

Final Report

SP-05-Red-Grocery List App CS 4850-Section 03: Fall 2025 Sharon Perry Dec 8, 2025

Website Link: <https://sp-05-grocery-list.github.io/>

GitHub Link: <https://github.com/AmanCantCode/SP05-Grocery-List.git>

Team Members:

| Roles | Name | Role | Cell Phone / Alt Email |
|--------------------------|--------------|---|--|
| Documentation | Drew | Responsible for project reports, risk assessment, and SDLC documentation. | 770-283-0379 prowlersdc@gmail.com |
| Documentation | Jose | Creates design diagrams, screen mockups, and compiles final report package. | 678-761-4883 gjose8120@gmail.com |
| Developer | Tyler | Focus on feature integration (grocery list) and testing. | 470.363.0302 tyler.chetty7@gmail.com |
| Developer | Aman | Develop basic app structure and UI. Set up user authentication. | 404.884.4149 amanbhayani608@gmail.com |
| Project Owner or Advisor | Sharon Perry | Facilitate project progress; advise on project planning and management. | 770.329.3895 Sperry46@kennesaw.edu |

Stats and Status

| | |
|------------------|---|
| Lines of Code | 3000 |
| Components/Tools | Flutter, Supabase, Figma, Android Studio, iOS/Android Emulators |
| Hours Estimate | 100 |
| Hours Actual | 100 |
| Status | Completed |

| | |
|--|-----------|
| Project Overview / Abstract | 4 |
| Project website | 4 |
| Deliverables | 4 |
| Design Considerations | 6 |
| General Constraints | 7 |
| Development Methods | 7 |
| Architectural Strategies | 7 |
| System Architecture | 9 |
| Detailed System Design | 10 |
| Classification | 10 |
| Definition | 11 |
| Constraints | 11 |
| Resources | 11 |
| Interface/Exports | 11 |
| Definitions and Acronyms | 13 |
| Assumptions | 13 |
| Design Constraints | 13 |
| Environment | 13 |
| User Characteristics | 14 |
| System | 14 |
| Functional Requirements | 14 |
| User Authentication | 14 |
| List Management | 14 |
| List Sharing & Collaboration | 14 |
| Item Management | 15 |
| Pricing Feature | 15 |
| Navigation | 15 |
| Non-Functional Requirements | 15 |
| Security | 15 |
| Capacity | 15 |
| Usability | 16 |
| Other | 16 |
| External Interface Requirements | 16 |
| User Interface Requirements | 16 |
| Hardware Interface Requirements | 16 |
| Software Interface Requirements | 16 |
| Communication Interface Requirements | 17 |
| Application Structure | 17 |
| User Interface | 17 |
| State Management | 17 |
| Authentication | 17 |

| | |
|---------------------------------------|-----------|
| Database Connection | 18 |
| Authentication | 18 |
| Database Structure | 18 |
| Row- Level Security | 18 |
| a. groups | 18 |
| b. group_members | 18 |
| c. grocery_items | 19 |
| SQL Function | 19 |
| Database Relationship | 20 |
| Project Setup | 20 |
| Install Flutter | 20 |
| Clone the Project Repository | 20 |
| Install Dependencies | 21 |
| Run the Application | 21 |
| Purpose | 21 |
| Scope | 21 |
| Test Objectives | 21 |
| Analysis of Scope and Testing | 22 |
| Progression of Test Objectives | 22 |
| Test Cases | 23 |
| Test Procedures | 24 |
| Test Environment | 25 |
| Software Test Report | 25 |
| Version Control Plan | 26 |
| Conclusion | 26 |
| Appendix | 28 |

Project Overview / Abstract

Managing household groceries can be a challenge, especially with larger families. Current solutions (notes, texting) lack efficiency. Duplicate items and store-specific goods can lead to confusion. This project aims to build a dedicated mobile app to solve these coordination issues within a defined user group (a “household”).

Primary Objectives

1. Real-Time Synchronization: Provide a sub-second sync engine to ensure all users in a household view a consistent list state
2. Implement a structured data model for grocery items to reduce ambiguity, allowing for features like smart grouping by store
3. Define a clear set of operations with strategies for resolving conflicting concurrent edits
4. Design a backend that keeps household data secured, authenticates users, and manages permissions (admin vs. standard user roles within household)

Scope

We aim to create a mobile app using a cross-platform framework (Flutter). This app will allow user authentication and household management (creating a household, joining, leaving). The app will have the ability to categorize and sort items at user’s convenience. The app will update in real time, ensuring users are always seeing the most recent list.

The app will not have predictive item addition or any complex machine learning features. The app will not be able to process payment. It will also not have location based services beyond user-defined store preferences.

Project website

The URL for the website we will develop for the project is (may be subject to change):

<https://sp-05-grocery-list.github.io/>

Deliverables

Phase 1: Core Functionality (MVP) User Account System

- ability to create an account and securely log in. Shared Grocery List
- add members to a list, enabling collaborative editing. Real-Time Updates
- instant syncing of grocery list items across all members’ devices. CRUD Operations
- add, edit, delete grocery items with immediate reflection. Basic UI/UX Mockups
- each team member creates an individual design. Group Screen Mockups & Information Flow Diagrams
- included in the Software Design Document (SDD).

Phase 2: Expansion / Monetization

Third-Party Integration: integration with grocery store APIs for item prices and availability.

Monetization Features: ads, premium subscription, or shopping recommendations.

Scalability Improvements: performance optimizations to handle larger user groups.

Technical Deliverables

Framework Tutorials Completion: all members complete the Flutter tutorial (Appendix B).

Source Code Repository: hosted on a GitHub Organization account (not individual accounts). Project Website: simple site containing project overview, documentation, and link to GitHub repo.

Unit and Integration Tests: validation of functionality and data synchronization.

Deployment Package: mobile build for both iOS and Android (demo-ready).

Documentation Deliverables

Software Requirements Specification (SRS): formal requirements documentation.

Software Design Document (SDD): includes:

- Architecture diagrams

- Group mockups

- Data flow diagrams

Final Report (and Draft): includes tutorials (Appendix B), mockups (Appendix C), testing evidence, and results.

User Manual / Quick Start Guide: for non-technical users to understand app functionality.

Presentation Deliverables

Final Presentation Video: overview of the project, demonstration of core features, and reflection on challenges/solutions.

Group Meeting Schedule Date/Time

The group meeting will be scheduled on Wednesdays every week at 8 PM.

Collaboration and Communication Plan

The group will meet synchronously (online) using Discord from wherever it is convenient.

Other Plans: Like Risk Assessment (if applicable)

Data Security & Privacy

Risk: Unauthorized access to user accounts or data leakage.

Impact: High: loss of user trust, potential data breach.

Mitigation: Enforce secure authentication, and proper database rules.

Scope Creep / Missed Deadlines

Risk: Spending too much time adding Phase 2 features (monetization, store integration) before MVP is complete.

Impact: High: final deliverables may not be ready on time.

Mitigation: Prioritize Phase 1 (MVP) and only expand if time allows.

Team Readiness (Learning Curve with Flutter)

Risk: Not all members complete the required Flutter training/tutorial.

Impact: High: uneven skill levels could slow development.

Mitigation: Require tutorial completion as proof (Appendix B) before coding begins.

Design Considerations

Goals & Priorities:

- Real-time collaboration: Minimal latency, consistent state across devices.
- Simplicity and UX: Easy onboarding, intuitive list interactions.
- Reliability and offline: App usable offline, syncs when online; conflict handling.
- Extensibility & modularity: Easy to add store integrations, analytics, monetization channels.
- Cross-platform: Single codebase for iOS & Android using Flutter.
- Scalability: The architecture should support multiple families, each with multiple users. Security: Authentication and row-level security (RLS) policies in Supabase will ensure that only authorized family members can view/edit their lists.

Key Scenarios:

- User creates an account, creates a family group, and invites members.
- A user adds an item to the list, and all members see the change instantly.
- A user edits or marks an item purchased, list updates in realtime.
- Offline changes are queued and synced when connection returns; conflicts resolved deterministically.
- Users browse nearby grocery stores via third-party data; optionally add items to list or place affiliate orders

Assumptions and Dependencies

Assumptions:

- The primary client is Flutter (Dart). Backend uses managed cloud services
- Users have smartphones with intermittent connectivity.
- Group sizes are modest per list (a typical group consists of less than 10 members).

Dependencies:

- Supabase services: Authentication, Postgres SQL- real-time database.
- Third-party grocery APIs (store product catalog/price/availability): vendor APIs or affiliate partners.
- Push notification provider (done with the help of supabase).
- Hardware: iOS and Android (mobile) devices.
- OS: iOS 14+ and Android 10+.

General Constraints

- Hardware/Software Environment: Runs on iOS/Android devices with limited CPU/GPU, battery, and memory
- End-User Environment: Users may be on the go or have limited interactivity (limited attention). Screen sizes may affect experience.
- Availability/Volatility of Resources: Mobile OS may kill the app (through crashes). Network and battery fluctuations.
- Standards Compliance: OAuth 2.0 / OIDC flows for social auth; platform store policies (e.g., “Sign in with Apple” on iOS when using third-party logins); secure transport (TLS).
- Interface/protocol requirements: Supabase Realtime uses WebSockets. DB access via HTTPS (REST/RPC) and PostgREST. Push notifications use APNs/FCM (future)
- Memory/Capacity Limits: Mobile memory pressure. Supabase plan quotas.
- Verification & Validation: Must validate RLS and concurrency; mobile UI across OS versions.
- User Management: Household roles (admin vs standard).
- Database: Supabase Postgres hosting.
- Performance: Real-time updates must propagate within <1s latency.
- Security: Use Supabase Row-Level Security (RLS) to ensure data isolation per group.
- Network: Requires active internet connection; offline mode may be a later enhancement.
- Interoperability: Must handle multiple OAuth providers (Google, Apple, etc.) via Supabase Auth.

Development Methods

Our approach is to use the Agile methodology with iterative sprints for a faster output. The team members work on their own branch and are later integrated into the main branch as needed. The tools that will be utilized are Flutter (for frontend), Supabase (for backend), and Github (for version control). We also think a code review would be beneficial by having at least one team member look at what the other has developed. Moreover, we will also refer to Flutter documentation to ensure using best practices.

Architectural Strategies

- Programming language and framework(s): Flutter (Dart) for cross-platform development with a single codebase, consistent UI, and near-native performance.

- Database & Backend: Supabase for authentication, PostgreSQL storage, and real-time synchronization. Considered Firebase but rejected for limited SQL flexibility.
 - System leverages Supabase for user authentication, PostgreSQL data storage, and real-time synchronization, thereby eliminating the need to develop a custom backend.
 - Flutter's Material and Cupertino widget libraries are utilized to provide a consistent, native-like user interface while minimizing redundant UI development.
 - Flutter's testing framework and GitHub Actions are integrated to support automated validation and continuous integration
- Future plans focus on expanding functionality and improving scalability:
 - Refine Third-Party API Integration: Redefine connections with grocery store APIs to display real-time item prices and availability.
 - Performance Improvements: Optimize database queries and network synchronization to support larger user groups with minimal latency.
 - Advanced Features: Add predictive item suggestions and analytics to further streamline the shopping experience.
- The system follows a mobile-first design using Flutter's Material Design (Android) and Cupertino (iOS) widgets to ensure a consistent, native-like experience across platforms.
 - Input Model: Users interact primarily through touch gestures, including taps, swipes, and long presses. Forms and dialogs are used for item creation, editing, and authentication.
 - Output Model: The application provides real-time visual feedback, automatically updating list views when items are added, edited, or marked as purchased. Status indicators ensure clarity during network operations.
 - Navigation: A bottom navigation bar or drawer offers quick access to core screens (Home, List Detail, Profile/Settings).
- Software Interfaces: The client application communicates with Supabase services over HTTPS for authentication and database operations, and uses WebSockets for real-time synchronization. The app runs on Android and iOS, leveraging their native SDKs through the Flutter framework to ensure consistent performance and UI rendering.
- Error detection and recovery:
 - Input Validation: User input is validated client-side to prevent malformed data from reaching the backend.
 - Network Error Handling: Connectivity issues are detected and communicated to the user through alerts and retry prompts.
 - Authentication Errors: Failed login or registration attempts return descriptive error messages while protecting sensitive information.
 - Data Consistency: Supabase's real-time synchronization automatically resolves conflicts by applying the most recent valid update.
- External databases and/or data storage management and persistence: Supabase (PostgreSQL) as the central data store for users, households, lists, and items. All data access is secured with Row-Level Security and transmitted over HTTPS.

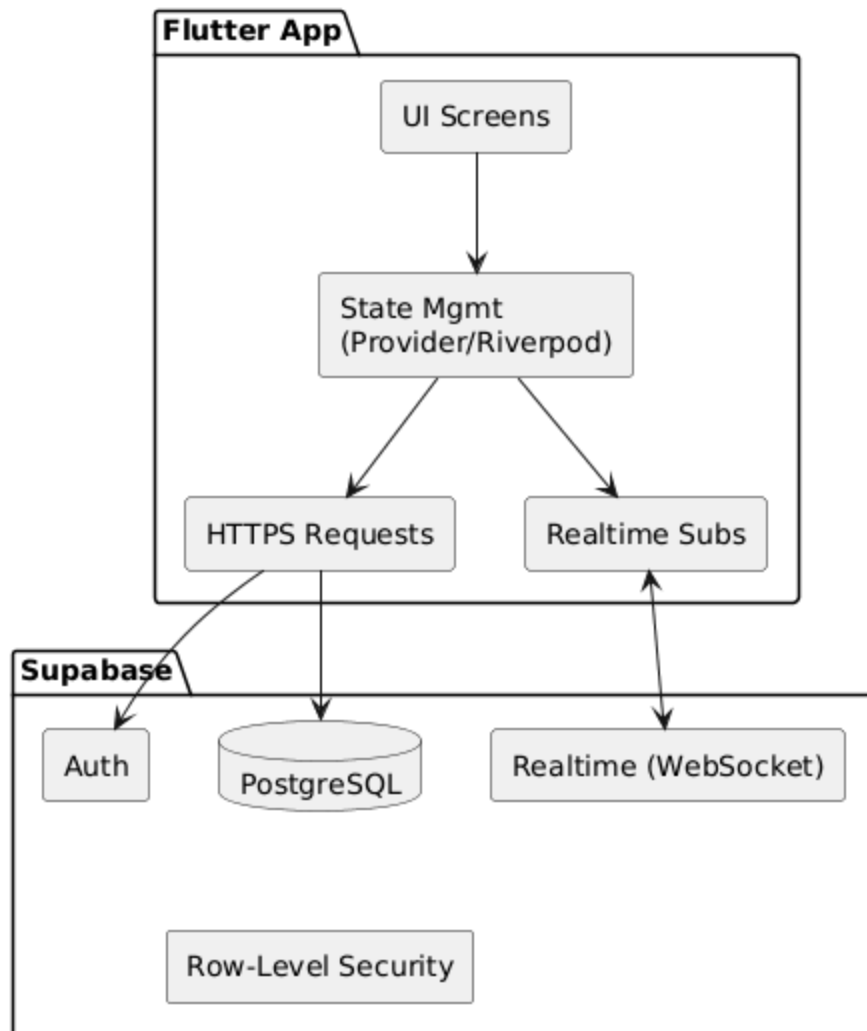
Real-time updates are handled through WebSocket subscriptions, ensuring sub-second synchronization across devices.

- **Concurrency & Synchronization:** System relies on Supabase's real-time subscriptions to synchronize list and item changes across all connected clients.
 - Updates are applied using a last-writer-wins strategy based on server timestamps to prevent conflicts.
 - On the client side, optimistic UI updates are used so users see changes immediately while the server confirms them in the background. Stream subscriptions are paused when a screen is backgrounded to reduce resource usage and resubscribed when resumed.

System Architecture

High-level components:

1. Frontend (Flutter):
 - a. Authentication screen (Sign in, Register, Google Auth)
 - b. Family or group management screens
 - c. Grocery List screen
 - d. Third party integration with other grocery stores screen.
2. Backend:
 - a. Auth: Handles user registration, login, and session management.
 - b. Database:
 - i. Users Table
 - ii. Families Table
 - iii. GroceryItems Table
 - c. Storage: For images and files.
3. External Services:
 - a. Grocery store(s) APIs.



Detailed System Design

Classification

- **Subsystem:** Subsystems included in the design of the app include a sign-in prompt at the beginning of the app.
- **Module:**
 - “Start a new family,” “Group 1,” “Family XYZ,” “My own list,” are all functions within the subsystem. These are functions where you can interact and start adding items to the specific list chosen.
- **Function:**
 - **addItem()**- Functions within the module include the **addItem()** where the program will add desired wants from the user.
 - **removeItem()**- In the case that the user will want to delete an item from a list they are able to do so by sliding left on the unwanted item.

- `updateItem()` - The grocery list will need to be updated in real time there for this method, it will be used to update
- Package/ File - Application is implemented using Flutter (framework) with Dart (language). The codebase is organized into separate Dart files for UI screens, business logic, and service integrations (Supabase API calls).

Definition

- The grocery list app is responsible for organizing items the user has added to a list. Furthermore, keeping a list of multiple users in a family or group. The app will be responsible for creating new grocery list, adding, updating, and deleting items
- Authentication System - The user will be prompted to sign in with email and password.
- Sharing Feature - When working on a shared list, multiple users are able to add items to the list. This feature allows this to be possible

Constraints

- Authentication Module: Requires internet; Supabase session tokens expire and must be refreshed.
- Supabase free plan:
 - Limited to 50,000 active users monthly (beyond needed, but limits future scalability)
 - Storage: 500 MB of storage is available, limiting how many grocery items and user accounts can be stored before needing an upgrade.
 - Row Level Security (RLS): needs to be enabled properly to prevent unauthorized access.
- API: Limited API for third party integration calls using a free version.

Resources

Authentication model/database: Supabase Auth and Postgres

Database Schema – Supabase PostgreSQL, RLS policies, triggers.

Third-Party Integration Module: Grocery store APIs.

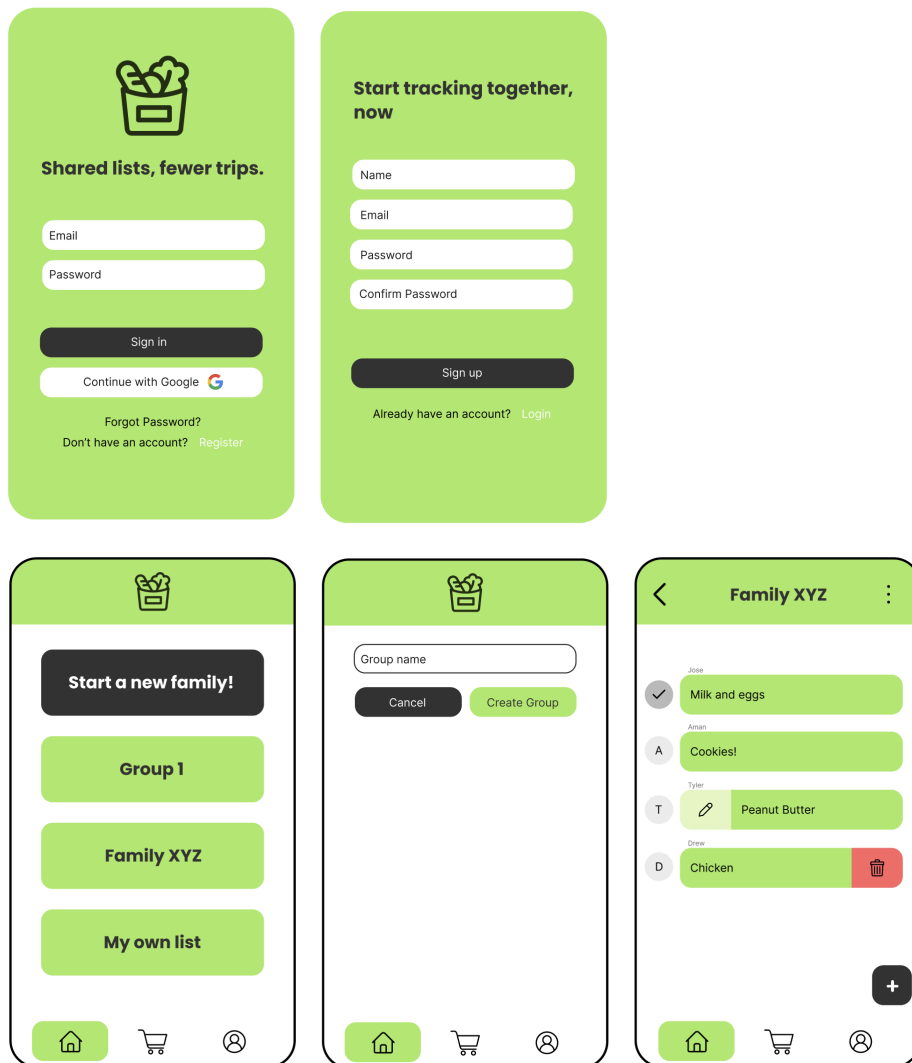
Grocery list module: Flutter ListView and Supabase tables.

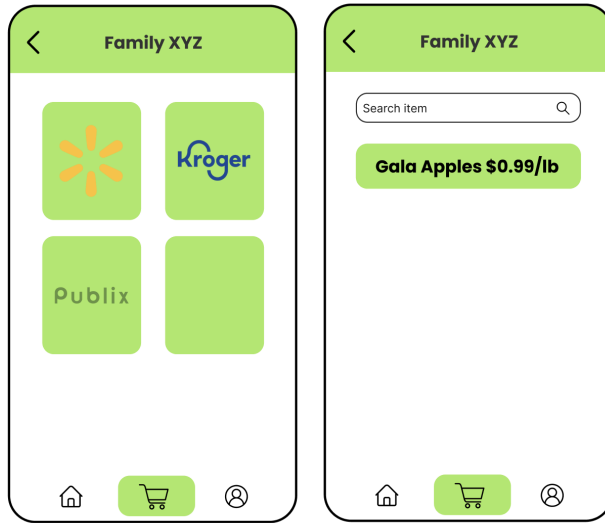
Interface/Exports

Services:

- `createList()` -This function is used to create a new list
- `deleteList()` - This function is used to delete a list
- `sharedList()` - This is used to access shared list where multiple users can add or delete items

- Items - objects that identify what items are which
- Int - integers are used to specify how much of an item is needed





Definitions and Acronyms

- SRS: Software Requirements Specification
- UI: User Interface
- UX: User Experience
- FAB: Floating Action Button
- Supabase: A backend as a service platform commonly used for authentication, databases, and cloud functions
- Widget: Fundamental building block of a flutter application's UI

Assumptions

- Users will have access to an Android or iOS device
- Users will have an internet connection for account creation, sharing lists, and synchronizing changes
- Users will have a valid email address for account registration and authentication
- The application will rely on a third-party backend service (Supabase) for real-time functionality

Design Constraints

Environment

This app works on Android and iOS. Internet access will be needed to update changes made by the user when dealing with a shared list. Otherwise accessing the list will not require internet access. Flutter is the framework in which this app is developed. This app is also developed using Dart.

User Characteristics

Target users include those who are looking for convenient ways to organize their shopping experience. These users will not have to be experts in navigating through their phone, the app is user friendly and minimal. These users will most likely be multitasking when shopping therefore needing a simple minimal app.

System

The system will be a client-server application. On the client-side, a mobile application built with the Flutter framework, running on Android and iOS systems. The server side will use Google Supabase, handling data storage, user authentication, and real-time synchronization.

Functional Requirements

User Authentication

- 3.1.1 The system shall allow a user to create a new account using an email address and a password.
- 3.1.2 The system shall allow a user to log in using their registered email and password.
- 3.1.3 The system shall provide a "Forgot Password" feature to allow users to reset their password via email.
- 3.1.4 The system shall allow a user to log out of their account.

List Management

- 3.2.1 The system shall allow an authenticated user to create a new grocery list.
- 3.2.2 The system shall allow a user to view a list of their owned and shared grocery lists on a home screen.
- 3.2.3 The system shall allow a user to edit the title of a list they own.
- 3.2.4 The system shall allow a user to delete a list they own.
- 3.2.5 The system shall allow a user to mark a list as "complete" or "in progress".

List Sharing & Collaboration

- 3.3.1 The system shall allow the owner of a list to share it with other users via their email address.
- 3.3.2 The system shall provide real-time synchronization of all changes (add, edit, delete items) for all users viewing a shared list.
- 3.3.3 The system shall display the name of the user who added or last modified an item.
- 3.3.4 The system shall allow the list owner to remove users from a shared list.

Item Management

- 3.4.1 The system shall allow a user to add a new item to a list.
- 3.4.2 The system shall allow a user to edit an item's properties: name, quantity (e.g., "2", "500g"), and notes (e.g., "organic", "for birthday cake").
- 3.4.3 The system shall allow a user to delete an item from a list.
- 3.4.4 The system shall allow a user to mark an item as "purchased" or "not purchased" (e.g., with a checkbox).
- 3.4.5 The system shall allow for quick item entry, suggesting recently added or frequently used items.

Pricing Feature

- 3.5.1 The system shall allow a user to manually enter a price for an item.
- 3.5.2 The system shall automatically calculate and display a running total cost for all items on the list.
- 3.5.3 The running total shall update in real-time as items are added, removed, or their prices/quantities are changed.

Navigation

- 3.6.1 The system shall provide a clear and consistent navigation structure (e.g., a bottom navigation bar or drawer) to access the Home (lists) screen, and user Profile/Settings.
- 3.6.2 The system shall allow users to tap on a list from the home screen to navigate to the detailed item view for that list.

Non-Functional Requirements

Security

- User passwords shall be hashed and salted before storage in the database (handled automatically by Supabase Auth).
- Data access shall be governed by security rules on the backend to ensure users can only read and write data for lists they own or are shared on.
- All communication between the client and server shall be encrypted using HTTPS/SSL.

Capacity

- The system shall be designed to handle a minimum of 10,000 concurrent users on shared lists. The scalability will be managed by the Supabase platform.
- The application should perform efficiently on devices with limited storage and memory.

Usability

- The User Interface shall be intuitive and require minimal instruction to use.
- The app shall follow modern Material Design guidelines on Android and Cupertino (iOS-style) guidelines where appropriate for a native feel on both platforms.
- Task completion times for core functions (e.g., adding an item) shall be under 3 seconds, including network latency.

Other

- Performance: The application should launch in under 2 seconds on a mid-range device. List scrolling should be smooth at 60fps.
- Reliability: The application should have a crash-free rate of over 99.5%. Data loss due to sync conflicts or poor network conditions should be prevented.
- Portability: The single Flutter codebase shall run correctly on all supported Android (API 21+) and iOS (iOS 11+) versions without major UI inconsistencies.

External Interface Requirements

User Interface Requirements

- Authentication Screen: A simple form for login and registration.
- Lists Home Screen: A scrollable list of the user's grocery lists, with a floating action button (FAB) to create a new list.
- List Detail Screen: The main shopping view, displaying items, checkboxes, quantities, prices, and a total. Features a FAB to add a new item.
- Add/Edit Item Dialog/Sheet: A modal form for entering item details.
- Sharing Settings Screen: A screen to add or remove users from a list via email.

Hardware Interface Requirements

- The application shall utilize the device's touch screen for all user input.
- The application shall require access to an internet connection (Wi-Fi or Cellular data) for synchronization and authentication features.

Software Interface Requirements

- Backend Services: The application shall interface with Supabase:
- Supabase Authentication: For user login and account management.
- Database: Primary SQL database for storing lists, items, and user relationships.
- Operating System: The application shall interface with the Android and iOS SDKs via the Flutter framework.

Communication Interface Requirements

Communication with Supabase services shall be accomplished using standard HTTPS REST APIs and WebSocket connections for real-time listeners, as provided by the official `supabase_core` and `supabase_auth` Flutter plugins.

Application Structure

The project follows the standard Flutter directory structure:

- `lib/` → Contains the application source code (`main.dart`)
- `android/`, `ios/`, `web/`, `macos/`, `windows/`, `linux/` → Platform-specific build files
- `test/` → Contains unit and widget tests
- `pubspec.yaml` → Defines dependencies and project metadata

The entry point of the application is located in `lib/main.dart`. The `main()` function runs the app and initializes the root widget, typically using Flutter's `MaterialApp` class.

User Interface

The UI is built using Flutter's widget tree. Core widgets used include:

- `Scaffold` for the main layout.
- `AppBar` for the title bar.
- `ListView` for displaying dynamic grocery items.
- `FloatingActionButton` for adding new items.
- `TextField` for user input.
- `CheckboxListTile` or similar widget for marking items as completed.

State management is handled locally using `StatefulWidget`, ensuring reactive updates whenever a grocery item is added, modified, or deleted.

State Management

State is managed using Flutter's native `setState()` function for local updates and Supabase's real-time listeners for synchronized changes. When a user adds or updates an item, the change is:

- Sent to Supabase via a REST or SQL call
- Reflected immediately in the local state
- Broadcasted to all connected clients in real time

Authentication

Supabase provides built-in email/password authentication:

- Users can register or log in to sync their personal grocery lists
- Supabase handles session tokens automatically
- Upon login, the app fetches only that user's grocery items

Database Connection

The Grocery List application uses a Supabase database which is hosted in the cloud. It stores product, customer, and order data. The application connects to Supabase using an API and secure API keys.

Authentication

- Uses Supabase's built in authentication system
- Users register with name, email, and passwords
- Supports Google login for sign in
- User credentials are securely stored and managed by Supabase Authentication

Database Structure

- **Groups:** Stores information about each shared grocery group
 - Primary Key (PK): group_id
 - Foreign Key (FK): created_by - auth.user
- **Group_members:** Links users to specific groups and defines role
 - Primary Key (PK): group_id, user_id
 - Foreign Key (FK): group_id, user_id
- **Grocery_items:** Stores the grocery list items to a group
 - Primary Key (PK): id
 - Foreign Key (FK): group_id

Row- Level Security

Supabase's Row-Level Security policies ensure users can only access data they are authorized to view or modify.

a. groups

- Select: All users can view groups
- Select: Users can view groups with invite code
- Insert: Users with an invite code can join

- Update: Admins can update group details
- Delete: Admins can delete the group

b. group_members

- Select: User can view their group membership and role
- Insert: Users can self-insert when joining a group
- Delete: Users or admins can remove themselves or others from the group

c. grocery_items

- Insert: Users can add items to the list
- Select: Users can view all items in the group's list
- Update: Users can edit their own items; admins can edit all items
- Delete: Users can delete their own items; admins can delete all items

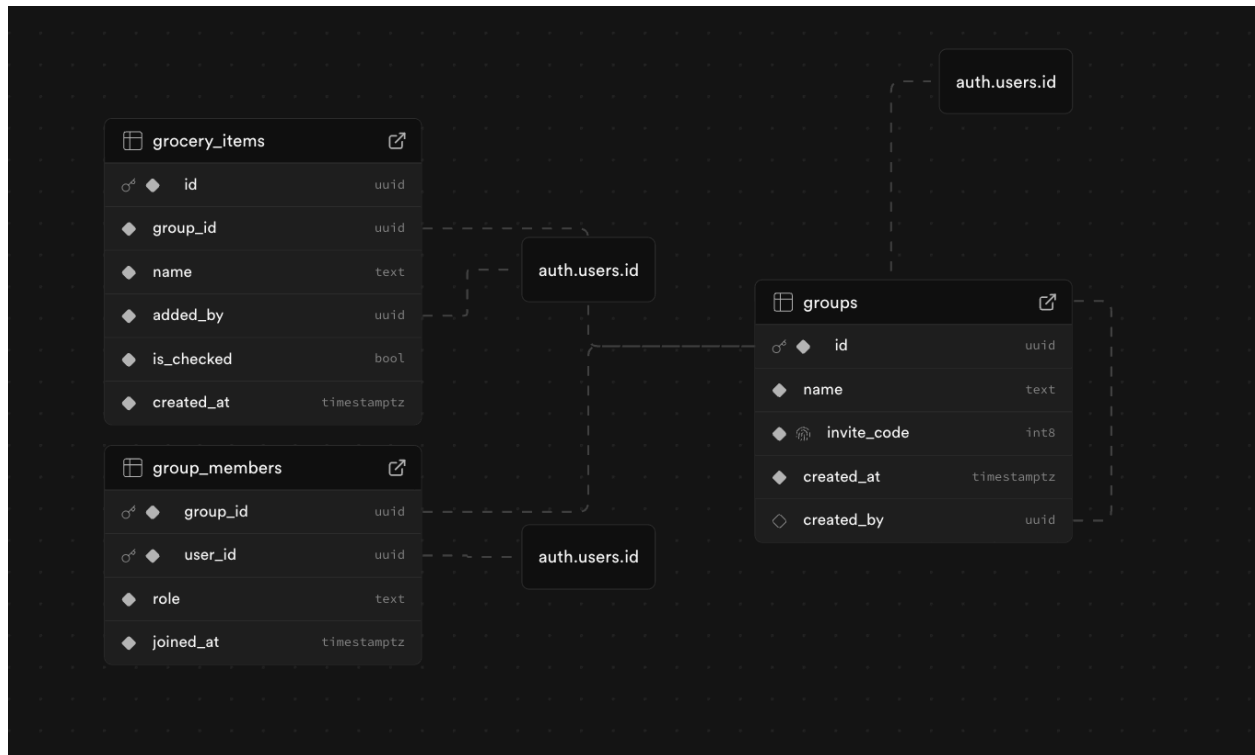
SQL Function

Supabase Remote Procedure Calls (RPC) are used to simplify complex logic and ensure secure data handling directly from the client.

- Create_new_group_and_admin - Creates a new entry in groups and assigns the first user as the group admin in group_members
- Get_group_grocery_items- returns all grocery items for a given group_id, including each user's display name from user's (auth) 'Display Name'

This configuration allows the Grocery List App to securely manage users, groups, and grocery items using Supabase's backend services. Authentication, access control, and SQL logic are all handled within Supabase, minimizing server-side complexity and improving security.

Database Relationship



This diagram depicts how the relational database works. The database contains three tables with specific attributes (columns) and the data that is actually stored in them (rows). Using primary and foreign keys specified in a previous section, the tables work with one another allowing the app to appropriately display data.

Project Setup

Install Flutter

- Download and install Flutter from the official website (<https://docs.flutter.dev/install>)
- Verify that Flutter is installed correctly by running a system check using the Flutter Doctor tool
- Make sure an IDE (Android Studio, Xcode, Visual Studio depending on platform) is installed and configured properly

Clone the Project Repository

- Obtain the GitHub repository link for the project (<https://github.com/AmanCantCode/SP05-Grocery-List>)

- Use a git client or the command line to clone the repository to your development environment
- Open the project folder in your preferred IDE

Install Dependencies

- Ensure that your terminal or IDE is pointed to the project directory
- Retrieve all necessary Flutter packages listed in the project's pubspec.yaml file
- Install proper emulator through IDE depending on desired platform
- Confirm that there are no dependency errors before continuing

Run the Application

- Use your IDE's run button or Flutter's run command to start the application
- Select your desired device or platform (Android, iOS, Windows, or Web)
- Wait for build process to complete and confirm application launches

Purpose

The purpose of the test plan is to verify and validate the functionality, reliability, and usability of our Flutter application; Shared Grocery List. Additionally, the plan aims to identify any defects in the application to ensure a consistent experience.

Scope

- User registration and authentication via Supabase
- Group creation, joining, and membership management
- CRUD operations for grocery items
- Food search and nutritional information display
- User profile management (name, avatar, email)
- Basic UI interaction and navigation

Test Objectives

- Ensure all users can register, login, and logout on both Android and iOS platforms.
- Ensure that all users can create, join, and leave/delete groups.
- Verify that all group members can add, edit, and delete grocery items.
- Ensure that the third-party integration functions accordingly where users can search food items properly and display its appropriate nutritional information.
- Confirm users can edit their personal info through the app.
- Ensure that the UI is consistent on both Android and iOS platforms.

Analysis of Scope and Testing

The testing covered all major functional and non-functional components of the application across both Android and iOS platforms. Testing focused on end to end user workflows to ensure that all features behaved as expected. Functional testing includes user authentication, group management, and grocery item operation. Additionally, the scope extends to validating third party nutritional data integration, ensuring that food searches return accurate and complete information

Non-functional aspects such as UI consistency, usability, navigation clarity, and cross platform behavior were also within scope. Testing confirmed that both platforms delivered a consistent user experience regardless of device.

Progression of Test Objectives

- Began by validating core authentication features including user registration, and login on both Android and iOS
- Tested group creation , joining, leaving, and deletion. Ensured both platforms handled group membership
- Proceeded to test the ability to add, edit, and delete grocery list items within their groups
- After, testing the external food search integration. Confirmed users can search food items, retrieve results, and view accurate nutritional information from the third party provider
- Validated personal information editing once other major functionalities are stable, ensuring user settings behave properly across platforms
- Lastly, performed UI/UX consistency checks across Android and iOS devices. This ensures the layout, design elements, and navigation remain and intuitive

Test Cases

| Test Case ID | Description | Expected Results |
|--------------|-------------------------|--|
| TC-001 | User Registration | User successfully registers their account to use for logging in. |
| TC-002 | User Login | User successfully logs in and navigates to Groups Page. |
| TC-003 | Google Login | User successfully logs in with a Google account, navigates to Group Page. |
| TC-004 | Navigation | User can access all pages of the app, no crashes. |
| TC-005 | User Logout | User is logged out of their account and navigated to Login Page |
| TC-006 | Create Group | New group is created, invite code displayed, group listed on Groups Page |
| TC-007 | Join Group | User is added to a group with an invite code, and displays new group on Groups Page. |
| TC-008 | Add Grocery Item | Item appears in grocery list for all group members. |
| TC-009 | Edit Grocery Item | Item is updated from grocery list for all group members. |
| TC-010 | Delete Grocery Item | Item is removed from grocery list for all group members. |
| TC-011 | Search Nutritional Info | Nutritional info displayed for the food item. Handles multiple food items. |
| TC-012 | Edit Display Name | Display name updated in profile and persists across sessions. |

Test Procedures

| Test Case | Test Procedure (Steps) |
|---------------------------------|---|
| TC-001: User Registration | <ol style="list-style-type: none">1. Open App2. Navigate to Register page using Register button3. Enter Name, Email, Password4. Tap Register |
| TC-002: User Login | <ol style="list-style-type: none">1. Open app2. Enter valid Email/Password3. Tap Login |
| TC-003: Google Login | <ol style="list-style-type: none">1. Open App2. Tap Continue with Google |
| TC-004: Navigation | <ol style="list-style-type: none">1. Login2. Tap between pages using nav bar3. Tap on groups4. Tap back button to verify |
| TC-005: User Logout | <ol style="list-style-type: none">1. Navigate to User Account Page2. Tap Logout |
| TC-006: Create Group | <ol style="list-style-type: none">1. Click "Start a New Family!" Button2. Tap Create group3. Enter group name4. Tap create |
| TC-007: Join Group | <ol style="list-style-type: none">1. Click "Start a New Family!" Button2. Tap Join group3. Enter Invite code4. Tap join |
| TC-008: Add Grocery Item | <ol style="list-style-type: none">1. Click '+' floating action button2. Type grocery item name3. Tap save |
| TC-009: Edit Grocery Item | <ol style="list-style-type: none">1. Locate item to edit2. Swipe left on item's name3. Type the updated name4. Tap Save |
| TC-010: Delete Grocery Item | <ol style="list-style-type: none">1. Locate item to edit2. Swipe right on item's name |
| TC-011: Search Nutritional Info | <ol style="list-style-type: none">1. Navigate to Nutrition Page2. Click and search bar |

| | |
|---------------------------|--|
| | <ol style="list-style-type: none">3. Enter food name4. Tap Search icon |
| TC-012: Edit Display Name | <ol style="list-style-type: none">1. Navigate to User Account Page2. Click Edit Name button3. Type in new name4. Tap Save |

Test Environment

In order to test the application, the Android (Google Pixel 9 Pro) and iOS (iPhone 17 Pro) emulators were used as they simulate the experience of an actual device.

Operating System Versions:

- Android 15 (latest version)
- iOS 18 (latest version)

Network Conditions:

- WiFi connection
- Cellular/LTE simulation
- Offline/limited connectivity scenarios

Third Party Integration:

- Access to nutritional data API
- API keys configured for test environment

Tools/Frameworks:

- Android Studio emulator
- Logging/monitoring tool (Supabase)

Software Test Report

This section summarizes the results of executing all planned test cases for the Shared Grocery List app.

| Area Tested | Result | Severity | Notes |
|------------------------|--------|----------|--|
| User Authentication | Passed | Critical | Email/Password and Google login work on both platforms |
| Group Management | Passed | Moderate | User can create, join, and view groups with no sync issues |
| CRUD for Grocery Items | Passed | Moderate | All list updates propagate instantly |

| | | | |
|--------------------|--------|----------|---|
| | | | across users |
| Nutritional Search | Passed | Minor | Search returns multiple food matches. No UI crashes |
| Profile Management | Passed | Minor | Name updates persist after app restart |
| Navigation/UI | Passed | Critical | No broken routes or navigation loops |
| Logout | Passed | Critical | Sessions clear correctly; user returned to login screen |

Version Control Plan

We will utilize Git as our version control system, hosted on a centralized platform (GitHub). The workflow will be based on the Feature Branch Workflow, which is well-suited for collaborative development and continuous integration. The main branch will always reflect a stable, production-ready state. All new work, whether it's a feature, bug fix, or chore, will be performed in a dedicated, short-lived branch created from the latest main branch. This isolation ensures that the main branch remains stable and allows for parallel development without conflicts.

Each feature branch will be named descriptively using a convention like feature/[ticket-number]-[short-description] (e.g., feature/GA-12-add-auth-ui). Development will be committed in small, logical units with clear and concise commit messages. To integrate changes, developers will open a Pull Request (PR). Each PR must pass all automated CI checks (linting, testing, builds) and require at least one approved code review from a peer before it can be squashed or rebased and merged back into the main branch. This process enforces code quality, facilitates knowledge sharing, and maintains a clean, linear project history.

Conclusion

The development of the Grocery List App provided our team with an opportunity to design, plan, and implement a real-time collaborative mobile application from the ground up. Throughout the project, we established a strong foundation using Flutter for cross-platform development and Supabase for authentication, database management, and synchronization. By focusing on core

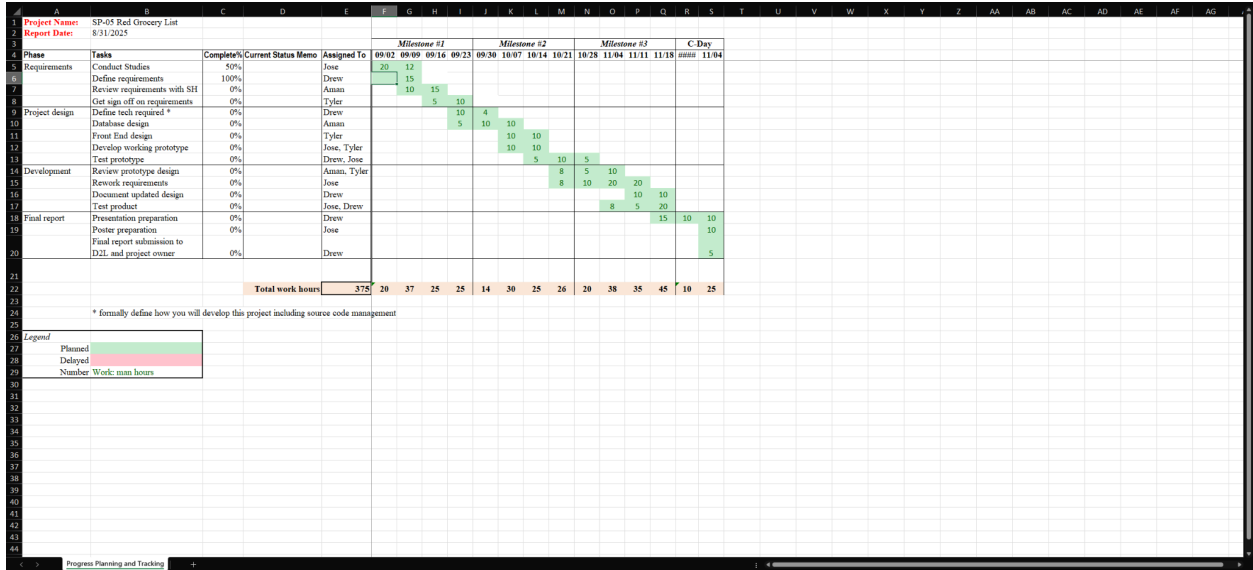
features, such as user authentication, shared list management, and real-time updates, we successfully created an app that addresses the primary goal of improving household coordination and reducing grocery-related confusion. The design process, from mockups to database schema creation, helped ensure a clean, intuitive, and scalable user experience.

In completing this project, we also gained practical insight into the challenges of mobile application development. Real-time synchronization, database security with row-level security, and state management required careful planning and repeated testing to ensure consistent behavior across devices. Our Agile approach enabled us to work iteratively, respond quickly to issues, and integrate features through structured version control practices. Each team member contributed to the system's architecture, documentation, testing, and UI development, leading to a well-rounded and functional product. These combined efforts also helped us better understand professional workflows such as pull requests, code reviews, and distributed collaboration.

Looking forward, the project has several promising opportunities for future expansion. Adding predictive item suggestions and enhancing the pricing system could significantly increase the app's utility. Additional features, such as improved offline support, push notifications, and household analytics, would make the app even more robust and appealing to larger user groups. Overall, the Grocery List App demonstrates a successful implementation of a real-time, user-centered solution and reflects our team's growth in software engineering practices, technical proficiency, and collaborative development.

Appendix

Project Schedule and Task Planning



Here is the proof of the team's Flutter tutorial completion:

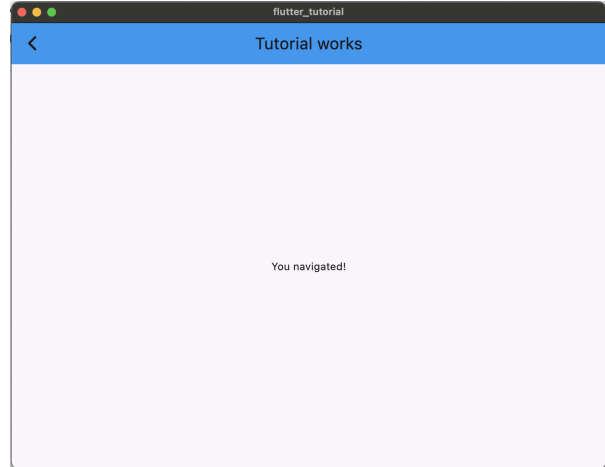
Drew:

The image shows a side-by-side comparison of Flutter development. On the left is a code editor window displaying the Dart code for a simple Flutter app. The code defines a `MyApp` class that extends `StatelessWidget` and a `HomeScreen` class that extends `StatelessWidget`. The `MyApp` class has a `build` method that returns a `MaterialApp` widget, and the `HomeScreen` class has a `build` method that returns a `Scaffold` widget. The `Scaffold` widget contains an `AppBar` with a title 'Flutter is Fun!', a green background color, and a 'Navigate' button. The button is a blue pill-shaped button with the text 'Navigate'. On the right is a screenshot of the Flutter application running on an Android emulator. The app has a green header bar with the text 'Flutter is Fun!'. Below the header is a large white area with a blue button labeled 'Navigate'. The bottom of the screen shows a black Android navigation bar with a back arrow and the text 'Flutter Tutorial'.

```

41   ); Scaffold
42   }
43 }
44
45 class AboutScreen extends StatelessWidget {
46   @override
47   Widget build(BuildContext context) {
48     return Scaffold(
49       appBar: AppBar(
50         title: Text('Tutorial works'),
51         backgroundColor: Colors.blue,
52       ),
53       body: Center(child: Text('You navigated!')),
54     ); Scaffold
55   }
56 }
57
58

```



Tyler:

```

1  import 'package:flutter/material.dart';
2
3  void main() {
4    runApp(const TutorialApp());
5  }
6
7  class TutorialApp extends StatelessWidget {
8    const TutorialApp({super.key});
9
10   @override
11   Widget build(BuildContext context) {
12     return MaterialApp(
13       debugShowCheckedModeBanner: false,
14       title: 'Flutter Tutorial',
15       home: const TutorialHomePage(),
16     ); MaterialApp
17   }
18 }
19
20 class TutorialHomePage extends StatefulWidget {
21   const TutorialHomePage({super.key});
22
23   @override
24   State<TutorialHomePage> createState() => _TutorialHomePageState();
25 }
26

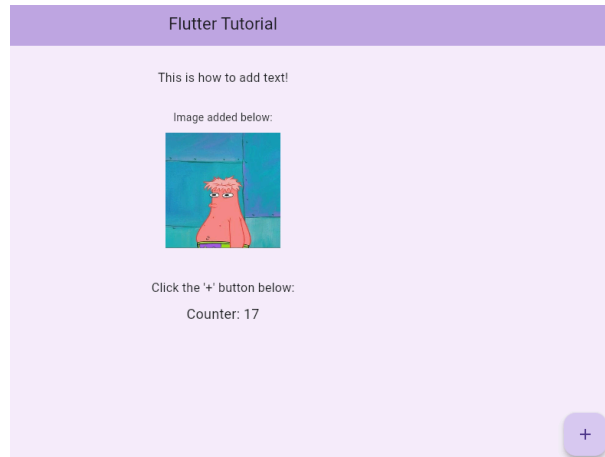
```



```

27 class _TutorialHomePageState extends State<TutorialHomePage> {
28   int count = 0;
29
30   @override
31   Widget build(BuildContext context) {
32     return Scaffold(
33       backgroundColor: const Color(0xFFFFE0FA),
34       appBar: AppBar(
35         backgroundColor: const Color(0xFFB0E0E6),
36         title: const Text('Flutter Tutorial'),
37         centerTitle: true,
38       ),
39       body: Center(
40         child: Padding(
41           padding: const EdgeInsets.symmetric(vertical: 30),
42           child: Column(
43             mainAxisAlignment: MainAxisAlignment.start,
44             children: [
45               const Text(
46                 'This is how to add text!',
47                 style: TextStyle(fontSize: 16),
48               ),
49               Text
50             ],
51             const SizedBox(height: 30),
52           ),
53         ),
54       ),
55     ); Scaffold
56   }
57 }
58

```



```

33     const Text(
34       "Image added below:",
35       style: TextStyle(fontSize: 14),
36     ), Text(
37
38     const SizedBox(height: 10),
39     SizedBox(
40       height: 150,
41       width: 150,
42       child: Image.network(
43         "https://i.imgur.com/ZI6umjx.jpeg",
44         fit: BoxFit.cover,
45       ), Image.network(
46     ), SizedBox
47
48     const SizedBox(height: 40),
49
50     const Text(
51       "Click the '+' button below:",
52       style: TextStyle(fontSize: 16),
53     ), Text(
54
55     const SizedBox(height: 10),
56
57     Text(
58       "Counter: $count",
59       style: const TextStyle(fontSize: 18),
60     ),
61   ),
62 ), Column(
63 ), Padding(
64 ), Center(
65
66   floatingActionButton: FloatingActionButton(
67     backgroundColor: const Color(0xFFD9C8F0),
68     onPressed: () {
69       setState() {
70         count++;
71       };
72     },
73     child: const Icon(Icons.add),
74   ), FloatingActionButton
75 ); Scaffold
76 }
77 }
78 }

```

```

76
77     Text(
78       "Counter: $count",
79       style: const TextStyle(fontSize: 18),
80     ), Text(
81     ),
82   ), Column(
83   ), Padding(
84   ), Center(
85
86   floatingActionButton: FloatingActionButton(
87     backgroundColor: const Color(0xFFD9C8F0),
88     onPressed: () {
89       setState() {
90         count++;
91       };
92     },
93     child: const Icon(Icons.add),
94   ), FloatingActionButton
95 ); Scaffold
96 }
97 }
98 }

```

Jose

```
import package:flutter/material.dart;

void main() {
  runApp(MyApp);
}

class MyApp extends StatelessWidget {
  value: build(App()
    return MaterialApp(
      title: 'Simple Flutter App',
      theme: ThemeData(primarySwatch: Colors.blue,
    )
    home: HomePage
  )
}

class HomePage extends StatefulWidget {
  createState() {return center();}
  int counter = 0;
  incrementCounter() { {
    setState(() { counter: ++1;
  })
}
}
in Scaffold() {
  return Scaffold(
    appBar: AppBar(title: 'Simple Flutter Tutorial'),
    center: Center(
      child: Text('Counter: $counter'),
      floatingActionButton: FloatingActionButton(
        onPressed: incrementCounter,
        icon: Icons.add,
      ),
    ),
  );
}
```

Simple Flutter Tutorial

Counter: 0

```

1  import package:flutter/material.dart;
2
3  void main() {
4    runApp(MyApp());
5  }
6
7  class MyApp extends StatelessWidget {
8    // This widget is the root of your application.
9    @override
10   Widget build(BuildContext context) {
11     return MaterialApp(
12       title: 'Simple Flutter App',
13       theme: ThemeData(primarySwatch: Colors.blue),
14       home: HomePage(),
15     );
16   }
17
18   class HomePage extends StatefulWidget {
19     @override
20     State<HomePage> createState() {
21       return _HomePageState();
22     }
23
24     class _HomePageState extends State<HomePage> {
25       int counter = 0;
26
27       void _incrementCounter() {
28         setState(() {
29           counter = counter + 1;
30         });
31       }
32
33       @override
34       Widget build(BuildContext context) {
35         return Scaffold(
36           appBar: AppBar(
37             title: Text('Simple Flutter Tutorial'),
38           ),
39           body: Center(
40             child: Text('Counter: $counter'),
41           ),
42           floatingActionButton: FloatingActionButton(
43             onPressed: _incrementCounter,
44             tooltip: 'Increment',
45             child: Icon(Icons.add),
46           ),
47         );
48       }
49     }
50   }

```

Simple Flutter Tutorial

Counter: 1


```

1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(const MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   const MyApp({super.key});
9
10  @override
11  Widget build(BuildContext context) {
12    return MaterialApp(
13      title: 'Flutter Demo',
14      theme: ThemeData(
15        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
16        useMaterial3: true,
17      ),
18      home: const MyHomePage(title: 'Flutter Tutorial'),
19    );
20  }
21 }
22
23 class MyHomePage extends StatefulWidget {
24   const MyHomePage({super.key, required this.title});
25
26   final String title;
27
28   @override
29   State<MyHomePage> createState() => _MyHomePageState();
30 }
31
32 class _MyHomePageState extends State<MyHomePage> {
33   String _message = "";
34
35   void _incrementCounter() {
36     setState(() {
37       _message = "You just clicked this button";
38     });
39   }
40 }

```

```

41 @override
42 Widget build(BuildContext context) {
43   return Scaffold(
44     appBar: AppBar(
45       backgroundColor: Theme.of(context).colorScheme.inversePrimary,
46       title: Text(widget.title),
47     ),
48     body: Center(
49       child: Column(
50         mainAxisAlignment: MainAxisAlignment.center,
51         children: <Widget>[
52           Text("This is how to add text!\n\n"),
53           Text("Image added below: "),
54           Image.asset("assets/image/flutter_img.jpeg",
55             height: 150,
56             width: 150,
57           ),
58           SizedBox(height: 50),
59           const Text(
60             'Click the '+' button below:',
61             style: TextStyle(fontSize: 20),
62           ),
63           const SizedBox(height: 20),
64
65           if (_message.isNotEmpty)
66             Text(
67               _message,
68               style: Theme.of(context).textTheme.headlineMedium,
69               textAlign: TextAlign.center,
70             ),
71         ],
72       ),
73     ),
74   ),
75   floatingActionButton: FloatingActionButton(
76     onPressed: _incrementCounter,
77     tooltip: 'Click',
78     child: const Icon(Icons.add_sharp),
79   ),
80 ),
81 );
82 }
83 }

```

Flutter Tutorial

DEBUG

This is how to add text!

Image added below:



Click the '+' button below:



