Pixora

Desarrollo de aplicaciones Android

Descargar Prueba Cerrada

https://play.google.com/store/apps/details?id=com.ibrepidevs.pixora

Máster Universitario en Informática Móvil

Israel Brea Piñero

Universidad Pontificia de Salamanca

Índice

1. Introducción

1.1. Motivación

En la era digital actual, la imagen se ha convertido en una de las formas más poderosas de comunicación. Desde redes sociales hasta entornos profesionales, las fotografías inspiran, conectan y permiten compartir ideas de forma visual y directa. Esta evolución ha fomentado el desarrollo de plataformas centradas en la creación y descubrimiento de contenido visual, como Pinterest o Instagram. La posibilidad de explorar imágenes temáticas, guardar inspiración, organizar ideas y compartir contenido visual ha transformado la manera en que los usuarios interactúan con la creatividad digital.

Ligado a esta transformación digital y al creciente protagonismo del contenido visual, surgió la idea de crear Pixora. Me pareció un proyecto interesante a nivel técnico y con suficiente profundidad como para aplicar los conocimientos aprendidos durante la asignatura de Desarrollo de Aplicaciones Android. Además, encajaba perfectamente con los requisitos establecidos.

1.2. Alcance

Pixora está pensada como una aplicación orientada a todos los usuarios que deseen buscar, explorar y organizar contenido visual de forma sencilla. Su diseño está centrado en la experiencia del usuario, permitiendo funcionalidades básicas como la búsqueda de imágenes, la creación de colecciones personalizadas, la posibilidad de dar "me gusta", y la carga de fotografías propias. Aunque la aplicación toma inspiración en otras plataformas similares, se ha desarrollado desde cero utilizando tecnologías modernas como Jetpack Compose, con un enfoque educativo y sin ánimo de lucro.

Actualmente, este proyecto está limitado a un entorno académico, sin proyección comercial. No obstante, la aplicación ha sido diseñada siguiendo buenas prácticas de desarrollo móvil, y podría escalarse en un futuro para incluir nuevas funcionalidades como interacción social, sistema de seguidores o recomendaciones personalizadas, si el objetivo evolucionara más allá del uso didáctico.

1.3. Objetivos

1.3.1. Objetivo general

Diseñar y desarrollar una aplicación móvil nativa para Android centrada en la exploración, organización y subida de contenido fotográfico, aplicando buenas prácticas de arquitectura, diseño visual y gestión de datos, como parte del

aprendizaje práctico de la asignatura Desarrollo de Aplicaciones Android del Máster en Informática Móvil (MIMO) de la Universidad Pontificia de Salamanca.

1.3.2. Objetivos específicos

1. Implementar una interfaz de usuario completamente nativa y adaptable con Jetpack Compose.
2. Aplicar una arquitectura moderna (MVVM + Clean Architecture) que separe las responsabilidades.
3. Uso de ViewModels para gestionar el estado de la UI.
4. Uso de Coroutines para la asincronía .
5. Uso de Flows/StateFlow para el flujo de datos reactivo.
6. Consumir servicios web externos para la obtención de datos mediante llamadas a una API REST, utilizando la librería Retrofit.
7. Persistir datos localmente de forma eficiente mediante el uso combinado de Preferences DataStore para configuraciones simples y la base de datos Room para la gestión de datos estructurados.
8. Desarrollar una navegación fluida e intuitiva dentro de la aplicación utilizando Jetpack Compose Navigation.
9. Establecer un sistema de diseño consistente y personalizable a través de MaterialTheme en Jetpack Compose, definiendo una paleta de colores propia y centralizando colores, tipografías y formas en ficheros de recursos.
10. Configurar variantes de compilación (build variants) con Gradle para gestionar diferentes versiones de la aplicación (desarrollo y producción) con configuraciones o funcionalidades específicas. Además de filtrado de variantes innecesarias.
11. Uso de gestión de permisos en tiempo de ejecución (Android 6.0+).
12. Crear un mínimo de tres componentes Composable reutilizables y personalizados.
13. Implementación de notificaciones compatibles con todas las versiones.
14. Uso de LaunchedEffects, SideEffect y DisposableEffect.
15. Uso de ActivityContracts (además de para permisos) para multimedia.
16. Subir la aplicación a la Google Play Store en fase Beta e invitar a los profesores.

2. Documentación Funcional

En esta sección se detalla la descripción funcional de la aplicación desde la perspectiva del usuario final.

2.1. Interfaz de usuario

2.1.1. Logo

Figura 2.1: Logo Pixora

2.1.2. Diseño app

Figura 2.2: SpashScreen

Figura 2.3: HomeScreen

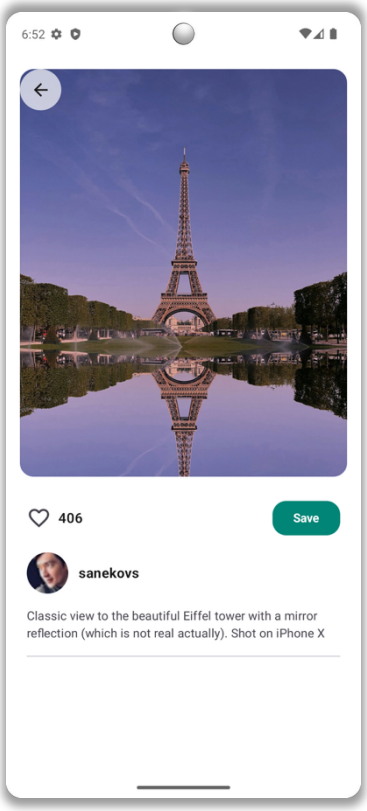

Figura 2.4: PhotoDetailScreen

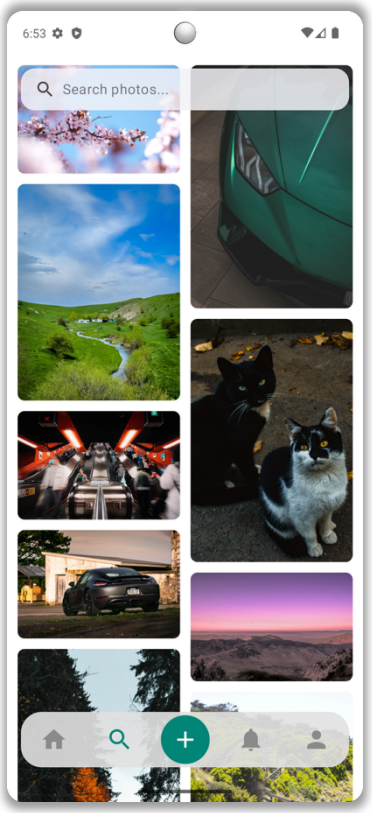

Figura 2.5. SearchScreen

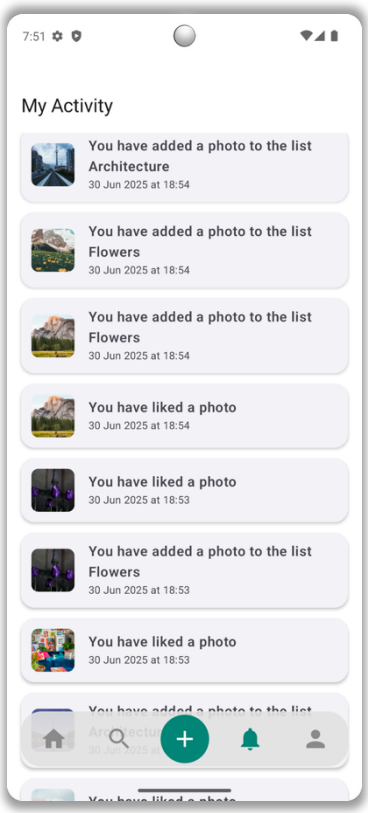

Figura 2.6. ActivityScreen

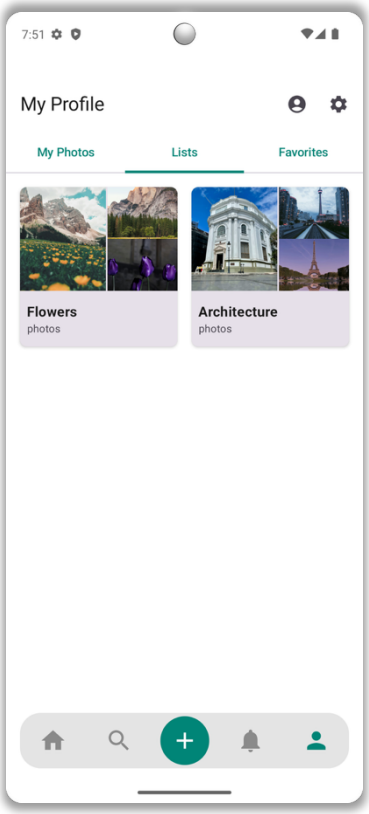

Figura 2.7. MyListsScreen

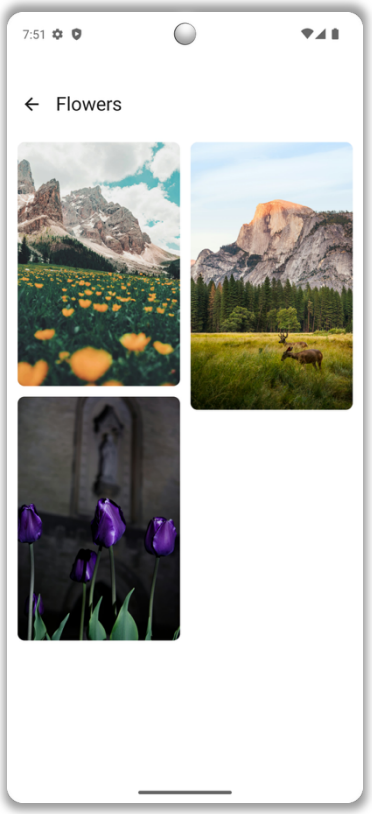

Figura 2.8. PhotoListDetailScreen

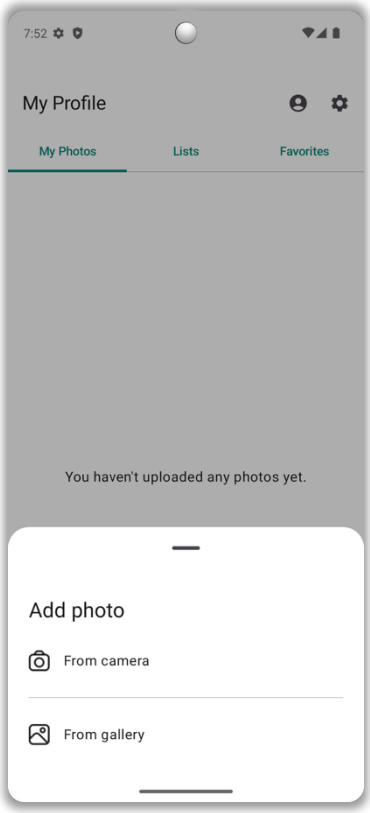

Figura 2.9. BottomSheetScreen

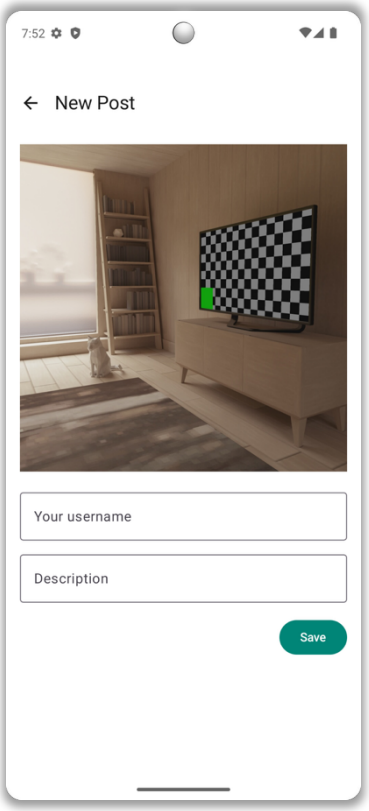

Figura 2.10. PostPhotoScreen

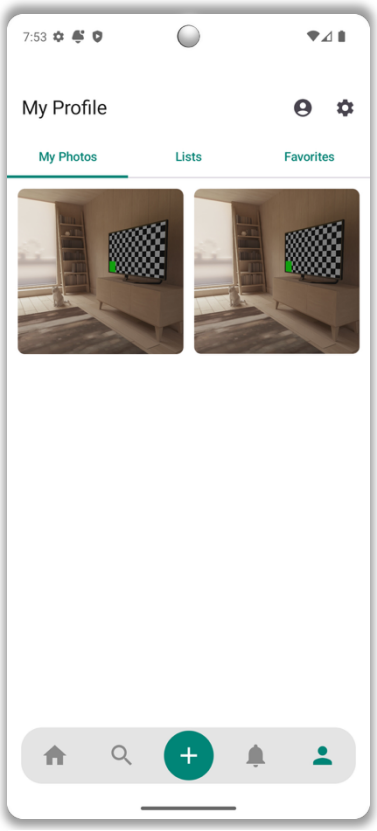

Figura 2.11. MyPhotosScreen

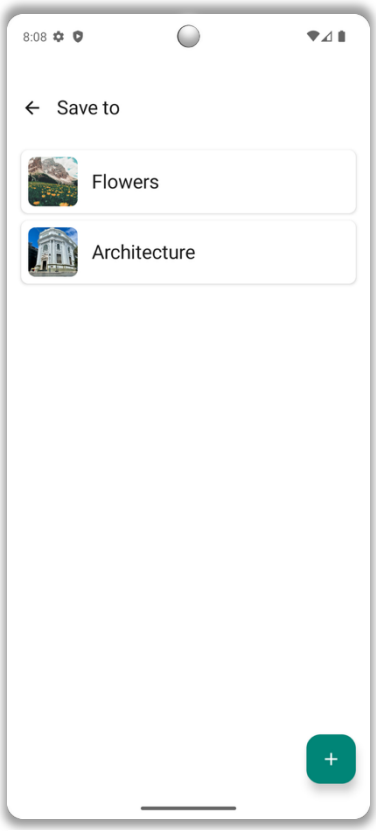

Figura 2.12. SavePhotoToListScreen

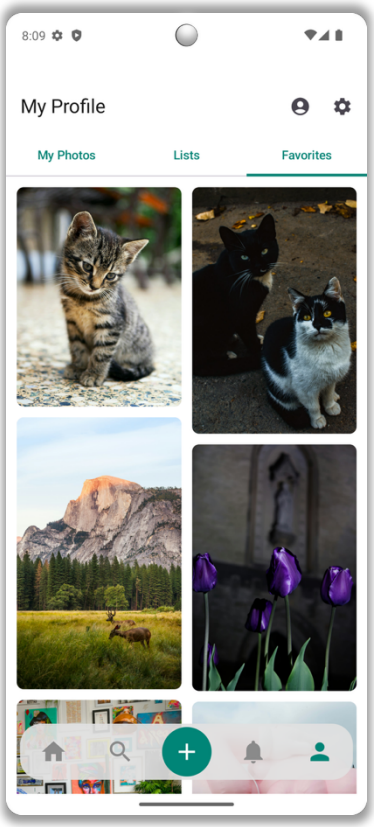

Figura 2.13. FavoritesScreen

2.2. Catálogo de requisitos

2.2.1. Requisitos de información

RI-01	Información sobre las fotos
Descripción	El sistema deberá almacenar la información correspondiente a las fotos. En concreto:
Datos específicos	• Imagen (URL o localURI) • Descripción • Nº de me gustas • Usuario del fotógrafo • Foto de perfil del fotógrafo • Color de fondo • Si me gusta la foto o no • Tamaño de la foto (width y height)

Tabla 2.1. Requisito de información de fotos

RI-02	Información sobre las listas de fotos
Descripción	El sistema deberá almacenar la información correspondiente a las fotos añadidas a listas. En concreto:
Datos específicos	• Nombre de la lista • Fecha de creación de la lista

Tabla 2.2. Requisito de Información de listas de fotos

RI-03	Información sobre las fotos añadidas a listas
Descripción	El sistema deberá almacenar la información correspondiente a las listas de fotos. En concreto:
Datos específicos	• Id de la foto • Id de la lista • Fecha a la que se añadió la foto a la lista

Tabla 2.3. Requisito de Información de fotos añadidas a listas

RI-04	Información sobre la actividad del usuario
Descripción	El sistema deberá almacenar la información correspondiente a la actividad del usuario en la aplicación. En concreto:
Datos específicos	• Tipo de actividad (ya sea dar me gusta o añadir a lista) • Fecha en la que se produce la actividad • Id de la foto • Id de la lista • Nombre de la lista

Tabla 2.4. Requisito de Información de actividad del usuario

2.2.2. Requisitos funcionales

RF-01	Navegación entre secciones
Descripción	El sistema deberá permitir al usuario navegar desde la pantalla de inicio a las secciones de búsqueda, añadir contenido, notificaciones y perfil.

Tabla 2.5. Requisito funcional – Navegación entre secciones

RF-02	Ver fotos por categoría
Descripción	El sistema deberá mostrar fotos clasificadas por categorías (ej. naturaleza, tecnología, viajes...) en la pantalla de inicio.

Tabla 2.6. Requisito funcional – Ver fotos por categoría

RF-03	Buscar fotos
Descripción	El sistema deberá permitir al usuario buscar fotos mediante un campo de texto en la pantalla de búsqueda.

Tabla 2.7. Requisito funcional – Buscar fotos

RF-04	Scroll infinito en búsqueda
Descripción	El sistema deberá ir cargando nuevas fotos automáticamente a medida que el usuario hace scroll hacia abajo en la pantalla de búsqueda.

Tabla 2.8. Requisito funcional – Scroll infinito de búsqueda

RF-05	Ver detalle de foto
Descripción	El sistema deberá mostrar una vista detallada de la foto al pulsar sobre ella.

Tabla 2.9. Requisito funcional – Ver detalle de foto

RF-06	Guardar en favoritos
Descripción	El sistema deberá permitir al usuario marcar una foto con "me gusta”.

Tabla 2.10. Requisito funcional – Guardar en favoritos

RF-07	Quitar de favoritos
Descripción	El sistema deberá permitir al usuario retirar el "me gusta" de una foto.

Tabla 2.11. Requisito funcional – Quitar de favoritos

RF-08	Añadir foto a lista
Descripción	El sistema deberá permitir al usuario guardar una foto en una lista existente.

Tabla 2.12. Requisito funcional – Añadir foto a lista

RF-09	Ver listas a las que puedo añadir foto
Descripción	El sistema deberá mostrar las listas existentes a la hora añadir una foto a lista.

Tabla 2.13. Requisito funcional – Ver listas a las que puedo añadir foto

RF-10	Crear nueva lista
Descripción	El sistema deberá permitir al usuario crear una nueva lista.

Tabla 2.14. Requisito funcional – Crear nueva lista

RF-11	Ver mis listas
Descripción	El sistema deberá permitir al usuario ver las listas que ha creado.

Tabla 2.15. Requisito funcional – Ver mis listas

RF-12	Ver detalle de una lista
Descripción	El sistema deberá mostrar las fotos guardadas dentro de una lista seleccionada.

Tabla 2.16. Requisito funcional – Ver detalle de una lista

RF-13	Ver fotos con me gusta
Descripción	El sistema deberá mostrar al usuario las fotos a las que ha dado "me gusta”.

Tabla 2.17. Requisito funcional – Ver fotos con me gusta

RF-14	Subir foto desde cámara
Descripción	El sistema deberá permitir al usuario seleccionar una imagen desde la cámara para subir una foto.

Tabla 2.18. Requisito funcional – Subir foto desde cámara

RF-15	Subir foto desde galería
Descripción	El sistema deberá permitir al usuario seleccionar una imagen desde la galería para subir una foto.

Tabla 2.19. Requisito funcional – Subir foto desde galería

RF-16	Rellenar datos al subir foto
Descripción	El sistema deberá permitir al usuario subir una foto introduciendo opcionalmente su nombre de usuario y la descripción de la foto.

Tabla 2.20. Requisito funcional – Rellenar datos al subir foto

RF-17	Ver fotos subidas
Descripción	El sistema deberá mostrar al usuario todas las fotos que ha subido.

Tabla 2.21. Requisito funcional – Ver fotos subidas

RF-18	Ver mi actividad
Descripción	El sistema deberá mostrar un historial de actividades del usuario (me gustas, fotos guardadas, etc.).

Tabla 2.22. Requisito funcional – Ver mi actividad

RF-19	Navegación atrás
Descripción	El sistema debe permitir la navegación a la página anterior para una experiencia fluida del usuario.

Tabla 2.23. Requisito funcional – Navegación atrás

RF-19	Cambiar modo de visualización
Descripción	El sistema debe permitir cambiar el modo de visualización de las fotos, permitiéndolas ver en modo mosaico o modo linear list

Tabla 2.24. Requisito funcional – Cambiar modo de visualización

RF-19	Cambiar de idioma
Descripción	El sistema debe permitir cambiar de idioma la aplicación, permitiendo cambiar a Español, Inglés o Francés.

Tabla 2.25. Requisito funcional – Cambiar de idioma

2.2.3. Requisitos no funcionales

A continuación, se presentan los diferentes requisitos no funcionales que se han identificado.

- **Rendimiento**: El tiempo de respuesta de las operaciones principales (carga de fotos, navegación entre pantallas, interacciones con imágenes) deberá ser inferior a 2 segundos para garantizar una experiencia fluida.
- **Escalabilidad**: El sistema debe estar diseñado siguiendo una arquitectura modular, que permita incorporar nuevas funcionalidades o ampliar el número de usuarios sin necesidad de rehacer la estructura principal.
- **Usabilidad**: La interfaz debe ser clara, intuitiva y accesible incluso para usuarios sin conocimientos técnicos. El sistema debe guiar al usuario en las principales tareas, como subir una foto, buscar contenido o guardar favoritos, sin ambigüedad ni elementos confusos.
- **Mantenibilidad**: El código deberá seguir buenas prácticas de programación (como separación de responsabilidades, gestión de dependencias…) y estar correctamente documentado para permitir futuras ampliaciones o mantenimiento.
- **Compatibilidad**: La aplicación debe ser compatible con la mayoría de dispositivos Android activos en el mercado actual.
- **Persistencia**: El sistema debe ser capaz de almacenar localmente información relevante (como fotos, listas o preferencias del usuario).
- **Notificaciones**: La aplicación debe implementar notificaciones locales.

2.3. Casos de uso

A continuación, se presenta el diagrama de casos de uso de la aplicación, en correspondencia con los requisitos funcionales ya expuestos. Posteriormente, se describirán los escenarios principales y alternativos asociados a estos diagramas.

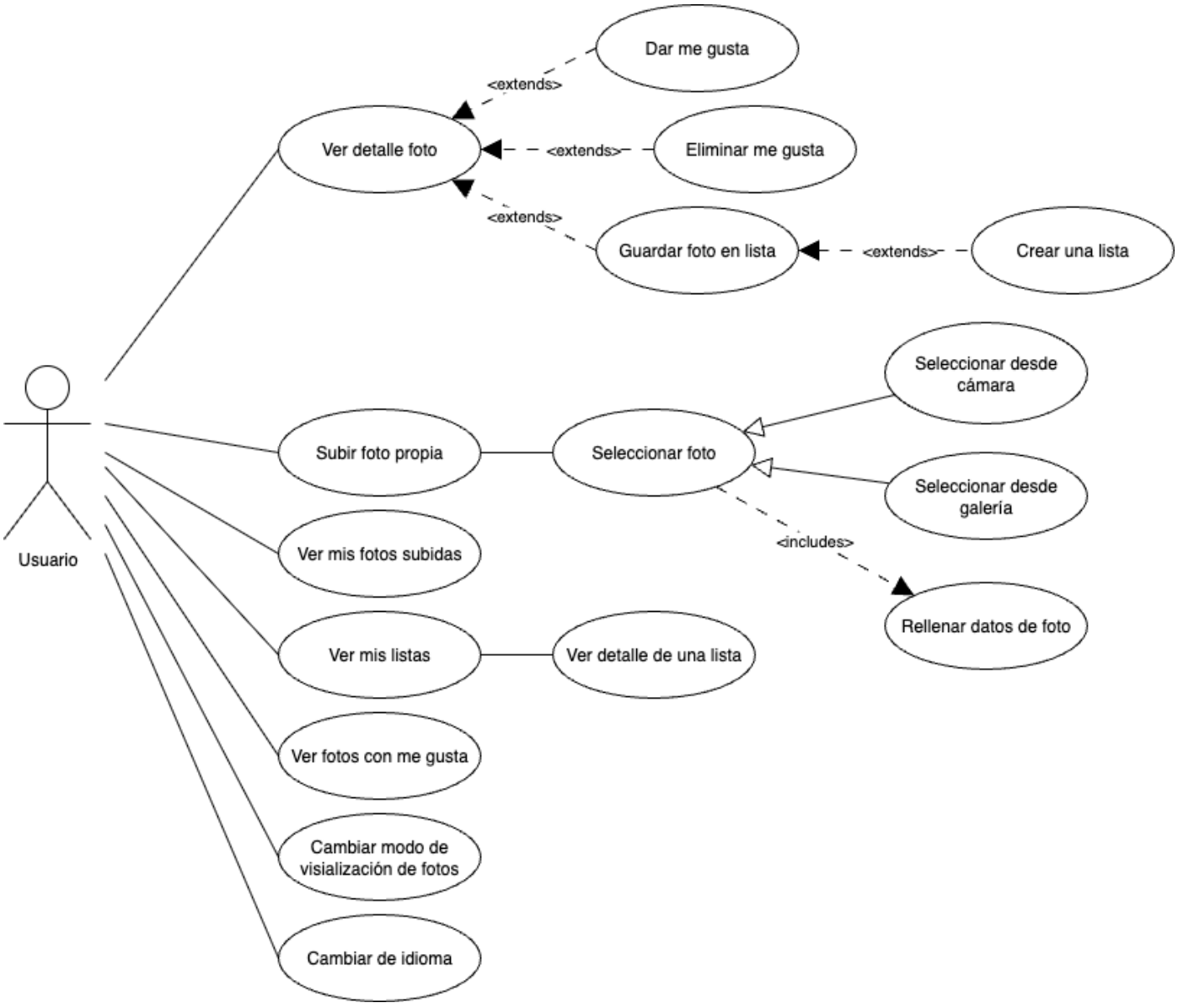

Figura 2.14. Diagrama de casos de uso de Pixora

2.4. Descripción de escenarios

En esta sección se describe la descripción de los escenarios de casos de uso más relevantes de la aplicación.

UC-01	**Ver detalle de foto**	
Descripción	El sistema permite al usuario ver los detalles de una foto específica.	
Precondición	El usuario debe estar en cualquier pantalla que muestre fotos (Home, Search, Likes, MyPhotos, Detalle de Lista).	
Secuencia	**Pasos**	**Acción**
	1	El usuario pulsa sobre una foto.
	2	El sistema muestra el detalle de la foto.
Postcondición	El sistema muestra la pantalla de detalle de la foto, incluyendo la imagen, información asociada, botón de "me gusta" (corazón) y botón de "guardar".	
Excepciones	**Pasos**	**Acción**
	2	El inicio de sesión es incorrecto.
	*	El sistema muestra un error en pantalla.

Tabla 2.26. Escenario UC-01

UC-02	**Dar me gusta**	
Descripción	El sistema permite al usuario marcar una foto como "me gusta".	
Precondición	El usuario se encuentra en la pantalla de "Ver detalle foto".	
Secuencia	**Pasos**	**Acción**
	1	El usuario pulsa el botón de "me gusta" (corazón).
	2	El sistema registra el "me gusta" para esa foto por parte del usuario.
	3	El sistema actualiza visualmente el botón de "me gusta" para indicar que la foto ahora tiene "me gusta”.
Postcondición	La foto queda marcada con "me gusta" por el usuario y será visible en la pantalla "Likes".	
Excepciones	**Pasos**	**Acción**
	2	Hay un error al registrar el "me gusta"
	*	El estado del botón no cambia.

Tabla 2.27. Escenario UC-02

UC-03	**Quitar me gusta**	
Descripción	El sistema permite al usuario quitar el "me gusta" de una foto.	
Precondición	El usuario se encuentra en la pantalla de "Ver detalle foto".	
Secuencia	**Pasos**	**Acción**
	1	El usuario pulsa el botón de "me gusta" (corazón).

Postcondición	2	El sistema elimina el "me gusta" para esa foto por parte del usuario.
	3	El sistema actualiza visualmente el botón de "me gusta" para indicar que la foto ya no tiene "me gusta".
	La foto ya no está marcada con "me gusta" por el usuario y no será visible en la pantalla "Likes"	
Excepciones	**Pasos**	**Acción**
	2	Hay un error al registrar el "me gusta"
	*	El estado del botón no cambia.

Tabla 2.28. Escenario UC-03

UC-04	**Guardar foto en lista**	
Descripción	El sistema permite al usuario guardar una foto en una de sus listas existentes.	
Precondición	El usuario ha pulsado el botón de save dentro de la pantalla de detalle de una foto	
Secuencia	**Pasos**	**Acción**
	1	El usuario pulsa el botón "Save".
	2	El sistema muestra una pantalla con las listas existentes del usuario.
	3	El usuario selecciona una lista existente.
	4	El sistema añade la foto a la lista seleccionada.
Postcondición	La foto está guardada en la lista seleccionada por el usuario.	
Excepciones	**Pasos**	**Acción**
	2	El usuario no tiene listas creadas.
	*	Solo se mostrará la opción de “Añadir nueva lista".
	4	Hay un error al guardar la foto en la lista.
	*	El sistema muestra un mensaje de error.

Tabla 2.29. Escenario UC-04

UC-05	**Crear una lista**	
Descripción	El sistema permite al usuario crear una nueva lista de fotos.	
Precondición	El usuario está en el proceso de guardar una foto en una lista y ha pulsado la opción "Añadir nueva lista”.	
Secuencia	**Pasos**	**Acción**
	1	El usuario pulsa el botón "Añadir nueva lista".
	2	El sistema muestra un alert en la pantalla para añadir un nombre a la lista.
	3	El usuario introduce el nombre de la lista.

<table>
<tr><td rowspan="4">Postcondición</td><td>4</td><td>El usuario confirma la creación.</td></tr>
<tr><td>5</td><td>El sistema crea la nueva lista.</td></tr>
<tr><td colspan="2">Se ha creado una nueva lista vacía. La nueva lista es visible en la pantalla "Lists".</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td rowspan="3">Excepciones</td><td>Pasos</td><td>Acción</td></tr>
<tr><td>5</td><td>Hay un error al crear la lista</td></tr>
<tr><td>*</td><td>El sistema muestra un mensaje de error.</td></tr>
</table>

Tabla 2.30. Escenario UC-05

<table>
<tr><td>UC-06, UC-07, UC-08, UC-09, UC-10</td><td colspan="2">Subir foto propia</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema permite al usuario subir una foto propia a la aplicación.</td></tr>
<tr><td>Precondición</td><td colspan="2">El usuario está en cualquier pantalla con el tabbar visible.</td></tr>
<tr><td rowspan="13">Secuencia

Postcondición</td><td>Pasos</td><td>Acción</td></tr>
<tr><td>1</td><td>El usuario pulsa el botón "Plus" en el tabbar.</td></tr>
<tr><td>2</td><td>El sistema invoca el caso de uso "Seleccionar foto" (CUC-07).</td></tr>
<tr><td>3</td><td>El sistema presenta al usuario las opciones para seleccionar una foto: "Seleccionar desde cámara" (CUC-08) o "Seleccionar desde galería" (CUC-09).</td></tr>
<tr><td>4</td><td>El usuario elige una opción.</td></tr>
<tr><td>5</td><td>Una vez la foto es seleccionada, el sistema navega a la pantalla de formulario.</td></tr>
<tr><td>6</td><td>El sistema invoca el caso de uso "Rellenar datos de foto" (CUC-10)</td></tr>
<tr><td>7</td><td>El usuario rellena opcionalmente los datos de usuario y descripción.</td></tr>
<tr><td>8</td><td>El usuario pulsa el botón de "Subir".</td></tr>
<tr><td>9</td><td>El sistema sube la foto con su nombre de usuario y descripción.</td></tr>
<tr><td>10</td><td>El sistema redirige al usuario a la pantalla "MyPhotos"</td></tr>
<tr><td colspan="2">Se ha subido la foto del usuario y es visible en la pantalla "MyPhotos".</td></tr>
<tr><td rowspan="5">Excepciones</td><td>Pasos</td><td>Acción</td></tr>
<tr><td>4</td><td>El usuario cancela la selección de foto.</td></tr>
<tr><td>*</td><td>El proceso de subida se interrumpe.</td></tr>
<tr><td>9</td><td>Hay un error al subir la foto.</td></tr>
<tr><td>*</td><td>El sistema muestra un mensaje de error.</td></tr>
</table>

Tabla 2.31. Escenario UC-06, UC-07, UC-08, UC-09, UC-10

<table>
<tr><td>UC-11</td><td colspan="2">Cambiar modo de visualización</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema permite al usuario cambiar el modo de visualización de las fotos.</td></tr>
<tr><td>Precondición</td><td colspan="2">El usuario debe estar en la pantalla de ProfileScreen</td></tr>
<tr><td>Secuencia</td><td colspan="2"><table>
<tr><th>Pasos</th><th>Acción</th></tr>
<tr><td>1</td><td>El usuario pulsa el botón con símbolo de setings</td></tr>
<tr><td>2</td><td>Se abre un menú con las opciones “Modo de visualización” e “Idioma”</td></tr>
<tr><td>3</td><td>El usuario pulsa en “Modo de visualización”</td></tr>
<tr><td>4</td><td>Se abren dos nuevas opciones, “Mosaico” o “Lista”</td></tr>
<tr><td>5</td><td>El usuario pulsa en el modo que prefiera</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td colspan="2">Se ha cambiado el modo de visualización de fotos de la aplicación</td></tr>
<tr><td>Excepciones</td><td colspan="2"><table>
<tr><th>Pasos</th><th>Acción</th></tr>
<tr><td>5</td><td>Hay un error al cambiar de modo</td></tr>
<tr><td>*</td><td>El sistema muestra un mensaje de error.</td></tr>
</table></td></tr>
</table>

Tabla 2.32. Escenario UC-11

<table>
<tr><td>UC-12</td><td colspan="2">Cambiar el idioma</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema permite al usuario cambiar el idioma de la aplicación.</td></tr>
<tr><td>Precondición</td><td colspan="2">El usuario debe estar en la pantalla de ProfileScreen</td></tr>
<tr><td>Secuencia</td><td colspan="2"><table>
<tr><th>Pasos</th><th>Acción</th></tr>
<tr><td>1</td><td>El usuario pulsa el botón con símbolo de settings</td></tr>
<tr><td>2</td><td>Se abre un menú con las opciones “Modo de visualización” e “Idioma”</td></tr>
<tr><td>3</td><td>El usuario pulsa en “Idioma”</td></tr>
<tr><td>4</td><td>Se abren tres nuevas opciones, “Español”, Francés e “Inglés”</td></tr>
<tr><td>5</td><td>El usuario pulsa en el idioma que prefiera</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td colspan="2">Se ha cambiado el idioma de la aplicación</td></tr>
<tr><td>Excepciones</td><td colspan="2"><table>
<tr><th>Pasos</th><th>Acción</th></tr>
<tr><td>5</td><td>Hay un error al cambiar de idioma.</td></tr>
<tr><td>*</td><td>El sistema muestra un mensaje de error.</td></tr>
</table></td></tr>
</table>

Tabla 2.33. Escenario UC-12

2.5. Manual de usuario

Para el manual de usuario se hará uso de las imágenes provistas en la sección de diseño de la aplicación. Estas figuras se referenciarán y se detallarán los pasos a seguir en cada uno de los escenarios a describir. Las instrucciones de uso del sistema se detallan a continuación.

2.5.1. Acceso a la aplicación

Para acceder a la aplicación Pixora, será necesario instalarla desde la versión beta de Google Play. Para ello, accedemos al siguiente enlace desde un dispositivo Android:

Enlace a Pixora: https://play.google.com/store/apps/details?id=com.ibrepidevs.pixora

Este enlace corresponde a una prueba cerrada - Alpha de la aplicación por lo que solo se podrás acceder a ella a través de las cuentas de Google que hayan sido añadidas a la lista de testers internos(en este caso los correos de los profesores de la asignatura).

Una vez instalada, al abrir la aplicación, veremos la SplashScreen (Figura 2.2).

2.5.2. Navegación principal

Al iniciar Pixora, la primera vista que aparece es la **pantalla de Home** (Figura 2.3), donde se muestran las fotos más populares de la comunidad. En esta pantalla podemos aplicar filtros por categorías como *People, Nature, Travel*, entre otras. Las fotos se presentan en un grid vertical de tipo Mansonry, y al pulsar sobre cualquiera de ellas, se accede a su detalle.

n la parte inferior de la pantalla encontraremos la barra de navegación (tabbar), desde la cual se puede acceder a las siguientes secciones:

- **Home**: Vista inicial donde se muestran las fotos filtradas por categoría (Figura 2.3).
- **Search**: Pantalla que dispone de una barra de búsqueda de imágenes y un scroll infinito (Figura 2.5).
- **Plus (+)**: Permite subir una foto propia desde cámara o galería (Figura 2.9 y Figura 2.10).
- **Activity**: Muestra las actividades recientes del usuario y sus interacciones (Figura 2.6).
- **Profile**: Desde donde se puede acceder a fotos con me gusta, listas creadas y fotos subidas (Figura 2.7, Figura 2.11 y Figura 2.13).

2.5.3. Buscar contenido

En la pantalla de Search (Figura 2.5), se encuentra una barra en la parte superior que permite escribir términos para buscar imágenes. A medida que el usuario desplaza hacia abajo, el sistema carga más imágenes automáticamente (scroll infinito).

2.5.4. Ver actividad

En la pantalla de Activity (Figura 2.6), el usuario puede consultar la actividad relacionada con su cuenta como las fotos a las que le ha dado me gusta o las fotos que ha añadido a una lista y a qué lista.

2.5.5. Subir una foto

Para añadir contenido propio, el usuario debe pulsar el icono de Plus (+) en la barra de navegación. Esto abrirá una hoja inferior (Figura 2.9) donde se podrá elegir entre:

- Seleccionar desde cámara
- Seleccionar desde galería

Una vez seleccionada la imagen, el sistema lleva al usuario a la pantalla de formulario (Figura 2.10), donde puede, de manera opcional, completar los siguientes datos:

- Nombre de usuario
- Descripción de la foto

Finalmente, al pulsar en “Subir”, la imagen se publica y aparece en su sección de fotos propias.

2.5.6. Ver perfil y secciones personales

Desde la pestaña de Profile se accede a tres secciones principales:

- **Likes**: Muestra todas las fotos a las que el usuario ha dado me gusta (Figura 2.13
- **Lists**: Listas personalizadas del usuario donde ha agrupado fotos (Figura 2.7).
- **MyPhotos**: Fotos subidas por el propio usuario (Figura 2.11).

En todas estas vistas, es posible pulsar sobre una foto para acceder a su detalle.

2.5.7. Ver detalles de una foto

Al pulsar sobre cualquier foto desde cualquier sección, se accede a la pantalla de detalle de la foto (Figura 2.4). En esta vista el usuario puede:

- Dar me gusta pulsando el icono del corazón o haciendo doble click en la foto.
- Quitar el me gusta si ya se había marcado.
- Guardar la foto en una lista pulsando el botón “Save”.

Si se desea guardar la foto, se abre la pantalla de selección de listas (Figura 2.12), donde se puede:

- Elegir una lista ya existente.
- Crear una nueva lista pulsando “Añadir nueva lista”.

2.5.8. Ver detalle de una lista

Desde la sección de Lists, el usuario puede pulsar sobre cualquier lista para acceder a su vista detallada (Figura 2.8), donde se muestran todas las fotos que pertenecen a esa colección. Desde aquí también se puede acceder al detalle individual de cada imagen.

3. Documentación técnica

Esta especificación técnica se centra en los detalles técnicos, como los requerimientos de hardware y software, la arquitectura de datos y los lenguajes de programación utilizados.

3.1. Arquitectura del proyecto

Para el desarrollo de esta aplicación, he optado por implementar una arquitectura que considero robusta y adecuada, combinando dos patrones arquitectónicos ampliamente reconocidos: MVVM (Model-View-ViewModel) y Clean Architecture. Elegí esta combinación porque, tras investigar y evaluar distintas opciones, vi que se complementan muy bien y aportan mejoras significativas en la organización y calidad del código, facilitando el desarrollo, la prueba y el futuro mantenimiento del proyecto.

3.1.1. MVVM (Model-View-ViewModel)

MVVM es un patrón arquitectónico de presentación que busca separar la lógica de la interfaz de usuario (la Vista) de la lógica de negocio (el Modelo). Introduce una capa intermedia, el ViewModel, que gestiona el estado de la Vista y expone los datos y acciones que la Vista necesita, actuando como un puente entre la UI y el Modelo.

Elegí MVVM en lugar de otros patrones porque consideré que es muy adecuado para este proyecto, especialmente al trabajar con Jetpack Compose, debido a su naturaleza reactiva. El uso de MVVM permite:

- Una clara separación entre la lógica de cómo se presentan los datos (en el ViewModel) y la lógica puramente visual (en la Vista).
- Mejorar la testabilidad, ya que la lógica de presentación en el ViewModel puede probarse fácilmente sin depender de la UI.
- Aprovechar eficientemente el sistema de enlace de datos de Jetpack Compose, lo que simplifica la actualización automática de la interfaz.

3.1.2. Clean Architecture

Además de MVVM para la gestión de la UI, he estructurado el proyecto siguiendo los principios de Clean Architecture. Esta arquitectura organiza el código en capas, asegurando que el flujo de las dependencias siempre vaya hacia el interior, aislando la lógica de negocio central de los detalles externos como la UI, bases de datos o APIs. He aplicado esta arquitectura dividiendo el proyecto en las siguientes capas principales:

1. **Capa de Dominio (Domain Layer)**. Esta es la capa más interna y fundamental. Lo crucial es que esta capa es totalmente independiente, no

sabe nada sobre la interfaz de usuario, bases de datos, o frameworks externos. Contiene:

- **Entidades (Entities)**: Objetos de negocio con sus reglas fundamentales.
- **Casos de Uso (Use Cases)**: Definen las operaciones específicas de la aplicación, orquestando la lógica y utilizando las interfaces de repositorio.
- **Interfaces de Repositorio (Repository Interfaces)**: Contratos que definen cómo se accede a los datos, sin especificar la implementación.

2. **Capa de Datos (Data Layer)**. Rodeando la Capa de Dominio, esta capa es responsable de la implementación concreta de las interfaces de repositorio definidas en la capa de Dominio. Gestiona la comunicación con fuentes de datos externas y la persistencia de datos. Sus componentes principales son:

   - **Implementaciones de Repositorio (Repository Implementations)**: Implementan las interfaces de repositorio del Dominio. Interactúan directamente con las fuentes de datos, ya sean remotas o locales.
   - **Mapeadores (Mappers)**: Objetos responsables de transformar los modelos de datos de las fuentes externas en las Entidades del Dominio, y viceversa.

3. **Capa de Presentación (Presentation Layer)**. Esta es la capa más externa y es responsable de la interfaz de usuario y la interacción con el usuario. Incluye:

   - **Vistas (Views - Compose)**: Componentes de la UI que muestran los datos al usuario y capturan sus entradas.
   - **ViewModels**: Actúan como intermediarios entre las Vistas y los Casos de Uso del Dominio. Preparan los datos para ser mostrados por las Vistas y reaccionan a las acciones del usuario invocando los Casos de Uso apropiados. Los ViewModels dependen de la capa de Dominio (específicamente de los Casos de Uso) para obtener datos y ejecutar lógica de negocio, pero no conocen la UI directamente, sino que exponen datos y estados que la Vista puede observar.

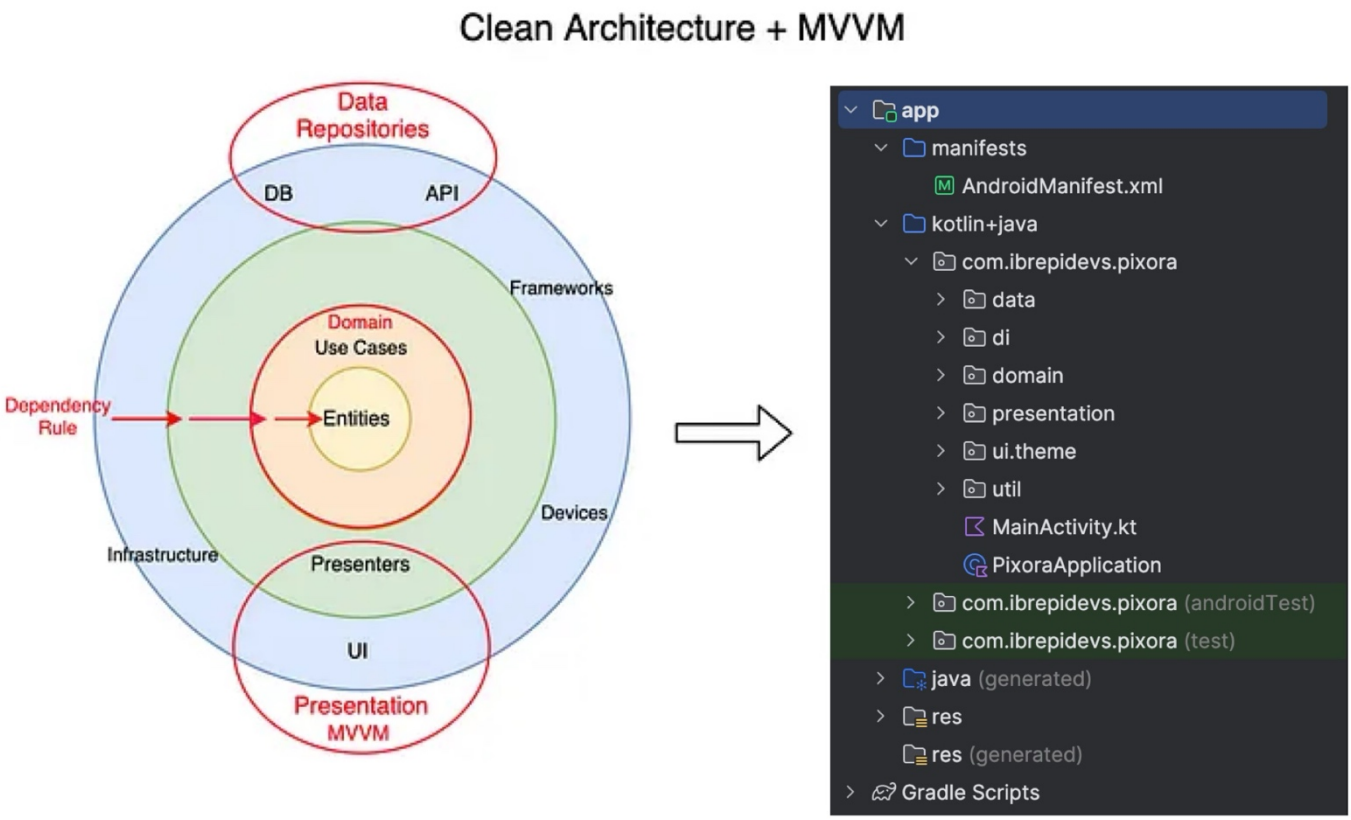

Figura 3.1: Clean Architecture + MVVM

3.1.3. Elección de Jetpack Compose

La elección de **Jetpack Compose** para construir la interfaz de usuario de Pixora fue debido a que Compose es el toolkit de UI moderno y nativo de Google. Las razones que me han motivado el desarrollado de la aplicación con esta herramienta son:

- **Modelo de UI declarativo:** A diferencia del enfoque imperativo de XML, donde se modifica manualmente la jerarquía de vistas, con Compose simplemente se describe cómo debe ser la UI para un estado determinado. Cuando el estado de la aplicación cambia (por ejemplo, al recibir nuevos datos del ViewModel), el framework se encarga de actualizar la interfaz de forma automática y eficiente. Lo que reduce el código repetitivo y elimina una fuente común de errores relacionados con la sincronización del estado.
- **Código más conciso e intuitivo:** El desarrollo de la UI se realiza íntegramente en Kotlin, lo que permite aprovechar toda la potencia del lenguaje. Esto se traduce en menos código, una mayor reutilización a través de funciones @Composable personalizadas y una estructura más fácil de leer y mantener en comparación con la gestión de múltiples archivos XML y el código de enlace correspondiente.
- **Aceleración del desarrollo:** Compose ofrece vistas previas en tiempo real y funcionalidades como *Live Edit*, que permiten ver los cambios al instante sin

necesidad de volver a compilar toda la aplicación. Esto agiliza enormemente el proceso de iteración y diseño de la interfaz.
- **Sinergia con la arquitectura elegida:** La naturaleza reactiva de Compose se integra a la perfección con los componentes de Jetpack y los patrones modernos utilizados en el proyecto, como el uso de StateFlow para exponer el estado desde los ViewModels y el flujo de datos unidireccional promovido por MVVM.

3.2. Estructura del proyecto

La organización del código fuente de la aplicación sigue una estructura de carpetas que refleja la arquitectura definida en el apartado anterior, facilitando la localización de los distintos componentes y la separación de responsabilidades entre capas. A continuación, se describe la organización principal del proyecto:

1. **data**: Este directorio corresponde a la Capa de Data (Data Layer) de la arquitectura Clean.
    a. **local:** Implementaciones para acceder a datos desde una fuente local, en este caso, la base de datos **Room**.
        - **db:** Contiene la definición de la base de datos con Room, incluyendo los DAOs (Data Access Objects) y la clase principal PixoraDataBase.kt
        - **model:** Definición de las entidades específicas para la base de datos Room (PhotoEntity, UserActivityEntity, etc.).
    b. **mappers**: Lógica para transformar los objetos de datos específicos de la Capa de Data (ej: DTOs de API, objetos de Core Data) a las entidades puras del Dominio y viceversa.
    c. **network**: Archivos y clases relacionados específicamente con la capa de red.
        - **api:** Contiene la definición de la API con Retrofit, constantes (ApiConstants) y modelos de respuesta (BaseResponse).
    d. **repository**: Implementaciones concretas de las interfaces de repositorio definidas en la Capa de Dominio.
2. **di**: Contiene el código relacionado con la Inyección de Dependencias (Hilt), organizado en módulos para proveer las dependencias de la capa de datos.
3. **domain**: Este directorio corresponde a la Capa de Dominio (Domain Layer).
    a. **entity**: Definición de las estructuras de datos puras que representan los conceptos fundamentales de la aplicación.
    b. **repository:** Contiene las definiciones de protocolos (interfaces) que especifican cómo la Capa de Dominio necesita interactuar con las

fuentes de datos. Las implementaciones de estos protocolos se encuentran en la Capa de Data.

c. **usecases:** Contiene las clases que representan las operaciones específicas de la lógica de negocio (los casos de uso).

4. **presentation**: Este directorio corresponde a la **Capa de Presentación** (Presentation Layer). Contiene los Composables (Vistas) y los ViewModels que interactúan con los Casos de Uso del Dominio.
    a. **navigation:** Define la estructura de navegación de la aplicación usando Jetpack Navigation Compose, incluyendo el grafo y las rutas.
    b. **screens:** Contiene los diferentes @Composable que representan una pantalla completa de la aplicación. También contiene los view models correspondientes a cada pantalla.
    c. **widgets:** Agrupa componentes de UI reutilizables que se usan en diferentes pantallas (o podrían usarse).
5. **theme:** Define el sistema de diseño de la aplicación para Jetpack Compose, incluyendo colores, tipografía y formas, siguiendo los principios de Material Design.
6. **util:** Carpeta destinada a clases, extensiones, funciones o utilidades generales que son usadas transversalmente en el proyecto y no pertenecen específicamente a una capa arquitectónica concreta.
7. **MainActivity.kt:** El archivo principal que define la Activity que aloja el contenido de Jetpack Compose y sirve como punto de entrada de la UI.
8. **PixoraApplication.kt:** Clase Application personalizada, utilizada para inicializar librerías globales al inicio de la aplicación, como la inyección de dependencias.

3.3. Composables principales

A continuación, se describen los Composables que actúan como las pantallas principales de la aplicación, detallando su responsabilidad técnica dentro de la capa de presentación.

- **SplashScreen**: Muestra una pantalla de bienvenida con una animación Lottie durante 2.5 segundos. Una vez finalizado el tiempo, navega automáticamente a la pantalla principal de la aplicación.
- **MainScreen**: Actúa como el contenedor principal y el "andamio" (Scaffold) de la aplicación. Alberga el grafo de navegación anidado y la barra de navegación inferior, y centraliza la lógica de la interfaz de usuario global, como la gestión de permisos, la subida de fotos y la visualización de componentes como el ModalBottomSheet.

- **HomeScreen**: Presenta la pantalla de inicio, donde el usuario puede explorar un feed de imágenes. Permite filtrar el contenido por categorías y muestra las fotos en formato de mosaico o lista, según la preferencia del usuario.
- **SearchScreen**: Implementa la funcionalidad de búsqueda, permitiendo al usuario introducir texto para encontrar imágenes. Gestiona la paginación automática (scroll infinito) y optimiza las llamadas a la API mediante un debounce para no realizar búsquedas con cada tecla pulsada.
- **ActivityScreen**: Muestra el historial de actividad del usuario, como los "me gusta" dados o las fotos añadidas a listas. Organiza el contenido en un HorizontalPager para una navegación cómoda por páginas.
- **ProfileScreen**: Funciona como la pantalla de perfil del usuario, utilizando un TabRow para organizar y dar acceso a las diferentes secciones personales: mis fotos, mis listas y mis favoritos. Además, integra el menú de ajustes de la aplicación.
- **MyPhotosScreen**: Muestra la galería de todas las fotos que el usuario ha subido a la aplicación, funcionando como su portafolio personal.
- **MyListsScreen**: Expone todas las colecciones o listas de fotos creadas por el usuario en una cuadrícula. Cada tarjeta (PhotoListCard) presenta una previsualización visual de la lista con algunas de sus imágenes.
- **FavoritesScreen**: Presenta una galería con todas las imágenes a las que el usuario ha dado "me gusta", permitiendo ver todo su contenido favorito en un único lugar.
- **PhotoDetailsScreen**: Muestra una foto en detalle junto con su información (descripción, autor). Es la pantalla principal de interacción, donde el usuario puede dar "me gusta" (con un gesto de doble toque), guardar la foto en una lista y ver los datos del fotógrafo.
- **PhotoListDetailsScreen**: Presenta el contenido de una lista de fotos específica que el usuario ha seleccionado previamente. Muestra todas las imágenes de esa colección en formato de galería.
- **PostPhotoScreen**: Es el formulario final para subir una nueva foto. Muestra la imagen seleccionada y permite al usuario añadir datos opcionales como un nombre de usuario y una descripción antes de publicarla.

3.4. Gestión de datos

Para la gestión y persistencia de datos se ha optado por utilizar **Room** como sistema principal de persistencia de datos en local. Room es una librería de abstracción sobre SQLite que permite gestionar bases de datos locales de manera

eficiente, segura y con una interfaz más sencilla y robusta. Su integración con el ciclo de vida de Android y el soporte de consultas SQL tipadas la convierten en una opción adecuada para aplicaciones que requieren persistencia estructurada de datos sin necesidad de conectividad remota.

Adicionalmente, se ha empleado **Preferences DataStore** para almacenar de forma persistente las preferencias del usuario, como el modo de visualización (mosaico o lista) y el idioma seleccionado (español, francés o inglés). Preferences DataStore es una alternativa moderna y segura a SharedPreferences, diseñada para manejar configuraciones simples y datos clave-valor.

No se ha considerado el uso de frameworks como Firebase, ya que la aplicación no requiere funcionalidades en la nube, sincronización remota ni almacenamiento en tiempo real. Dado que todos los datos pueden mantenerse y gestionarse de forma local, el uso de Room y DataStore se considera más apropiado para los objetivos y alcance de este proyecto. Si se quisiera ampliar el alcance del proyecto hasta necesitar relaciones entre usuarios, etc… se podría optar por Firebase por ejemplo.

3.4.1. Modelo conceptual

Para complementar la explicación sobre la gestión de datos locales con ROOM, el siguiente diagrama conceptual ilustra la estructura del modelo de datos utilizado en la aplicación. Este diagrama muestra las entidades principales, sus atributos clave y las relaciones que existen entre ellas, proporcionando una vista general de cómo se organiza la información persistida localmente.

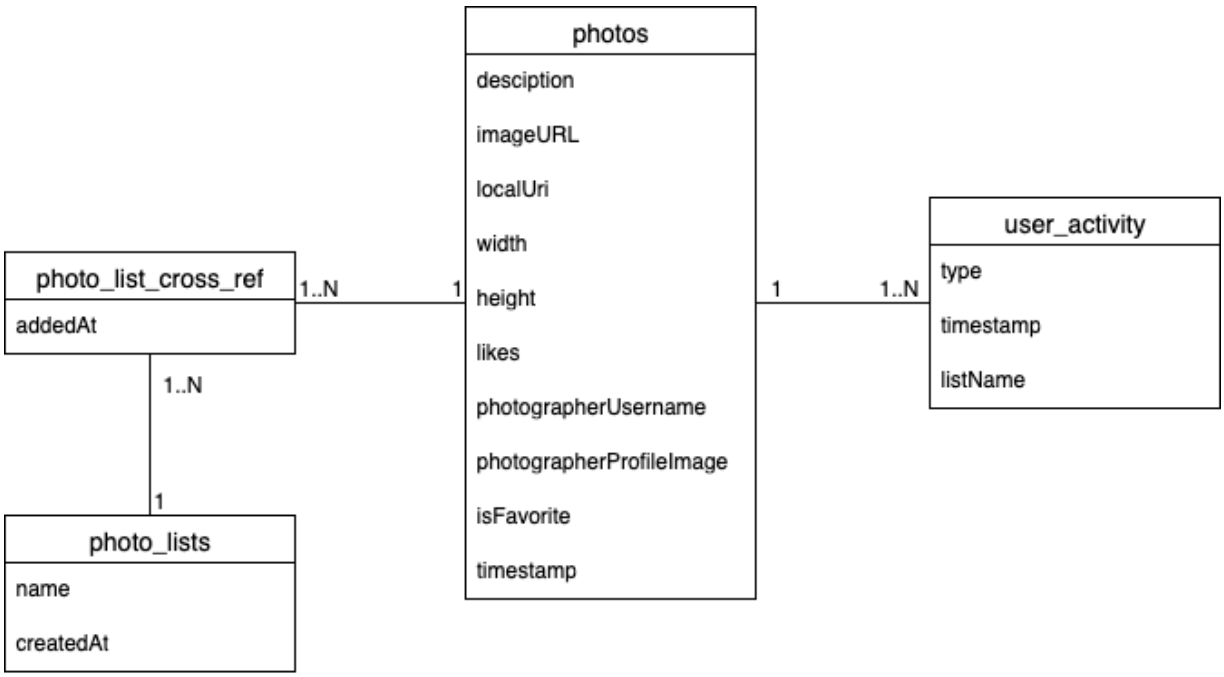

Figura 3.2: Modelo conceptual

Nombre	Descripción	Tipo	Entidad
description	Descripción de la foto	String	photos
imageURL	URL remota de la imagen	String	photos
localUri	URI local de la imagen	String	photos
width	Ancho de la imagen	Integer	photos
height	Alto de la imagen	Integer	photos
likes	Número de likes que tiene la foto	Integer	photos
photographer Username	Nombre de usuario del fotógrafo	String	photos
photographer ProfileImage	Imagen de perfil del fotógrafo	String	photos
isFavorite	Indica si la foto ha sido marcada como favorita	Boolean	photos
timestamp	Fecha y hora en que se guardó la foto	Long	photos
name	Nombre de la lista de fotos	String	photo_lists
createdAt	Fecha de creación de la lista	Long	photo_lists
addedAt	Fecha en la que se añadió la foto a la lista	Long	photo_list_cross_ref
type	Tipo de actividad realizada (añadir a lista o marcar favorito)	String	user_activity
timestamp	Fecha y hora en la que se produce la actividad	Long	user_activity
listName	Nombre de la lista relacionada con la actividad (opcional)	String	user_activity

Tabla 3.1: Descripción de atributos del modelo conceptual de datos

3.4.2. Modelo relacional

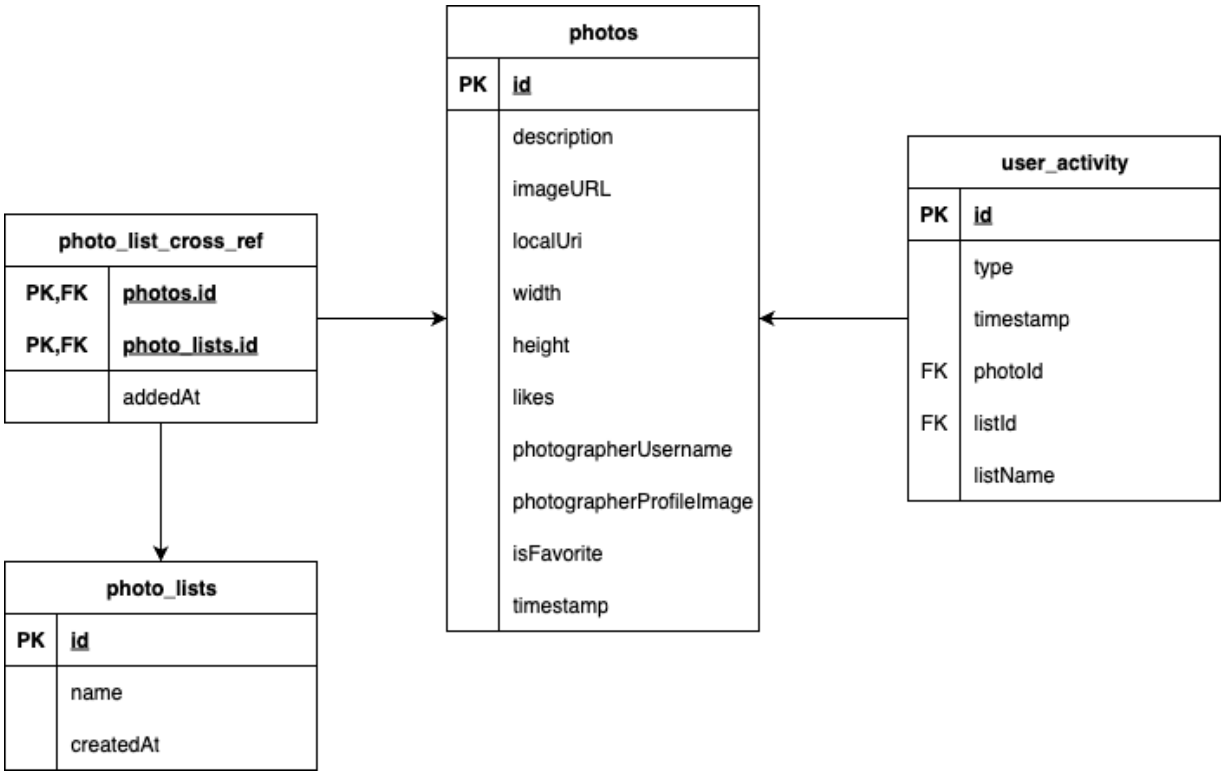

Figura 3.3: Modelo relacional

3.5. Integraciones con APIs / Servicios externos

La aplicación integra servicios externos. Concretamente, se conecta a la **API de Unsplash** para acceder a un amplio catálogo de imágenes de alta calidad, que constituyen el contenido principal presentado al usuario.

La gestión de las llamadas HTTP a estos servicios se implementa dentro de la **Capa de Data** de la arquitectura Clean, concretamente en la subcapa encargada de las **fuentes de datos remotas**. Para realizar las peticiones de red, en este caso se utiliza **Retrofit**, una librería ampliamente adoptada en el ecosistema Android, combinada con **OkHttp** para la gestión avanzada de interceptores y logs de red. Esta elección se debe a dos motivos principales: por un lado, Retrofit simplifica la definición de interfaces de API de forma declarativa y reduce el código boilerplate; por otro lado, permite mantener un control granular sobre la configuración de la conexión y el tratamiento de respuestas, gracias a la integración con OkHttp y la posibilidad de añadir interceptores personalizados.

3.5.1. Unsplash API

Como mencioné anteriormente, la aplicación Pixora hace uso de servicios externos para la obtención de contenido visual. En concreto, se ha integrado la API de Unsplash, una de las plataformas de imágenes más reconocidas y utilizadas a nivel mundial. Esta API permite acceder a una amplia colección de fotografías de alta calidad de forma gratuita, bajo una licencia abierta.

Unsplash es una plataforma en línea lanzada en 2013 que ofrece fotografías de alta resolución de manera gratuita, contribuidas por una comunidad global de fotógrafos. Su objetivo es democratizar el acceso a imágenes de calidad, permitiendo su uso sin necesidad de licencias comerciales. En 2017, Unsplash lanzó su propia API pública con el fin de permitir a desarrolladores y empresas integrar estas imágenes en sus aplicaciones, productos o servicios.

3.5.1.1. Tipo de peticiones y funcionamiento general

La API de Unsplash es RESTful, por lo tanto, utiliza métodos HTTP estándar, principalmente:

- ***GET*** para obtener listas de imágenes, realizar búsquedas o consultar detalles de una foto.

Algunas de las rutas más comúnmente utilizadas incluyen:

- ***GET*** */photos* – Lista de fotos generales.
- ***GET*** */search/photos* – Buscar fotos por palabra clave.
- ***GET*** */photos/:id* – Obtener detalles de una foto específica.
- ***GET*** */topics* – Listar temas populares y categorías.

Además, cada petición requiere autenticación mediante una Access Key generada al registrar la aplicación en el portal de desarrolladores de Unsplash.

3.5.1.2. Uso de la API en Pixora

En el contexto de Pixora, la API de Unsplash se utiliza como fuente principal de imágenes para alimentar distintas secciones de la aplicación:

- **Pantalla de inicio (Home)**: donde se muestran imágenes categorizadas por temas como "popular", "nature", "travel", etc.
- **Pantalla de búsqueda (Search)**: que permite a los usuarios buscar imágenes usando texto libre.
- **Pantalla de detalles**: donde se accede a información más específica de cada imagen mostrada.

3.5.1.3. Limites y restricciones

Por defecto, la API de Unsplash impone un límite de 50 peticiones por hora para aplicaciones que se encuentren en modo de desarrollo o no cumplan ciertos criterios

de producción. Estos límites están diseñados para garantizar un uso responsable de los recursos de la plataforma.

Si una aplicación en producción cumple con una serie de requisitos establecidos por Unsplash, puede aumentar su cuota hasta 2000 peticiones por hora. Para ello, la aplicación debe ofrecer un valor añadido y no reproducir la misma experiencia que el sitio oficial de Unsplash. Es decir, no debe tener como funcionalidad principal la simple visualización de imágenes, sino que debe integrar estas fotos como parte de un flujo de valor más amplio.

En el caso de Pixora, debido a que su experiencia de usuario se basa principalmente en mostrar, buscar y gestionar imágenes de forma similar a Unsplash, no es elegible para la ampliación del límite de peticiones.

3.6. Sistema de temas (Modo Claro y Oscuro)

La aplicación implementa soporte completo para los modos claro y oscuro.

Aprovechando el sistema de temas de Material Design 3 y Jetpack Compose, se han definido dos esquemas de color (lightColorScheme y darkColorScheme) en el fichero Theme.kt. La aplicación escucha automáticamente la configuración del sistema operativo del usuario y aplica el tema correspondiente sin necesidad de ninguna acción manual.

Figura 3.4: Ejemplo modo claro

Figura 3.5: Ejemplo modo oscuro

3.7. Gestión de notificaciones

La aplicación implementa un sistema de **notificaciones locales** diseñado de manera proactiva para cumplir con los requisitos del proyecto y mejorar la retención de usuarios. Dado que la aplicación no cuenta con un backend propio para gestionar interacciones en tiempo real (como nuevos seguidores o comentarios), lo que se me ocurrió es crear una notificación de ejemplo que salte cuando estés 5 segundos fuera de la aplicación

3.7.1. Implementación y Lógica de Disparo

El mecanismo de disparo es una solución ligada al ciclo de vida de la MainActivity para simular actividad y recordar al usuario que vuelva a la aplicación.

- **Lógica de Disparo**: Cuando el usuario sale de la aplicación (se invoca el método onStop), se inicia una corrutina que programa el envío de una notificación tras un breve retardo de 5 segundos. Si el usuario regresa a la aplicación antes de que finalice ese tiempo (se invoca onStart), la corrutina y la notificación pendiente se cancelan. Este enfoque asegura que la notificación solo se envíe cuando el usuario realmente ha dejado la aplicación, actuando como un recordatorio para volver a explorar contenido.
- **Centralización del Código**: Toda la lógica para construir y mostrar las notificaciones está encapsulada en el objeto NotificationHelper, manteniendo el código limpio y fácil de mantener. La inicialización del canal de notificación se realiza de forma segura en la clase PixoraApplication, asegurando que esté disponible desde el momento en que se lanza la app.

3.7.2. Compatibilidad

Para garantizar una experiencia consistente y funcional desde la minSdk 24 en adelante, la implementación sigue las mejores prácticas de compatibilidad de Android:

- **NotificationCompat.Builder**: Se utiliza esta clase de la librería AndroidX, que gestiona automáticamente las diferencias entre versiones del sistema operativo, asegurando que la notificación se muestre correctamente en todos los dispositivos.
- **Canales de Notificación**: El código comprueba la versión del SDK y crea un NotificationChannel solo en Android 8.0 (API 26) o superior, cumpliendo con los requisitos de las versiones más modernas sin generar errores en las más antiguas.
- **Permiso en Tiempo de Ejecución**: Se incluye la verificación del permiso POST_NOTIFICATIONS, necesario para Android 13 (API 33) y superior, garantizando el funcionamiento correcto en los dispositivos más recientes.

3.8. Gestión de variantes con Gradle

Para el ciclo de desarrollo y facilitar la gestión de diferentes entornos, se ha creado la siguiente matriz de variantes a partir de dos dimensiones:

- **Build Types:**
    - **debug**: Versión utilizada durante el desarrollo. Está configurada para permitir la depuración de la aplicación.
    - **release**: Versión preparada para la publicación. Utiliza la configuración de firma (signingConfig) para generar un artefacto listo para ser distribuido en Google Play.
- **Product Flavors:**
    - **dev (Development)**: Flavor configurado para el desarrollo. Añade el sufijo .dev al applicationId, lo que permite tener la app de desarrollo instalada en un dispositivo junto a la de producción. También añade el sufijo -dev al nombre de la versión para una fácil identificación.
    - **pro (Production)**: Flavor que corresponde a la versión final de la aplicación, con el applicationId definitivo.

Ambos flavors utilizan buildConfigField para gestionar de forma segura las claves de la API de Unsplash y otras variables de entorno.

3.8.1. Filtrado de variantes

La combinación de 2 build types y 2 flavors genera 4 variantes (devDebug, devRelease, proDebug, proRelease). Para optimizar el proceso, se ha aplicado un filtro de variantes en el fichero build.gradle para deshabilitar las combinaciones que no son necesarias:

- **devRelease**: Una versión de desarrollo firmada para producción no tiene utilidad.
- **proDebug**: Una versión de producción no debería ser depurable.

Gracias a este filtro, el ciclo de trabajo se centra exclusivamente en las dos variantes útiles: **devDebug** para el desarrollo diario y **proRelease** para la generación del APK/AAB final que se sube a Google Play.

3.9. Control de Versiones

Desde el comienzo del proyecto se hace uso de Github para llevar un seguimiento claro de los cambios en el código que experimentan las aplicaciones. El repositorio Git utilizado para este proyecto puede consultarse en

https://github.com/israelbrea12/Pixora-Android.git

3.10. Librerías de terceros

A continuación, se comentan las librerías externas que se han utilizado para tareas específicas dentro de la aplicación:

3.10.1. Networking: Retrofit y OkHttp

Para gestionar la comunicación con servicios externos, como la API de Unsplash, se ha implementado **Retrofit**. Esta librería se ha elegido por ser un cliente HTTP robusto y type-safe que simplifica la conversión de una API REST en una interfaz de Kotlin. Frente a alternativas como Ktor o Volley, Retrofit destaca por su madurez, su extensa comunidad y su facilidad para integrar conversores de datos. En este proyecto, se combina con retrofit.converter.gson para la serialización y deserialización de objetos JSON y con okhttp.logging.interceptor para facilitar la depuración de las peticiones de red durante el desarrollo.

3.10.2. Carga de imágenes: Coil (Coroutine Image Loader)

Siendo Pixora una aplicación centrada en el contenido visual, la gestión de imágenes es un pilar fundamental. Se ha optado por **Coil** en lugar de otras alternativas populares como Glide o Picasso. La razón principal es que Coil es una librería moderna, escrita desde cero en Kotlin y basada en Corrutinas. Esto la hace extremadamente ligera, rápida y eficiente. Además, su integración con Jetpack Compose a través de la dependencia coil-compose es nativa y muy sencilla, permitiendo cargar imágenes directamente en los Composables con un rendimiento optimizado.

3.10.3. Persistencia de datos: Room y DataStore

La persistencia local se resuelve con dos librerías de Jetpack.

- **Room**: Para los datos estructurados (fotos, listas, actividad del usuario), se utiliza Room. Se eligió esta librería por ser la capa de abstracción sobre SQLite recomendada por Google. Ofrece seguridad en tiempo de compilación para las consultas SQL y se integra de forma nativa con los flujos de datos reactivos (Flow), lo cual es clave para la arquitectura MVVM de la aplicación. Se descartó el uso de Firebase porque la aplicación no requiere sincronización en la nube ni funcionalidades en tiempo real para su alcance académico actual. Frente a otros ORMs como Realm, Room fue la opción preferida por su integración total con el ecosistema Jetpack.
- **Preferences DataStore**: Para almacenar datos simples clave-valor, como las preferencias de idioma o el modo de visualización, se usa DataStore. Se eligió como el sustituto moderno de SharedPreferences, ya que resuelve sus principales inconvenientes al ofrecer una API asíncrona basada en Corrutinas

y Flow, lo que evita bloqueos en el hilo principal y garantiza la consistencia de los datos.

3.10.4. Inyección de dependencias: Hilt

Para aplicar correctamente la arquitectura limpia y desacoplar las distintas capas del proyecto, se utiliza **Hilt**. La elección de Hilt sobre otras alternativas como Koin se basó en varios factores clave, principalmente su **seguridad en tiempo de compilación** y su rendimiento.

A diferencia de Koin, que resuelve las dependencias en tiempo de ejecución y puede provocar fallos inesperados en la aplicación (como NoBeanDefFoundException), Hilt valida el grafo de dependencias durante la compilación. Este enfoque proactivo permite detectar y solucionar errores de configuración en la fase de desarrollo, garantizando una mayor robustez y estabilidad en la versión final del producto.

Además, al generar el código necesario en tiempo de compilación, Hilt ofrece un **rendimiento superior en tiempo de ejecución**, evitando la sobrecarga que supone la resolución de dependencias en el arranque de la aplicación. Finalmente, Hilt es la **solución recomendada oficialmente por Google** para la inyección de dependencias en Android, lo que asegura una integración perfecta con los componentes de Jetpack (como ViewModel y Navigation), un mejor soporte de herramientas en Android Studio y un mantenimiento a largo plazo.

3.10.5. Lottie

Con el fin de mejorar la experiencia de usuario se ha incluido la librería **Lottie**. Lottie permite renderizar animaciones de Adobe After Effects exportadas como JSON de forma nativa. Es la opción estándar en la industria para implementar animaciones vectoriales complejas sin el sobrecoste de rendimiento que supondría usar ficheros de vídeo o GIFs. En concreto, esta librería se usa para la animación de la pantalla de carga inicial (Splash Screen) de la aplicación.

3.10.6. Serialización de datos: Kotlinx Serialization

En Pixora, la librería **Kotlinx Serialization** se utiliza de una manera muy específica: para habilitar un sistema de **navegación type-safe** con Jetpack Compose.

Su función es definir las rutas de la aplicación como objetos de Kotlin serializables mediante la anotación @Serializable. Este enfoque moderno permite pasar destinos de navegación, incluso aquellos con argumentos complejos, como objetos completos en lugar de simples cadenas de texto.

3.11. Especificaciones y compatibilidad

En esta sección se detallan las decisiones técnicas relacionadas con la compatibilidad de la aplicación y el uso de APIs específicas del ecosistema Android y Kotlin.

3.11.1. minSdk y compatibilidad

La aplicación establece un minSdkVersion de 24, correspondiente a **Android 7.0 (Nougat)**.

Además de escogerse debido a que los requisitos de la aplicación lo indicaban, al establecer el nivel de API 24, se garantiza la compatibilidad con la gran mayoría de dispositivos Android activos en el mercado actual, al tiempo que se reduce la necesidad de implementar código de retrocompatibilidad complejo, permitiendo un desarrollo más ágil y el uso de APIs más recientes. Y ese ha sido el motivo de su elección.

3.11.2. Uso de APIs deprecadas

La aplicación **no hace uso de ninguna API marcada como obsoleta (deprecated)**.

3.11.3. Uso de APIs experimentales

La aplicación hace uso de varias APIs marcadas como experimentales. El uso de estas APIs es una decisión consciente que permite construir una interfaz de usuario más rica y eficiente, asumiendo el riesgo controlado de que sus firmas puedan cambiar en futuras versiones estables. Para ello, se utiliza la anotación @OptIn en los puntos necesarios.

Las principales APIs experimentales utilizadas son:

- **@ExperimentalMaterial3Api**: Esta anotación se utiliza de forma transversal en toda la aplicación. Es necesaria para acceder a componentes clave de la librería Material Design 3, como TopAppBar, Scaffold, ModalBottomSheet y OutlinedTextField.
- **@ExperimentalFoundationApi**: Se utiliza para implementar componentes de bajo nivel que ofrecen funcionalidades avanzadas de layout. En Pixora, es fundamental para el uso del HorizontalPager en la pantalla de actividad del usuario, permitiendo crear una interfaz basada en páginas que el usuario puede deslizar horizontalmente.
- **@FlowPreview**: Esta anotación se emplea para acceder a operadores avanzados de Kotlin Flow. Concretamente, se utiliza para el operador debounce en la pantalla de búsqueda. Este operador se utiliza para optimizar el rendimiento, ya que evita que se lance una nueva búsqueda a la API con cada tecla que pulsa el usuario, esperando en su lugar a que haya una pequeña pausa en la escritura.

3.12. Animaciones de la UI

Para mejorar la experiencia de usuario y proporcionar un feedback visual claro, Pixora incorpora varias animaciones clave en su interfaz.

Animación en la SplashScreen con Lottie: Durante la pantalla de carga inicial, se muestra una animación vectorial personalizada. Para ello, se ha utilizado la librería Lotti. La animación se reproduce una sola vez, ofreciendo una bienvenida a la aplicación mucho más atractiva y profesional que una imagen estática.

Animación de Visibilidad de la Barra de Navegación: La barra de navegación inferior (BottomBar) no aparece o desaparece de forma abrupta al cambiar de pantalla. En su lugar, se utiliza el componente AnimatedVisibility de Jetpack Compose. Este gestiona una transición suave de entrada y salida (slideInVertically y slideOutVertically), haciendo que la barra se deslice hacia la parte inferior de la pantalla al ocultarse y vuelva al aparecer. Esto crea una transición fluida y visualmente agradable para el usuario.

Animaciones de Interacción al dar "Me Gusta": Para dar "me gusta" a una foto, se implementó dos animaciones:

1. **Gesto de doble toque:** Al hacer doble toque sobre la imagen en la pantalla de detalle, aparece un icono de corazón superpuesto que crece con un efecto de resorte (spring) y se desvanece suavemente.
2. **Icono en la barra de interacción:** Simultáneamente, el icono del corazón en la barra de acciones realiza una pequeña animación de escala, contrayéndose y expandiéndose con un efecto de "rebote" para dar una sensación táctil y reactiva.

Ambas animaciones se gestionan con Animatable y corrutinas, proporcionando una respuesta instantánea y satisfactoria al usuario.

4. Puntos de mejora

Aunque la versión actual de Pixora cumple con todos los objetivos académicos propuestos , su arquitectura modular y escalable permite plantear diversas evoluciones a futuro. A continuación, se describen algunas posibles líneas de mejora:

Implementación de Funcionalidades Sociales: Añadir perfiles de usuario completos, un sistema de seguidores, y la posibilidad de dejar comentarios en las fotos para fomentar la interacción.

Migración a un Backend en la Nube: Para soportar las funcionalidades sociales y permitir que los usuarios accedan a sus datos desde cualquier dispositivo, sería necesario implementar un backend en la nube (por ejemplo, con Firebase). Esto centralizaría el almacenamiento de fotos, listas y perfiles, superando el enfoque local actual.

Sistema de Recomendaciones Personalizadas: Se podría desarrollar un motor que analice la actividad del usuario (me gustas, guardados, búsquedas) para sugerirle contenido e imágenes de su interés, mejorando significativamente la experiencia de descubrimiento.

Editor de Imágenes Integrado: Añadir funcionalidades de edición básica dentro de la aplicación, como aplicar filtros, recortar, o ajustar el brillo y contraste de las imágenes antes de subirlas.

5. Problemas durante el desarrollo

5.1. Implementación de cambio manual de idioma mediante preferencias

5.1.1. Problema

Uno de los problemas en los que más me atasqué fue permitir al usuario cambiar el idioma de la aplicación desde el menú de ajustes, de forma que el cambio fuera persistente e instantáneo, sin tener que reiniciar la aplicación.

El cambio de idioma con respecto al idioma configurado en el dispositivo fue sencillo, pero para el cambio manual vinieron varias dificultades:

El primer enfoque consistió en guardar la preferencia de idioma en DataStore y luego forzar la recreación de la Activity con activity.recreate() para que el cambio fuera visible al instante. Sin embargo, aunque funcionaba en tiempo real, la elección se perdía al cerrar y volver a abrir la aplicación, ya que esta volvía a tomar el idioma del sistema. Se intentó usar AppCompatDelegate.setApplicationLocales() para comunicar al sistema operativo la nueva preferencia de idioma. A pesar de

implementarlo, el problema persistía y el cambio no se aplicaba correctamente. Esto me llevó bastante tiempo y varios intentos.

5.1.2. Solución

Después de revisar la implementación múltiples veces y comparar el código con varios tutoriales sin encontrar la causa, la solución se encontró en un pequeño pero crucial detalle dentro de la documentación oficial de Android.

El problema no estaba en la lógica del cambio de idioma, sino en la configuración base de la Activity principal de la aplicación. Para que AppCompatDelegate.setApplicationLocales() funcione correctamente en una aplicación de Compose, la MainActivity **debe heredar de AppCompatActivity**. En mi caso, estaba heredando de ComponentActivity, que es la clase base por defecto en los proyectos de Compose.

Una vez realizado este simple cambio en la declaración de la clase MainActivity, todo el sistema comenzó a funcionar como se esperaba.

5.1.3. Bibliografía de la solución

Portal de desarrolladores de Android:
https://developer.android.com/guide/topics/resources/app-languages

En concreto fue esta advertencia la que hizo que lo solucionara:

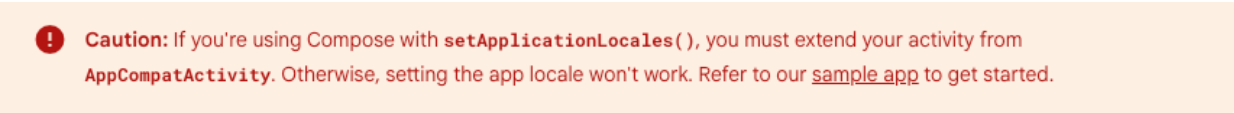

Figura 5.1. Advertencia de uso de AppCompactActivity en Compose

5.2. Estructura de Navegación Anidada para la Gestión de la UI

Otro desafío relevante durante el desarrollo se presentó en la arquitectura de la navegación de la aplicación, concretamente en cómo gestionar la coexistencia de un grafo de navegación principal y otro anidado para las secciones accesibles a través de una barra de navegación inferior (BottomNavigationBar).

5.2.1. Descripción del problema

La estructura de la aplicación se divide en dos niveles de navegación principales:

1. **Grafo de Navegación Principal (PixoraNavigation)**: Gestiona las transiciones de alto nivel, como el paso de la pantalla de carga

(SplashScreen) a la pantalla principal (MainScreen) que contiene toda la funcionalidad post-autenticación.

2. **Grafo de Navegación Anidado (MainScreen)**: Una vez en MainScreen, un segundo NavHost gestiona la navegación entre las diferentes pestañas de la barra inferior (Inicio, Búsqueda, Perfil, etc.) y las pantallas de detalle que se abren desde ellas.

El problema surgió al intentar **unificar ambos grafos de navegación** en un único NavHost centralizado en PixoraNavigation. El objetivo era tener una única fuente de verdad para la navegación, lo que teóricamente simplificaría el código y la gestión de rutas.

Sin embargo, este enfoque demostró ser contraproducente, al menos de la manera en la que yo lo estaba enfocando. Mover toda la lógica de MainScreen (que incluye la gestión de la visibilidad de la barra de navegación, la lógica de los permisos, la apertura de la cámara/galería y la gestión de BottomSheet) al NavHost principal resultó en un **acoplamiento excesivo y mayor complejidad**. El Composable PixoraNavigation se sobrecargó de responsabilidades que no le correspondían, haciendo el código difícil de leer, mantener y escalar.

5.2.2. Solución

Finalmente, tomé la decisión de mantener la estructura de navegación anidada. Aunque implica tener dos NavHost, esta solución presenta varias ventajas prácticas:

- **Encapsulación**: MainScreen actúa como un contenedor autocontenido que gestiona su propio estado de navegación y de interfaz de usuario (como la BottomNavigationBar). Esto lo convierte en un componente modular y reutilizable.
- **Claridad del Código**: La lógica de navegación de alto nivel (Splash → Main) permanece separada de la navegación interna de la aplicación, resultando en un código más limpio y fácil de entender en ambos niveles.
- **Mantenibilidad**: Aislar la complejidad en MainScreen facilita la depuración y la adición de nuevas pantallas en el futuro sin afectar al flujo de navegación principal.

5.2.3. Bibliografía de la solución

Aunque leí la documentación oficial de Android como https://developer.android.com/guide/navigation/design/nested-graphs y algún artículo, o encontré un solución exacta para este problema.