Personal Expense Tracker

1. Introduction

Managing personal finances effectively is crucial for financial stability, and the **Expense Tracker App** provides a seamless solution for tracking and analyzing income and expenses. Designed for **Android** using **Kotlin** and **Material3**, the app offers a modern and intuitive interface that enables users to effortlessly record their financial transactions.

With secure **Google OAuth authentication**, users can sign in and access their data across devices, ensuring a seamless experience. The app supports **local storage with SQLite**, allowing users to manage expenses and incomes offline while providing an option to **sync data with Firebase**, ensuring data security and accessibility.

A comprehensive **dashboard** presents a clear financial overview, displaying **net expenses, net income, and net balance** for informed decision-making.

With a robust cloud backup mechanism through **Firebase Storage**, financial data remains safe and retrievable whenever needed. The **Expense Tracker App** is designed to simplify financial management, making it an essential tool for individuals looking to gain better control over their personal finances.

2. Architecture of the Expense Tracker App

The Personal Expense Tracker app follows a well-structured **Clean Architecture** approach, ensuring modularity, maintainability, and scalability. It is developed using **Kotlin** for Android and leverages **Material3, Firebase, SQLite, and Firebase Authentication**. This architecture promotes clear separation of concerns, making the app robust and easier to extend.

2.1 Layered Architecture

The architecture is divided into three key layers:

1. **Presentation Layer –** Handles UI and user interactions.
2. **Domain Layer –** Contains business logic and use cases.
3. **Data Layer –** Manages data sources, including local (SQLite) and remote (Firestore).

Each layer interacts only with its adjacent layer, ensuring loose coupling and better testability.

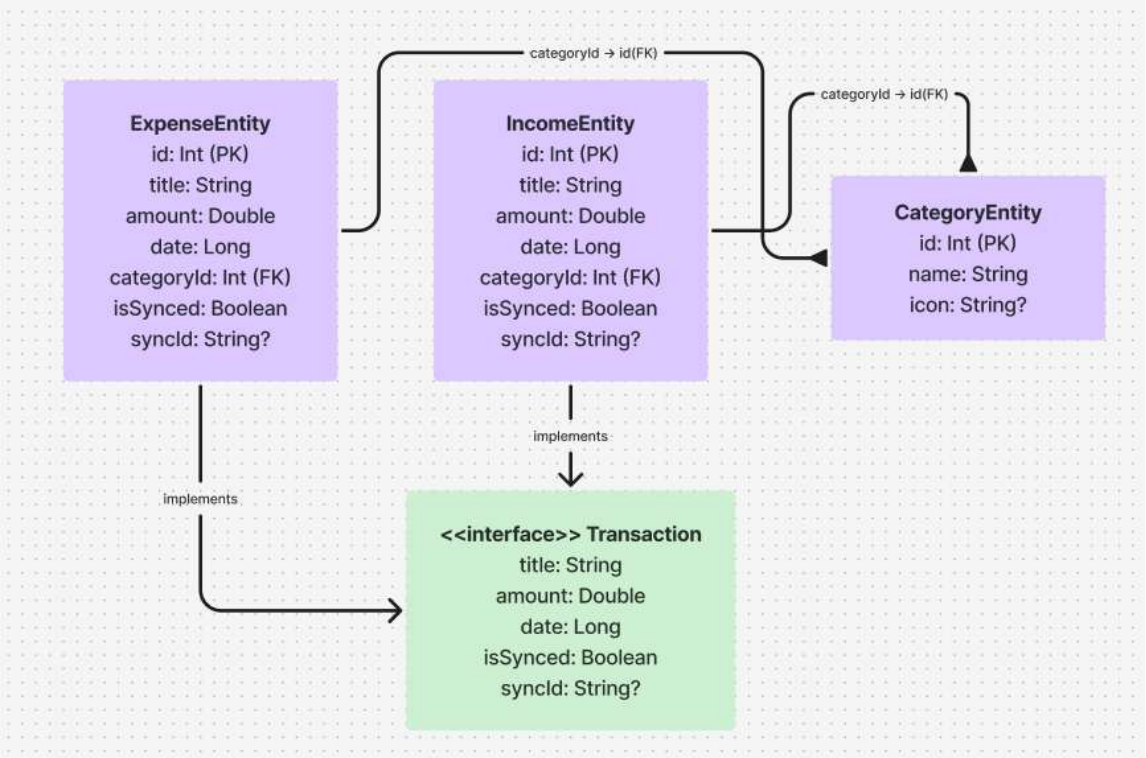

Fig 2.1: Entity Diagram of Expense Tracker App

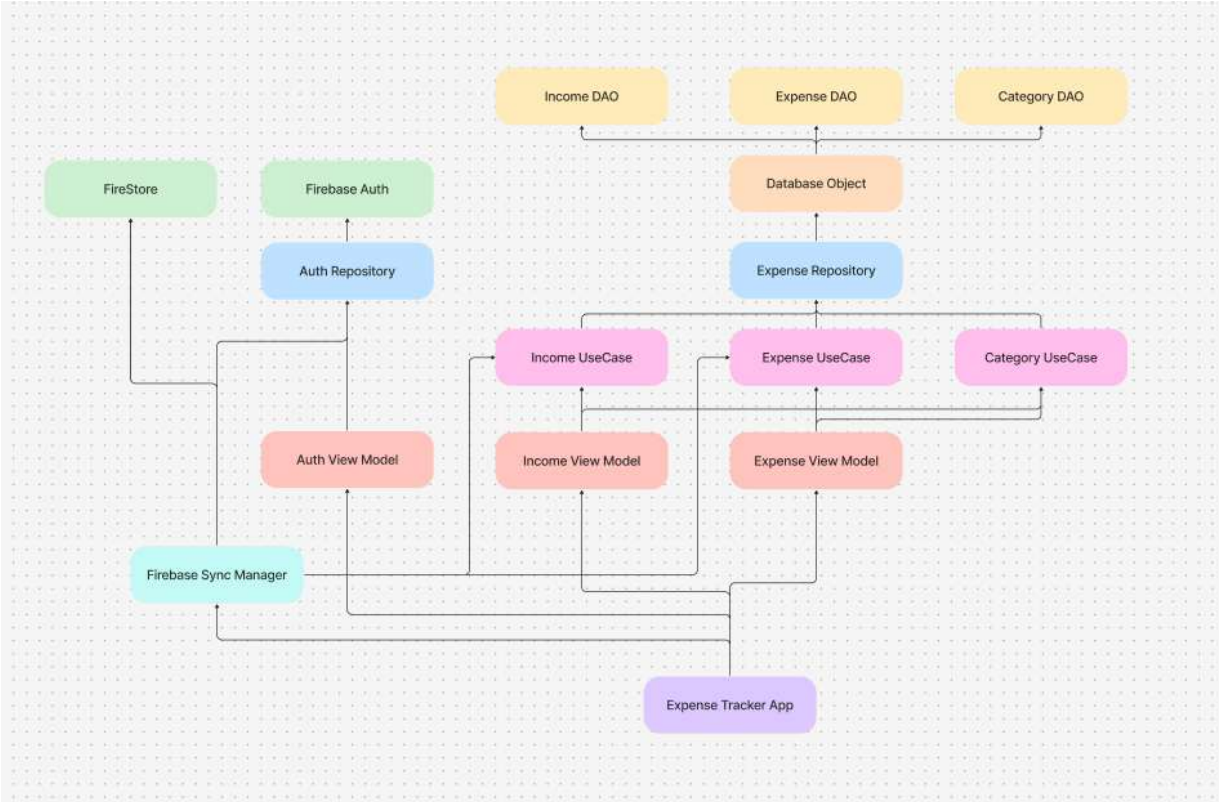

Fig 2.2: Dependency Chart of Expense Tracker App

2.1.1 Presentation Layer

The Presentation Layer is responsible for rendering the user interface and responding to user actions. It is built using **Jetpack Compose** and follows **MVVM (Model-View-ViewModel) architecture**. ViewModels manage UI state using **StateFlow**, ensuring a reactive and lifecycle-aware approach to state management.

Navigation between screens is handled through a dedicated navigation system, ensuring a smooth user experience. The dashboard provides insights into **total expenses, incomes, and net balance**, while separate screens allow users to **add, edit, and delete transactions**.

2.1.2 Domain Layer

The Domain Layer contains the core business logic of the application. It includes **Use Cases**, which represent individual actions like **adding an expense, deleting an income, retrieving transactions, or syncing data**. This layer remains independent of UI and data sources, making it reusable and testable.

Each use case interacts with the repository to fetch or manipulate data. By encapsulating business logic in this layer, the app ensures that rules remain consistent across different parts of the system.

2.1.3 Data Layer

The Data Layer is responsible for data persistence and retrieval, ensuring seamless offline access and structured storage. It consists of **entities, a local SQLite database (Room), and repositories**. **Room Database** efficiently manages financial records, allowing users to store, retrieve, and modify expenses and incomes locally.

Repositories act as an **abstraction layer**, interacting solely with the local database. They expose structured data to the Domain Layer without directly handling network operations. Synchronization with Firestore is managed at the **Use Case level**, ensuring a clear separation between offline storage and cloud syncing. This approach maintains a strict boundary, preventing direct interactions between the Data Layer and Firebase.

2.2 Design of The Application

2.2.1 Authentication & Security

The app secures user authentication using **Firebase Authentication (OAuth)**, ensuring safe sign-ins via Google accounts. **Firestore Security Rules** restrict access, allowing only authenticated users to manage their data. All communication is encrypted with **HTTPS (TLS encryption)** to protect sensitive financial information.

2.2.2 Network & Data Synchronization

Offline-first support is achieved using **SQLite (Room Database)**, allowing users to manage transactions without an internet connection. **Firestore syncs data efficiently**, only updating modified records to reduce network usage. Users can manually sync data, and **batched reads** optimize data fetching based on a selected time range.

2.2.3 State Management

The app uses **StateFlow** for reactive state updates, ensuring UI changes reflect database and Firestore modifications instantly. **Cold flows and caching** improve efficiency, while **coroutines** handle asynchronous operations, keeping the UI responsive.

2.2.4 Performance & Optimization

Room's indexing and lazy loading optimize database queries. **Jetpack Compose's remember functions** minimize recompositions, while **pagination** ensures smooth scrolling for large datasets.

2.2.5 Scalability & Maintainability

Built with **Clean Architecture**, the app maintains **modular separation** between UI, business logic, and data layers. **Repositories act as the single source of truth**, simplifying testing, debugging, and future enhancements.

3. Features of the Application

3.1. Secure Authentication with Firebase OAuth

The app utilizes Firebase Authentication with OAuth to provide a seamless and secure login experience. Users can sign in using their Google accounts, ensuring quick access while maintaining security. Authentication tokens are managed securely, allowing for multi-device access without compromising user data.

3.2. Expense and Income Management

Users can efficiently **add, edit, and delete** expenses and incomes, categorizing them based on predefined or custom categories. Each transaction includes details such as amount, date, and category, ensuring structured financial tracking.

3.3. Dashboard Overview

A dedicated dashboard provides a real-time financial summary, displaying **net expenses, net income, and overall balance**. This allows users to track their financial health at a glance, empowering better budgeting decisions.

3.4. Offline Storage with SQLite

The app ensures offline accessibility by storing all financial transactions locally using **SQLite (Room Database)**. Users can access and manage their data without an internet connection, with changes automatically synced when online.

3.5. Cloud Synchronization with Firestore

For data backup and cross-device access, expenses and incomes can be synced with **Firebase Firestore**. Users can manually trigger synchronization to ensure their financial records are stored securely in the cloud.

3.6. Selective Data Syncing

Instead of automatic sync, users can **fetch and sync data based on a specific time range**, optimizing cloud usage and ensuring they retrieve only the necessary records. Additionally, synced data can be removed from local storage to free up space.

3.7. Categorized and Sorted Expense Tracking

Users can filter and sort expenses based on **categories.** This allows for easier analysis and better financial insights.

3.8. Secure and Efficient Data Handling

Sensitive user information, such as authentication tokens and financial records, is securely handled. The app enforces **strict data access policies**, ensuring privacy and preventing unauthorized access.

3.9. Modern UI with Material 3

The app follows Google's **Material 3 design principles**, ensuring a clean, intuitive, and visually appealing user experience. **Jetpack Compose** is used for UI development, ensuring a smooth and responsive interface.

This comprehensive set of features makes the **Personal Expense Tracker** a powerful tool for managing finances, offering both **offline reliability and cloud convenience** with a user-friendly approach.

4. Overview of App Screens

4.1. Auth Screen

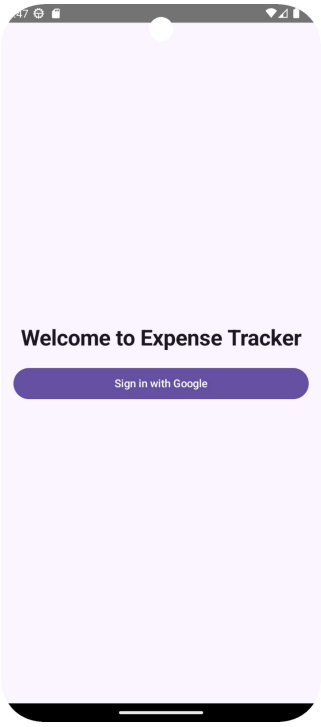

Fig 4.1: Authentication Screen of Expense Tracker App

The authentication screen handles user login through Firebase Authentication using OAuth. It ensures that only authenticated users can access the app's features. The screen provides a smooth and secure login experience, allowing users to sign in with their preferred authentication provider, such as Google. Once authenticated, users are redirected to the dashboard.

4.2. Dashboard Screen

The dashboard serves as the financial hub of the app, providing an overview of the user's financial status. It displays key metrics such as total income, total expenses, and the net balance. Users can visualize their financial trends through graphs and statistics, helping them make informed decisions. The dashboard also provides quick access to recent transactions, ensuring that users stay updated on their spending and earnings.

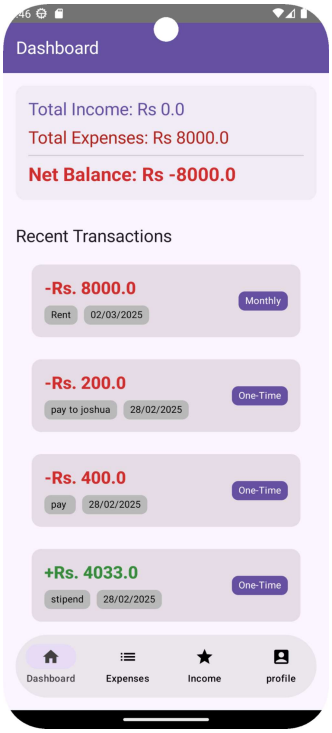

Fig 4.2: Dashboard Screen of Expense Tracker App

4.3. Income Screen

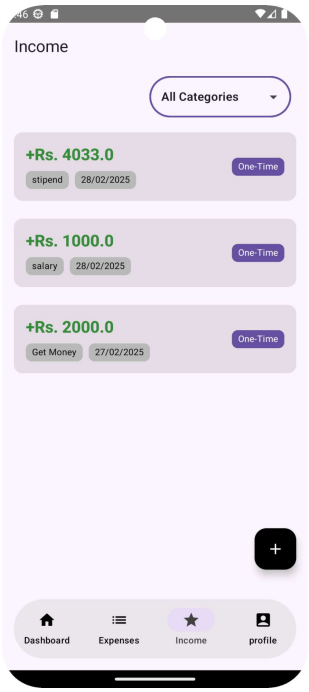

Fig 4.3: Income Screen of Expense Tracker App

The income screen is dedicated to tracking earnings. Users can add new income transactions, specifying details such as the source, amount, date, and category. They can also edit or delete

existing entries as needed. Categorization helps users analyze income sources, making it easier to manage finances. The screen may also include sorting and filtering options to view income records for specific time periods.

4.4. Expense Screen

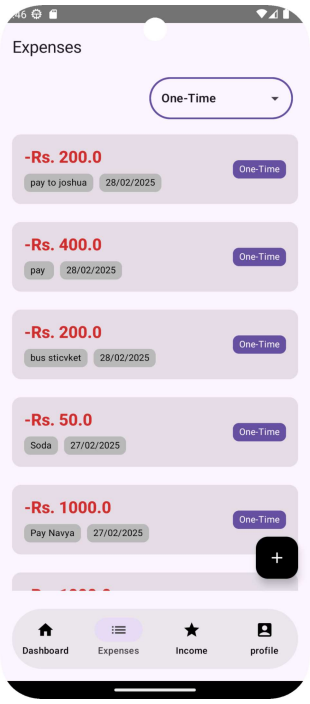

Fig 4.4: Expense Screen of Expense Tracker App

The expense screen allows users to log and monitor their spending habits. Users can add expense transactions, selecting a category, amount, and date. This structured approach helps in identifying spending patterns and managing budgets effectively. The screen also supports editing and deleting transactions, ensuring that financial records remain accurate and up to date. Users can filter expenses by category or time period for better financial analysis.

4.5. Profile Screen

The profile screen provides user account details and authentication status. It displays basic user information such as name, email, and profile picture. Users can access account-related settings, manage preferences, and log out from the application. In the future, this screen can be extended to include additional features such as theme customization, notification settings, or account linking options.

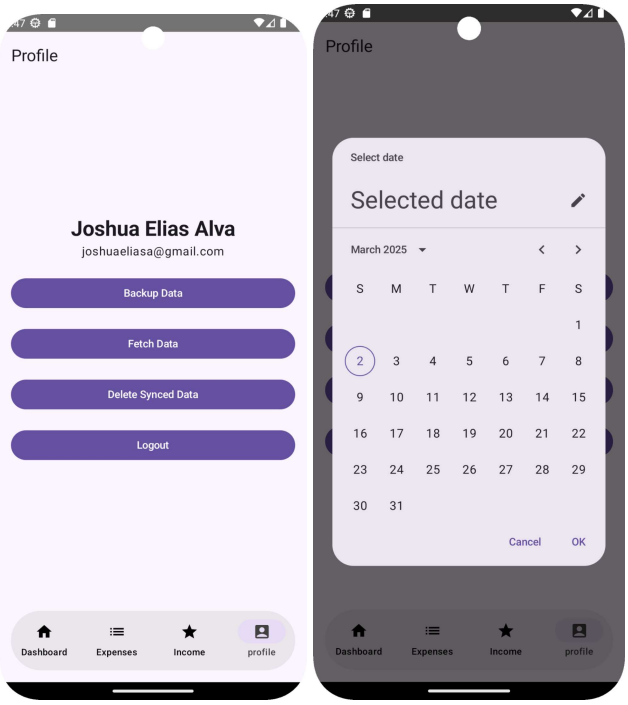

Fig 4.5: Profile Screen of Expense Tracker App

5. Conclusion

The **Personal Expense Tracker App** offers a comprehensive and user-friendly solution for managing personal finances efficiently. By leveraging **Kotlin, Jetpack Compose, Material3, Firebase, and SQLite**, the app ensures a seamless experience for users to track their income and expenses both online and offline.With a **Clean Architecture or MVVM** approach, the app maintains modularity, scalability, and ease of maintenance, ensuring a well-structured and robust financial management tool. Features such as **secure authentication, real-time dashboard insights, cloud synchronization, and offline storage** provide users with flexibility and security in managing their financial records.

The app's intuitive **Material3 UI** enhances usability, while **StateFlow and coroutine-based architecture** ensure efficient state management and responsiveness. Additionally, the integration of **Firestore and Room Database** ensures seamless data synchronization and accessibility across multiple devices.Overall, the **Expense Tracker App** empowers users with a **secure, efficient, and feature-rich** platform to gain better control over their personal finances. Future enhancements could include **AI-powered financial insights, budget planning recommendations, and multi-currency support**, making the app even more powerful and adaptable to diverse user needs.

6. References

[1] Get Started with Kotlin - https://kotlinlang.org/docs/getting-started.html

[2] Develop Android apps with Kotlin - https://developer.android.com/kotlin

[3] Material3 for Jetpack Compose - https://m3.material.io/develop/android/jetpack-compose

[4] Material3 Components - https://m3.material.io/components

[5] Firebase Setup for Android - https://firebase.google.com/docs/android/setup

[6] Read Write Firebase Data on Android - https://firebase.google.com/docs/database/android/read-and-write

[7] Firebase Authentication on Android - https://firebase.google.com/docs/auth/android/start

[8] Guide to app architecture - https://developer.android.com/topic/architecture

[9] MVVM architecture - http://netguru.com/blog/mvvm-architecture

[10] Clean Architecture - https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

[11] Better Android Apps Using MVVM With Clean Architecture - https://www.toptal.com/android/android-apps-mvvm-with-clean-architecture