

15-6-2025

PROYECTO DE FIN DE CICLO

APLICACIÓN DE GESTIÓN DE
LECTURAS PERSONALES



Xoel Rivas Pérez
CIFP A CARBALLEIRA - MARCOS VALCÁRCEL
CICLO SUPERIOR DE DESARROLLO DE APLICACIONES
MULTIPLATAFORMA

Contenido

INTRODUCCIÓN	3
OBJETIVOS DEL PROYECTO	3
Objetivo principal	3
Objetivos técnicos.....	3
Objetivos académicos.....	3
ANÁLISIS Y ESPECIFICACIÓN DE REQUISITOS.....	3
Requisitos funcionales	3
Requisitos no funcionales	4
Casos de uso.....	5
GESTIÓN DE LA INFORMACIÓN Y DATOS	8
PLANIFICACIÓN.....	11
DESARROLLO.....	11
Estructura del proyecto	11
Pruebas	12
Pruebas de interfaz	12
Pruebas funcionales.....	13
Pruebas de validación	13
Gestión de errores	13
Control de versiones con Git.....	14
Herramientas y librerías utilizadas.....	14
ESTUDIO DE MERCADO	15
Introducción del estudio de mercado	15
Análisis de la competencia	15
Oportunidades de mejora	16
Conclusión del estudio de mercado.....	16
MANUAL DE INSTALACIÓN	17
MANUAL DE USUARIO	19
MEJORAS FUTURAS	26
CONCLUSIÓN FINAL	27
ANEXO.....	28
Presupuesto.....	28



Webgrafía	29
-----------------	----



INTRODUCCIÓN

Este proyecto consiste en el desarrollo de una aplicación de escritorio, diseñada para amantes de la lectura que deseen llevar un registro organizado de sus libros leídos. La herramienta permitirá no solo almacenar información básica (título, autor, género), sino también añadir valoraciones y reseñas personales, facilitando un seguimiento enriquecedor de sus hábitos de lectura.

El proyecto surge como requisito para aprobar el Ciclo Superior de Desarrollo de Aplicaciones Multiplataforma, por lo que probará diversos conocimientos de programación, gestión de bases de datos, diseño de interfaces gráficas y demás habilidades relacionadas con el desarrollo de aplicaciones.

OBJETIVOS DEL PROYECTO

Objetivo principal

Desarrollar una aplicación funcional e intuitiva que permita:

- Registrar los libros leídos por el usuario.
- Autocompletar determinados campos (vía API) como título, autor, género, año de publicación, etc.
- Rellenar determinados campos con información personal como la reseña o la puntuación del libro.
- Visualizar el historial de lectura con filtros avanzados (por año, género, autor, puntuación...)

Objetivos técnicos

- Implementar una base de datos SQLite para almacenar libros.
- Garantizar la integridad de los datos con relaciones y consultas SQL optimizadas.
- Crear una interfaz gráfica vistosa con CustomTkinter.
- Conectar con una API para autocompletar datos de los libros.

Objetivos académicos

Demostrar la competencia en el desarrollo de aplicaciones y aplicar metodologías de planificación y documentación propias del ciclo.

ANÁLISIS Y ESPECIFICACIÓN DE REQUISITOS

Requisitos funcionales

- Gestión de libros:
 - **RF-01:** Búsqueda de libros en una API externa.



El sistema permitirá buscar libros por título/autor en Open Library y mostrará resultados en un formato de lista.

- **RF-02:** CRUD de libros en la biblioteca personal.
Los libros se añaden desde la API o manualmente (con campos: título, autor, género, etc.). También se pueden editar o eliminar libros existentes.
- **RF-03:** Gestión de estados de lectura.
Marcado de libros con estados: “Leído”, “Leyendo”, “Pendiente”, “Abandonado”.
- **RF-04:** Visualización de detalles.
Mostrar información extendida: sinopsis, editorial, año de publicación, etc.

➤ Interfaz de Usuario:

- **RF-05:** Pantalla principal.
Aparece el listado de los libros en formato tarjeta (ordenados alfabéticamente). Cada tarjeta tendrá un indicador visual que mostrará el estado de la lectura del libro:
Verde → Leído
Naranja → Leyendo
Morado → Pendiente
Rojo → Abandonado
- **RF-06:** Formularios interactivos.
Autocompletado vía API.

➤ Base de datos:

- **RF-07:** Persistencia de datos en SQLite.
Almacenamiento local de la biblioteca personal.

Requisitos no funcionales

➤ Usabilidad:

- **RNF-01:** Interfaz gráfica sencilla y fácil de navegar.
Menú de navegación intuitivo con diseño responsive.

➤ Eficiencia:

- **RNF-02:** La base de datos debe manejar cientos de libros sin problemas.

➤ Seguridad:

- **RNF-03:** Integridad de los datos.
Validación de campos para evitar errores y restricción de unicidad en campos críticos.



Casos de uso

CASO DE USO 1: INICIAR APLICACIÓN	
Campo	Descripción
Actor	Usuario.
Descripción	Al arrancar la aplicación, se crea la base de datos si no existe, y se muestra la ventana principal con el listado de libros guardados (en el caso de haberlos).
Precondición	Ninguna.
Flujo principal	Ejecución script principal → Se crea o se abre la base de datos → Se instancian los componentes de la ventana principal → Se consultan las diferentes tablas de la base de datos para obtener el listado → Se muestran los libros (de haberlos) o un mensaje "No hay libros guardados" (de no haberlos)
Postcondición	Se muestra la ventana principal.

Tabla 1: Caso de uso 1: Iniciar aplicación.

CASO DE USO 2: VER LISTADO DE LIBROS	
Campo	Descripción
Actor	Usuario.
Descripción	Se muestran todos los libros guardados en la base de datos, con su título, autor, año, estado y portada.
Precondición	La ventana principal está iniciada y la base de datos creada.
Flujo principal	Se obtienen los libros guardados en la base de datos → Se itera sobre cada libro → Para cada libro se determina el color de fondo según su estado y se crea un frame con el título, el autor, el año y la portada → Se muestran todos los frames
Flujos alternativos	Si no hay libros se muestra: "No hay libros guardados". Si falla la carga de portada, se usa una imagen predeterminada "Sin portada".
Postcondición	Se muestra la lista en pantalla y el usuario puede interactuar.

Tabla 2: Caso de uso 2: Ver listado de libros.



CASO DE USO 3: BUSCAR LIBROS GUARDADOS	
Campo	Descripción
Actor	Usuario.
Descripción	Se realiza una búsqueda de los libros guardados por título o autor.
Precondición	Ventana principal abierta y listado inicial cargado.
Flujo principal	El usuario introduce texto en el campo de búsqueda y pulsa el botón de buscar → La app recoje el texto → Se ejecuta sentencia SQL para hacer búsqueda por título o autor → Si hay resultados se muestran los libros filtrados, si no se muestra: "No se encontraron libros que coincidan" → Si el buscador está vacío y se pulsa el botón de búsqueda se recarga el listado completo
Flujos alternativos	Error en la consulta: mostrar messagebox de error.
Postcondición	Vista refrescada con los libros filtrados o con todos si la búsqueda está vacía.

Tabla 3: Caso de uso 3: Buscar libros guardados.

CASO DE USO 4: AÑADIR UN LIBRO	
Campo	Descripción
Actor	Usuario.
Descripción	El usuario busca un libro por título, autor o ISBN, se muestran los resultados y se guarda el libro seleccionado en la base de datos.
Precondición	El usuario accede a la opción de "Añadir libro" desde la interfaz principal.
Flujo principal	El usuario pulsa el botón "+" → Se abre una ventana de elección (Crear o Añadir) → Selecciona "Añadir libro" → Se abre la ventana de añadir libro → El usuario escribe el texto de búsqueda y pulsa buscar → Se hace una llamada a la API de OpenLibrary y se extraen los datos de búsqueda correspondientes → Se muestra cada resultado en una tarjeta con título, autor, año y portada (si los datos existen) → El usuario selecciona un libro cuyos datos se insertan en la BD → Se muestra: "Libro guardado correctamente" → Se recarga el listado de la ventana principal → El usuario cierra la ventana de búsqueda o sigue buscando
Flujos alternativos	Si el libro ya existe (ISBN duplicado) se ignora. El usuario puede cancelar sin seleccionar un libro.
Postcondición	El libro queda registrado en la biblioteca del usuario. A nivel interno, si el autor, género o editorial no existiera previamente en la base de datos, se crean automáticamente para mantener las relaciones correctas.

Tabla 4: Caso de uso 4: Añadir un libro.



CASO DE USO 5: CREAR UN LIBRO	
Campo	Descripción
Nombre	Crear un libro.
Actor	Usuario.
Descripción	Permite al usuario crear un libro manualmente (todos los campos editables) cuando la búsqueda no arroja los resultados deseados.
Precondición	Ventana principal abierta.
Flujo principal	El usuario pulsa el botón "+" y elige "Crear libro" en la ventana de elección → Se abre la ventana de editar libro con todos los campos del formulario vacíos → El usuario rellena los campos que crea convenientes y puede insertar una imagen en la portada → El usuario guarda los cambios → Se muestra: "Libro creado correctamente" → Se recarga el listado principal
Flujos alternativos	Error al insertar: se muestra el error. El usuario puede pulsar "Cancelar": la ventana se cierra sin guardar nada.
Postcondición	Si guardó, aparece en el listado principal.

Tabla 5: Caso de uso 5: Crear un libro.

CASO DE USO 6: EDITAR UN LIBRO	
Campo	Descripción
Actor	Usuario.
Descripción	El usuario accede a los detalles de un libro previamente añadido y modifica uno o varios campos: fechas de lectura, reseña personal, calificación, estado, etc.
Precondición	El libro ya debe estar registrado en la biblioteca del usuario.
Flujo principal	El usuario selecciona un libro del listado principal → Se abre la ventana de edición → Se cargan los datos del libro → El usuario modifica uno o varios campos → El usuario guarda los cambios → Se muestra: "Libro actualizado correctamente" → Se cierra la ventana y se recarga el listado principal
Flujos alternativos	Error en la BD: se muestra el error. El usuario pulsa "Eliminar libro" → Se solicita confirmación → Se eliminan datos y relaciones → Se muestra: "Libro eliminado" → Se cierra la ventana y se recarga el listado principal. El usuario pulsa "Cancelar": descarta cambios y cierra la ventana.
Postcondición	Los cambios realizados por el usuario se guardan correctamente en la base de datos, actualizando el registro del libro.

Tabla 6: Caso de uso 6: Editar un libro.



CASO DE USO 7: ELIMINAR UN LIBRO	
Campo	Descripción
Actor	Usuario.
Descripción	El usuario selecciona un libro de su biblioteca personal y lo elimina permanentemente.
Precondición	El libro ya debe estar registrado en la biblioteca del usuario.
Flujo principal	El usuario selecciona un libro del listado principal → Se abre la ventana de edición → Se cargan los datos del libro → El usuario pulsa "Eliminar libro" → Aparece diálogo de confirmación → Si confirma se eliminan los datos y relaciones del libro → Se muestra: "Libro eliminado correctamente" → Se cierra la ventana y se recarga el listado principal → Si el usuario cancela, se cierra el diálogo y no hace nada
Flujos alternativos	Error durante la eliminación: mostrar error.
Postcondición	El libro se elimina de la base de datos. Las relaciones con autores, géneros o editorial también se eliminan si ya no están vinculadas a otros libros.

Tabla 7: Caso de uso 7: Eliminar un libro.

GESTIÓN DE LA INFORMACIÓN Y DATOS

La base de datos de la app estará formada por cuatro tablas: libros, autores, géneros y editoriales. Las tablas “autores”, “géneros” y “editoriales” están relacionadas con la tabla principal “libros”, de esta manera se evita la repetición de datos.

A continuación, se muestra el diagrama entidad-relación resultante:



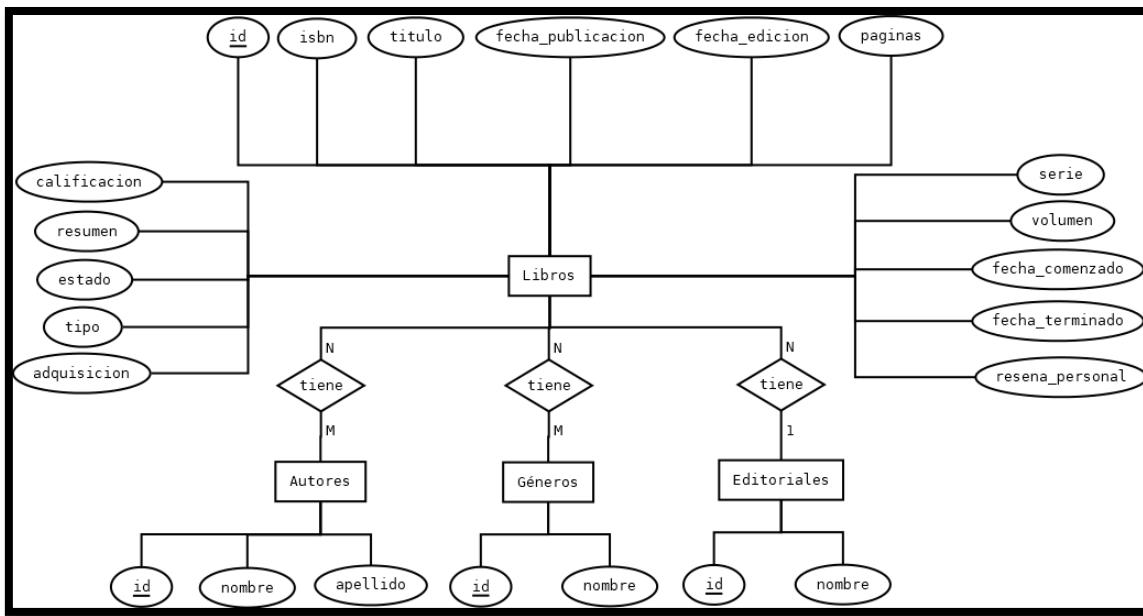


Ilustración 1: Diagrama entidad-relación.

Y a partir de este diagrama se crearon las siguientes tablas, que muestran las restricciones de cada campo y el resultado de las relaciones entre entidades:

LIBROS																
id	título	fecha_publicacion	fecha_edicion	paginas	isbn	serie	volumen	fecha_comenzado	fecha_terminado	estado	resumen	reseña_personal	calificacion	tipo	adquisicion	id_editorial
PK NN					UNIQUE											FK_Editoriales

Tabla 8: Tabla "Libros".

- ❖ Contiene los datos principales de cada libro registrado.

AUTORES		
id	nombre	apellido
PK NN	NN	
NN	UNIQUE	

Tabla 9: Tabla "Autores".

- ❖ Almacena los nombres y apellidos de los autores.

GÉNEROS	
id	nombre
PK NN	NN UNIQUE

Tabla 10: Tabla "Géneros".

- ❖ Incluye los distintos géneros literarios disponibles.



EDITORIALES	
id	nombre
PK	NN
NN	UNIQUE

Tabla 11: Tabla "Editoriales".

- ❖ Lista las editoriales asociadas a los libros.

LIBROS_AUTORES	
id_libro	id_autor
	PK
FK_Libros	FK_Autores

Tabla 12: Tabla "Libros-Autores".

- ❖ Representa la relación muchos a muchos entre libros y autores.

LIBROS_GÉNEROS	
id_libro	id_genero
	PK
FK_Libros	FK_Géneros

Tabla 13: Tabla "Libros-Géneros".

- ❖ Representa la relación muchos a muchos entre libros y géneros.

LEYENDA	
PK	Primary Key
FK_NombreTabla	Foreign Key y tabla de referencia
NN	Not Null

Tabla 14: Leyenda.

Como se puede observar, al existir dos relaciones N:M, se crearon dos tablas adicionales para mostrar la relación entre las entidades.

Este diseño relacional permite mantener la integridad de los datos, evitar la redundancia y facilita la escalabilidad del sistema ante futuras ampliaciones.



PLANIFICACIÓN

Fase	Tiempo estimado	Inicio	Fin
Inicio y organización	0,5h	04-abr-25	05-abr-25
Redacción de Introducción y Objetivos	0,5h	05-abr-25	07-abr-25
Análisis y especificación de requisitos	1h	08-abr-25	10-abr-25
Diagrama ER y diseño de la base de datos	2h	15-abr-25	20-abr-25
Creación del script de la base de datos	1h	19-abr-25	20-abr-25
Planificación del proyecto	0,5h	21-abr-25	21-abr-25
Diseño de casos de uso	1h	22-abr-25	24-abr-25
Diseño para la gestión de la información y datos	1h	22-abr-25	25-abr-25
Implementación de la base del sistema	3h	26-abr-25	02-may-25
Implementación del formulario de añadir libros	3h	03-may-25	10-may-25
Integración de relaciones N:M (autores y géneros)	2,5h	11-may-25	18-may-25
Desarrollo de la interfaz gráfica básica (CustomTkinter)	3h	11-may-25	18-may-25
Búsqueda de libros	2h	19-may-25	25-may-25
Implementación de guardado y carga de datos	1,5h	26-may-25	30-may-25
Mejora de la interfaz y validaciones	2h	31-may-25	04-jun-25
Pruebas funcionales y depuración	2h	05-jun-25	08-jun-25
Finalización de funcionalidades (detalles extra)	1h	09-jun-25	10-jun-25
Redacción y finalización de la documentación	2h	11-jun-25	13-jun-25
Entrega final y revisión	0,5h	14-jun-25	15-jun-25
	30h		

Tabla 15: Planificación del proyecto.

DESARROLLO

Estructura del proyecto

La aplicación está organizada en distintos archivos y carpetas que separan las responsabilidades principales del sistema. Esta modularidad facilita el mantenimiento y la comprensión del código. Su estructura es la siguiente:

- **mi_biblio_app/**: carpeta raíz de la app.
 - **ui/**: carpeta en la que se almacenan las interfaces de usuario y su lógica.
 - **ventana_principal.py**: contiene la clase principal que lanza la interfaz principal de la aplicación.
 - **ventana_anadir_libro.py**: define la ventana donde se realiza la búsqueda y selección de libros para añadir a la biblioteca.
 - **ventana_editar_libro.py**: ventana en la que aparecen los diferentes campos editables de cada libro que se ha añadido a la biblioteca.



- **ventana_elegir_modo.py**: ventana que sirve para preguntarle al usuario si desea crear un libro nuevo o añadir uno ya existente.
- **utils/**: carpeta que contiene los útiles de la app.
 - **rutas.py**: contiene funciones utilitarias para gestionar rutas de archivos.
- **api.py**: módulo encargado de gestionar la conexión con la API de OpenLibrary y recuperar los datos de libros.
- **database.py**: define y crea la estructura de la base de datos SQLite, con sus tablas y relaciones.
- **ímágenes/**: carpeta donde se almacenan los iconos utilizados en la interfaz gráfica.
- **portadas/**: carpeta con portadas predeterminadas.
- **dist/**: carpeta en la que se genera el ejecutable de la aplicación y dónde se guardan las portadas que se descargan durante la ejecución de la app.

Pruebas

Durante el desarrollo de la aplicación, se realizaron pruebas funcionales, de validación y de comportamiento, aunque no se siguió una metodología formal de testing automatizado. Las pruebas fueron ejecutadas manualmente durante las distintas etapas del proyecto. A continuación, se resumen los principales tipos de pruebas aplicadas:

Pruebas de interfaz

Se realizaron pruebas manuales para verificar que la interfaz gráfica cumpliera con criterios de usabilidad, consistencia y estética. Dado que se empleó la biblioteca *CustomTkinter*, se evaluaron tanto el diseño visual como la interacción de los elementos. A continuación, se resumen los aspectos probados:

- **Distribución de componentes**: se comprobó que todos los elementos (botones, campos de entrada, etiquetas, etc.) estuvieran correctamente alineados y con un espaciado adecuado en diferentes ventanas.
- **Interacción de elementos**: se verificó que los botones y formularios reaccionaran adecuadamente al clic, mostrando las ventanas correspondientes o ejecutando las acciones esperadas. Destacando también aquellos que permiten interacción ya sea cambiando de color o cambiando la forma del pointer del ratón.
- **Cambios de estado visual**: el sistema de colores para el estado de lectura funcionó correctamente, cambiando la apariencia de la tarjeta del libro según la opción seleccionada, y que el color del sombreado al pasar el botón por encima fuera el correcto.
- **Feedback visual**: cada acción importante (como guardar, eliminar, buscar o añadir un libro) muestra una notificación visual o mensaje confirmando



la operación, para que el usuario no tenga dudas de que la acción se realizó correctamente.

- **Comprobación de portabilidad:** la aplicación fue probada en distintas resoluciones y entornos de escritorio para asegurar que la interfaz no se descuadre ni pierda funcionalidad.

Pruebas funcionales

Se verificó que todas las funcionalidades implementadas operaran correctamente:

- Aparición de resultados al hacer una búsqueda vía API. La búsqueda se puede realizar introduciendo el título, el autor y el ISBN del libro. Estos resultados muestran el título, autor, año de publicación y portada del libro.
- Se añadieron libros y se verificó su aparición en la biblioteca con los datos que proporciona la API.
- Se comprobó que se pueden crear libros sin necesidad de estar conectados a internet.
- El sistema de filtrado funciona correctamente al hacer búsquedas por título o autor.

Pruebas de validación

Se realizaron pruebas para asegurar la integridad de los datos. Hay que tener en cuenta que esta app busca proporcionar flexibilidad al usuario a la hora de completar los datos de los libros que guarda en su biblioteca, esto quiere decir que no hay reglas excesivamente restrictivas para guardar información. Por ejemplo, en el campo “Autor”, nada impide al usuario rellenarlo con números. El que el usuario rellene este campo de manera “incorrecta” no afecta al funcionamiento de la aplicación, por eso es algo que se deja a responsabilidad del usuario en pro de proporcionarle mayor versatilidad para llenar los datos de los libros. Los únicos campos con restricción son las fechas, ya que solo se pueden llenar por medio del calendario emergente.

- Se almacenan, se devuelven y se eliminan correctamente los datos de los libros al guardar, editar o eliminar.
- La app permite el guardado de campos vacíos.
- Se comprobó el correcto formato de las fechas.
- Se comprobó a añadir libros duplicados y se verificó que no se repiten claves primarias o foráneas.

Gestión de errores

La app es capaz de gestionar errores por medio de bloques *try/except* que, en caso de producirse algún error, se informa mediante un mensaje al usuario sin bloquear la aplicación.



Las actividades críticas en las que es más probable que suceda algún error son en las peticiones a la API, en la importación de imágenes o en el guardado o eliminación de datos.

Control de versiones con Git

Durante el desarrollo del proyecto se utilizó el sistema de control de versiones *Git* como herramienta fundamental para organizar, documentar y proteger el código fuente. Este sistema permitió llevar un seguimiento detallado de los avances, facilitó el trabajo estructurado por fases y ayudó a gestionar los posibles errores o regresiones de forma segura.

A lo largo del desarrollo se realizaron uno o varios commits por jornada de trabajo, cada uno acompañado de un mensaje descriptivo que resumía los cambios efectuados en la jornada correspondiente. Esto permitió documentar el progreso paso a paso, volver a versiones anteriores del proyecto en caso de errores e identificar rápidamente cuándo y dónde se introdujo una funcionalidad o se modificó una parte concreta del código.

También se emplearon ramas separadas para desarrollar funcionalidades nuevas o realizar mejoras importantes, sin afectar directamente al código estable de la rama principal (*main*). Algunas de las ramas creadas fueron “*mejoras*”, para añadir mejoras estéticas a la versión final funcional de la app, o “*pruebas*”, para realizar diversas experimentaciones funcionales para la app. Una vez que las ramas cumplieron su cometido, se realizó una fusión (*merge*) controlada con la rama principal y se eliminaron para mantener una única rama en el proyecto (*main*).

El repositorio completo del proyecto está disponible en GitHub:

 <https://github.com/XoelRivas/MiBiblio>

Gracias al uso de Git, se mantuvo un entorno de desarrollo limpio, estructurado y con posibilidad de retroceso ante errores. Esta práctica resultó clave para mejorar la calidad del código y reducir riesgos durante el desarrollo.

Herramientas y librerías utilizadas

Para el desarrollo de la app se han utilizado las siguientes herramientas y librerías:

- **Python 3.9.13**: lenguaje principal del desarrollo.
- **Visual Studio Code**: editor de código.
- **API OpenLibrary**: API de la que se obtienen los datos de los libros.
- **Tkinter** (incluido en Python): librería estándar de Python para la creación de interfaces gráficas.



- **Tkcalendar 1.6.1:** complemento que permite incorporar calendarios funcionales en las interfaces.
- **CustomTkinter 5.2.2:** framework moderno basado en Tkinter que proporciona una interfaz gráfica más estilizada.
- **Pillow (PIL) 11.0.0:** utilizada para cargar y mostrar imágenes (iconos) dentro de los diferentes widgets de la interfaz.
- **Urllib.request** (incluido en Python): permite realizar solicitudes HTTP para descargar imágenes desde internet.
- **Requests 2.32.3:** permite realizar peticiones HTTP a la API de OpenLibrary para recuperar datos de libros.
- **Re** (incluido en Python): Módulo de expresiones regulares, utilizado para validar o procesar texto.
- **Io.BytesIO** (incluido en Python): facilita la lectura de datos binarios en memoria.
- **Threading** (incluido en Python): se utiliza para ejecutar tareas en segundo plano.
- **SQLite3** (incluido en Python): usado en database.py para crear y gestionar la base de datos local.
- **Datetime** (incluido en Python): Para manejar y formatear fecha, especialmente en combinación con tkcalendar.
- **Time** (incluido en Python): Utilizada en algunas funciones para añadir retardos o gestionar tiempos de espera.

ESTUDIO DE MERCADO

Introducción del estudio de mercado

La gestión personal de libros es una necesidad creciente para los aficionados a la lectura. Muchos lectores buscan soluciones digitales para organizar su biblioteca personal de manera cómoda y accesible. En un contexto donde la mayoría de las aplicaciones de gestión de libros están diseñadas para móviles o como servicios web, existe una necesidad poco cubierta de herramientas de escritorio orientadas a usuarios que prefieren una solución local, privada y sin dependencia de servicios en la nube. Este estudio analiza el mercado el mercado actual de apps de gestión de libros, identifica los principales competidores y detecta oportunidades de mejora que justifican el desarrollo de esta aplicación.

Análisis de la competencia

Existen diversas aplicaciones en el mercado que ofrecen funcionalidades similares. Algunas de las más conocidas son:

- Goodreads: Popular por su red social para lectores. Permite registrar libros, valorarlos y recibir recomendaciones. Sin embargo, depende



mucho de conexión online y no está completamente orientada a una gestión privada de libros.

- LibraryThing: Más enfocada a coleccionistas serios y bibliotecarios. Ofrece herramientas potentes para catalogar libros, pero su interfaz es poco intuitiva y puede resultar abrumadora para el usuario medio.
- Calibre: Es una de las herramientas más completas para la gestión de ebooks. Permite convertir formatos, gestionar metadatos, sincronizar con dispositivos electrónicos y más. No obstante, está más enfocada en libros electrónicos y carece de una experiencia de usuario sencilla o atractiva para quienes solo desean registrar sus lecturas físicas.
- Libib: Aplicación web que permite catalogar libros, películas, videojuegos y música. Es muy visual y fácil de usar, pero también depende de conexión online y cuenta con funciones limitadas en su versión gratuita.

Oportunidades de mejora

Según la competencia nombrada anteriormente, existen varias áreas en las que existe una oportunidad para desarrollar una herramienta diferente:

- Privacidad y uso local: La mayoría de las apps existentes requieren conexión a internet o guardan los datos en la nube. Esta app se orienta a usuarios que valoran la privacidad y prefieren guardar sus datos localmente.
- Flexibilidad en la edición: Mientras que muchas apps imponen valores fijos en los diferentes campos de los libros, *MiBiblio* permite al usuario modificar los valores de todos ellos a su gusto, incluyendo el título, autor, portada, etc.
- Simplicidad y enfoque: Se busca una aplicación liviana, sin funcionalidades innecesarias, pensada para el uso rápido y directo, ideal para lectores que solo quieren llevar un registro simple y personal de sus libros leídos.
- Creación manual: Permite crear libros desde cero en caso de que no se encuentren resultados mediante la API, lo cual da control total al usuario.

Conclusión del estudio de mercado

Este proyecto está enfocado a lectores que quieren gestionar su biblioteca personal de forma local, flexible, sencilla y sin depender de servicios externos. Frente a la complejidad de algunas soluciones actuales o a la orientación social de otras, esta aplicación destaca por su enfoque minimalista y privado.



MANUAL DE INSTALACIÓN

MiBiblio se puede obtener a través del siguiente enlace de GitHub:

🔗 <https://github.com/XoelRivas/MiBiblio.git>

- Paso 1: Descargar la carpeta que contiene el ejecutable.

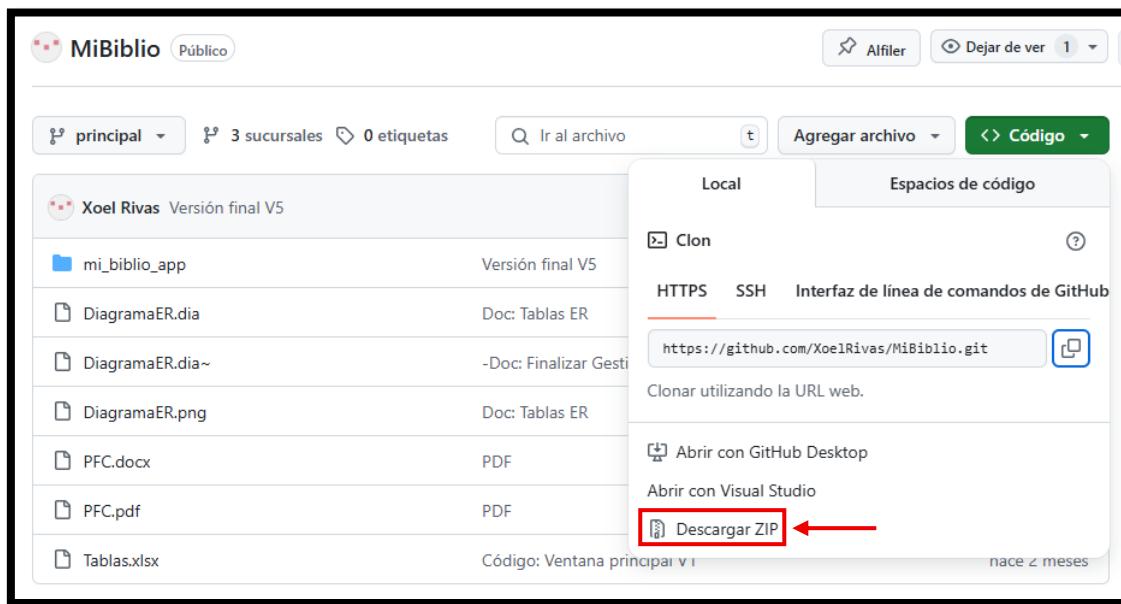


Ilustración 2: Repositorio de GitHub.

- Paso 2: Acceder a la carpeta descargada desde las descargas del propio navegador o desde las descargas del explorador de archivos.

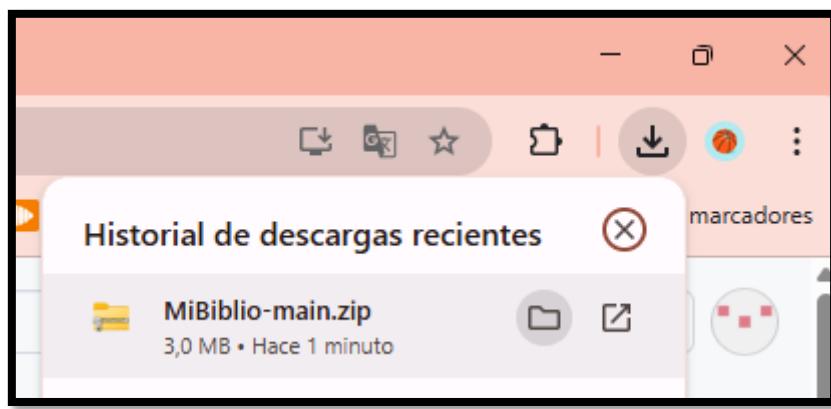


Ilustración 3: Descargas desde el navegador.



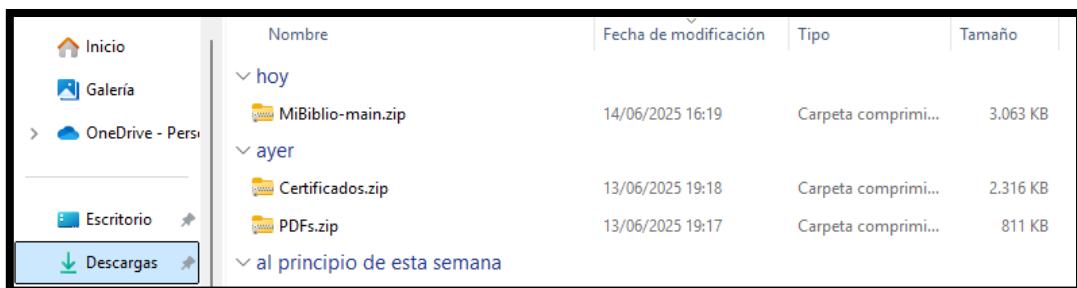


Ilustración 4: Descargas desde el explorador de archivos.

- Paso 3: Descomprimir la carpeta en la ubicación deseada.

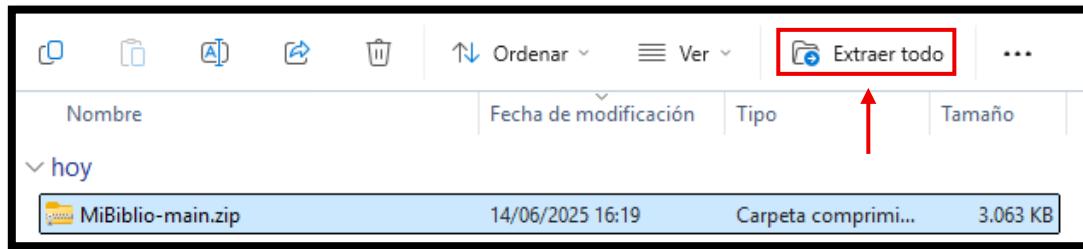


Ilustración 5: Descomprimir carpeta.

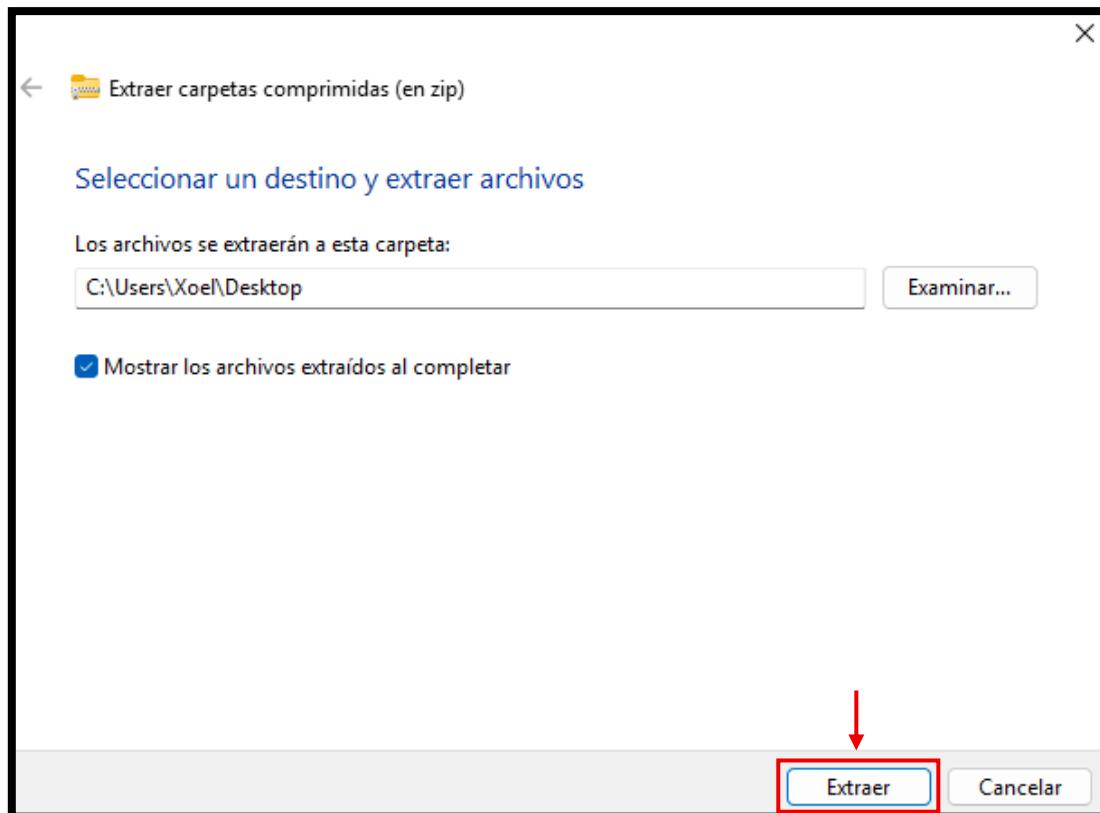


Ilustración 6: Ubicación de la extracción.



- Paso 4: La carpeta contendrá múltiples archivos. El usuario deberá acceder a la carpeta *dist*, que es donde se encuentra el ejecutable.

📁 .git	13/06/2025 21:36	Carpeta de archivos
📁 build	14/06/2025 16:46	Carpeta de archivos
📁 dist	14/06/2025 16:46	Carpeta de archivos
📁 mi_biblio_app	13/06/2025 21:09	Carpeta de archivos
DiagramaER.dia	18/04/2025 18:17	diaFile 4 KB
DiagramaER.dia~	18/04/2025 16:12	Archivo DIA~ 4 KB
DiagramaER.png	18/04/2025 16:12	Archivo PNG 62 KB
main.spec	14/06/2025 16:46	Archivo SPEC 1 KB
PFC.docx	13/06/2025 21:35	Documento de Mi... 868 KB
PFC.pdf	13/06/2025 21:35	Documento Adob... 4.740 KB
Tablas.xlsx	10/06/2025 23:38	Hoja de cálculo d... 29 KB

Ilustración 7: Carpeta *dist*.

- Paso 5: Abrir el ejecutable.

Nombre	Fecha de modificación	Tipo	Tamaño
main.exe	14/06/2025 16:46	Aplicación	22.199 KB

Ilustración 8: Ejecutable de la app.

MANUAL DE USUARIO

Una vez se ejecuta la aplicación, al usuario se le presenta directamente su biblioteca personal (inicialmente vacía). Esta es la ventana principal, de la que derivan las demás funciones de la app.



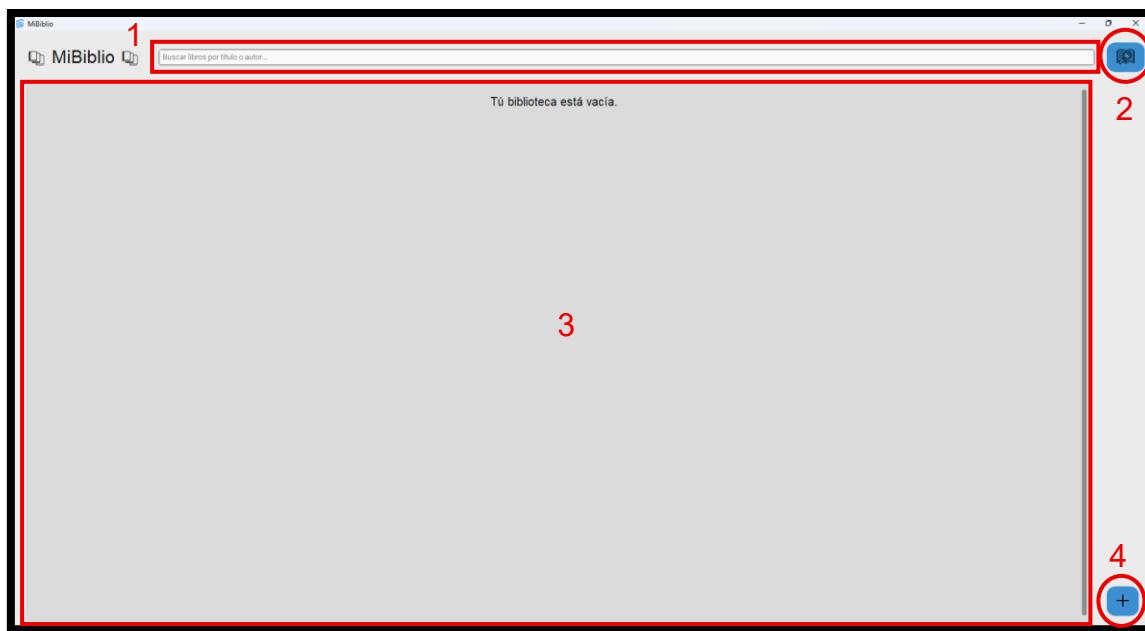


Ilustración 9: Ventana principal.

1. Buscador de los libros guardados (se puede buscar por título o por autor y si está vacío devuelve el listado completo).
2. Botón de búsqueda. Ejecuta la búsqueda con el contenido del buscador.
3. Frame en el que aparecen los libros guardados.
4. Botón para añadir o crear libro.

El usuario procede ahora a añadir un libro, para ello pulsa el botón “+”. A continuación, se despliega la siguiente ventana:

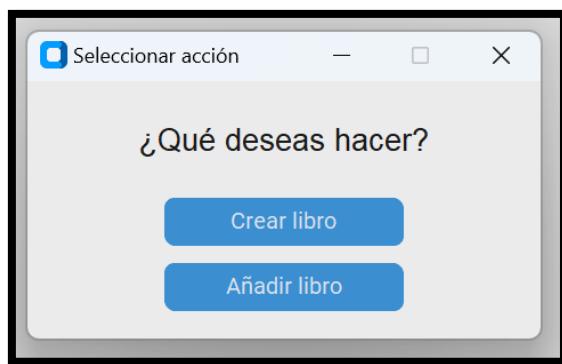


Ilustración 10: Ventana de selección de modo.

En primer lugar, se establece la casuística de que el usuario quiere añadir un libro. Pulse “Añadir libro”. Se despliega la siguiente ventana:





Ilustración 11: Ventana de búsqueda de libros.

1. Buscador de libros. Introduzca el título, el autor o el ISBN del libro que desea buscar.
2. Botón de búsqueda. Ejecuta la búsqueda.
3. Frame en el que aparecen las coincidencias de la búsqueda.

Al realizar una búsqueda:



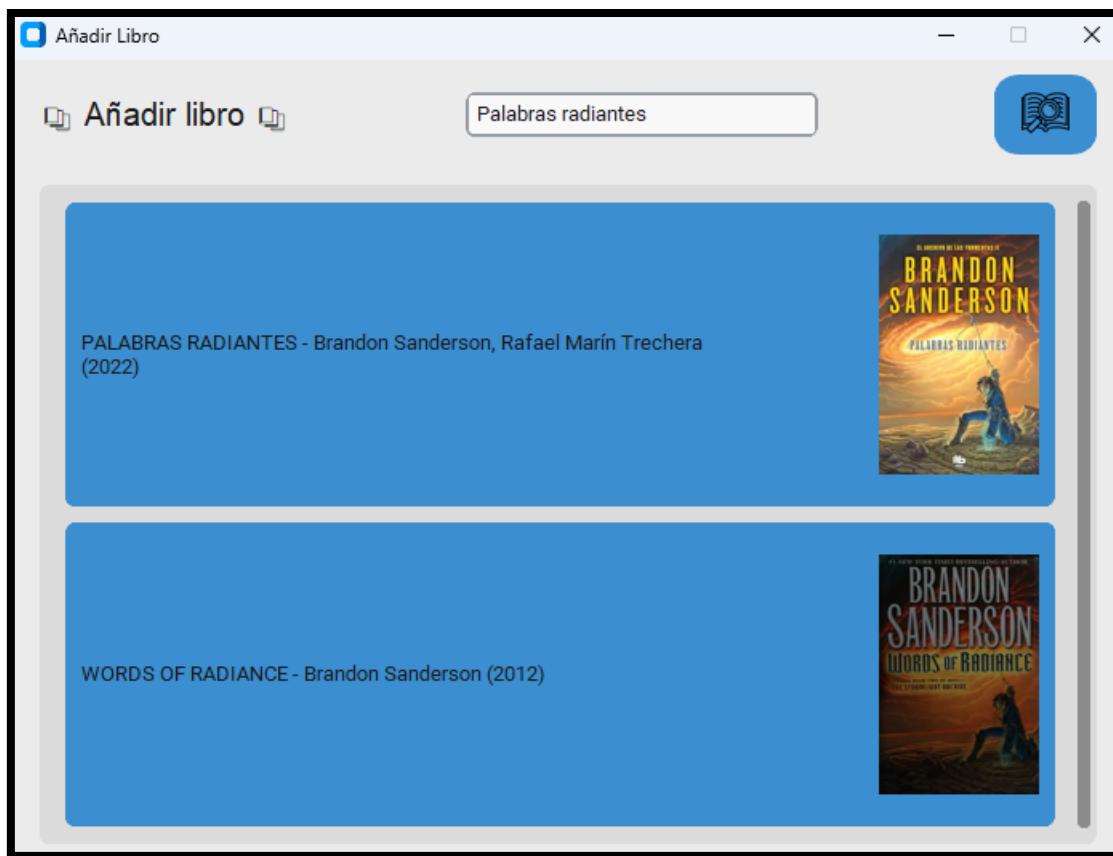


Ilustración 12: Resultados de la búsqueda por medio de la API OpenLibrary.

Pulse en el libro que desea añadir a su biblioteca. Debería ver el siguiente mensaje, que indica que el libro se añadió correctamente:

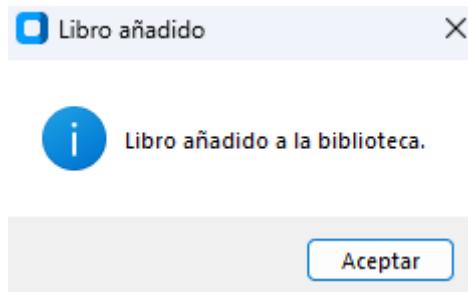


Ilustración 13: Mensaje informativo.

Ahora el libro aparece en la ventana principal:



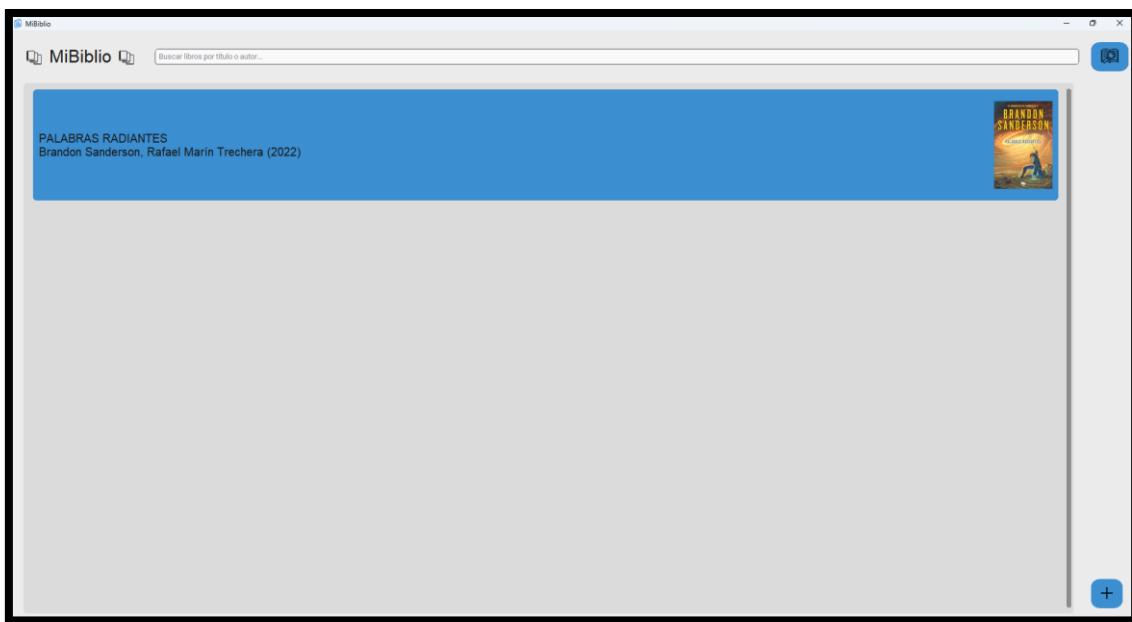


Ilustración 14: Ejemplo de libro guardado en la ventana principal.

La siguiente acción será comprobar la información del libro añadido y modificarla como guste. A continuación, si pulsa en el libro añadido, se abrirá la ventana de edición:



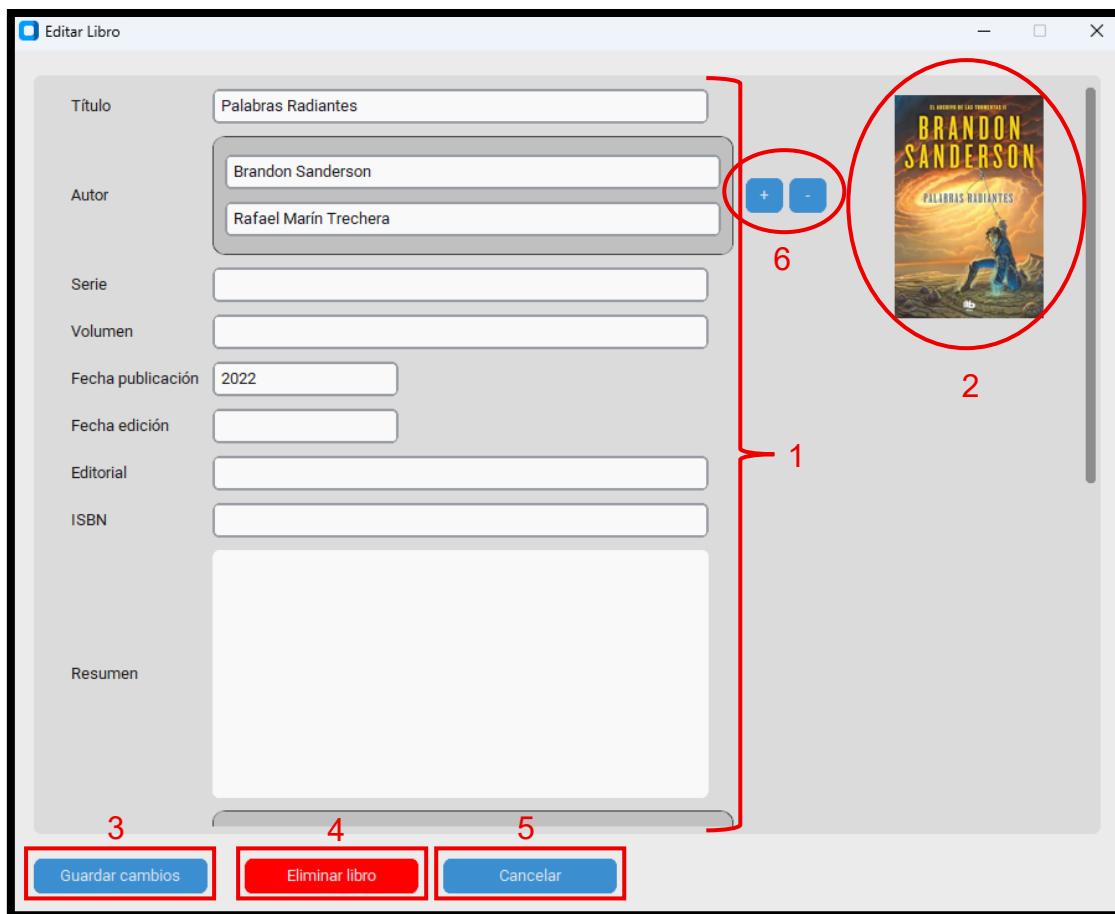


Ilustración 15: Ventana de edición.

1. Campos para completar con la información deseada (opcional). Los campos se pueden autocompletar según la información de la que disponga OpenLibrary acerca del libro.
2. Portada. Si se pulsa en la portada aparece:

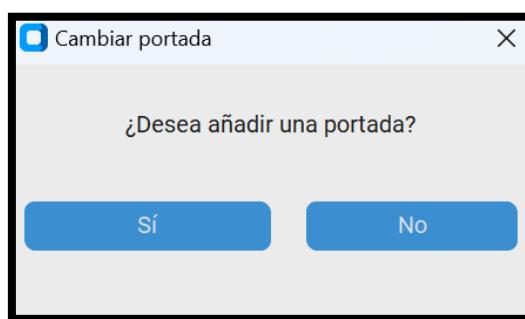


Ilustración 16: Ventana de confirmación.

- Si pulsa "Sí", puede añadir una imagen local para sustituir a la portada actual. Si pulsa "No", se mantendrá la portada actual.
3. Botón para guardar los cambios realizados.
 4. Botón para eliminar el libro seleccionado.



5. Botón para cancelar los cambios realizados.
6. Botones para añadir o quitar campos.

En esta ventana hay un campo que tiene una función especial. Se trata del campo “Estado”. Este campo ofrece cinco opciones:

- “—”: Opción por defecto. Es la que se asigna nada más se agrega el libro a la biblioteca.
- “Leyendo”
- “Leído”
- “Pendiente”
- “Abandonado”

Cada una de ellas mostrará el libro en la ventana principal con un color diferente, para que el usuario pueda identificar con mayor facilidad qué libros se ha leído, qué libros está leyendo, cuáles tiene pendientes y cuáles ha abandonado. El sistema de colores es el siguiente:

- “—”: → Azul 
- “Leyendo” → Naranja 
- “Leído” → Verde 
- “Pendiente” → Morado 
- “Abandonado” → Rojo 

Segunda casuística: el usuario quiere crear un libro desde cero. Pulse el botón “+” de la ventana principal y seleccione “Crear libro” en la ventana de elección. Se abrirá la ventana de edición nuevamente:



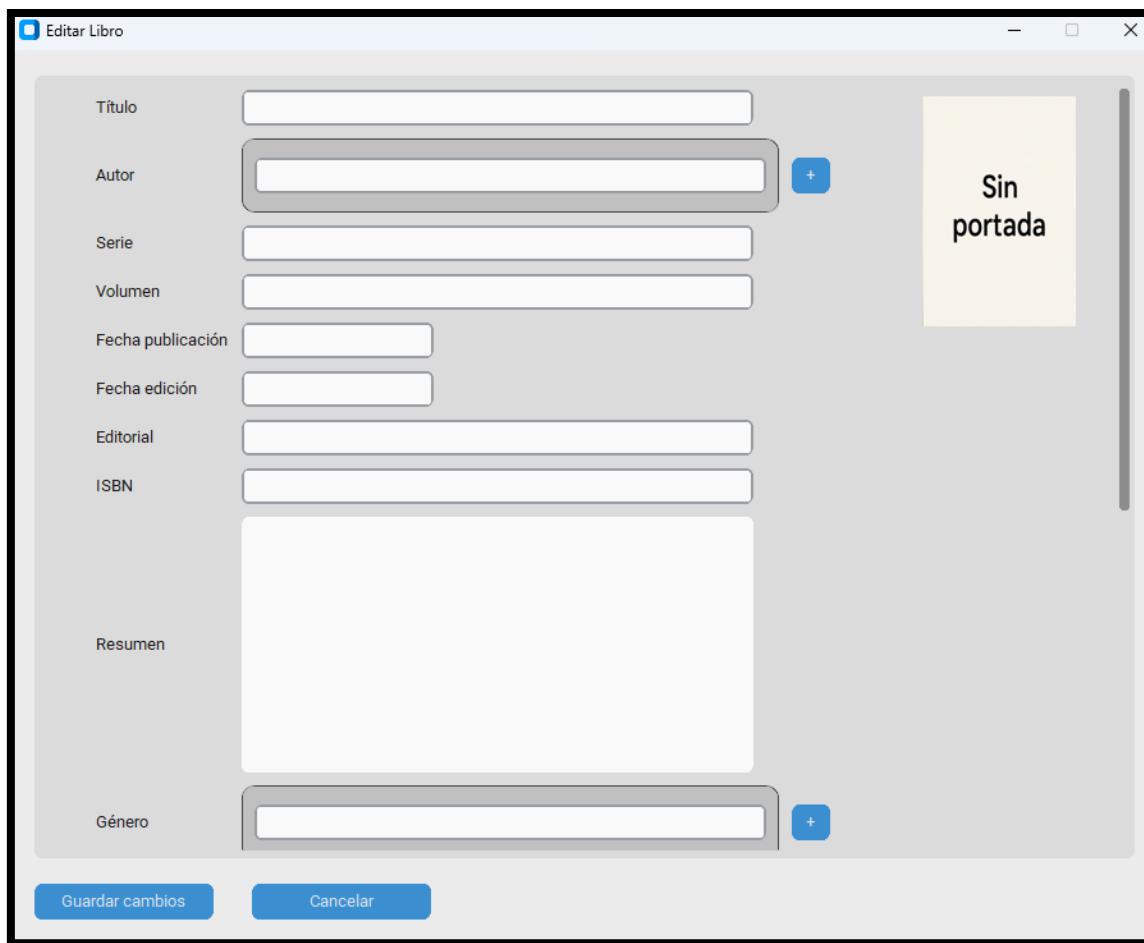


Ilustración 17: Ventana de creación de libro.

Las acciones que puede realizar el usuario aquí son las mismas que cuando edita un libro ya creado. La única diferencia es que no se autocompletará ningún campo y que no dispone de botón de eliminar, ya que el libro aún no está creado.

MEJORAS FUTURAS

En este apartado se enumeran una serie de mejoras que por falta de tiempo no se han podido implementar:

- **Estadísticas:** esto consistiría en una venta nueva a la que el usuario podría acceder para consultar diversas estadísticas de su biblioteca, tales como: histórico de libros leídos, cantidad de libros leídos en un año, autor más leído, total de páginas leídas, editoriales más leídas, géneros más leídos..., y más. Gracias a esta ventana el usuario podría tener un mayor conocimiento de su biblioteca.
- **Animaciones:** actualmente, la app es estática. Añadir diversas animaciones resultaría en una app más atractiva para su consumo.



- **Tipo de vistas:** en la ventana principal solo se dispone de un tipo de vista para visualizar los libros, esto es la vista tipo tarjeta o lista. Añadir otro tipo de vista, como la vista cuadricular facilitaría al usuario una mejor visualización de sus libros de acuerdo con sus preferencias.
- **Filtros:** el único filtro del que dispone la app es el de buscar por título o por autor los libros almacenados. Añadir una mayor variedad de filtros (filtros por género, fechas, estado...) facilitaría la búsqueda de libros al usuario.
- **Idiomas:** añadir varios idiomas para que la app sea accesible para una mayor cantidad de público.

CONCLUSIÓN FINAL

El desarrollo de *MiBiblio* ha sido una experiencia profundamente enriquecedora a nivel técnico. A lo largo del proyecto he podido aplicar de forma práctica gran parte de los conocimientos adquiridos durante el Ciclo Superior de Desarrollo de Aplicaciones Multiplataforma: diseño de interfaces gráficas, manejo de bases de datos relacionales, integración con APIs externas, etc.

Quedaron muchas mejoras por implementar, pero el proyecto ha resultado en una aplicación completamente operativa, algo que quería llevando hacer desde hace tiempo ya que ninguna aplicación de este tipo se adapta por completo a mis gustos.

Este proyecto me ha permitido enfrentarme a retos reales de programación y diseño, tomar decisiones de arquitectura, aplicar buenas prácticas como el control de versiones con Git, y ponerme en la piel del usuario para mejorar la experiencia de uso. La implementación de una interfaz intuitiva, junto con una base de datos sólida y una conexión funcional con OpenLibrary, dan como resultado una aplicación útil, ligera y funcional para los que, como yo, quieren llevar un recuento de sus lecturas de una manera más cómoda que, por ejemplo, hacerlo en un Excel, que era mi caso. Y de eso nació la idea de desarrollar esta app.



ANEXO

Presupuesto

Costes de desarrollo			
Tarea	Horas	Tarifa/h	Subtotal (€)
Inicio y organización	0,50	20 €	10,00 €
Redacción de Introducción y Objetivos	0,5	20 €	10,00 €
Análisis y especificación de requisitos	1	20 €	20,00 €
Diagrama ER y diseño de la base de datos	2	20 €	40,00 €
Creación del script de la base de datos	1	20 €	20,00 €
Planificación del proyecto	0,5	20 €	10,00 €
Diseño de casos de uso	1	20 €	20,00 €
Diseño para la gestión de la información y datos	1	20 €	20,00 €
Implementación de la base del sistema	3	20 €	60,00 €
Implementación del formulario de añadir libros	3	20 €	60,00 €
Integración de relaciones N:M (autores y géneros)	2,5	20 €	50,00 €
Desarrollo de la interfaz gráfica básica (CustomTkinter)	3	20 €	60,00 €
Búsqueda de libros	2	20 €	40,00 €
Implementación de guardado y carga de datos	1,5	20 €	30,00 €
Mejora de la interfaz y validaciones	2	20 €	40,00 €
Pruebas funcionales y depuración	2	20 €	40,00 €
Finalización de funcionalidades (detalles extra)	1	20 €	20,00 €
Redacción y finalización de la documentación	2	20 €	40,00 €
Entrega final y revisión	0,5	20 €	10,00 €
Total	30,00	20 €	600,00 €

Tabla 16: Costes de desarrollo.



Costes de software y herramientas		
Herramienta	Licencia/Coste	Subtotal (€)
Visual Studio Code	Gratis	0 €
Python	Gratis	0 €
SQLite	Gratis	0 €
OpenLibrary (API)	Gratis	0 €
Total		0 €

Tabla 17: Costes de software y herramientas.

Costes de hardware		Subtotal (€)
Recurso		Subtotal (€)
Equipo de trabajo (ordenador, monitores, periféricos, etc.)		800 €
Total		800 €

Tabla 18: Costes de hardware.

Costes de electricidad/internet			
Recurso	Meses	Tarifa/mes	Subtotal (€)
Electricidad	3	62,00 €	186 €
Internet	3	65,00 €	195 €
Total			381 €

Tabla 19: Costes de electricidad/internet.

Total presupuesto	
Concepto	Subtotal
Desarrollo	600 €
Software y herramientas	0 €
Hardware	800 €
Electricidad/internet	381 €
Total estimado	1.781 €

Tabla 20: Total presupuesto.

Webgrafía

- 🔗 [Documentación API OpenLibrary](#)
- 🔗 [Documentación Tkinter](#)
- 🔗 [Documentación Customtkinter](#)
- 🔗 [Documentación Tkcalendar](#)



- [!\[\]\(9aedebbc0e10803492e4a1bee3db97ab_img.jpg\) Documentación Pillow](#)
- [!\[\]\(07f30b8c2111b1275646b95bd5f90099_img.jpg\) Documentación Urllib request](#)
- [!\[\]\(f7b5f958681e1d8d9b0235f9208c62ae_img.jpg\) Documentación Requests](#)
- [!\[\]\(0b28a03349cadf6e557d5898759789be_img.jpg\) Documentación Re](#)
- [!\[\]\(a55e324722d963909eaedf28feb38310_img.jpg\) Documentación Io](#)
- [!\[\]\(cd9f0ac46c92e8ce77cbff66cb048fe0_img.jpg\) Documentación Threading](#)
- [!\[\]\(36acf430a0812fa13c3bcc4bcf9e2eb6_img.jpg\) Documentación Sqlite3](#)
- [!\[\]\(3f0c79218043ca40f80d4b0df15dfdf2_img.jpg\) Documentación Datetime](#)
- [!\[\]\(2b881fffae476b429a956df921571466_img.jpg\) Documentación Time](#)

