**PERSONALIZED DIET & WORKOUT PLANNER**

BY

JEREMY TAN JIM HONG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEM (HONOURS) INFORMATION SYSTEM ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

# COPYRIGHT STATEMENT



This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Systems (Honours) Information Systems Engineering at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

# ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors Dr. Chai Meei Tyng who has given me this opportunity to engage in the development of a personalized diet and workout planning system. It is an important step in building my skills in software development and health technology. A million thanks to you.

To all of my family and friends in my life, appreciate for their patience, unconditional support, and love, and for standing by my side during hard times. Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

# ABSTRACT

Flux is a personalized diet and workout planning mobile application developed to help users convert health goals into practical daily actions. The project addresses the gap between generic calorie trackers or fixed workout templates and the need for guidance that considers personal health data, dietary restrictions, workout constraints, and real progress. Flux is implemented as a full-stack system using a Flutter mobile front end, a Node.js/Express REST API, and a MySQL relational database. Through guided onboarding, the system collects profile, goal, diet, and workout preferences, then calculates BMI, BMR, TDEE, calorie targets, and macronutrient targets as the foundation for plan generation.

The system contains three core modules. The diet recommendation engine generates seven-day meal plans by filtering and scoring recipes according to calorie targets, macronutrient fit, meal type, dietary preference, allergies, and selected chronic-condition guardrails such as hypertension, hyperlipidemia, and type 2 diabetes. The workout recommendation engine generates weekly workout plans based on goal, intensity, session duration, preferred locations, and available equipment, while using MET-based calculations to estimate calorie expenditure and display goal-support feedback. The logging and progress modules record planned and custom meals, workouts, water intake, and body weight so users can monitor daily summaries, weekly adherence, weight trends, and plan freshness.

Flux also includes optional AI-assisted features for meal image scanning, recipe step generation, custom workout calorie estimation, and weekly progress insights. However, the main contribution remains the explainable rule-based recommendation logic. By combining evidence-informed calculations, transparent plan adjustments, swap impact previews, and secure account management, Flux demonstrates a complete undergraduate-level solution that fulfils the project objectives of diet recommendation, workout recommendation, and adaptive logging with progress monitoring.

Area of Study (Minimum 1, Maximum 2): Mobile Application Development, Rule-Based Recommendation System
Keywords (Minimum 5, Maximum 10): Personalized health planning, diet recommendation, workout recommendation, progress tracking, mobile health

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| kg | Kilogram, used for body weight and target weight. |
| cm | Centimetre, used for body height. |
| kcal | Kilocalorie, used for energy intake and expenditure. |
| g | Gram, used for macronutrient quantities. |
| mg | Milligram, used for sodium and selected micronutrient quantities. |
| mmHg | Millimetres of mercury, used for blood pressure thresholds. |
| % | Percentage, used for adherence, progress, and ratio-based indicators. |
| MET | Metabolic equivalent of task, used to estimate exercise energy expenditure. |
| min | Minute, used for workout duration and weekly activity totals. |

# LIST OF ABBREVIATIONS

| | |
|---|---|
| ACSM | American College of Sports Medicine |
| ADA | American Diabetes Association |
| AHA | American Heart Association |
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| BMI | Body Mass Index |
| BMR | Basal Metabolic Rate |
| ISSN | International Society of Sports Nutrition |
| JWT | JSON Web Token |
| REST | Representational State Transfer |
| SDLC | Software Development Life Cycle |
| TDEE | Total Daily Energy Expenditure |
| UI | User Interface |
| UX | User Experience |
| WHO | World Health Organization |

# Chapter 1 Introduction

## 1.1 Problem Statement and Motivation

Managing diet and physical activity is a continuous process that involves daily decisions about food intake, exercise, hydration, and body-weight progress. These decisions are important because unhealthy diets and physical inactivity are recognized risk factors for noncommunicable diseases. The World Health Organization reported that noncommunicable diseases killed at least 43 million people in 2021, while nearly one third of adults globally were physically inactive in 2022 [1], [2].

Mobile health applications can support this need by making tracking and guidance available through a device that users already carry. However, continued use is a major challenge: adherence to mHealth apps is affected by personalization, user-friendly design, reminders, and support that matches the user's context [3]. Therefore, a useful health application should not only display generic calorie values or fixed workout templates. A user's age, sex, height, weight, activity level, goal, dietary restrictions, disease conditions, workout preference, and daily logged behavior should influence the guidance provided by the system.

Flux is developed as a personalized diet and workout planning mobile application that connects profile collection, health-metric calculation, plan generation, daily logging, and progress monitoring in one workflow. The system uses a Flutter mobile application as the user interface, a Node.js/Express backend for authentication and recommendation logic, and a MySQL database for user data, recipes, exercises, generated plans, and logs. Recommendations are generated using rule-based engines so that the output can be explained in terms of calorie targets, macronutrient targets, dietary restrictions, workout constraints, and progress data.

### 1.1.1 Background Study

Personalized health planning begins with a reliable user profile. In Flux, onboarding collects personal information, health data, goal settings, diet preferences, and workout preferences before any plan is generated. The health-metrics module then calculates body mass index, basal metabolic rate, total daily energy expenditure, daily calorie target, and macronutrient targets; these are reported as BMI, BMR, and TDEE where appropriate. BMR is computed using the Mifflin-St Jeor equation, which is widely used for estimating resting energy expenditure in adults [4]. Protein targets are adjusted by goal because the International Society of Sports Nutrition states that exercising individuals commonly require higher protein intake than sedentary individuals [5].

For diet planning, the system must consider more than total calories. A suitable meal plan should also respect the user's preferred meal types, dietary preference, allergies, and selected chronic conditions. Reliable food-composition and label data are needed for this process; FoodData Central demonstrates the value of traceable nutrient data, while the Edamam Recipe API documents diet and health labels that can support filtering by nutrient and ingredient characteristics [6], [7]. For example, hypertension-aware meal planning should consider sodium and DASH-style eating patterns, diabetes-aware planning should consider fibre and carbohydrate quality, and hyperlipidaemia-aware planning should limit saturated fat exposure [8], [9], [10].

For workout planning, the system uses exercise information that includes activity category, intensity, target muscle group, difficulty level, environment, required equipment, and Metabolic Equivalent of Task value, abbreviated as MET. MET values are commonly used to estimate the energy cost of physical activities and are provided in the Compendium of Physical Activities [11]. Flux uses these values to estimate workout calories and to generate structured weekly workout plans. The generated plan is also evaluated against WHO and ACSM physical activity guidance so the user can understand whether the week supports the selected goal [12], [13].

A complete planning system also needs to connect recommendations with what the user does. Flux therefore includes logging functions for planned meals, custom meals, planned workouts, custom workouts, water intake, and body weight. Planned items become closed after

logging so that user history is not overwritten by later plan changes. When the user changes health data or preferences, the system can identify that the current plan may no longer match the latest profile. This design supports continuity between recommendation, action, and feedback.

The application also includes optional AI-assisted features, such as meal image scanning, recipe step generation, custom workout calorie estimation, and weekly progress insights. These functions support usability, but they do not replace the rule-based recommendation core. This separation is important because image-based and large-language-model dietary assessment can identify foods, but published reviews and evaluations still report limitations in portion-size and nutrient estimation accuracy [25], [26].

### 1.1.2 Problem Statement

Public health data shows that lifestyle-related health risks remain significant: noncommunicable diseases remain a major global cause of death, and physical inactivity affects a large proportion of adults worldwide [1], [2]. Although the general advice to eat appropriately and exercise regularly is widely known, users often need a more specific answer to daily questions such as what to eat, how much to eat, which workout to perform, how long to exercise, and whether their actions still support their goals.

The first problem is insufficient personalization in everyday diet and fitness guidance. Many users have different calorie requirements, body-weight goals, food preferences, allergies, chronic conditions, workout locations, and equipment availability. A generic meal plan or workout template may not match these constraints. As a result, the user may need to manually interpret and modify the plan, which increases effort and reduces the chance of consistent use. Research on mHealth adherence also shows that personalization and user-friendly design are important factors for continued engagement [3].

The second problem is weak connection between planning and progress monitoring. A plan is only useful when it remains connected to the user's actual behavior. If a user logs extra meals, misses workouts, changes weight, or updates preferences, the system should help the user understand the effect of these changes. A scoping review of recommender systems in weight-management mHealth interventions notes that user engagement remains limited and

that many studies measure engagement inconsistently, which supports the need for clearer feedback and stronger links between recommendation and user action [14].

The third problem is **limited explainability and flexibility**. Users may need to swap a meal because ingredients are unavailable, replace an exercise because equipment is missing, or adjust a plan because of time constraints. If the system does not show the impact of these changes, users cannot easily know whether the replacement still fits their calorie target, macro target, workout goal, or weekly activity balance. In health-related applications, unexplained recommendations can reduce trust because users cannot see why an item was recommended or why a substitution is suitable.

## 1.2 Objectives

1. To develop an integrated logging and progress monitoring module that records planned and custom meals, planned and custom workouts, water intake, and body weight, then presents daily summaries, weekly summaries, adherence indicators to help users keep track on their goals progress.
2. Develop a rule-based diet recommendation system that generates personalized meal plans from calorie targets and health goals while honoring allergies and disease-specific constraints, plotting the health and diet label of food, and use reputable databases to ensure correctness.
3. Develop a rule-based workout recommendation system that recommends weekly workout plan by goal, preference, and intensity, supports flexible exercise modification, and maps weekly loads to WHO and ACSM recommendations [12], [33] using Compendium MET values [11] to keep energy budgets consistent.

## 1.3 Project Scope

The scope of this project is to design and implement Flux as a cross-platform mobile application for adult users who want structured support for diet planning, workout planning, daily logging, and progress monitoring. The system focuses on personal wellness planning rather than clinical diagnosis or medical treatment. It provides evidence-informed calculations and planning rules, but it does not replace advice from doctors, dietitians, or certified fitness professionals.

The frontend scope includes the Flutter mobile application screens and state management needed for user authentication, email verification, password reset, Google sign-in, onboarding, metrics display, home dashboard, diet plan viewing, workout plan viewing, recipe and exercise search, swap flows, impact preview, daily logging, progress charts, and profile editing. The mobile application presents data in a user-friendly form and communicates with the backend through REST API calls.

The backend scope includes a Node.js/Express REST API that manages authentication, user onboarding, health-metrics calculation, diet recommendation, workout recommendation, logging, progress summaries, and optional AI-assisted features. The backend is organized using routes, controllers, services, and repositories. Business rules are placed in service modules so that recommendation logic is separated from request handling and database queries.

The database scope includes a MySQL relational database that stores user accounts, refresh tokens, verification records, password reset records, user profiles, diet preferences, workout preferences, computed metrics, recipe records, exercise records, generated diet plans, generated workout plans, meal logs, workout logs, water logs, weight logs, saved custom meals, saved custom workouts, and goal-completion records. Foreign-key relationships are used to maintain data consistency between users, plans, logs, and reference datasets.

The diet recommendation scope covers seven-day meal plan generation, meal-slot distribution, recipe filtering, calorie and macronutrient scoring, portion scaling, dietary-preference handling, allergy exclusion, disease-related guardrails, recipe health tags, recipe search, swap recommendations, impact preview, and day regeneration. The workout recommendation scope covers weekly plan generation, session distribution, exercise selection, structured prescriptions, MET-based calorie estimation, goal-support assessment, exercise search, swap recommendations, and impact preview.

The logging and progress scope covers planned and custom meal logging, planned and custom workout logging, water tracking, body-weight tracking, daily summary, weekly nutrition summary, diet adherence, workout adherence, calorie history, macro history, workout frequency, weight trend, and goal timeline. Optional AI-assisted features are included only as

supportive functions for meal image scanning, recipe step generation, custom workout calorie estimation, and weekly progress insights.

## 1.4 Contributions

Flux contributes an integrated approach to personal diet and workout planning. Instead of treating diet planning, exercise planning, logging, and progress monitoring as separate functions, the system connects these activities through the user's profile, targets, generated plans, and daily logs. This helps users understand how planned recommendations and actual behaviour relate to their selected goal.

The diet component contributes a rule-based meal planning process that considers calories, macronutrients, meal types, dietary preference, allergies, and selected chronic conditions. Recipes are filtered and scored before being assembled into a seven-day plan. The system also provides recipe search, meal swap options, day regeneration, and impact previews so that users can make changes without losing sight of daily nutrition targets.

The workout component contributes a structured weekly training planner that uses goal, intensity, workout frequency, session duration, location, and equipment availability to select suitable exercises. The system calculates estimated calorie expenditure using MET values and provides structured prescriptions for the generated workout sessions. Swap and search functions allow users to replace exercises while still reviewing the effect on the overall workout plan.

The logging and progress component contributes continuity after the plan has been generated. Users can record planned and custom meals, planned and custom workouts, water intake, and body weight. These logs are used to present daily and weekly summaries, adherence measures, and progress trends. Logged planned items are protected from later regeneration or swapping, which preserves the accuracy of user history.

The implementation also contributes a maintainable full-stack structure. The Flutter frontend focuses on user interaction and presentation, the Node.js/Express backend handles authentication and business rules, and the MySQL database stores profiles, plans, logs, and reference datasets. Security-related functions such as password hashing, email verification,

refresh tokens, Google account linking, and protected API routes support safer handling of user accounts and health-related data.

## 1.5 Report Organization

This report is organized into seven chapters. Chapter 2 reviews related technologies, recommendation approaches, and existing diet and fitness applications. Chapter 3 explains the development methodology and system approach used for Flux. Chapter 4 presents the system design artefacts, including architecture, use case, data, sequence, and flow diagrams. Chapter 5 documents the system implementation, setup, configuration, and module operation. Chapter 6 evaluates the system through testing results, project challenges, and objective achievement. Chapter 7 concludes the project and provides recommendations for future improvement.

# Chapter 2 Literature Review

## 2.1 Review of the Technologies

This section reviews the technologies that directly support Flux as a development-based mobile application, the target hardware platform, mobile operating environment, database layer, programming languages and frameworks, and the recommendation algorithms used for diet, workout, logging, and progress feedback.

Flux is not a research prototype that only discusses formulas. It is a working full-stack system. Therefore, the technology review connects published health guidance and recommender-system literature with practical implementation choices such as Flutter mobile development, a Node.js/Express backend, MySQL relational storage, rule-based filtering, MET-based workout estimation, and explainable health labels.

### 2.1.1 Hardware Platform

The primary hardware platform for Flux is a smartphone. A mobile device is suitable because diet and exercise decisions happen throughout the day, not only at a desktop computer. The phone gives the user immediate access to onboarding, meal plans, workout plans, logging, and progress summaries at the time the action occurs. This supports the mHealth adherence literature, which emphasizes the importance of accessible, user-friendly, and personally relevant app interactions [3].

The development and verification hardware also includes a laptop or desktop workstation capable of running the Flutter toolchain, Android simulator, Node.js backend server, and MySQL database. This setup allows the system to be tested end to end: the mobile interface sends requests to the backend, the backend executes business rules, and the database stores users, plans, logs, recipes, exercises, and computed progress data.

Because Flux depends on daily user behaviour, the hardware platform must support frequent interaction rather than occasional batch entry. Planned meal completion, custom meal logging, water tracking, workout completion, and body-weight entry are all diary-like tasks. A mobile-first platform therefore matches the practical workflow of the application better than a desktop-only system.

### 2.1.2 Firmware/OS

Flux is implemented as a cross-platform mobile application using Flutter, which is Google's UI toolkit for building applications from a single codebase [15]. This is suitable for an FYP development project because the same Dart codebase can target Android and iOS-style mobile experiences while keeping the project maintainable for a beginner developer.

The app is verified mainly through an Android simulator during development, while the architecture remains portable because Flutter abstracts much of the platform-specific rendering and interaction layer. The operating-system layer is still relevant because camera access, secure token storage, network connectivity, and screen navigation must behave consistently on the target mobile environment.

Using a mobile OS environment also supports real-world workflows such as capturing a meal photo, checking the current day's plan, logging a workout immediately after completion, and reviewing progress while away from the development workstation. These workflows would be weaker if the application were designed only for a browser or desktop platform.

### 2.1.3 Database

Flux uses MySQL as the relational database layer. MySQL is appropriate for this project because the system contains strongly related data: users, profiles, preferences, computed metrics, recipe records, workout records, generated plan days, plan items, swap history, and daily logs. The official MySQL 8.0 reference manual documents the database system and its SQL features, which support structured relational storage for these connected records [19].

The database is also important for nutrition and exercise traceability. FoodData Central is used as an evidence point for reliable food-composition data, while Edamam's diet and health labels show how recipe metadata can support filtering by macronutrient level, allergens, and dietary pattern [6], [7]. Flux stores cleaned recipe and workout datasets so recommendation results can be generated consistently rather than relying on manual user interpretation.

A relational schema is especially useful for preserving the link between generated plans and user actions. When a planned meal or workout is logged, the log can remain connected to the original plan item. This supports adherence calculations, progress summaries, and regeneration rules that preserve completed items instead of overwriting user history.

### 2.1.4 Programming Language

The frontend uses Dart with Flutter. Dart is described by its official documentation as a language designed for multi-platform development and as the language that powers Flutter apps [16]. This makes Dart suitable for building Flux's guided onboarding screens, bottom-tab navigation, plan views, logging forms, progress charts, and profile-management flows within a single mobile codebase.

The backend uses JavaScript with Node.js and Express. Node.js is a JavaScript runtime built for server-side applications, while Express is a minimal and flexible Node.js web application framework for web and mobile applications [17], [18]. This stack is suitable for a REST API that handles authentication, onboarding, recommendation generation, logging, progress aggregation, and AI-assisted endpoints.

The separation between Dart on the frontend and JavaScript on the backend keeps the implementation understandable. Flutter screens remain focused on user interaction, while backend services contain formulas, scoring rules, filtering rules, and workflow decisions. This separation also makes the project easier to explain during FYP evaluation because each layer has a clear responsibility.

The chosen languages and frameworks are therefore practical rather than experimental. They support a maintainable full-stack implementation while still allowing evidence-based health rules and recommendation logic to be centralized on the backend.

### 2.1.5 Algorithm

The core personalization algorithms in Flux are rule-based and evidence-informed. Health metrics begin with profile-based calculations, including BMR estimation through the Mifflin-St Jeor equation and goal-aware protein targets informed by ISSN protein guidance [4], [5]. These values provide the numerical basis for calorie targets, macronutrient targets, and progress comparisons.

The diet recommendation algorithm filters recipes by dietary preference, allergies, health conditions, meal type, and nutrient constraints before ranking candidates. This design is supported by food recommender-system literature, which notes that food recommendation is complex because it must consider nutrition, preference, ingredients, restrictions, and evaluation criteria rather than only item popularity [20], [21]. FDA label-claim guidance and Edamam's diet and health label definitions also support the use of computed labels such as low sodium, high protein, high fibre, and allergen-free badges [7], [22].

The workout recommendation algorithm selects exercises by goal, intensity, duration, location, equipment, and target muscle group. Energy expenditure is estimated using MET values from the Compendium of Physical Activities, while plan quality is compared against WHO and ACSM physical activity guidance [11], [12], [13]. The updated 2024 Adult Compendium confirms that MET-based activity classification remains a current method for comparing the energy costs of many activity categories [23].

### 2.1.6 Summary of the Technologies Review

The reviewed technologies support Flux as an integrated mobile health application rather than a simple calorie counter. Smartphone hardware supports daily interaction, Flutter and Dart support the mobile frontend, Node.js and Express support the REST API backend, and MySQL stores the persistent profile, plan, recipe, workout, and logging data needed for traceable recommendations.

The algorithm review shows that Flux should remain explainable. Rule-based filtering is appropriate because users may not have enough historical data for machine-learning personalization, but the system still needs to respect health constraints, preferences, and practical workout conditions from the first plan. Logging and progress feedback remain important because self-monitoring of diet, exercise, and weight is a recognized behaviour-change component, although it must be made easy enough for users to sustain [24].

Overall, the technology choices align with the FYP objectives: personalized diet planning, personalized workout planning, logging, progress monitoring, and adaptive feedback can be implemented in a maintainable full-stack architecture while keeping the recommendation logic evidence-informed and defensible.

## 2.2 Review of the Existing Systems/Applications

This section reviews existing diet and fitness applications that are relevant to Flux. The comparison focuses on five systems: MyFitnessPal, Noom, Fitbit, WeightWatchers, and Lose It!. These applications were selected because they represent different approaches to diet tracking, behaviour coaching, wearable-centered fitness monitoring, structured weight-management programmes, and lightweight calorie-budget tracking.

The review does not treat these systems as direct copies for Flux. Instead, each system is used to identify useful features and limitations. The evaluation dimensions are recommendation approach, health or diet condition support, personalization and tracking, meal planning, and workout guidance.

Table 2.1 summarizes the comparison. The discussion after the table explains how the strengths and gaps of each system inform the design of Flux.

The comparison shows that commercial applications often specialize in one area. Some are strong trackers, some are stronger coaching systems, some are hardware-centered fitness monitors, and some are lightweight calorie-budget tools. Flux aims to combine the most relevant ideas into a development-based FYP system: evidence-based targets, generated plans, planned and unplanned logging, transparent swap impact, and progress feedback from stored logs.

| Application | Recommendation Approach | Health/Diet Condition Support | Personalization & Tracking | Meal Planning/Generation | Workout Guidance |
|---|---|---|---|---|---|
| MyFitness Pal | User-driven food logging with newer Premium Meal Planner filtering based on goals, preferences, cooking time, allergies, restrictions, and dislikes [27]. | Supports manual goal adjustment and selected meal-plan restrictions, but disease-specific planning is not the main workflow. | Strong calorie and nutrient tracking; personalization depends mainly on user-entered goals and logged intake. | Meal Planner can generate plans from user answers, but workout recommendation is outside its main scope [27]. | No full personalized workout-plan generator; exercise data is mainly logged or synced. |
| Noom | Behaviour-change and coaching approach using lessons, food logging, and a colour system based on calorie density and nutritional value [28]. | General weight-management support; condition-specific programmes exist separately, but detailed allergy or disease filtering is not the standard | Personalized calorie budget, food colour feedback, weight tracking, and coaching-style prompts. | Provides recipes and food guidance, but does not function mainly as an automatic seven-day meal-plan generator. | Encourages activity and step goals, but structured workout generation is limited. |

| | | | | | |
|---|---|---|---|---|---|
| | | food-planning workflow. | | | |
| Fitbit | Wearable-centered monitoring with Premium daily workout picks and recommendations based on recovery, preferences, activity type, intensity, equipment, and duration [29]. | Strong activity and recovery monitoring; diet-specific chronic-condition planning is limited. | Highly personalized fitness tracking through device and app data, including insights and readiness-style feedback. | Food logging exists, but detailed disease-aware meal-plan generation is not the main strength. | Strong guided workout content and personalized workout recommendations through Fitbit Premium [29]. |
| WeightWatchers (WW) | Points-based weight-management system that converts nutritional quality into a user-facing points budget. | Provides a diabetes-tailored programme, blood sugar tracking, and diabetes-specific nutrition guidance [30]. | Personalized points budget, ZeroPoint foods, progress tracking, workshops, and coaching/community support. | Strong recipe and meal guidance within the points system. | Activity can be tracked, but the app is primarily diet and weight-management focused rather than a full workout-plan generator. |
| Lose It! | Lightweight calorie-budget and nutrition-tracking approach with app-based food and exercise logging [31]. | Supports custom goals and advanced tracking fields, but disease-aware generated plans are not the core | Personalized calorie budget, weight goal tracking, food logging, exercise tracking, and progress tracking. | Supports meal and nutrition tracking, but does not provide the same disease-aware seven-day generated meal-plan workflow as Flux. | Tracks exercise calories but does not function as a full personalized workout-plan generator. |

| | | | | | |
|---|---|---|---|---|---|
| | | workflow [31]. | | | |
| Flux (Proposed) | Rule-based diet and workout recommendation system using health profile, goals, preferences, dietary restrictions, disease guardrails, recipe nutrition data, and workout MET values. | Supports dietary preferences, allergies, and selected disease-aware guardrails such as hypertension, hyperlipidaemia, and type 2 diabetes through rule-based filtering and nutrition targets. | Personalized calorie and macronutrient targets, generated weekly diet and workout plans, daily meal/exercise/water/weight logs, and progress monitoring. | Generates seven-day meal plans from the filtered recipe pool with per-recipe nutrition details and meal-swap support. | Generates weekly workout plans by goal, duration, intensity, location, and equipment, with exercise swaps and calorie estimates. |

*Table 2.1: Comparison of Features in Selected Diet and Fitness Applications*

### 2.2.1 MyFitnessPal

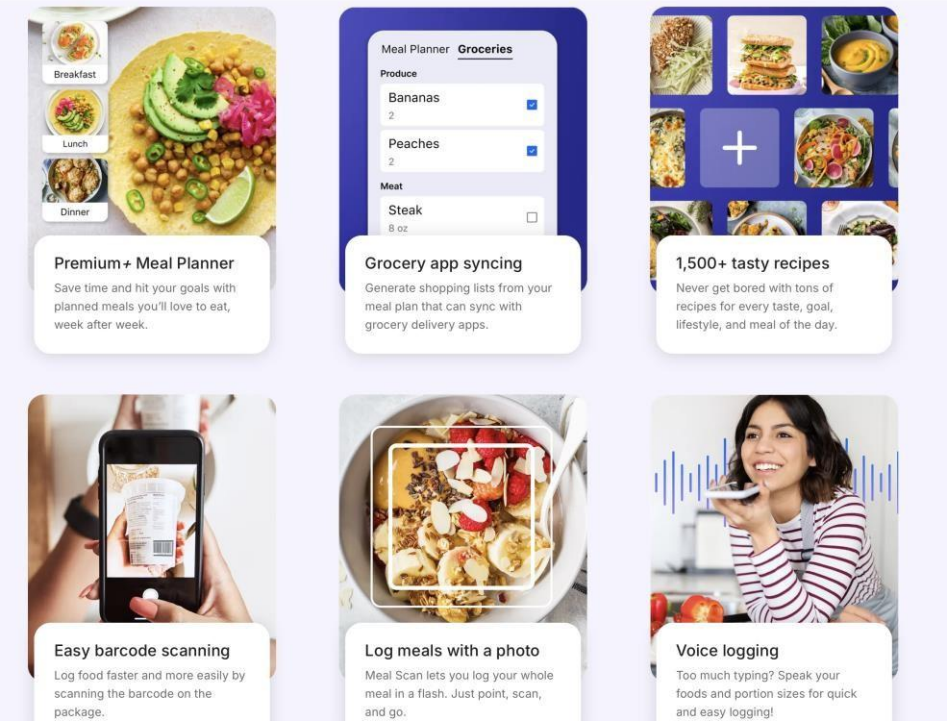


*Figure 2.1: MyFitnessPal*

MyFitnessPal is strongest as a nutrition and calorie-tracking application. Its large food database and Premium Meal Planner help users create plans based on goals, diet and cuisine preferences, cooking time, allergies, restrictions, and dislikes [27]. However, the system is still mainly user-driven: users log food, review totals, and decide how to adjust their behavior. Compared with Flux, its limitation is that diet planning, workout planning, disease-aware filtering, and generated-plan adherence are not combined into one rule-based workflow.

### 2.2.2 Noom

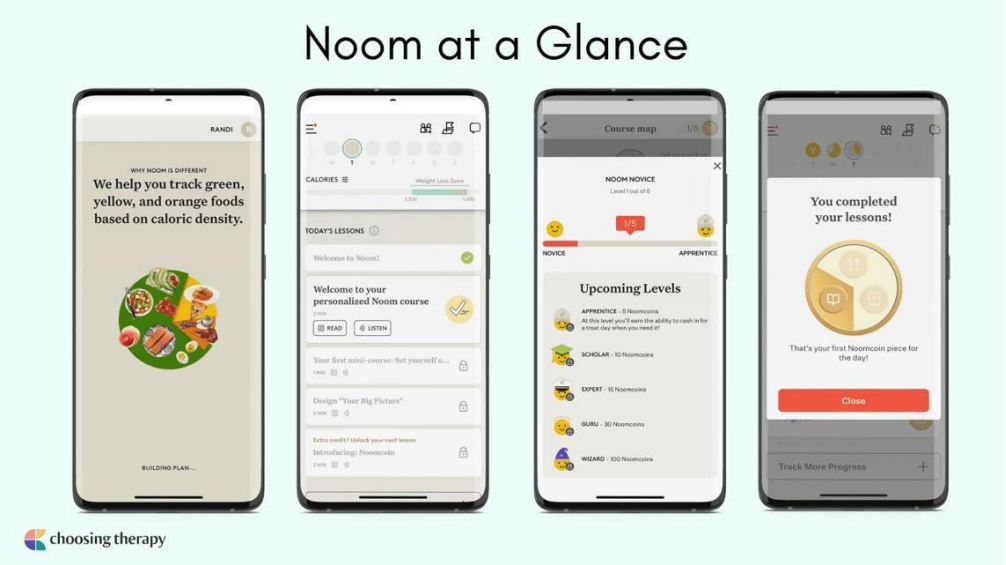


*Figure 2.2: Noom*

Noom focuses on behavior change rather than full automatic plan generation. Its food colour system classifies foods as green, yellow, or orange based on calorie density and nutritional value, helping users make simpler food choices [28]. This supports user education, but the standard workflow does not provide the same level of recipe-level disease filtering, workout-plan generation, or transparent swap impact preview required for Flux.

### 2.2.3 Fitbit

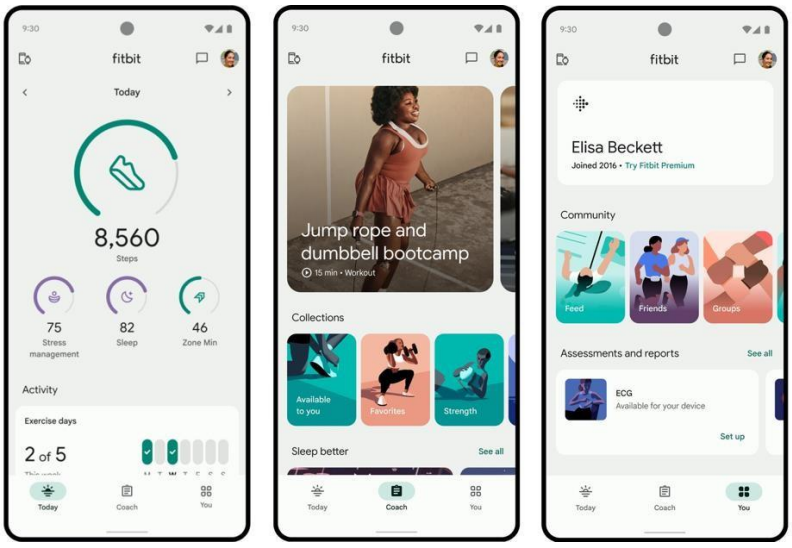


*Figure 2.3: Fitbit*

Fitbit is strongest when paired with wearable hardware. Fitbit Premium provides personalized workout content and daily picks based on recovery, user preferences, workout type, intensity,

equipment availability, and duration [29]. This makes Fitbit strong for activity tracking and guided workout content, but it is less focused on disease-aware recipe selection and seven-day meal-plan generation. Flux therefore adopts the idea of progress-aware feedback while keeping diet and workout planning inside one application flow.

### 2.2.4 WeightWatchers

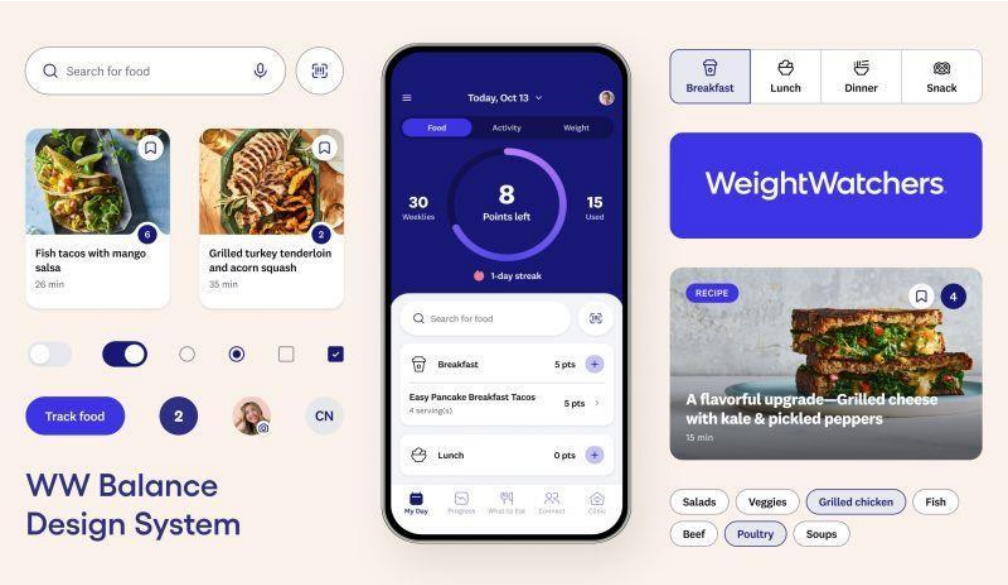


*Figure 2.4: WeightWatchers (WW)*

WeightWatchers uses a Points system that simplifies nutritional quality into a single user-facing budget. Its diabetes programme provides a tailored app experience with blood sugar tracking, diabetes-specific nutrition guidance, dietitian visits, and diabetes-focused workshops [30]. This shows strong support for condition-aware weight management. However, WW is centered on its proprietary points heuristic and coaching ecosystem, while Flux implements transparent rule-based meal and workout generation that can be explained through formulas, thresholds, and stored plan/log data.

### 2.2.5 Lose It!

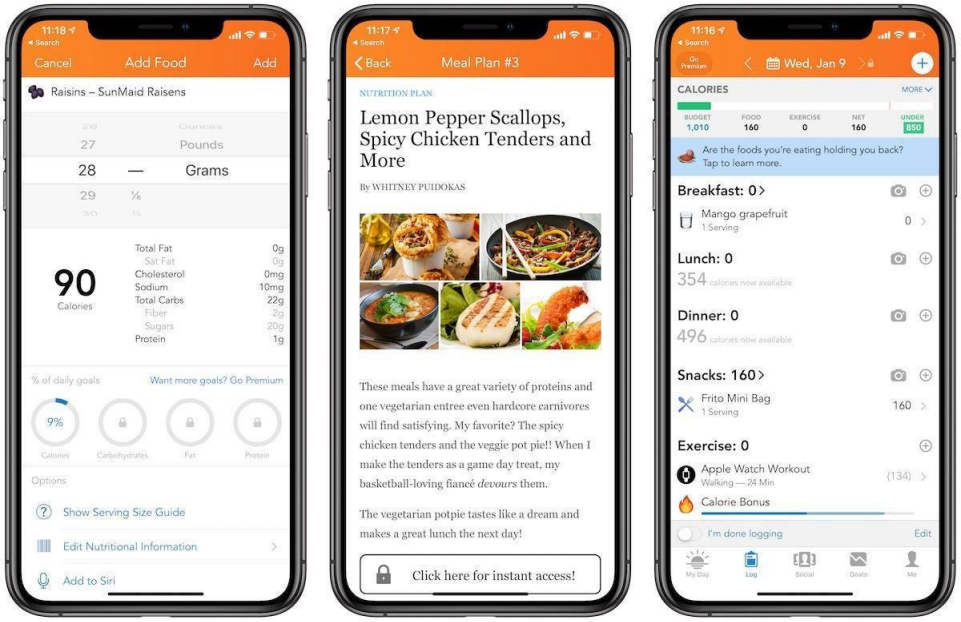


*Figure 2.5: Lose It!*

Lose It! is a lightweight calorie counter, nutrition tracker, and weight-loss progress application. Its official app listing describes calorie tracking, exercise tracking, macro and nutrition tracking, and health metrics such as blood pressure, glucose, cholesterol, body measurements, and sleep cycles [31]. Lose It! is useful as a simple self-management tool, but it does not provide the same integrated workflow that Flux targets: evidence-based health metrics, disease-aware recipe filtering, workout-plan generation, planned and custom diary logging, and transparent plan-adherence feedback.

### 2.2.6 Summary of the Existing Systems

The review of existing applications shows that no single application fully covers the target scope of Flux. MyFitnessPal is strong for calorie tracking and meal-plan support, Noom is strong for behavior education, Fitbit is strong for wearable-centered activity feedback, WeightWatchers is strong for structured weight-management support and Lose It! is strong for lightweight calorie-budget tracking. However, these systems usually emphasize one main strength rather than combining evidence-based diet planning, workout planning, planned and custom logging, progress monitoring, disease-aware guardrails, and transparent swap impact in one beginner-friendly development project.

This gap supports the design direction of Flux. The proposed application should keep the accessibility and daily logging convenience of commercial apps but add a clearer rule-based recommendation structure. Users should be able to understand why a meal or exercise is recommended, how it affects their daily targets, and whether their logged actions still support their objective.

Compared with commercial systems, Flux also emphasizes academic traceability. Health metrics, MET estimates, disease-aware nutrient checks, and protein targets are linked to named sources, while interface labels and progress indicators are derived from stored data. This makes the report easier to defend during FYP evaluation because the design is based on explainable system behaviour rather than broad claims about personalization.

In conclusion, Chapter 2 shows that the project should not simply duplicate an existing tracker. The reviewed technologies and applications justify a system that integrates profile-based targets, recipe and workout datasets, rule-based recommendation, logging, progress monitoring, and transparent feedback into one structured mobile application.

# Chapter 3 System Methodology/Approach

## 3.1 System Design Diagram/Equation

This section presents the main system design diagrams for Flux. The diagrams explain the overall architecture, how users interact with the system, and the major activity flows for key functions.

### 3.1.1 System Architecture Diagram

Flux follows a three-tier client-server architecture. The Flutter mobile application handles the user interface and local interaction flow, the Node.js/Express backend exposes protected REST APIs and contains the business logic, and the MySQL database stores user, diet, workout, logging, and progress data. External services support AI meal scanning, Google OAuth, and email OTP delivery.

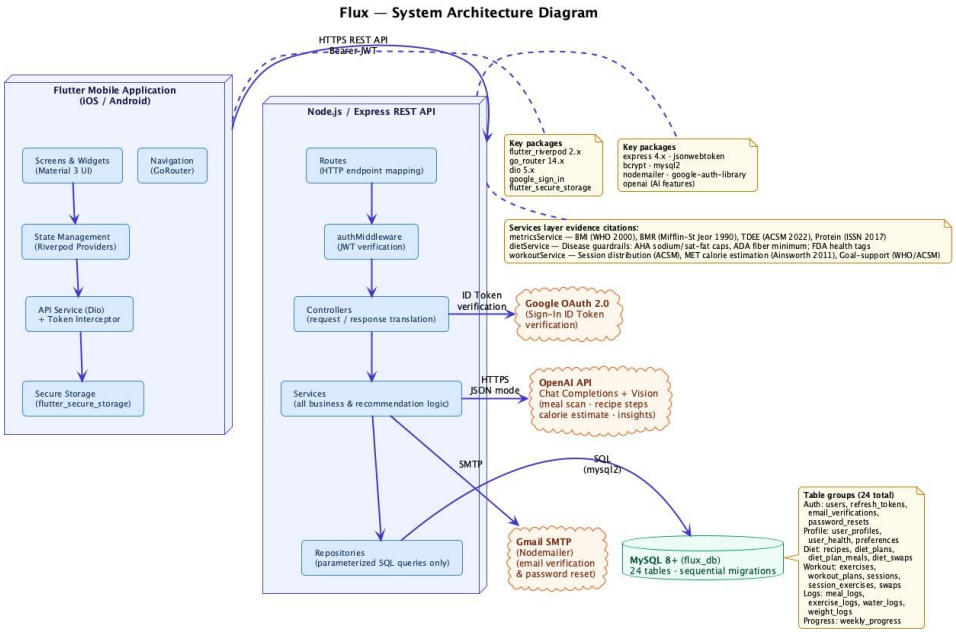


*Figure 3.1: Flux System Architecture Overview*

### 3.1.2 Use Case Diagram and Description

The use case diagram defines the functional scope of the system from the perspective of users and supporting external services. It shows the difference between guest functions, authenticated user functions, and external service integrations.

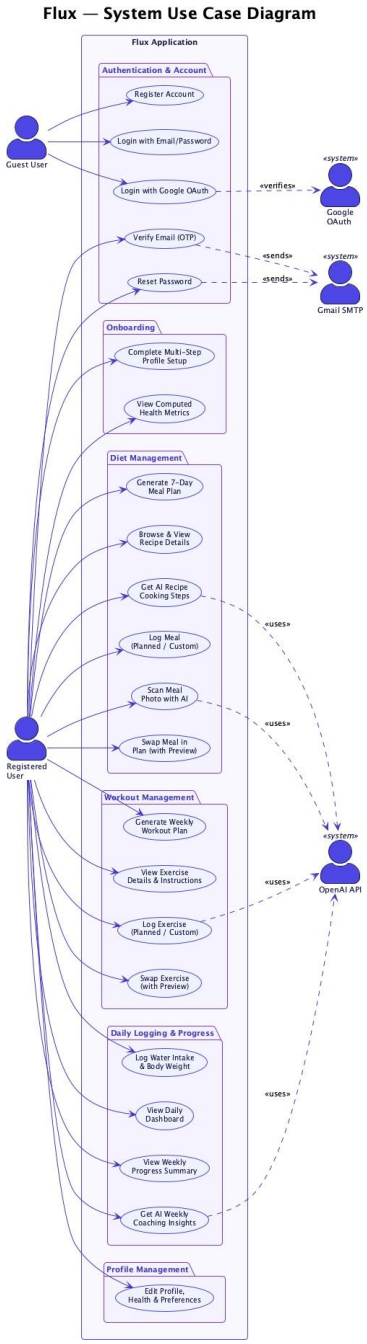


*Figure 3.2: Flux System Use Case Diagram*

The Use Case Diagram in Figure 3.2 provides a high-level overview of the Flux system from the perspective of its actors and their interactions with the system's functional capabilities. Two primary human actors interact with the system: the Guest User (unauthenticated) who can

register and log in, and the Registered User (authenticated) who has access to all application features including plan generation, daily logging, progress tracking, and AI-assisted meal scanning.

Three external system actors are also depicted: the OpenAI API (which provides AI-powered nutrition analysis, calorie estimation, and weekly coaching insight generation), Google OAuth (which verifies Google Sign-In identity tokens), and Gmail SMTP (which handles OTP email delivery for email verification and password reset). The use cases are logically grouped into six modules: Authentication & Account, Onboarding, Diet Management, Workout Management, Daily Logging & Progress, and Profile Management. Dashed dependency arrows indicate use cases that invoke external system actors.

### 3.1.3 Activity Diagram

Figure 3.3 shows the activity flow for user registration and email verification. The user enters credentials, which are validated client-side before being sent to the backend. The backend checks for duplicate emails, hashes the password, creates the user record, and sends a 6-digit OTP via Gmail SMTP. The user verifies the OTP (15-minute expiry) to activate the account and proceed to onboarding.

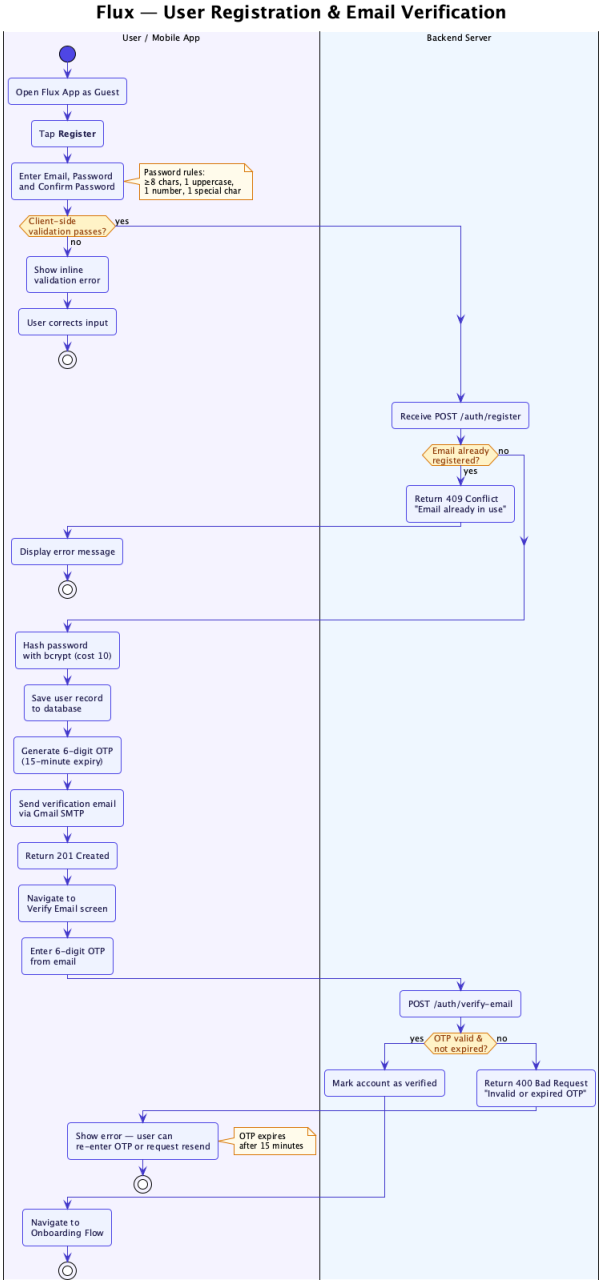


*Figure 3.3: Activity Diagram — User Registration and Email Verification*

Figure 3.4 shows the login activity for both email/password and Google OAuth paths. On successful authentication via either method, JWT access and refresh tokens are issued and stored securely on the device.

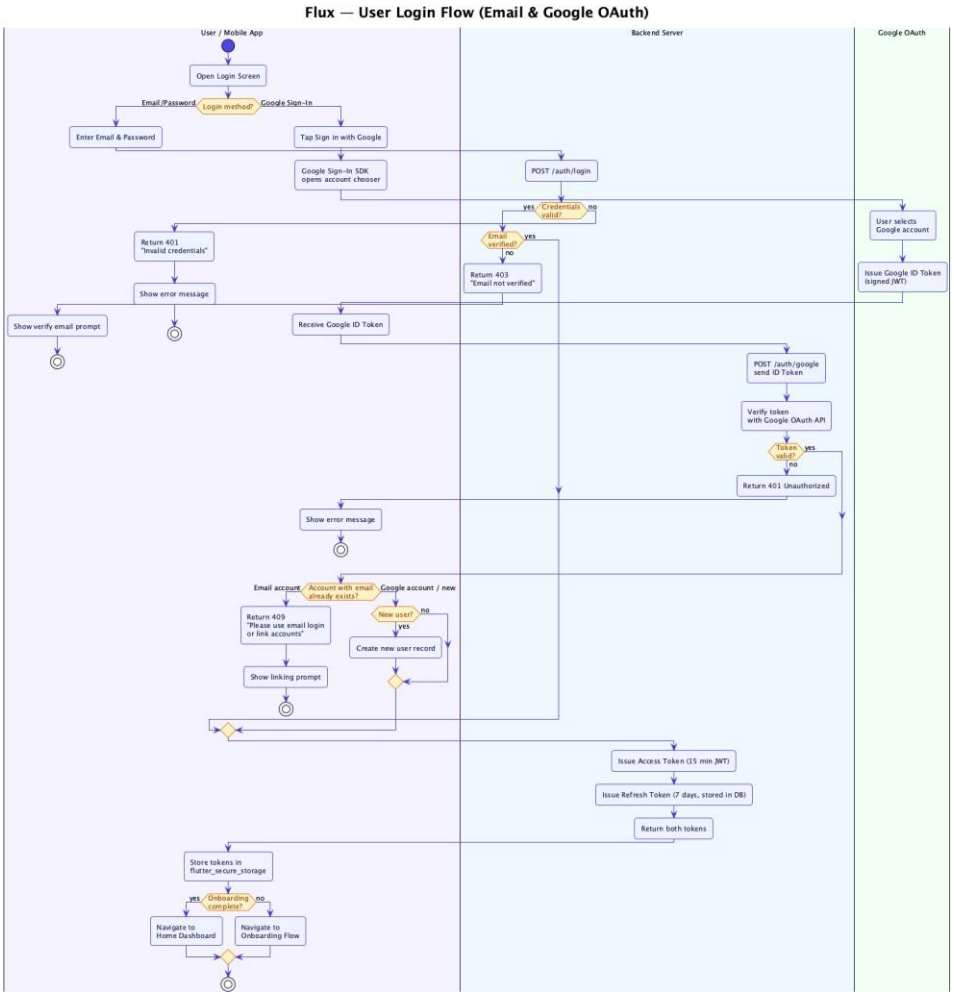


*Figure 3.4: Activity Diagram — User Login (Email / Password and Google OAuth)*

Figure 3.5 shows the nine-step onboarding flow, each screen collecting a distinct category of user data (personal details, body measurements, activity level, fitness goal, health conditions, dietary preferences, food allergies, and meal frequency). After all steps are submitted, the backend computes BMI, BMR, TDEE, goal-adjusted calorie target, and macronutrient targets in parallel before directing the user to the home dashboard.

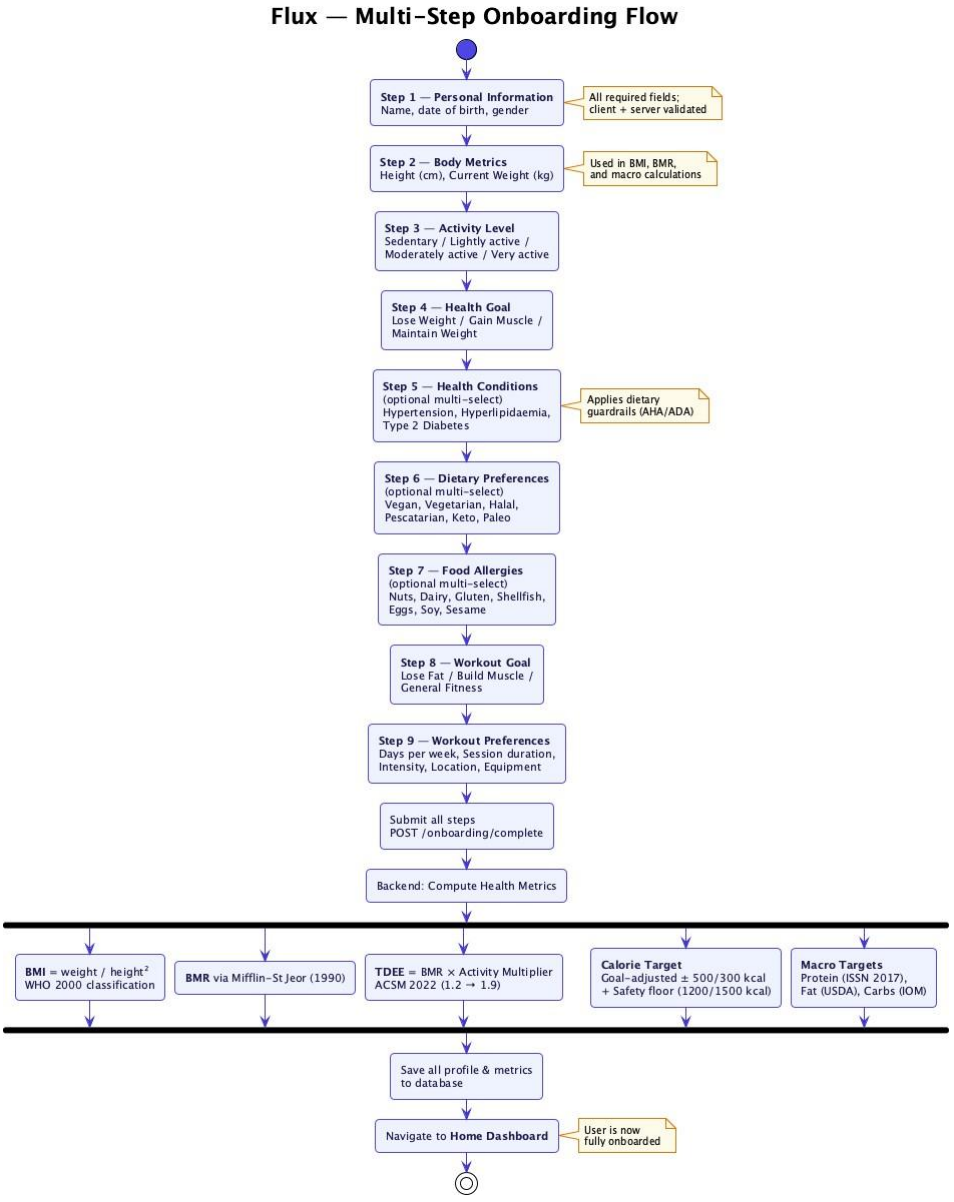


*Figure 3.5: Activity Diagram — Multi-Step User Onboarding*

Figure 3.6 shows the three-path meal logging activity: planned meal entry (from the active diet plan), custom manual entry, and AI-assisted meal scan (via OpenAI vision). All three paths converge at POST /log/meal to persist the entry to the meal_logs table and update daily nutrition totals.

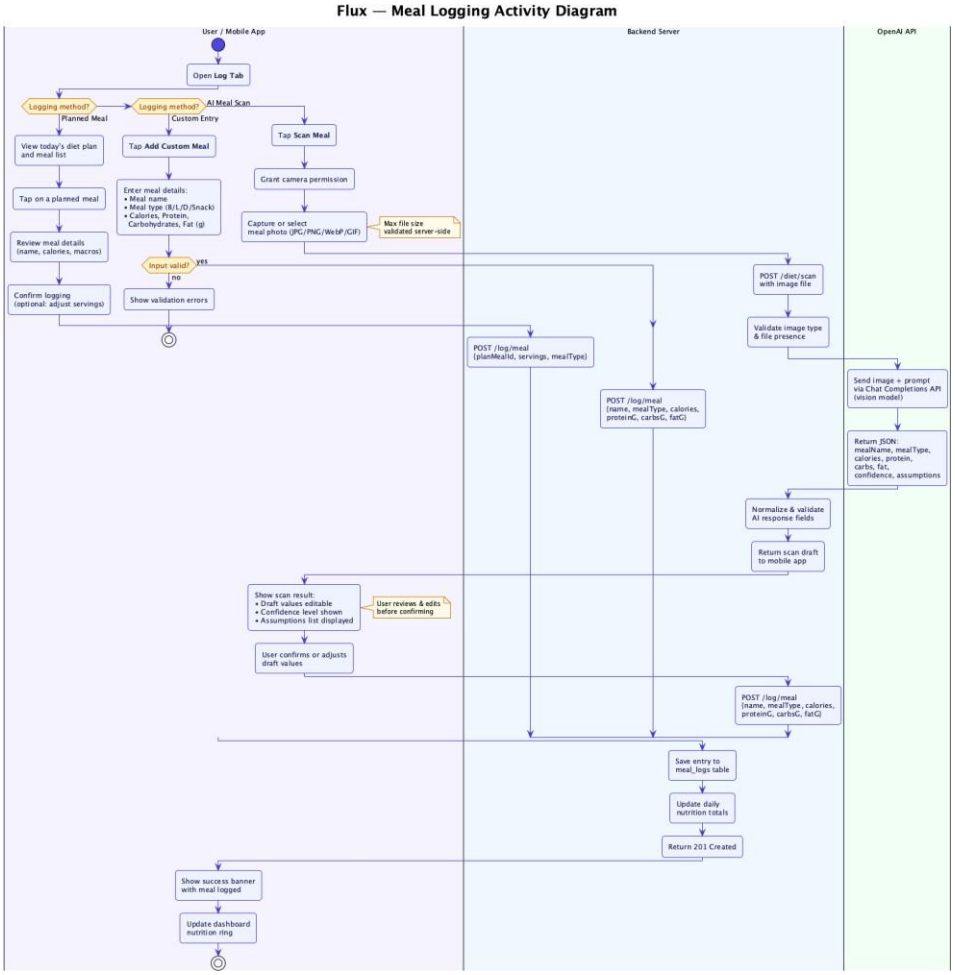


*Figure 3.6: Activity Diagram — Meal Logging (Planned / Custom / AI Scan)*

Figure 3.7 shows the two-path exercise logging activity: planned exercise entry (from the active workout plan, with MET-based calorie calculation) and custom free-form activity entry (with AI calorie estimation via OpenAI).

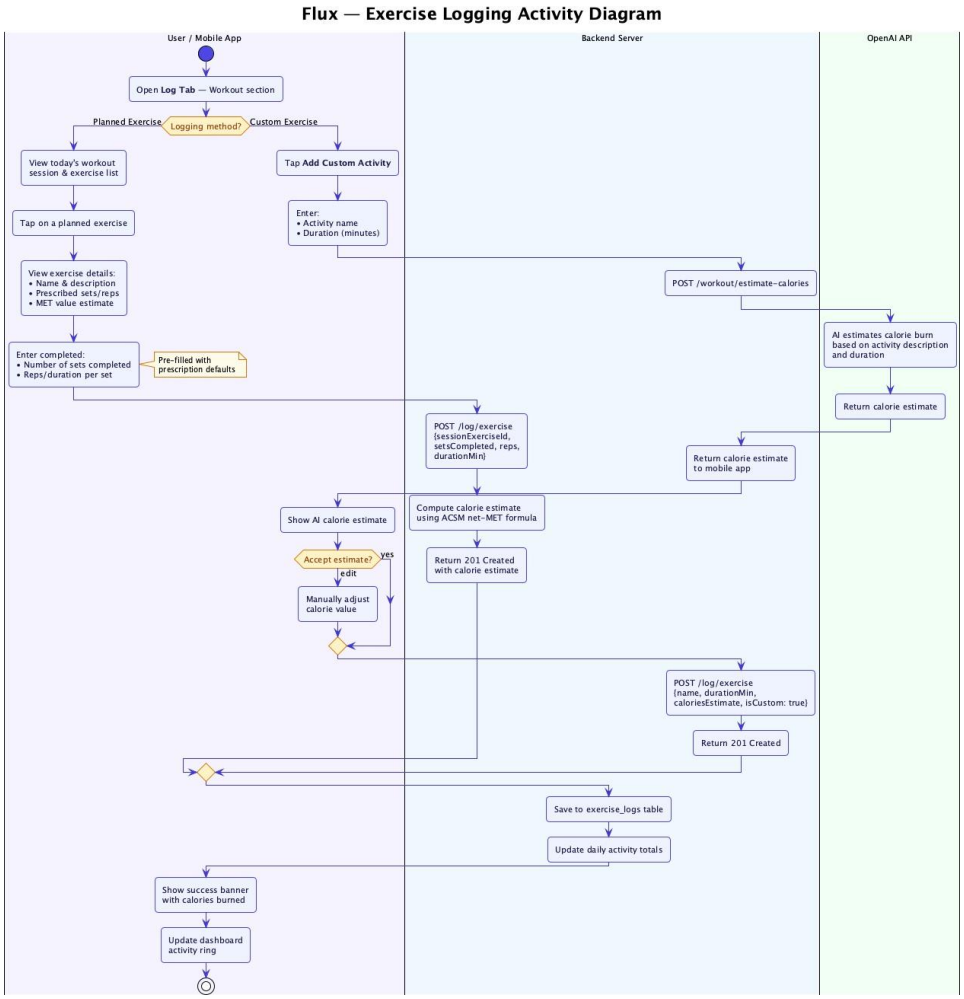


*Figure 3.7: Activity Diagram — Exercise Logging (Planned / Custom)*

Figure 3.8 shows the swap flow that is shared between diet and workout plans. For meal swaps, the backend retrieves ranked alternative recipes filtered by preferences, allergies, and disease guardrails, computes a nutritional impact comparison (before and after), and presents it for user review before confirming the change. For exercise swaps, an equivalent comparison is computed for calorie burn. All swaps are recorded for traceability.

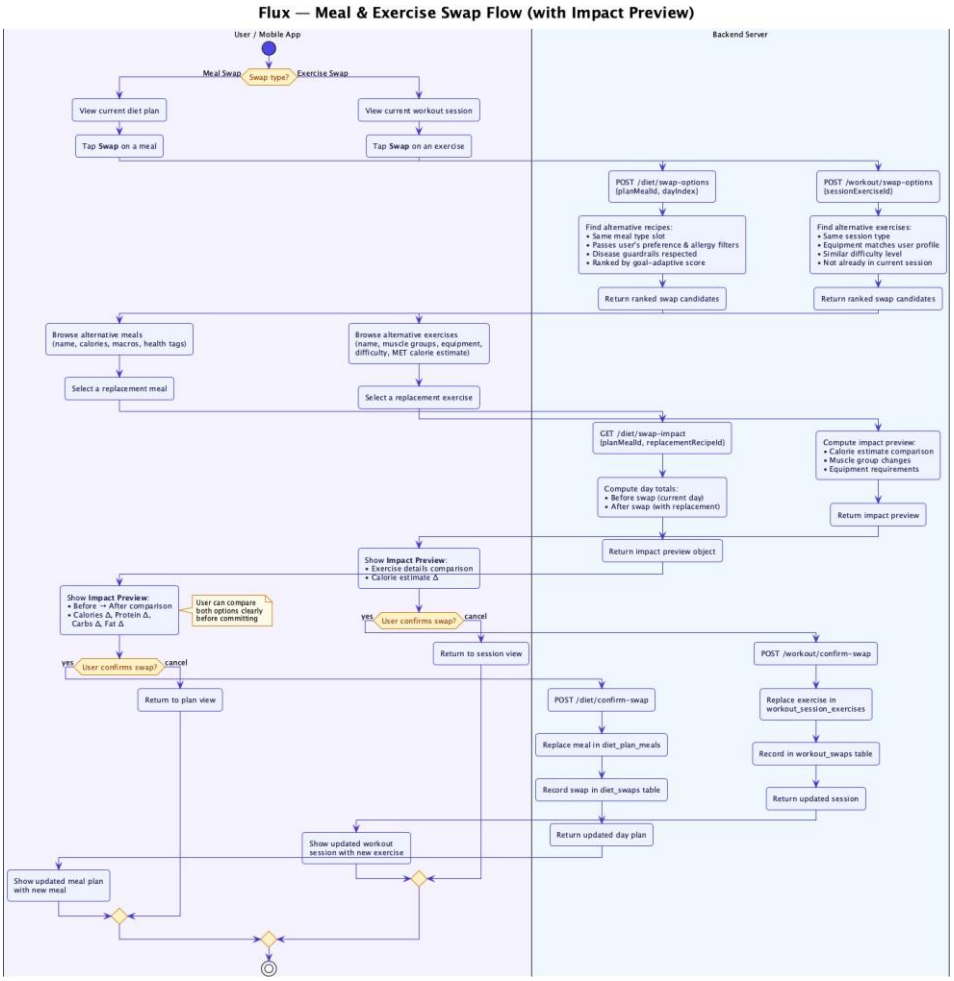


*Figure 3.8: Activity Diagram — Meal and Exercise Swap Flow (with Impact Preview)*

Figure 3.9 shows the progress tracking and AI weekly insights flow. Logged data is aggregated for the current week and presented as adherence metrics. For AI insights, the backend assembles an anonymized aggregate summary (no PII) and sends it to the OpenAI Chat Completions API, which returns three structured coaching insights. A disclaimer is displayed alongside all AI-generated content.

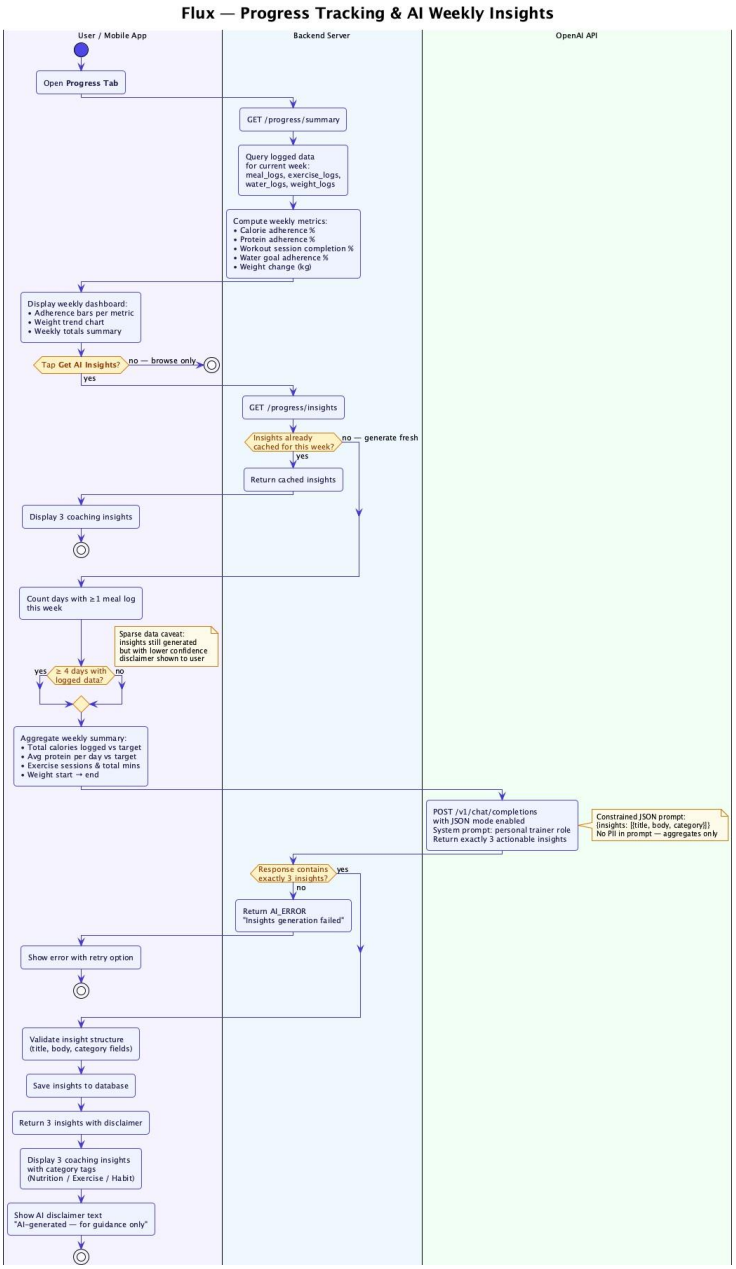


*Figure 3.9: Activity Diagram — Progress Tracking and AI Weekly Insights*

## 3.2 Development Methodology

The Agile model was selected as the Software Development Life Cycle (SDLC) framework for the Flux project. Unlike traditional waterfall methodologies, which require all requirements to be fully specified before development begins, Agile embraces iterative and incremental development. This is particularly well-suited to a mobile application with multiple interconnected modules, such as authentication, onboarding, recommendation engines, logging, and AI integration where early feedback and continuous refinement are necessary to deliver a coherent and high-quality product.

The Agile process was applied to Flux through a series of short development sprints, each lasting approximately one to two weeks. At the start of each sprint, a set of features or tasks was prioritized based on dependency order and estimated effort. Development, testing, and review were conducted within each sprint before moving to the next set of features. This cycle allowed design decisions and evidence-based logic, such as macronutrient formulas, disease guardrails, and goal-adaptive scoring to be validated incrementally rather than deferred until the end of the project.

### 3.2.1 Planning Phase

The planning phase involved gathering and analyzing the functional requirements of the Flux application. This included defining the target users, their health and fitness goals, the types of data the system must collect, and the recommendation logic required. A high-level system architecture was designed to establish the three-tier client–server model: Flutter mobile frontend, Node.js Express backend, and MySQL relational database. The tech stack, external API integrations (OpenAI, Google OAuth, Gmail SMTP), and database schema were scoped during this phase.

### 3.2.2 Design Phase

The design phase produced the system artefacts required to guide implementation. This included the entity–relationship diagram (ERD) defining all 24 database tables, the use case diagram capturing all functional capabilities and actors, data flow diagrams (DFD Level 0 and Level 1) showing how data moves through the system, sequence diagrams for key interactions, and activity and flowchart diagrams for all major user journeys. The UI/UX design system based on a amber styling with card-based layouts and micro-animations was also established during this phase.

### 3.2.3 Implementation Phase

Implementation was conducted module by module in dependency order. The authentication module was built first, establishing the JWT access and refresh token infrastructure. User onboarding and health metrics computation followed, producing the calorie and macronutrient targets required by the recommendation engines. The diet recommendation engine and workout recommendation engine were then developed, incorporating evidence-based scoring, disease-specific guardrails, and goal-adaptive logic. Subsequent sprints delivered the daily logging module, AI meal scanning via OpenAI, the progress and insights module, and the meal and exercise swap flows.

### 3.2.4 Testing Phase

Testing was performed at multiple levels. Each backend module was verified by running API calls against the live MySQL database to confirm that data persisted correctly. The Flutter application was tested against the real backend (not mocked data) to verify that API responses were parsed and rendered accurately on-screen. Edge cases such as expired tokens, infeasible dietary constraints, and invalid image uploads were explicitly tested for each module. Integration testing confirmed that cross-module flow, for example, onboarding metrics feeding into plan generation, and plan generation linking to logging behaved correctly end to end.

### 3.2.5 Review and Deployment Phase

At the end of each sprint, the completed features were reviewed against the project requirements and evidence standards defined in the project guidelines. Recommendation logic, formula citations, and scoring rules were verified against named evidence sources, including WHO, ACSM, ISSN, AHA, ADA, and FDA guidance [5], [10], [12], [13], [22], [33]. Documentation in the project reports was updated to reflect any changes to system behavior. The final review phase covers overall system validation, report finalization, and preparation of the FYP submission.

## 3.3 Software and Development Tools

In this section, the tools, frameworks, and techniques used in the project are discussed with reference to their relevance and rationale.

### 3.3.1 Programming Languages

**1. JavaScript**


*Figure 3.10: JavaScript*

JavaScript is a programming language that can run in web browsers and on servers. In this project, JavaScript is used to write the entire backend server code for Flux. It processes requests from the mobile application, runs business logic such as plan generation and health metric computation, and returns structured responses. Node.js allows JavaScript to run on the server side, while Express provides a framework for defining API routes and handling HTTP methods [17], [18].

**2. Dart (Flutter)**


*Figure 3.11: Flutter (Dart)*

Dart is the programming language used by Flutter. It allows a single codebase to target both Android and iOS devices, which reduces the development effort needed for cross-platform support. In Flux, Dart is used to write the screen layouts, state management providers, navigation logic, and API communication code. Dart also supports asynchronous programming, which is important for sending requests to the backend and waiting for responses without freezing the user interface [16].

### 3.3.2 Frameworks and Libraries

**1. Node.js and Express**


*Figure 3.12: Node js Express Framework*

Node.js is a runtime environment that allows JavaScript to run on the server side, outside of a web browser. Express is a lightweight web framework built on top of Node.js that makes it easier to define API routes and handle HTTP requests. In Flux, the backend server is built using Node.js and Express. Express is used to set up the API endpoints for each feature, such as user authentication, plan generation, meal and exercise logging, and progress tracking, and to connect each endpoint to the correct controller and service function [17], [18].

### 3.3.3 Database

**MySQL**

*Figure 3.13: MySQL*

MySQL serves as the relational database for the system. It stores user accounts, profile information, health metrics, workout datasets, and generated plans. Its ACID compliance ensures data reliability and consistency, while structured schema design with foreign keys enforces integrity across tables such as accounts, user_info, user_metrics, workout_plans, and workout_history. MySQL integrates with Node.js using the mysql2 library, which is used to run parameterized SQL queries from the backend service and repository layers.

### 3.3.4 Development Tools

**1. Visual Studio Code**

*Figure 3.14: Visual Studio Code*

VS Code is the primary lightweight editor for Dart, Flutter, JavaScript, and Node.js development. It is used to write and debug Dart and Flutter code on the frontend, and JavaScript and Node.js code on the backend. The built-in terminal is also used to run the backend server and manage the database during development.

**2. Android Studio**


*Figure 3.15: Android Studio*

Android Studio is used for mobile builds, testing on emulators, and deploying APKs. It provides a comprehensive environment for managing Gradle builds, monitoring logs, and ensuring the Flutter mobile app runs correctly on Android devices. On macOS, it also supports iOS simulator testing for cross-platform validation.

### 3.3.5 Hardware

| Description | Specifications |
|---|---|
| Model | MacBook Air (M2-2022) |
| Processor | Apple M2 Chip |
| Operating System | Mac OS 18.5 |
| Graphic | Media Engine |
| Memory | 16GB unified memory |
| Storage | 1TB SSD |

*Table 3.1: Hardware Specification*

## 3.4 Gantt Chart

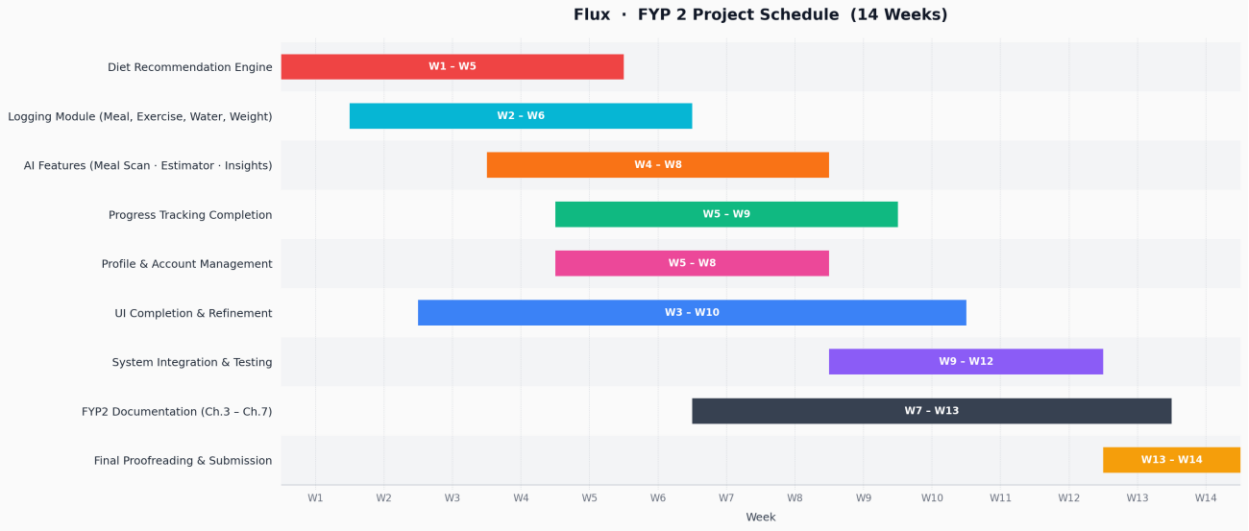


*Figure 3.16: Flux FYP2 Project Gantt Chart*

# Chapter 4 System Design

## 4.1 System Block Diagram

This chapter explains how the Flux system is designed. It covers the system block structure, the role of each major component, the database and recommendation logic design, and how the components interact during system operation.

Figure 4.1 shows the system block diagram for Flux. The diagram presents the four main tiers of the system: the mobile user, the Flutter client tier, the Node.js/Express application tier, and the MySQL data tier. External services including Google OAuth 2.0, Gmail SMTP, and the OpenAI API connect to the application tier. Pre-loaded datasets for recipes and exercises are seeded into the database at system startup.

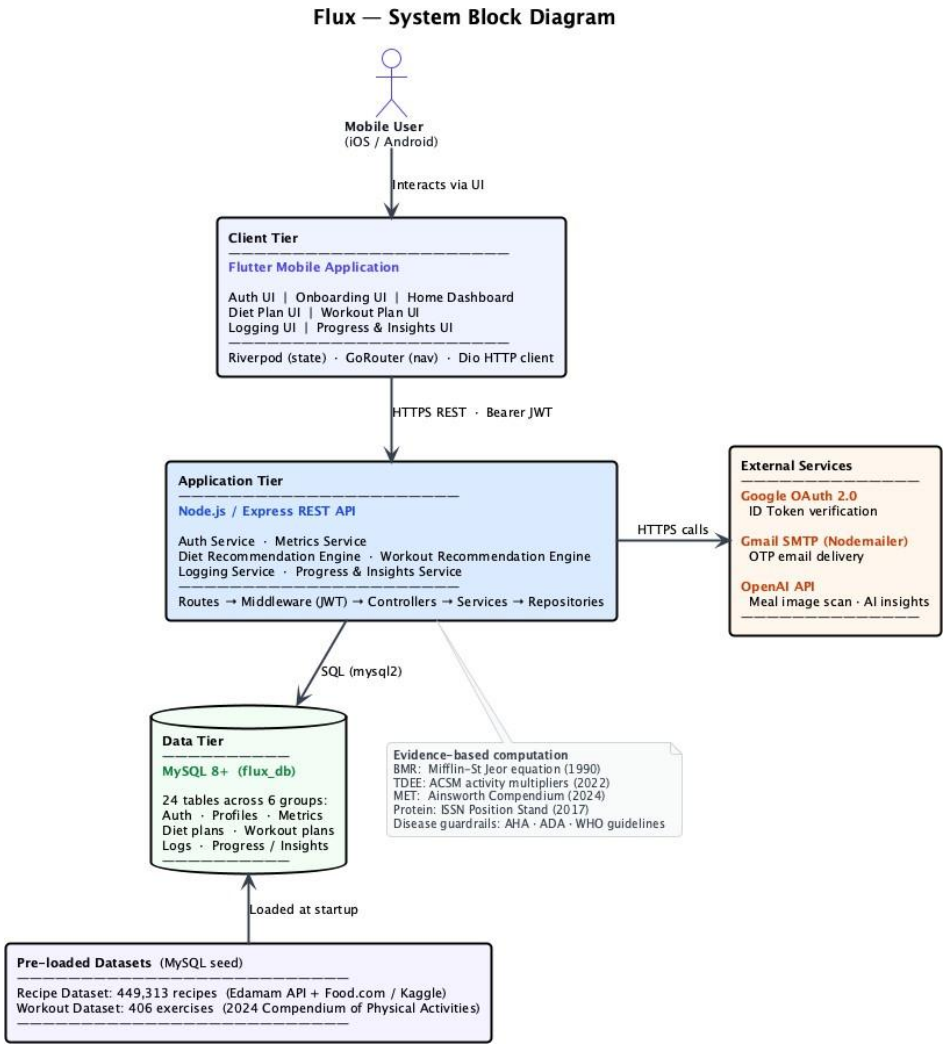


*Figure 4.1: Flux System Block Diagram*

## 4.2 System Components Specifications

### 4.2.1 Mobile Application Component

The mobile application is implemented using Flutter and Dart. It contains the user interface screens, Riverpod state providers, Dio HTTP client integration, GoRouter navigation, secure token storage, and local form validation used across authentication, onboarding, plan generation, logging, progress tracking, and profile management.

### 4.2.2 Backend API Component

The backend component is implemented using Node.js and Express. It exposes REST API routes, validates protected requests using JWT access tokens, delegates request handling to controllers, places business and recommendation logic in services, and uses repositories for database access.

### 4.2.3 Database Component

The database component is implemented using MySQL. It stores authentication data, user profiles, computed health metrics, dietary preferences, recipe records, workout data, generated plans, logged meals and workouts, progress values, and token records.

### 4.2.4 External Service Components

The external service components include OpenAI for AI-assisted meal photo interpretation and weekly insight generation, Google OAuth for Google sign-in, and Gmail SMTP for email OTP delivery. These services are accessed through the backend rather than directly from the mobile client.

## 4.3 Software Components Design

This section documents the software component design of Flux. The design is organised into three areas: the database schema design that defines the relational structure used to persist all application data, the recommendation and security logic flowcharts that describe the evidence-based computation pipelines, and the architecture and data flow diagrams that illustrate the relationships between system components.

### 4.3.1 Database Design

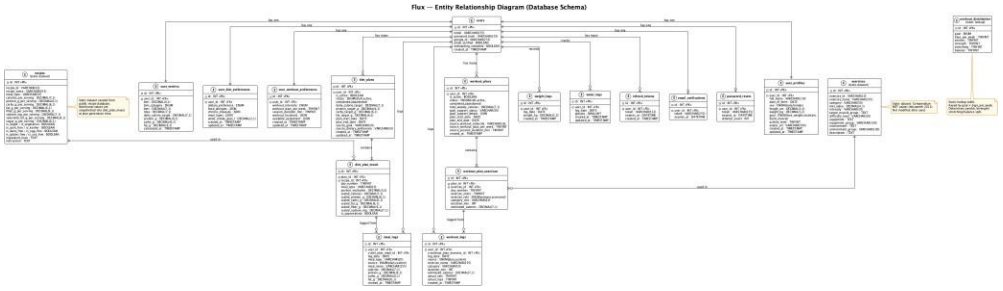


*Figure 4.2: Flux Database Entity-Relationship Diagram (Full Schema)*

The Entity-Relationship Diagram in Figure 4.2 depicts the full relational schema of the Flux database (flux_db). The schema comprises 19 core tables organised into five logical domains, connected via foreign key relationships shown using crow's foot notation.

The Authentication domain stores user credentials (users table), email verification OTPs (email_verification_otps), and JWT refresh tokens (refresh_tokens). The User Profile domain stores the extended user profile (user_profiles), computed health metrics including BMI, BMR, TDEE, and macro targets (user_metrics), dietary preferences (user_diet_preferences), and food allergies (user_food_allergies). The Diet domain stores the recipe catalogue with pre-computed nutritional data (recipes), active diet plans (diet_plans), and individual meal assignments per plan day (diet_plan_meals). The Workout domain stores the exercise catalogue with MET values and difficulty ratings (exercises), workout plans (workout_plans), individual exercise assignments per session (workout_plan_exercises), and exercise swap history (exercise_swaps). The Activity Logs domain stores daily meal logs (meal_logs), exercise logs (exercise_logs), water intake logs (water_logs), weight entries (weight_logs), and AI-generated weekly coaching insights (weekly_insights). All tables use an integer auto-increment primary key, and the schema is initialised through sequential migration scripts.

### 4.3.2 Recommendation and Security Logic Design

Figure 4.3 shows the sequential pipeline for computing all user health targets. The pipeline applies BMI classification [32], BMR estimation using the Mifflin-St Jeor equation [4], TDEE estimation using ACSM activity multipliers [33], a goal-adjusted calorie target based on an energy-deficit heuristic [34] with ACSM-informed safety-floor enforcement [33], and macronutrient targets for protein [5], fat [35], and carbohydrate minimums [36]. A protein feasibility check is performed against the filtered recipe pool before plan generation proceeds.

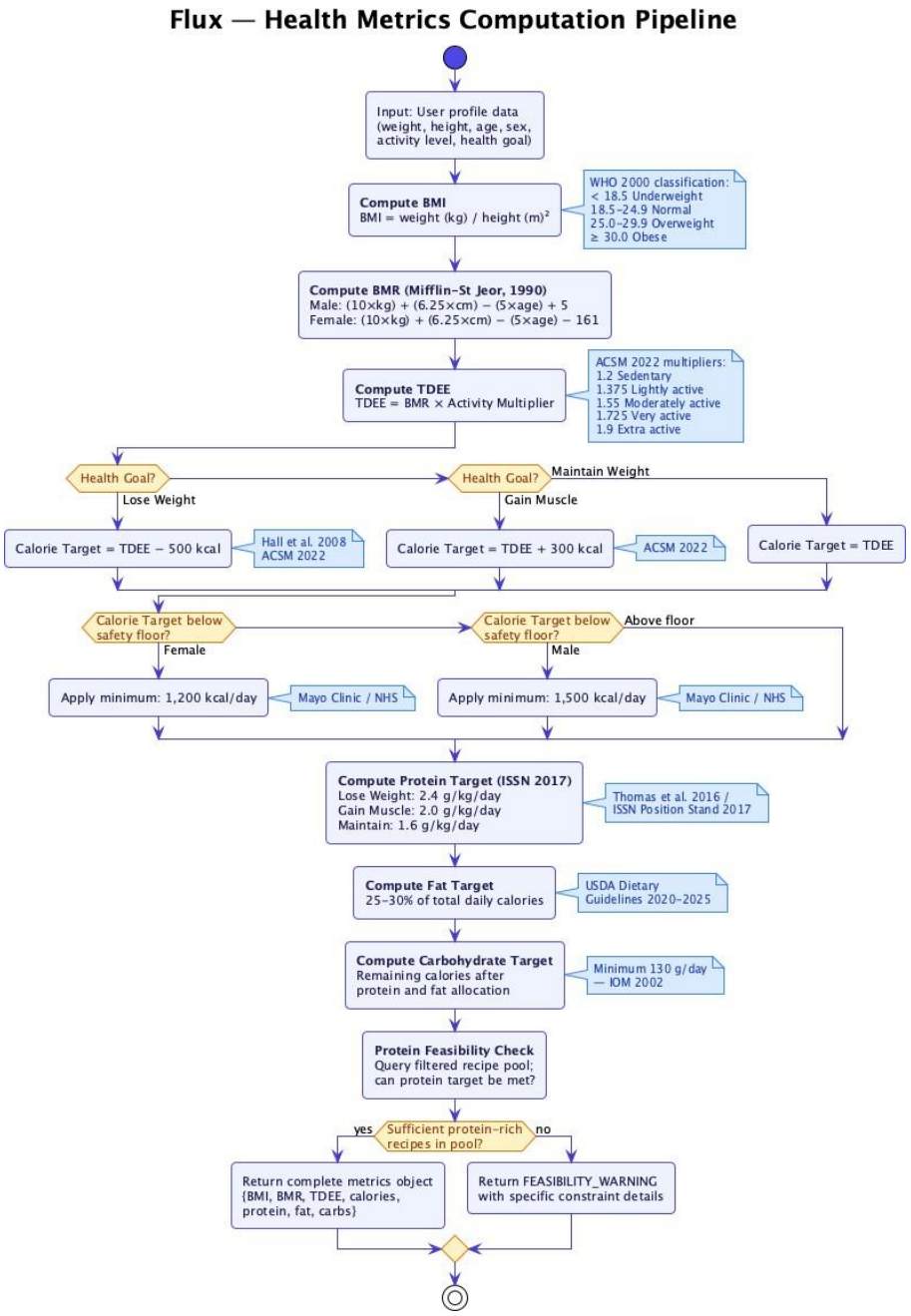


*Figure 4.3: Flowchart — Health Metrics Computation Pipeline*

Figure 4.4 shows the four-stage diet recommendation engine. Stage 1 filters the recipe pool by dietary preference and allergy flags. Stage 2 assembles meals per day using beam search (width 5) for disease conditions or standard slot fill otherwise. Stage 3 applies goal-adaptive scoring (protein-weighted for muscle gain, calorie-weighted for fat loss) and validates each day within ±25%/±35% tolerance. Stage 4 applies FDA-based health tags [22] to all recipes in the final plan.

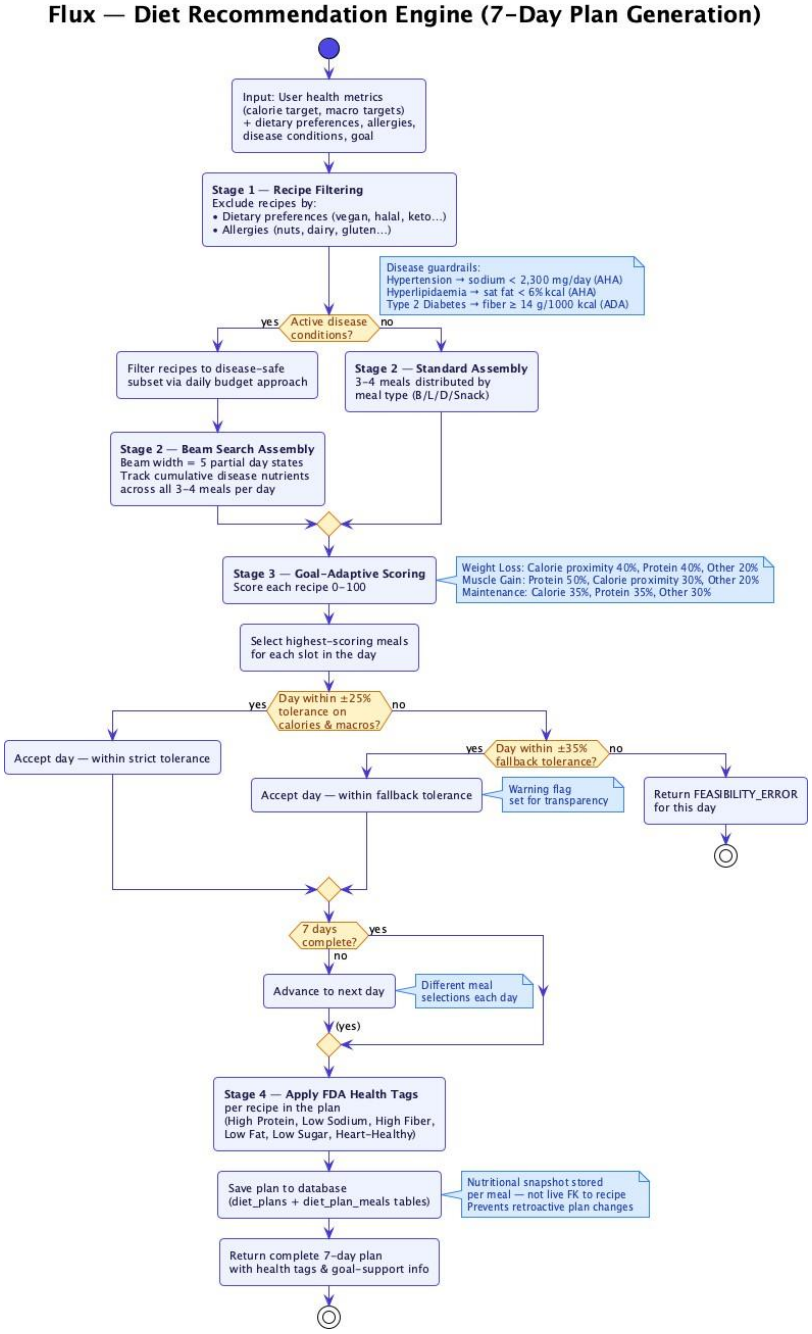


*Figure 4.4: Flowchart — Diet Recommendation Engine (7-Day Plan Generation)*

Figure 4.5 shows the six-stage workout recommendation engine. Stage 1 determines session distribution by goal, days per week, and location (gym/home). Stage 2 filters the exercise pool by session type, equipment, and difficulty. Stage 3 selects compound exercises first, then fills remaining slots with accessories, scaled by session duration. Stage 4 generates structured prescriptions (sets × reps or duration). Stage 5 estimates calorie burn using the ACSM net-MET formula [33] with Compendium MET values [11]. Stage 6 evaluates the plan against WHO and ACSM guidelines [12], [33] to produce a goal-support rating (Strong / Moderate / Limited).

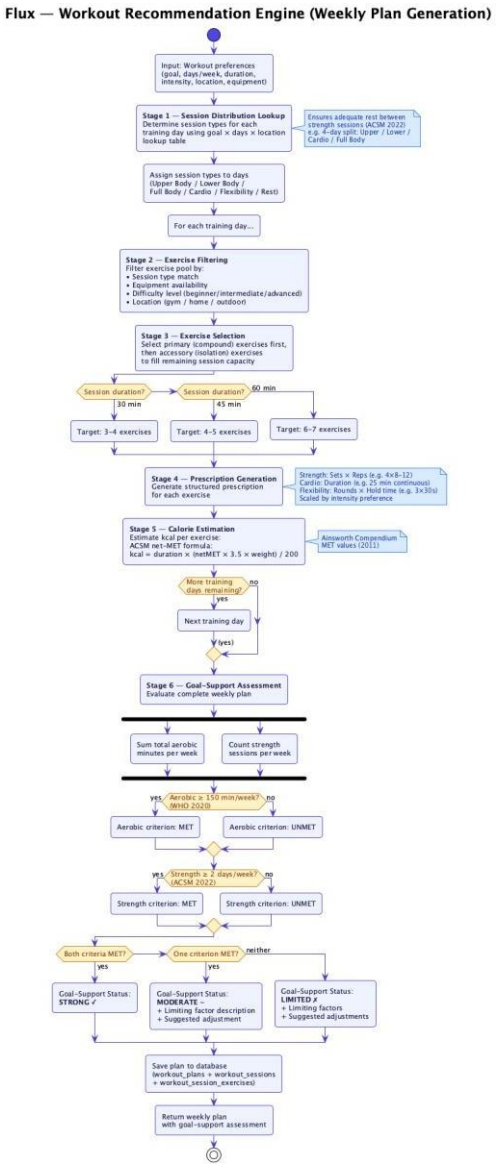


*Figure 4.5: Flowchart — Workout Recommendation Engine (Weekly Plan Generation)*

Figure 4.6 shows the Completer-lock-based JWT token refresh mechanism. When any API call returns 401 (expired access token), the Dio interceptor acquires a lock before sending a single POST /auth/refresh request. All concurrent failing requests await the lock result and retry with the new token once issued. If the refresh token is invalid or expired, all tokens are cleared and the user is redirected to the login screen.

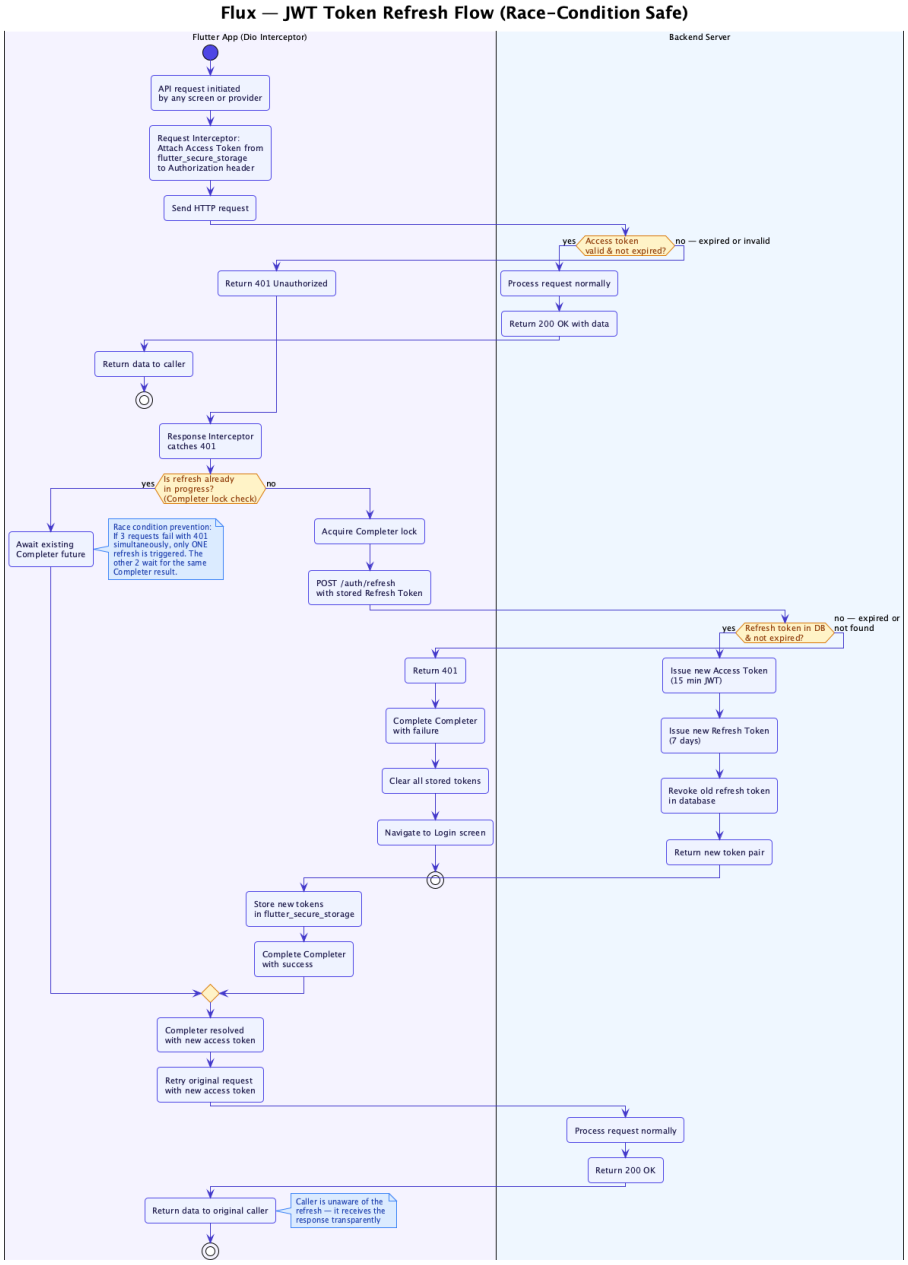


*Figure 4.6: Flowchart — JWT Token Refresh Flow (Race-Condition Safe)*

## 4.4 System Components Interaction Operations

This section explains how the system components interact during operation. Data flow diagrams describe movement of data between users, external services, backend processes, and data stores. Sequence diagrams then show the time-ordered interactions for selected high-risk workflows.

### 4.4.1 Data Flow Operations

The Context Diagram (Data Flow Diagram Level 0) in Figure 4.7 treats the entire Flux system as a single process bubble and shows all external entities that interact with it, along with the data flows between them. This provides the highest-level view of the system boundary, clearly distinguishing what is inside the system from what is external to it.

Two user roles interact with the system: Guest Users who submit registration and login requests and receive authentication tokens in return, and Registered Users who provide health profiles, generate plan requests, submit daily logs, and receive personalized plans, progress data, and AI coaching insights. Three external systems are shown: the OpenAI API (bidirectional - receives meal images, activity descriptions, and progress summaries; returns structured JSON responses), the Google OAuth API (bidirectional - receives Google ID tokens; returns verified user identity), and Gmail SMTP (outbound - receives requests to send OTP emails).

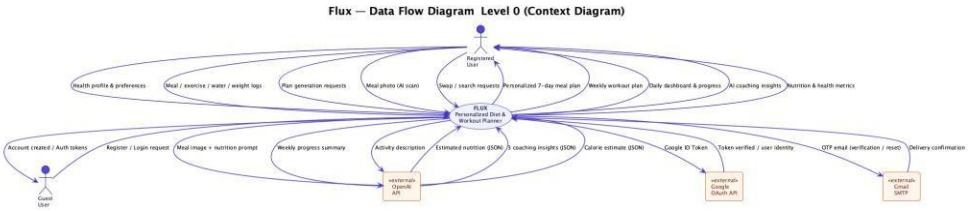


*Figure 4.7: Data Flow Diagram — Level 0 (Context Diagram)*

Figure 4.8 presents the Data Flow Diagram Level 1, which decomposes the Flux system into six core processes (P1 to P6) and five data stores (D1 to D5). P1 handles Authentication and Account Management, interfacing with D1 (Users & Auth), Google OAuth, and Gmail SMTP. P2 manages Onboarding and Health Metrics Computation, writing computed data to D2 (User Profile). P3 covers Diet Plan Generation and Management, reading preferences from D2 and the recipe pool from D3 (Diet Data), and calling the OpenAI API for meal scanning and recipe enrichment. P4 handles Workout Plan Generation and Management using D2 preferences and the exercise pool from D4 (Workout Data). P5 manages Daily Logging, reading plan references from D3 and D4 and writing all log entries to D5 (Activity Logs). P6 handles Progress and AI Insights, reading weekly logs from D5 and calling the OpenAI API to generate coaching insights.

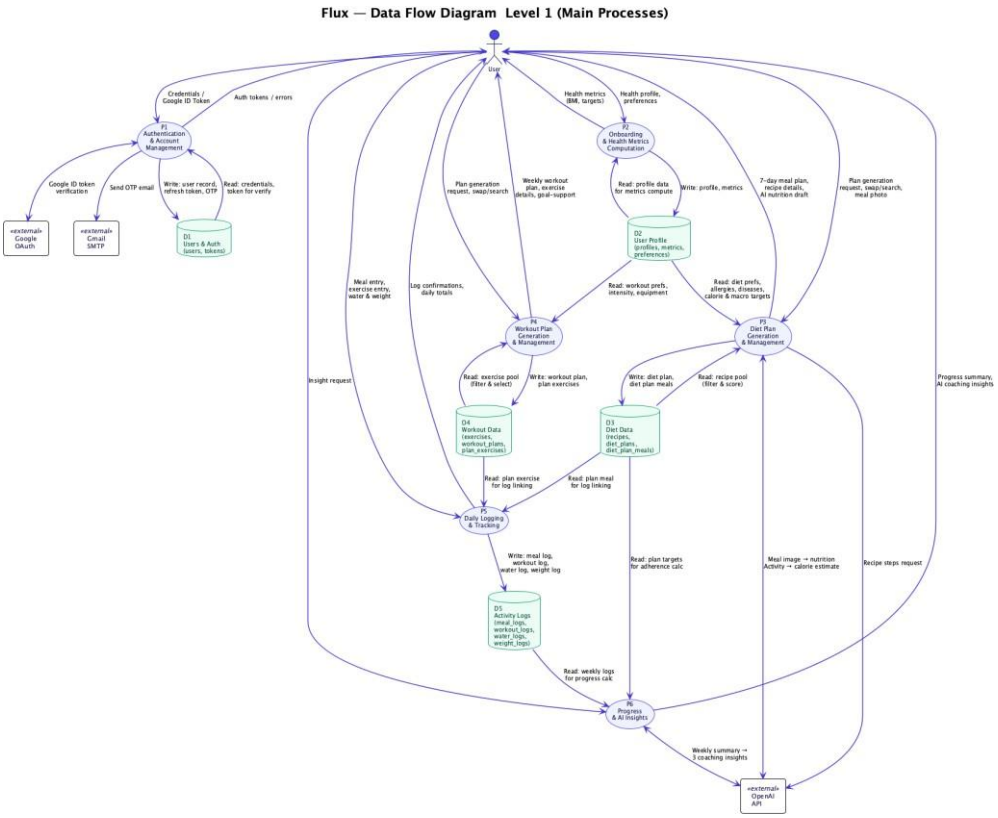


*Figure 4.8: Data Flow Diagram — Level 1 (Main Processes)*

### 4.4.2 Sequence Operations

Sequence diagrams show the time-ordered interactions between system components for specific use cases. Three key flows are documented below: user authentication and token refresh, diet plan generation, and AI meal image scanning and logging.

Figure 4.9 traces the email/password login flow and the automatic token refresh mechanism. The Flutter application sends login credentials to the Express API, which delegates to authService for bcrypt password verification against the database. On success, a 15-minute JWT access token and a 7-day refresh token are issued and written to flutter_secure_storage on the device. When a subsequent API call returns 401 (expired access token), the Dio interceptor acquires a Completer-based lock to serialize the refresh only the first failing request sends POST /auth/refresh, while all other concurrent failing requests await the Completer result before retrying with the new token.

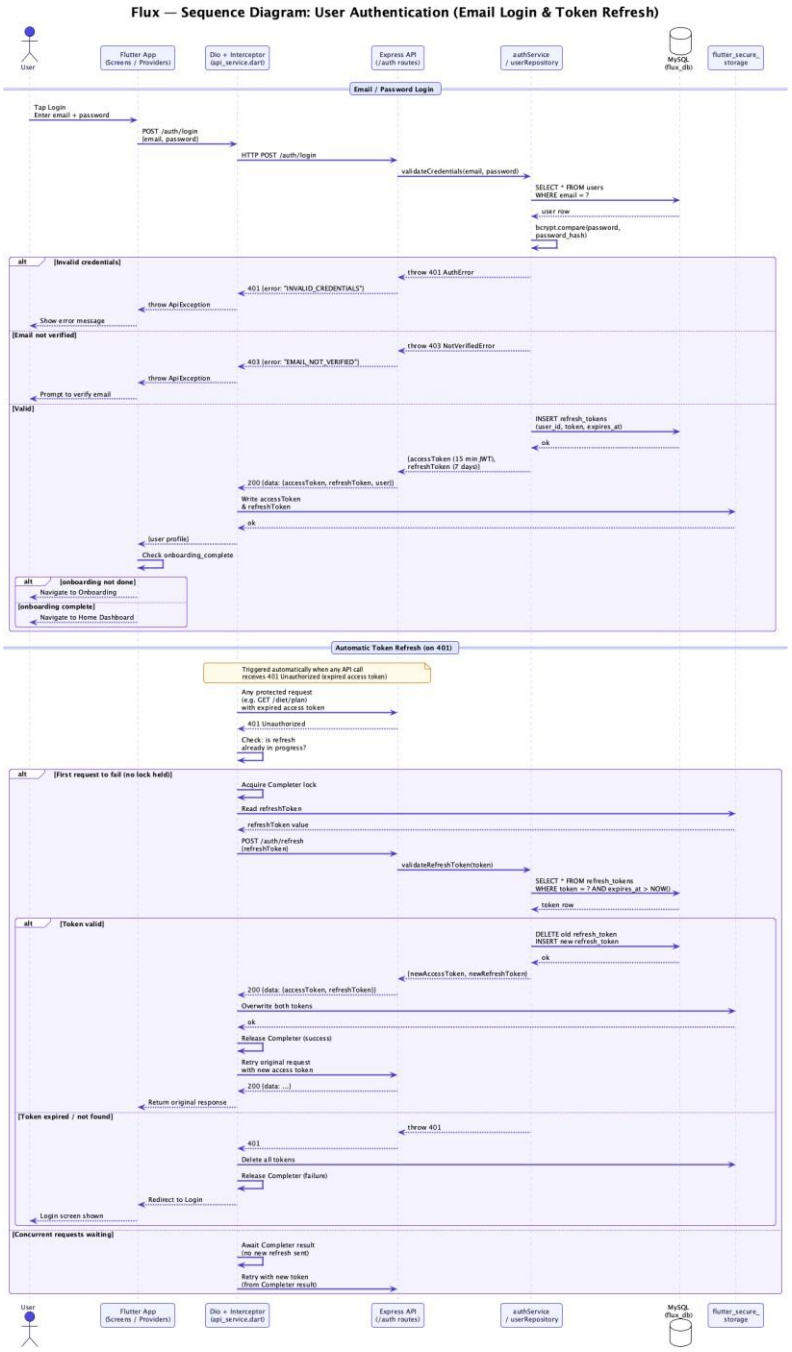


*Figure 4.9: Sequence Diagram — User Authentication and Token Refresh*

Figure 4.10 traces the diet plan generation pipeline from the mobile client through to the database. The dietController extracts the user ID from the JWT, fetches computed health metrics from metricsService, and reads dietary preferences and allergies from the database before calling dietService.generatePlan(). The service executes four internal stages: recipe pool filtering by preference and allergy flags using parameterised SQL; per-day scoring and meal slot assembly using beam search (width 5) for users with disease conditions, or standard slot fill otherwise; day-level tolerance validation (±25%/±35% on calories and macros); and FDA health tag application [22]. The completed plan is written to diet_plans and diet_plan_meals, and any previously active plan is deactivated atomically.

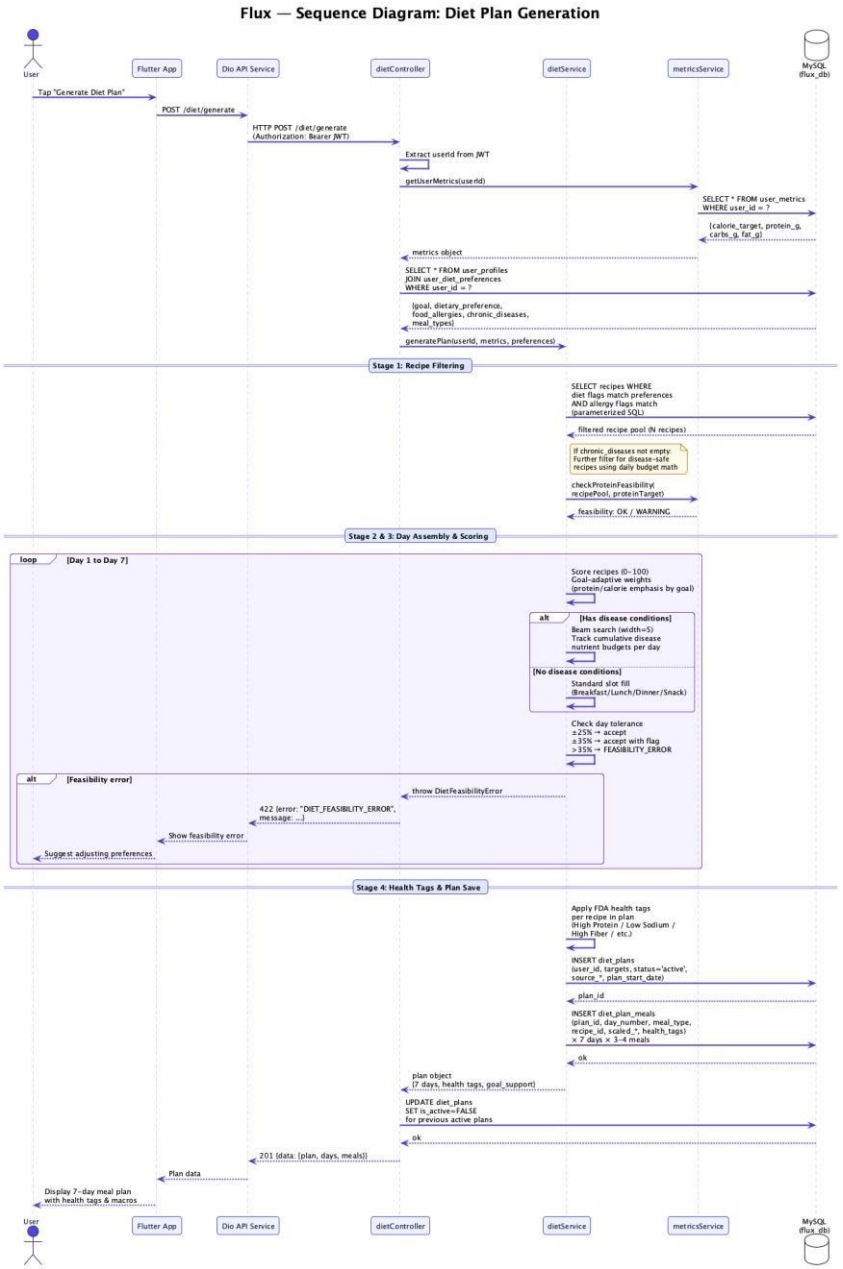


*Figure 4.10: Sequence Diagram — Diet Plan Generation*

Figure 4.11 traces the AI meal scan flow. The user captures a meal photo in the Flutter app, which sends it as multipart/form-data to POST /diet/scan. The mealScanService validates the image MIME type and buffer signature, converts the image to a base64 data URL, and sends it to the OpenAI Chat Completions API (GPT-5.5 vision) with a structured prompt. The model returns a JSON object with estimated meal name, meal type, calories, macronutrients, a confidence level, and a list of assumptions. The service normalises and validates the response before returning a draft to the client. The user reviews and may edit the values before logging the entry via POST /log/meal.

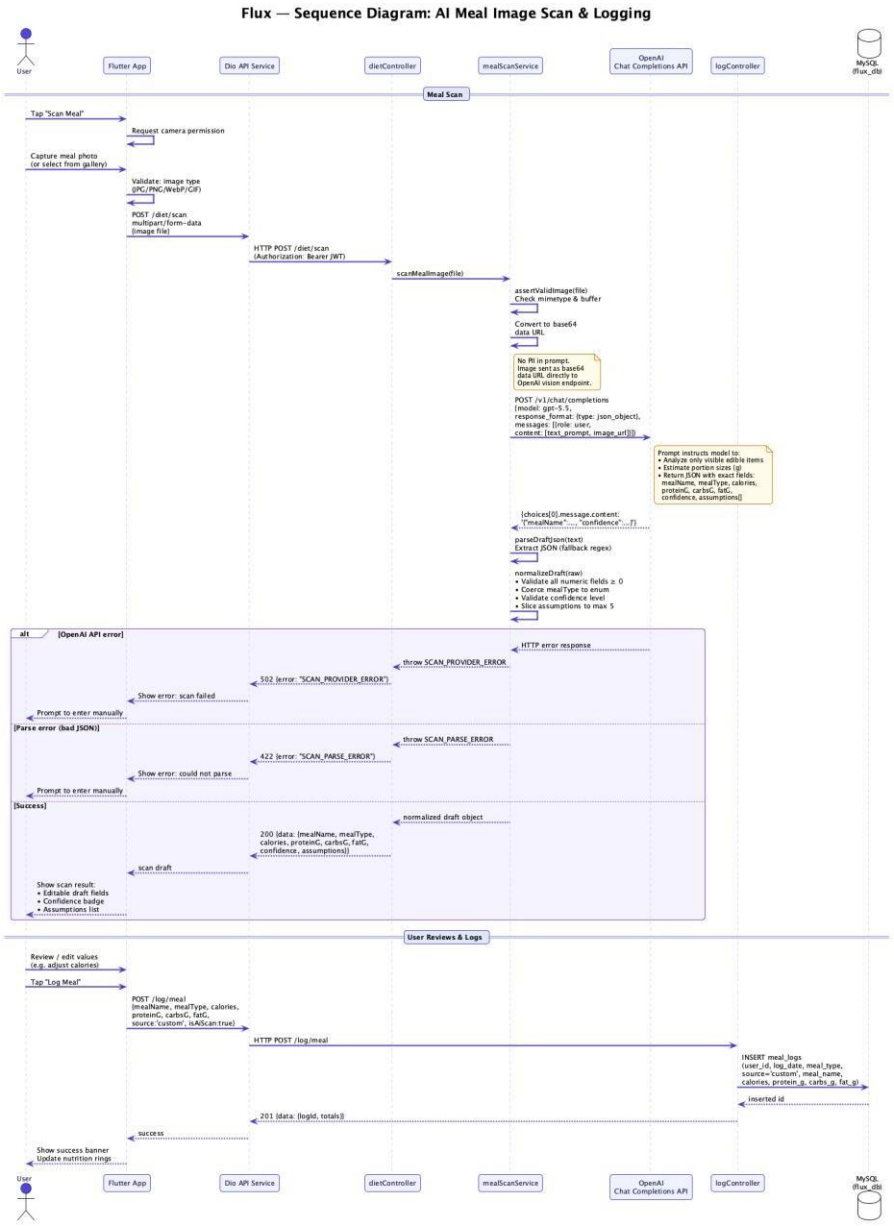


*Figure 4.11: Sequence Diagram — AI Meal Image Scan and Logging*

## 4.5 Summary of Chapter

This chapter has documented the detailed system design of Flux. The system block view explains how the mobile application, backend API, database, and external service components are arranged, while the component specifications describe the responsibility of each major part. The software component design section presents the database schema and the core recommendation and security flowcharts. The component interaction section then explains data flow operations and sequence-level behavior for authentication, diet plan generation, and AI meal image scanning. Together, these design artefacts provide enough structure for another developer to understand, rebuild, and verify the Flux system.

# Chapter 5 System Implementation

## 5.1 Hardware Setup

Flux was developed and tested on a MacBook running macOS Sonoma 14.x. The machine ran the full development stack simultaneously: the Flutter mobile toolchain, an iOS Simulator (iPhone 16 Pro Max via Xcode 15), the Node.js backend server, and a local MySQL 8 database instance. Mobile UI verification was performed on the iOS Simulator. Backend API behaviour and database state were verified by running the automated test suite and by querying the live MySQL database directly after each significant change.

## 5.2 Software Setup

The software environment for Flux consists of the following tools and versions. Flutter 3.x with Dart was used for the mobile application. Node.js 20.x with Express 4.x was used for the REST API backend. MySQL 8 was used for relational data storage. Android Studio and VS Code with the Flutter and Dart extensions were used as the primary development IDEs. Git was used for version control. Supporting external integrations include the Google OAuth 2.0 API for social sign-in, Gmail SMTP via Nodemailer for OTP email delivery, and the OpenAI API (GPT-5.5) for AI-assisted meal scanning, recipe step generation, custom workout calorie estimation, and weekly coaching insights. The project is organized into two top-level folders: flux_app for the Flutter mobile application and backend for the Node.js API server, keeping the frontend and backend independently runnable.

## 5.3 Setting and Configuration

The backend configuration is managed through a .env file that is not committed to version control. It stores the following categories of environment-specific values: MySQL connection details (host, port, database name, username, and password), JWT secrets and token expiry settings (15-minute access token, 7-day refresh token), Gmail SMTP credentials for OTP delivery (sender address and app password), Google OAuth client ID for server-side ID token verification, and the OpenAI API key and base URL. The Flutter application stores the backend API base URL and uses flutter_secure_storage to persist JWT access and refresh tokens between sessions. Riverpod providers are used across all modules to manage remote state, automatically re-fetching data from the backend after plan generation, logging, swaps, and profile updates to keep the UI consistent with the live database.

## 5.4 Module Implementation Details

The following subsections present the functional test cases for each module. Screenshots of the user interface and detailed module walkthroughs are provided in Section 5.5 of Chapter 5.

### 5.4.1 Implementation Overview

This section describes the implementation of the Flux application, describing how each module was built, the key technical decisions made, and the algorithms and external integrations that power the system. The implementation spans the full-stack architecture: a Flutter mobile frontend, a Node.js Express REST API backend, and a MySQL relational database. Each section maps one functional module to its corresponding API endpoints, database tables, backend service logic, and Flutter screens.

The implementation follows the four-layer backend architecture established in the design phase: route handlers that map HTTP endpoints, controllers that validate and delegate requests, services that contain business and recommendation logic, and repositories that execute SQL queries. All inter-layer communication uses plain JavaScript objects; no ORM framework is used, keeping the data access layer transparent and easy to reason about. The Flutter frontend communicates with the backend exclusively via HTTPS REST calls through the Dio HTTP client, which is configured with automatic JWT token injection and transparent token refresh on authentication expiry.

### 5.4.2 Data Preparation

Two datasets underpin the Flux recommendation engines: a recipe dataset that powers the diet plan generation, and a workout exercise dataset that powers the workout plan generation and calorie estimation. Both datasets were collected from external sources, processed through cleaning pipelines, and loaded into MySQL tables before the recommendation services could operate.

Table 5.1 summarizes the four dataset artefacts produced during data preparation, the source of each, and its role in the system.

| Dataset | Source | Raw Records | Final Records | Purpose |
|---|---|---|---|---|
| Recipe – Edamam API [7] | Edamam Recipe Search API v2, edamam.com [7] | 58,095 | 58,095 | Nutritionally complete recipes |

| | | | | |
|---|---|---|---|---|
| | | | | with full macro and micro data |
| Recipe – Food.com | Food.com Recipes and Interactions dataset on Kaggle [37] | 391,218 | 391,218 | Large public recipe corpus with ingredients and instructions |
| Recipe – Merged | Edamam and Food.com unified and cleaned [7], [37] | 449,313 | 449,313 | Single recipe pool used by the Flux diet recommendation engine |
| Workout exercises | 2024 Adult Compendium of Physical Activities [23] | ~1,000 activities | 406 exercises | Exercise pool with evidence-based MET values for calorie estimation |

*Table 5.1: Dataset Overview*

**Recipe Dataset Collection**

The recipe dataset was built by combining two complementary sources: the Edamam Recipe Search API [7], which provides structured nutritional data and dietary labels for each recipe, and the Food.com public recipe dataset available on Kaggle [37], which contributes a much larger pool of recipes with ingredient lists and cooking instructions.

**Edamam API Collection [7]**

The Edamam Recipe Search API v2 [7] was used to collect recipes with verified macronutrient and micronutrient data. A custom Python crawler was written to automate the collection process. The crawler sent paginated search requests using two types of seed parameters: filter seeds (diet labels such as balanced, high-protein, low-carb, and health labels such as dairy-free, egg-free, gluten-free) and query seeds (dish names grouped into Malaysian, Asian, and global priority tiers). The priority grouping ensured that the resulting dataset had stronger coverage of Malaysian and Asian recipes, which are most relevant to the application's target user base.

To handle the Edamam API's rate limits [7], the crawler implemented automatic retry logic with exponential back-off and a configurable throttle wait period on HTTP 429 responses. Crawl state (the last pagination token for each category) persisted to a JSON file so that interrupted collection runs could resume from the last completed page without duplicating records. Duplicate recipes were detected by the recipe URI field and merged in place, with the

seen count and last-seen timestamp updated on each re-encounter. The full collection run produced 58,095 unique recipe records, stored in a raw CSV file.

```
# Key crawl loop structure (edamam_collector/scripts/fetch_edamam_recipes.py)
for category_type, category_value in build_category_jobs(filters, query_seeds):
    url = build_search_url(category_type, category_value)
    while url:
        response = fetcher.fetch(url)          # auto-retry + back-off
        for hit in response["hits"]:
            row = flatten_hit(hit, category_type, category_value, run_id)
            store.upsert(row)                  # deduplicates by recipe_uri
        url = extract_next_url(response)       # None when last page reached
        state.save(category_type, category_value, url)
```

Each raw record stored the full Edamam API payload [7] as JSON strings inside CSV columns. A separate transformation step flattened the nested nutritional fields into per-serving scalar values for the unified recipe schema.

**Food.com Kaggle Dataset**

The Food.com Recipes and Interactions dataset on Kaggle [37] is a large public recipe corpus containing 391,218 recipes collected from the Food.com cooking community. Unlike the Edamam dataset, which provides per-serving nutritional data directly from a trusted food database, the Food.com dataset contains raw nutritional totals that required normalization by serving count, and its dietary labels were not pre-computed.

The Food.com dataset [37] was downloaded as a CSV file and read into a pandas DataFrame. Nutritional values were divided by the serving count for each recipe to produce per-serving figures consistent with the Edamam schema. Recipes with zero or missing calorie values were removed from the dataset, as they cannot be used by the calorie-matching recommendation engine. After cleaning, the Food.com dataset contributed 391,218 recipe records to the merged pool.

**Recipe Dataset Merging and Cleaning**

Before the two datasets could be merged, their column schemas were unified. Both datasets were transformed to share the same column names and value formats. The unified schema has 31 columns: core recipe fields (recipe_id, recipe_name, meal_type, dish_type, servings, recipe_url, ingredient_lines, instruction), per-serving nutritional fields (calories, protein, carbohydrates, fat, saturated fat, fibre, sugar, cholesterol, and sodium), and 12 binary dietary label columns (is_vegan, is_vegetarian, is_pescatarian, is_pork_free, is_kosher, is_dairy_free, is_egg_free, is_gluten_free, is_soy_free, is_peanut_free, is_alcohol_free, is_kidney_friendly). Table 5.6 shows how the main Edamam API fields [7] were mapped to the unified schema.

| Edamam API Field | Unified CSV Column | Notes |
|---|---|---|
| recipe.label | recipe_name | Recipe display name |
| recipe.url | recipe_url | Source web page URL |
| recipe.yield | servings | Number of servings the recipe produces |
| recipe.calories / yield | calories_per_serving | Total calories divided by servings |
| totalNutrients.PROCNT.quantity / yield | protein_g_per_serving | Protein per serving in grams |
| totalNutrients.CHOCDF.quantity / yield | carbs_g_per_serving | Total carbohydrates per serving |
| totalNutrients.FAT.quantity / yield | fat_g_per_serving | Total fat per serving |
| totalNutrients.FASAT.quantity / yield | saturated_fat_g_per_serving | Saturated fat per serving |
| totalNutrients.FIBTG.quantity / yield | fiber_g_per_serving | Dietary fibre per serving |
| totalNutrients.SUGAR.quantity / yield | sugar_g_per_serving | Total sugar per serving |
| totalNutrients.NA.quantity / yield | sodium_mg_per_serving | Sodium per serving in milligrams |
| healthLabels (array) | is_vegan, is_vegetarian, … (12 binary columns) | Dietary label flags re-derived from ingredients for consistency |
| mealType (array) | meal_type | Mapped to breakfast / lunch / dinner / snack |
| dishType (array) | dish_type | First dish-type value retained |
| ingredientLines (array) | ingredient_lines | Pipe-separated ingredient list |

*Table 5.2: Edamam API to Unified Schema Mapping*

After merging, the combined dataset contained 449,313 recipe records. The primary data quality challenge was the dietary label columns. The Edamam API [7] returned health labels for some recipes, but these were not consistent across all records and did not include all 12 required binary flags. The Food.com dataset [37] had no pre-computed dietary labels at all. Both datasets therefore required a uniform label computation pass applied to every record in the merged dataset.

**Ingredient-Based Label Computation**

A Python-based rule engine was developed to assign each of the 12 binary dietary labels from the recipe's ingredient list. The engine operated in two passes for each label: a negative pass that searched for any blocked ingredient terms, and a positive pass that searched for explicit label-confirming phrases. A recipe was assigned a label value of 1 only if no blocked terms were found; if confirmed ingredient terms were present, the positive assignment was further validated. Ambiguous edge cases such as recipes using plant-based substitute products that contain terms like "vegan sausage" or "dairy-free cheese" were handled by a set of safe-pattern overrides that prevented false negatives.

```
# Simplified label computation rule structure (final_cleaned/Untitled.ipynb)
RULES = {
    "is_vegan": [
        "beef", "chicken", "pork", "fish", "salmon", "egg",
        "butter", "milk", "cheese", "cream", "honey", ...
    ],
    "is_dairy_free": [
        "milk", "butter", "cheese", "cream", "yogurt", "ghee", ...
    ],
    # ... one blocklist per label
}

SAFE_PATTERNS = [
    r"\b(?:vegan|dairy[- ]free)\s+(?:butter|cheese|cream)\b",
    r"\b(?:almond|oat|coconut|soy)\s+milk\b",
]

def relabel_row(row):
    text = (row["ingredient_lines"] + " " + row["recipe_name"]).lower()
    is_safe = any(re.search(p, text) for p in SAFE_PATTERNS)
    result = {}
    for label, blocked_terms in RULES.items():
        has_blocked = any(re.search(rf"\b{t}\b", text) for t in blocked_terms)
        result[label] = 0 if (has_blocked and not is_safe) else 1
    return pd.Series(result)
```

The label computation was applied iteratively across four versions of the dataset (v1 through v4), with each version refining the blocklists and safe patterns based on manual review of a sample of flagged rows. A review queue was generated at each pass by selecting rows where the assigned label appeared inconsistent with visible ingredient terms. Table 5.7 shows sample blocklist entries and positive override patterns for selected dietary labels.

| **Label** | **Blocklist (sample terms)** | **Positive override (sample patterns)** |
|---|---|---|
| is_vegan | beef, chicken, pork, fish, salmon, egg, butter, milk, cheese, honey | "vegan", "plant-based", "meatless" |

| | | |
|---|---|---|
| is_vegetarian | beef, chicken, lamb, fish, salmon, shrimp, anchovy, bacon, ham | "vegetarian", "veggie", "meatless" |
| is_pescatarian | beef, chicken, pork, lamb, turkey, duck, bacon, ham, sausage | "pescatarian", "seafood", "fish" |
| is_pork_free | pork, bacon, ham, prosciutto, sausage, salami, pepperoni, lard | (no pork terms in name/ingredients) |
| is_dairy_free | milk, butter, cheese, cream, yogurt, ghee, whey | "dairy-free", "non-dairy", "vegan" |
| is_gluten_free | flour, wheat, barley, rye, oats, breadcrumb, pasta, soy sauce | "gluten-free", "gluten free" |
| is_kidney_friendly | tomato sauce, orange juice, banana, potato, high-sodium terms | "kidney-friendly", "renal-friendly" |

*Table 5.3: Sample Label Rule Definitions per Dietary Flag*

A final structural validation pass (v5) confirmed that the merged dataset contained 449,313 records with no duplicate recipe_id values. Residual label sanity checks flagged a small number of rows where the vegan or dairy-free flag was set despite the presence of meat or dairy ingredient terms; these were reviewed and corrected. After all cleaning passes, the final recipe dataset was exported and loaded into the MySQL recipes table used by the Flux diet recommendation engine.

**Workout Dataset Collection and Preparation**

The workout exercise dataset was built from the 2024 Adult Compendium of Physical Activities [23], which is the primary peer-reviewed reference for Metabolic Equivalent of Task values, abbreviated as MET, across a wide range of physical activities. The Compendium was obtained from the official website at sites.google.com/site/compendiumofphysicalactivities in HTML format and parsed programmatically to extract activity codes, MET values, and activity descriptions.

**Compendium Parsing and Activity Selection**

The Compendium organises activities under major headings such as Bicycling, Conditioning Exercise, Dancing, Running, Sports, Walking, and Water Activities. A Python script read the raw Compendium text, applied a regular expression to locate rows in the format of heading, activity code, MET value, and description, and extracted all matching entries. Entries belonging to non-exercise headings including household activities, medical device use, spectating, and occupational tasks were excluded because they are not relevant to a fitness planning application.

```
# Compendium row extraction (Workout Dataset/scripts/build_workout_dataset.py)
ROW_RE = re.compile(
    r"^(?P<heading>[A-Za-z& -]+?)\s+(?P<code>\d{5})\s+
    (?P<met>\d+(?:\.\d+)?)\s*(?P<description>.*)$"
)

INCLUDED_HEADINGS = {
    "Bicycling", "Conditioning Exercise", "Dancing",
    "Running", "Sports", "Walking", "Water Activities", "Winter Activities"
}

EXCLUDED_PATTERNS = (
    "spectator", "household", "using crutches", "loading and/or unloading",
    "wheelchair", "walking the dog", "bird watching", ...
)
```

For each selected Compendium activity, the MET value was used to assign an intensity level: light for MET below 3.0, moderate for MET from 3.0 to 5.9, and vigorous for MET of 6.0 or above, following the WHO physical activity intensity classification [12]. Where the same exercise name appeared at multiple intensity levels (for example, Aerobics at moderate and vigorous), both entries were retained as separate rows with distinct MET values, allowing the recommendation engine to match exercises to the user's preferred intensity.

**Descriptions, Instructions, and Enrichment**

The Compendium provides activity names and MET codes but does not include exercise descriptions, technique instructions, equipment requirements, or training environment classifications. These fields were added manually or sourced from authoritative fitness references including the NHS Fitness Studio, the ACE Exercise Library, the NASM Exercise Library, and the ExRx Exercise Directory. Each exercise entry in the audit CSV records the source URL used for each field so that the provenance of every description, instruction, environment, and equipment assignment can be traced.

After enrichment, the dataset was filtered to remove entries that were too vague to be useful as specific workout recommendations such as generic entries for "Resistance Training", "Home Workout Video", or "Gym Workout" and entries involving high-risk activities such as skydiving or bungee jumping. The final workout exercise dataset contains 406 exercises across four training categories: aerobic, strength, balance, and stretching. Table 5.8 describes the columns in the final workout dataset schema.

| Column | Type | Description |
|---|---|---|
| ExerciseID | TEXT (PK) | Unique identifier (EX0001 … EX0406) |

| | | |
|---|---|---|
| ExerciseName | TEXT | Human-readable exercise name |
| Category | TEXT | Training type: aerobic, strength, balance, stretching |
| EquipmentBased | BOOLEAN | Whether the exercise requires equipment |
| Equipment | TEXT | Pipe-separated list of required equipment items |
| Environment | TEXT | Pipe-separated preferred locations (Home, Gym, Outdoor, Studio) |
| METValue | DECIMAL | Metabolic Equivalent of Task from the 2024 Compendium |
| Intensity | TEXT | light / moderate / vigorous based on MET thresholds |
| Description | TEXT | Plain-language overview of the exercise |
| Instructions | TEXT | Step-by-step technique guidance |
| TargetMuscleGroup | TEXT | Pipe-separated primary and secondary muscles targeted |
| DifficultyLevel | TEXT | Beginner / Intermediate / Advanced |

*Table 5.4: Workout Dataset Column Definitions*

The final workout dataset (workout_dataset.csv) was loaded into the MySQL exercises table at application startup via a seed script. Each row in the exercises table maps directly to one exercise entry, and the MET value stored in the database is used at runtime to estimate the calorie expenditure of each planned session using the formula: Calories = MET × body weight in kilograms × duration in hours.

### 5.4.3 User Authentication (Login and Registration)

The authentication module provides account creation, email verification, email/password login, Google OAuth login, account linking, password reset, and session management. It is implemented across authController.js and authService.js on the backend, with six Flutter screens handling the corresponding user journeys. All endpoints are rate-limited using express-rate-limit to prevent brute-force attacks.

## API Endpoints

| Method | Endpoint | Auth Required | Description |
|---|---|---|---|
| POST | /auth/register | No | Registers a new user account. Validates email format and password complexity, hashes the password with bcrypt (10 rounds), saves the user record, and sends a 6-digit OTP to the email via Gmail SMTP. |
| GET | /auth/verify-email | No | Verifies the email OTP (15-minute expiry). Marks the user account as verified on success. |
| POST | /auth/resend-verification | No | Resends the email verification OTP to the specified address, subject to rate limiting. |
| POST | /auth/login | No | Validates email and password with bcrypt.compare(). Issues a 15-minute JWT access token and a 7-day refresh token (stored in refresh_tokens table) on success. |
| POST | /auth/google | No | Verifies a Google ID Token via the Google Auth Library. Creates a new account or issues tokens for an existing account with a matching email. |
| POST | /auth/link-google | No | Links a Google identity to an existing email-registered account after the user confirms the link. |
| POST | /auth/refresh | No | Validates a stored refresh token, rotates it (DELETE old + INSERT new), and issues fresh access and refresh tokens. |
| POST | /auth/logout | Yes | Deletes the supplied refresh token from the database, effectively invalidating the session. |
| GET | /auth/me | Yes | Returns the authenticated user's public profile fields from the JWT claims. |
| POST | /auth/forgot-password | No | Sends a 6-digit password-reset OTP to the registered email address. |
| POST | /auth/verify-reset-otp | No | Verifies the password-reset OTP and returns a short-lived reset token used to authorise the password update. |
| POST | /auth/reset-password | No | Validates the reset token and updates the user's hashed password. |
| DELETE | /auth/me | Yes | Permanently deletes the authenticated user account and all associated data. |

*Table 5.5: Authentication API Endpoints*

### Database Tables

| Table | Key Columns | Purpose |
|---|---|---|
| users | id, email, password_hash, is_email_verified, google_id, auth_provider, onboarding_complete | Stores user credentials, Google identity link, and account state flags. |
| email_verification_otps | user_id, otp_code, expires_at, is_used | Stores one-time OTP codes for email verification and password reset, with a 15-minute expiry window. |
| refresh_tokens | user_id, token, expires_at, created_at | Stores active JWT refresh tokens (7-day expiry). Rotated on each use to prevent replay attacks. |

*Table 5.6: Authentication Database Tables*

### Implementation Details

#### Password Security

All passwords are hashed using the bcrypt library with a salt round factor of 10 before storage. Plain-text passwords are never logged, stored, or transmitted beyond the point of initial receipt. On login, bcrypt.compare() performs a constant-time comparison to prevent timing attacks.

```
// authService.js — password hashing on registration
const bcrypt = require('bcryptjs');
const SALT_ROUNDS = 10;
const passwordHash = await bcrypt.hash(password, SALT_ROUNDS);

// Password comparison on login
const isValid = await bcrypt.compare(password, user.password_hash);
if (!isValid) throw new AuthError('INVALID_CREDENTIALS');
```

#### JWT Token Architecture

Authentication uses a dual-token system. A short-lived JWT access token (15-minute expiry) is issued for each request and is verified by the authMiddleware on all protected routes. A long-lived refresh token (7-day expiry) is stored in the refresh_tokens database table. When the access token expires and a protected API call returns 401, the Flutter Dio interceptor automatically sends POST /auth/refresh to obtain a new token pair, using a Dart Completer-based lock to ensure that only one refresh request is sent even when multiple concurrent requests fail simultaneously.

```
// authService.js — token issuance
const jwt = require('jsonwebtoken');

```

```
function signAccessToken(userId) {
  return jwt.sign({ userId }, process.env.JWT_SECRET, { expiresIn: '15m' });
}

async function issueRefreshToken(userId) {
  const token = crypto.randomBytes(40).toString('hex');
  const expiresAt = new Date(Date.now() + 7 * 24 * 60 * 60 * 1000);
  await authRepo.insertRefreshToken({ userId, token, expiresAt });
  return token;
}
```

### Google OAuth Flow

The Google Sign-In flow uses the flutter google_sign_in SDK on the client to initiate the sign-in and receive a Google ID Token. This token is sent to POST /auth/google, where the backend verifies it using the google-auth-library package against Google's public keys. The verified payload provides the user's email, name, and Google sub (subject) identifier. If the email already belongs to an existing email-registered account, the user is prompted to link the accounts; otherwise a new account is created.

### Flutter Screens

| Screen File | Purpose |
|---|---|
| login_screen.dart | Email/password login form with inline validation and error banner integration. |
| register_screen.dart | Registration form with real-time password strength feedback. |
| verify_email_screen.dart | OTP entry screen for email verification with resend option. |
| forgot_password_screen.dart | Email entry screen for initiating password reset. |
| reset_otp_screen.dart | OTP entry screen for password reset verification. |
| reset_password_screen.dart | New password entry with confirmation and complexity rules. |
| auth_google_flow.dart | Manages the Google Sign-In SDK flow and error handling. |
| link_account_screen.dart | Handles the prompt to link a Google identity to an existing email account. |

*Table 5.7: Authentication Flutter Screens*

### 5.4.4 User Onboarding and Health Metrics Computation

The onboarding module collects user data across nine sequential, single-purpose screens before triggering health metrics computation on the backend. The nine steps are: (1) personal details, (2) body measurements, (3) activity level, (4) fitness goal and pace, (5) health conditions, (6) dietary preference, (7) food allergies, (8) meal frequency, and (9) workout preferences. Each step is a focused screen that collects one data category, keeping the flow structured and easy to navigate. Steps 5, 6, and 7 are optional multi-select screens.

**API Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| POST | /onboarding/profile | Saves the user's personal details (name, date of birth, sex, height, weight, activity level, goal, goal pace). |
| POST | /onboarding/diet | Saves dietary preferences (preference type, meal frequency), food allergies, and chronic disease flags. |
| POST | /onboarding/workout | Saves workout preferences (location, intensity, days per week, preferred session duration). |
| POST | /onboarding/complete | Marks onboarding as complete. Triggers server-side computation of all health metrics (BMI, BMR, TDEE, calorie target, macros) and saves them to user_metrics. |
| GET | /onboarding/status | Returns the current onboarding completion status and which steps have been saved. |
| GET | /onboarding/profile | Returns the saved profile for display in onboarding or profile edit screens. |
| GET | /metrics/current | Returns the latest computed health metrics and targets for the authenticated user. |

*Table 5.8: Onboarding and Metrics API Endpoints*

**Database Tables**

| Table | Key Columns | Purpose |
|---|---|---|
| user_profiles | user_id, name, date_of_birth, sex, height_cm, weight_kg, activity_level, goal, goal_pace_kg_per_week | Stores the personal and physical profile submitted during onboarding. |
| user_metrics | user_id, bmi, bmi_category, bmr, tdee, calorie_target, protein_g, carbs_g, fat_g, computed_at | Stores computed health metrics and macronutrient targets derived from the profile. |
| user_diet_preferences | user_id, dietary_preference, meal_types, chronic_diseases | Stores dietary preference type, |

| | | |
|---|---|---|
| | | selected meal types, and disease flags used by the diet engine. |
| user_food_allergies | user_id, allergy_name | Stores individual food allergy records for filtering during recipe selection. |

*Table 5.9: Onboarding and Metrics Database Tables*

**Health Metrics Computation**

All health metrics are computed server-side in metricsService.js when the user completes onboarding or updates their profile. The computation pipeline applies the following evidence-based formulas and thresholds in sequence.

| Metric | Formula / Rule | Source |
|---|---|---|
| BMI | weight (kg) ÷ height² (m²) | WHO Technical Report Series 894 [32] |
| BMR | (10 × weight) + (6.25 × height) − (5 × age) ± 5/161 | Mifflin-St Jeor equation [4] |
| TDEE | BMR × activity multiplier (1.2 to 1.9) | ACSM Guidelines for Exercise Testing and Prescription, 11th ed. [33] |
| Calorie target | TDEE ± (pace_kg/week × 7700 ÷ 7) | Hall energy-deficit heuristic [34] |
| Protein target | 1.6–2.4 g/kg body weight (goal-dependent) | ISSN position stand [5] |
| Fat target | 25–30% of calorie target | Dietary Guidelines for Americans [35] |
| Carb minimum | ≥ 130 g/day | Institute of Medicine Dietary Reference Intakes [36] |
| Safety floor | ≥ 1,200 kcal/day (women), ≥ 1,500 kcal/day (men) | Heuristic — minimum energy intake for micronutrient sufficiency |

*Table 5.10: Health Metrics Computation Formulas and Evidence Sources*

A protein feasibility check is performed after computing targets. The service queries the filtered recipe pool and estimates the maximum achievable daily protein under the user's dietary constraints. If the protein target cannot be met within the recipe pool, a FEASIBILITY_WARNING flag is returned, and the user is informed to consider adjusting their preferences before generating a plan.

```
// metricsService.js — Mifflin-St Jeor BMR [4]
function calculateBmr(weightKg, heightCm, age, sex) {
  const base = (10 * weightKg) + (6.25 * heightCm) - (5 * age);
  return sex === 'male' ? base + 5 : base - 161;
}
```

```
// ACSM activity multipliers [33], sedentary to extra active
const ACTIVITY_MULTIPLIERS = { 1: 1.2, 2: 1.375, 3: 1.55, 4: 1.725, 5: 1.9 };

function calculateTdee(bmr, activityLevel) {
  return bmr * ACTIVITY_MULTIPLIERS[activityLevel];
}
```

**Flutter Screens**

The onboarding flow is managed by onboarding_screen.dart using a PageController to step through all nine screens. Each step validates its own inputs before allowing progression. The metrics_result_screen.dart presents the computed BMI, calorie target, and macronutrient breakdown after onboarding is completed.

### 5.4.5 Diet Plan Generation and Management

The diet recommendation engine generates a personalised 7-day meal plan from the user's computed health metrics and dietary preferences. The engine runs in four sequential stages inside dietService.js, supported by five dedicated sub-service files: dietEligibilityService.js (preference and allergy filtering), dietSelectionService.js (recipe pool loading), dietRankingService.js (goal-adaptive scoring), dietDayPlanningService.js (day assembly and beam search), and dietValidationService.js (tolerance checking and feasibility analysis).

**API Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| POST | /diet/generate | Generates a personalised 7-day diet plan. Runs the four-stage pipeline (filter → assemble → score → tag) and saves the plan to the database. |
| GET | /diet/plan | Returns the user's current active diet plan with all meal slots, recipes, nutritional data, and health tags for all 7 days. |
| POST | /diet/regenerate-day | Regenerates the meal slots for a single specified day within the active plan, using the same pipeline as full plan generation. |
| GET | /diet/swap-options | Returns a ranked list of alternative recipes for a specified meal slot, filtered by preference and disease constraints, with nutritional comparison data. |
| GET | /diet/recipes/search | Searches the recipe catalogue by name or filter criteria (meal type, dietary preference, health tag). |
| GET | /diet/recipes/dish-types | Returns the distinct dish types available in the recipe catalogue for use in search filters. |

| | | |
|---|---|---|
| POST | /diet/swap-pick | Confirms a meal swap. Replaces the specified diet_plan_meal record with the chosen alternative recipe. |
| GET | /diet/recipes/:id/impact | Returns a nutritional impact preview comparing the current meal and a proposed replacement for the same day. |
| POST | /diet/recipes/:id/generate-steps | Calls the OpenAI API to generate step-by-step cooking instructions for the specified recipe on demand. |

*Table 5.11: Diet Plan API Endpoints*

**Database Tables**

| Table | Key Columns | Purpose |
|---|---|---|
| recipes | id, name, calories, protein_g, carbs_g, fat_g, fiber_g, sodium_mg, sugar_g, saturated_fat_g, dietary_preference, allergy_flags, disease_flags, dish_type | Recipe catalogue with per-serving nutritional data and compatibility flags used for filtering and scoring. |
| diet_plans | id, user_id, calorie_target, protein_g, carbs_g, fat_g, status, source_goal, source_dietary_preference, plan_start_date, created_at | One active plan record per user, storing the target values and preference snapshot at the time of generation. |
| diet_plan_meals | id, plan_id, day_number, meal_type, recipe_id, scaled_calories, scaled_protein_g, scaled_carbs_g, scaled_fat_g, health_tags | Individual meal assignments. One row per meal slot per day in the plan, with scaled nutritional values and health tags. |

*Table 5.12: Diet Plan Database Tables*

**Four-Stage Generation Pipeline**

**Stage 1 — Recipe Pool Filtering**

The engine first builds a filtered recipe pool from the full recipes table. Parameterised SQL queries select only recipes whose dietary_preference flag matches the user's preference (e.g., vegetarian, vegan, standard) and whose allergy_flags do not include any of the user's declared allergens. For users with chronic disease conditions (hypertension, hyperlipidaemia, or type 2 diabetes), an additional filter applies disease-safe thresholds derived from AHA and diabetes dietary guidance [9], [10], removing recipes that exceed disease-specific nutrient caps.

**Stage 2 — Day Assembly (Standard and Beam Search)**

For users without chronic disease conditions, the engine assembles each day using standard slot fill: for each meal slot (e.g., breakfast, lunch, dinner, snack), the highest-scoring recipe is

selected from the filtered pool. For users with active disease conditions, a beam search algorithm (beam width 5) is used instead. The beam search tracks cumulative disease-relevant nutrients across all meal slots of the day, ensuring that daily totals remain within clinical budget thresholds derived from AHA and diabetes dietary guidance [9], [10] rather than per-meal caps.

```
// dietDayPlanningService.js — beam search sketch
// Each beam state tracks: selected meals so far + cumulative nutrient totals
const beam = [{ meals: [], nutrients: zeroBudget() }];

for (const slot of daySlots) {
  const candidates = getCandidatesForSlot(pool, slot);
  const expanded = [];
  for (const state of beam) {
    for (const candidate of candidates) {
      const newState = mergeState(state, candidate, slot);
      if (withinDiseaseBudget(newState.nutrients, diseaseBudgets)) {
        expanded.push(newState);
      }
    }
  }
  beam.length = 0;
  beam.push(...topN(expanded, BEAM_WIDTH)); // keep top 5
}
```

**Stage 3 — Goal-Adaptive Scoring**

Recipes are scored on a 0–100 scale using weights that adapt to the user's fitness goal. For weight loss plans, calorie proximity to the slot target (40%) and protein content (40%) are weighted highest, with minor contributions from fibre (10%) and health tag count (10%). For muscle-gain plans, protein content receives the highest weight (50%). For maintenance plans, calorie proximity and protein are weighted equally (35% each). The top-scoring recipe for each slot is selected after beam search or standard fill completes.

```
// dietRankingService.js — goal-adaptive weights (excerpt)
const GOAL_WEIGHTS = {
  lose_weight:  { calorieProximity: 0.40, protein: 0.40, fibre: 0.10, tags: 0.10 },
  build_muscle: { calorieProximity: 0.25, protein: 0.50, fibre: 0.10, tags: 0.15 },
  maintain:     { calorieProximity: 0.35, protein: 0.35, fibre: 0.15, tags: 0.15 },
};
```

**Stage 4 — FDA Health Tags and Plan Saving [22]**

After assembly, each recipe in the plan is evaluated against FDA food labelling thresholds [22] to assign health tags. Tags assigned include: High Protein (≥ 10g per serving), High Fibre (≥ 5g), Low Sodium (≤ 140mg), Low Saturated Fat (≤ 1g), Low Sugar (≤ 5g), and Heart-Healthy (meets both Low Sodium and Low Saturated Fat criteria). The plan is saved to diet_plans and

all meal slot records are saved to diet_plan_meals. Any previously active plan for the user is deactivated atomically within the same database transaction.

### Swap Flow with Impact Preview

Users can request alternative recipes for any meal slot. GET /diet/swap-options returns a ranked list of alternatives that pass all preference, allergy, and disease filters. Before confirming a swap, GET /diet/recipes/:id/impact returns a side-by-side nutritional comparison of the current meal and the proposed replacement for the same day, including the delta in calories, protein, carbohydrates, and fat. This impact preview is shown in the impact_preview_screen.dart screen and must be reviewed by the user before POST /diet/swap-pick is called to commit the change.

### Flutter Screens

| Screen File | Purpose |
|---|---|
| recipe_recommendation_screen.dart | Displays the generated 7-day meal plan with per-day meal slots, health tags, and macronutrient summaries. |
| diet_plan_view.dart | Detailed per-day view of the diet plan with individual recipe cards and nutritional breakdown. |
| recipe_search_screen.dart | Recipe search and browse screen for manual recipe exploration. |
| swap_bottom_sheet.dart | Bottom sheet listing alternative recipes for a selected meal slot with ranking and nutritional info. |
| impact_preview_screen.dart | Side-by-side nutritional comparison of the current meal and the proposed swap before confirmation. |

*Table 5.13: Diet Plan Flutter Screens*

### 5.4.6 Workout Plan Generation and Management

The workout recommendation engine generates a personalised weekly workout plan based on the user's fitness goal, training location (gym or home), intensity preference, days per week, and preferred session duration. The engine runs in six stages inside workoutService.js, supported by dedicated sub-services: workoutEligibilityService.js, workoutSelectionService.js, workoutRankingService.js, workoutConstructionService.js, workoutEstimateService.js, and workoutValidationService.js.

**API Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| POST | /workout/generate-preview | Returns a preview of the workout plan configuration (session distribution, estimated total minutes) without saving it, for user review before commitment. |
| POST | /workout/generate | Generates and saves a personalised weekly workout plan based on goal, preferences, and available equipment. |
| GET | /workout/plan | Returns the user's current active workout plan with all sessions, exercises, prescriptions, calorie estimates, and goal-support assessment. |
| GET | /workout/swap-options | Returns ranked alternative exercises for a specified session slot, filtered by session type and equipment. |
| GET | /workout/exercises/search | Searches the exercise catalogue by name, session type, muscle target, or equipment. |
| GET | /workout/swap-impact | Returns a calorie and prescription comparison between the current exercise and a proposed replacement. |
| POST | /workout/swap-pick | Confirms an exercise swap, replacing the specified workout_plan_exercise record with the chosen alternative. |

*Table 5.14: Workout Plan API Endpoints*

## Database Tables

| Table | Key Columns | Purpose |
|---|---|---|
| exercises | id, name, category, session_type, primary_muscle, equipment, difficulty, met_value, is_compound | Exercise catalogue with MET values for calorie estimation and difficulty/equipment filters for selection. |
| workout_plans | id, user_id, goal, days_per_week, intensity_preference, session_duration_min, location, goal_support_status, goal_support_message, created_at | One active plan record per user, storing preferences and the computed goal-support assessment. |
| workout_plan_exercises | id, plan_id, session_number, exercise_order, exercise_id, sets, reps, duration_sec, rest_sec, estimated_calories, exercise_role | Individual exercise assignments per session, with structured prescriptions and calorie estimates. |

*Table 5.15: Workout Plan Database Tables*

**Six-Stage Generation Pipeline**

**Stage 1 — Session Distribution**

The engine determines how many sessions of each type (strength, cardio, flexibility, balance) to schedule across the user's chosen days per week and goal. A lookup table keyed by goal and days-per-week maps to a distribution (e.g., muscle-gain users training 4 days receive 3 strength sessions and 1 cardio session). ACSM guidelines [33] are applied to ensure adequate rest between consecutive strength sessions.

**Stage 2 — Exercise Pool Filtering**

For each session type, the exercise catalogue (exercises table) is filtered by session type compatibility, available equipment (gym or home), and difficulty level (matched to the user's intensity preference). Compound exercises are flagged separately via the is_compound column to ensure they are always selected first in Stage 3.

**Stage 3 — Exercise Selection (Compound First)**

For each session, compound exercises are selected first to fill primary slots, with accessory exercises filling the remaining capacity. The total exercise count per session is scaled by the session duration: shorter sessions (30 minutes) include fewer exercises while longer sessions (60 minutes) include more, using a lookup table keyed by session type and duration.

```
// workoutService.js — exercise count by session type and duration
const EXERCISE_COUNTS = {
  strength:    { 30: 4, 45: 6, 60: 8 },
  cardio:      { 30: 2, 45: 3, 60: 4 },
  flexibility: { 30: 5, 45: 7, 60: 9 },
};

function getExerciseCountForSession(sessionType, durationMin) {
  const table = EXERCISE_COUNTS[sessionType] || EXERCISE_COUNTS.strength;
  return nearestDurationKey(table, durationMin);
}
```

**Stage 4 — Structured Prescriptions**

Each selected exercise receives a structured prescription generated by buildStructuredExercisePrescription(). Strength exercises receive sets × reps (e.g., 3×10) scaled by the intensity preference. Cardio exercises receive a duration in minutes. Flexibility and balance exercises receive rounds × hold duration. Rest periods between sets are also

prescribed based on intensity (lighter intensity = longer rest to support aerobic recovery; heavier intensity = shorter rest for strength adaptation).

**Stage 5 — MET-Based Calorie Estimation**

Calorie burn for each exercise is estimated using the ACSM net-MET formula [33] applied to the exercise's stored Compendium MET value [11]. The formula computes net calories, meaning the energy expended above resting metabolic rate, using the user's body weight and the exercise duration from the prescription.

```
// workoutEstimateService.js — ACSM net-MET calorie formula [33]
// ACSM Guidelines for Exercise Testing and Prescription, 11th ed. [33]
// Compendium of Physical Activities [11]
function calculateNetExerciseCalories(durationMin, metValue, weightKg) {
  const netMet = metValue - 1; // subtract resting MET of 1.0
  return (netMet * 3.5 * weightKg * durationMin) / 200;
}
```

**Stage 6 — Goal-Support Assessment**

After the full plan is assembled, workoutValidationService.js evaluates it against evidence-based physical activity guidelines to produce a goal-support rating. For the "maintain" and "lose weight" goals, the assessment checks whether the plan meets WHO physical activity guidelines [12] of ≥ 150 minutes per week of moderate-intensity aerobic activity and ACSM recommendations [33] of ≥ 2 strength sessions per week. For "build muscle", a higher weekly volume is required. The result is one of three statuses: Strong, Moderate, or Limited, displayed with an explanatory message in the app.

**Swap Flow with Impact Preview**

Exercise swaps follow the same pattern as diet swaps. GET /workout/swap-options returns alternative exercises filtered by session type and equipment. GET /workout/swap-impact provides a calorie and prescription comparison before confirmation. The workout_impact_preview_screen.dart displays the side-by-side comparison, and POST /workout/swap-pick commits the replacement.

## Flutter Screens

| Screen File | Purpose |
|---|---|
| workout_recommendation_screen.dart | Displays the generated weekly workout plan with sessions, exercise cards, prescriptions, and calorie estimates. |
| workout_plan_screen.dart | Detailed session view with exercise list, sets/reps or duration, rest periods, and cumulative session calorie burn. |
| workout_search_screen.dart | Exercise catalogue search and browse screen. |
| workout_swap_action_sheet.dart | Bottom sheet listing alternative exercises for a selected session slot. |
| workout_impact_preview_screen.dart | Side-by-side comparison of current exercise and proposed replacement before confirming the swap. |
| workout_plan_support_dialog.dart | Dialog displaying the goal-support rating and explanation. |

*Table 5.16: Workout Plan Flutter Screens*

### 5.4.7 AI Meal Image Scanning

The AI meal image scanning feature allows users to photograph a meal and receive an estimated nutritional breakdown without manual data entry. The feature is implemented in mealScanService.js on the backend and log_meal_scan_screen.dart on the Flutter client. It integrates with the OpenAI Chat Completions API using the GPT-5.5 vision model.

#### API Endpoint

| Method | Endpoint | Description |
|---|---|---|
| POST | /log/meals/scan | Accepts a meal image as multipart/form-data. Validates the image format, sends it to the OpenAI vision API, normalises the response, and returns a structured nutrition draft for user review. |

*Table 5.17: AI Meal Scan Endpoint*

#### Implementation Details

#### Image Validation

The backend uses Multer with a memory storage adapter to receive the uploaded image. Before sending the image to OpenAI, mealScanService.js validates both the declared MIME type (must be one of image/jpeg, image/png, image/webp, image/gif) and the actual binary buffer to ensure the file is a valid image. Invalid files are rejected with a 400 Validation Error before any API call is made.

#### OpenAI Vision API Integration

The validated image is converted to a base64-encoded data URL and sent to the OpenAI Chat Completions API as an image_url content block within a user message. The prompt instructs the model to analyse only visible edible items, estimate portion sizes in grams, and return a strict JSON object. The response_format is set to json_object to reduce parsing failures. No personally identifiable information is included in the prompt.

```
// mealScanService.js — OpenAI vision prompt structure
const payload = {
  model: 'gpt-5.5',
  response_format: { type: 'json_object' },
  messages: [{
    role: 'user',
    content: [
      { type: 'text', text: MEAL_SCAN_PROMPT },
      { type: 'image_url', image_url: { url: base64DataUrl, detail: 'low' } },
    ],
  }],
  max_tokens: 400,
```

```
};

// Required JSON fields in the model response:
// mealName, mealType, calories, proteinG, carbsG, fatG,
// confidence (low/medium/high), assumptions (string[])
```

**Response Normalisation**

The raw JSON response from OpenAI is passed through normalizeDraft() in mealScanService.js, which enforces the following constraints: all numeric fields (calories, proteinG, carbsG, fatG) must be finite non-negative numbers; mealType is coerced to one of the four valid enum values (breakfast, lunch, dinner, snack); confidence is coerced to low/medium/high; and the assumptions array is capped at five items. If the model returns an unparseable response, a SCAN_PARSE_ERROR is raised and the user is prompted to enter nutritional values manually.

**Flutter Screen**

The log_meal_scan_screen.dart allows the user to capture a photo using the device camera or select one from the gallery. After the scan, the returned draft is displayed in editable fields so the user can adjust any value before tapping Log Meal. A confidence badge (Low / Medium / High) and a list of AI assumptions are shown below the draft to communicate the estimated accuracy to the user.

### 5.4.8 Daily Logging (Meal, Exercise, Water, and Weight)

The logging module enables users to record their daily food intake, physical activity, water consumption, and body weight. All logging operations are handled by logController.js and logService.js. The module supports three entry paths for both meal and exercise logging: planned entries (from the active diet or workout plan), manual custom entries, and saved custom templates for quick re-logging of frequently consumed meals or activities.

**API Endpoints**

| Method | Endpoint | Description |
|---|---|---|
| POST | /log/meals | Logs a planned meal entry (from diet plan) or a custom meal entry (manual or AI scan result) to meal_logs. |
| GET | /log/meals | Returns the authenticated user's meal log entries for a specified date, with daily nutrition totals. |
| PATCH | /log/meals/:id | Updates an existing meal log entry (e.g., corrects calories or serving size). |
| DELETE | /log/meals/:id | Removes a meal log entry. |
| GET | /log/saved-meals | Returns the user's library of saved custom meal templates. |
| POST | /log/saved-meals | Creates a new saved custom meal template for reuse. |
| POST | /log/saved-meals/:id/log | Logs a saved custom meal template directly without re-entering nutritional values. |
| POST | /log/workouts | Logs a planned workout entry (from workout plan, with MET-based calorie calculation) or a custom activity entry (with AI calorie estimation). |
| GET | /log/workouts | Returns the user's workout log entries for a specified date. |
| PATCH | /log/workouts/:id | Updates an existing workout log entry. |
| DELETE | /log/workouts/:id | Removes a workout log entry. |
| POST | /log/saved-workouts/:id/log | Logs a saved custom workout template directly. |
| POST | /log/water/increment | Adds a specified volume (mL) to the user's water intake for the current day. |
| POST | /log/water/decrement | Subtracts a specified volume from the user's water intake. |
| GET | /log/water | Returns the current day's total water intake for the user. |
| POST | /log/weight | Records a new weight measurement. Triggers a recalculation of health metrics using the new weight. |
| GET | /log/weight | Returns the user's weight log history. |

| | | |
|---|---|---|
| GET | /log/weight/trend | Returns a smoothed weight trend line for the progress chart. |

*Table 5.18: Logging API Endpoints*

### Database Tables

| Table | Key Columns | Purpose |
|---|---|---|
| meal_logs | id, user_id, log_date, log_time, meal_type, source, is_ai_scan, meal_name, calories, protein_g, carbs_g, fat_g, diet_plan_meal_id | One row per meal log entry. The source column distinguishes planned meals from custom entries. The diet_plan_meal_id links planned entries back to the diet plan. |
| exercise_logs | id, user_id, log_date, log_time, source, exercise_name, duration_min, sets, reps, estimated_calories, met_value, workout_plan_exercise_id | One row per exercise log entry. Links planned entries to the workout plan via workout_plan_exercise_id. |
| water_logs | id, user_id, log_date, amount_ml | Stores the cumulative daily water intake. Updated via increment/decrement operations. |
| weight_logs | id, user_id, log_date, weight_kg | Stores individual weight measurements. Each new entry triggers a metrics recalculation. |
| saved_custom_meals | id, user_id, meal_name, meal_type, calories, protein_g, carbs_g, fat_g, created_at | User-specific library of reusable custom meal templates. |
| saved_custom_workouts | id, user_id, exercise_name, duration_min, estimated_calories, notes, created_at | User-specific library of reusable custom workout templates. |

*Table 5.19: Logging Database Tables*

### Meal Logging Implementation

Meal logging supports three distinct entry paths, distinguished by the source field in the request payload.

- Planned meal entry: The user selects a meal from their active diet plan by its diet_plan_meal_id. The backend retrieves the nutritional values from the plan record and inserts a meal_log row with source = "plan".
- Custom meal entry: The user manually enters or adjusts nutritional values (from memory, packaging, or an AI scan draft). The backend inserts a meal_log row with source = "custom". If the entry originated from an AI scan, isAiScan is set to true for traceability.

– Saved custom meal: The user selects from their saved custom meals library and logs it directly without re-entering values. The backend copies the template values to a new meal_log row.

**Exercise Logging and Custom Activity Calorie Estimation**

Exercise logging similarly supports three paths. Planned exercise entries reference a workout_plan_exercise_id and use the stored MET value [11] and prescription duration to compute the calorie estimate on the server using the ACSM net-MET formula [33]. For custom free-form activity entries, the user enters the activity name and duration. The backend calls workoutEstimateService.js, which searches the exercises table for a matching activity to retrieve its MET value. If no match is found, the OpenAI API is called with the activity description and duration to estimate a MET value, which is then used in the same formula.

**Weight Logging and Metrics Update**

When a user logs a new weight measurement via POST /log/weight, logService.js persists the weight_log record and then calls refreshLatestWeightProfileMetrics() to update the user's profile weight and recompute all health metrics (BMR, TDEE, calorie target, macros) using the new body weight. This ensures that calorie and macro targets remain aligned with the user's most recent measurements throughout their journey.

**Flutter Screens**

| Screen File | Purpose |
|---|---|
| log_tab.dart | Main logging hub showing today's meals, exercises, water intake, and nutrition ring summaries. |
| log_meal_screen.dart | Meal logging screen for selecting planned meals or initiating custom entry. |
| log_meal_custom_screen.dart | Manual nutritional value entry form for custom meal logging. |
| log_meal_scan_screen.dart | AI meal scan screen (camera capture and scan result review). |
| log_saved_custom_meals_screen.dart | Saved custom meal library for quick re-logging. |
| log_workout_screen.dart | Exercise logging screen for selecting planned exercises. |
| log_workout_custom_screen.dart | Custom activity entry form with AI calorie estimation. |

| | |
|---|---|
| log_saved_custom_workouts_screen.dart | Saved custom workout library for quick re-logging. |
| log_water_screen.dart | Water intake tracking screen with increment/decrement controls. |
| log_weight_screen.dart | Body weight entry screen. |

*Table 5.20: Logging Flutter Screens*

### 5.4.9 Progress Tracking and AI Coaching Insights

The progress module aggregates the user's logged data into weekly summaries and visualizations and generates personalized AI coaching insights using the OpenAI Chat Completions API. The module is implemented in progressController.js, progressService.js, and progressInsightService.js.

#### API Endpoints

| Method | Endpoint | Description |
|---|---|---|
| GET | /progress/today | Returns a summary of today's logged calories, macros, water intake, and completion status relative to targets. |
| GET | /progress/weekly-summary | Returns aggregated weekly statistics including average daily calories, protein, workout days, total workout minutes, and adherence percentages. |
| GET | /progress/calorie-history | Returns daily calorie totals for the past N days for the calorie trend chart. |
| GET | /progress/macro-history | Returns daily protein, carbohydrate, and fat totals for the macro trend chart. |
| GET | /progress/workout-frequency | Returns per-day workout session counts for the workout frequency chart. |
| GET | /progress/goal-timeline | Returns the estimated goal completion date based on current pace and goal target. |
| POST | /progress/goal-completion/confirm | Records goal completion and resets the goal timeline for a new cycle. |
| GET | /progress/weekly-insights | Fetches the current week's AI coaching insights. Generates new insights via OpenAI if no cached insights exist for the week. |

*Table 5.21: Progress and Insights API Endpoints*

#### Weekly Data Aggregation

The weekly summary is computed by progressService.js by aggregating the user's meal_logs, exercise_logs, water_logs, and weight_logs for the current ISO week. The aggregation calculates average daily calories and protein compared to targets, the number of days on which nutrition was logged, the number of workout sessions and total workout minutes, and adherence percentages for both diet and workout plans. These metrics are used both for display in the progress tab and as inputs to the AI insight generation prompt.

#### AI Coaching Insights Generation

Weekly coaching insights are generated by progressInsightService.js using the OpenAI Chat Completions API. The service builds a structured prompt containing only the user's aggregated

metrics (no personally identifiable information) and instructs the model to return exactly three short, actionable, non-medical insights as a JSON object. The prompt explicitly constrains the model to: use only the provided numbers, not infer unstated conditions or diagnoses, and acknowledge limited logging if the user logged on fewer than four days.

```
// progressInsightService.js — prompt construction (excerpt)
const prompt = [
  `Goal: ${goalLabel}`,
  `Avg daily calories: ${avgCalories} kcal (target: ${calorieTarget})`,
  `Avg daily protein: ${avgProtein}g (target: ${proteinTarget}g)`,
  `Nutrition logged: ${loggedDays} of 7 days`,
  `Workouts: ${workoutDays} days, ${totalMinutes} minutes total`,
  `Give exactly 3 short actionable insights as JSON:`,
  `{ "insights": ["...", "...", "..."] }`,
].join('\n');
```

Generated insights are cached in the weekly_insights database table keyed by user and ISO week number. Subsequent requests within the same week return the cached insights without calling the OpenAI API again. A disclaimer is always displayed alongside AI-generated insights to inform the user that the content is AI-generated and not professional medical or nutritional advice.

**Progress Visualizations**

The progress_tab.dart screen presents the following visualisations and metrics for the current week:

- Daily calorie bar chart showing actual versus target calories for each logged day.
- Macronutrient breakdown rings showing average protein, carbohydrate, and fat against targets.
- Workout frequency chart showing how many sessions were logged on each day of the week.
- Weight trend line chart showing recorded body weight over time.
- Goal timeline card estimating the projected date of goal achievement based on current weekly pace.
- AI insights section showing the three generated coaching insights with the disclaimer.

## 5.5 System Operation

This section presents the key user-facing screens and interaction flows for each module of the Flux application. Screenshots were captured from an iOS Simulator running iPhone 16 Pro Max. The walkthrough covers authentication, onboarding, home dashboard, diet plan, workout plan, daily logging, progress tracking, and profile management.

### 5.5.1 User Authentication

The authentication module manages user registration, login, email verification, and password recovery. These screens are the entry point to the Flux application and are presented before the main navigation is accessible.

#### Login and Registration

When a user opens Flux for the first time, they are presented with the Login screen. The screen displays a "Welcome back" heading, an email field, a password field, and a "Sign in" button. The user can enter their registered email address and password, then tap "Sign in" to access the application. A "Continue with Google" option is also available for users who prefer to sign in with their Google account. If the user does not have an account yet, they can tap the "Register" link at the bottom of the screen to navigate to the Registration screen.

On the Registration screen, the user enters their full name, email address, and a password. The screen provides a real-time password strength checker that validates four requirements as the user types: a minimum of eight characters, at least one uppercase letter, at least one number, and at least one special character. Each requirement is shown as a tick or cross that updates immediately, giving the user clear feedback before they submit the form. Once the form is valid, the user taps "Create account" to register.

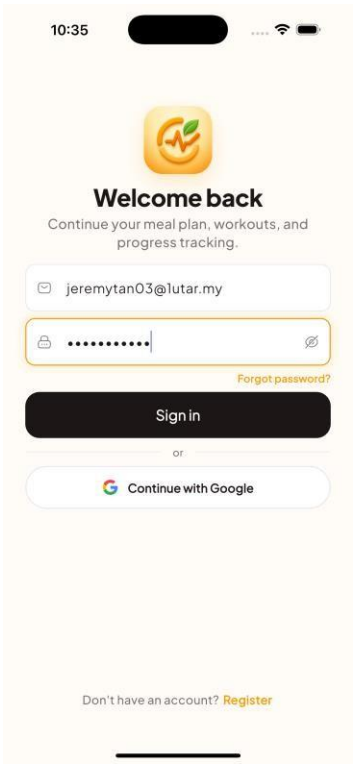


*Figure 5.1: Login Screen*

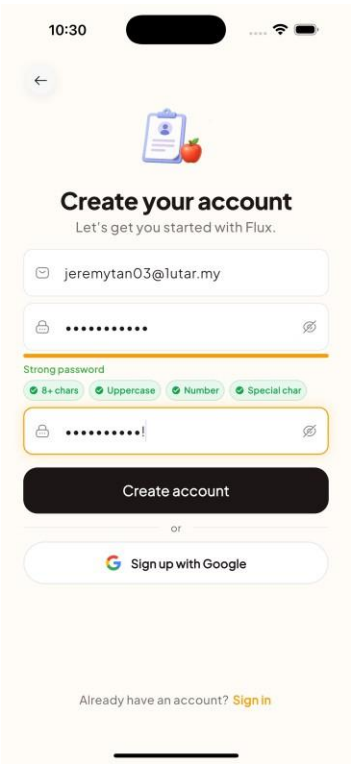


*Figure 5.2: Registration Screen*

**Email Verification**

After successful registration, the user is taken to the Email Verification screen. The screen displays a confirmation message stating that the account has been created and that a verification email has been sent. The user should open their email inbox, click the verification link, and then return to the app and tap "I've verified — continue" to proceed. If the verification email was not received, the user can tap "Resend Email" to request a new message.

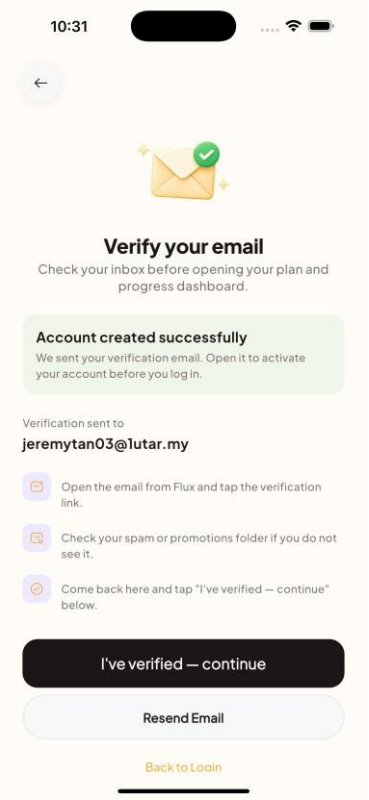


*Figure 5.3: Email Verification Screen*

**Password Reset**

If a user forgets their password, they can tap the "Forgot password?" link on the Login screen. This opens the Reset Password screen, where the user enters the email address associated with their account and taps "Send code". The system sends a six-digit one-time password (OTP) to that email address.

The OTP Verification screen presents six individual digit input boxes and a countdown timer. The user types of each digit of the code they received. If the timer expires before they enter the code, a "Resend code" link becomes available. Once the correct OTP is entered and verified, the user is taken to the Set New Password screen, which shows the same four-rule password strength indicator seen during registration. After entering a valid new password and tapping "Reset Password", the account password is updated, and the user is redirected to the Login screen.

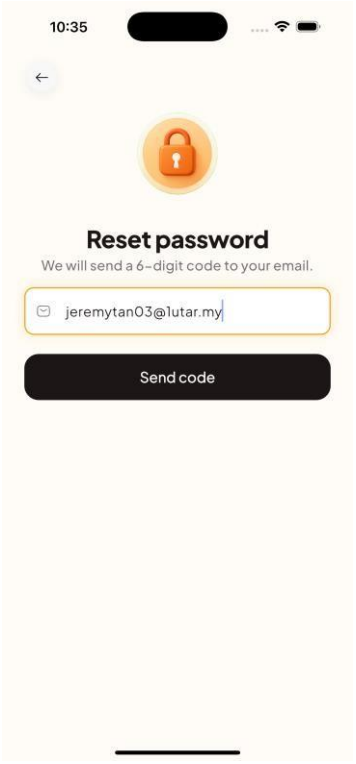


*Figure 5.4: Forgot Password Screen*

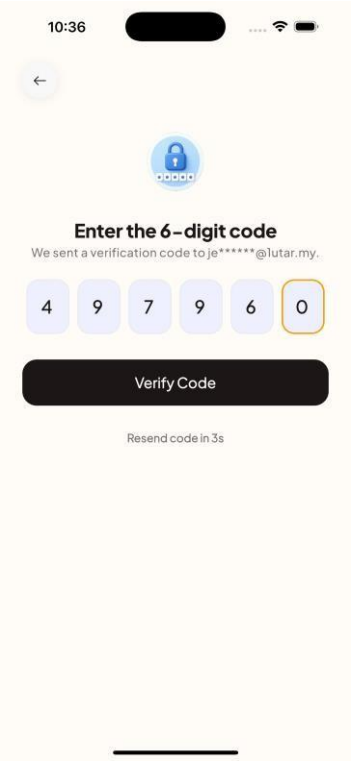


*Figure 5.5: OTP Verification Screen*

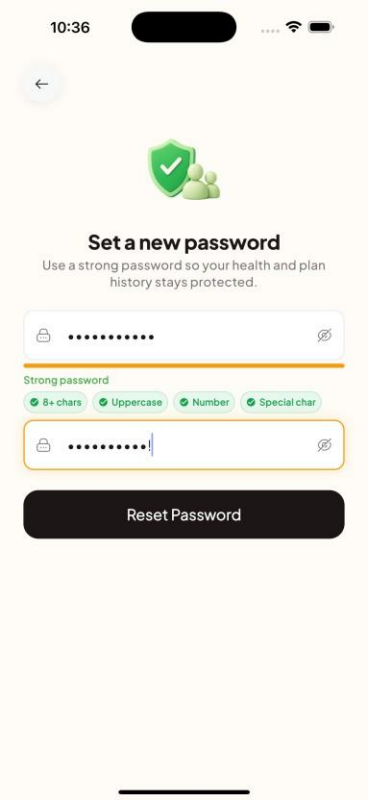


*Figure 5.6: Set New Password Screen*

### 5.5.2 User Onboarding and Health Metrics

The onboarding module collects the health data needed to personalize the user's diet and workout plans. It is a guided 14-step flow presented to first-time users immediately after email verification. Each screen has a progress indicator showing "Step X of 14" so the user always knows how far along they are.

#### Personal Information and Body Measurements

The first step asks the user to enter their full name, date of birth, and biological sex. This information is used in the Mifflin-St Jeor equation [4] to calculate the user's resting metabolic rate. The second step asks for the user's current body height and weight, which together with the previous inputs complete the baseline calculation. The user taps "Continue" after each step to advance to the next screen.


*Figure 5.7: Personal Information (Step 1)*

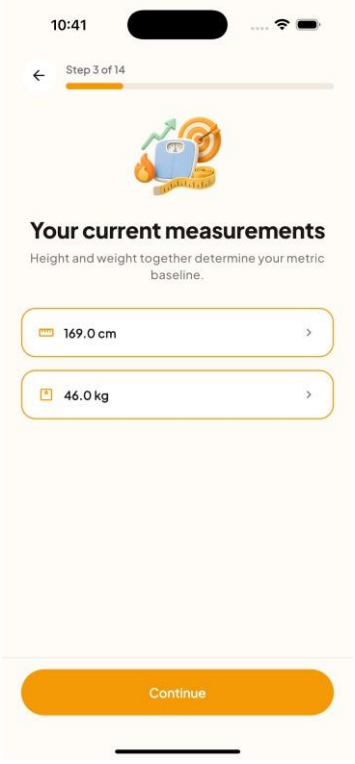


*Figure 5.8: Body Measurements (Step 2)*

**Activity Level, Goal, and Dietary Preferences**

The activity level step asks the user to describe their typical daily movement on a scale from Sedentary to Very Active. This value is used as the activity multiplier in the TDEE calculation. The following step asks for the user's primary health goal (lose weight, build muscle, or maintain) and, for weight-loss and muscle-gain goals, a target body weight. These inputs determine the calorie surplus or deficit applied to the daily calorie target.

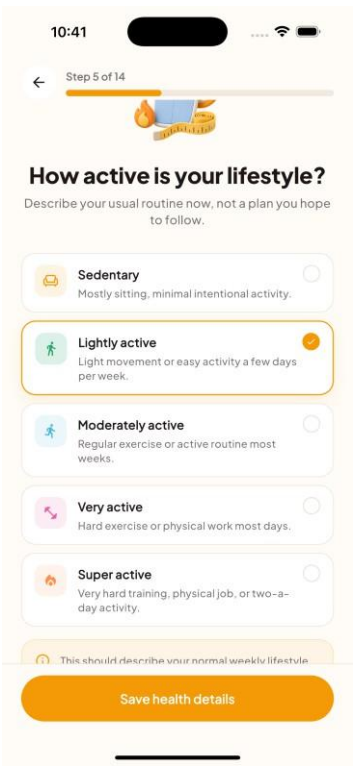


*Figure 5.9: Activity Level (Step 3)*

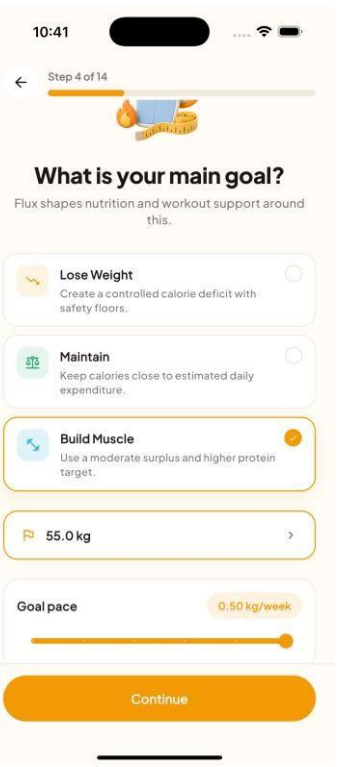


*Figure 5.10: Goal and Target Weight (Step 4)*

The dietary preference step allows the user to select a base eating style such as No Restriction, Vegan, Vegetarian, Pescatarian, Pork-free, or Kosher. Flux uses this selection to filter the recipe pool when generating the diet plan. The following step asks about chronic health conditions such as hypertension, hyperlipidemia, or type 2 diabetes. These are used to apply clinical dietary guardrails to the meal recommendations. A subsequent step asks about food allergies, so those ingredients are excluded from all suggested recipes.

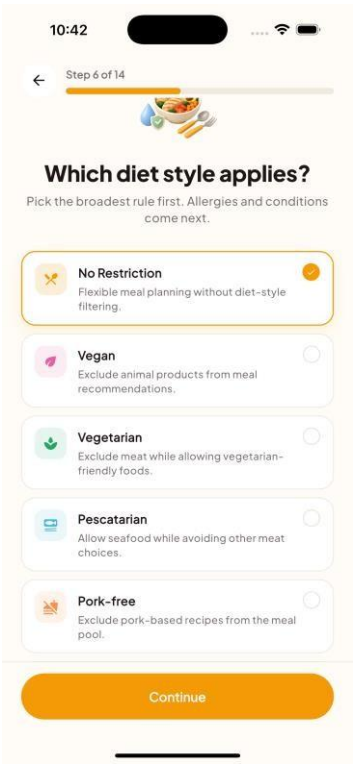


*Figure 5.11: Diet Style Selection (Step 6)*

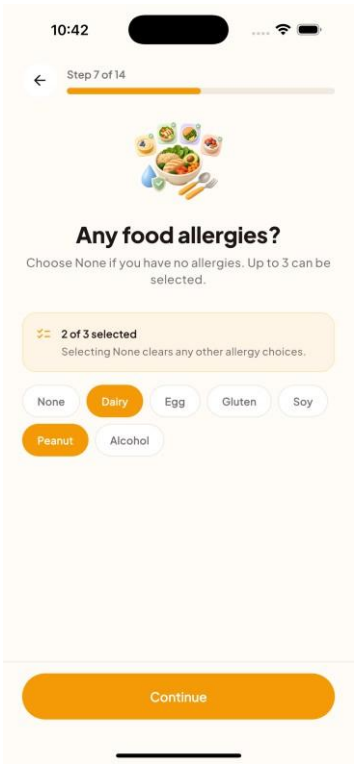


*Figure 5.12: Allergy Selection (Step 7)*

**Meal Setup and Workout Preferences**

The user then selects which meal types they plan to eat each day (breakfast, lunch, dinner, snack) and sets a daily water intake target. The number of selected meal types determines how the daily calorie budget is distributed across the plan.

The workout section starts with an opt-in step: the user can choose to include a weekly workout plan or skip workouts entirely. If workouts are enabled, the following steps collect the preferred training intensity, number of training days per week, preferred session duration, training location (gym, home, or outdoor), and available equipment. These settings are used to generate a weekly workout plan that fits the user's real environment and schedule.

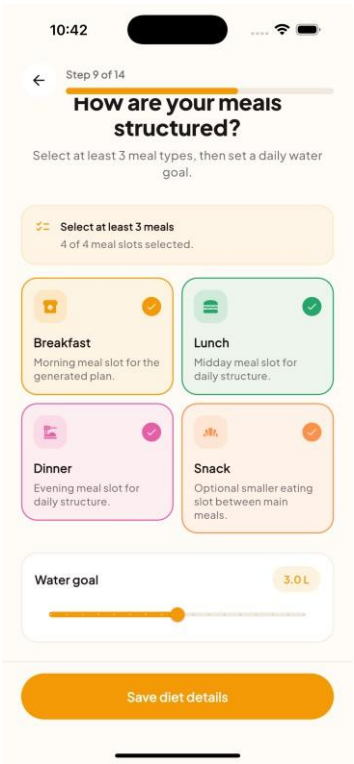


*Figure 5.13: Meal and Water Setup (Step 8)*

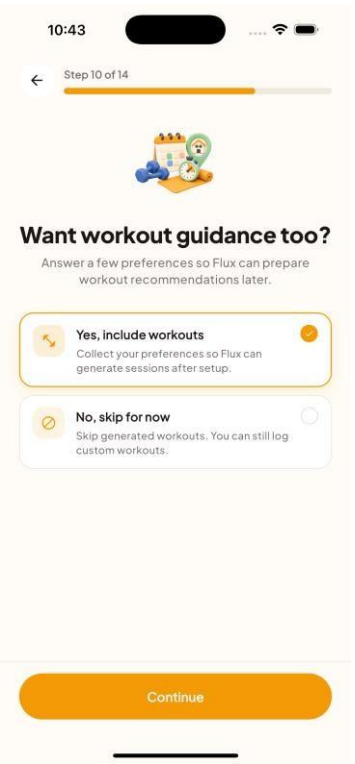


*Figure 5.14: Workout Opt-in (Step 9)*

**Review Summary and Metrics Result**

After completing all preference steps, the user is shown a Review screen that summarizes the key inputs: goal, diet style, meal types, activity level, and target weight. The user can tap "Continue" to submit all settings or go back to change any entry.

Once the onboarding is submitted, Flux calculates the user's BMI, BMR, TDEE, daily calorie target, and macronutrient targets, then displays a Metrics Result screen. This screen shows the computed daily calorie target, gram targets for protein, carbohydrates, and fat, the user's BMI classification, and an estimated timeline to reach their goal. The user taps "Go to Flux" to enter the main application.

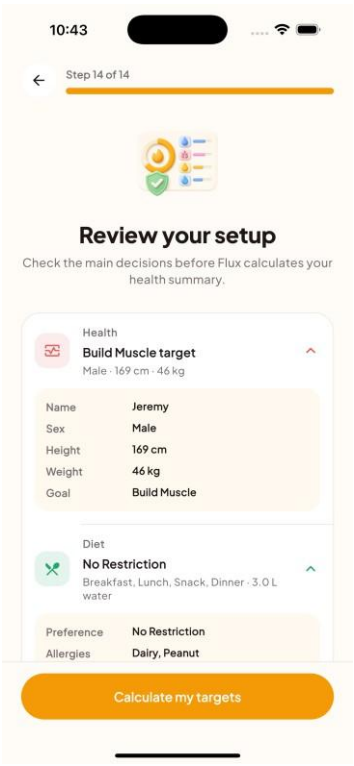


*Figure 5.15: Onboarding Review (Step 14)*

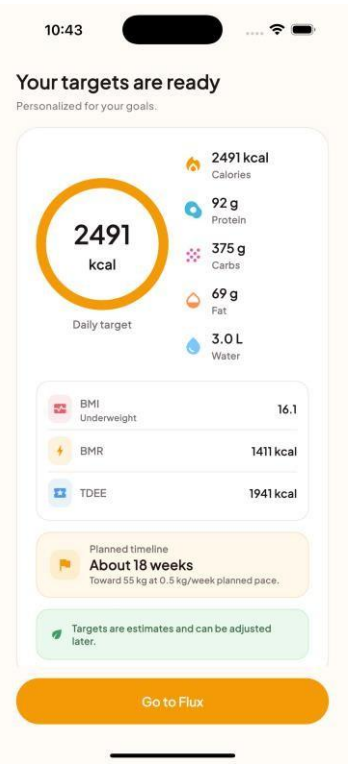


*Figure 5.16: Metrics Result Screen*

### 5.5.3 Home Dashboard

After onboarding, the user lands on the Home dashboard. This screen provides an at-a-glance summary of the current day's nutrition progress, water intake, workout status, and weight check-in. It is accessible at any time by tapping the "Home" icon in the bottom navigation bar.

The top section greets the user by name and displays a circular ring that shows how many calories have been consumed against the daily target. Below the ring, three progress bars display the protein, carbohydrate, and fat intake relative to the respective macro targets. A water section allows the user to quickly increment or decrement the day's water intake using the plus and minus buttons without leaving the Home screen.

Scrolling down, the Home screen lists the planned meals for the day, each with a shortcut to log it, and the scheduled workout sessions. A weight check-in section at the bottom prompts the user to record today's weight. The entire Home screen updates in real time as the user logs meals, workouts, and water throughout the day.

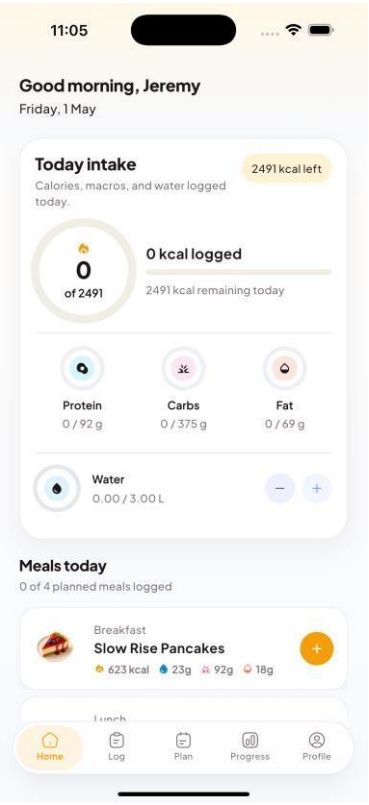


*Figure 5.17: Home Dashboard Screen*

### 5.5.4 Diet Plan Generation and Management

The diet plan module generates and displays a personalized seven-day meal plan based on the user's daily calorie target, macronutrient targets, dietary style, disease constraints, allergies, and selected meal types. It is accessed by tapping the "Plan" tab in the bottom navigation bar and then selecting "Diet plan".

#### Plan Generation

When no diet plan exists, the user sees a "Build your 7-day meal plan" screen. Three informational chips  Review first, Daily targets, and Health checks indicate that the system will review the user's saved preferences, calorie targets, and disease constraints before generating. The user can tap "Generate diet plan" to proceed. A "Review your diet setup" confirmation dialog appears, summarizing the current goal, diet style, meal types, and daily calorie and macro targets. If everything looks correct, the user taps "Generate diet plan" to confirm; otherwise, they can tap "Adjust diet preferences" to return to the diet preference editor before generating.

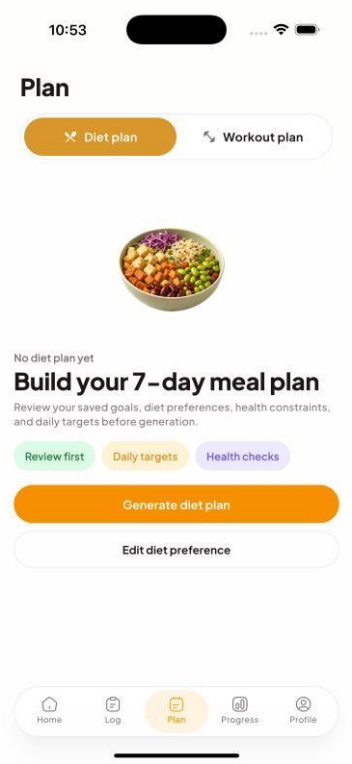


*Figure 5.18: Build Diet Plan Screen*

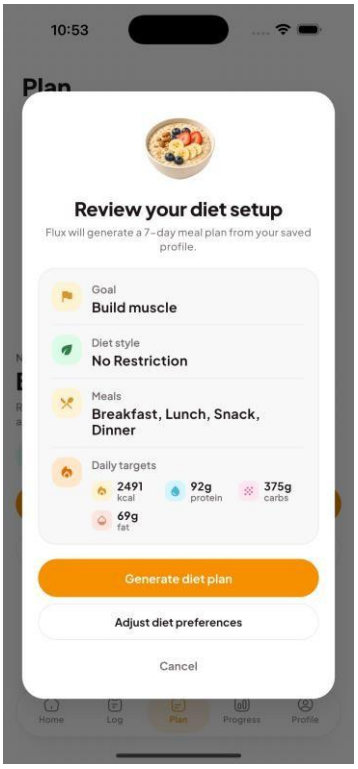


*Figure 5.19: Diet Setup Review Dialog*

**Diet Plan Overview**

Once the plan is generated, the Diet Plan Overview screen is displayed. A horizontal day selector at the top shows the seven days of the plan cycle; the user can tap any day to view its meals. The selected day shows a daily summary of the calorie target and the macro totals across all planned meals. Below the summary, each planned meal is shown as a card with the meal type label (for example, Breakfast), the recipe name, a food image, and four macro chips displaying calories, protein, carbohydrates, and fat. Each meal card has a "Swap" button the user can tap to replace that meal.

A three-dot menu icon at the top-right of the plan overview opens a "Manage diet plan" bottom sheet with three options: Regenerate Day (to refresh only the open meals for the current day), Regenerate Weekly Plan (to rebuild the full seven-day meal structure from scratch), and Force Regenerate with Latest Metrics (to rebuild the plan using the most recently saved health metrics, effective from the next day so that any already-logged meals for today are preserved).

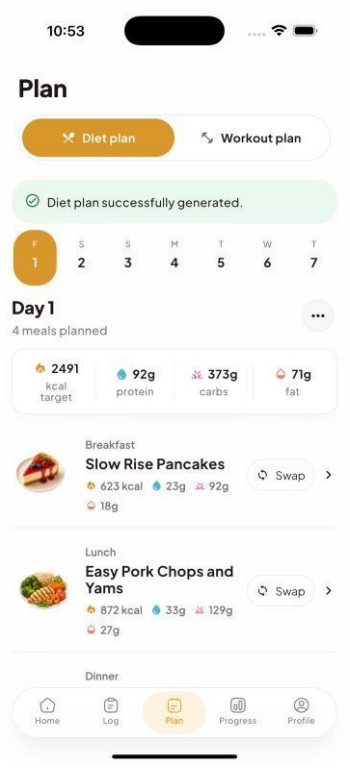


*Figure 5.20: Diet Plan Overview*

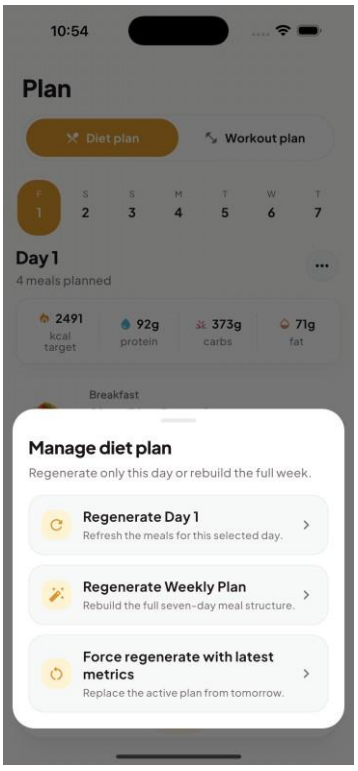


*Figure 5.21: Manage Diet Plan Options*

**Recipe Details**

Tapping on any meal card opens the Recipe Detail screen. This screen shows the recipe image, the meal type and recipe name, and four macro chips for the adjusted serving. A "Nutrition snapshot" section lists the exact grams of protein, carbohydrates, fat, fiber, sugar, and sodium. Two preview sections "Ingredients" and "How to prepare" give a brief overview; the user can tap the arrow on each to expand to the full list.

Tapping "View all" on the Ingredients section opens a dedicated Ingredients screen that lists every ingredient with a bullet point and a note indicating that amounts are based on the source recipe serving. Tapping "View all steps" on the How to prepare section opens the Preparation Steps screen with numbered cooking instructions. If a recipe has no source instructions, an AI-powered "Generate steps with AI" button is shown; tapping it calls the backend, which uses the recipe ingredients and name to generate a set of cooking steps, displayed after a brief loading period with a "Preparation steps generated. Review before cooking." banner.

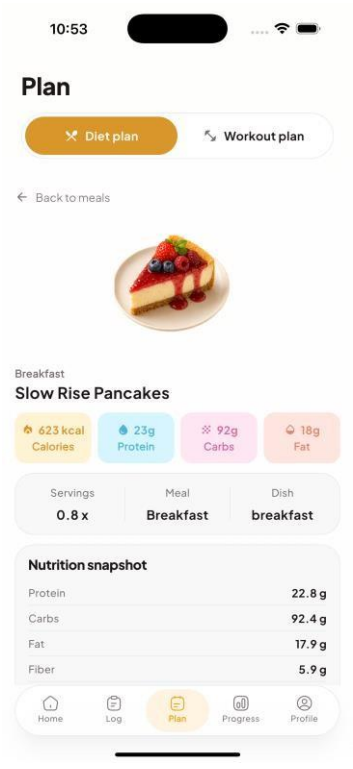


*Figure 5.22: Recipe Detail Screen*

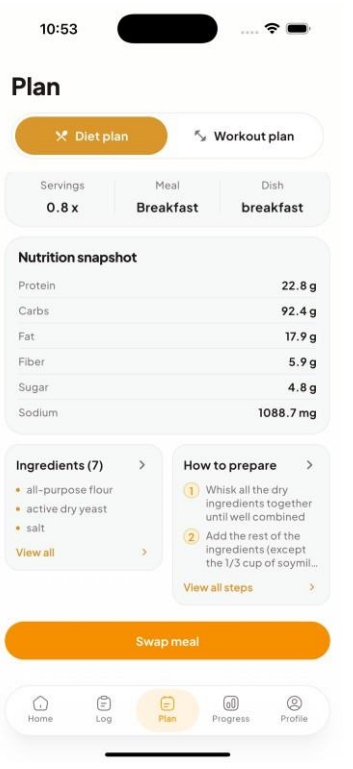


*Figure 5.23: Recipe Detail (Nutrition and Swap)*

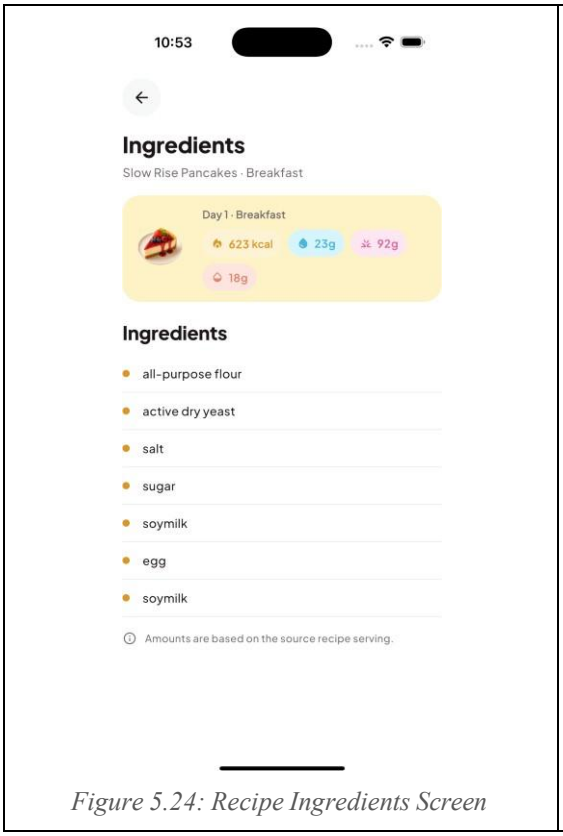


*Figure 5.24: Recipe Ingredients Screen*

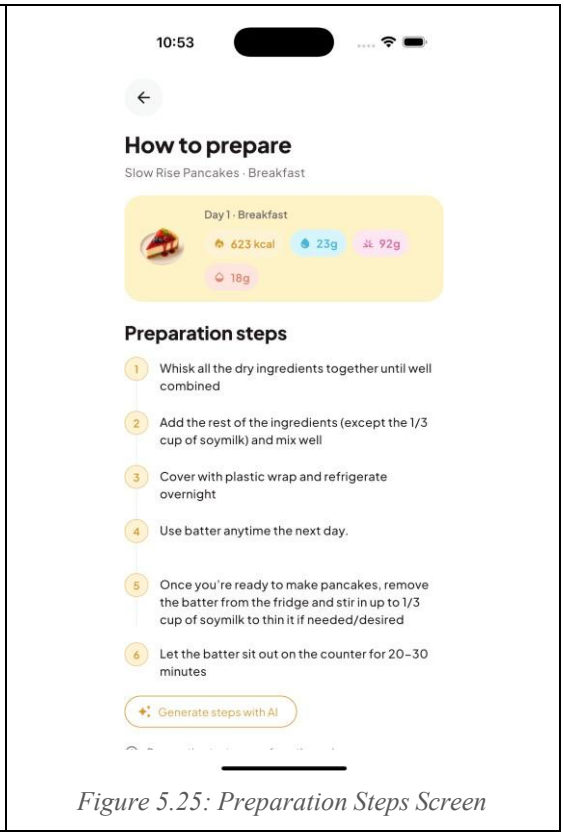


*Figure 5.25: Preparation Steps Screen*


*Figure 5.26: AI-Generated Preparation Steps*

**Meal Swap Flow**

To change a planned meal, the user taps the "Swap" button on any unlogged meal card. A "Change [meal type]" bottom sheet appears with two options: Recommended Swaps (best-fit alternatives selected by the recommendation engine) and Search & Pick (which lets the user browse the full recipe dataset). Both options show the impact preview before the user confirms any change.

The Recommended Swaps screen displays the current meal at the top and lists the best matching alternatives below it, each showing its macro chips and a "Best fit" badge. The Search & Pick screen provides a search bar, meal-type filter chips, and dish-type filter chips; results are sorted with the best-fit recipes shown first. When the user selects a recipe from either screen, the Impact Preview screen appears, showing a before/after/target comparison table for all four macros. If the swap keeps all macros within the daily targets and any disease guardrails, a "Plan-safe before confirm" banner is displayed. The user taps "Confirm swap" to apply the change or "Cancel" to go back.

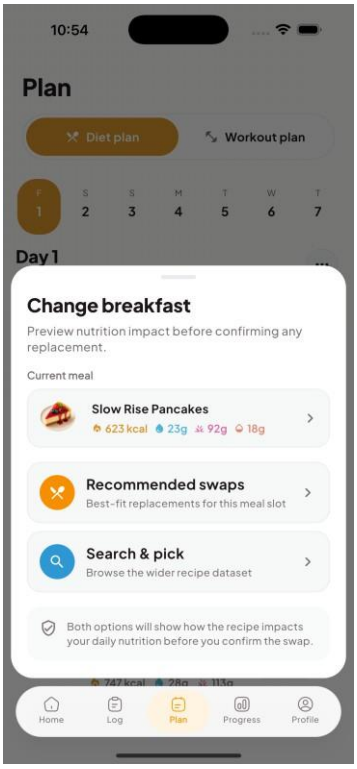


*Figure 5.27: Change Meal Bottom Sheet*

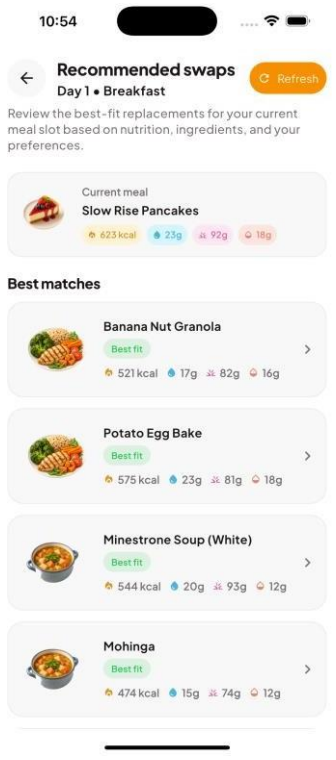


*Figure 5.28: Recommended Swaps Screen*

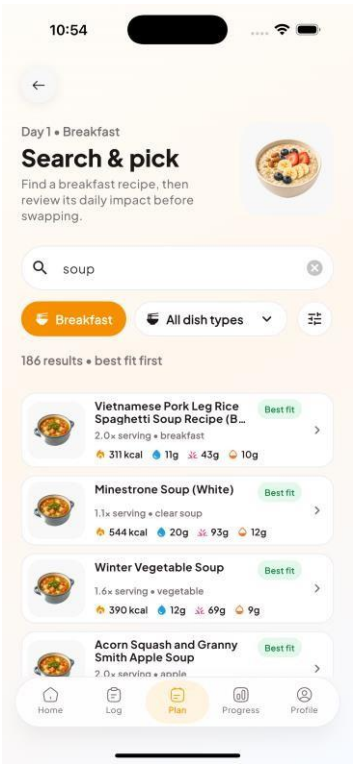


*Figure 5.29: Search and Pick Screen*

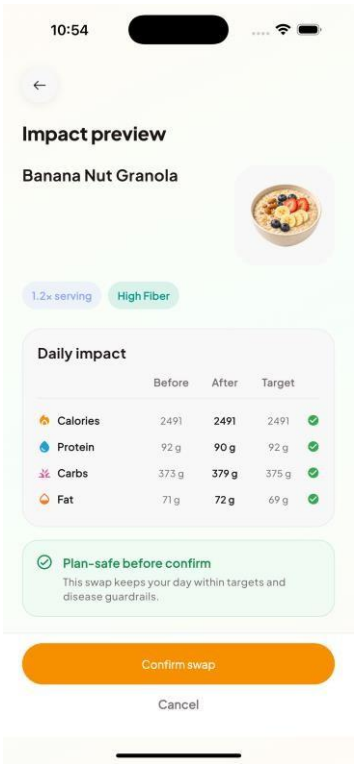


*Figure 5.30: Swap Impact Preview Screen*

### 5.5.5 Workout Plan Generation and Management

The workout plan module generates and displays a personalized weekly training plan based on the user's goal, preferred training intensity, number of days per week, session duration, location, and available equipment. It is accessed by tapping the "Plan" tab in the bottom navigation bar and then selecting "Workout plan".

#### Plan Generation

When no workout plan exists, the user sees a "Build your weekly training plan" screen. Three informational chips Goal support, Equipment fit, and Session time indicate the factors the recommendation engine will use. The user taps "Generate weekly plan" to start the generation process.

Before the plan is created, a confirmation dialog appears showing the plan quality assessment: a support rating (for example, Strong), the total number of sessions, the total training time in minutes, and the estimated weekly energy expenditure in kilocalories. A "Why this matters" explanation describes how the plan aligns with the user's goal. If the user is satisfied, they tap

"Generate Anyway" to confirm; otherwise, they can tap "Adjust Preferences" to modify their workout settings first.

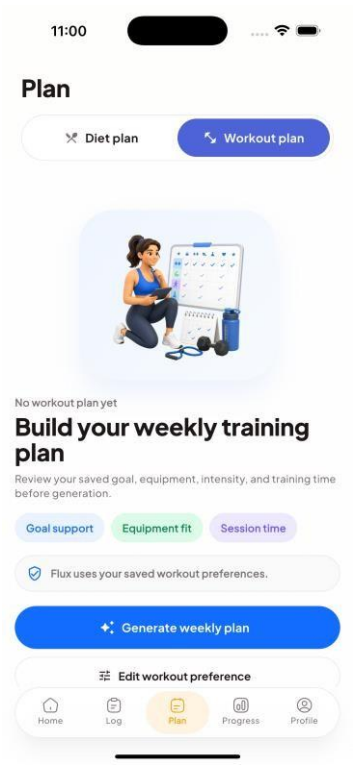


*Figure 5.31: Build Workout Plan Screen*

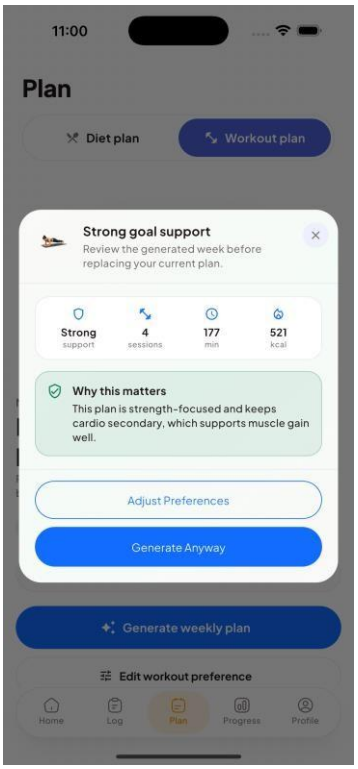


*Figure 5.32: Workout Generation Confirmation*

**Workout Plan Overview**

Once generated, the Workout Plan Overview screen shows a tab bar at the top with one tab for each scheduled session (S1, S2, S3, etc.), labelled with the session type. A weekly summary section below displays four key numbers: total sessions, total time, total estimated energy expenditure, and the goal support rating.

A support-quality banner explains the rating in plain language, for example, "This plan is strength-focused and keeps cardio secondary, which supports muscle gain." Below this, the primary exercise for the selected session is displayed with its sets, reps, estimated time, and estimated calories. Scrolling down reveals the accessory exercises for the session. At the bottom of the screen, "Regenerate Weekly Plan" and "Force regenerate with latest metrics" buttons allow the user to rebuild the plan without navigating away.

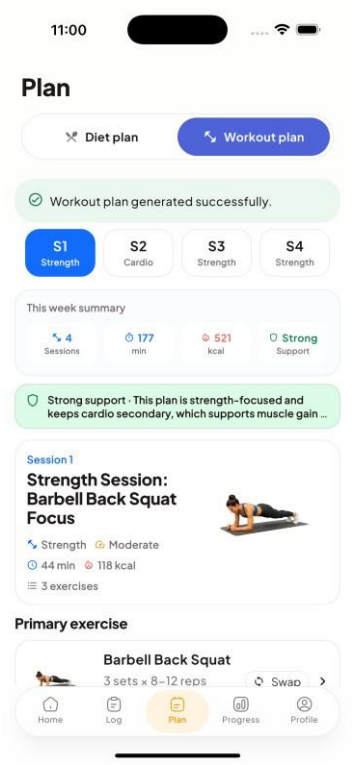


*Figure 5.33: Workout Plan Overview*

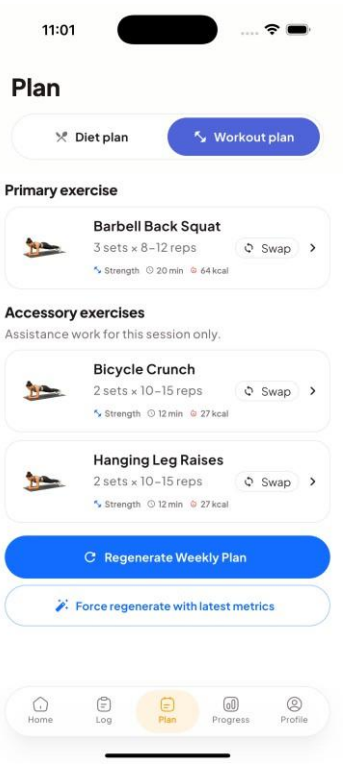


*Figure 5.34: Workout Plan (Exercises)*

**Exercise Detail**

Tapping on any exercise opens the Exercise Detail screen. This screen shows the exercise name, the set and repetition prescription (for example, 3 sets × 8–12 reps), the recommended rest time, and five attribute chips: training type, effort level, estimated time, estimated calories, and training environment. A "Prescription" section presents the sets, reps, and rest time in a clear three-column layout. A "How to perform" section provides numbered step-by-step technique guidance. A "Muscles and equipment" section lists the target muscle groups, required equipment, and training environment. A "Why it is planned" note explains the training rationale, for example, advising the user to stop when form breaks down. The user can tap "Swap this exercise" at the bottom to replace the exercise with an alternative.

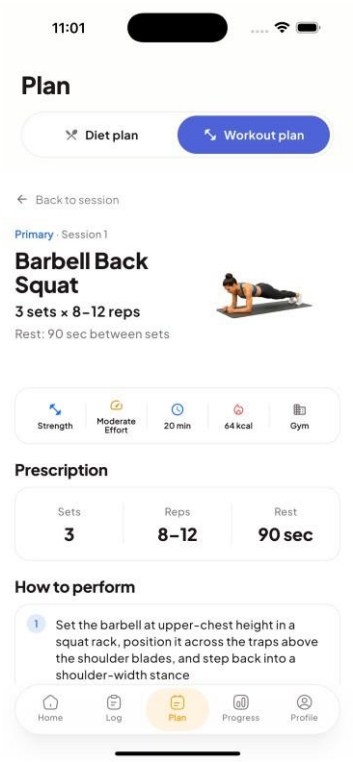


*Figure 5.35: Exercise Detail Screen*

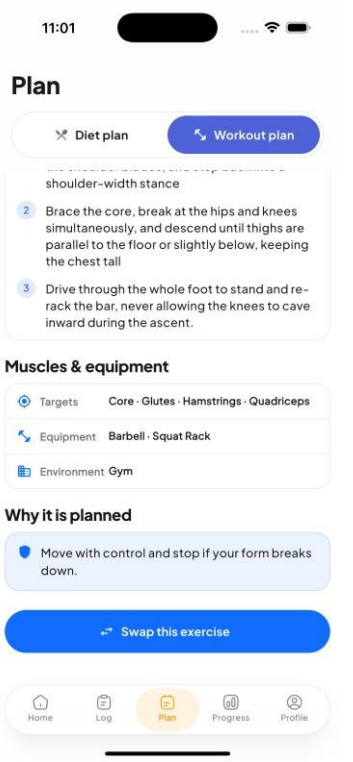


*Figure 5.36: Exercise Detail (Form and Equipment)*

**Exercise Swap Flow**

Tapping "Swap this exercise" on any exercise card in the plan overview or tapping "Swap this exercise" on the Exercise Detail screen, opens a "Change exercise" bottom sheet. This sheet shows the current exercise name with its key metrics and a note that the impact preview will appear before the user confirms. Two options are presented: Recommended Swaps (best-fit alternatives identified by the engine) and Search & Pick (which lets the user browse the full exercise dataset).

The Recommended Swaps screen lists up to fifteen compatible alternatives, each tagged with a "Best fit" badge, the training type, estimated time, and estimated calories. The Search & Pick screen provides a search bar that accepts exercise names, muscle groups, or equipment names, together with filter chips for training type and difficulty. Results are ranked by compatibility with the session slot. When the user selects an exercise from either screen, the Workout Swap Preview screen is shown. This screen displays the weekly impact of the change: weekly energy expenditure before and after, goal support rating before and after, and muscle coverage before and after. A "This swap keeps weekly support stable" confirmation banner appears when the change does not reduce the plan quality. The user taps "Confirm swap" to apply the change.

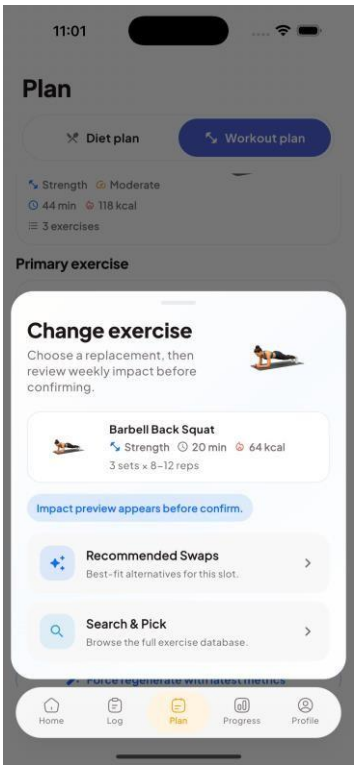


*Figure 5.37: Change Exercise Bottom Sheet*

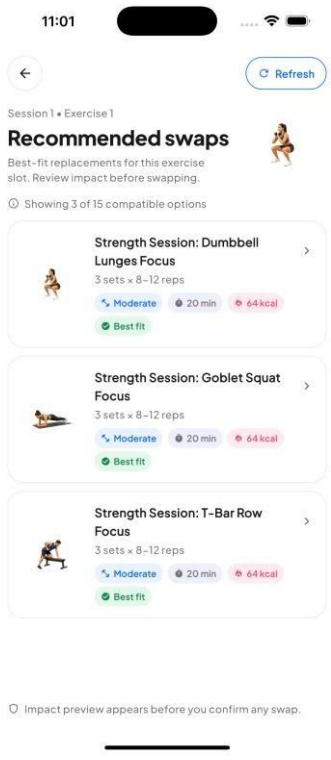


*Figure 5.38: Recommended Exercise Swaps*

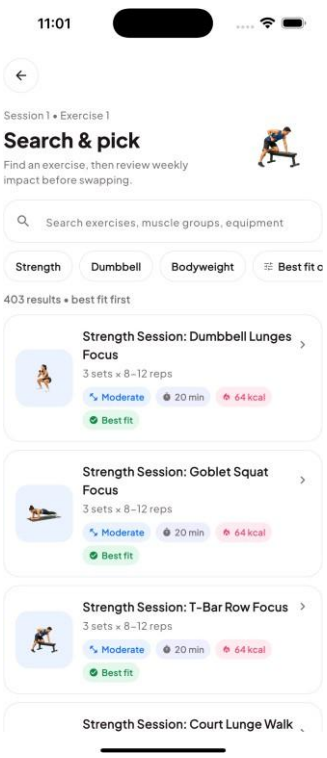


*Figure 5.39: Exercise Search Screen*

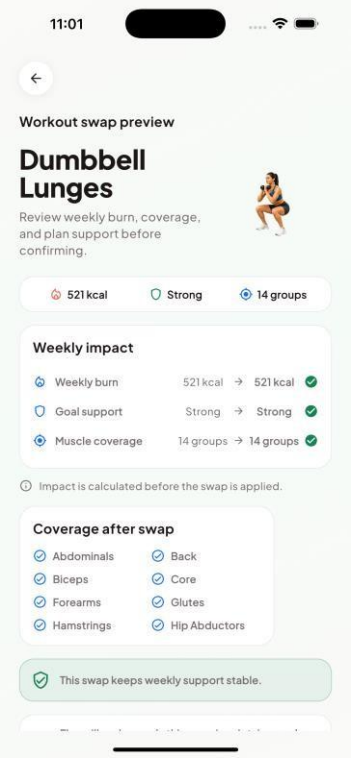


*Figure 5.40: Exercise Swap Impact Preview*

### 5.5.6 Daily Logging

The logging module is the user's daily diary. It allows the user to record what they ate, what exercise they completed, how much water they drank, and their current body weight. All entries are stored against the selected date so the user can backfill past days if needed. The module is accessed by tapping the "Log" tab in the bottom navigation bar.

#### Log Tab Overview and Diary

The Log screen opens to the current date, shown with a weekly calendar strip at the top. The user can tap any date in the strip to switch to that day. The diary section below the date strip lists all logged meals and logged workouts for the selected date, grouped under "Meals" and "Workouts" headings with a count of logged entries beside each heading. A floating orange plus button in the bottom-right corner opens the "Add to diary" menu.

The "Add to diary" bottom sheet presents five logging options: Scan meal (to estimate nutrition from a food photo), Log meal (to pick from planned meals or add a custom entry), Log workout (to record a planned or custom exercise session), Add water (to update the daily water intake),

and Log weight (to record today's body weight). The user taps whichever option applies and is taken to the corresponding logging screen.

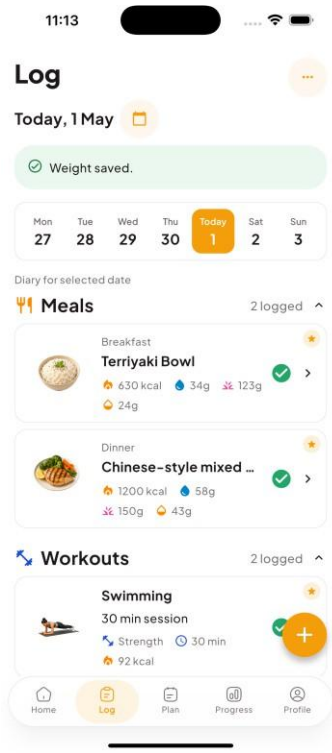


*Figure 5.41: Log Tab Overview*

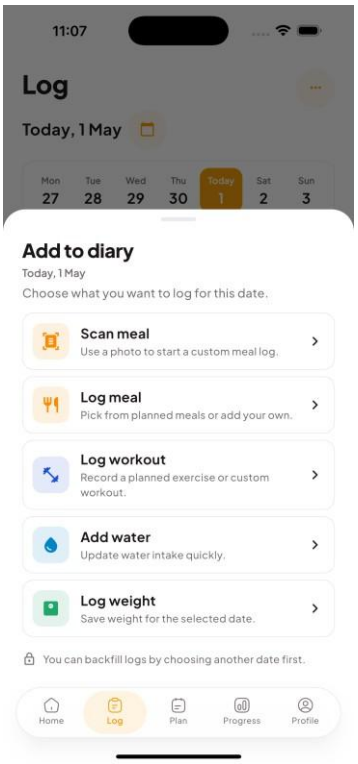


*Figure 5.42: Add to Diary Menu*

**Meal Logging**

The Log Meal screen has two tabs at the top: Planned and Custom. In the Planned tab, the user sees the meals from today's active diet plan, each shown with the meal type, recipe name, image, and macros. The user taps the orange plus button next to a meal to log it. Once logged, the plus button changes to a green checkmark to confirm the entry. At the bottom of the Planned tab, an "Add custom meal" button and a "Scan meal" button provide alternative entry paths.

In the Custom tab, the user fills in a form with a time selector, meal type chips (Breakfast, Lunch, Dinner, Snack), a meal name, and four nutrition fields for calories, protein, carbohydrates, and fat. An optional note field is also available. A "Save as reusable meal" checkbox allows the user to save the entry as a template for quick re-use on future days. The user taps "Save to diary" to confirm.

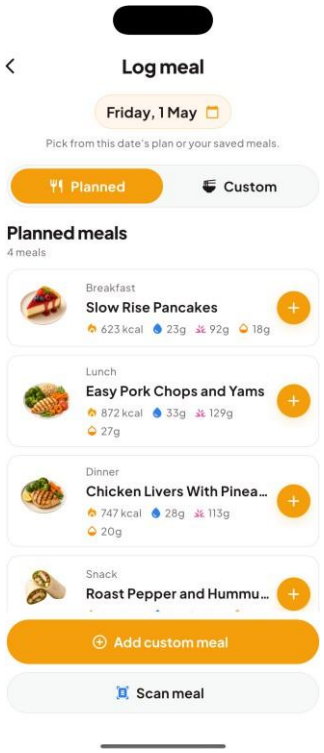


*Figure 5.43: Log Meal – Planned Tab*

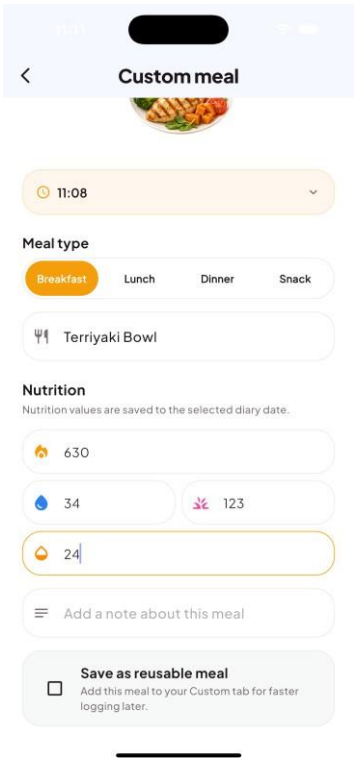


*Figure 5.44: Log Meal – Custom Form*

**Meal Photo Scan**

The Scan Meal screen explains that Flux will estimate the calories and macros from a food photo and then send the user to the custom meal form to review the values before saving. The user can choose to take a new photo with the camera or select an existing photo from the device gallery. Alternatively, a link at the bottom lets the user skip the scan and enter values manually.

After the photo is analyzed, the result is displayed as a "Scanned draft" with a confidence level (for example, Medium confidence). The AI description of what was identified in the photo is shown above the pre-filled custom meal form. The user reviews the estimated values, adjusts any fields that look incorrect, and taps "Save to diary" to log the entry.

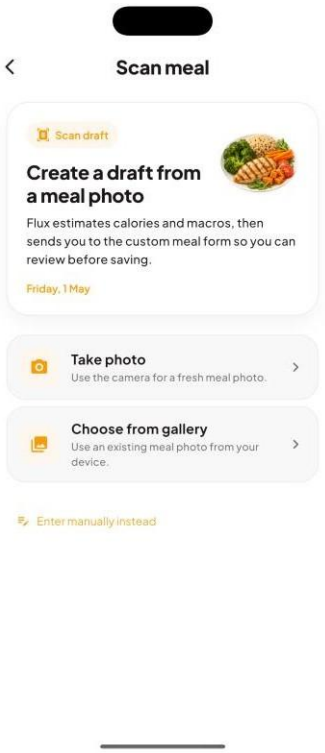


*Figure 5.45: Scan Meal Screen*

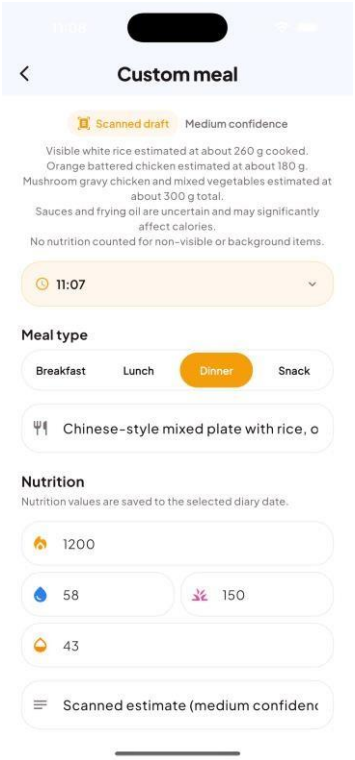


*Figure 5.46: Scan Meal Result*

### Workout Logging

The Log Workout screen has a Planned tab and a Custom tab, similar to the meal logging screen. In the Planned tab, the user sees the exercises from today's active workout plan, each showing the exercise name, set and rep scheme, training type, estimated duration, and estimated calories. The user taps the plus button next to each exercise to log it. Once logged, the exercise row shows a "Logged" badge with a green checkmark.

In the Custom Workout form, the user selects a category (Strength, Cardio, Stretching, or Sports), enters the workout name, the duration in minutes, and the estimated calories burned. An "AI-assisted calorie estimate" section appears below the calorie field: if the user is unsure of the calorie burn, they can tap "Estimate calories with AI" and the backend will use Compendium of Physical Activities MET values [11] to calculate an estimate based on the activity name and duration. Optional performance fields (sets, reps, and weight) and a notes field allow the user to record additional detail. A "Save as reusable workout" toggle allows the entry to be saved as a template. The user taps "Save to diary" to confirm.

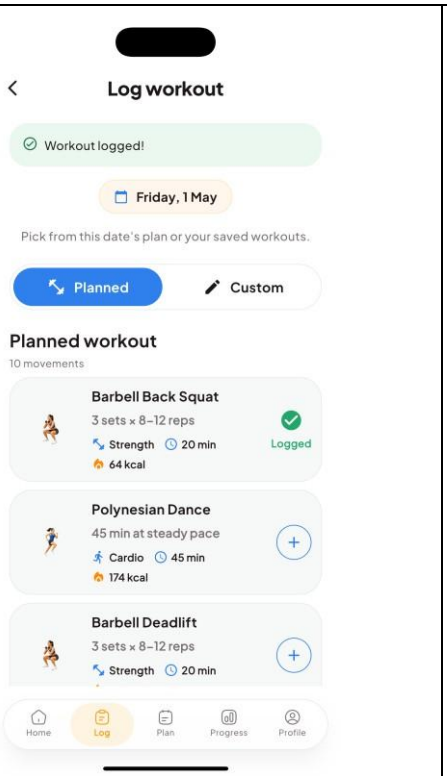


*Figure 5.47: Log Workout – Planned Tab*

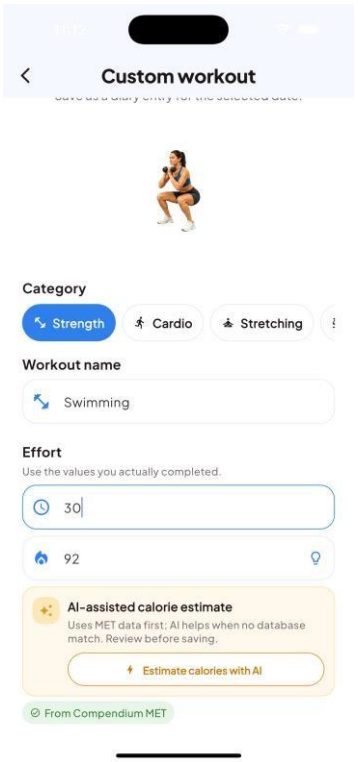


*Figure 5.48: Custom Workout Form*

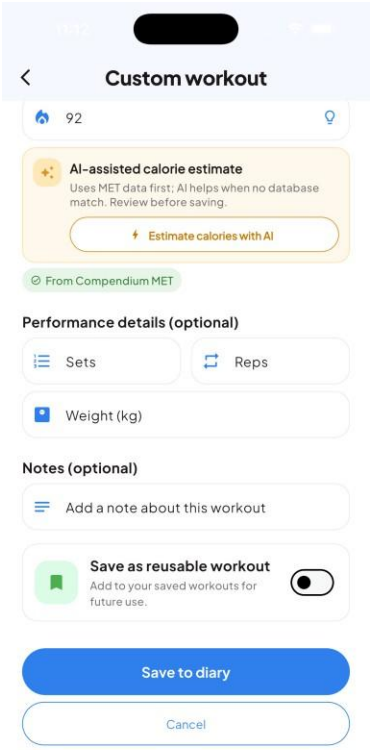


*Figure 5.49: AI Calorie Estimate for Custom Workout*

**Water and Weight Logging**

The Add Water screen shows the total waterlogged so far for the selected date against the daily target. Quick-select chips offer common amounts (250 ml, 500 ml, 750 ml, and 1 L). The user can also type a custom amount in the text field below. Selecting a quick chip clears the custom field, and typing a custom amount deselects the chips. The user taps "Add water" to save the entry.

The Log Weight screen displays the currently selected weight value with a scrollable picker. The user adjusts the value to their measured weight and taps "Save weight" to store it. Only one weight entry is kept per date; saving a new value for the same date replaces the previous entry. The saved weight is also used to update the current weight shown in the Profile tab and feeds into the body trend chart in the Progress tab.

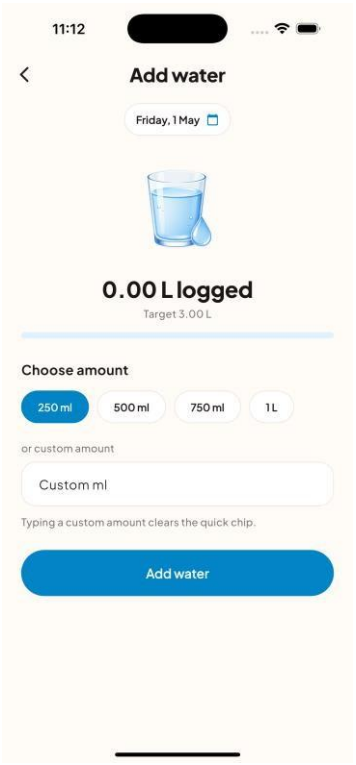


*Figure 5.50: Add Water Screen*

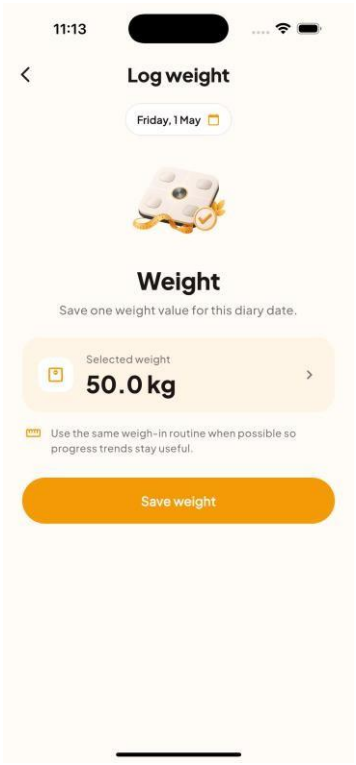


*Figure 5.51: Log Weight Screen*

**Log History and Saved Items**

Tapping the three-dot icon at the top-right of the Log screen opens a menu with three options: View full log history, Add/Edit saved custom meal, and Add/Edit saved custom workout.

The Log History screen displays all saved diary entries, sorted by date. Filter tabs (All, Meals, Workouts, Water) allow the user to narrow the list. A search bar and a month-picker help the user find entries from specific dates. Entries are grouped by date and show the entry type, name, and value (for example, calories or litres).

The Saved Custom Meals screen lists all reusable meal templates the user has saved. Tapping any saved meal opens an "Edit saved custom meal" bottom sheet where the user can update the name, meal type, and nutrition values, then tap "Save template" to apply changes or "Delete template" to remove it permanently. The Saved Workouts screen works in the same way for saved workout templates.

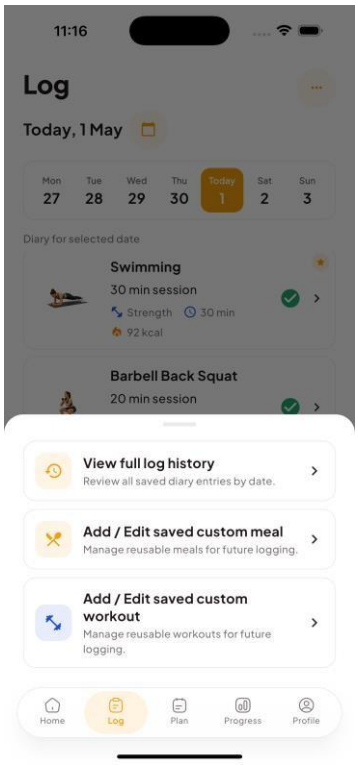


*Figure 5.52: Log More Actions Menu*

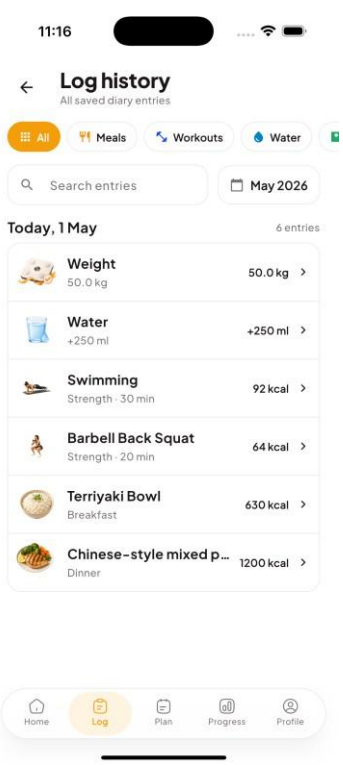


*Figure 5.53: Log History Screen*

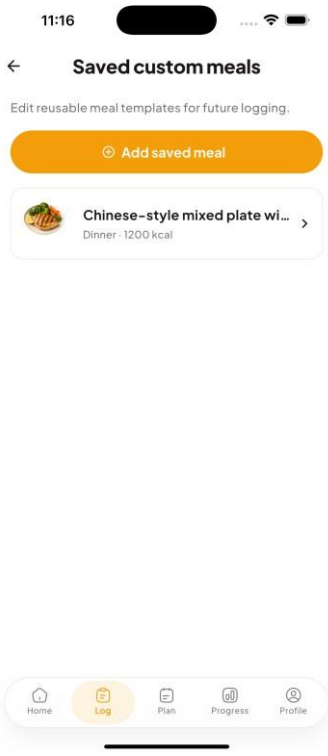


*Figure 5.54: Saved Custom Meals*

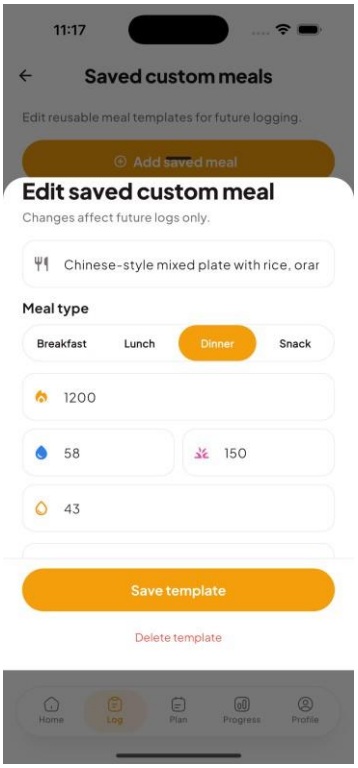


*Figure 5.55: Edit Saved Meal*

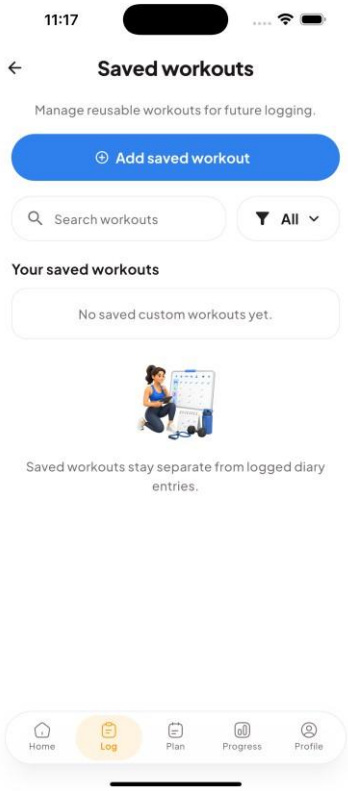


*Figure 5.56: Saved Workouts Screen*

### 5.5.7 Progress Tracking and AI Coaching Insights

The progress module gives the user visibility into how well they are following their plan over time. It is accessed by tapping the "Progress" tab in the bottom navigation bar. The screen has three tabs: Today, 7-day view, and Body trend.

#### Today View

The Today tab shows a summary of everything the user has logged for the current date. A large calorie ring at the top displays the total kilocalories consumed against the daily target, with the remaining balance shown below. Three progress bars below the ring show the protein, carbohydrates, and fat logged against their respective targets. A water row shows the current logged amount against the daily water goal. A workout sessions row shows how many sessions have been logged and how many remain open. A weight row confirms whether the day's weight has been recorded.

A "Plan follow-through today" section at the bottom compares how many planned meals have been logged against the total planned and shows the workout session status. This gives the user a clear picture of their adherence to the generated plan for the day.

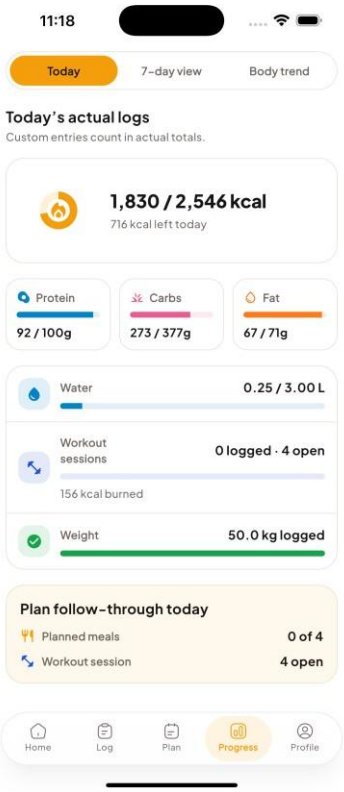


*Figure 5.57: Progress – Today View*

**Seven-Day View**

The 7-day view tab shows two charts side by side: a "Diet cycle intake" bar chart that plots daily calorie consumption for each day of the active plan cycle against the target line, and a "Workout cycle burn" bar chart that plots daily energy expenditure against the average plan-day target line. Below the charts, a macro summary table shows the total logged protein, carbohydrates, fat, and workout burn for the full cycle against the cycle targets, along with the Acceptable Macronutrient Distribution Range, abbreviated as AMDR, reference [36].

A "Generated plan follow-through" section at the bottom shows how many diet cycle meals have been logged (for example, 0 of 28) and how many workout cycle sessions have been completed (for example, 0 of 4) for the active cycle.


*Figure 5.58: Progress – 7-Day View*

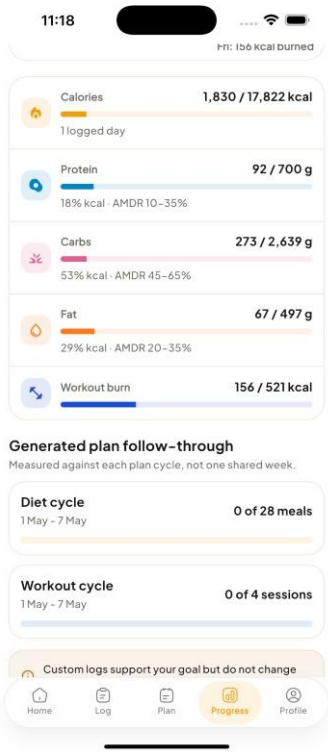


*Figure 5.59: Progress – 7-Day Macro Summary*

**Body Trend**

The Body trend tab displays a line chart that plots the user's logged body weight over time against the target weight. The chart requires at least two logged weight entries before the trend line can be drawn. Until then, a prompt appears asking the user to log weight on at least two days to unlock the trend. A summary below the chart shows the target weight, the calculated actual trend, and the gap between current weight and target weight. The trend data feeds into the goal timeline projection shown in the Today view.

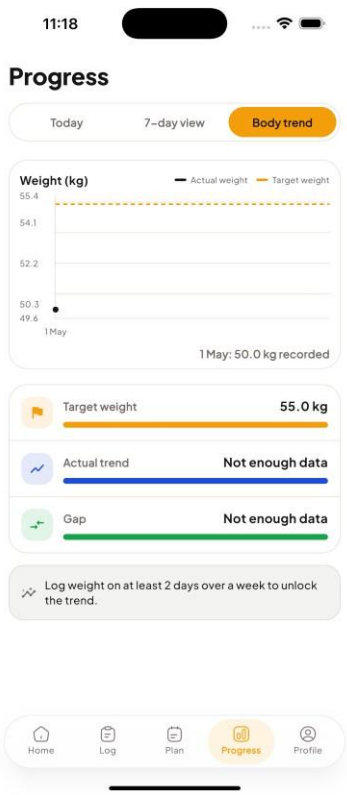


*Figure 5.60: Progress – Body Trend Tab*

**AI Weekly Insights**

At the bottom of the 7-day view tab, a "AI weekly insights" section appears once enough data has been logged. The system generates exactly three personalized, non-medical insights based on the user's logged calories, protein, and workout frequency for the current cycle. Each insight is displayed as a bullet point in plain language, for example, noting that the user's average calorie intake was below the target and suggesting logging more consistently next week. If logging has been sparse (for example, only one out of seven days was logged), the insight explicitly discloses this limitation and bases the guidance on the available data. A "Regenerate" button allows the user to refresh the insights.

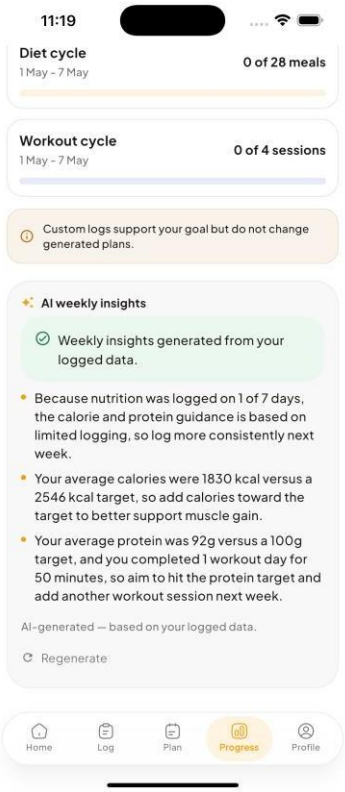


*Figure 5.61: AI Weekly Insights*

### 5.5.8 Profile Management

The Profile module allows the user to view their account summary, edit their health details and dietary or workout preferences, and manage their account. It is accessed by tapping the "Profile" tab in the bottom navigation bar.

#### Profile Overview

The Profile screen shows the user's name and email address at the top, followed by a quick-summary row of four key values: primary goal, daily calorie target, protein macro target, and target body weight. Below this, a "Profile setup" section links to three editors: Health details (body stats, goals, and activity level), Diet preferences (diet style, allergies, meals, and hydration), and Workout preferences (intensity, days per week, and equipment). A "Targets" section shows the user's current weight, target weight, calculated BMI, and activity level for quick reference. At the bottom of the screen, a "Log out" button ends the current session, and a "Delete account" button permanently removes all data associated with the account.

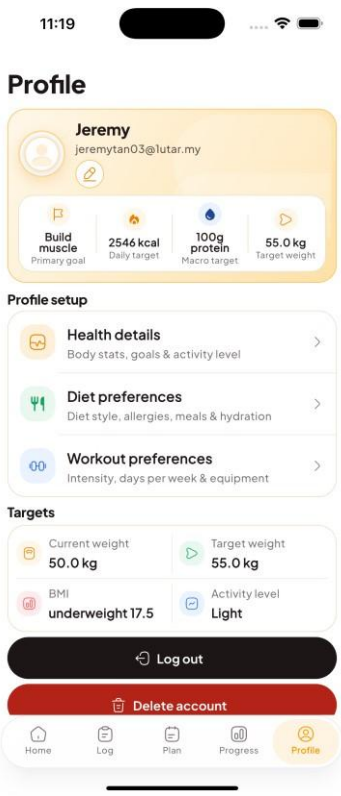


*Figure 5.62: Profile Overview Screen*

**Editing Health Details, Diet Preferences, and Workout Preferences**

Tapping "Health details" opens a four-step guided editor. The first step (Body basics) asks for the user's date of birth and biological sex. Each subsequent step collects more detail such as current weight, height, activity level, and goal. When the user saves changes, the system recalculates the user's metrics (BMR, TDEE, and macronutrient targets) and returns the updated values to the Profile and Home screens immediately.

Tapping "Diet preferences" opens a five-step guided editor beginning with dietary style selection. The remaining steps collect disease conditions, allergies, meal types, and the daily water target. Changes saved here affect future plan generation. If a plan is already active with logged entries, the backend signals that the changes will take effect at the start of the next fresh cycle rather than immediately replacing the active plan.

Tapping "Workout preferences" opens a six-step guided editor beginning with the workout opt-in choice (Include workouts or Skip workouts). If workouts are enabled, the following steps collect intensity, days per week, session duration, location, and equipment. Saving changes follows the same cycle-protection logic as diet preferences.

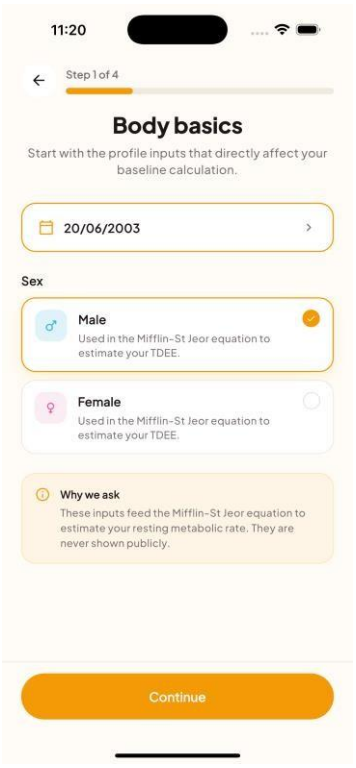


*Figure 5.63: Edit Health Details (Step 1 of 4)*

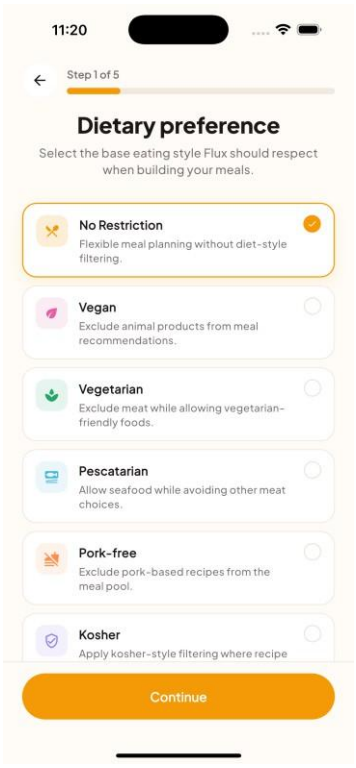


*Figure 5.64: Edit Diet Preferences (Step 1 of 5)*

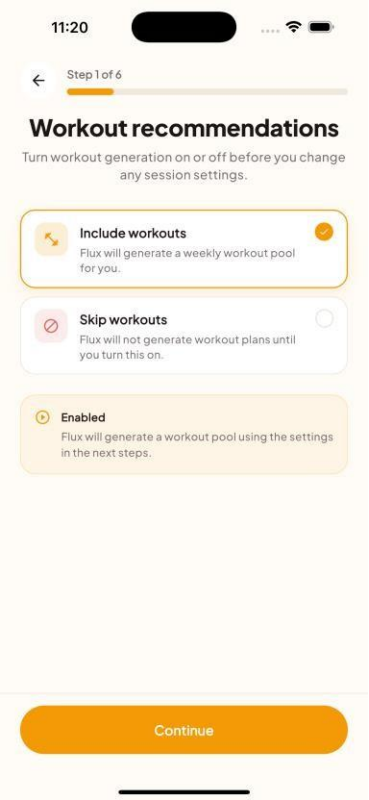


*Figure 5.65: Edit Workout Preferences (Step 1 of 6)*

**Account Management**

Tapping "Delete account" on the Profile screen opens a confirmation dialog that clearly lists all data that will be permanently deleted: the profile and account access, all diet and workout plans, all diary logs, and all progress history. To proceed, the user must enter their current password in the first field and type the word DELETE in the second field. Only when both conditions are met does the "Delete account" button become active. The deletion is irreversible, and the dialog makes this explicit with the statement "This action is permanent and cannot be undone." The user can tap "Cancel" at any time to dismiss the dialog without deleting the account.

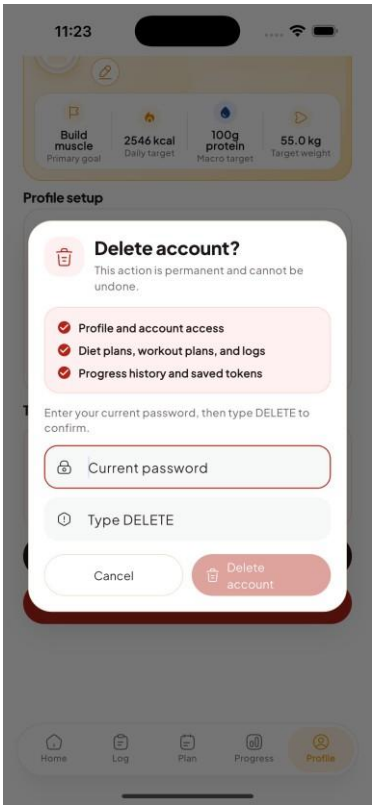


*Figure 5.66: Delete Account Confirmation Dialog*

## 5.6 Implementation Issues and Challenges

Several implementation challenges arose during the development of Flux and required careful analysis before being resolved. The recipe and workout datasets required cleaning, normalization, and filtering before they could support reliable recommendations. The recommendation engines had to respect user goals, allergies, disease-related constraints, and logged history without overwriting completed user actions. The frontend also had to stay synchronized with backend state after onboarding changes, plan regeneration, meal or exercise swaps, and daily logging. These issues were handled through service-layer rules, protected database relationships, API validation, and repeated backend and Flutter verification cycles.

## 5.7 Concluding Remark

This chapter has documented the implementation of all core modules of the Flux application. The authentication module provides a secure, multi-path account management system supporting email/password and Google OAuth sign-in, JWT-based session management with automatic token refresh, and OTP-based email verification and password reset. The onboarding module collects nine categories of user data and triggers server-side computation of all health metrics using a pipeline of evidence-based formulas: Mifflin-St Jeor BMR [4], ACSM TDEE multipliers [33], ISSN protein targets [5], and IOM carbohydrate minimums [36], resulting in personalized calorie and macronutrient targets.

The diet recommendation engine implements a four-stage generation pipeline recipe pool filtering, disease-aware beam search assembly, goal-adaptive scoring, and FDA health tag application to produce a personalized 7-day meal plan. The workout recommendation engine applies a six-stage pipeline with session distribution, exercise pool filtering, compound-first selection, structured prescription generation, MET-based calorie estimation [11], and WHO/ACSM goal-support assessment [12], [33] to produce a personalized weekly training plan. Both recommendation modules include a swap flow with nutritional impact previews before any modification is committed.

The AI meal image scanning feature uses the OpenAI GPT-5.5 vision model to analyze meal photographs and return structured nutritional estimates, with rigorous input validation and response normalization. The daily logging module records all food, exercise, water, and weight entries across three entry paths (planned, custom, and saved templates) and automatically updates health metrics when a new weight measurement is recorded. The progress module

aggregates weekly data into visualizations and generates three personalized coaching insights via the OpenAI Chat Completions API, with insights cached to avoid redundant API calls within the same week.

# Chapter 6 System Evaluation and Discussion

## 6.1 System Testing and Performance Metrics

This chapter evaluates the completed Flux system using functional test cases, backend regression tests, Flutter verification, live database checks, and user-interface walkthrough evidence. The evaluation focuses on whether each implemented module behaves correctly, preserves user data, and supports the project objectives defined in Chapter 1.

The user interface is built with Flutter 3 and follows a consistent design system throughout the application. The bottom navigation bar provides access to the five main areas of the app: Home, Log, Plan, Progress, and Profile. The overall visual style uses an orange-amber primary accent, clean white backgrounds, and a warm neutral palette, with pill-shaped buttons and rounded card surfaces for a modern feel.

All quality assurance tests were executed against a live Node.js backend connected to a real MySQL database and an iOS Simulator running Flutter on iPhone 16 Pro Max. The backend automated regression suite passed 408 of 408 tests, and the Flutter static analysis and widget test suite passed 274 of 274 tests before and after each fix cycle.

## 6.2 Testing Setup and Result

Testing was organized by module so that each major user flow could be evaluated from screen interaction to API response and database state. The test result tables in the following subsections summarize the expected behaviour, test action, and observed result for authentication, onboarding, home dashboard, diet plan, workout plan, logging, progress, and profile functions.

### 6.2.1 Authentication Module

Section 5.5.1 presents the user authentication screens and interaction flows. The test cases below verify the authentication API endpoints and security behaviour.
Table 6.1 below presents the test cases for the authentication module.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|

| | | | |
|---|---|---|---|
| AUTH-01 | Register new account with valid information | API creates user record and initiates email verification flow. Server returns HTTP 201. | Pass |
| AUTH-02 | Register second account with valid information | API creates user record and starts email verification flow. Server returns HTTP 201. | Pass |
| AUTH-03 | Login with valid credentials | API returns access token, refresh token, and user object. Server returns HTTP 200 for all test accounts. | Pass |
| AUTH-04 | Login with wrong password | API rejects the attempt. Returns 401 INVALID_CREDENTIALS. | Pass |
| AUTH-05 | Refresh access token | API rotates and returns valid new access credentials. Returns HTTP 200. | Pass |
| AUTH-06 | Logout invalidates refresh token | API marks refresh token as used. Returns HTTP 200. | Pass |
| AUTH-07 | Forgot password request for valid email | API creates reset flow and returns a generic success message. Returns HTTP 200. | Pass |
| AUTH-08 | Verify OTP and reset password | OTP verification returns reset token; password reset completes; login works with new password. | Pass |
| AUTH-09 | Reset password screen shows strength indicators | Screen shows Set a new password title, strength labels (8+ chars, Uppercase, Number, Special char), no manual token field. | Pass |
| AUTH-10 | Reject invalid Google token | API rejects the request. Returns 400 INVALID_GOOGLE_TOKEN. | Pass |
| AUTH-11 | Delete account with wrong credentials | API rejects wrong confirmation (400) and wrong password (401 INVALID_PASSWORD). | Pass |
| AUTH-12 | Delete account with correct credentials | User row removed from the live database. API returns HTTP 200. | Pass |

*Table 6.1: Authentication Module Test Cases*

### 6.2.2 Onboarding Module

Section 5.5.2 presents the onboarding flow screens. The test cases below verify that each onboarding step saves correctly and that health metrics are computed accurately.

Table 6.2 below presents the test cases for the onboarding module.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|
| ONB-01 | Save health profile for a weight-loss user | Profile stores full name, sex, height, current weight, target weight, goal, and activity level in the database. | Pass |
| ONB-02 | Save diet preferences for a weight-loss user | Diet preference row stores dietary restriction, disease condition, meal types, and water goal. | Pass |
| ONB-03 | Save workout preferences for a weight-loss user | Workout preference row stores enabled plan settings, intensity, and days per week. | Pass |
| ONB-04 | Save muscle-gain onboarding profile | Metrics reflect a higher-calorie muscle-gain scenario: 3188.9 kcal, 144.0 g protein stored. | Pass |
| ONB-05 | Save maintenance profile with workout disabled | Vegan diet metrics calculated; workout_recommendation stored as 0. | Pass |
| ONB-06 | Mark onboarding as complete | onboarding_complete flag set to 1 for all test accounts after all steps submitted. | Pass |
| ONB-07 | Retrieve current metrics after onboarding | API returns BMI, BMR, TDEE, calorie target, and macro targets matching the stored user_metrics row. | Pass |
| ONB-08 | Calorie floor applied for weight-loss user | Weight-loss daily calorie target does not fall below the female safety floor. Stored as 1200.0 kcal. | Pass |
| ONB-09 | Profile avatar upload | API saves the uploaded image and returns a profile-avatars URL. Returns HTTP 200. | Pass |
| ONB-10 | Auth state after onboarding complete | GET /api/auth/me returns verified and onboarding-complete flags for the signed-in user. | Pass |
| ONB-11 | Metrics refresh when workout preference is absent | Metrics recalculate from profile and diet data only; user_metrics upserted without error. | Pass |

*Table 6.2: Onboarding Module Test Cases*

### 6.2.3 Home Dashboard

Section 5.5.3 presents the Home Dashboard screen. The Home Dashboard has no dedicated test case table; its correctness is verified through the logging and progress module tests.

### 6.2.4 Diet Plan Module

Section 5.5.4 presents the diet plan screens and swap flow. The test cases below verify diet plan generation, display, and meal swap behaviour.

Table 6.3 below presents the test cases for the diet plan module.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|
| DIET-01 | Generate 7-day vegetarian weight-loss meal plan | Plan generated with 7 days and 28 meals (4 meal types per day) matching the user diet and disease constraints. | Pass |
| DIET-02 | Generate 7-day muscle-gain meal plan | Plan generated with 7 days and 21 meals (3 meal types per day) at higher calorie target. | Pass |
| DIET-03 | Generate 7-day vegan maintenance meal plan | Plan generated without workout preference dependency. 7 days and 21 meals returned. | Pass |
| DIET-04 | Retrieve active diet plan | API returns current plan with days[].meals nested structure and daily macro totals. | Pass |
| DIET-05 | Validate API response shape | Nested day structure confirmed; old flat-meal assumption corrected. Passes. | Pass |
| DIET-06 | Recipe search for an open meal slot | API returns candidate recipes matching the meal type and search query. 5 candidates returned. | Pass |
| DIET-07 | Dish type filter list | API returns 10 curated dish-type filter options for the search UI. | Pass |
| DIET-08 | Swap options for an open meal slot | API returns 5 best-fit alternative recipes for the selected slot. | Pass |
| DIET-09 | Impact preview before confirming swap | API returns before/after nutrition comparison using numeric recipe ID. Returns HTTP 200. | Pass |
| DIET-10 | Confirm meal swap | Planned meal slot updated to the selected recipe. Returns HTTP 200. | Pass |
| DIET-11 | Reject swap on a logged meal slot | API rejects swap on a slot that was already logged. Returns 400 ITEM_CLOSED. | Pass |
| DIET-12 | Regenerate one day (open slots only) | Open meal slots for that day are regenerated; previously logged meals stay unchanged. | Pass |

| | | | |
|---|---|---|---|
| DIET-13 | AI recipe steps for a recipe with no source instructions | API generates 6 cooking steps without overwriting the original recipe row. Returns HTTP 200. | Pass |
| DIET-14 | Generate steps with AI from within the app | Empty preparation state shows AI button; tapping it calls the backend and renders generated steps on screen. | Pass |
| DIET-15 | AI cooking-step grounding guards | Generated prompt stays recipe-grounded; unrelated food topics are rejected. | Pass |
| DIET-16 | Force regeneration after metrics change starts from the next day | A plan generated with updated metrics starts on the following day, not the current day. | Pass |

*Table 6.3: Diet Plan Module Test Cases*

### 6.2.5 Workout Plan Module

Section 5.5.5 presents the workout plan screens and swap flow. The test cases below verify workout plan generation, display, and exercise swap behaviour.

Table 6.4 below presents the test cases for the workout plan module.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|
| WO-01 | Preview workout plan without saving | API returns a plan preview without writing final plan rows to the database. Returns HTTP 200. | Pass |
| WO-02 | Generate workout plan for weight-loss user | Plan matches moderate home or outdoor settings; generated with 5 exercises across the weekly cycle. | Pass |
| WO-03 | Generate workout plan for muscle-gain user | Plan matches vigorous gym settings; generated with 18 exercises emphasising strength work. | Pass |
| WO-04 | Reject workout generation when recommendation is disabled | API rejects the request for a user who opted out of workout plans. Returns 400 WORKOUT_RECOMMENDATION_DISABLED. | Pass |
| WO-05 | Retrieve active workout plan | API returns plan rows with planExerciseId exposed for frontend use. | Pass |
| WO-06 | Search for exercise swap candidates | API returns compatible exercise alternatives for the selected slot. | Pass |
| WO-07 | Workout swap impact preview | API returns weekly impact metrics (burn, goal support, muscle coverage) before any change is confirmed. Returns HTTP 200. | Pass |
| WO-08 | Confirm exercise swap | Selected exercise replaces the planned slot; plan weekly totals updated. Returns HTTP 200. | Pass |
| WO-09 | Force regeneration after metrics change starts from the next day | A workout plan generated with updated metrics starts on the following day; any logged exercises for today are preserved. | Pass |

*Table 6.4: Workout Plan Module Test Cases*

### 6.2.6 Logging Module

Section 5.5.6 presents the logging screens. The test cases below verify meal, exercise, water, and weight logging behaviour and AI meal scan integration.

Table 6.5 below presents the test cases for the logging module.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|
| LOG-01 | Log a planned meal from the active diet plan | Planned meal row inserted into diary; returns HTTP 201. Meal slot marked as logged. | Pass |
| LOG-02 | Reject duplicate logging of the same planned meal | API rejects a second log attempt for the same planned meal slot. Returns 400 ALREADY_LOGGED. | Pass |
| LOG-03 | Create a custom meal log with user-entered nutrition | Custom meal row inserted separately from plan rows. Returns HTTP 201. | Pass |
| LOG-04 | Edit a custom meal log | Nutrition values and notes updated in the diary row. Returns HTTP 200. | Pass |
| LOG-05 | Delete a custom meal log | Custom meal log row removed from the database. Returns HTTP 200. | Pass |
| LOG-06 | Saved custom meal CRUD | Saved meal template can be created, updated, listed, used to log, and deleted. | Pass |
| LOG-07 | Meal photo scan returns reviewable draft | AI scan returns a draft with name, meal type, confidence level, and estimated nutrition. No meal log inserted automatically. | Pass |
| LOG-08 | Log a planned workout from the active workout plan | Planned exercise row inserted into diary using planExerciseId. Returns HTTP 201. | Pass |
| LOG-09 | Reject duplicate logging of the same planned exercise | API rejects a second log attempt for the same planned exercise slot. Returns 400 ALREADY_LOGGED. | Pass |
| LOG-10 | Estimate calorie burn for a custom workout | Matched exercise returns calorie estimate from the Compendium MET table; unmatched exercise returns an AI-assisted MET estimate. No workout log inserted automatically. | Pass |
| LOG-11 | Create a custom workout log | Custom workout row inserted separately from plan rows. Returns HTTP 201. | Pass |
| LOG-12 | Edit a custom workout log | Custom workout values updated. Returns HTTP 200. | Pass |
| LOG-13 | Delete a custom workout log | Custom workout log row removed from the database. Returns HTTP 200. | Pass |
| LOG-14 | Saved custom workout CRUD | Saved workout template can be created, updated, listed, used to log, and deleted. | Pass |
| LOG-15 | Water increment, decrement, and retrieval | Water total updates correctly; never goes below zero; stored by selected date. | Pass |

| | | | |
|---|---|---|---|
| LOG-16 | Weight upsert and trend update | Latest weight refreshes the stored profile weight and recalculates the BMI shown in the Profile tab. | Pass |
| LOG-17 | Log history retrieval by date | API returns all logs for the selected date across meals, workouts, water, and weight. | Pass |
| LOG-18 | Scan screen overlay and error messages | Scan shows an analysing overlay; timeout and connection errors display clear messages without layout errors. | Pass |
| LOG-19 | Success feedback after custom and scanned saves | Diary screen shows success message after saving a custom meal, scanned meal, or custom workout. | Pass |

*Table 6.5: Logging Module Test Cases*

### 6.2.7 Progress Module

Section 5.5.7 presents the progress tracking screens. The test cases below verify daily aggregation, weekly summary, and AI coaching insight generation.

Table 6.6 below presents the test cases for the progress module.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|
| PROG-01 | Today summary combines planned and custom logs | API returns planned meals, custom meals, planned workouts, and custom workouts in separate arrays alongside water and weight data. | Pass |
| PROG-02 | Custom logs support progress without replacing generated plan slots | Custom entries counted in daily totals; generated plan slots remain open for the user to log separately. | Pass |
| PROG-03 | Weekly summary uses the active plan cycle dates | API returns summary anchored to the active diet and workout cycle start date, not the current calendar week. | Pass |
| PROG-04 | Goal timeline projection | API returns current goal projection based on stored metrics and logged progress. | Pass |
| PROG-05 | Calorie history retrieval | API returns logged calorie totals over time for chart display. | Pass |
| PROG-06 | Macro history retrieval | API returns macro trends over time. | Pass |
| PROG-07 | Workout frequency retrieval | API returns workout session frequency data for the progress charts. | Pass |
| PROG-08 | Goal completion guard before target is reached | API rejects completion confirmation if the goal has not been reached. Returns 400 GOAL_NOT_REACHED. | Pass |
| PROG-09 | AI weekly insights generated from logged data | API returns exactly 3 non-medical personalised insights; sparse logging is disclosed. Returns HTTP 200. | Pass |
| PROG-10 | Weekly AI prompt uses correct protein target and day count | Prompt uses the stored protein_g metric; goal completion counts same-day completion as 1 day. Tests passed. | Pass |

*Table 6.6: Progress Module Test Cases*

### 6.2.8 Profile Module

Section 5.5.8 presents the profile management screens. The test cases below verify profile editing, preference updates, and account management.

Table 6.7 below presents the frontend simulator test cases covering all five main tabs.

| Test ID | Test Case | Expected Result | Status |
|---|---|---|---|
| UI-01 | Login screen rendering in iOS Simulator | Screen shows email, password, Sign in, Forgot password, Continue with Google, and Register controls. | Pass |
| UI-02 | Login with QA test account | App navigates to the authenticated main app because account is verified and onboarding is complete. | Pass |
| UI-03 | Home dashboard rendering | Home shows greeting, calorie intake ring, macro bars, water controls, planned meals, workout sessions, and weight check-in. | Pass |
| UI-04 | Log tab rendering | Log tab shows the selected-date diary with meals, workouts, water, weight, and the add button. | Pass |
| UI-05 | Plan tab rendering | Plan tab shows the diet/workout segment, day picker, plan totals, and Swap buttons for unlogged meals. | Pass |
| UI-06 | Progress tab rendering | Progress tab shows the Today, 7-day view, and Body trend tabs with calorie chart, workout burn chart, and adherence summary. | Pass |
| UI-07 | Profile tab rendering | Profile tab shows account email, goal, targets, preference sections, Log out, and Delete account controls. | Pass |
| UI-08 | AI preparation steps for a recipe with no source instructions | Empty preparation state shows AI button; tapping it calls the backend and renders generated steps on screen. | Pass |

*Table 6.7: Frontend Simulator Test Cases*

Table 6.8 below presents the real-user workflow test cases that simulate end-to-end user scenarios across multiple modules.

| Test ID | Scenario | Expected Result | Status |
|---|---|---|---|
| REAL-01 | Regenerate a started diet day (some meals already logged) | Logged meals remain unchanged. Open meal slots regenerate to new recipes. | Pass |
| REAL-02 | Regenerate a started workout cycle (one exercise already logged) | Logged exercise stays in the active plan. Open exercises regenerate. | Pass |
| REAL-03 | Edit diet preferences after a plan has already been started with logged items | Backend signals apply_next_fresh_cycle; the current cycle is not interrupted. | Pass |
| REAL-04 | Edit preferences before any logs exist | Backend signals regenerate_now; plan is immediately replaced with the updated preferences. | Pass |
| REAL-05 | Regenerate plan immediately after a preference edit | New diet and workout plans generated reflecting the changed meal count and workout settings. | Pass |
| REAL-06 | Restore account to original preferences after preference-edit test | Active diet and workout plans return to the original expected shapes after restoration. | Pass |
| REAL-07 | Swap on a logged meal slot is rejected; swap on an open slot succeeds | Logged slot returns 400 ITEM_CLOSED. Open slot returns search results normally. | Pass |

*Table 6.8: Real-User Workflow Test Cases*

## 6.3 Project Challenges

The development of Flux involved a number of technical and conceptual challenges that required careful analysis and iterative refinement before they were resolved.

1. Designing the disease guardrail system. Translating clinical dietary guidelines for chronic conditions into practical per-meal and per-day constraints required interpreting primary sources from AHA, WHO, and diabetes dietary guidance [1], [9], [10]. The initial design used hardcoded per-meal gram limits, which proved too rigid and frequently blocked recipe selection unnecessarily. The approach was revised to use a daily budget model, where the total permissible intake for a constrained nutrient is divided across all planned meals for the day. This change made the guardrail system both more clinically accurate and more practically usable.
2. Handling the calorie floor for weight-loss users. The weight-loss calorie target is calculated as a deficit below the user's TDEE. For users with a low TDEE particularly sedentary females, this deficit can produce a target below safe thresholds. Implementing and testing the calorie floor logic required careful consideration of how the floor interacts with the macronutrient allocation to ensure that protein, carbohydrate, and fat targets are recalculated consistently after the floor is applied.
3. Managing plan cycle integrity during preference edits. When a user edits their health metrics or dietary preferences after a plan has already been generated and partially logged, the system must decide whether to regenerate the plan immediately or defer changes to the next cycle. Implementing the cycle-protection logic — which checks whether any generated items have been logged before deciding between regenerate now and apply_next_fresh_cycle modes required coordinating state across the onboarding, diet, and workout services and ensuring that the frontend handled both response modes correctly.
4. AI output reliability and token budget management. The optional AI features particularly the custom workout calorie estimator and the recipe step generator, initially failed under a smaller token completion budget, returning incomplete JSON that caused parse errors. Increasing the token budget and adding robust fallback handling resolved the issue. This experience highlighted that AI-assisted features require defensive parsing and graceful degradation to remain reliable across different model versions and response sizes.

## 6.4 Objectives Evaluation

Objective 1 was achieved through the implemented logging and progress modules. Users can record planned and unplanned meals, workouts, water intake, and weight, while the system presents daily and weekly progress indicators based on stored logs.

Objective 2 was achieved through the rule-based diet recommendation engine. The system generates seven-day meal plans from user calorie and macronutrient targets, applies preference, allergy, and disease-related filters, displays recipe nutrition information, and supports meal swap impact previews.

Objective 3 was achieved through the rule-based workout recommendation engine. The system generates weekly workout plans using goal, preference, intensity, equipment, and duration inputs, estimates exercise energy expenditure using MET values, supports exercise search and swap flows, and evaluates workout output against goal-support expectations.

## 6.5 Concluding Remark

The evaluation shows that Flux met the main functional objectives of the development-based project. The implemented modules work together as an integrated planning, logging, and progress-monitoring system, while the test cases provide evidence that the major user workflows were completed and verified.

# Chapter 7 Conclusion and Recommendation

## 7.1 Conclusion

This project set out to design and develop Flux, a personalized diet and workout planner mobile application built as a Final Year Project. Three core objectives were established at the outset. The first objective was to deliver meal and exercise logging that accepts both planned and unplanned entries, with progress monitoring that evaluates user adherence and goal progress over time, supported by adaptive plan recommendations that update as users record their daily intake and activities. The second objective was to develop a rule-based diet recommendation system that generates personalized seven-day meal plans from calorie and macronutrient targets derived from user health profiles, while honoring dietary preferences, allergies, and disease-specific constraints such as hypertension, hyperlipidemia, and type 2 diabetes. The third objective was to develop a rule-based workout recommendation system that generates weekly workout plans by goal, preference, and intensity, supports flexible exercise modification through guided swap and search flows, and evaluates generated plans against WHO and ACSM physical activity guidelines [12], [33] to communicate goal-support quality to the user.

All three objectives were successfully delivered. The Flux application provides a complete end-to-end user experience from account registration through personalized plan generation, daily logging, progress monitoring, and profile management.

Beyond the minimum deliverables, Flux incorporates several features that elevate it above a basic logging application. The diet recommendation engine applies disease-specific daily budget guardrails derived from AHA, WHO, and diabetes dietary thresholds [1], [9], [10]. The workout recommendation engine assesses the generated plan against WHO and ACSM physical activity guidelines [12], [33] and communicates a goal-support rating to the user. AI-assisted features including meal photo scanning for nutrition estimation, AI-generated cooking steps for recipes without source instructions, an AI calorie estimator for unmatched custom workouts, and AI weekly insights summarizing the user's logged behavior, supplement the rule-based core and provide a richer user experience where an OpenAI-compatible model is available.

### 7.1.1 Lessons Learned

The development of Flux provided a broad and practical learning experience that covered full-stack mobile application development, evidence-based system design, and structured software engineering practices. Several of the most significant lessons are described below.

The importance of a clean layered architecture became apparent early in the project. Separating the backend into route handlers, controllers, services, and repositories made it straightforward to locate and change a single business rule without disturbing unrelated parts of the system. When the diet scoring algorithm or the workout goal-support rating needed to be revised, only the relevant service file required changes. This separation of concerns reduced the risk of regressions and made the codebase easier to reason about as it grew in size and complexity.

Working with evidence-based health and nutrition logic introduced a discipline of citing sources for every formula and threshold. The Mifflin-St Jeor equation for BMR estimation [4], the ISSN position stand for macronutrient targets [5], and AHA and diabetes dietary guardrails for chronic disease management [9], [10] all required consulting and interpreting primary literature before they could be implemented. This discipline made the recommendation logic transparent and verifiable, and highlighted the difference between a heuristic design decision and a clinically grounded rule.

State management in Flutter using Riverpod taught the value of separating UI state from business logic. Providers that manage API calls, error handling, and local caching keep individual screens lightweight and focused. The architecture also made it easier to write widget and provider unit tests, because each provider can be tested in isolation by overriding its dependencies.

Managing the optional AI integration required careful separation between the deterministic rule-based core and the probabilistic AI-assisted features. AI features are optional enhancements: the diet planner, workout planner, metrics calculation, logging, and progress tracking all function correctly without any AI model available. This design decision meant that the application could be fully tested and demonstrated without relying on an external API, and that any AI service failure degrades gracefully rather than breaking core functionality.

### 7.1.2 Advantages of the Project

Flux offers a number of advantages compared to a generic fitness or nutrition tracking application, arising from the evidence-based design principles applied throughout the project.

1. Evidence-based nutrition and exercise logic. Every formula, threshold, and scoring rule in the recommendation engine is grounded in a named clinical or academic source. The Mifflin-St Jeor equation is used for BMR estimation [4], ISSN protein targets guide the macronutrient allocation [5], and AHA, WHO, and diabetes dietary guidance defines the disease-specific dietary guardrails [1], [9], [10]. This approach ensures that the app's advice is medically reasonable rather than based on arbitrary values.
2. Disease-aware meal planning. The diet recommendation engine applies daily sodium, saturated fat, and sugar budgets for users with hypertension, hyperlipidaemia, or type 2 diabetes. These guardrails operate at the daily budget level rather than using hardcoded per-meal gram limits, which allows the engine to accommodate user meal preferences while still keeping total daily intake within clinically recommended ranges.
3. Transparent swap and impact preview. Before any meal or exercise swap is confirmed, the user is shown a before-and-after comparison of the nutritional or fitness impact. For diet swaps, the impact preview shows how each macro will change relative to the daily target. For workout swaps, the preview shows the effect on weekly energy expenditure, goal support rating, and muscle group coverage. This transparency gives the user informed control over their plan and avoids silent changes that could undermine their goals.
4. Goal-adaptive plan quality assessment. The workout recommendation engine evaluates every generated plan against WHO and ACSM physical activity guidelines [12], [33] and produces a support rating (for example, Strong, Moderate, or Insufficient) that is explained to the user in plain language. The rating changes if the user edits preferences or swaps exercises, giving real-time feedback on whether the plan still supports their stated goal.
5. Graceful integration of AI-assisted features. AI features such as meal photo scanning, recipe step generation, and weekly insight generation supplement the rule-based core without depending on it. The application remains fully functional if no AI model is available, and all AI outputs are presented as reviewable drafts that the user can modify before saving. This design prevents the application from blindly accepting AI-generated values and keeps the user in control of their own data.

## 7.2 Recommendation

While the current version of Flux meets all three FYP objectives, several enhancements have been identified that would further improve the application's functionality, accuracy, and user experience in future development cycles.

1. Barcode and nutritional label scanning. The current meal scan feature uses a general-purpose AI model to estimate nutrition from a food photograph. Adding a barcode scanner that queries a food product database such as Open Food Facts would provide exact nutritional values for packaged foods, improving the accuracy of the calorie and macro logging without relying on AI estimation.
2. Wearable device integration. Integrating with platforms such as Apple HealthKit or Google Fit would allow Flux to import step counts, heart rate data, and active energy burned directly from the user's wearable device. This would improve the accuracy of the daily energy balance calculation and reduce the manual effort required to log physical activity.
3. Adaptive plan regeneration based on logged adherence. The current plan regeneration is user initiated. A future enhancement could analyze the user's logging adherence over the current plan cycle and automatically suggest a plan adjustment. For example, switching from four meals to three if the user consistently skips the snack slot without requiring the user to manually edit their preferences through Machine Learning models.
4. Expanded disease and condition support. The current disease guardrail system supports hypertension, hyperlipidemia, and type 2 diabetes. Future development could extend this to include chronic kidney disease, irritable bowel syndrome, and coeliac disease, each with their own evidence-based dietary filters and daily budget constraints.

In summary, Flux demonstrates that a beginner-friendly, maintainable full-stack mobile application can be built to a mature product standard when evidence-based design principles are applied consistently, a clean layered architecture is maintained from the start, and all changes are verified against a real database rather than mocked data. The project has provided a thorough grounding in mobile and backend development, nutrition and exercise science, and structured software engineering practice, and leaves a solid foundation for the enhancements described above.

# REFERENCES

[1] World Health Organization, “Noncommunicable diseases,” WHO Fact Sheet. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/noncommunicable-diseases. Accessed: May 2, 2026.

[2] World Health Organization, “Physical activity,” WHO Fact Sheet. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/physical-activity. Accessed: May 2, 2026.

[3] R. Jakob, S. Harperink, A. M. Rudolf, M. Fleisch, S. Haug, J. L. Mair, A. Salamanca-Sanabria, and T. Kowatsch, “Factors influencing adherence to mHealth apps for prevention or management of noncommunicable diseases: Systematic review,” Journal of Medical Internet Research, vol. 24, no. 5, e35371, 2022, doi: 10.2196/35371.

[4] M. D. Mifflin, S. T. St Jeor, L. A. Hill, B. J. Scott, S. A. Daugherty, and Y. O. Koh, “A new predictive equation for resting energy expenditure in healthy individuals,” The American Journal of Clinical Nutrition, vol. 51, no. 2, pp. 241-247, 1990, doi: 10.1093/ajcn/51.2.241.

[5] R. Jäger et al., “International Society of Sports Nutrition Position Stand: protein and exercise,” Journal of the International Society of Sports Nutrition, vol. 14, no. 1, Art. no. 20, 2017, doi: 10.1186/s12970-017-0177-8.

[6] N. K. Fukagawa, J. M. Haytowitz, P. R. Pehrsson, A. Moshfegh, J. Harnly, and J. Finley, “USDA’s FoodData Central: what is it and why is it needed today?,” The American Journal of Clinical Nutrition, vol. 115, no. 3, pp. 619-624, 2022, doi: 10.1093/ajcn/nqab397.

[7] Edamam, “Recipe Search API Documentation,” Edamam Developer Portal. [Online]. Available: https://developer.edamam.com/edamam-docs-recipe-api. Accessed: May 2, 2026.

[8] National Heart, Lung, and Blood Institute, “DASH Eating Plan,” NHLBI. [Online]. Available: https://www.nhlbi.nih.gov/health/dash-eating-plan. Accessed: May 2, 2026.

[9] A. Reynolds and J. Mitri, “Dietary advice for individuals with diabetes,” in Endotext. South Dartmouth, MA, USA: MDText.com, Inc., updated Apr. 27, 2024. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK279012/. Accessed: May 2, 2026.

[10] American Heart Association, “Saturated fats,” AHA. [Online]. Available: https://www.heart.org/en/healthy-living/healthy-eating/eat-smart/fats/saturated-fats. Accessed: May 2, 2026.

[11] B. E. Ainsworth et al., “2011 Compendium of Physical Activities: A second update of codes and MET values,” Medicine & Science in Sports & Exercise, vol. 43, no. 8, pp. 1575-1581, 2011, doi: 10.1249/MSS.0b013e31821ece12.

[12] F. C. Bull et al., “World Health Organization 2020 guidelines on physical activity and sedentary behaviour,” British Journal of Sports Medicine, vol. 54, no. 24, pp. 1451-1462, 2020, doi: 10.1136/bjsports-2020-102955.

[13] C. E. Garber et al., “Quantity and quality of exercise for developing and maintaining cardiorespiratory, musculoskeletal, and neuromotor fitness in apparently healthy adults: Guidance for prescribing exercise,” Medicine & Science in Sports & Exercise, vol. 43, no. 7, pp. 1334-1359, 2011, doi: 10.1249/MSS.0b013e318213fefb.

[14] B. A. de Castro, S. M. Levens, M. Sullivan, and G. Shaw Jr., “Recommender systems use in weight management mHealth interventions: A scoping review,” Obesity Reviews, vol. 26, no. 3, e13863, 2025, doi: 10.1111/obr.13863.

[15] Flutter, “What is Flutter?,” Flutter Documentation. [Online]. Available: https://docs.flutter.dev/resources/faq#what-is-flutter. Accessed: May 2, 2026.

[16] Dart, “Dart overview,” Dart Documentation. [Online]. Available: https://dart.dev/overview. Accessed: May 2, 2026.

[17] Node.js, “About Node.js,” Node.js Documentation. [Online]. Available: https://nodejs.org/en/about. Accessed: May 2, 2026.

[18] Express, “Express - Node.js web application framework,” Express Documentation. [Online]. Available: https://expressjs.com/. Accessed: May 2, 2026.

[19] Oracle, “MySQL 8.0 Reference Manual,” MySQL Documentation. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/. Accessed: May 2, 2026.

[20] Y. Cai, M. Fan, and C. F. Harvey, “Health recommender systems development, usage, and evaluation from 2010 to 2022: A scoping review,” International Journal of Environmental Research and Public Health, vol. 19, no. 22, 15115, 2022, doi: 10.3390/ijerph192215115.

[21] J. N. Bondevik, K. E. Bennin, Ö. Babur, and C. Ersch, “A systematic review on food recommender systems,” Expert Systems with Applications, vol. 238, Art. no. 122166, 2024, doi: 10.1016/j.eswa.2023.122166.

[22] U.S. Food and Drug Administration, “Label claims for conventional foods and dietary supplements,” FDA. [Online]. Available: https://www.fda.gov/food/nutrition-food-labeling-and-critical-foods/label-claims-conventional-foods-and-dietary-supplements. Accessed: May 2, 2026.

[23] S. D. Herrmann et al., “2024 Adult Compendium of Physical Activities: A third update of the energy costs of human activities,” Journal of Sport and Health Science, vol. 13, no. 1, pp. 6-12, 2024, doi: 10.1016/j.jshs.2023.10.010.

[24] L. E. Burke, J. Wang, and M. A. Sevick, “Self-monitoring in weight loss: A systematic review of the literature,” Journal of the American Dietetic Association, vol. 111, no. 1, pp. 92-102, 2011, doi: 10.1016/j.jada.2010.10.008.

[25] P. Chotwanvirat, A. Prachansuwan, P. Sridonpai, and W. Kriengsinyos, “Advancements in using AI for dietary assessment based on food images: Scoping review,” Journal of Medical Internet Research, vol. 26, e51432, 2024, doi: 10.2196/51432.

[26] C. O’Hara, G. Kent, A. C. Flynn, E. R. Gibney, and C. M. Timon, “An evaluation of ChatGPT for nutrient content estimation from meal photographs,” Nutrients, vol. 17, no. 4, 607, 2025, doi: 10.3390/nu17040607.

[27] MyFitnessPal, “How do I use the Meal Planner?,” MyFitnessPal Help, updated Oct. 29, 2025. [Online]. Available: https://support.myfitnesspal.com/hc/en-us/articles/34603055097869-How-do-I-use-the-Meal-Planner. Accessed: May 2, 2026.

[28] Noom, “How Noom’s food color system works,” Noom Support. [Online]. Available: https://www.noom.com/support/faqs/question-topics/food-logging/2016/08/where-can-i-find-my-color-budget/. Accessed: May 2, 2026.

[29] Google Fitbit Help, “What should I know about Fitbit Premium?,” Fitbit Help Center. [Online]. Available: https://support.google.com/fitbit/answer/14237941. Accessed: May 2, 2026.

[30] WeightWatchers, “Diabetes support through Weight Watchers: Lose weight and lower your blood sugar,” WeightWatchers. [Online]. Available: https://www.weightwatchers.com/us/how-it-works/diabetes-program. Accessed: May 2, 2026.

[31] Google Play, “Calorie Counter by Lose It!,” Google Play Store. [Online]. Available: https://play.google.com/store/apps/details?id=com.fitnow.loseit. Accessed: May 2, 2026.

[32] World Health Organization, Obesity: Preventing and Managing the Global Epidemic: Report of a WHO Consultation, WHO Technical Report Series 894. Geneva, Switzerland: World Health Organization, 2000. [Online]. Available: https://iris.who.int/handle/10665/42330. Accessed: May 2, 2026.

[33] G. Liguori, Ed., ACSM's Guidelines for Exercise Testing and Prescription, 11th ed. Philadelphia, PA, USA: Wolters Kluwer, 2022.

[34] K. D. Hall, “What is the required energy deficit per unit weight loss?,” International Journal of Obesity, vol. 32, no. 3, pp. 573-576, 2008, doi: 10.1038/sj.ijo.0803720.

[35] U.S. Department of Agriculture and U.S. Department of Health and Human Services, Dietary Guidelines for Americans, 2020-2025, 9th ed. Washington, DC, USA, 2020. [Online]. Available: https://www.dietaryguidelines.gov/about-dietary-guidelines/previous-editions/2020-dietary-guidelines. Accessed: May 2, 2026.

[36] Institute of Medicine, Dietary Reference Intakes for Energy, Carbohydrate, Fiber, Fat, Fatty Acids, Cholesterol, Protein, and Amino Acids. Washington, DC, USA: National Academies Press, 2005. doi: 10.17226/10490.

[37] S. Li, “Food.com Recipes and Interactions,” Kaggle, 2019. [Online]. Available: https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions. Accessed: May 2, 2026.

# APPENDIX

## Poster


**xs**