SP-5 — Purple Groceries App

SOFTWARE DESIGN DOCUMENT (SDD)

Course: CS 4850 Senior Project

Section 03

Semester: Fall 2025

Date: September 12

Professor: Sharon Perry

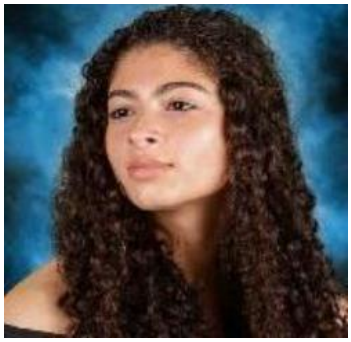 Carinne Tzurdecker Documentation	 Vanessa Espinoza Documentation	 Shane Hazeur Developer	 Piao Hou Developer

Table of Contents

1.0 Introduction

1.1 Document Outline

This Software Design Document describes the architecture and detailed design of the Purple Groceries App. It translates the requirements defined in the SRS into a technical design suitable for implementation. The document covers system architecture, components, data processing, and user interface design.

1.2 Document Description

1.2.1 he grocery app is a mobile application designed to allow users to generate personalized meal plans and grocery lists from both structured profile data and free-text natural language input. The application simulates intelligent behavior through keyword matching, contextual analysis, and deduplication of grocery items, while also enforcing user constraints such as allergens, dietary preferences, and budget.

1.2.2 The system is implemented in React with Hooks for state management and Tailwind CSS for styling. All processing occurs on the client, with no backend or persistent database required. The application features an event input interface, interactive calendar, and grocery list manager, which together provide users with a responsive and accessible grocery planning experience.

2.0 Design Considerations

2.1 Assumptions and Dependencies

The grocery app assumes that users will access the system on Android devices running Android 10 or later. The system depends on a stable client–server connection, with the Android client communicating over HTTPS to a Spring Boot backend. The backend relies on a PostgreSQL database with pgvector support for recipe storage and semantic search. Authentication and authorization assume that JWT tokens will be securely issued and managed by the server. The app assumes users will input goals, constraints, and free-text briefs in English, and current keyword matching is limited to predefined dictionaries. Any future requirement for multi-language input, offline persistence, or integration with third-party APIs would require modifications to both the Android client and backend services.

2.2 General Constraints

The grocery app is constrained to run on Android devices (Android 10+ minimum, minSdk 26) and depends on a Spring Boot backend for authentication, meal plan generation, and grocery list consolidation. Persistent data storage and recipe search require the PostgreSQL database with

pgvector support, meaning the app cannot function without a server connection. Performance is constrained by the requirement that plan generation must complete in under 6 seconds (SRS §4.2) and recipe search must return first-page results within 800 ms p95. Security is constrained by reliance on Spring Security with JWT, enforcing HTTPS/TLS 1.2+ for all communications. Memory and processing limits on client devices restrict very large meal plans or grocery lists from being handled locally, so computation is distributed between the Android client and backend services. Any future requirements for iOS support, offline functionality, or integration with external APIs would necessitate additional redesign beyond the current scope.

2.3 Development Methods

The grocery app is developed using an iterative, agile-inspired approach that emphasizes incremental prototyping and continuous refinement. The Android client is implemented in Java for Android 10+ using modular components that map directly to the functional requirements in the SRS. The backend is implemented in Java 21 with Spring Boot 3.x, providing RESTful APIs secured with Spring Security and JWT. PostgreSQL serves as the data repository, extended with pgvector for semantic recipe search. Gradle is used for builds and dependency management, while GitHub supports version control and team collaboration. Continuous integration ensures that new features are automatically tested, with a focus on validation of allergen enforcement, authentication, and grocery list generation

Alternative methods, such as building a lightweight client-only app or using machine learning–based NLP models for free-text processing, were considered but rejected. A browser-only app lacked persistent storage and multi-device synchronization, while ML-based NLP introduced unnecessary complexity, cost, and latency. The chosen methods align with project goals of reliability, security, and predictable performance under the environment constraints defined in the SRS.

3.0 Architectural Strategies

The grocery app is designed as a client–server system with an Android client and a Spring Boot backend. This architecture was chosen to meet the functional requirements for persistent storage, authentication, and secure communication described in the SRS. The client focuses on user interaction and presentation, while the backend manages business logic, recipe storage, and grocery list generation.

The Android client is implemented in Java using a modular activity and fragment structure, ensuring separation of concerns across user profile management, meal plan generation, and grocery list features. This decision supports maintainability and aligns with the requirement for Android 10+ support.

The backend is implemented in Java 21 with Spring Boot 3.x, providing RESTful APIs that are secured with Spring Security and JWT. This approach ensures compliance with the security requirements defined in SRS §4.1 (all traffic over HTTPS, password hashing, token-based authentication). Data persistence is handled by PostgreSQL, extended with pgvector to enable semantic recipe search. Alternatives such as NoSQL databases or external recipe APIs were considered but rejected due to added complexity and reduced control over query performance.

Meal plan generation and grocery list processing are implemented with deterministic rule-based algorithms, keyword matching, and context analysis rather than machine learning. This approach guarantees predictable results, enforces allergen and exclusion constraints (SRS §3.8 Safety & Compliance), and satisfies the performance requirement of generating plans within 6 seconds (SRS §4.2 Capacity).

The system employs a structured UI paradigm on the client, with distinct screens for onboarding, free-chat brief input, meal plan preview, and grocery list management (SRS §5.1 User Interface Requirements). This layout was selected to reduce cognitive load and improve usability on mobile devices.

All persistent data is stored server-side in the PostgreSQL database, while the client maintains temporary state for active sessions. This ensures security and consistency across devices, but also introduces a dependency on network connectivity. Future enhancements may include expanded offline capabilities, iOS support, or the use of more advanced AI models for free-text analysis.

4.0 System Architecture

The grocery app follows a modular client–server design, with responsibilities divided between the Android client and the Spring Boot backend. This separation ensures that input capture, user interaction, and display logic remain on the client, while authentication, meal planning, and data persistence are handled by the server. The design corresponds directly to the functional and non-functional requirements outlined in the SRS, ensuring that each subsystem has clearly defined responsibilities while maintaining smooth communication over secure RESTful APIs.

At the highest level, the system is organized into five core subsystems:

- **Event Input Subsystem** — Implements SRS §3.3 (Free-Chat Brief) by capturing natural-language text or quick examples provided by the user. Input is validated and routed to the AI Processor for interpretation.
- **Calendar Subsystem** — Supports SRS §3.7 (Planner Controls & Modes) by providing date selection, duration control, and event scheduling. It communicates with the Event Input and Grocery List subsystems to contextualize item usage.
- **Events List Subsystem** — Manages active events, satisfying SRS requirements related to user profile constraints and planning (e.g., §3.2 User Profile). It displays events as interactive cards with options for deletion or editing.
- **Grocery List Subsystem** — Implements SRS §3.6 (Grocery List Builder) by consolidating ingredients, normalizing units, and grouping categories. The backend generates the list, and the client provides editing, marking, and exporting functions.
- **AI Processor Subsystem** — Enforces SRS §3.5 (Meal Plan Generation) and §3.8 (Safety & Compliance) by applying keyword matching, context analysis, and allergen filtering. This subsystem integrates with PostgreSQL (with pgvector) to perform semantic recipe retrieval and ensures all constraints are validated before results are returned to the client.

This architecture ensures that performance-critical tasks (such as grocery list consolidation and recipe search) are handled by the backend, while responsiveness and usability are maintained on the Android client. Persistent data storage and security are centralized on the server, meeting the constraints defined in the SRS environment.

5.0 Detailed System Design

5.1 Classification

The Event Input Subsystem is implemented as an Android activity and supporting fragments. It is a UI-driven subsystem on the client that captures and validates user input before sending it to the backend.

5.2 Definition

The subsystem provides a text field and quick-select examples where users can enter natural-language descriptions of events, meals, or grocery needs. Input is forwarded to the AI Processor Subsystem for interpretation and mapping to grocery items, supporting SRS §3.3 (Free-Chat Brief).

5.3 Constraints

The subsystem assumes English-language input only. It is dependent on network connectivity to communicate with backend services. Input must be submitted and acknowledged in under 10 seconds, in accordance with SRS §4.2 (Capacity).

5.4 Resources

The subsystem uses Android device resources for UI rendering and temporary in-memory storage. It communicates with the backend over HTTPS using RESTful API calls secured with JWT.

5.5 Interface/Exports

The subsystem exports structured user input (free-text brief, timestamp, and related metadata) to the backend API. Data is transmitted as JSON payloads and consumed by the Meal Plan Generation and Grocery List Builder subsystems.

6.0 Glossary

Android Client — The mobile application built for Android 10+ using Java. Provides the user interface for input, meal planning, and grocery list management.
Activity / Fragment (Android) — Android UI components used to structure screens such as onboarding, free-text input, meal plan preview, and grocery list editing.
API (Application Programming Interface) — RESTful endpoints exposed by the Spring Boot backend that allow the Android client to send requests and receive responses
Calendar Subsystem — The component responsible for managing event scheduling, date selection, and range selection.
Client-Side State — Data stored and managed entirely in the user's browser using React hooks, with no external persistence.

Context Analysis — A rule-based process that interprets user input to identify meal types, occasions, and dietary constraints.
Deduplication — The process of detecting and removing duplicate grocery items when consolidating a shopping list.
Event Input Subsystem — The user interface for entering natural-language text or selecting predefined event examples.
Events List Subsystem — The display of created events as interactive cards that can be modified or removed by the user.
Events Management Subsystem — The combination of client and backend logic for storing, editing, and displaying meal plans.
Grocery List Subsystem — The component that consolidates, categorizes, and displays grocery items, with support for editing and completion tracking.
JWT (JSON Web Token) — A secure token format used for authenticating users and authorizing API calls.
Keyword Matching — The technique of mapping specific user input terms to predefined grocery categories or items.
REST (Representational State Transfer) — The architectural style used for communication between the Android client and Spring Boot backend over HTTPS.
Simulated AI Processor — The subsystem that interprets user input through keyword matching and context analysis, without machine learning.
Spring Boot — A Java-based backend framework used to implement the server-side logic and expose REST APIs.
Spring Security — The security framework that enforces authentication and authorization policies using JWT.
PostgreSQL — The relational database used to store user profiles, recipes, and meal plan data.
pgvector — A PostgreSQL extension that supports semantic recipe search using vector embeddings.

7.0 Bibliography

[1] Nielsen Norman Group. *Response Times: The 3 Important Limits*.