# Food Logging & Nutrition Intelligence System — Design Document

## 1. System Overview

The system provides automated meal logging through food image recognition, volume estimation, and nutritional inference. Users can capture or upload food images, review predicted ingredients and nutrition information, and log meals with optional corrections. The platform integrates feedback loops for model retraining and continuous improvement.

### 1.1 User Flow

1. **Authentication**
    a. User signs in via AWS Cognito (supports Google and Apple SSO).
2. **Image Input**
    a. Authenticated user captures a food photo or uploads an existing image.
3. **Image Upload**
    a. Image is uploaded to Amazon S3 using a pre-signed URL generated by the backend.
4. **Inference Request**
    a. Mobile app calls the API to trigger ML inference on AWS SageMaker.
    b. Models perform:
        i. Multi-label food recognition
        ii. Segmentation and volume estimation
        iii. Nutrition mapping
5. **Prediction Review**
    a. Backend returns predicted ingredients, portion size, calories/macros, and confidence scores.
    b. User edits or confirms the prediction via the preview UI.
6. **Meal Logging**
    a. The finalized meal record is stored in the database.
    b. User feedback (corrections and confidence issues) is automatically captured.
7. **User Insights**
    a. App displays basic calorie trends (e.g., daily calories, missed days).

# 2. System Architecture

## 2.1 Mobile Application (Android-first, Flutter)

**Primary Responsibilities**

- Image capture and file upload
- Upload preview and manual nutrition entry
- Editing and deleting logged meals
- Displaying calorie trend charts
- Managing user authentication with AWS Cognito

**Key Features**

- Android-first deployment, cross-platform portability via Flutter
- Local caching for fast UI interactions (no requirement for offline inference)
- Consistent design for future iOS rollout

## 2.2 API Layer (AWS API Gateway + AWS Lambda)

**Functions**

- Serves REST endpoints for:
    - Image inference requests
    - Meal CRUD operations
    - User profile and settings
    - Feedback collection
- Generates pre-signed S3 upload URLs
- Routes inference requests to SageMaker or ECS-hosted models

**Notes**

- Stateless Lambda functions ensure scalability
- API Gateway enables rate limiting, usage plans, and authentication enforcement

## 2.3 ML Inference (AWS SageMaker or ECS)

**Components**

- **Food Recognition Model**
  - Multi-label classifier
  - Optional segmentation model
- **Volume Estimation Module**
  - Uses segmentation masks + heuristic or depth estimation
- **Nutrition Engine**
  - Maps each ingredient to calories/macros using USDA/Canadian datasets

**Outputs**

- Detected ingredients
- Estimated portion size and volume
- Total calories and macro breakdown
- Confidence metrics

## 2.4 Data Storage Layer

| Component | Purpose |
|---|---|
| **Amazon S3** | Stores raw food images, thumbnails, and user feedback datasets |
| **DynamoDB** | Stores user profiles, meal records, prediction metadata, feedback, and food database entries |
| **Food Database** | USDA + Canadian nutrition datasets mapped to model labels |

## 2.5 Feedback & Retraining Pipeline

**Flow**

- Predictions with low confidence or user corrections trigger feedback entries
- Feedback data stored in S3 and DynamoDB

- AWS Step Functions orchestrate periodic automated retraining workflows
- SageMaker Model Monitor tracks:
    - Data drift
    - Accuracy degradation
    - Missing classes or underrepresented foods

**Outputs**

- Updated weights deployed to SageMaker endpoints
- Alerts if performance drops below thresholds

# 3. Data Flow (Simplified)

1. **Capture Image**
    a. User takes or selects an image.
2. **Upload → S3**
    a. App receives pre-signed URL from backend and uploads the file.
3. **Inference Request**
    a. App triggers backend to call the ML model.
4. **Prediction Returned**
    a. SageMaker returns ingredient predictions, volume estimation, calories/macros.
5. **User Review**
    a. User edits or confirms values.
6. **Store Meal**
    a. Final meal is persisted in DynamoDB.
7. **Feedback Storage**
    a. Corrections or low-confidence detections logged for retraining.

# 4. Key Priorities and Technical Goals

## 4.1 Recognition Accuracy

- Target: **94%+ food recognition accuracy** (top-k based for multi-label)

- Emphasis on common dishes and multi-ingredient meals

### 4.2 User Controls

- Manual entry workflows
- Editable predictions
- Ability to flag incorrect outputs

### 4.3 Volume Estimation

- Robust segmentation and portion prediction
- Prioritized for MVP due to nutritional accuracy dependence

### 4.4 System Characteristics

- Cloud-first design
- No offline inference required
- Designed for cross-platform scalability
- Clear path for expansion to iOS and Web

## 5. Source Code

https://github.com/LD-UO/SEG4105-POC