SP-5 Purple Grocery List App

Course Information: 4850-03, Fall 2025
Date: 11/30/25
Instructor: Sharon Perry

Project Team

Carinne Tzurdecker Documentation	Vanessa Espinoza Documentation	Shane Hazeur Developer	Piao Hou Developer

SP-5 Purple Grocery List App

Test Plan

The purpose of this Software Test Plan is to describe the testing objectives, scope, test cases, environment, and procedures for the SP-5 Purple Grocery List

Revision History

Date	Version	Author	Description
11/30/2025	1.0	Vanessa Espnoza	Initial Draft
11/30/2025	1.1	Carinne Tzurdecker	Added section 7 Test Strategy

Table of Contents

1. Overview

1.1. Purpose

- The scope, focus areas, and testing objectives for the current MVP
- The responsibilities of the project testing team
- The testing strategy for validating core features such as authentication, recipe search, meal planning, and grocery list generation
- The entry and exit criteria for each testing stage
- The basis for estimating the testing effort
- Any risks, issues, assumptions, or dependencies that may affect testing activities
- The testing schedule and major milestones
- The deliverables produced as part of the testing process

1.2. Scope

The scope of this Software Test Plan includes:

- Testing all MVP features currently implemented in the SP-5 Purple Grocery List Application.
- Verifying core functionality**,** including authentication, onboarding, recipe search, recipe viewing, meal planning, grocery list management, AI chat responses, navigation, settings, and user profile configuration.
- Ensuring that user workflows (registration -> onboarding -> navigation -> feature use) operate correctly and consistently.
- Performing manual functional testing to confirm expected behavior across screens and supported platforms.
- Validating UI/UX flow**,** screen transitions, and correct handling of user inputs.
- Excluding any features that require backend services, advanced AI models, semantic search, nutrition analysis, or other unimplemented functionality

2. Testing Summary

2.1. Scope of Testing

2.1.1. In scope

- Complete authentication flow
- User onboarding and profile setup
- Recipe search with filtering
- Basic meal planning interface
- Grocery list management
- AI chat interface
- Settings and account management
- Navigation and routing

2.1.2. Out of scope

- Real backend authentication or JWT validation
- Database integration (PostgreSQL or pgvector search)
- AI-powered recipe recommendations or meal-plan optimization
- Advanced natural language processing beyond simple keyword responses
- Real-time nutrition analysis or dynamic nutrition tracking
- Cloud storage for user data, meal plans, or grocery lists
- Admin recipe import or recipe curation features
- Pantry management and budget optimization
- Allergy enforcement logic
- Offline mode, caching, or advanced performance features
- Accessibility testing (screen reader support, WCAG compliance)
- Security testing, load testing, or performance testing
- Unit testing, widget testing, or integration testing

3. Analysis of Scope and Test Focus Areas

3.1. Release Content

- Mock user authentication (email/password)
- User onboarding and profile setup
- Recipe search with category and diet filters
- Recipe details (ingredients, instructions, mock nutrition info)
- Adding recipes to meal plan slots
- Grocery list generation from meal plans
- Editing and checking off grocery list items
- AI chat with simple keyword responses
- Bottom navigation and routing
- Basic settings and profile management
-

3.2. Regression Testing

Regression testing will be performed to ensure that new changes to the application do not break existing functionality. Because the SP-5 Purple Grocery List Application has several connected UI features, any update may affect previously working screens or workflows.

For this release, regression testing applies to the following areas:

- User authentication (mock email/password login)
- User onboarding and profile setup
- Navigation and routing between all screens
- Recipe search and recipe detail views
- Meal plan creation and editing
- Grocery list generation, editing, and item removal
- AI chat interface (keyword-based responses)
- Settings and account management

Only front-end Flutter functionality will be included in regression testing.
Backend APIs, real authentication, database features, and AI-powered functionality are **not** part of regression testing for this MVP.

3.3. Platform Testing

For this release, testing will be performed only on the platform used during development. The SP-5 Purple Grocery List Application was built and tested as a Flutter Web application.

Hardware Used

- MacBook Pro (16-inch, 2019)
  - 2.6 GHz 6-Core Intel Core i7
  - Intel UHD Graphics 630
  - 16 GB 2667 MHz DDR4 RAM
  - macOS Sonoma 26.1

Software Environment

- Flutter SDK 3.3.4+
- Dart SDK 3.0.0+
- Google Chrome (latest version) — used for all functional and UI testing
- VS Code / Terminal for build and run commands

Tested Platform

- Web (Chrome browser) — All functional testing, navigation flows, and UI verification were executed using Flutter’s web renderer.

Testing focused solely on verifying UI behavior, navigation, state management, and local storage functionality within the Chrome browser environment on macOS.

4. Progression Test Objectives

This section details the progression test objectives that will be covered by the project team. Please note that this is at a high level. For large projects, a suite of test cases would be created which would reference directly back to this master.

This could be documented in bullet form or in a table similar to the one below, to assist with Requirements Traceability

Ref	Function	Test Objective	Evaluation Criteria	X-Ref	P
Function to be tested					
Test reference	Name of the function or sub-function being tested	The objective the test is trying to demonstrate	The criteria that will be evaluated to demonstrate the test is successful	Any cross references. For example, a functional requirement, a design document etc	The priority of the test
Repeat for each function					
T1	User Authentication	Ensure mock email/password login functions correctly	User can register, log in, and access home screen without an error	Req ID (Auth)	H
T2	Onboarding & Profile Setup	Validate onboarding screens and successful collection of user preferences	Onboarding flow completes	Req ID (Onboarding)	H
T3	Navigation & Routing	Confirms all tabs and routes transition correctly across the app	Home, Recipes, Meal Plan, Grocery, and Profile tabs, load expected screens	Req ID (Nav)	H
T4	Recipe Search & Details	Verify recipe filtering, category browsing, and recipe detail view	Recipes load from mock data; filters work; details display nutrition and instructions	Req ID (Recipes)	M

Ref	Function	Test Objective	Evaluation Criteria	X-Ref	P
T5	Meal Plan Creation	Validate creation of 3–7 day meal plans and adding/removing recipes	Meal plan saves, updates, and loads correctly	Req ID (MealPlan)	H
T6	Grocery List Management	Ensure grocery list generation, editing, and check-off functionality work	List groups ingredients, supports CRUD operations	Req ID (Grocery)	H
T7	AI Chat Interface	Validate chat input, keyword-based responses, and saved history	Messages send, mock responses display, chat history persists	Req ID (Chat)	L
T8	Settings & Account	Confirm updating user settings and logout behaviour	Toggles update local storage; logout clears user session	Req ID (Settings)	M

5. Progression Test Objectives

This section details the regression test objectives that will be covered by the project team. Please note that this is at a high level. For large projects, a suite of test cases would be created which would reference directly back to this master.

This could be documented in bullet form or in a table similar to the one below, to assist with Requirements Traceability

Ref	Function	Test Objective	Evaluation Criteria	X-Ref	P
Regression testing					
Test reference	Name of the function or sub-function being regression tested	The objective the test is trying to demonstrate	The criteria that will be evaluated to demonstrate the test is successful	Any cross references. For example previous regression suite or requirement document	The priority of the test
R1	User Authentication	Ensure login still works after updates	Valid login succeeds; errors show correctly	Auth-Req	H
R2	Onboarding & Profile Setup	Ensure onboarding screens and saved preferences still function	All slides load; preferences persist	Onboard-Req	H
R3	Navigation & Routing	Confirm that all tabs/routes still load correct pages	Home, Recipes, Meal Plan, Grocery, and Profile tabs, load without errors	Nav-Req	H
R4	Recipe Search & Details	Ensure recipe search and details still load correctly	Filters work; recipes load from mock DB	Recipe-Req	M

Ref	Function	Test Objective	Evaluation Criteria	X-Ref	P
R5	Meal Plan Creation	Ensure validating creation of 3–7 day meal plans and adding/removing recipes still works	Meal plan saves, updates, and loads correctly	MealPlan-Req	H
R6	Grocery List Management	Ensure grocery list generation, editing, and check-off functionality still work	Items persist; CRUD works	Grocery-Req	H
R7	AI Chat Interface	Ensure keyword-based mock responses work	Chat loads; mock responses appear without errors	Chat-Req	L
R8	Settings & Account	Ensures settings toggles and logout behaviour remain functional	Toggles save; logout clears session	Settings-Req	M

6. Other Testing

6.1. Security

Because the application runs only on Flutter Web (Chrome) and uses mock data with no real authentication**,** security testing is limited.

Scope:

- Verify that no sensitive data is stored or exposed (only mock JWT tokens)
- Ensure input fields do not crash the UI when invalid text is entered.
- Confirm that logout clears local session/state.

Performed by: Project tester (Vanessa Espinoza).

6.2. Stress & Volume Testing (S&V)

- Not applicable for this MVP.
- The system does **not** use real backend servers, databases, or large datasets. Only small mock data is used (sample recipes, sample grocery items).
- No stress or performance load testing performed.

6.3. Connectivity Testing (CT)

Connectivity testing for this MVP was limited and focused only on verifying that the Flutter Web application runs correctly within the local development environment.

Scope:

- Confirm the application launches successfully through VS Code using Flutter Web.
- Validate that the application loads and runs in Google Chrome on a MacBook Pro (2019)**.**
- Confirm that temporary internet loss does not crash the UI (Chrome may freeze or require reload, which is acceptable for this MVP).

Platform Used:

- macOS Sonoma (MacBook Pro 2019)
- Google Chrome (latest version)
- VS Code for running and debugging the Flutter Web build

6.4. Disaster Recovery/Back Up

Disaster Recovery and Backup testing are not applicable to this project.
The SP-5 Purple Grocery List Application is an MVP Flutter Web application that uses:

- No backend servers
- No cloud database
- No persistent remote storage
- Only local storage (SharedPreferences) for saving temporary user preferences

Because all data is mock/sample data and stored only in the browser's local storage, there are no backup systems, no failover servers, and no restoration procedures to test.

.

6.5. Unit Testing

Unit testing will not be performed for this release. Because the application is an MVP built with Flutter Web and uses only front-end UI components and mock data, no individual code modules or units were isolated for automated unit testing.

All verification was completed through manual functional testing of UI screens, navigation flows, and local storage behavior during progression and regression testing.

6.6. Integration Testing

No integration testing will be performed for this release. The application does not include any backend services, APIs, external systems, or database integrations. All functionality is contained within the Flutter front-end and uses mock data stored locally.

Testing focuses solely on UI flow validation, navigation behavior, and local storage, which do not require integration testing.

7. Test Strategy

7.1. Test level responsibility

Detail the testing levels expected to be applied and who has primary (P) and secondary (S) responsibility for performing this testing (example below).

Test Level	External Party	Proj Team	Business
Unit Testing		P	
Integration Testing		P	
Security Testing		S	
Connectivity Testing		P	
User Acceptance Testing		S	P
Production Verification Testing		S	P

7.2. Test Type & Approach

Detail the types of testing covered by the project team and their standard objectives (example below)

Test Type	Objectives
Progression Requirements	The objectives are to verify that the application: • Meets the defined requirements; • Performs and functions accurately; • Correctly handles error conditions; • Interfaces function correctly; • Data load is successful. Functional testing will occur in an iterative and controlled manner, ensuring the solution matches the defined requirements.
Regression testing	

7.3. Build strategy

The Purple Groceries App is implemented as a single Flutter install (one unified codebase) rather than separate modules or services. All core features — login (mock authentication), recipe browsing, grocery list management, settings, and AI chat — are packaged inside one application and deployed together. This strategy keeps deployment simple (one build, one install) while still organizing the codebase by feature so that each part can be developed, tested, and maintained independently inside the same Flutter project.

7.4. Test Execution Schedule

Date	type of testing	functions tested

Sept 29 – Oct 5	Unit Testing	• Login screen (mock authentication) • Navigation routing • Basic widget rendering (cards, list items, buttons) Goal: Ensure core screens load and respond correctly.
Oct 6 – Oct 12	Functional Testing	• Add/Edit/Delete grocery items • Recipe browsing • Category filtering Goal: Validate all primary app functions work as intended.
Oct 13 – Oct 19	Integration Testing	• Local storage read/write (SharedPreferences) • Data persistence between sessions • Navigation flow between Recipe → Grocery List → Settings Goal: Ensure features interact properly and maintain consistent state.
Oct 27 – Nov 2	AI Feature Testing	• Gemini API connection • Sending prompts and receiving responses • Handling invalid or missing API keys Goal: Ensure AI chat features work reliably under typical and edge-case inputs.

7.5. Facility, data, and resource provision plan

7.5.1. Test environment

The project uses a simple local test environment. All testing is done on each team member's computer using the Flutter SDK, Chrome browser, and SharedPreferences for local data storage. No external servers, databases, or backend systems are required. Internet access is only needed when testing the Gemini AI feature.

7.5.2. Access to other applications

No external systems are required for testing the Purple Groceries App. All core features run locally within the Flutter environment using SharedPreferences for storage. The only external access needed is optional:

- Google Chrome: Required for web-based testing (flutter run -d chrome)

- Gemini API (optional): Only needed when testing the AI chat feature, which requires a valid Gemini API key and internet access

No databases, servers, or third-party applications are required for normal testing.

7.5.3. Testing Requirements

Before testing can begin, each person involved in the testing process must have the following:

- A computer with the Flutter SDK installed
- An IDE such as VS Code, Cursor, or Android Studio
- Google Chrome for running and testing the web build
- Local device access for SharedPreferences storage
- Optional: A valid Gemini API key (required only for AI feature testing)
- Ability to document defects using Word, Google Docs, or Excel

7.5.4. Data Requirements

To begin testing, only minimal local data setup is required:

- At least one mock user login (email and password that pass basic format validation)
- Sample grocery list items created by the tester (e.g., “Milk,” “Eggs,” “Chicken”)
- Sample recipe entries added or accessed through the app’s built-in recipe data
- A valid Gemini API key for testing AI chat functionality

7.5.5. Resources & Skills

The following resources and skills are required during the testing window:

- Flutter/Dart competency: To execute tests, identify UI or functional defects, and understand widget behavior or navigation issues.
- General QA/Testing skills: Ability to create test cases, reproduce defects, document issues clearly, and evaluate user flows for correctness and usability.
- Local storage knowledge (SharedPreferences): Understanding how local key-value data is saved and retrieved to verify persistence, clearing, and edge-case handling.
- Basic troubleshooting and debugging skills: Ability to interpret console logs, use IDE debugging tools, and pinpoint where failures occur within the app.

7.6. Testing Tools

Detail the tools to be used for testing.

For example:

The following tools will be used for testing:

Process	Tool
Test case creation	Microsoft Word
Test case tracking	Microsoft Excel
Test case execution	Manual
Test case management	Microsoft Excel
Defect management	Microsoft Excel

7.7. Testing Handover Procedure

Handover from Developers to Testers

- Once a feature is completed (e.g., grocery list, recipes, settings), the two developers push the updated build to the shared repository.
- Developers provide a short note summarizing what was implemented and any known limitations.
- Testers are notified that the feature is ready for review.

Tester Review Phase

- The two testers/commentors review the feature in the Flutter web environment.
- They check functionality, navigation flow, UI layout, and overall user experience.
- Any issues are documented and sent back to the developers through group chat or comments.

Handover Back to Developers for Fixes

- Developers review the notes from testers and make corrections or improvements.
- Fixes are pushed as a new updated build along with a brief summary of what was changed.

Final Acceptance Handover

- Once testers confirm that the fixes address all issues, the feature is marked as "accepted."
- The final build is reviewed by the full team to ensure that all features work together.
- Final documentation is updated before submission.

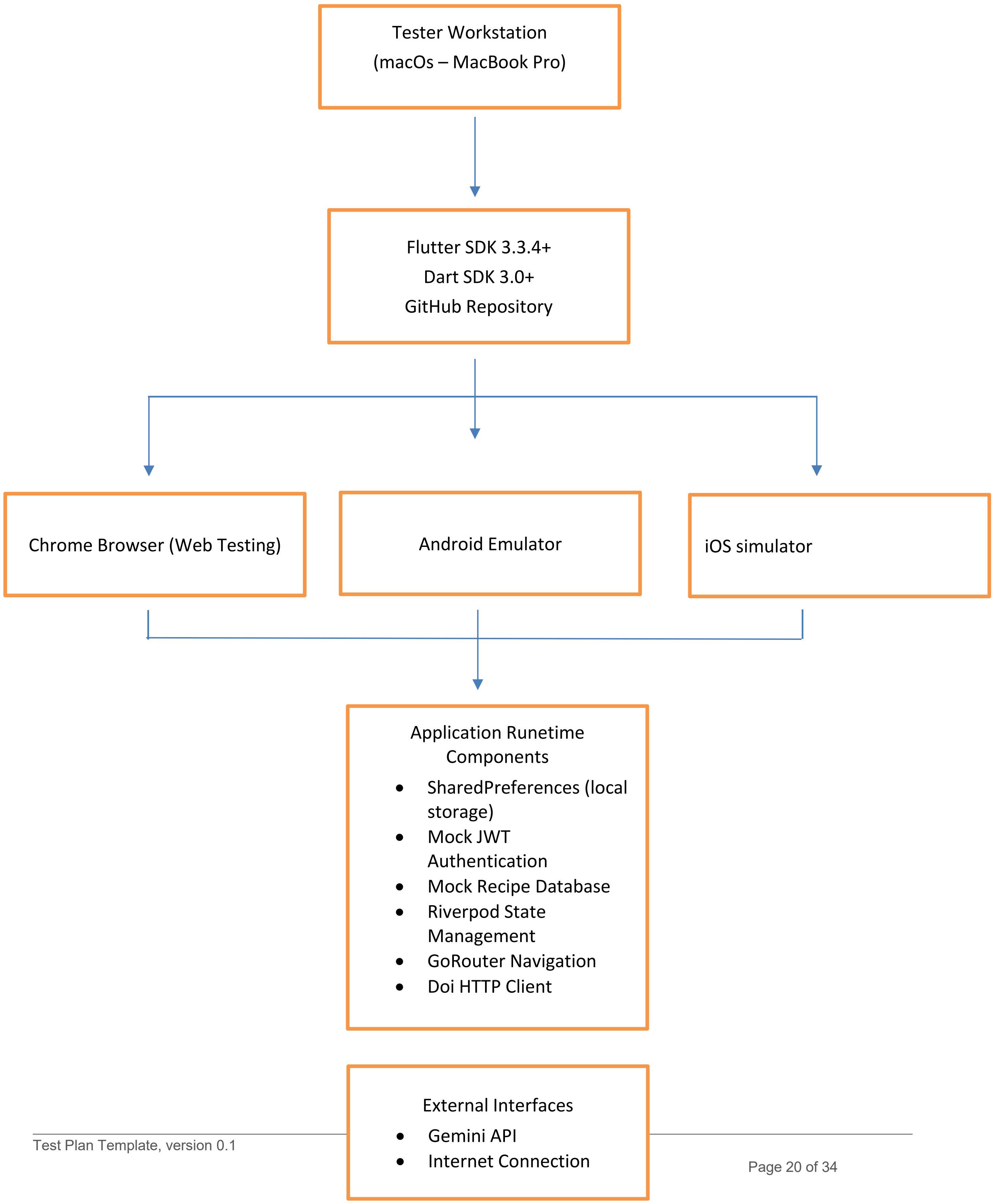
7.8. Testing Metrics

Title	What We Track	Why	How

Number of Issues Found per Feature	How many bugs or problems were found in each feature (Login, Recipes, Grocery List, Settings, AI Chat).	Helps us see which parts of the app need the most attention or redesign.	Recorded in a simple shared Google Doc or Sheet after each testing session.
Issue Severity	Whether an issue is minor, noticeable, or breaks the feature	Helps the developers know what needs to be fixed immediately versus later.	Testers label each issue as Low, Medium, or High.
Fix Time /Turnaround	How fast an issue goes from "reported" to "fixed."	Shows whether any feature is harder to fix or needs cleanup.	Testers add the date an issue was found, and developers add the date it was resolved.
Repeated Issues (Did the same bug come back?)	Issues that reappear after being marked fixed.	Helps identify unstable parts of the code or unclear requirements.	Testers mark any reopened issues in the shared sheet.
User Flow Stability	Whether major flows can be completed without errors.	Ensures the app is usable from start to finish.	Testers check off each flow once per week.

8. Test Environment Plan

8.1. Test Environment Man



8.2. Test Environment Details

8.2.1. Testers

Number of testers: **1**
(Tested individually by Vanwssa Espinoza.)

System access requirements:
• Access to the Flutter project source code
• Ability to run Flutter Web using Chrome
• Ability to run the Android Emulator via VS Code or Android Studio
• Access to GitHub repository for code updates
• System-level permissions to install and run Flutter SDK and Dart SDK

Hardware requirements:
• macOS laptop (MacBook)
• Chrome browser installed
• Stable internet connection (for Gemini API and package downloads)

8.2.2. Hardware and Firmware

Hardware:
• Apple MacBook Pro (16-inch, 2019)
• Processor: 2.6 GHz 6-Core Intel Core i7
• Memory: 16 GB 2667 MHz DDR4
• Graphics: Intel UHD Graphics 630 (1536 MB)
• Operating System: macOS 14 Sonoma
• Chrome browser (latest version)

Purpose of use:
• Running the Flutter application for Web, Android, and iOS
• Debugging UI components and navigation flows
• Simulating device behavior across multiple platforms
• Testing SharedPreferences storage and in-app state management
• Running Gemini API calls for the AI Chat module

Provided by:
• Tester (Vanessa Espinoza – personal MacBook Pro)

8.2.3. Software

Software used in the test environment includes the following components:

Flutter SDK 3.3.4+
Dart SDK 3.0.0+
VS Code (primary IDE used for running and debugging the Flutter project)
Android Studio (used for Android Emulator tools)
Chrome browser (used as the primary execution platform for Flutter Web testing)
GitHub repository containing the project source code

SharedPreferences package for local storage
Riverpod 2.4.9 for state management
GoRouter 12.1.3 for navigation
Dio 5.4.0 for HTTP handling
Gemini API integration for the AI chat module
Mock recipe database and mock JWT authentication components
macOS Sonoma 14.x (host operating system)

These software tools and frameworks support development, execution, debugging, and validation of the Flutter Web, Android, and iOS versions of the application. Software was installed and managed directly on the tester’s machine.

8.2.4. Interfaces

Interfaces to external applications:
The current MVP integrates with *one* external service:

- Gemini API (AI Chat Module)
  Used to generate keyword-based or mock AI responses within the Chat feature.
  Communication occurs through HTTP requests executed by the Flutter application.

Local Interfaces:

- **SharedPreferences (Local Storage)**
  Used for storing user profile preferences, onboarding completion, and cached app data.
  Operates fully on-device and requires no external connection.

Who establishes the interface:

- The **tester (Vanessa Espinoza)** establishes and configures access through:
  - Installing required Flutter packages (Dart SDK, Dio HTTP client)
  - Setting local API key environment variables (if used)
  - Configuring emulator/browser environment

How the interface is established:

- Gemini API requests are performed through the Flutter application using the Dio client.
- SharedPreferences is automatically initialized when the app launches.
- No separate interface agreement, backend configuration, or enterprise integration is required for this MVP.

When the interface is used:

- During functional testing of:
  - AI Chat
  - Onboarding preference extraction
  - Navigation and data persistence
  - Recipe and meal plan flows involving stored preferences

8.2.5. Other Materials

The following materials are required to support testing for this release:

- **User documentation and README**
  Used for setup instructions, feature descriptions, and platform requirements.
- **Gemini API Key (if enabled)**
  Required for testing AI chat responses.
  Provided and configured locally by the tester.
- **GitHub Repository**
  Hosts the Flutter project source code and is used to pull updates during development and testing.
- **Flutter Packages and Dependencies**
  Auto-installed via flutter pub get and used by the tester to run the application.

No additional manuals, licensing files, printed documentation, or enterprise materials are required for this

8.3. Establishing Environment

Define the plan for establishing the testing environment, and responsibilities. This should include acquisition of each element, setup, installation and testing the environment.

Task	Requirements	Responsibility	Start Date	End Date
Install Flutter SDK & Dart SDK	Flutter 3.3.4+, Dart 3.0+; Stable internet connection	Tester (Vanessa Espinoza)	11/30/2025	11/30/2025
Configure Development Tools	VS Code installed, Flutter extensions, Git, Chrome browser	Tester	11/30/2025	11/30/25
Install Application Dependencies	Run flutter pub get; ensure Riverpod, GoRouter,Dio, Sharedpreferences installed	Tester	11/30/2025	11/30/2025
Clone GitHub	Access GitHub project & pull latest code	Tester	11/30/2025	11/30/2025
Run and Validate Build	Verify working builds	Tester	11/30/2025	11/30/2025

8.4. Environment Control

This section defines the control measures placed on the testing environment to ensure consistency, stability, and integrity across all test cycles.

Software Release Control

- Only approved project branches from GitHub (e.g., main or dev) may be used for testing.

- Code updates must be pulled before test execution.
- Version control through GitHub ensures reproducibility of test results.

Environment Access

- Only the tester (Vanessa Espinoza) has access to the local development/testing environment.
- Access to API keys (Gemini API) is restricted and stored individually by the tester.

Environment Monitoring and Support

- System performance monitored manually during testing (build errors, crashes, emulator stability).
- Flutter doctor (flutter doctor -v) used to confirm the environment is healthy before test execution.

8.5. Environment Roles and Responsibilities

Define the roles and responsibilities of persons who will be responsible for, or interface with the environment

Role	Staff Member	Responsibilities
Release Manager	Vanessa Espinoza	Responsible for overall establishment, coordination and support of the test environment
Test Manager	Carinne Tzurdecker	Responsible for advising release manager of environment requirements for planning, establishment and ongoing
Project Manager	Shane Hazeur & Piao Hou	Escalation points for environment issues.

9. Assumptions and Dependencies

9.1. Assumptions

The following assumptions apply to the testing of the PurpleGroceries Meal Planning Application:

- The tester (Vanessa Espinoza) has full access to the project's Flutter source code through GitHub.
- All required software components (Flutter SDK, Dart SDK, VS Code, Chrome, Android Emulator) will remain stable and functional throughout the testing period.
- Internet connectivity will be available for package installation, emulator setup, and optional Gemini API calls.
- Mock authentication, mock recipe database, and local SharedPreferences storage will behave consistently during testing.
- AI-related testing assumes the Gemini API key remains valid (if used).
- The application build will compile successfully for Web and Android without requiring external backend services.
- No additional team members or external support resources are required for defect resolution, as this is a solo-tested project.

9.2. Dependencies

Testing activities depend on the following components being available and functional:

- **GitHub Repository**
  The test environment depends on successful access to the project repository for pulling source code updates.
- **Flutter SDK & Dart SDK**
  Testing requires the correct SDK versions (Flutter 3.3.4+, Dart 3.0+) to be installed and properly configured.
- **Device Emulators / Browsers**
  Web testing depends on Chrome.
  Android testing depends on Android Emulator (API 31+).
  iOS Simulator is optional and only available if configured.
- **Package Dependencies**
  Flutter packages such as Riverpod, GoRouter, Dio, and SharedPreferences must install without errors via flutter pub get.
- **Local Device Storage**
  SharedPreferences must be functional for storing user profiles, meal plans, and preferences.
- **System Resources**
  The MacBook Pro must provide enough RAM and CPU to run emulators and builds reliably.

10. Entry and Exit Criteria

Testing may begin only when all of the following conditions are satisfied:

Entry Criteria

Test Environment Readiness

- Flutter SDK and Dart SDK are successfully installed and configured.
- Required IDE/tools (VS Code, Chrome, Android Emulator) are available and functioning.
- All Flutter dependencies install without errors using flutter pub get.
- The application builds successfully for the target platforms (Web and Android).

Application Readiness

- Latest version of the source code is pulled from GitHub.
- Mock authentication, mock recipe data, and SharedPreferences local storage are functional.
- Navigation, UI screens, and major flows compile without crashes on app launch.
- Optional: Gemini API key is valid (if AI chat module testing is included).

Test Preparation

- Test plan and test scenarios have been documented.
- Acceptance criteria for features (recipes, grocery list, meal plan, profile, chat) are defined.
- Test data (sample recipes, sample user profiles, mock meal plans) is ready.
- All required hardware is available (MacBook Pro + Emulator/Browser).

Testing cannot begin until all entry criteria are met.

Exit Criteria

Testing is considered complete when the following conditions are met:

Test Coverage

- All planned test cases for core features have been executed:
  - Authentication & Onboarding
  - Navigation (Bottom Nav, Routing)
  - Recipes (Search, Filters, Details)
  - Meal Planning (3–7 day plan creation)
  - Grocery List (Add, Remove, Check-off, Consolidation)
  - Profile Setup & Editing
  - Settings Page
  - AI Chat Interface (if Gemini enabled)

Defect Status

- All critical and major defects have been resolved or retested with acceptable workarounds.

- No showstopper defects remain (app does not crash on launch or navigation).
- Minor defects have been logged and documented for future development.

Test Documentation Completion

- All test results, observations, and evidence are recorded.
- All defects are documented in the issue tracker (or report).
- Test summary report is completed.

Stability Criteria

- Application builds consistently without errors for Web and Android.
- App demonstrates stable performance during navigation, UI display, and user flows.
- Local data (SharedPreferences) behaves consistently across sessions.

Exit Approval

- Tester (Vanessa Espinoza) confirms that all exit criteria have been met.

Testing is officially complete only when all exit criteria are satisfied.

11. Administrative Plan

11.1. Approvals

Detail the responsibilities for testing signoff. For example, the following persons are responsible for the critical aspects of testing:

Task	Responsible Person	Escalation/ Approver
Systems Integration Signoff	Piao Hao & Shane Hazeur	Piao Hao, Shane Hazeur, Vanessa Espinoza, and Carinne Tzurdecker
User Acceptance Testing Signoff	Vanessa Espinoza & Carinne Tzurdecker	Piao Hao, Shane Hazeur, Vanessa Espinoza, and Carinne Tzurdecker
Production Verification Testing Signoff	Piao Hao & Shane Hazeur	Piao Hao, Shane Hazeur, Vanessa Espinoza, and Carinne Tzurdecker

11.2. Test Milestones and Schedule

Detailed below are the high-level testing milestones.

Milestone	Planned End Date	Actual End Date	Resource
Requirements Gathering	9/16	9/26	Team
Requirement Review & Approval	9/23	9/23	Team
Project Design Completed (UI & Database & Flows)	9/30	10/2	Developers and Designer
Working Prototype Built	10/14	10/16	Developers
Prototype Testing Completed	10/21	10/22	Tester
Full Feature Development Completed	11/11	11/11	Developers
Integration & Bug Fixing	11/18	11/20	Developers
Functional Testing Completed	11/25	11/25	Tester
Regression Testing Completed	11/30	11/30	Tester
Final Report and Presentation Submitted	12/08	12/08	Entire Team

11.3. Training

The following training requirements have been identified to ensure testing can commence:

Training Requirement	Staff	Date
Project Architecture (Flutter etc.)	Shane Hazeur & Piao Hao	9/23
Training on testing environment setup	Shane Hazeur & Piao Hao	9/30
Review of test plan, test cases, and testing workflow	Vanessa Espinoza & Carinne Tzurdecker	11/25

11.4. Defect Management

Defects for this project will be managed using a lightweight, manual defect tracking approach suitable for a student MVP project.

Defect Management Process:

- All defects found during testing will be recorded in a shared Google Doc or Excel sheet.
- Each defect entry will include:
  - Defect ID
  - Description
  - Steps to reproduce
  - Expected vs. actual result
  - Severity
  - Status (Open / In Progress / Resolved)
  - Date reported
  - Person responsible for fix
- Defects will be reviewed after each test cycle.
- Fixes will be re-tested before marking the defect as resolved.

Tool Used:

- Google Sheets (shared with all team members)

12. Definitions

The following acronyms and terms have been used through out this document

Term/Acronym	Definition
CRUD	Create, Read, Update, Delete
JWT	JSON Web Token (used for mock authentication)
SDK	Software Development Kit (Flutter/Dart)
MVP	Minimum Viable Product (Current state of the app)
API	Application Programming Interface (Gemini API used for AI chat module)
UI/UX	User Interface/User Experience User interface being the visual layout of the app and User Experience being the overall usability and flow

13. References

The following documents have been used to assist in creation of this document.

#	Document name	Version	Comments
1	Grocery List App GitHub Repository README.md	Latest	Used for feature descriptions, supported platforms, and implementation notes
2	STP – Template	0.1	Base structure used for creating this Software Test Plan

14. Points of Contact

The following people can be contacted in reference to this document

Primary Contact	
Name	Vanessa Espinoza
Title/Organisation	Documentation
Phone	404-796-6075
Email	vespino1@students.kennesaw.edu
Secondary Contact	
Name	Carinne Tzurdecker
Title/Organisation	Documentation
Phone	470-365-9019
Email	ctzurdec@students.kennesaw.edu

Software Test Report (STR)

The Software Test Report summarizes the results of all executed test cases for the Purple Groceries App. Testing focused on verifying the stability and functionality of all implemented features, including mock authentication, grocery list management, recipe browsing, settings updates, SharedPreferences storage, and the AI chat module. All testers used the shared Flutter web testing environment as defined in the STP.

Severity ratings (Low, Minor, Moderate, Critical) reflect how much a defect would impact user experience or feature usability. Most core features passed without major issues, showing stable behavior across screens and interactions. Features not included in project scope (Create Account, Password Recovery) were recorded as expected failures.

Requirement	Pass	Fail	Severity	Notes
Create account		Low	Not implemented; app uses mock login only	
Login with mock credentials		Minor	Accepts valid format; generates placeholder token.	
Password Recovery		Low	Not included in scope.	
Add grocery items		Moderate	Not included in scope.	
Edit grocery item		Minor	Updates render correctly.	
Delete grocery item		Minor	Stable and consistent.	
Persistent storage (SharedPreferences)		Moderate	Data persists properly between sessions	
View recipe list		Minor	Loads sample recipes correctly.	
Filter recipes by category		Minor	Accurate filtering results.	

Modify user settings (Weight, height, preferences)		Minor	Settings save and load correctly through SharedPreferences.	
AI Chat (Gemini)		Moderate	Works with valid API key; errors handled gracefully.	
Navigation across all screens		Minor	No route issues or crashes encountered.	

Summary of Results

Testing confirms that the Purple Groceries App functions reliably across all implemented features. Grocery list creation and editing, recipe browsing, navigation, settings updates, and AI chat all passed successfully.

The only failed test cases correspond to features intentionally omitted from the project scope (account creation and password recovery). No critical issues were identified, and no defects prevented testers from completing full user flows.

Overall, the application demonstrates strong stability for its intended scope and is ready for final submission.