2.1 Introduction

2.1.1 Purpose

This Software Requirements Specification (SRS) defines the functional and non-functional requirements for the Personal Food Log App commissioned by Digital Health Inc. It describes what the system shall do in order to:

- allow users to log food intake via smartphone photos,
- identify visible ingredients,
- estimate caloric content using volume and density approximations, and
- support scalability for millions of images per day.

This SRS is used by:

- The client (Digital Health Inc., represented by product management and other stakeholders) to validate that the proposed system meets their needs.
- The project team as the baseline for design, implementation, and testing.
- Future development teams at Digital Health Inc. to evolve the prototype into a production system.

Note: The proof-of-concept (PoC) implemented for the initial phase of the project will realize a subset of these requirements. Each requirement is marked as PoC (must be met in the current project phase) or Future (planned for the full product).

2.1.2 Scope

The Personal Food Log App is a mobile application (Android and iOS) with a cloud backend hosted on Amazon Web Services (AWS).

At a high level, the system will:

- Allow users to capture or upload photos of meals.
- Use AI-based visual analysis to detect visually distinguishable food ingredients in the image.
- Approximate ingredient volume and weight by:
    - segmenting ingredients on the visible surface,
    - estimating 3D volume assuming uniform depth,
    - converting volume to weight using density tables, and
    - converting weight to calories using nutrition tables.
- Provide users with a per-meal summary (ingredients, portion estimates, calories) and daily intake summaries.

- Support incremental model improvement by leveraging newly uploaded images over time.
- Be architected to support high scalability (millions of images per day) in production.

Out of scope for this SRS (but potentially future extensions beyond the initial release):

- Integration with external health platforms (Fitbit, Apple Health, EHR systems).
- Advanced nutritional analysis (micronutrients, non-visible ingredients beyond reasonable estimation).
- Billing, subscriptions, and social features.

2.1.3 Definitions, Acronyms, and Abbreviations

AI – Artificial Intelligence.
AWS – Amazon Web Services.
PoC – Proof of Concept.
VBM – Visual-Based Measurement systems for food logging.
User – End user of the mobile application (patients, athletes, fitness users).
Clinician – Dietician, doctor, or coach who may later review user data (primarily future scope).
Ingredient – A visually distinguishable food component (e.g., lettuce, tomato, rice).
CCB – Change Control Board (PM + Customer).

2.1.4 References

- "Personal Food Log App – Product Brief", Fall 2025.
- Project Charter v1.1 – "Personal Food Log App", 2025-10-11.
- SPMP v1.1 – "Software Project Management Plan – Personal Food Log App", 2025-11-05.
- IEEE Std 1058-1998 – Software Project Management Plans.

2.1.5 Overview

The remainder of this SRS is structured as follows:

- 2.2 Assumptions and Constraints – domain and technical assumptions, plus limiting factors.
- 2.3 Functional Requirements – detailed use cases and functional requirements.
- 2.4 Non-Functional Requirements – quality attributes such as performance, usability, security, and scalability.
- 2.5 External and Internal Interfaces – user interfaces, external systems, and internal component interfaces.

2.2 Assumptions and Constraints (brief)

2.2.1 Assumptions

A1. Ingredient-level accuracy is the primary value.
The system's main selling point is its ability to accurately identify visible ingredients and estimate calories; PoC may use placeholder or simplified calculations, but the architecture must support higher accuracy later.

A2. Visible ingredients extend uniformly in depth.
For volume estimation, any ingredient visible on the surface of the dish is assumed to continue downward with approximately the same density and composition.

A3. Certain nutrients are inherently non-visible.
Non-visible elements such as salt, oil within mixtures, spices, and micronutrients cannot be reliably detected. Approximations or default assumptions are acceptable.

A4. Auto-calibration uses image-based methods only.
The system assumes that camera auto-calibration techniques can estimate scale from the image itself and must not rely on external reference objects (coins, thumbs, rulers).

A5. Availability of public food datasets.
The system assumes that appropriate public food image datasets and food density/nutrition tables are available and legally usable for model training and calibration.

A6. AWS cloud availability and free-tier usage.
For the initial PoC, the team assumes access to the AWS free tier, including services such as S3 and Lambda.

A7. Smartphone capabilities.
Target users possess modern Android or iOS smartphones with a functional camera, a stable internet connection for upload, and sufficient storage and processing power for the app.

A8. User consent and non-PII images.
Images used for PoC and training are assumed to be either non-personally identifiable or captured with informed consent, in accordance with privacy norms (e.g., PIPEDA).

2.2.2 Constraints

C1. Platform Constraint – Mobile OS.
The app must support both Android and iOS platforms for end users (PoC may target at least one platform with design ready for both).

C2. Cloud Constraint – AWS Only.
The backend must use AWS and be compatible with the company's existing Amazon Cloud infrastructure. No alternate cloud providers are allowed.

C3. Scalability Constraint.
The architecture must be designed to scale to millions of images per day, even though the PoC will process far fewer images.

C4. Auto-Calibration Constraint.
The calibration algorithm must not require external calibration objects placed in the image by the user.

C5. Project Deadline Constraint.
All PoC functionality must be delivered by November 20, 2025, with presentation and demo during November 21–28, 2025. This places a hard constraint on PoC scope.

C6. Team and Resource Constraint.
Implementation is performed by a 4-person team, within typical part-time availability (≈ 6 h/week per team member), using primarily free tools and services.

C7. Regulatory / Privacy Constraint.
The system must handle any real user data in a way that is consistent with privacy regulations (e.g., PIPEDA) and internal policies of Digital Health Inc.

2.3 Functional Requirements

2.3.1 Actors

End User – Individual using the app to log meals and track calorie intake.
Clinician / Coach (Future) – Reviews user logs and trends.
System Administrator (Future) – Manages configurations and datasets.
Data Scientist / ML Engineer (Future) – Uses collected data to improve AI models.

For the initial PoC, the primary actor is the End User; admin and clinician features are mainly future scope.

2.3.2 Use Case Overview

Core use cases (PoC / Future):

- UC-1 Log Meal via Photo (PoC)
- UC-2 View Daily Intake Summary (PoC – basic)
- UC-3 View Meal History (PoC – basic list)
- UC-4 Manage Profile & Preferences (PoC – minimal)
- UC-5 Manage Datasets & Training Hooks (Future)
- UC-6 Clinician Reviews User Logs (Future)

2.3.3 Use Case Descriptions

UC-1: Log Meal via Photo

Primary Actor: End User
Goal: Capture a photo of a meal, upload it, and receive estimated ingredients and calories.
Scope: Mobile app + backend.

Preconditions:

- User has installed the app and is authenticated (if required for PoC).
- Device has a working camera and network connectivity (for online path).

Postconditions:

- A new meal entry is stored in the user's log with attached image, detected ingredients, and calorie estimates (or placeholders in PoC).

Main Flow:

1. User selects "Log Meal" from the home screen.
2. System opens the camera interface.
3. User captures a photo of the meal (or selects an existing photo from gallery if allowed).
4. System displays a preview and allows the user to confirm or retake the photo.
5. On confirmation, the app sends the photo and basic metadata (timestamp, device info) to the backend over a secure channel.
6. Backend performs:
   6.1 Auto-calibration to determine scale from the image.
   6.2 Visual segmentation of ingredients.
   6.3 Volume and weight estimation for each visible ingredient.
   6.4 Calorie estimation using nutrition tables.

7. Backend returns structured data: ingredient list, estimated weights, per-ingredient calories, and total calories.
8. App displays a result screen summarizing the thumbnail of the meal photo, list of detected ingredients, and estimated calories.
9. User optionally edits ingredient labels or flags incorrect items.
10. User confirms to save the meal to their log.

Alternate / Exception Flows:

A1 – Low confidence detection:

- If the system's confidence in ingredient detection falls below a threshold, it marks items as low confidence and/or prompts the user for manual input.

A2 – Network unavailable:

- If network is not available during upload, the app offers to save the photo locally and retry later.

A3 – Backend error:

- If backend processing fails, the app displays an error and allows retry or cancel.

UC-2: View Daily Intake Summary

Primary Actor: End User
Goal: See a summary of calories consumed during a specific day.

Main Flow:

1. User opens the app and navigates to "Today" (or selects a date).
2. System loads all meal entries for that day.
3. System displays total estimated daily calories, number of meals logged, and a list of meals with thumbnails and per-meal calorie totals.

Alternate Flow:

- If no meals exist for the selected day, system displays a friendly message encouraging user to log meals.

UC-3: View Meal History

Primary Actor: End User

Main Flow:

1. User selects “History” from the menu.
2. System displays a paginated list of past meals with date/time, thumbnails, and calories.
3. User taps a meal to view detailed ingredient breakdown, calorie estimates, and original photo.

UC-4: Manage Profile & Preferences

Primary Actor: End User

Main Flow (PoC – Minimal):

1. User accesses “Profile” screen.
2. System displays fields such as name, email, and optional dietary preferences.
3. User edits and saves; system validates and persists changes.

2.3.4 Detailed Functional Requirements

Each requirement is labeled FR-x and tagged as [PoC] or [Future].

2.3.4.1 User Management

FR-1 [Future] – Basic Account / Session
The system shall allow users to maintain a session so that meal logs are associated with a specific user or device.

FR-2 [Future] – Full Authentication
The system shall provide secure user registration and authentication using email/password or federated identity providers, integrated with AWS authentication services.

FR-3 [Future] – Multi-User Organization
The system shall support linking end-user accounts with clinician/coach accounts for data sharing, restricted by user consent.

2.3.4.2 Meal Capture and Upload

FR-4 [PoC] – Capture Meal Photo
The mobile app shall allow the user to capture a photo using the device camera from within the app.

FR-5 [PoC] – Photo Preview and Confirmation
After capturing, the app shall display a preview and allow the user to retake or confirm the photo.

FR-6 [PoC] – Secure Photo Upload
Upon confirmation, the app shall upload the image and relevant metadata to the backend over an encrypted connection.

FR-7 [Future] – Multi-Angle Capture (Optional)
The app should support capturing multiple angles of the same meal to improve volume estimation accuracy.

2.3.4.3 Auto-Calibration and Ingredient Detection

FR-9 [Future] – Auto-Calibration without External Objects
The backend shall apply camera auto-calibration methods that infer scale and depth from visual features alone, without requiring external reference objects.

FR-10 [PoC] – Calibration Stub
For the PoC, the backend shall provide a calibration stub that simulates or approximates scale sufficiently to produce plausible portion sizes for demo images.

FR-11 [Future] – Ingredient Segmentation
The backend shall segment the image into regions corresponding to distinct visible ingredients.

FR-12 [Future] – Ingredient Classification
For each segmented region, the system shall classify the ingredient type using an AI model trained on public food datasets.

FR-13 [Future] – Confidence Scoring
For each detected ingredient, the system shall compute and return a confidence score between 0 and 1.

2.3.4.4 Volume, Weight, and Calorie Estimation

FR-14 [Future] – Volume Estimation
The system shall estimate the volume of each visible ingredient based on its segmented area, estimated depth, and calibration parameters.

FR-15 [Future] – Weight Estimation Using Density Tables
The system shall convert estimated ingredient volumes to weights using a configurable food density table.

FR-16 [Future] – Calorie Estimation Using Nutrition Tables
The system shall calculate calories per ingredient using known nutrition information and aggregate them to a total per meal.

FR-17 [PoC] – Placeholder Calorie Estimates
For the PoC, the system shall return plausible placeholder calorie estimates to demonstrate the full pipeline even if AI accuracy is not production-grade.

2.3.4.5 Meal Log and History

FR-18 [PoC] – Persist Meal Entries
The system shall persist meal entries including user identifier, timestamp, image reference, ingredient list, and calorie estimates.

FR-19 [Future] – Daily Summary View
The system shall provide a view that summarizes total calories and meal count for a selected day.

FR-20 [Future] – Meal Detail View
The system shall allow the user to view a detailed breakdown of any logged meal.

FR-21 [Future] – Browse by Date Range
The system shall allow users to filter and browse meal history by date range.

FR-22 [Future] – Trends and Basic Analytics
The system shall provide simple visualizations (e.g., weekly average calories) to support behavior changes.

2.3.4.6 User Editing and Feedback

FR-23 [Future] – Ingredient Label Editing
The user shall be able to correct ingredient labels before saving the meal.

FR-24 [Future] – Flagging Incorrect Results
The user shall be able to flag ingredients or entire meals as incorrect or low-quality.

FR-25 [Future] – Basic Notes
The system shall allow users to add a short text note to a meal.

2.3.4.7 Data and AI Training Hooks

FR-26 [Future] – Data Storage for Model Training
The system shall store selected images and their associated labels in a format suitable for training.

FR-27 [Future] – Incremental Learning Interface
The system shall expose an internal interface for incremental training, enabling models to be updated with new data subsets.

FR-28 [PoC] – Logging for Synthetic Training
The PoC shall at minimum log processed images and outputs in a way that could be reused later to bootstrap model training.

2.3.4.8 Administration and Configuration (Future)

FR-29 [Future] – Admin Portal Access
The system shall provide an admin interface restricted to authorized staff to manage food datasets, nutrition tables, and density tables.

FR-30 [Future] – Configuration of Thresholds
Admins shall be able to configure parameters such as confidence thresholds and default densities.

2.4 Non-Functional Requirements

2.4.1 Performance

NFR-1 – Response Time (PoC)
For typical demo images (under 5 MB) over a stable connection, the PoC system should return analysis results within ≤ 5 seconds in at least 80% of cases.

NFR-2 – Response Time (Target System)
The production-grade system shall return results within ≤ 3 seconds for 95% of requests under normal load.

NFR-3 – Throughput and Scalability
The architecture shall support horizontal scaling to handle millions of images per day by adding more compute resources.

2.4.2 Reliability and Availability

NFR-4 – Robustness to Failure
The system shall handle backend errors gracefully, returning appropriate error messages and not crashing the client application.

NFR-5 – Availability (Target System)
The production-grade system should target 99.5% uptime per month, excluding scheduled maintenance.

NFR-6 – Data Integrity
Meal entries shall not be partially saved; they must be persisted atomically or not at all.

2.4.3 Usability and Accessibility

NFR-7 – Ease of Use
A novice user shall be able to complete UC-1 (log a meal) the first time without training, in ≤ 5 steps from the home screen.

NFR-8 – Consistent UI
The app shall use a consistent color scheme, typography, and iconography across all major screens.

NFR-9 – Accessibility
The app should follow common mobile accessibility guidelines, including sufficient contrast, readable fonts, and support for screen readers.

2.4.4 Security and Privacy

NFR-10 – Secure Transport
All communication between the mobile app and backend shall be encrypted using industry-standard protocols.

NFR-11 – Access Control
Users shall not be able to view or modify other users' meal logs.

NFR-12 – Data Minimization
The system shall avoid storing unnecessary personal information and shall not store sensitive identifiers.

NFR-13 – Regulatory Alignment
Data handling shall be designed to be compatible with relevant privacy regulations (e.g., PIPEDA).

2.4.5 Scalability and Extensibility

NFR-14 – Cloud-Native Design
The backend shall follow a cloud-native architecture that can be scaled up by adding more instances without redesign.

NFR-15 – Modularization
The system shall separate concerns for image storage, ML processing, and user data storage in distinct components or services.

2.4.6 Maintainability

NFR-16 – Code Organization
The codebase shall be organized into separate modules for mobile front-end, backend services, and ML processing, with clear API contracts.

NFR-17 – Documentation
Key APIs and data models shall be documented so that a new developer can onboard to the project and implement a small change within 2 working days.

2.4.7 Portability

NFR-18 – Mobile Portability
The app shall be implemented using a cross-platform framework (e.g., Flutter) or equivalent approaches that minimize duplication between Android and iOS implementations.

NFR-19 – Cloud Portability (Best Effort)
While AWS is the mandated infrastructure, the system should avoid unnecessary tight coupling to proprietary services where feasible.

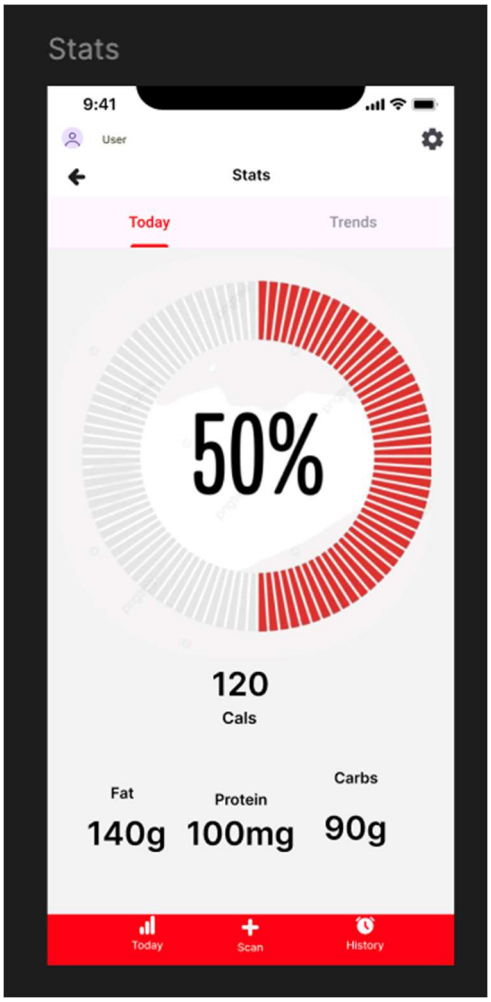
2.5 External and Internal Interfaces

2.5.1 External Interfaces

2.5.1.1 User Interface (Mobile App)

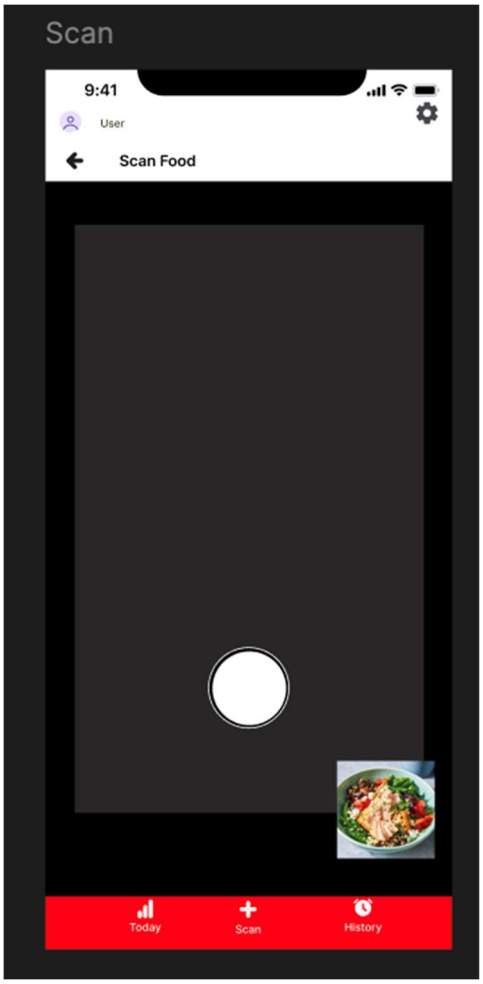
UI-1 – Home Screen

- Displays quick access actions: “Log Meal”, “Today’s Summary”, “History”, and “Profile”.



UI-2 – Camera / Capture Screen

- Provides a live camera viewfinder, standard camera controls, and preview of last captured image.

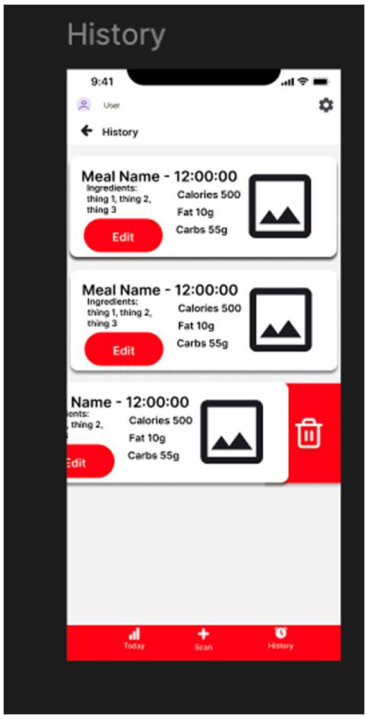


UI-3 – Results Screen

- Shows the captured image thumbnail and the detected ingredients with estimated portion and calories.

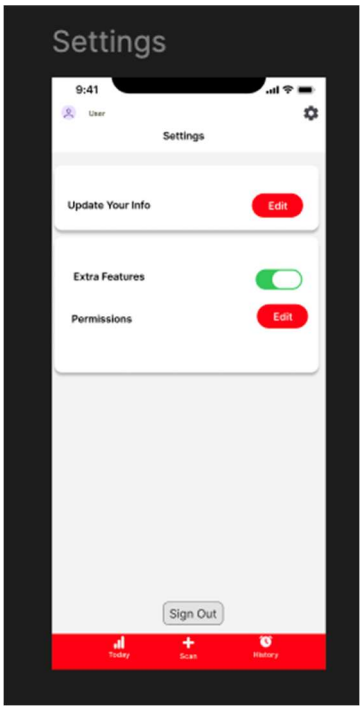
UI-5 – History Screen

- Lists meals by date and allows navigation to meal details.



UI-6 – Settings Screen

- Displays and allows editing of basic user information and optional dietary preferences.



2.5.1.2 Hardware Interfaces

- Mobile Device Camera: The app shall use the platform's native camera APIs to capture images.
- Device Storage: Used for temporary caching of images prior to upload and for offline queued uploads.

2.5.1.3 Software and Communication Interfaces

Backend API (HTTPS)

- POST /api/v1/meals – Upload a new meal photo and metadata; returns analysis results.
- GET /api/v1/meals?date=YYYY-MM-DD – Fetch meals for a given date.
- GET /api/v1/meals/{mealId} – Fetch detailed info about a specific meal.

Communication uses JSON payloads for requests and responses.

AWS Services

- S3 – Storage of original and processed images.
- Lambda / Container Service – Running the processing pipeline.
- Database (e.g., DynamoDB/RDS) – Storing meal metadata and user data.
- Cognito or Equivalent (Future) – Authentication and user management.

2.5.2 Internal Interfaces

Mobile App ↔ API Gateway

- Sends photos and receives structured results over HTTPS.

API Gateway ↔ Processing Service

- Forwards requests to a processing service, passing image references and metadata; receives processed ingredient and calorie data.

Processing Service ↔ Storage Layer

- Reads and writes images from/to S3; retrieves and persists meal records in the database.

Processing Service ↔ ML Models (Future)

- Invokes ML models for ingredient segmentation and classification through an internal API or library interface.

Storage Layer ↔ Analytics/Training Pipelines (Future)

- Provides data export or streaming interface for incremental training systems to retrieve labeled food images and metadata.