Aplicación multiplataforma de compra y gestión de libros electrónicos

Andrés García Payá y Alexis Willy Andía Rosales

Contenido

[1. Resumen 4](#_Toc196834692)

[2. Introducción 4](#_Toc196834693)

[3. Objetivos 5](#_Toc196834694)

[4. Palabras clave 5](#_Toc196834695)

[5. Análisis de contexto 6](#_Toc196834696)

[5.1. Análisis del Contexto 6](#_Toc196834697)

[5.1.1. Competencia 6](#_Toc196834698)

[5.1.2. DAFO del Proyecto 7](#_Toc196834699)

[5.1.2.1. Fortalezas 7](#_Toc196834700)

[5.1.2.2. Debilidades 7](#_Toc196834701)

[5.1.2.3. Oportunidades 7](#_Toc196834702)

[5.1.2.4. Amenazas 7](#_Toc196834703)

[5.2. Innovación 7](#_Toc196834704)

[6. Modelo de datos 8](#_Toc196834705)

[6.1. Resumen 8](#_Toc196834706)

[6.2. Tablas 8](#_Toc196834707)

[6.2.1. Usuario 8](#_Toc196834708)

[6.2.2. Biblioteca 8](#_Toc196834709)

[6.2.3. Libro 9](#_Toc196834710)

[6.2.4. Deseo 9](#_Toc196834711)

[6.2.5. Biblio\_libro 10](#_Toc196834712)

[6.2.6. Saga 10](#_Toc196834713)

[6.2.7. Idioma 10](#_Toc196834714)

[6.2.8. Genero 10](#_Toc196834715)

[6.2.9. Autor 10](#_Toc196834716)

[6.2.10. Libro\_idioma 11](#_Toc196834717)

[6.2.11. Libro\_autor 11](#_Toc196834718)

[6.2.12. Libro\_genero 11](#_Toc196834719)

[6.2.13. Editorial 11](#_Toc196834720)

[6.2.14. Libro\_edit 12](#_Toc196834721)

[6.3. Diagrama E.R 12](#_Toc196834722)

[7. Diseño 12](#_Toc196834723)

[7.1. Elementos comunes 13](#_Toc196834724)

[7.1.1. Login, registro, recuperar contraseña 13](#_Toc196834725)

[7.2. UI de la aplicación de escritorio 14](#_Toc196834726)

[7.2.1. Vista usuario 14](#_Toc196834727)

[7.2.2. Vista Editorial 15](#_Toc196834728)

[7.3. UI de la aplicación móvil 16](#_Toc196834729)

[8. Casos de uso 17](#_Toc196834730)

[9. Diagrama de clases 18](#_Toc196834731)

[9.1. Versión de escritorio 18](#_Toc196834732)

[9.1.1. ControllerLogin 18](#_Toc196834733)

[9.1.2. ManejaImagen 19](#_Toc196834734)

[9.1.3. ManejaBibliotecaLocal 19](#_Toc196834735)

[9.1.4. ManejaBibliotecaLibro 19](#_Toc196834736)

[9.1.5. ControllerBiblioteca 19](#_Toc196834737)

[9.1.6. ControllerTienda 19](#_Toc196834738)

[9.1.7. Conexión 19](#_Toc196834739)

[9.2. UML del API REST 20](#_Toc196834740)

[9.3. UML del Servidor 21](#_Toc196834741)

[9.4. UML de Android 22](#_Toc196834742)

[10. Planificación 22](#_Toc196834743)

[10.1. Diagrama de Gantt 22](#_Toc196834744)

[10.2. Definición de Recursos y Logística 23](#_Toc196834745)

[11. Implementación 24](#_Toc196834746)

[11.1. Aplicación de escritorio 24](#_Toc196834747)

[11.1.1. Introducción 24](#_Toc196834748)

[11.1.2. Clase Properties 24](#_Toc196834749)

[11.1.3. Clase ManejaArchivos 26](#_Toc196834750)

[11.1.4. Login 28](#_Toc196834751)

[11.1.5. Biblioteca 32](#_Toc196834752)

[11.2. OCR 34](#_Toc196834753)

[11.2.1. Introducción 34](#_Toc196834754)

[11.2.2. Implementación 35](#_Toc196834755)

[11.2.3. Tratamiento de imagen 35](#_Toc196834756)

[11.2.4. Futuras mejoras 36](#_Toc196834757)

[11.2.5. Motivos de la no implementación 36](#_Toc196834758)

[11.3. Pagos 36](#_Toc196834759)

[11.3.1. Introducción 36](#_Toc196834760)

[11.3.2. Implementación 36](#_Toc196834761)

[11.3.3. Flujo del proceso 37](#_Toc196834762)

[12. Bibliográfica 37](#_Toc196834763)

# 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.

# 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 unas forma sencilla, eficiente, centralizada y menos restrictiva que el reste de opciones del mercado.

# 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 brindara 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 a la tienda. Para llevar a cabo esto se utilizará tesseract OCR, que estará integrado en un servidor y al que se le enviaran las fotos a través de una API. Ya que uno de los motivos de la aplicación es no depender en sobremedida de grandes se empresas se ha optado por este OCR sobre Google Cloud visión.

La versión de escritorio contara 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 las permitirá subir libros al servidor, aumentando así su catálogo disponible en la tienda. Para ello, se les asignara 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.

# Palabras clave

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

# Análisis de contexto

## 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.

## 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.

## DAFO del Proyecto

### 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.

### 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.

### 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.

### Amenazas

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

## 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.

# Modelo de datos

## Resumen

Para hacer la base 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 basa relacional orientada a objetos, lo que la hace ideal cuando se usa en conjunto de un lenguaje vasado 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.

## Tablas

### 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 haseada 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 creo 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.

### 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 está en varias bibliotecas y barias bibliotecas pueden tener un libro

### 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\_publi:** 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.

### 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

### 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.

### 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.

### 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.

### 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.

### Autor

* **Resumen:** Esta tabla contiene información básica del autor, ya que son las editoriales quienes tratan con ellos.
* **Campso:**
  + **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.

### 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.

### 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.

### Libro\_genero

* **Resumen:** Es la tabla intermedia creada a partir de la relación n:m entre la tabla libro y la tabla género.
* **Campso:**
  + **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.

### 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.

### 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.

## Interfaz de usuario gráfica Descripción generada automáticamenteDiagrama E.R

# 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 el en esta sección.

## Elementos comunes

### Login, registro, recuperar contraseña

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

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.

## 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.

### Vista usuario

Diagrama

Descripción generada automáticamente

En la imagen se puede ver el flujo de las ventanas que de un usuario. La primera ventana que vera siempre nada mas 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 titulo. 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 contara 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.

Interfaz de usuario gráfica

Descripción generada automáticamente

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.

### Vista Editorial

Captura de pantalla de un celular

Descripción generada automáticamente

Ya que la editorial no es realmente un usuario que vaya a consumir la aplicación para otra cosa que no sean negocios, se le proporcionara 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.

## 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.

Diagrama

Descripción generada automáticamente

Al igual que con la versión de escritorio, lo primero que vera el usuario nada mas 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 activity, 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.

# Casos de uso

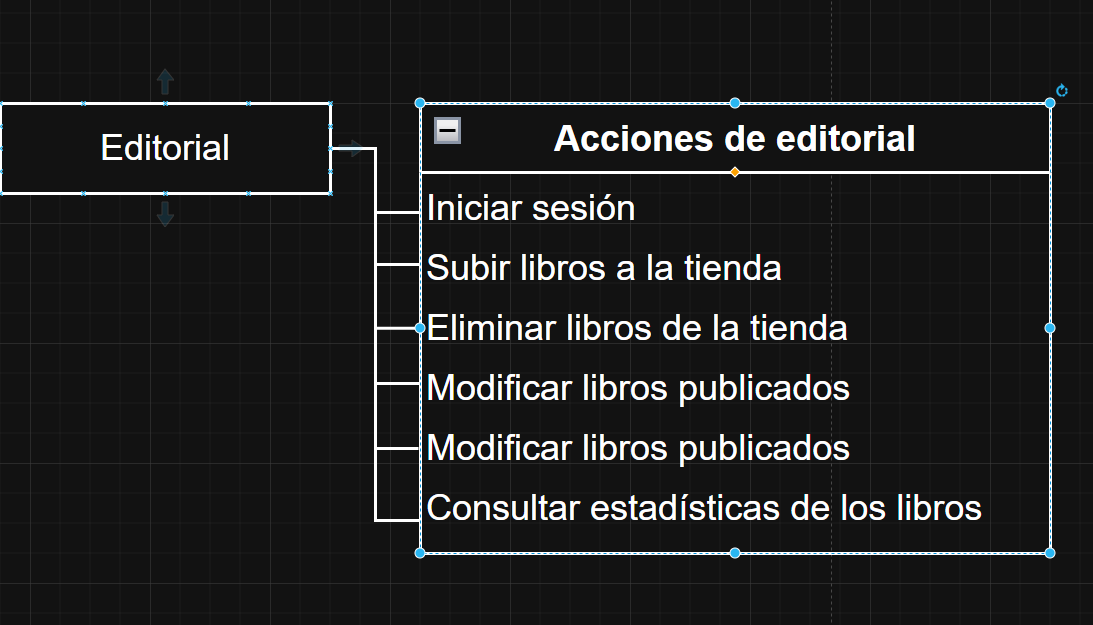
La aplicación estará enfocada en dos tipos de usuarios distintos, las empresas o editoriales y los usuarios comunes, siendo estos últimos los usuarios a los que está enfocada principalmente y que más uso le darán. Ya que el proyecto consta de dos aplicaciones, se procederá a explicar que usuario tiene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

**Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.**

* **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.

****

# Diagrama de clases

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

## Versión de escritorio

Pantalla de computadora

Descripción generada automáticamente con confianza media

### ControllerLogin

Esta clase será la encargada de controlar tanto la interfaz como la funcionalidad que incumban 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.

### 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.

### ManejaBibliotecaLocal

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.

### ManejaBibliotecaLibro

Clase centrada en la gestión del contenido de las bibliotecas de libros electrónicos. Utilizará la librería epulib para acceder a los metadatos de los archivos EPUB y extraer información relevante como título, autor y el número de páginas de aquellos libros que haya añadido el usuario a su biblioteca local.

### 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.

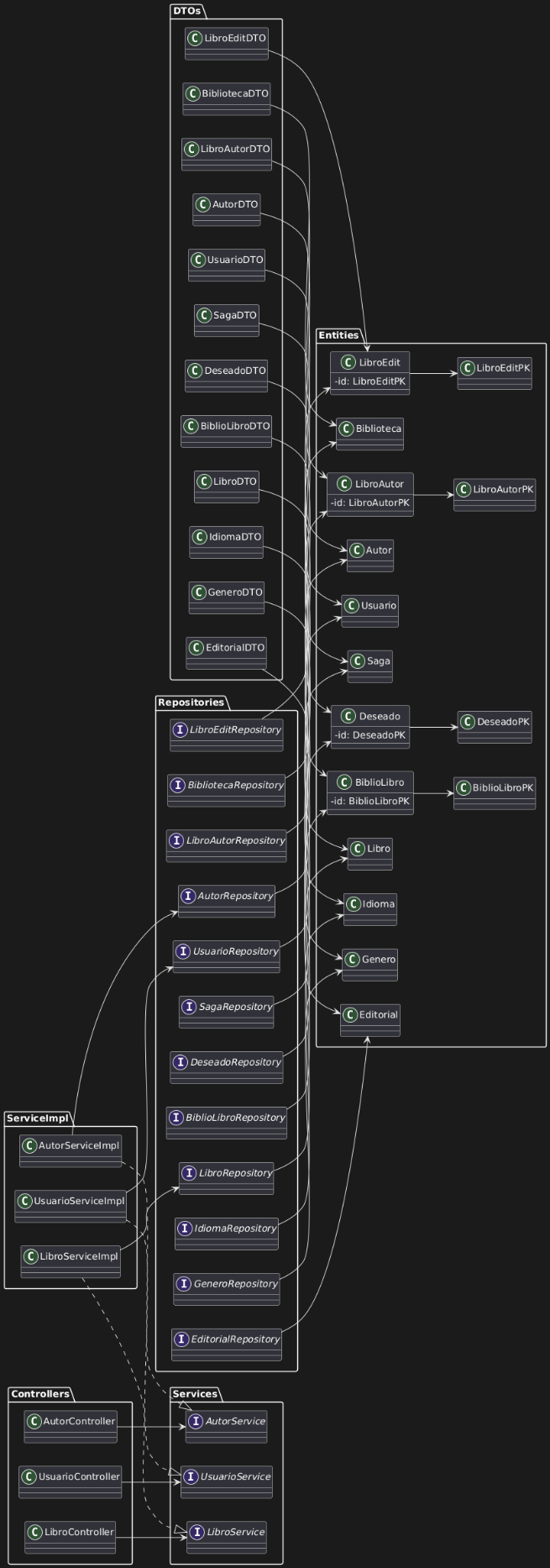
### ControllerTienda

Esta clase será la encargada de manejar la interfaz gráfica referente a la tienda. Trabajará 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.

### Conexión

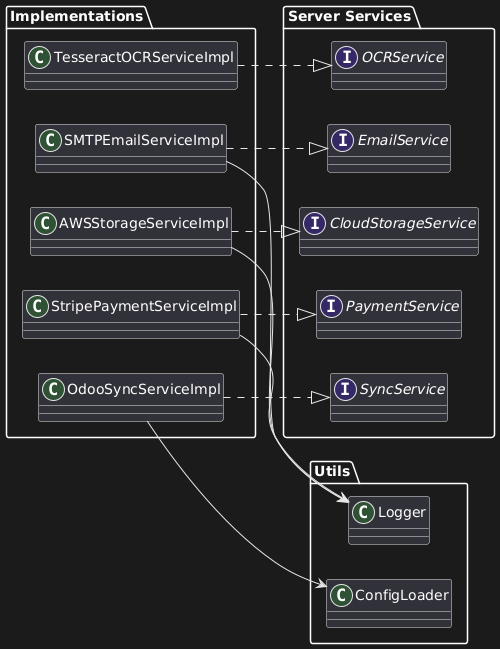
Clase responsable de establecer y gestionar la comunicación entre la aplicación de escritorio y el servidor. Esta clase será la encargada de consumir la API que conecta con la base de datos, enviando y recibiendo datos de forma segura. Implementará métodos para realizar operaciones como login, registro,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

## UML del API REST



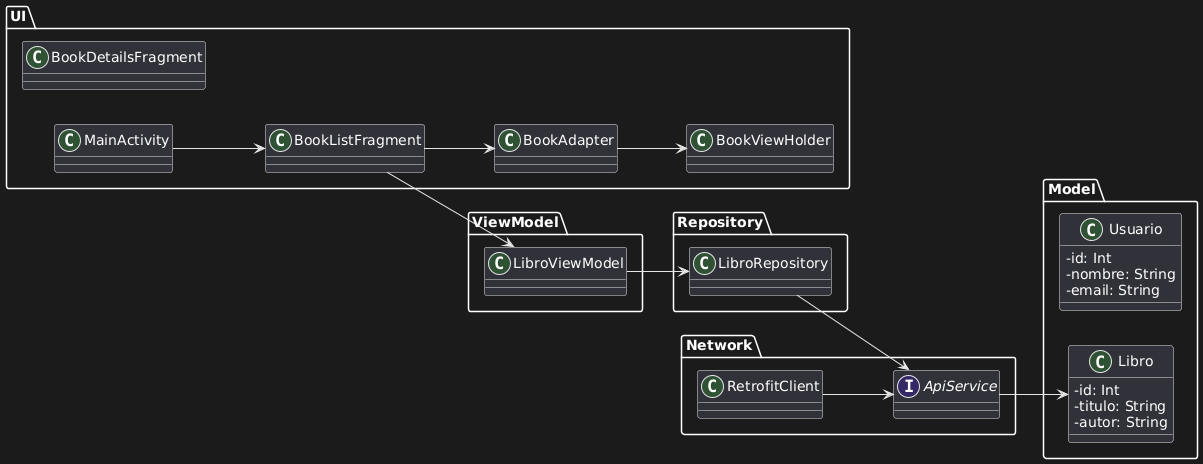
Este diagrama representa la arquitectura de una API RESTful construida con Spring Boot. Organiza el código en capas: modelos (entidades), DTOs, repositorios, servicios, implementaciones de servicio y controladores. Cada capa se comunica con la siguiente para estructurar una aplicación mantenible, clara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

## 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.

## 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.

# 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.

## 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.

## 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.

# Implementación

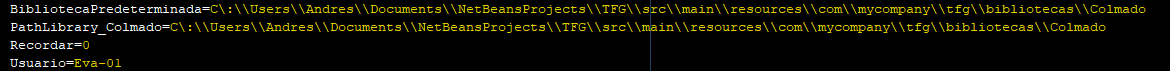
## Aplicación de escritorio

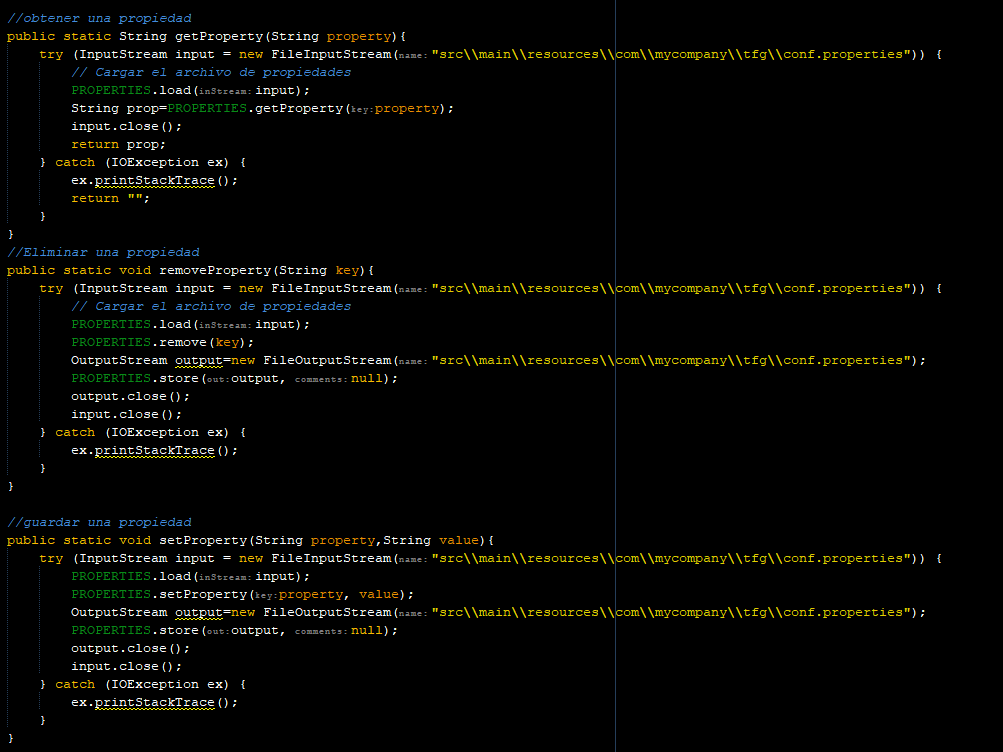
### Introducción

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.

### Clase Properties

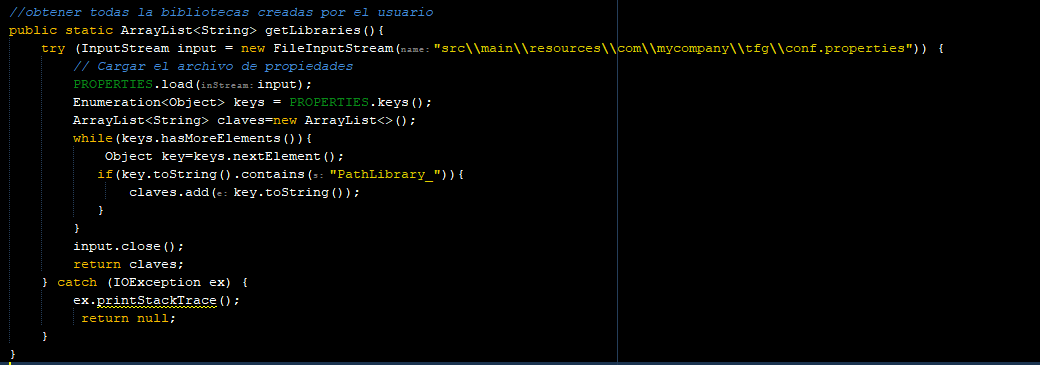
Esta clase es la encargada de darle acceso a las demás clases al archivo .properties, que es donde se guardan cosas como si el usuario quiere que se le recuerde, los paths a las distintas librerías, etc. Para ello cuenta con métodos genéricos para eliminar, añadir y modificar las propiedades pasándole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





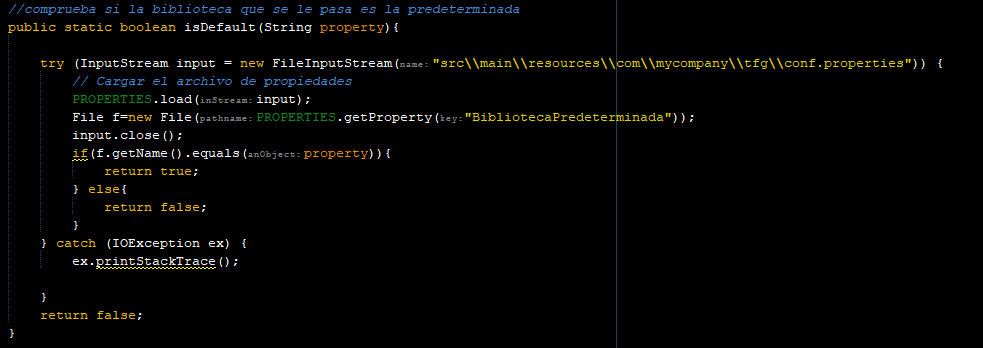
Como se puede ver en la imagen, estos métodos acceden a un archivo .properties que va incluido en el proyecto. Para acceder a él, primero hay que crear un InputStream para que el programa sea capaz de leer el archivo y utilizando la clase Properties de java se carga el archivo en una variable. En caso 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.





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.

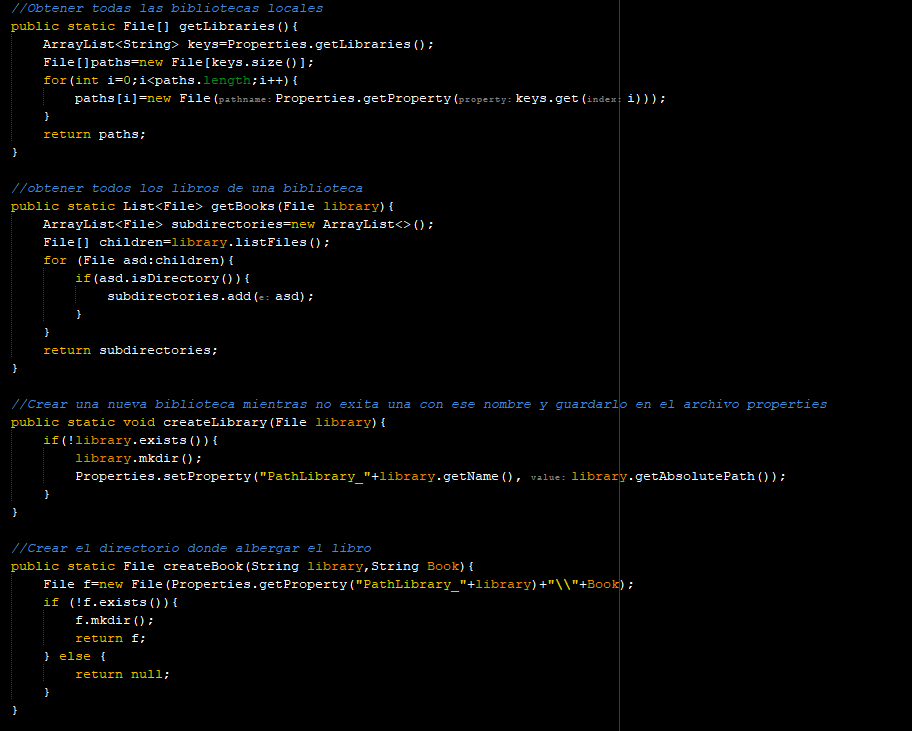




Para saber si una biblioteca está guardada como predeterminada, primero se obtiene la propiedad correspondiente usando el argumento property que pide el método. Este argumento será el nombre de la biblioteca a eliminar. Posteriormente se crea una variable tipo File ya que la propiedad es un path, y usando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

### Clase ManejaArchivos

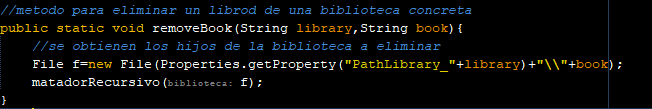
Esta clase será la encargada de interactuar con directorios, ya sea creándolos, eliminándolos o modificándolos. También será la encargada de manejar las propiedades del archivo .properties relacionadas con las rutas de las bibliotecas y los libros.



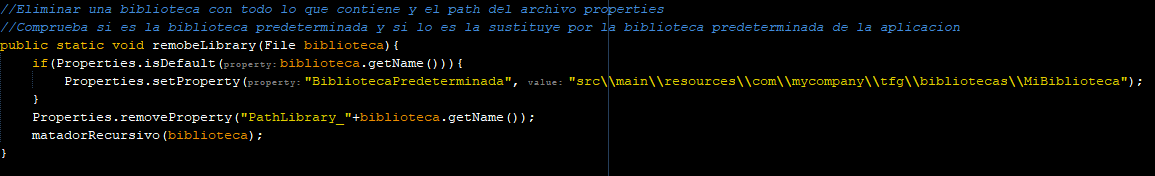
Cuenta con métodos para obtener las bibliotecas creadas por el usuario, obtener los libros de una biblioteca concreta, crear una biblioteca, crear el directorio donde se va a guardar el libro, etc. Aquí se puede ver en uso la clase Properties, por ejemplo, en el método createLibrary, para crear una nueva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



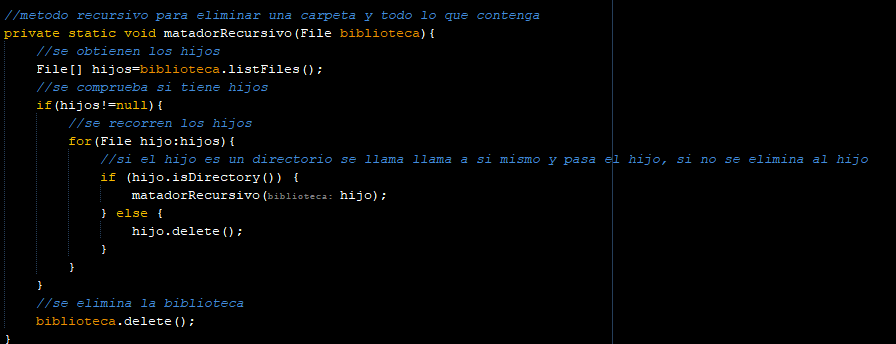
Esta clase también cuenta con métodos para eliminar tanto los libros como las bibliotecas. Para llevarlo a cabo se usan dos métodos, uno específico para eliminar un libro o una biblioteca entera y un método recursivo para eliminar todo el contenido. Es necesario hacer la distinción a la hora de eliminar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



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 ultimo se llama al método recursivo.

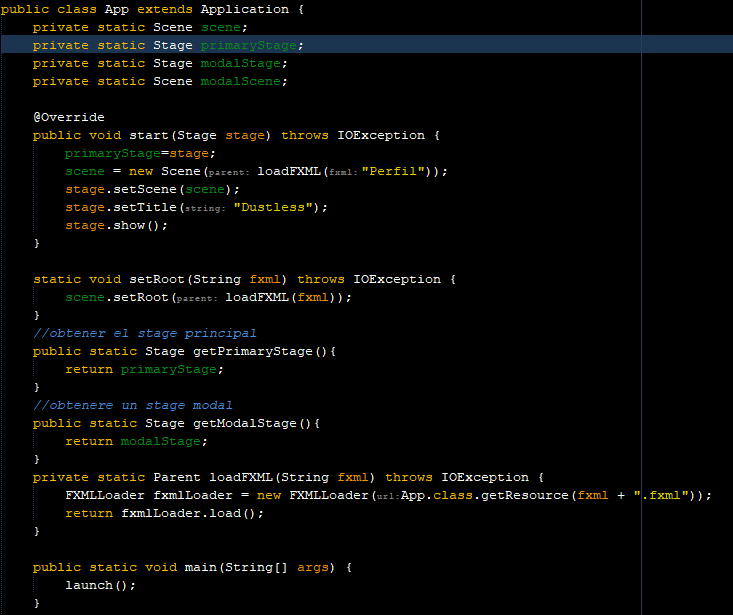


Por otra parte, para eliminar una biblioteca solo se necesita el nombre de la esta. Primero se comprueba si la biblioteca esta guardada como predeterminada y en caso afirmativo se sustituye por la ruta de la biblioteca predeterminada viene al principio. Por último se obtiene el path absoluta de la biblioteca 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.



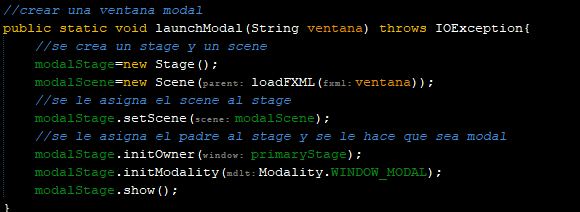
El método recursivo empieza por obtener todos los hijos del directorio que se le pase y se guarda en un Array de tipo File. Se comprueba si es nulo y en caso de que no lo sea, se procede a recorrer con un for el array de hijos. En caso de que el hijo sea un directorio, se llama a si mismo pasando la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

### Login



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 sera la ventana que se abra cada vez que se abra una ventana modal y el método para crear el stage modal.



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.

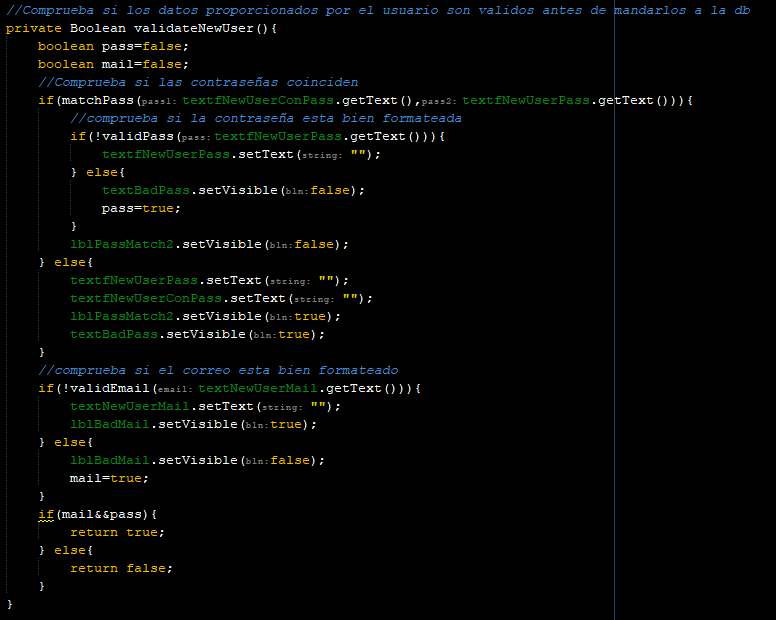
Para implementar la lógica a la interfaz gráfica hecha con javafx es necesario crear una clase controlador. Se ha creado una clase controlador emparejará con su respectivo archivo .fxml para cada vista de la aplicación. Para hace la clase controlador simplemente hay que asignarle el controlador desde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

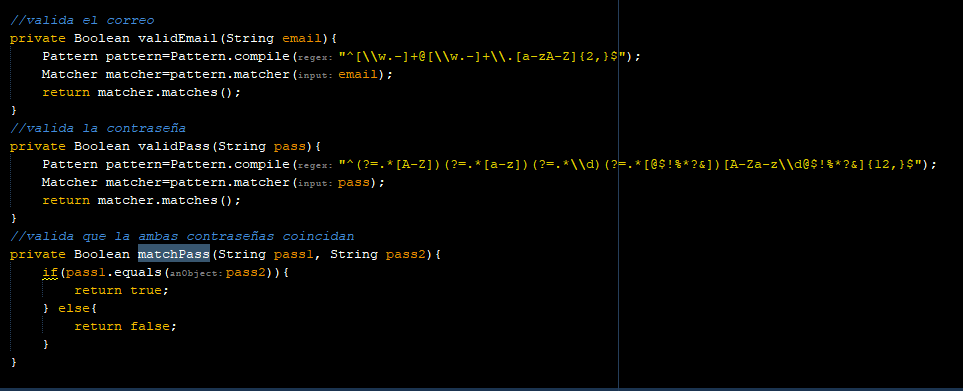






Para poder acceder a los objetos de javafx creados en el archivo .fxml se utilizará la anotación @FXML en la declaración del objeto y en la creación de los métodos, como pueda ser el que se ejecuta al hacer click en un botón. Hay que tener en cuenta que si se le asigna uno de estos métodos a un control y no se implementa saltará una excepción.





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.

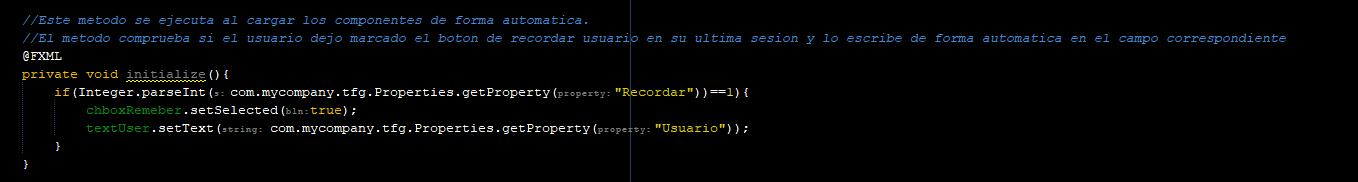
Para comprobar el correo coge el texto del TextField correspondiente y se lo pasa a un método que utilizando una expresión regular buscara si se cumple un formato y devuelve el resultado. En caso de que el correo sea válido se guardará en una variable de tipo booleana la confirmación, en caso negativo,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

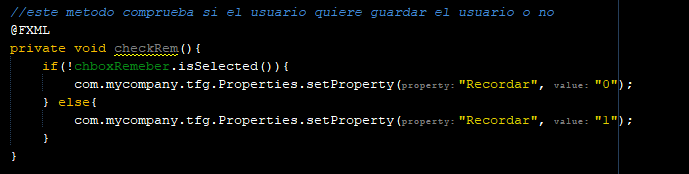


El controlador del login también cuenta con métodos para manejar el autocompletar del campo usuario a la hora de iniciar sesión. Lo primero es comprobar si el usuario marcó la opción la última vez o no. Para ello se comprueba la propiedad “Recordar” del archivo .properties. En caso de que dejase marcado 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.





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.



### 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.

Interfaz de usuario gráfica, Aplicación, Tabla, Excel

El contenido generado por IA puede ser incorrecto.

Nada mas 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 vallan a mostrar como String para evitar posibles fallos.

Texto

El contenido generado por IA puede ser incorrecto.

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 esta 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.

Texto

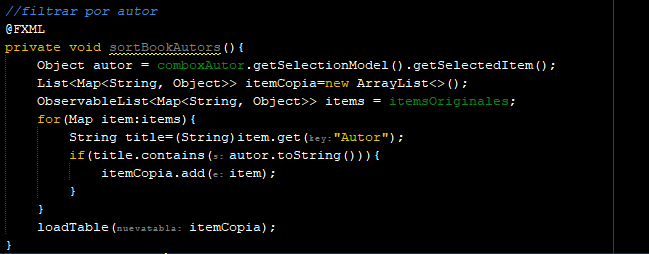
El contenido generado por IA puede ser incorrecto.

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.

Texto

El contenido generado por IA puede ser incorrecto.

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.



También se puede filtrar por género, autor y saga. Los métodos recogen el item 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().

## OCR

### 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.

### 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 tratara 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++.

### 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.

### 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

### 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 llevaria a errores imprevistos.

## Pagos

### 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 done 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.

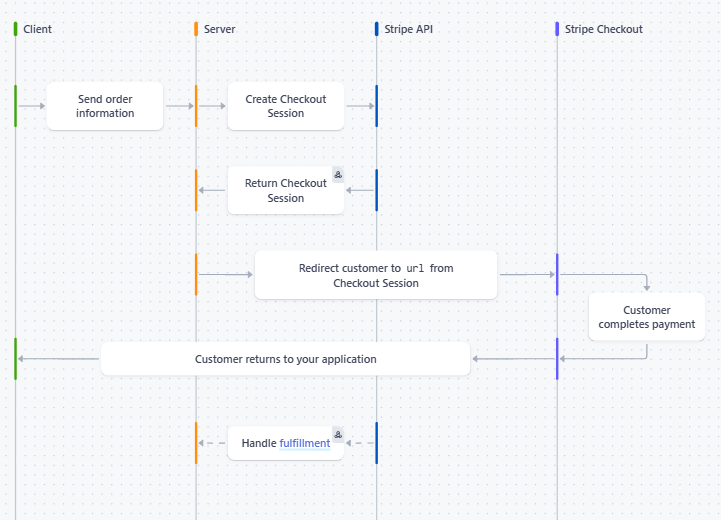
Si bien Stripe cuenta con distintas formas de tratar con los pagos, nos hemos decantado por utilizar Checkout Session ya que es una forma sencilla para manejar pagos únicos o subscripciones ya que delega gran parte del proceso a Stripe. Si se viese necesario también se podría manejar de forma más “personal”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

### 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 mandara 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.

### Flujo del proceso

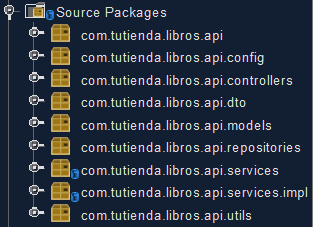


## 11.2. API RESTful

### 11.2.1. 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)



### 11.2.2. 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

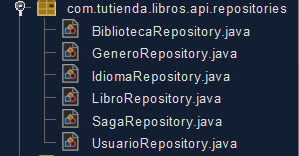
* Jerarquía de categorías
* Sistema de etiquetado

Idioma

* Traducciones disponibles
* Metadatos específicos por idioma

### 11.2.3. Repositorios

Implementan el patrón Repository para acceso a datos.



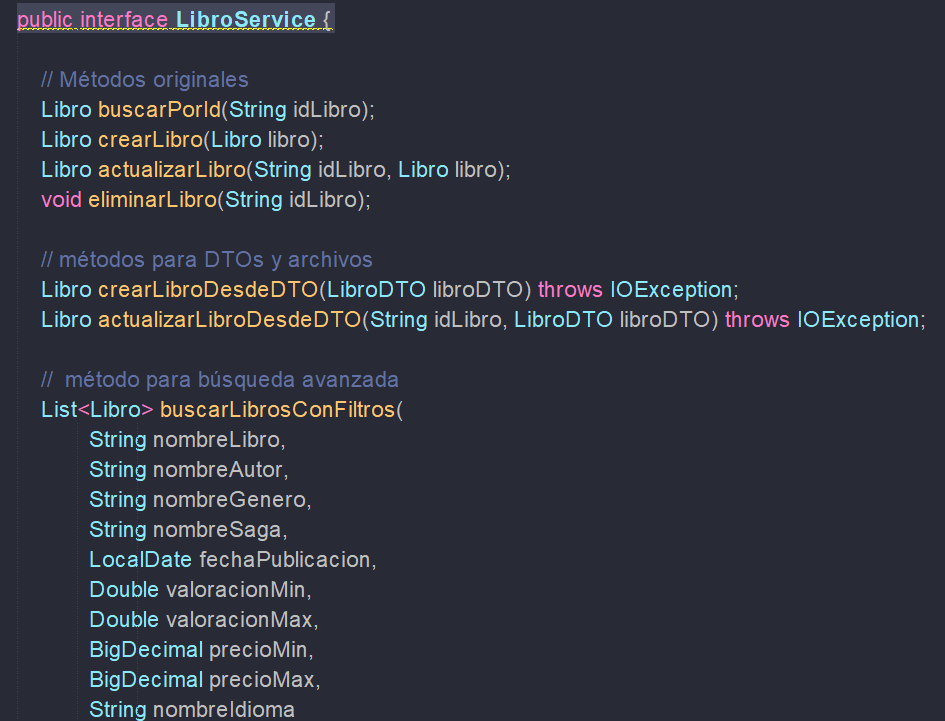
### 11.2.4. Servicios

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.

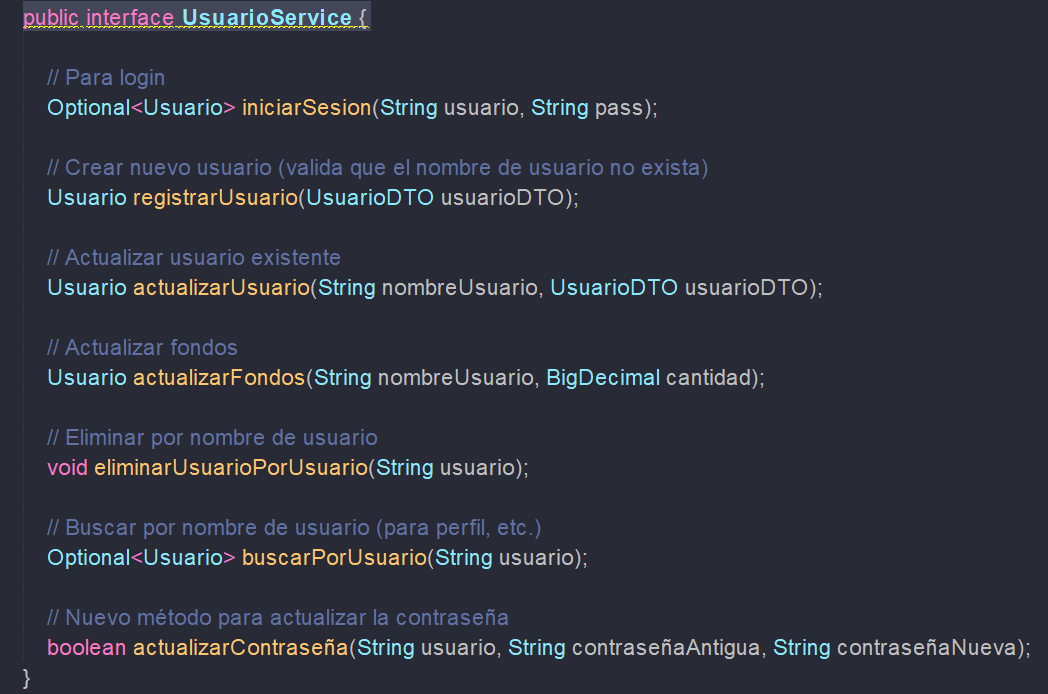


UsuarioService

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.



GeneroService

Descripción: Lógica de negocio para la gestión de géneros.

Métodos clave:

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

IdiomaService

Descripción: Lógica de negocio para la gestión de idiomas.

Métodos clave:

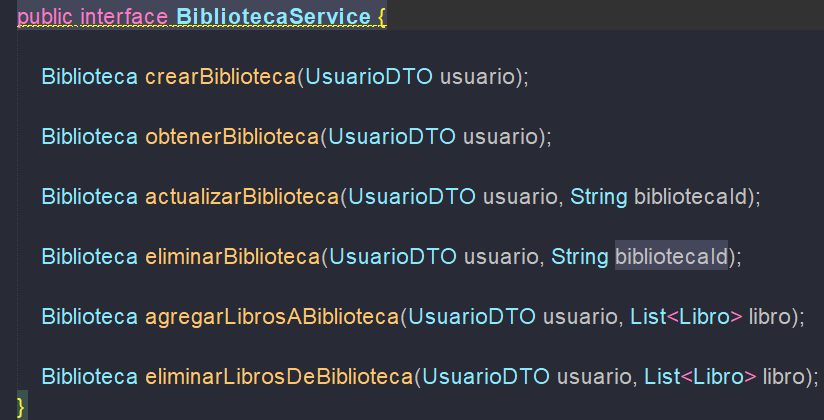
* obtenerIdiomas(): Obtiene una lista de todos los idiomas disponibles en el sistema.

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.



### 11.2.5. 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

### 11.2.6. 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

### 11.2.7. Controladores

Los endpoints REST se organizan por recurso:

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.

GeneroController

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

IdiomaController

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

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.

UsuarioController

* + POST /api/usuarios/registrar: 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.

### 11.2.8. 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

# 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://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>