Informe: Aplicación de Monitorización Financiera Multiplataforma en Compose Multiplatform

1. Introducción

Este informe presenta el diseño y planificación de una aplicación multiplataforma desarrollada con **Compose Multiplatform**. La app permitirá monitorizar precios de activos financieros (acciones, índices y criptomonedas) en tiempo real, empleando únicamente **APIs gratuitas** y aplicando principios de **Clean Architecture**.

---

2. MVP y Funcionalidades Clave

1. **Listado de activos**: visualización de ticker, nombre, precio actual y variación porcentual (24 h).
2. **Detalle de activo**: gráfico de líneas (1D, 1W, 1M), volumen, máximos/mínimos y capitalización.
3. **Favoritos**: gestionar lista de activos preferidos.
4. **Alertas básicas**: notificaciones locales cuando se crucen umbrales de precio.
5. **Modo offline**: mostrar datos cacheados en ausencia de conectividad.

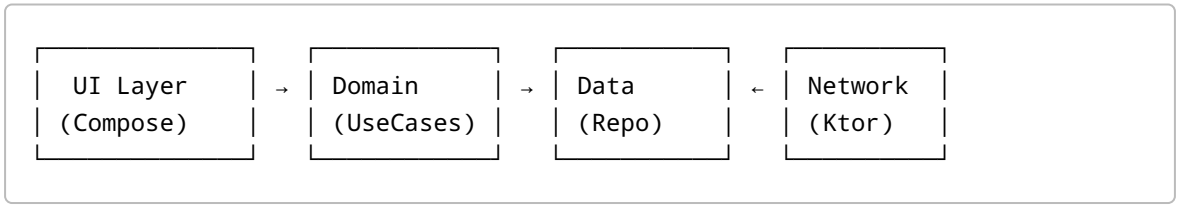
---

3. APIs Gratuitas Recomendadas

API	Cobertura	Límite gratuito	Endpoint ejemplo
CoinGecko	Criptomonedas	60 req/min ilim.	`/coins/markets`
Finnhub	Acciones, índices	60 req/min	`/quote`
Alpha Vantage	Acciones, forex	5 req/min, 500 req/día	`TIME_SERIES_DAILY_ADJUSTED`

---

4. Arquitectura (Clean Architecture)



- **UI Layer**: Composables, ViewModels con `StateFlow`, navegación.
- **Domain Layer**: entidades (`Asset`, `PricePoint`, `Alert`), casos de uso (`GetAssetsListUseCase`, etc.).
- **Data Layer**: repositorios, `RemoteDataSource` (Ktor), `LocalDataSource` (SQLDelight).
- **Network**: Ktor Client con serialización, interceptores.
- **Cache/Offline**: SQLDelight para tablas de `assets`, `price_points`, `alerts`.

---

5. Roadmap de Implementación

1. **Inicializar proyecto**: Compose MP targets Android, Desktop y web.
2. **Configurar Ktor Client**: prueba llamada a CoinGecko.
3. **Definir modelos de datos** y configurar Kotlinx.serialization.
4. **Implementar RemoteDataSource** para listado y detalle de activos.
5. **Agregar SQLDelight** y definir esquema de base de datos.
6. **Crear Repository** con estrategia "network first, fallback cache".
7. **ViewModel y UseCase** para pantalla de listado (`StateFlow<List<Asset>>`).
8. **Pantalla de listado**: `LazyColumn`, tarjetas, pull-to-refresh.
9. **Detalle de activo**: gráfico de líneas con Compose Chart API.
10. **Favoritos**: toggle y sección dedicada.
11. **Alertas en background**: Worker API en Android, cron en Desktop.
12. **Notificaciones locales** en cada plataforma.
13. **Modo offline**: datos cacheados y alerta visual.
14. **Tests unitarios** con MockK.
15. **Documentación**: arquitectura, README y diagramas.

---

6. Buenas Prácticas

- Inyección de dependencias con **Koin** o **Hilt**.
- Modularización en módulos `:core`, `:data`, `:domain`, `:ui`.
- Uso de **Coroutines** y **Flow** para concurrencia.
- Nombres de paquetes y clases claros.
- Commits atómicos y descriptivos.
- **CI/CD** con GitHub Actions para compilaciones y tests en múltiples plataformas.

---

7. Conclusión

Este plan de desarrollo proporciona una base sólida para crear una aplicación de monitorización financiera profesional. El uso de Compose Multiplatform junto a Clean Architecture y APIs gratuitas garantiza un proyecto estructurado, escalable y preparado para mostrar en tu portfolio.