Technical Documentation

Table of Contents

1. INTRODUCTION

1.1. Purpose of the Document

This document serves as the **Technical Design Document (TDD)** for the **CampusExpense Manager project**. The main purpose of the document is to provide insight and comprehensiveness into the system architecture, data structures, and logical processing flows within the application.

This document has been compiled to serve for:

- **Developers:** Understand the source code, class diagram, and how modules work for future maintenance or feature expansion.
- **Assessors/Stakeholders:** Demonstrate the technical complexity and amount of coding effort undertaken, meeting stringent offline-first, security, and performance requirements.

1.2. Project Overview

CampusExpense Manager is a Native Android mobile application developed by the BudgetWise Solutions team, to solve the problem of personal financial management for university students in Vietnam2.

The core difference of the system is the **Offline-first** architecture . The application operates completely independently of the user's device, regardless of the Internet connection or cloud server, ensuring absolute data privacy and instant access in all network conditions3.

1.3. Technology Stack

The project is built on the most stable and popular technology platform for Android development today, ensuring good backward compatibility.

Component	Specification	Description
Language	**Java 11**	Primary programming language, ensuring rigor and object-orientation (OOP)[4].
GOES	**Android Studio Iguana**	Version 2024.2.1, supports the latest debug and layout inspector tools5.
Min SDK	**API 29 (Android 10)**	Guaranteed support for >98% of student devices, good compatibility with Scoped Storage6.
Target SDK	**API 36**	Comply with the latest Google Play performance and security standards7.
Database	**SQLite**	Embedded relational database, retrieved directly via SQLiteOpenHelper (not using Room to optimize query control)[8].
Architecture	**MVC / Layered**	Clear layering model: Activities (View/Controller) is separate from Models and DatabaseHelper (Model)[9].

1.4. Third-Party Libraries & Dependencies

To optimize development time and enhance user experience, the system integrates 10 proven open-source libraries:

1. **MPAndroidChart:** Used to plot a Pie Chart for spending distribution and a Line Chart for a 6-month trend.
2. **iText7:** Powerful PDF processing library for creating professional financial reports with tables and printed page formats.

3. **AndroidX WorkManager:** Manage reliable background tasks to automatically create recurring expenses even when the app is turned off.

4. **AndroidX Biometric:** Provides a standardized biometric (Fingerprint/Face) security layer across multiple devices.

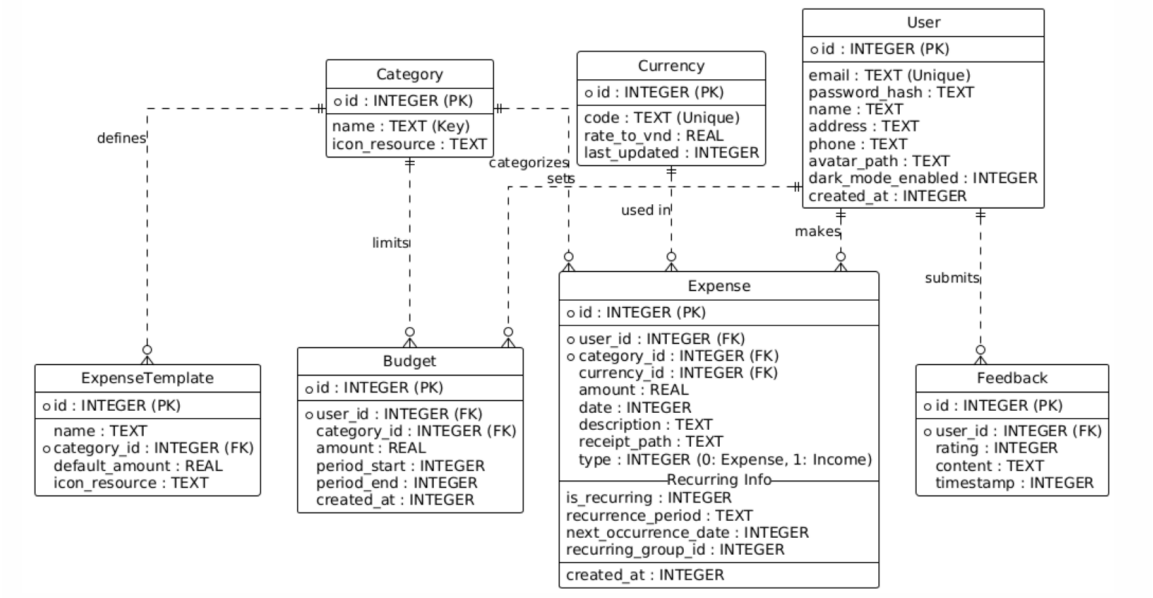
1.5. System Architecture Overview

The system follows the **Monolithic Mobile Application** architecture with **a Layered Architecture** design to ensure Modularity and Maintainability11.

- **Presentation Layer (UI):** Includes Activities and XML Layouts, which are responsible for displaying data and receiving interaction from users.
- **Business Logic Layer:** Adapters, CurrencyConverter, RecurringExpenseWorker handle computation, currency formatting, and repetitive logic.
- **Data Access Layer (DAL):** The DatabaseHelper class serves as the only communication gateway to SQLite, executing complex SQL statements (CRUD).
- **Data Layer:** Models (POJO) such as User, Expense, Budget represent the entity data structure.

2. Database Schema

2.1. Entity Relationship Diagram (ERD) (update)



2.2. Data Dictionary

1. Table: users

Column Name	Data Type	Constraints	Description
id	INTEGER	PK, Auto Increment	Unique identifier (ID) of the user.
email	TEXT	UNIQUE, NOT NULL	Login email (must not be the same).
password_hash	TEXT	NOT NULL	Encrypted password (SHA-256).
name	TEXT	NOT NULL	User display name.
address	TEXT	Nullable	Contacts.
phone	TEXT	Nullable	Phone number.
avatar_path	TEXT	Nullable	The link to the profile picture file on the device.
dark_mode_enabled	INTEGER	Default 0	1: Dark Mode on, 0: Off.

created_at	INTEGER	NOT NULL	Account Creation Time (Unix Timestamp).

2. Table: categories

Column Name	Data Type	Constraints	Description
id	INTEGER	PK, Auto Increment	Category identification.
name	TEXT	NOT NULL, Unique	Category codes (e.g., 'cat_food') are multilingual.
icon_resource	TEXT	NOT NULL	The name of the icon file in res/drawable (e.g., 'ic_food').

3. Table: expenses

Column Name	Data Type	Constraints	Description
id	INTEGER	PK, Auto Increment	Transaction identifiers.
user_id	INTEGER	FK (users.id)	Transaction creator ID.
category_id	INTEGER	FK (categories.id)	Spending category ID.
currency_id	INTEGER	FK (currencies.id)	Currency ID.
amount	REAL	NOT NULL	Transaction amount.
date	INTEGER	NOT NULL	Execution Date (Unix Timestamp).
description	TEXT	Nullable	Additional Notes/Descriptions.
receipt_path	TEXT	Nullable	Invoice photo link.
type	INTEGER	Default 0	0: Expense, 1: Income.

is_recurring	INTEGER	Default 0	1: Is a repeat transaction, 0: Regular transaction.
recurrence_period	TEXT	Nullable	Repeat cycles ('daily', 'weekly', 'monthly').
next_occurrence	INTEGER	Nullable	The next transaction is expected to be created.
recurring_group_id	INTEGER	Default 0	A group ID to link repeat transactions together.

4. Table: budgets

Column Name	Data Type	Constraints	Description
id	INTEGER	PK, Auto Increment	Budget identification.
user_id	INTEGER	FK (users.id)	Budget Creator.
category_id	INTEGER	FK (categories.id)	Applicable categories (0 = Total budget).
amount	REAL	NOT NULL	Maximum money limit.
period_start	INTEGER	NOT NULL	The start date of the counting.
period_end	INTEGER	NOT NULL	End date.
created_at	INTEGER	NOT NULL	The date the record was created.

5. Table: expense_templates

Column Name	Data Type	Constraints	Description
id	INTEGER	PK, Auto Increment	Sample identifiers.
name	TEXT	NOT NULL	Sample display name (e.g., 'Rent').
category_id	INTEGER	FK (categories.id)	The default category of the template.
default_amount	REAL	Default 0	Suggested amount.

icon_emoji	TEXT	Nullable	Emoji icons (e.g. 🏠,).

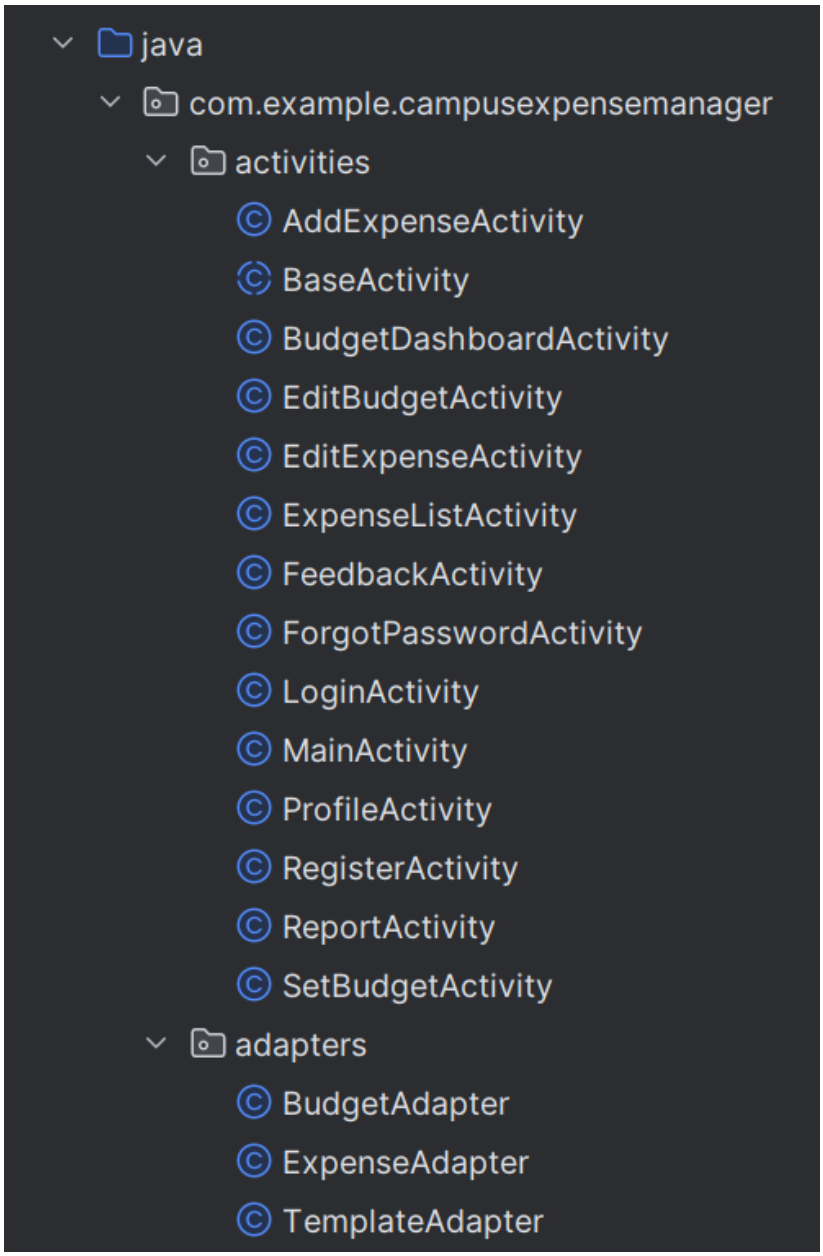
6. Table: feedback

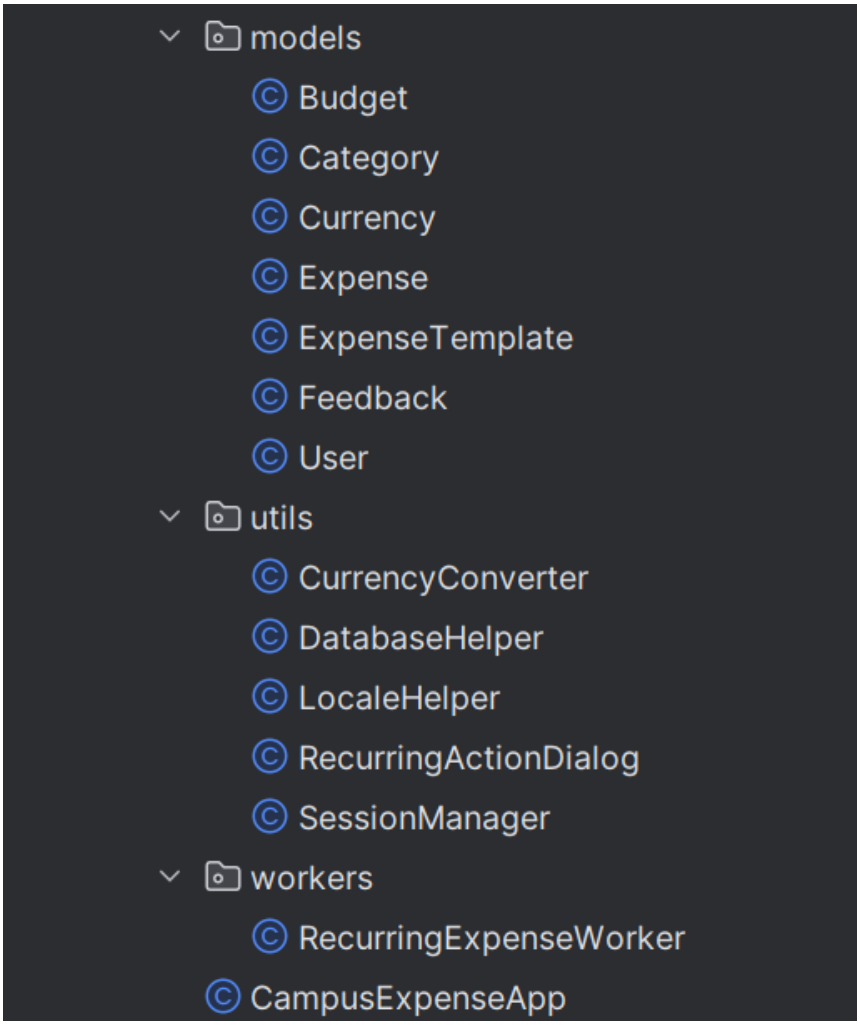
Column Name	Data Type	Constraints	Description
id	INTEGER	PK, Auto Increment	Feedback identifiers.
user_id	INTEGER	FK (users.id)	Sender.
rating	INTEGER	NOT NULL	The number of review stars (1-5).
content	TEXT	Nullable	Detailed content.
timestamp	INTEGER	NOT NULL	Submission time.

3. SYSTEM ARCHITECTURE & DESIGN

3.1. Project Package Structure

The project structure is organized according to **the Package-by-Layer model**, which provides a clear separation between Interface (UI), Data (Data) and Processing Logic. Here is the main source code directory tree of the application:


java
com.example.campusexpensemanager
activities
AddExpenseActivity
BaseActivity
BudgetDashboardActivity
EditBudgetActivity
EditExpenseActivity
ExpenseListActivity
FeedbackActivity
ForgotPasswordActivity
LoginActivity
MainActivity
ProfileActivity
RegisterActivity
ReportActivity
SetBudgetActivity
adapters
BudgetAdapter
ExpenseAdapter
TemplateAdapter



3.2. Component Responsibilities

Detailed description of the function of each package (Package):

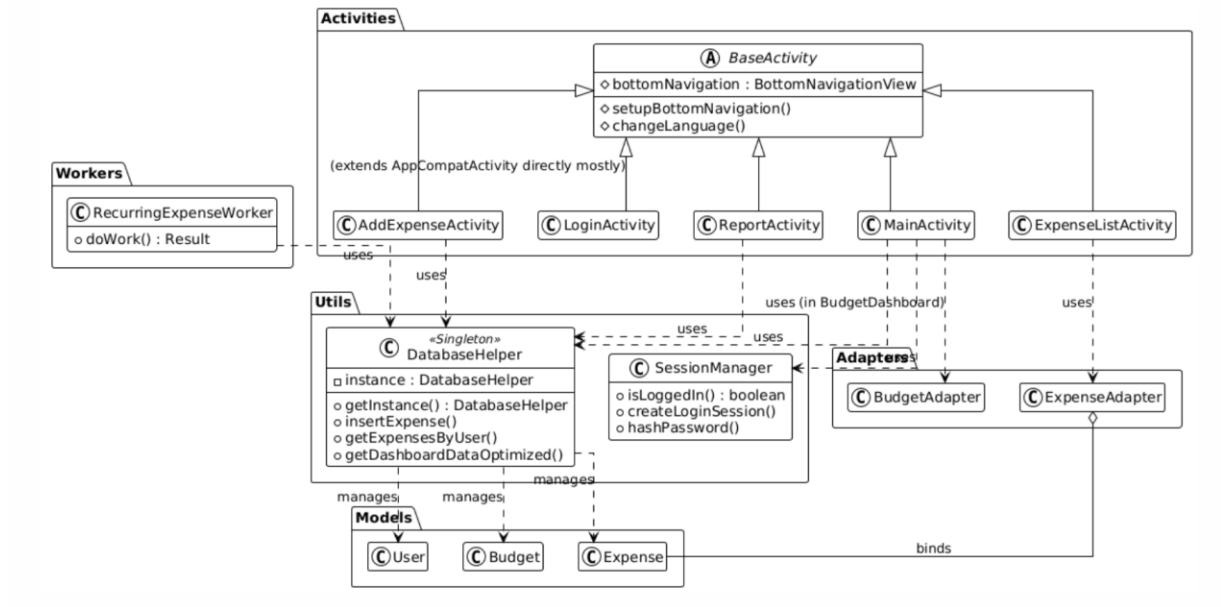
- **activities**: Contains classes inherited from AppCompatActivity. In particular, BaseActivity acts as the Parent Class that handles common logic such as **Navigation Bottom Bar**, **Locale,** and **Dark Mode** for all child screens.
- **models**: Defines data objects (Entities). These classes contain only attributes, constructor, and getter/setter, which directly reflect the table structure in SQLite.
- **utils**: Contains the core business logic:
    - DatabaseHelper: Manages the database lifecycle and executes any SQL statements.

- SessionManager: Manage login sessions, encrypt passwords (SHA-256), and save user settings.

- **workers**: Use Android WorkManager to perform background tasks even when the app is turned off, ensuring that recurring expenses are always created on time.

3.3. Class Diagram

The diagram below illustrates the relationship between the main components of the system, including inheritance and dependency.

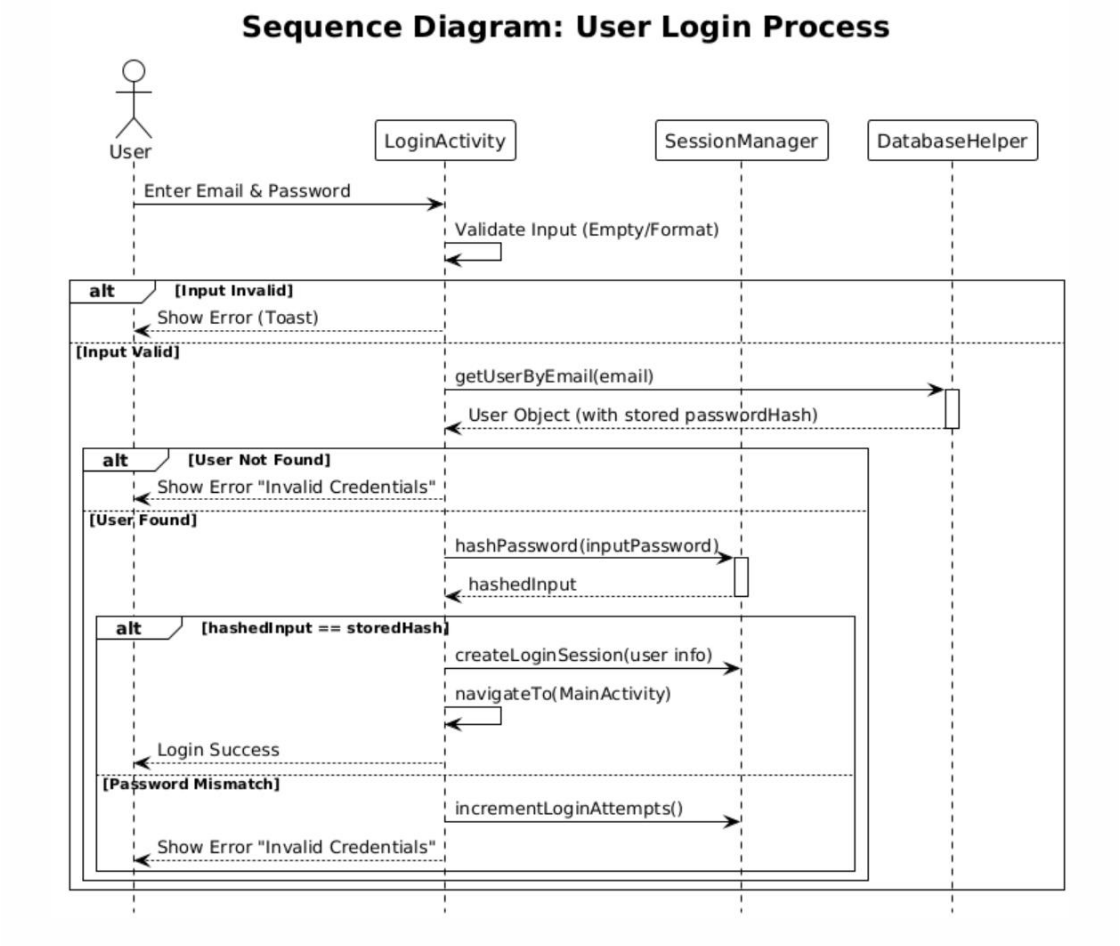


4. SYSTEM LOGIC & DATA FLOWS

This chapter describes in detail the Internal Data Flow of the most important business functions in the application. The charts below illustrate how Activities, DatabaseHelper, and Models interact with each other.

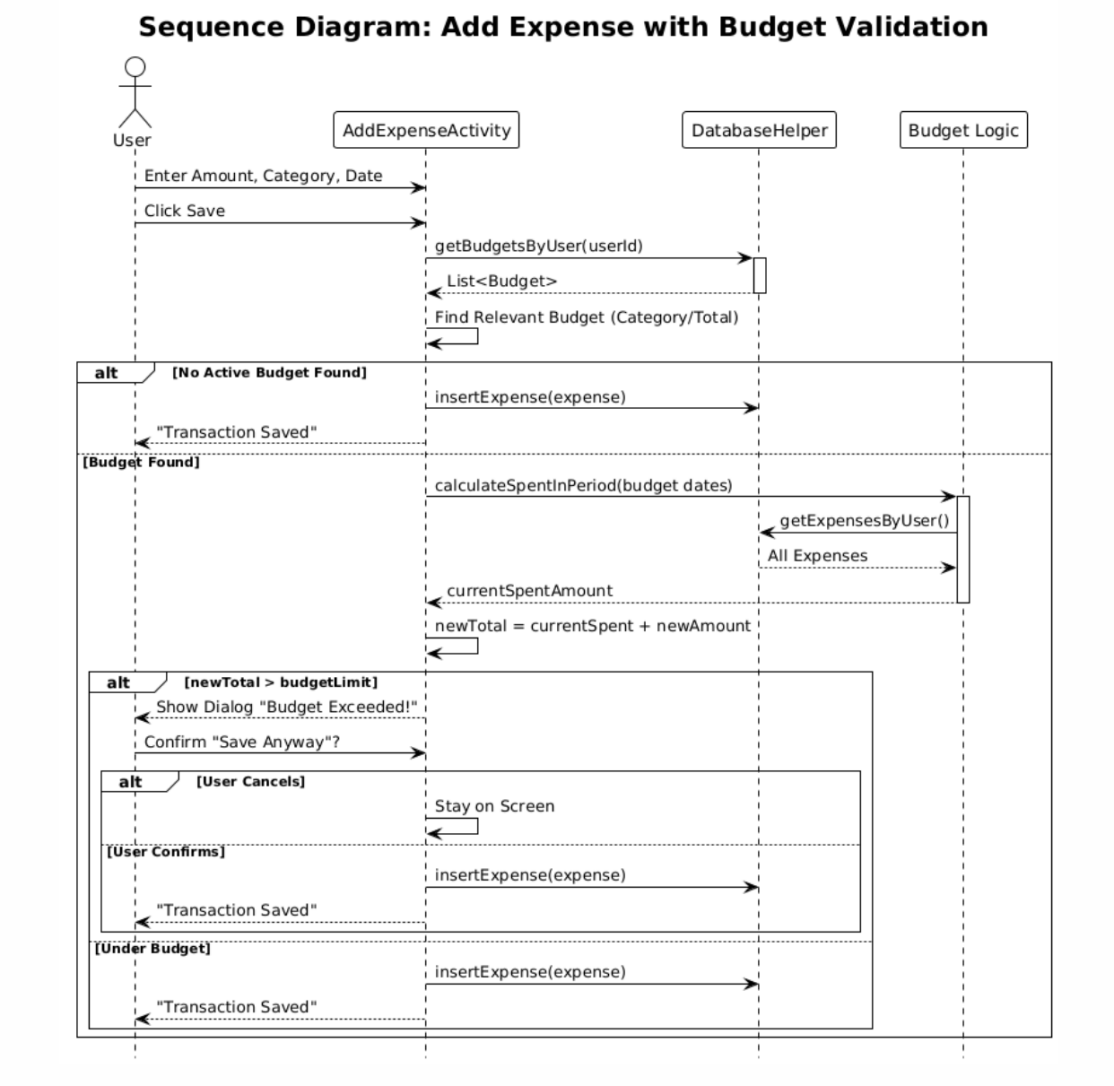
4.1. User Authentication Flow (Secure Login)

The login process ensures security by never comparing plain text passwords. The system hashes the password entered by the user and compares it with the hash string saved in SQLite.



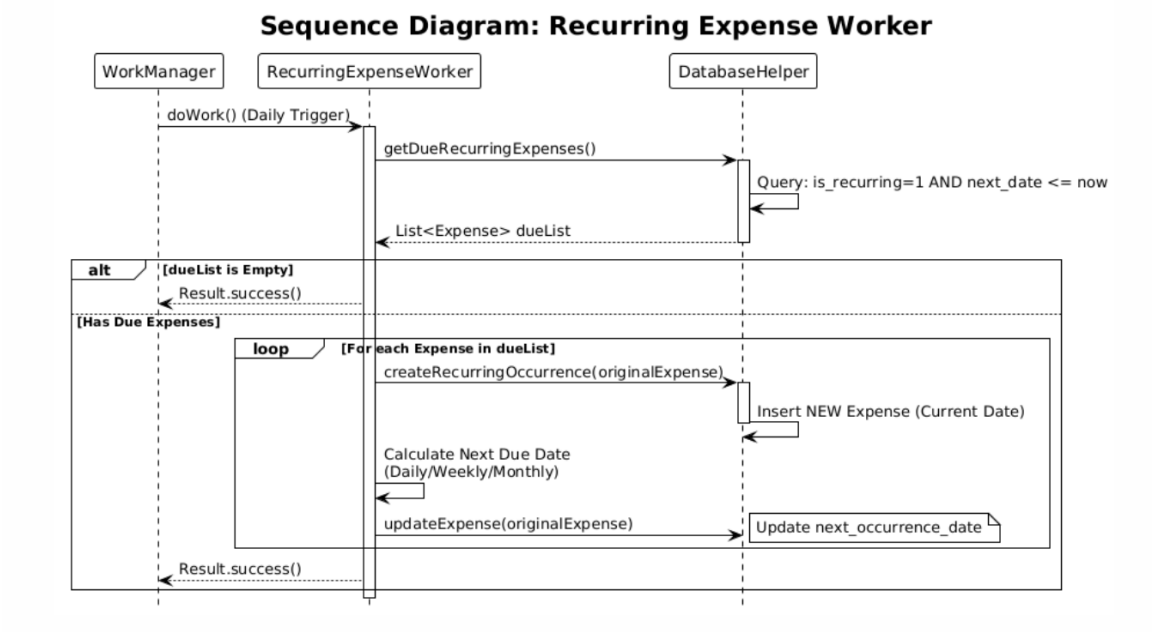
4.2. Expense Creation Flow (with Budget Check)

This is the most complex business flow. Before saving an expense transaction, the system must calculate whether this transaction exceeds the budget limit (Budget) or not to issue a timely warning.



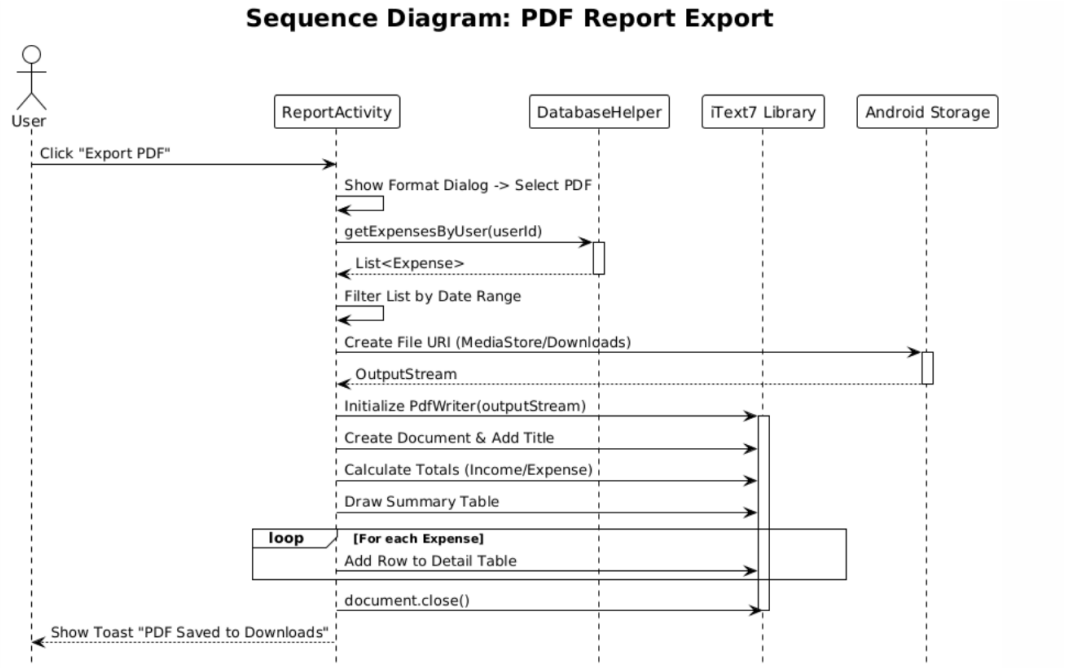
4.3. Recurring Expense Automation Flow

This automation uses Android WorkManager. Workers run in the background every day, scan the database for recurring transactions that are due, and automatically create new records without the user opening the application.



4.4. PDF Report Generation Flow

The report export feature uses the iText7 library. The processing flow includes querying data, formatting tables, and writing files to the device's memory according to the Android 10+ Scoped Storage standard.



5. THIRD-PARTY LIBRARIES & IMPLEMENTATION

To ensure performance and stability, the project uses industry-standard open-source libraries. Here are the technical details on how to integrate and use them.

5.1. MPAndroidChart (Visual Analytics)

- **Purpose:** Financial data visualization makes it easy for users to grasp spending trends.
- **Implementation:**
    - **PieChart:** Used in ReportActivity to show the percentage of spend by category. The data is aggregated from SQLite (getExpensesByUser), grouped by categoryId, and then fed into PieDataSet. The slices are dynamically colored based on the Sera UI color palette.
    - **LineChart:** Shows spending trends for the last 6 months. The X-axis uses IndexAxisValueFormatter to display the month name (Jan, Feb...), the Y-axis displays the amount.

5.2. iText 7 (Professional Reporting)

- **Purpose:** Generate high-quality PDF reports, which can be printed or emailed.
- **Implementation:**
    - Use the PdfWriter and PdfDocument classes to initialize PDF files.
    - **Table Layout:** The data is presented as a table (com.itextpdf.layout.element.Table) with a custom column width (UnitValue.createPercentArray).
    - **Scoped Storage: Logical** saving files that strictly comply with Android 10+ security. On Android Q+, use the MediaStore and ContentValues to write the data stream (OutputStream) to the Downloads folder without WRITE_EXTERNAL_STORAGE permissions.

5.3. AndroidX WorkManager (Background Automation)

- **Purpose:** Reliably handle the "Recurring Expenses" feature even when the app is shut down or the device restarts.

- **Implementation:**
    - **Worker Class:** RecurringExpenseWorker inherited from Worker. The doWork() function contains the logic that scans the database for transactions that are due (next_occurrence_date <= current_time).
    - **Scheduling:** In MainActivity, a PeriodicWorkRequest is configured to run every 24 hours (1, TimeUnit.DAYS) with the ExistingPeriodicWorkPolicy.KEEP policy to avoid duplicate tasks.

5.4. AndroidX Biometric (Security)

- **Purpose:** To provide a secure passwordless login method.
- **Implementation:**
    - Use BiometricPrompt to display the system's standard authentication dialog.
    - The hardware test logic (BiometricManager.canAuthenticate) is performed before the feature is activated.
    - Security Key: The user's email is saved in SharedPreferences (KEY_BIOMETRIC_EMAIL) only if fingerprint authentication is successful for the first time.

6. CONCLUSION & FUTURE WORK

6.1. Technical Achievements

The CampusExpense Manager **project** has accomplished the goal of building a powerful personal finance management application, complying with modern technical standards:

1. **Architecture:** Layered architecture is clear and easy to maintain.
2. **Offline-first:** Works perfectly without the Internet thanks to SQLite optimization.
3. **Security:** Applies SHA-256 password encryption and biometric security.
4. **Performance:** Use Raw SQL Queries for big data aggregation tasks, ensuring a UI response of < 50ms.

6.2. Known Limitations

- **Sync:** Cloud Sync between multiple devices is not yet supported.
- **Automation: There** is no automatic invoice scanning (OCR) feature with AI.

6.3. Future Roadmap (v2.0)

Based on the current solid technical foundation, the following features are recommended for the next version:

- Firebase Realtime Database **integration** for data backup.
- Use **Google ML Kit** to extract information from invoice snapshots.
- Upgrade to **MVVM architecture** with **Jetpack Compose** to modernize the Codebase UI.