💎 Deluxe Shop

Aplicación Android segura y offline-first

Autor: **Daniel Imbert**

Fecha: **2025**

Versión: **1.0**

🧭 Descripción general

Deluxe Shop es una aplicación Android desarrollada en **Kotlin** que combina una arquitectura moderna, seguridad avanzada y sincronización offline-first.
Permite gestionar joyas y categorías, visualizar datos aún sin conexión, y mantener sincronizados los cambios con un backend en **Appwrite**.

🎯 Objetivos principales

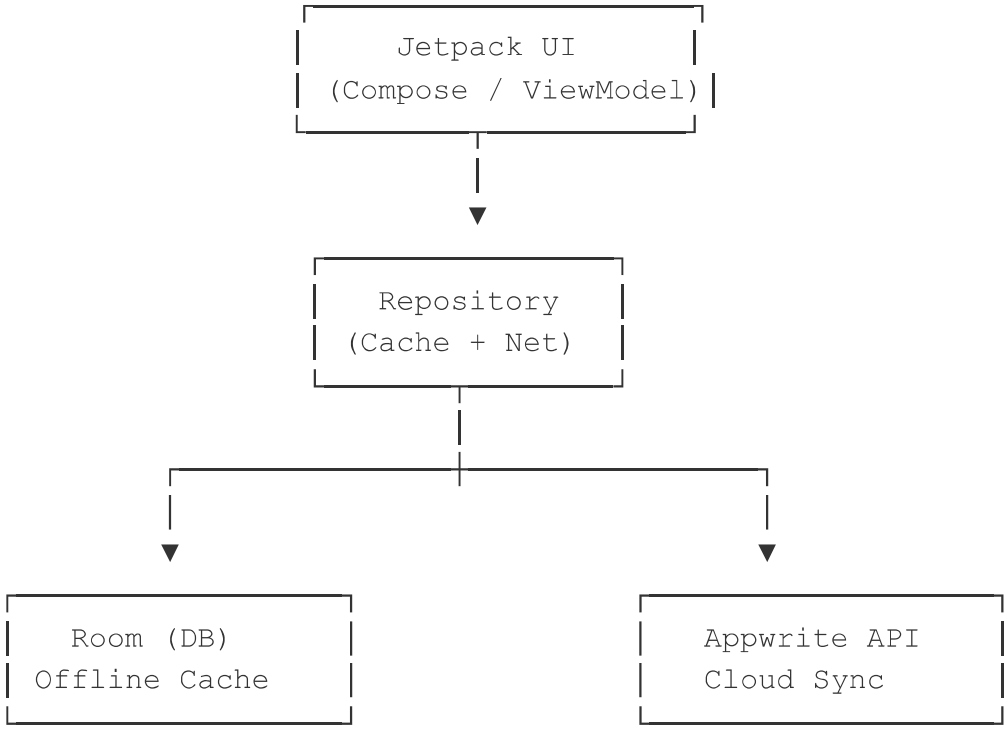
- Mantener la aplicación funcional sin conexión a Internet (offline-first).
- Asegurar las credenciales, claves y datos locales mediante cifrado.
- Cumplir principios **SOLID**, aplicando un enfoque **limpio y modular**.
- Facilitar la escalabilidad y el mantenimiento.

🧱 Tecnologías principales

Componente	Tecnología	Propósito
Lenguaje	Kotlin	Lógica principal y UI
UI	Jetpack Compose	Interfaz reactiva
Base de datos	Room + SQLCipher	Caché y cifrado local
Backend	Appwrite	Sincronización de datos
DI	Koin	Inyección de dependencias
Seguridad	KeyStore + AES/GCM	Protección de claves
Asincronía	Coroutines + Flow	Flujo reactivo de datos

🧩 Arquitectura y flujo de datos

Deluxe Shop implementa una arquitectura **MVVM + Repository + DI**.
Los datos se obtienen de un repositorio que combina el acceso **local (Room)** y **remoto (Appwrite)**, manteniendo la consistencia y el acceso inmediato.



🔍 Características clave

- **UI reactiva** con `StateFlow`.
- **Repositorio genérico** para minimizar código duplicado.
- **Sincronización automática** entre cache y servidor.
- **Capa de dominio separada** de la lógica de presentación.

🔐 Seguridad avanzada

Deluxe Shop implementa una **capa de seguridad reforzada** tanto en datos locales como en el manejo de credenciales.

Protección local

- Base de datos cifrada con **SQLCipher**.
- Llave AES generada y guardada en **Android KeyStore**.
- Claves cifradas almacenadas de forma segura en `SharedPreferences`.

Protección de API keys y credenciales

- Encriptación con **AES/GCM** antes de compilar.
- No se almacenan claves planas en el código fuente.
- Integración con **Appwrite Function** para obtener tokens temporales.

Fragmento de código clave

```
class SqlCipherKeyManager(
    private val sharedPreferences: SharedPreferences
) {
    private val keyStore = KeyStore.getInstance("AndroidKeySt

    init {
        generateKeystoreKeyIfNeeded()
        if (!sharedPreferences.contains("encrypted_key")) {
            generateAndEncryptSqlCipherKey()
        }
    }

    private fun generateAndEncryptSqlCipherKey() {
        val secretKey = getSecretKey("sqlcipher_keystore_key"
        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(Cipher.ENCRYPT_MODE, secretKey)
        val sqlCipherKey = ByteArray(32).also { SecureRandom(
        val encryptedKey = cipher.doFinal(sqlCipherKey)
        sharedPreferences.edit {
            putString("encrypted_key", Base64.encodeToString(
            putString("encryption_iv", Base64.encodeToString(
        }
    }
}
```

🗄️ Repositorios y sincronización

La app sigue el patrón offline-first, donde el repositorio controla tanto el acceso local como el remoto.

Interfaz genérica

```
interface BaseRepository<T> {

    // Flow siempre refleja la caché local
    val itemsFlow: Flow<List<T>>

    // Obtener por ID: cache + fallback network
    suspend fun getById(id: String, forceRefresh: Boolean = fa

    // Sincronización incremental
    suspend fun sync(): Result<Unit>
}
```

Implementaciones concretas

```
class CategoriasRepositoryImpl(
    private val categoriasDao: CategoriasDao,
    private val categoriasNet: CategoriasNetRepoImpl,
): BaseRepository<Categoria> {
    override val itemsFlow: Flow<List<Categoria>> get() = cate

    override suspend fun getById(id: String, forceRefresh: Boo
        if (forceRefresh) {
            val remoteItem = categoriasNet.getById(id).getOrTh
            categoriasDao.insert(remoteItem)
            remoteItem
        } else categoriasDao.getById(id)
    }

    override suspend fun sync(): Result<Unit> = runCatching {
        val remoteItems = categoriasNet.getAll().getOrThrow()
        categoriasDao.insertAll(remoteItems)
    }
}
```

```
class JoyaRepositoryImpl(
    private val joyasDao: DeluxeDao<Joya>,
```

```
    private val joyasNet: JoyaNetRepoImpl
): BaseRepository<Joya> {
    override val itemsFlow: Flow<List<Joya>> get() = joyasDao

    override suspend fun getById(id: String, forceRefresh: Bo
        if (forceRefresh) {
            val remoteItem = joyasNet.getById(id).getOrThrow(
            joyasDao.insert(remoteItem)
            remoteItem
        } else joyasDao.getById(id)
    }

    override suspend fun sync(): Result<Unit> = runCatching {
        val remoteItems = joyasNet.getAll().getOrThrow()
        joyasDao.insertAll(remoteItems)
    }
}
```

🌐 Funcionalidades principales

- Listado y detalle de joyas y categorías.
- CRUD local con sincronización a Appwrite.
- Caché persistente y cifrada.
- Modo offline automático.
- Envío de datos seguro.
- Manejo de notificaciones.
- Integración con WhatsApp para compartir carritos.

📦 Escalabilidad y mantenibilidad

- Arquitectura modular (UI, dominio, data, seguridad).
- Código desacoplado mediante interfaces y DI.
- Repositorios genéricos reutilizables.
- Seguridad y sincronización implementadas en capas independientes.

- Facilita testing unitario e integración CI/CD.

🔧 Mejoras futuras

- Pruebas unitarias con MockK y Turbine para flujos.
- Sincronización incremental basada en timestamps.
- Implementación de workers para tareas diferidas (WorkManager).
- Migración a arquitectura multi-módulo (feature modules).
- Dashboard web administrativo conectado al mismo backend Appwrite.

🧠 Reflexión profesional

Este proyecto demuestra competencias clave para roles Android:

- ✅ Arquitectura limpia y modular
- ✅ Seguridad avanzada de datos locales y remotos
- ✅ Manejo de sincronización y offline-first
- ✅ Buenas prácticas de inyección y asincronía
- ✅ Capacidad de integrar múltiples tecnologías coherentemente

"El verdadero valor de un desarrollador no está solo en escribir código, sino en diseñar sistemas seguros, mantenibles y escalables."