Pixora

Desarrollo de Aplicaciones iOS

Máster Universitario en
Informática Móvil

Israel Brea Piñero

Universidad Pontificia de Salamanca

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 iOS. 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 SwiftUI, 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 iOS 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 iOS del Máster en Informática Móvil (MIMO) de la Universidad Pontificia de Salamanca.

1.3.2. Objetivos específicos

1. Implementar una interfaz adaptable y funcional utilizando SwiftUI, compatible con distintos tamaños de pantalla en iPhone y iPad.
2. Aplicar una arquitectura limpia que separe correctamente las capas de presentación, dominio y data.
3. Consumir servicios web externos mediante llamadas HTTP usando URLSession.
4. Persistir información localmente mediante CoreData.
5. Integrar librerías externas a través de Swift Package Manager para mejorar la experiencia visual y la usabilidad.
6. Ofrecer una navegación fluida mediante listas dinámicas y cuadrículas (List y LazyVGrid).
7. Garantizar una experiencia de uso estable, sin errores en ejecución básica y con soporte para orientación vertical y horizontal.
8. Incorporar elementos avanzados como Combine o notificaciones
9. Subir la app a la App Store cumpliendo los requisitos técnicos y de distribución de Apple.

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. iPhone vertical

Figura 2.2. Vista Launch Screen

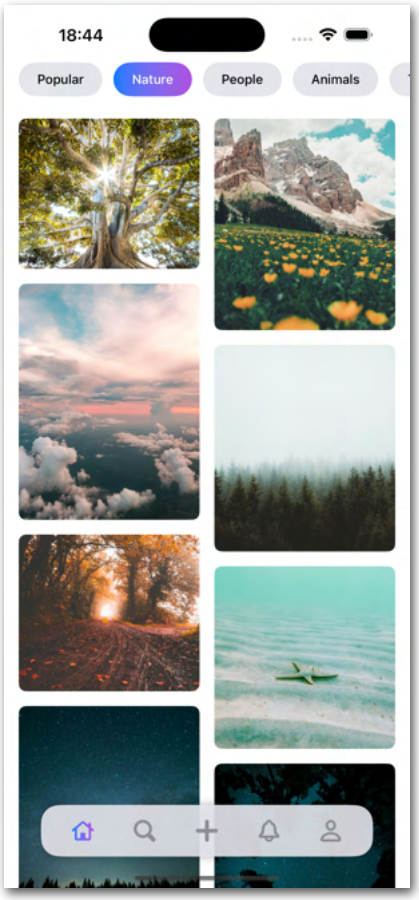


Figura 2.3. Vista Home

Figura 2.4. Vista Search

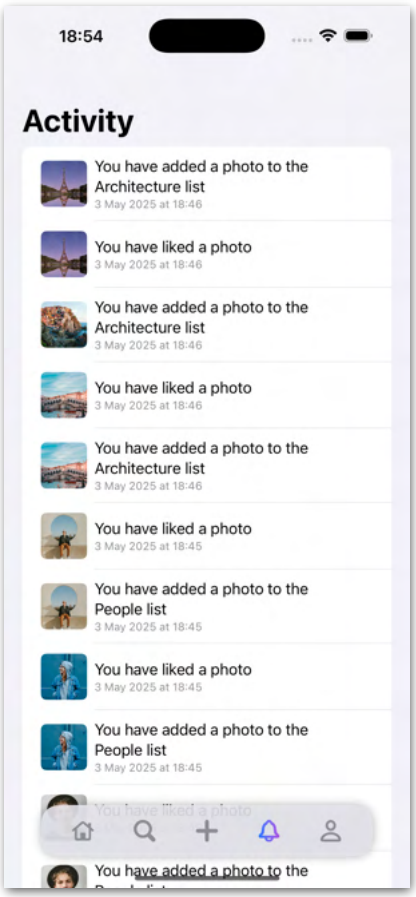


Figura 2.5. Vista Notifications

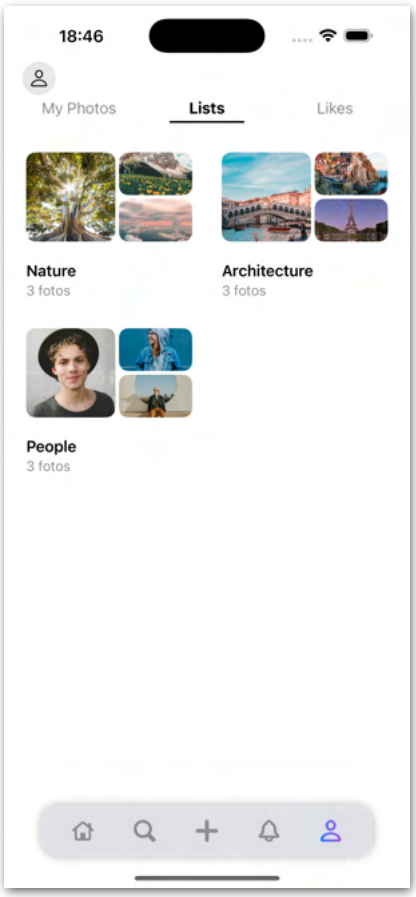


Figura 2.6. Vista Lists

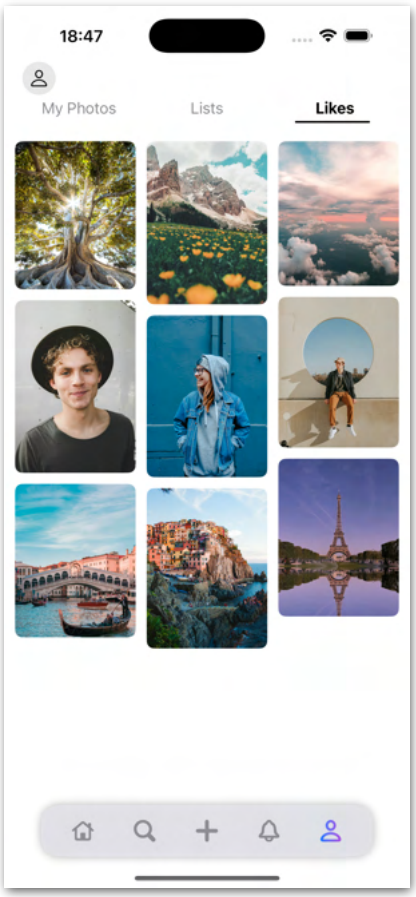


Figura 2.7. Vista Likes

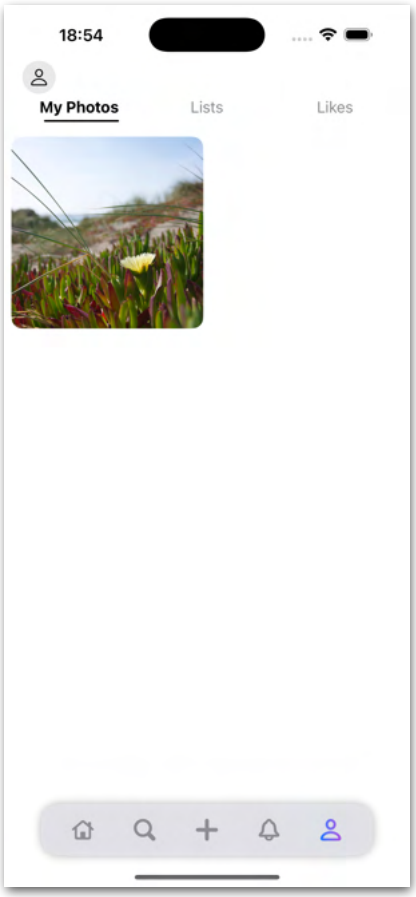


Figura 2.8. Vista MyPhotos

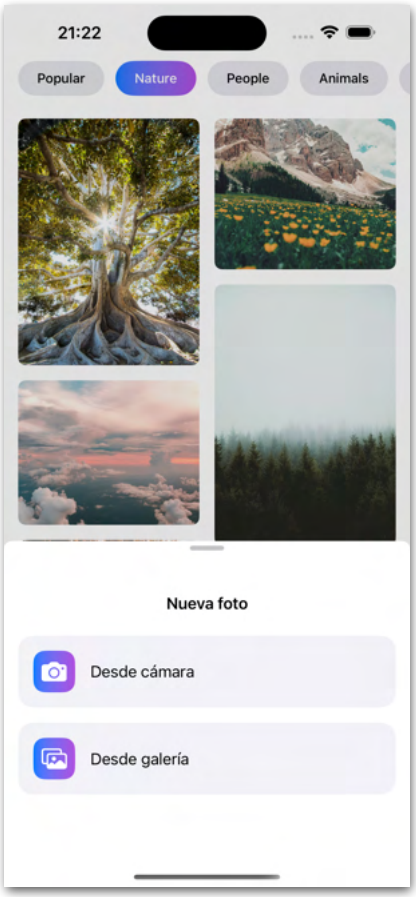


Figura 2.9. Vista BottomSheet

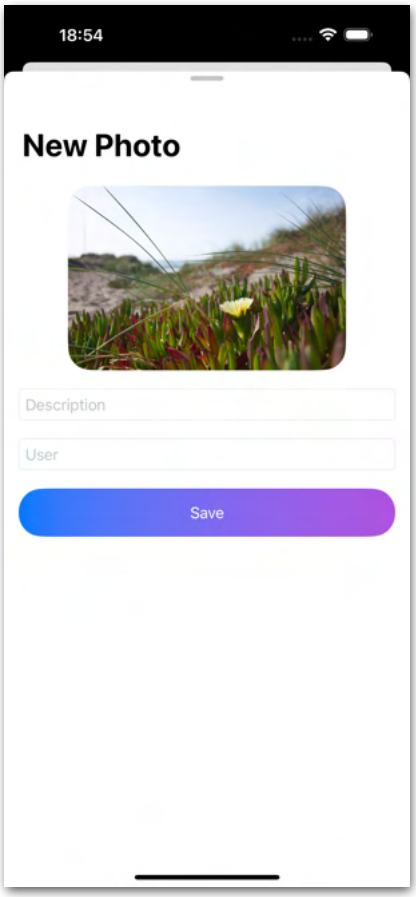


Figura 2.10. Vista AddPhotoForm

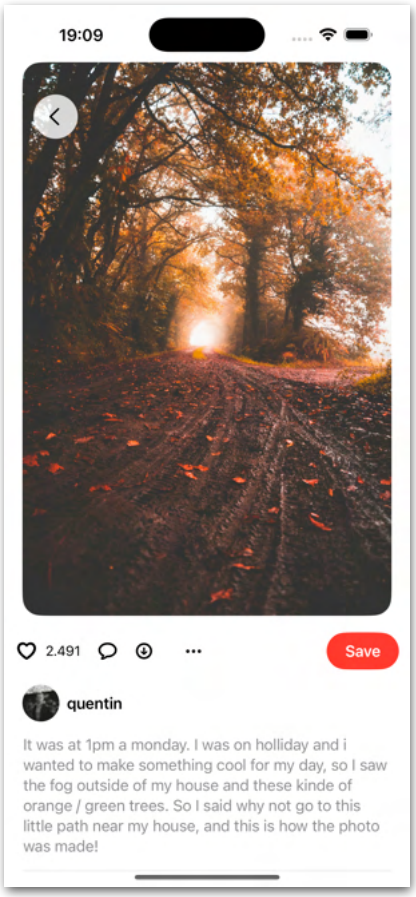


Figura 2.11. Vista PhotoDetails

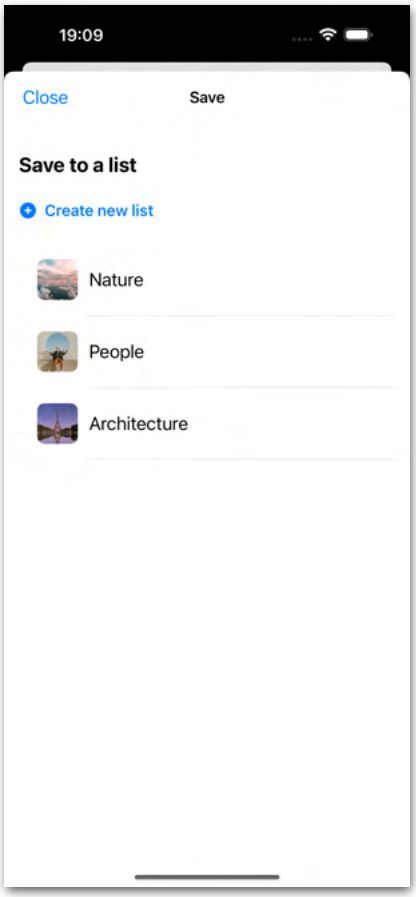


Figura 2.12. Vista NewList

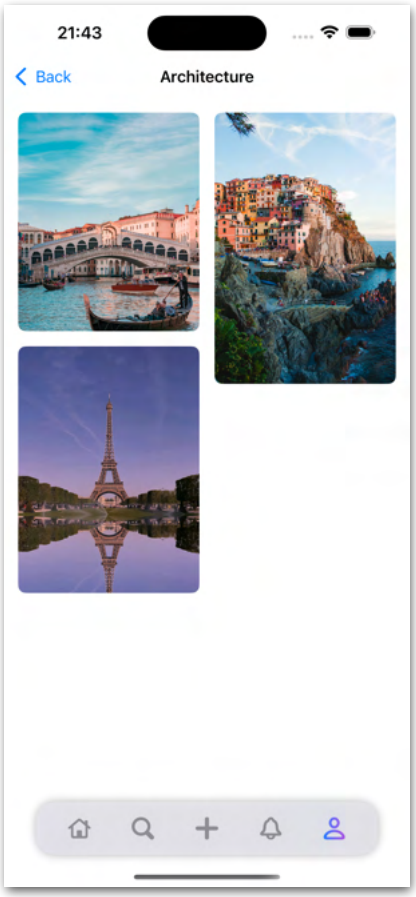


Figura 2.13. Vista DetailsPhotoList

2.1.2. iPhone horizontal

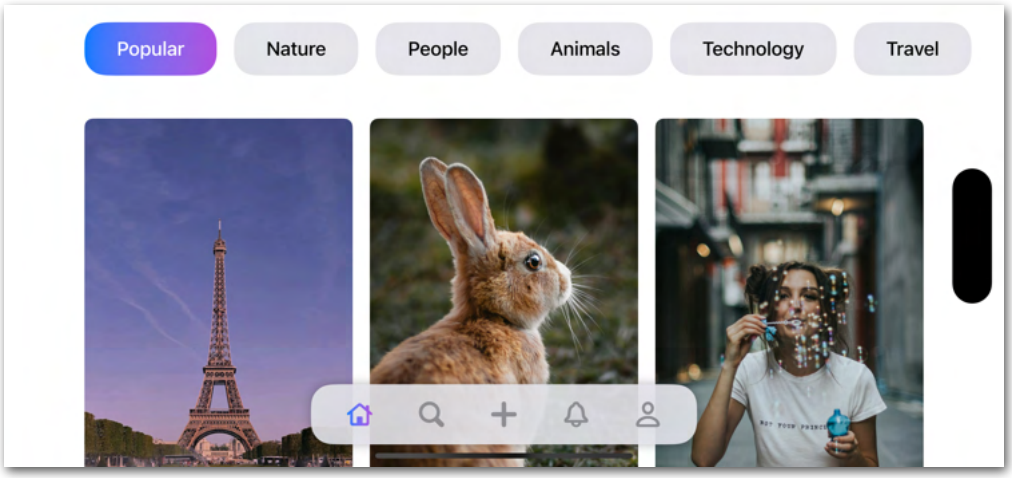


Figura 2.14. Vista Home horizontal

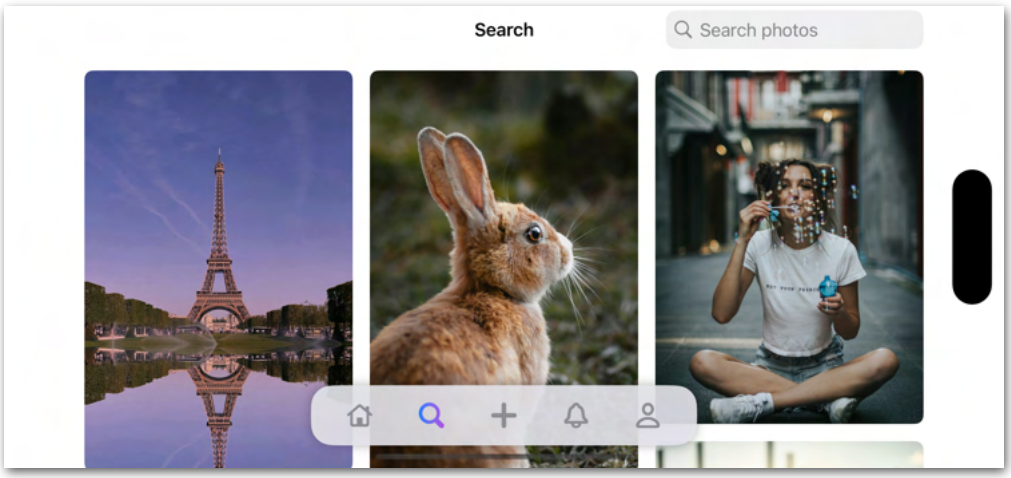


Figura 2.15. Vista Search horizontal

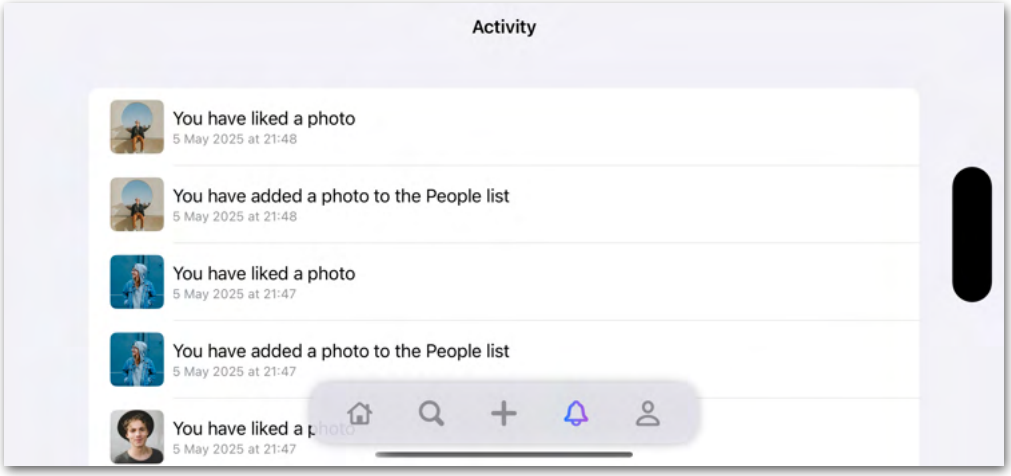


Figura 2.16. Vista Notifications horizontal

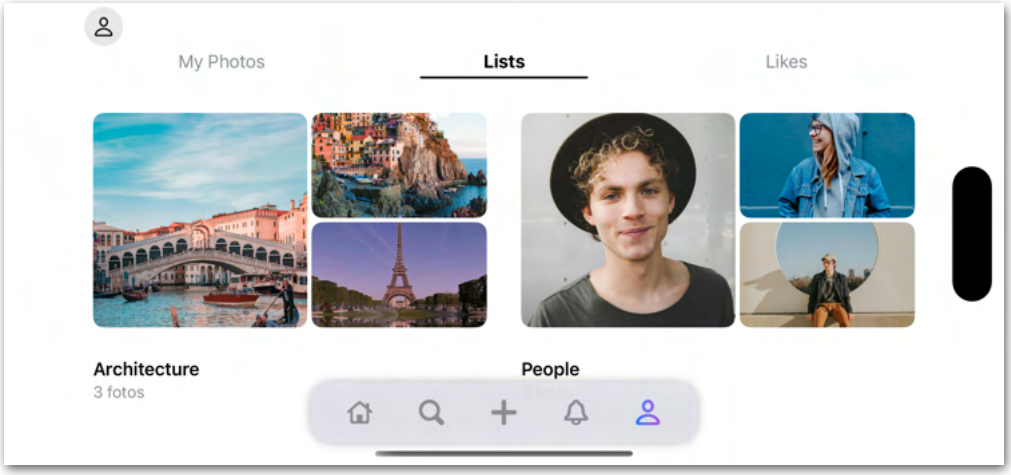


Figura 2.17. Vista Lists horizontal

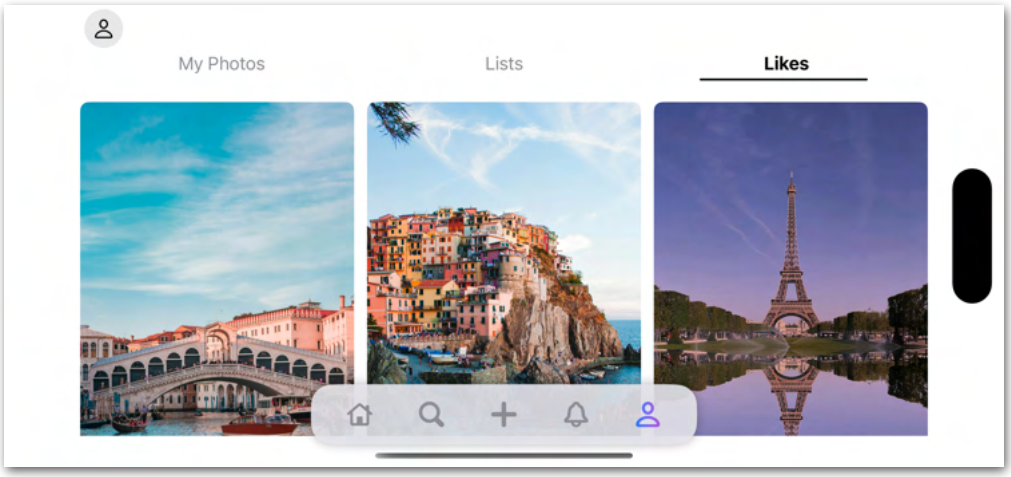


Figura 2.18. Vista Likes horizontal

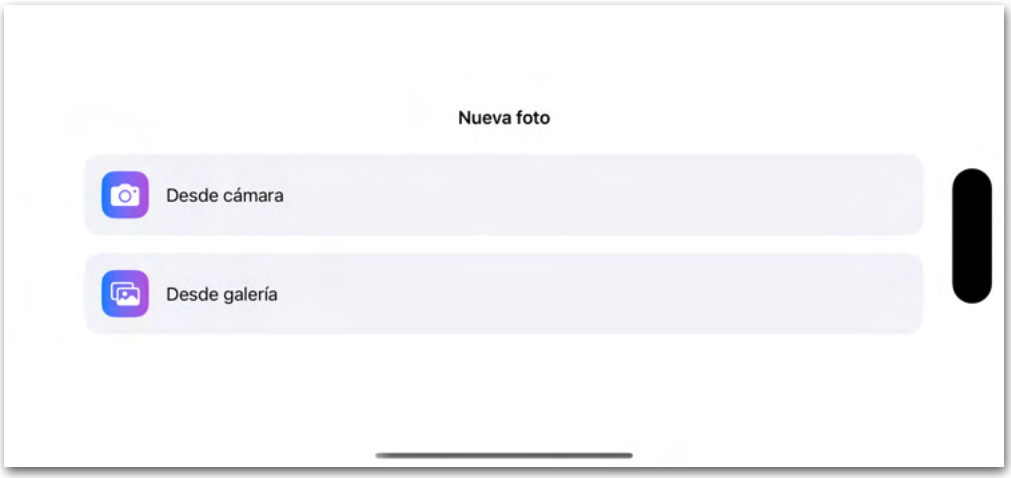


Figura 2.19. Vista BottomSheet horizontal

Figura 2.20. Vista MyPhotos horizontal

2.1.3. iPad

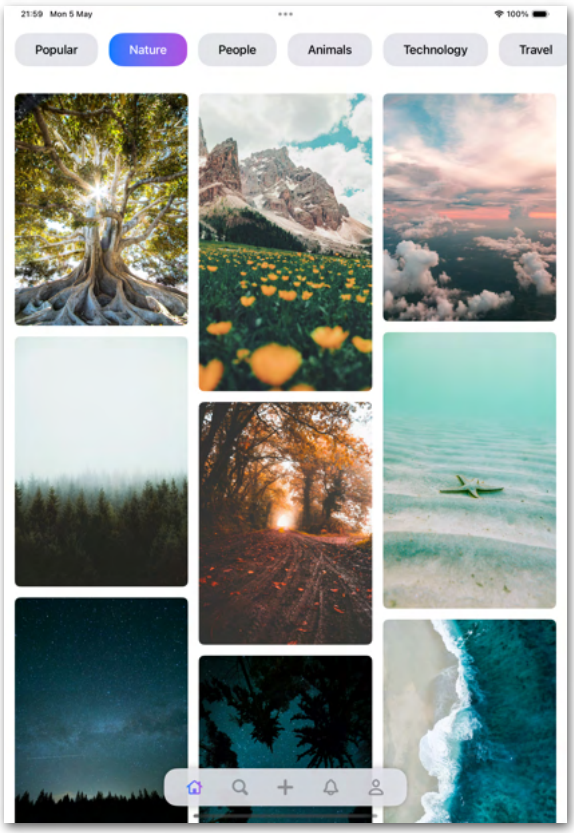


Figura 2.21. Vista Home iPad

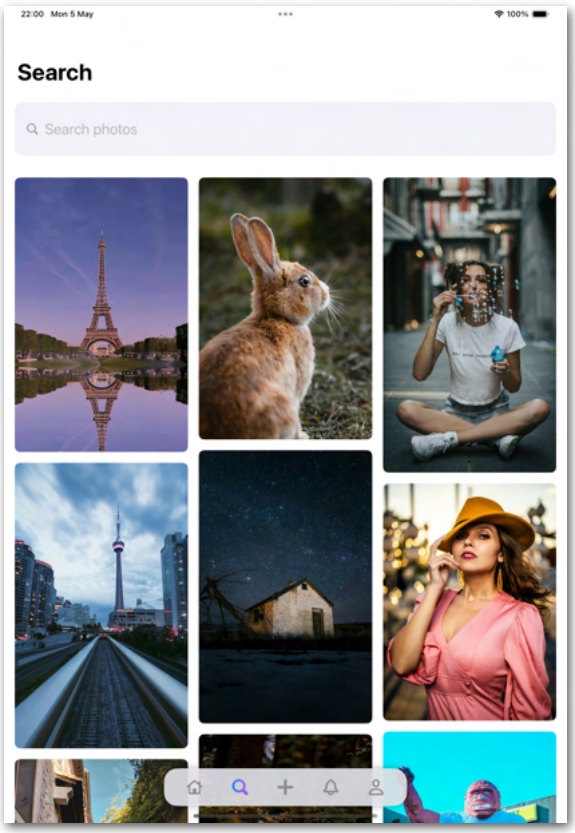


Figura 2.22. Vista Search iPad

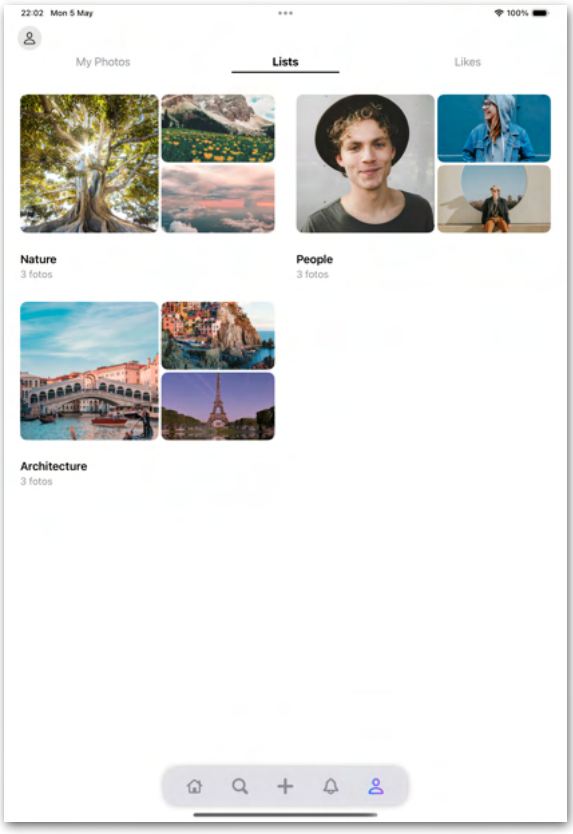


Figura 2.23. Vista Lists iPad

Figura 2.24. Vista Likes iPad

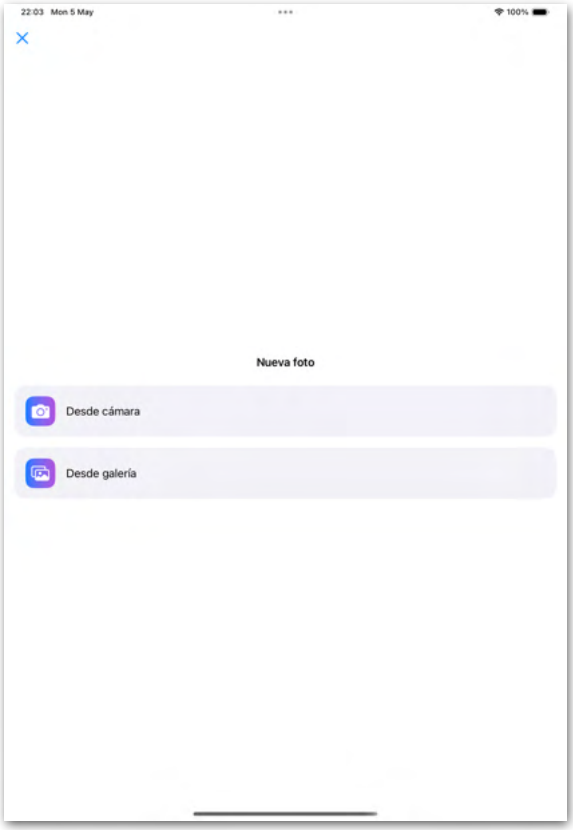


Figura 2.25. Vista BottomSheet iPad

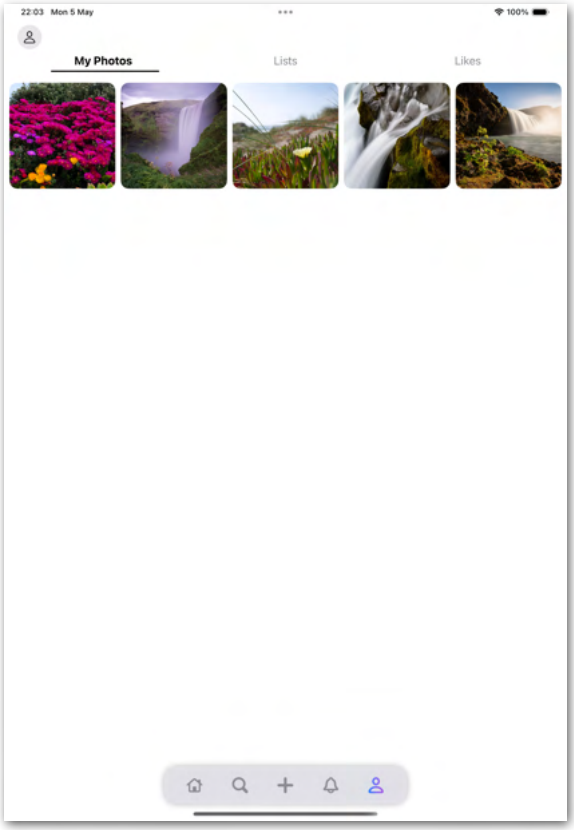


Figura 2.26. Vista MyPhotos iPad

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 Data) • Descripción • Nº de me gustas • Usuario del fotógrafo • Foto de perfil del fotógrafo • Color de fondo • Si me gusta la foto o no

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 listas de fotos. En concreto:
Datos específicos	• Nombre 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 fotos añadidas a listas. En concreto:
Datos específicos	• Fecha a la que se ha añadido 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 una lista) • Nombre de la lista • Fecha y hora de la actividad

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 en 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 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	Redirección tras subir foto
Descripción	El sistema deberá redirigir automáticamente al usuario a la pantalla "MyPhotos" tras subir una nueva foto.

Tabla 2.23. Requisito funcional - Redirección tras subir foto

RF-20	Navegación atrás
Descripción	El sistema debe permitir la navegación a la página anterior para una experiencia fluida del usuario.

Tabla 2.24. Requisito funcional - Navegación atrás

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.
- **Disponibilidad**: La aplicación debe ofrecer un nivel de disponibilidad superior al 99,5%, especialmente considerando que los usuarios podrán acceder en distintos momentos del día desde múltiples dispositivos.
- **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 las últimas versiones de iOS, incluyendo soporte para iPhone e iPad. El layout debe adaptarse a todos los tamaños de pantalla disponibles, desde iPhone SE hasta iPad Pro, tanto en orientación vertical como horizontal.
- **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, como recordatorios tras cerrar la app, enviadas a los 5 y 30 segundos para fomentar la reentrada del usuario.

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.

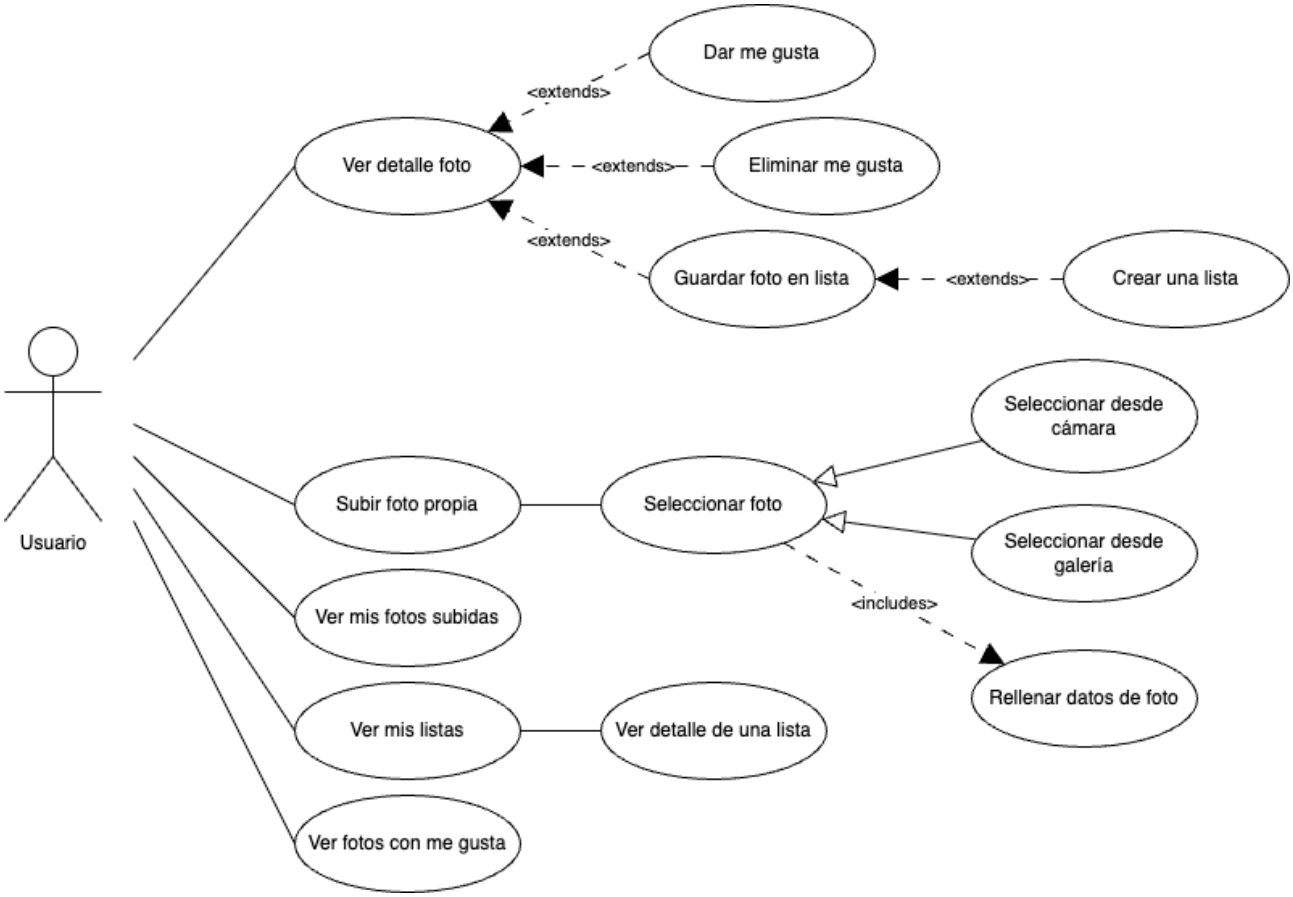


Figura 2.27. Diagrama de casos de uso Pixora

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 2.25. 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 2.26. 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).
	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".
Postcondición	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 2.26. 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 2.27. Escenario UC-04

<table>
<tr><td>UC-05</td><td colspan="2">Crear una lista</td></tr>
<tr><td>Descripción</td><td colspan="2">El sistema permite al usuario crear una nueva lista de fotos.</td></tr>
<tr><td>Precondición</td><td colspan="2">El usuario está en el proceso de guardar una foto en una lista y ha pulsado la opción "Añadir nueva lista”.</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 "Añadir nueva lista".</td></tr>
<tr><td>2</td><td>El sistema muestra un alert en la pantalla para añadir un nombre a la lista.</td></tr>
<tr><td>3</td><td>El usuario introduce el nombre de la lista.</td></tr>
<tr><td>4</td><td>El usuario confirma la creación.</td></tr>
<tr><td>5</td><td>El sistema crea la nueva lista.</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td colspan="2">Se ha creado una nueva lista vacía. La nueva lista es visible en la pantalla "Lists".</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 crear la lista</td></tr>
<tr><td>*</td><td>El sistema muestra un mensaje de error.</td></tr>
</table></td></tr>
</table>

Tabla 2.28. 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>Secuencia</td><td colspan="2"><table>
<tr><th>Pasos</th><th>Acción</th></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>
</table></td></tr>
<tr><td>Postcondición</td><td colspan="2">Se ha subido la foto del usuario y es visible en la pantalla "MyPhotos".</td></tr>
<tr><td>Excepciones</td><td colspan="2"><table>
<tr><th>Pasos</th><th>Acción</th></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></td></tr>
</table>

Tabla 2.29. Escenario UC-06, UC-7, UC-8, UC-09, UC-10

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 App Store. Para ello, accedemos al siguiente enlace desde un dispositivo iOS:

Enlace a Pixora: https://apps.apple.com/es/app/pixora/id6745432045?l=en-GB

Una vez instalada, al abrir la aplicación, veremos la Launch Screen (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.

En 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 2.4).
- **Plus (+)**: Permite subir una foto propia desde cámara o galería (Figura 2.9 y Figura 2.10).
- **Notifications**: Muestra las actividades recientes del usuario y sus interacciones (Figura 2.5).
- **Profile**: Desde donde se puede acceder a fotos con me gusta, listas creadas y fotos subidas (Figura 2.6, Figura 2.7 y Figura 2.8).

2.5.3. Buscar contenido

En la pantalla de Search (Figura 2.4), 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 notificaciones

En la pantalla de Notifications (Figura 2.5), 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 2.7).
- **Lists**: Listas personalizadas del usuario donde ha agrupado fotos (Figura 2.6).
- **MyPhotos**: Fotos subidas por el propio usuario (Figura 2.8).

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 2.11). 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 (AddToList, 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 2.13), 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

Una 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 SwiftUI, 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 SwiftUI, 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 vayan 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. Utilizan las Fuentes de Datos.
   - **Fuentes de Datos (Data Sources)**: 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 - SwiftUI)**: 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.

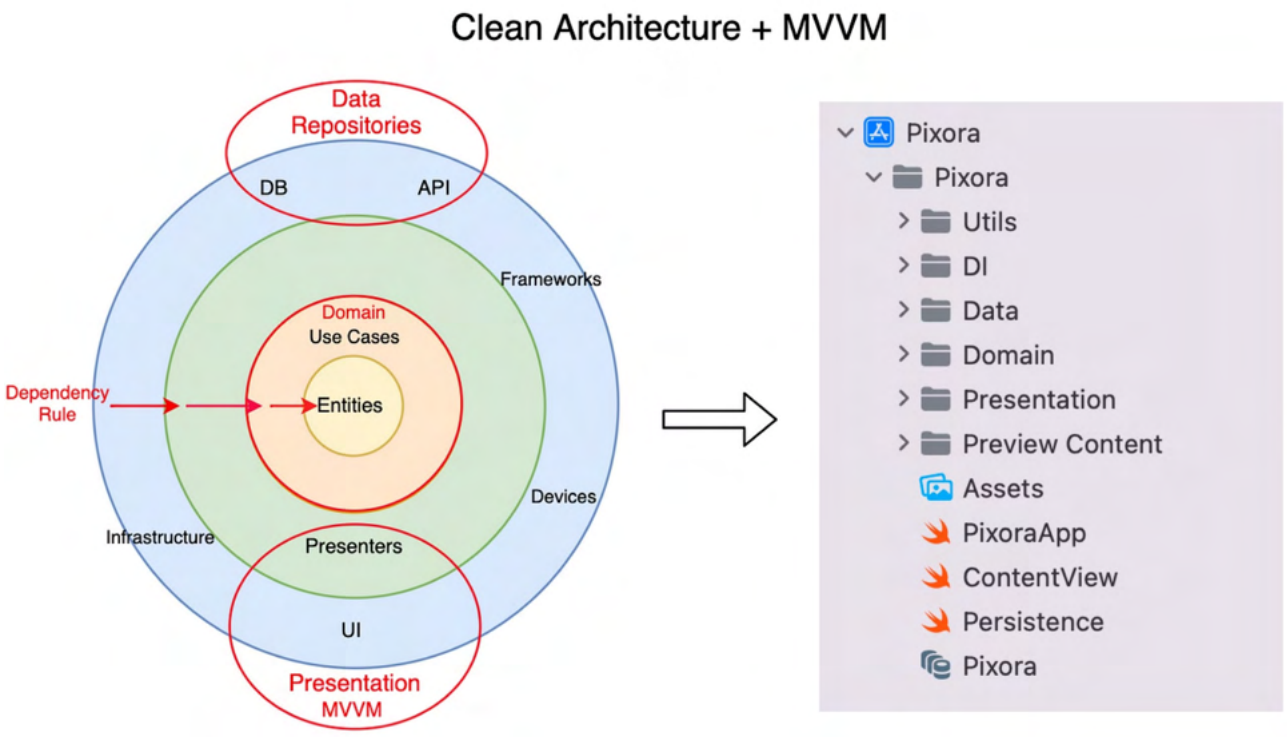


Figura 3.1. Clean Architecture + MVVM

3.2. 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. **Utils**: 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.
2. **DI**: Contiene el código relacionado con la Inyección de Dependencias.
3. **Data**: Este directorio corresponde a la Capa de Data (Data Layer) de la arquitectura Clean.
    a) **DataSource**: Contiene las implementaciones específicas de las fuentes de datos.
        - **Remote**: Implementaciones para acceder a datos desde una fuente remota, en este caso la API.
        - **Local**: Implementaciones para acceder a datos desde una fuente local, en este caso Core Data.
    b) **Network**: Archivos y clases relacionados específicamente con la capa de red (configuración, clientes HTTP, manejo de peticiones).
    c) **Repository**: Implementaciones concretas de los protocolos de repositorio definidos en la Capa de Dominio.
    d) **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.
    e) **Model**: Definición de estructuras de datos específicas de la Capa de Data.
    f) **DataConstants.swift**: Archivo para agrupar constantes específicas de la Capa de Data (URLs base, claves de API, etc.).
4. **Domain**: Este directorio corresponde a la Capa de Dominio (Domain Layer).
    a) **UseCase**: Contiene las clases que representan las operaciones específicas de la lógica de negocio (los casos de uso).
    b) **Entity**: Definición de las estructuras de datos puras que representan los conceptos fundamentales de la aplicación (las entidades de negocio).
    c) **Errors**: Definición de tipos de errores personalizados que pueden ocurrir dentro de la lógica de negocio del Dominio.
    d) **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.
5. **Presentation**: Este directorio corresponde a la Capa de Presentación (Presentation Layer). Contiene las Vistas de SwiftUI y los ViewModels asociados que interactúan con los Casos de Uso del Dominio.
6. **Preview Content**: Carpeta estándar de Xcode para archivos y recursos utilizados únicamente para las previsualizaciones de las Vistas de SwiftUI.

7. **Assets.xcassets**: Catálogo de assets para gestionar imágenes, colores y otros recursos gráficos de la aplicación.
8. **PixoraApp.swift**: El archivo principal que define el punto de entrada de la aplicación utilizando la estructura App de SwiftUI.
9. **ContentView.swift**: El archivo de Vista inicial, utilizado como la primera pantalla de la aplicación.
10. **Persistence.swift**: Contiene la configuración y la lógica para inicializar y gestionar el contenedor de Core Data, utilizado para la persistencia local de datos.
11. **Pixora.xcdatamodeld**: El archivo de modelo de datos visual de Core Data, donde se definen las entidades, atributos y relaciones de la base de datos local.

3.3. Gestión de datos

Para la gestión y persistencia de datos a nivel local en la aplicación, he utilizado el framework Core Data proporcionado por Apple. Core Data no es una base de datos en sí misma, sino un framework que gestiona un grafo de objetos y proporciona funcionalidades para la persistencia, recuperación, validación y gestión de cambios en esos datos.

Otra opción para la persistencia de datos es SwiftData, un framework más reciente introducido por Apple y que ofrece una sintaxis más Swift-centric e integrada con el lenguaje, y, desde mi punto de vista inicial, más sencillo. Sin embargo, para este proyecto, me decanté por Core Data, ya que mi principal motivación fue la de aprender y trabajar con la tecnología subyacente y más establecida antes de pasar a la abstracción más sencilla que ofrece SwiftData.

3.3.1. Modelo Conceptual

Para complementar la explicación sobre la gestión de datos locales con Core Data, 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.

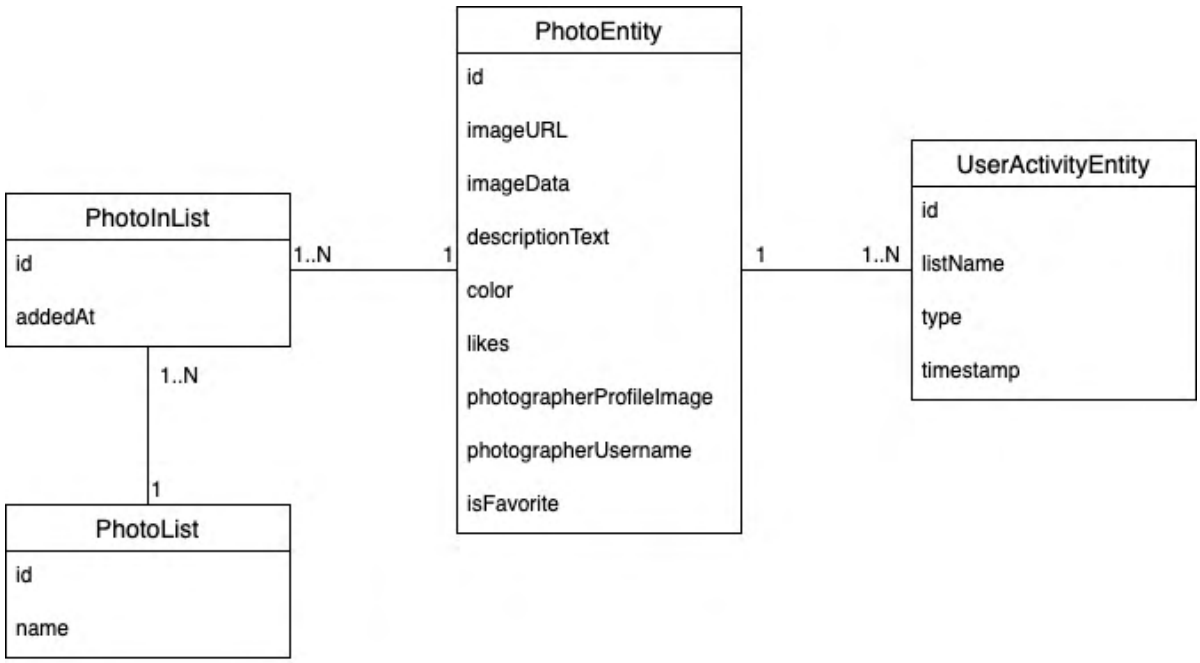


Figura 3.2. Diagrama conceptual

Nombre	Descripción	Tipo	Entidad
id	Id de la foto	String	PhotoEntity
imageURL	URL de la foto que llega de la API	String	PhotoEntity
imageData	Data de la imagen que se sube mediante multimedia	Binary Data	PhotoEntity
descriptionText	Descripción de la foto	String	PhotoEntity
color	Color de fondo de la foto	String	PhotoEntity
likes	Número de likes de la foto	Integer 32	PhotoEntity
photographerProfileImage	Foto de perfil del usuario	String	PhotoEntity
photographerUsername	Nombre de usuario al que corresponde la foto	String	PhotoEntity
isFavorite	Indica si la has marcado como me gusta o no	Boolean	PhotoEntity
id	Id de la foto en la lista	UUID	PhotoInListEntity
addedAt	Fecha y hora en la que se añade la foto a la lista	Date	PhotoInListEntity
id	Id de la lista	UUID	PhotoListEntity
name	Nombre de la lista	String	PhotoListEntity
id	Id de la actividad del usuario	UUID	UserActivityEntity
listName	Nombre de la lista en la que ocurre la actividad	String	UserActivityEntity
type	Indica si la actividad es que se ha añadido una foto a una lista o se ha dado me gusta a una foto	String	UserActivityEntity
timestamp	Fecha y hora en la que ocurre la actividad	Date	UserActivityEntity

Table 3.1. Descripción de atributos del modelo conceptual de datos

3.4. Integraciones con APIs / Servicios externos

La aplicación integra servicios externos para enriquecer su funcionalidad. Concretamente, se conecta a la API externa de Unsplash para poder acceder a un gran catálogo de imágenes de alta calidad, que es el contenido principal que se presenta al usuario.

La gestión de las llamadas HTTP a estos servicios se implementa dentro de la Capa de Data de la arquitectura Clean, específicamente en la subcapa encargada de las fuentes de datos remotas. He decidido realizar las peticiones de red directamente con URLSession, el framework nativo de Apple, principalmente por dos motivos: para tener un control total sobre el proceso de la petición y la respuesta, sin la abstracción completa de librerías de terceros, y para profundizar mi conocimiento del framework de red nativo de iOS.

Con respecto a como está organizado en el código fuente, he optado por un diseño de la capa de red bastante modular, estructurado en diferentes abstracciones. Se ha definido un RequestProtocol que estandariza cómo se construyen las peticiones (definiendo URL, método HTTP, parámetros, etc.). Un NetworkManager es el responsable de ejecutar físicamente esta URLRequest resultante utilizando URLSession y de manejar la respuesta bruta de la red. Finalmente, un APIManager coordina el flujo completo: toma una petición definida por RequestProtocol, la pasa al NetworkManager para su ejecución y, una vez obtenida la respuesta en formato Data, la entrega a un DataParser para transformar esos datos brutos (JSON) a los modelos Swift.

3.4.1. 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.

3.4.1.1. 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.

3.4.1.2. 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.

3.4.1.3. 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.

3.5. 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-iOS.git

3.6. Justificación de tecnologías y herramientas empleadas

En esta sección se justifica la elección de las diferentes tecnologías y herramientas utilizadas durante el proyecto.

3.7.1. Elección del lenguaje y framework (Swift y SwiftUI)

Como lenguaje de programación principal, se ha utilizado Swift. La elección de Swift se debe a que es el lenguaje moderno y recomendado por Apple para el desarrollo en sus plataformas, destacando por su seguridad (gracias a su fuerte tipado y gestión automática de memoria), su rendimiento y su sintaxis expresiva y concisa que facilita la lectura y escritura del código.

Para la construcción de la interfaz de usuario, se ha optado por SwiftUI en lugar de UIKit. SwiftUI es el framework de UI declarativo de Apple, que simplifica significativamente el proceso de diseño de interfaces al permitir describir la apariencia y el comportamiento de la UI de forma más intuitiva. Su integración nativa con Swift, su diseño reactivo y su compatibilidad con patrones como MVVM lo convierten en una elección ideal para desarrollar aplicaciones modernas en el ecosistema de Apple de manera más rápida y eficiente que con el enfoque imperativo de UIKit.

3.7.2. Librerías externas utilizadas

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

- **CryptoSwift**: Es una colección de algoritmos criptográficos implementados en Swift. En este proyecto, se utiliza específicamente para generar el hash MD5 necesario como parte de la autenticación y construcción de la URL en las peticiones a la API externa, como se observa en la implementación de la capa de red.
- **Lottie**: Es una librería de Airbnb que permite renderizar animaciones de After Effects de forma nativa en aplicaciones móviles. Se ha integrado para mostrar una animación fluida y visualmente atractiva durante la Launch Screen de la aplicación, mejorando la experiencia de inicio del usuario.
- **SDWebImageSwiftUI**: Esta librería es una extensión de SDWebImage, diseñada específicamente para funcionar con SwiftUI. Su propósito principal es la carga asíncrona y el cacheo eficiente de imágenes desde URLs. Se utiliza en el proyecto para mostrar las imágenes obtenidas de la API de Unsplash en la interfaz de usuario de manera optimizada, evitando bloqueos de la UI y gestionando la caché de imágenes automáticamente.
- **Swinject**: Es un ligero framework de Inyección de Dependencias (DI) para Swift. Se utiliza en el proyecto para gestionar las dependencias entre los diferentes componentes y capas de la arquitectura. Facilita la definición del Resolver que se encarga de proporcionar las instancias necesarias donde son requeridas, haciendo el código más modular, testable y fácil de mantener.

Para la integración y gestión de dependencias de estas librerías se ha utilizado la herramienta de Apple de **Swift Package Manager** (SPM).

3.7.3. Gestión de notificaciones locales

Para interactuar con el usuario incluso cuando la aplicación no está activa en primer plano, se ha implementado un sistema de notificaciones locales. Estas notificaciones se generan y gestionan directamente en el dispositivo del usuario y no requieren de un servidor externo para ser enviadas.

La gestión de estas notificaciones se realiza utilizando el framework nativo de Apple UserNotifications. Este framework permite solicitar permiso al usuario para mostrar notificaciones, definir el contenido de la notificación (título, cuerpo, sonido) y programar su entrega basándose en diversas condiciones, como un intervalo de tiempo o una ubicación.

En la aplicación, se utiliza una clase NotificationManager que encapsula las interacciones con UserNotifications, manejando la solicitud de autorización y la programación/cancelación de notificaciones. Un NotificationUseCase (dentro de la Capa de Dominio) define la lógica de negocio asociada a las notificaciones, como por ejemplo, programar recordatorios cuando la aplicación pasa a segundo plano (.background scenePhase), y cancelarlas cuando vuelve a primer plano (.active). Esto permite recordar al usuario que vuelva a la aplicación después de un tiempo de inactividad.

3.7.4. Frameworks de Apple empleados

Además de Swift y SwiftUI como lenguaje y framework principal de UI, la aplicación hace uso de varios frameworks nativos proporcionados por Apple para implementar funcionalidades específicas. A continuación, se comentan algunos de los más relevantes empleados en el proyecto:

- **UserNotifications**: Este framework se ha utilizado para gestionar las notificaciones locales de la aplicación, como se describió en el apartado 3.6.3. Permite solicitar permisos al usuario y programar la entrega de notificaciones directamente en el dispositivo para recordar al usuario o informarle sobre eventos relevantes.
- **AVFoundation**: Es un framework potente para trabajar con medios audiovisuales en las plataformas de Apple. En este proyecto, se ha empleado AVFoundation para implementar la funcionalidad de acceso a la cámara del dispositivo y la captura de fotografías. Permite configurar la sesión de captura, gestionar las entradas (cámara frontal/trasera) y procesar las fotos capturadas.
- **UIKit**: Aunque la interfaz de usuario principal está construida con SwiftUI, UIKit, el framework de UI anterior, se sigue utilizando en ciertas partes del proyecto. Principalmente, se recurre a UIKit para funcionalidades que aún no tienen una API equivalente o madura en SwiftUI, o para aprovechar extensiones y helpers existentes. En este caso, se utiliza para obtener información del dispositivo (como verificar si es un iPad con UIDevice.current.userInterfaceIdiom) y para la implementación del CameraPreview, que actúa como un puente (UIViewRepresentable) para integrar la vista previa de la cámara de AVFoundation (basada en UIView) dentro de un entorno SwiftUI.
- **Foundation**: Este es un framework fundamental que proporciona tipos de datos primitivos, colecciones, y acceso a servicios del sistema operativo. Es la base sobre la que se construyen muchos otros frameworks. En este proyecto, Foundation se utiliza para tareas básicas como la gestión de cadenas de texto (String), fechas (Date), identificadores únicos (UUID), manejo de errores, y la interacción con el centro de notificaciones del sistema (NotificationCenter y Notification.Name).

3.7.5. Compatibilidad de layout para diferentes dispositivos y tamaños

Para poder dotar a la aplicación de soporte universal (iPhone/iPad), adaptarla a todos los tamaños de pantalla y soportar las diferentes orientaciones tanto vertical como horizontal, he empleado diversas técnicas y herramientas proporcionadas por SwiftUI y el SDK de iOS:

- **GeometryReader**: Este contenedor de layout lo que permite es obtener información sobre el tamaño y las coordenadas del espacio disponible para una vista. Lo he utilizado para crear layouts flexibles ajustando por ejemplo el número de columnas de un grid basándome en el valor de ancho de pantalla que GeometryReader me proporciona.
- **Detección del Tipo de Dispositivo** (UIDevice.current.userInterfaceIdiom): En algunos puntos, he necesitado detectar específicamente si la aplicación se está ejecutando en un iPad. Por ejemplo, se ha utilizado la propiedad userInterfaceIdiom de UIDevice para detectar si la aplicación se está ejecutando en un iPad (.pad) o

iPhone (.phone) y adaptar la presentación, para mostrar sheet o fullScreenCover dependiendo del dispositivo.

- **Grids Adaptativas** (LazyVGrid con .adaptive): Para mostrar colecciones de elementos (como las fotos) en un diseño de cuadrícula que se ajuste automáticamente al espacio disponible, se ha utilizado LazyVGrid con ítems adaptativos (GridItem(.adaptive(minimum:))). Esta configuración me ha permitido adaptar mediante un tamaño mínimo, el número de columnas que caben en el ancho disponible.
- **Clases de Tamaño de Entorno**: \.horizontalSizeClass y \.verticalSizeClass indican la cantidad de espacio disponible en una dimensión (compacto o regular).

3.7.6. Animaciones

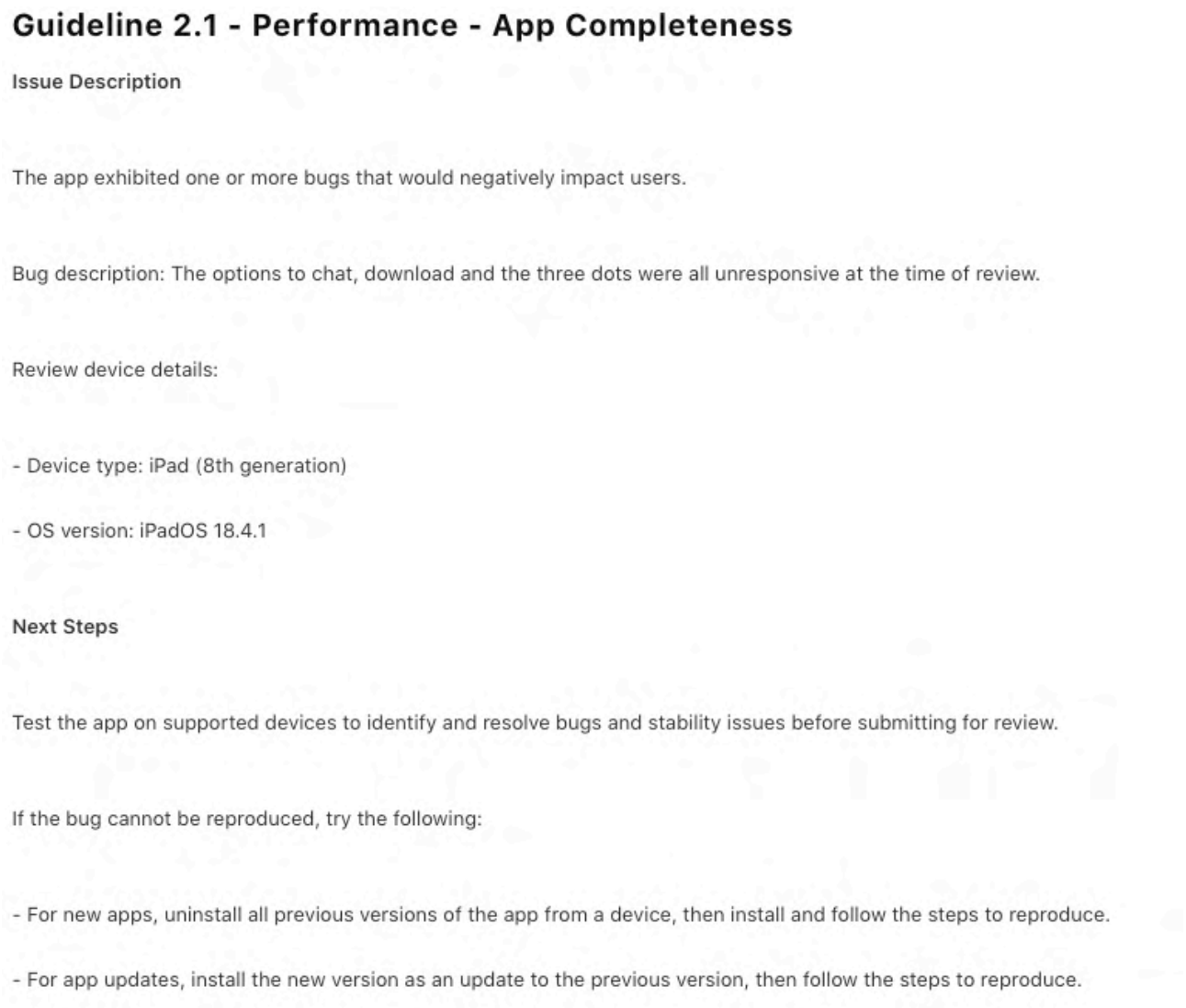
Las animaciones son un elemento clave para mejorar la experiencia de usuario, proporcionando retroalimentación visual, guiando la atención y haciendo la interfaz más dinámica y agradable. La aplicación incorpora varias animaciones para estos fines:

- **Animación en la Launch Screen con Lottie**: Durante la pantalla de inicio (Launch Screen), se muestra una animación personalizada. Para implementar esto, se ha utilizado la librería externa Lottie, que permite integrar animaciones vectoriales creadas en herramientas como After Effects de forma nativa. Esta animación se reproduce una vez y proporciona un inicio de aplicación visualmente atractivo.
- **Animación de Visibilidad de la Tab Bar**: La Tab Bar inferior se muestra u oculta en ciertas pantallas. En lugar de desaparecer o aparecer instantáneamente, se ha aplicado una animación suave utilizando el modificador .offset para desplazar la barra verticalmente fuera de la pantalla y el modificador .animation para interpolar este cambio de posición a lo largo del tiempo. Esto crea una transición fluida y menos abrupta para el usuario.
- **Animación en el Botón "Me Gusta"**: Al interactuar con el botón que indica si una foto es favorita (el "corazón"), se desencadena una animación. Aunque el código específico de la animación del icono del corazón no se mostró, la estructura indica que la acción de tocar el botón (onTap) actualiza el estado (isLiked), lo cual, en una implementación típica de un botón de "Me Gusta" en SwiftUI, estaría acompañado de una animación visual en el icono (por ejemplo, un escalado o un cambio de color con resorte) para confirmar la acción al usuario.
- **Animación en el Botón "Save"**: El botón para guardar una foto en una lista también presenta una animación. Cuando se toca, se utiliza withAnimation con una curva de resorte (.spring) para animar el cierre del sheet una vez se añade la foto a una lista. Posteriormente, el propio botón anima su texto ("Save" a "Saved") y su color de fondo cambia, utilizando el modificador .animation para interpolar suavemente estos cambios visuales.

3.8. Limitaciones y desafíos técnicos

3.8.1. Proceso de publicación

Durante el proceso de distribución de la aplicación me rechazaron la publicación de esta debido al siguiente error:



Guideline 2.1 - Performance - App Completeness

Issue Description

The app exhibited one or more bugs that would negatively impact users.

Bug description: The options to chat, download and the three dots were all unresponsive at the time of review.

Review device details:

- Device type: iPad (8th generation)
- OS version: iPadOS 18.4.1

Next Steps

Test the app on supported devices to identify and resolve bugs and stability issues before submitting for review.

If the bug cannot be reproduced, try the following:

- For new apps, uninstall all previous versions of the app from a device, then install and follow the steps to reproduce.
- For app updates, install the new version as an update to the previous version, then follow the steps to reproduce.

Figura 3.3. Feedback proceso de publicación

Básicamente lo que me comenta es que a la hora de la revisión los botones de chat, download y los tres puntos de la pantalla de detalles de una foto no respondían (Figura 2.11). Esto era normal porque hice esos botones meramente estéticos y no tenían ninguna funcionalidad especifica. Para solucionar este problema lo que hice fue directamente quitar esas opciones quedando la pantalla de la siguiente manera:

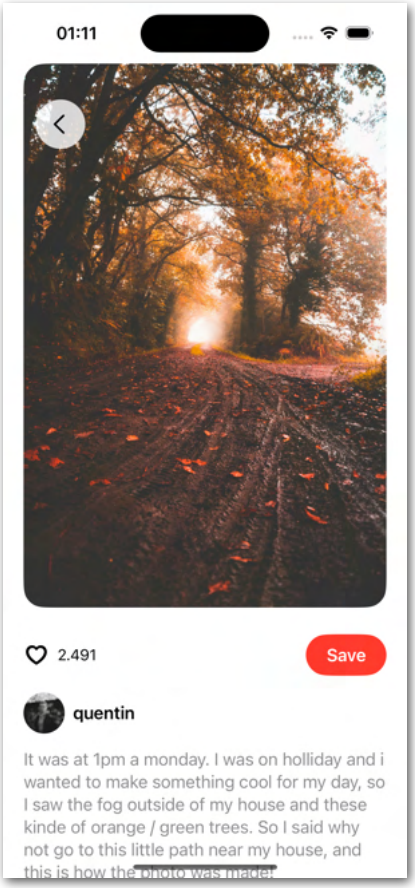


Figura 3.4. Pantalla de detalle de foto corregida

Tras este cambio la aplicación fue vuelta a poner en revisión y aceptada. Se encuentra publicada en la App Store desde el 06/05/2025 y se puede acceder desde el siguiente enlace:
https://apps.apple.com/es/app/pixora/id6745432045?l=en-GB

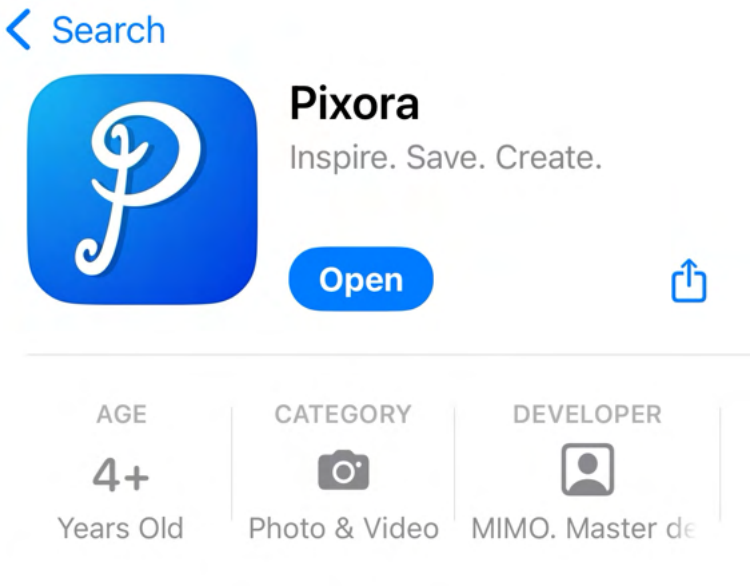


Figura 3.5. Aplicación de Pixora en App Store