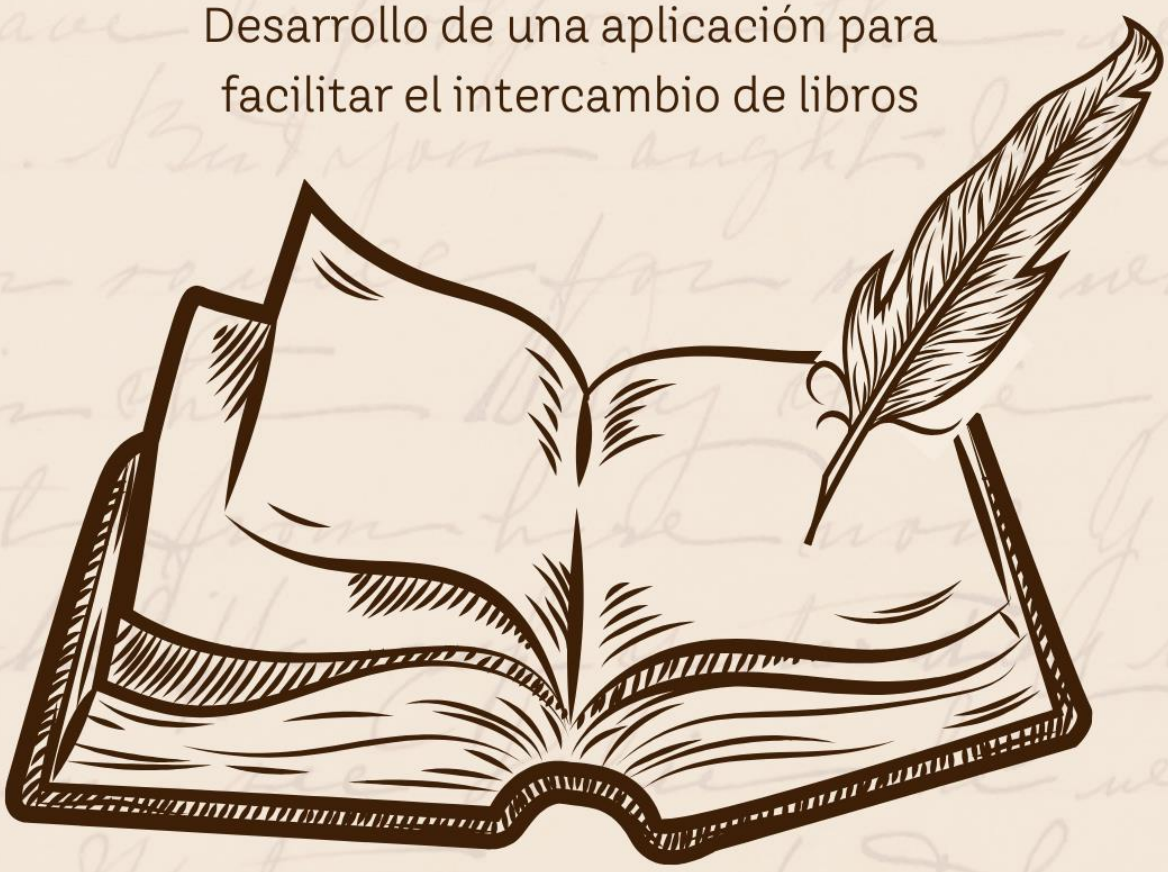
UNIR FP

Trabajo Fin de Ciclo

SCRIPTUM

Desarrollo de una aplicación para facilitar el intercambio de libros



Elvira Medina Velilla
María Moreno Rodríguez
Álvaro Tercero Nieves

Tutora: Raquel Cerdá

Índice

1. Definición del proyecto

La digitalización de la información que se ha producido en las últimas décadas ha supuesto una serie de cambios en los hábitos de consumo que han afectado a muchas de las formas de manifestación cultural tradicionales, como por ejemplo la música, el cine, la fotografía o la lectura. Por ejemplo, en la comercialización de estos productos de carácter cultural se ha abandonado el formato físico, sustituyéndose por otro digital.

La lectura es uno de los pocos casos en los que la digitalización no ha modificado de manera radical el modelo de distribución, sino que lo ha hecho de manera parcial (López Suárez & Larrañaga Rubio, 2010). Muchos lectores siguen prefiriendo el formato físico tradicional en papel debido principalmente a que permite una lectura prolongada sin forzar la vista, frente a otros soportes digitales que, en mayor o menor medida, implican un mayor cansancio.

Sin embargo, otros cambios sociales y culturales están provocando una disminución en la compra de libros en papel. Muchas personas viven en hogares pequeños y apenas cuentan con espacio para poder mantener una biblioteca personal de tamaño considerable, por lo que pueden terminar desprendiéndose de algunos ejemplares para poder darle espacio a otros nuevos. Dichos ejemplares podrían tener una segunda vida siendo prestados, intercambiados o regalados a otros lectores.

Actualmente no existe ninguna manera diseñada específicamente para conectar a lectores con el objetivo de prestar o regalar libros de papel. Para realizarlo se puede recurrir a redes sociales como X o Threads, o a foros de lectura en Internet, aunque encontrar algo que resulte cercano a nuestra ubicación geográfica pueda ser una tarea imposible. Para el intercambio de libros sí existe una aplicación en Android llamada Bimdu. Las reseñas más recientes indican que esta aplicación ha dejado de funcionar al no permitir subir imágenes.

Scriptum pretende llenar ese vacío que ha dejado y completarlo con nuevas funcionalidades. Como se ha indicado anteriormente, con esta aplicación diseñada para Android es posible realizar tres tipos diferentes de conexiones entre lectores cercanos: intercambiar un libro por otro, prestar un libro o regalarlo.

Todas las interacciones entre usuarios comienzan con la incorporación de un nuevo libro. Al añadirlo se facilita información sobre su título, autor, editorial y año de publicación. Además, se debe incorporar una fotografía del mismo, información sobre su estado actual y determinar qué tipo de interacción se desea realizar con él.

Los usuarios pueden incorporar una ubicación aproximada en la información de su perfil, lo que permite mostrar en un mapa los libros disponibles más cercanos mediante una búsqueda, ya sea escribiendo una dirección o arrastrando el propio mapa. Esto contribuye a crear una iniciativa sostenible desde el punto de vista ecológico, ya que de este modo se reduce el impacto que conllevan los desplazamientos en vehículos o los envíos mediante una empresa de transportes.

2. Módulos formativos aplicados en el trabajo

Para poder realizar este proyecto es necesario emplear los conocimientos adquiridos en diversas asignaturas de las que han constituido el Ciclo Formativo de Grado Superior de Desarrollo de Aplicaciones Multiplataforma. Aunque muchos de los conceptos utilizados son transversales y han estado presentes en casi todos los módulos del ciclo, los más destacados serían los siguientes:

- Programación.
    - UF3 – El lenguaje de programación Java. Los conceptos básicos de programación en este lenguaje que se ha utilizado en Android Studio para desarrollar la aplicación.
    - UF4 – Programación orientada a objetos. Tema fundamental para entender los objetos y sus características.
    - UF5 – Estructuras de datos compuestas. Para guardar los resultados de las búsquedas ha sido necesario emplear colecciones.
    - UF6 – Interrelación entre clases. En Android, las clases están relacionadas entre sí, estando también presentes otros conceptos como la herencia de otras clases.
- Bases de Datos
    - UF1 – Fundamentos conceptuales. Aunque esta asignatura solamente se ha centrado en las bases de datos relacionales, los conceptos básicos también son necesarios para entender una base de datos NoSQL como Firebase Cloud Firestore.
- Entornos de Desarrollo
    - UF3 – Diseño y realización de pruebas. Uso de JavaDoc para comentar los métodos del código de la aplicación.
    - UF5 – Lenguaje unificado de modelado UML. Para poder llevar a cabo el diagrama UML de clases y de casos de uso.
- Lenguajes de Marcas y Sistemas de Gestión de Información
    - UF2 – Lenguajes para la visualización de la información. Para poder implementar la funcionalidad de los mapas en la aplicación se ha utilizado tanto HTML como CSS.

    - UF3 – Lenguajes de almacenamiento y transmisión de información. Para programar en Android resulta fundamental entender el lenguaje XML, ya que es utilizado para definir las características de sus elementos.
- Desarrollo de Interfaces
    - UF3 – Componentes. JavaScript. Ha sido empleado poder implementar la funcionalidad de los mapas en la aplicación.
    - UF5 – Usabilidad. La aplicación de estas reglas en la interfaz de la aplicación ha permitido detectar problemas de legibilidad y realizar cambios significativos para evitarlos.
- Programación Multimedia y Dispositivos Móviles
    - UF1 – Análisis de tecnologías para aplicaciones en dispositivos móviles. Conceptos básicos para comprender los elementos disponibles en Android Studio.
    - UF2 – Programación de aplicaciones para dispositivos móviles. Resulta de gran utilidad comprender los métodos necesarios para programar con Android Studio.
- Inglés Técnico para Grado Superior
    - El vocabulario y las estructuras utilizadas en esta asignatura han contribuido en el desarrollo de la aplicación en inglés.
- Empresa e Iniciativa Emprendedora
    - UF3 – Estudio de mercado. Ha resultado fundamental realizar un análisis previo sobre otras aplicaciones o servicios similares para buscar la diferenciación.

3. Tecnologías y herramientas

Para desarrollar los mockups se ha utilizado la herramienta **Figma** (Figma: The Collaborative Interface Design Tool, 2024) con la que se han podido plasmar los bocetos iniciales de la aplicación. Estos esquemas han ayudado a crear una identidad de marca no solamente con el logo, sino con el diseño en general. Igualmente, realizar un diseño previo ha servido como base para construir las diferentes partes de la aplicación, realizando pequeñas modificaciones cuando ha sido conveniente pero manteniendo el estilo visual.

Debido a que esta aplicación ha sido concebida para ser utilizada en dispositivos móviles, se ha utilizado **Android Studio** (Android Developers, 2024) como entorno de desarrollo integrado (IDE). De este modo es posible abarcar una gran cuota de mercado debido a que millones de usuarios utilizan el sistema operativo Android. Otro de los motivos que han determinado esta decisión ha sido la versatilidad de este IDE, aspecto que ha permitido probar diversas posibilidades ante la resolución de determinados retos técnicos para dotar de ciertas funcionalidades a la aplicación. Además, Android Studio resulta muy útil para trabajar de forma colaborativa, puesto que tiene capacidad de integrarse en sistemas de control de versiones como **Git**.

Como lenguaje de programación se ha empleado **Java** para poder desarrollar las distintas funcionalidades con las que cuenta la aplicación. Se ha tomado esta decisión debido a la versatilidad de este lenguaje y a la experiencia en el mismo que como desarrolladores hemos obtenido a lo largo de esta formación. También ha determinado esta elección la gran cantidad de recursos formativos que existen sobre este lenguaje, siendo sencillo encontrar guías para casi cualquier funcionalidad.

Para realizar el diseño de la aplicación se ha utilizado el lenguaje de marcas **XML**. No es de extrañar que Android apueste por él en el desarrollo de sus aplicaciones, puesto que permite realizar un desarrollo más organizado separado de la capa lógica de la aplicación. Su sencillo formato permite que la información sea más fácil de leer e interpretar por el ser humano. También facilita el desarrollo de la aplicación en diferentes idiomas gracias a que permite una gestión de las cadenas de

texto utilizadas, que se guardan en un documento XML y se identifican por un id, lo cual, a su vez, ayuda a evitar la redundancia de código en la aplicación.

Para almacenar la información se ha decidido utilizar **Firebase Cloud Firestore**, que es una base de datos en la nube que permite proporcionar datos en tiempo real, con todas las ventajas de eficiencia que ofrecen las bases de datos no relacionales. Además, Firebase ofrece otras herramientas integradas como la autenticación, que permite a los usuarios registrarse de forma segura, o Storage, que otorga almacenamiento de archivos en sus servidores.

Para implementar la funcionalidad de los mapas, se ha utilizado **Leaflet** (Leaflet)**,** una biblioteca de JavaScript de código abierto que permite implementar mapas interactivos en dispositivos móviles. Para llevar a cabo dicha implementación, se han utilizado archivos **HTML**, siendo combinados con **CSS** y **JavaScript** para estructurar la página, estilizar el mapa y agregar funcionalidad interactiva, incluyendo la capacidad de buscar direcciones y colocar marcadores en el mapa.

4. Componentes del equipo y aportación realizada

Elvira Medina Velilla

- Diseño en Figma de los mockups iniciales que sirvieron como guía para desarrollar los elementos visuales finales.
- Gestión de fragmentos para la subida y guardado de Libros: Implementación de la funcionalidad para guardar libros en Firebase Firerstorage, subida de fotografías de los libros correspondientes a su carpeta en Firebase Storage y obtención de las URL correspondientes.
- Gestión de fragmentos para la visualización de Intercambios, Préstamo y Regalos de los libros. Gestión de la edición de datos en la aplicación.
- Interfaz de Usuario: Implementación de un carrusel en el HomeFragment para destacar libros o funcionalidades.

María Moreno Rodríguez

- Gestión de usuarios: registro de usuarios y autenticación de los usuarios ya existentes, perfil de usuarios y edición de los datos de usuario.
- Mapas de la aplicación: búsqueda de libros mediante el uso de mapa, así como la selección y actualización de la ubicación de los usuarios.
- Ficha de los libros: con los datos de los mismos y un mapa que muestra su ubicación, así como la posibilidad de ver aumentada la fotografía del libro.
- Toasts personalizados, splash e icono de la aplicación.

Álvaro Tercero Nieves

- Bottom Navigation Bar: barra inferior de menú que proporciona acceso directo a las principales partes de la aplicación.
- Búsqueda de libros: consultas en la base de datos por título y/o autor, ordenándose los resultados por relevancia y conectando con el propietario.
- Mensajes entre usuarios: vista de perfiles con los que se ha mantenido comunicación y chat que muestra en tiempo real los mensajes intercambiados.
- Validación de campos en la edición de libros.

5. Fases del proyecto

5.1 Estudio de mercado

- Comparación con competidores

A diferencia de sus competidores, que, en su mayoría, se centran más en las recomendaciones y la comunidad de lectores, Scriptum está diseñada para permitir interactuar a los usuarios con los libros ya sea mediante su intercambio, préstamo o mediante donación. Además, Scriptum utiliza la ubicación geográfica del usuario para facilitar la conexión entre lectores cercanos, promoviendo la sostenibilidad al reducir la necesidad de desplazamientos largos o envíos.

- Competidores directos e indirectos:

1. **Librología**: Plataforma que se enfoca en compartir y descubrir libros a través de recomendaciones y la creación de una comunidad de lectores. Aunque es fuerte en fomentar discusiones y descubrimiento de nuevos libros, no facilita tanto el intercambio físico de libros entre usuarios.

2. **Bimdu**: Aplicación diseñada para el intercambio de libros. Según reseñas recientes, ha dejado de funcionar correctamente debido a problemas técnicos que impiden subir imágenes de los libros. Esto limita su efectividad y deja insatisfechos a sus usuarios.

3. **Goodreads**: Aunque no permite el intercambio físico de libros, es una red social muy popular entre lectores para reseñar y recomendar libros. Su gran base de usuarios y su amplio abanico de funcionalidades de interacción social la hacen un competidor indirecto importante.

4. **BookMooch**: Plataforma internacional que permite a los usuarios intercambiar libros. Aunque cumple con la función de intercambio, su interfaz y experiencia de usuario podrían mejorarse, y, además, no se enfoca en conectar a usuarios que se encuentren cerca unos de otros, como sí hace Scriptum.

- Propuesta de valor de Scriptum

Scriptum destaca porque permite tres tipos de interacciones con los libros: intercambio, préstamo y donación. Los usuarios pueden añadir un libro proporcionando detalles como título, autor, editorial, año de publicación, estado y una fotografía del libro. La ubicación geográfica de los usuarios facilita encontrar libros cercanos, promoviendo así una iniciativa sostenible al minimizar desplazamientos y envíos. También, mediante su funcionalidad de chat entre usuarios, puede llegar a servir como una plataforma que ayude a que los amantes de la lectura conecten entre sí.

- Diferenciadores clave:

- Múltiples tipos de interacción: No solo permite el intercambio, sino también el préstamo y la donación de libros entre los usuarios.

- Ubicación geográfica: Facilita la conexión entre usuarios ubicados cerca unos de otros, lo que ayuda a reducir el impacto ambiental y los costes asociados a los envíos.

- Sostenibilidad: Promueve la reutilización de libros, contribuyendo a una práctica ecológica y sostenible.

- Estrategia de marketing

El lanzamiento de Scriptum se enfocará en universidades, clubes de lectura y eventos literarios, ya que son lugares donde la densidad de lectores es alta. Un ejemplo sería la Feria del Libro de Madrid. También, con el fin de captar público joven y adulto, se podrán realizar campañas de publicidad en redes sociales enfocadas a ese segmento de la población, para así poder aumentar la visibilidad de la aplicación.

Scriptum tiene un gran potencial para capturar una audiencia significativa de amantes de los libros al ofrecer una solución integral y sostenible para el intercambio, préstamo y donación de libros. Su enfoque en la comunidad local y su funcionalidad la diferencian de los competidores actuales, posicionándola favorablemente en el mercado. Con una estrategia de marketing efectiva, Scriptum puede establecerse como una herramienta valiosa para los lectores que buscan compartir y descubrir libros de manera sostenible, e incluso llegar a forjar amistades con otros amantes de los libros.

5.2 Modelo de datos

Como base de datos se ha utilizado Firebase Cloud Firestore, que es una base de datos NoSQL en la nube, compuesta por colecciones y documentos, que permite almacenar y sincronizar datos en tiempo real entre la aplicación y el servidor de Firebase. Para el desarrollo del proyecto, se han elaborado dos bases de datos:

- booksData

Esta base de datos se ha elaborado para guardar los datos pertenecientes a los libros que se cargan en la aplicación, almacenando los siguientes campos:

  - author: el autor del libro.
  - bookId: el identificador del libro.
  - editorial: la editorial del libro.
  - photo: la url de la fotografía de perfil del usuario.
  - status: comentarios sobre el estado del libro
  - timestamp: el momento en que se subió el libro a la aplicación.
  - title: el título del libro.
  - type: en este campo se establece si el libro es para prestar, regalar o intercambiar.
  - user: el identificador único del usuario que ha subido el libro a la aplicación.
  - year: el año de publicación del libro.

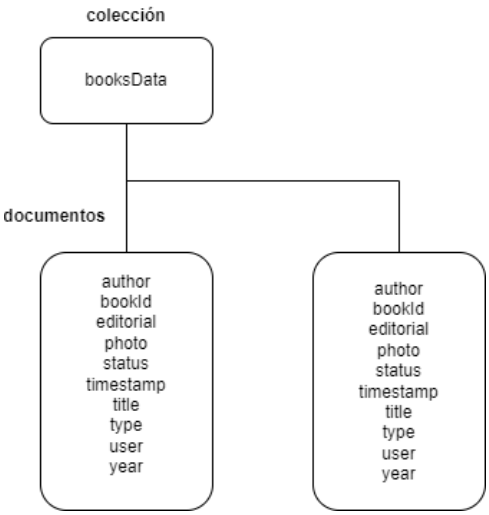


Figura 1. Esquema de base de datos booksData.

- usersData

Esta base de datos se ha elaborado para guardar los datos pertenecientes a los usuarios que se registran, almacenando los siguientes campos:

  - address: la dirección del usuario.
  - latitude: la latitud de la ubicación del usuario.
  - longitude: la longitud de la ubicación del usuario.
  - nameSurname: nombre y apellidos del usuario.
  - profileImageUrl: la url de la imagen de perfil del usuario.
  - user: el identificador único asignado al usuario.

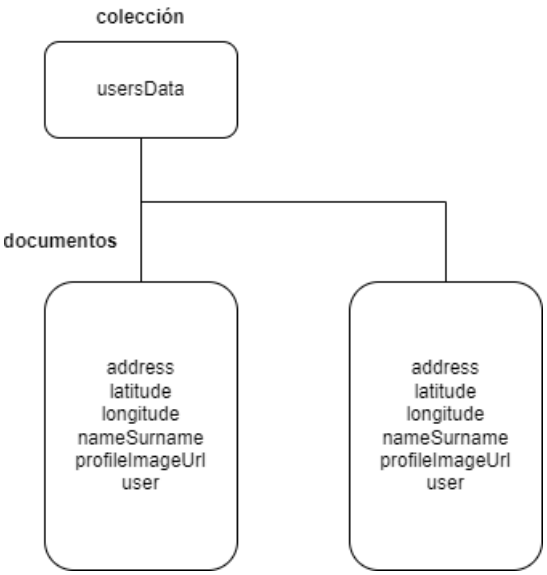


Figura 2. Esquema de base de datos usersData.

5.3 Diagramas UML

Por su extensión, el diagrama de clases está ubicado dentro del repositorio de GitHub bajo el nombre “diagram.png”. Se encuentra en la siguiente URL: https://github.com/Mariamore/ScriptumApp/blob/master/diagram.png

5.4 Diseño de interfaces

Para realizar el diseño inicial de la interfaz, se ha utilizado Figma, y los prototipos pueden consultarse en la siguiente URL: https://www.figma.com/file/cZqAMhy3bkqdajl5YwvSNI/Prototipo?type=design&node-id=0%3A1&mode=dev&t=mpm8P6IRtya2iXRs-1

Según este diseño previo, lo primero que va a encontrar el usuario al abrir la aplicación es un **Splash Screen** que muestra su icono. Esta vista solamente aparece unos segundos y no se vuelve a mostrar en ningún otro momento, sirviendo únicamente de presentación. A continuación, aparece la **página principal** de la aplicación. En ella se pueden ver libros destacados y vistos recientemente. Es posible ver los libros disponibles mediante búsquedas antes de crear una cuenta o iniciar sesión, para atraer a los usuarios potenciales que descarguen la aplicación por primera vez. En la parte inferior hay una barra de menú que se mantiene en todas las vistas y que contiene cuatro botones: **Inicio**, **Perfil**, **Buscar** y **Mensajes**.

Al pulsar sobre **Perfil** es posible iniciar sesión mediante el correo electrónico y la contraseña. También se ofrece la opción de crear una cuenta en el caso de que no estemos dados de alta. Otra de las posibilidades es la de recuperar la cuenta en caso de olvidar la contraseña.

Figura 3. Prototipos de Splash Screen, pantalla de inicio y pantalla de login.

Para crear una cuenta nueva se ha diseñado otra vista. En ella se solicitan algunos datos como el nombre, la dirección, el correo electrónico y la contraseña, terminando con un botón para crear la cuenta y dando la opción de cancelar.

La pantalla del perfil de usuario, a la que se accede pulsando el botón correspondiente en la barra de menú inferior, cuenta en la parte superior con la foto de perfil, mostrando un icono genérico en caso de no haber facilitado ninguna. Esta vista cuenta con cuatro botones: **Datos Personales**, para ver y editar los datos del usuario; **Subir Libro**, para dar de alta un libro que se quiera intercambiar, prestar o regalar; **Consultas**, para ver las transacciones que ha realizado con otros usuarios; y **Ubicación**, donde es posible añadir una ubicación al perfil para que los libros puedan aparecer en el mapa.

La pantalla de **Buscar** muestra por defecto una barra de búsqueda donde escribir las palabras que queremos enviar a la base de datos para que devuelva los registros que coinciden. Dicha búsqueda se realiza sobre los campos de título y autor, ya que son los datos más importantes de cada libro.

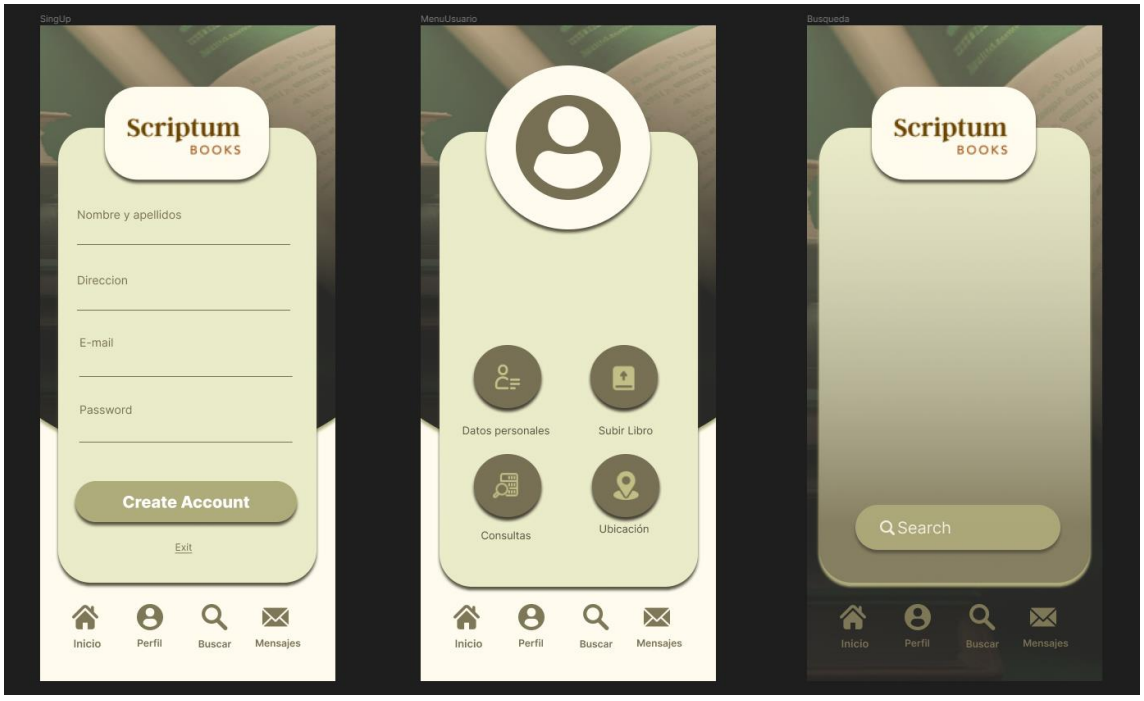


Figura 4. Prototipos de pantalla para crear cuenta, perfil de usuario y pantalla de búsqueda.

En el apartado **Mensajes** aparece una pantalla en la que se ven los nombres de los usuarios con los que se ha hablado anteriormente. Al pulsar en ellos se accede a otra vista en la que aparecen los mensajes que se han recibido y enviado.

La vista de **Subir Libro**, accesible desde el perfil del usuario, permite añadir una fotografía del mismo para que otros usuarios tengan información visual sobre el ejemplar. Otros de los datos que se pueden añadir son su título, autor, editorial, año de publicación, estado actual y si el libro es para prestar, regalar o intercambiar.

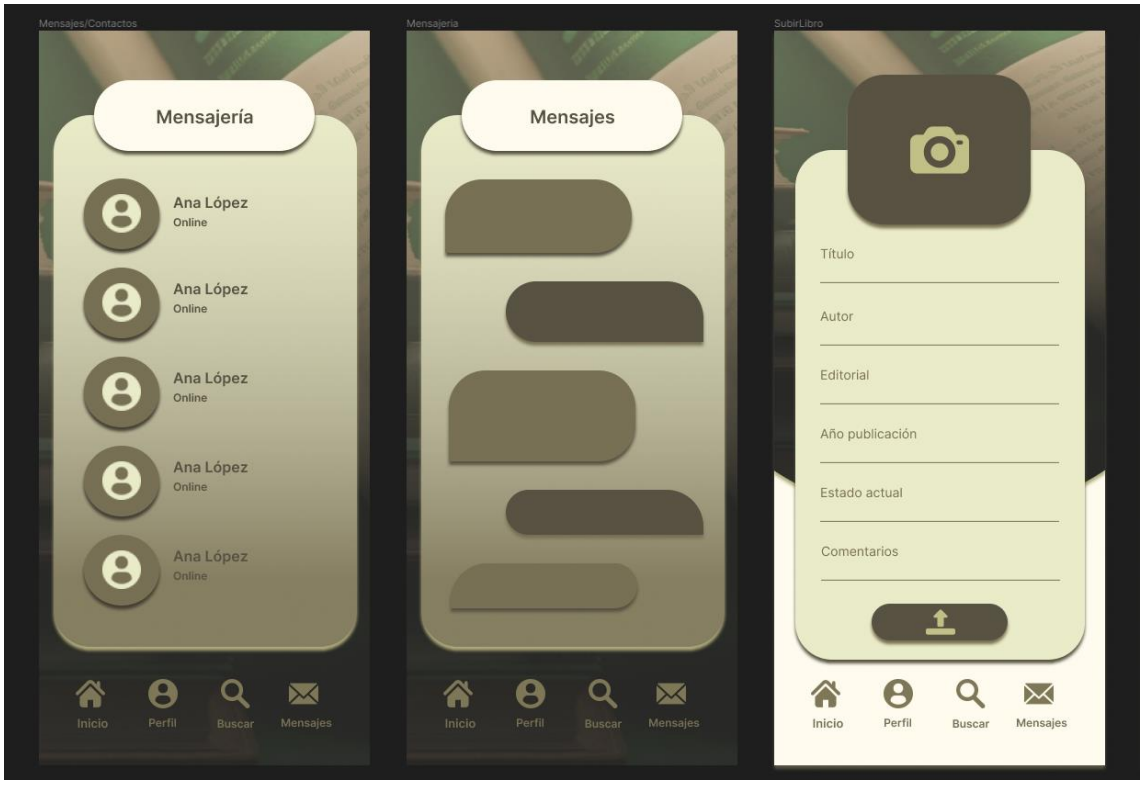


Figura 5. Prototipos de pantalla de mensajes a usuarios, de mensajes a un determinado usuario y de añadir un libro a Scriptum.

Otro de los botones con los que cuenta el menú del perfil de usuario es el de **Consultas**. Al pulsarlo aparecen tres botones para poder ver las diferentes categorías de interacciones que se pueden realizar con Scriptum: préstamos, intercambios o regalos. Al pulsar en cada uno de dichos botones aparece una vista con un listado de los libros que se hayan añadido. En este diseño inicial están disponibles las opciones de **Añadir** y **Editar**, además de otro botón para volver atrás a la página de perfil.

Figura 6. Prototipos de pantalla de consulta de interacciones, de préstamos y de intercambios.

También desde el perfil de usuario se puede acceder a los **Datos Personales**. En esta vista aparece la información que guarda la base de datos del usuario que está utilizando Scriptum: nombre, dirección, localidad, teléfono, email visible en el perfil y comentarios adicionales. Es posible modificar los datos, enviando los cambios mediante el botón **Guardar**. También se puede volver al menú de perfil de usuario con el botón **Atrás**.

El último de los botones que aparecen en la pantalla de perfil, **Ubicación**, conduce a una nueva vista en la que es posible ver la ubicación del usuario. Si no ha añadido una ubicación a su perfil, al pulsar sobre el botón **Añadir** puede incorporar una nueva para que los libros que ha dado de alta aparezcan referenciados en ese lugar. Si previamente ha añadido una ubicación pero desea cambiarla, el mismo botón elimina la anterior ubicación y la sustituye por la actual.

Figura 7. Prototipos de pantalla de consulta de regalos, editar datos personales y ubicación.

A la hora de implementar este prototipo en el proyecto, se ha sido bastante fiel al mismo, implementando pequeños cambios y mejoras que han ido estimándose oportunas durante el desarrollo de la aplicación. Los cambios más significativos en la interfaz de la aplicación tras su desarrollo han sido los siguientes:

- Eliminación de botones añadir y editar en los fragmentos de BookLoanFrangment, BookExchangeFragment, GifBookFragment. Anteriormente, los usuarios podían añadir y editar libros directamente desde los fragmentos. Sin embargo, se ha optimizado esta funcionalidad para mejorar la experiencia del usuario y simplificar la interfaz.
- Adición de un Spinner en el Fragmento UploadBook. Ahora, en lugar de tener botones separados para añadir y editar libros en los fragmentos de préstamo y cambio de libros, se ha introducido un Spinner en el fragmento UploadBookFragment, que permite a los usuarios seleccionar la acción que desean realizar cuando suben un libro a la aplicación. Las opciones disponibles en el Spinner incluyen añadir libro para préstamo (Loan), para intercambio (Exchange) o para regalar (Gift). Esto tiene la ventaja de simplificar la interfaz, reduciendo la cantidad de botones visibles y posibles confusiones.

- Ubicación del botón de edición y eliminación de libros. En lugar de tener botones dispersos para la edición y eliminación de libros, estos se han ubicado de manera más accesible y lógica para el usuario.
- Nueva ubicación: dentro de la ficha del libro. Los botones de edición y eliminación de libros ahora están ubicados dentro de la vista de detalle del libro. Esta vista se accede al seleccionar cualquier libro en las listas de préstamos o intercambios o regalos.

Figura 8. Diseños definitivos de pantalla de añadir un libro y de libros disponibles para prestar.

5.5 Planificación del desarrollo

El desarrollo de este proyecto comenzó con la creación de las interfaces de usuario principales, que consistió en trasladar los diseños creados en Figma a layouts mediante Android Studio. Posteriormente se decidió que el menú inferior se realizase mediante un Bottom Navigation Bar en el layout principal, lo que supuso trasladar todos estos diseños a fragments para ser cargados encima de dicho layout.

Tras crear entre todos las interfaces de usuario, las siguientes tareas fueron repartidas de forma equitativa entre los miembros del equipo, asignando en un primer lugar la realización de la autenticación de usuarios, el registro de usuarios, el menú de usuarios, la búsqueda de libros por texto, la subida de libros a la aplicación y la consulta de los libros según sean para intercambio, regalo o préstamo.

A medida que avanzó el desarrollo de esas funcionalidades, al terminar con las tereas asignadas, cada integrante comenzó a implementar algunos de los elementos restantes del proyecto. Por último, las tres tareas de mayor complejidad, que han sido la mensajería, la implementación de los mapas y la página principal, se asignaron cada una a un integrante del equipo. Aun así, en todo momento ha existido colaboración e interconexión en el trabajo llevado a cabo por todos los integrantes, brindando ayuda y consejo cuando han surgido problemas o dificultades.

5.6 Funcionalidad del proyecto

Al arrancar la aplicación, lo primero con lo que se encuentra el usuario es un splash de carga con el logo y una barra de progreso circular formada por los colores básicos de la aplicación. Tras 3 segundos de espera, se muestra la MainActivity y la página principal.

El proyecto cuenta únicamente con una activity. En ella se encuentra el Bottom Navigation View, que es la barra de menú inferior con acceso directo a las principales partes de la aplicación, y un espacio sobre el que cargar los fragments que componen el resto de la aplicación. La decisión de optar por este modelo fue debido a que resulta mucho más rápido y eficiente cargar solamente dichos fragments sin cambiar de activity. Además, el menú que podíamos utilizar de esta forma es mucho más dinámico, evitando ser recargado constantemente al cambiar de activity.

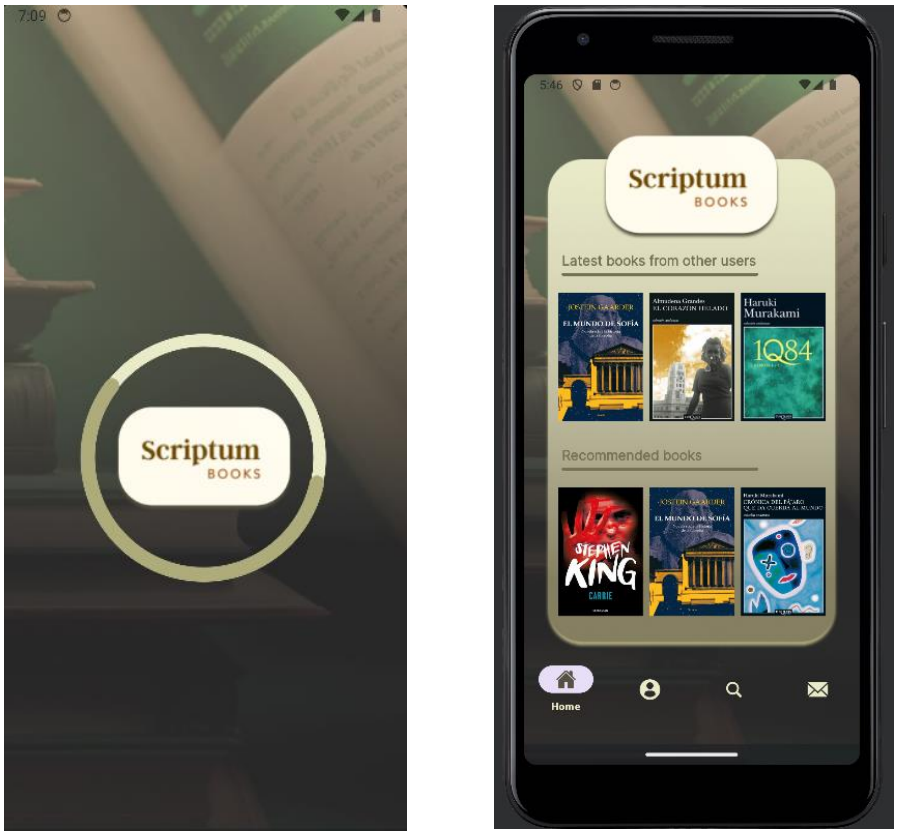


Figura 9. Pantalla de Splash y página de inicio de Scriptum.

La clase MainActivity es la actividad principal del proyecto, y es la encargada de manejar la navegación entre fragmentos de la interfaz de usuario, utilizando la barra de navegación inferior de la aplicación.

En la barra de menú hay configurado un oyente para manejar los clics en la barra de navegación inferior, dependiendo del elemento seleccionado Home, Profile, Search, Messages se reemplaza el fragmento actual por el correspondiente, lo que permite la navegación a través de los diferentes fragmentos usando la barra de navegación inferior.

Una vez ejecutado el splash, la aplicación muestra por defecto el HomeFragment, que contiene dos carruseles de imágenes. Estas son cargadas desde Firebase Firestore y, dependiendo si un usuario está autenticado, puede ver acceder a la ficha del libro haciendo clic en la fotografía del mismo, lo que da lugar a que se abra el BookInfoFragment, el cual se explicará más adelante. Pero, si por el contrario, el usuario no está autenticado, se muestra un toast con un mensaje que indica que el usuario ha de estar autenticado.

En este fragmento, se configura el enlace a la vista usando FragmentHomeBinding, Se inicializa Firebase Firestore (db) y se obtiene el usuario actual (user), el método onCreateView() infla el diseño del fragmento y devuelve la vista raíz, a la vez que verifica si el usuario está autenticado y obtiene su Id de usuario (idUser).

El método onViewCreated() configura los adaptadores para los carruseles de imágenes (viewPager y viewPager2), que son los que gestionan las listas de URLs de imágenes (imageUrls e imageUrls2).

Cada adaptador (ImageCarouselAdapter y ImageCarouselAdapter2) se asegura de que las imágenes correctas se muestren en el carrusel, son componentes clave en el HomeFragment. Su función principal es proporcionar las vistas necesarias para que los carruseles de imágenes funcionen correctamente dentro de los ViewPager2.

En onViewCreated(), se sitúan los métodos para obtener imágenes desde Firestore (fetchLatestImagesFromFirestore y fetchLatestImagesFromFirestore2).

La función fetchLatestImagesFromFirestore() obtiene imágenes de la colección “booksData” ordenadas por fecha y excluye las del usuario actual a su vez agrupa las imágenes de tres en tres y actualiza el adaptador ImageCarouselAdapter1. La función fetchLatestImagesFromFirestore2() obtiene imágenes de la colección “recommendedBooks” de manera similar a la anterior y actualiza el adaptador ImageCarouselAdapter2.

Los clics en las imágenes del carrusel son manejados por el método onItemClick(), de forma que si el usuario está autenticado, obtiene el ID del libro correspondiente y abre BookInfoFragment. Si el usuario no está autenticado, muestra un mensaje de que necesita iniciar sesión.

El método openDetailFragment2(), obtiene el ID del libro desde Firestore basado en la URL de la imagen y abre el fragmento ‘BookInfoFragment’ con el ID del libro, permitiendo al usuario ver detalles adicionales. Este fragmento también cuenta con un toast personalizado que aparecerá si el usuario no está autenticado.

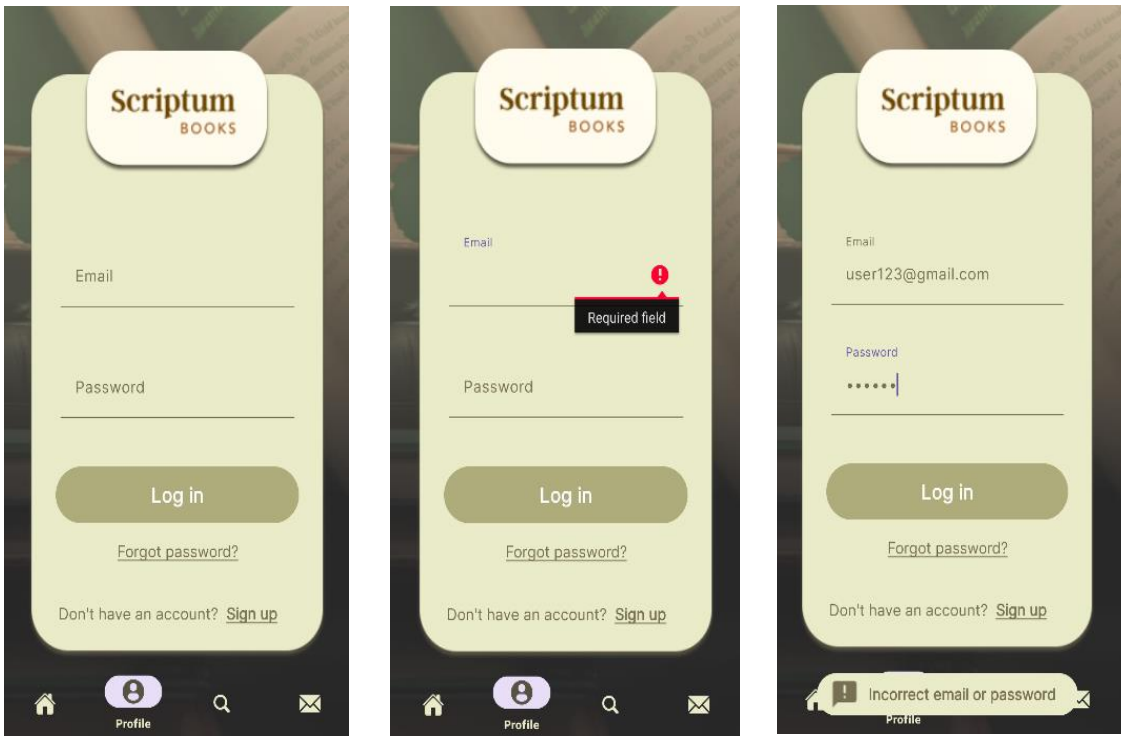


Figura 10. Pantalla de Login, ejemplo de mensaje de error y ejemplo de mensaje en Toast.

Al seleccionar el icono del usuario del menú inferior, si no hay un usuario autenticado, se muestra el LoginFragment, en el que en la parte superior se muestra el logo de la aplicación, y bajo este existen dos campos de texto, uno para introducir el email y otro para introducir la contraseña del usuario. También cuenta con un botón con el texto "Log in" que permite al usuario autenticarse, y dos textos clicables: "Forgot password", que al hacerle clic pasa al ResetPasswordFragment; y "Sign up", que al hacerle clic pasa al SignUpFragment. En este fragment se da funcionalidad a los botones y textos clicables mediante la implementación de la interfaz View.OnClickListener. Así, si se hace clic en el botón de "Log in" se ejecuta la validación del contenido de los campos de texto, de forma que si están vacíos, mediante el método .setError() aparece un mensaje de error junto al campo, en el que se indica que es obligatorio rellenarlo. También se comprueba que el email tenga el formato adecuado y que la contraseña tenga una longitud superior a 6 caracteres. En ambos casos mediante setError() se establece un mensaje de error "Enter a valid address" en el caso del campo de texto del email y "6 characters minimum" en el caso de la contraseña.

Si se cumplimentan los dos campos de texto y los datos introducidos concuerdan con los existentes en la base de datos, aparece un toast positivo con el mensaje "User

logged", y se pasa al ProfileFragment. En caso de que los datos no sean iguales, aparece un toast negativo con el mensaje "Incorrect email or password".

Si se selecciona el texto "Forgot password?", como se ha indicado anteriormente, se pasa al ResetPasswordFragment, el cual cuenta en su parte superior con el logo de la aplicación, y bajo este un campo de texto para introducir el email del que se quiere recuperar la contraseña para poder acceder a la aplicación. Bajo el campo de texto existen dos botones: "Reset password" y "Back". El primero permite enviar un email de recuperación de la contraseña al correo indicado y el segundo lleva de vuelta al LoginFragment. En el código, se valida el campo de texto, de forma que si al hacer clic en el botón "Reset password", el campo está vacío, aparece el mensaje de error "Required field" y, si no cumple con el formato de un email, aparece el mensaje de error "Enter a valid email". Cuando se introduce un email válido y se hace clic en el botón "Reset password", aparece un toast con el mensaje "Sending password reset link (if email is registered)…" y, si el email estaba registrado en la base de datos, se recibirá un email con un link para poder resetear la contraseña. En caso de que no esté registrado el email en la base de datos, no se recibe ningún correo.

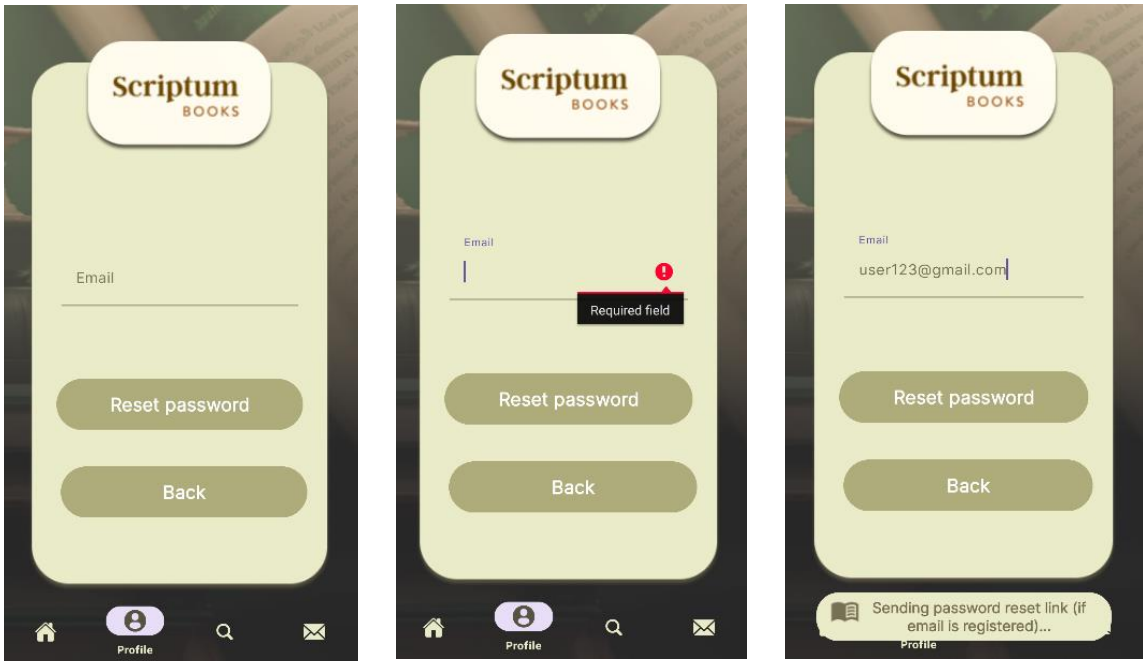


Figura 11. Pantalla de ResetPassword, ejemplo de mensaje de error y mensaje en Toast.

Si en el ProfileFragment se selecciona "Sign up", como se ha mencionado anteriormente, se abre el SignUpFragment, que muestra en la parte superior el logo de la aplicación, y bajo este, cuatro campos de texto. El primero permite introducir al usuario

su nombre y apellidos. El siguiente actúa como un botón de forma que, al hacer clic en él, lleva al MapFragment, donde se podrá seleccionar la ubicación del usuario. En el siguiente campo de texto se ha de introducir el email y, en el último, la contraseña elegida por el usuario. Bajo los campos de texto está el botón “Create account” que, al hacer clic en él, guarda los datos introducidos en los campos de texto en la base de datos y pasa al ProfileFragment, en el que se muestra el perfil del usuario. Por último, está el texto “Exit”, que lleva de vuelta al LoginFragment.

La validación de los campos de texto se realiza al hacer clic en el botón “Create account” y, en caso de que alguno esté vacío, aparece el mensaje de error “Required field”, que indica que el campo tiene que estar cumplimentado para poder crear la cuenta. En el caso de los campos de texto de email y contraseña, como en fragments anteriores, se comprueba que el primero tenga el formato adecuado y que la contraseña contenga más de 6 caracteres, y en caso contrario, aparece un mensaje de error con un texto adecuado para cada caso.

Si todos los campos están correctamente cumplimentados, al hacer clic en el botón, se procede a registrar el email y la contraseña en Firebase. En caso de que surja un problema, se muestra un toast con el texto “Sign up failed” y la cuenta no se crea; pero en caso de que el registro en Firebase sea satisfactorio, se procederá a añadir los datos aportados por el usuario en los campos de texto a la base de datos “usersData”, y si esto ocurre de forma satisfactoria, aparece un toast con el mensaje “Account created” indicando así que la cuenta se ha creado, y se abre el ProfileFragment. En caso contrario, aparece un toast con el texto “Sign up failed” y la cuenta no se da de alta.

En este fragment, en el método onCreate() se ha implementado el método “getParentFragmentManager().setFragmentResultListener("key", this, new FragmentResultListener() { ... })” que se utiliza para establecer un listener en un fragmento y escuchar resultados enviados por otros fragmentos, en este caso los enviados por el MapFragment.

En primer lugar, “getParentFragmentManager()” obtiene el “FragmentManager” asociado con la activity o el fragmento padre. “FragmentManager” es la clase que gestiona los fragmentos dentro de una actividad o de otros fragmentos, permitiendo

añadir, quitar, reemplazar y realizar otras operaciones sobre los fragmentos. Luego, con “setFragmentResultListener("key", this, new FragmentResultListener() { ... })” se establece un listener que escucha los resultados asociados con una clave específica, en este caso, "key". Dentro de la implementación de “FragmentResultListener”, el método “onFragmentResult” se invoca cuando otro fragmento establece un resultado utilizando la clave “key” mencionada anteriormente.

Este método recibe dos parámetros:

- “requestKey”: es la clave con la que se identificó el resultado, que debería coincidir con "key".
- “bundle”: un “Bundle” que contiene los datos enviados por el otro fragmento.

Dentro de “onFragmentResult”, se extraen los datos del “Bundle”. En este caso, se obtienen tres valores:

- “addressReceived”: se extrae del “Bundle” usando la clave “address” y se asigna a la variable `addressReceived`.
- “latitude”: se extrae del “Bundle” usando la clave “latitude” y se asigna a la variable `latitude`.
- “longitude”: se extrae del `Bundle` usando la clave "longitude" y se asigna a la variable `longitude`.

Si el campo de texto `addressInputEditText` no es nulo, se actualiza su texto con el valor de `addressReceived`. Esto permite actualizar la interfaz de usuario del fragmento receptor con los datos proporcionados por el fragmento emisor, de forma que en el campo de texto “Address” se mostrará la dirección recibida desde el MapFragment. Los datos “latitude” y “longitude”, que contienen respectivamente la latitud y longitud de la ubicación seleccionada por el usuario se utilizarán para guardarlos en la base de datos usersData, ya que serán de utilidad en otros fragments para poder llevar a cabo acciones con la ubicación de los usuarios.

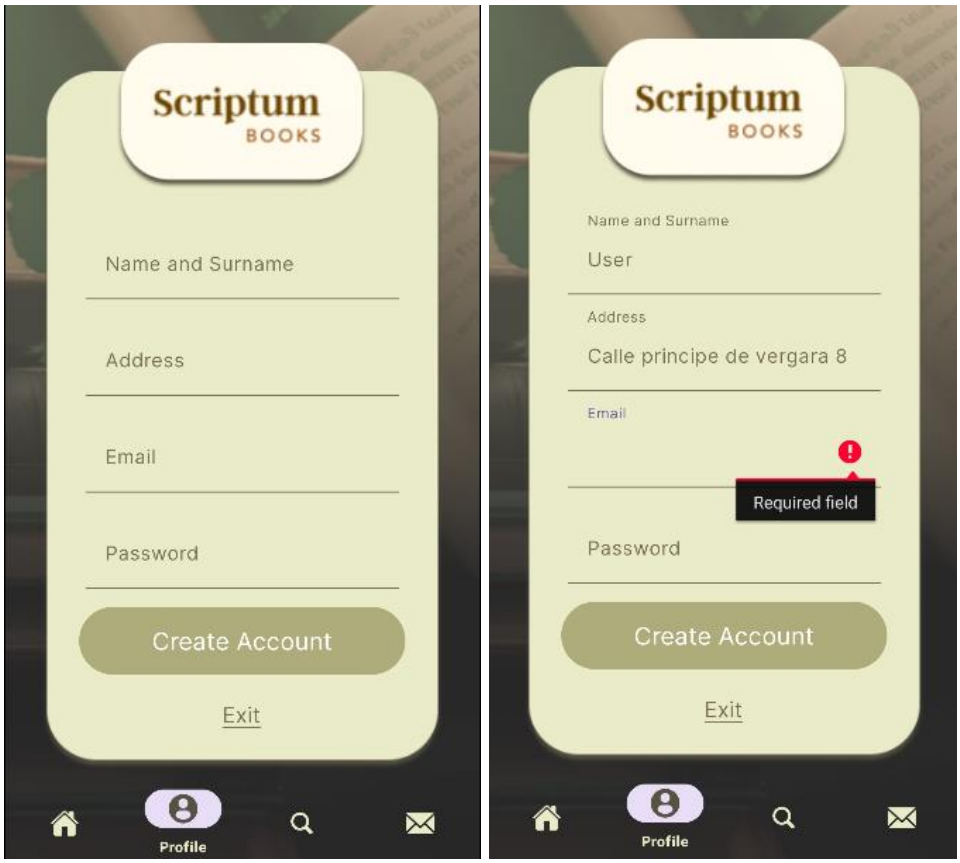


Figura 12. Pantalla de SignUp y ejemplo de mensaje de error al no introducir datos requeridos.

Al seleccionar el campo de texto “Address”, como se ha mencionado con anterioridad, se abre el MapFragment, cuya funcionalidad es permitir al usuario seleccionar su ubicación y visualizarla en un mapa. Este fragment cuenta con un campo de texto “Enter your address” para que el usuario pueda introducir la dirección a buscar, bajo el mismo, un botón “Search address” que permite buscar la ubicación introducida y, en caso de que sea correcta, mostrarla en el mapa con un marcador, y si es incorrecta, se mostrará un toast con el mensaje “Enter a valid address”. En caso de que el campo de texto esté vacío al hacer clic en el botón, se mostrará un toast con el mensaje “Enter an address”.

Bajo el botón hay un webView que mediante el código proporciona la configuración y carga de una página web que es un mapa de Leaflet. Al cargar el fragment, se llevan a cabo las acciones necesarias para dotar de funcionalidad al webview. Primero, con “webView.getSettings().setJavaScriptEnabled(true);” se habilita la ejecución de JavaScript dentro del webView.

Con “webView.addJavascriptInterface(this, "Android");” se añade una interfaz entre el JavaScript de la página web y el código Java de la aplicación, permitiendo que el JavaScript de la página web pueda llamar a métodos en la clase Java de la aplicación,

como es el caso del método setCoordinates(), el cual permite que el código JavaScript de la página web del mapa pueda enviar datos a la aplicación, esto se consigue usando la anotación `@JavascriptInterface`, que se utiliza para indicar que este método puede ser invocado desde el código JavaScript. Al ocurrir esto, este método asigna las coordenadas de latitud y longitud recibidas a las variables de instancia "latitude" y "longitude" respectivamente. Luego, verifica si ambas coordenadas son cero, y si es así, indica que se debe ingresar una dirección válida mostrando un toast con el mensaje "Enter a valid address", ya que las coordenadas 0,0 indican que la dirección no existe.

Después, el método "webView.setWebViewClient(new WebViewClient());" establece un `WebViewClient` personalizado para el webView. Esto es importante para manejar eventos de navegación dentro del webView, como abrir enlaces dentro del mismo webView en lugar de en el navegador externo. Finalmente, "webView.loadUrl("file:///android_res/raw/map.html");" carga una página web desde los recursos locales de la aplicación. En este caso, la página web se encuentra en la carpeta `raw` dentro de los recursos de Android, y se carga utilizando la URL "file:///android_res/raw/map.html".

El archivo map.html define una página web que muestra un mapa interactivo utilizando Leaflet. En la sección "<head>", se incluyen los metadatos y enlaces a recursos externos, también se define la escala inicial y el ancho de la ventana de visualización en dispositivos móviles. Además, se establece el título de la página y se enlazan las hojas de estilo de Leaflet para dar formato al mapa.

El cuerpo de la página, contenido dentro de la etiqueta "<body>", contiene un div con el id "map" donde se renderizará el mapa. En la sección de script, se inicializa el mapa de Leaflet con una vista centrada en las coordenadas [40.416775, -3.703790] (Madrid, España) y se añade una capa de OpenStreetMap como base del mapa.

Se define una función "searchAddress(address)" que toma una dirección como argumento y busca las coordenadas correspondientes utilizando el servicio de geocodificación de Nominatim de OpenStreetMap. Si se encuentra la dirección, se crea un marcador en el mapa en la ubicación correspondiente y se llama a la función

“Android.setCoordinates(lat, lon)” para pasar las coordenadas al código Java de la aplicación, la cual se ejecuta como se ha mencionado anteriormente.

Prosiguiendo con la estructura del fragment, bajo el webView se encuentran dos botones: “Save” y “Back”. Si se hace clic en el primero, se obtiene el contenido actual del campo de dirección con “String currentAddress = addressInput.getText().toString()” y después se compara el contenido actual con la última dirección buscada. En caso de que no coincidan, se muestra un toast con el mensaje “You must search the address first”, obligando al usuario a validar la dirección introducida buscándola en el mapa, para evitar que se guarde una dirección inexistente. Si el contenido actual de la caja de texto coincide con la última dirección buscada, se crea un “Bundle” y se añade la dirección, latitud y longitud actuales. Luego, con el método “getParentFragmentManager().setFragmentResult("key", bundle)” se envía el resultado al fragmento padre. Después, obtiene el “FragmentManager” y se retrocede al fragmento anterior en la pila de retroceso con “fragmentManager.popBackStack()”, lo que permite volver al fragment anterior, en este caso SignUpFragment, conservando los datos introducidos en dicho fragment antes de pasar al MapFragment.

Si se hace clic en el botón “Back” se obtiene el “FragmentManager” y se retrocede al fragmento anterior en la pila de retroceso con “fragmentManager.popBackStack()”, pero en este caso, no se guarda la dirección contenida en el campo de texto, ni se envía al fragment anterior.

Una vez el usuario ha iniciado sesión o ha creado su cuenta, se abre el ProfileFragment. En la parte superior del mismo, se muestra la imagen de perfil del usuario y, en caso de que no se haya elegido ninguna aún, una foto por defecto, que se encuentra en la carpeta drawable del proyecto bajo el nombre de “userlogo”. Para que la imagen se muestre redonda, se utiliza la vista CircleImageView. Para que el usuario elija fotografía de perfil, ha de hacer clic sobre item CircleImageView llamado userImage, activándose el método “openFileChooser()”, de forma que se abre un selector de archivos para que el usuario pueda elegir una imagen de su dispositivo.

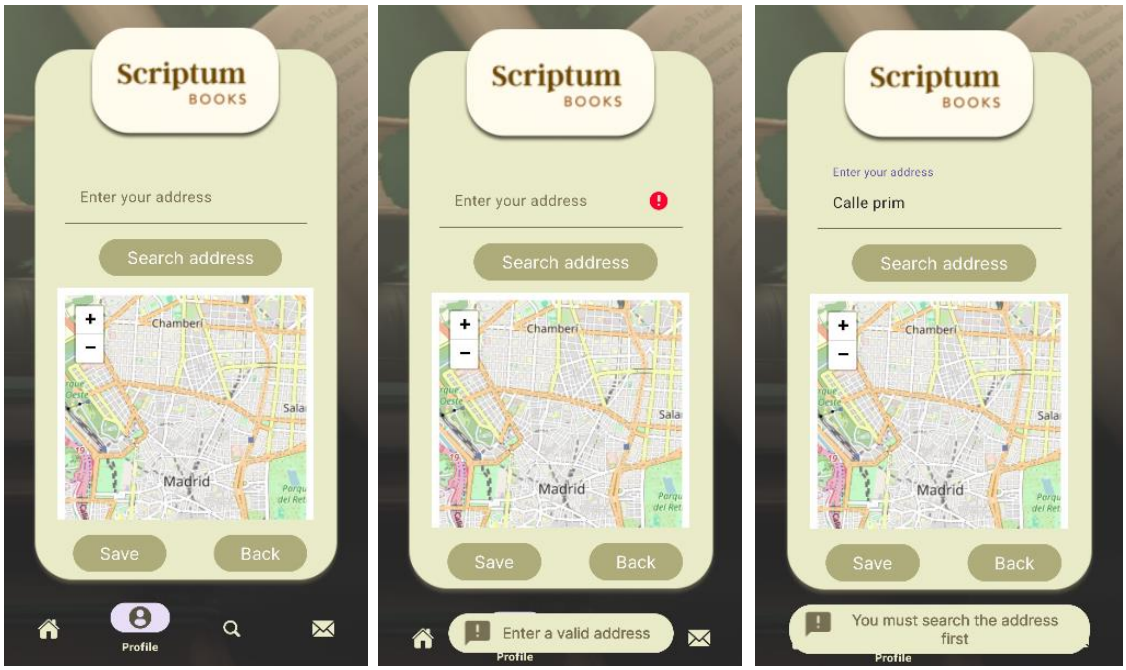


Figura 13. Página de MapFragment y ejemplos de Toast en problemas relacionados con la dirección.

El método “onActivityResult()” es invocado después de que una actividad lanzada ha completado su ejecución y devuelve un resultado. En este caso, se usa para recibir la URI de la imagen seleccionada del selector de archivos. Después se ejecuta el método “uploadFile()”, que gestiona la carga de la imagen seleccionada a Firebase Storage mediante “putFile()”. Una vez completada la carga, se obtiene la URL de descarga de la imagen utilizando “getDownloadUrl()”. Luego, se actualiza el perfil del usuario en Firestore en la base de datos “usersData”, buscando el documento que pertenece al usuario actual, comparando el campo “user” de cada documento de la base de datos, con el identificador del usuario actual. Una vez se localiza el documento, se actualiza el campo “profileImageUrl” del mismo con la URL de descarga de la imagen, para que la imagen de perfil se muestre correctamente en la aplicación. Si todo sale bien, se muestra un toast con el mensaje “User photo uploaded”, y en caso contrario, se muestra otro con el mensaje “Error uploading user photo”.

El ProfileFragment cuenta también con cuatro botones. Al hacer clic en el de “Personal Data”, se pasa al fragment PersonalDataFragment, que muestra la información del perfil del usuario. Si se hace clic en el botón “Upload book”, se pasa al fragment UploadBookFragment, en el cual el usuario puede subir libros a la aplicación. Si el botón en el que se hace clic es el de “Check books”, se pasa al QueriesFragment,

donde el usuario podrá consultar los libros que ha subido a la aplicación. Por último, si se hace clic sobre el botón “Location”, se abre el LocationFragment, el cual permite al usuario actualizar su ubicación.

En la parte inferior del fragment, existe un icono que actúa como botón, que permite al usuario cerrar la sesión, de forma que al hacer clic sobre él, se pasa al LoginFragment y aparece un toast con el mesaje “User logged out”.

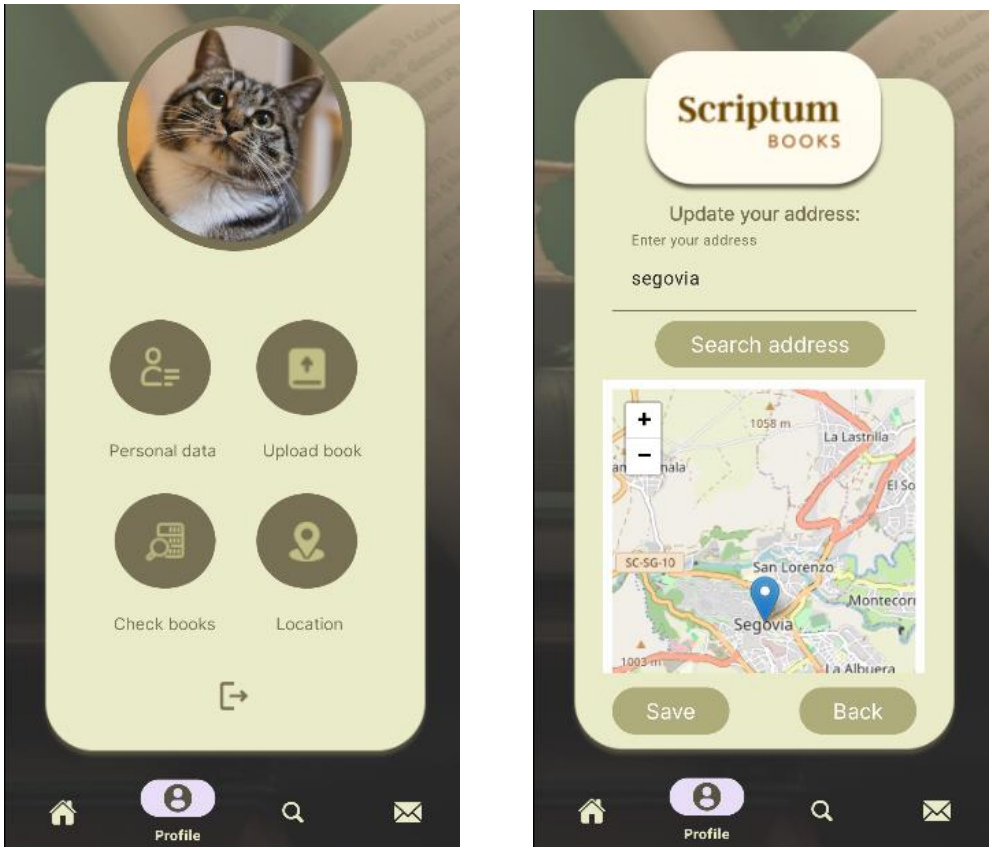


Figura 14. Páginas Profile y Location.

El LocationFragment presenta la misma estructura que el MapFragment, salvo por la diferencia de que tiene el texto “Upload your address” sobre el campo de texto del fragment. En cuanto a funcionalidad, es muy similar a la del MapFragment, salvando algunas diferencias como que para cargar inicialmente los datos del mapa, se consulta la base de datos usersData, seleccionando el documento en el que el campo “user” coincide con el identificador del usuario actual, y extrae los datos de los campos “latitude” y “longitude” del documento. Si el valor de estos es cero, como quiere decir que no existe ubicación, se utilizaría para cargar el mapa el archivo map.html que se

usaba en el MapFragment, para mostrar como ubicación por defecto Madrid; pero en caso de que los valores de latitud y longitud sean distintos de cero, se utiliza el archivo mapuser.html, que utiliza dichas coordenadas para mostrar en el mapa con un marcador esa ubicación. Para pasar los datos de Java al HTML, se utilizan dos métodos "getLongitude()" y "getLatitude()", ambos anotados con la etiqueta @JavaScriptInterface para que sean visibles por el script del archivo HTML.

Para poder seleccionar la nueva ubicación, igual que en el MapFragment, se utiliza la función searchAddress del script de mapuser.html.

Si el usuario hace clic en el botón "Personal data" del ProfileFragment, se abre el PersonalDataFragment, en este fragment se permite tanto que el usuario vea sus datos del perfil, como que los modifique. En la parte superior del fragmet se muestra el logo de la aplicación, y bajo este, tres campos de texto. El primero contiene el nombre y apellidos del usuario, el segundo su email y el tercero la contraseña que en realidad, para evitar hardcodearla en el código, contiene el String "password".

Estos tres campos están creados de forma que no se hace focus sobre ellos al hacer clic, si no que al hacer clic sobre el botón de editar que se encuentra al lado de uno de los campos, se hará focus sobre dicho campo y, en caso de que no se introduzca texto en ellos y se haga clic o se toque en cualquier otra parte de la pantalla, recuperen la información que contenían inicialmente, evitando así que los campos queden vacíos. En caso de que se tenga focalizado un campo vacío y se haga clic en el botón "Save" saltará un error que impedirá que se actualice la información en la base de datos. Todo esto se consigue gracias a la función "onFocusChange()" y el método "onTouchListener()".

Debajo de los campos de texto, existen dos botones, "Back" que permite volver al ProfileFragment sin guardar los datos modificados y "Save" que permite guardar los datos modificados y actualizarlos en la base de datos. Al hacer clic en este botón, se validan los campos, saltando un mensaje de error si alguno está vacío o si la contraseña tiene menos de 6 caracteres. También se comprueba si el contenido de alguno de los campos es diferente al contenido que tenían inicialmente al abrir el fragment, y en caso de que así sea, se actualiza el campo o los campos en cuestión por medio de la ejecución

de los siguientes métodos: “updateNameSurname()”, en el cual se busca en la base de datos “usersData” el documento del usuario actual mediante su identificador único, y actualizando el campo “nameSurname” con la información contenida en el campo de texto. En caso de que el proceso ocurra satisfactoriamente, se muestra un toast con el mensaje “Name updated” y, en caso contrario, se mostraría un toast con el mensaje “Error updating name”.

Para actualizar la contraseña se ejecuta el método “updatePassword()” y utilizando el Firebase user actual y la función específica existente para actualizar la contraseña, se procede a realizar esta acción. En caso de que ocurra satisfactoriamente este proceso, se muestra un toast con el mensaje “Password updated!”, y en caso contrario, se muestra otro toast con el mensaje “Error updating password. Please try logging in again”, ya que muchas veces no es que ocurra un error per sé, sino que la aplicación requiere que el usuario se haya logeado recientemente para poder llevar a cabo este proceso.

Por último, con el método “updateEmail()” se actualiza el email del usuario, utilizando el Firebase user actual y la función verifyBeforeUpdateEmail(), la cual, si el proceso transcurre de forma adecuada, hará que se muestre un toast con el mensaje “Email updated. Please verify your new email address”, indicando al usuario que ha de abrir el correo electrónico que le haya llegado a su bandeja de entrada y hacer clic en el link que contiene para verificar el nuevo email y así poder utilizar su cuenta de Scriptum con esa dirección de correo electrónico. También, para que se apliquen los nuevos cambios, es necesario que el usuario se autentique de nuevo con las nueva scredenciales, por lo que se deslogea al usuario, pasando al LoginFragment, y se muestra un toast con el mensaje “User logged out. Please log in with your new credentials”. En caso de que ocurriese un error al actualizar el email del usuario, se muestra un toast con el mensaje “Error updating email”.

Figura 15. Página PersonalData.

Volviendo al ProfileFragment, cuando el usuario hace clic en el botón de "Upload Book" se abre el fragmento UploadFragment, y el usuario ve un formulario con varios campos para introducir la información del libro y dos botones: uno para seleccionar una imagen del libro y otro para subir los datos del libro. El usuario introduce la información del libro en los campos de texto y selecciona una imagen del libro haciendo clic en el icono de la cámara, lo cual abre el selector de imágenes. Una vez seleccionada la imagen, esta se muestra, y cuando el usuario hace clic en botón para subir los datos y la imagen del libro, el fragmento valida que todos los campos estén cumplimentados correctamente y que la imagen esté seleccionada. Si la validación es correcta, se suben los datos del libro a Firebase Firestore, a su vez la imagen se sube a Firebase Storage, y la URL de descarga se guarda en el documento de Firestore del libro. Si la subida es exitosa, se muestra un mensaje de éxito, pero si hay algún problema durante la subida, se muestra un mensaje de error.

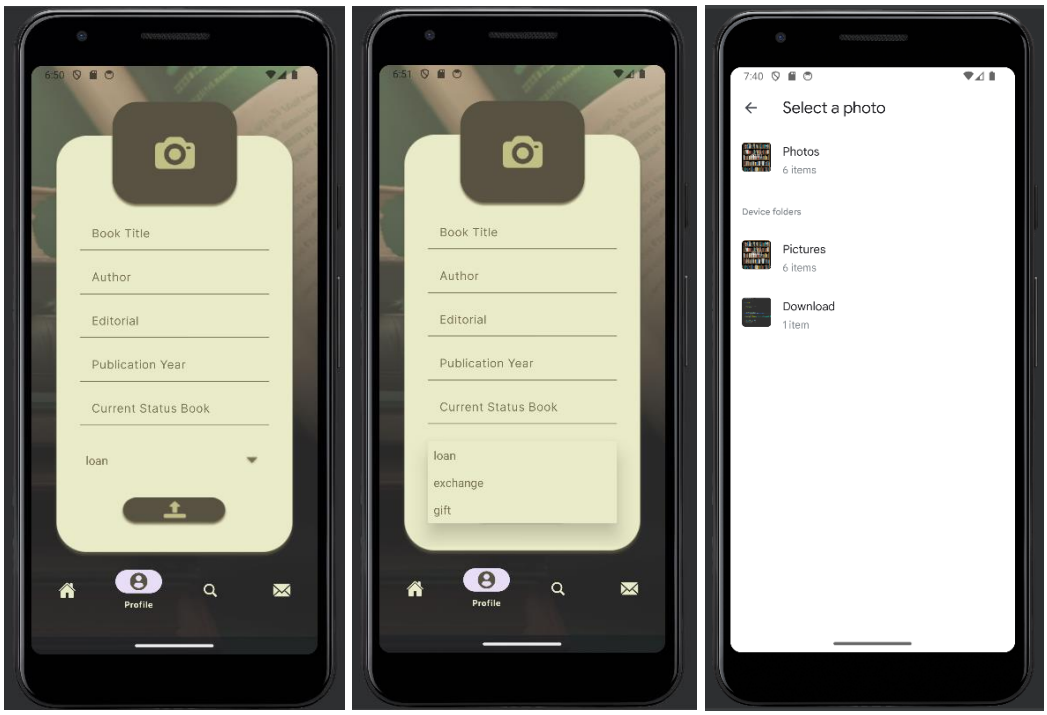


Figura 16. Página para agregar libros y selección de fotografías del dispositivo.

En primer lugar, se definen las variables y los componentes de la interfaz de usuario como son los botones "bookUploadButton' y 'rectanglePhotoBook' estos permiten al usuario subir los datos del libro y seleccionar una imagen, respectivamente. Los campos de texto como 'titleBookEditText', 'authorEditText', editorialEditText, publicationYearEditText, currentStatus, y 'commentsEditTex' son utilizados para introducir la información del libro. El menú desplegable (Spinner) permite al usuario seleccionar el tipo de transacción ya sea préstamo, intercambio o regalo.

En el método onCreateView(), se infla el diseño del fragmento y se configuran los componentes de la interfaz de usuario. Se inicializan Firebase Authentication, Firebase Firestore y Storage. El Spinner está configurado con las opciones préstamo, intercambio o regalo usando un ArrayAdapter.

Se configuran los listeners para los botones de seleccionar imagen 'rectanglePhotoBook' y subir libro 'bookUploadButton' cuando el usuario hace clic en rectanglePhotoBook, se llama a selectImage(), que abre el selector de imágenes mediante un intent que carga la galería del dispositivo. Al hacer clic en bookUploadButton, se obtienen los datos introducidos por el usuario en los campos de texto y el Spinner y se valida que todos los campos estén correctamente y que una

imagen esté seleccionada. Si la validación es correcta, se suben los datos del libro a Firestore y la imagen a Firebase Storage llamando a uploadPhoto().

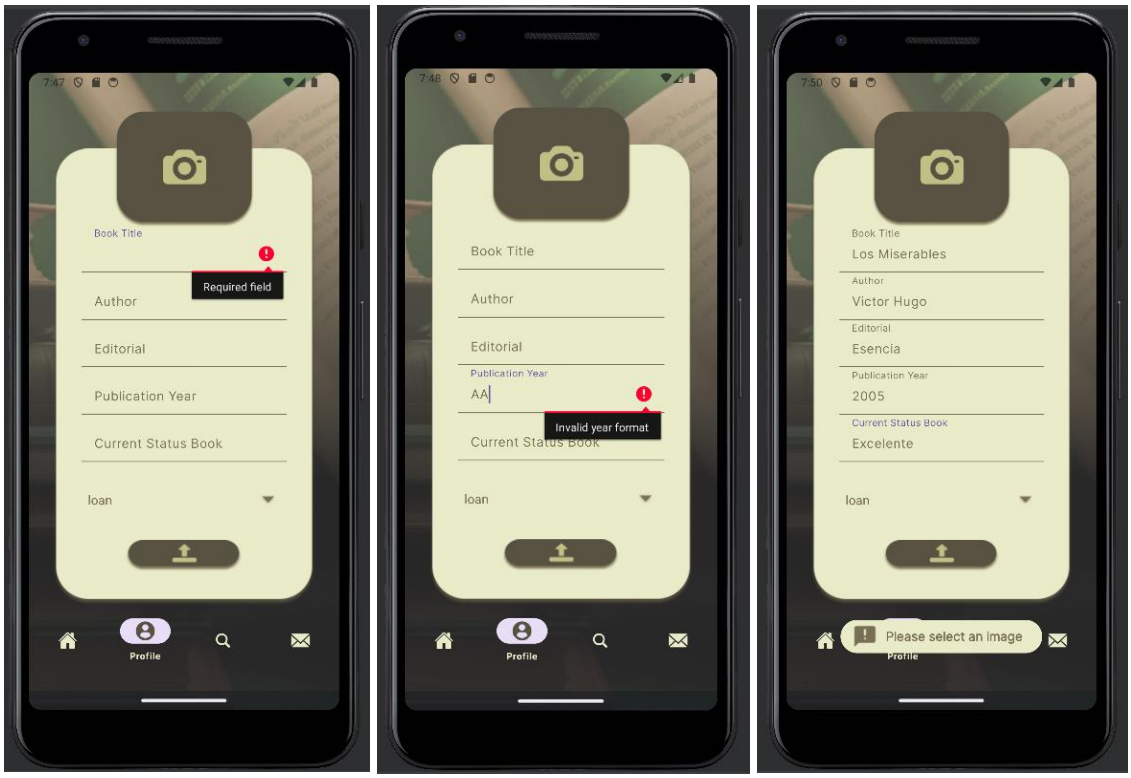


Figura 17. Notificaciones de errores en campos requeridos y en imagen requerida.

El método selectImage() abre un intent para seleccionar una imagen de la galería. El resultado de la selección de imagen se maneja en onActivityResult(), donde se verifica que la selección de imagen fue exitosa y se muestra la imagen seleccionada en 'rectanglePhotoBook'. En el método uploadPhoto(), la imagen seleccionada se sube a Firebase Storage. Luego, se obtiene la URL de descarga de la imagen y se guarda en Firestore en el documento correspondiente del libro.

Para notificar al usuario si las operaciones se han realizado de forma satisfactoria, se utilizan mensajes tipo toast. PositiveToast muestra un mensaje de éxito cuando los datos del libro se han subido correctamente, mientras que negativeToast muestra un mensaje de error si hay algún problema durante la subida. Si el usuario decide dejar de subir libros, haciendo clic en el botón de la barra de menú 'profile', volverá a encontrarse con el fragmento 'ProfileFragment' para desarrollar la tarea que prefiera.

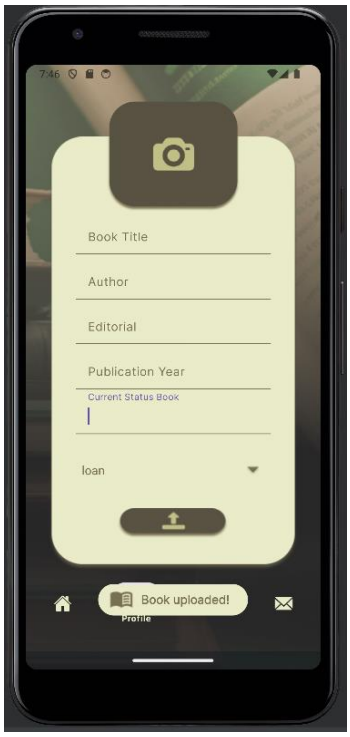


Figura 18. Notificación creada cuando el libro ha sido añadido correctamente.

Cuando el usuario hace clic en el botón ‘Check Books’, la aplicación abre ‘QueriesFragment’ permitiendo navegar a diferentes secciones basadas en las opciones seleccionadas: préstamo de libros, intercambio de libros y donación de libros, por medio de botones.

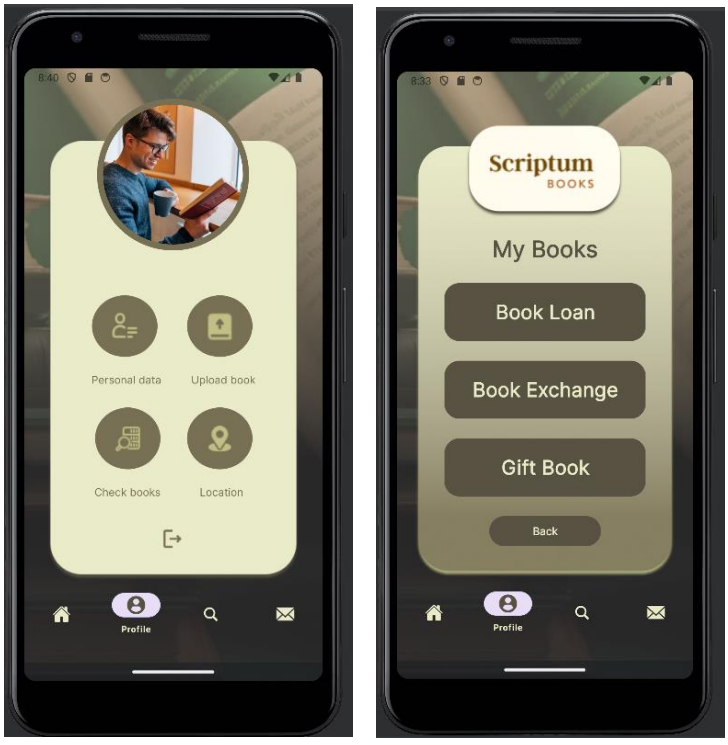


Figura 19. Página de perfil de usuario y de consulta de libros subidos a la base de datos.

Se defienen varios componentes de la interfaz de usuario como son los botones ‘book_loan’, ‘book_exchange’, ‘book_gift’, y back que permiten al usuario navegar a las distintas secciones o regresar al perfil. En el método onCreateView(), se configuran estos componentes. Se inicializan los botones utilizando ‘findViewById’ para vincularlos con los elementos de la interfaz definidos en el archivo de diseño XML (‘fragment_queries’). Además, se asignan OnClickListener() a los botones para manejar los clics.

Cuando el usuario interactúa con estos botones, se ejecuta la lógica definida en el método onClick(). Si el usuario hace clic en book_loan, se llama a replaceFragment para reemplazar el fragmento actual con ‘BookLoanFragment’. De manera similar, si hace clic en book_exchange, se reemplaza el fragmento actual con ‘BookExchangeFragment’, y si hace clic en ‘book_gift’, se reemplaza con ‘GiftBookFragment’. Si el usuario hace clic en back, se reemplaza el fragmento actual con ProfileFragment.

El método replaceFragment() se encarga de realizar estos reemplazos. Utiliza ‘FragmentManager’ y ‘FragmentTransaction’ para gestionar el cambio de fragmentos.

Figura 20. Vista de libros agregados para prestar, intercambiar y regalar.

En la aplicación, los fragmentos ‘BookExchangeFragment’, ‘GiftBookFragment’ y ‘BookLoanFragment’ se utilizan para permitir a los usuarios gestionar diferentes tipos de transacciones de libros: intercambio, regalo y préstamo, respectivamente como anteriormente hemos expuesto. Estos fragmentos comparten una estructura y funcionalidad similares, utilizando Firebase Firestore para obtener y mostrar datos en un ‘RecyclerView’ a través de FirebaseUI.

Cuando el usuario abre uno de estos fragmentos, se inicializan Firebase Authentication y Firebase Firestore. Firebase Authentication (FirebaseAuth) se utiliza para identificar al usuario actual, mientras que Firestore (FirebaseFirestore) proporciona acceso a la base de datos de la aplicación donde se almacenan los datos de los libros. Cada fragmento configura un RecyclerView (mRecycler) para mostrar una lista de libros y un adaptador específico para manejar la visualización de los datos en el RecyclerView. El ‘LinearLayoutManager’ se utiliza para gestionar el diseño vertical de la lista.

La consulta a Firestore (query) se personaliza en cada fragmento para filtrar los libros del usuario actual según el tipo de transacción. En ‘BookExchangeFragment’, la consulta filtra los libros donde el tipo es "exchange" (intercambio). En ‘GiftBookFragment’, filtra los libros donde el tipo es "gift" (regalo). En ‘BookLoanFragment’, filtra los libros donde el tipo es "loan" (préstamo). Estas consultas aseguran que solo los libros relevantes para la transacción específica se muestren al usuario.

Las opciones de consulta se crean utilizando FirestoreRecyclerOptions, lo que permite que FirebaseUI maneje automáticamente los cambios en los datos y actualice el RecyclerView en consecuencia. El adaptador específico para cada fragmento (BookAdapterExchange, BookAdapterGift, BookAdapterLoan) se inicializa con estas opciones y se asigna al RecyclerView para manejar la visualización de los datos.

Cada fragmento también tiene un botón de regreso (‘backBookExchange’, ‘backBookGift’, ‘backBookLoan’) que permite al usuario volver al ‘QueriesFragment’. Este botón llama a un método replaceFragment(), que utiliza ‘FragmentManager’ y ‘FragmentTransaction’ para reemplazar el fragmento actual con el ‘QueriesFragment’.

Esto proporciona una navegación fluida y fácil de usar, permitiendo al usuario moverse entre diferentes secciones de la aplicación sin problemas.

En los métodos onStart() y onStop(), el adaptador comienza y deja de escuchar los cambios en los datos, respectivamente. onStart() llama a startListening() en el adaptador para comenzar a escuchar los cambios en los datos cuando el fragmento está visible, asegurando que el 'RecyclerView' se mantenga actualizado. onStop() llama a stopListening() para detener la escucha de los cambios en los datos cuando el fragmento no está visible, evitando el consumo innecesario de recursos.

Los adaptadores de cada fragment esta compuestos para mostrar listas de libros en un 'RecyclerView', como hemos indicado anteriormente cada uno maneja la visualización y la interacción con los datos de Firestore para diferentes tipos de transacciones de libros: regalo, intercambio y préstamo. Estos adaptadores permiten a la aplicación presentar los datos en tiempo real y gestionar las interacciones del usuario con los elementos de la lista.

Cada adaptador se inicializa con las opciones de consulta de Firestore (FirestoreRecyclerOptions<Book>) y el 'FragmentManager'. El 'FragmentManager' permite a los adaptadores gestionar los cambios de fragmento cuando el usuario interactúa con los elementos de la lista.

El método onCreateViewHolder() infla el diseño de cada elemento en el RecyclerView usando LayoutInflater. El diseño de cada elemento se define en un archivo XML (item_view_book_single.xml), que incluye vistas como 'TextView' para el título, autor y estado del libro, 'ImageView' para la foto del libro, y 'ImageButton' para las acciones de edición y eliminación. Al crear una instancia de 'ViewHolder', se establecen referencias a estas vistas.

El método onBindViewHolder() se encarga de enlazar los datos de un libro específico a las vistas del ViewHolder. Esto incluye establecer el título, autor y estado del libro en los 'TextView' correspondientes y cargar la imagen del libro en el ImageView utilizando Picasso. Además, se configuran listeners para los botones de edición y eliminación.

Cuando el usuario hace clic en el botón de edición, el adaptador obtiene el ID del documento del libro en Firestore y lanza un nuevo fragmento (‘BookEditFragment’, ‘BookEditFragmentExchange’, ‘BookEditFragmentLoan’), pasando este ID como argumento. Esto permite al usuario editar la información del libro. Si el usuario hace clic en el botón de eliminación, el adaptador elimina el documento correspondiente en Firestore. Si la eliminación es exitosa, se muestra un mensaje de confirmación; de lo contrario, se muestra un mensaje de error.

Figura 21. Edición de los datos de un libro y mensaje de que se ha actualizado correctamente.

Los fragmentos de edición

‘BookEditFragmentExchange’, ‘BookEditFragmentGift’, y ‘BookEditFragmentLoan’ son fragmentos de la aplicación que permiten a los usuarios editar la información de sus libros para intercambio, regalo y préstamo, respectivamente. Estos fragmentos facilitan la actualización de detalles como el título, autor, editorial, año de publicación, estado y la imagen del libro. Utilizan Firebase Firestore para almacenar los datos y Firebase Storage para manejar las imágenes. . Además, se proporciona retroalimentación mediante toasts que indican si la operación de guardado fue exitosa o si hubo algún error. Esto asegura una gestión efectiva y actualizada de los libros en la aplicación.

En los fragmentos se obtienen específicamente los IDs del documento del libro en Firestore, lo que permite cargar y actualizar los datos correctos. Firebase Firestore (db) y Firebase Storage (stRe) se inicializan para acceder a los datos del libro y almacenar las imágenes.

En el método onCreateView(), se infla el diseño del fragmento y se configuran los componentes de la interfaz de usuario, que incluyen campos de texto (EditText) para el título, autor, editorial, año de publicación y estado del libro, así como un ImageView para mostrar la imagen del libro. También se configura un botón de guardado. Cuando el usuario presiona este botón, los cambios se guardan en Firestore.

Para cargar los datos actuales del libro en cada uno de los fragments, se utiliza un método (por ejemplo, dataBookExchange() en 'BookEditFragmentExchange') que obtiene la información del libro desde Firestore y la muestra en los campos de texto y el ImageView. Esto asegura que el usuario pueda ver y editar los detalles existentes del libro.

El ImageView permite al usuario seleccionar una nueva imagen del libro. Al hacer clic en el ImageView, se abre un selector de imágenes mediante un intent. El resultado de esta selección se maneja en onActivityResult(), donde la imagen seleccionada se muestra en el ImageView del fragmento.

Cuando el usuario hace clic en el botón de guardado, se llama a un método como saveBookEditExchange(). Este método obtiene los valores de los campos de texto, los valida y, si son válidos, los guarda en Firestore. Si el usuario ha seleccionado una nueva imagen, esta se carga en Firebase Storage y su URL se actualiza en Firestore.

El método para manejar la carga de la imagen (como uploadPhotoLoan() en 'BookEditFragmentLoan') sube la imagen seleccionada a Firebase Storage. Una vez que la imagen se ha cargado, se obtiene la URL de descarga y se actualiza el documento correspondiente en Firestore con esta URL.

Búsqueda de libros

Cuando el usuario pulsa sobre el tercer icono del menú inferior, que representa una lupa, se abre un nuevo fragment en el que aparecen dos botones: uno para acceder a las búsquedas de libros mediante texto y otro para poder visualizar las ubicaciones donde hay libros dados de alta por otros usuarios. Si pulsamos en el primero se abrirá una página con una barra de búsqueda y un botón. Con la segunda opción se visualiza un mapa con puntos de ubicación con los usuarios que tienen libros dados de alta.

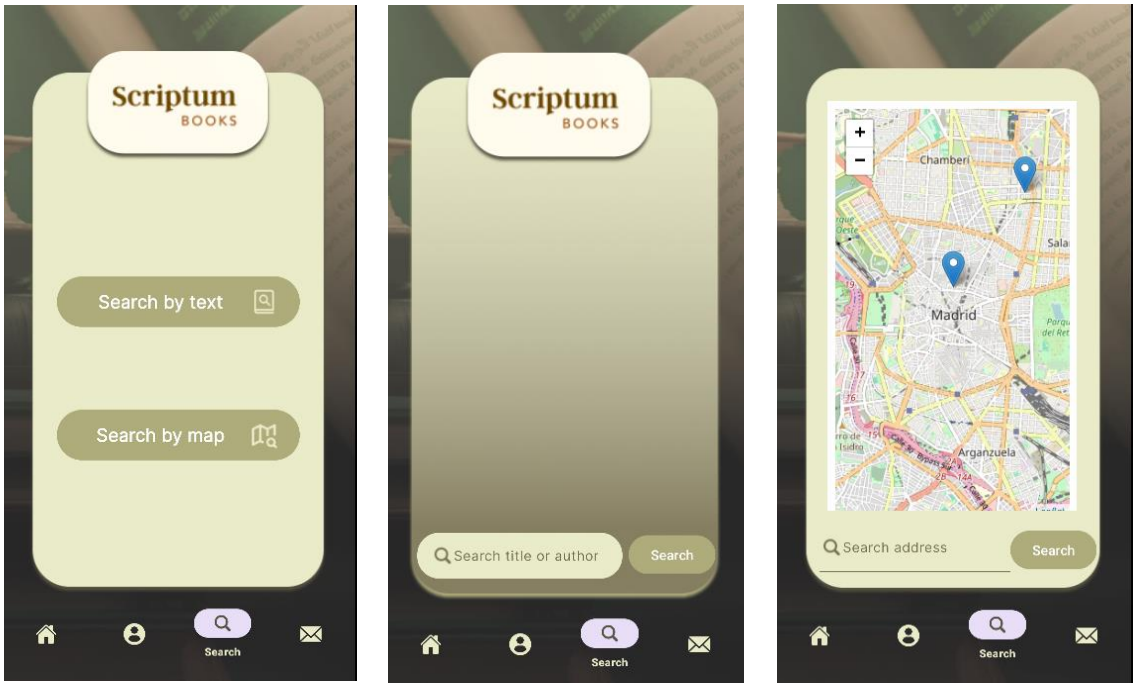


Figura 22. Selector de búsquedas, búsqueda de libros por texto y búsqueda por ubicaciones en mapa.

La lógica del selector de búsquedas se maneja con la clase SearchMenuFragment. Su implementación consiste en una variable de la clase FirebaseUser, para saber si el usuario está identificado o no, y en otras dos para identificar a los botones. En el método onCreateView se crea la vista, se identifica a estos dos botones con su ID correspondiente en el archivo FragmentMenuSearch.xml y se les pone a la escucha con el método setOnClicListener. El método onClic determina, mediante una estructura condicional, sobre qué botón se ha pulsado, reemplazando cada uno la vista actual con su respectivo fragment.

La búsqueda por texto se controla con la clase SearchFragment, utilizando también el adaptador BookAdapterSearch para mostrar los resultados de manera visual.

Tras declarar las variables e identificarlas con los elementos correspondientes del fragment FragmentSearch, se comprueba si la segunda lista donde guardan los títulos de las búsquedas, llamada savedTitlesList, contiene resultados, en cuyo caso se muestran mediante el adaptador. Esto se utiliza para conservar la búsqueda cuando volvemos a ella hacia atrás, evitando que se borren los resultados.

A continuación, se pone a la escucha el botón de búsqueda searchButton, que al hacer clic sobre él recoge el texto que se ha escrito en searchEditText para realizar la búsqueda al pasárselo como parámetro al método searchAndUpdateUi. En él se sitúa un EventListener sobre la colección booksData que responde ante la obtención de resultados a la base de datos. Cuando esto ocurre, limpia todas las listas y recorre los resultados, guardando cada título en titlesList, cada autor en authosList, cada URL de foto en photosList, cada ID de usuario en usersList y cada ID de libro en booksIdList.

También se calcula la relevancia de cada resultado con el método calculateRelevance, que otorga mayor puntuación a aquellos resultados con mayor coincidencia de palabras en los campos title y author de la colección en la base de datos. De este modo, al buscar varias palabras se ordenan los resultados para mostrar primero los que tengan una mayor cantidad de los términos buscados, sin descartar los que solamente tengan una coincidencia parcial, que tendrán menor puntuación y aparecerán después. Este método recorre cada documento y añade un punto cada vez que encuentra una coincidencia. La utilización de múltiples listas para guardar los resultados se debe a que de este modo se pueden pasar al adaptador todos los títulos, autores y demás campos por separado, uniéndose en el controlador según su posición en las listas. Así, los datos del resultado número 1 se guardarán en cada lista en la posición 0.

Como se ha mencionado anteriormente, cada campo se guarda en tres listas diferentes. La primera de ellas sirve para recoger los resultados sin ordenar. En la segunda, estos resultados se ordenan según la puntuación obtenida. La tercera lista se utiliza para guardar los datos del libro cuya ficha se abrirá tras pulsar sobre el item en el que se ven los datos resumidos del libro. Después de realizar la búsqueda y guardar los resultados en las listas, el método envía las listas que se han ordenado por relevancia al adaptador BookAdapterSearch, que se encuentra en otra clase. Allí se infla el item_book, que es el elemento que mostrará los resultados por pantalla, se asigna cada

una de sus partes a una variable y se rellena su contenido con los valores recibidos. En el caso de la URL de la imagen del libro, se carga mediante el método load de Picasso, que admite ciertas opciones de ajustes.

El item en el que se muestran los libros, además de la imagen, el título y el autor, cuenta con un botón para enviarle mensajes al usuario propietario. Por ello, el adaptador presenta un método setOnClickListener para ese botón que a su vez llama a la interfaz OnMessageButtonClickListener para abrir el chat con ese usuario, pasándole la posición del usuario en la lista a SearchFragment, desde donde se lanza el método onMessageButtonClick y se abre definitivamente el fragment MessagesChatFragment. Por otro lado, BookAdapterSearch cuenta con otro setOnClickListener que escucha de la View, por lo que se activa pulsando en cualquier parte del item de cada libro. De esta forma accedemos a la ficha de un libro concreto, gestionada por la clase BookInfoFragment. En este cambio de fragment se utilizan los datos guardados en el tercer grupo de listas, agrupadas en la lista sortedBooksIds. El método onItemClick envía estos datos al nuevo fragment mediante un objeto de la clase Bundle, que se encarga de este tipo de transmisiones de datos entre fragments.

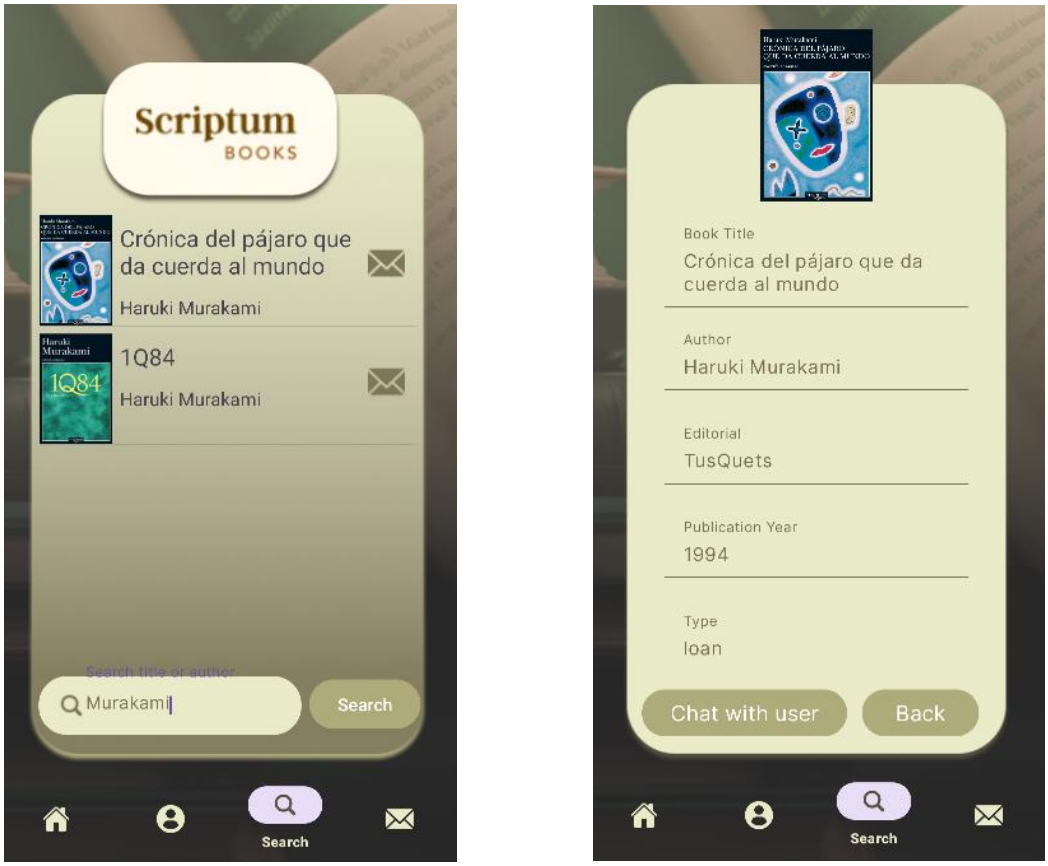


Figura 23. Resultados de búsqueda por texto y ficha del libro.

La segunda forma de búsqueda se realiza a través de un mapa en la clase SearchByMapFragment, la cual presenta un webView que ocupa gran parte de la pantalla, cuya funcionalidad es similar a la mencionada anteriormente en el MapFragment y el LocationFragment, solo que esta vez se vale del archivo searchmap.html, ubicado en la carpeta raw del proyecto.

Mediante el script incluido en dicho archivo, se establece mediante la función "searchAddress()" que la ubicación inicial que muestre el mapa sea la ciudad de Madrid. Lo novedoso de este script respecto a los anteriores, es la implementación de la función "addUserLocation()" la cual es llamada desde el código Java del fragmento, y permite crear marcadores en el mapa, mediante datos de latitud y longitus.

Previo a la llamada de la función, en el código Java se ejecuta el método "obtainUsersLocations()" el cual recorre la base de datos "usersData", extrayendo los campos "latitude" y "longitude". Solo se extraerán estos datos en el caso de usuarios que tengan libros subidos a la aplicación y que no sean el usuario actual de la aplicación, para no mostrar en la búsqueda sus propios libros. Esto se consigue mediante el método "obtainFirebaseLocations()", que genera un array de usuarios teniendo en cuenta las restricciones ya mencionadas, y ese array es el que utiliza el ya mencionado método "obtainUsersLocations()" para obtener la latitud y longitud de los usuarios. Estos datos se mandanal código JavaScript mediante el uso de la ya mencionada función "addUserLocation()", pasándole esos dos valores como parámetro. Así, en el webView se generan marcadores con las ubicaciones de los usuarios.

En el código del archivo searchmap.html, se implementa el método "Android.onMarkerClick()", el cual devuelve al código Java la latitud y longitud del marcador creado. En Java encontramos ese mismo método anotado con la etiqueta @JavascriptInterface y permite, que al hacer clic en un marcador concreto del mapa, se obtenga el identificador único del usuario asociado a ese marcador, y mediante un "Bundle" se pase dicha información al "UsersBookFragment", que muestra una lista con todos los libros subidos por ese usuario.

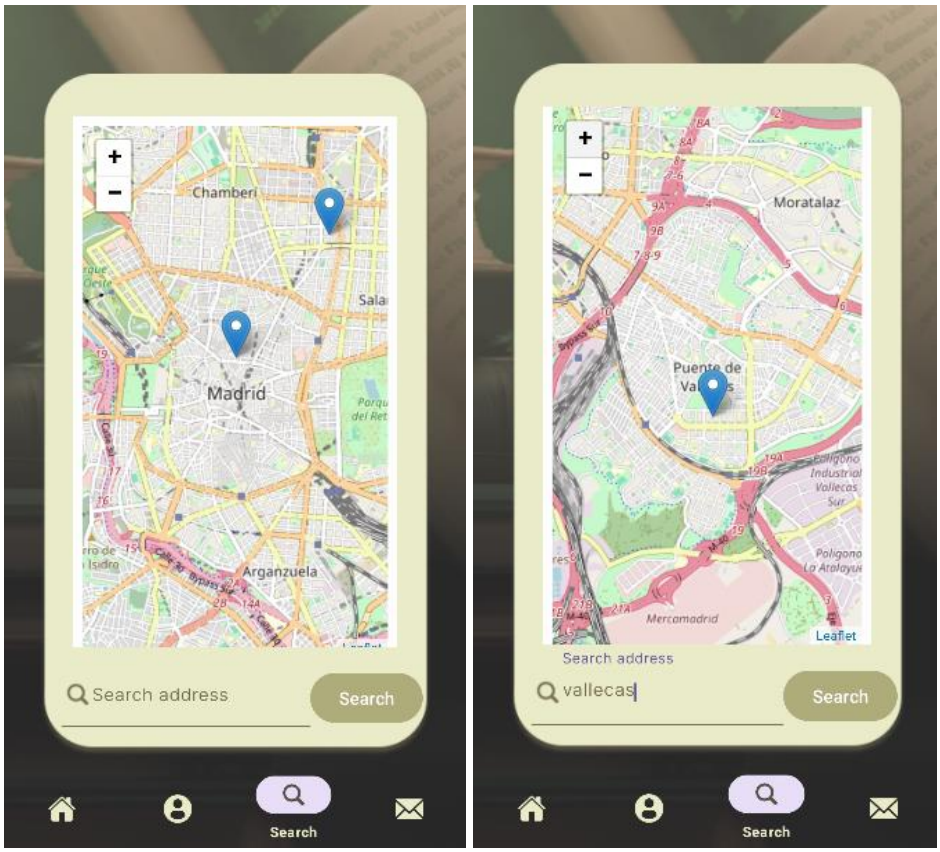


Figura 14. Página de SearchByMap

También, mediante un campo de texto y el botón “Search” se permite que el usuario busque si hay libros disponibles en una ubicación concreta.

El UsersBooksFragment tiene la misma estructura y funcionalidad que el SearchFragment, salvo porque se ha eliminado la barra de búsqueda y se ha añadido el botón “Back”, de forma que si se hace clic en él, se vuelve al SearchByMapFragment; y porque mediante un “Bundle” se obtiene el identificador del usuario del cual queremos mostrar los libros.

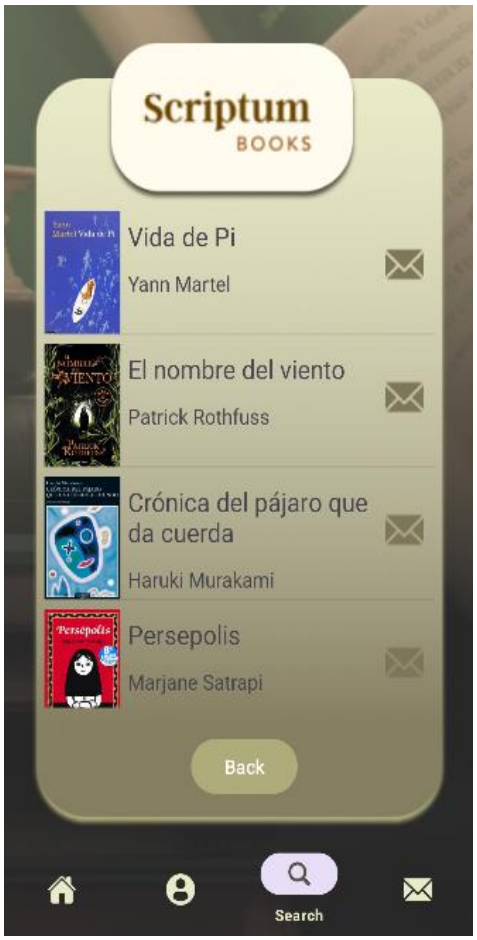


Figura 25. Página UsersBooks

Al hacer clic sobre cualquiera de los ítems de la lista, se ejecuta el método "OnItemClick()" procedente de "BookAdapterSearch", y se genera un "Bundle" que contiene el identificador del libro sobre el que se ha hecho clic, y se le pasa al BookInfoFragment, que mostrará la ficha del libro.

El BookInfoFragment, en la parte superior, muestra la fotografía del libro, y si se hace clic sobre la misma, abre el BookScreenImageFragment, en el que se muestra la imagen ampliada, junto al botón "Back" que permite volver al fragmento anterior, es decir a la ficha del libro al que pertenece la foto.

Debajo de la imagen del libro, se muestran unos campos de texto, sobre los cuales no se puede hacer clic ni focus, que permiten al usuario ver los datos del libro: el título del libro, autor, editorial, año de publicación, si el libro es para intercambiar, regalar o prestar y el usuario a quien pertenece el libro. Debajo aparece un webView que mediante el archivo mapuser.html, muestra la ubicación del usuario en un mapa. Como

todos estos elementos abarcan más que la pantalla de la aplicación, se ha implementado la posibilidad de hacer scroll por eset fragment, pero que al situarse sobre el mapa, no se haga scroll en la pantalla si no que se permita al usuario moverse libremente en el mapa.

En la parte inferior del fragment, se encuenta el botón “Chat with user” que permite iniciar una conversación con el propietario del libro, y el botón “Back”, que al hacer clic sobre él, te lleva al fragmento anterior.

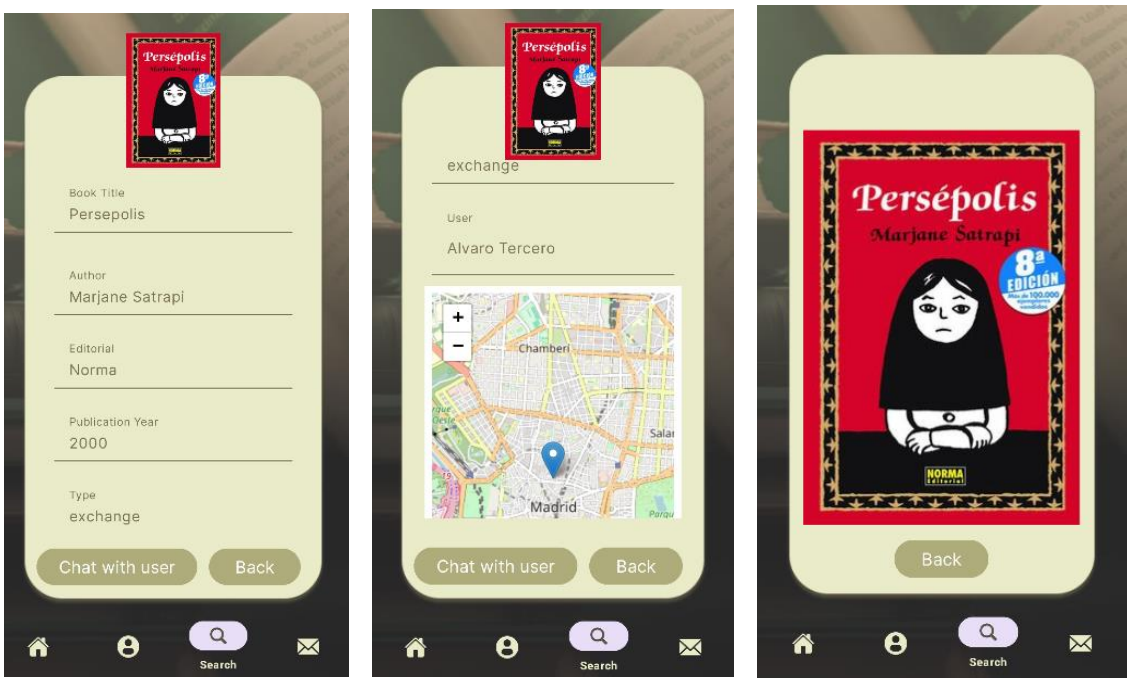


Figura 26. Página BookInfo y FullScreenImage

Mensajes entre usuarios

Como se ha explicado anteriormente, el usuario accede a la opción de enviarle mensajes a un usuario propietario de un libro mediante varias formas: desde la búsqueda mediante texto, la búsqueda en mapa, o desde la propia ficha del libro. En todos estos casos se abre el chat con ese usuario, que se gestiona con la clase MessagesChatFragment.

Al crearse con onCreateView, se infla el fragment y recibe mediante un objeto Bundle el ID del usuario con el que se desea mantener una conversación. Con este dato y el ID de usuario se genera un chatId único mediante el método getChatId, que primero ordena ambos datos alfabéticamente y luego los concatena. Así, el chatId es el mismo tanto en un sentido de la comunicación como en el contrario. Después se identifica a

cada elemento del fragment mediante una variable y se crea un Toast de advertencia avisando de que el usuario no debe compartir datos comprometidos, como por ejemplo contraseñas.

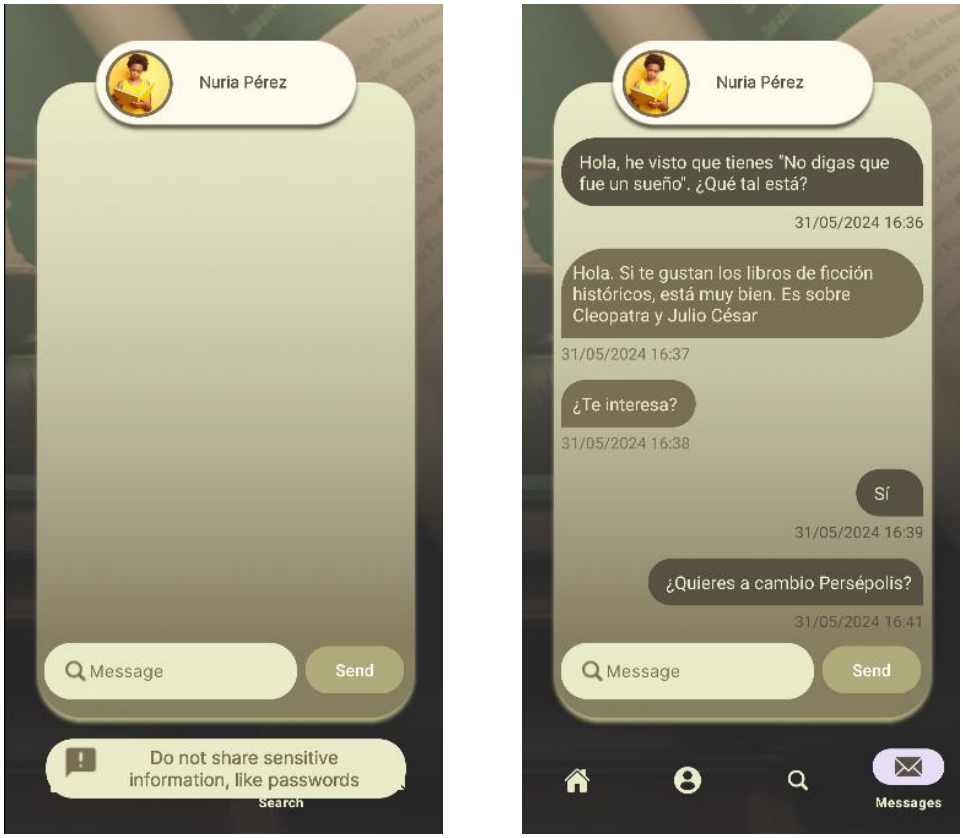


Figura 27. Mensaje de advertencia y ejemplo de mensajes entre usuarios.

Posteriormente se llama al método getUserProfile, que hace una consulta a la base de datos para extraer de la colección usersData los datos del perfil de la persona con la que se va a intercambiar mensajes. Los datos recogidos se utilizan en los componentes de la parte superior del chat, mostrándose el nombre y la foto de perfil. Esta última es gestionada por la clase Picasso, que la ajusta a un tamaño determinado, la centra y la convierte en circular con la clase ImageCircle. Esta clase dibuja un círculo base, ajusta el rectángulo al borde y superpone la imagen recortada en el círculo, creando finalmente un pequeño borde.

La siguiente tarea que se realiza es la inicialización de la lista de mensajes y el adaptador, pasándose al recyclerView. El botón de enviar mensajes se pone a la escucha

con setOnClickListener para enviar mensajes con el método sendMessage. A su vez, se llama al método receiveMessages para recuperar los mensajes anteriores, si los hubiese.

Al llamar a sendMessage se le otorgan como parámetros un chatId, senderId, receiverId y messageText en formato de cadena de texto. Se genera un ID único para cada mensaje y un timestamp para recoger el momento en el que fue enviado.Con todo ello se crea un objeto de la clase Message, el cual es añadido a la colección messages. Al terminar se limpia el Edit Text sobreescribiendo su texto con una cadena vacía.

El método receiveMessages hace una consulta a messages en la que recupera todos los documentos que tengan el chatId igual al que se calculó anteriormente, ordenándose de forma cronológica. Al añadirle un SnapshotListener, permanece constantemente a la escucha, por si se introducen nuevos documentos con el mismo chatId, en cuyo caso aparecerían en tiempo real. Al detectarse, se añadiría a messageList y se notificaría al adaptador del cambio, haciendo un scrollToPosition al recyclerView para que el chat se desplace al último mensaje.

La visualización en pantalla de los mensajes se realiza gracias al adaptador que se encuentra en la clase MessageAdapter. En ella se distingue entre si el usuario es el que ha enviado el mensaje o lo ha recibido, para mostrarse alineado a la derecha o a la izquierda, respectivamente, y de un color u otro. La diferenciación se hace en la clase getItemViewType, devolviendo un número en cada opción. El inflado del mensaje en un item u otro se realiza en el método onCreateViewHolder, y el enlace de los mensajes con las vistas se tramita con onBindViewHolder.

La clase SentMessageHolder sirve para representar las vistas de los mensajes enviados, guardando el texto del mensaje en una variable y el timestamp en otra, para ser añadidas a un Bind que será enviado a MessagesChatFragment. La clase ReceivedMessageHolder hace lo mismo, pero con los mensajes recibidos. Por último, el método formatTimestamp transforma los milisegundos del timestamp a una fecha y una hora reconocible a simple vista por el ser humano.

Por otro lado, al pulsar en el cuarto icono del Bottom Navigation View se abre MessagesFragment. En él se cargan los usuarios con los que hayamos mantenido alguna

interacción comunicativa. Tras inflar el fragmento y asignar las variables a sus respectivos elementos, se asigna un LayoutManager al RecyclerView. Después se carga el método loadChatUsers, que obtiene dichos usuarios. También se encuentra el método allUsers, que carga todos los usuarios de la base de datos. Este método solamente se ha utilizado para realizar pruebas durante el transcurso del desarrollo de Scriptum, por lo que actualmente solo se mantiene con ese fin.

En loadChatUsers se establece un HashSet para guardar los IDs de los usuarios que hemos intentado cargar. Posteriormente se obtienen los documentos de la colección messages que coinciden con el ID del usuario actual, tanto en el campo senderId como en receiverId. En ambas consultas se añaden al HashSet en el caso de que no existan ya, evitando duplicidades. Después se llama al método getUserData para que, al terminar las búsquedas, se obtengan los nombres de perfil y las URLs de las fotos para añadirlas a listas. El método updateUi llamará al UserAdapter para pasarle dichas listas y establece que cuando se ejecute onMessageButtonClick en en adaptador, se creará un nuevo fragment de MessagesChatFragment pasándole el ID del otro usuario mediante un Bundle. Por último, se añadirá el adapter al RecyclerView para así cargar los usuarios de manera satisfactoria.

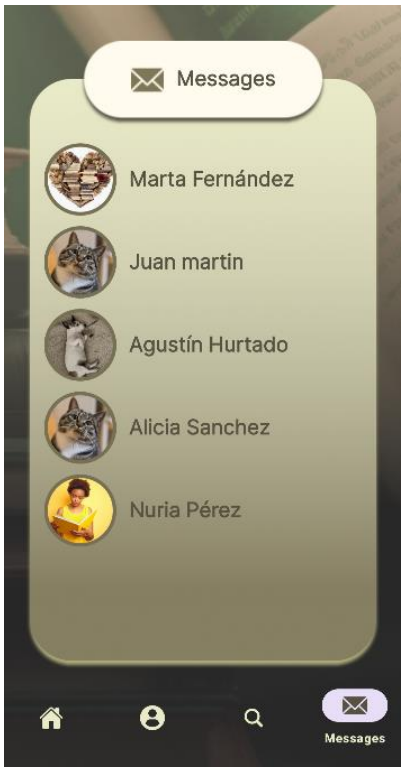


Figura 29. Listado de usuarios con los que se ha mantenido una conversación anteriormente.

El UserAdapter funciona de manera similar a otros adaptadores explicados anteriormente. Al crearse con onCreateViewHolder, es inflado con un item_contacts_messages, que muestra los datos más representativos del usuario: su nombre y fotografía de perfil. Con onBindViewHolder se gestiona tanto el nombre como la fotografía, que es mostrada gracias a la clase Picasso, ajustándose al espacio determinado para ello. Se crea también un onClickListener al itemView para poder abrir el chat con el usuario seleccionado.

6. Conclusiones y mejoras

Tras llevar a cabo todo el proceso de desarrollo de la aplicación, se han implementado los puntos que fueron propuestos al inicio del mismo prácticamente en su totalidad. No obstante, siempre queda margen de mejora y existen ciertas funcionalidades que, de cara a futuro, podrían implementarse. Por ejemplo, sería interesante incluir el género literario en los libros y añadir filtros de búsqueda para facilitar descubrir nuevos títulos.

La realización del desarrollo de este proyecto de fin de ciclo de forma colaborativa ha ayudado a que sus integrantes se aproximen a una modalidad real de trabajo en un proyecto fuera del ámbito académico, así como a aprender a trabajar en equipo, colaborando entre todos, y a comprobar lo fundamental que resulta la organización en un equipo de trabajo, tanto de cara al reparto de tareas como al cumplimento de las fechas de entrega de las mismas.

Este proyecto también ha supuesto un gran aprendizaje de cara a saber afrontar los problemas que surgen en el desarrollo de una aplicación más compleja que las que se han diseñado en las diferentes asignaturas durante el curso, ya que es entonces cuando se aprende a superar los obstáculos que se presentan y donde se resalta la importancia de contar con un buen equipo de trabajo en el que poder apoyarse ante las adversidades.

7. Bibliografía

Android Developers. (2024). Recuperado el 19 de 03 de 2024, de Download Android Studio & App Tools - Android Developers: https://developer.android.com/studio?hl=es-419

Figma: The Collaborative Interface Design Tool. (2024). Recuperado el 19 de 03 de 2024, de Figma: The Collaborative Interface Design Tool: https://www.figma.com/

Git. (2024). Recuperado el 19 de 03 de 2024, de Git: https://git-scm.com/

Canva. (s.f.). Recuperado el 10 de 05 de 2024, de Canva: https://www.canva.com/

Documentación de Firebase > Authentication. (s.f.). Recuperado el 15 de 05 de 2024, de Documentación de Firebase > Authentication: https://firebase.google.com/docs/auth/android/manage-users?hl=es-419

FlatIcon. (s.f.). Recuperado el 10 de 05 de 2024, de FlatIcon: https://www.flaticon.es/iconos

Leaflet. (s.f.). Recuperado el 30 de 05 de 2024, de Leaflet: https://leafletjs.com/

López Suárez, M., & Larrañaga Rubio, J. (Ene - Jun de 2010). El e–book y la industria editorial española. *Revista Interamericana de Bibliotecología, 33*(1), 85-103.