Mobile Application Development Assignment 2 Report

Name: Kunal Pandya
Student Number: 100792272
GitHub Repository: https://github.com/kunalpanda/Mobile_Dev_Assign_3

Introduction

This report outlines the design and development of the Mobile Application Development Assignment 3 project. The purpose of this assignment was to build a Food Ordering mobile application using Flutter and Dart that integrates a local SQLite database with full Create, Read, Update, and Delete (CRUD) functionality. The application allows users to manage a collection of food items, create budget centric meal plans, and query order plans by date. This project demonstrates practical use of persistent data storage, real-time validation, and responsive UI design within a cross-platform mobile environment developed entirely in Flutter.

Activities Overview

The application consists of multiple activities that perform task related operations in an straightforward manner, enhancing readability and maintainability.

HomeScreen serves as the central dashboard that displays real-time statistics including the total number of food items and order plans in the database. It provides descriptive navigation cards for all three main features of the application. The screen automatically refreshes statistics when returning from other screens, ensuring users always see up to date information.

CreateOrderScreen enables users to plan their daily meals. Users select a date with an interactive date picker, set their target daily budget, and choose food items from the full menu. The screen implements smart budget validation that dynamically disables food items if selecting them would exceed the target budget, providing immediate visual feedback. A progress bar displays budget usage in real-time, and a summary section shows the number of selected items and remaining budget. The screen replaces existing order plans if one is already saved at the selected date.

ViewOrdersScreen provides a query feature that allows users to search for order plans by date. Users select a date and tap the search button to retrieve the meal plan for that specific day. The screen displays detailed information including each food item name and its associated cost, along with a summary card showing the total number of items, total cost, target budget, and remaining amount. A delete button allows users to remove the entire order plan with a confirmation dialog.

ManageFoodScreen implements full CRUD operations for the food item database. Users can view all food items in an organized list with clear pricing information. A floating action button opens a dialog for adding new food items with validated input fields for name and cost. Each food item card includes edit and delete buttons, allowing users to modify existing items or

remove them from the database. The delete operation includes a confirmation dialog that warns users about cascade deletion of related order plans, ensuring data integrity.

Each activity uses consistent design principles and follows the same layout style, maintaining a clean, user-friendly interface. All CRUD operations are responsive and give immediate visual feedback via the snack bar feature. Asynchronous operations are handled consistently using async/await syntax for all database calls, ensuring responsive UI behavior and preventing blocking operations or timeout triggers in case of any race conditions.

Database Integration

The database layer is implemented using the sqflite package, which provides a structured approach to local data management in Flutter. The system follows a singleton pattern through the DatabaseHelper class, ensuring a single shared instance across the entire application. The database consists of two tables with a foreign key relationship. The food_items table stores the id, name, and cost for each menu item, while the order_plans table maintains id, date, targetCost, and foodItemId columns. The foreign key relationship between order_plans and food_items includes CASCADE DELETE functionality, ensuring that when a food item is deleted, all related order plan entries are automatically removed to maintain referential integrity.

The data models use dedicated classes for FoodItem and OrderPlan with proper encapsulation and methods for database operations. Each model includes toMap methods for converting objects to database-compatible maps and factory constructors (fromMap) for creating objects from database query results. The OrderPlan model supports both simple queries and JOIN operations, with optional fields for foodItemName and foodItemCost that are populated when retrieving order details with associated food item information.

Additional functionality:

The application incorporates several key enhancements that elevate user experience and ensure reliable operation. These features work together to create a robust, user-friendly application:

- **Smart Budget Validation**: Real-time feedback system that continuously calculates item totals and automatically disables food items that would exceed the target budget. Visual cues include disabled checkboxes, grayed-out text, and lock icons on items outside the user allocate budget, while a color-coded progress bar turns red when budgets are exceeded. The red bar is not an expected behavior rather a fail safe if the user somehow manages to select items beyond their allocated budget.
- **Efficient Database Queries**: JOIN-based query operations retrieve complete order plan information in single database calls. The getOrdersForDate method performs INNER JOIN operations between order_plans and food_items tables, minimizing database overhead.

- ➢ **Comprehensive Error Handling**: Try-catch blocks wrap all asynchronous database operations with user-friendly SnackBar error messages. Mounted checks prevent widget disposal errors, loading states provide visual feedback during operations by implementing a loading screen, empty states guide user actions, and confirmation dialogs protect against accidental data loss in destructive operations.
- ➢ **Consistent Visual Design**: Simplistic theme maintains unified visual identity across all screens through deep purple primary colors for app bars and buttons, soft lavender backgrounds for comfortable viewing, cyan accents for pricing and interactive elements creating a polished, cohesive experience.

Conclusion

This project successfully demonstrates the integration of persistent local data storage with responsive user interface design in a cross-platform Flutter mobile application. The SQLite database implementation ensures reliable and efficient data management with proper normalization, foreign key relationships, and cascade operations. The smart budget validation system enhances UX by preventing errors before they occur, while the comprehensive CRUD functionality provides users with complete control over their food ordering data. The modular, screen-based structure combined with consistent theming, robust error handling, and professional coding practices results in a cohesive, functional application that meets all assignment objectives and demonstrates best practices in mobile app development.