Expense Budget App Documentation

In today's fast-paced world, keeping track of expenses and maintaining a budget is essential for financial stability. This application solves the problem of manual expense tracking by providing users with an intuitive, mobile-based solution. The app allows users to record transactions, set budget limits, and receive alerts when their spending exceeds a set threshold, ensuring they can manage their finances effectively. The application was built using Swift and SwiftUI, with a Flask backend connected to a MySQL database.

From a technical perspective, the project integrates a user-friendly front end with a backend logic. The application consists of five main views: a dashboard for navigating different sections, a budget management screen, a transaction history, an alerts page, and a category view for organizing spending habits. One advanced feature of this app is its integration with API endpoints that handle real-time CRUD operations for managing budgets, transactions, and alerts. The app also includes a visual alert system, where users receive notifications if their spending exceeds their budget limits. This feature not only enhances user experience but also makes the app more interactive and engaging.

The backend database is organized into five key tables: User, Budget, Category, Transaction, and Alert. Each table is designed with clear relationships to ensure data consistency. For example, the Transaction table references both the User and Budget tables through foreign keys, enabling accurate tracking of which user made a transaction and which budget it belongs to. This design ensures scalability and maintains data integrity. Additionally, the Alert table generates alerts dynamically when the user's transactions exceed their budget limits, further enhancing the app's functionality. These relationships are critical for ensuring the app performs as expected and provides meaningful insights to the user.

To run the application, users first need to set up the Flask backend by installing dependencies and running the server locally. This can be done by activating a virtual environment, ensuring the required Python libraries, such as Flask and python-dotenv are installed, and then starting the server. On the front end, the Swift application can be launched in Xcode. Users can then interact with the app through an iPhone simulator, adding transactions, creating budgets, and viewing their financial activity. As the preview/content view function only works on static data, the user would have to build and then run the code. The integration between the backend and front end ensures real-time synchronization of data.

During development, one of the most significant blockers was dealing with foreign key constraints in the database. Errors arose when trying to add transactions without valid budget IDs. After debugging, we ensured the Transaction table had the correct references and set up cascading updates and deletions. This resolved the issue, allowing transactions to be linked to budgets dynamically. Another challenge was integrating the alert functionality, which required implementing real-time logic in both the backend and frontend to ensure alerts generated dynamically appeared on the app.

This project was an individual. I focused on integrating the backend with the front end and ensuring smooth API communication; other aspects included designing the database schema and creating user-friendly views in SwiftUI. Working on this project has provided valuable experience in mobile app development, database design, and debugging real-world issues.