

Aplicación multiplataforma de compra y gestión de libros electrónicos

Andrés García Payá y Alexis Willy Andía Rosales

Contenido

1. Resumen	4
2. Introducción	4
3. Objetivos	5
4. Palabras clave.....	5
5. Análisis de contexto	6
5.1. Análisis del Contexto.....	6
5.1.1. Competencia	6
5.1.2. DAFO del Proyecto	7
5.1.2.1. Fortalezas	7
5.1.2.2. Debilidades.....	7
5.1.2.3. Oportunidades	7
5.1.2.4. Amenazas	7
5.2. Innovación	7
6. Modelo de datos	8
6.1. Resumen	8
6.2. Tablas	8
6.2.1. Usuario	8
6.2.2. Biblioteca	8
6.2.3. Libro.....	9
6.2.4. Deseo.....	9
6.2.5. Biblio_libro	10
6.2.6. Saga.....	10
6.2.7. Idioma	10
6.2.8. Genero.....	10
6.2.9. Autor	10
6.2.10. Libro_idioma.....	11
6.2.11. Libro_autor.....	11
6.2.12. Libro_genero	11
6.2.13. Editorial.....	11
6.2.14. Libro_edit.....	12
6.3. Diagrama E.R.....	12
7. Diseño	12
7.1. Elementos comunes	13
7.1.1. Login, registro, recuperar contraseña.....	13
7.2. UI de la aplicación de escritorio	14

7.2.1.	Vista usuario.....	14
7.2.2.	Vista Editorial	15
7.3.	UI de la aplicación móvil.....	16
8.	Casos de uso.....	17
9.	Diagrama de clases.....	18
9.1.	Versión de escritorio	18
9.1.1.	App	18
9.1.2.	ControllerLogin.....	18
9.1.3.	ManejaImagen.....	19
9.1.4.	Properties	19
9.1.5.	ManejaArchivos	19
9.1.6.	Ebook	19
9.1.7.	ControllerBiblioteca.....	19
9.1.8.	ControllerTienda	19
9.1.9.	Conexión	19
9.2.	UML del API REST.....	20
9.3.	UML del Servidor	21
9.4.	UML de Android.....	22
10.	Planificación.....	22
10.1.	Diagrama de Gantt.....	22
10.2.	Definición de Recursos y Logística	23
11.	Implementación.....	24
11.1.	Aplicación de escritorio	24
11.1.1.	Introducción	24
11.1.2.	Clase Properties.....	24
11.1.3.	Clase ManejaArchivos	26
11.1.4.	Login	28
11.1.5.	Biblioteca	32
11.1.6.	Tienda.....	38
11.1.7.	Editorial.....	38
11.1.8.	Ventana Perfil	40
11.2.	OCR.....	40
11.2.1.	Introducción	40
11.2.2.	Implementación.....	41
11.2.3.	Tratamiento de imagen.....	41
11.2.4.	Futuras mejoras	41

11.2.5.	Motivos de la no implementación	42
11.3.	Pagos.....	42
11.3.1.	Introducción	42
11.3.2.	Implementación.....	42
11.3.3.	Flujo del proceso	43
11.4.	API RESTful	43
11.4.1.	Introducción	43
11.4.2.	Modelos de Dominio	43
11.4.3.	Repositorios	45
11.4.4.	Servicios	45
11.4.5.	Utilidades	47
11.4.6.	DTOs	47
11.4.7.	Controladores.....	48
11.4.8.	Configuraciones.....	49
12.	Puesta en marcha.....	49
12.1.	Configuración	49
12.2.	Seguridad	50
12.3.	Legalidad.....	50
12.4.	Pase a producción.....	50
13.	Control de calidad	51
13.1.	Aplicación de escritorio	51
13.1.1.	Login.....	51
13.1.2.	Biblioteca.....	54
13.1.3.	Tienda.....	59
13.1.4.	Perfil.....	60
13.1.5.	Ventana editorial.....	60
14.	Plan de empresa.....	¡Error! Marcador no definido.
15.	Bibliográfica.....	61

1. Resumen

Esta aplicación constara de una versión de PC y una versión móvil. Ambas versiones tendrán funciones similares. Tanto la versión de PC como la versión de móvil están planteadas para que un usuario normal no esté obligado a usar las dos. Algunas de las funciones compartidas entre las versiones, son la capacidad de utilizar la tienda, descargar libros, leerlos, modificar ciertas características.

Las principales diferencias entre las dos versiones vienen de dos funciones concretas siendo estas la capacidad de la versión móvil de escanear una imagen que pueda contener el título de los libros, permitiendo buscar en la base de datos si ese libro está disponible en la tienda y la versión de escritorio, que tendrá un apartado exclusivo para las editoriales, lo que las dará la capacidad de subir libros a la tienda. La versión de escritorio también contara con la capacidad administrar la biblioteca interna de un libro electrónico.

2. Introducción

La idea de este proyecto surge de una reciente noticia relacionada con Amazon, en la que comentan que la empresa elimina la capacidad de descargarse los libros que tenga uno ya comprados, así como también eliminarlos y o actualizarlos del propio dispositivo. Si bien esto es algo que solo afectaría a los usuarios de Kindle, estos mismos son gran parte del mercado, por lo que puede suponer un problema. Esto también imposibilita a los usuarios de Kindle a acceder a la parte de su biblioteca que pertenezca a Amazon si no disponen de una conexión a internet.

Otra de las razones por las que se escogió el proyecto fue que, como consumidores, nos parecía más cómodo tener todo lo relacionado con el tema en una sola aplicación, así como hacen otros sectores como pueda ser el de los videojuegos, que en PC cuenta con diversas aplicaciones desde donde se puede comprar el juego deseado, así como ejecutarlo, entre otras cosas. La idea es aplicar el enfoque usado en estos sectores al mundo de los libros electrónicos, ya que actualmente, no existe una aplicación que fusione la idea de la compra con el apartado de gestión de los libros.

De estos dos motivos principalmente nace la idea de este proyecto, una alternativa que le brinde al usuario un control mayor sobre el contenido digital que posee, así como también una mayor comodidad a la hora de tratar con él.

En resumen, este proyecto nace de la intención de mejorar la experiencia de los usuarios, dándoles una herramienta para adquirir y administrar sus libros de una forma sencilla, eficiente, centralizada y menos restrictiva que el resto de opciones del mercado.

3. Objetivos

El objetivo de esta aplicación es acercar lo más posible las ventajas que tienen los libros físicos sobre los digitales y viceversa. También se tiene la intención de centralizar todo el proceso relacionado a la compra de libros y el manejo de estos.

Para cumplir con estos objetivos, se ha decidido crear dos versiones de la aplicación, la de móvil y la de escritorio, estando la versión móvil enfocada en un tipo de usuario más inexperto por decirlo de alguna manera, ya que esta presenta unas funcionalidades más limitadas en comparación con la versión de escritorio que está diseñada con un usuario más avanzado o una editorial en mente.

La versión móvil brindará a los usuarios la opción de acceder a la tienda de la aplicación, añadir libros a su lista de deseados, descargar los libros que tenga comprados y leerlos. También podrán escanear la foto de una portada o una hoja donde tengan apuntado el título, para comprobar si se dispone de este en la tienda. Para llevar a cabo esto se utilizará tesseract OCR, que estará integrado en un servidor y al que se le enviarán las fotos a través de una API. Ya que uno de los motivos de la aplicación es no depender en sobremedida de grandes empresas se ha optado por este OCR sobre Google Cloud visión.

La versión de escritorio contará con las mismas funciones a excepción de la capacidad escanear imágenes. También contará con la capacidad de tener no solo la biblioteca digital, sino también con la opción de tener varias bibliotecas locales, en las que el usuario podrá almacenar también libros obtenidos de otras tiendas. Si lo desea, podrá leer los libros de su biblioteca desde la propia aplicación. Esta no es la única diferencia, también tendrá la capacidad convertir el formato de sus libros, entre .pdf y .epub, que es el utilizado en los libros electrónicos. Por último, también podrá manejar de estos, permitiéndole añadir o eliminar libros.

En lo que respecta a las editoriales, estas tendrán una sección oculta a la que solo podrán acceder ellas, que les permitirá subir libros al servidor, aumentando así su catálogo disponible en la tienda. Para ello, se les asignará a los usuarios un rol en función de si son una editorial o un consumidor. El rol de editorial se dará a mano mientras el que el rol de usuario común se le dará automáticamente al resto.

Para complementar las dos versiones, se dispondrá de un servidor de descarga, donde estará integrado el OCR, y desde donde se manejarán los pagos a través de la API de Stripe.

4. Palabras clave

API, versión de escritorio, versión móvil, libro electrónico, OCR, roles, aplicación multiplataforma, conversión de formatos

5. Análisis de contexto

5.1. Análisis del Contexto

En los últimos años, el mundo de los libros electrónicos ha ganado mucha popularidad gracias a la facilidad que ofrecen para acceder, transportar y leer contenido desde prácticamente cualquier dispositivo. Sin embargo, este crecimiento no ha estado libre de problemas. Muchas plataformas conocidas, como Amazon Kindle o Google Books, aplican restricciones que afectan directamente al usuario, como el uso de DRM que limita el acceso a los libros solo desde ciertos dispositivos o cuentas. Incluso, en algunos casos, empresas como Amazon han eliminado la posibilidad de descargar libros comprados o los han modificado sin permiso del usuario.

Estas limitaciones han encendido la alarma entre muchos lectores digitales, que empiezan a cuestionarse si realmente "poseen" los libros que han comprado. Además, para aquellos que buscan una experiencia más libre o desean organizar sus libros como prefieran, las alternativas actuales son pocas o demasiado técnicas.

Inspirándonos en cómo otras industrias han resuelto problemas similares —como el mundo de los videojuegos en PC, donde existen plataformas como Steam o Epic Games que permiten comprar, gestionar y jugar desde una misma aplicación— nace este proyecto. Queremos trasladar ese mismo enfoque al mundo de los libros electrónicos: una única aplicación que no solo permita comprar libros, sino también gestionarlos, organizarlos y leerlos, todo desde un solo lugar y con total control por parte del usuario.

5.1.1. Competencia

Al analizar lo que ya existe en el mercado, identificamos varias plataformas populares:

Amazon Kindle: muy conocida, con un catálogo enorme, pero extremadamente cerrada y restrictiva. Los libros tienen DRM y están limitados a su propio ecosistema.

Google Play Books: más flexible, pero sin funciones avanzadas de organización local o acceso editorial.

Kobo: otra buena opción, pero también con restricciones técnicas y de formato.

Calibre: excelente para la gestión local de libros, sin tienda integrada, pero poco accesible para usuarios no técnicos.

Ninguna de ellas combina todas las funcionalidades que los usuarios modernos desean: compra, organización personalizada, gestión de libros externos, lectura, OCR, y además, herramientas para editoriales. Ese vacío es justamente el que nuestra aplicación busca llenar.

5.1.2. DAFO del Proyecto

5.1.2.1. Fortalezas

- Aplicación multiplataforma: misma experiencia desde PC y móvil.
- Escaneo de portadas mediante OCR para encontrar libros fácilmente.
- Conversión entre formatos de lectura (.pdf y .epub).
- Posibilidad de añadir libros propios y organizarlos en bibliotecas locales.
- Acceso exclusivo para editoriales para publicar directamente en la tienda.
- Uso de tecnologías libres y sostenibles como PostgreSQL y Tesseract.

5.1.2.2. Debilidades

Se requiere una inversión inicial para mantener servidores y servicios. Al principio puede costar atraer usuarios y editoriales si ya usan otras plataformas. Doble desarrollo y mantenimiento: versión móvil y versión de escritorio.

5.1.2.3. Oportunidades

Muchos usuarios buscan alternativas más libres y completas. Editoriales independientes necesitan espacios donde publicar sin intermediarios. Hay un hueco en el mercado para una plataforma “todo en uno” para ebooks.

5.1.2.4. Amenazas

Grandes empresas podrían adoptar ideas similares. Obstáculos legales o cambios en normativas de protección de contenido. Avances tecnológicos que exijan adaptación constante.

5.2. Innovación

Este proyecto nace con la intención de romper barreras y ofrecer a los lectores digitales una experiencia mucho más completa, intuitiva y libre. A diferencia de las plataformas tradicionales, esta aplicación no solo se centra en la venta de libros, sino que da poder al usuario para organizar, personalizar y conservar su contenido como prefiera.

Una de las funciones más innovadoras es el uso de OCR (Reconocimiento Óptico de Caracteres) integrado a través de Tesseract. Gracias a esta tecnología, los usuarios pueden escanear la portada de un libro o una nota donde esté escrito el título, y la aplicación buscará automáticamente si ese libro está disponible en la tienda. Esto convierte al móvil en una herramienta de descubrimiento literario práctica y potente.

Otra gran innovación es la posibilidad de convertir libros entre formatos .epub y .pdf, algo especialmente útil para quienes usan distintos dispositivos de lectura. Además, la app permite gestionar no solo los libros comprados, sino también aquellos que el usuario ya tenga en su ordenador o que haya adquirido por otras vías. Todo esto se podrá hacer desde una biblioteca personal organizada por el propio usuario, sin depender de ninguna nube externa o permisos especiales.

Y por supuesto, no nos olvidamos de las editoriales. La versión de escritorio incluye una sección especial para ellas, donde podrán subir sus libros, gestionar su catálogo y publicar directamente en la tienda. Esto es algo poco común en otras plataformas, y permite dar visibilidad a editoriales más pequeñas o independientes que buscan conectar con los lectores sin pasar por intermediarios.

En definitiva, nuestra aplicación apuesta por la libertad, el control y la centralización, ofreciendo una solución que no existe hoy en el mercado. Todo esto usando tecnologías abiertas, pensadas para durar y crecer con el tiempo.

6. Modelo de datos

6.1. Resumen

Para hacer el apartado de base de datos se ha decidido utilizar PostgreSQL ya que es un proyecto open source por lo que se alinea con nuestra visión de marca. Otra de las razones por la que se decidió usar PostgreSQL fue que se trata de una base relacional orientada a objetos, lo que la hace ideal cuando se usa en conjunto de un lenguaje basado orientado a objetos como pueda ser java. Por último, se escogió PostgreSQL por su escalabilidad, ya que, en el contexto de este proyecto no es necesario manejar una gran carga de trabajo, pero deja la puerta abierta a esa capacidad.

6.2. Tablas

6.2.1. Usuario

- **Resumen:** Esta tabla contiene la información de las cuentas de los usuarios que hayan creado una cuenta. Se crea de forma automática cada vez que un usuario se registra en la aplicación, ya sea la versión de móvil o la de escritorio.
- **Campos:**
 - **ID_usuario:** Es la clave principal del usuario, es incremental.
 - **Nombre:** Es el nombre real del usuario, puede ser NULL ya que la información se termina de añadir si se quiere en el apartado de perfil.
 - **Apellidos:** Son los apellidos reales del usuario, pueden ser NULL ya que la información se termina de añadir si se quiere en el apartado de perfil.
 - **Usuario:** Es el nombre con el que podrá iniciar sesión el usuario.
 - **Correo:** Es el correo asociado a la cuenta.
 - **Contraseña:** Es la contraseña hasheada del usuario.
 - **Semilla:** Es la semilla utilizada para encriptar la contraseña del usuario.
 - **Cartera:** Es el saldo del que dispone el usuario.
 - **Fecha_registro:** Es la fecha en la que el usuario creó su cuenta.
 - **Ultimo_registro:** Es la fecha de la última conexión de la cuenta asociada
- **Relaciones:**
 - Relación 0:1 con editorial ya que una editorial solo puede tener un usuario, pero un usuario no tiene por qué tener editorial.
 - Relación n:m con usuario ya que un usuario puede desear n libros y un libro puede ser deseado por n usuarios.

6.2.2. Biblioteca

- **Resumen:** Es la tabla que almacena los libros que se hayan comprado en la aplicación, por lo que no tiene en cuenta los libros de las posibles bibliotecas locales del usuario.
- **Campos:**
 - **ID_biblioteca:** Es la clave primaria de la biblioteca, es incremental.
 - **ID_FK_usuario:** Es la clave foránea de usuario.
 - **Ultimo_registro:** Es la fecha de la última vez que se añadió un libro a esta biblioteca.

- **Relaciones:**
 - Relación 1:1 con la tabla usuario ya que un usuario solo puede tener una biblioteca.
 - Relación n:m con libro ya que un libro puede estar en varias bibliotecas y varias bibliotecas pueden tener un libro

6.2.3. Libro

- **Resumen:** Es la tabla que almacena la información de los libros que están a la venta al público desde la aplicación. Tiene una relación n:1 con saga.
- **Campos:**
 - **ID_libro:** Es la clave primaria de la tabla, es incremental. En este caso es el ISBN
 - **Título:** Es el título del libro.
 - **Fecha_public:** Es la fecha en la que se publicó el libro.
 - **Precio:** Es el precio actual del libro.
 - **Descuento:** Es el descuento que pueda tener el libro en el momento. Puede ser NULL ya que no tiene por qué tener descuento alguno.
 - **DRM:** Indica si el libro tiene DRM.
 - **N_paginas:** Indica el número de páginas que tiene el libro.
 - **N_votos:** Es el número de veces que este libro se ha calificado, se utilizara para sacar la puntuación medio del libro
 - **Sinopsis:** Es una breve descripción de la trama del libro
 - **Valoración:** Es la nota media del libro.
 - **URLibro:** Es la dirección desde la que se puede descargar el libro.
 - **URLportada:** Es la dirección desde la que se puede descargar la portada del libro.
 - **ID_FK_saga:** Es la clave foránea de la tabla saga. Puede ser NULL ya que un libro no tiene por qué pertenecer a una saga.
- **Relaciones**
 - Relación n:0 con saga ya que un libro puede o no estar en una saga, pero una saga puede tener n libros.
 - Relación n:m con autor ya que un libro puede tener n autores y un autor escribir n libros.
 - Relación n:m con idioma ya que un libro puede estar en n idiomas y un idioma puede estar en n libros.
 - Relación n:m con genero ya que un libro puede tener n géneros y un género puede estar en n libros.
 - Relación n:m con usuario ya que un usuario puede desear n libros y un libro puede ser deseado por n usuarios.

6.2.4. Deseo

- **Resumen:** Tabla intermedia resultante de la relación n:m de usuario y libro.
- **Campos:**
 - **ID_FK_usuario:** Es la clave tanto primaria como foránea relacionada con la tabla usuario.
 - **ID_FK_libro:** Es la clave tanto primaria como foránea relacionada con la tabla libro.

- **Fecha_registro:** Fecha en la que se creó el registro

6.2.5. Biblio_libro

- **Resumen:** Es la tabla intermedia resultado de la relación n:m entre las tablas libro y biblioteca.
- **Campos:**
 - **ID_FK_biblioteca:** Es la clave tanto primaria como foránea relacionada con la tabla biblioteca.
 - **ID_FK_libro:** Es la clave tanto primaria como foránea relacionada con la tabla libro.
 - **Fecha_compra:** Es la fecha en la que se compró el libro.
 - **Precio:** Precio al que se compró el libro.

6.2.6. Saga

- **Resumen:** Es la tabla que almacena información de las distintas sagas de libros
- **Campos:**
 - **ID_saga:** Es la clave principal, es incremental.
 - **Saga:** Es el nombre de la saga a la que pertenece el libro.
- **Relaciones:**
 - Relación 0:n con libro ya que un libro puede o no pertenecer a una saga pero una saga tiene n libros.

6.2.7. Idioma

- **Resumen:** Es la tabla que contiene los distintos idiomas en los que puede estar un libro.
- **Campos:**
 - **ID_idioma:** Es la clave primaria, es incremental.
 - **Idioma:** El nombre del idioma.
- **Relaciones:**
 - Relación n:m con libro ya que un libro puede estar en n idiomas y un idioma puede estar en n libros.

6.2.8. Genero

- **Resumen:** Es la tabla que contiene los distintos géneros que puede tener un libro.
- **Campos:**
 - **ID_genero:** Es la clave primaria, es incremental.
 - **Genero:** Es el género al que puede pertenecer el libro.
- **Relaciones:**
 - Relación n:m con genero ya que un libro puede tener n géneros y un género puede estar en n libros.

6.2.9. Autor

- **Resumen:** Esta tabla contiene información básica del autor, ya que son las editoriales quienes tratan con ellos.
- **Campo:**
 - **ID_autor:** Es la clave primaria, es incremental.
 - **Nombres:** Es el nombre del autor.
 - **Apellidos:** Son los apellidos del autor.

- **Relaciones:**
 - Relación n:m con idioma ya que un libro puede tener n autores y un autor puede tener en n libros.

6.2.10. Libro_idioma

- **Resumen:** Es la tabla intermedia creada a partir de la relación n:m entre la tabla libro y la tabla idioma.
- **Campos:**
 - **FK_ID_libro:** Es la clave tanto primaria como foránea relacionada con la tabla libro.
 - **FK_ID_idioma:** Es la clave tanto primaria como foránea relacionada con la tabla idioma.

6.2.11. Libro_autor

- **Resumen:** Es la tabla intermedia creada a partir de la relación n:m entre la tabla libro y la tabla autor.
- **Campos:**
 - **FK_ID_libro:** Es la clave tanto primaria como foránea relacionada con la tabla libro.
 - **FK_ID_autor:** Es la clave tanto primaria como foránea relacionada con la tabla autor.

6.2.12. Libro_genero

- **Resumen:** Es la tabla intermedia creada a partir de la relación n:m entre la tabla libro y la tabla género.
- **Campo:**
 - **FK_ID_libro:** Es la clave tanto primaria como foránea relacionada con la tabla libro.
 - **FK_ID_genero:** Es la clave tanto primaria como foránea relacionada con la tabla género.

6.2.13. Editorial

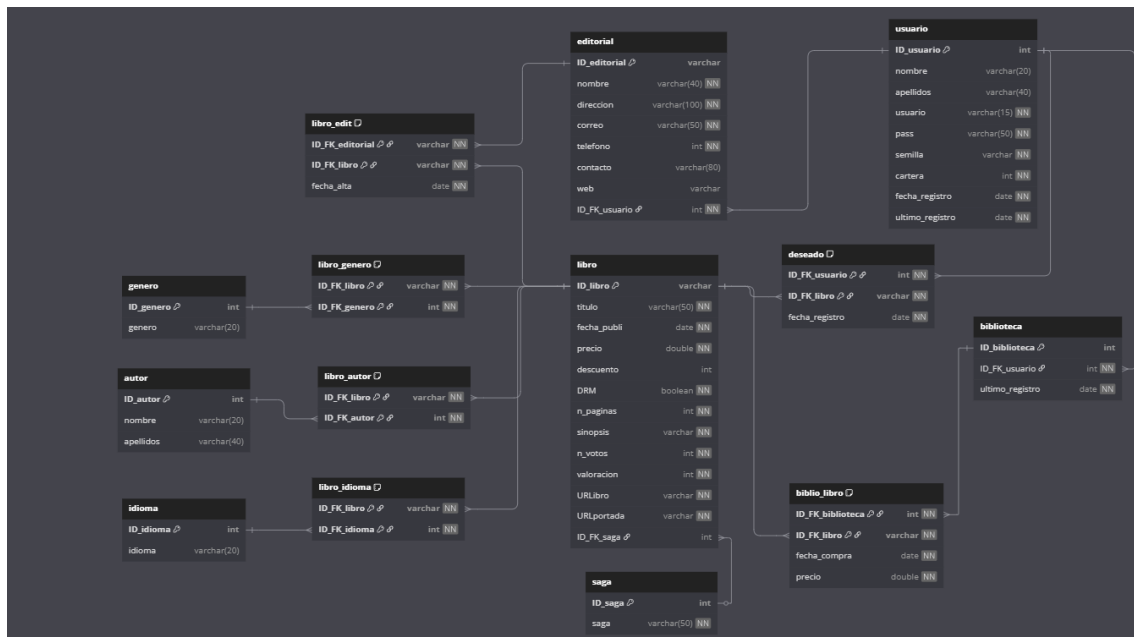
- **Resumen:** Es la tabla que contiene información referente a las editoriales con las que está asociada la aplicación. Desde dentro de la aplicación no se puede crear un registro de esta tabla ni el usuario al que va asociado.
- **Campos:**
 - **ID_editorial:** Es la clave primaria. En este caso es el NIF de la empresa.
 - **Nombre:** Nombre asociado a esa empresa.
 - **dirección:** Dirección física de la empresa.
 - **Correo:** Es el correo de la empresa.
 - **Teléfono:** Es el teléfono de la empresa.
 - **Contacto:** Es el nombre de persona de contacto. Puede ser NULL.
 - **Web:** Es la dirección de la web de la editorial en caso de que tengan. Puede ser NULL.
 - **ID_FK_usuario:** Es la clave foránea de la tabla usuario.
- **Relaciones:**

- Relación n:m con la tabla libros ya que una editorial puede tener varios libros publicados y un libro puede ser publicado en varias editoriales a la vez.
- Relación 0:1 con la tabla usuario ya que una empresa siempre tendrá un usuario en la aplicación, pero un usuario puede no tener asociado una editorial.

6.2.14. Libro_edit

- **Resumen:** Es la tabla intermedia creada a partir de la relación n:m de la tabla libro y la tabla editorial.
- **Campos:**
 - **ID_FK_editorial:** Es la clave tanto primaria como foránea relacionada con la tabla editorial.
 - **ID_FK_libro:** Es la clave tanto primaria como foránea relacionada con la tabla libro.
 - **Fecha_alta:** Es la fecha en la que el libro fue publicado por esa editorial.

6.3. Diagrama E.R

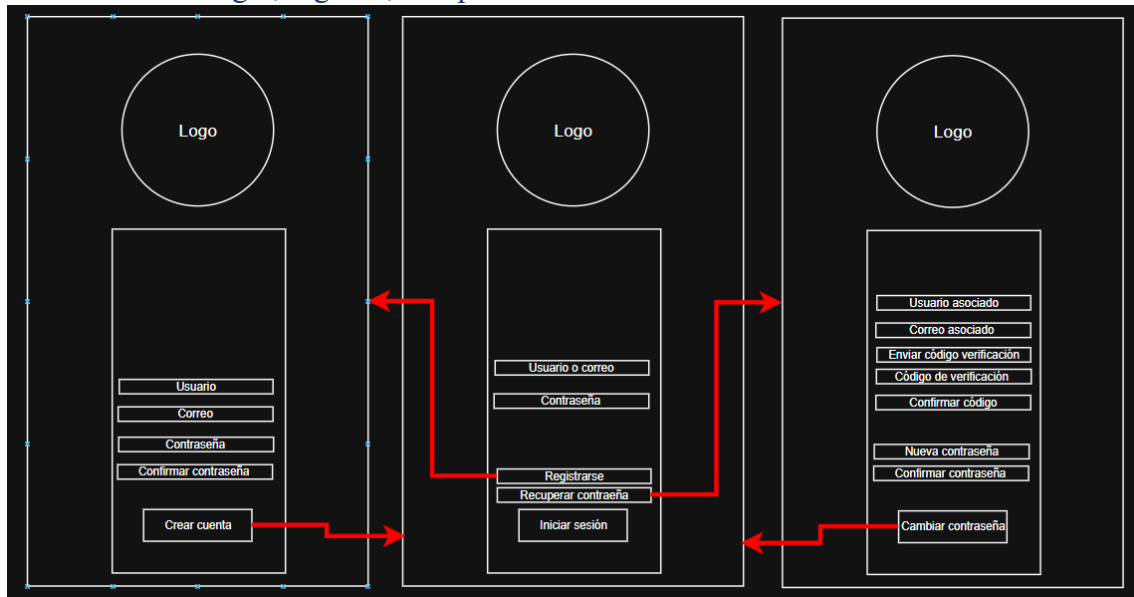


7. Diseño

Ya que la aplicación cuenta con dos versiones, una para escritorio y una móvil, cada aplicación contará con su respectiva interfaz gráfica. El objetivo es que sea lo más natural e intuitivo pasar de una versión a la otra, para ello se seguirá un diseño parecido en ambas, teniendo en cuenta las limitaciones del entorno con respecto a su contraparte. A su vez, también tendrán elementos comunes como puede ser el login, el registro, la recuperación de contraseña, y la pagina de ayuda, ya que esta última simplemente redirigirá a una url de que estará colgada en la página web de la aplicación. Para la interfaz gráfica de la versión de escritorio se utilizará javafx y para la versión móvil se utilizará kotlin. Como nota a tener en cuenta, los diseños que se mostraran son prototipos, por lo que no se verá reflejado cosas como colores, iconos, etc. También se contará con un servidor, pero no se hablará de él en esta sección.

7.1. Elementos comunes

7.1.1. Login, registro, recuperar contraseña



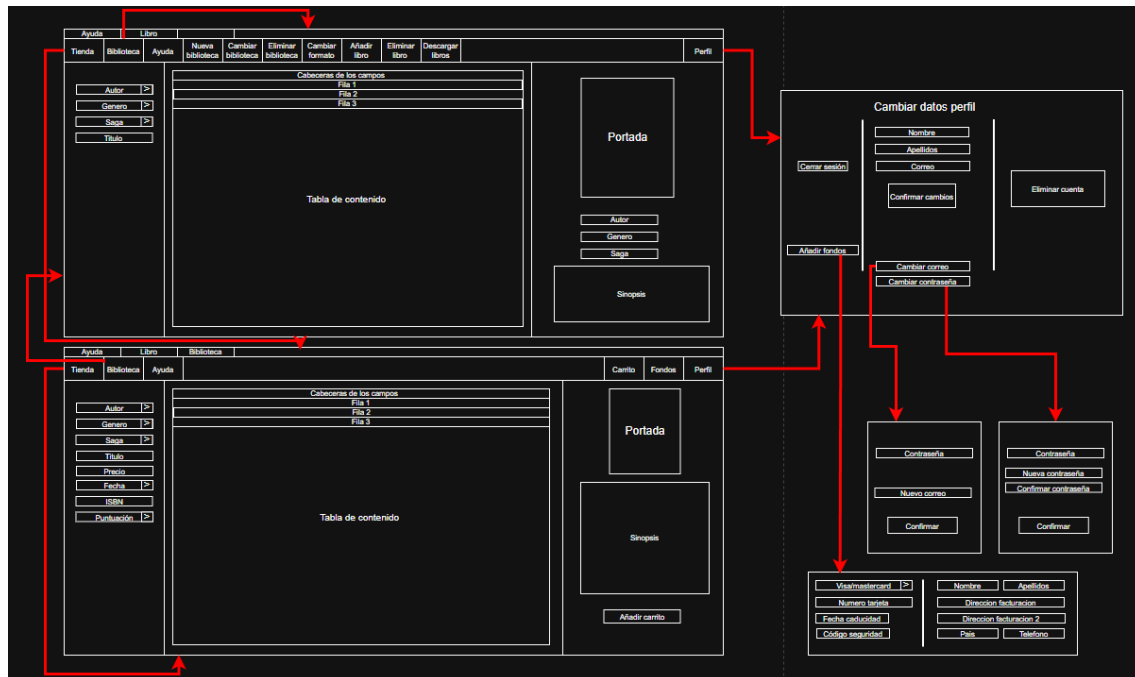
El login será lo primero que vea el usuario nada más abrir la aplicación en ambas versiones. Desde aquí se podrá navegar entre las distintas pestañas de registro y recuperar contraseña. Una vez el usuario haya insertado sus datos en la pestaña de login y suponiendo que todo sea correcto, pasara a la ventana principal.

Para reutilizar lo más posible el código, todo estará en un solo frame, y la navegación se hará con paneles. En el caso de la pestaña de recuperar contraseña, lo único visible al principio serán los campos de usuario, correo, código de verificación, y los botones de enviar código y confirmar código. En el momento que el usuario le dé al botón de confirmar código, se hagan todas las comprobaciones pertinentes y estas resulten correctas, revelara el botón de cambiar contraseña.

7.2. UI de la aplicación de escritorio

Debido a que la versión de escritorio está pensada también para ser usada por las editoriales, lo que vea el usuario nada más entrar cambiará entre la ventana de usuario y la de editorial. Este hecho de esta forma para que un usuario promedio no pueda acceder a la ventana de subir libros y ya que a una editorial no le interesa las funciones del usuario promedio, esta es redirigida a la ventana de editorial directamente. En principio no habrá navegación entre las dos y las editoriales tampoco podrán acceder a la tienda.

7.2.1. Vista usuario



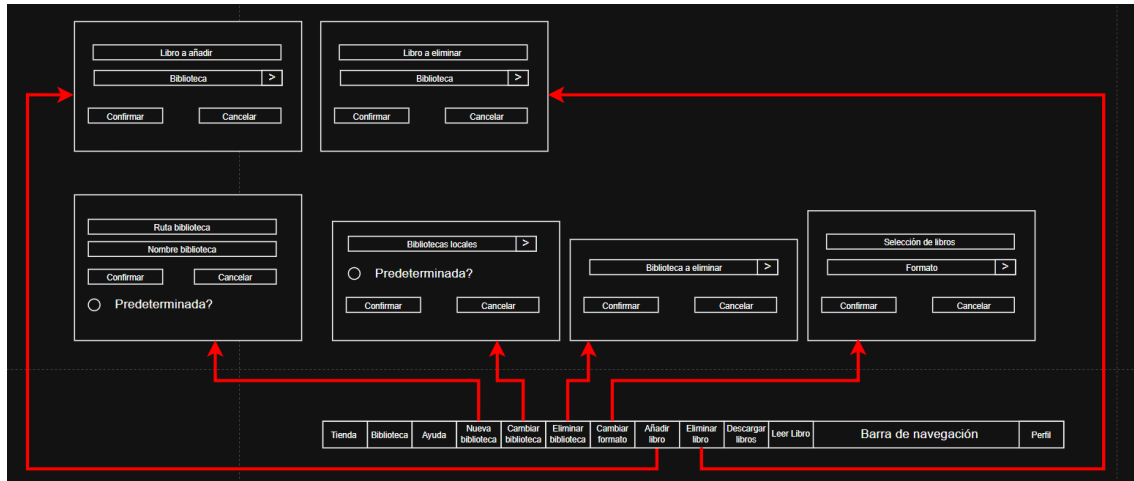
En la imagen se puede ver el flujo de las ventanas que de un usuario. La primera ventana que verá siempre nada más logearse será la de biblioteca, desde donde podrá acceder al resto de ventanas. Se ha diseñado pensando en utilizar un frame principal e ir cambiando entre la ventana de biblioteca y la de tienda. Ambas ventanas tendrán una barra de herramientas, desde donde podrán acceder a las opciones de sus respectivas ventanas.

La ventana de biblioteca estará dividida en tres secciones, filtros de búsqueda, tabla de contenido e información básica. En la sección de filtros se le permitirá al usuario buscar por género, autor, saga, y título. En la tabla de contenido se mostrarán todos los libros de los que dispone el usuario, así como los que tenga en la biblioteca local actualmente. El contenido de esta tabla se verá afectado por los filtros que se le apliquen y se podrá escoger que campos del libro se mostrarán. En el panel de información se mostrará el autor, la saga, el género, la portada y la sinopsis del libro seleccionado.

La ventana de tienda estará distribuida de forma similar a la ventana de biblioteca, contando esta también con tres secciones, filtros de búsqueda, tabla de contenido e información básica. La sección de búsqueda cuenta con los mismos filtros que la biblioteca, con el añadido de poder buscar por fecha de publicación, ISBN, Precio y editorial. La tabla de contenido también será prácticamente idéntica, excepto por el añadido de que se mostrará el precio y si hay un descuento. Por último, en el panel de

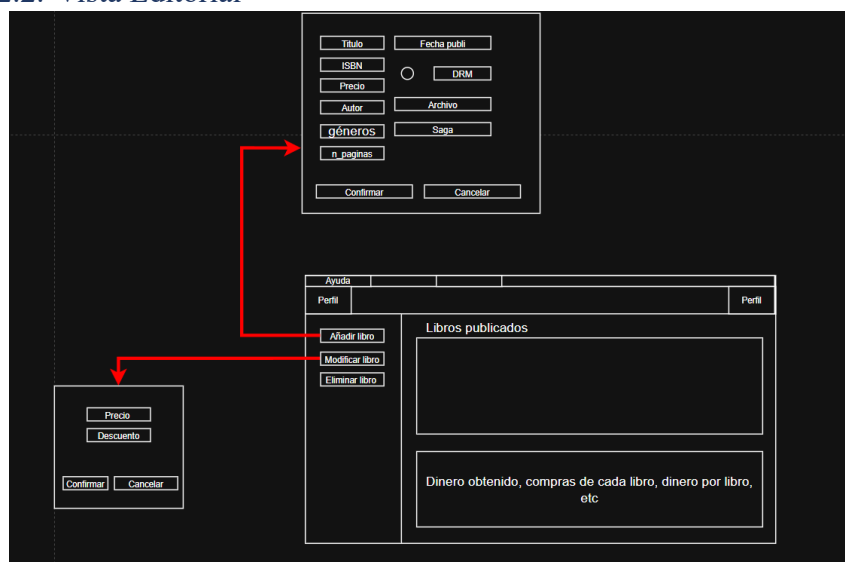
información se mostrará la portada del libro seleccionado y la sinopsis. También contará con el botón de añadir al carrito.

En la ventana de perfil se podrá cambiar directamente los datos como el nombre, los apellidos, la contraseña y el correo, aunque estos últimos abrirán un cuadro de dialogo para hacer comprobaciones antes de llevarse a cabo. También se podrá cerrar la sesión actual, abrir la ventana para añadir fondos que pedirá la información de la tarjeta a la que cobrar, y eliminar la cuenta por completo.



En la barra herramientas se podrá acceder a distintas opciones, como la navegación por la aplicación, añadir o eliminar un libro, crear una biblioteca o eliminarla, etc. Exceptuando tienda, biblioteca, perfil, ayuda y descargar libro abrirán su respectivo pop-up como aparece en la imagen. El botón de ayuda abrirá en el navegador la sección de ayuda de página web. El botón de “descargar libro” descargará todos los libros comprados que no tengamos ya descargados en la biblioteca local.

7.2.2. Vista Editorial



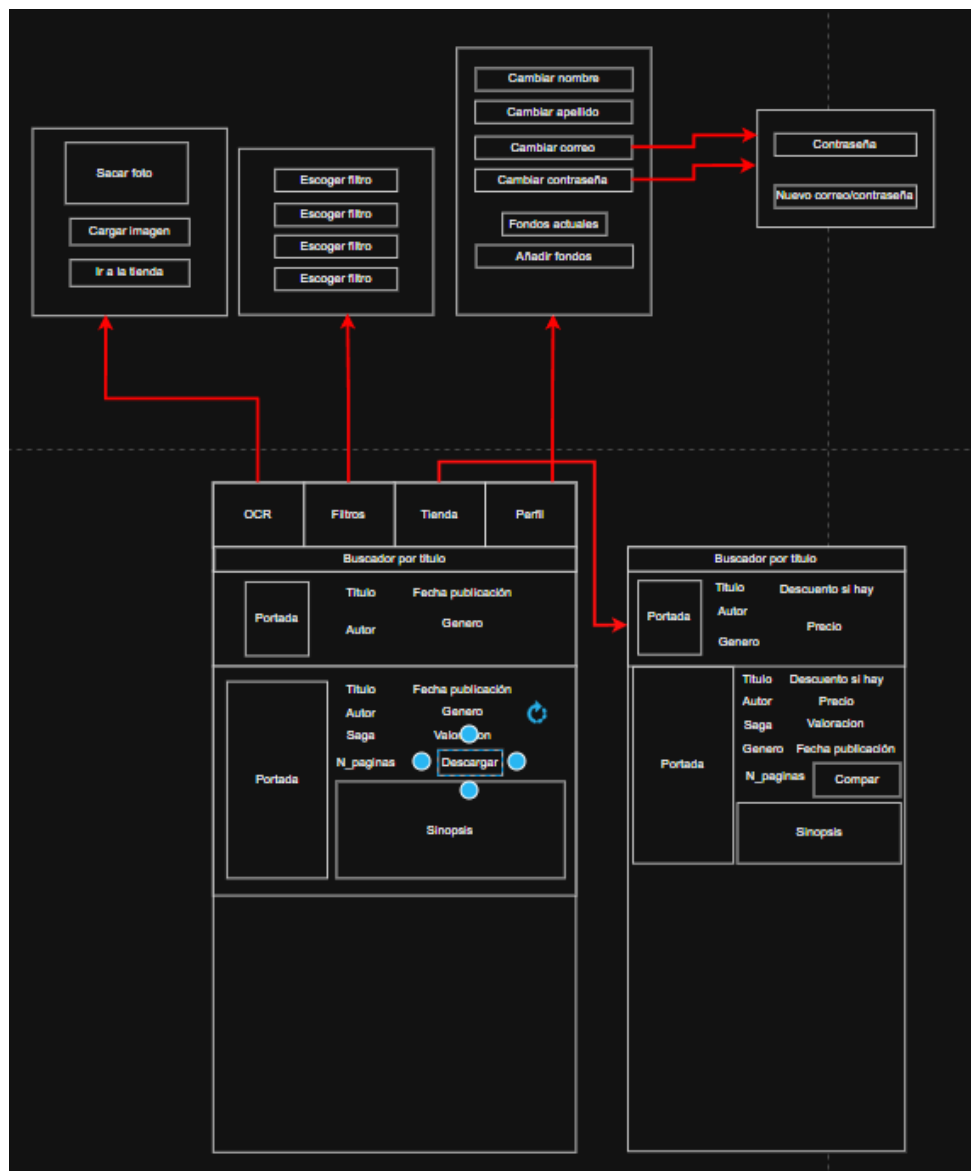
Ya que la editorial no es realmente un usuario que vaya a consumir la aplicación para otra cosa que no sean negocios, se le proporcionará una vista diferente a la del usuario común, dándole acceso a la capacidad de añadir libros con un formulario simple, ver una

métrica rápida de cuanto ha generado cada libro, y ver los libros publicados. Como se ha mencionado antes, las editoriales solo tendrán acceso a esta vista.

A diferencia de la vista de usuario, la vista de editorial está pensada para ser lo más concisa y rápida de usar ya que es lo que más les suele interesar a las empresas. En el apartado de las métricas, las editoriales podrán acceder dinero total ganado con la venta de libros, cuanto han ganado por libro, cuanto han ganado por mes, etc.

7.3. UI de la aplicación móvil

Para la versión móvil se utilizarán fragments y framelayout, ya que nos permitirá crear una interfaz bastante más flexible a la hora de hacer la navegación entre pantallas y nos permitirá aprovechar gran parte del código que tengamos.



Al igual que con la versión de escritorio, lo primero que verá el usuario nada más logearse será su biblioteca, desde donde tendrá acceso a la tienda, el perfil, y el OCR. Tanto la tienda como la biblioteca serán fragments y se crearán en la misma actividad, teniendo acceso ambos a la barra de navegación. Los botones de OCR como filtros abrirán una ventana modal que llevará a cabo sus respectivas tareas.

Desde el fragment de biblioteca, el usuario verá todos los libros que tiene comprados en la tienda. Tendrá una vista previa con la portada, el título, el autor y el género. Cuando pulse en esa vista previa, esta se expandirá, mostrando la información del libro en más detalle y el botón de descarga. También se dispondrá de un buscador, en el que se podrá escribir el título del libro deseado. Si se ha activado algún filtro en su ventana correspondiente también se verá reflejado en la lista de libros.

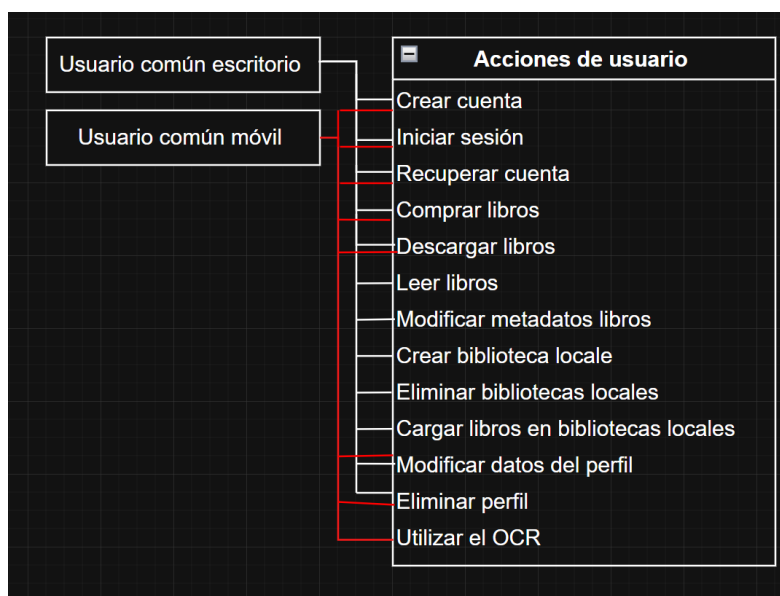
El fragment de tienda reutiliza la idea de la biblioteca y contara con la misma lista con información básica del libro que se expandirá en el momento que el usuario pulse sobre él. También contara con un buscador por título y la ventana de filtros. Una vez escogido el libro deseado se procederá a la compra.

En la ventana de perfil, al igual que en la versión de escritorio, se podrá cambiar la información de la cuenta, añadir fondos y eliminarla, abriendo cada uno su respectiva ventana.

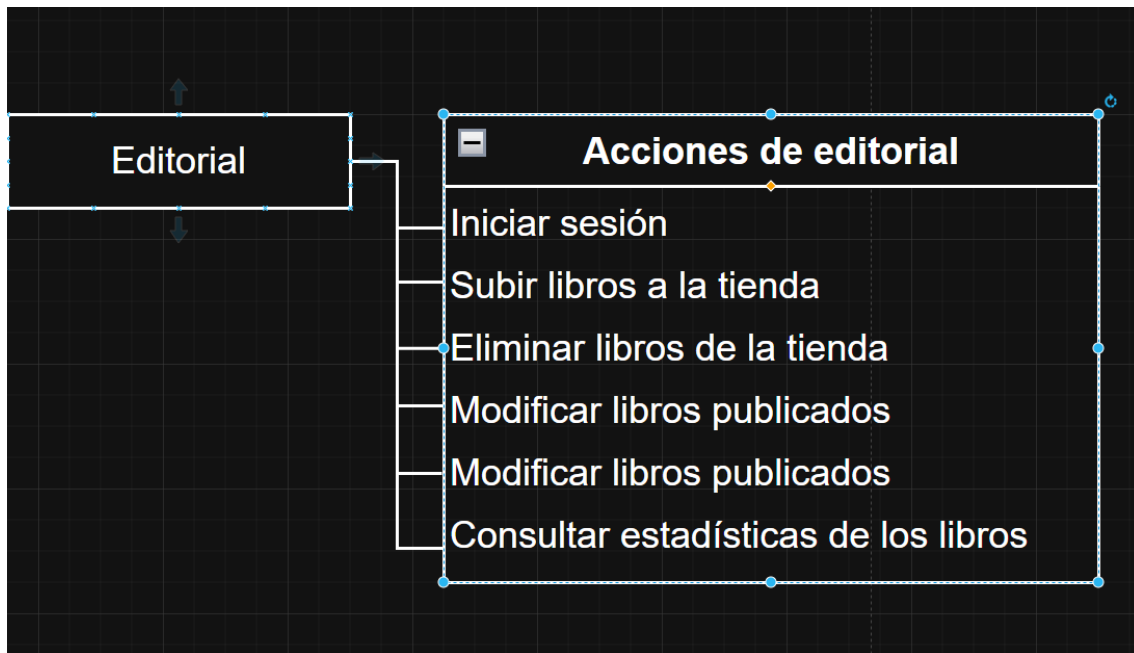
8. Casos de uso

La aplicación estará enfocada en dos tipos de usuarios distintos, las empresas o editoriales y los usuarios comunes, siendo estos últimos los usuarios a los que está enfocada principalmente y que más uso le darán. Ya que el proyecto consta de dos aplicaciones, se procederá a explicar que usuario tiene acceso a que y desde donde.

- **Usuario común:** Este usuario tendrá acceso tanto a la aplicación de escritorio como a la aplicación móvil.
 - **Escritorio:** Desde esta aplicación podrá administrar su cuenta, administrar tanto su biblioteca local como la virtual, podrá comprar libros y leer sus libros.
 - **Móvil:** En esta versión podrá administrar su cuenta, comprar libros, descargar los libros comprados, escanear imágenes de portadas para saber si el libro se encuentra en la tienda.



- **Editoriales:** Las editoriales solo cuenta con la versión de escritorio de la aplicación. Desde ella podrán subir a la tienda, eliminarlos, administrar sus precios, y consultar estadísticas de ventas.



9. Diagrama de clases

A continuación, se mostrará los diagramas UML de las versiones de escritorio, de móvil, y el servidor.

9.1. Versión de escritorio



9.1.1. App

Será la clase principal de la aplicación de escritorio. Es la encargada de crear la ventana donde se trabajará y cargar contenido en esta. También será la encargada de crear y manejar las ventanas modales.

9.1.2. ControllerLogin

Esta clase será la encargada de controlar tanto la interfaz como la funcionalidad que incumbran al login como pueda ser la creación de nuevas cuentas, y recuperación de contraseñas. Cuenta con métodos para verificar que el correo que se introduzca sea un

correo valido y que la contraseña tenga un cierto formato. También se encargará de autocompletar el campo de usuario si este lo desea, a la hora de iniciar sesión a través del archivo .properties.

9.1.3. ManejaImagen

Sera la clase encargada de gestionar las imágenes que utilice la aplicación. Mas específicamente, se encargará de redimensionar imágenes y cambiar el formato según sea necesario.

9.1.4. Properties

Esta clase será la encargada de acceder al archivo .properties y manejar la información que este contiene.

9.1.5. ManejaArchivos

Esta clase será la encargada de la gestión de las bibliotecas locales del usuario. Permitirá crear nuevas bibliotecas, eliminar las existentes y cambiar entre ellas. Las rutas y configuraciones se almacenarán en un archivo .properties, donde también se indicará cuál de ellas es la biblioteca predeterminada.

9.1.6. Ebook

Clase centrada en la gestión de los metadatos de los libros electrónicos. Utilizará la librería epublib para acceder a estos y extraer información relevante como título, autor y el número de páginas de aquellos libros que haya añadido el usuario a su biblioteca local.

9.1.7. ControllerBiblioteca

Esta clase será la encargada de manejar la interfaz gráfica referente a la biblioteca. También se encargará de manejar las ventanas de perfil, crear libro, etc. Estará ligada a un archivo .fxml, que será el que contenga toda la información de la interfaz.

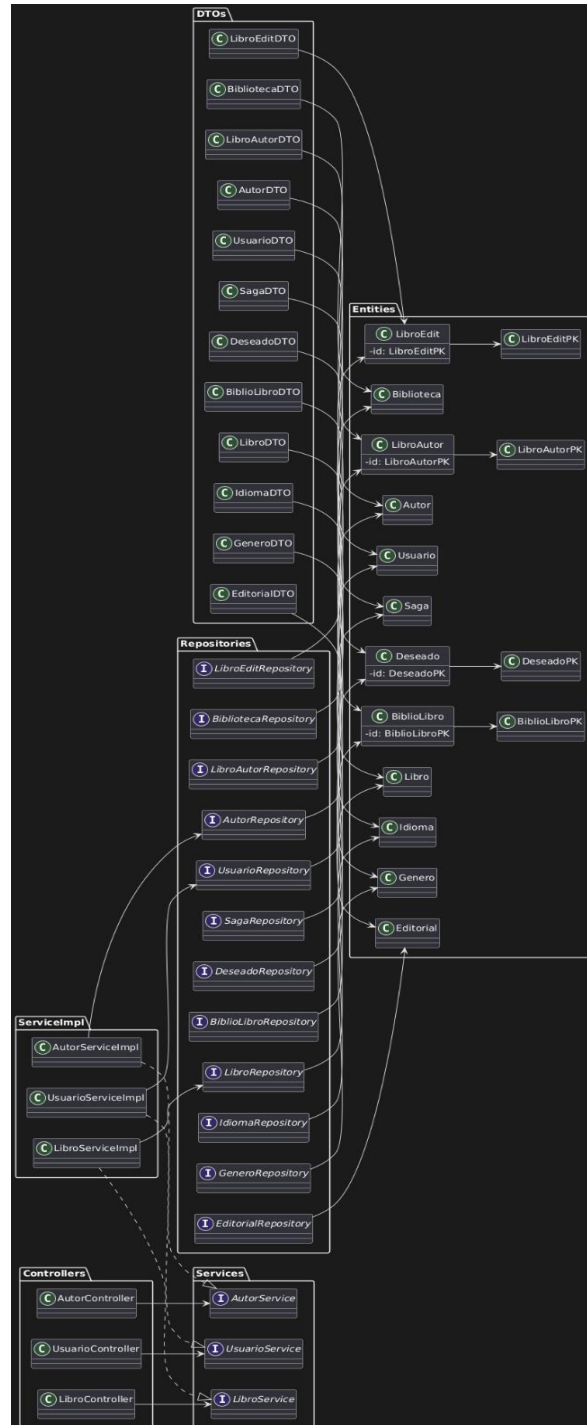
9.1.8. ControllerTienda

Esta clase será la encargada de manejar la interfaz gráfica referente a la tienda. Trabjará junto con la clase de conexión para hacer las peticiones a la API del servidor. Estará ligada a un archivo .fxml, que será el que contenga toda la información de la interfaz.

9.1.9. Conexión

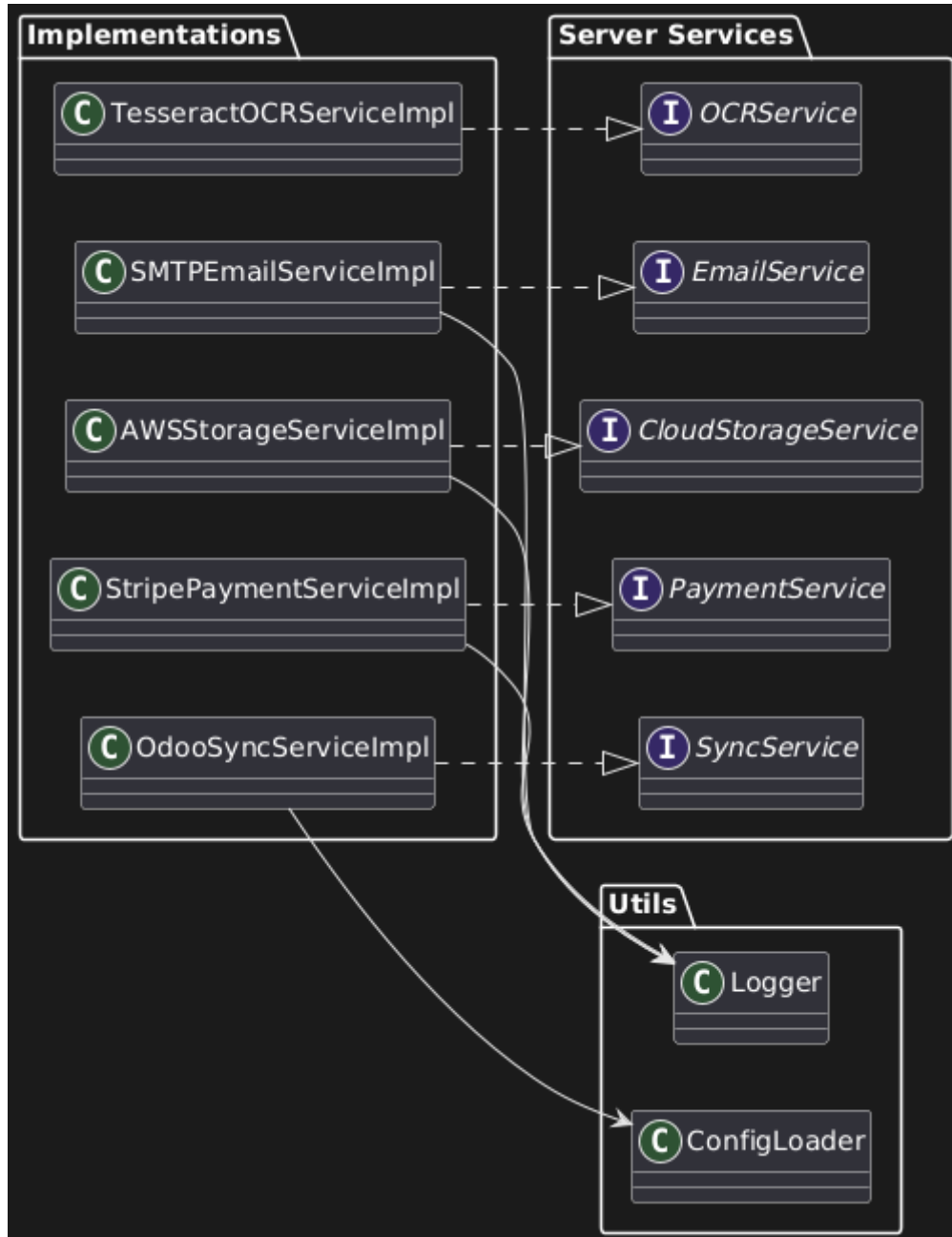
Clase responsable de establecer y gestionar la comunicación entre la aplicación de escritorio y el servidor. Esta clase será la encargada de consumir la API que conecta con la base de datos, enviando y recibiendo datos de forma segura. Implementará métodos para realizar operaciones como login, registro, sincronización de bibliotecas, descarga de libros o cualquier otra interacción entre cliente y servidor.

9.2. UML del API REST



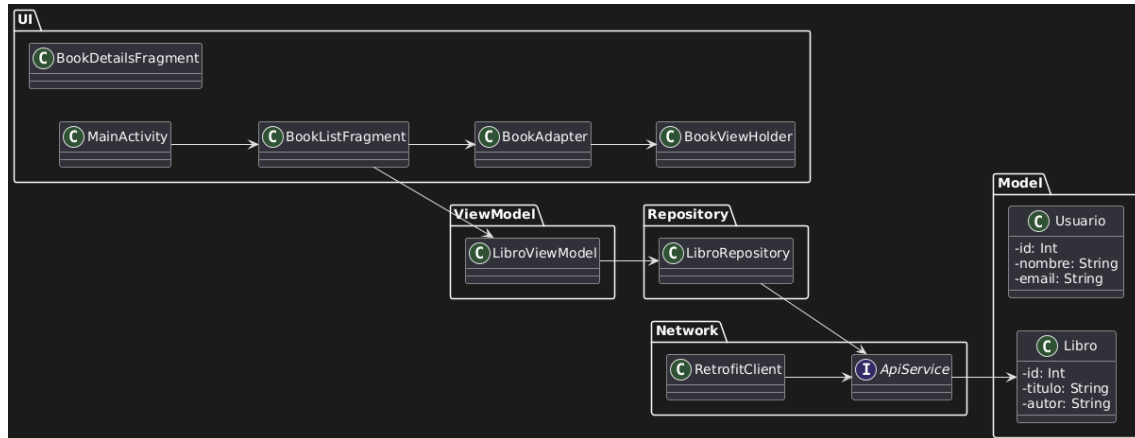
Este diagrama representa la arquitectura de una API RESTful construida con Spring Boot. Organiza el código en capas: modelos (entidades), DTOs, repositorios, servicios, implementaciones de servicio y controladores. Cada capa se comunica con la siguiente para estructurar una aplicación mantenible, clara y escalable, siguiendo el patrón MVC y principios SOLID.

9.3. UML del Servidor



Este diagrama muestra los componentes del backend que manejan lógica auxiliar o tareas externas, como integración con Odoo, reconocimiento de texto con Tesseract, pagos con Stripe, almacenamiento en la nube y envío de correos. Cada servicio tiene su interfaz y su clase de implementación correspondiente, siguiendo el patrón de inyección de dependencias y separación de responsabilidades.

9.4. UML de Android



Este diagrama describe la estructura de una aplicación móvil desarrollada en Kotlin usando arquitectura MVVM. Incluye capas de UI (Activities, Fragments, Adapters), ViewModels que gestionan el estado, repositorios que acceden a los datos, y servicios de red configurados con Retrofit. Permite una separación clara de lógica de negocio y presentación, facilitando el testing y la escalabilidad.

10. Planificación

La planificación es clave para llevar el proyecto a buen puerto, ya que nos permite organizar las tareas y tener un camino claro hacia los objetivos. Al dividir el trabajo en fases, con un enfoque flexible que permita trabajar en paralelo en algunos casos, buscamos optimizar el tiempo y la colaboración.

10.1. Diagrama de Gantt

En este cronograma se agrupan las actividades principales del proyecto, estableciendo su secuencia y considerando tareas que podemos ejecutar simultáneamente. Cada fase es importante, y la idea es avanzar de manera iterativa para adaptarnos a cualquier cambio.

Fases del proyecto:

1. **Montaje y prueba de la base de datos en entorno local:** Asegurarnos de que la base de datos funcione correctamente en un entorno controlado antes de pasar a la siguiente fase.
2. **Desarrollo de la API RESTful con Spring Boot y PostgreSQL:** Aquí empezamos con el backend, utilizando tecnologías que nos permitan manejar solicitudes y respuestas de forma eficiente.
3. **División de tareas:**
 - Un integrante se encargará del **servidor** (backend).
 - Otro se centrará en el **desarrollo de la aplicación de escritorio**.
4. **Implementación del OCR con Tesseract y la pasarela de pagos Stripe:** Incluir el reconocimiento de texto y las opciones de pago en el sistema.

5. **Desarrollo de la aplicación móvil con Kotlin para Android:** Desarrollar la versión móvil, donde Kotlin será clave para la interfaz y el comportamiento de la app.
 6. **Integración con Odoo para la gestión empresarial:** Conectar nuestro sistema con Odoo para optimizar procesos internos.
 7. **Migración y despliegue en un entorno cloud (Render, AWS, etc.):** Llevar todo a la nube para un mejor rendimiento y escalabilidad.
 8. **Pruebas de calidad, documentación final y despliegue:** Finalmente, realizar pruebas, documentar el trabajo y poner todo en producción.
-

10.2. Definición de Recursos y Logística

Para cada fase, hemos identificado los recursos necesarios, asegurándonos de que todo esté listo para llevar a cabo el desarrollo de manera efectiva.

Recursos materiales:

- **Ordenadores personales:** Para trabajar en el desarrollo de backend y frontend. Con un entorno adecuado de desarrollo Java (NetBeans o IntelliJ).
- **Dispositivos móviles Android:** Para realizar pruebas durante el desarrollo de la app móvil.
- **Conexión a internet:** Esencial para pruebas, descargas de dependencias y despliegues.
- **Hosting en la nube (Render, AWS):** Para la fase final de despliegue en producción.

Recursos tecnológicos:

- **Spring Boot, PostgreSQL y Lombok:** Para construir la API de backend.
- **Tesseract OCR:** Para la funcionalidad de reconocimiento de texto.
- **Stripe API:** Para las transacciones de pago seguras.
- **JavaFX/Scene builder:** Para el desarrollo de la aplicación de escritorio.
- **Kotlin + Android Studio:** Para la aplicación móvil en Android.
- **Odoo:** Para la integración con el sistema de gestión empresarial.
- **Servicios cloud (Render, AWS):** Para el despliegue final.

Recursos humanos:

- **Desarrollo colaborativo:** Un integrante se dedicará al backend, mientras que el otro se encargará de las interfaces (escritorio y móvil).
- Ambos trabajarán juntos en pruebas, despliegue y documentación final.

Logística:

- **Control de versiones (GitHub/GitLab):** Para mantener el código sincronizado.
- **Reuniones periódicas:** Para revisar el progreso, identificar obstáculos y ajustar el plan.
- **Herramientas colaborativas (Trello, Notion):** Para gestionar las tareas y asegurarnos de que todo esté en su lugar.

Documentación centralizada: Para evitar errores de integración y asegurar que todo el equipo esté alineado.

11. Implementación

11.1. Aplicación de escritorio

11.1.1. Introducción

Para la aplicación de escritorio se ha utilizado javafx junto con scene builder para crear los archivos .fxml que darán forma a la interfaz. Para la lógica se ha utilizado java junto con NetBeans.

11.1.2. Clase Properties

Esta clase es la encargada de darle acceso a las demás clases al archivo .properties, que es donde se guardan cosas como si el usuario quiere que se le recuerde, los paths a las distintas librerías, etc. Para ello cuenta con métodos genéricos para eliminar, añadir y modificar las propiedades pasándole una clave o un valor donde sea requerido, lo que los hace reutilizables en varias secciones de código.

```
BibliotecaPredeterminada=C:\Users\Andres\Documents\NetBeansProjects\TFG\src\main\resources\com\mycompany\tfg\bibliotecas\Colmado
PathLibrary_Colmado=C:\Users\Andres\Documents\NetBeansProjects\TFG\src\main\resources\com\mycompany\tfg\bibliotecas\Colmado
Recorder=0
Usuario=Eva-01
```

```
//obtener una propiedad
public static String getProperty(String property){
    try (InputStream input = new FileInputStream(name: "src\main\resources\com\mycompany\tfg\conf.properties")) {
        // Cargar el archivo de propiedades
        PROPERTIES.load(input);
        String prop=PROPERTIES.getProperty(key:property);
        input.close();
        return prop;
    } catch (IOException ex) {
        ex.printStackTrace();
        return "";
    }
}

//Eliminar una propiedad
public static void removeProperty(String key){
    try (InputStream input = new FileInputStream(name: "src\main\resources\com\mycompany\tfg\conf.properties")) {
        // Cargar el archivo de propiedades
        PROPERTIES.load(input);
        PROPERTIES.remove(key);
        OutputStream output=new FileOutputStream(name: "src\main\resources\com\mycompany\tfg\conf.properties");
        PROPERTIES.store(out:output, comments: null);
        output.close();
        input.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

//guardar una propiedad
public static void setProperty(String property,String value){
    try (InputStream input = new FileInputStream(name: "src\main\resources\com\mycompany\tfg\conf.properties")) {
        PROPERTIES.load(input);
        PROPERTIES.setProperty(key:property, value);
        OutputStream output=new FileOutputStream(name: "src\main\resources\com\mycompany\tfg\conf.properties");
        PROPERTIES.store(out:output, comments: null);
        output.close();
        input.close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
```

Como se puede ver en la imagen, estos métodos acceden a un archivo .properties que va incluido en el proyecto. Para acceder a él, primero hay que crear un InputStream para que el programa sea capaz de leer el archivo y utilizando la clase Properties de java se carga el archivo en una variable. En caso de que queramos escribir en el archivo también tendremos que crear un OutputStream. Como se puede apreciar en el ejemplo, estos métodos pares clave-valor para acceder a las propiedades, ya sea para obtenerlas, eliminarlas, crearlas, o modificarlas.

También cuenta con métodos específicos cuando son requeridos por alguna clase. En estos casos, estos métodos son utilizados para sacar todas las bibliotecas que ha creado el usuario y para saber si una biblioteca en concreto es la predeterminada.

```
//obtener todas la bibliotecas creadas por el usuario
public static ArrayList<String> getLibraries() {
    try (InputStream input = new FileInputStream(name: "src\\main\\resources\\com\\mycompany\\tfg\\conf.properties")) {
        // Cargar el archivo de propiedades
        PROPERTIES.load(input);
        Enumeration<Object> keys = PROPERTIES.keys();
        ArrayList<String> claves=new ArrayList<>();
        while (keys.hasMoreElements()) {
            Object key=keys.nextElement();
            if(key.toString().contains(": PathLibrary_")){
                claves.add(key.toString());
            }
        }
        input.close();
        return claves;
    } catch (IOException ex) {
        ex.printStackTrace();
        return null;
    }
}

Properties.setProperty("PathLibrary_"+library.getName(), value: library.getAbsolutePath());
```

Para obtener todas las bibliotecas, lo primero es obtener todas las claves del archivo, a partir de ahí, se buscan las claves que contengan “PathLibrary_” ya que es algo que se pone de forma “manual” cada vez que se crea una biblioteca. Cuando se encuentre una coincidencia, se añade a un ArrayList que será lo que devuelva el método.

```
//comprueba si la biblioteca que se le pasa es la predeterminada
public static boolean isDefault(String property) {
    try (InputStream input = new FileInputStream(name: "src\\main\\resources\\com\\mycompany\\tfg\\conf.properties")) {
        // Cargar el archivo de propiedades
        PROPERTIES.load(input);
        File f=new File(pathname: PROPERTIES.getProperty(key: "BibliotecaPredeterminada"));
        input.close();
        if(f.getName().equals(property)){
            return true;
        } else{
            return false;
        }
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return false;
}
```

```
BibliotecaPredeterminada=C:\\Users\\Andres\\Documents\\NetBeansProjects\\TFG\\src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\Colmado
```

Para saber si una biblioteca está guardada como predeterminada, primero se obtiene la propiedad correspondiente usando el argumento property que pide el método. Este argumento será el nombre de la biblioteca a eliminar. Posteriormente se crea una variable tipo File ya que la propiedad es un path, y usando el método getName se comprueba si el nombre proporcionado y el nombre guardado de las bibliotecas coincide.

Dependiendo del resultado, se devuelve uno u otro. En caso de que el método falle devolverá false para evitar problemas.

11.1.3. Clase ManejaArchivos

Esta clase será la encargada de interactuar con directorios, ya sea creándolos, eliminándolos o modificándolos. También será la encargada de manejar las propiedades del archivo .properties relacionadas con las rutas de las bibliotecas y los libros.

```
//Obtener todas las bibliotecas locales
public static File[] getLibraries() {
    ArrayList<String> keys=Properties.getLibraries();
    File[] paths=new File[keys.size()];
    for(int i=0;i<paths.length;i++){
        paths[i]=new File(pathname: Properties.getProperty(property: keys.get(index: i)));
    }
    return paths;
}

//obtener todos los libros de una biblioteca
public static List<File> getBooks(File library){
    ArrayList<File> subdirectories=new ArrayList<>();
    File[] children=library.listFiles();
    for (File asd:children){
        if(asd.isDirectory()){
            subdirectories.add(e: asd);
        }
    }
    return subdirectories;
}

//Crear una nueva biblioteca mientras no exista una con ese nombre y guardarlo en el archivo properties
public static void createLibrary(File library){
    if(!library.exists()){
        library.mkdir();
        Properties.setProperty("PathLibrary_"+library.getName(), value: library.getAbsolutePath());
    }
}

//Crear el directorio donde albergar el libro
public static File createBook(String library,String Book){
    File f=new File(Properties.getProperty("PathLibrary_"+library)+"\\"+Book);
    if (!f.exists()){
        f.mkdir();
        return f;
    } else {
        return null;
    }
}
```

Cuenta con métodos para obtener las bibliotecas creadas por el usuario, obtener los libros de una biblioteca concreta, crear una biblioteca, crear el directorio donde se va a guardar el libro, etc. Aquí se puede ver en uso la clase Properties, por ejemplo, en el método createLibrary, para crear una nueva entrada guardando en la clave el nombre junto con "PathLibray_" y en el valor la ruta absoluta que apunta a esa librería.

```

//metodo recursivo para eliminar una carpeta y todo lo que contenga
private static void matadorRecursoivo(File biblioteca){
    //se obtienen los hijos
    File[] hijos=biblioteca.listFiles();
    //se comprueba si tiene hijos
    if(hijos!=null){
        //se recorren los hijos
        for(File hijo:hijos){
            //si el hijo es un directorio se llama a si mismo y pasa el hijo, si no se elimina al hijo
            if (hijo.isDirectory()) {
                matadorRecursoivo(biblioteca: hijo);
            } else {
                hijo.delete();
            }
        }
    }
    //se elimina la biblioteca
    biblioteca.delete();
}

//metodo para eliminar un libro de una biblioteca concreta
public static void removeBook(String library,String book){
    //se obtienen los hijos de la biblioteca a eliminar
    File f=new File(Properties.getProperty("PathLibrary_"+library)+"\\"+book);
    matadorRecursoivo(biblioteca: f);
    //se borra la biblioteca
    f.delete();
}

//hasta aqui funcionan todos

//Eliminar una biblioteca con todo lo que contiene y el path del archivo properties
//Comprueba si es la biblioteca predeterminada y si lo es la sustituye por la biblioteca predeterminada de la aplicacion
public static void remobeLibrary(File biblioteca){
    if(Properties.isDefault(property: biblioteca.getName())){
        Properties.setProperty(property: "BibliotecaPredeterminada", value: "src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\MiBiblioteca");
    }
    Properties.removeProperty("PathLibrary_"+biblioteca.getName());
    matadorRecursoivo(biblioteca);
}

```

Esta clase también cuenta con métodos para eliminar tanto los libros como las bibliotecas. Para llevarlo a cabo se usan dos métodos, uno específico para eliminar un libro o una biblioteca entera y un método recursivo para eliminar todo el contenido. Es necesario hacer la distinción a la hora de eliminar ya que el procedimiento es distinto para cada uno y no serviría llamar directamente al método recursivo.

```

//metodo para eliminar un libro de una biblioteca concreta
public static void removeBook(String library,String book){
    //se obtienen los hijos de la biblioteca a eliminar
    File f=new File(Properties.getProperty("PathLibrary_"+library)+"\\"+book);
    matadorRecursoivo(biblioteca: f);
}

```

En el caso de eliminar un libro, primero se tiene que obtener la biblioteca donde esta guardado. Para ello se pide como argumento el nombre de la biblioteca y el del libro a eliminar. Se obtiene el path absoluto de la biblioteca y se concatena con el nombre del libro para sacar el hijo donde esta guardado y por último se llama al método recursivo.

```

//Eliminar una biblioteca con todo lo que contiene y el path del archivo properties
//Comprueba si es la biblioteca predeterminada y si lo es la sustituye por la biblioteca predeterminada de la aplicacion
public static void remobeLibrary(File biblioteca){
    if(Properties.isDefault(property: biblioteca.getName())){
        Properties.setProperty(property: "BibliotecaPredeterminada", value: "src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\MiBiblioteca");
    }
    Properties.removeProperty("PathLibrary_"+biblioteca.getName());
    matadorRecursoivo(biblioteca);
}

```

Por otra parte, para eliminar una biblioteca solo se necesita el nombre de la esta. Primero se comprueba si la biblioteca esta guardada como predeterminada y en caso afirmativo se sustituye por la ruta de la biblioteca predeterminada viene al principio. Por último, se obtiene el path absoluta de la biblioteca y se llama al método recursivo. Hay que tener en cuenta que esto no solo elimina la biblioteca, sino que también elimina todos los libros y demás archivos que contenga esta.

```
//metodo recursivo para eliminar una carpeta y todo lo que contenga
private static void matadorRecurso(File biblioteca){
    //se obtienen los hijos
    File[] hijos=biblioteca.listFiles();
    //se comprueba si tiene hijos
    if(hijos!=null){
        //se recorren los hijos
        for(File hijo:hijos){
            //si el hijo es un directorio se llama a si mismo y pasa el hijo, si no se elimina al hijo
            if (hijo.isDirectory()) {
                matadorRecurso(biblioteca: hijo);
            } else {
                hijo.delete();
            }
        }
    }
    //se elimina la biblioteca
    biblioteca.delete();
}
```

El método recursivo empieza por obtener todos los hijos del directorio que se le pase y se guarda en un Array de tipo File. Se comprueba si es nulo y en caso de que no lo sea, se procede a recorrer con un for el array de hijos. En caso de que el hijo sea un directorio, se llama a si mismo pasando la ruta de ese hijo, si por el contrario es un archivo, se elimina directamente. Por último, se elimina el path original que recibe el método.

11.1.4. Login

```
public class App extends Application {
    private static Scene scene;
    private static Stage primaryStage;
    private static Stage modalStage;
    private static Scene modalScene;

    @Override
    public void start(Stage stage) throws IOException {
        primaryStage=stage;
        scene = new Scene(parent: loadFXML(fxml: "Perfil"));
        stage.setScene(scene);
        stage.setTitle(string: "Dustless");
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(parent: loadFXML(fxml));
    }

    //obtener el stage principal
    public static Stage getPrimaryStage(){
        return primaryStage;
    }

    //obtener un stage modal
    public static Stage getModalStage(){
        return modalStage;
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(url:App.class.getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }

    public static void main(String[] args) {
        launch();
    }
}
```

Para poder usar javafx en la aplicación la clase principal debe extender la clase Application. Esta clase se genera de forma automática con los métodos de start, setRoot, loadFXML y main desde NetBeans al generar un proyecto de Maven de javafx.

Se le ha añadido a la clase métodos para recuperar el stage principal, que será la ventana principal de la aplicación, el stage modal, que será la ventana que se abra cada vez que se abra una ventana modal y el método para crear el stage modal.

```
//crear una ventana modal
public static void launchModal(String ventana) throws IOException{
    //se crea un stage y un scene
    modalStage=new Stage();
    modalScene=new Scene(parent: loadFXML(fxml: ventana));
    //se le asigna el scene al stage
    modalStage.setScene(scene: modalScene);
    //se le asigna el padre al stage y se le hace que sea modal
    modalStage.initOwner(window: primaryStage);
    modalStage.initModality(modlty: Modality.WINDOW_MODAL);
    modalStage.show();
}
```

Para crear la ventana modal se crea un nuevo Stage y un nuevo Scene, (que es donde se guarda todo el contenido de la ventana), al que se le carga el archivo .fxml que se le pasa como parámetro a través del método loadFXML. Se le asigna el scene, el padre y la modalidad al stage y se muestra.

Para implementar la lógica a la interfaz gráfica hecha con javafx es necesario crear una clase controlador. Se ha creado una clase controlador emparejará con su respectivo archivo .fxml para cada vista de la aplicación. Para hacer la clase controlador simplemente hay que asignarle el controlador desde SceneBuilder o asignárselo directamente al contenedor que esté más alto en la jerarquía en el archivo .fxml.

Controller class

com.mycompany.tfg.ControllerLogin

```
<VBox prefHeight="800.0" prefWidth="400.0" xmlns="http://javafx.com/javafx/23.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="com.mycompany.tfg.ControllerLogin">
```

```
public class ControllerLogin {
    @FXML private StackPane stackPaneLogin;
    @FXML private Pane paneLogin;
    @FXML private Pane paneNewPass;
    @FXML private Button btnSendCode;
    @FXML private Button btnSendPass;
    @FXML private Button btnSendMail;
    @FXML private CheckBox checkBoxRemember;
    @FXML private TextField textNewUser;
    @FXML private Label lblBadCode;
    @FXML private Label lblBadPass;
    @FXML private Label lblPassMatch;
    @FXML private Label lblPassMismatch;
    @FXML private PasswordField textNewUserPass;
    @FXML private PasswordField textNewUserConfPass;
    @FXML private TextField textNewUserMail;
    @FXML private Text textBadPass;
    @FXML private Label lblBadMail;
    //queda por hacer metodo para mandar hasseada la pass

    //Este metodo se ejecuta al cargar los componentes de forma automatica.
    //El metodo comprueba si el usuario dejo marcado el boton de recordar usuario en su ultima sesion y lo escribe de forma automatica en el campo correspondiente
    @FXML
    private void initialize(){
        if(Integer.parseInt(com.mycompany.tfg.Properties.getProperty("Recordar"))!=1){
            checkBoxRemember.setSelected(false);
            textNewUser.setText(com.mycompany.tfg.Properties.getProperty("Usuario"));
        }

        //este metodo comprueba si el usuario quiere guardar el usuario o no
        @FXML
        private void checkRem(){
            if(!checkBoxRemember.isSelected()){
                com.mycompany.tfg.Properties.setProperty("Recordar", value: "0");
            } else{
                com.mycompany.tfg.Properties.setProperty("Recordar", value: "1");
            }
        }
    }
}
```

Para poder acceder a los objetos de javafx creados en el archivo .fxml se utilizará la anotación @FXML en la declaración del objeto y en la creación de los métodos, como

pueda ser el que se ejecuta al hacer click en un botón. Hay que tener en cuenta que si se le asigna uno de estos métodos a un control y no se implementa saltará una excepción.

```
//Comprueba si los datos proporcionados por el usuario son validos antes de mandarlos a la db
private Boolean validateNewUser() {
    boolean pass=false;
    boolean mail=false;
    //Comprueba si las contraseñas coinciden
    if(matchPass(pass1: textfNewUserConPass.getText(),pass2: textfNewUserPass.getText())){
        //comprueba si la contraseña esta bien formateada
        if(!validPass(pass: textfNewUserPass.getText())){
            textfNewUserPass.setText(string: "");
        } else{
            textBadPass.setVisible(bln:false);
            pass=true;
        }
        lblPassMatch2.setVisible(bln:false);
    } else{
        textfNewUserPass.setText(string: "");
        textfNewUserConPass.setText(string: "");
        lblPassMatch2.setVisible(bln:true);
        textBadPass.setVisible(bln:true);
    }
    //comprueba si el correo esta bien formateado
    if(!validEmail(email: textNewUserMail.getText())){
        textNewUserMail.setText(string: "");
        lblBadMail.setVisible(bln:true);
    } else{
        lblBadMail.setVisible(bln:false);
        mail=true;
    }
    if(mail&&pass){
        return true;
    } else{
        return false;
    }
}
```

```
//valida el correo
private Boolean validEmail(String email){
    Pattern pattern=Pattern.compile(regex: "^[\\w.-]+@[\\w.-]+\\. [a-zA-Z]{2,}$");
    Matcher matcher=pattern.matcher(input: email);
    return matcher.matches();
}

//valida la contraseña
private Boolean validPass(String pass){
    Pattern pattern=Pattern.compile(regex: "(?=.*[A-Z]) (?=.*[a-z]) (?=.*\\d) (?=.*[!@#$%^&]) [A-Za-z\\d@!%*?&]{12,}$");
    Matcher matcher=pattern.matcher(input: pass);
    return matcher.matches();
}

//valida que la ambas contraseñas coincidan
private Boolean matchPass(String pass1, String pass2){
    if(pass1.equals(input: pass2)){
        return true;
    } else{
        return false;
    }
}
```

También se han creado métodos que no se ejecutarán en base a una acción sobre un control, sino cuando sean llamados. Este método en concreto valida si los datos introducidos por un usuario a la hora de crear una cuenta son correctos. Para ello comprueba dos cosas principalmente, el correo y las contraseñas.

Para comprobar el correo coge el texto del TextField correspondiente y se lo pasa a un método que utilizando una expresión regular buscara si se cumple un formato y devuelve el resultado. En caso de que el correo sea válido se guardará en una variable de tipo booleana la confirmación, en caso negativo, se mostraran los mensajes de error correspondientes.

Para validar la contraseña primero se comprueba que ambas contraseñas coincidan, en caso negativo se mostraran los mensajes de error, en caso afirmativo procederá a la siguiente comprobación. Ahora se comprueba que la contraseña tenga una mayúscula, una minúscula, un número, un carácter especial y doce caracteres mínimo. Al igual que con el correo se guardará el resultado en una variable.

Por último, se comprueba que ambas variables sean verdaderas, confirmando así que ambos datos están validados y se devuelve la respuesta.



El controlador del login también cuenta con métodos para manejar el autocompletar del campo usuario a la hora de iniciar sesión. Lo primero es comprobar si el usuario marcó la opción la última vez o no. Para ello se comprueba la propiedad “Recordar” del archivo .properties. En caso de que dejase marcado en el último inicio de sesión el botón “Recuérdame”, utilizando el método initialize (que es un método de javafx que se ejecuta al cargarse el archivo) se deja otra vez marcado el botón y se escribe en el TextField el usuario que se tenga guardado en el archivo .properties.

```
//Este metodo se ejecuta al cargar los componentes de forma automatica.  
//El metodo comprueba si el usuario dejo marcado el boton de recordar usuario en su ultima sesion y lo escribe de forma automatica en el campo correspondiente  
@FXML  
private void initialize() {  
    if(Integer.parseInt(com.mycompany.tfg.Properties.getProperty(property: "Recordar"))==1){  
        checkBoxRemember.setSelected(true);  
        textUser.setText(texting: com.mycompany.tfg.Properties.getProperty(property: "Usuario"));  
    }  
}
```



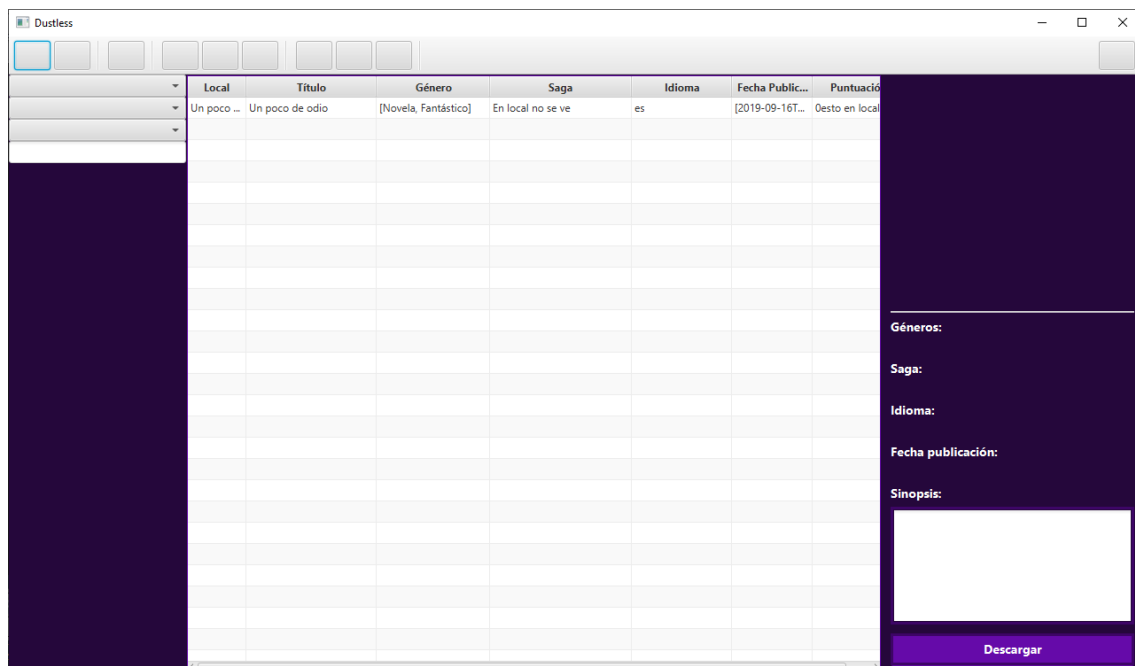
```
# 1=si 0=no
Recordar=0
Usuario=Eva-01
```

Por otra parte, cuando el usuario presiona el botón “Recordarme” este comprobará su estado, y dependiendo de si esta seleccionado o no guardara un resultado u otro el archivo .properties. Por último, cuando el usuario completa el inicio de sesión, se guarda el usuario que ha usado en la propiedad Usuario para la próxima vez.

```
//este metodo comprueba si el usuario quiere guardar el usuario o no
@FXML
private void checkRem(){
    if(!chboxRemeber.isSelected()){
        com.mycompany.tfg.Properties.setProperty(property: "Recordar", value: "0");
    } else{
        com.mycompany.tfg.Properties.setProperty(property: "Recordar", value: "1");
    }
}
```

11.1.5. Biblioteca

Una vez el usuario inicie sesión y se hagan las comprobaciones pertinentes pasará a la ventana de biblioteca desde donde se dará uso a la mayoría de los métodos de la clase ManejaArchvio. También contará con métodos propios para cargar el contenido de la biblioteca y mostrárselo al usuario, abrir las distintas ventanas modales y cargar la ventana de tienda, que será prácticamente idéntica a estas.



Nada más entrar, lo primero que se ejecutará será el método initialize que configurará la forma en la que reciben datos las columnas de la tabla central y cargará los datos iniciales en esta.

Ya que la aplicación cuenta con la biblioteca en la base de datos, que contienen los libros que ha comprado el usuario en la tienda, y las diversas bibliotecas locales, hay que tenerlo en cuenta a la hora de manejar el contenido que se muestra. Para ello se ha

decidido tratar todos los datos que se vayan a mostrar como String para evitar posibles fallos.

```

/*
Hace la carga de la tabla con todos los libros de una biblioteca cogiendo los datos de los metadatos del libro si es local o de la DB si es de la tienda
falta por implementar la db
*/
public void loadInitialTable(){
    List<File> books = ManejaArchivos.getBooks(library:bibliotecaSeleccionada);
    ObservableList<Map<String,Object>> ol=FXCollections.observableArrayList();
    Map<String,Object>[] libros=new HashMap[books.size()];
    for(int i=0;i<books.size();i++){
        //aquí se comprobará si los libros son locales o de la db
        libros[i]=new HashMap<>();
        libros[i].put(key:"Local", value: books.get(index: i).getName());
        libros[i].put(key:"Titulo",value: Ebooks.getTitulo(new File(books.get(index: i).getName()+".epub")));
        libros[i].put(key:"Genero",value: Ebooks.getGeneros(new File(books.get(index: i).getName()+".epub")));
        libros[i].put(key:"Saga", value: "En local no se ve");
        libros[i].put(key:"Idioma", value: Ebooks.getLanguage(new File(books.get(index: i).getName()+".epub")));
        libros[i].put(key:"Fecha", value: Ebooks.getDate(new File(books.get(index: i).getName()+".epub")));
        libros[i].put(key:"Puntuacion", i+"esto en local no se ve");
        libros[i].put(key:"Autor",value: Ebooks.getAutor(new File(books.get(index: i).getName()+".epub")));
        libros[i].put(key:"Sinopsis",value: Ebooks.getSinopsis(new File(books.get(index: i).getName()+".epub")));
        libros[i].put(key:"Path", value: new File(books.get(index: i).getName()+".jpg").toString());
    }
    ol.addAll(ol);
    tableCentral.setItems(ol);
    itemsOriginales=tableCentral.getItems();
}

```

El método initialize utilizará el método loadInitialTable para cargar la tabla por primera vez. Este método obtiene los libros de una variable llamada bibliotecaSeleccionada, que de primeras será la biblioteca predeterminada, pero se podrá cambiar a posteriori. Se crea un objeto tipo ObservableList<Map<String,Object>> y se crea un Array de Map<String,Object> con el tamaño de los libros que tenga la biblioteca. Se creará un bucle for que itere sobre el Array y lo cargará de datos. A continuación, se comprobará si el libro está en local o en la base de datos, ya que la forma de obtener los datos de cada uno cambia. Una vez determinado el origen de la información, se crean atributos clave-valor que contendrán la información y se rellenan con datos. En el caso de las bibliotecas locales se utilizarán los metadatos de los libros, que se obtienen de la clase Ebook. Una vez obtenidos los datos se cargarán todas las entradas del Array en el objeto ObservableList creado anteriormente, este se cargará en la tabla y se guardará una copia de los objetos para utilizarlos en otros métodos.

```

//carga una tabla a partir de una lista de de maps que representan las filas de las tablas siendo cada uno un libro
private void loadTable(List<Map<String, Object>> nuevatabla){
    Map<String, Object>[] libros=new HashMap[nuevatabla.size()];
    for (int i=0;i<libros.length;i++){
        libros[i]=new HashMap<>();
        libros[i].put(key:"Local", value: nuevatabla.get(index: i).get(key:"Local"));
        libros[i].put(key:"Titulo",value: nuevatabla.get(index: i).get(key:"Titulo"));
        libros[i].put(key:"Genero",value: nuevatabla.get(index: i).get(key:"Genero"));
        libros[i].put(key:"Saga", value: nuevatabla.get(index: i).get(key:"Saga"));
        libros[i].put(key:"Idioma", value: nuevatabla.get(index: i).get(key:"Idioma"));
        libros[i].put(key:"Fecha", value: nuevatabla.get(index: i).get(key:"Fecha"));
        libros[i].put(key:"Puntuacion", value: nuevatabla.get(index: i).get(key:"Puntuacion"));
        libros[i].put(key:"Autor",value: nuevatabla.get(index: i).get(key:"Autor"));
        libros[i].put(key:"Sinopsis",value: nuevatabla.get(index: i).get(key:"Sinopsis"));
        libros[i].put(key:"Path",value: nuevatabla.get(index: i).get(key:"Path"));
    }
    ObservableList<Map<String,Object>> ol=FXCollections.observableArrayList();
    ol.addAll(ol);
    tableCentral.setItems(ol);
}

```

El controlador cuenta con otro método que carga campos en la tabla, utilizado sobre todo para el filtrado. Este método, a diferencia del otro, carga los datos en base a una lista que se le proporciona.

```
/*coge el texto del textfield de busqueda del titulo cada vez que se suelta una tecla
si encuentra coincidencias con ese texto las guarda
si el textfield se queda en blanco se cargara la tabla original entera
*/
@FXML
private void findBook(){
    if(!textfTitulo.getText().isBlank()){
        List<Map<String, Object>> itemCopia=new ArrayList<>();
        ObservableList<Map<String, Object>> items = itemsOriginales;
        for(Map item:items){
            String title=(String)item.get(key:"Titulo");
            if(title.startsWith(prefix: textfTitulo.getText())){
                itemCopia.add(e: item);
            }
        }
        loadTable(nuevatabla: itemCopia);
    } else {
        loadInitialTable();
    }
}
```

El método findBook se ejecutará cada vez que el usuario deje de presionar una tecla, cogiendo el texto escrito en el buscador y creando una nueva lista de libros que se mandará como parámetro al método loadTable. En caso de que el buscador esté en blanco se cargará la tabla inicial.

```
//filtrar por autor
@FXML
private void sortBookAutors(){
    Object autor = comboBoxAutor.getSelectionModel().getSelectedItem();
    List<Map<String, Object>> itemCopia=new ArrayList<>();
    ObservableList<Map<String, Object>> items = itemsOriginales;
    for(Map item:items){
        String title=(String)item.get(key:"Autor");
        if(title.contains(s: autor.toString())){
            itemCopia.add(e: item);
        }
    }
    loadTable(nuevatabla: itemCopia);
}
```

También se puede filtrar por género, autor y saga. Los métodos recogen el ítem seleccionado en el combobox y la lista de registros iniciales de la tabla, luego recorre los registros con un bucle for y se saca el dato de autor de cada uno. Por último, se compara el autor seleccionado, con el autor del registro y si coinciden se guarda el registro en una tabla que se pasará como argumento al método loadTabla()

```

//
@FXML
private void setInfoPanel() {
    Map<String, Object> selectedItem = tableCentral.getSelectionModel().getSelectedItem();
    if(selectedItem!=null){
        //cambiar en tre el boton de leer y el de descargar
        if(selectedItem.get(key:"Local").equals(obj:"Si")){
            cahngeDonwloadButton();
        } else{
            changeReadButton();
        }
        lblIdioma.setText((String)selectedItem.get(key:"Idioma"));
        lblAutor.setText((String)selectedItem.get(key:"Autor"));
        lblFecha.setText((String)selectedItem.get(key:"Fecha"));
        lblSaga.setText((String)selectedItem.get(key:"Saga"));
        lblTitulo.setText((String)selectedItem.get(key:"Titulo"));
        lblGeneros.setText((String)selectedItem.get(key:"Genero"));
        textSinopsis.setText((String)selectedItem.get(key:"Sinopsis"));
        setPortada((String)selectedItem.get(key:"Path"));
    }
}

```

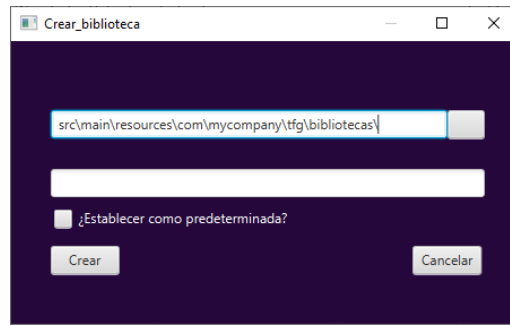
Para cargar información en el panel derecho, la tabla cuenta con un método que se llama cada vez que se hace click sobre un ítem de esta. Se obtiene el ítem seleccionado y se hace la comprobación de que no sea null. En caso de que el ítem tenga contenido, se hace otra comprobación sobre la propiedad “Local” del Map para saber si ya está en la biblioteca y cambiar el botón de descarga por el de leer y viceversa. Una vez se hacen todas las comprobaciones, se cargan los datos en sus respectivas labels.

```

@FXML
private void loadBook() {
    try {
        launchModal(ventana:"Cargar_libro");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

```

La biblioteca también cuenta con los botones que abrirán las ventanas modales encargadas de tratar con las bibliotecas y libros. En su mayor parte, estas clases interactúan con los distintos métodos de la clase ManejaArchivo.



```
//carga la ruta predeterminada para las bibliotecas
@FXML
private void initialize() {
    txtNewLibraryPath.setText(string: "src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\");
}

//Metodo para crear una biblioteca nueva haciendo las comprobaciones adecuadas
@FXML
private void createNewLibrary() {
    File comp=new File(pathname: txtNewLibraryPath.getText());
    if(!txtLibraryName.getText().isBlank() || comp.exists() || comp.getAbsolutePath().isBlank()){
        File f=new File(txtNewLibraryPath.getText()+"\\"+txtLibraryName.getText());
        ManejaArchivos.createLibrary(library:f);
        lblEmptyName.setVisible(bln:false);
        lblEmptyPath.setVisible(bln:false);
        if(chboxDefaultLibrary.isSelected()){
            ManejaArchivos.setDefaultLibrary(f);
        }
    } else {
        lblEmptyPath.setVisible(bln:true);
        lblEmptyName.setVisible(bln:true);
    }
}

//Crea un DirectoryChooser, lo hace visible y lo muestra en la interfaz una vez escogido
@FXML
private void newFileChooser() {
    DirectoryChooser dchooser=new DirectoryChooser();
    dchooser.setTitle(string: "Seleccionar ruta para la biblioteca");
    File newLibraryPath=dchooser.showDialog(window: App.getPrimaryStage());
    txtNewLibraryPath.setText(string: newLibraryPath.getAbsolutePath());
}

@FXML
private void close() {
    App.getModalStage().close();
}
}
```

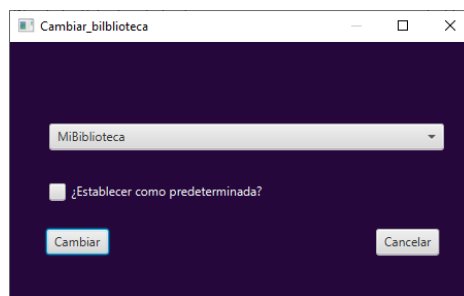
Una de estas clases sería la utilizada para crear nuevas bibliotecas. Lo primero que hace esta clase es cargar el textfield con la ruta predeterminada para las nuevas bibliotecas, que será siempre la misma. Esta clase cuenta con dos métodos, uno para abrir un directoryChooser para seleccionar la ruta de la biblioteca y otro método que creará la biblioteca. El método createNewLibrary empezará obteniendo la ruta de la biblioteca, ya sea la predeterminada o la que ha escogido el usuario y comprobará que no está en blanco, que no existe ya un archivo con ese nombre o que el nombre de la biblioteca está en blanco. Si alguna de estas condiciones no se cumple se harán visibles los mensajes de error adecuados, si se cumplen, se creará un objeto File con esa ruta y se comprobará si el checkbox de biblioteca predeterminada está marcado. Si lo está, se guardará la ruta en el archivo .properties. Por último, se hacen invisibles los mensajes de error y se guarda la ruta en el archivo .properties.

```

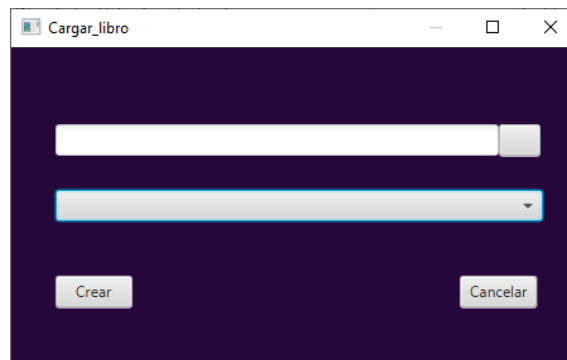
//cambia la biblioteca seleccionada y recarga la ventana principal
@FXML
private void changeLibrary() {
    SingleSelectionModel selectionModel = comboBoxLibrary.getSelectionModel();
    if (!selectionModel.isEmpty()) {
        if (comboBoxDefault.isSelected()) {
            Properties.setProperty("BibliotecaPredeterminada", selectionModel.getSelectedItem());
        }
        App.BibliotecaSeleccionada = new File(selectionModel.getSelectedItem());
    } else {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error al cambiar la biblioteca");
        alert.setHeaderText("Debe seleccionar una biblioteca");
        alert.showAndWait();
    }
    try {
        App.setRoot("Biblioteca");
        App.getMainWindow().close();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}

//carga el combobox con las bibliotecas del archivo properties
private void loadComboBox() {
    ObservableList<String> items = FXCollections.observableArrayList();
    File[] libraries = ManejaArchivos.getLibraries();
    for (File libreria : libraries) {
        items.add(libreria.getName());
    }
    comboBoxLibrary.setItems(items);
}

```



Otro ejemplo de estas clases sería `ControllerCambiarBiblioteca`, que cuenta con dos métodos propios, el `loadComboBox` y el `changeLibrary`. El método `loadComboBox` es llamado nada más iniciarse la aplicación y será el que cargue el combobox de biblioteca con los nombres de las bibliotecas que el usuario haya creado en la aplicación. Para ello, se obtendrán las rutas de los libros utilizando la clase `ManejaArchivo` y se cargarán todos los ítems. El método `changeLibrary` está asociado al botón `cambiar` y será el que cambie la biblioteca activa en el momento. Para ello, obtiene el ítem seleccionado en el combobox y comprueba que no esté vacío. En caso de estarlo saltará una ventana con un mensaje de error. Si el ítem no está vacío, se hará una comprobación para saber si el checkbox de biblioteca predeterminada está marcado, en caso afirmativo, se guardará la ruta en la propiedad correspondiente. Por último, se guardará la ruta de la nueva biblioteca en un atributo de la clase `main` y recargará la ventana de biblioteca.



```
//Cargar un libro de la biblioteca en una carpeta a especificar
//Se creara una carpeta en el destino con el nombre del libro si no existe ya, y se copiara
@FXML
private void loadBook(){
    String selectedBook = (String) comboBoxBook.getSelectionModel().getSelectedItem();
    List<File> books = ManejaArchivos.getBooks(library.App.BibliotecaSeleccionada);
    //se saca el archivo .epub de la carpeta del libro
    File libraryBookPath=new File(App.bibliotecaSeleccionada.getAbsolutePath()+"\\"+selectedBook+"\\ "+selectedBook+".epub");
    //path de la biblioteca del dispositivo
    //Se crea en el libro electronico el directorio donde guardar el libro
    File ebookLibraryPath=new File(txtEBookPath.getText()+"\\"+selectedBook);
    if(!ebookLibraryPath.exists() && !ebookLibraryPath.getAbsolutePath().isBlank()){
        ebookLibraryPath.mkdir();
        ebookLibraryPath=new File(txtEBookPath.getText()+"\\"+selectedBook+"\\ "+selectedBook+".epub");
        try {
            Files.copy(source: libraryBookPath.toPath(),target: ebookLibraryPath.toPath());
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    } else {
        Alert alert=new Alert(Alert.AlertType.ERROR);
        alert.setTitle(string: "Error al cargar el libro");
        alert.setHeaderText(string: "El campo de libro esta en blanco o no es valido");
        alert.showAndWait();
    }
}
```

La clase ControllerCargarLibro será la que cargue los libros de la biblioteca en los distintos dispositivos. Primero se rellenará el combobox con los libros de la biblioteca activa. Una vez el usuario pulse el botón de crear, se llamará al método loadBook. Este método obtendrá el item seleccionado en el combobox y se creará un objeto File que será la unión del path de la ruta seleccionada, el item seleccionado, y ese mismo item con la extensión correcta. El siguiente paso es crear un objeto File a partir de la ruta destino y el nombre del libro. Se comprueba que la ruta no exista ya y se crea el directorio donde se copiará el archivo .epub. Una vez creado el directorio, se copia el libro en dicha ruta. Como mejora a futuro se ha planteado que la app detecte automáticamente los dispositivos para así evitar posibles errores por parte del usuario.

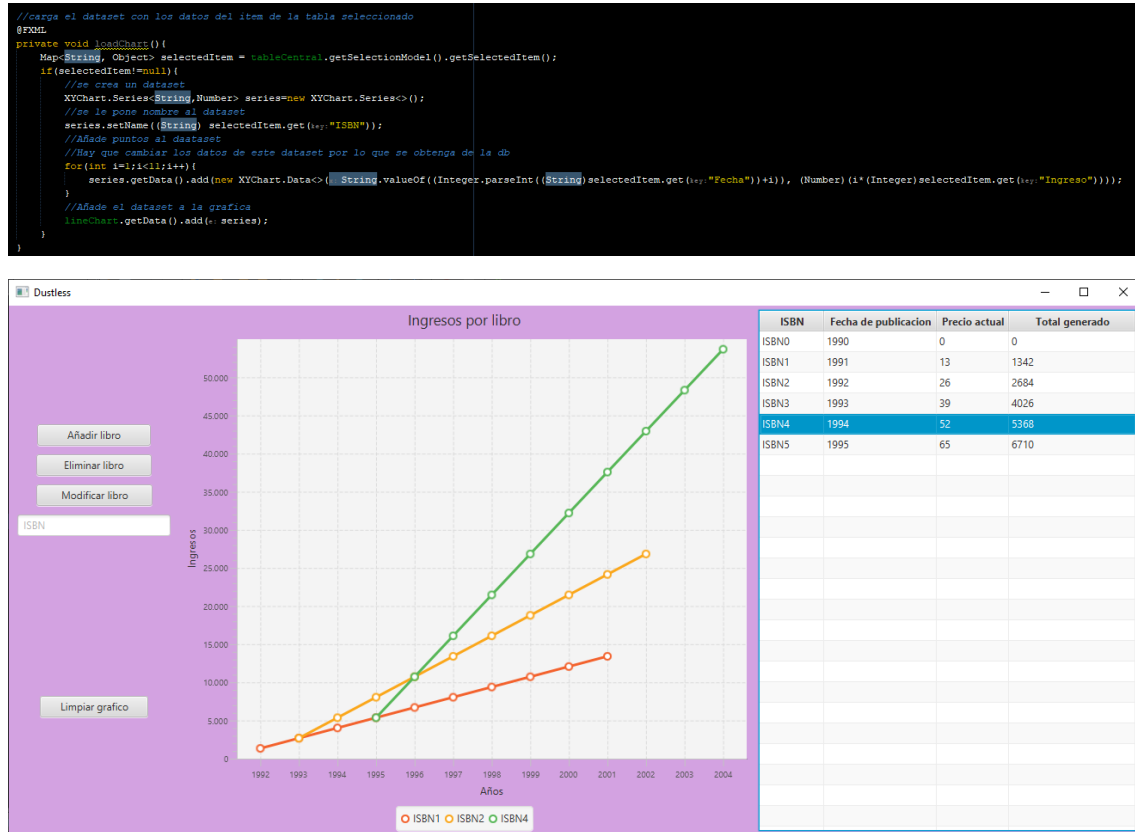
11.1.6. Tienda

La ventana de tienda funcionará igual que la ventana de biblioteca con las diferencias de que esta no cuenta con los botones para controlar las bibliotecas y los libros locales y saca obtiene la información de los libros de la base de datos. El apartado de tienda también cuenta con el botón de comprar libro, que cogiendo la información del libro seleccionado creará un checkout session de Stripe. Una vez se haya recibido el url para proceder al pago por parte de Stripe, se creará una ventana emergente con un WebView donde cargar el URL. Por último, al recibir la respuesta de la compra, ya sea exitosa o no, se redirigirá al usuario a la página correspondiente y en caso de ser exitosa, se le habilitará en la cuenta el libro comprado.

11.1.7. Editorial

En el caso de que el usuario pertenezca a una editorial cuando se inicie sesión será redirigido a la pestaña de editorial, donde verá una tabla que contenga los libros que haya

publicado, un pequeño panel de control para añadir más libros, modificarlos, eliminarlos y buscarlos dentro de la base de datos, y un gráfico que mostrara los ingresos del libro seleccionado en cada año desde el momento en el que dicha editorial lo publicó. Al igual que las demás pestañas, esta contará con su clase controlador, que en este caso será la encargada de manejar la navegación y el contenido del gráfico y de la tabla.

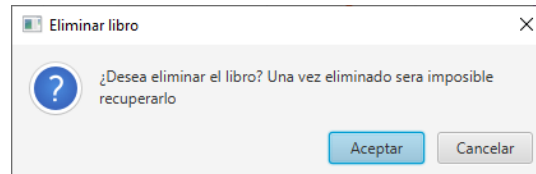
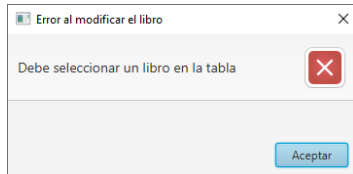


Este método será llamado cada vez que se seleccione un item de la tabla. Una vez se compruebe que el item no es null, se crea un dataset, que será plasmado en el grafico una vez tenga datos. Para ello, a través de la información obtenida de la base de datos, se cargará en el eje X los años en los que ha reportado veneficios dicho libro y en el eje Y los beneficios de ese año. Por último, se añade el dataset al gráfico.

```
@FXML
private void modifyBook(){
    if (tableCentral.getSelectionModel().getSelectedItem()!=null){
        try {
            launchModal(modal:"Modificar_libro");
            ControllerModificarLibro.selectedBookISBN=(String)tableCentral.getSelectionModel().getSelectedItem().get(key:"ISBN");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    } else {
        Alert alert=new Alert(Alert.AlertType.ERROR);
        alert.setTitle(modal:"Error al modificar el libro");
        alert.setHeaderText(modal:"Debe seleccionar un libro en la tabla");
        alert.showAndWait();
    }
}

@FXML
private void deleteBook(){
    //se comprueba si el usuario ha seleccionado un item de la tabla
    if (tableCentral.getSelectionModel().getSelectedItem()!=null){
        //se muestra un mensaje de confirmacion
        Alert alert=new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle(modal:"Eliminar libro");
        alert.setHeaderText(modal:"");
        alert.setContentText(modal:"¿Desea eliminar el libro? Una vez eliminado sera imposible recuperarlo");
        //en caso afirmativo se elimina el registro de la base de datos
        //en este caso se limpia la tabla
        if(alert.showAndWait().get() == ButtonType.OK){
            cleanChart();
        }
    } else {
        Alert alert=new Alert(Alert.AlertType.ERROR);
        alert.setTitle(modal:"Error al eliminar el libro");
        alert.setHeaderText(modal:"Debe seleccionar un libro en la tabla");
        alert.showAndWait();
    }
}
}
```


Para evitar posibles errores, los botones de eliminar y modificar libros harán una comprobación antes de nada, de si hay un item seleccionado en la tabla, ya que a través de ese item se obtendrá el ISBN que es la id del libro en la base de datos. En el caso de eliminar un libro, mostrará un cartel de confirmación para evitar posibles borrados por accidente y en caso de que no lo sea cogerá el libro seleccionado y eliminará la relación de ese libro con la editorial en la base de datos.



En el caso de actualizar un libro, no se mostrará el mensaje de confirmación. Aparte de eso, también mandará el ISBN del libro seleccionado a la clase controlador correspondiente, para que esta pueda hacer la consulta a la base de datos y que esta le devuelva el libro correspondiente.

11.1.8. Ventana Perfil



Desde esta ventana el usuario tendrá la capacidad de cambiar información privada como pueda ser el nombre, apellido, correo y contraseña.

11.2. OCR

11.2.1. Introducción

Una de las características que tendrá la aplicación para distinguirnos de la competencia será la capacidad de usar una foto de la portada del libro para saber si se encuentra disponible en la tienda y ver su información. Para llevar a cabo esta funcionalidad se utilizará Tesseract OCR, debido a que es open-source, lo que nos permitirá adaptarlo a nuestras necesidades en caso de ser necesario. También nos permitirá ahorrar ciertos costes, como los que suman las tarifas de en base a las peticiones hechas con Google Cloud Vision o Amazon Textract. Por otra parte, utilizar Tesseract conlleva costes adicionales que los otros no tendrían, como el mantenimiento del servidor donde se implemente el servicio y el tiempo requerido para configurarlo correctamente entre otros. También se ha decidido que, aunque se podría implementarse el OCR directamente en la aplicación móvil, este se llevará a cabo en el servidor para reducir la carga que se pueda generar en este por el uso de la aplicación.

Se ha optado implementar un OCR en vez de una IA capaz de reconocer imágenes porque, si bien se quiere que la funcionalidad sea principalmente sacar la información a través de la portada, queremos que se pueda usar también con un texto escrito a mano o un documento.

11.2.2. Implementación

Para la comunicación, se utilizará una clase en la aplicación móvil que transmita las imágenes sin procesar al servidor, donde serán tratadas para facilitar el reconocimiento del texto. Una vez detectada la información necesaria, se consultará en la base de datos si el libro está disponible. En caso afirmativo, se redirigirá al usuario a una ventana específica de la tienda para ese libro. Si el libro no se encuentra en la base de datos o la imagen no es lo suficientemente clara para su procesamiento, se mostrará al usuario un mensaje correspondiente.

Lo primero que se debe hacer es la instalación. Esta se puede realizar de dos formas: a través de la línea de comandos utilizando los comandos “apt install tesseract-ocr” y “apt install libtesseract-dev”, o compilando el programa por nosotros mismos. Una vez instalado el motor, será necesario obtener los archivos .traineddata correspondientes para cada idioma que se desee implementar. Estos archivos se pueden descargar directamente desde el repositorio o, si se prefiere, se pueden gestionar manualmente. En este último caso, habrá que descargar los archivos .traineddata y colocarlos manualmente en una carpeta llamada tessdata.

Como se ha mencionado antes, se dispondrá de una clase en el servidor que será la encargada de recibir los datos necesarios. Una vez se disponga de dicha información que será la imagen y el idioma, se llamará a otra clase que será la que tratará la imagen para facilitar el reconocimiento de esta. Cuando se complete este proceso, se utilizará tessj4, un wrapper que nos permitirá trabajar con Tesseract ya que está escrito en c++.

11.2.3. Tratamiento de imagen

Para mejorar la precisión del OCR a la hora de detectar los títulos de los libros desde las portadas se tratarán las imágenes para aumentar las posibilidades de un reconocimiento correcto. Lo primero será convertir el formato de imagen a uno de los que admite Tesseract en caso de que sea necesario. Lo siguiente será poner en vertical la imagen. A continuación, se reescalara la imagen para que tenga 300 DPI (dots per inch) ya que eso aumentara la tasa de éxito del OCR. Después se procederá a binarizar la imagen, ya que los OCR trabajan mejor con imágenes en blanco y negro o de alto contraste. El siguiente paso será intentar eliminar la mayor cantidad de ruido de la imagen. Por último, se acotarán las zonas con texto para intentar facilitar aún más el reconocimiento del texto. Para ayudarnos con el proceso utilizaremos OpenCV, que es una librería de visión artificial, que nos ayudará a automatizar procesos más complicados. Hay que tener en cuenta que no siempre se conseguirá un resultado positivo ya que esto es una tecnología que no es 100% infalible.

11.2.4. Futuras mejoras

- Entrenar a Tesseract con portadas de libros para mejorar la tasa de éxito. Esto será un objetivo a largo plazo ya que habría que introducir cada portada de forma manual indicándole cual debería ser el resultado esperado.

- Implementar un proceso previo al procesado de la imagen que determine si merece la pena o el intentarlo.
- Entrenar a Tesseract para aumentar la tasa de éxito con texto escrito a mano, ya que de base no está enfocado en ello.
- Integrar en la versión de escritorio el sistema de OCR

11.2.5. Motivos de la no implementación

El motivo por el que se ha decidido no llevar a cabo la implementación de este servicio es que se ha determinado que la complejidad junto con la falta de tiempo dificultara una correcta integración, lo que llevaría a errores imprevistos.

11.3. Pagos

11.3.1. Introducción

Para la administración de pagos se utilizará la API de Stripe ya que este servicio ofrece una integración sencilla tanto para la aplicación móvil como para la de escritorio, proporciona un entorno de pruebas donde se podría hacer un ejemplo si se implementa, compatibilidad con distintos lenguajes.

Para poder usar el servicio será necesario tener cuenta creada de Stripe ya que nos dará acceso a las API keys, que son necesarias para trabajar con su API. Estas keys se dividen en dos, secret key que va en la aplicación de servidor, y publishable key que va en la aplicación cliente.

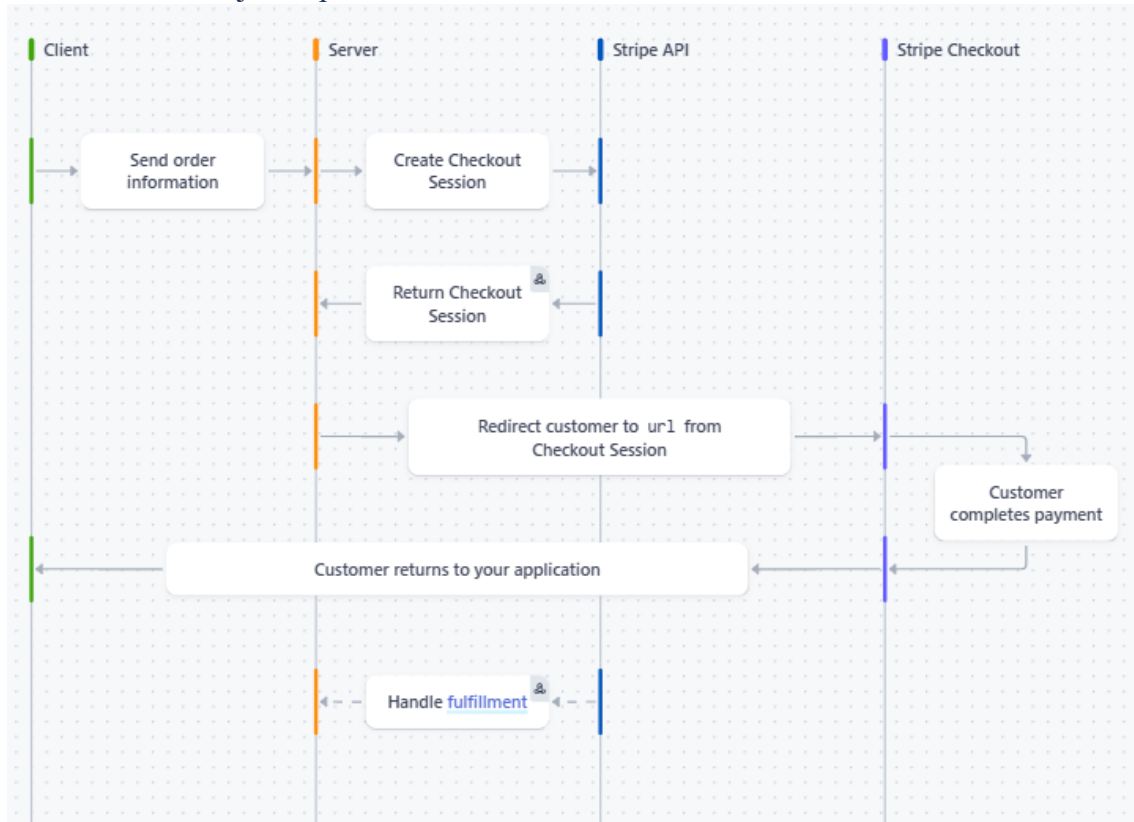
Si bien Stripe cuenta con distintas formas de tratar con los pagos, nos hemos decantado por utilizar Checkout Session ya que es una forma sencilla para manejar pagos únicos o suscripciones ya que delega gran parte del proceso a Stripe. Si se viese necesario también se podría manejar de forma más “personal” el flujo de pago sin delegar tanto.

11.3.2. Implementación

A la hora de trabajar con Checkout Session hay 3 formas, haciendo que el propio Stripe albergue la página de los pagos, incrustando la página en un formulario o utilizando componentes personalizados de pago desde nuestra página. Se ha optado por que sea Stripe la que aloje el formulario de pagos ya que es la opción más eficiente para gestionar los recursos de los que se dispone en este proyecto.

Cuando el usuario pulse el botón de compra, este recogerá la información necesaria y se la mandará al servidor. Una vez recibidos los datos y creará con un Checkout Session a través de la API de Stripe, que devuelve un objeto Session que contiene una URL única. Esta URL es devuelta al usuario, donde se abrirá. En la página, el usuario introduce los datos de pago, y es el propio Stripe quien valida los mismos. Dependiendo del resultado del pago, Stripe conducirá al usuario a la página correspondiente. Desde el servidor se verificará el resultado del pago y se le añadirá al usuario el libro correspondiente al pago.

11.3.3. Flujo del proceso

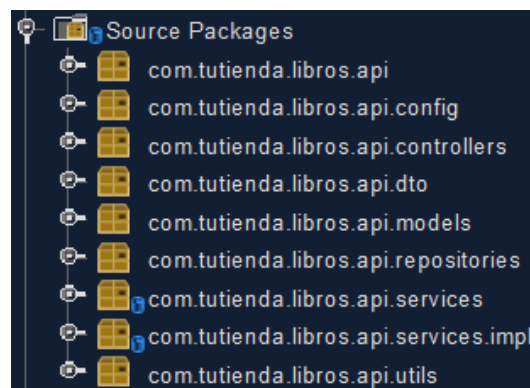


11.4. API RESTful

11.4.1. Introducción

La API RESTful desarrollada con Spring Boot constituye el núcleo del sistema de gestión de biblioteca digital. Implementa un diseño orientado a recursos que sigue los principios REST, ofreciendo:

- Interoperabilidad mediante JSON
- Autenticación JWT robusta
- Gestión completa del ciclo de vida de libros
- Sistema de recomendaciones y listas de deseos
- Administración de relaciones complejas (autores, sagas, editoriales)



11.4.2. Modelos de Dominio

Usuario

- Datos personales (nombre, email)
- Credenciales seguras (contraseña hasheada)
- Relación con editoriales (para usuarios editores)
- Cartera virtual para compras
- Biblioteca personal y lista de deseos

Libro

- Metadatos básicos (título, ISBN, páginas)
- Información comercial (precio, descuentos)
- Clasificación (género, idioma)
- DRM y formatos digitales
- Relaciones N:M con autores y editoriales
- Valoraciones y reseñas (agregadas)

Biblioteca

- Colección personal de libros por usuario
- Organización por estanterías virtuales
- Estadísticas de lectura
- Sincronización multiplataforma

Autor

- Datos personales
- Relación con libros

Editorial

- Datos corporativos
- Catálogo de libros publicados
- Relación con usuarios editores
- Historial de publicaciones

Saga

- Colección de libros relacionados
- Orden cronológico
- Autores participantes
- Información de continuidad

Deseado

- Lista de libros deseados por usuario
- Sistema de prioridades
- Integración con cartera virtual

Género

- Jerarquía de categorías

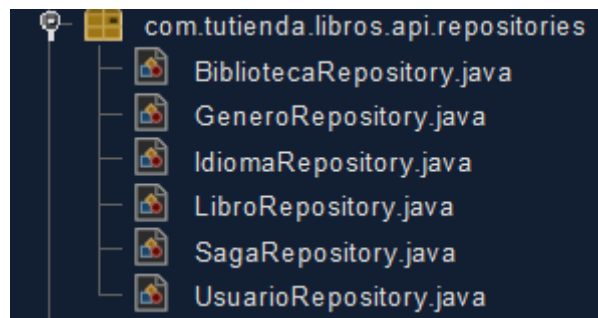
- Sistema de etiquetado

Idioma

- Traducciones disponibles
- Metadatos específicos por idioma

11.4.3. Repositorios

Implementan el patrón Repository para acceso a datos.



11.4.4. Servicios

11.4.4.1. LibroService

Descripción: Lógica de negocio para la gestión de libros.

Métodos clave:

- crearLibroDesdeDTO(LibroDTO libroDTO): Crea un libro desde un DTO.
- buscarLibrosConFiltros(String titulo, String autor, String genero, String saga, LocalDate fechaPublicacion, Double valoracionMin, Double valoracionMax, BigDecimal precioMin, BigDecimal precioMax, String idioma): Permite buscar libros mediante varios filtros.
- buscarPorId(String idLibro): Busca un libro por su ID.
- actualizarLibroDesdeDTO(String idLibro, LibroDTO libroDTO): Actualiza un libro utilizando un DTO.
- eliminarLibro(String idLibro): Elimina un libro según su ID.

```
public interface LibroService {  
    // Métodos originales  
    Libro buscarPorId(String idLibro);  
    Libro crearLibro(Libro libro);  
    Libro actualizarLibro(String idLibro, Libro libro);  
    void eliminarLibro(String idLibro);  
  
    // métodos para DTOs y archivos  
    Libro crearLibroDesdeDTO(LibroDTO libroDTO) throws IOException;  
    Libro actualizarLibroDesdeDTO(String idLibro, LibroDTO libroDTO) throws IOException;  
  
    // método para búsqueda avanzada  
    List<Libro> buscarLibrosConFiltros(  
        String nombreLibro,  
        String nombreAutor,  
        String nombreGenero,  
        String nombreSaga,  
        LocalDate fechaPublicacion,  
        Double valoracionMin,  
        Double valoracionMax,  
        BigDecimal precioMin,  
        BigDecimal precioMax,  
        String nombreIdioma  
    );  
}
```

11.4.4.2. UsuarioService

Descripción: Lógica de negocio para la gestión de usuarios.

Métodos clave:

- registrarUsuario(UsuarioDTO usuarioDTO): Registra un nuevo usuario en el sistema.
- iniciarSesion(String usuario, String pass): Realiza el login de un usuario.
- actualizarUsuario(String usuario, UsuarioDTO usuarioDTO): Actualiza la información de un usuario.
- actualizarContraseña(String usuario, String contraseñaAntigua, String contraseñaNueva): Actualiza la contraseña de un usuario.
- eliminarUsuarioPorUsuario(String usuario): Elimina un usuario por su nombre de usuario.

```
public interface UsuarioService {  
  
    // Para login  
    Optional<Usuario> iniciarSesion(String usuario, String pass);  
  
    // Crear nuevo usuario (valida que el nombre de usuario no exista)  
    Usuario registrarUsuario(UsuarioDTO usuarioDTO);  
  
    // Actualizar usuario existente  
    Usuario actualizarUsuario(String nombreUsuario, UsuarioDTO usuarioDTO);  
  
    // Actualizar fondos  
    Usuario actualizarFondos(String nombreUsuario, BigDecimal cantidad);  
  
    // Eliminar por nombre de usuario  
    void eliminarUsuarioPorUsuario(String usuario);  
  
    // Buscar por nombre de usuario (para perfil, etc.)  
    Optional<Usuario> buscarPorUsuario(String usuario);  
  
    // Nuevo método para actualizar la contraseña  
    boolean actualizarContraseña(String usuario, String contraseñaAntigua, String contraseñaNueva);  
}
```

11.4.4.3. GeneroService

Descripción: Lógica de negocio para la gestión de géneros.

Métodos clave:

- obtenerGeneros(): Obtiene una lista de todos los géneros disponibles en el sistema.

11.4.4.4. IdiomaService

Descripción: Lógica de negocio para la gestión de idiomas.

Métodos clave:

- obtenerIdiomas(): Obtiene una lista de todos los idiomas disponibles en el sistema.

11.4.4.5. BibliotecaService

Descripción: Lógica de negocio para la gestión de bibliotecas de los usuarios.

Métodos clave:

- obtenerBibliotecaPorUsuario(UsuarioDTO usuario): Obtiene la biblioteca de un usuario específico.
- crearBiblioteca(UsuarioDTO usuario): Crea la biblioteca de un usuario.
- actualizarBiblioteca(UsuarioDTO usuario, String bibliotecaId): Crea o actualiza la biblioteca de un usuario.
- eliminarBibliotecaPorUsuario(UsuarioDTO usuario, String bibliotecaId): Elimina la biblioteca de un usuario.

- `agregarLibrosABiblioteca(UsuarioDTO usuario, List<String> libroIds)`: Añade libros a la biblioteca de un usuario.
- `eliminarLibrosDeBiblioteca(UsuarioDTO usuario, List<String> libroIds)`: Elimina libros de la biblioteca de un usuario.

```
public interface BibliotecaService {  
  
    Biblioteca crearBiblioteca(UsuarioDTO usuario);  
  
    Biblioteca obtenerBiblioteca(UsuarioDTO usuario);  
  
    Biblioteca actualizarBiblioteca(UsuarioDTO usuario, String bibliotecaid);  
  
    Biblioteca eliminarBiblioteca(UsuarioDTO usuario, String bibliotecaid);  
  
    Biblioteca agregarLibrosABiblioteca(UsuarioDTO usuario, List<Libro> libro);  
  
    Biblioteca eliminarLibrosDeBiblioteca(UsuarioDTO usuario, List<Libro> libro);  
}
```

11.4.5. Utilidades

Clases auxiliares que proporcionan funcionalidad transversal:

Seguridad

- JWT Generator/Validator
- Password Encoder
- Rate Limiter

Archivos

- EPUB Processor
- PDF Generator
- Image Optimizer

Validación

- Email Verification

Conversión

- DTO Mapper

11.4.6. DTOs

Los DTOs facilitan la comunicación segura entre cliente y servidor al controlar la información expuesta. Además, centralizan la validación de entradas y adaptan formatos entre capas, mejorando mantenibilidad y escalabilidad.

LibroDTO

- Todos los metadatos editables
- Soporte para archivos multipart
- Relaciones (IDs o nested DTOs)
- Validación de campos

UsuarioDTO

- Datos públicos/perfil

- Credenciales (solo para registro/login)
- Cartera y estadísticas

11.4.7. Controladores

Los endpoints REST se organizan por recurso:

11.4.7.1. *BibliotecaController*

- POST /api/bibliotecas/usuario/{usuario}:
 1. Descripción: Crea o actualiza la biblioteca de un usuario. Si no existe, se crea una nueva; si ya existe, se actualiza con los datos proporcionados.
 2. Cuerpo de la solicitud: BibliotecaDTO
 3. Respuesta: Si la operación es exitosa, se devuelve el objeto Biblioteca con estado HTTP 201 Created. En caso de error, se retorna 400 Bad Request.
- GET /api/bibliotecas/usuario/{usuario}:
 1. Descripción: Recupera la biblioteca asociada a un usuario específico.
 2. Respuesta: Si la biblioteca existe, se devuelve el objeto Biblioteca con estado HTTP 200 OK. Si no se encuentra, se retorna 404 Not Found.
- GET /api/bibliotecas:
 1. Descripción: Obtiene todas las bibliotecas existentes en el sistema.
 2. Respuesta: Devuelve una lista de objetos Biblioteca con estado HTTP 200 OK.
- PUT /api/bibliotecas/usuario/{usuario}/libros:
 1. Descripción: Añade o elimina libros de la biblioteca del usuario.
 2. Añadir: Se agregan los libros especificados a la biblioteca del usuario.
 3. Eliminar: Se eliminan los libros especificados de la biblioteca del usuario.
 4. Cuerpo de la solicitud: Lista de IDs de libros a añadir o eliminar.
 5. Respuesta: Si la operación es exitosa, se devuelve un estado 200 OK. En caso de error, se retorna 400 Bad Request.
- DELETE /api/bibliotecas/usuario/{usuario}:
 1. Descripción: Elimina la biblioteca asociada a un usuario específico.
 2. Respuesta: Si la eliminación es exitosa, se devuelve un mensaje de éxito con estado HTTP 200 OK. Si no se encuentra la biblioteca, se devuelve 404 Not Found.

11.4.7.2. *GeneroController*

- GET /api/generos: Obtiene todos los géneros disponibles.
 1. Llama al servicio GeneroService para obtener la lista de géneros.

11.4.7.3. *IdiomaController*

- GET /api/idiomas: Obtiene todos los idiomas disponibles.
 1. Llama al servicio IdiomaService para obtener la lista de idiomas.

11.4.7.4. *LibroController*

- POST /api/libros: Crea un libro con archivos.
 1. Recibe datos de libro en formato multipart/form-data, los procesa con LibroService y crea un nuevo libro.
- GET /api/libros/buscar: Busca libros con filtros.

1. Permite buscar libros por título, autor, género, saga, fecha, valoración, precio e idioma.
- GET /api/libros/{idLibro}: Obtiene un libro por ID.
 1. Llama al servicio LibroService para obtener los detalles de un libro específico.
- PUT /api/libros/{idLibro}: Actualiza un libro existente con archivos.
 1. Recibe datos para actualizar un libro existente, manejados por LibroService.
- DELETE /api/libros/{idLibro}: Elimina un libro por ID.
 1. Llama a LibroService para eliminar el libro especificado.

11.4.7.5. UsuarioController

- POST /api/usuarios/regar: Registra un nuevo usuario.
 1. Recibe los datos del usuario y los envía al servicio UsuarioService para su registro.
- POST /api/usuarios/login: Inicia sesión con un usuario.
 1. Recibe las credenciales del usuario, verifica la autenticidad y devuelve un DTO con la información del usuario.
- PUT /api/usuarios/actualizar/{usuario}: Actualiza los datos de un usuario.
 1. Recibe nuevos datos para actualizar la información de un usuario.
- PUT /api/usuarios/actualizar/contraseña/{usuario}: Actualiza la contraseña de un usuario.
 1. Verifica que la contraseña antigua sea correcta y actualiza la nueva.
- PATCH /api/usuarios/fondos/{usuario}: Agrega fondos a la cuenta de un usuario.
 1. Recibe la cantidad a agregar y la aplica al balance del usuario.
- DELETE /api/usuarios/eliminar/{usuario}: Elimina un usuario.
 1. Elimina un usuario especificado por su nombre de usuario.

11.4.8. Configuraciones

Las configuraciones principales incluyen seguridad con JWT y Spring Security, almacenamiento de archivos, mapeo de recursos estáticos y personalización de comportamientos globales, garantizando un funcionamiento seguro y optimizado del sistema.

Seguridad

- JWT Filter Chain
- CSRF Protection

Almacenamiento

- Local/Cloud Storage
- File Type Validation
- Backup Strategy

12. Puesta en marcha

12.1. Configuración

Para hospedar el servidor y el OCR se utilizará AWS más en concreto EC2 ya que es un servicio flexible que nos ahorra los costos de montar la infraestructura por nuestra

cuenta. Para la base de datos se utilizará RDS ya que esta opción facilita mucho el proceso de configuración, manejo y monitorización de la base de datos.

12.2. Seguridad

Antes de almacenar en la base de datos la contraseña de un usuario, esta será hasheada para garantizar la seguridad de la información. Ya que para los pagos se utilizará Stripe se delegará a ellos la verificación de pagos, almacenamiento de información bancaria, etc.

Respecto a la aplicación, se ha creado pensando en una aplicación basada en roles, dándole acceso a su respectivo apartado a cada rol. Para ello, al momento de hacer un login se comprobará si el usuario es empresa o no y se llevarán a cabo los procedimientos adecuados en cada caso.

12.3. Legalidad

Sera necesario cumplir las siguientes leyes relacionadas con la protección de datos: Reglamento (UE) 2016/679 del Parlamento Europeo y del de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos; la Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales y el Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de desarrollo de la Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal.

En lo referente al comercio online será necesario cumplir la Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico; la Ley 7/1998, de 13 de abril, sobre condiciones generales de la contratación; el Real Decreto Legislativo 1/2007, de 16 de noviembre, por el que se aprueba el texto refundido de la Ley General para la Defensa de los Consumidores y Usuarios y otras leyes complementarias.

12.4. Pase a producción

El primer paso será configurar una instancia de Amazon EC2 encargada de albergar el servidor junto con la API. Se instalará el OCR en otra instancia de EC2 ya que a la hora de procesar imágenes son necesarios bastantes recursos. Otro de los motivos por los que se ha decidió separar el OCR en otra instancia ha sido para prevenir que en caso de fallo colapse todo. También será necesario configurar la b

Para asegurar que todo funcione de forma correcta, en la instancia del servidor se instalará java y Tesseract en la del OCR. Una vez todo esté instalado, habrá que configurar las distintas instancias para que se puedan comunicar unas con las otras, teniendo en cuenta las medidas de seguridad necesarias.

Para terminar con esto, se llevarán a cabo pruebas para comprobar el correcto funcionamiento tanto del servidor como de la API antes de que reciba uso real.

Una vez estén las instancias configuradas, el siguiente paso será comprobar que tanto la aplicación de escritorio como la aplicación móvil tengan un comportamiento estable. Para ello, se llevarán a cabo pruebas de funcionamiento para comprobar que haya errores inesperados y que las aplicaciones den los resultados esperados. Hay que tener en

cuenta sobre todo la parte de pagos y de inicio de sesión, ya que el fallo en alguna de estas puede llevar a problemas graves.

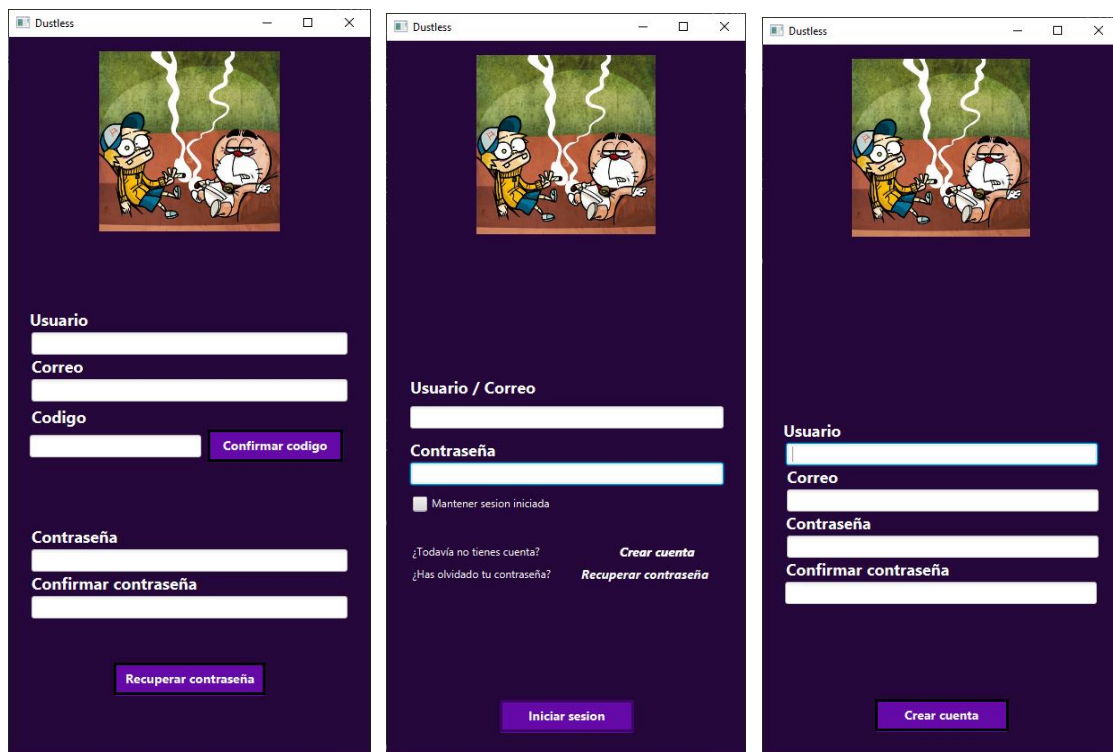
13. Control de calidad

13.1. Aplicación de escritorio

13.1.1. Login

En la ventana de login se deberán hacer prueba de la creación de usuarios, cambios de contraseña, autenticación, validación y del guardado de los datos en el archivo .properties.

La primera prueba que se llevará a cabo será el desplazamiento del usuario entre los distintos paneles de la ventana.



Lo siguiente será comprobar la validación de nuevos usuarios, para ello, se cuenta con un método que validara si el correo es válido, si la contraseña esta formateada correctamente, si ambas contraseñas coinciden y si todo queda registrado correctamente en la base de datos. Para que la contraseña este bien formateada debe de tener 12 caracteres mínimo, incluir una mayúscula, una minúscula, un número y un carácter especial.



Usuario
prueba

Correo
prueba@prueba.pru

Contraseña

Confirmar contraseña

Crear cuenta



Usuario
prueba

Correo
Correo Incorrecto

Contraseña

Confirmar contraseña

Las contraseñas no coinciden
La contraseña debe tener 12 caracteres, mayuscula, minuscula, un numero y un caracter especial

Crear cuenta

En caso afirmativo se devolverá un booleano true, y se cambiará al inicio de sesión. En el caso contrario, se mostrarán los mensajes de error correspondientes, se borran los campos incorrectos y el método devolverá, un false. El usuario no podrá salir de esta vista hasta que introduzca unos datos correctos o cierre la aplicación.



Usuario
prueba

Correo
prueba@prueba.pru

Codigo

Enviar codigo



Usuario
prueba

Correo
prueba@prueba.pru

Codigo

Confirmar codigo



Usuario
prueba

Correo
prueba@prueba.pru

Codigo

Confirmar codigo

Contraseña

Confirmar contraseña

Recuperar contraseña

Para comprobar el cambio de contraseña se introducirá un usuario y un correo que ya estén guardados en la base de datos y se mandaran al servidor como primer método de verificación. En caso de que exista ambos existan y estén relacionados, desde el servidor se enviara un correo con un código a la cuenta asociada al usuario. En la ampliación se le habilitará al usuario el botón de confirmar código. Este botón enviará al servidor el código que haya introducido el usuario y lo comprobará, en caso de coincida con el código que se mandó al correo devolverá un booleano true al cliente o un false en caso contrario. En caso de que la respuesta sea false, se mostrarán los mensajes de error correspondientes,

en el caso contrario, se le habilitará al usuario el panel que contiene los campos para cambiar la contraseña. Estos campos se validarán igual que los campos de creación de nuevo usuario. En caso de que sean correctos, se cambiará en la base de datos los datos relacionados. En el caso contrario se mostrarán los mensajes de error correspondientes.

A la hora de iniciar sesión, se comprobará si el usuario y la contraseña pertenecen a un mismo registro. En caso de que no pertenezcan al mismo usuario se mostrará el mensaje de error correspondiente, en el caso contrario se hará la comprobación de si el usuario es una editorial o un usuario promedio. Si es un usuario promedio se le redirigirá a la ventana de editorial, si es un usuario normal se le dirigirá a la biblioteca directamente.

ISBN	Fecha de publicación	Precio actual	Total generado
0000	1999	5	5
0001	1991	10	100
0002	1992	20	2000
0003	1993	30	3000
0004	1994	40	4000
0005	1995	50	5000

Autor	Título	Editorial	Fecha	Precio
Robert A. Rulov	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no
El mundo	El mundo del viento	Prueba, Random	En final no se ve	no

Por ultimo, se debe comprobar que el boton de recordar usuario guarda correctamente el usuario en el archivo .properties y que la proxima vez que se abra la aplicación se autocomplete el campo de usuario. Ya que el boton es un checkbox, cuando se marque la opcion se comprobará el estado. Si el boton esta ya marcado cuando se pulse, se desmarcará y guardara un 0 en la propiedad correspondiente, si no esta marcado, se marcará y se guardara un 1.

```
Recordar=1
Usuario=pruebadeguardado
```

Una vez que se haya autenticado el usuario, se guardará en el archivo properties es el texto del campo Usuario/Correo, para la proxima vez que abra la aplicación, en caso de que haya dejado marcado el boton de recordar.

13.1.2. Biblioteca

En la biblioteca, las partes esenciales a probar son el manejo de las rutas a la hora de tratar con bibliotecas y los libros locales. También se comprobará que a la hora de descargar libros desde el servidor no haya problemas.

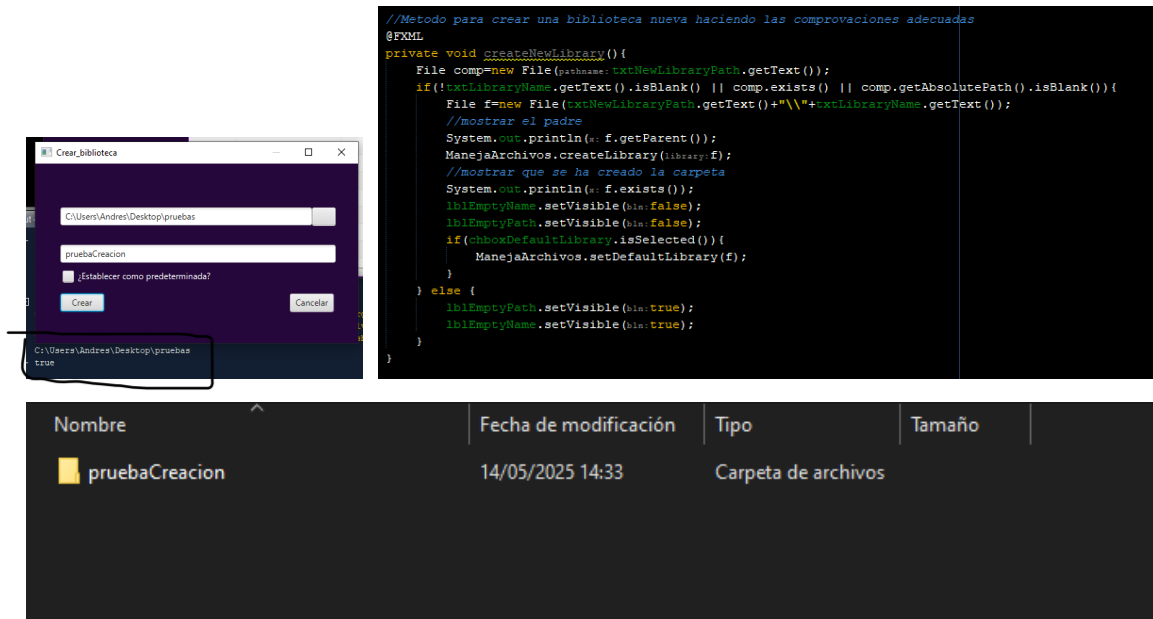
The screenshot shows the 'Dustless' application window. It features a table with columns: Autor, Local, Título, Género, Saga, Idioma, Fecha Public..., and Puntuación. The table lists several books, including 'El nombre del viento' and 'El problema de la paz'. On the right side, there is a panel for the selected book, 'El nombre del viento' by Patrick Rothfuss. This panel includes a book cover image, the title, author, genres, saga, language, publication date, and a synopsis. A 'Descargar' button is located at the bottom of the panel.

```
//esto solo es para las pruebas de libros que no estén en local
ObservableList<Map<String,Object>> o1=FXCollections.observableArrayList();
Map<String,Object>[] libros=new Map[10];
for(int i=0;i<libros.length;i++){
    libros[i]=new HashMap<>();
    libros[i].put(key:"Local", value:"No");
    libros[i].put(key:"Titulo", value:"Libro no local "+i);
    libros[i].put(key:"Genero", value:"Aventuras");
    libros[i].put(key:"Saga", value:"");
    libros[i].put(key:"Idioma", value:"es");
    libros[i].put(key:"Fecha", value:"1998-5-6");
    libros[i].put(key:"Puntuacion", value:"5.9");
    libros[i].put(key:"Autor", value:"Don Juan Palomo");
    libros[i].put(key:"Sinopsis", value:"");
    libros[i].put(key:"Path", value:"");
}
```

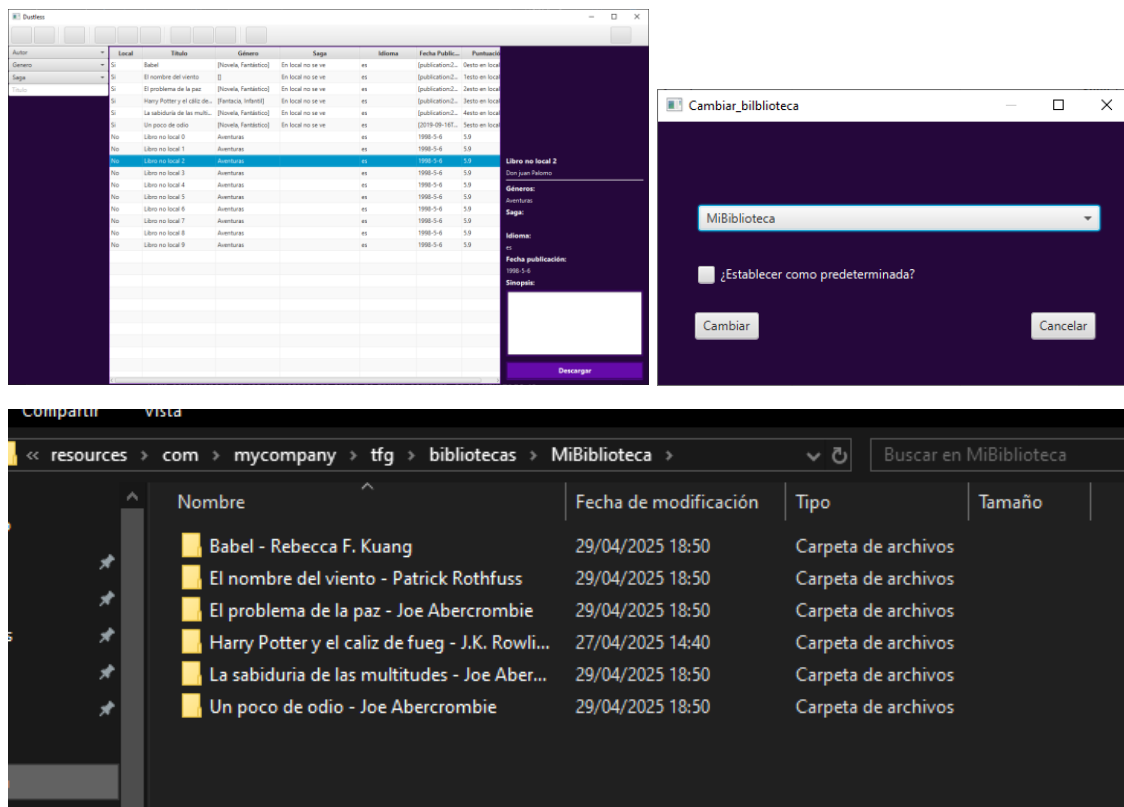
Babel - Rebecca F. Kuang	29/04/2025 18:50
El nombre del viento - Patrick Rothfuss	29/04/2025 18:50
El problema de la paz - Joe Abercrombie	29/04/2025 18:50
Harry Potter y el caliz de fuego - J.K. Rowli...	27/04/2025 14:40
La sabiduría de las multitudes - Joe Aber...	29/04/2025 18:50
Un poco de odio - Joe Abercrombie	29/04/2025 18:50

La primera prueba que se debe realizar en esta sección es la carga inicial de la pantalla. para ello, se hará una llamada a la API que devolverá todos los libros del usuario logeado. Una vez recibida la lista de libros que ha comprado el usuario, se comprobará si este ya cuenta con alguno en la biblioteca local, y se eliminaran los repetidos. A continuación, se extraerán los datos como el título, autor, géneros, etc y se cargarán en un Array de Map. Acto seguido, se llamará a un método que obtendrán los libros de la biblioteca que esté seleccionada como primaria. A diferencia de los libros de la base de datos, para obtener la información de los libros locales será necesario extraerla de los metadatos de los archivos. Con toda la información ya recolectada, se cargará en la tabla.

Otras de las pruebas a llevar a cabos son las del botón de descarga y el botón leer. Para el botón de descarga, se comprobará que una vez pulsado, se cree en la biblioteca seleccionada una carpeta con el libro y la portada, que se obtendrán del servidor, se cambia la propiedad "Local" del libro en la tabla y se cambia el botón por el de leer el libro seleccionado. Por otra parte, el botón de leer, al ser pulsado, deberá abrir una ventana nueva que mostrará el contenido del libro seleccionado.

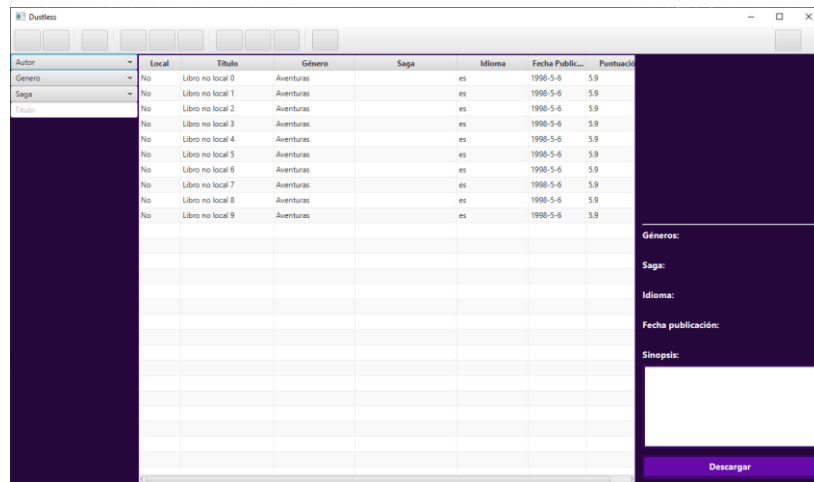


Para comprobar que las bibliotecas se crean de forma correcta, se ha utilizado la consola. Se modificó el código para que imprima por pantalla el padre del archivo y la respuesta al método `exists()` de la clase `File` para saber si se ha creado bien el archivo. También habrá que comprobar que en caso de que se deje marcado el checkbox de biblioteca predeterminada se guarde correctamente en el archivo `properties`.



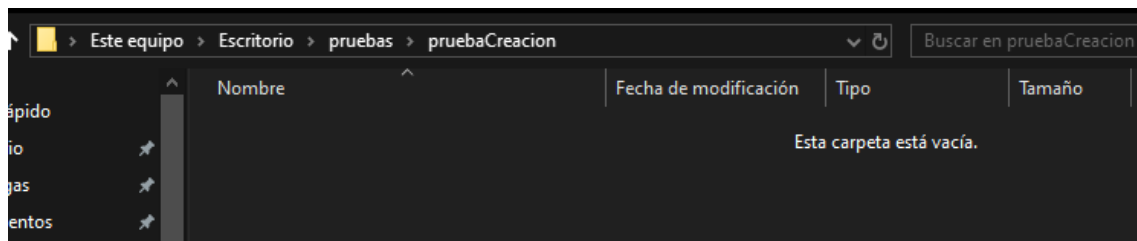
Para comprobar el funcionamiento del cambio de bibliotecas se ha utilizado la biblioteca previamente creada, ya que no contiene ningún libro en local. Al abrir la ventana, el usuario tendrá seleccionada la biblioteca predeterminada y podrá escoger

desde el combobox a cuál cambiar. Para esta prueba se cambiará de la biblioteca predeterminada.

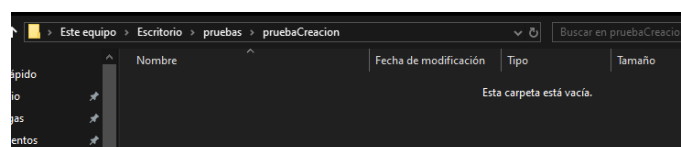
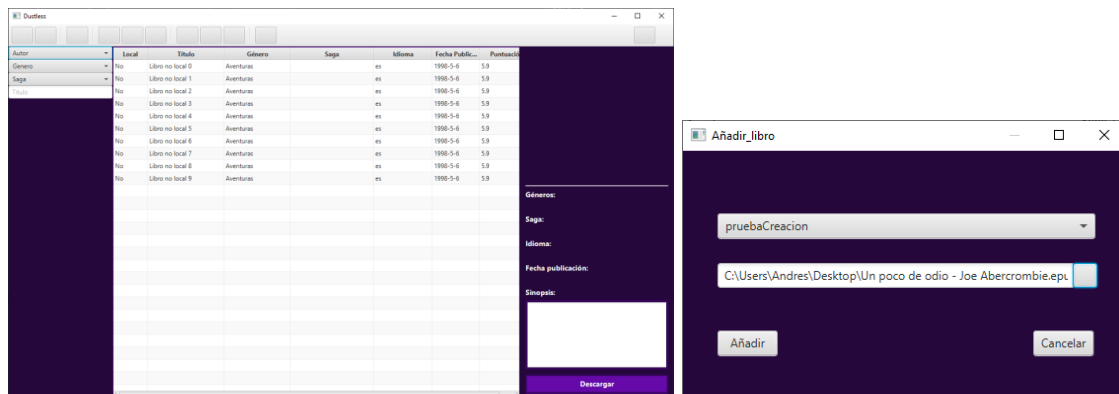


Autor	Local	Título	Género	Saga	Idioma	Fecha Public...	Puntuación
No	Libro no local 0	Aventuras			es	1990-5-6	5.9
No	Libro no local 1	Aventuras			es	1990-5-6	5.9
No	Libro no local 2	Aventuras			es	1990-5-6	5.9
No	Libro no local 3	Aventuras			es	1990-5-6	5.9
No	Libro no local 4	Aventuras			es	1990-5-6	5.9
No	Libro no local 5	Aventuras			es	1990-5-6	5.9
No	Libro no local 6	Aventuras			es	1990-5-6	5.9
No	Libro no local 7	Aventuras			es	1990-5-6	5.9
No	Libro no local 8	Aventuras			es	1990-5-6	5.9
No	Libro no local 9	Aventuras			es	1990-5-6	5.9

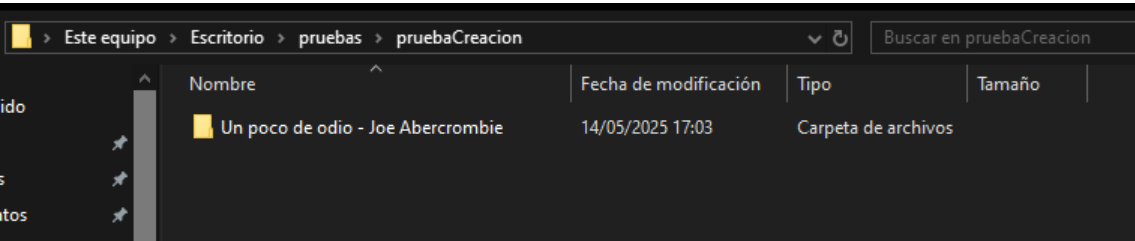
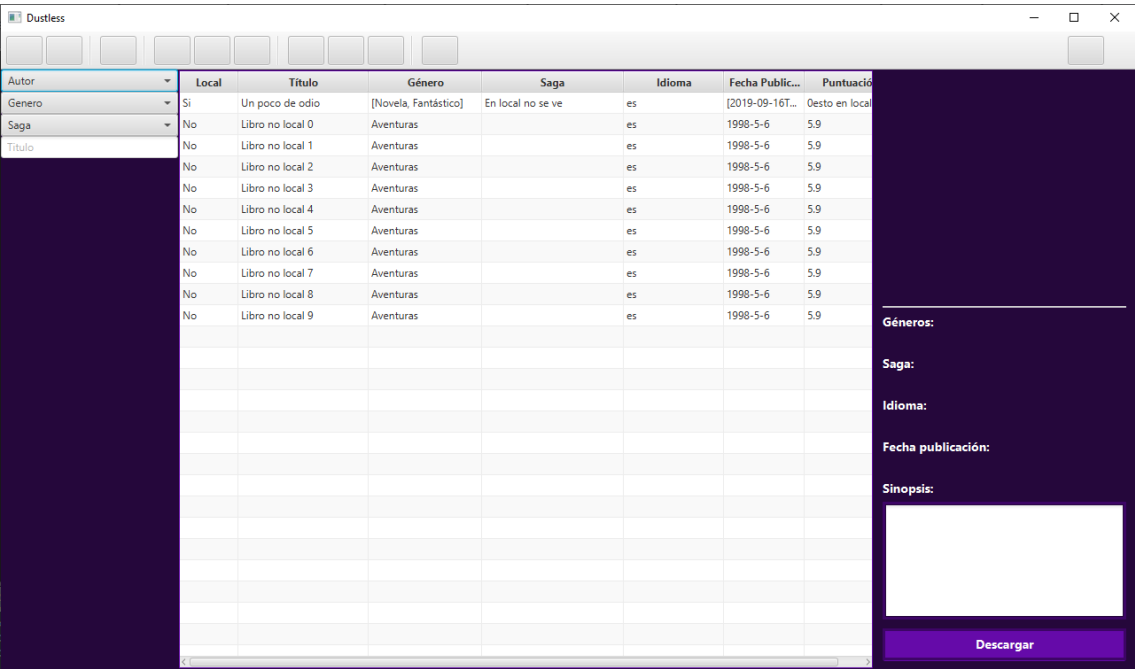
Generos:
Saga:
Idioma:
Fecha publicación:
Sinopsis:
Descargar



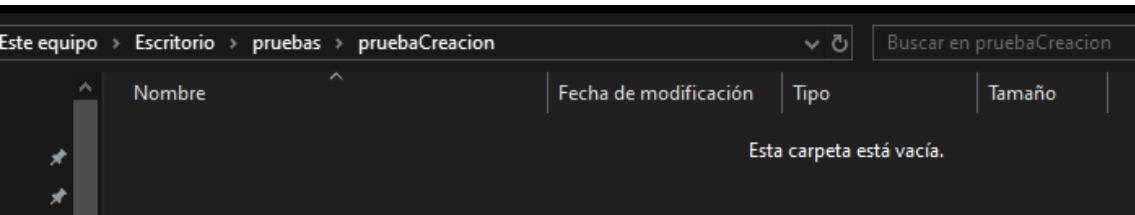
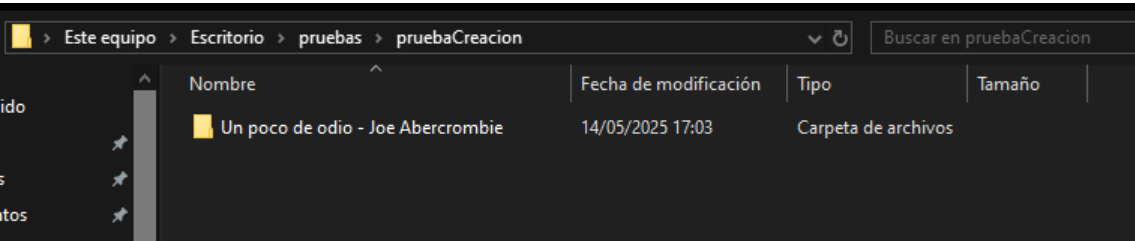
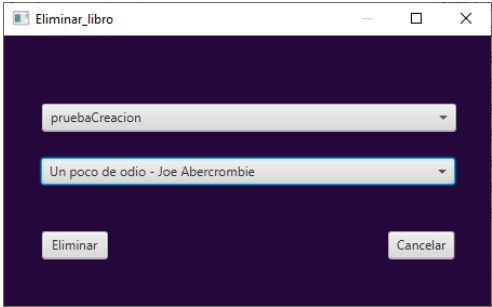
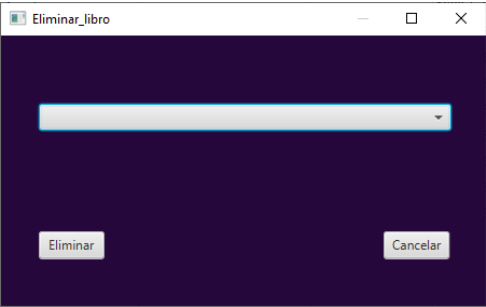
El cambio se verá reflejado directamente en la tabla central, ya que esta pasará a solo tener los libros de la base de datos. Al igual que con la creación de las bibliotecas, será necesario comprobar si se guarda de forma correcta en el archivo .properties en caso de que se deje marcado el checkbox de predeterminada.



Lo siguiente a comprobar será el añadir un libro una de las bibliotecas locales. Al igual que con la prueba anterior, los cambios se verán reflejados en la tabla central y la carpeta a la que apunte la biblioteca. Para evitar posibles fallos con los tipos de archivo, se ha limitado el directoryChooser para que solo reconozca archivos con extensión .epub.



Para comprobar si se eliminan los libros de forma correcta, se eliminará el libro previamente añadido a la biblioteca.

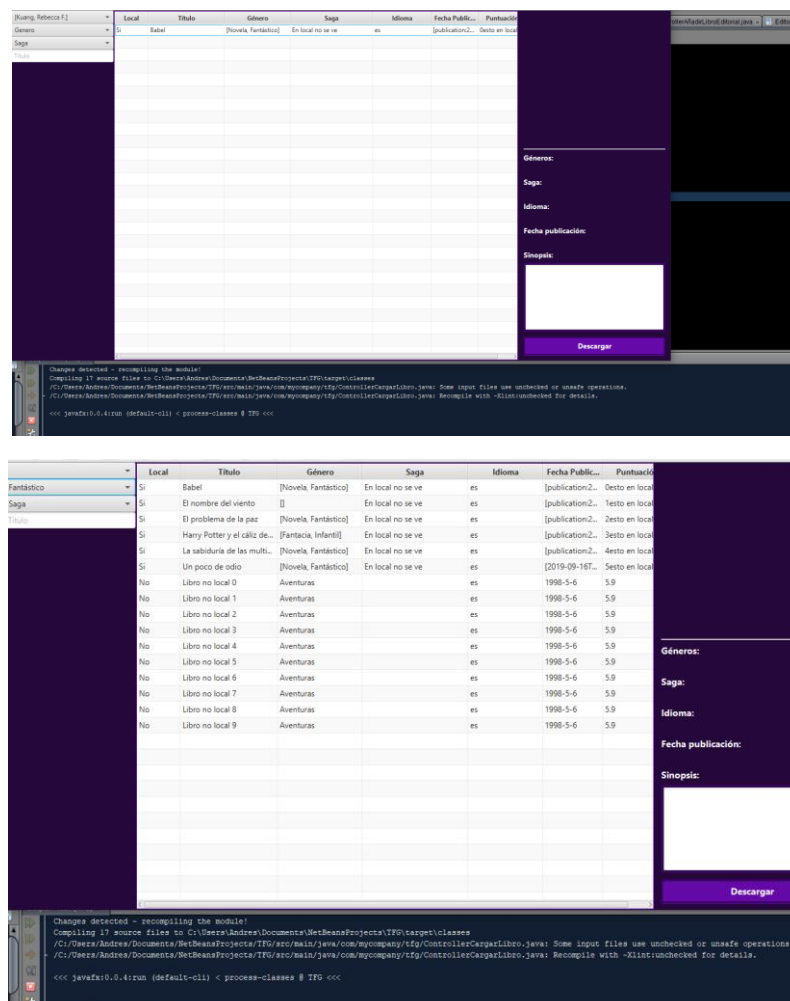


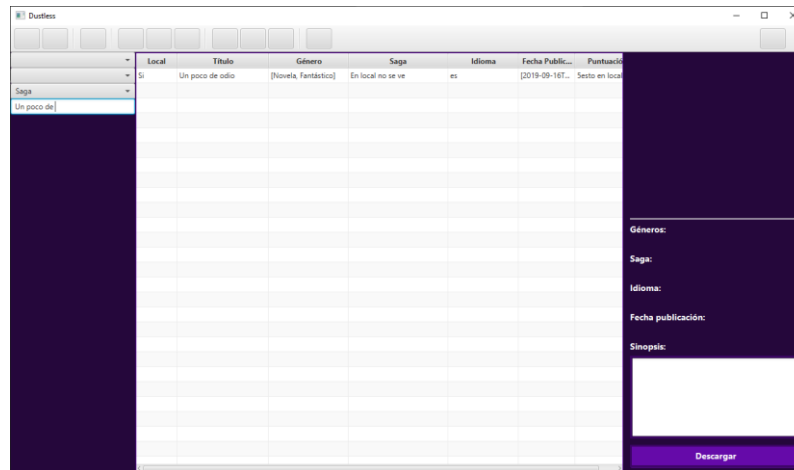
Por último, se comprobará la eliminación de bibliotecas. Para ello una vez pulsado el botón, se comprobará en el explorador de archivos se ha eliminado la carpeta. Como puntos a tener en cuenta, no solo se eliminará la biblioteca, sino que también se eliminará todo su contenido. Para evitar que el usuario borre lo que no debe se ha limitado la capacidad de borrar a solo las bibliotecas. También habrá que comprobar que se ha borrado correctamente la propiedad del archivo .properties.

```
#Wed May 14 14:33:43 CEST 2025
BibliotecaPredeterminada=src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\MiBiblioteca
PathLibrary_MiBiblioteca=src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\MiBiblioteca
PathLibrary_pruebaCreacion=C:\\Users\\Andres\\Desktop\\pruebas\\pruebaCreacion
Recordar=1
Usuario=pruebadeguardado

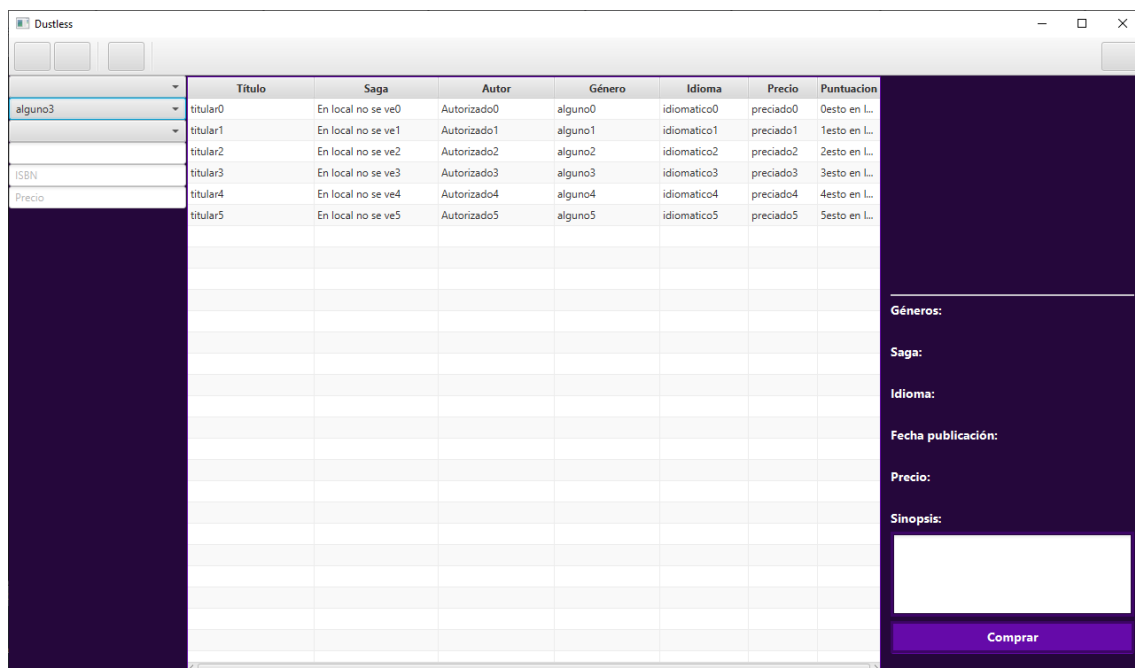
#Wed May 14 18:06:32 CEST 2025
BibliotecaPredeterminada=src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\MiBiblioteca
PathLibrary_MiBiblioteca=src\\main\\resources\\com\\mycompany\\tfg\\bibliotecas\\MiBiblioteca
Recordar=1
Usuario=pruebadeguardado
```

Los filtros se probarán introduciendo un género, un autor, una saga e introduciendo un titulo en el buscador, se podrá ver los resultados en la tabla central y en caso de error se vera a traves de la consola.





13.1.3. Tienda



Al igual que en el apartado de biblioteca, en la ventana tienda habrá que llevar a cabo pruebas para ver si se obtienen los libros de la base de datos, si se carga de forma correcta la información en la tabla, los filtros y la compra de un libro. La única diferencia a tener en cuenta con las pruebas del apartado anterior es el origen de los datos, ya que aquí siempre se obtendrán de la base de datos.

Para comprobar si la compra se realiza correctamente se puede aprovechar el entorno de pruebas de Stripe, lo que nos permitirá simular una venta. Para ello haremos uso de la API key publica de prueba que proporcionan.

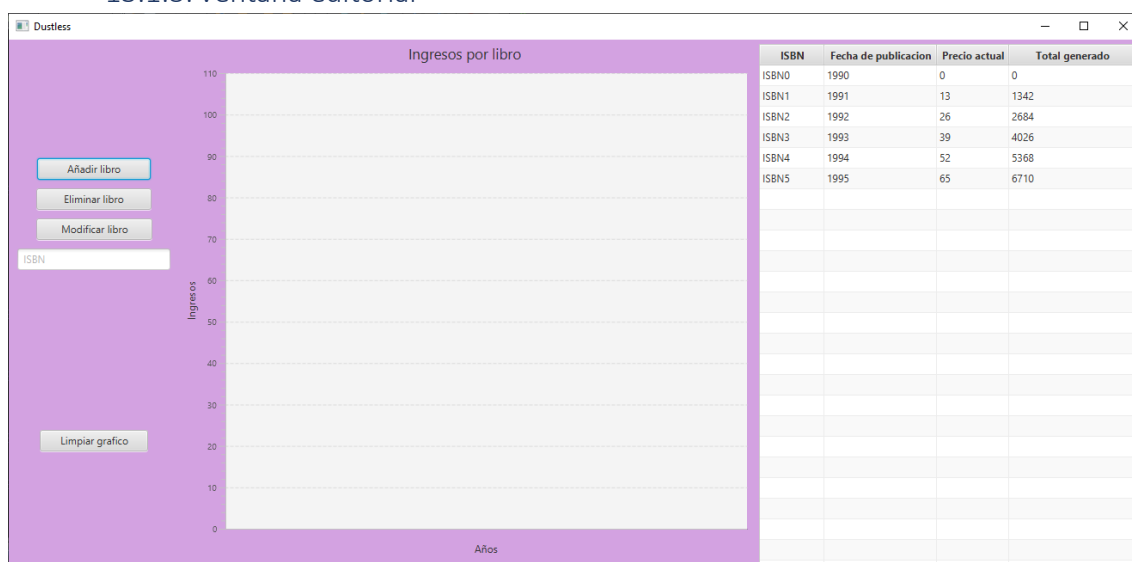
Desde el cliente le pasará al servidor la información necesaria para crear un Checkout Session. Una vez creado, el servidor debería devolver un UR, el cual se utilizará para redirigir al usuario a la página de venta. Dependiendo de si el usuario realiza de forma exitosa la compra o la cancela, será redirigido a su página correspondiente. Si la transacción ha salido bien, la próxima vez que el usuario entre a la página de tienda deberá ver reflejada la compra en la tabla central.

13.1.4. Perfil

En este apartado habrá que comprobar inicialmente que el usuario recibe su información personal y se carga correctamente en la ventana. A su vez, habrá que comprobar que la información se recibe correctamente en la base de datos. Para ello, utilizaremos el perfil de prueba creado al comprobar el login, le cambiaremos los datos y por último le borraremos.

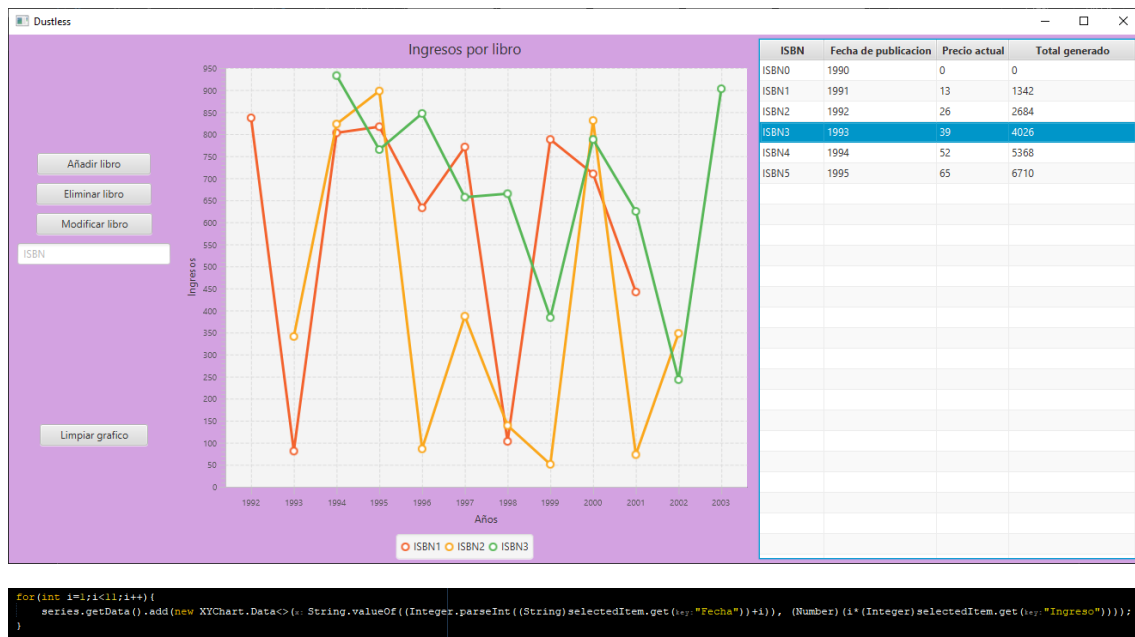
Para comprobar el cambio de contraseña desde la ventana de perfil, se repetirán los pasos que se llevaron a cabo en la sección de login. Estos consistirán en comprobar que se envía el correo a la cuenta asociada con un código de verificación que el usuario deberá introducir en la aplicación y se enviará de vuelta al servidor. Una vez ahí, se comprobará que el código es el mismo. Dependiendo del resultado se devolverá una respuesta u otra y en base a ella la aplicación permitirá al usuario cambiar la contraseña o no. En caso afirmativo, la nueva contraseña deberá pasar por las mismas validaciones que en el login y una vez las cumpla, se cambiará en la base de datos. El apartado de cambio de correo seguirá los mismos pasos que el cambio de contraseña por lo que las comprobaciones serán las mismas.

13.1.5. Ventana editorial



En este apartado se comprobará que la aplicación recibe los datos de la base de datos de forma correcta, que estos se pueden borrar y alterar sin problemas. También habrá que comprobar que la gráfica representa los datos de forma correcta.

Nada mas iniciar esta pantalla, se cargarán en la tabla los libros pertenecientes a esa editorial. Se comparará con los datos de la base de datos para asegurarse de que devuelve los mismos registros. Para comprobar las modificaciones, se llevarán a cabo cambios desde la aplicación y se comprobará en la base de datos si se ha guardado correctamente. Para el borrado se eliminará un libro y se comprobará en la base de datos si se ha borrado como es debido.



Para comprobar que el grafico muestra de forma correcta, se ha utilizado la clase Random para crear datos de prueba. En

14. Bibliográfica

- <https://github.com/stripe/stripe-java> - API para manejar pagos
- <https://github.com/tesseract-ocr/tessdoc> - OCR usado para el reconocimiento de imágenes
- <https://github.com/nguyenq/tess4j> - Wrapper para tesseract
- <https://github.com/psiegman/epublib> - Libreria para manejar ebooks
- https://web.stanford.edu/class/ee368/Project_Autumn_1617/Reports/report_yang_shen.pdf - Procesamiento de imágenes
- <https://medium.com/analytics-vidhya/a-hitchhikers-guide-to-ocr-8b869f4e3743> - Procesamiento de imágenes.
- <https://opencv.org/> - Apoyo para el tratamiento de imágenes
- <https://github.com/stripe/stripe-java> - Server side SDK pagos
- <https://docs.stripe.com/sdks/android> - Móvil SDK pagos
- Ejemplos uso stripe
 - <https://docs.stripe.com/sdks/server-side?examples=list&lang=java>
 - <https://docs.stripe.com/payments/accept-a-payment?platform=android&ui=payment-sheet>
 - <https://docs.stripe.com/api/checkout/sessions/create>