**SOFTWARE ENGINEERING LAB MINI-PROJECT**

**TOPIC: DIET AND CALORIE TRACKER**

**Problem Statement:**

* Maintaining a healthy diet is getting increasingly challenging in most people’s fast-paced lifestyle, especially for students and professionals with irregular schedules. People often consume food without understanding its nutritional value, leading to long-term health risks such as obesity, malnutrition, and fatigue.
* There is a need for a solution to help users track their daily calorie intake, monitor progress, and stay aware of their habits with respect to their diet.
* This project aims to develop a system that allows individuals to log their meals, calculate total daily calories, and compare their intake with dietary targets, thus helping them make informed eating decisions.

**Synopsis:**

This will be a simple calorie tracking app that will help users log their meals and monitor daily intake. It lets users set a personal calorie goal and shows progress through a user-friendly interface. Key features include:

* Adding food items with calories
* Viewing dietary summaries
* Setting and tracking dietary targets

**Software Lifecycle Model - Iterative Waterfall Model:**

* **Requirements & Planning:** features like meal logging, calorie goals, and summary views.
* **Design:** Create UI, plan database structure, and flow diagrams.
* **Implementation (Iterative):** Develop modules in phases - food logging, goal tracking, then weekly stats.
* **Testing:** Perform unit and system testing at each step to ensure smooth functionality.
* **Maintenance:** Fix bugs, update food lists, and polish the UI as needed.

**Team Members:**

Aarav Bhatia (230905562)

Samarth Patil (230905342)

Dhruv Kaushik Thacker (230905566)

# 

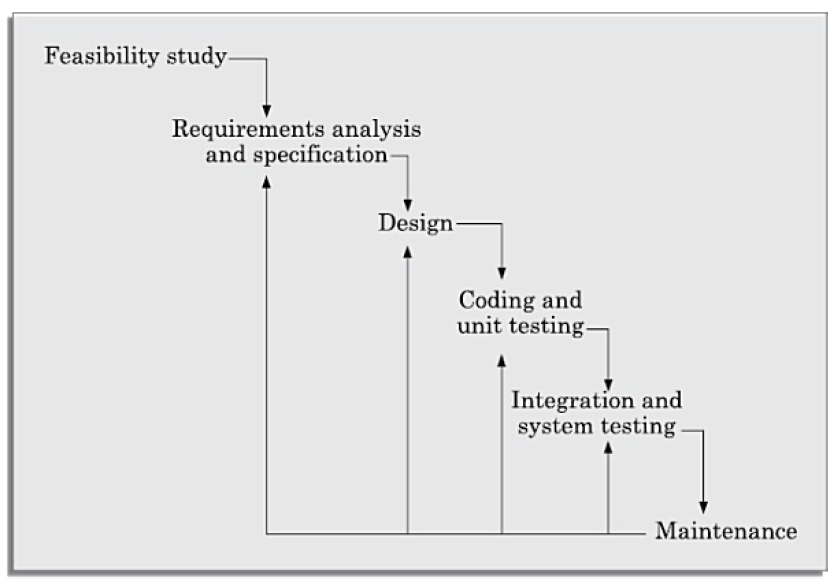
# 

Table of Contents

1. [Life Cycle Model 3](#_c1vwo6gzek14)
2. [Software Requirements Specification (SRS) 5](#_uipk7rrj14j9)
3. [Context Diagram (DFD Level-0) 10](#_w0so42o7ycgx)
4. [Data Flow Diagram 10](#_eosg8h2ouioz)
5. [Structured Diagram 14](#_t4owo5gknx0l)
6. [Data Dictionary 15](#_upw1lqooopsi)
7. [Use-Case Diagram: 16](#_5uqdo8fjaneo)
8. [Class Diagram 18](#_66siy79mi1bx)
9. [Activity Diagram for meal entry Use Case 20](#_bqbfivaacht5)
10. [Statechart diagram for meal entry 21](#_m4zcccmd2fv4)
11. [Statechart Diagram for Goal Object 22](#_s5cg8if0x63f)
12. [Sequence Diagram for Add Meal Entry 23](#_dn01phwaut7y)
13. [Collaboration Diagram: 25](#_4j5iv3md0841)
14. [Deployment Diagram 26](#_qvzwlog9a51q)
15. [Package Diagram 27](#_l74cj6manv1g)
16. [Object Diagram 28](#_sjb3j0cf96x7)
17. [Component Diagram 29](#_g7gcwacbsrbx)
18. Sample Screenshot 30
19. Conclusion 31

# Life Cycle Model

**Iterative Waterfall**



* The **Iterative Waterfall Model** is a refinement of the classical waterfall model.
* Unlike the pure waterfall, it allows **feedback loops** between phases — if errors are found in later phases, you can go back and correct earlier phases.
* This makes it more realistic and practical for projects, since requirements often evolve.

## **Phases of Iterative Waterfall (Applied to Diet & Calorie Tracker)**

1. **Feasibility Study**
   * Assess whether developing a web-based calorie tracker is technically and economically feasible.
   * Consider factors like available time, skill level, browser-only environment, and simple storage options.
   * Outcome: decision to proceed with a lightweight browser-based app.
2. **Requirements Analysis and Specification**
   * Collect and document functional requirements: add/manage food items, log meals, set daily calorie goals, view daily/weekly summaries.
   * Identify non-functional needs: simplicity, web-only, fast response.
   * Deliverable: **Software Requirements Specification (SRS)**, which serves as the baseline for design.
3. **System Design**
   * Translate requirements into system architecture and models.
   * Design data stores: **FoodDB (D1)**, **MealLogs (D2)**, **Goals (D3)**.
   * Prepare **Context, Level-1, Level-2 DFDs**, and structured chart to define module interactions.
   * Deliverable: design diagrams and logical database schema.
4. **Coding and Unit Testing**
   * Implement each module:  
     + Manage Food Items
     + Log Meals
     + Goals & Comparison
     + Generate Summaries
   * Write and test each module individually to ensure correctness.  
     Technology: browser-based web app using HTML, CSS, and JavaScript (with localStorage).
5. **Integration and System Testing**
   * Integrate all modules into a complete working system.
   * Perform **system testing** to check flows across modules (e.g., logged meals correctly update summaries, goals are applied properly).
   * Verify the system against the requirements specified in the SRS.
6. **Maintenance**
   * Provide long-term support: fixing bugs, adapting to browser updates, and enhancing features (like new types of summaries).
   * Iteration ensures that if problems are discovered during deployment or usage, the team can revisit earlier stages for corrections.

# Software Requirements Specification (SRS)

## **1. Introduction**

### **1.1 Purpose**

This SRS defines the requirements for the **Diet & Calorie Tracker**. It specifies all functional and non-functional requirements, interfaces, and constraints to guide design, development, and testing.

### **1.2 Scope**

The system is a simple calorie tracking web application for individuals. It enables users to log meals, track calories, set daily goals, and view summaries directly from a browser. The app helps raise awareness of dietary habits and supports healthier choices.

### **1.3 Environmental Characteristics**

* Must run exclusively on a web browser.
* Persistent storage handled through browser-supported methods (e.g., localStorage or IndexedDB).
* Should function without requiring installation.

## **2. Overall Description of Organization of SRS Document**

### **2.1 Product Perspective**

Browser-based web app. No dependency on native installation.

### **2.2 Product Features**

* Manage food items (name, unit, kcal).
* Log meals with multiple food items.
* Compute daily totals.
* Track daily goals.
* Display daily and weekly summaries.

### **2.3 User Classes**

* **End User:** Individual who wants to track calorie intake. Basic technical proficiency.

### **2.4 Operating Environment**

* Runs entirely within modern web browsers (Chrome, Firefox, Edge, Safari).
* Works with browser local storage.

### **2.5 Design & Environment Constraints**

* Web-only; no desktop or mobile native apps.
* Deterministic rounding of calorie totals.

### **2.6 User Documentation**

* Quick start guide/manual.

## **3. Functional Requirements**

### **3.1 Manage Food Items**

* **R.1.1 Add Food Item** **Description:** Register a new food item with name, unit, kcal-per-unit#.  
   **Input:** Food details.  
   **Output:** Confirmation of item saved.
* **R.1.2 Update Food Item** **Description:** Modify details of an existing food item.  
   **Input:** Food name, updated details.  
   **Output:** Confirmation of update.
* **R.1.3 Delete Food Item** **Description:** Delete an item from the list. Historical meals remain unchanged.  
   **Input:** Food name.  
   **Output:** Confirmation message.
* **R.1.4 View Food List** **Description:** Display all stored food items.  
   **Output:** Food list.

### **3.2 Manage Meal Entries**

* **R.2.1 Add Meal Entry** **Description:** Log a new meal with date, time, and multiple (food-item, quantity) pairs.  
   **Input:** Meal details.  
   **Output:** Confirmation of entry saved; computed entry total.
* **R.2.2 Update Meal Entry** **Description:** Modify existing meal entry.  
   **Input:** Meal ID, new details.  
   **Output:** Confirmation.
* **R.2.3 Delete Meal Entry** **Description:** Delete a logged meal.  
   **Input:** Meal ID.  
   **Output:** Confirmation message.
* **R.2.4 View Daily Meals** **Description:** Display all meals logged for a date.  
   **Input:** Date.  
   **Output:** List of meals.

### **3.3 Manage Goals & Summaries**

* **R.3.1 Set Daily Goal** **Description:** Allow user to set/update a daily calorie target.  
   **Input:** Target kcal.  
   **Output:** Confirmation.
* **R.3.2 Compare Totals vs Goal** **Description:** Calculate difference between daily total and goal.  
   **Output:** Remaining or over-goal status.
* **R.3.3 View Daily Summary** **Description:** Display total intake, goal, and difference.  
   **Input:** Date.  
   **Output:** Summary report.
* **R.3.4 View Weekly Summary** **Description:** Display 7-day totals with goal comparisons.  
   **Input:** Week range.  
   **Output:** Weekly summary report.

### **3.4 Data Management**

* **R.4.1 Undo/Delete Safety** **Description:** Provide confirmation prompts or Undo for destructive actions.  
   **Output:** Confirmation or restored item.

## **4. External Interface Requirements**

### **4.1 User Interface**

* Clean, responsive web UI. Core actions ≤2 clicks away.
* Progress bar for goal tracking.

### **4.2 Hardware Interfaces**

* Standard input/output via browser (keyboard, mouse, touchscreen).

### **4.3 Software Interfaces**

* Browser local storage (IndexedDB/localStorage).

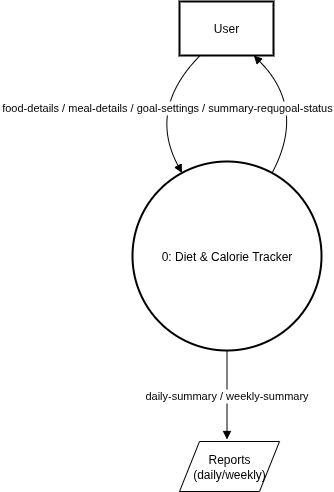
### **4.4 Communication Interfaces**

* None required.

## **5. Non-Functional Requirements**

* **N.1 Performance:** Web app loads home page in ≤5s. Meal entry saves in ≤1s.
* **N.2 Reliability:** No data loss after confirmed save.
* **N.3 Usability:** First-time user can add a meal in <180s. Responsive design across devices.
* **N.4 Security:** Local storage only.
* **N.5 Portability:** Runs in modern browsers across OS.
* **N.6 Maintainability:** Code modular, unit tested. Uses standard web frameworks.
* **N.7 Database:** Use browser-supported storage.

# Context Diagram (DFD Level-0)



At Level 0, the single process “0: Diet & Calorie Tracker” receives user inputs—food-details, meal-details, goal-settings, and summary-requests—and returns goal-status to the user while producing daily/weekly summaries as reports.

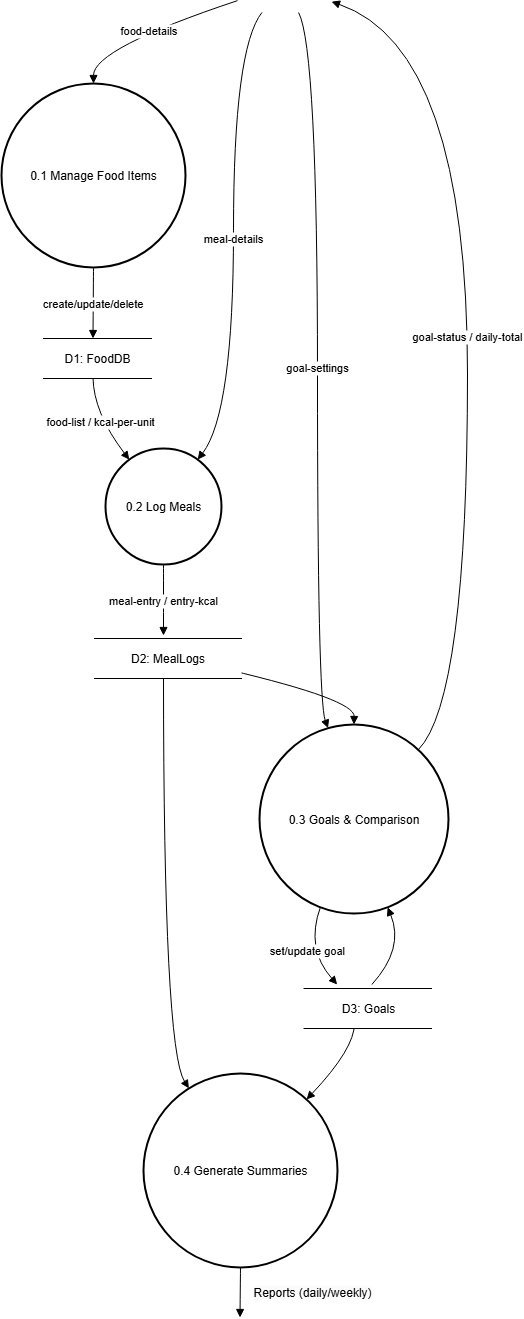
• Example: A user submits a summary-request; the system compiles recent logs and goals to output a daily or weekly report.

• Goal-status (e.g., progress vs. target) is sent back to the user in real time.

# 

# Data Flow Diagram

**DFD Level-1**

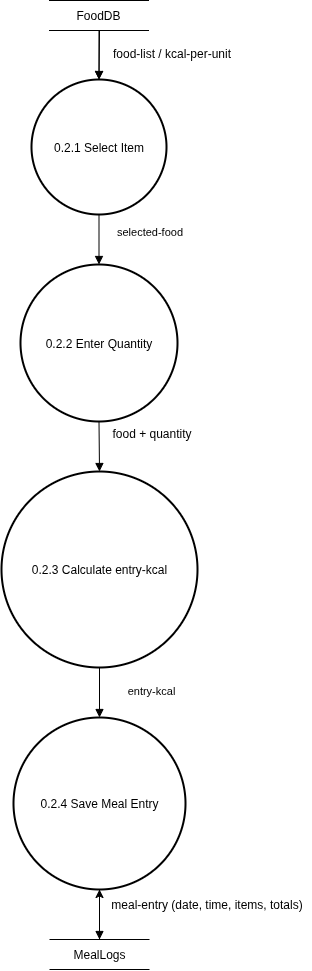
****

At Level 1, the Level 0 process is decomposed into four major sub-processes:

1. Manage Food Items (0.1)
2. Log Meals (0.2)
3. Goals & Comparison (0.3)
4. Generate Summaries (0.4)

Each process communicates with the appropriate data stores—FoodDB (D1), MealLogs (D2), and Goals (D3)—and produces periodic reports for users.  
 • Example: The user provides food-details to “Manage Food Items (0.1),” which creates/updates/deletes items in FoodDB (D1).  
 • When the user logs a meal, “Log Meals (0.2)” retrieves food-list and kcal-per-unit from FoodDB (D1), then writes meal-entry with computed entry-kcal to MealLogs (D2).  
 • “Goals & Comparison (0.3)” stores new or updated goal-settings in Goals (D3), reads both MealLogs (D2) and Goals (D3), and returns goal-status/daily-total to the user.  
 • “Generate Summaries (0.4)” aggregates MealLogs (D2) and Goals (D3) to produce daily/weekly Reports.

**DFD Level-2**

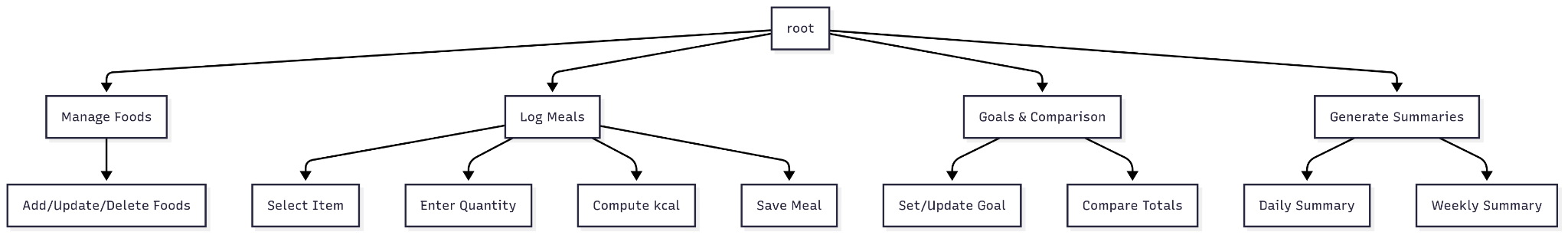
****

DFD Level-2 (for 0.2 Log Meals): the Level 1 process “Log Meals (0.2)” is decomposed into four sub-processes:

1. Select Item (0.2.1)
2. Enter Quantity (0.2.2)
3. Calculate entry-kcal (0.2.3)
4. Save Meal Entry (0.2.4)

Each sub-process interacts with FoodDB (D1) and MealLogs (D2) as needed.  
 • Example: “Select Item (0.2.1)” fetches the chosen food’s data from FoodDB (D1); “Enter Quantity (0.2.2)” captures serving size; “Calculate entry-kcal (0.2.3)” multiplies quantity by kcal-per-unit from FoodDB (D1); “Save Meal Entry (0.2.4)” persists the finalized meal-entry—including computed entry-kcal—into MealLogs (D2).

# Structured Diagram

****

The structured diagram presents a hierarchical decomposition of the Diet & Calorie Tracker into four primary modules: Manage Foods, Log Meals, Goals & Comparison, and Generate Summaries. Each top-level module is further refined into focused subfunctions that clarify responsibilities and simplify maintenance.

* Manage Foods → Add/Update/Delete Foods: Centralizes CRUD operations for the food catalog. Ensures consistency of name, unit, and kcal-per-unit across the system.
* Log Meals → Select Item → Enter Quantity → Compute kcal → Save Meal: Orchestrates user flow for adding meals, validating quantities, deriving per-entry kcal, and persisting meal entries.
* Goals & Comparison → Set/Update Goal → Compare Totals: Stores daily goal targets and computes progress by contrasting daily totals against the goal.
* Generate Summaries → Daily Summary → Weekly Summary: Aggregates meal logs into day- and week-level summaries for reporting and user feedback.

This decomposition emphasizes separation of concerns and supports module-level testing and evolution without cross-module impact.

# Data Dictionary

The data dictionary defines canonical structures and units used across modules:

* **food-item** = name (string) + unit (string) + kcal-per-unit (int)
* **meal-entry** = date (dd/mm/yyyy) + time (hh:mm) + set of (food-item + quantity:int)
* **goal** = date + target-kcal:int
* **summary** = date + total-kcal:int + goal-kcal:int + difference:int
* **Data Stores**: **D1: foodDB** (set of food-item), **D2: mealLogs** (set of meal-entry), **D3: goals** (set of goal)
* **Outputs**: **daily-summary** (summary per day), **weekly-summary** (7-day collection of summaries)

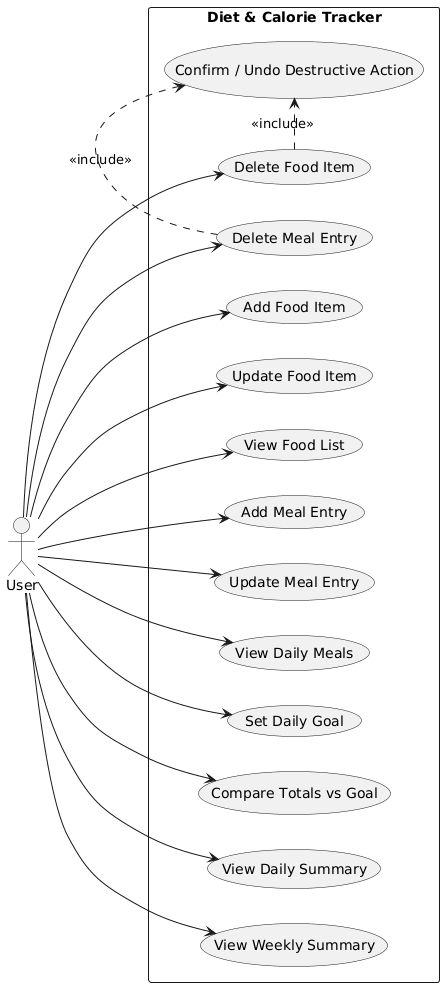
### **Outputs**

reports = { summary }\*

daily-summary = summary (per day)

weekly-summary = { summary }\* (7 days)

# Use-Case Diagram:

****

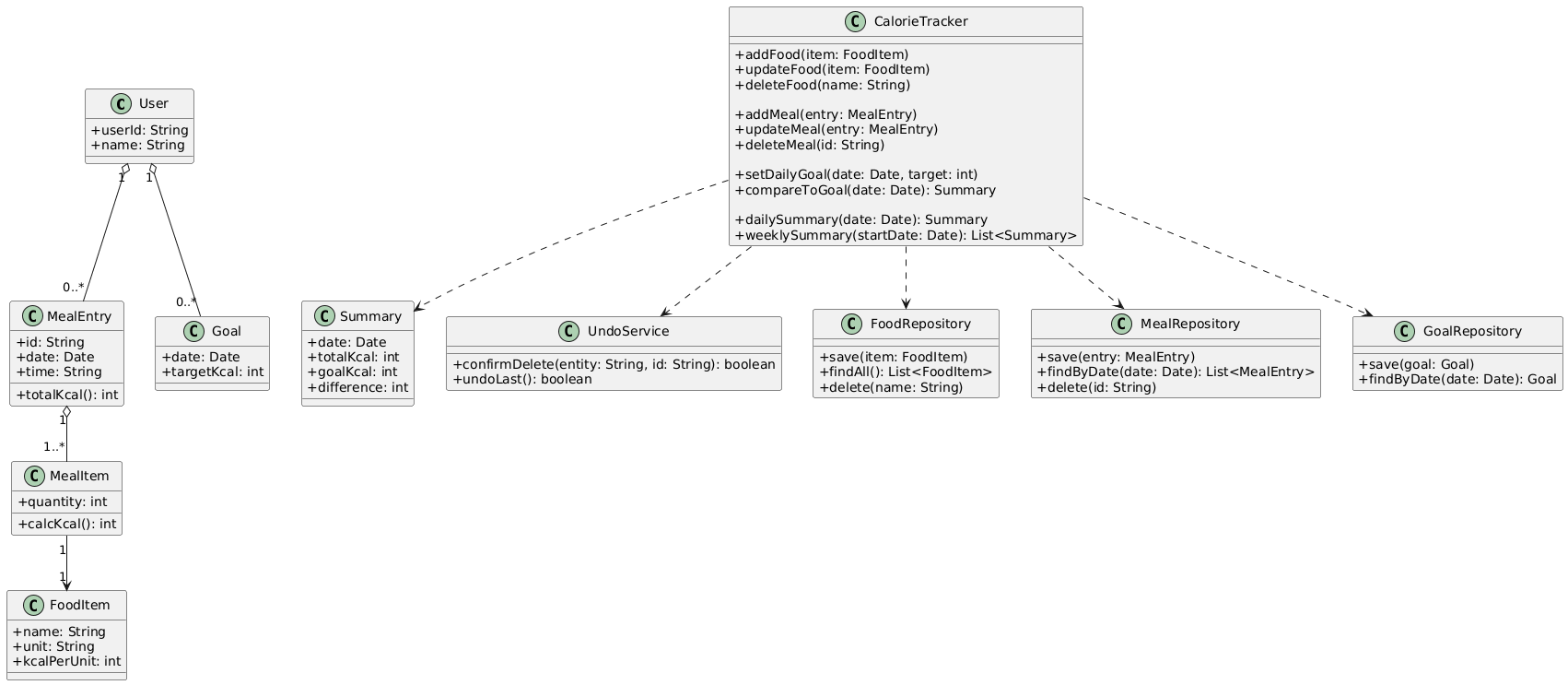
The use-case model captures how the User interacts with application functions: managing foods and meals, setting goals, and viewing summaries. It also introduces a safety mechanism for destructive operations.

Primary Use Cases

* Add/Update/Delete Food Item; View Food List: Maintain the food catalog displayed to users when logging meals.
* Add/Update/Delete Meal Entry; View Daily Meals: Support full lifecycle of meal logging with visibility into the day’s entries.
* Set Daily Goal; Compare Totals vs Goal: Persist the target kcal for a date and display progress (met/remaining).
* View Daily Summary; View Weekly Summary: Generate quick feedback artifacts for short- and medium-term tracking.
* Confirm / Undo Destructive Action (<<include>> for Deletes): Ensures irreversible actions (deleting food/meal) are confirmed or reversible, preserving data integrity.

The actor User initiates each use case; delete operations include a confirmation/undo flow to reduce accidental data loss.

# Class Diagram:

****

The class model defines core entities, application services, and repository interfaces that isolate persistence:

**Entities**

* **User** (userId, name)
* **FoodItem** (name, unit, kcalPerUnit)
* **MealItem** (quantity, calcKcal()) links to **FoodItem**
* **MealEntry** (id, date, time, totalKcal()) aggregates **MealItem**
* **Goal** (date, targetKcal)
* **Summary** (date, totalKcal, goalKcal, difference)

**Application Layer**

* **CalorieTracker** orchestrates: food CRUD; meal CRUD; goal setting; comparisons; dailySummary() and weeklySummary().
* **UndoService** supports confirmation and undo for destructive actions.

**Persistence Ports**

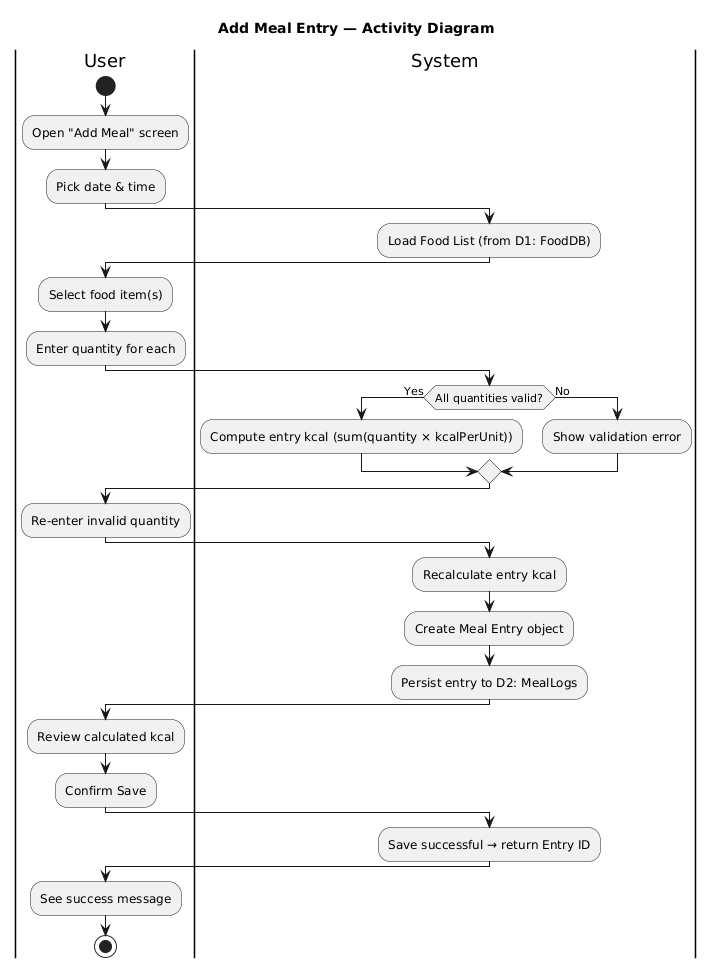
* **FoodRepository**, **MealRepository**, **GoalRepository** expose minimal, intention-revealing methods for storage operations.

**Associations**

* A **User** owns many **MealEntry** and **Goal** objects.
* **MealEntry** composes **MealItem**; each **MealItem** references one **FoodItem**.
* **CalorieTracker** depends on repositories, **Summary**, and **UndoService**.

This layout enforces clean boundaries, testability, and the ability to swap storage without altering business logic.

# Activity Diagram for meal entry Use Case:

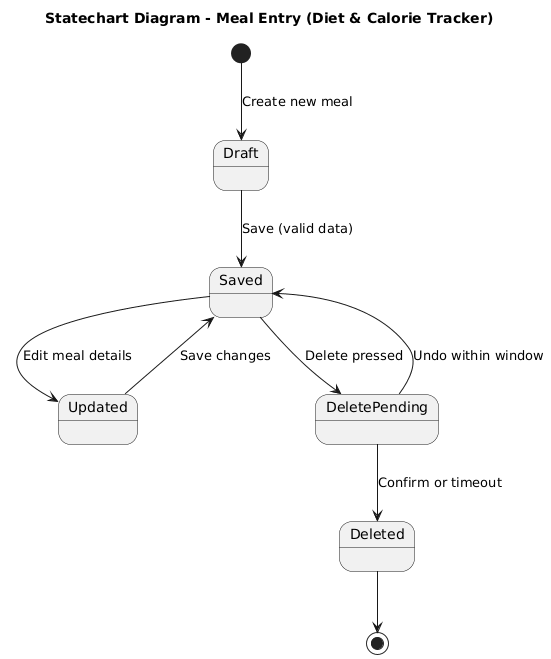
****

The activity flow partitions responsibilities between **User** and **System**:

1. The user opens **Add Meal** and selects date/time.
2. The system loads the **Food List** (from D1).
3. The user selects food items and enters quantities.
4. System validates quantities; on success, computes **entry kcal** (Σ quantity × kcalPerUnit).
5. The system creates a **Meal Entry** and persists to **D2: MealLogs**.
6. User reviews and confirms; system returns success and the new entry ID.

Validation loops guard against invalid input; computation occurs prior to persistence to ensure stored entries include their derived kcal.

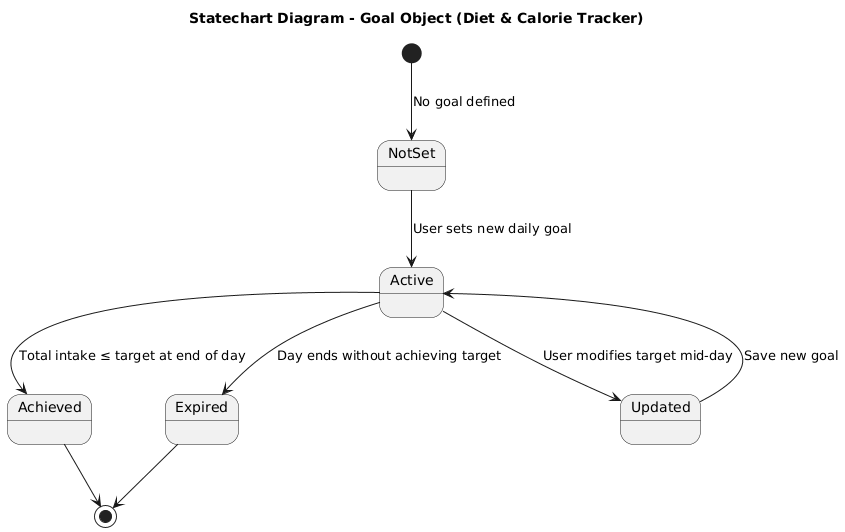
# Statechart diagram for meal entry:

****

* **Draft**: Newly created, editable entry.
* **Saved**: Validated and persisted.
* **Updated**: Post-edit state; saving returns to **Saved**.
* **DeletePending**: Deletion requested; **Undo** reverts to **Saved**, confirmation or timeout finalizes to **Deleted**.

The model captures the safety window around deletion, aligning with the undo/confirm requirement for destructive actions.

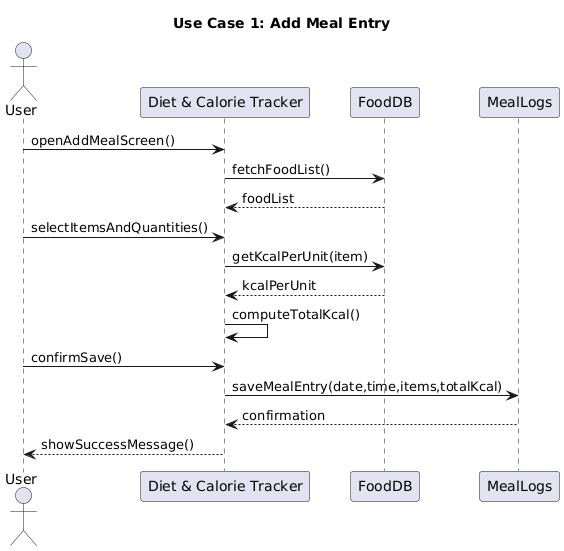
# Statechart Diagram for Goal Object

****

* **NotSet** → **Active** when a new daily goal is saved.
* **Active** → **Achieved** if total intake ≤ target at end of day; → **Expired** otherwise.
* **Active** → **Updated** when user modifies the target mid-day; saving returns to **Active**.

This lifecycle formalizes day-scoped evaluation of progress and mid-day adjustments.

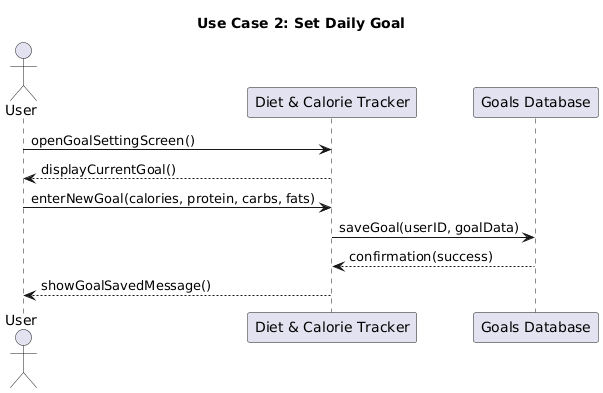
# Sequence Diagram for Add Meal Entry:



1. **User → DCT**: Open add screen.
2. **DCT → FoodDB (D1)**: Fetch food list.
3. **User → DCT**: Provide items & quantities.
4. **DCT ↔ D1**: Lookup kcal per unit; **DCT** computes total kcal.
5. **User → DCT**: Confirm save.
6. **DCT → MealLogs (D2)**: Persist entry; return confirmation to user.

The sequence emphasizes pre-save computation, repository-backed persistence, and clear success feedback.

**Sequence Diagram for Set Daily Goal**

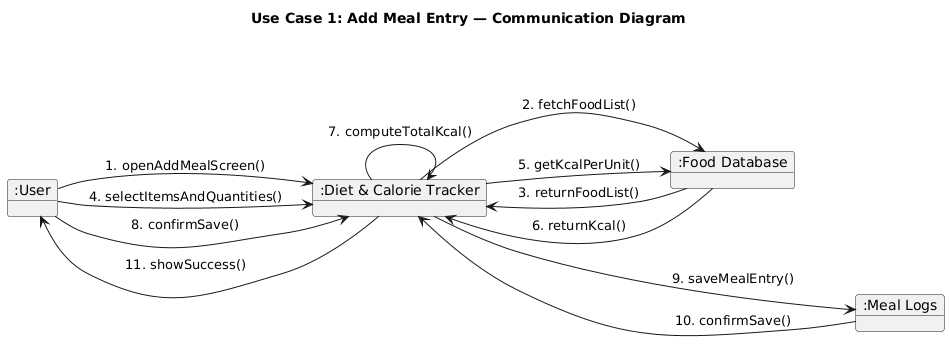


1. **User → DCT**: Open goal setting screen; current goal displayed (if any).
2. **User → DCT**: Enter new goal (calories or macro targets as supported).
3. **DCT → Goals DB (D3)**: Save goal; confirmation returned.
4. **DCT → User**: Show “goal saved” message.

This interaction ensures idempotent storage per date and immediate acknowledgement.

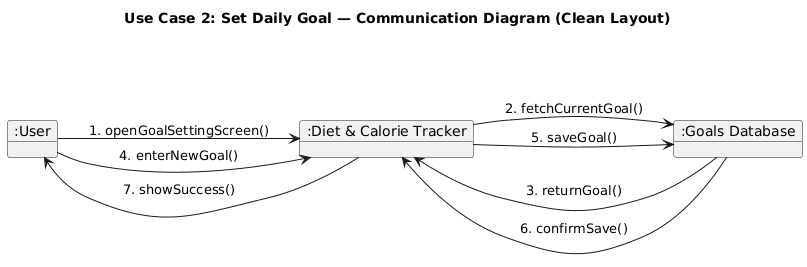
# Collaboration Diagram:

1. **For Add Meal Entry:**



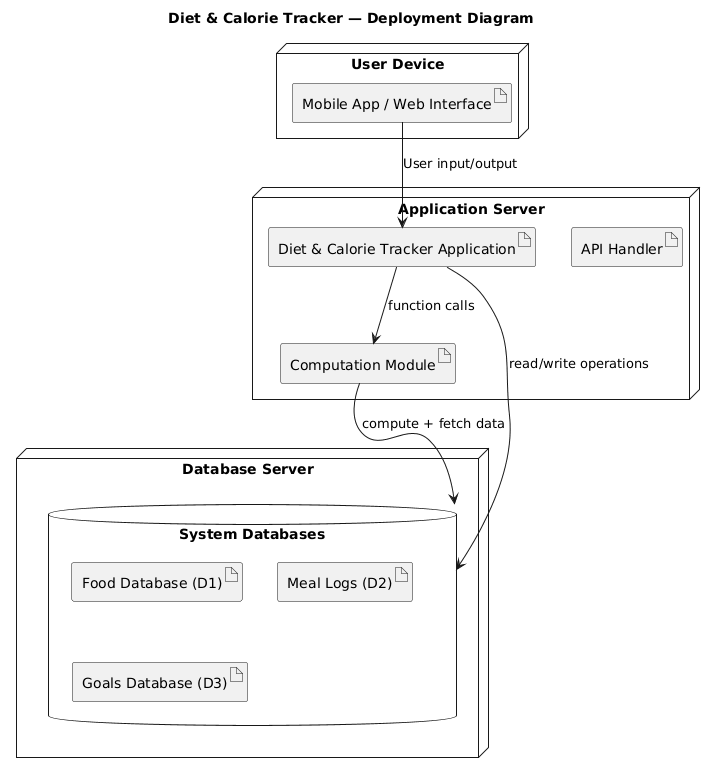
Objects (**User**, **Diet & Calorie Tracker**, **Food Database**, **Meal Logs**) exchange numbered messages: open screen → fetch list → return list → select items → fetch kcal → compute → confirm save → persist → confirm → show success. The collaboration view complements the sequence diagram by highlighting object roles and message order on a single canvas.

1. **For Set Daily Goal:**



Objects (**User**, **Diet & Calorie Tracker**, **Goals Database**) exchange: open goal screen → fetch/return current goal → enter new goal → save → confirm → show success. The diagram stresses minimal coupling and straightforward persistence.

# Deployment Diagram:

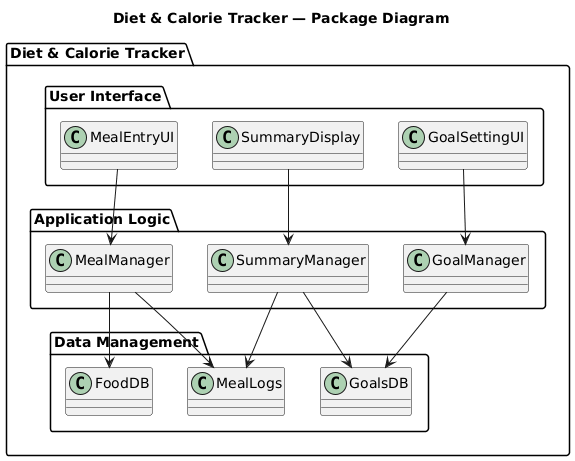


**User Device** runs the **Mobile/Web UI**, connecting over HTTPS to the **Application Server** hosting the **Diet & Calorie Tracker Application**, its **Computation Module**, and **API Handler**. A separate **Database Server** hosts **Food Database (D1)**, **Meal Logs (D2)**, and **Goals Database (D3)**.

* **UI ↔ App**: user I/O and API calls
* **App ↔ DB**: transactional reads/writes
* **App ↔ Computation Module**: function calls for kcal and summaries

This topology supports clear separation of concerns and horizontal scalability (app and DB tiers can scale independently).

# Package Diagram:

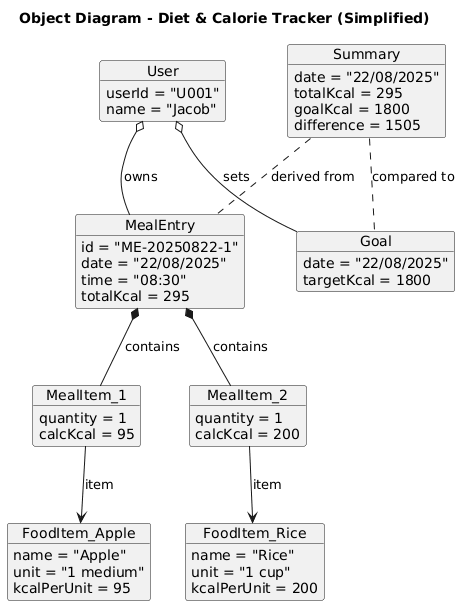


Within **Diet & Calorie Tracker**:

* **User Interface**: MealEntryUI, GoalSettingUI, SummaryDisplay (presentation and input handling)
* **Application Logic**: MealManager, GoalManager, SummaryManager (business rules and orchestration)
* **Data Management**: FoodDB, MealLogs, GoalsDB (persistence abstractions)

Directed dependencies flow from UI → Logic → Data, enforcing a layered architecture and easing unit/integration testing.

# Object Diagram:

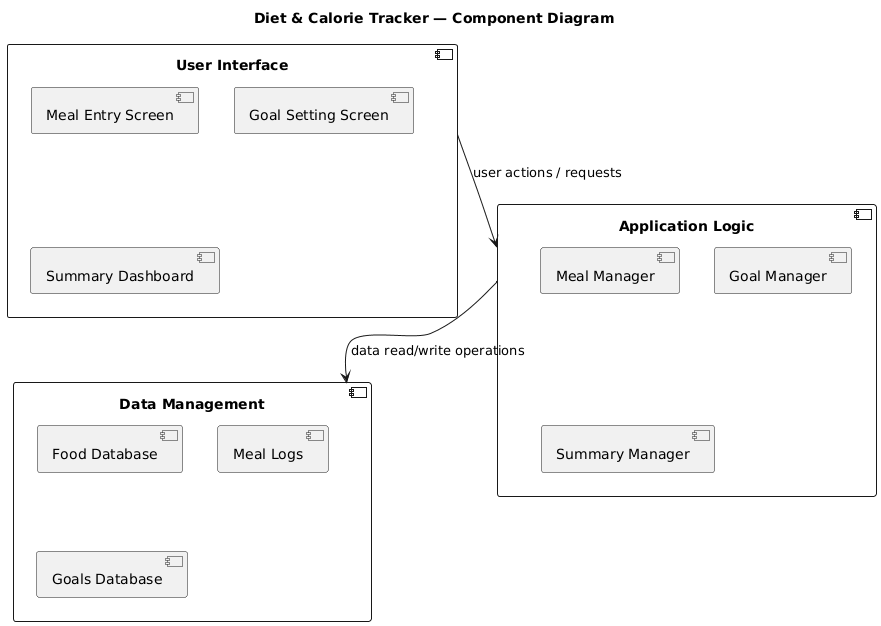


Instances illustrate a concrete day:

* **User** U001/Jacob
* **FoodItem\_Apple (95 kcal)**, **FoodItem\_Rice (200 kcal)**
* **MealItem\_1 (qty 1 → 95 kcal)**, **MealItem\_2 (qty 1 → 200 kcal)**
* **MealEntry** ME-20250822-1 totaling **295 kcal** at **08:30**
* **Goal** for **22/08/2025** with **targetKcal = 1800**
* **Summary** for **22/08/2025**: totalKcal=295, goalKcal=1800, difference=1505

Associations show ownership (User→MealEntry/Goal), composition (MealEntry→MealItems), references (MealItem→FoodItem), and derivations (Summary from MealEntry + Goal).

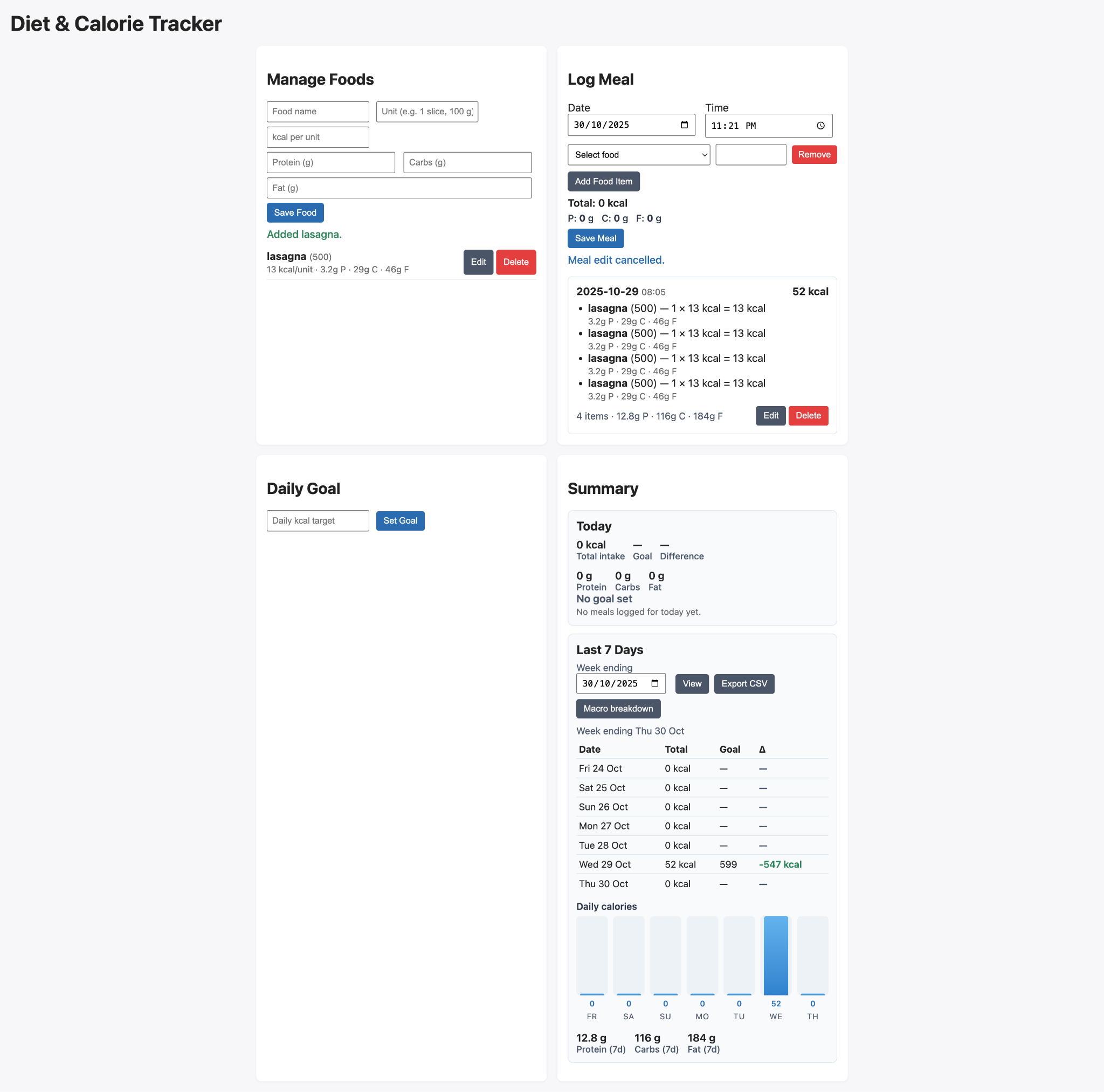
# Component Diagram:



* **User Interface**: Meal Entry Screen, Goal Setting Screen, Summary Dashboard
* **Application Logic**: Meal Manager, Goal Manager, Summary Manager
* **Data Management**: Food Database, Meal Logs, Goals Database

Interfaces between components align with the package layering: UI components use Logic services, which in turn depend on Data components. This modularization promotes maintainability and targeted reuse.

# Sample Screenshot of the Application:



**Conclusion**

The project successfully demonstrates the complete software development process using the Iterative Waterfall Model. Each phase ,requirements, design, implementation, testing, and maintenance was clearly defined and executed.

Through this system, users can:

* Log daily meals and calories,
* Set and compare calorie goals, and
* View daily and weekly summaries of progress.

The design emphasizes modularity, simplicity, and maintainability, achieved through layered architecture (UI → Logic → Data) and structured diagrams (DFDs, use-case, class, sequence, and component). By leveraging browser-based technologies (HTML, CSS, JavaScript, localStorage), it provides accessibility without installation, ensuring lightweight performance and data persistence on the client side.

Overall, the Diet & Calorie Tracker achieves its aim of helping users monitor and manage calorie intake efficiently while reinforcing practical understanding of software engineering principles like system decomposition, abstraction, and iterative refinement.