Universidad del País Vasco
Euskal Herriko Unibertsitatea

Ingeniería Informática de Gestión y Sistemas de Información

Desarrollo Avanzado de Software

LibreBook

Estudiante:
Gabiña Barañano, Xabier

22 de abril de 2025

Índice general

Índice de figuras

1. Introducción

Al igual que en la primera entrega, LibreBook es una aplicación móvil para Android diseñada para los amantes de la lectura. Es una plataforma que permite a los usuarios crear una biblioteca personal digital donde pueden registrar, organizar y seguir su progreso en los libros que están leyendo o desean leer.

La aplicación está desarrollada completamente en Java y sigue las mejores prácticas de desarrollo para Android, incluyendo el uso de Room para la persistencia de datos[3], arquitectura MVVM[1], y componentes de la biblioteca de Material Design[4] para ofrecer una interfaz moderna y funcional.
El repositorio de la aplicación se encuentra en GitHub y está disponible para su descarga y uso bajo la licencia MIT.

Repositorio de la aplicación

Además en el repositorio, tambien esta disponible en GitHub el binario de la aplicación en formato apk para su descarga e instalación en dispositivos Android.

Archivo APK de la aplicación

En la aplicación existen por defecto unicamente 10 libros.

1. Crimen y Castigo
2. Los Hermanos Karamazov
3. El Idiota
4. Memorias del Subsuelo
5. El Jugador
6. Los Demonios
7. Humillados y Ofendidos
8. El eterno marido
9. Noches Blancas
10. El doble

Todos de Dostoievski. Tenlo en cuenta a la hora de buscar los libros ya que no se pueden añadir libros directamente desde la aplicación.

La aplicación cuenta con dos cuentas de usuario por defectos:

- **Administrador**
  - Email: admin@xabierland.com
  - Contraseña: admin
- **Xabier**
  - Email: xabierland@gmail.com
  - Contraseña: 123456

No obstante, se pueden crear nuevas cuentas de usuario mediante el registro en la aplicación.

2. Objetivos

2.1. Elementos obligatorios

- **Uso de una base de datos remota para el registro y la identificación de usuarios mediante registro.**
  - Se ha migrado la base de datos de local a remota, manteniendo la misma estructura pero alojándola en un servidor externo.
  - La definición de la base de datos se puede encontrar en **/sql/librebook_db.sql**, con las mismas tablas que se utilizaban en la versión local (usuarios, libros, usuarios_libros).
  - Se ha desarrollado una API REST en PHP que sirve como intermediaria entre la aplicación Android y la base de datos MySQL (**/php/api.php**).
  - Se han implementado nuevas clases en el paquete `com.xabierland.librebook.api` (ApiClient, ApiConfig, ApiParsers) para gestionar la comunicación con la API.
  - Los repositorios (LibroRepository, UsuarioRepository, BibliotecaRepository) se han actualizado para realizar las operaciones CRUD a través de la API en lugar de hacerlo localmente.
- **Integrar los servicios Google Maps y Open Street Map y Geolocalización en una actividad.**
  - Se ha implementado OpenStreetMap en la aplicación mediante la biblioteca OSMdroid.
  - Se ha creado un nuevo fragmento (`MapFragment`) que muestra un mapa interactivo donde se visualizan las librerías cercanas a la ubicación del usuario.
  - El fragmento gestiona los permisos de ubicación y utiliza el proveedor de ubicación de alta precisión de Google (FusedLocationProviderClient).
  - Se ha implementado la clase utilitaria `BookstoreFinder` que utiliza la API de Overpass para buscar librerías y bibliotecas cercanas.
  - Se muestran marcadores personalizados en el mapa para cada librería, con ventanas de información detallada al hacer clic (implementadas mediante `CustomInfoWindow`).
  - El mapa se puede abrir desde la MainActivity a través de un card dedicado, mostrándose como un diálogo a pantalla completa.
- **Captar imágenes desde la cámara, guardarlas en el servidor y mostrarlas en la aplicación. Por ejemplo, una foto de perfil.**
  - Se ha ampliado la funcionalidad de `ProfileActivity` para permitir a los usuarios tomar fotos con la cámara además de seleccionarlas desde la galería.
  - Se han implementado los permisos necesarios para acceder a la cámara y al almacenamiento externo, con diálogos explicativos cuando se necesitan permisos.
  - Se ha utilizado FileProvider para gestionar el acceso a archivos de imagen entre diferentes aplicaciones (cámara y LibreBook).
  - Se ha agregado la clase utilitaria `FileUtils` para procesar las imágenes, corregir su orientación y convertirlas a formato base64 para su almacenamiento.
  - Las imágenes de perfil se transmiten al servidor codificadas en base64 a través de la API y se almacenan en la base de datos remota.
  - Las imágenes se cargan correctamente en el menú de navegación y en las vistas de perfil propio y de otros usuarios.

2.2. Elementos opcionales

- **Uso de algún Content Provider para añadir, modificar o eliminar datos.**
  - Se ha implementado un Content Provider personalizado (`LibreBooksProvider`) que permite a otras aplicaciones acceder a los datos de libros de la biblioteca del usuario.
  - El Content Provider expone dos tipos de URI: una para acceder a la lista de libros (`books`) y otra para obtener archivos de texto con información detallada de un libro específico (`book_files`).
  - Se ha implementado la funcionalidad de compartir libros mediante este Content Provider en `BookDetailActivity` (botón de compartir en la barra de herramientas) y en `BookCardAdapter` (menú contextual al mantener pulsado un libro).
  - La clase `ShareUtils` proporciona métodos para compartir información de libros como texto plano o como archivos de texto a través del Content Provider.
  - El Content Provider permite compartir no solo la información básica del libro, sino también datos adicionales como el estado de lectura, calificación y notas personales.
- **Implementación de un servicio en primer plano y gestión de mensajes broadcast durante el servicio**
  - Se ha desarrollado un servicio en primer plano (`ReadingTimerService`) que permite al usuario registrar su tiempo de lectura incluso cuando la aplicación está en segundo plano.
  - El servicio muestra una notificación persistente con el tiempo transcurrido y controles para detener el temporizador.
  - Se ha implementado un sistema de comunicación bidireccional entre la actividad (`ReadingTimerActivit`) y el servicio mediante un mecanismo de binding y una interfaz de callback (`TimerListener`).
  - El servicio gestiona intents de tipo broadcast para responder a acciones como iniciar, detener y reiniciar el temporizador.
  - Se ha configurado correctamente el servicio para ejecutarse en Android 12+ siguiendo las restricciones de ejecución en segundo plano y solicitando el tipo de servicio adecuado (FOREGROUND_SERVICE_DATA_SYNC).
  - La actividad se asegura de que el servicio siga funcionando incluso si el usuario cierra la aplicación, y puede reconectarse al servicio cuando se vuelve a abrir.
- **Uso de mensajería FCM. Se debe incluir alguna forma de que se pueda probar de forma externa (por ejemplo, con un servicio web PHP adicional en el servidor de la asignatura).**
  - Se ha implementado Firebase Cloud Messaging (FCM) para recibir notificaciones push en la aplicación.
  - Se ha creado un servicio (`LibreBookMessagingService`) que extiende `FirebaseMessagingService` para gestionar los mensajes recibidos.
  - La aplicación se suscribe automáticamente al tema .ªll_devices"para recibir notificaciones generales.
  - Se solicitan los permisos necesarios para mostrar notificaciones en Android 13+.
  - Se ha implementado un panel de administración web accesible desde esta URL que permite enviar notificaciones a todos los dispositivos suscritos.
  - Las notificaciones pueden contener un título, un mensaje y abren la aplicación al ser pulsadas.
- **Desarrollar un widget que tenga, al menos, un elemento que se actualice automáticamente de manera periódica.**

- Se ha implementado un widget de escritorio (`BookRecommendationWidgetProvider`) que muestra recomendaciones de libros aleatorios de la biblioteca del usuario.
- El widget se actualiza automáticamente cada 15 segundos mediante un AlarmManager para mostrar diferentes recomendaciones.
- El widget muestra la portada del libro, el título y el autor, y al pulsarlo, abre la actividad de detalle del libro correspondiente.
- Se ha implementado la clase utilitaria `WidgetImageLoader` para cargar imágenes desde URLs en el widget de forma asíncrona.
- El widget gestiona adecuadamente su ciclo de vida, incluyendo la cancelación de actualizaciones cuando se elimina y la programación de nuevas actualizaciones cuando se crea.
- Se han utilizado RemoteViews para diseñar la interfaz del widget, que se adapta a diferentes tamaños de pantalla.

- **Uso de algún servicio o tarea programada mediante alarma (no valen las alarmas del widget).**

3. Descripción de la aplicación

3.1. Clases

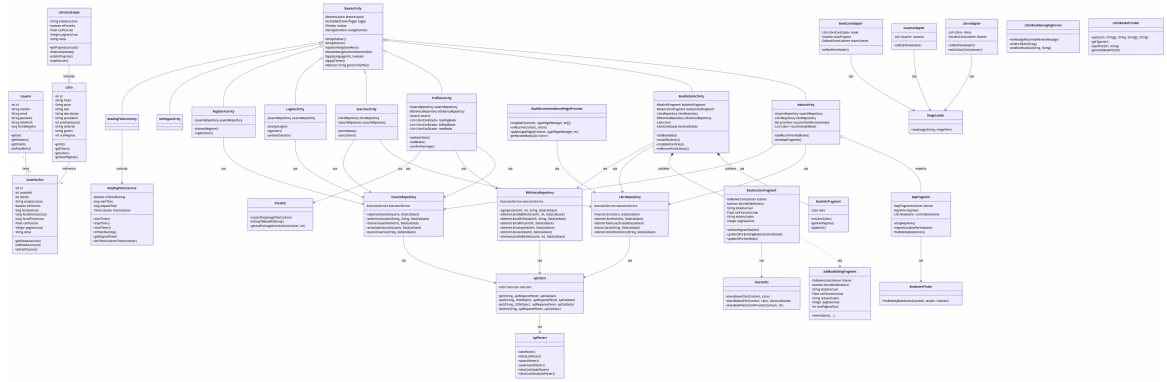

Figura 3.1: Diagrama de clases de la aplicación LibreBook

Como se muestra en la Figura 3.1, la ha aumentado el número de clases respecto a la entrega anterior pero manteniendo las capas con una clara separación de responsabilidades[2]: En este apartado me voy a limitar a describir los cambios realizados en la aplicación respecto a la entrega anterior.

3.1.1. Activities

- **Modificaciones en ProfileActivity:** Se ha ampliado para permitir la captura de imágenes mediante la cámara o su selección desde la galería, ajustarlas y guardarlas como fotos de perfil.

3.1.2. Fragments

- **MapFragment:** Fragment que muestra un mapa interactivo con las librerías cercanas a la ubicación del usuario utilizando OpenStreetMap.

3.1.3. Services

- **ReadingTimerService:** Implementa un servicio en primer plano que mantiene un temporizador de lectura activo incluso cuando la aplicación está en segundo plano. Muestra una notificación persistente que se actualiza con el tiempo transcurrido.
- **LibreBookMessagingService:** Extiende FirebaseMessagingService para recibir y procesar mensajes push. Gestiona tanto mensajes de notificación como mensajes de datos.

3.1.4. API

- **ApiClient:** Proporciona métodos genéricos para realizar peticiones HTTP (GET, POST, PUT, DELETE) a nuestra API REST. Gestiona la comunicación asíncrona y el manejo de errores.

- **ApiConfig:** Contiene la configuración básica como la URL base de la API.
- **ApiParsers:** Conjunto de parseadores que convierten las respuestas JSON en objetos del dominio de la aplicación (Libro, Usuario, LibroConEstado, etc.).

3.1.5. Provider

- **LibreBooksProvider:** Content Provider que permite a otras aplicaciones acceder a la información de libros de LibreBook.

3.1.6. Widget

- **BookRecommendationWidgetProvider:** Implementa un widget de escritorio que muestra recomendaciones de libros y se actualiza periódicamente.

3.1.7. Utils

- **BookstoreFinder:** Clase utilitaria que utiliza la API de Overpass para buscar librerías y bibliotecas cercanas a una ubicación dada.
- **CustomInfoWindow:** Componente personalizado para mostrar información detallada sobre una librería al seleccionar su marcador en el mapa.
- **ShareUtils:** Proporciona métodos para compartir información de libros como texto plano o como archivos de texto.
- **WidgetImageLoader:** Clase utilitaria para cargar imágenes de portadas de libros en el widget.
- **FileUtils:** Proporciona métodos para gestionar archivos, convertir entre diferentes formatos de imagen y corregir orientación de imágenes capturadas.

3.2. Base de datos

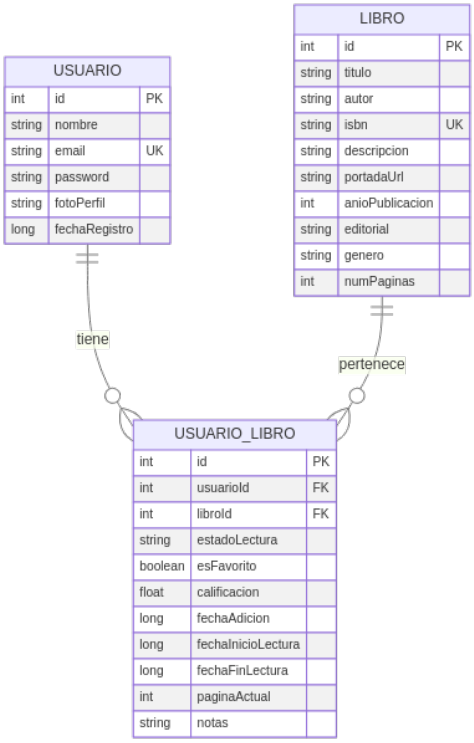

Figura 3.2: Diagrama de la base de datos de la aplicación LibreBook

La base de datos no ha sufrido cambios respecto a la entraga anterior.

4. Manual de usuario

5. Dificultades

5.1. Base de datos remota

Mover la base de datos de local a remota ha supuesto un reto importante. Primero al usar la arquitectura Room no tenia unas sentencias SQL definidas para la estructura de la base de datos por lo que tuve que crearla de cero. Ademas, era necesario el uso de una API REST para poder interactuar con la base de datos.

5.2. Servicio de tiempo de lectura

Implementar el servicio de lectura tubo sus dificultades especialmente la parte de broadcast. Yo queria implementar una notificacion que constantemente se estuviera actualizando para que estando fuera de la aplicacion y de actividad fuera posible saber el tiempo de lectura e incluso deternerlo. Lo primero que intente fue usar un WorkManager y poner el PeriodicWorkRequest a 1 segundo pero Android pone que el tiempo minimo es de 15 minutos por lo que tuve que rescribir el servicio u usar un ForegroundService. Esto supuso reescribir el servicio y la actividad de tiempo de lectura completamente.

6. Conclusiones

Aunque esta entrega haya sido más corta en cuanto a funcionalidades se me ha hecho bastante más complejas de implementar.

Bibliografía

[1] Microsoft. *The MVVM Pattern*. 2012. URL: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10).

[2] Android Developers. *Guide to app architecture*. 2023. URL: https://developer.android.com/topic/architecture.

[3] Android Developers. *Room Persistence Library*. 2023. URL: https://developer.android.com/training/data-storage/room.

[4] Google. *Material Design*. 2023. URL: https://material.io/design.