Software Design Specifications

for

EcoMunch - Food Waste Reducer

Prepared by:

Group Name: EcoMunch

Prachi Sarda	**SE22UARI130**	**se22uari130@mahindrauniversity.edu.in**
G Aniketh	**SE22UARI211**	**se22uari211@mahindrauniversity.edu.in**
P Mohith Krishna	**SE22UARI138**	**se22uari138@mahindrauniversity.edu.in**
Sneha Sharma	**SE22UARI162**	**se22uari162@mahindrauniversity.edu.in**
Shreya Patil	**SE22UARI207**	**se22uari207@mahindrauniversity.edu.in**
Shilpa	**SE22UARI152**	**se22uari152@mahindrauniversity.edu.in**

Document Information

Title: Food Waste Reducer	
Project Manager:	**Document Version No:2**
	Document Version Date:10-03-2025
Prepared By:team EcoMunch	**Preparation Date:10-03-2025**

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
SRS 2	Prachi Sarda, G Aniketh, P Mohith Krishna, Sneha Sharma, Shreya Patil, Shilpa	The SRS2 documents includes updates to the use case model and modifications to the appendices. The remodeled use case diagram enhances clarity, while the appendix revisions improve documentation accuracy.	**10 March, 2025**
SRS 1	Prachi Sarda, G Aniketh, P Mohith Krishna, Sneha Sharma, Shreya Patil, Shilpa	The SRS1 document outlines the functional and non-functional requirements of a software system. It typically includes an introduction with project objectives, scope and definitions along with system features, user requirements and interface specifications. Functional requirements define system behavior, while non-functional aspects like performance, security, and usability constraints are also specified.	**04 March, 2025**

Table of Contents

1. INTRODUCTION

1.1 PURPOSE 5

1.2 SCOPE 5

1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS 5

1.4 REFERENCES 5

2 . USE CASE VIEW 5

2.1 USE CASE 5

3. DESIGN OVERVIEW 6

3.1 DESIGN GOALS AND CONSTRAINTS 7

3.2 DESIGN ASSUMPTIONS 7

3.3 SIGNIFICANT DESIGN PACKAGES 7

3.4 DEPENDENT EXTERNAL INTERFACES 7

3.5 IMPLEMENTED APPLICATION EXTERNAL INTERFACES 7

4 . LOGICAL VIEW 7

4.1 DESIGN MODEL 8

4.2 USE CASE REALIZATION 8

5 . DATA VIEW 11

5.1 DOMAIN MODEL 11

5.2 D ATA MODEL (PERSISTENT DATA VIEW) 11

5.2.1 Data Dictionary 11

6 . EXCEPTION HANDLING 12

7 . CONFIGURABLE PARAMETERS 12

8 . QUALITY OF SERVICE 13

8.1 AVAILABILITY 13

8.2 SECURITY AND AUTHORIZATION 13

8.3 LOAD AND PERFORMANCE IMPLICATIONS 13

8.4 MONITORING AND CONTROL 13

1 Introduction

1.1 Purpose

This Software Design Document (SDD) provides a comprehensive blueprint for the design and development of the "Food Waste Reducer" system. It translates the system requirements into detailed technical specifications and design architecture, ensuring a clear pathway for developers to implement and maintain the system effectively.

1.2 Scope

This Software Design Specification (SDS) applies to the development of the Food Waste Processor App, a mobile and web-based application intended to minimize food waste through tracking, categorization, and redistribution of surplus food. This document outlines the architecture, components, data flow, and design decisions for the system.

The design specifications influence and guide all aspects of the app's development lifecycle, including user interface design, backend logic, database structure, API integration, and deployment strategies. Stakeholders affected by this document include developers, designers, project managers, testers, and end-users such as households, food businesses, NGOs, and food banks.

1.3 Definitions, Acronyms, and Abbreviations

1. **MVC**: Model-View-Controller
2. **API**: Application Programming Interface
3. **UI**: User Interface
4. **DBMS**: Database Management System
5. **JWT**: JSON Web Token
6. **ERD**: Entity Relationship Diagram

1.4 References

IEEE 1016-2009 Standard for Software Design Descriptions

2 Use Case View

This section outlines the central use cases of the Food Waste Reducer system based on the functional requirements identified in the SRS. These use cases represent core functionalities critical to the system's purpose of reducing food waste. Each use case exercises multiple design elements and supports interactions between users and backend systems.

2.1 Use Case

Use Case ID	Use Case Name	Priority	Description
U1	Manage Food Inventory	**High**	Users can add, edit, or remove food items, including barcode scanning
U2	Receive Expiry Alerts	**High**	Users receive timely alerts for soon-to-expire items
U3	Plan Meals	**Medium**	Users can create meal plans prioritizing expiring ingredients.
U4	Share/Donate Food	**Medium**	Users can donate surplus food via local donation networks
U5	View Analytics	**High**	Users can view waste trends, savings, and consumption analytics

3 Design Overview

This section outlines the overall structure and organization of the Food Waste Reducer system, describing the architectural foundation, design goals, modular decomposition, and dependencies. The design complies with

the functional and non-functional requirements as outlined in the SRS and supports scalable, secure, and efficient implementation across web and mobile platforms.

3.1 Design Goals and Constraints

Design Goals

1. Minimize Food Waste: Ensure the system prioritizes reduction of food waste through real-time alerts, planning tools, and community sharing.
2. Cross-Platform Accessibility: Deliver a seamless user experience on both web and mobile platforms (iOS, Android).
3. User-Centric Design: Keep the interface intuitive, visually clean, and accessible for all age groups.
4. Performance and Scalability: Support real-time notifications, responsive UI/UX, and scalable backend services.

Design Constraints

1. Multi-Platform Compatibility: Must be deployable on iOS, Android, and browsers using a shared codebase (e.g., React Native + Node.js backend).
2. Real-Time Alerts: Use Firebase Cloud Messaging (FCM) for delivering instant expiry notifications.
3. Data Integrity & Backup: Relational database (e.g., PostgreSQL or Firebase Realtime DB) to maintain structured and synced inventory data.
4. Development Tools: React Native, Node.js, Express.js, Firebase (Messaging + Authentication), PostgreSQL, GitHub CI/CD.
5. Team Structure: 6-person dev team with roles divided across frontend, backend, UI/UX, and testing.
6. Schedule Constraints: Agile development over 6 sprints within an academic semester (as per university schedule).

3.2 Design Assumptions

1. Users have access to smartphones or computers with internet connectivity.
2. Community food-sharing programs or NGOs are present and accessible via APIs or custom listings.
3. Barcode databases are accessible either through third-party APIs or internal mapping.
4. Users are willing to input or scan food items into the app.
5. Notifications permissions are enabled by users for the app.
6. Backend services remain operational for syncing data across devices.

3.3 Significant Design Packages

The Food Waste Reducer is modularly decomposed into the following major design packages:

Package Name	Responsibility
User Interface (UI)	Handles all user interactions via mobile/web screens. Built using React Native for consistency.
Inventory Management	Manages CRUD operations for food items, categorization, and expiration tracking.
Notification Engine	Sends expiry alerts and reminders using Firebase Cloud Messaging (FCM).
Meal Planning Module	Suggests meal plans based on inventory; includes user-defined meal entry and recipe matching.
Food Sharing Module	Connects to nearby donation programs and allows users to list or schedule food donations.
Analytics Dashboard	Generates visual insights from historical inventory and waste data.
Authentication & User Profiles	Secure login, registration, and personalized settings using Firebase Auth.
External API Integrator	Facilitates interaction with barcode databases and donation/composting APIs.

These modules are loosely coupled and follow a layered architecture: **Presentation Layer → Business Logic → Data Access Layer → External Interfaces**.

3.4 Dependent External Interfaces

External Application and Interface Name	Module Using the Interface	Functionality/ Description
Firebase Cloud Messaging	Notification Engine	To push real-time alerts to user devices.
Firebase Authentication	Authentication Module	To securely manage user logins and sign-ups.
Barcode Database API	Inventory Management Module	To auto-fill food item details from scanned barcodes
Location Services API	Food Sharing Module	To find nearby donation centers
NGO/Food Banks API	Food Sharing Module	To find nearby donation centres

3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

Interface Name	Implementing Module	Functionality / Description
/api/inventory	Inventory Management Module	Provides CRUD operations for food items (add, edit, delete, view). Supports barcode input.
/api/alerts	Notification Engine	Handles push notification triggers and status queries for expiring items. Uses Firebase backend.
/api/meal-planner	Meal Planning Module	Returns suggested meals based on user inventory, expiry dates, and dietary preferences.
/api/donate	Food Sharing Module	Allows users to post donations and find local food-sharing centers based on geolocation.
/api/analytics	Analytics Dashboard	Returns user-specific data visualizations: waste trends, category breakdowns, cost savings.
/api/auth	Authentication & User Module	Manages registration, login, and profile management. Integrates Firebase Authentication.
/api/external/barcode	External API Integrator	Queries third-party barcode databases and returns standardized food item data.
/api/location-services	External API Integrator	Fetches nearby food-sharing facilities using Google Maps API or similar.
/api/user/settings	User Preferences Module	Stores and retrieves custom notification settings, dietary filters, and meal planning configs.

4 Logical View

The Food Waste Reducer App follows a layered design, with the UI interacting with core functional modules like Inventory Manager, Donation Manager, and Analytics Engine, which in turn connect to the database layer. Each use case is implemented through coordinated interactions between these layers.

Modules are further decomposed into key classes with clear responsibilities. For example, the Inventory Manager includes classes like `FoodItem`, `BarcodeScanner`, and `InventoryHandler`, which work together to handle inventory tasks.

This structure promotes modularity, making the system easier to develop, test, and maintain. Where needed, class behavior can be further detailed using pseudocode or text

4.1 Design Model

The system is designed using the Model-View Controller (MVC) pattern, split across the following major modules:

Module Overview:

1. **User Interface Module (UI)**
   - Platform-specific UIs for web and mobile
   - Displays inventory, alerts, meal plans, donation interface, analytics dashboard
2. **Food Inventory Module**
   - Manages CRUD operations on food items
   - Integrates barcode scanner API
3. **Notification Module**
   - Handles scheduling and sending expiry alerts
4. **Meal Planner Module**
   - Suggests recipes and meal plans based on expiring items
5. **Donation Module**
   - Connects to food donation networks (via REST APIs)
6. **Analytics Module**
   - Visualizes food waste data, usage trends, and user savings
7. **Authentication & User Management Module**
   - Manages registration, login, roles (user, admin, NGO, etc.)
8. **Database Module**
   - Stores persistent data (inventory, users, meal plans, donations, analytics)

4.2 Use Case Realization

Goal: Users can add, edit, or remove food items, including using barcode scanning.

High-Level Module Interaction:

User → UI Module → Inventory Manager → Barcode Scanner (if used) → Database Module

Activity Flow:

- User opens inventory screen.
- Selects 'Add Item' and either inputs manually or scans barcode.
- UI passes data to Inventory Manager.
- Barcode Scanner fetches product info if barcode is scanned.
- Inventory Manager creates/updates/removes Food Item.
- Database stores the updated inventory list.

Class-Level Collaboration:

User → InventoryUI → InventoryManager → BarcodeScanner (if barcode is scanned) → FoodItem (create/edit/delete) → DatabaseHandler

U2: Receive Expiry Alerts

Goal: The system notifies users about items that are near expiry.

High-Level Module Interaction:

System Scheduler → Inventory Manager → Notification Manager → UI Module → User

Activity Flow:

- A scheduled background task runs daily.
- It checks for each user's expiring items via Inventory Manager.
- Notification Manager creates alerts for items expiring soon.
- Notifications are shown via UI or sent via push/email.

Class-Level Collaboration:

System → InventoryManager → FoodItem.daysLeft() → NotificationManager → Notification (create, schedule) → NotificationSender

---

U3: Plan Meals

Goal: Users plan meals that use ingredients nearing expiration.

High-Level Module Interaction:

User → UI Module → Meal Planner Module → Inventory Manager → Database

---

Activity Flow:

- - User accesses the Meal Planner screen.
- - MealPlanner fetches expiring ingredients from InventoryManager.
- - It matches recipes that use those ingredients.
- - User selects and finalizes a meal plan.
- - MealPlan is stored in the Database.

Class-Level Collaboration:

User → MealPlannerUI → MealPlanner → InventoryManager → FoodItem → RecipeEngine → Recipe → MealPlan → DatabaseHandler

---

U4: Share/Donate Food

Goal: Users donate food to NGOs through local networks.

High-Level Module Interaction:

User → UI Module → Donation Manager → External Donation API → Database

---

Activity Flow:

- User selects items for donation.
- Donation Manager fetches nearby NGOs using location & API.
- User selects a recipient and confirms the donation.
- Donation is logged and status tracked.
- Confirmation sent via UI.

Class-Level Collaboration:

User → DonationUI → DonationManager → FoodItem → APIConnector → NGO → Donation → DatabaseHandler

---

U5: View Analytics

Goal: Users view insights like food waste trends, consumption, and savings.

High-Level Module Interaction:

User → UI Module → Analytics Engine → Database Module → Charts/Graphs UI

Activity Flow:

- User opens the Analytics screen.
- Analytics Engine pulls data on food usage and waste.
- Calculates savings, waste patterns, and usage frequency.
- Results are visualized via graphs and charts.

Class-Level Collaboration:

User → AnalyticsUI → AnalyticsEngine → AnalyticsReport → DatabaseHandler

5. Data View

This section describes the persistent data storage structure of the Food Waste Reducer application. It includes the domain model, the data model, and the data dictionary that represent how data is stored and accessed across the system.

5.1 Domain Model

The domain model consists of key entities and the relationships among them to support core functionalities such as inventory management, meal planning, notifications, food sharing, and analytics.

Primary Entities:

- **User**: Represents a registered user of the system.
- **FoodItem**: Represents an individual food item stored by the user.
- **MealPlan**: Represents planned meals using items from the inventory.
- **Notification**: Represents alerts and reminders for expiration or meal planning.
- **Donation**: Represents food items listed for sharing/donation.
- **AnalyticsReport**: Captures waste patterns and user insights.

Relationships:

- A **User** can have many **FoodItems**, **MealPlans**, **Notifications**, and **Donations**.
- A **MealPlan** references multiple **FoodItems**.
- An **AnalyticsReport** is associated with a **User** and aggregates data from **FoodItems** and **Donations**.

5.2 Data Model (persistent data view)

Table Name	Column Name	Data Type	Key Type	Relationships / Notes
Users	UserID	UUID	Primary Key	One-to-many with FoodItems, MealPlans,

				Notifications, Donations
FoodItems	ItemID	UUID	Primary Key	Belongs to User (UserID as Foreign Key)
MealPlans	PlanID	UUID	Primary Key	Belongs to User (UserID as Foreign Key)
MealPlanItems	PlanID	UUID	Foreign Key	Junction table: connects MealPlans & FoodItems
MealPlanItems	ItemID	UUID	Foreign Key	Each entry links a food item to a plan
Notifications	NotificationID	UUID	Primary Key	Belongs to User (UserID as Foreign Key), linked to FoodItems
Donations	DonationID	UUID	Primary Key	Belongs to User (UserID as Foreign Key), related to FoodItems
AnalyticsReports	ReportID	UUID	Primary Key	Belongs to User (UserID as Foreign Key)
FoodItems	ExpiryDate	DATE	-	Used for alert generation

5.2.1 Data Dictionary

Field Name	Table Name	Data Type	Required	Description
UserID	Users	UUID	Yes	Unique identifier for each user
Name	Users	VARCHAR(100)	Yes	Full name of the user
Email	Users	VARCHAR(100)	Yes	Email address (must be unique)
ItemID	FoodItems	UUID	Yes	Unique ID for each food item
ExpiryDate	FoodItems	DATE	Yes	The date on which the food item is expected to expire
Category	FoodItems	VARCHAR(50)	Yes	Category of the food (e.g., dairy, vegetable, meat)
PlanID	MealPlans	UUID	Yes	Unique ID for the meal plan
ScheduledDate	Donations	DATE	No	Date on which food donation is scheduled
AlertDate	Notifications	DATE	Yes	Date on which alert should be sent to user

6 Exception Handling

[This section should describe exceptions that are defined within the application, the circumstances in which they can be thrown and handled, how the exceptions are logged, and the expected follow-up action needed.]

7 Configurable Parameters

This table describes the simple configurable parameters (name / value pairs).

Configuration Parameter Name	Definition and Usage	Dynamic?
EXPIRY_ALERT_DAYS_BEFORE	Number of days before expiration to notify users (e.g., 3 days before item expiry)	Yes
SESSION_TIMEOUT	Time duration after which the user is auto-logged out due to inactivity (e.g., 15 mins)	Yes
MAX_DONATION_ITEMS	Limit on how many items a user can list for donation at once	No
API_RETRY_ATTEMPTS	Number of retry attempts if a third-party API (like food-sharing) fails	Yes
DATA_SYNC_INTERVAL_MINUTES	Interval for syncing data between client and server (e.g., every 10 minutes)	Yes
MAX_IMAGE_SIZE_MB	Maximum size of an uploaded image (e.g., food item image)	No
DEFAULT_LANGUAGE	Default language set for the user interface	Yes
NOTIFICATION_BATCH_INTERVAL	Time gap between bundled notifications to reduce alert fatigue	Yes

8 Quality of Service

8.1 Availability

- The system is designed for high availability with 99.9% uptime, achieved by deploying the application on scalable cloud services such as AWS or Azure.
- Load balancing and redundant server architecture will ensure uninterrupted service during high usage periods or hardware failures.

8.2 Security and Authorization

- All user passwords are encrypted using secure hashing algorithms (bcrypt or SHA-256).
- All communications between client and server are secured using HTTPS protocols.
- Authentication is managed using JWT (JSON Web Tokens), which ensures session validation without storing user sessions on the server.
- Access control is role-based to restrict permissions appropriately for admins and users.

8.3 Load and Performance Implications

- The system is optimized to support up to 10,000 concurrent users, utilizing indexing, caching, and connection pooling to reduce query response times.
- Backend APIs are stateless and lightweight, enabling horizontal scaling.
- Database operations use parameterized queries to prevent SQL injection and improve performance.

8.4 Monitoring and Control

- All backend API requests and errors are logged with a timestamp and request ID for traceability.
- Application health and performance are monitored using tools like PM2 (Node.js Process Manager) and logging systems like Winston or Morgan.
- Future integrations will include dashboards (e.g., Grafana or Kibana) to track real-time metrics, user behavior, and server load.