SmartSplit Technical Specification

Prepared by: Kevin Fasan &
Nkemjika Onwumereh

Table Of Contents

Introduction

The Bill Splitter Application is a mobile application designed to simplify and streamline the process of splitting bills among group members. By leveraging advanced AI and machine learning technologies, the application automates the tedious process of calculating individual expenses, ensuring accurate and efficient bill division. Users can assign specific items to individuals within their groups and receive an instant breakdown of each member's share, including service charges and tips.

1.1 Overview

This system is developed for the Android platform, utilizing Jetpack Compose for building a responsive and intuitive user interface. The architecture follows the Model-View-ViewModel (MVVM) design pattern, ensuring a clear separation of concerns and facilitating maintainable, scalable code. The application integrates with Firebase services such as Firestore, Storage, and Authentication to securely manage user data, store digital receipts, and authenticate users.

A built-in camera interface enables users to capture images of receipts directly within the app. Using Optical Character Recognition (OCR) technology, the app extracts text from receipt images and converts it into digital, itemized lists. Users can then assign items to group members, and the system calculates the exact amount each member owes, dividing service charges and tips proportionally. This process eliminates manual data entry and minimizes human error, making bill splitting quick, accurate, and hassle-free.

1.2 Glossary

- Jetpack Compose: A modern UI toolkit for building native Android applications using a declarative programming model.
- MVVM (Model-View-ViewModel): An architectural pattern that separates the user interface from the application logic, improving maintainability and testability.
- Firebase Firestore: A cloud-based NoSQL database that allows real-time data storage and synchronization across devices.
- Firebase Storage: A cloud service for storing and retrieving files, such as images of receipts captured using the Camera API.

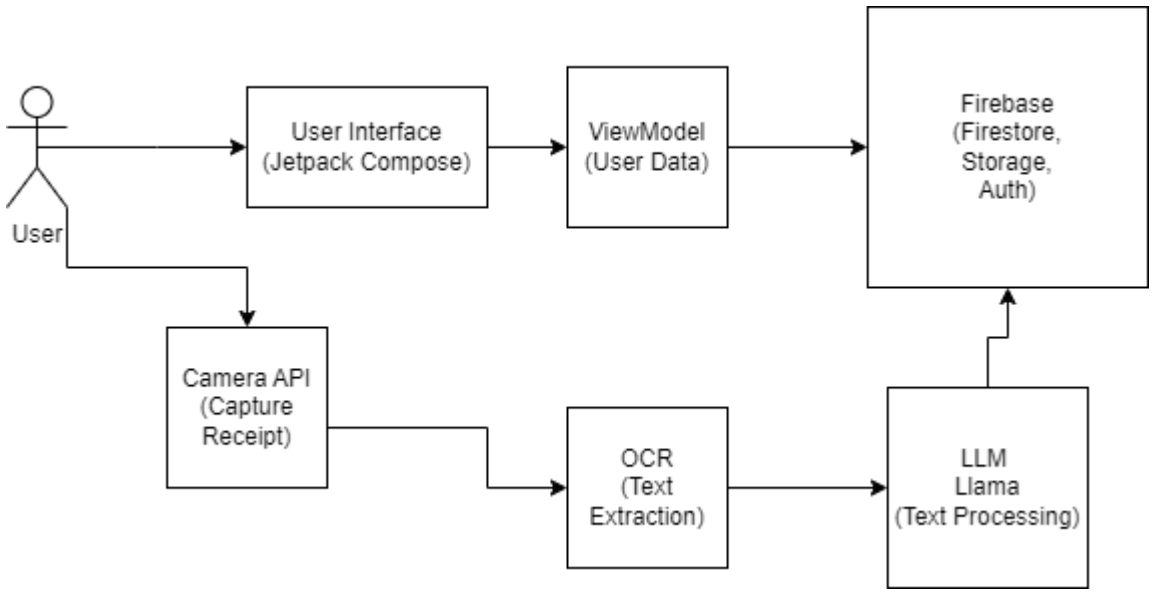
- Firebase Authentication: A secure authentication service that enables users to log in using various identity providers.
- Camera API: A library that provides functionality for capturing images using the device's camera.
- OCR (Optical Character Recognition): A technology that extracts text from images, converting printed or handwritten characters into digital text.
- LLM (Large Language Model): An advanced AI model capable of understanding and generating human language, such as LLAMA 8b, used to improve text interpretation and context recognition within the app.

---

2. System Architecture

The system architecture is designed to be modular, scalable, and user-friendly.

2.1 High-Level Architecture Diagram



2.2 Components

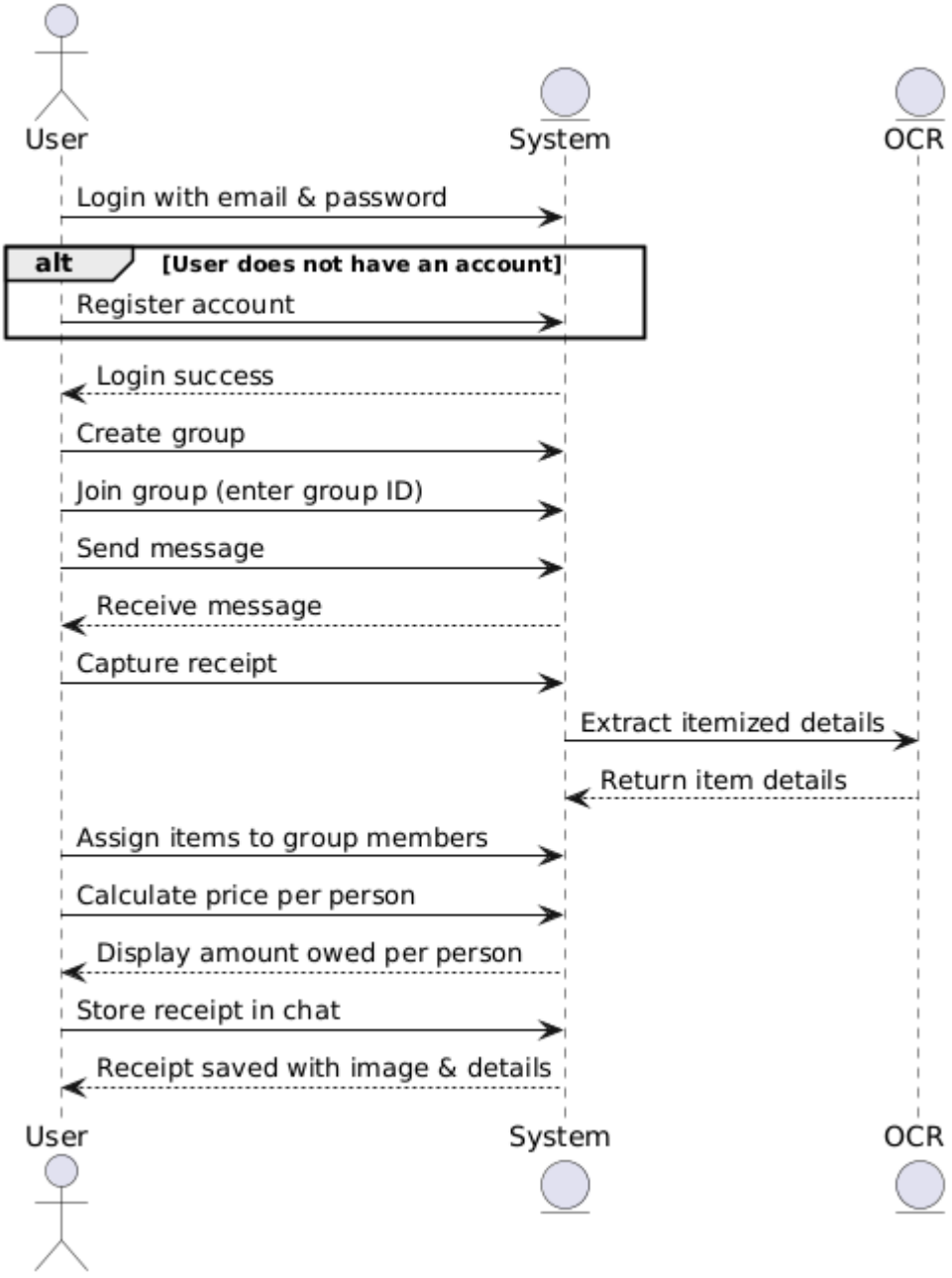
- User Interface (UI): Built using Jetpack Compose, the UI includes screens for capturing receipts, itemizing expenses, assigning items to members, and displaying calculated amounts.
- ViewModel: Manages, state, and communication with Firebase services. It ensures separation of concerns between UI and data storage.
- Firebase:

    - Firestore: Stores User Accounts, Messages, Groups, Image Links, structured bill details, item assignments, and calculated amounts.
    - Storage: Manages receipt images captured by the camera API.
    - Authentication: Handles user authentication for accessing application features.
- Camera API: Captures receipts and passes image to OCR API for text extraction
- OCR: Extracts raw data from receipt and passes this to LLM for processing
- LLM (Llama 8b): Parses this data and itemized details alongside owed amounts is then uploaded to Firebase Firestore

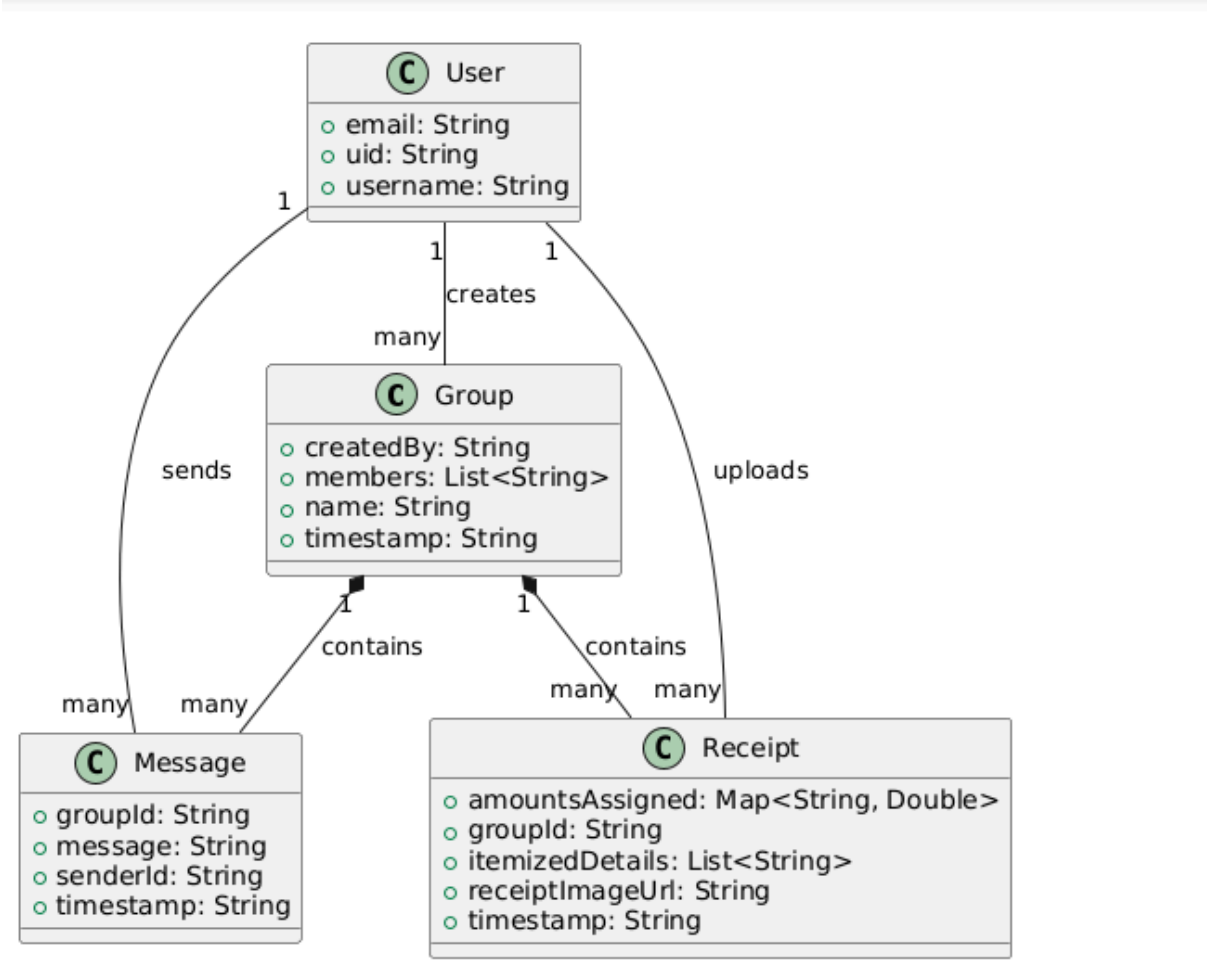
---

3. High-Level Design

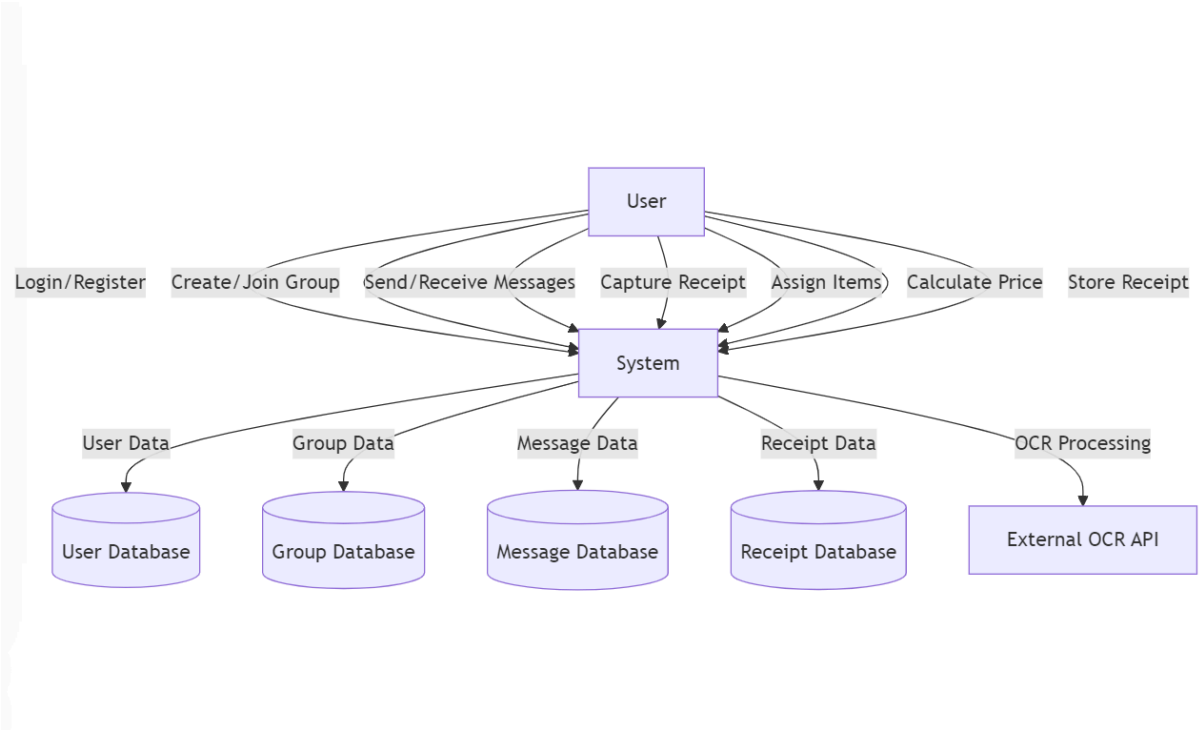
Sequence Diagram



3.1 Object Models



3.2 Data Flow Diagram (DFD)



3.3 Key Design Decisions

1. Jetpack Compose's MutableState is used to manage UI state, ensuring reactivity and efficient updates during the bill splitting process.

2. State Management: Initially managed states within each composable function, but this state was not reflected as the user traversed through the application so we opted for better state management using a ViewModel. This allowed the state to be “remembered” as the user moved through different screens. This was highly efficient as instead of resetting and fetching data on a new page the application fetches data once and uses it throughout pages.

3. Firebase Integration: Firebase Firestore and Storage are chosen for their ability to handle real-time data synchronization and storage efficiently.

4. Dialog Management: Dialogs (e.g., for assigning items, edit, group selection) are managed using state variables in Jetpack Compose, ensuring they appear immediately when needed without requiring manual interaction from the user.

---

4. Problems and Resolution

This section outlines the major problems encountered during development and their solutions.

4.1 Problem: Depreciated Features

- Description: Many of the features researched / implemented at first were dependent on depreciated libraries
- Refactored code after replacing older libraries and synced Gradle files with the project files to make use of the newer libraries.

4.2 Problem: Dialog Not Displaying Immediately

- Description: When the "Assign to" button was clicked, the dialog did not appear immediately. Users had to scroll to the bottom of the page for the dialog to display.
- Resolution: The dialog was moved outside the LazyColumn and placed in a Box composable. This ensured the dialog was always part of the composition hierarchy and displayed immediately upon clicking the button.

4.3 Problem: Calculation State Not Resetting

- Description: The "Store Receipt" button remained enabled even after reassigning items to members, leading to incorrect calculations.
- Resolution: The isCalculationComplete state was reset to false whenever the "Assign to" button was clicked. This ensured the button was disabled until a new calculation was performed.

4.4 Problem: Image Upload Failure

- Description: Receipt images were not being uploaded to Firebase Storage due to incorrect URI handling.
- Resolution: The URI handling logic was fixed, and proper error handling was added to ensure images were uploaded successfully.

---

5. Installation Guide

5.1 Prerequisites

- Android Studio (version ≥2023.1)
- Firebase Account with Firestore and Storage enabled in the project settings
- Android device or emulator running API level 26 (or higher)

5.2 Installation Steps

1. Clone the Repository:

   git clone https://gitlab.computing.dcu.ie/fasank2/2025-csc1049-kfasan-waggify.git
   cd 2025-csc1049-kfasan-waggify
2. Set Up Firebase Console:
   - Create a new project in Firebase Console.
   - Enable Firestore and Storage for your project.
3. Download Google Services File:
   - Obtain google-services.json from the Firebase console under "Files" > "Storage".
   - Place this file in the app directory of the project.
4. Sync Project with Grade Dependencies.
   - Head over to ‘Files’ > ‘Sync Project with Gradle Files’ or use the shortcut Ctrl + Shift + O
5. Build and Run the Application:
   - Connect an Android device or start an emulator.
   - Click Run in Android Studio to build and deploy the application.
6. Verify Installation:
   - Launch the app on your device/emulator.

- Ensure you can capture a receipt, itemize expenses, assign items, and calculate amounts accurately.

---

6. Machine Learning Integration

6.1 Google ML Kit for OCR

The Google ML Kit is integrated into the system to enhance the text processing capabilities of the application:

- OCR (Optical Character Recognition): Extracts text from images captured by the camera API and converts it into structured data that can be stored in Firebase Storage.
- Error Handling: The ML Kit provides error detection during OCR, ensuring accurate transcription of text.

This integration improves the system's ability to recognize and process text accurately, even in low-light or challenging imaging conditions.

6.2 LLM (LLAMA 8b) for Text Processing

The application leverages an LLM such as LLAMA 8b for advanced text processing tasks:

1. Text Processing: After capturing a receipt, the app generates a summary of the bill details using the LLM, providing a concise overview for quick decision-making. This LLM corrects any spelling errors that OCR may have created.
2. Contextual Understanding: Beyond basic OCR, the LLM enhances the system's ability to understand and interpret text in various formats (e.g., multi-line invoices, spaces between text, etc.).

This integration ensures that the app not only processes text accurately but also leverages advanced AI capabilities for enhanced functionality.

---

7. References

1. Google ML Kit Documentation:

   https://developers.google.com/ml-kit

2. LLAMA 8b Documentation:

   https://www.llama.com/docs/get-started/

3. Firebase Firestore:

   https://firebase.google.com/docs/firestore

4. Jetpack Compose Quick Start Guide:

   https://developer.android.com/develop/ui/compose/setup