Receipt Manager - Project Setup Document

Overview

Receipt Manager is a Flutter application that allows users to scan, manage, and analyze their receipts efficiently. The app integrates Firebase for user authentication and data storage and supports expense tracking, data visualization, and budget management.

Prerequisites

Before setting up this project on your local machine, make sure you have the following:

- **Flutter**: Ensure you have Flutter installed. Flutter installation guide.
- **Xcode** (for iOS development) or **Android Studio** (for Android development).
- **Firebase Account**: You will need a Firebase account to configure Firebase authentication, Firestore, and other services.
- **Google Cloud API Key**: You'll need a Google Cloud Vision API key for receipt scanning.
- **Open Exchange Rates API Key**: To access currency conversion rates, you'll need an API key from Open Exchange Rates.

Setup Instructions

1. Clone the Repository

To get started with the project, clone the repository to your local machine:

```
git clone https://github.com/jingjingyang0803/Receipt-Manager-App.git
cd receipt_manager
```

2. Install Dependencies

Run the following command to install the required Flutter dependencies:

```
flutter pub get
```

3. Configure Firebase

- Set up Firebase for both **iOS** and **Android**.

- Download the **google-services.json** (for Android) and **GoogleService-Info.plist** (for iOS) from the Firebase console and place them in the respective directories.
    - For Android: Place **google-services.json** in the android/app/ directory.
    - For iOS: Place **GoogleService-Info.plist** in the ios/Runner/ directory.
- Follow Firebase Instructions
    - Once the configuration files are in place, you need to follow the specific instructions provided by Firebase for further setup and integration, especially for enabling Firebase services like authentication, Firestore, and storage.

4. Set Up Firebase Cloud Functions (For OCR)

- Ensure that Firebase Cloud Functions are set up to handle OCR requests.
- Create a Firebase project and configure the Cloud Functions with the Google Cloud Vision API.

5. Run the Application

To run the app on your device or emulator:

- **For Android**:
    - Ensure you have an Android emulator running or a physical device connected.
    - Run the following command: `flutter run`
    - Or, press the **Run** button in your IDE to launch the app directly on the connected Android device.
- **For iOS**:
    - Make sure you have an iOS simulator running or a physical device connected.
    - Ensure you have set up your iOS device for development.
    - Run the following command: `flutter run`
    - Or, press the **Run** button in your IDE to launch the app directly on the connected iOS device.

Key Features

- **Receipt Scanning**: Uses the device camera and OCR API to scan receipts and extract data.
- **Expense Management**: Users can categorize their expenses and manage them easily.
- **Data Visualization**: View your expenses in graphs and charts for better understanding.
- **Budget Setting**: Set budgets for different categories and get alerts when limits are reached.
- **Currency Conversion**: Supports real-time currency conversion using the Open Exchange Rates API.

Dependencies

- **Core**: flutter SDK
- **Firebase**: firebase_core, firebase_auth, cloud_firestore, firebase_storage, cloud_functions
- **Local Storage**: shared_preferences, path_provider
- **Camera and Image Picker**: camera, image_picker
- **Data Visualization**: fl_chart
- **Currency API**: http, intl
- **And more.**

Troubleshooting

- If the app does not launch, make sure your **Android/iOS setup** is correct and dependencies are installed.
- Check the Firebase and Google Cloud configurations to ensure they are properly linked.

Known Problems

1. **Batch Delete Limitations**: Users must delete expenses one at a time.
2. **Lack of Goal Tracking**: Users can't set and track financial goals.
3. **No Feedback System**: The app doesn't currently collect user feedback.

4. **Limited Responsiveness**: The app does not fully adapt to various screen sizes.
5. **State Management Error (Assertion Failure)**:
   - **Issue**: The app triggers an error when trying to update the state of a widget after it has been disposed of, caused by asynchronous operations.
   - **Solution**: Ensure that setState() is called only when the widget is still mounted using if (mounted).

Additional Documentation

Technical Document: https://tuni-my.sharepoint.com/:w:/r/personal/jingjing_yang_tuni_fi/_layouts/15/Doc.aspx?sourcedoc=%7B186B3738-33B3-4B57-A948-9466EBCC4F63%7D&file=Technical%20Document_TeamC.docx&action=default&mobileredirect=true

---

Conclusion

This document provides the necessary steps to set up the **Receipt Manager** project locally. If you encounter any issues or need further assistance, feel free to reach out.