PDE-Marketplace App

Realizado por: Álvaro Martín Romero y Pablo García Moya

Índice

1. Breve Resumen del proyecto

PDE-MarketPlace es una aplicación Android en Java que **implementa una prueba de concepto de un marketplace centrado en el flujo principal de compra**: autenticación, exploración del catálogo, búsqueda, detalle de producto, carrito y confirmación de pedido, con historial consultable.

El proyecto se ha **diseñado para demostrar programación dirigida por eventos** en Android (listeners, callbacks y navegación entre Activities), **apoyándose en componentes estándar como RecyclerView** y en servicios externos como **Firebase Authentication** para la gestión de usuarios.
La solución **prioriza claridad y modularidad, separando presentación (UI), lógica de navegación/orquestación y acceso a datos**, con el objetivo de facilitar futuras extensiones como persistencia avanzada, notificaciones, roles o integración con un backend real.

2. Alcance, requisitos y supuestos

La POC cubre el proceso esencial de un marketplace: registro/inicio de sesión, visualización de un catálogo, búsqueda de productos, gestión de carrito y confirmación de pedidos (sin pasarela bancaria), además de consulta de pedidos realizados.

Requisitos del enunciado (trazabilidad)

- Autenticación: implementada mediante FirebaseAuth (registro e inicio de sesión).
- Cuenta: pantalla de cuenta con información del usuario y cierre de sesión.
- Búsqueda de productos: búsqueda en catálogo mediante barra de búsqueda (filtrado por nombre).
- Visualización de pedidos realizados: historial de pedidos confirmados.
- Carrito de la compra: añadir productos, ver total y confirmar pedido (sin pago).
- Componentes: RecyclerView para catálogo/carrito/historial, navegación entre Activities, integración con FirebaseAuth y persistencia local (Room) para entidades definidas en el proyecto.

Supuestos

- No se implementa pago real por estar fuera del alcance de la POC.
- El catálogo se alimenta con datos de ejemplo; en una versión de producción se obtendría desde un backend.

3. Análisis de producto: problema, usuarios y propuesta de valor

Problema
Los usuarios necesitan un proceso simple para explorar productos, comparar rápidamente (precio y descripción), añadir al carrito y confirmar una compra, con trazabilidad mínima (historial de pedidos).

Usuarios objetivo (clientes)

- Usuario final: compra y consulta pedidos.
- Administrador (futuro): gestión de catálogo, precios y stock.

Propuesta de valor

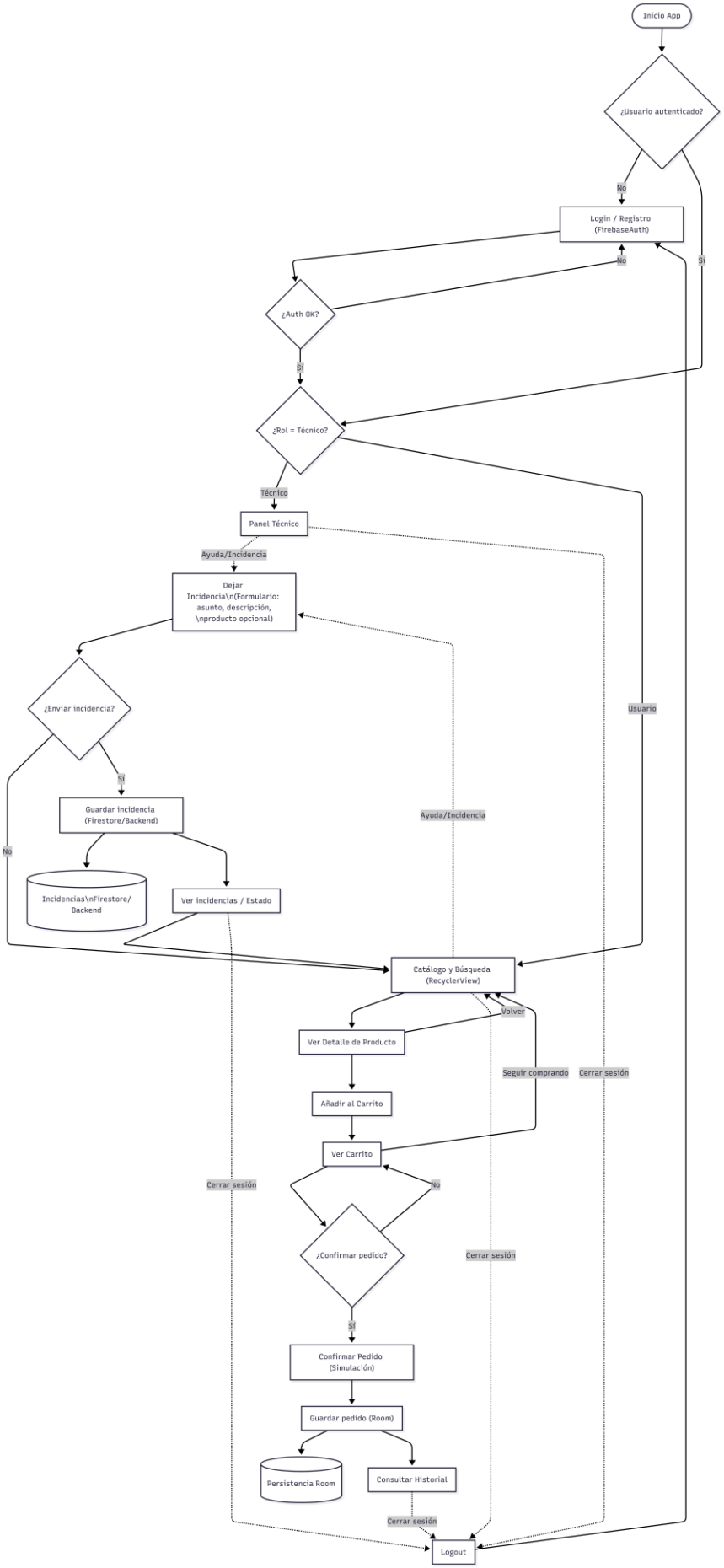
- Flujo de compra rápido (mínimos pasos).
- Interfaz centrada en catálogo + búsqueda.
- Persistencia de sesión y acceso a cuenta.

4. Diagrama de arquitectura

Nuestra aplicación sigue una arquitectura modular dividida en tres capas fundamentales para garantizar la escalabilidad y el mantenimiento:

- **Vista (Capa de Presentación):** Responsable de renderizar la interfaz gráfica y mostrar los datos al usuario mediante *Activities* y *Adapters* (para listados). Su función principal es capturar las interacciones y eventos del usuario (como clics o formularios) para delegarlos a la lógica de control.
- **Lógica de Control (Orquestación):** Actúa como intermediario entre la vista y los datos. Se encarga de procesar los eventos de navegación, gestionar el flujo de la aplicación y coordinar las peticiones de información a través de la clase **Repository**, asegurando que la interfaz siempre refleje el estado actual de la aplicación.
- **Modelo (Datos y Negocio):** Es el núcleo de la información. Esta capa integra las definiciones de objetos (**POJOs** en el paquete model), la persistencia de datos local mediante **Room** (paquete data) y la gestión de usuarios en la nube con **Firebase** (auth), centralizando todas las reglas de negocio y sincronización.

Diagrama de flujo de la aplicación


Inicio App
¿Usuario autenticado?
No
Login / Registro (FirebaseAuth)
No
Sí
¿Auth OK?
Sí
¿Rol = Técnico?
Técnico
Panel Técnico
Ayuda/Incidencia
Dejar Incidencia\n(Formulario: asunto, descripción, \nproducto opcional)
¿Enviar incidencia?
Sí
Guardar incidencia (Firestore/Backend)
No
Incidencias\nFirestore/ Backend
Ver incidencias / Estado
Usuario
Ayuda/Incidencia
Catálogo y Búsqueda (RecyclerView)
Volver
Ver Detalle de Producto
Seguir comprando
Cerrar sesión
Añadir al Carrito
Ver Carrito
Cerrar sesión
No
¿Confirmar pedido?
Cerrar sesión
Sí
Confirmar Pedido (Simulación)
Guardar pedido (Room)
Persistencia Room
Consultar Historial
Cerrar sesión
Logout

5. Backend realizado

El backend gestiona la lógica y los datos "detrás" de la app, sin depender de la interfaz. Se encarga de autenticación, almacenamiento/recuperación de información (productos, carrito, pedidos) y reglas de seguridad. En nuestra app se materializa como servicios externos (FirebaseAuth) y una capa de datos local (Room + Repository) que soporta el funcionamiento del marketplace.

5.1. Backend externo: Firebase Authentication (paquete auth)

Este paquete centraliza las llamadas a FirebaseAuth y se encarga además (mediante las pertinentes llamadas a Firebase) de:

- Registrar usuarios nuevos
- Permitir el inicio de sesión a usuarios ya existentes en la base de datos
- Permitir el cierre de sesión a los usuarios ya autentificados

5.2. Backend local: persistencia Room (paquete data)

La implementación de una base de datos Room permite **modo offline**, persistencia y recuperación de estado. Con el uso de Repository permite mantener una arquitectura por capas

Desglose de las clases del paquete:

AppDatabase	Se encarga de crear la base de datos Room
CartItemDao	Realiza operaciones sobre los productos que se añaden al carrito (añadir más, borrarlos, comprarlos y añadirlos al historial…)
OrderDao	Realiza las operaciones pertinentes sobre el pedido (borrar los distintos productos del carrito, realizar el pago y enviar el pedido al historial…)
Repository	Capa intermedia entre la base de datos y la interfaz del usuario

5.3. Modelos de datos (paquete model)

Se encargan de representar la información principal de la aplicación (como producto, artículo del carrito y pedido).
Define qué campos existen, cómo se agrupan y cómo se intercambian entre pantallas y la capa de datos (Room/Repository).

Sirven para mantener consistencia entre la UI y el almacenamiento, evitando duplicidades y errores al manejar la información.

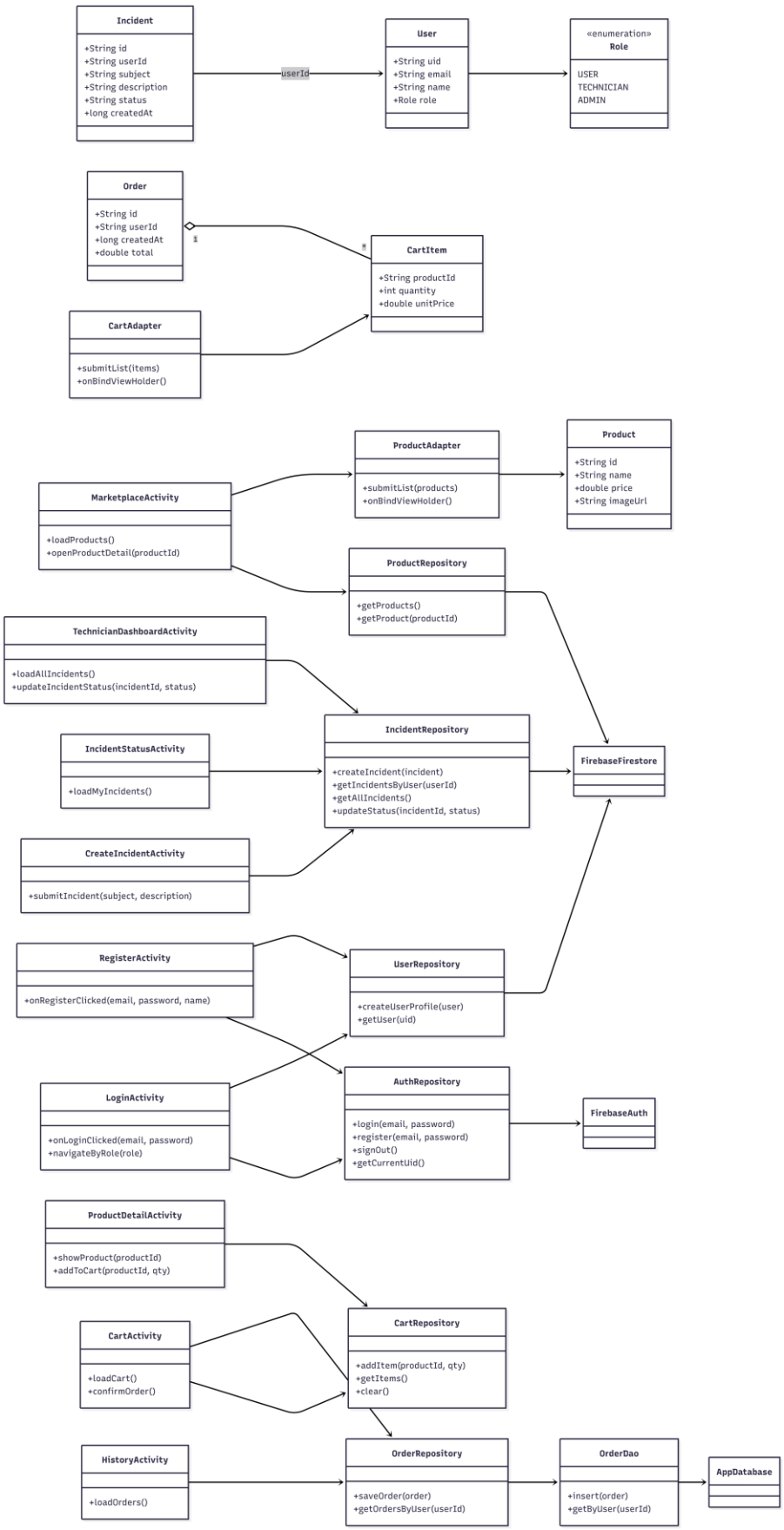
Desglose de las clases del paquete

CartItem (Entidad Room)	Define la entidad item del carrito (los productos que se añaden al carrito para realizar un pedido mas adelante)
Order (Entidad Room)	Define la entidad pedido la cual aparecerá en el historial de pedidos
Product	Define la entidad producto que se encuentra en la Marketplace (no es una entidad Room)
User	Define la entidad asociada al usuario con datos como la localización, email, teléfono…

5.4. Control de sesión MainActivity

La clase MainActivity se encarga de la redirección según si el usuario esta o no autenticado, si esta autenticado redirige a la actividad principal del Marketplace, si no lo mando a la actividad de inicio de sesión

5.5. Diagrama de clases



6.Gestión de roles (técnico/usuario)

El rol **Técnico** representa un perfil interno orientado a **operación y soporte** dentro del marketplace. Su objetivo es permitir tareas de mantenimiento que no corresponden al usuario final, principalmente:

- **Gestión de incidencias** reportadas por usuarios (visualizar, marcar como resueltas y eliminar).
- **Administración del catálogo** (crear productos; y, en diseño, facilitar mantenimiento del contenido).

Activación del rol

En esta app, el rol **no se almacena** como atributo role en el modelo User. En su lugar, la identificación es **implícita por dominio de correo**:

- **Clase:** auth/LoginActivity
- **Criterio:** email.endsWith("@tecnico.com")
- **Efecto:** tras login, se redirige a:
    - ui/TechnicianActivity si es técnico
    - ui/HomeActivity si es usuario estándar

Implicación técnica: es un mecanismo de "routing por email" pero **no es autorización fuerte**

Capacidades del Técnico (qué puede hacer)

Panel Técnico (punto de entrada)

Clase: ui/TechnicianActivity, Funciona como "hub" del rol y expone:

- Acceso a **catálogo técnico** (RecyclerView)
- Acción de **crear producto** (CreateProductActivity)
- Acceso a **incidencias** (TechnicianIncidentsActivity)
- Acceso a **perfil técnico** (TechnicianProfileActivity)

Gestión de incidencias

Creación de incidencias (usuario estándar)

La clase AccountActivity construye un incidente con email de usuario, mensaje y timestamp y lo envia a IncidentRepository.addIncident(incident) y muestra un mensaje por pantalla que muestra: "incidencia enviada al técnico"

Visualización y operaciones (técnico)

TechnicianIncidentsActivity se encarga de renderizar una lista con IncidentAdapter usando IncidentRepository.getIncidents()

IncidentAdapter puede marcar como resulta la incidencia recibida y eliminar dicha incidencia

Persistencia actual de incidencias (limitación del diseño)

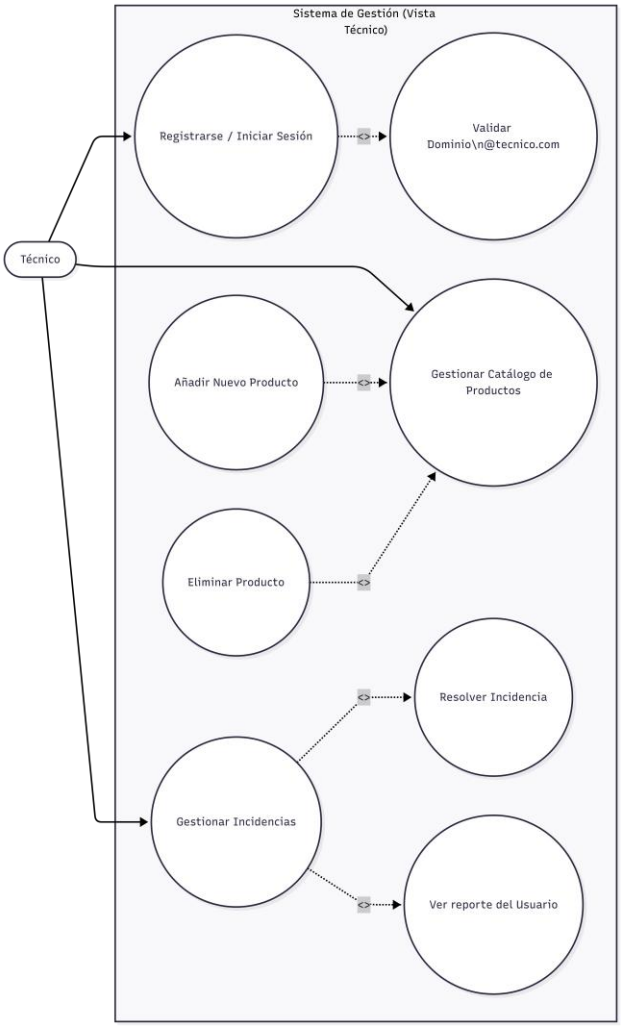
IncidentRepository es una lista **estática en memoria**.

- Consecuencia:
    - No persiste si se cierra la app.
    - No sincroniza entre dispositivos/usuarios (no hay BD remota para incidencias).

Administración de catálogo

CreateProductActivity: Permite crear un producto y llama a: ProductRepository.addProduct(name, desc, price)

Diagrama de flujo del rol de técnico



7.Descripción del trabajo realizado

Funcionalidades Implementadas

- Registro de usuarios mediante Firebase Authentication
- Inicio de sesión seguro
- Persistencia de sesión
- Visualización de productos/películas mediante RecyclerView
- Adaptadores dinámicos (Grid / Linear)
- Navegación entre actividades (Login → Marketplace → Cesta)
- Menú superior con opciones (Settings / Logout)
- Gestión avanzada de roles (técnico/usuario)

Funcionalidades no implementadas

- Persistencia compleja de productos creados por usuarios
- Pagos o transacciones reales
- Notificaciones push

Funcionalidades adicionales realizadas

- Control de errores de login
- Validación de campos
- Interfaz adaptable

8.Explicación de las decisiones tomadas

Firebase: elegido por su aportación de servicios en la nube como bases de datos en tiempo real, autenticación y hosting

Recyclerview: optimiza la memoria reutilizando las vistas de elementos que salen en pantalla permite actualizar la lista de forma **reactiva** mediante adaptadores que refrescan automáticamente la interfaz cada vez que los datos de origen cambian.

Separación por paquetes: mejora la claridad del proyecto al indicar claramente donde se encuentran los modelos (La información que mostramos), los adaptadores (Los que conectan la información con el diseño) y las actividades (donde se muestran los datos de los modelos).

Eventos explícitos (onClick, listeners): sirven para registrar las peticiones del usuario (onClick) y para escuchar activamente si el usuario realiza alguna petición (Listeners) y así agilizar la interacción con el usuario.

Evitar lógica pesada en XML: En nuestro proyecto mantenemos la lógica de eventos en Java para que sea capaz de gestionar las interacciones de los usuarios.

Gestión avanzada de roles (Técnico/usuario): Para una gestión más sencilla hemos decidido añadir un sistema de roles con tecnico y usuario, el tecnico tiene la posibilidad de gestionar los productos que se encuentran presentes en el Marketplace y de resolver incidencias

9.Conclusiones

En resumen, la aplicación diseñada funciona de forma fluida y organizada, permite a los usuarios registrarse, cerrar sesión, navegar por la aplicación añadir artículos a la cesta y finalmente pagar por esos artículos añadidos a un pedido

Principales problemas encontrados

- Configuración inicial de Firebase
- Errores de dependencias (Gradle)
- Gestión del ciclo de vida de Activities
- Problemas de sincronización de sesión
- Internacionalización y recursos
- Gestion de roles

Comentarios personales y aspectos que mejorar

- Se podría mejorar:
- Aplicando MVVM: **M**odel (Modelo), **V**iew (Vista) y **V**iew**M**odel
- Aplicando LiveData
- Mejorando UX/UI
- Implementando persistencia avanzada y roles
- Añadiendo tests
- Internalización de la aplicación
- Reforzando la seguridad de la aplicación frente a autentificaciones ilegitimas