**Desarrollo de Aplicaciones Móviles Nativas**

**Práctica 3**



**Nombre**

Parra Mancilla José Ramón

**Boleta**

2021630506

**Grupo**

7CV2

**Profesor**

Gabriel Hurtado Avilés

6 de Octubre del 2025

Índice

[Introducción 1](#_Toc210664821)

[Marco Teórico 2](#_Toc210664822)

[1. Aplicaciones Nativas vs Híbridas vs Web Apps 2](#_Toc210664823)

[2. Arquitectura Android y componentes principales 3](#_Toc210664824)

[3. Patrones de diseño (Clean Architecture) 4](#_Toc210664825)

[Descripción general y flujo de usuario 6](#_Toc210664826)

[Arquitectura técnica 6](#_Toc210664827)

[Acceso a recursos nativos 6](#_Toc210664828)

[Interfaz de Usuario 6](#_Toc210664829)

[Persistencia de Datos 6](#_Toc210664830)

[Gestión de Permisos 6](#_Toc210664831)

[Conclusiones 6](#_Toc210664832)

[Bibliografía 6](#_Toc210664833)

# Introducción

# Marco Teórico

## Aplicaciones Nativas vs Híbridas vs Web Apps

El desarrollo de software móvil moderno se enfrenta a una elección fundamental entre tres paradigmas principales: aplicaciones nativas, aplicaciones híbridas y aplicaciones web (AWS, s.f.). Esta decisión impacta directamente en el rendimiento, la experiencia del usuario (UX), el presupuesto y la estrategia de distribución del producto digital.

* 1. **Aplicaciones Nativas**

Las **aplicaciones nativas** representan el estándar de oro en cuanto a rendimiento y funcionalidad. Se construyen específicamente para un sistema operativo (SO) particular, ya sea **iOS** (utilizando Swift u Objective-C) o **Android** (con Kotlin o Java), aprovechando el lenguaje de programación y las herramientas de desarrollo (*kits* SDK) propios de la plataforma (TRBL Services, 2024). Esta optimización intrínseca les confiere el **mayor rendimiento** y velocidad, siendo ideales para proyectos que demandan un alto poder de procesamiento, como juegos en 3D, realidad aumentada o aplicaciones con animaciones complejas (EMMA, s.f.; Berna Network, s.f.). Su principal ventaja reside en el **acceso completo a todas las funcionalidades del *hardware*** y las API del dispositivo, permitiendo una integración fluida con la cámara, el GPS, los sensores biométricos, el *Bluetooth* y el funcionamiento *offline* (AWS, s.f.; TRBL Services, 2024). Sin embargo, su principal desventaja es el **alto costo y tiempo de desarrollo**, ya que se requiere un equipo y una base de código separada para cada plataforma, dificultando la portabilidad (Hiberus, 2022).

* 1. Aplicaciones Híbridas

Las **aplicaciones híbridas** buscan ofrecer un punto de equilibrio entre las apps nativas y la eficiencia del desarrollo web. Se desarrollan utilizando tecnologías web estándar (**HTML, CSS y JavaScript**), pero se envuelven en un **contenedor nativo** (como *WebView*) que les permite ser instaladas a través de las tiendas de aplicaciones (*App Store* o *Google Play*) y acceder, aunque de manera limitada o a través de *plugins*, a algunas funciones nativas del dispositivo (AWS, s.f.; EMMA, s.f.). Su principal atractivo es la **multiplataforma con código único**, lo que reduce significativamente la inversión inicial en tiempo y dinero, haciendo el mantenimiento y las actualizaciones menos complejos (TRBL Services, 2024; Hiberus, 2022). A pesar de su rapidez de desarrollo, presentan inconvenientes en la **experiencia de usuario** (ya que la UI puede no sentirse completamente nativa) y, fundamentalmente, en el **rendimiento**, que es inferior al de las aplicaciones nativas, especialmente en integraciones de *hardware* complejas (TRBL Services, 2024; Berna Network, s.f.).

* 1. Aplicaciones Web (Web Apps)

Finalmente, las **aplicaciones web** son sitios web adaptables que se ejecutan directamente en el navegador del dispositivo sin requerir descarga o instalación. Estas aplicaciones se construyen completamente con estándares web y son la opción **más económica y rápida** de implementar (EMMA, s.f.). Al estar atadas a las capacidades del navegador, su principal limitación es la **funcionalidad restringida**, ya que carecen de acceso a la mayoría de las características del *hardware* del dispositivo (cámara, agenda, etc.) y dependen, en gran medida, de una conexión a Internet (AWS, s.f.; Scribd, s.f.). Un subtipo avanzado son las *Progressive Web Apps* (**PWA**), que intentan subsanar estas carencias adoptando comportamientos nativos, como el funcionamiento sin conexión, pero aún sin la distribución plena en tiendas de aplicaciones (EMMA, s.f.).

## Arquitectura Android y componentes principales

El sistema operativo **Android** es una pila de *software* de código abierto basada en el **Kernel de Linux**, diseñada para una amplia variedad de dispositivos y factores de forma (AOSP, 2025). Su arquitectura está organizada en una estructura de capas jerárquica, lo que garantiza la seguridad, la gestión eficiente de recursos y una plataforma robusta para el desarrollo de aplicaciones.

* 1. **Estructura Jerárquica del SO Android**

La arquitectura del sistema Android se compone de varias capas, donde cada una se apoya en la capa inferior:

* + 1. **Kernel de Linux**: Es la base de la arquitectura. Se encarga de las funciones de bajo nivel del sistema, como la gestión de memoria, la seguridad (SELinux) y los drivers de hardware. Su uso permite a Android aprovechar las características clave de seguridad de Linux (Scribd, 2021).
    2. **Capa de Abstracción de Hardware (HAL)**: Esta capa proporciona interfaces estandarizadas que exponen las capacidades del hardware del dispositivo (ej., cámara, Bluetooth) al Framework de la API de Java. El HAL es fundamental porque permite a Android ser independiente de las implementaciones de los controladores de bajo nivel (AOSP, 2025; Scribd, 2021).
    3. **Capa de Abstracción de Hardware (HAL)**: Esta capa proporciona interfaces estandarizadas que exponen las capacidades del hardware del dispositivo (ej., cámara, Bluetooth) al Framework de la API de Java. El HAL es fundamental porque permite a Android ser independiente de las implementaciones de los controladores de bajo nivel (AOSP, 2025; Scribd, 2021).
    4. **Bibliotecas Nativas C/C++ y Android Runtime (ART)**: Las bibliotecas nativas, como SQLite (base de datos ligera), OpenGL ES (gráficos 3D) y la Bionic C Library, están disponibles para los desarrolladores a través del framework. Por otro lado, el ART es el entorno de ejecución que interpreta el código de bytes de la aplicación (DEX) y lo traduce a instrucciones específicas del procesador, un reemplazo moderno de la antigua máquina virtual Dalvik (AOSP, 2025; Develou, s.f.).
    5. **Framework de la API de Java (Application Framework)**: Es el conjunto de clases e interfaces Java que los desarrolladores utilizan para crear aplicaciones. Este framework incluye componentes modulares esenciales que gestionan el ciclo de vida, la comunicación y los recursos del sistema:
* **Activity Manager**: Administra el ciclo de vida de las aplicaciones y sus pantallas (Activities).

**Content Providers**: Permiten que las aplicaciones compartan y accedan a datos de otras aplicaciones (ej., contactos).

**Resource Manager**: Proporciona acceso a elementos no codificados (imágenes, layouts XML, cadenas de texto).

* + 1. **Capa de Aplicaciones**: Es la capa superior que engloba tanto las aplicaciones preinstaladas por el sistema (correo electrónico, calendario) como las instaladas por el usuario.

## Patrones de diseño (Clean Architecture)

La **Arquitectura Limpia (*Clean Architecture*)** es una filosofía de diseño de *software* popularizada por **Robert C. Martin** (conocido como *Uncle Bob*) en su libro del mismo nombre (Martin, 2018). Su objetivo central es construir sistemas que sean **independientes, mantenibles, flexibles y altamente testeables**, centrándose en las reglas de negocio esenciales del sistema y relegando los detalles de implementación (como la base de datos, la interfaz de usuario o los *frameworks* de terceros) a capas externas (Adictos al trabajo, 2018).

* 1. **Fundamentos y Principios de Inversión de Dependencias (DIP)**

**Clean Architecture** se estructura en un modelo de capas concéntricas, donde la regla de oro es la Regla de Dependencia: el código en los círculos internos no debe depender de nada en los círculos externos (Martin, 2018). Esto significa que la lógica de negocio central (el dominio) debe ser completamente ajena a la interfaz de usuario (UI), la base de datos o el framework web utilizado.

Para hacer cumplir esta regla, la arquitectura se basa en el Principio de Inversión de Dependencias (DIP) (parte de los principios SOLID). El DIP establece que los módulos de alto nivel (los casos de uso) no deben depender de módulos de bajo nivel (las bases de datos); ambos deben depender de abstracciones (interfaces) (Paradigmadigital, 2017). De este modo, la capa interna define las interfaces (puertos) y la capa externa proporciona las implementaciones concretas (adaptadores), asegurando que el flujo de control se invierta hacia el centro, manteniendo el núcleo del sistema independiente de la infraestructura (Adictos al trabajo, 2018).

* 1. **Capas de la Arquitectura limpia**

La arquitectura se divide en cuatro capas conceptuales, desde la más interna y abstracta, hasta la más externa y concreta:

* + 1. **Entidades (Entities)**: Contienen las reglas de negocio de la empresa. Son las estructuras de datos y métodos que representan el dominio de la aplicación y deben ser las menos propensas al cambio.
    2. **Casos de Uso (Use Cases)**: Contienen las reglas de negocio específicas de la aplicación. Orquestan el flujo de datos desde la interfaz de usuario a las entidades y viceversa, definiendo las funciones que la aplicación puede realizar (Code & Chill, 2023).
    3. **Adaptadores de Interfaz (Interface Adapters)**: Esta capa actúa como un puente. Incluye Presentadores, Controladores y Gateways que convierten el formato de datos entre el núcleo (Entidades y Casos de Uso) y las capas externas (UI y Base de Datos) (Code & Chill, 2023).
    4. **Frameworks y Drivers**: La capa más externa. Contiene todos los detalles de implementación, como los frameworks de UI (ej., Jetpack Compose en Android), las herramientas de base de datos (ej., Room) y cualquier otra librería de terceros.

# Descripción general y flujo de usuario

# Arquitectura técnica

Diagramas, estructura, clases principales

# Acceso a recursos nativos

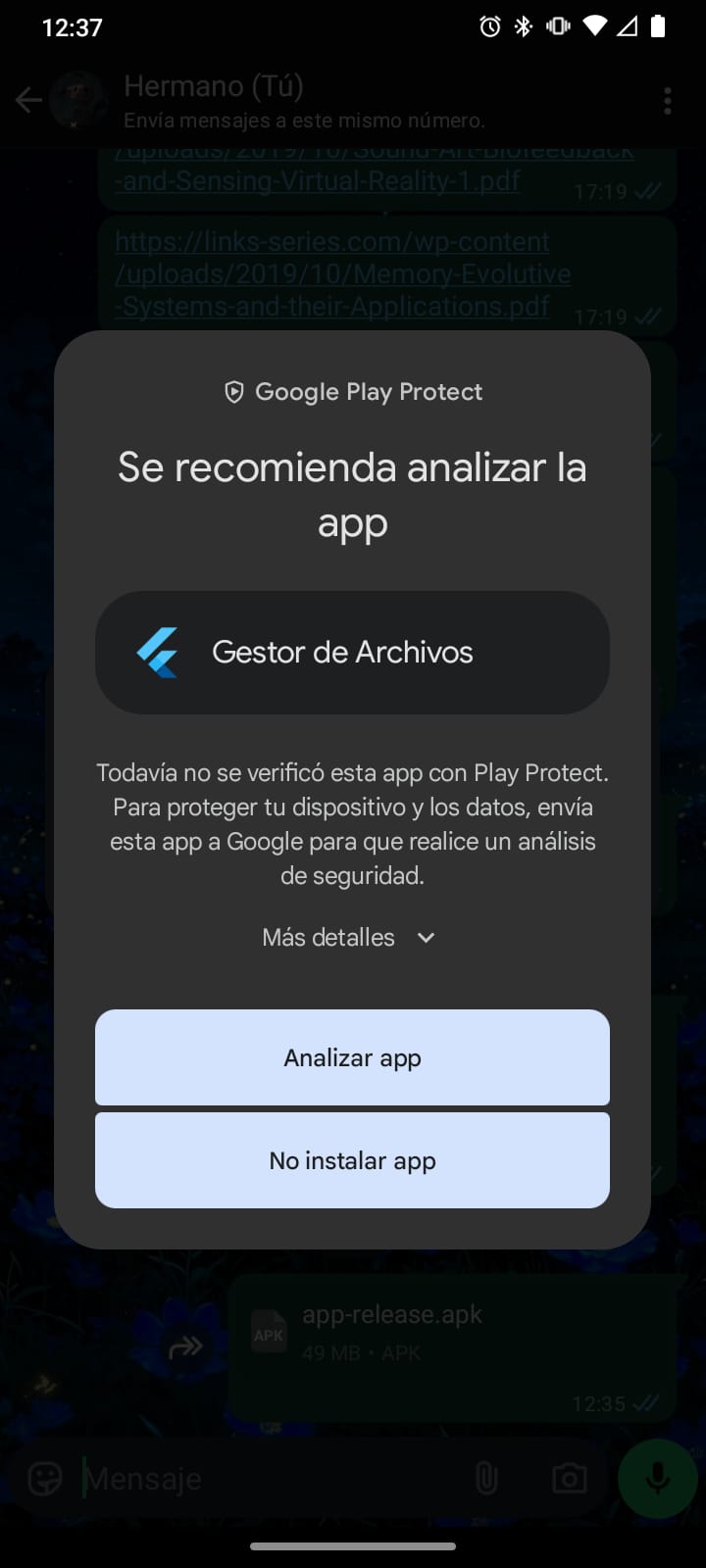
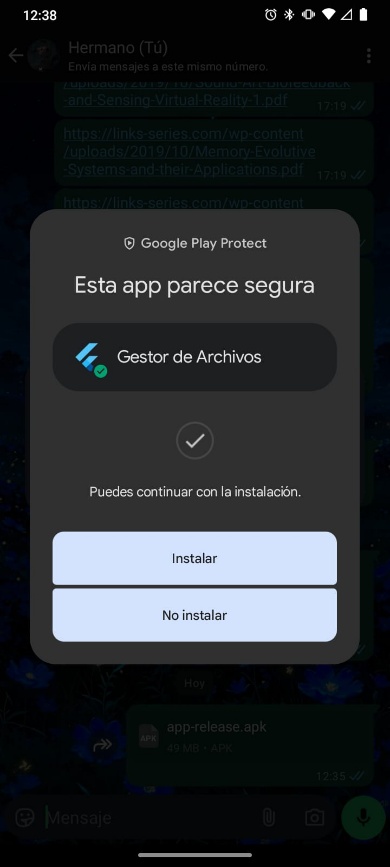
Implementación código ejemplo comentado

# 

# Interfaz de Usuario

Se muestran las capturas de pantalla de la aplicación, en donde al instalar la aplicación, lo primero que aparece es una advertencia por parte de Google Play Protect, además tenemos que darle permiso a la aplicación para que pueda acceder a los archivos de sistema.





A continuación, se presenta el cambio de temas en la interfaz de usuario, teniendo una combinación de 4 tonos (negro – azul, negro – guinda, blanco – azul, blanco – guinda).

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

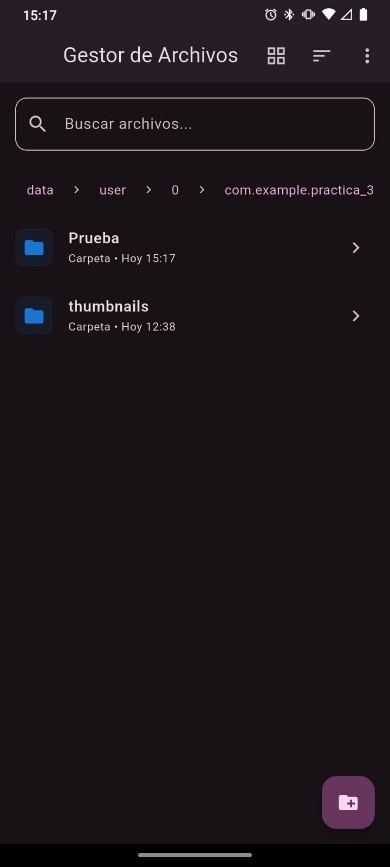
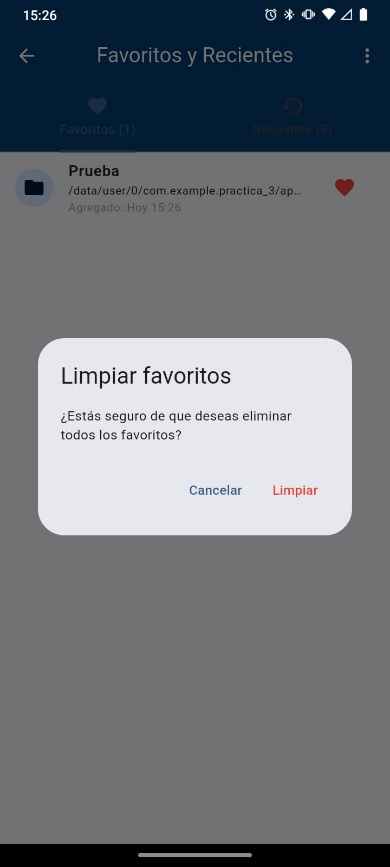
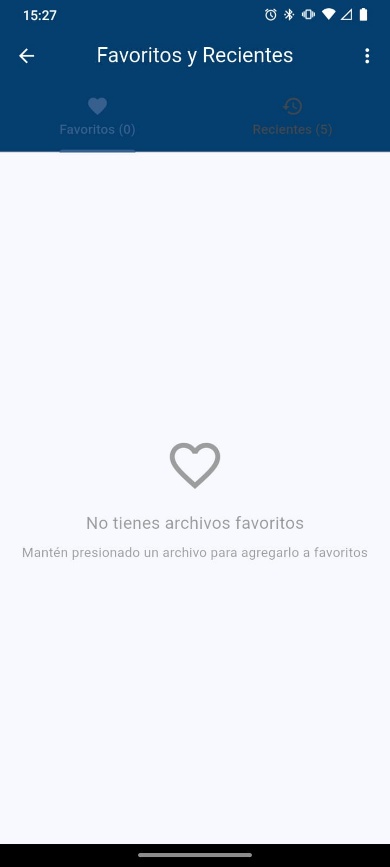
Observamos el estado inicial de la aplicación, en donde podemos crear una carpeta nueva y agregarla a favoritos.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.Una pantalla de un celular

El contenido generado por IA puede ser incorrecto.Imagen que contiene Patrón de fondo

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

# Persistencia de Datos

# Gestión de Permisos

# Conclusiones

# Bibliografía

* Adictos al trabajo. (2018, 23 de enero). *Revisión del nuevo libro de Uncle Bob: Clean Architecture*. Recuperado de <https://adictosaltrabajo.com/2018/01/23/revision-del-nuevo-libro-de-uncle-bob-clean-architecture/>
* AOSP. (2025, julio 24). *Descripción general de la arquitectura*. Android Open Source Project. Recuperado de <https://source.android.com/docs/core/architecture?hl=es-419>
* AWS. (s.f.). *Comparación entre aplicaciones web, aplicaciones nativas e Aplicaciones híbridas: diferencia entre los tipos de aplicaciones web y móviles*. Amazon Web Services.
* Berna Network. (s.f.). *Diferencias entre app híbrida y nativa*. Recuperado de <https://www.bernanetwork.com/diferencias-entre-app-hibrida-y-nativa>
* Code & Chill. (2023, 1 de agosto). *Introducción a las “Clean Architectures”*. Medium.
* Develou. (s.f.). *Aprendiendo Sobre La Arquitectura De Android*. Recuperado de <https://www.develou.com/aprendiendo-la-arquitectura-de-android/>
* EMMA. (s.f.). *Tipos de aplicaciones, características, ejemplos y comparativa*. Recuperado de <https://emma.io/blog/tipos-aplicaciones-caracteristicas-ejemplos/>
* Hiberus. (2022, 28 de mayo). *App híbrida o nativa: diferencias y ejemplos*. Blog de Hiberus.
* Martin, R. C. (2018). **Arquitectura Limpia: Guía para profesionales para la estructura y el diseño de *software***. Anaya Multimedia.
* Paradigmadigital. (2017, 22 de junio). *Introducción a los componentes de arquitectura Android: go Clean!*. Recuperado de <https://www.paradigmadigital.com/dev/introduccion-los-componentes-arquitectura-android-go-clean/>
* Scribd. (2021). *Arquitectura Android*. Recuperado de <https://es.scribd.com/document/517673979/ArquitecturaAndroid>
* Scribd. (s.f.). *Cuadro Comparativo Aplicaciones Nativas Web e Híbridas*. Recuperado de <https://es.scribd.com/document/398753567/Cuadro-Comparativo-Aplicaciones-Nativas-Web-e-Hibridas>
* TRBL Services. (2024, 5 de marzo). *Aplicaciones híbridas y nativas: qué son y diferencias entre ambas*. Recuperado de <https://trbl-services.eu/aplicaciones-hibridas-y-nativas-que-son-y-diferencias-entre-ambas/>