MARKETPLACE APP

Prueba de Concepto (POC) - Desarrollo de Aplicaciones Móviles

https://github.com/maridilo/MarketplaceApp.git

AUTORES:
MARIA DIAZ – HEREDERO LOPEZ
CINTIA SANTILLAN GARCIA

Asignatura: PROGRAMACIÓN DIRIGIDA POR EVENTOS

Contenido

1. Resumen del Documento

Detallamos el desarrollo de una Prueba de Concepto (POC) para una aplicación Android de comercio electrónico, Marketplace App. El objetivo principal ha sido implementar una solución cliente funcional que permita la gestión de usuarios, exploración de catálogo y simulación de compra, aplicando los conocimientos de la asignatura.

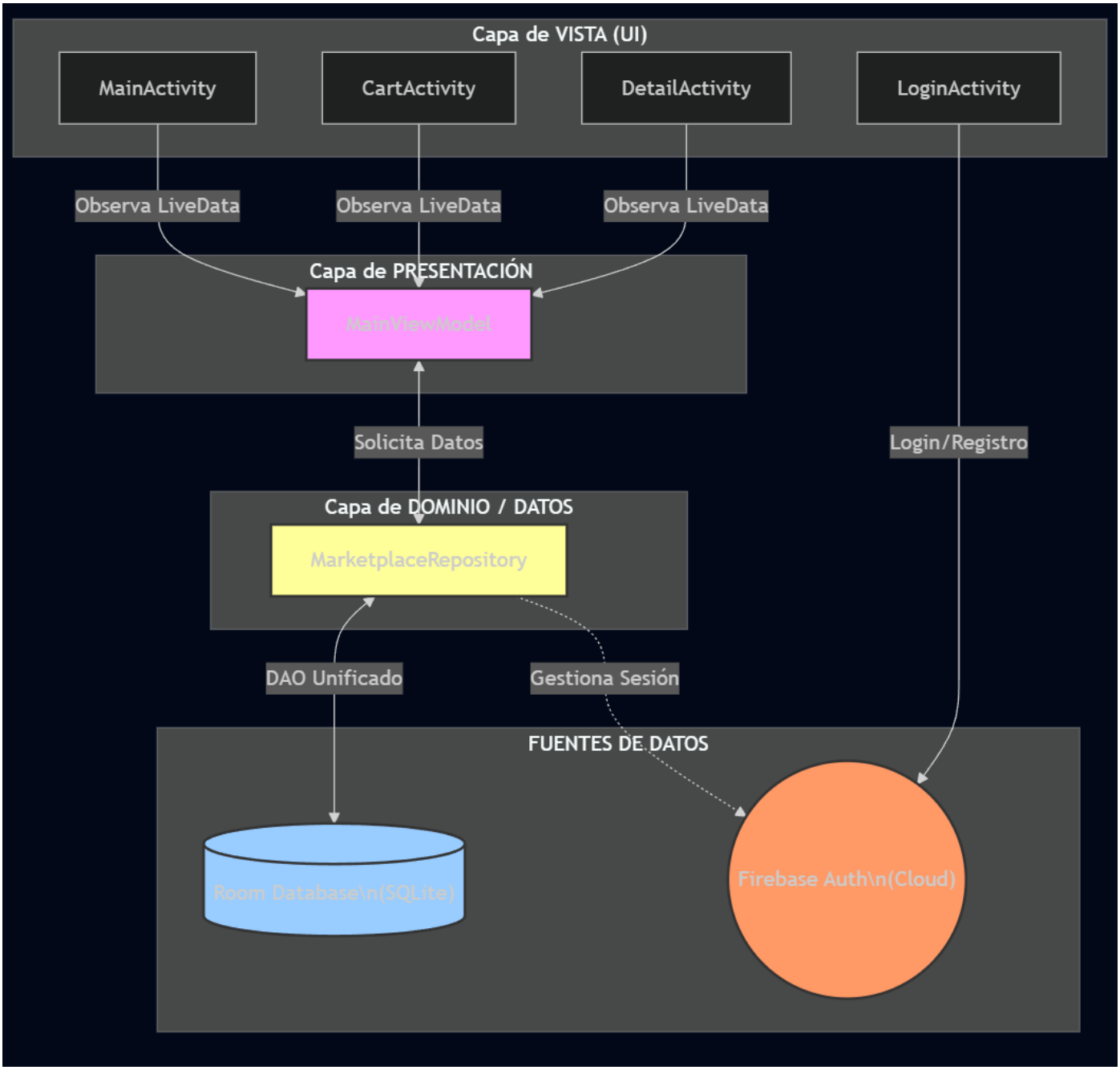
El proyecto se ha desarrollado siguiendo el patrón de arquitectura MVVM (Model-View-ViewModel), integrando autenticación Firebase para la gestión de identidades en la nube y Room Database para la persistencia de datos en local, asegurando así una experiencia de usuario fluida y con soporte offline.

2. Diagrama de Arquitectura

Se ha elegido una arquitectura **MVVM (Model-View-ViewModel)** acompañada del patrón **Repository** para desacoplar la lógica de negocio de la interfaz de usuario, facilitando el mantenimiento y la escalabilidad.

Componentes del Sistema:

- **Vista (UI Layer):**
    - Compuesta por Activities (MainActivity, CartActivity, LoginActivity, etc.) y Fragments (InfoDialogFragment).
    - **Responsabilidad:** Mostrar datos al usuario y capturar eventos (clicks, input). No contiene lógica de negocio.
    - Se comunica con el ViewModel observando objetos LiveData.
- **ViewModel (Presentation Layer):**
    - Clase: MainViewModel.
    - **Responsabilidad:** Mantiene el estado de la UI y sobrevive a cambios de configuración. Actúa como intermediario entre la Vista y el Repositorio.
- **Repositorio (Domain Layer):**
    - Clase: MarketplaceRepository.
    - **Responsabilidad:** Decide si obtener los datos de la base de datos local o de la red (simulada en este caso para productos).
- **Fuentes de Datos (Data Layer):**
    - **Local (Room):** AppDatabase y ProductDao. Se ha implementado un DAO unificado que centraliza todas las operaciones de persistencia (catálogo, gestión del carrito y registro de pedidos) para simplificar la capa de datos.
    - **Remota (Firebase):** FirebaseAuth. Gestiona el inicio de sesión y registro de usuarios.


Capa de VISTA (UI)
MainActivity
CartActivity
DetailActivity
LoginActivity
Observa LiveData
Observa LiveData
Observa LiveData
Capa de PRESENTACIÓN
MainViewModel
Solicita Datos
Login/Registro
Capa de DOMINIO / DATOS
MarketplaceRepository
DAO Unificado
Gestiona Sesión
FUENTES DE DATOS
Room Database\n(SQLite)
Firebase Auth\n(Cloud)

3. Descripción del Trabajo Realizado

3.1 Funcionalidades Incluidas

El proyecto cubre todos los **requisitos funcionales mínimos** exigidos:

- **Autenticación y Cuenta:**
  - Registro de nuevos usuarios mediante correo y contraseña.
  - Inicio de sesión seguro con validación de credenciales a través de FirebaseAuth.
  - Persistencia de sesión (Auto-login al abrir la app si el usuario ya estaba autenticado).
  - Pantalla de Perfil (ProfileActivity) con visualización del correo del usuario y funcionalidad de "Cerrar Sesión" con diálogo de confirmación.
- **Catálogo y Búsqueda:**
  - Visualización de productos en un RecyclerView con diseño de cuadrícula (GridLayoutManager).
  - Carga asíncrona de imágenes mediante la librería Glide.
  - Barra de búsqueda (SearchView) en la Toolbar que filtra los productos en tiempo real consultando la base de datos local.
- **Carrito de la Compra:**
  - Persistencia local mediante Room: los productos añadidos permanecen en el carrito, aunque se cierre la aplicación.
  - Gestión de ítems: Posibilidad de incrementar/decrementar cantidad y eliminar productos.
  - Cálculo dinámico del total a pagar.
  - Control de estado: Visualización de un "Empty State" (mensaje amigable) cuando el carrito está vacío.
- **Pedidos e Historial:**
  - Simulación de proceso de Checkout: Al confirmar la compra, se vacía el carrito y se genera un registro persistente en la tabla de pedidos.
  - Visualización de listado de pedidos históricos con fecha, estado y monto total.

3.2 Funcionalidades Adicionales y Mejoras (Extras)

Para mejorar la calidad de la POC y la experiencia de usuario (UX), se han implementado las siguientes características no obligatorias:

1. **DialogFragments Personalizados:** En lugar de usar Toasts genéricos, se han implementado diálogos modales (InfoDialogFragment) con iconos animados y colores semánticos (Verde para éxito, Rojo para error/borrado) que cumplen con el requisito de uso de Fragments.
2. **Empty States Reactivos:** Implementación lógica en la UI para alternar automáticamente entre las listas (RecyclerView) y mensajes de aviso cuando no hay datos (en Historial y Carrito).

4. Explicación de las Decisiones Tomadas

Durante el desarrollo se tomaron las siguientes decisiones técnicas:

1. **Uso de Room para el Catálogo y Carrito:** Aunque Firebase Realtime Database permitiría sincronización en la nube, el enunciado solicitaba explícitamente persistencia local con ROOM. Se decidió cargar un set de datos inicial ("Mock Data") en la base de datos local al arrancar la app por primera vez, permitiendo que la aplicación funcione completamente offline excepto para el login.

2. **Navegación basada en Activities vs Navigation Component:** Se optó por utilizar **Activities** independientes (MainActivity, CartActivity, DetailActivity) gestionadas mediante Intents explícitos.
   - Justificación: Esta estructura separa claramente los ciclos de vida de cada funcionalidad principal, evitando la complejidad de gestionar una sola actividad ("Single Activity Architecture") para una POC de este tamaño.
   - Cumplimiento de Fragments: El requisito de usar Fragments se hizo mediante la implementación de DialogFragments para las notificaciones modales.

3. **Programación Reactiva con LiveData:** Se decidió utilizar LiveData en los ViewModels para exponer los datos a la Vista.
   - Justificación: Esto permite que la UI reaccione automáticamente a los cambios en la base de datos (por ejemplo, al borrar un ítem del carrito, el precio total se recalcula solo).

5. Conclusiones

5.1 Resumen Final

El proyecto ha resultado en una aplicación funcional, robusta y estéticamente cuidada que cumple con todos los objetivos propuestos. La arquitectura MVVM mantiene el código organizado, facilitando la implementación de nuevas características como el buscador o el historial sin afectar a la lógica existente.

5.2 Principales Problemas Encontrados

1. **Conflictos en ViewBinding/Adapters:** Inicialmente, hubo problemas donde las listas aparecían vacías. Se diagnosticó que se estaban instanciando múltiples adaptadores en el MainActivity. La solución fue centralizar la instancia del adaptador y usar submitList para actualizar los datos.
2. **Integridad de Base de Datos:** Al modificar las entidades (Product, Order) durante el desarrollo, Room lanzaba excepciones de migración. Se solucionó incrementando la versión de la base de datos y utilizando. fallbackToDestructiveMigration() para el entorno de desarrollo.
3. **Diseño de Interfaz:** Ajustar los layouts para que se vieran bien con y sin datos (Empty States) requirió el uso de FrameLayouts y lógica de visibilidad dinámica en los observadores del ViewModel.

5.3 Evaluación y Aspectos de Mejora

La aplicación actual es una base sólida para un producto real. Como mejoras para una versión futura (v2.0) se proponen:

- Implementar una pasarela de pago real (Stripe o PayPal).
- Sincronizar el carrito con la nube (Firestore) para que el usuario pueda acceder a él desde diferentes dispositivos.
- Añadir animaciones de transición compartida (Shared Elements) al entrar en el detalle del producto para mejorar la fluidez visual.