código y la configuración del entorno, especialmente en lo referente a la gestión de dependencias y a la base de datos. Problemas como discrepancias entre la configuración de Alembic y los modelos del proyecto pusieron de manifiesto que la arquitectura no se limita únicamente al código fuente, sino que también incluye archivos de configuración y herramientas auxiliares. Mantener esta coherencia fue clave para evitar errores difíciles de detectar y para garantizar la estabilidad del sistema.

Por último, el uso de la IA en el contexto arquitectónico evidenció que estas herramientas son más eficaces cuando trabajan sobre una base bien estructurada, pero menos fiables cuando se les delega la definición completa de la arquitectura. La experiencia confirmó que la IA resulta adecuada como apoyo para validar decisiones, generar alternativas o resolver problemas concretos, pero que la responsabilidad del diseño arquitectónico debe recaer en el desarrollador. Esta lección refuerza la idea de que una arquitectura bien pensada no solo mejora la calidad del sistema, sino que también maximiza el valor del uso de herramientas de inteligencia artificial durante el desarrollo.

6.3 Implementación

Durante la fase de implementación del proyecto se obtuvieron diversas lecciones relacionadas con el uso de herramientas de inteligencia artificial como apoyo al desarrollo del código. Una de las conclusiones más relevantes fue la importancia de abordar la implementación de forma progresiva e incremental, especialmente cuando se trabaja con ayuda de una IA. Definir inicialmente una estructura básica de la API permitió establecer una base sólida y comprensible sobre la que ir incorporando, de manera gradual, nuevas funcionalidades y el frontend. Este enfoque facilitó la validación continua del sistema, la detección temprana de

errores y una mejor comprensión de la evolución del proyecto, reduciendo la complejidad de los cambios y mejorando el control sobre el desarrollo.

En contraposición, la experiencia demostró que intentar implementar el proyecto completo en una única iteración con ayuda de la IA constituye una práctica poco recomendable. Generar grandes volúmenes de código de una sola vez incrementó notablemente la aparición de errores desde fases tempranas y dificultó su localización posterior. La ausencia de validaciones intermedias provocó que los fallos fueran más costosos de identificar y corregir, generando una falsa sensación de avance rápido que, en la práctica, derivó en un mayor retrabajo y pérdida de tiempo. Además, este enfoque redujo la comprensión del proceso de construcción del sistema y de las decisiones técnicas adoptadas, afectando negativamente a la mantenibilidad y evolución futura del proyecto.

Otra lección clave estuvo relacionada con la necesidad de proporcionar a la IA un contexto completo sobre la estructura del proyecto antes de solicitar la creación o modificación de funcionalidades. Facilitar el árbol de archivos, la organización del código y las relaciones entre módulos permitió que la IA comprendiera mejor el sistema y generara soluciones más coherentes. Asimismo, preguntar explícitamente qué archivos necesitaba analizar para realizar un cambio concreto redujo errores frecuentes, como se muestra en la CONVERSACIÓN 5, como la suposición de funciones inexistentes, nombres incorrectos o interpretaciones erróneas del comportamiento del sistema. Esta práctica contribuyó a disminuir el número de iteraciones necesarias por funcionalidad y a mejorar la calidad del código generado.

Quiero normalizar los codigos HTTP dime que documentos necesitas que te pase para que lso edites

Perfecto. Para normalizar correctamente los códigos HTTP (sin romper tests ni comportamiento), necesito ver solo los puntos donde FastAPI decide el status code, es decir: routers + algunos services.

📄 Archivos que necesito que me pases (en este orden)

1 Routers (imprescindibles)

Aquí es donde se definen los `@router.post(...)`, `@router.delete(...)`, etc.

Pásame completos:

- `app/routers/auth_router.py`
- `app/routers/user_router.py`
- `app/routers/book_router.py`
- `app/routers/loan_router.py`
- `app/routers/reservation_router.py`
- `app/routers/sanction_router.py`

👉 Con esto puedo:

- poner `201 Created` donde toca
- `204 No Content` en deletes
- asegurar `401 / 403 / 404 / 409` coherentes
- dejar la API REST "canónica"

Conversación 5: Modificaciones con contexto

De forma complementaria, la experiencia evidenció que la falta de contexto es una de las principales fuentes de ineficiencia en el uso de IA durante la implementación. Cuando no se proporciona información suficiente sobre el estado actual del proyecto, su arquitectura o las decisiones técnicas previas, la IA tiende a basarse en suposiciones que pueden no ajustarse a la realidad del sistema. Esto da lugar a código que no se integra correctamente, introduce incoherencias o requiere múltiples correcciones posteriores, aumentando innecesariamente el esfuerzo de desarrollo.

Otra lección aprendida fue la importancia de supervisar activamente la organización del código generado por la IA. En algunos casos, la herramienta tendió a concentrar demasiadas responsabilidades en un único archivo o módulo, especialmente en el desarrollo del frontend. Este tipo de soluciones contraviene principios básicos de diseño software como la modularidad y la separación de responsabilidades, generando código difícil de leer, mantener y escalar. Guiar a la IA para distribuir correctamente la lógica en componentes y módulos bien definidos resultó fundamental para preservar la claridad y sostenibilidad del proyecto como se ve en la CONVERSACIÓN 6

Hay alguna forma de dividir las funciones del js en varios archivos o organizarlo de otra forma, ya que con el catalogo del admin se mete mucho texto, y aun faltan muchas partes más de la web, meterlo todo en 3 archivos (index.html, app.js y styles.css) me parece poco escalable y reparable

Sí, totalmente: meter todo en un solo `app.js` es una bomba de relojería 🤡

Te propongo una forma **ordenada y sencilla** de modularizar el frontend sin meter aún Webpack/Vite ni nada raro.

1 Nueva estructura de carpetas para el front

Dentro de `src/app/static` puedes dejar algo así:

```
text Copiar código

static/
├─ index.html
├─ styles.css
└─ js/
  ├─ main.js           # Punto de entrada
  ├─ api.js           # Funciones para llamar a la API (fetch, headers...)
  ├─ auth.js          # Login, logout, currentUser
  ├─ router.js        # Cambio de vistas (login / user / admin)
  ├─ userCatalog.js   # Catálogo para usuarios normales
  ├─ adminCatalog.js  # Catálogo para bibliotecario
  └─ ...              # más módulos (prestamos.js, sanciones.js, etc.)
```

Conversación 6: Dividir las funciones en distintos archivos

Por último, se constató que la IA resulta especialmente eficaz en tareas complementarias como la generación de documentación técnica, por ejemplo en la elaboración del archivo README del proyecto. Aunque el contenido generado

requirió posteriormente un proceso de revisión y unificación, la IA fue capaz de producir documentación completa y estructurada incluso con instrucciones mínimas, llegando incluso a aportar valor añadido mediante la propuesta de posibles líneas de evolución o backlog de tareas futuras.

En conjunto, estas lecciones ponen de manifiesto que el uso eficaz de la inteligencia artificial durante la implementación no depende únicamente de la herramienta en sí, sino de cómo se integra dentro de un proceso de desarrollo controlado, incremental y supervisado, en el que el desarrollador mantiene el control sobre las decisiones técnicas y utiliza la IA como un apoyo, no como un sustituto.

6.4 Depuración

Durante la fase de depuración del proyecto se constató que el uso eficaz de herramientas de inteligencia artificial depende en gran medida de la forma en que se formulan las solicitudes y del contexto proporcionado. Una de las principales lecciones aprendidas fue la necesidad de identificar y compartir con la IA las partes relevantes del código antes de solicitar una corrección. Preguntar explícitamente qué archivos, módulos o fragmentos necesita analizar permite que la herramienta comprenda mejor el origen del problema y genere soluciones más ajustadas al funcionamiento real del sistema, evitando respuestas genéricas o basadas en suposiciones.

Esta práctica resultó especialmente importante en un proyecto con múltiples módulos interrelacionados, donde un error aparente en una parte del sistema podía tener su origen en otra. Al facilitar a la IA acceso a los componentes implicados, las correcciones propuestas tendían a ser más coherentes y a integrarse mejor con el resto del código, reduciendo el número de iteraciones necesarias para resolver cada incidencia.

De forma complementaria, la experiencia puso de manifiesto una limitación habitual: proporcionar únicamente el mensaje de error generado por el sistema no es suficiente para una depuración eficaz. Los mensajes de error rara vez reflejan el contexto completo en el que se produjo el fallo, por lo que la IA carece de información sobre la funcionalidad que se estaba desarrollando, el comportamiento esperado o los cambios recientes introducidos en el código. Esta falta de contexto conduce con frecuencia a soluciones parciales o incorrectas que no abordan la causa real del problema. Como se ve en las CONVERSACIONES 7-9, en las que se ve como le paso el código de error pero y

chatGPT no es capaz de saber a que se debe (duda entre dos opciones) CONVERSACIÓN 7 Y me pide que le mande unos documentos, en la siguiente,

```
apiPost @ api.js:38
sancionarPrestamo @ adminPrestamos.js:285
handleLoansTableClick @ adminPrestamos.js:209Entender
este error
adminPrestamos.js:294 Error creando sanción Error:
{"detail":"Not Found"}
  at apiPost (api.js:45:11)
  at async sancionarPrestamo (adminPrestamos.js:285:5)
  at async
HTMLTableSectionElement.handleLoansTableClick
(adminPrestamos.js:209:5)
```

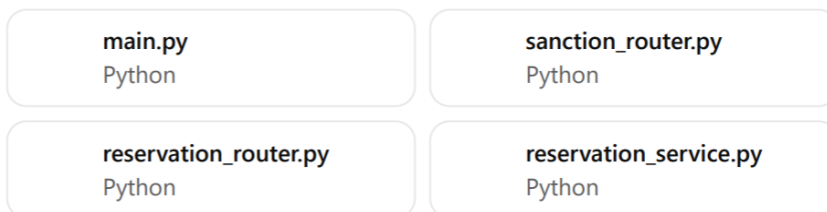
Ese 404 en POST /sanctions/ ya NO puede ser "falta la barra" (ya la pusiste).

Significa una de estas dos cosas (casi seguro):

1. Tu router de sanciones NO está incluido en main.py , por eso FastAPI no conoce /sanctions/ .
2. Tu API está montada con otro prefijo (por ejemplo /api/sanctions/) y tú estás llamando a /sanctions/ .

Conversación 7: Le Mando solo el error y no es capaz de saber a que se debe el error

CONVERSACIÓN 7 le paso algunos de los documentos que me pidió, y ya es capaz de darme las modificaciones y la causa, aun así le faltan archivos para arreglar el error por completo



Aqui te subo 2 de los 4 archivos que me has pedido, no existe __init__.py en routers ni un routers padre, ademas te he pasado el router y el service de reservations para que

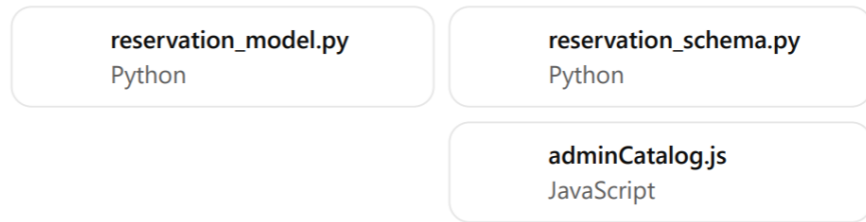
18e23b04d3d6dd5e0-deberes/c/69497280-6200-832a-a630-143315aedf46

ChatGPT - Deberes

los hagas, tienes que hacer funciones para que más adelante a través del front se puedan reservar libros que estén todas sus copias prestadas

Conversación 8: Añadir documentos para contexto y cambio

y ya en la última CONVERSACIÓN 8 le mando todos los archivos necesarios y me corrige el error



Aparte de cambiar reservation_schema y reservation_model, modifica admin_catalog para poder crear reservas de libros

Pensó durante 14s >

Genial — con lo que has subido ya puedo hacerlo “bien” y coherente con tu arquitectura actual.

Conversación 9: Añadir los Documentos que faltaban

A partir de esta experiencia, se concluyó que una depuración efectiva con ayuda de la IA requiere describir no solo el error, sino también el escenario en el que se produce y las modificaciones realizadas previamente. Este enfoque permite obtener respuestas más precisas y refuerza el papel del desarrollador como responsable del análisis y la validación de las soluciones propuestas, utilizando la IA como una herramienta de apoyo dentro de un proceso de razonamiento crítico.

6.5 Pruebas

Durante la fase de pruebas se obtuvieron principalmente lecciones relacionadas con la validación del contenido generado con ayuda de herramientas de inteligencia artificial, más que con aspectos puramente funcionales del sistema. En este contexto, una de las principales lecciones aprendidas fue la necesidad de revisar y ajustar de forma crítica los textos generados por la IA antes de integrarlos en la aplicación. Los modelos de lenguaje tienden en ocasiones a utilizar un tono excesivamente informal o artificial, así como a incluir elementos innecesarios como emojis, lo que puede resultar inadecuado en un entorno profesional o académico.

Esta revisión manual permitió adaptar el estilo del contenido al contexto del proyecto, garantizando una comunicación clara, coherente y alineada con el público objetivo de la aplicación. La fase de pruebas sirvió, por tanto, no solo para verificar el correcto funcionamiento del sistema, sino también para evaluar la calidad y adecuación del contenido generado, reforzando el papel del desarrollador como responsable último de la validación.

En cuanto a las limitaciones, no se identificaron problemas relevantes asociados al uso de la IA durante esta fase. El proceso de pruebas fue breve y estuvo centrado en ajustes menores, lo que puso de manifiesto que, una vez estabilizado el sistema, el impacto de la IA en esta etapa es reducido y

principalmente complementario. Esta experiencia sugiere que el mayor valor de las herramientas de inteligencia artificial se concentra en fases previas del desarrollo, como la implementación y la depuración, mientras que en la fase de pruebas su uso se limita a tareas de apoyo y refinamiento como se puede ver en la CONVERSACIÓN 10 con pedirle que hiciera pruebas era capaz de crearmelas incluso dandome opciones de tipos de pruebas.

Vamos a parar la implementacion por ahora, ahora quiero realizar un testeo completo de la API automatizado, como me recomiendas empezar?

Empieza por montar una base de tests que puedas ejecutar en 1 comando y que no dependa de tu BD real. La receta típica (y la que mejor escala) es: **pytest + TestClient + SQLite en memoria + fixtures para usuario/token + factories de datos.**

1) Decide el "nivel" de test (y en qué orden)

Conversación 10: Petición de creación de pruebas

7 Conclusiones

Este capítulo presenta las conclusiones del Trabajo Fin de Grado, centrado en el análisis del uso de inteligencias artificiales generativas —en concreto GitHub Copilot y ChatGPT— como apoyo al desarrollo de software. Se sintetizan los principales hallazgos del proyecto, destacando que estas herramientas resultan especialmente útiles cuando se integran en procesos iterativos y controlados, donde su función es complementar al desarrollador en tareas concretas y no sustituirlo en el diseño o la toma de decisiones. Asimismo, se ofrecen recomendaciones prácticas para su uso adecuado, se identifican limitaciones técnicas y metodológicas, y se examinan los principales riesgos asociados, como la pérdida de control del sistema o la posible dependencia de la herramienta. El capítulo concluye subrayando la importancia de mantener un equilibrio entre automatización y supervisión humana y relaciona el trabajo con el Objetivo de Desarrollo Sostenible 9, al evidenciar cómo la innovación tecnológica puede contribuir al fortalecimiento de procesos y capacidades en la industria del software.

7.1 Resumen de hallazgos

El desarrollo de este Trabajo Fin de Grado ha permitido analizar de forma práctica y crítica el impacto del uso de inteligencias artificiales generativas, y en particular de GitHub Copilot y ChatGPT, en el desarrollo de sistemas software. El alcance del trabajo se ha centrado en estudiar cómo estas herramientas influyen en el proceso de desarrollo, en la toma de decisiones técnicas y en la adopción de buenas y malas prácticas a lo largo de un ciclo de vida completo, desde la toma de requisitos hasta la implementación, depuración y pruebas.

A partir de un desarrollo iterativo apoyado por la IA, se ha observado que estas herramientas resultan especialmente eficaces como apoyo en tareas concretas y acotadas, tales como la generación de estructuras iniciales de proyecto, la creación de código repetitivo, la consulta rápida de configuraciones técnicas o la resolución puntual de errores. Asimismo, se ha comprobado que la formulación adecuada de prompts y el trabajo por iteraciones cortas y controladas incrementan notablemente la utilidad de la IA y reducen la aparición de errores difíciles de rastrear.

Sin embargo, el trabajo también pone de manifiesto que el uso indiscriminado de la IA, especialmente en intentos de generación completa del sistema en una sola iteración, conduce a soluciones frágiles, con acumulación de errores, pérdida de comprensión del sistema por parte del desarrollador y una fuerte dependencia de la herramienta. En este sentido, el valor real de la inteligencia artificial no reside en sustituir al desarrollador, sino en complementar su trabajo dentro de un proceso iterativo bien definido, donde el control, la validación y la toma de decisiones sigan recayendo en la persona.

7.1.1 Recomendaciones prácticas para futuros desarrolladores

A partir de las experiencias recogidas en el desarrollo del proyecto, se pueden extraer una serie de recomendaciones prácticas para futuros desarrolladores que deseen integrar herramientas de IA generativa en su flujo de trabajo:

En primer lugar, resulta fundamental evitar la generación masiva de código en una única interacción. La experiencia demuestra que dividir el desarrollo en iteraciones pequeñas y funcionales facilita la comprensión del sistema, reduce la complejidad de la depuración y permite aprovechar mejor las capacidades de la IA.

En segundo lugar, la IA debe utilizarse como apoyo y no como fuente única de decisiones técnicas. El desarrollador debe mantener una visión global del sistema, revisar críticamente el código generado y adaptar las soluciones propuestas al contexto concreto del proyecto.

Asimismo, la calidad de los resultados obtenidos depende en gran medida de la calidad de los prompts. Proporcionar contexto suficiente, describir claramente los objetivos de cada iteración y limitar el alcance de las peticiones mejora significativamente la utilidad de las respuestas generadas.

Por último, se recomienda combinar el uso de la IA con metodologías de desarrollo bien conocidas, como el enfoque iterativo e incremental empleado en este trabajo, que permite integrar las sugerencias de la IA sin comprometer la estabilidad ni la coherencia del sistema.

7.1.2 Limitaciones del uso de IAs

El uso de inteligencias artificiales generativas en el desarrollo de software presenta una serie de limitaciones que han quedado claramente reflejadas en este trabajo. En primer lugar, estas herramientas no garantizan la corrección ni la calidad del código generado, siendo habitual la aparición de errores sintácticos, lógicos o de diseño que requieren una revisión cuidadosa por parte del desarrollador.

Además, el rápido ritmo de evolución de las tecnologías de IA implica que algunas de las conclusiones obtenidas pueden perder vigencia con el tiempo. Las capacidades, limitaciones y comportamientos de estas herramientas cambian con frecuencia, lo que dificulta la generalización absoluta de los resultados.

Por otro lado, los beneficios del uso de la IA dependen en gran medida del contexto de uso, del tipo de proyecto y del nivel de experiencia del desarrollador. Tal y como se desprende del trabajo, los desarrolladores con mayor experiencia son capaces de extraer más valor de estas herramientas, mientras que un uso poco crítico puede generar dependencia y una comprensión superficial del sistema.

Finalmente, al tratarse de un análisis en gran parte cualitativo, existen limitaciones a la hora de medir de forma completamente objetiva aspectos como la productividad o la calidad del software, ya que estos factores están influidos por variables externas difíciles de aislar y controlar.

7.2 Riesgos y desafíos

El uso de IA generativa en el desarrollo de software conlleva una serie de riesgos y desafíos que deben ser considerados de forma explícita. Uno de los principales riesgos identificados es la pérdida de control sobre el sistema cuando el desarrollador delega en exceso en la IA, lo que puede derivar en arquitecturas poco coherentes, código difícil de mantener y una elevada dependencia de la herramienta para realizar cualquier modificación.

Otro desafío relevante es la falsa sensación de productividad. Aunque la IA puede acelerar la generación de código, este ahorro de tiempo inicial puede verse compensado negativamente por un mayor esfuerzo posterior en depuración, comprensión y corrección de errores si no se ha mantenido un control adecuado desde el inicio.

Asimismo, existe el riesgo de adopción acrítica de malas prácticas, ya que la IA puede generar soluciones que funcionan en apariencia pero no cumplen con estándares de calidad, seguridad o mantenibilidad exigibles en entornos

profesionales. Este aspecto resulta especialmente crítico en contextos formativos, donde el uso inadecuado de estas herramientas puede afectar negativamente al proceso de aprendizaje.

7.3 Equilibrio entre automatización y control humano

Uno de los principales aprendizajes derivados de este trabajo es la necesidad de encontrar un equilibrio adecuado entre la automatización que ofrecen las inteligencias artificiales y el control humano del proceso de desarrollo. La IA demuestra ser una herramienta muy valiosa cuando se utiliza para automatizar tareas repetitivas, generar borradores de código o apoyar la resolución de problemas concretos, pero no debe sustituir la capacidad de análisis, diseño y toma de decisiones del desarrollador.

El desarrollo iterativo seguido en este proyecto ha demostrado ser un marco adecuado para alcanzar este equilibrio, permitiendo integrar la IA como un asistente que acelera el trabajo sin perder el control del sistema. En este contexto, el desarrollador mantiene la responsabilidad sobre la arquitectura, el diseño del dominio y la validación final del código, mientras que la IA actúa como un apoyo que reduce la carga cognitiva y agiliza determinadas tareas.

En conjunto, este trabajo refuerza la idea de que el valor real de la inteligencia artificial en el desarrollo de software no reside en la automatización completa, sino en su uso consciente y crítico como complemento al conocimiento y la experiencia humana, dentro de procesos bien estructurados y controlados.

7.4 Sostenibilidad, objetivos del milenio: industria, innovación e infraestructura

El presente Trabajo Fin de Grado guarda una relación directa con el Objetivo de Desarrollo Sostenible 9 (Industria, innovación e infraestructura), al abordar el uso de tecnologías innovadoras en el ámbito del desarrollo de sistemas software y analizar su impacto real sobre los procesos de producción tecnológica.

En particular, este trabajo se centra en el estudio y aplicación de inteligencias artificiales generativas como herramientas de apoyo al desarrollo, evaluando su capacidad para mejorar la eficiencia, la calidad y la sostenibilidad de los procesos de ingeniería del software. A lo largo del proyecto se ha demostrado que el uso adecuado de estas tecnologías puede contribuir a la modernización de los procesos de desarrollo, alineándose con las metas del ODS 9 orientadas a fomentar la innovación y el fortalecimiento de la capacidad tecnológica.

El desarrollo iterativo de una API funcional para la gestión de una biblioteca, apoyado por herramientas de IA, constituye un ejemplo práctico de cómo la innovación tecnológica puede integrarse en la construcción de infraestructuras digitales robustas, modulares y escalables. El sistema resultante incorpora buenas prácticas de arquitectura, control de acceso, gestión de bases de datos y pruebas automatizadas, aspectos clave para garantizar la sostenibilidad y mantenibilidad de infraestructuras software a medio y largo plazo.

Asimismo, el enfoque crítico adoptado en el uso de la IA refuerza la contribución al ODS 9, ya que no se limita a la adopción acrítica de nuevas tecnologías, sino que analiza sus limitaciones, riesgos y desafíos, promoviendo un uso responsable y eficiente de la innovación. Este planteamiento es especialmente relevante para evitar dependencias tecnológicas innecesarias y asegurar que la automatización contribuya realmente a mejorar los procesos industriales, en lugar de degradar la calidad del producto o el conocimiento técnico del desarrollador.

Por último, el trabajo pone de manifiesto que la innovación en el desarrollo de software no reside únicamente en la incorporación de herramientas avanzadas, sino en su correcta integración dentro de metodologías de trabajo bien definidas. El equilibrio alcanzado entre automatización y control humano permite maximizar los beneficios de la IA, manteniendo la fiabilidad y calidad de la infraestructura desarrollada, lo que refuerza la alineación del proyecto con los principios del ODS 9 en materia de industria, innovación e infraestructuras sostenibles.

8 Bibliografía

- [1] S. M. Hosseini and G. Lichtinger, *Generative AI as Seniority-Biased Technological Change: Evidence from U.S. Résumé and Job Posting Data*, SSRN working paper, Nov. 2025.
- [2] E. Brynjolfsson, B. Chandar, and R. Chen, *Canaries in the Coal Mine? Six Facts About the Recent Employment Effects of Artificial Intelligence*, Stanford Digital Economy Lab working paper, Aug. 2025.
- [3] K. Kellogg, et al., *Don't Expect Juniors to Teach Senior Professionals to Use Generative AI*, Working Paper 24-074, 2024.
- [4] Microsoft, "Use Copilot in Microsoft Teams meetings." Microsoft Support, 2025. [Online]. Available: <https://support.microsoft.com/en-us/office/use-copilot-in-microsoft-teams-meetings-0bf9dd3c-96f7-44e2-8bb8-790bedf066b1>
- [5] OpenAI, "GPT-5 is here," 2025. [Online]. Available: <https://openai.com/gpt-5/>
- [6] S. Piñeiro (Tiangolo), "Tutorial – User Guide – FastAPI," FastAPI documentation, 2024. [Online]. Available: <https://fastapi.tiangolo.com/tutorial/>
- [7] pytest-dev, "Get Started – pytest documentation," 2024. [Online]. Available: <https://docs.pytest.org/en/stable/getting-started.html>

9 Anexos

ANEXO [A]

El repositorio se encuentra dividido en dos grandes bloques principales. Por un lado, la carpeta Docs incluye los archivos correspondientes a cada una de las fases que se han llevado a cabo durante el desarrollo del proyecto, tales como la definición de requisitos, el diseño de la arquitectura, la implementación, la depuración y las pruebas, constituyendo así el registro completo del proceso seguido a lo largo del Trabajo Fin de Grado. Por otro lado, la carpeta src contiene toda la implementación del sistema software, organizada siguiendo una estructura modular que separa claramente los distintos componentes de la API. En ella se incluye la lógica del servidor, los modelos de datos, los controladores, los servicios, la configuración de la base de datos, las utilidades auxiliares y las pruebas automatizadas, lo que facilita la mantenibilidad del código y refleja una arquitectura coherente y alineada con buenas prácticas de desarrollo.

Docs

TFG/

- |— Notas/
 - | |— Conversaciones/
 - | |— Entrevistas/
 - | |— Briefs/
 - | |— Casos de Usos/
 - | |— Requisitos funcionales/
 - | |— Wireframes/
 - | | |— UseCase Sencillo/
 - | | |— UseCase Completo/
- |— Grabación/
 - | |— Conversaciones/
 - | |— Grabaciones/
 - | |— Briefs/
 - | |— Casos de Usos/
 - | |— Requisitos funcionales/
 - | |— Wireframes/
 - | | |— Grabacion 1 Usecase sencillo/
 - | | |— Use cases completos/
 - | | |— Wireframe Modificaciones/
- |— Arquitectura/
- |— Implementación/
- |— Tests/
- |— Memorias/
- |— Memoria.docx

- |— Memoria seguimiento.docx
- └─ Diagrama.docx

Src

biblioteca/

- └─ src/

- |— app/

- | |— main.py

- | |— core/

- | |— models/

- | |— schemas/

- | |— routers/

- | |— services/

- | |— utils/

- |— tests/

- | |— conftest.py

- | |— test_auth.py

- | |— test_books.py

- | |— test_loans.py

- | |— test_reservations.py

- | |— test_sanctions.py

- | |— test_users.py

- └─ pytest.ini

ANEXO [B]

En el siguiente anexo se encuentra una especificación de requisitos en la que nos basamos para hacer las reuniones de cliente desarrollador.

Requirements Specification

**ACME University Library Information
System**

(ACMELIS)

TABLE OF CONTENTS

1	<i>Introduction</i>	3
1.1	Purpose	3
1.2	Scope of System	3
1.3	Definitions, Acronyms and Abbreviations	4
1.4	References	4
1.5	Overview	4
2	<i>General Description</i>	5
2.1	Product Perspective	5
2.2	Product Functions	5
2.2.1	Reservation of bibliographical resources	5
2.2.2	Loan of bibliographical resources	6
2.2.3	Return of bibliographical resources.....	6
2.3	User Characteristics	7
2.4	Constraints	7
2.5	Assumptions and Dependencies	7
3	<i>Specific Requirements</i>	8
3.1	Functional Requirements	8
3.1.1	Reservation of bibliographical resources	8
3.1.2	Loan of bibliographical resources	9
3.1.3	Return of bibliographical resources.....	10
3.2	Performance Requirements	11
3.3	Technological Requirements	11
3.4	Attributes	11
3.4.1	Security	11
3.4.2	Audit.....	11
3.4.3	Initial data load	12
4	<i>Appendix A</i>	13

1 Introduction

This document is a software requirements specification (SRS) for the ACME University Library Information System. This SRS was validated by an independent team of testers and is considered to be error free with the usual reservations in this respect in the context of software development.

This SRS conforms to the guidelines established in the IEEE Recommended Practice for Software Requirements Specification ANSI/IEEE Standard 830-1998.

1.1 Purpose

The target audience of this SRS is the software development team, the ACME University Library management and the system end users. As this SRS has already been validated, the main stakeholders shall be the system developers especially with a view to the immediate commencement of design activities.

1.2 Scope of System

The ACME University Library now operates fully manually, without any type of automated support whatsoever. This is an enormous obstacle to the everyday operation of the library, as the increasing number of students, faculty and bibliographical resources at the university every year renders manual data processing extremely inefficient.

The main intent of the ACME University Library Information System, hereafter ACMELIS, is to automate certain library tasks, specifically:

- Reservation of bibliographical resources
- Loan of bibliographical resources
- Return of bibliographical resources

ACMELIS shall not, on the other hand, automate any of the following tasks:

- Library user data entry/deletion/modification
- Bibliographical resources entry/deletion/modification
- Bibliographical resources acquisition/withdrawal

The above processes shall be the object of a later development project, which is scheduled to kick off when ACMELIS has been implemented and has proved to be useful in routine library use.

1.3 Definitions, Acronyms and Abbreviations

ACMELIS	ACME University Library Information System
Suspension	Period of time during which a library user cannot borrow books.
Library users	People entitled to use the library services. They are students, faculty, administrative and support staff and library staff.
SMTP	Simple Mail Transfer Protocol

1.4 References

IEEE Recommended Practice for Software Requirements Specification. ANSI/IEEE Std. 830- 1998.

1.5 Overview

This document is structured in conformity with the IEEE Recommended Practice for Software Requirements Specification ANSI/IEEE Standard 830-1998. Sections that are not considered applicable have been omitted.

Section 3 is the only part of the SRS to deviate from the standard. It describes requirements in list format, which is a simplification of the templates recommended by the standard for that section.

Section 3 lists some requirements that cannot be classed as such, as they do not describe services that the software should provide, but procedural aspects that should be overlooked. These requirements are Req(23) and Req(32). These requirements are added to improve the clarity of the explanation.

2 General Description

This section gives a high-level description of ACMELIS, focusing on the functions that it is to perform, as well as its general constraints.

2.1 Product Perspective

ACMELIS shall interact with four software products. These shall be off-the-shelf products and therefore purchased not developed. These products are as follows:

- Database. The use of a relational database shall be the only demand. Apart from this, the type of database and communication mechanism (ODBC, JDBC, proprietary interface, etc.) shall be selected during the Architecture definition.
- ACME's Single Sign-On system. This system will also provide the user information indicated in Appendix A.
- SMTP software. The decision on which SMTP software is to be used shall be made during the Architecture definition.

2.2 Product Functions

The aim of ACMELIS shall be to automate the following ACME University Library tasks:

- Reservation of bibliographical resources
- Loan of bibliographical resources
- Return of bibliographical resources

These tasks and how they shall be supported by the software system are described in more detail in the following.

2.2.1 Reservation of bibliographical resources

As the library's bibliographical resources are a limited resource, not all users shall be able access a book when they need to. Therefore, ACMELIS shall have to implement a mechanism whereby users can reserve a book that they shall later take out on loan.

Any user shall be entitled to make any number of reservations. The only existing constraint is that reference works shall not be able to be reserved, as such books are not available for borrowing either.

The reservations shall be valid for a limited period. The aim of this constraint is to prevent users from forgetting to collect the bibliographical resources that they have reserved. Users shall be notified that they should collect a reserved book when the book becomes available. If they fail to do so within two days, the reservation shall be cancelled.

With the aim of maximizing functionality, however, users shall be able to cancel or extend the term of their reservations, if they so wish.

2.2.2 Loan of bibliographical resources

Library users shall be able to borrow books whether or not they have made a reservation. Users shall be entitled to borrow any books except reference works. Additionally, only the person or persons that reserved a reserved book shall be able to borrow that book.

The term of the loan shall depend on the user type and the period of the academic year in which the loan is made.

The process shall differ depending on the user type. Students and administrative and support staff shall have to go to the library to request a loan. Faculty shall be able to use ACMELIS to request a loan anywhere, and the requested books shall be sent to their office by internal mail.

2.2.3 Return of bibliographical resources

The return of bibliographical resources shall depend on the type of user that borrowed the book. Students and administrative and support staff shall have to visit the library to return the book. Faculty shall be able to send the library books to the library via the internal mail system. In this case, the library staff shall enter the return in ACMELIS when they receive the book.

Failure to return the book within the loan period shall entail a two-day suspension for every day of delay. To prevent pointless suspensions, ACMELIS shall notify users when their books are due, as well as when they are overdue.

2.3 User Characteristics

There are three types of users: faculty, students and administrative and support staff. User familiarity with software systems may differ depending on their education and development. On this ground, ACMELIS shall have to offer an intuitive, easy-to-learn and easy-to-use user interface. On-line help should be provided for all system functions.

2.4 Constraints

ACMELIS shall have to provide a WEB interface only. This shall facilitate its use on multiple platforms at a lower cost.

ACMELIS shall have to be available at all times, except in the case of communications network crashes.

Users shall be able to use ACMELIS from both the ACME University Intranet and from the Internet. On this ground, a robust system user authentication protocol shall have to be defined where the only constraint is that the protocol should be easily implementable and usable.

2.5 Assumptions and Dependencies

ACMELIS shall be considered to be independent of any other software system.

3 Specific Requirements

This section presents the functional requirements that the system shall have to satisfy. All the requirements stated here are ESSENTIAL, that is, a system that does not satisfy any of the requirements presented here would not be acceptable.

The key data on users and bibliographical resources, which are necessary for a proper understanding of the functional requirements, are set out in Appendix A.

3.1 Functional Requirements

3.1.1 Reservation of bibliographical resources

Req(01) Users shall be able to reserve books, irrespective of whether or not the requested book is on loan.

Req(02) Users shall not be able to reserve reference works.

Req(03) A user shall be able to reserve any number of books. Similarly, multiple users shall be able to reserve any particular book.

Req(04) The reservation shall be valid for any copy of a particular book. For example, if there are five copies of *Don Quixote*, any one of these copies shall be considered to be reserved.

Req(05) Reservations are organized in chronological order, where the longest-standing shall take the highest and the newest the lowest priority.

Req(06) Users shall be notified by e-mail when a reserved book becomes available. Only the user with the oldest reservation shall be notified, save as stated in Req(12).

Req(07) A reservation shall be valid for two days as of the date when the reserved book becomes available. After this period has expired, the reservation shall be cancelled.

Req(08) Only the reservations of users that have been notified of the availability of a particular book shall be cancelled. Once a reservation has been cancelled, the next user shall be notified, as specified in Req(06) and Req(07).

Req(09) Users shall be able to consult their reservations.

Req(10) Users shall be able to cancel their reservations.

Req(11) Users shall be able to extend the validity period of their reservations. If the validity period of a reservation is extended, this reservation becomes the newest of all the reservations.

Req(12) If several copies of a reserved book are available, the same number of users shall be notified at the same time. For example, if eight users have requested *Don Quixote* and three copies are available, three users shall be notified of availability at one go.

3.1.2 Loan of bibliographical resources

Req(13) Library users shall be able to borrow books.

Req(14) The loan period shall be 7 days for students and administrative and support staff and 15 days for faculty.

Req(15) If the loan period expires on a non-working day, the loan shall be considered to be due on the next working day.

Req(16) Users shall not be able to borrow reference works.

Req(17) Suspended users shall not be able to borrow books.

Req(18) In order to borrow a book, the user shall have to be the holder of the longest-standing reservation for the book in question. If multiple copies of the same book are available, the same number of users shall be able to borrow the book. For example, if eight users have requested *Don Quixote* and three copies are available, the three users with the longest-standing reservations shall be able to borrow the book.

Req(19) If a book has not been reserved, any user shall be able to borrow it.

Req(20) Students and administrative and support staff shall have to place a request for a loan at the library, and the librarian shall have to enter the data in ACMELIS.

Req(21) Faculty shall be able to use ACMELIS to borrow books without having to visit the library in person. To do this, and respecting the constraints set out in Req(16), Req(17), Req(18) and Req(19), they shall enter the loan data in ACMELIS.

Req(22) Books borrowed by a faculty member according to the specifications in Req(21) shall be sent to the faculty member's office by internal mail. For this purpose, the following shall be generated on a daily basis:

- A list of book loans requested by each faculty member;
- A list of labels with the name of the faculty member and his or her office.

Req(23) The library staff shall use the list of book loans to search for the books that have to be sent to the faculty. The list of labels shall be used to send the books to faculty members by internal mail.

Req(24) Users shall be able to consult their loans, as well as the return date.

3.1.3 Return of bibliographical resources

Req(25) Library users shall have to return the books before the loan period expires.

Req(26) If a book is returned late, the system shall suspend the user for two days for every day the book is overdue.

Req(27) Library staff shall be able to access and cancel user suspensions.

Req(28) ACMELIS shall send an e-mail advising users of the expiry date of a book's loan period. The notification shall be issued two days before the book's return date.

Req(29) If a book is returned late, ACMELIS shall e-mail the user every two days advising that the book is overdue, serving up to three notices.

Req(30) ACMELIS shall generate a list of users with overdue books. Users that have books that are 8 or more days overdue shall be listed.

Req(31) Students and administrative and support staff shall have to return a loan to the library, and the librarian shall have to enter the data in ACMELIS.

Req(32) Faculty shall be able to return books as specified in Req(31) or shall be able to send the books by internal mail to the library, upon receipt of which the library staff shall enter the book as returned.

3.2 Performance Requirements

ACMELIS shall serve 20 users simultaneously with a response time of less than 1 seconds. If the number of users using ACMELIS is greater than 20, the service time shall necessarily decay gradually up to a maximum of 5 seconds for 100 users. As of 100 users there shall be a denial of service.

The service time shall be met both on the Intranet and on the Internet, save in the event of network congestion.

3.3 Technological Requirements

The ACMELIS implementation platform shall be defined during the detailed design activity in conformity with the stipulations in Section 2.1.

3.4 Attributes

3.4.1 Security

Only library staff shall be able to perform the loan and return operations and lift suspensions, save as indicated in Req(21).

3.4.2 Audit

All reservations, loans and returns shall be recorded for possible statistical use.

3.4.3 Initial data load

The ACME University' computer centre is responsible for the initial bibliographical resources' data load.

4 Appendix A

The essential user data and bibliographical resources data are as follows

Users:




- Given name
- Family name
- E-mail address
- Type: faculty, student, administrative and support staff, library staff
- Postal address
- Office for faculty or administrative and support staff

Bibliographical resources:

- Book code (identical for all copies of the same book)
- Unique code for each copy of the book
- Title
- Authors
- Edition
- Publisher
- Year
- Subject matter
- Type: reference or open access
- Materials included (e.g. diskettes, CD-ROM, offprints, etc.)

DIEGO OJEDA HIGUERA

Memoria.pdf

-  Turnitin Memoria Final
-  TFG ETSIINF (Moodle PP)
-  Universidad Politecnica de Madrid

Document Details

Submission ID

trn:oid::1:3457683621

Submission Date

Jan 14, 2026, 11:32 PM GMT+1

Download Date

Jan 14, 2026, 11:35 PM GMT+1

File Name

Memoria.pdf

File Size

2.0 MB

60 Pages

18,841 Words

109,819 Characters




6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.


Filtered from the Report

- ▶ Bibliography
 - ▶ Quoted Text
-

Top Sources

- 6%  Internet sources
 - 1%  Publications
 - 0%  Submitted works (Student Papers)
-

Este documento esta firmado por

	Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jan 14 23:41:21 CET 2026
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)