AI NUTRITIONIST

A Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

BACHELOR OF SCIENCE (COMPUTER SCIENCE)

By Sarang Irfan Mustak

Seat No. _____

Under the esteemed guidance of

Prof. Javed Pathan

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(***Affiliated to University of Mumbai***)

MUMBAI – 400050

MAHARASHTRA

2025 – 2026

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated of University of Mumbai)

Mumbai, Maharashtra – 400050

DEPARTMENT OF COMPUTER SCIENCE

CERTIFICATE

This is to certify that the project entitled, **"AI NUTRITIONIST"**, is Bonafide work of **Sarang Irfan Mustak** bearing Seat No: _____ Roll No. **36,** submitted in **partial fulfilment of the requirements for the award of the degree of BACHELOR OF SCIENCE in COMPUTER SCIENCE from the University of Mumbai.**

Project Guide **HOD**

External Examiner

Date: _____ **College Seal**

ACKNOWLEDGEMENT

I would like to extend my sincere appreciation to the **Department of Computer Science** at **Rizvi College of Arts, Science, and Commerce** for providing me with the opportunity to undertake and complete this project dissertation. I am deeply grateful to our **Principal**, **Dr. S. Arunachalam**, for his exceptional leadership and effective management. I also wish to express my gratitude to the Head of the Department, **Professor Arif Patel**. His support in terms of providing essential resources and invaluable guidance throughout our course has been instrumental in the completion of this project. I would also like to convey my profound thanks to our project guide, **Professor Javed Pathan**. His mentorship and support have played an important role in the success of this project. Lastly, I am deeply appreciative of my dear parents for their unwavering support.

AI NUTRITIONIST APP
USING REACT NATIVE

RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

(Affiliated to University of Mumbai)

MUMBAI - MAHARASHTRA – 400050

DEPARTMENT OF COMPUTER SCIENCE

DECLARATION

I, **Sarang Irfan Mustak**, Roll No. **36,** hereby declare that the project synopsis entitled **"AI NUTRITIONIST"** is submitted for approval **for a Bachelor of Science** in Computer Science Sem V project for the academic year **2025-26**.

Signature of the Guide **Signature of the Student**

Place: _____

INTRODUCTION [1]

In today's world, people face many challenges in maintaining a healthy diet. Processed foods, hidden additives, and confusing food labels make it difficult to know whether daily food choices are truly safe or nutritious. Certifications can sometimes be misleading, and most individuals do not have the time or expertise to analyse detailed nutritional information. As a result, lifestyle-related health problems such as obesity, diabetes, and hypertension are increasing rapidly. The AI Nutritionist App is designed to address these issues by acting as a virtual dietitian that provides instant, reliable, and personalized guidance. Built using React Native and TypeScript, the app works on both Android and iOS platforms.

The app begins with an AI Diagnosis, where users enter details like age, height, weight, medical conditions, and allergies. Based on this data, it generates a personalized health profile. A Meal Planner then calculates daily needs such as calories, BMI, water intake, protein, carbohydrates, and fats, while also giving AI-based prompts to guide healthier eating.

The app also includes a Health Tips section, offering both general and personalized suggestions for improving lifestyle habits. Through the AI Scanner, users can scan product barcodes to get ingredient details, identify harmful additives, check health risks, and receive healthier alternatives.

By combining problem awareness with AI-powered solutions, the AI Nutritionist App empowers users to make informed food choices, build sustainable eating habits, and reduce the risk of diet-related diseases.

OBJECTIVES [2]

The objectives of the AI Nutritionist App are:

- To provide personalized health insights using AI: By analyzing user details such as age, height, weight, medical conditions, and allergies, the app helps users understand their current health status and make informed dietary choices.
- To offer a Meal Planner: By tracking calories, BMI, water intake, and nutrients like proteins, carbohydrates, and fats, the app guides users to maintain a balanced diet tailored to their needs.
- To generate AI-based dietary suggestions: The app provides actionable prompts to help users plan meals effectively, ensuring nutritional requirements are met.
- To deliver general and personalized health tips: By educating users on healthy habits and providing recommendations based on individual profiles, the app encourages sustainable lifestyle improvements.
- To scan and analyze food products: Users can check ingredients, detect harmful additives, evaluate health impacts, and receive suggestions for healthier alternatives, helping them make safer food choices.
- To raise awareness about healthy eating: The app empowers users to understand the impact of their daily food choices, aiming to reduce risks of nutrition-related diseases.
- To build a scalable and adaptable platform: The modular design allows for future enhancements such as pediatric guidance, bill scanning, and voice assistant support, ensuring long-term relevance and usability.

SCOPE [2]

The AI Nutritionist App aims to provide a comprehensive solution for personalized nutrition and health management. The scope of the project includes:

- Personalized Health Assessment: Evaluates users' current health status using AI, based on age, height, weight, medical conditions, and allergies.
- Meal Planning and Dietary Guidance: Calculates daily requirements for calories, BMI, water intake, proteins, carbohydrates, and fats, and provides AI-based prompts for balanced meals.
- Health Tips and Awareness: Offers both general and personalized recommendations to encourage healthier lifestyle choices and better eating habits.
- Food Product Analysis: Enables users to scan barcodes or input food details to identify ingredients, harmful additives, health risks, and healthier alternatives.
- Cross-Platform Support: Works on both Android and iOS devices, ensuring accessibility for a wide range of users.
- Future Scalability: Designed as a modular system, allowing easy integration of additional features like pediatric guidance, bill scanning, and voice-assistant support.
- Educational Value: Raises awareness about the impact of daily dietary choices on long-term health and encourages informed decision-making.

METHODOLOGY [3]

The development of the AI Nutritionist App followed the Spiral Model, an iterative, risk-driven process that refines the system through repeated cycles. Each phase ensured thorough design, implementation, and testing before moving forward.

1. Planning Phase: Project requirements were gathered and analyzed. Core objectives included personalized health assessment, meal planning, AI-based dietary suggestions, health tips, and barcode scanning. Technologies such as React Native, TypeScript, MongoDB, and the Open Food Facts API were selected.
2. Risk Analysis Phase: Key risks were identified, including inaccurate AI predictions, unreliable nutritional data, and cross-platform issues. Mitigation strategies involved algorithm validation, database verification, and multi-device testing.
3. Engineering Phase: The system was built iteratively in modules:
   - AI Diagnosis: Generates personalized health insights from user data.
   - Meal Planner: Calculates daily nutrition requirements (calories, BMI, water, macros) and provides AI-driven diet prompts.
   - Health Tips: Delivers lifestyle and nutrition advice tailored to users.
   - AI Scanner: Processes barcode data to detect harmful additives, analyze nutrition, and suggest healthier alternatives.
4. Evaluation Phase: Each module underwent iterative testing for accuracy, usability, and performance on Android and iOS. Feedback from test users refined functionality and recommendations.
5. Next Spiral Planning: Future improvements were outlined, including pediatric guidance, shopping bill scanning, and voice assistant integration.

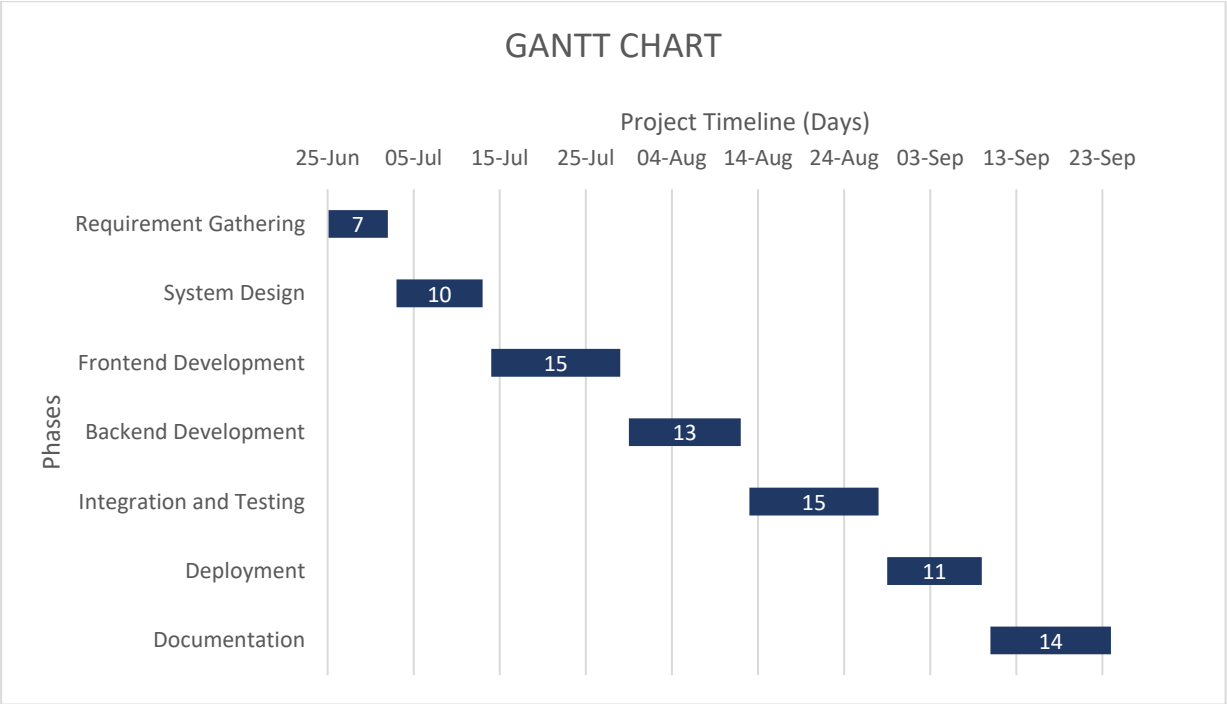
By applying the Spiral Model, the AI Nutritionist App achieved continuous improvement, effective risk management, and user-focused development, resulting in a scalable and reliable personalized nutrition solution.

TOOLS AND TECHNOLOGIES

The development of the AI Nutritionist App involved the use of modern tools and technologies to ensure cross-platform functionality, scalability, and efficient development. The main tools and technologies used are as follows:

1. React Native: A cross-platform mobile development framework used to build the app for both Android and iOS platforms. React Native allows fast development and reusable components.
2. TypeScript: A strongly-typed programming language used for building robust and maintainable code. TypeScript helps in detecting errors during development and improves code quality.
3. Node.js & Express: Express, a Node.js framework, was used to develop the backend server for handling API requests, managing data flow, and ensuring seamless communication between the app and the database.
4. MongoDB: A NoSQL database used to store user profiles, nutrition data, and scanned food information. MongoDB provides flexibility, scalability, and fast access to large datasets.
5. ADB (Android Debug Bridge): Used for connecting and testing the app on mobile devices over Wi-Fi, allowing real-time debugging and performance checks during development.
6. npm Modules: Various npm (Node Package Manager) modules were used to add functionality, manage dependencies, and simplify development tasks, including packages for UI components, API handling, and barcode scanning.
7. Git: A version control system used to track changes in the codebase, manage collaborative development, and maintain project history.

TIMELINE



Phase	Start Date	End Date	Duration (Days)
Requirement Gathering	25-Jun-2025	02-Jul-2025	7
System Design	03-Jul-2025	13-Jul-2025	10
Frontend Development	14-Jul-2025	29-Jul-2025	15
Backend Development	30-Jul-2025	12-Aug-2025	13
Integration & Testing	12-Aug-2025	28-Aug-2025	15
Deployment	29-Aug-2025	09-Sep-2025	11
Documentation	10-Sep-2025	24-Sep-2025	14

RESOURCES [5]

1. Human Resources: The project was developed by a single developer who handled design, coding, testing, and documentation with guidance from the project guide. This ensured complete ownership and consistency throughout the project.

2. Hardware Resources: A laptop/computer with at least 8GB RAM, a good processor, and stable internet was used. A mobile device was required for testing the app. These resources provided a reliable setup for both development and real-time testing.

3. Software Resources: The app was built using React Native and TypeScript for the frontend, Node.js and Express.js for the backend, and MongoDB as the database. Git and GitHub were used for version control, and ADB for mobile testing. Tools like NPM modules, Visual Studio Code, Android Studio, and Postman supported development and testing. This combination of tools made the development process smooth and efficient.

4. Content and Information Resources: Food datasets such as Open Food Facts, research papers, and health guidelines like BMI charts and WHO recommendations were used. Custom health tips and awareness content were also included. These sources ensured the app delivered accurate and reliable health information.

5. Collaborative Resources: GitHub was used for managing the code, while Google Docs and MS Word were used for documentation. Discussions with the guide provided feedback. Such collaboration helped in maintaining progress and improving the quality of the work.

6. Time and Timelines: The project was completed in about 10–12 weeks, with stages covering research, design, development, testing, and documentation. Breaking the work into phases ensured timely delivery and better organization.

By ensuring access to these necessary resources, the project was successfully developed and completed within the planned timeline.

EXPECTED OUTCOMES

The AI Nutritionist App is designed to provide the following outputs for the user:

1. Personalized Health Assessment: Based on user details like age, height, weight, medical conditions, and allergies, the app gives an easy-to-understand overview of the user's current health status.

2. Meal Planning and Diet Suggestions: The app calculates daily needs such as calories, BMI, water intake, and nutrients (proteins, carbs, fats) and provides simple AI-based prompts to help users eat a balanced diet.

3. Food Product Analysis: When a product barcode is scanned or food details are entered, the app shows ingredient information, highlights harmful additives, and suggests healthier alternatives in an easy way to understand.

4. Health Tips: The app offers both general and personalized advice to improve lifestyle and encourage healthy eating habits.

5. User-Friendly Interface: All outputs are shown clearly so that users can easily understand and follow the recommendations without confusion.

6. Cross-Platform Functionality: The app works on both Android and iOS devices, allowing anyone to access health guidance on their mobile device.

7. Awareness and Education: The app helps users learn about healthy eating, processed food risks, and better nutrition choices for long-term health.

ADVANTAGES

- Personalized Health Guidance: Provides tailored recommendations based on user details such as age, height, weight, medical conditions, and allergies.
- Balanced Diet Support: Helps users plan meals by calculating calories, BMI, water intake, and nutrient distribution (proteins, carbs, fats).
- Food Awareness: Analyzes ingredients and additives in products through barcode scanning, helping users make safer and healthier choices.
- Health Education: Offers general and personalized tips, raising awareness about processed foods, nutrition, and healthy lifestyle habits.
- User-Friendly Interface: Simple and clear design allows users to easily understand and follow recommendations.
- Time and Effort Saving: Provides instant dietary insights and meal planning, eliminating the need for manual calculations or research.
- Scalable and Upgradable: Modular design allows adding future features like pediatric guidance, bill scanning, and voice assistant support.

LIMITATIONS

- Not a Medical Replacement: The app provides health and nutrition guidance but cannot replace professional medical advice or consultations.
- Data Dependence: Accuracy depends on the quality of food datasets and APIs used; incomplete or incorrect data may affect results.
- Limited Scope: Features like teleconsultation, wearable device integration, or advanced AI analysis are not included in the current version.
- Device Dependency: The app requires a smartphone with sufficient hardware and internet access for full functionality.
- User Input Accuracy: Health assessment depends on accurate user-provided information (age, weight, allergies); wrong data can affect recommendations.
- Regional Limitations: Some food products or ingredients may not be available in the database, especially for local or regional brands.

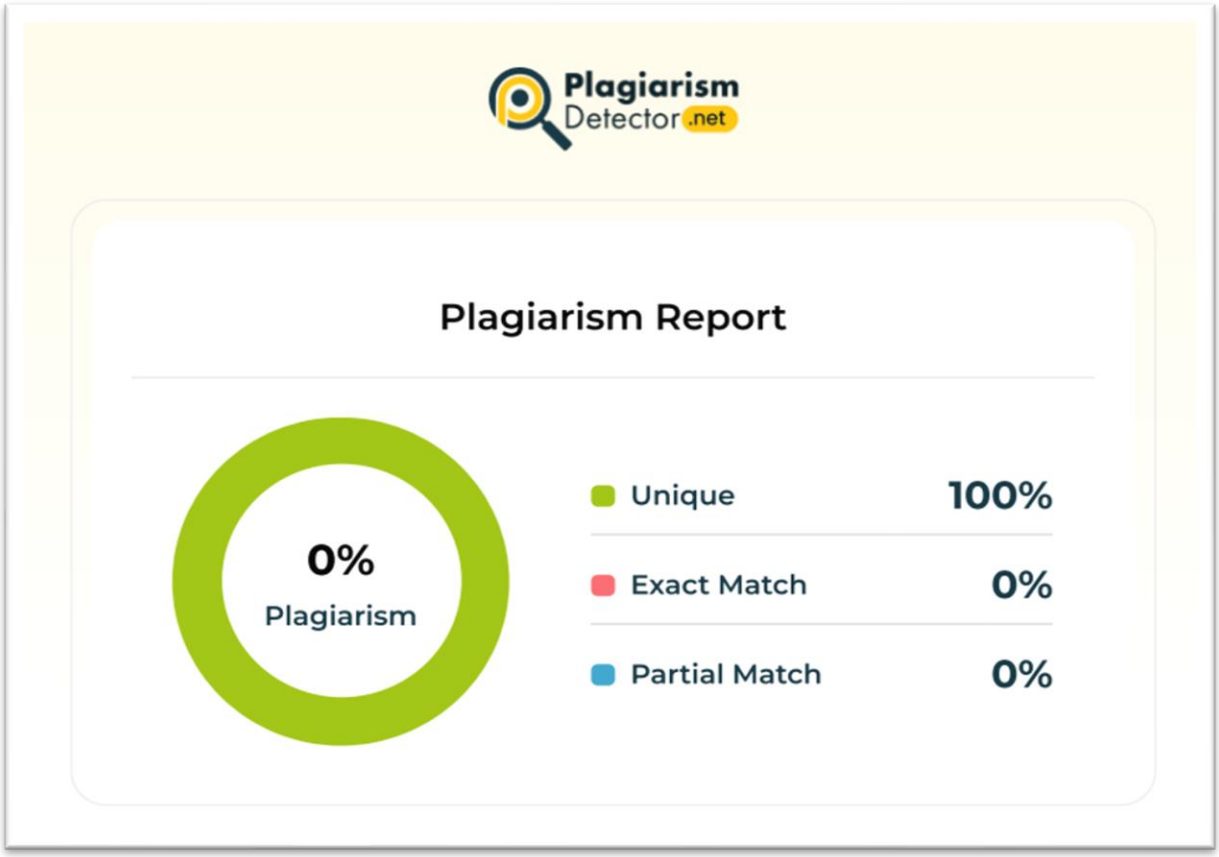
REFERENCES

1. Introduction: https://www.who.int/news-room/fact-sheets/detail/healthy-diet
2. Objectives and Scope: https://nutritionsource.hsph.harvard.edu/
3. Methodology: https://www.geeksforgeeks.org/software-engineering/software-engineering-spiral-model/
4. Gantt Chart: https://www.youtube.com/watch?v=oc6zX6vEWrY
5. Resources: https://world.openfoodfacts.org/data

PLAGIARISM REPORT

A plagiarism report is a document or a summary that provides information about the presence of plagiarism in a piece of written or academic work. Plagiarism refers to the act of using someone else's words, ideas, or work without proper attribution or permission, presenting them as your own. Plagiarism is considered unethical and can have serious consequences, particularly in academic and professional settings.

A plagiarism report is typically generated by plagiarism detection or services. It scans a given document or text for similarities to existing sources, such as published articles, books, websites, and other written material. When the software identifies matching or highly similar content, it highlights or marks the specific passages that may be considered plagiarized.



DECLARATION

I hereby declare that the project entitled, "**AI NUTRITIONIST**" done at **Rizvi College of Arts, Science and Commerce**, has not been in any case duplicated to be submitted to any other university for the award of any degree. To the best of my knowledge, other than me, no one has submitted this to any other university.

The project is done in partial fulfillment of the requirements for the award of the degree of **BACHELOR OF SCIENCE (COMPUTER SCIENCE)** to be submitted as 5th semester project as part of our curriculum.

Sarang Irfan Mustak

ABSTRACT

Maintaining a healthy diet is increasingly difficult due to processed foods, hidden additives, confusing labels, and misleading certifications. Many people lack the time or expertise to analyse nutritional information, leading to lifestyle-related health issues such as obesity, diabetes, and hypertension. The AI Nutritionist App addresses these challenges by acting as a virtual dietitian, providing instant, reliable, and personalized guidance.

Users begin by creating a profile, with each account supporting up to three individual profiles. The AI Diagnosis uses profile information—including age, height, weight, medical conditions, and allergies—to generate a personalized health assessment. Based on this, the app creates a diet plan that calculates daily requirements such as calories, BMI, water intake, and nutrients (proteins, carbohydrates, fats), and provides AI-based prompts and recommendations for balanced and healthy eating. The Health Tips section offers both general and personalized advice to improve lifestyle habits.

Through the AI Scanner, users can scan product barcodes to access ingredient details, detect harmful additives, assess health risks, and receive healthier alternatives. Built using React Native and TypeScript, the app is cross-platform, running on both Android and iOS devices. By combining AI-powered health assessment, personalized diet planning, and nutritional awareness, the app empowers users to make informed food choices, develop sustainable eating habits, and reduce the risk of diet-related diseases.

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

1.1 Introduction

In recent years, Artificial Intelligence (AI) has transformed healthcare by enabling preventive care, personalized treatment, and lifestyle management. Nutrition, being central to health and disease prevention, is one of the most impactful areas. Poor diets are strongly linked to chronic conditions like obesity, diabetes, heart disease, and hypertension.

An AI-based nutritionist addresses this challenge by analyzing individual health profiles and offering tailored diet recommendations. Unlike generic charts, it considers factors such as medical conditions, allergies, preferences, and lifestyle to deliver personalized, science-backed guidance. This not only supports fitness goals but also helps in managing existing health issues effectively.

1.2 Problem Statement

Maintaining proper nutrition is a major challenge as most people rely on generic diet plans, lack access to dieticians, or struggle with confusing food labels and hidden additives. This often leads to poor dietary management, nutritional deficiencies, and rising lifestyle diseases such as obesity, diabetes, and hypertension. Existing mobile health apps provide generalized advice without considering individual health conditions, allergies, or evolving needs, and rarely integrate AI-driven diagnosis for early risk detection. Hence, there is a need for an affordable, accessible, and intelligent AI-based system that delivers personalized nutrition guidance, quick insights through features like barcode scanning, and preventive recommendations to help individuals and families adopt healthier, long-term eating habits.[1]

1.3 Aim

The aim of this project is to design and develop an AI-based Nutritionist System that provides personalized dietary recommendations, AI-assisted diagnosis, and preventive nutrition guidance to improve overall health and well-being. The system will allow users to create multiple profiles (up to three per account), enabling family-wide usage. It will analyze health parameters such as age, weight, dietary preferences, allergies, and medical conditions to

generate customized diet plans while also supporting early diagnosis of nutritional deficiencies and potential risks. Beyond personalization, the project aims to empower users to make healthier food choices, reduce the risk of lifestyle-related diseases such as obesity, diabetes, and hypertension, and promote sustainable eating habits for long-term wellness. Additionally, it seeks to bridge the gap between dieticians and the general public by offering expert-like advice through AI and to provide a scalable, adaptable solution that evolves with more datasets, advanced features, and AI improvements in the future.[2]

1.4 Objectives

The main objective of the AI Nutritionist project is to provide users with a personalized and reliable nutrition assistant. The app is designed to create customized health profiles, offer AI-driven diagnosis, and generate diet plans based on factors such as age, weight, height, medical conditions, and allergies. Another important objective is to make nutrition guidance accessible for families by allowing multiple profiles under a single account. The project also aims to integrate a barcode scanner that can analyze food ingredients, detect harmful additives, and suggest healthier alternatives. Along with this, the app provides a meal planner to calculate calories, BMI, water intake, and essential nutrients, as well as AI-based lifestyle and health tips. Overall, the project seeks to make healthy living simple, affordable, and user-friendly.[3]

CHAPTER 2. REQUIREMENTS SPECIFICATION

2.1 Introduction

The Requirement Specification defines the essential features, capabilities, and constraints of the AI Nutritionist App. It ensures that the system is designed, developed, and implemented according to user needs and expectations. The specifications are divided into functional and non-functional requirements, along with hardware and software requirements.

2.2 Functional Requirements

Functional requirements describe what the system should do:

- User Profile Management: Users can create an account and manage up to three individual profiles. Each profile stores details like age, height, weight, medical conditions, allergies, and dietary preferences.
- AI Diagnosis: The system performs AI-based health assessments using profile information. It identifies potential nutritional deficiencies and health risks, providing personalized guidance tailored to each user.
- Meal Planner: The Meal Planner calculates daily calorie needs, BMI, water intake, and essential nutrients like proteins, carbohydrates, and fats. It also provides AI-generated prompts and recommendations to help users maintain balanced diets.
- Health Tips: The app offers both general and personalized tips for improving eating habits and overall lifestyle.
- Food Analysis using Barcode: Through barcode scanning, the system retrieves detailed information about food products, detects harmful additives, assesses health risks, and suggests healthier alternatives.
- Data Storage and Retrieval: All user profiles, diet plans, and scan history are securely stored and efficiently retrieved when needed, ensuring smooth functionality and data integrity.

2.3 Non-Functional Requirements

Non-functional requirements define how the system should perform:

- Usability: The system must have a user-friendly interface with clear navigation, ensuring an intuitive experience for all users.
- Performance: The app should respond quickly during AI diagnoses, barcode scanning, and diet plan generation to maintain smooth functionality.
- Reliability: The system must provide accurate health assessments and diet suggestions consistently.
- Scalability: The app should handle an increasing number of users and allow future expansion of features.
- Security: Sensitive user data must be securely stored and protected from unauthorized access.
- Maintainability: The system should be easy to update, including improvements to databases, AI algorithms, and app features.

2.4 Hardware Requirements

- User Side: Users need a smartphone or tablet running Android or iOS to access the app.
- Developer Side:
    - Processor: Intel i3 / AMD equivalent
    - RAM: 8 GB
    - Storage: 512 GB HDD
    - Display: 1366 x 768 resolution
    - OS: Windows
    - Internet connection for package installation

2.5 Software Requirements

- Development Environment: The app is developed using React Native and TypeScript.
- Database and Backend: MongoDB is used for storing user and product information, while Express.js handles backend API requests.

- Additional Tools: Dependencies are managed using NPM modules, and Git is used for version control. Developers can use IDEs like VS Code for coding and testing the system.

2.6 User Requirements

Users should be able to easily create and manage profiles and receive personalized AI-based diet and health guidance. The barcode scanner should provide quick insights into food products, while health tips offer actionable advice to improve lifestyle habits. Overall, the system should be fast, reliable, and secure, functioning efficiently on mobile devices for daily use.

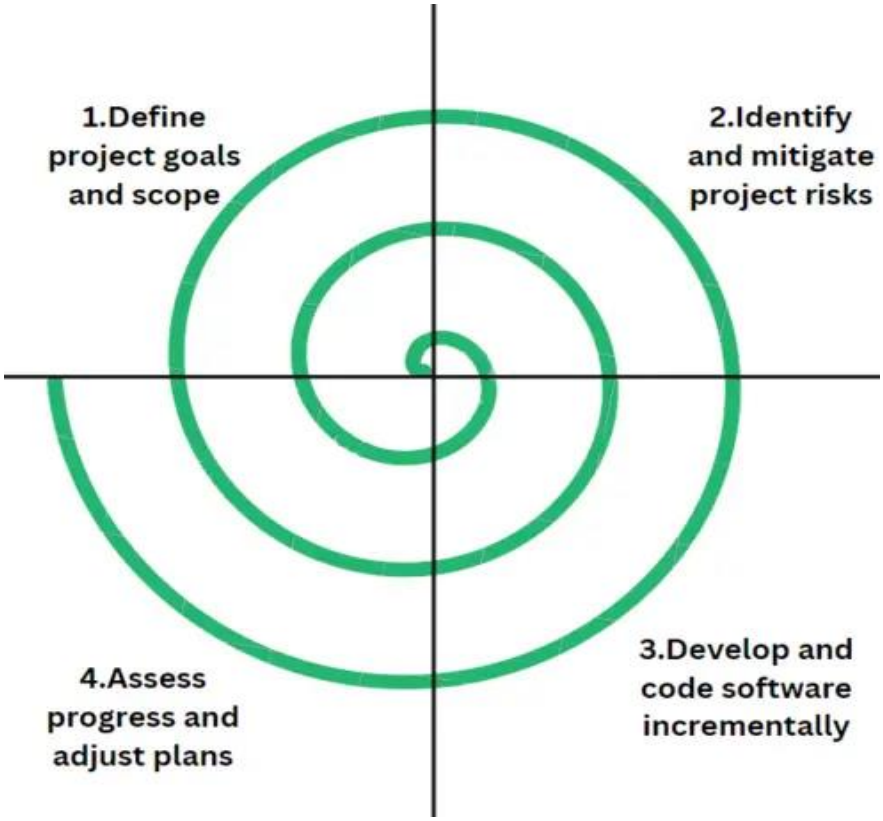
2.7 Methodology

The development of the AI Nutritionist App follows the Spiral Model, which combines iterative development with continuous risk analysis for a structured yet flexible approach. In the Planning phase, objectives, scope, and user requirements were defined. The Risk Analysis phase identified potential issues such as inaccurate AI predictions, database errors, and performance challenges, with mitigation strategies prepared accordingly. During Development, core modules—AI Diagnosis, Meal Planner, Health Tips, and AI Scanner—were built iteratively using React Native and TypeScript for the frontend, Express.js for the backend, and MongoDB for data storage. Testing ensured module-level and system-wide accuracy, usability, and performance, with feedback integrated into further iterations. Finally, the Deployment and Maintenance phase delivered the app on Android and iOS platforms, with regular updates planned for databases, AI algorithms, and features to maintain scalability, reliability, and accuracy.

2.8 Spiral Model

The Spiral Model is a software development methodology that combines iterative development with systematic risk analysis. Unlike traditional linear models, it emphasizes repeated cycles (or spirals) of planning, development, testing, and evaluation. Each cycle allows developers to refine the system, address risks early, and incorporate user feedback, making it suitable for complex projects like the AI Nutritionist App.

The Spiral Model consists of four main activities in each iteration: Planning, Risk Analysis, Development, and Evaluation. In the planning stage, objectives, requirements, and resources are defined. Risk analysis identifies potential problems and outlines mitigation strategies. Development involves designing, coding, and implementing system modules, while evaluation ensures the deliverables meet requirements and function correctly.[4]



CHAPTER 3. SYSTEM ANALYSIS

3.1 Introduction

System analysis involves understanding the requirements, structure, and functionality of the system before its design and implementation. For the AI Nutritionist App, this process helps identify user needs, functional modules, and potential challenges to ensure the development of a reliable, efficient, and user-friendly system.

3.2 Feasibility Study

A feasibility study was conducted to assess the practicality of the project from three perspectives:

- Technical Feasibility: The project uses technologies like React Native, TypeScript, Express.js, and MongoDB, which are suitable for building a cross-platform AI-based nutrition app. AI algorithms can be integrated to provide personalized diet and health recommendations.
- Economic Feasibility: The project requires minimal hardware and software costs, mainly for development and testing, making it cost-effective. Open-source tools and databases reduce expenses further.
- Operational Feasibility: Users can easily interact with the app to create profiles, receive AI-based health assessments, plan meals, and scan food products, ensuring smooth and practical usage.[5]

3.3 System Requirements Analysis

The system analysis identified the following requirements:

- Functional Requirements: User profile management, AI diagnosis, meal planning, health tips, barcode scanning, data storage, and cross-platform compatibility.
- Non-Functional Requirements: Usability, performance, reliability, security, scalability, and maintainability.

- Hardware and Software Requirements: Smartphones/tablets for users, computers for development, React Native, TypeScript, MongoDB, Express.js, NPM modules, and Git for version control.

3.4 Data Flow and System Modules

The system is divided into modules to ensure clarity and modular development:

1. User Profile Module: Allows creating and managing up to three profiles per account.
2. AI Diagnosis Module: Generates health assessments based on profile data.
3. Meal Planner Module: Calculates calories, BMI, nutrients, and water intake.
4. Health Tips Module: Provides general and personalized lifestyle advice.
5. AI Scanner Module: Scans food products, identifies harmful additives, and provides healthier alternatives.

Data flows from user inputs through the AI modules, generating insights and recommendations that are stored and displayed in the app.

3.5 Problem Analysis

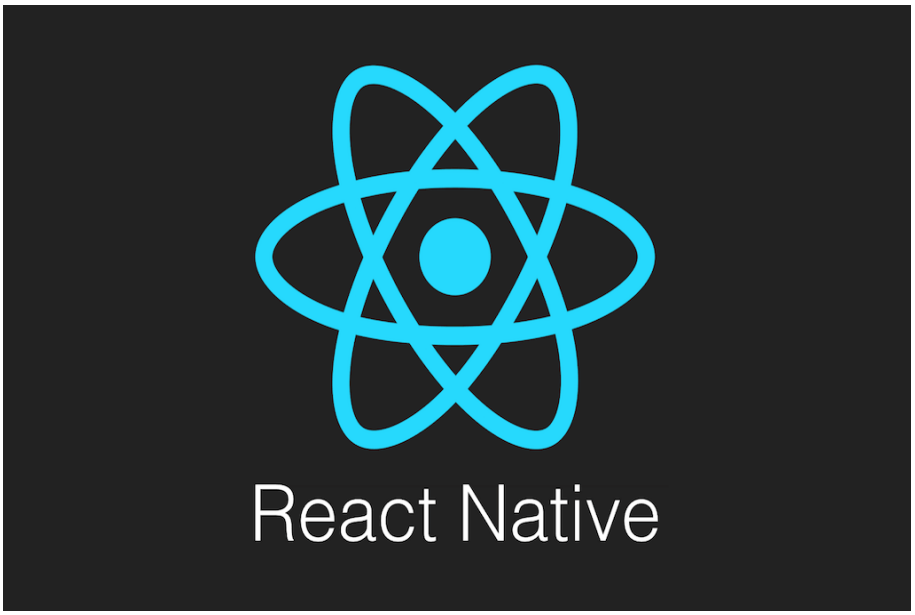
Current challenges in nutrition include confusing food labels, hidden additives, and lack of personalized guidance. The AI Nutritionist App addresses these issues by providing instant, AI-powered diet recommendations, food analysis, and health tips. Potential risks, such as incorrect AI predictions or performance delays, were identified and mitigated through testing and iterative development.

CHAPTER 4. SURVEY OF TECHNOLOGY

4.1 React Native

React Native is an open-source framework developed by Facebook that allows developers to build cross-platform mobile applications using a single codebase. With React Native, the same code can run on both Android and iOS devices, which reduces development time and effort compared to building separate native apps for each platform.

React Native uses JavaScript (or TypeScript) and allows developers to create mobile apps with a native look and feel by rendering components using native APIs. It also provides hot-reloading, which enables developers to see changes instantly without rebuilding the entire app, making development faster and more efficient.[6]



4.2 TypeScript

TypeScript is an open-source programming language developed by Microsoft that is a superset of JavaScript. This means it includes all features of JavaScript but adds additional capabilities, such as static typing, interfaces, and type checking at compile time.[7]

Using TypeScript helps developers catch errors early in the development process, before running the code, which improves the reliability and maintainability of applications. It also makes the code more readable and structured, especially for large-scale projects like the AI Nutritionist App.

4.3 Node.js and Express

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript on the server side. It is built on Google Chrome's V8 engine and is designed for high-performance, scalable, and event-driven applications. Node.js is particularly suitable for applications that require handling multiple requests simultaneously, like the AI Nutritionist App.

Express.js is a lightweight and flexible web application framework built on top of Node.js. It simplifies backend development by providing tools and features to handle routing, HTTP requests, middleware, and APIs efficiently. Express.js helps developers create RESTful APIs, which allow the frontend (React Native app) to communicate seamlessly with the backend and the database.

4.4 MongoDB

MongoDB is an open-source NoSQL database that stores data in a flexible, document-oriented format called BSON, which is similar to JSON. Unlike traditional relational databases, it does not require a fixed schema, making it ideal for handling dynamic and unstructured data. MongoDB provides fast read and write operations, scalability, and seamless integration with Node.js and Express.js, which is essential for real-time applications. In the AI Nutritionist App, MongoDB is used to store user profiles, diet plans, scan history, and product information. Its flexibility allows the app to manage multiple profiles per user, store detailed food and nutritional data, and retrieve information quickly to generate accurate AI-based recommendations.

CHAPTER 5. SYSTEM DESIGN

5.1 Introduction

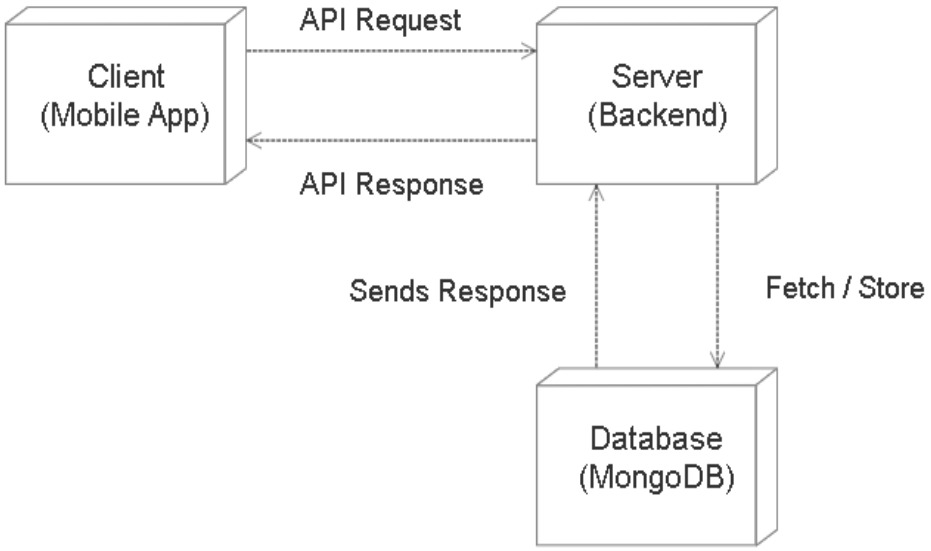
System Design is the process of defining the architecture, components, and interactions of the system to ensure it meets the specified requirements. It provides a blueprint for how the system will function and how different modules will communicate with each other.

In the AI Nutritionist App, system design plays a crucial role in translating user needs, such as personalized diet planning, AI diagnosis, barcode scanning, and health tips - into a structured model. The design outlines how the client (mobile app) interacts with the backend server, and how the server communicates with the database to deliver real-time results to users.

This stage also involves the use of UML diagrams such as use case diagrams, activity diagrams, sequence diagrams, and deployment diagrams to represent the system from different perspectives. By focusing on scalability, security, and usability, the system design ensures that the app can provide accurate insights, maintain user trust, and adapt to future improvements.[8]

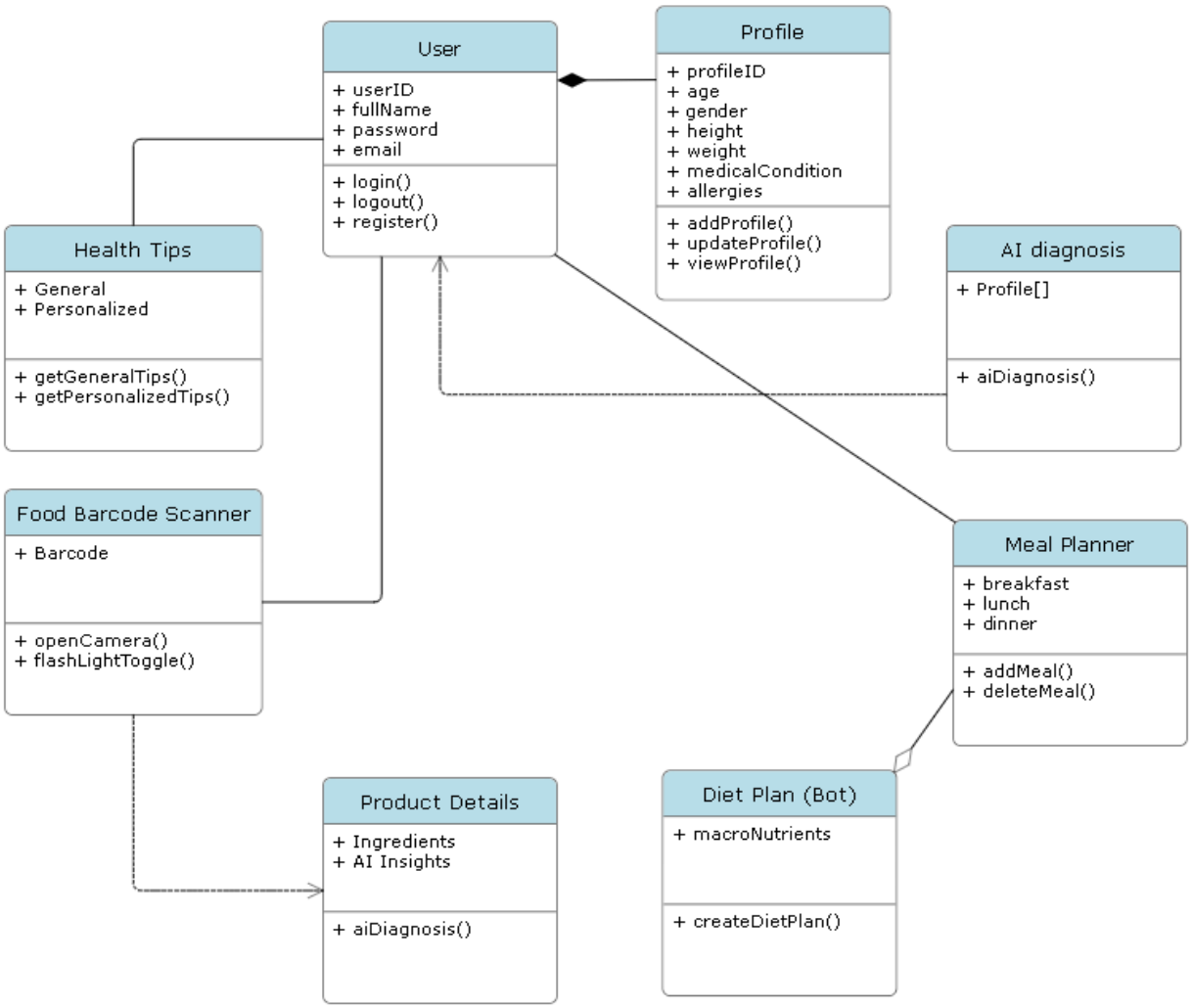
5.2 Client – Server Model

The AI Nutritionist App follows a client–server architecture, where the mobile application acts as the client and communicates with the backend server through API requests. The backend processes these requests, interacts with the database, and returns the appropriate responses to the client.



5.3 Class Diagram

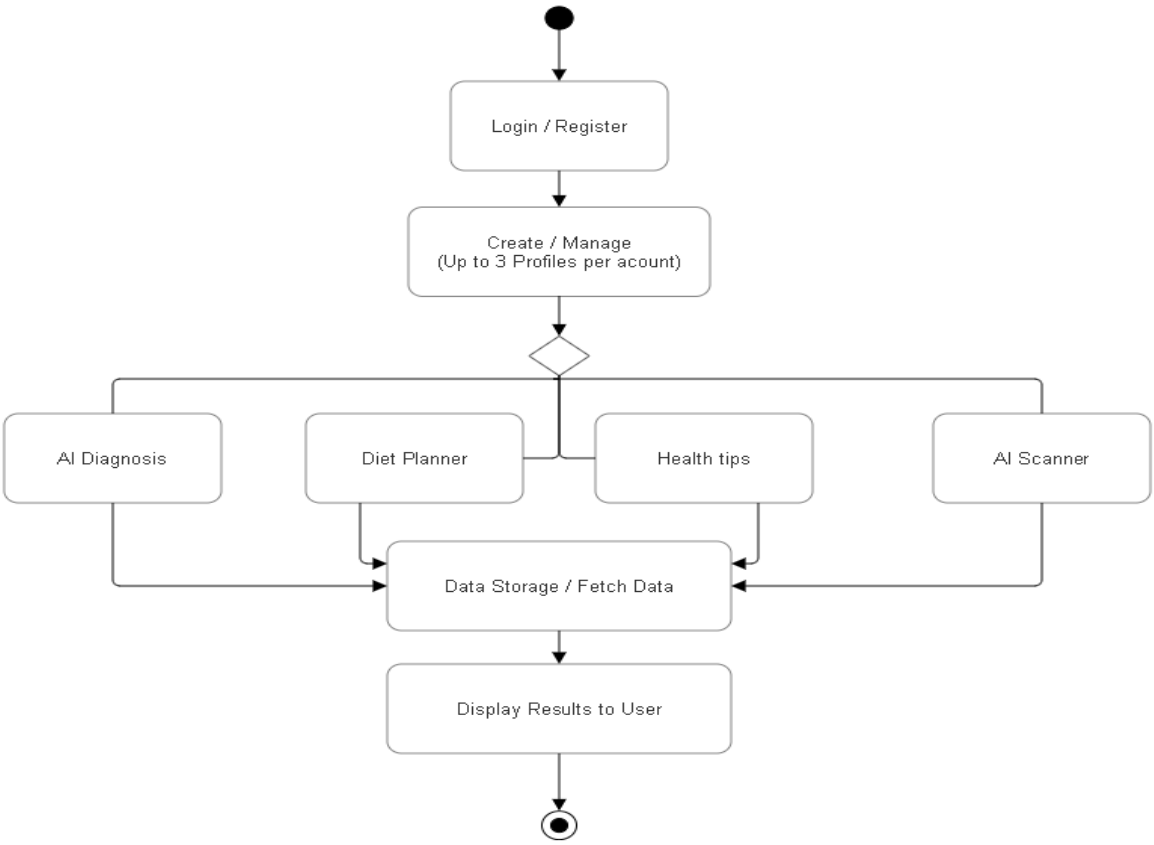
The class diagram of the AI Nutritionist App shows the main parts of the system and how they connect. The User class manages up to three Profiles, each storing details like age, weight, and medical conditions. The AI Diagnosis class uses this profile data to create health assessments. The Meal Planner class generates a Diet Plan, which includes daily calorie and nutrient needs. The Health Tips class provides both general and personalized lifestyle advice to the user. The Barcode Scanner class lets users scan food items and connects with the Food Product class to show ingredients, risks, and alternatives. Together, these classes work to give users personalized diet guidance in a clear and organized way.



5.4 Activity Diagram

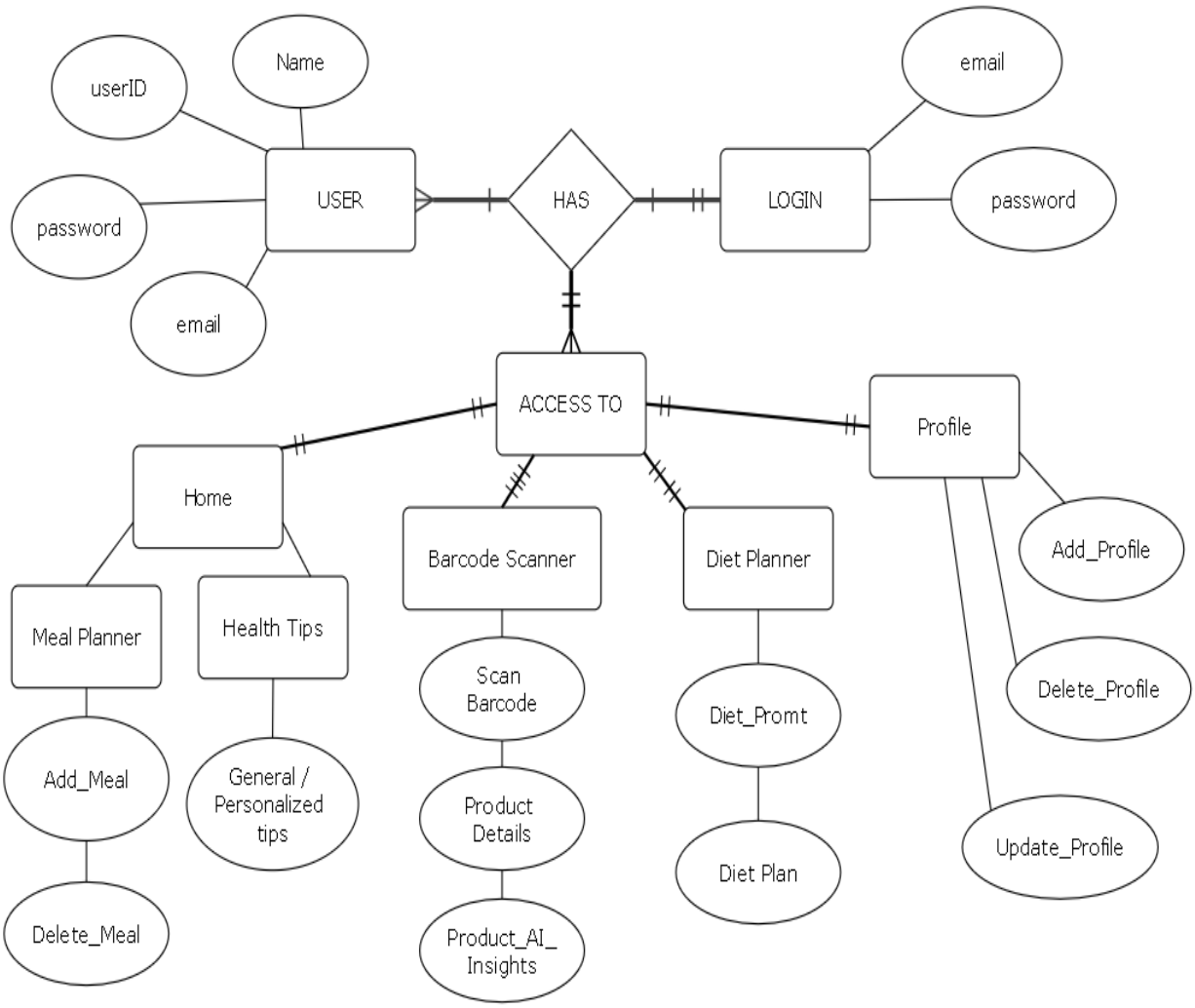
The activity diagram explains how users interact with the system step by step. It begins at the Start node, where the user either logs in or registers. Once logged in, the user manages their profile, where they can create and update health details such as age, weight, height, medical conditions, and allergies. From here, the system gives the user a choice of services through a decision node. The user can select AI Diagnosis, which analyses profile data and generates a health assessment. They may choose the Meal Planner, which calculates calorie needs, BMI, and nutrients to create a diet plan. Another option is Health Tips, where users receive both general and personalized lifestyle advice. Alternatively, the user can use the AI Scanner, which scans product barcodes to show ingredients, harmful additives, and healthier alternatives.

After completing any of these activities, all paths merge into a common flow where results are processed and stored in the database. Finally, the system displays the output back to the user, and the process ends at the End node.



5.5 Entity-Relationship (ER) Diagram

An ER (Entity-Relationship) diagram is a simple visual tool used to represent the data in a system and the relationships between them. Entities represent objects or concepts, attributes describe the properties of those entities, and relationships show how entities are connected. Cardinality indicates how many instances of one entity can relate to another. ER diagrams are used to organize data clearly and help design databases efficiently.



CHAPTER 6. SYSTEM IMPLEMENTATION

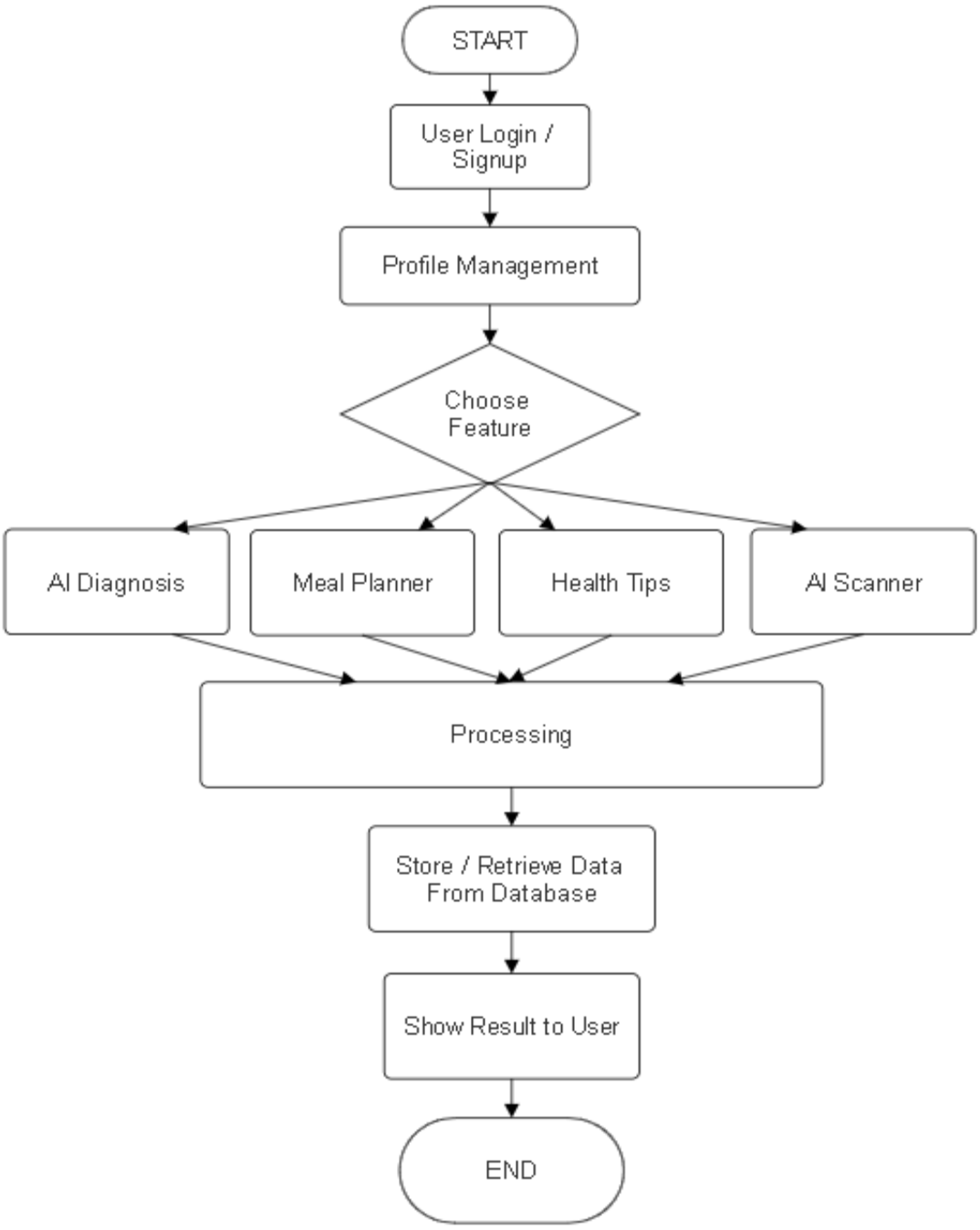
6.1 Introduction

System implementation is a critical phase in the software development life cycle where theoretical designs and plans are transformed into a fully functional system. Unlike the planning and design stages, this phase focuses on practical execution, bringing the project to life. It involves writing the actual code based on the system specifications and ensuring that the software architecture is properly translated into working modules. The emphasis is on creating a system that is efficient, maintainable, and scalable.

A key part of implementation is testing and quality assurance, which ensures the system is reliable and meets user requirements. This includes unit testing of individual components, integration testing of combined modules, system testing of the complete software, and user acceptance testing to confirm it fulfills the intended purpose. Additionally, data migration, integration with other systems, and configuration for the target environment are performed to prepare the system for real-world usage.

Finally, system implementation includes deployment, user training, and documentation. Deployment strategies may vary, such as phased rollouts or pilot testing, depending on organizational needs. Users are trained to operate the system effectively, supported by manuals and guides, while technical documentation helps in future maintenance. Post-deployment support ensures that any issues are resolved quickly, stabilizing the system and allowing it to function smoothly in its operational environment.

6.2 Flowchart



6.3 Coding

App.tsx

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import AppNavigator from './src/navigation/AppNavigator';
import { UserProvider } from './backend/context/UserIdContext';
import { ActiveProfileProvider } from './backend/context/ActiveProfileContext';

const App = () => {
  return (
    <UserProvider>
      <ActiveProfileProvider>
        <NavigationContainer>
          <AppNavigator />
        </NavigationContainer>
      </ActiveProfileProvider>
    </UserProvider>
  );
};

export default App;
```

MealPlanner.tsx

```
import React, { useState, useEffect } from 'react';
import {
  View,
  Text,
  TouchableOpacity,
  FlatList,
  Modal,
  TextInput,
  StyleSheet,
  ScrollView,
  ActivityIndicator,
  Alert,
} from 'react-native';
import { useUser } from '../../backend/context/UserIdContext';
import { useActiveProfile } from '../../backend/context/ActiveProfileContext';
import { BASE_URL } from '../../config';

type MealType = 'breakfast' | 'lunch' | 'dinner';

interface IMealPlan {
```

```
  breakfast: string[];
  lunch: string[];
  dinner: string[];
}
interface IMealResponse {
  breakfast: string[];
  lunch: string[];
  dinner: string[];
  message?: string;
}

const MealPlanner = () => {
  const [meals, setMeals] = useState<IMealPlan>({
    breakfast: [],
    lunch: [],
    dinner: [],
  });
  const [loading, setLoading] = useState(true);
  const [modalVisible, setModalVisible] = useState(false);
  const [currentMealType, setCurrentMealType] = useState<MealType>('breakfast');
  const [mealInput, setMealInput] = useState('');
  const { userId } = useUser();
  const { activeProfileId } = useActiveProfile();

  // Fetch meals from backend
  // Fetch meals from backend
  const fetchMeals = async () => {
    if (!userId || !activeProfileId) {
      setLoading(false); // stop loading if no profile
      return;
    }
    setLoading(true);
    try {
      const res = await fetch(
        `${BASE_URL}/api/users/${userId}/${activeProfileId}/fetchMeal`,
      );
      const data = (await res.json()) as IMealResponse;

      if (data) {
        setMeals({
          breakfast: data.breakfast || [],
          lunch: data.lunch || [],
          dinner: data.dinner || [],
        });
      }
    } catch (err) {
      console.error(err);
```

```
    Alert.alert('Error', 'Failed to fetch meals');
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  fetchMeals();
}, [userId, activeProfileId]);

// Add meal
const addMeal = async () => {
  if (!activeProfileId) {
    Alert.alert(
      'No Profile',
      'Please create a profile first before adding meals.',
    );
    return;
  }
  if (mealInput.trim() === '') return;

  try {
    const res = await fetch(
      `${BASE_URL}/api/users/${userId}/profiles/${activeProfileId}/addMeal`,
      {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          mealType: currentMealType,
          item: mealInput.trim(),
        }),
      },
    );
    const data = (await res.json()) as IMealResponse;
    if (res.ok) {
      setMeals(prev => ({
        ...prev,
        [currentMealType]: [...prev[currentMealType], mealInput.trim()],
      }));
      setModalVisible(false);
      setMealInput('');
    } else {
      Alert.alert('Error', data.message || 'Failed to add meal');
    }
  } catch (err) {
    console.error(err);
    Alert.alert('Error', 'Failed to add meal');
```

```
  }
};

// Delete meal
const deleteMeal = async (type: MealType, item: string) => {
  try {
    const res = await fetch(
      `${BASE_URL}/api/users/${userId}/profiles/${activeProfileId}/deleteMeal`,
      {
        method: 'DELETE',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ mealType: type, item }),
      },
    );
    const data = (await res.json()) as IMealResponse;
    if (res.ok) {
      setMeals(prev => ({
        ...prev,
        [type]: prev[type].filter(mealItem => mealItem !== item),
      }));
    } else {
      Alert.alert('Error', data.message || 'Failed to delete meal');
    }
  } catch (err) {
    console.error(err);
    Alert.alert('Error', 'Failed to delete meal');
  }
};

const renderMealSection = (title: string, type: MealType) => (
  <View style={styles.section}>
    <View style={styles.sectionHeader}>
      <Text style={styles.sectionTitle}>{title}</Text>
      <TouchableOpacity
        style={styles.addButton}
        onPress={() => {
          setCurrentMealType(type);
          setModalVisible(true);
        }}
      >
        <Text style={styles.addText}>+ Add</Text>
      </TouchableOpacity>
    </View>
    {meals[type].length === 0 ? (
      <Text style={{ color: '#888', marginBottom: 8 }}>No items added</Text>
    ) : (
      <FlatList
```

```
        data={meals[type]}
        keyExtractor={(_, index) => index.toString()}
        renderItem={({ item, index }) => (
          <View style={styles.mealItem}>
            <Text style={styles.mealText}>
              Option {index + 1}: {item}
            </Text>
            <TouchableOpacity
              onPress={() => deleteMeal(type, item)}
              style={styles.deleteButton}
            >
              <Text style={styles.deleteText}>X</Text>
            </TouchableOpacity>
          </View>
        )}
        scrollEnabled={false}
      />
    )}
  </View>
);

if (loading)
  return (
    <ActivityIndicator size="large" color="#007AFF" style={{ flex: 1 }} />
  );

return (
  <ScrollView style={styles.container}>
    {renderMealSection('Breakfast', 'breakfast')}
    {renderMealSection('Lunch', 'lunch')}
    {renderMealSection('Dinner', 'dinner')}

    {/* Modal for adding meal */}
    <Modal visible={modalVisible} transparent animationType="fade">
      <View style={styles.modalBackground}>
        <View style={styles.modalBox}>
          <Text style={styles.modalTitle}>Add {currentMealType}</Text>
          <TextInput
            style={styles.input}
            placeholder="Enter meal name"
            value={mealInput}
            onChangeText={setMealInput}
          />
          <View style={styles.modalButtons}>
            <TouchableOpacity style={styles.saveButton} onPress={addMeal}>
              <Text style={styles.buttonText}>Save</Text>
```

```
            </TouchableOpacity>
            <TouchableOpacity
              style={styles.cancelButton}
              onPress={() => setModalVisible(false)}
            >
              <Text style={styles.buttonText}>Cancel</Text>
            </TouchableOpacity>
          </View>
        </View>
      </View>
    </Modal>
  </ScrollView>
 );
};

const styles = StyleSheet.create({
 container: { flex: 1, backgroundColor: 'transparent', padding: 16 },
 section: {
  marginBottom: 20,
  backgroundColor: '#fefefeff',
  borderRadius: 16,
  padding: 16,
  shadowColor: '#000',
  shadowOpacity: 0.08,
  shadowRadius: 6,
  elevation: 2,
 },
 sectionHeader: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 10,
 },
 sectionTitle: { fontSize: 20, fontWeight: '700', color: '#004E92' },
 addButton: {
  backgroundColor: '#007AFF',
  paddingHorizontal: 14,
  paddingVertical: 8,
  borderRadius: 8,
 },
 addText: { color: 'white', fontWeight: '600' },
 mealItem: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  backgroundColor: 'white',
  padding: 12,
```

```
  borderRadius: 10,
  marginBottom: 8,
  borderWidth: 1,
  borderColor: '#E3EFFF',
},
mealText: { fontSize: 16, color: '#333', width: '90%' },
deleteButton: {
  width: '10%',
  backgroundColor: '#FFECEC',
  paddingHorizontal: 8,
  paddingVertical: 4,
  borderRadius: 6,
},
deleteText: { color: '#E63946', fontWeight: 'bold' },
modalBackground: {
  flex: 1,
  backgroundColor: 'rgba(0,0,0,0.3)',
  justifyContent: 'center',
  alignItems: 'center',
},
modalBox: {
  backgroundColor: 'white',
  width: '80%',
  borderRadius: 16,
  padding: 20,
  shadowColor: '#000',
  shadowOpacity: 0.2,
  shadowRadius: 8,
  elevation: 5,
},
modalTitle: {
  fontSize: 18,
  fontWeight: 'bold',
  color: '#007AFF',
  marginBottom: 12,
  textAlign: 'center',
},
input: {
  borderWidth: 1,
  borderColor: '#B0CFFF',
  borderRadius: 10,
  padding: 12,
  marginBottom: 16,
  backgroundColor: '#F9FBFF',
},
modalButtons: { flexDirection: 'row', justifyContent: 'space-around' },
saveButton: {
```

```
    backgroundColor: '#007AFF',
    padding: 12,
    borderRadius: 10,
    width: '40%',
    alignItems: 'center',
  },
  cancelButton: {
    backgroundColor: '#A0AAB5',
    padding: 12,
    borderRadius: 10,
    width: '40%',
    alignItems: 'center',
  },
  buttonText: { color: 'white', fontWeight: '600' },
});

export default MealPlanner;
```

AppNavvigator.tsx

```
import React from 'react';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import GetStarted from '../Auth/GetStarted';
import LoginScreen from '../Auth/LoginScreen';
import SignupScreen from '../Auth/SignupScreen';
import HomeScreen from '../screens/HomeScreen';
import BottomTabs from './BottomTabs';
import ProfileDetails from '../screens/ProfileDetails';
import ConsumptionPlanner from '../screens/ConsumptionPlanner';
import HealthTips from '../screens/HealthTips';
import ProductDetails from '../screens/ProductDetails';
import AIScreen from '../screens/AIScreen';
import ProfileForm from '../screens/ProfileForm';
import ProfileScreen from '../screens/ProfileDetails';
import Scanner from '../screens/Scanner';
import ForgotPasswordScreen from '../Auth/ForgetPasswordScreen';
import ChangePasswordScreen from '../screens/ChangePasswordScreen';

export type RootStackParamList = {
  GetStarted: undefined;
  Login: undefined;
  Signup: undefined;
  Home: undefined;
  MainApp: undefined;
```

```
  ProfileDetails: undefined;
  ConsumptionPlanner: undefined;
  HealthTips: undefined;
  ProductDetails: undefined;
  AIScreen: undefined;
  ProfileForm: undefined;
  ProfileScreen: undefined;
  Scanner: undefined;
  ForgotPassword: undefined;
  ChangePasswordScreen: undefined;
};

const Stack = createNativeStackNavigator<RootStackParamList>();

export default function AppNavigator() {
  return (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
      <Stack.Screen name="GetStarted" component={GetStarted} />
      <Stack.Screen name="AIScreen" component={AIScreen} />
      <Stack.Screen name="Login" component={LoginScreen} />
      <Stack.Screen name="Signup" component={SignupScreen} />
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="ProfileDetails" component={ProfileDetails} />
      <Stack.Screen name="MainApp" component={BottomTabs} />
      <Stack.Screen name="ConsumptionPlanner" component={ConsumptionPlanner} />
      <Stack.Screen name="HealthTips" component={HealthTips} />
      <Stack.Screen name="ProductDetails" component={ProductDetails} />
      <Stack.Screen name="ProfileForm" component={ProfileForm} />
      <Stack.Screen name="ProfileScreen" component={ProfileScreen} />
      <Stack.Screen name="Scanner" component={Scanner} />
      <Stack.Screen name="ForgotPassword" component={ForgotPasswordScreen} />
      <Stack.Screen
        name="ChangePasswordScreen"
        component={ChangePasswordScreen}
      />
    </Stack.Navigator>
  );
}

import React from 'react';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import GetStarted from '../Auth/GetStarted';
import LoginScreen from '../Auth/LoginScreen';
import SignupScreen from '../Auth/SignupScreen';
import HomeScreen from '../screens/HomeScreen';
import BottomTabs from './BottomTabs';
import ProfileDetails from '../screens/ProfileDetails';
```

```
import ConsumptionPlanner from '../screens/ConsumptionPlanner';
import HealthTips from '../screens/HealthTips';
import ProductDetails from '../screens/ProductDetails';
import AIScreen from '../screens/AIScreen';
import ProfileForm from '../screens/ProfileForm';
import ProfileScreen from '../screens/ProfileDetails';
import Scanner from '../screens/Scanner';
import ForgotPasswordScreen from '../Auth/ForgetPasswordScreen';
import ChangePasswordScreen from '../screens/ChangePasswordScreen';

export type RootStackParamList = {
  GetStarted: undefined;
  Login: undefined;
  Signup: undefined;
  Home: undefined;
  MainApp: undefined;
  ProfileDetails: undefined;
  ConsumptionPlanner: undefined;
  HealthTips: undefined;
  ProductDetails: undefined;
  AIScreen: undefined;
  ProfileForm: undefined;
  ProfileScreen: undefined;
  Scanner: undefined;
  ForgotPassword: undefined;
  ChangePasswordScreen: undefined;
};

const Stack = createNativeStackNavigator<RootStackParamList>();

export default function AppNavigator() {
  return (
    <Stack.Navigator screenOptions={{ headerShown: false }}>
      <Stack.Screen name="GetStarted" component={GetStarted} />
      <Stack.Screen name="AIScreen" component={AIScreen} />
      <Stack.Screen name="Login" component={LoginScreen} />
      <Stack.Screen name="Signup" component={SignupScreen} />
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="ProfileDetails" component={ProfileDetails} />
      <Stack.Screen name="MainApp" component={BottomTabs} />
      <Stack.Screen name="ConsumptionPlanner" component={ConsumptionPlanner} />
      <Stack.Screen name="HealthTips" component={HealthTips} />
      <Stack.Screen name="ProductDetails" component={ProductDetails} />
      <Stack.Screen name="ProfileForm" component={ProfileForm} />
      <Stack.Screen name="ProfileScreen" component={ProfileScreen} />
      <Stack.Screen name="Scanner" component={Scanner} />
      <Stack.Screen name="ForgotPassword" component={ForgotPasswordScreen} />
```

```
      <Stack.Screen
        name="ChangePasswordScreen"
        component={ChangePasswordScreen}
      />
    </Stack.Navigator>
  );
}
```

BottomTabs.tsx

```
import React from 'react';
import { Image } from 'react-native';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import AIScreen from '../screens/AIScreen';
import HomeScreen from '../screens/HomeScreen';
import ProfileScreen from '../screens/ProfileScreen';
import ScanScreen from '../screens/ScanScreen';

const Tab = createBottomTabNavigator();

const BottomTabs = () => {
  return (
    <Tab.Navigator
      screenOptions={{
        tabBarStyle: {
          height: 75,
        },
      }}
    >
      {/* Home Tab */}
      <Tab.Screen
        name="Home"
        component={HomeScreen}
        options={{
          headerShown: false,
          tabBarLabel: 'Home',
          tabBarLabelStyle: {
            fontSize: 12,
            fontWeight: '600',
            color: '#1e90ff',
            paddingTop: 2,
          },
          tabBarIcon: () => (
            <Image
              source={require('../assets/icons/homeIcon.png')}
              style={{ width: 26, height: 26, tintColor: '#1e90ff' }}
```

```
      />
    ),
  }}
/>

{/* Scan Tab */}
<Tab.Screen
  name="Scan"
  component={ScanScreen}
  options={{
    headerShown: false,
    tabBarLabel: 'Scan',
    tabBarLabelStyle: {
      fontSize: 12,
      fontWeight: '600',
      color: '#1e90ff',
      paddingTop: 2,
    },
    tabBarIcon: () => (
      <Image
        source={require('../assets/icons/scanIcon.png')}
        style={{ width: 26, height: 26, tintColor: '#1e90ff' }}
      />
    ),
  }}
/>

{/* AI Tab */}
<Tab.Screen
  name="AI"
  component={AIScreen}
  options={{
    headerShown: false,
    tabBarLabel: 'AI',
    tabBarLabelStyle: {
      fontSize: 12,
      fontWeight: '600',
      color: '#1e90ff',
      paddingTop: 2,
    },
    tabBarIcon: () => (
      <Image
        source={require('../assets/icons/robotLogo.png')}
        style={{ width: 40, height: 40 }}
      />
    ),
  }}
```

```
      />

      {/* Profile Tab */}
      <Tab.Screen
        name="Profile"
        component={ProfileScreen}
        options={{
          headerShown: false,
          tabBarLabel: 'Profile',
          tabBarLabelStyle: {
            fontSize: 12,
            fontWeight: '600',
            color: '#1e90ff',
            paddingTop: 2,
          },
          tabBarIcon: () => (
            <Image
              source={require('../assets/icons/profileIcon.png')}
              style={{ width: 26, height: 26, tintColor: '#1e90ff' }}
            />
          ),
        }}
      />
    </Tab.Navigator>
  );
};

export default BottomTabs;
```

AIScreen.tsx

```
import React, { useState, useRef, useEffect } from 'react';
import {
  View,
  Text,
  TextInput,
  StyleSheet,
  TouchableOpacity,
  Image,
  ScrollView,
  ActivityIndicator,
} from 'react-native';
import { indian_foods } from '../assets/dataSet/indianFood';
import { GOOGLE_API_KEY_1 } from '@env';
```

```
export default function AIScreen() {
  const [message, setMessage] = useState('');
  const [chat, setChat] = useState([
    {
      sender: 'bot',
      text: "Hi! I'm Nova, your personal AI Nutritionist. How can I help you?",
    },
  ]);
  const [isLoading, setIsLoading] = useState(false);
  const scrollViewRef = useRef<ScrollView>(null);

  const sendMessageToAI = async () => {
    if (!message.trim()) return;

    // Add user message immediately
    setChat(prev => [...prev, { sender: 'user', text: message }]);
    setIsLoading(true);

    try {
      const conversationHistoryJSON = chat.map(c => ({
        sender: c.sender,
        text: c.text,
      }));

      const response = await fetch(
                          `https://generativelanguage.googleapis.com/v1beta/models/gemini-2.5-
flash:generateContent?key=${GOOGLE_API_KEY_1}`,
        {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            contents: [
              {
                parts: [
                  {
                    text: JSON.stringify({
                      assistantProfile: {
                        name: 'Nova',
                        role: 'AI Nutritionist',
                        description:
                        'Hi! I am Nova, your AI Nutritionist. I provide friendly, practical, and supportive
guidance. I first try to understand your problem before suggesting anything. I explain why your
issue might be happening in simple, easy-to-understand language, then ask clarifying questions
to know more about your habits or diet. Only after you indicate that you want a meal plan do I
provide one. When suggesting meals, I include protein (dal, paneer, eggs, chicken, tofu,
legumes), carbs (rice, chapati, oats, quinoa), vegetables and fruits for vitamins and fiber, and
healthy fats (nuts, seeds, small amounts of oil), using healthy cooking methods like boiling,
```

steaming, grilling, or roasting. I respect dietary restrictions such as allergies, vegetarian or vegan preferences, keep portions realistic and varied, encourage hydration, and provide simple, practical options. Meal plans are formatted with headings and emojis like Breakfast , Lunch , and Dinner with 3 options per meal, leaving a line between meals. All answers are short, clear, under 100 words, and include line breaks after every 2 lines. I always prioritize understanding the problem, explaining it, asking for clarification, and then giving actionable guidance, only giving full meal plans when requested.',

```
                dataset: indian_foods,
              },

              userMessage: message,
              conversationHistory: conversationHistoryJSON,
            }),
          },
        ],
      },
    ],
  }),
 },
);

const data: any = await response.json();
let botReply =
 data?.candidates?.[0]?.content?.parts?.[0]?.text ||
 'Sorry, I couldn't understand.';
botReply = botReply.trim();

setChat(prev => [...prev, { sender: 'bot', text: botReply }]);
} catch (error) {
 console.error('Error calling Gemini API:', error);
 setChat(prev => [
  ...prev,
  { sender: 'bot', text: 'Oops! Something went wrong.' },
 ]);
}

setIsLoading(false);
setMessage('');
};

useEffect(() => {
 scrollViewRef.current?.scrollToEnd({ animated: true });
}, [chat]);

const clearChat = () => {
 setChat([
  {
```

```
      sender: 'bot',
      text: "Hi! I'm Nova, your personal AI Nutritionist. How can I help you?",
    },
  ]);
};

return (
  <View style={styles.container}>
    {/* Header */}
    <View style={styles.header}>
      <Image
        source={require('../assets/icons/botLogoback.png')}
        style={styles.headerIcon}
      />
      <Text style={styles.headerText}>NovaBot</Text>

      <View style={{ padding: 10, alignItems: 'center' }}>
        <TouchableOpacity style={styles.clearButton} onPress={clearChat}>
          <Text style={styles.clearButtonText}>Clear Chat</Text>
        </TouchableOpacity>
      </View>
    </View>

    {/* Chat area */}
    <ScrollView ref={scrollViewRef} style={styles.chatArea}>
      {chat.map((c, index) => (
        <View
          key={index}
          style={c.sender === 'bot' ? styles.botRow : styles.userRow}
        >
          {c.sender === 'bot' && (
            <Image
              source={require('../assets/icons/askAiLogo.png')}
              style={styles.chatIcon}
            />
          )}
          <View
            style={c.sender === 'bot' ? styles.botBubble : styles.userBubble}
          >
            <Text style={styles.chatText}>{c.text}</Text>
          </View>
          {c.sender === 'user' && (
            <Image
              source={require('../assets/icons/profileIcon.png')}
              style={styles.chatIconUser}
            />
          )}
```

```
          </View>
        ))}

        {isLoading && (
          <View
            style={{
              flexDirection: 'row',
              alignItems: 'center',
              marginVertical: 5,
            }}
          >
            <ActivityIndicator size="small" color="#4A90E2" />
            <Text style={{ marginLeft: 5, color: '#4A90E2' }}>
              Nova is typing...
            </Text>
          </View>
        )}
      </ScrollView>

      {/* Input row */}
      <View style={styles.inputRow}>
        <TextInput
          style={styles.textInput}
          placeholder="Type Your Message Here..."
          placeholderTextColor="#666"
          value={message}
          onChangeText={setMessage}
        />
        <TouchableOpacity onPress={sendMessageToAI}>
          <Image
            source={require('../assets/icons/sendIcon.png')}
            style={styles.sendIcon}
          />
        </TouchableOpacity>
      </View>

      {/* Clear Chat Button */}
    </View>
  );
}

const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: '#fff' },
  header: {
    flexDirection: 'row',
    alignItems: 'center',
    backgroundColor: '#4A90E2',
```

```
  paddingVertical: 12,
  paddingHorizontal: 16,
},
headerIcon: { width: 45, height: 45, marginRight: 10, borderRadius: 22.5 },
headerText: {
  color: '#fff',
  fontSize: 23,
  fontWeight: 'bold',
  letterSpacing: 3,
},
chatArea: { flex: 1, paddingHorizontal: 8, marginBottom: 10 },
botRow: { flexDirection: 'row', alignItems: 'flex-start', marginVertical: 5 },
userRow: {
  flexDirection: 'row',
  justifyContent: 'flex-end',
  alignItems: 'flex-start',
  marginVertical: 5,
},
chatIcon: { width: 40, height: 40, marginHorizontal: 3 },
chatIconUser: {
  width: 30,
  height: 30,
  marginHorizontal: 6,
  marginVertical: 3,
  tintColor: '#4A90E2',
},
botBubble: {
  backgroundColor: '#f1f1f1',
  padding: 10,
  borderRadius: 10,
  maxWidth: '80%',
},
userBubble: {
  backgroundColor: '#e0eaff',
  padding: 10,
  borderRadius: 10,
  maxWidth: '80%',
},
chatText: { fontSize: 16, color: '#000', letterSpacing: 1 },
inputRow: {
  backgroundColor: '#4A90E2',
  flexDirection: 'row',
  alignItems: 'center',
  borderTopWidth: 1,
  borderColor: '#ddd',
  paddingHorizontal: 17,
  paddingVertical: 6,
```

```
    height: 55,
  },
  textInput: {
    flex: 1,
    fontSize: 14,
    color: '#000',
    backgroundColor: '#fff',
    borderRadius: 10,
    paddingHorizontal: 12,
    paddingVertical: 4,
    height: 35,
  },
  sendIcon: { width: 35, height: 35, marginLeft: 5, tintColor: '#fff' },
  clearButton: {
    backgroundColor: '#f53b2eff',
    paddingVertical: 8,
    paddingHorizontal: 12,
    borderRadius: 20,
    left: 40,
  },
  clearButtonText: {
    color: '#fff',
    fontSize: 16,
    fontWeight: 'bold',
  },
});
```

ConsumtionPlanner.tsx

```
import React, { useState, useRef } from 'react';
import {
  View,
  Text,
  TextInput,
  StyleSheet,
  TouchableOpacity,
  ScrollView,
  Alert,
} from 'react-native';
import { Picker } from '@react-native-picker/picker';
import Clipboard from '@react-native-clipboard/clipboard';

const ConsumptionPlanner: React.FC = () => {
  const [age, setAge] = useState('22');
```

```
const [gender, setGender] = useState('M');
const [height, setHeight] = useState('180');
const [weight, setWeight] = useState('80');
const [activity, setActivity] = useState('Low');
const [condition, setCondition] = useState('');
const [showResults, setShowResults] = useState(false);

// Results state
const [results, setResults] = useState({
  calories: 0,
  bmi: 0,
  bmiStatus: '',
  protein: 0,
  carbs: 0,
  fats: 0,
  water: 0,
  proteinPct: 0,
  carbsPct: 0,
  fatsPct: 0,
  normalWeightRange: 'N/A',
});

// refs for scrolling
const scrollViewRef = useRef<ScrollView>(null);

// Input validation helper
const isValidNumber = (val: string) => {
  if (!val) return false;
  return /^\d+(\.\d+)?$/.test(val);
};

// Dynamic formulas
const calculateResults = () => {
  const ageNum = parseInt(age || '0', 10);
  const heightNum = parseFloat(height || '0');
  const weightNum = parseFloat(weight || '0');

  if (
    !isValidNumber(age) ||
    !isValidNumber(height) ||
    !isValidNumber(weight) ||
    ageNum < 10 ||
    ageNum > 120 ||
    heightNum < 100 ||
    heightNum > 250 ||
    weightNum < 30 ||
    weightNum > 300
```

```
  ) {
    Alert.alert(
      'Invalid Input',
      'Please enter valid numbers for age (10-120), height (100-250cm), and weight (30-300kg).',
    );
    return null;
  }

  // BMR (Mifflin-St Jeor Equation)
  let bmr =
    gender === 'M'
      ? 10 * weightNum + 6.25 * heightNum - 5 * ageNum + 5
      : 10 * weightNum + 6.25 * heightNum - 5 * ageNum - 161;

  // Activity multiplier
  let activityMult = 1.2;
  if (activity === 'Moderate') activityMult = 1.55;
  if (activity === 'High') activityMult = 1.725;

  // Final calories
  const calories = Math.round(bmr * activityMult);

  // BMI
  const bmi = +(weightNum / (heightNum / 100) ** 2).toFixed(1);
  let bmiStatus = 'Normal';
  if (bmi < 18.5) bmiStatus = 'Underweight';
  else if (bmi < 25) bmiStatus = 'Normal';
  else if (bmi < 30) bmiStatus = 'Overweight';
  else bmiStatus = 'Obese';

  // Normal weight range for this height
  const minNormal = 18.5 * (heightNum / 100) ** 2;
  const maxNormal = 24.9 * (heightNum / 100) ** 2;

  /// Macro calculation based on activity and calories
  let protein = weightNum * 1.0; // sedentary / low activity
  if (activity === 'Moderate') protein = weightNum * 1.2;
  if (activity === 'High') protein = weightNum * 1.6;
  protein = Math.round(protein);

  const fats = Math.round((calories * 0.25) / 9); // 25% of calories from fats
  const carbs = Math.round((calories - (protein * 4 + fats * 9)) / 4); // remaining calories from
carbs

  // Percentages for UI
  const proteinPct = Math.round((protein * 4 * 100) / calories);
  const fatsPct = Math.round((fats * 9 * 100) / calories);
```

```
  const carbsPct = 100 - proteinPct - fatsPct;
  // Water intake (simple: 35ml per kg)
  const water = +(weightNum * 0.035).toFixed(2);

  return {
    calories,
    bmi,
    bmiStatus,
    proteinPct,
    carbsPct,
    fatsPct,
    protein,
    carbs,
    fats,
    water,
    normalWeightRange: `${minNormal.toFixed(1)} kg - ${maxNormal.toFixed(
      1,
    )} kg`,
  };
};

const handleCalculate = () => {
  const res = calculateResults();
  if (!res) return;
  setResults(res);
  setShowResults(true);
};

return (
  <ScrollView
    ref={scrollViewRef}
    contentContainerStyle={styles.screen}
    keyboardShouldPersistTaps="handled"
  >
    <Text style={styles.heading}>
      How Much{'\n'}Should I Eat{'\n'}Daily?
    </Text>

    {/* FORM */}
    <View style={styles.formBox}>
      <Text style={styles.formTitle}>Enter Your Details</Text>

      {/* Age & Gender */}
      <View style={styles.row}>
        <View style={[styles.inputCol, { marginRight: 8 }]}>
          <Text style={styles.label}>Age:</Text>
          <TextInput
```

```
      style={styles.textInput}
      keyboardType="numeric"
      value={age}
      onChangeText={text => /^\d*$/.test(text) && setAge(text)}
      placeholder="e.g., 22"
      placeholderTextColor="#9bbbd6"
      maxLength={3}
    />
  </View>
  <View style={styles.inputCol}>
    <Text style={styles.label}>Gender:</Text>
    <View style={styles.pickerWrap}>
      <Picker
        selectedValue={gender}
        onValueChange={setGender}
        style={styles.picker}
      >
        <Picker.Item label="Male" value="M" />
        <Picker.Item label="Female" value="F" />
      </Picker>
    </View>
  </View>
</View>

{/* Height */}
<View style={styles.inputCol}>
  <Text style={styles.label}>Height (cm):</Text>
  <TextInput
    style={styles.textInput}
    keyboardType="numeric"
    value={height}
    onChangeText={text => /^\d*\.?\d*$/.test(text) && setHeight(text)}
    placeholder="e.g., 170"
    placeholderTextColor="#9bbbd6"
    maxLength={5}
  />
</View>

{/* Weight */}
<View style={styles.inputCol}>
  <Text style={styles.label}>Weight (kg):</Text>
  <TextInput
    style={styles.textInput}
    keyboardType="numeric"
    value={weight}
    onChangeText={text => /^\d*\.?\d*$/.test(text) && setWeight(text)}
    placeholder="e.g., 80"
```

```
        placeholderTextColor="#9bbbd6"
        maxLength={5}
      />
    </View>

    {/* Activity Level */}
    <View style={styles.inputCol}>
      <Text style={styles.label}>Activity Level:</Text>
      <View style={styles.pickerWrap}>
        <Picker
          selectedValue={activity}
          onValueChange={setActivity}
          style={styles.picker}
        >
          <Picker.Item label="Low (Sedentary)" value="Low" />
          <Picker.Item label="Moderate (Active)" value="Moderate" />
          <Picker.Item label="High (Very Active)" value="High" />
        </Picker>
      </View>
    </View>

    {/* Condition (Changed to TextInput) */}
    <View style={styles.inputCol}>
      <Text style={styles.label}>Medical Conditions:</Text>
      <TextInput
        style={styles.textInput}
        value={condition}
        onChangeText={setCondition}
        placeholder="e.g., Diabetes, Malnutrition"
        placeholderTextColor="#9bbbd6"
      />
    </View>

    {/* Button */}
    <TouchableOpacity style={styles.button} onPress={handleCalculate}>
      <Text style={styles.buttonText}>Calculate</Text>
    </TouchableOpacity>
  </View>

  {/* RESULTS */}
  {showResults && (
    <View style={styles.resultsBox}>
      <View style={styles.resultCard}>
        <Text style={styles.resultTitle}> Daily Calories</Text>
        <Text style={styles.resultValue}>{results.calories} kcal</Text>
        <Text style={styles.resultSub}>Based on your selections</Text>
      </View>
```

```
<View style={styles.resultCard}>
  <Text style={styles.resultTitle}>BMI Analysis</Text>
  <Text style={styles.resultValue}>
    Your BMI is {results.bmi} ({results.bmiStatus})
  </Text>
  <Text style={styles.resultSub}>
    Calculated from height & weight
  </Text>
  <Text style={styles.resultSub}>
    Normal weight range for your height:{' '}
    {results.normalWeightRange || 'N/A'}
  </Text>
</View>

<View style={styles.resultCard}>
  <Text style={styles.resultTitle}>Water Intake</Text>
  <Text style={styles.resultValue}>{results.water} L / day</Text>
  <Text style={styles.resultSub}>Based on your weight</Text>
</View>

<View style={styles.resultCard}>
  <Text style={styles.resultTitle}> Macro Split</Text>

  <View style={styles.legendRow}>
    <View
      style={[styles.legendDot, { backgroundColor: '#65aef2' }]}
    />
    <Text style={styles.legendText}>
      Protein {results.proteinPct}%
    </Text>
    <View
      style={[styles.legendDot, { backgroundColor: '#f2b365' }]}
    />
    <Text style={styles.legendText}>Carbs {results.carbsPct}%</Text>
    <View
      style={[styles.legendDot, { backgroundColor: '#6cc070' }]}
    />
    <Text style={styles.legendText}>Fats {results.fatsPct}%</Text>
  </View>

  <View style={styles.macroBarOuter}>
    <View
      style={[
        styles.macroSeg,
        { flex: results.proteinPct, backgroundColor: '#65aef2' },
      ]}
```

```
            />
            <View
              style={[
                styles.macroSeg,
                { flex: results.carbsPct, backgroundColor: '#f2b365' },
              ]}
            />
            <View
              style={[
                styles.macroSeg,
                { flex: results.fatsPct, backgroundColor: '#6cc070' },
              ]}
            />
          </View>

          <View style={styles.macroNumbers}>
            <Text style={styles.macroNumText}>
              Protein {results.protein} g
            </Text>
            <Text style={styles.macroNumText}>Carbs {results.carbs} g</Text>
            <Text style={styles.macroNumText}>Fats {results.fats} g</Text>
          </View>
        </View>
        <View style={styles.resultCard}>
          <Text style={styles.resultTitle}>AI Diet Prompt</Text>
          <Text style={styles.resultSub}>
            Copy & paste this into your AI tool to generate a diet plan
          </Text>
          <ScrollView style={styles.promptBox}>
            <Text selectable style={styles.promptText}>
              {`Create a Indian diet plan based on:
- Daily Calories: ${results.calories} kcal
- Protein: ${results.protein} g
- Carbs: ${results.carbs} g
- Fats: ${results.fats} g
- Water Intake: ${results.water} L/day
Provide meal-wise breakdown (Breakfast, Lunch, Dinner).`}
            </Text>
          </ScrollView>
          <TouchableOpacity
            style={styles.copyButton}
            onPress={() =>
              Clipboard.setString(
                `Create a Indian diet plan based on:
- Daily Calories: ${results.calories} kcal
- Protein: ${results.protein} g
- Carbs: ${results.carbs} g
```

```
- Fats: ${results.fats} g
- Water Intake: ${results.water} L/day
Provide meal-wise breakdown (Breakfast, Lunch, Dinner).`,
              )
            }
           >
             <Text style={styles.copyButtonText}>Copy Prompt</Text>
           </TouchableOpacity>
         </View>
       </View>
     )}
   </ScrollView>
 );
};

const styles = StyleSheet.create({
 screen: {
   backgroundColor: '#fefffff',
   padding: 18,
   alignItems: 'center',
   minHeight: '100%',
 },
 heading: {
   fontSize: 28,
   fontWeight: '600',
   color: '#65aef2',
   textAlign: 'center',
   marginVertical: 24,
   letterSpacing: 1,
   lineHeight: 36,
 },
 formBox: {
   backgroundColor: '#fff',
   borderRadius: 16,
   borderColor: '#c7c7c7',
   borderWidth: 1,
   padding: 20,
   width: '100%',
   maxWidth: 400,
   marginBottom: 18,
   shadowColor: '#000',
   shadowOpacity: 0.04,
   shadowRadius: 6,
   elevation: 2,
 },
 formTitle: {
   fontSize: 21,
```

```
  fontWeight: 'bold',
  textAlign: 'center',
  marginBottom: 22,
  letterSpacing: 1,
  color: '#222',
},
row: {
  flexDirection: 'row',
  marginBottom: 10,
},
inputCol: {
  flex: 1,
  marginBottom: 14,
},
label: {
  fontSize: 16,
  marginBottom: 2,
  fontWeight: '500',
  color: '#161d25',
},
textInput: {
  borderWidth: 1,
  borderColor: '#65aef2',
  borderRadius: 7,
  padding: 10,
  fontSize: 16,
  backgroundColor: '#f4fafd',
  marginTop: 2,
  color: '#222',
},
pickerWrap: {
  borderWidth: 1,
  borderColor: '#65aef2',
  borderRadius: 7,
  overflow: 'hidden',
  backgroundColor: '#f4fafd',
  marginTop: 2,
},
picker: {
  height: 52,
  width: '100%',
  color: '#222',
},
button: {
  alignSelf: 'center',
  backgroundColor: '#65aef2',
  borderRadius: 10,
```

```
  paddingVertical: 12,
  paddingHorizontal: 42,
  marginTop: 18,
  shadowColor: '#65aef2',
  shadowOpacity: 0.15,
  shadowRadius: 4,
  elevation: 2,
},
buttonText: {
  color: '#fff',
  fontSize: 17,
  fontWeight: '600',
  letterSpacing: 0.5,
},
resultsBox: {
  marginTop: 25,
  width: '100%',
  maxWidth: 400,
},
resultCard: {
  backgroundColor: '#f9f9f9',
  borderRadius: 16,
  padding: 18,
  marginVertical: 9,
  alignItems: 'center',
  shadowColor: '#000',
  shadowOpacity: 0.05,
  shadowRadius: 5,
  elevation: 6,
},
resultTitle: {
  fontSize: 18,
  fontWeight: '600',
  marginBottom: 6,
  color: '#222',
},
resultValue: {
  fontSize: 20,
  fontWeight: 'bold',
  color: '#65aef2',
},
resultSub: {
  marginTop: 4,
  fontSize: 12,
  color: '#6c7a89',
},
legendRow: {
```

```
  flexDirection: 'row',
  alignItems: 'center',
  flexWrap: 'wrap',
  justifyContent: 'center',
  marginTop: 6,
  marginBottom: 10,
},
legendDot: {
  width: 10,
  height: 10,
  borderRadius: 5,
  marginHorizontal: 4,
},
legendText: {
  fontSize: 12,
  color: '#333',
  marginRight: 8,
},
macroBarOuter: {
  height: 18,
  width: '100%',
  backgroundColor: '#e9eef3',
  borderRadius: 10,
  overflow: 'hidden',
  flexDirection: 'row',
},
macroSeg: {
  height: '100%',
},
macroNumbers: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  width: '100%',
  marginTop: 10,
},
macroNumText: {
  fontSize: 13,
  color: '#444',
},
promptBox: {
  maxHeight: 120,
  width: '100%',
  marginVertical: 8,
},
promptText: {
  fontSize: 13,
  color: '#333',
```

```
  },
  copyButton: {
    backgroundColor: '#65aef2',
    paddingVertical: 10,
    paddingHorizontal: 25,
    borderRadius: 10,
    marginTop: 6,
  },
  copyButtonText: {
    color: '#fff',
    fontSize: 14,
    fontWeight: '600',
  },
});

export default ConsumptionPlanner;
```

HealthTips.tsx

```
import React, { useState } from 'react';
import {
  View,
  Text,
  TouchableOpacity,
  StyleSheet,
  ScrollView,
} from 'react-native';

import { generalTips, Tip } from '../assets/dataSet/generalTips';
import { personalizedTips } from '../assets/dataSet/personalizedtips';

const HealthTipsScreen: React.FC = () => {
  const [selectedTab, setSelectedTab] = useState<'General' | 'Personalized'>(
    'General',
  );
  const [subTab, setSubTab] = useState<'Underweight' | 'Normal' | 'Overweight'>(
    'Normal',
  );

  const [generalIndex, setGeneralIndex] = useState(0);
  const [personalizedIndex, setPersonalizedIndex] = useState(0);

  // Choose active tips list
  const getActiveTips = () => {
    if (selectedTab === 'General') {
      return generalTips;
```

```
  } else {
    return personalizedTips[
      subTab.toLowerCase() as 'underweight' | 'normal' | 'overweight'
    ];
  }
};

const tips = getActiveTips();
const activeTip =
  selectedTab === 'General' ? tips[generalIndex] : tips[personalizedIndex];

// Next button handler
const handleChangeTip = () => {
  if (selectedTab === 'General') {
    setGeneralIndex(prev => (prev + 1) % generalTips.length);
  } else {
    const list =
      personalizedTips[
        subTab.toLowerCase() as 'underweight' | 'normal' | 'overweight'
      ];
    setPersonalizedIndex(prev => (prev + 1) % list.length);
  }
};

return (
  <ScrollView contentContainerStyle={styles.container}>
    {/* Heading */}
    <Text style={styles.heading}>Daily Health Tips</Text>
    <Text style={styles.subText}>
      Stay consistent!{'\n'}Small changes lead to big results
    </Text>

    {/* Top Toggle Buttons */}
    <View style={styles.toggleContainer}>
      {['General', 'Personalized'].map(tab => (
        <TouchableOpacity
          key={tab}
          style={[
            styles.toggleButton,
            selectedTab === tab && styles.activeToggle,
          ]}
          onPress={() => {
            setSelectedTab(tab as 'General' | 'Personalized');
            setGeneralIndex(0);
            setPersonalizedIndex(0);
          }}
        >
```

```
      <Text
        style={[
          styles.toggleText,
          selectedTab === tab && styles.activeToggleText,
        ]}
      >
        {tab}
      </Text>
    </TouchableOpacity>
  ))}
</View>

{/* Sub Buttons only if Personalized */}
{selectedTab === 'Personalized' && (
  <View style={styles.toggleContainer}>
    {['Underweight', 'Normal', 'Overweight'].map(tab => (
      <TouchableOpacity
        key={tab}
        style={[
          styles.toggleButton,
          subTab === tab && styles.activeToggle,
        ]}
        onPress={() => {
          setSubTab(tab as 'Underweight' | 'Normal' | 'Overweight');
          setPersonalizedIndex(0);
        }}
      >
        <Text
          style={[
            styles.toggleText,
            subTab === tab && styles.activeToggleText,
          ]}
        >
          {tab}
        </Text>
      </TouchableOpacity>
    ))}
  </View>
)}

{/* Tip Card */}
{activeTip && (
  <View style={[styles.tipCard, { borderColor: activeTip.color }]}>
    <View
      style={[styles.tipHeader, { backgroundColor: activeTip.color }]}
    >
      <Text style={styles.emoji}>{activeTip.emoji}</Text>
```

```
          <View>
            <Text style={styles.tipTitle}>{activeTip.title}</Text>
            <Text style={styles.tipSubtitle}>{activeTip.subtitle}</Text>
          </View>
        </View>

        <View style={styles.tipBody}>
          <Text style={styles.bold}>Benefit:</Text>
          <Text style={styles.normal}>{activeTip.benefit}</Text>

          <Text style={[styles.bold, { marginTop: 6 }]}>Bonus:</Text>
          <Text style={styles.normal}>{activeTip.bonus}</Text>

          <View style={styles.aiBox}>
            <Text style={[styles.aiText, { color: activeTip.color }]}>
              AI Insight: {activeTip.ai}
            </Text>
          </View>
        </View>
      </View>
    )}

    {/* Change Tip Button */}
    <TouchableOpacity style={styles.changeButton} onPress={handleChangeTip}>
      <Text style={styles.changeButtonText}>Next</Text>
    </TouchableOpacity>
  </ScrollView>
 );
};

const styles = StyleSheet.create({
 container: {
  flex: 1,
  padding: 18,
  backgroundColor: '#ffffffff',
  alignItems: 'center',
 },
 heading: {
  fontSize: 28,
  fontWeight: '700',
  color: '#1a73e8',
  textAlign: 'center',
  marginBottom: 6,
 },
 subText: {
  fontSize: 15,
  color: '#444',
```

```
  textAlign: 'center',
  marginBottom: 16,
},
toggleContainer: {
  flexDirection: 'row',
  flexWrap: 'wrap',
  justifyContent: 'center',
  marginBottom: 16,
  backgroundColor: '#f2f2f2',
  borderRadius: 25,
  padding: 4,
},
toggleButton: {
  paddingVertical: 8,
  paddingHorizontal: 14,
  borderRadius: 20,
  margin: 4,
},
toggleText: {
  fontSize: 15,
  fontWeight: '500',
  color: '#555',
},
activeToggle: {
  backgroundColor: '#fff',
  elevation: 3,
  shadowColor: '#000',
  shadowOpacity: 0.1,
  shadowRadius: 4,
},
activeToggleText: {
  color: '#000',
  fontWeight: '600',
},
tipCard: {
  width: '95%',
  maxWidth: 370,
  borderWidth: 1.5,
  borderRadius: 16,
  overflow: 'hidden',
  backgroundColor: '#fff',
  marginBottom: 20,
},
tipHeader: {
  flexDirection: 'row',
  alignItems: 'center',
  padding: 14,
```

```
},
emoji: {
  fontSize: 32,
  marginRight: 12,
},
tipTitle: {
  fontSize: 19,
  fontWeight: '700',
  color: '#fff',
},
tipSubtitle: {
  fontSize: 14,
  color: '#f0f8ff',
},
tipBody: {
  padding: 16,
},
bold: {
  fontWeight: '700',
  fontSize: 16,
},
normal: {
  fontSize: 15,
  marginTop: 2,
  color: '#333',
},
aiBox: {
  backgroundColor: '#f7f5f5ff',
  padding: 10,
  borderRadius: 12,
  marginTop: 12,
},
aiText: {
  fontSize: 14,
  fontWeight: '400',
},
changeButton: {
  backgroundColor: '#1a73e8',
  paddingVertical: 12,
  paddingHorizontal: 24,
  borderRadius: 25,
  marginTop: 10,
},
changeButtonText: {
  color: '#fff',
  fontWeight: '600',
  fontSize: 16,
```

```
  },
});

export default HealthTipsScreen;
```

HomeScreen.tsx

```
import React from 'react';
import {
  Text,
  View,
  StyleSheet,
  TouchableOpacity,
  Image,
  TextInput,
  ScrollView,
} from 'react-native';
import LinearGradient from 'react-native-linear-gradient';
import MealPlanner from '../components/MealPlanner';
import { useNavigation } from '@react-navigation/native';
import { NativeStackNavigationProp } from '@react-navigation/native-stack';
import { useActiveProfile } from '../../backend/context/ActiveProfileContext';

type RootStackParamList = {
  ConsumptionPlanner: undefined; // corrected spelling
  HealthTips: undefined;
  ProfileScreen: undefined;
  AI: undefined; // added for 2nd card navigation
};

const HomeScreen = () => {
  const navigation =
    useNavigation<NativeStackNavigationProp<RootStackParamList>>();
  const { activeProfileName } = useActiveProfile();

  const today = new Date();
  const options: Intl.DateTimeFormatOptions = {
    weekday: 'long',
    month: 'long',
    day: 'numeric',
  };
  const formattedDate = today.toLocaleDateString('en-US', options);

  return (
    <LinearGradient
      colors={['#ffffff', '#88E9FF', '#ffffff']}
```

```
  style={styles.container}
>
  {/* Header Row */}
  <View style={styles.header}>
    <View>
      <Text style={styles.greeting}>
        Hello, {activeProfileName ? activeProfileName : 'Guest'}
      </Text>
      <Text style={styles.date}>{formattedDate}</Text>
    </View>

    {/* Profile button */}
    <TouchableOpacity
      style={styles.profileButton}
      onPress={() => navigation.navigate('ProfileScreen')}
    >
      <Image
        source={require('../assets/icons/profileIcon.png')}
        style={styles.profileIcon}
      />
    </TouchableOpacity>
  </View>

  {/* Search Input */}
  <TouchableOpacity
    style={styles.inputContainer}
    onPress={() => navigation.navigate('AI')} //  navigate to AI screen
  >
    <Image
      source={require('../assets/icons/askAiLogo.png')}
      style={styles.icon}
    />
    <Text style={styles.searchInputPlaceholder}>Ask AI...</Text>
  </TouchableOpacity>

  <ScrollView showsVerticalScrollIndicator={false}>
    {/* Two Cards */}
    <View style={styles.twoCardContainer}>
      {/* Card One */}
      <TouchableOpacity
        style={styles.cardOne}
        onPress={() => navigation.navigate('ConsumptionPlanner')}
      >
        <Image
          source={require('../assets/images/consumeDaily.png')}
          style={styles.cardOneImage}
        />
```

```
          </TouchableOpacity>

          {/* Card Two */}
          <TouchableOpacity
            style={styles.cardTwo}
            onPress={() => navigation.navigate('HealthTips')}
          >
            <Image
              source={require('../assets/images/healthTips.png')}
              style={styles.cardTwoImage}
            />
          </TouchableOpacity>
        </View>

        {/* Meal Planner */}
        <MealPlanner />
      </ScrollView>
    </LinearGradient>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 15,
  },
  header: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    paddingHorizontal: 25,
    paddingVertical: 15,
  },
  greeting: {
    fontSize: 25,
    fontWeight: '600',
    color: '#000',
  },
  date: {
    fontSize: 16,
    color: '#777',
    marginTop: 4,
    paddingHorizontal: 2,
  },
  profileButton: {
    width: 60,
    height: 60,
```

```
  borderRadius: 30,
  backgroundColor: '#1e90ff',
  justifyContent: 'center',
  alignItems: 'center',
},
profileIcon: {
  width: 40,
  height: 40,
  tintColor: '#fff',
  elevation: 7,
},
inputContainer: {
  width: '90%',
  height: 50,
  backgroundColor: 'rgba(255, 255, 255, 0.9)',
  borderRadius: 15,
  alignSelf: 'center',
  justifyContent: 'center',
  paddingLeft: 50,
  marginVertical: 3,
  elevation: 7,
},
icon: {
  width: 40,
  height: 40,
  position: 'absolute',
  left: 10,
  top: '50%',
  transform: [{ translateY: -20 }],
},
searchInputPlaceholder: {
  fontSize: 17,
  color: 'rgba(0,0,0,0.5)',
  marginHorizontal: 5,
},

searchInput: {
  fontSize: 17,
  color: '#000',
  height: '100%',
},
twoCardContainer: {
  height: 250,
  flexDirection: 'row',
  justifyContent: 'space-evenly',
  alignItems: 'center',
  paddingHorizontal: 25,
```

```
    paddingVertical: 2,
    gap: 8,
  },
  cardOne: {
    backgroundColor: '#fff',
    width: '55%',
    height: '75%',
    justifyContent: 'center',
    alignItems: 'center',
    borderRadius: 20,
    elevation: 7,
  },
  cardTwo: {
    backgroundColor: '#fff',
    width: '38%',
    height: '75%',
    borderRadius: 15,
    justifyContent: 'center',
    alignItems: 'center',
    elevation: 7,
  },
  cardOneImage: {
    width: '115%',
    height: '100%',
    resizeMode: 'contain',
  },
  cardTwoImage: {
    width: '100%',
    height: '87%',
    resizeMode: 'contain',
    position: 'absolute',
    top: 24,
    left: 4,
  },
});

export default HomeScreen;
```

Scanner.tsx

```
import React, { useState, useEffect, useCallback } from 'react';
import { useNavigation } from '@react-navigation/native';
import {
  View,
  Text,
```

```
  StyleSheet,
  Alert,
  SafeAreaView,
  TouchableOpacity,
  Image,
} from 'react-native';
import {
  Camera,
  useCameraDevices,
  useCameraPermission,
  useCodeScanner,
  Code,
} from 'react-native-vision-camera';

const Scanner: React.FC = () => {
  const navigation = useNavigation<any>();

  const devices = useCameraDevices();
  const device = devices.filter(device => device.position === 'back')[0]; // back camera
  const [barcode, setBarcode] = useState<string | null>(null);
  const [scanningEnabled, setScanningEnabled] = useState<boolean>(true);

  const [torchOn, setTorchOn] = useState<boolean>(false);

  // Correct usage based on your hook implementation
  const { hasPermission, requestPermission } = useCameraPermission();

  useEffect(() => {
    (async () => {
      if (!hasPermission) {
        await requestPermission();
      }
    })();
  }, [hasPermission, requestPermission]);

  // Callback invoked when codes are detected
  const onCodesDetected = useCallback(
    (codes: Code[]) => {
      if (!scanningEnabled) return;

      if (codes.length > 0) {
        const first = codes[0];
        if (first.value) {
          setBarcode(first.value);
          setTorchOn(false);
          setScanningEnabled(false); // disable further scanning until reset
          Alert.alert('Scanned Code', first.value, [
```

```
          {
            text: 'OK',
            onPress: () => setScanningEnabled(true),
          },
        ]);
      }
    }
  },
  [scanningEnabled],
);

const codeScanner = useCodeScanner({
  codeTypes: ['qr', 'ean-13', 'ean-8', 'code-128'], // supported formats
  onCodeScanned: onCodesDetected,
});

if (!device) {
  return (
    <View style={styles.center}>
      <Text>Loading camera...</Text>
    </View>
  );
}

if (!hasPermission) {
  return (
    <View style={styles.center}>
      <Text>No camera permission</Text>
    </View>
  );
}

return (
  <SafeAreaView style={styles.container}>
    <Camera
      style={StyleSheet.absoluteFill}
      device={device}
      isActive={scanningEnabled}
      codeScanner={codeScanner}
      torch={torchOn ? 'on' : 'off'}
    />
    <View style={styles.resultBox}>
      <Text style={styles.resultText}>
        {barcode ? `Scanned Code: ${barcode}` : 'Scan a barcode'}
      </Text>
      {barcode && (
        <TouchableOpacity
```

```
            style={{
              marginTop: 12,
              backgroundColor: '#1e90ff',
              paddingVertical: 10,
              paddingHorizontal: 20,
              borderRadius: 8,
            }}
            onPress={() => {
              const scannedCode = barcode; // store code for navigation
              setBarcode(null); //  clear previous scanned code
              setScanningEnabled(true);
              navigation.navigate('ProductDetails', { code: scannedCode });
            }}
          >
            <Text style={{ color: '#fff', fontWeight: 'bold' }}>
              Know More About the Product
            </Text>
          </TouchableOpacity>
        )}

        <TouchableOpacity
          onPress={() => setTorchOn(prev => !prev)}
          style={{
            flexDirection: 'row',
            alignItems: 'center',
            backgroundColor: torchOn
              ? 'rgba(255,255,255,0.7)'
              : 'rgba(0,0,0,0.3)',
            paddingVertical: 8,
            paddingHorizontal: 10,
            borderRadius: 8,
            marginTop: 10,
          }}
        >
          <Image
            source={require('../assets/icons/torch.png')}
            style={{
              tintColor: torchOn ? 'black' : '#fff',
              height: 35,
              width: 35,
            }}
          />
        </TouchableOpacity>
      </View>
    </SafeAreaView>
  );
};
```

```
const styles = StyleSheet.create({
  container: { flex: 1, backgroundColor: '#000' },
  center: { flex: 1, justifyContent: 'center', alignItems: 'center' },
  resultBox: {
    position: 'absolute',
    bottom: 40,
    width: '100%',
    alignItems: 'center',
  },
  resultText: {
    fontSize: 18,
    fontWeight: 'bold',
    color: '#fff',
    backgroundColor: 'rgba(0,0,0,0.6)',
    padding: 10,
    borderRadius: 8,
  },
});

export default Scanner;
```

Backend

db.ts

```
import mongoose from 'mongoose';
const connectDB = async () => {
  try {
    if (!process.env.MONGO_URL) throw new Error('MONGO_URL not defined');
    await mongoose.connect(process.env.MONGO_URL);
    console.log('MongoDB connected');
  } catch (err) {
    console.error('MongoDB connection failed: ', err);
    process.exit(1);
  }
};

export default connectDB;
```

url.ts

```
const BASE_URL =
  process.env.NODE_ENV === 'production'
    ? process.env.PROD_URL
```

```
    : process.env.LOCAL_URL;

export default BASE_URL;
```

planController.tsx

```
import { Request, Response } from 'express';
import User, { IPlan } from '../models/user';
import mongoose from 'mongoose';

type MealType = 'breakfast' | 'lunch' | 'dinner';

// Fetch meal plan for a profile
export const fetchMeal = async (req: Request, res: Response) => {
  const { userId, profileId } = req.params;
  try {
    const user = await User.findById(userId);
    if (!user) return res.status(404).json({ message: 'User not found' });

    const profile = user.profile.find(p => p._id.toString() === profileId);
    if (!profile) return res.status(404).json({ message: 'Profile not found' });

    return res.status(200).json({
      breakfast: profile.plan[0]?.breakfast || [],
      lunch: profile.plan[0]?.lunch || [],
      dinner: profile.plan[0]?.dinner || [],
    });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Server error' });
  }
};

// Add single meal item
export const addMealItem = async (req: Request, res: Response) => {
  const { userId, profileId } = req.params;
  const { mealType, item } = req.body as { mealType: MealType; item: string };

  try {
    if (!['breakfast', 'lunch', 'dinner'].includes(mealType))
      return res.status(400).json({ message: 'Invalid meal type' });

    const user = await User.findById(userId);
    if (!user) return res.status(404).json({ message: 'User not found' });
```

```
    const profile = user.profile.find(p => p._id.toString() === profileId);
    if (!profile) return res.status(404).json({ message: 'Profile not found' });

    // Initialize plan if missing
    if (!profile.plan || profile.plan.length === 0) {
      profile.plan = [{ breakfast: [], lunch: [], dinner: [] }];
    }

    const plan = profile.plan[0] as IPlan;
    plan[mealType] = plan[mealType] || [];
    plan[mealType].push(item);

    await user.save();

    return res.status(201).json({
      message: `Item added to ${mealType}`,
      [mealType]: plan[mealType],
    });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Server error' });
  }
};

// Delete a meal item
export const deleteMealItem = async (req: Request, res: Response) => {
  const { userId, profileId } = req.params;
  const { mealType, item } = req.body as { mealType: MealType; item: string }; // Get the 'item'
value instead of 'index'

  try {
    const user = await User.findById(userId);
    if (!user) return res.status(404).json({ message: 'User not found' });

    const profile = user.profile.find(p => p._id.toString() === profileId);
    if (!profile) return res.status(404).json({ message: 'Profile not found' });

    const plan = profile.plan[0];
    if (!plan || !plan[mealType]) {
      return res.status(400).json({ message: 'Invalid meal plan or type' });
    }

    // Find the index of the item to delete
    const itemIndex = plan[mealType].indexOf(item);

    if (itemIndex === -1) {
      return res.status(404).json({ message: 'Meal item not found' });
```

```
    }

    plan[mealType].splice(itemIndex, 1);
    await user.save();

    return res.status(200).json({
      message: `Item removed from ${mealType}`,
      [mealType]: plan[mealType],
    });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Server error' });
  }
};

///fetching ProfileId
export const fetchProfile = async (req: Request, res: Response) => {
  const { userId } = req.params;

  try {
    const user = await User.findById(userId);

    if (!user) return res.status(404).json({ message: 'User not found' });
    if (!user.profile || user.profile.length === 0) {
      return res.status(404).json({ message: 'No profiles found' });
    }

    // Safe access with optional chaining
    const firstProfileId = user.profile?.[0]?._id;

    if (!firstProfileId) {
      return res.status(404).json({ message: 'No profile ID found' });
    }

    return res.status(200).json({ profileId: firstProfileId });
  } catch (err) {
    console.error(err);
    return res.status(500).json({ message: 'Server error' });
  }
};
```

profileFecthController.tsx

```
import { Request, Response } from 'express';
```

```
import User from '../models/user'; // Adjust path as needed

// Get profile by ID
export const getProfileById = async (req: Request, res: Response) => {
  const { id } = req.params;
  try {
    // Find the user containing this profile
    const user = await User.findOne({ 'profile._id': id });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Use Array.find() instead of .id()
    const profile = user.profile.find((p: any) => p._id.toString() === id);
    if (!profile) {
      return res.status(404).json({ message: 'Profile not found' });
    }

    return res.status(200).json(profile);
  } catch (error) {
    return res.status(500).json({ message: 'Internal server error' });
  }
};

//  Update profile by ID
export const updateProfileById = async (req: Request, res: Response) => {
  const { id } = req.params;
  const updates = req.body;

  try {
    const user = await User.findOne({ 'profile._id': id });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    // Use Array.find() instead of .id()
    const profile = user.profile.find((p: any) => p._id.toString() === id);
    if (!profile) {
      return res.status(404).json({ message: 'Profile not found' });
    }

    // Merge updates into profile
    Object.assign(profile, updates);

    await user.save();
    return res.status(200).json(profile);
  } catch (error) {
```

```
    return res.status(500).json({ message: 'Internal server error' });
  }
};
```

userController.tsx

```
import { Request, Response } from 'express';
import User from '../models/user';
import { profile } from 'console';

// Register User
export const registerUser = async (req: Request, res: Response) => {
  const { fullName, email, password } = req.body;

  try {
    // Check if user already exists
    const existingUser = await User.findOne({ email });
    if (existingUser) {
      return res.status(400).json({ message: 'User already exists' });
    }

    // Create new user
    const newUser = new User({ fullName, email, password });
    await newUser.save();

    return res.status(201).json({ message: 'User registered successfully' });
  } catch (err) {
    console.error('Register error:', err);
    res.status(500).json({ message: 'Error registering user', error: err });
  }
};

// Login User
export const loginUser = async (req: Request, res: Response) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });

    if (!user) {
      // Email not found
      return res.status(401).json({ errors: { email: 'Email not found' } });
    }
```

```
    if (user.password !== password) {
      // Password incorrect
      return res
        .status(401)
        .json({ errors: { password: 'Incorrect password' } });
    }

    // Successful login
    return res.json({
      message: 'Login successful',
      user: user,
    });
  } catch (err) {
    console.error('Login error:', err);
    return res.status(500).json({ errors: { general: 'Error logging in' } });
  }
};

//Add New Profile
export const addProfile = async (req: Request, res: Response) => {
  const { userId, profile } = req.body;

  if (!userId || !profile) {
    return res.status(400).json({ message: 'UserId and Profile are required' });
  }

  try {
    //Find the user by ID
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    if (!user.profile) {
      user.profile = [];
    }

    if (user.profile.length >= 3) {
      return res
        .status(400)
        .json({ message: 'You can only make up to 3 profiles' });
    }
    user.profile.push(profile);
    await user.save();

    return res
      .status(200)
```

```
      .json({ message: 'Profile added successfully', user });
  } catch (error) {
    console.error('Error adding profile:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
};

// Get profiles of a user
export const getProfiles = async (req: Request, res: Response) => {
  const { userId } = req.params;

  try {
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }

    return res.status(200).json(user.profile || []);
  } catch (error) {
    console.error('Error fetching profiles:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
};

//fetch profiles and return
export const fetchData = async (req: Request, res: Response) => {
  try {
    const users = await User.find({}, { fullName: 1, email: 1, profile: 1 });
    // returning only necessary fields

    return res.status(200).json(users);
  } catch (error) {
    console.error('Error fetching data:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
};

// Delete a profile from a user
export const deleteProfile = async (req: Request, res: Response) => {
  const { userId, profileId } = req.params;
  try {
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    // Remove profile by _id (MongoDB ObjectId)
    user.profile = user.profile.filter(
```

```
      (p: any) => String(p._id ?? p.id) !== profileId,
    );
    await user.save();
    return res.status(200).json({ message: 'Profile deleted' });
  } catch (error) {
    console.error('Error deleting profile:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
};
```

Modal/users.tsx

```
import mongoose, { Schema, Document, Mongoose } from 'mongoose';

export interface IPlan {
  breakfast: string[];
  lunch: string[];
  dinner: string[];
}
const planSchema: Schema = new Schema({
  breakfast: { type: Array, required: false },
  lunch: { type: Array, required: false },
  dinner: { type: Array, required: false },
});

export interface IProfile extends Document {
  _id: mongoose.Types.ObjectId;
  name: string;
  age: string;
  gender: string;
  weight: string;
  height: string;
  dietType: string;
  allergies?: string;
  healthGoal: string;
  activityLevel: string;
  medicalConditions?: string;
  plan: IPlan[];
}

const profileSchema: Schema = new Schema({
  name: { type: String, required: true },
  age: { type: String, required: true },
  gender: { type: String, required: true },
```

```
  weight: { type: String, required: true },
  height: { type: String, required: true },
  dietType: { type: String, required: true },
  allergies: { type: String },
  healthGoal: { type: String, required: true },
  activityLevel: { type: String, required: true },
  medicalConditions: { type: String },
  plan: {
    type: [planSchema],
    default: [{ breakfast: [], lunch: [], dinner: [] }], // <-- default empty plan
  },
});

export interface IUser extends Document {
  fullName: string;
  email: string;
  password: string;
  profile: IProfile[];
  resetToken?: string | null;
  resetTokenExpiry?: Date | null;
}

const userSchema: Schema = new Schema({
  fullName: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  profile: { type: [profileSchema], default: [] },
  resetToken: { type: String, default: '' },
  resetTokenExpiry: { type: Date, default: null },
});

const User = mongoose.model<IUser>('User', userSchema);
export default User;
```

routes/userRoutes.tsx

```
import express from 'express';
import {
  registerUser,
  loginUser,
  addProfile,
  getProfiles,
  fetchData,
  deleteProfile,
```

```
} from '../controllers/userController';

import {
  getProfileById,
  updateProfileById,
} from '../controllers/profileFetchController';

import {
  fetchMeal,
  addMealItem,
  deleteMealItem,
  fetchProfile,
} from '../controllers/planController';

import { fetchProfileName } from '../controllers/nameController';
import {
  forgotPassword,
  verifyOtp,
  resetPassword,
} from '../controllers/forgetPassController';

const router = express.Router();

//User Authentication
router.post('/register', registerUser);
router.post('/login', loginUser);

//ProfileManagement
router.post('/addProfile', addProfile);
router.get('/:userId/profiles', getProfiles);

// Fetch all users + profiles (admin/global)
router.get('/fetchData', fetchData);

//deleteing profile
router.delete('/:userId/profiles/:profileId', deleteProfile);

//editing profile details + fetchig
router.get('/profile/:id', getProfileById);
router.put('/profile/:id', updateProfileById);

//Fetching meal
router.get('/:userId/:profileId/fetchMeal', fetchMeal);
router.post('/:userId/profiles/:profileId/addMeal', addMealItem);
router.delete('/:userId/profiles/:profileId/deleteMeal', deleteMealItem);

//fetching profile
```

```
router.get('/:userId/firstProfile', fetchProfile);

//fetching profileName
router.get('/:profileId/fetchName', fetchProfileName);

//forget password
router.post('/forgot-password', forgotPassword);
router.post('/verify-otp', verifyOtp);
router.post('/reset-password', resetPassword);

export default router;
```

server.ts

```
import express from 'express';
import cors from 'cors';
import userRoutes from './routes/userRoutes';
import connectDB from './config/db';
import dotenv from 'dotenv';
dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

connectDB();

//Routes
app.use('/api/users', userRoutes);

//start server
// Choose port dynamically
const PORT = process.env.PORT || 5000; // Render sets process.env.PORT automatically

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

6.4 Testing Approaches

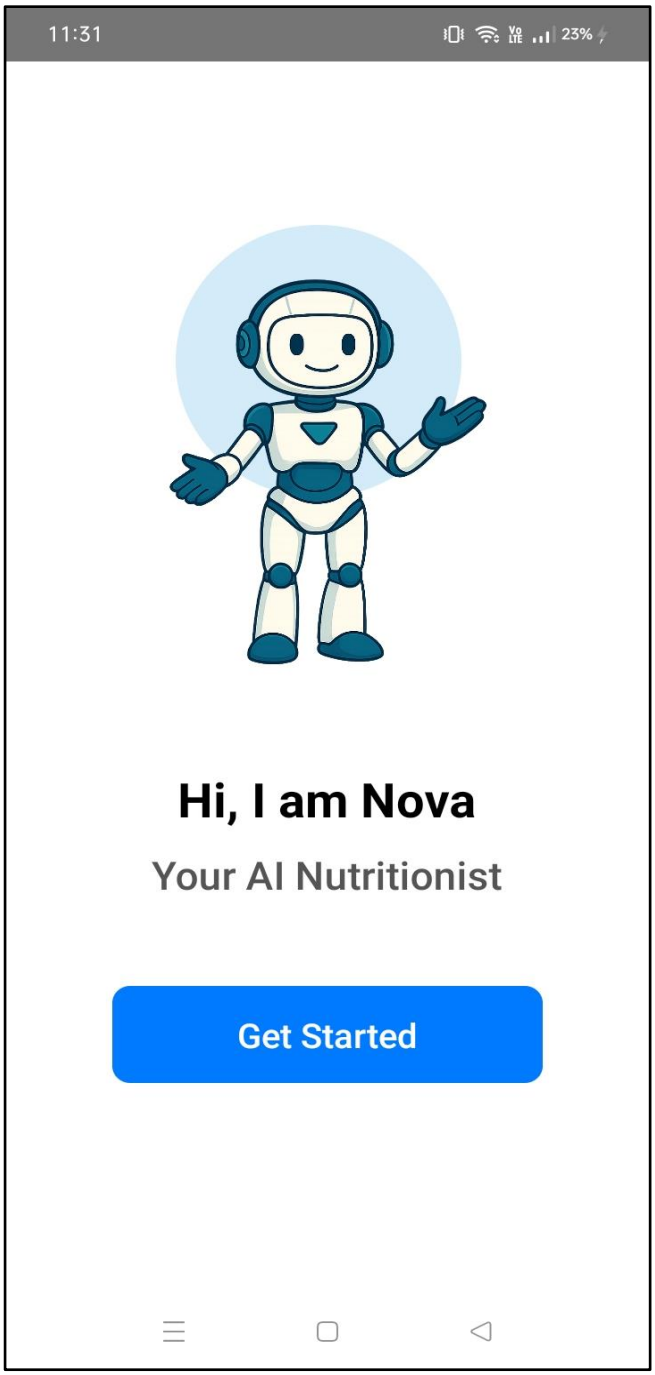
1. Unit Testing: Each individual component of the app (such as profile creation, AI diagnosis, meal planner, health tips, and barcode scanner) was tested separately to ensure that they function correctly in isolation. This helped detect small issues early during development.
2. Integration Testing: After unit testing, different modules were combined (e.g., profile data with AI diagnosis, or barcode scanning with food database) and tested together. This ensured smooth communication between components and consistent data flow across the system.
3. System Testing: The entire app was tested as a whole to verify that all features—profile management, diagnosis, diet planning, health tips, and scanning—work correctly on multiple Android Devices. This validated the overall functionality against the requirements.
4. User Acceptance Testing: Sample users tested the app to check usability, clarity of results, and ease of navigation. Their feedback was collected to improve the interface and enhance user experience.
5. Performance Testing: The app was tested under different conditions to measure response time for AI diagnosis, barcode scanning, and meal planning. The goal was to ensure that the system provides quick and reliable results even when handling multiple users.
6. Security Testing: User profile data and health details are sensitive. Security testing was done to verify that data is stored securely in the database, protected during transmission, and accessible only to authorized users.

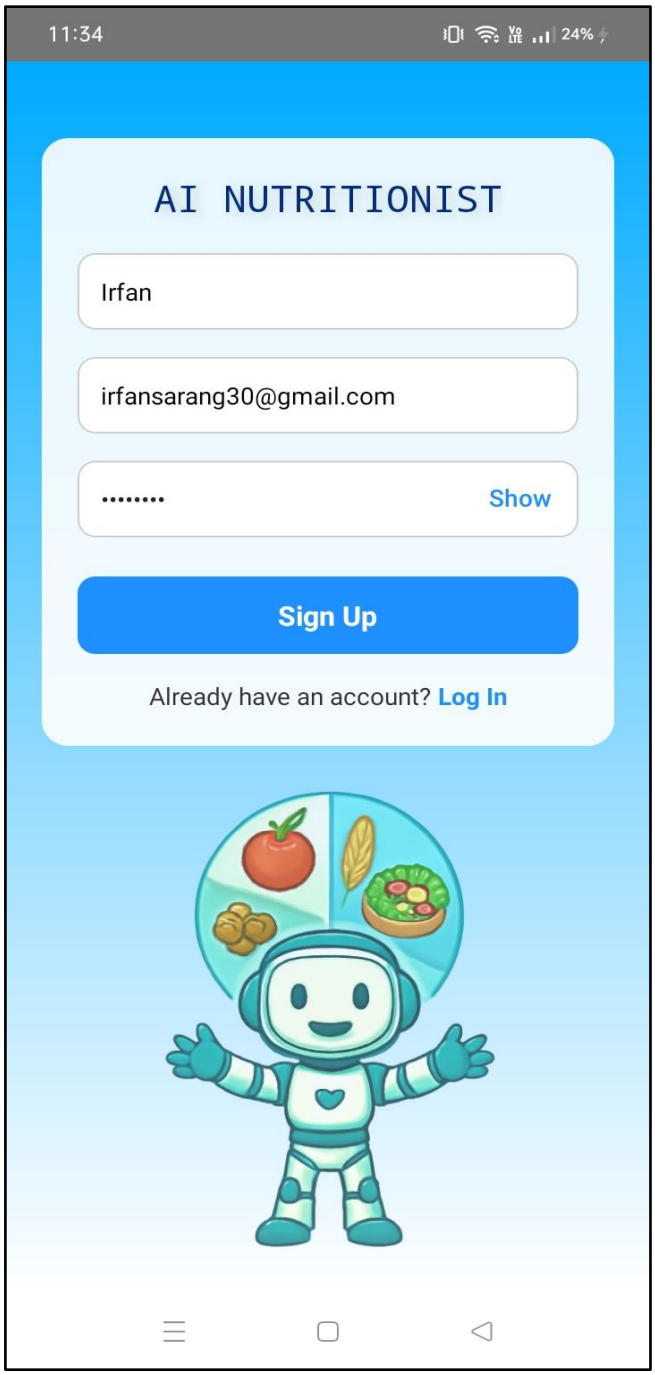
6.5 Test Cases

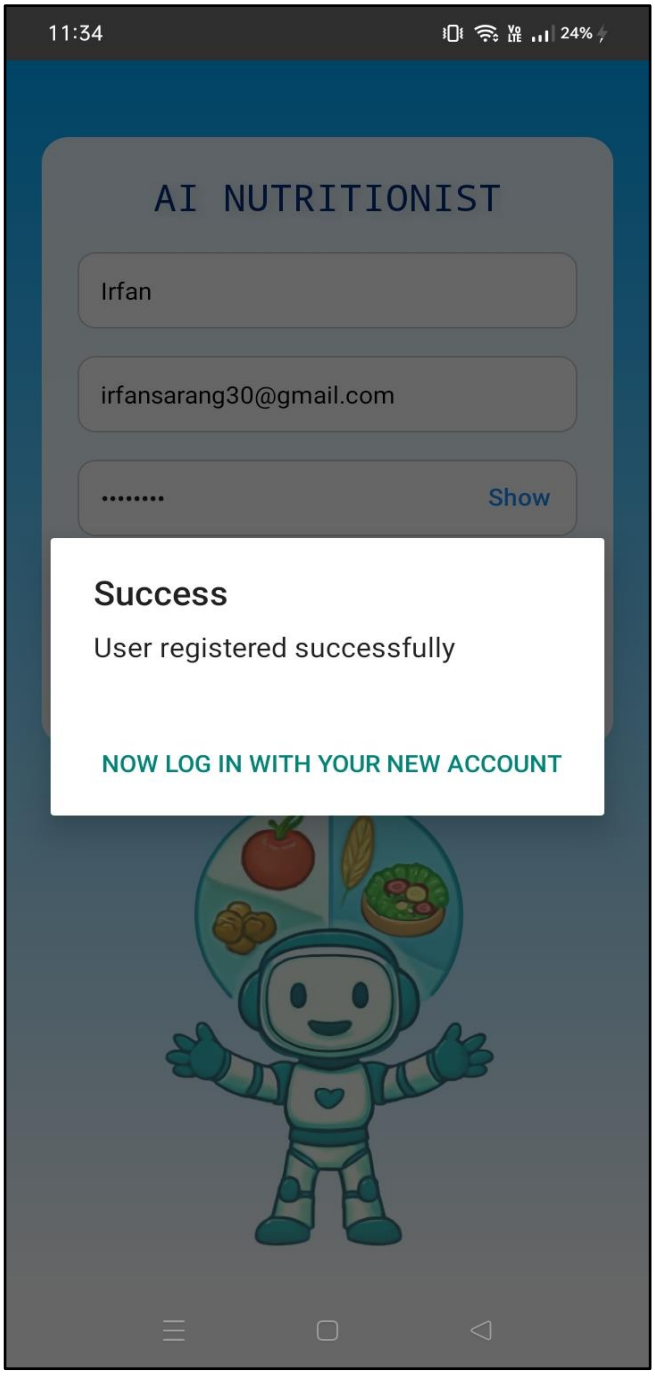
Test Case ID	Case	Actions	Expected Output	Actual Output	Status
1.	Signup	Enter Name, Email, Password → Tap Signup	Account created → Redirect to login page	Account created → Redirect to login page	Pass
2.	Login	Enter Email & Password → Tap Login	Login Successfully.	Login Successfully.	Pass
3.	Create Profile	Fill the Profile Registration and click Create Profile.	Profile Created Successfully.	Profile Created Successfully.	Pass
4.	Delete Profile	Tap Delete on selected profile	Profile Deleted	Profile Deleted	Pass
5.	AI Diagnosis	Tap Get AI Diagnosis in Profile Details.	Show diagnosis based on profile details.	Show diagnosis based on profile details.	Pass
6.	How Much Should I Consume Daily?	Fill form → Tap Calculate	Show daily calories, BMI analysis, water intake, and macro split	Show daily calories, BMI analysis, water intake, and macro split	Pass
7.	Diet Planner	Enter diet prompt → Send	Diet plan created from dataset	Diet plan created from dataset	Pass
8.	Add Meal in Meal Planner	Tap Add → Enter meal → Tap Save	Meal added to Meal Planner	Meal added to Meal Planner	Pass
9.	Delete Meal in Meal Planner	Tap X on the meal	Meal removed from Meal Planner	Meal removed from Meal Planner	Pass
10.	Scan Food Product Barcode	Tap Scan → Use camera to scan barcode	Barcode scanned and displayed	Barcode scanned and displayed	Pass
11.	Product Details	Tap Know More after scanning barcode	Show product details	Show product details	Pass
12.	AI Insights	Tap Get AI Insight in Product Details	Display AI insights of the product	Display AI insights of the product	Pass

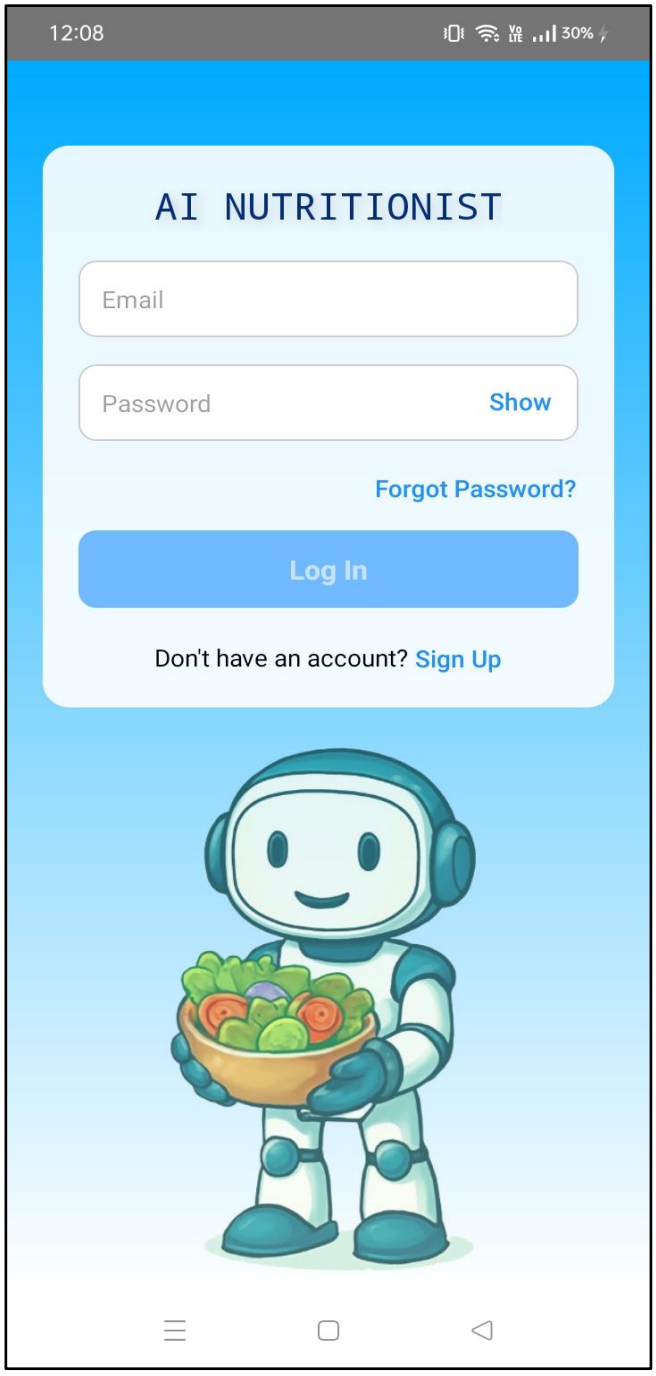
CHAPTER 7. RESULTS

1. Signup

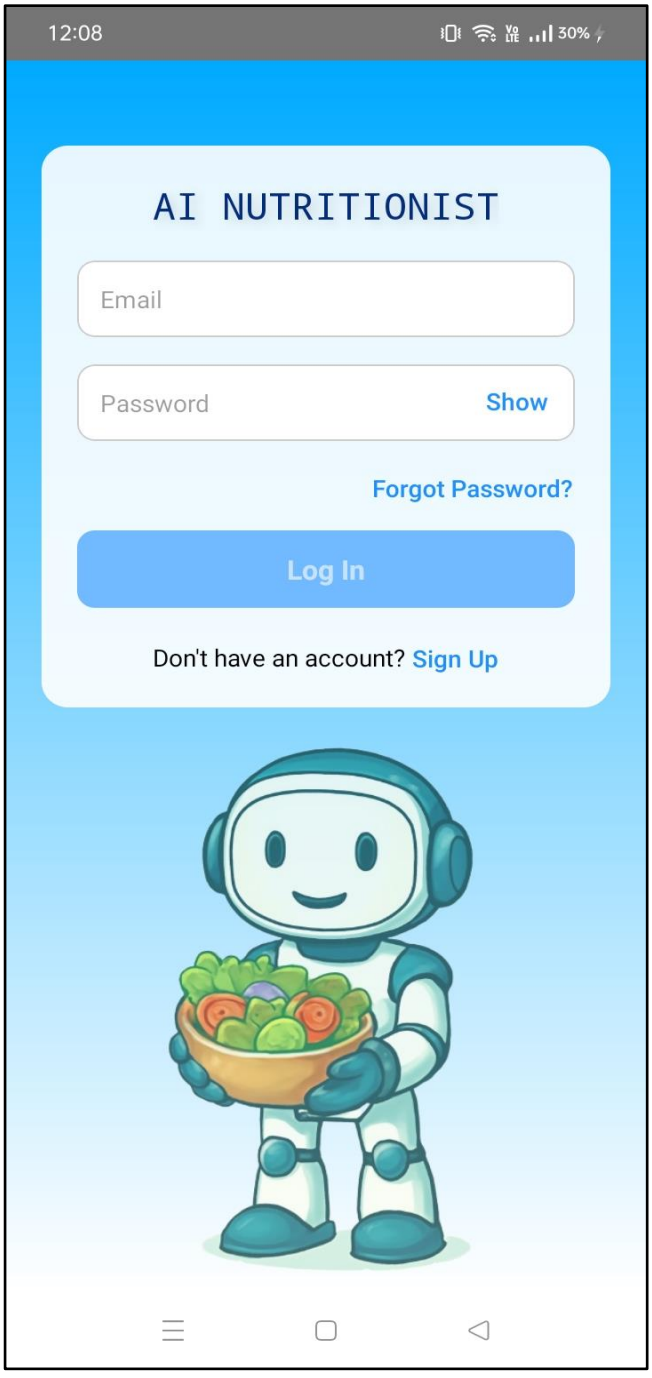


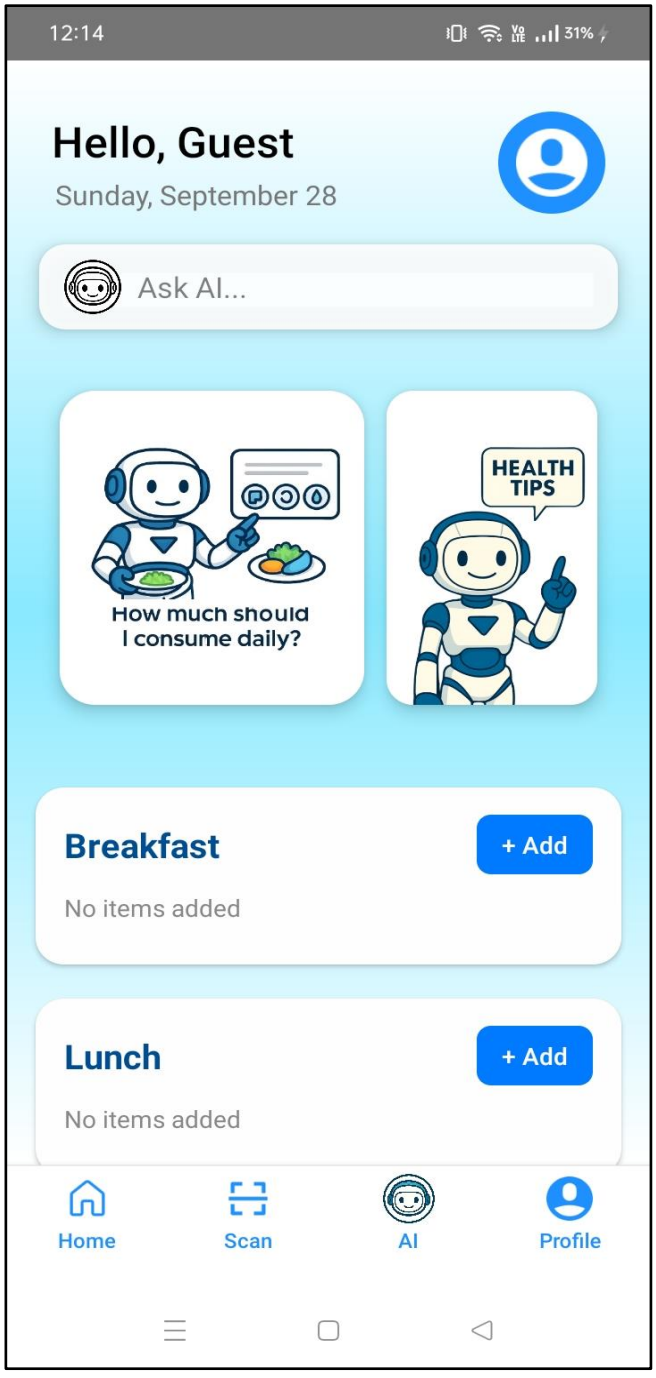



11:34
24%
AI NUTRITIONIST
Irfan
irfansarang30@gmail.com
Show
Success
User registered successfully
NOW LOG IN WITH YOUR NEW ACCOUNT

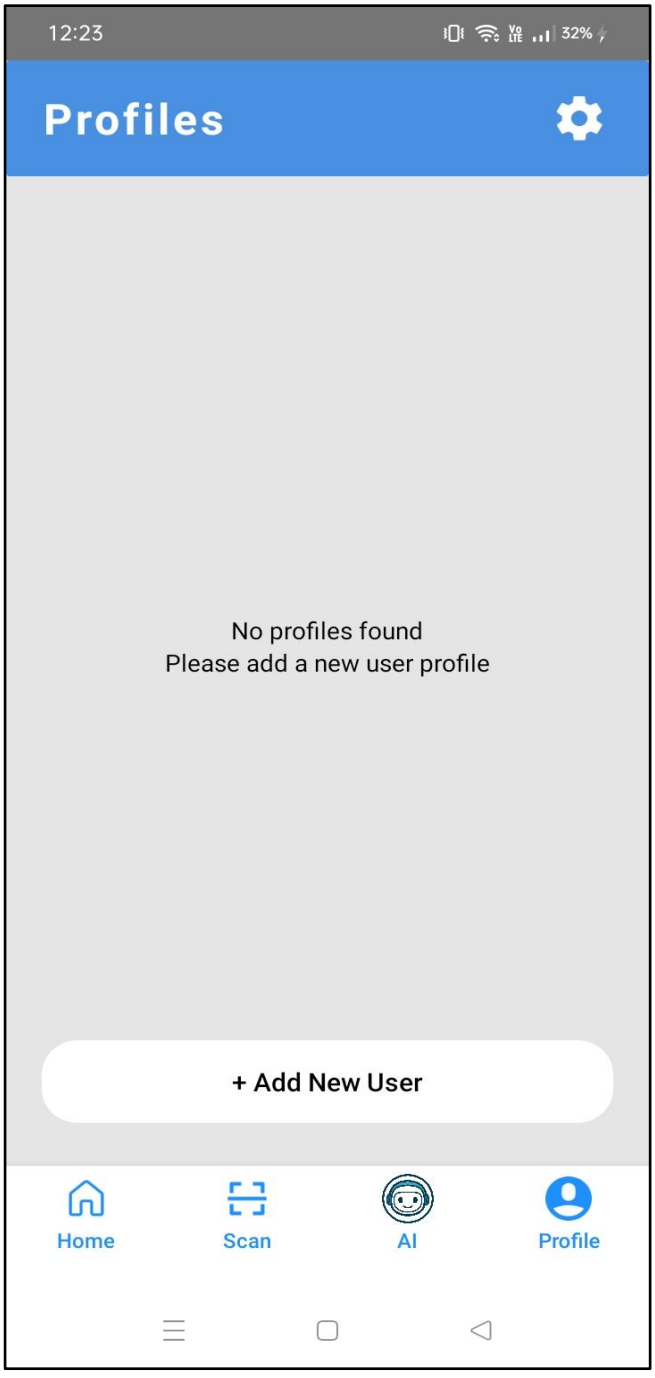

12:08
30%
AI NUTRITIONIST
Email
Password
Show
Forgot Password?
Log In
Don't have an account? Sign Up

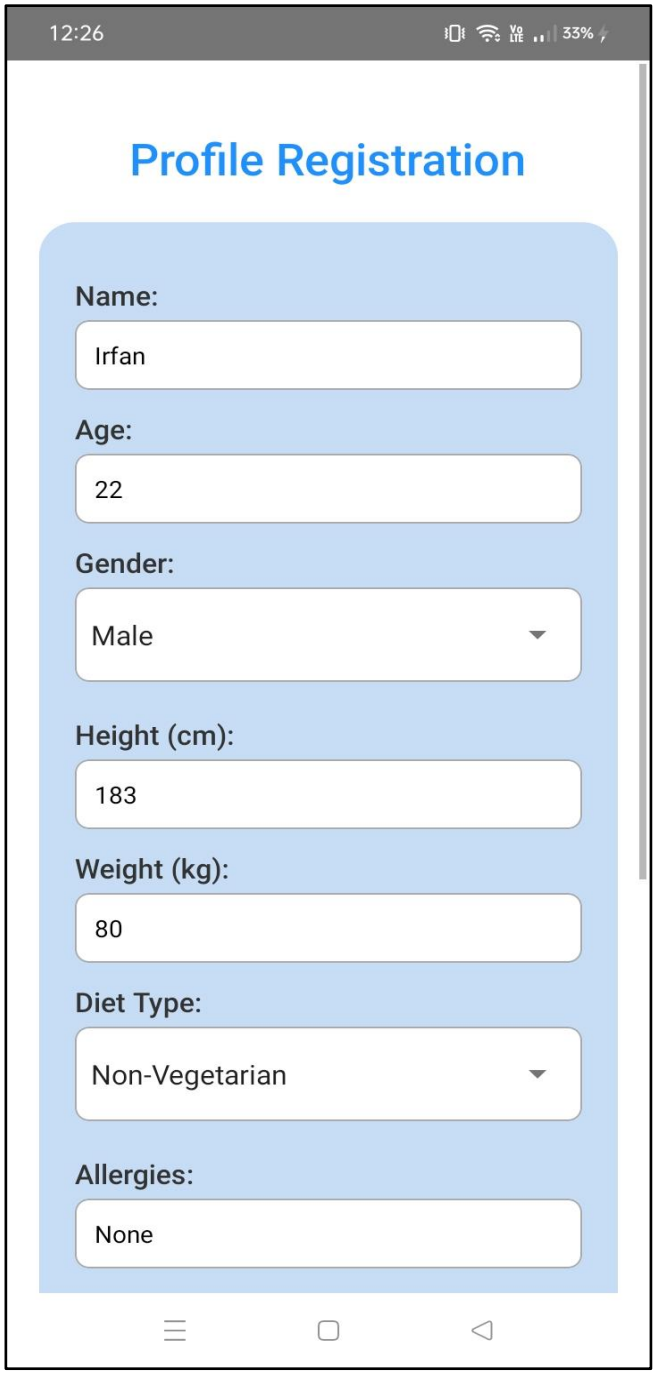
2. Login

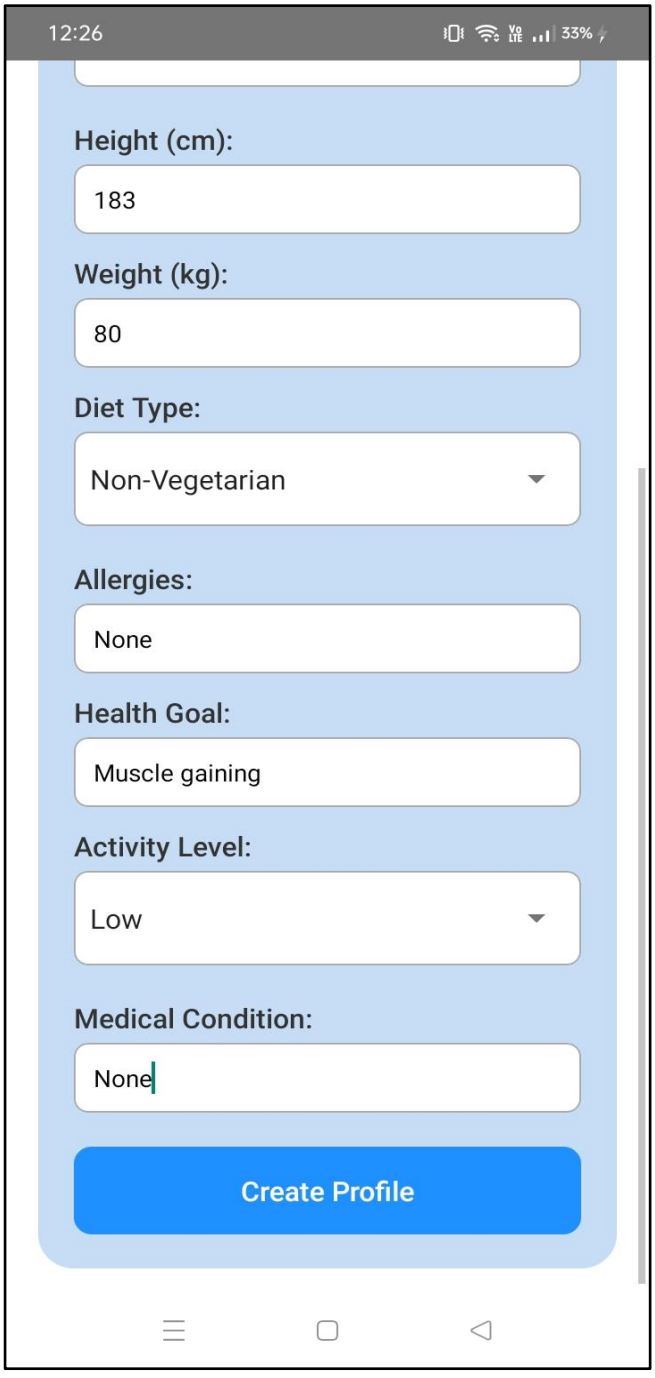

12:08
30%
AI NUTRITIONIST
Email
Password
Show
Forgot Password?
Log In
Don't have an account? Sign Up

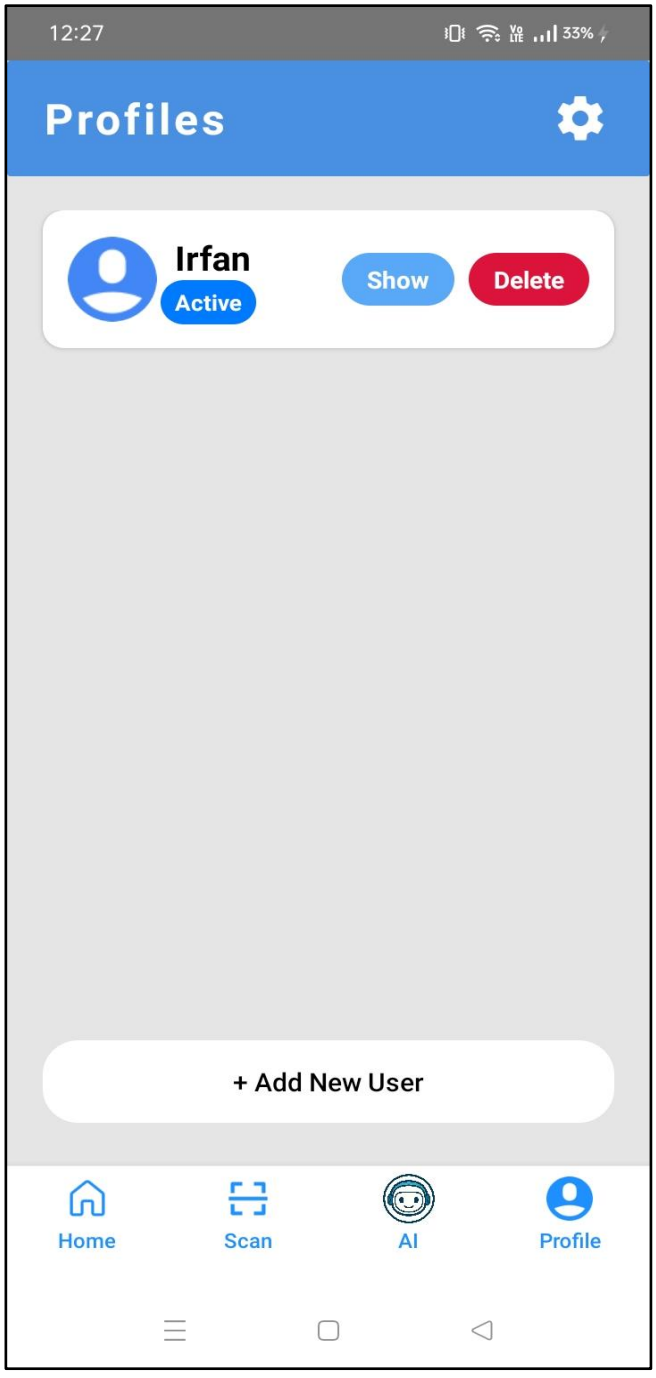

12:14
31%
Hello, Guest
Sunday, September 28
Ask AI...
How much should I consume daily?
HEALTH TIPS
Breakfast
+ Add
No items added
Lunch
+ Add
No items added
Home
Scan
AI
Profile

3. Create Profile

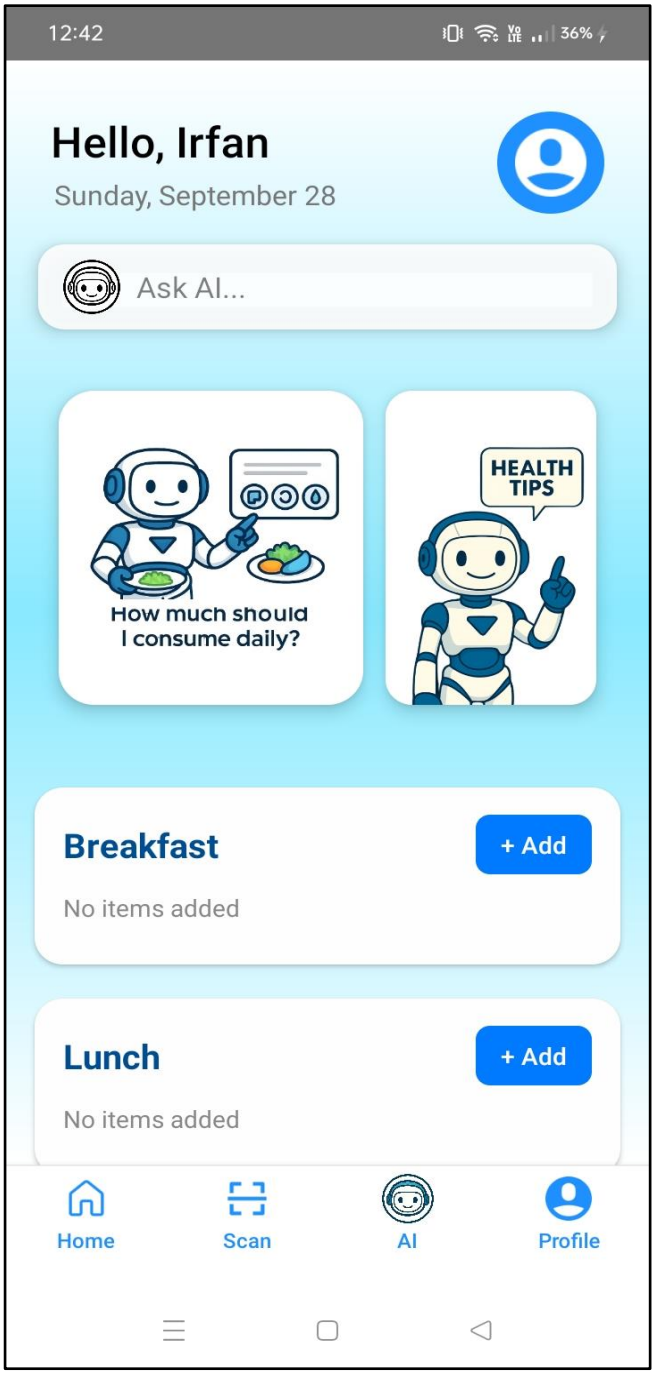


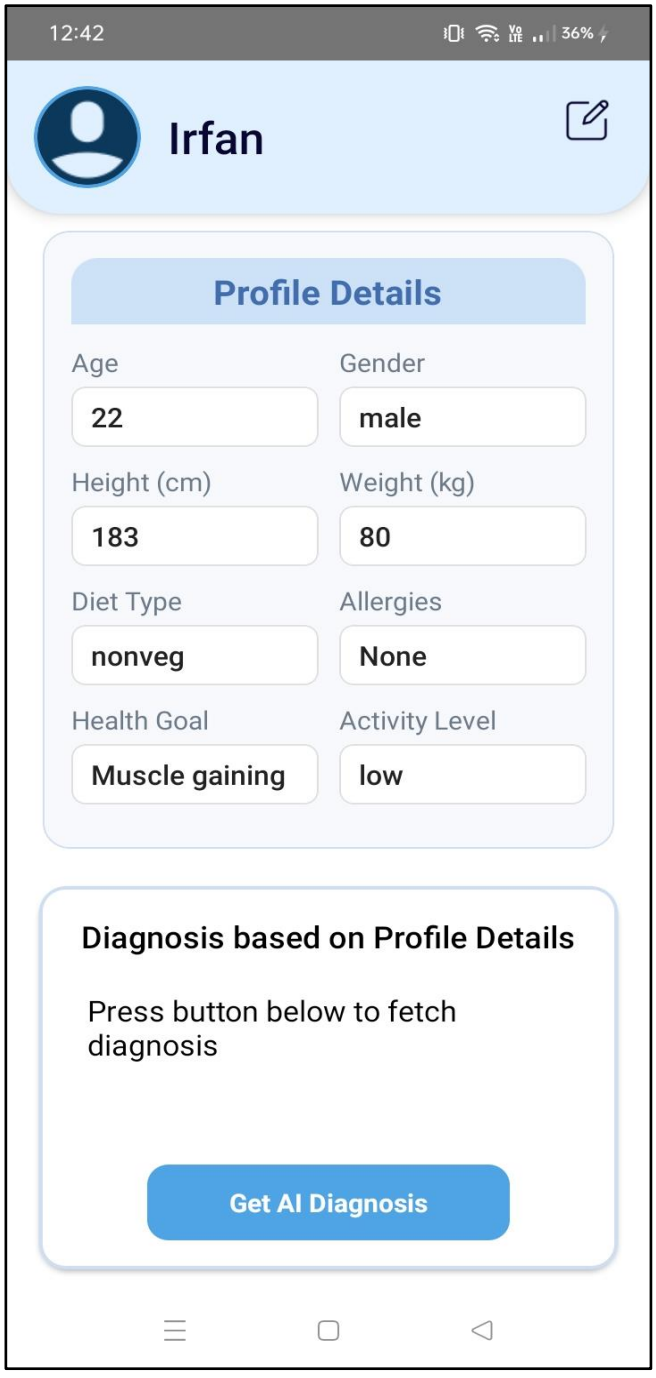


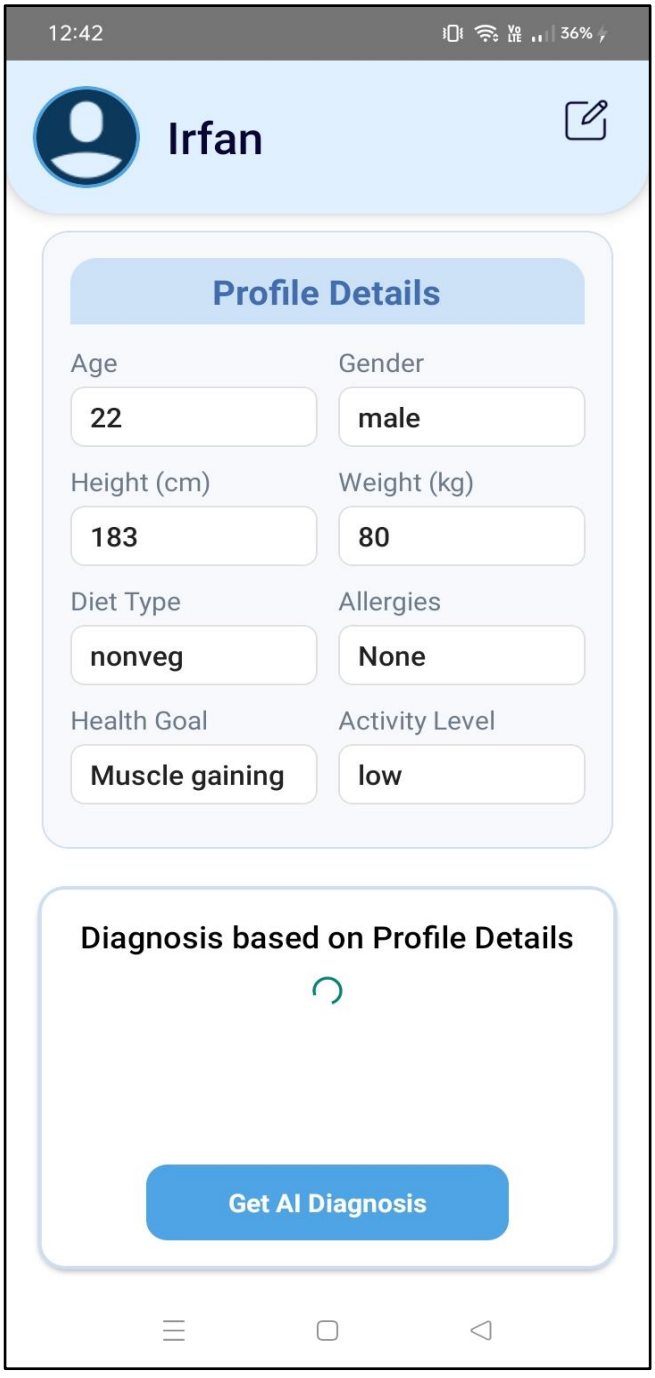

12:26
33%
Height (cm):
183
Weight (kg):
80
Diet Type:
Non-Vegetarian
Allergies:
None
Health Goal:
Muscle gaining
Activity Level:
Low
Medical Condition:
None
Create Profile

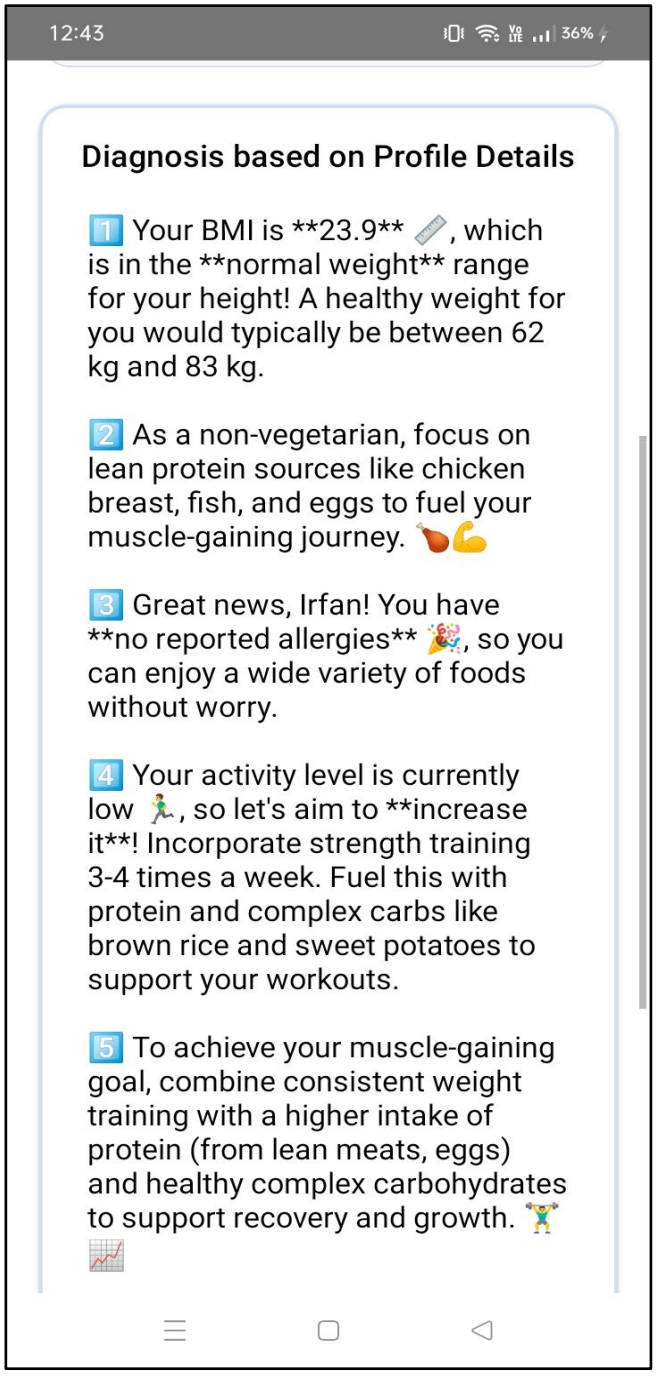

12:27
33%
Profiles
Irfan
Active
Show
Delete
+ Add New User
Home
Scan
AI
Profile

4. AI Diagnosis

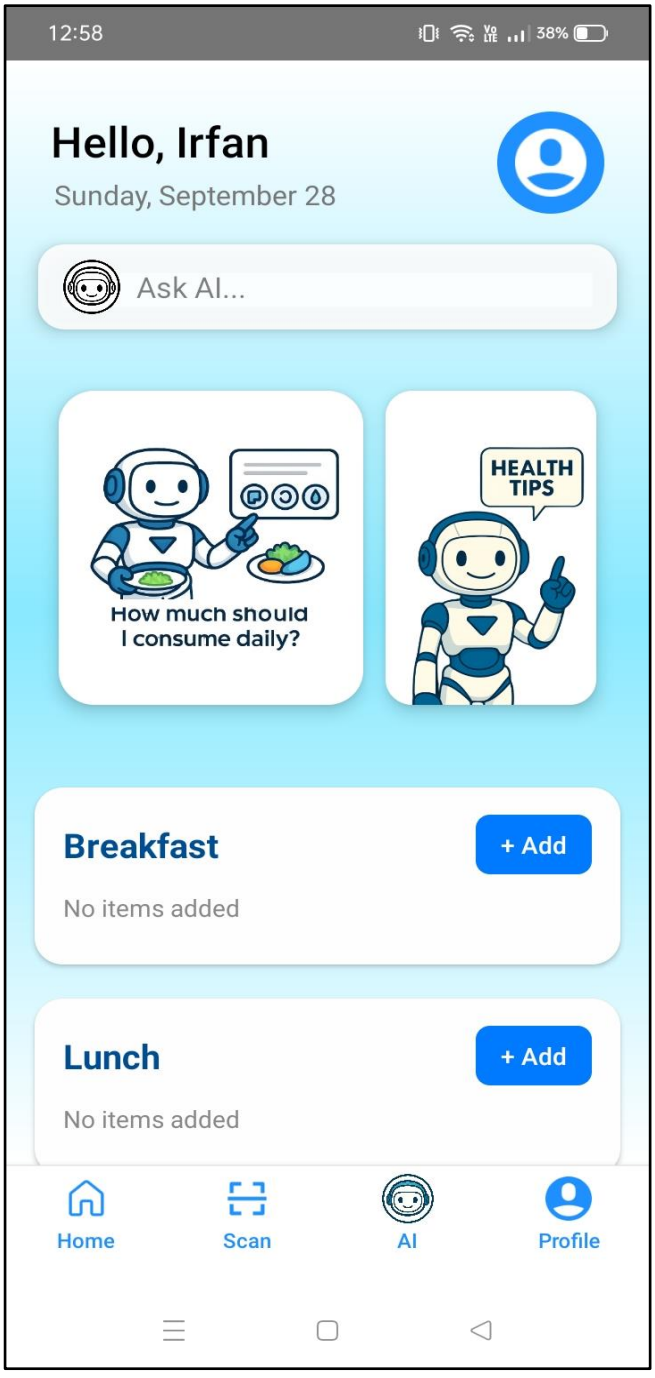


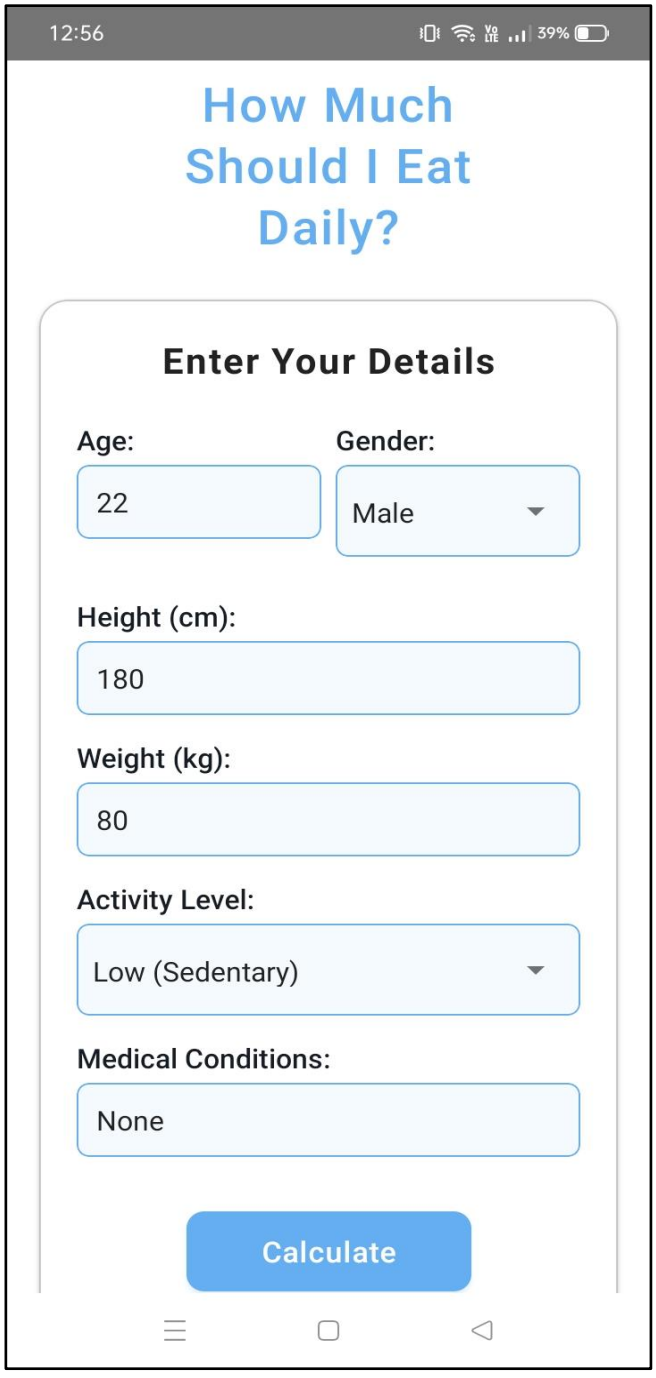


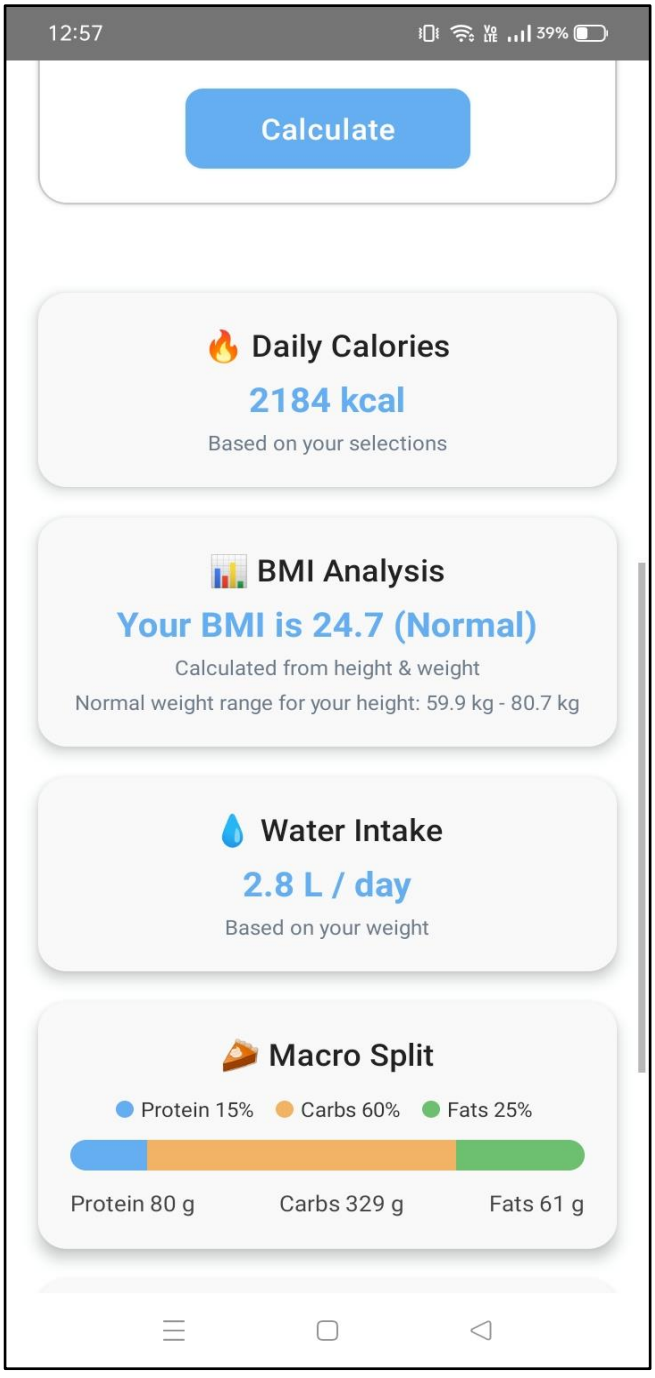

12:42
36%
Irfan
Profile Details
Age
22
Gender
male
Height (cm)
183
Weight (kg)
80
Diet Type
nonveg
Allergies
None
Health Goal
Muscle gaining
Activity Level
low
Diagnosis based on Profile Details
Get AI Diagnosis

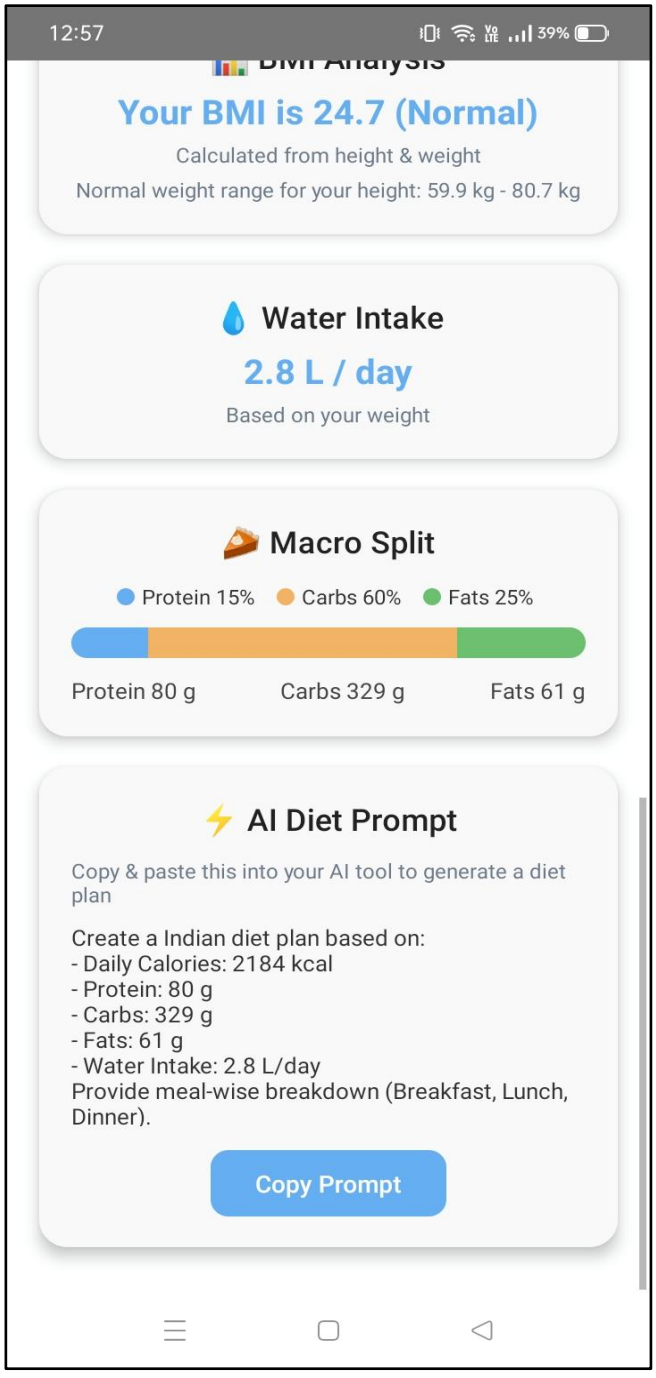

12:43
36%
Diagnosis based on Profile Details
1️⃣ Your BMI is **23.9** 📏, which is in the **normal weight** range for your height! A healthy weight for you would typically be between 62 kg and 83 kg.
2️⃣ As a non-vegetarian, focus on lean protein sources like chicken breast, fish, and eggs to fuel your muscle-gaining journey. 🍗💪
3️⃣ Great news, Irfan! You have **no reported allergies** 🎉, so you can enjoy a wide variety of foods without worry.
4️⃣ Your activity level is currently low 🏃, so let's aim to **increase it**! Incorporate strength training 3-4 times a week. Fuel this with protein and complex carbs like brown rice and sweet potatoes to support your workouts.
5️⃣ To achieve your muscle-gaining goal, combine consistent weight training with a higher intake of protein (from lean meats, eggs) and healthy complex carbohydrates to support recovery and growth. 🏋️📈

5. How Much Should I Consume Daily?

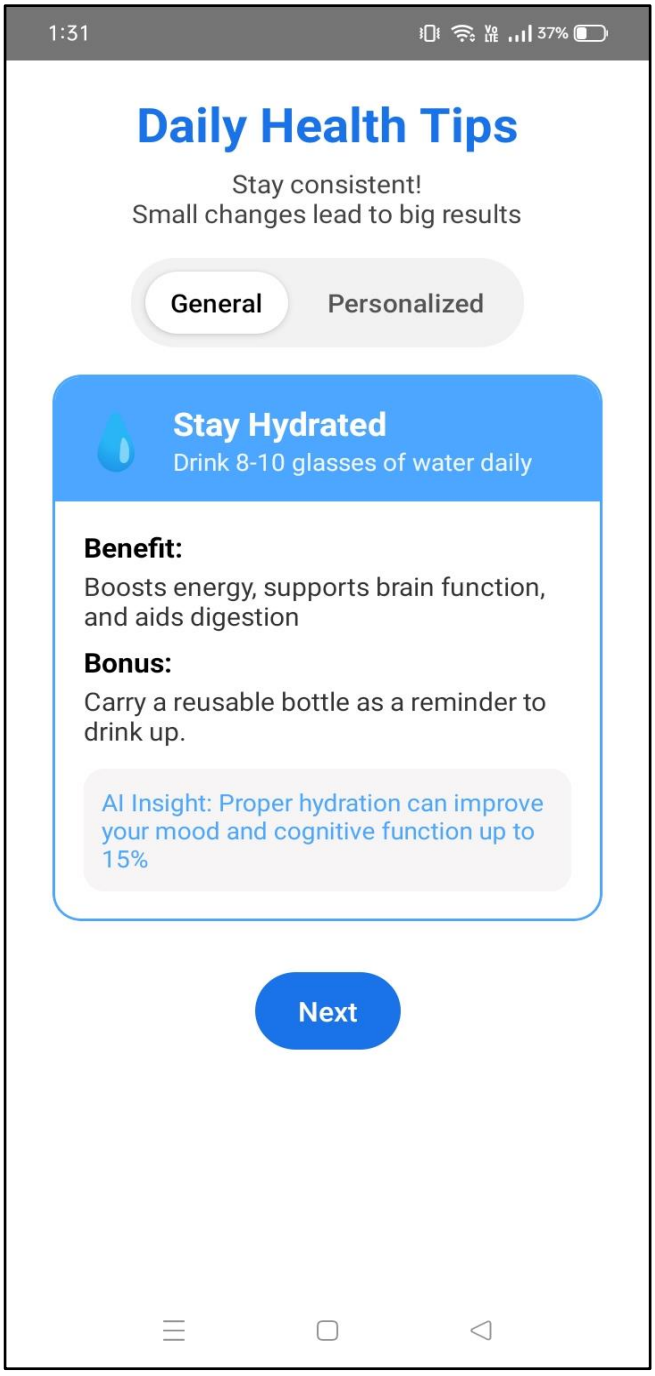


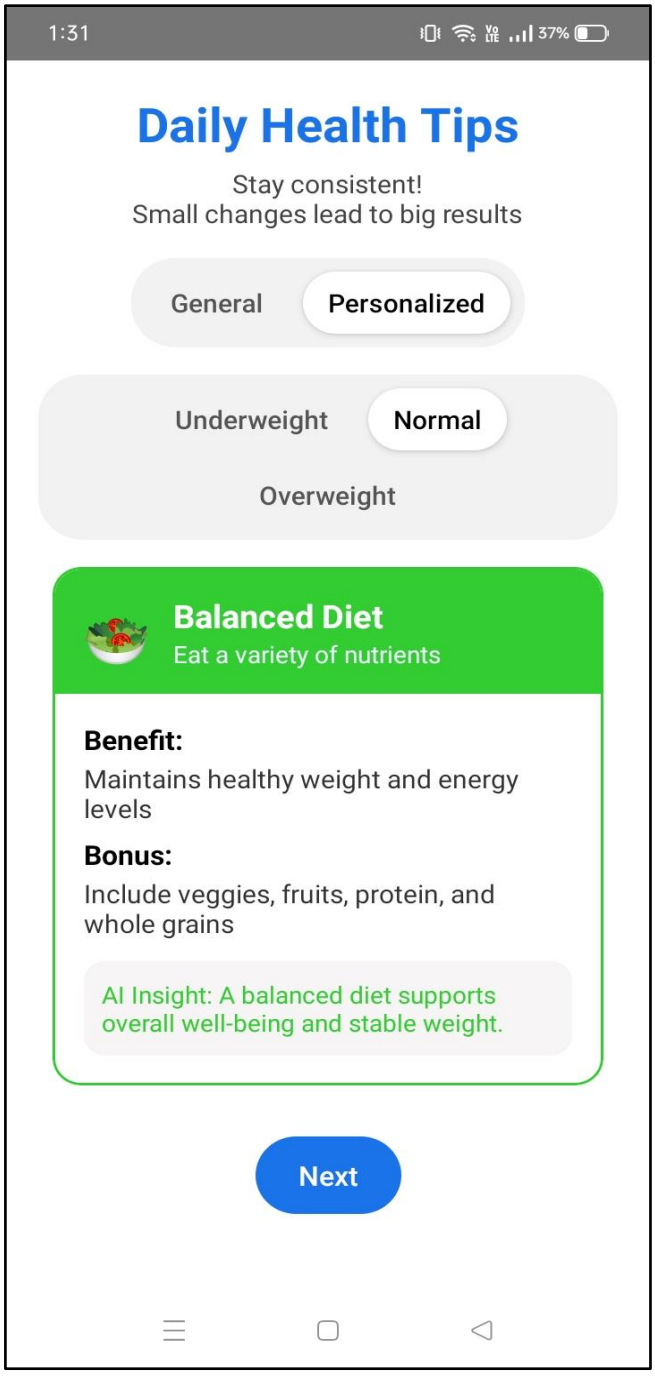



12:57
39%
Calculate
Daily Calories
2184 kcal
Based on your selections
BMI Analysis
Your BMI is 24.7 (Normal)
Calculated from height & weight
Normal weight range for your height: 59.9 kg - 80.7 kg
Water Intake
2.8 L / day
Based on your weight
Macro Split
Protein 15%
Carbs 60%
Fats 25%
Protein 80 g
Carbs 329 g
Fats 61 g

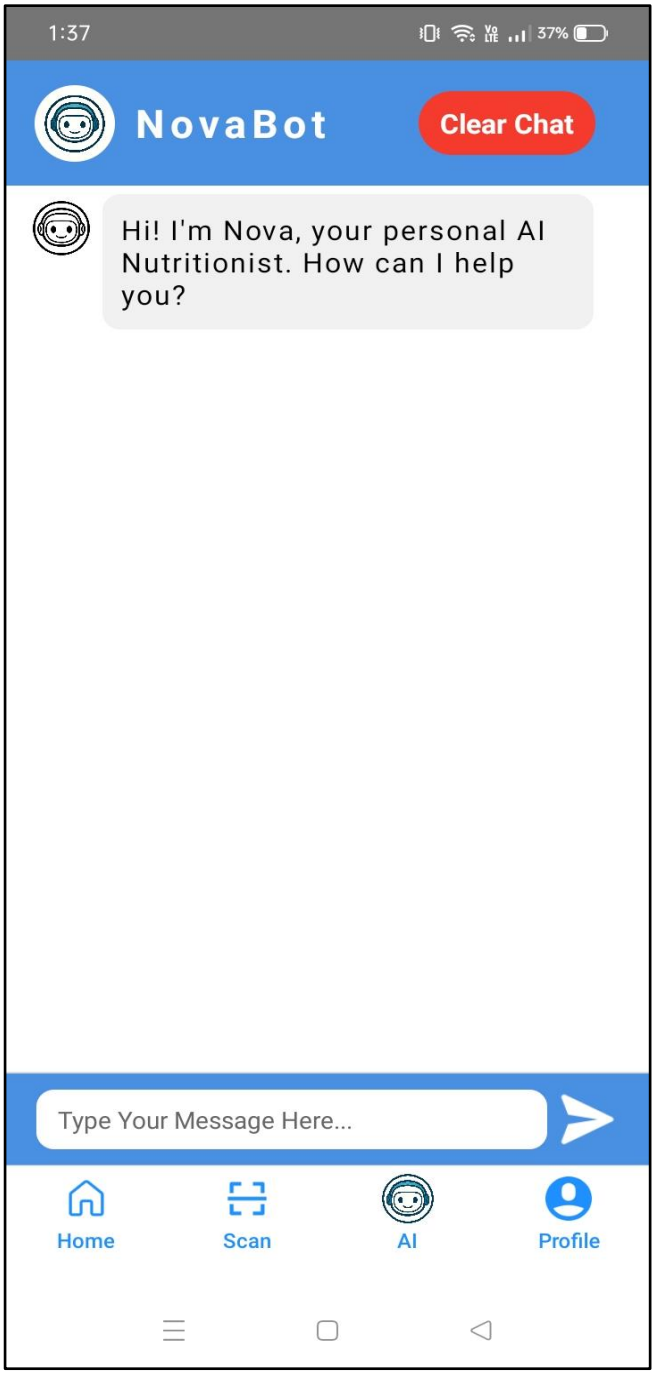

12:57
39%
Your BMI is 24.7 (Normal)
Calculated from height & weight
Normal weight range for your height: 59.9 kg - 80.7 kg
Water Intake
2.8 L / day
Based on your weight
Macro Split
Protein 15%
Carbs 60%
Fats 25%
Protein 80 g
Carbs 329 g
Fats 61 g
AI Diet Prompt
Copy & paste this into your AI tool to generate a diet plan
Create a Indian diet plan based on:
- Daily Calories: 2184 kcal
- Protein: 80 g
- Carbs: 329 g
- Fats: 61 g
- Water Intake: 2.8 L/day
Provide meal-wise breakdown (Breakfast, Lunch, Dinner).
Copy Prompt

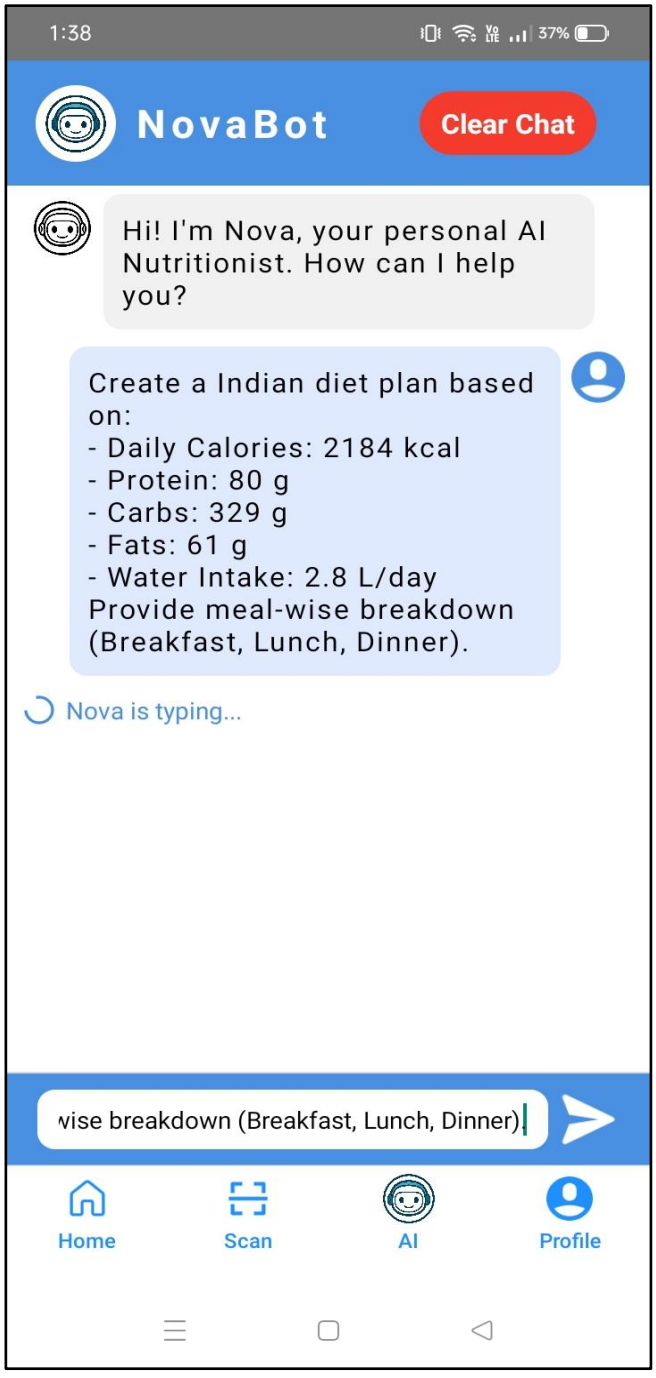
6. Health Tips

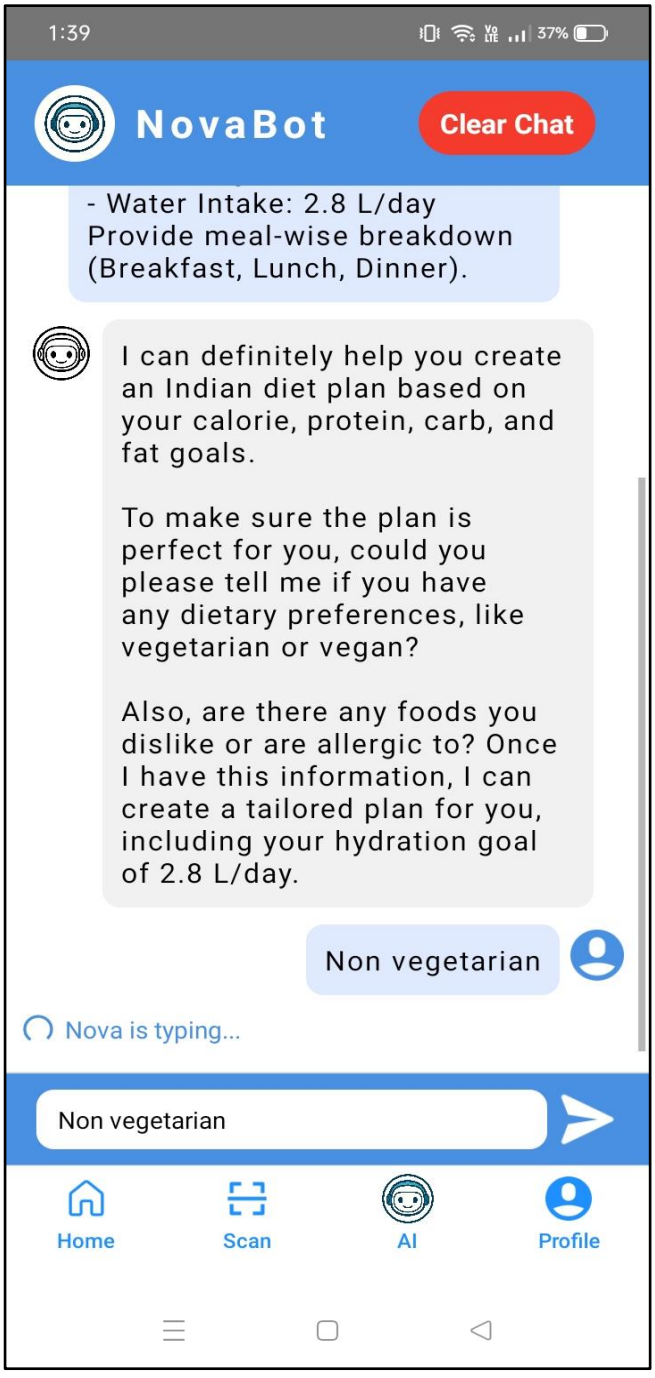


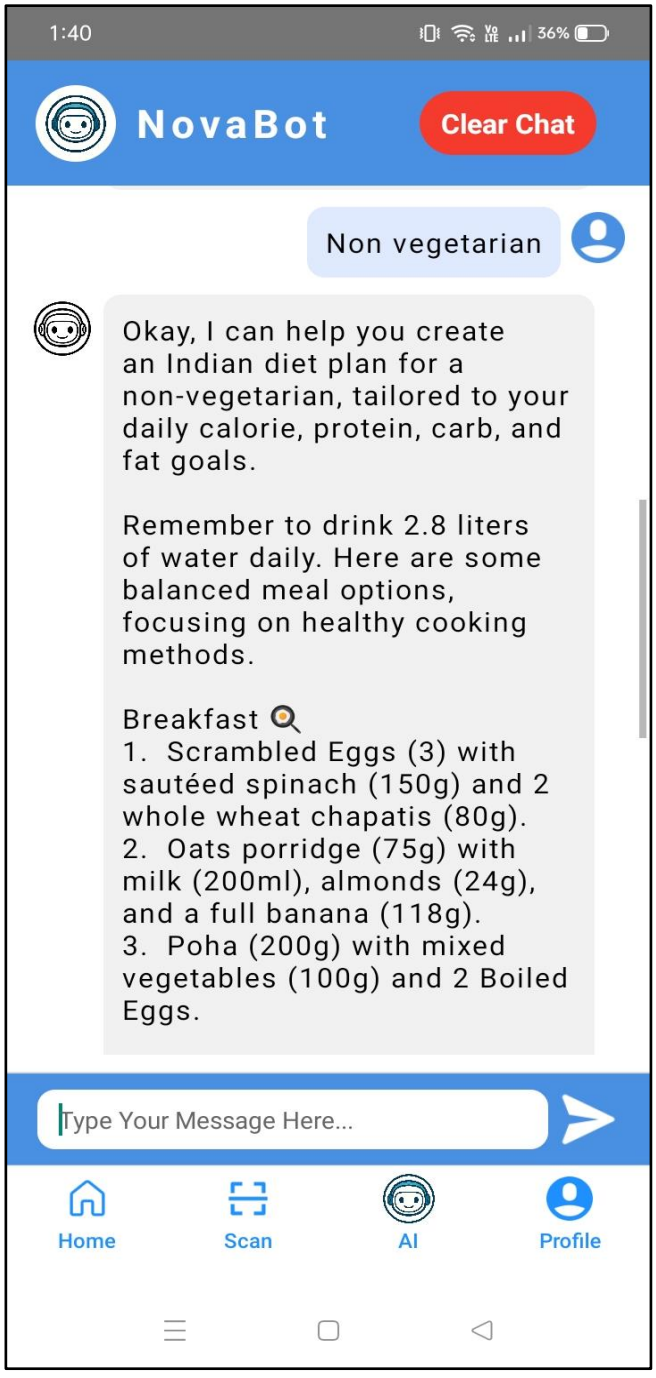


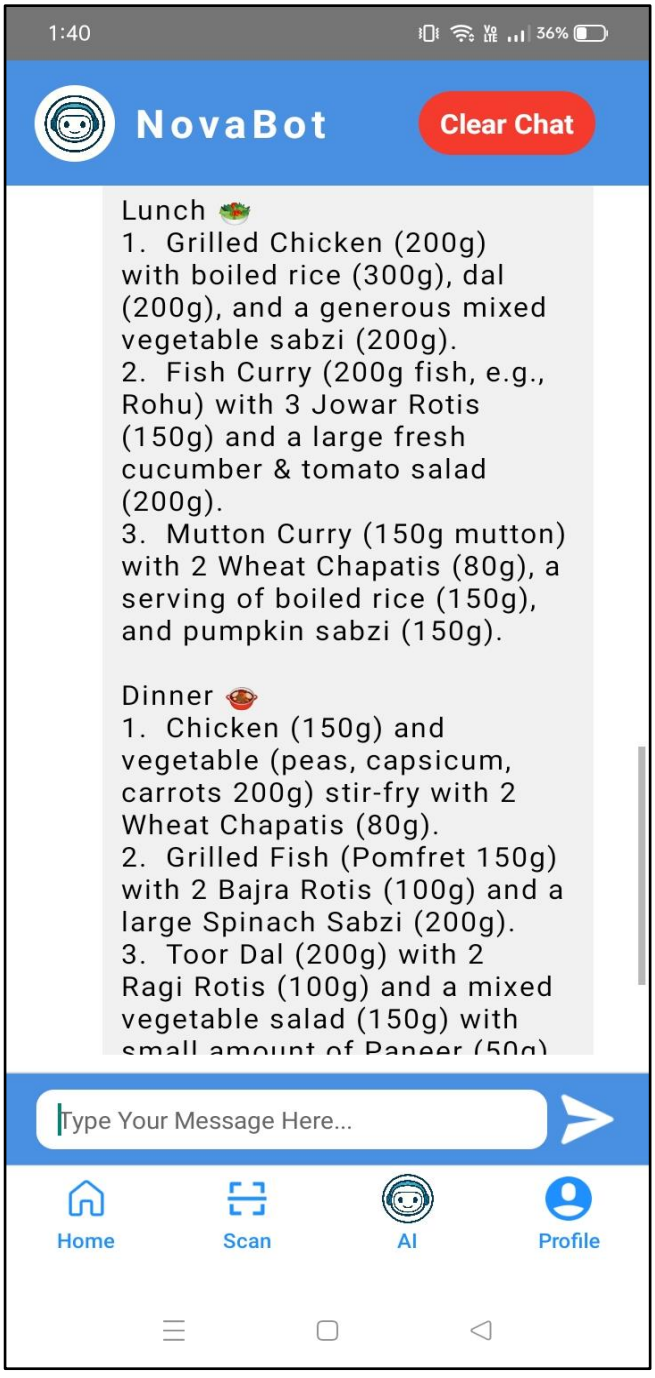
7. Diet Planner

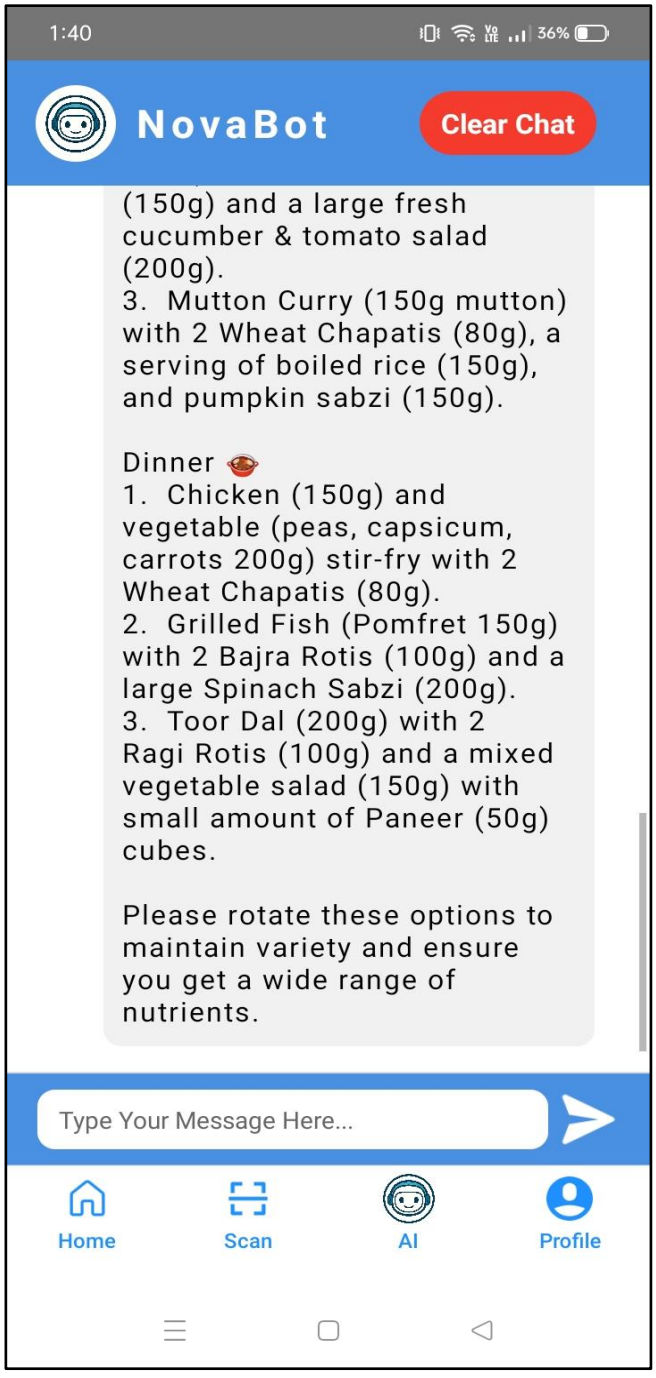




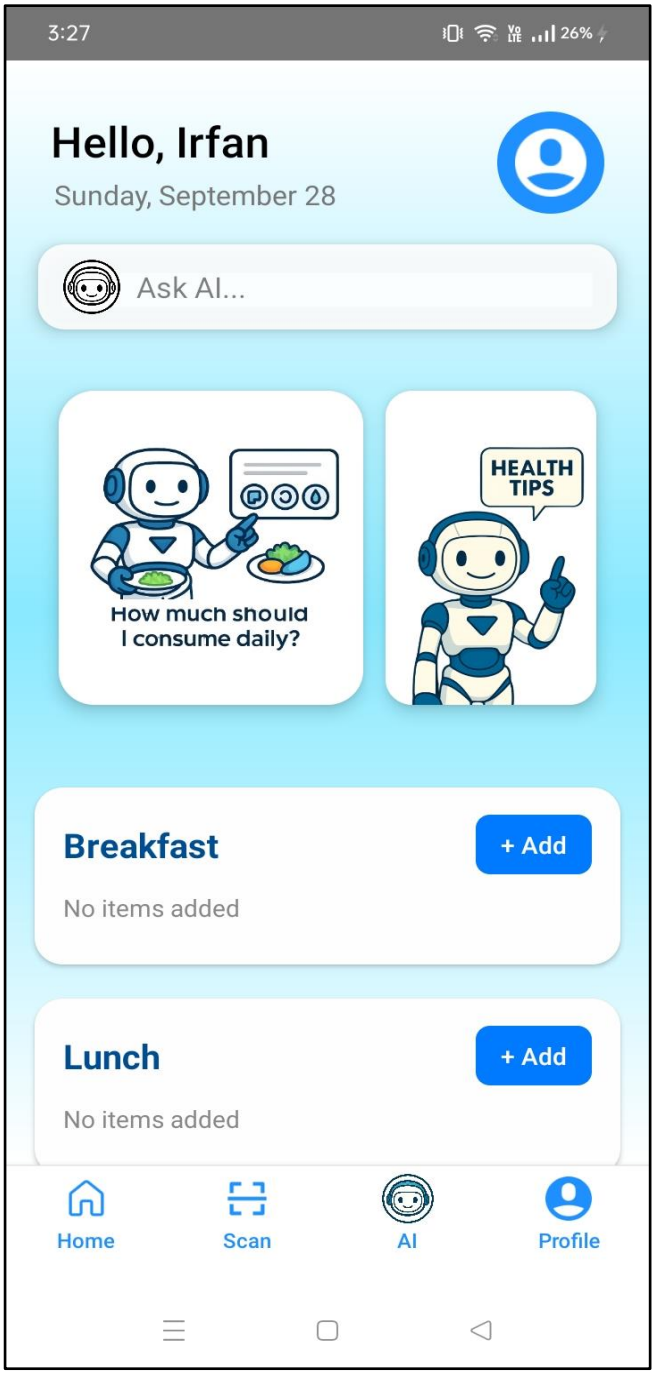

1:39
37%
NovaBot
Clear Chat
- Water Intake: 2.8 L/day
Provide meal-wise breakdown (Breakfast, Lunch, Dinner).
I can definitely help you create an Indian diet plan based on your calorie, protein, carb, and fat goals.
To make sure the plan is perfect for you, could you please tell me if you have any dietary preferences, like vegetarian or vegan?
Also, are there any foods you dislike or are allergic to? Once I have this information, I can create a tailored plan for you, including your hydration goal of 2.8 L/day.
Non vegetarian
Nova is typing...
Non vegetarian
Home
Scan
AI
Profile

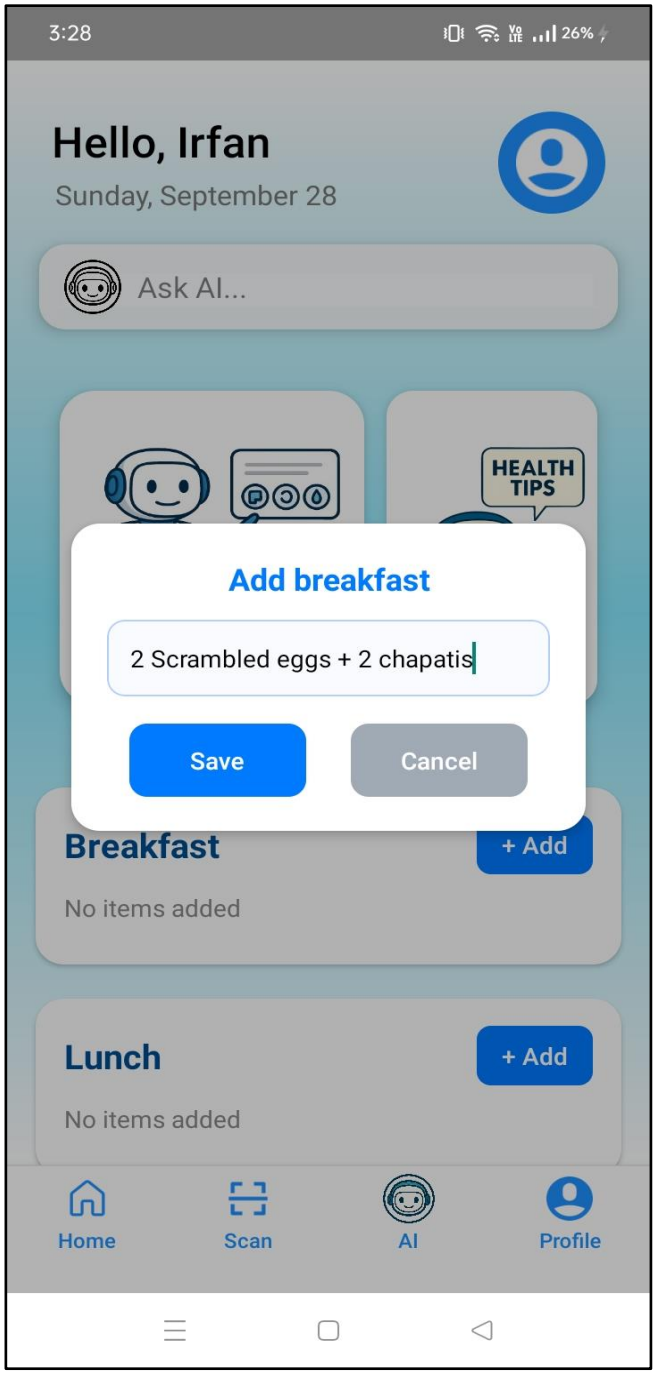

1:40
36%
NovaBot
Clear Chat
Non vegetarian
Okay, I can help you create an Indian diet plan for a non-vegetarian, tailored to your daily calorie, protein, carb, and fat goals.
Remember to drink 2.8 liters of water daily. Here are some balanced meal options, focusing on healthy cooking methods.
Breakfast
1. Scrambled Eggs (3) with sautéed spinach (150g) and 2 whole wheat chapatis (80g).
2. Oats porridge (75g) with milk (200ml), almonds (24g), and a full banana (118g).
3. Poha (200g) with mixed vegetables (100g) and 2 Boiled Eggs.
Type Your Message Here...
Home
Scan
AI
Profile

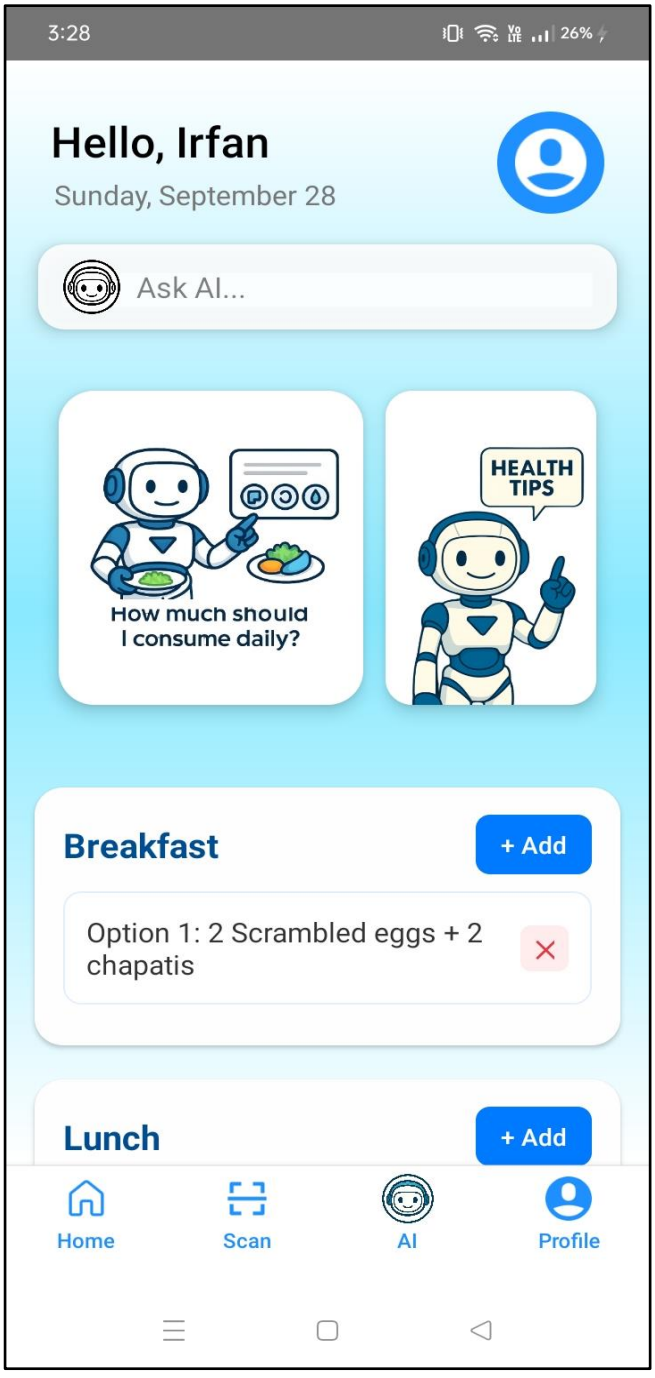

1:40
36%
NovaBot
Clear Chat
Lunch
1. Grilled Chicken (200g) with boiled rice (300g), dal (200g), and a generous mixed vegetable sabzi (200g).
2. Fish Curry (200g fish, e.g., Rohu) with 3 Jowar Rotis (150g) and a large fresh cucumber & tomato salad (200g).
3. Mutton Curry (150g mutton) with 2 Wheat Chapatis (80g), a serving of boiled rice (150g), and pumpkin sabzi (150g).
Dinner
1. Chicken (150g) and vegetable (peas, capsicum, carrots 200g) stir-fry with 2 Wheat Chapatis (80g).
2. Grilled Fish (Pomfret 150g) with 2 Bajra Rotis (100g) and a large Spinach Sabzi (200g).
3. Toor Dal (200g) with 2 Ragi Rotis (100g) and a mixed vegetable salad (150g) with
Type Your Message Here...
Home
Scan
AI
Profile

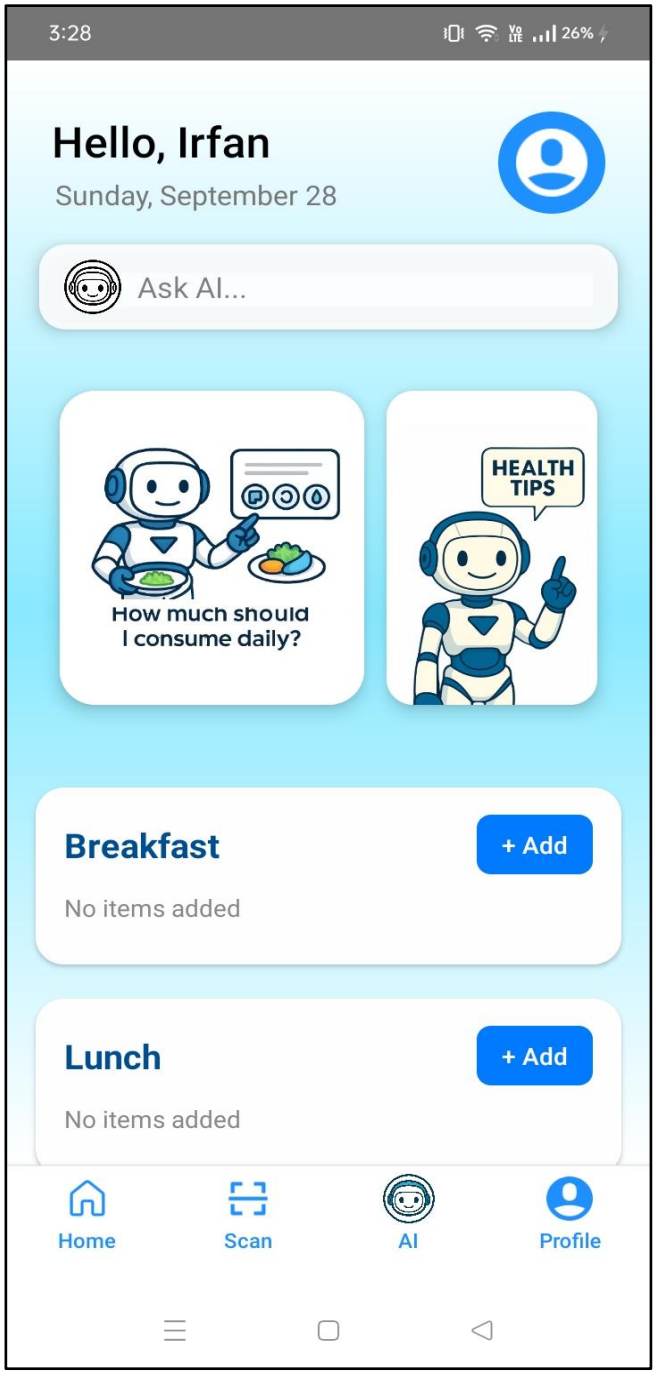

1:40
36%
NovaBot
Clear Chat
(150g) and a large fresh cucumber & tomato salad (200g).
3. Mutton Curry (150g mutton) with 2 Wheat Chapatis (80g), a serving of boiled rice (150g), and pumpkin sabzi (150g).
Dinner
1. Chicken (150g) and vegetable (peas, capsicum, carrots 200g) stir-fry with 2 Wheat Chapatis (80g).
2. Grilled Fish (Pomfret 150g) with 2 Bajra Rotis (100g) and a large Spinach Sabzi (200g).
3. Toor Dal (200g) with 2 Ragi Rotis (100g) and a mixed vegetable salad (150g) with small amount of Paneer (50g) cubes.
Please rotate these options to maintain variety and ensure you get a wide range of nutrients.
Type Your Message Here...
Home
Scan
AI
Profile

8. Add Meal / Delete Meal

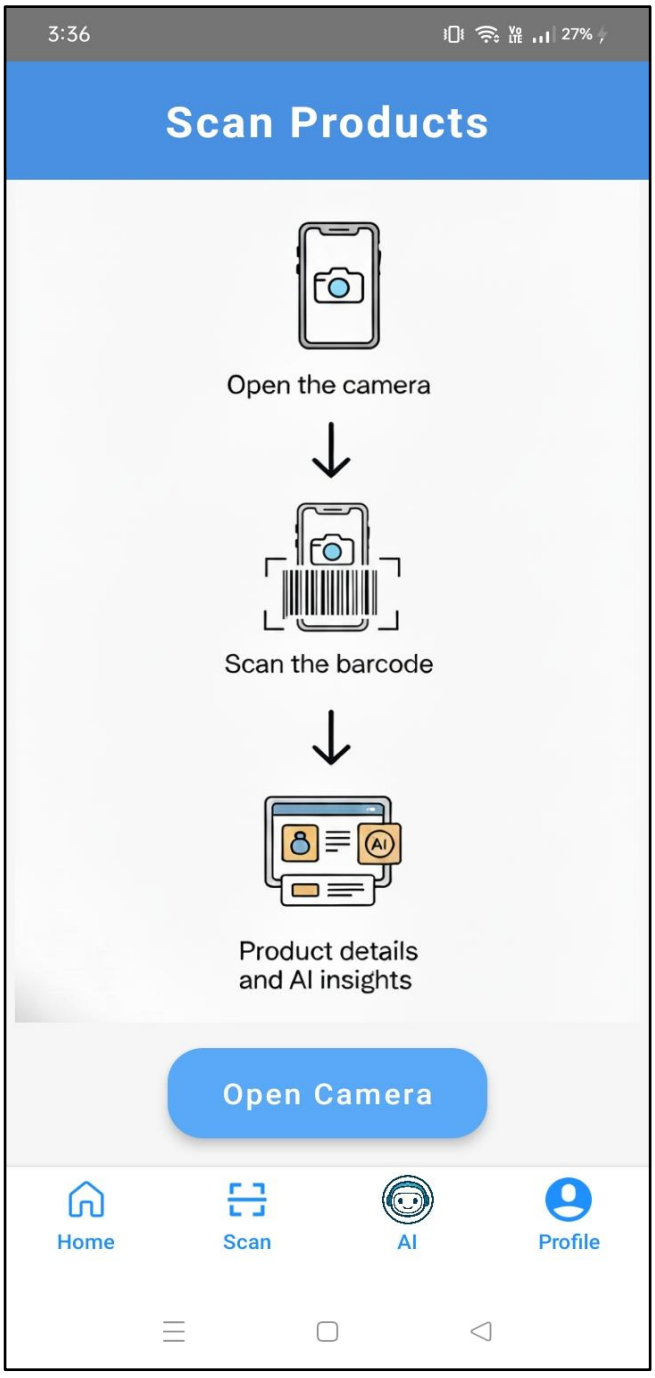


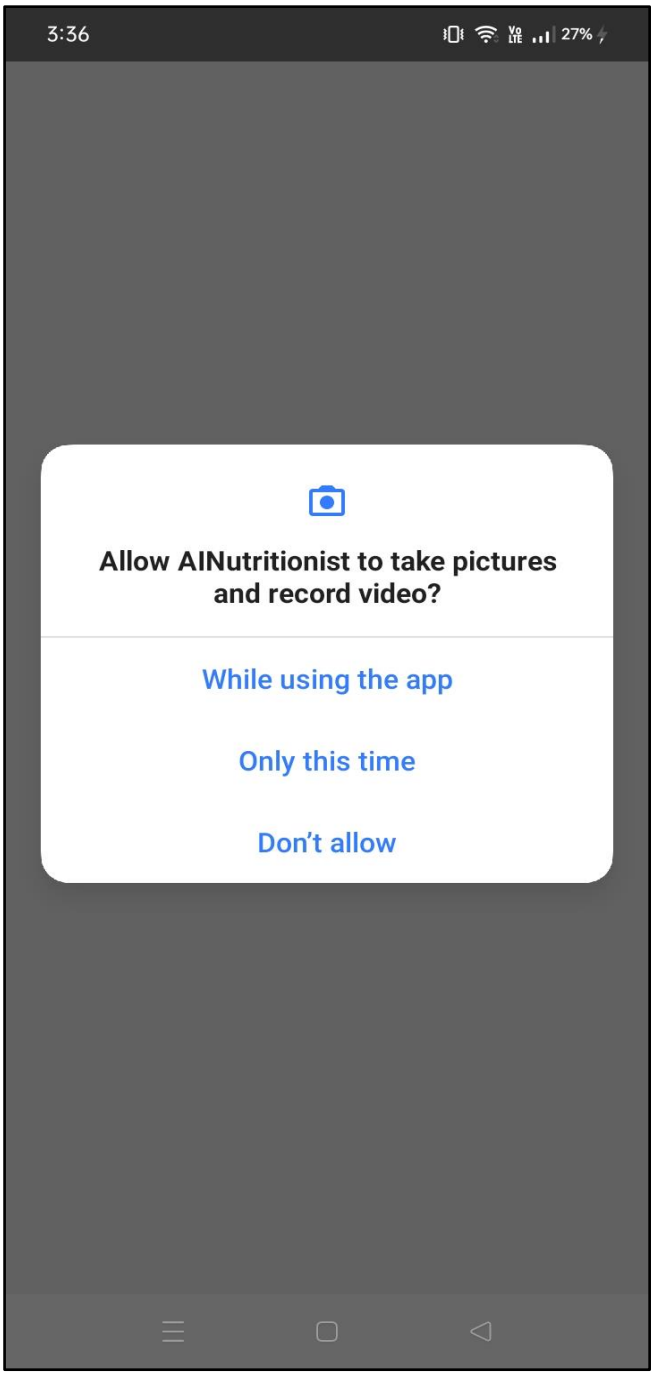


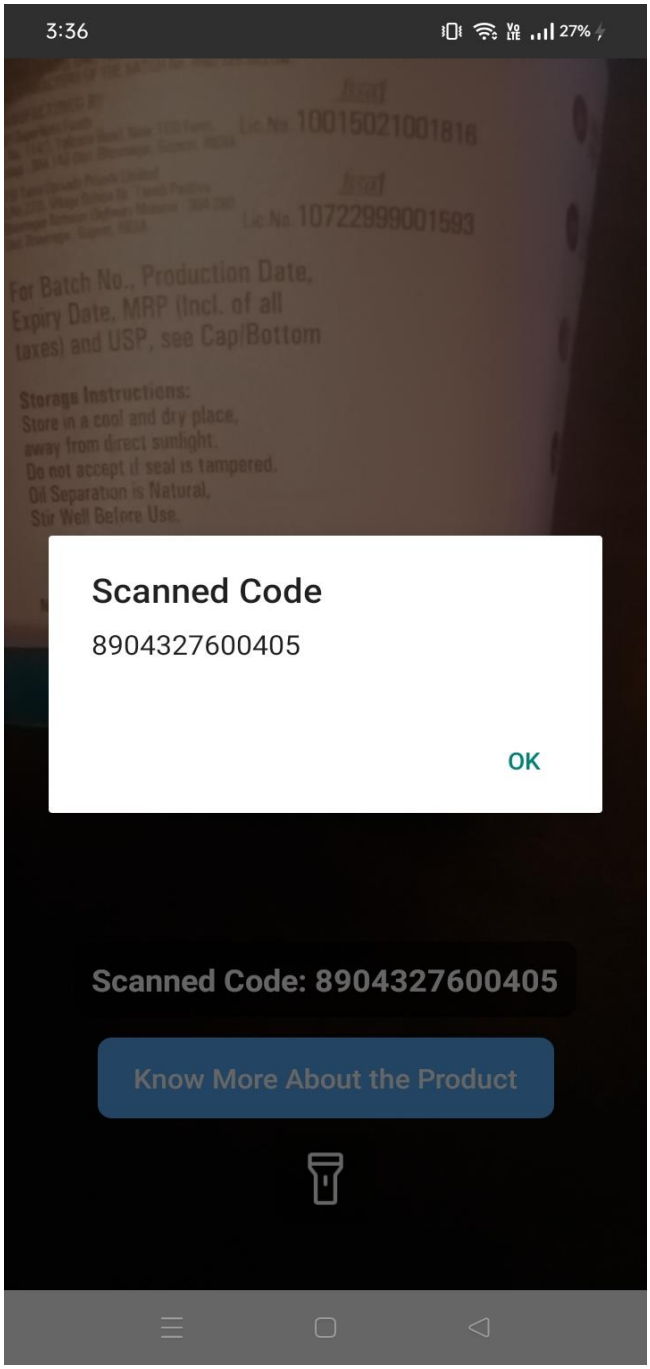

3:28
26%
Hello, Irfan
Sunday, September 28
Ask AI...
How much should I consume daily?
HEALTH TIPS
Breakfast
+ Add
Option 1: 2 Scrambled eggs + 2 chapatis
Lunch
+ Add
Home
Scan
AI
Profile

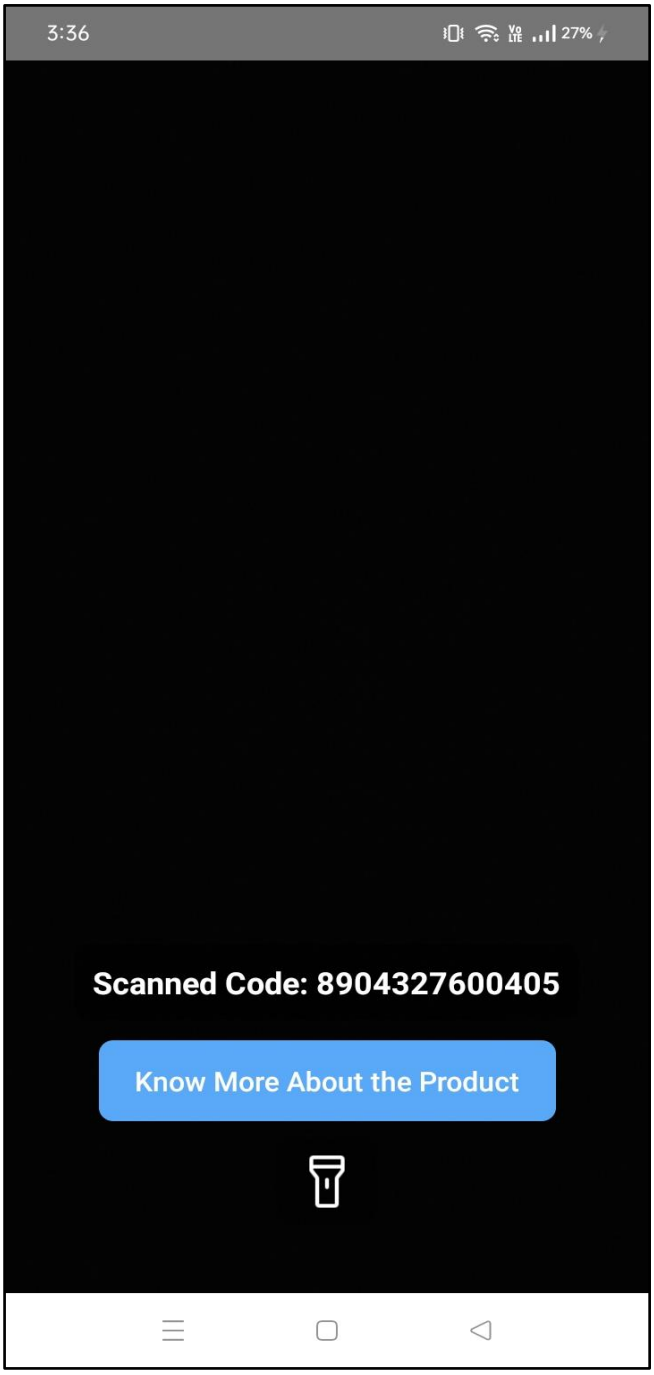

3:28
26%
Hello, Irfan
Sunday, September 28
Ask AI...
How much should I consume daily?
HEALTH TIPS
Breakfast
+ Add
No items added
Lunch
+ Add
No items added
Home
Scan
AI
Profile

9. Scanning Barcode + AI insights of Product

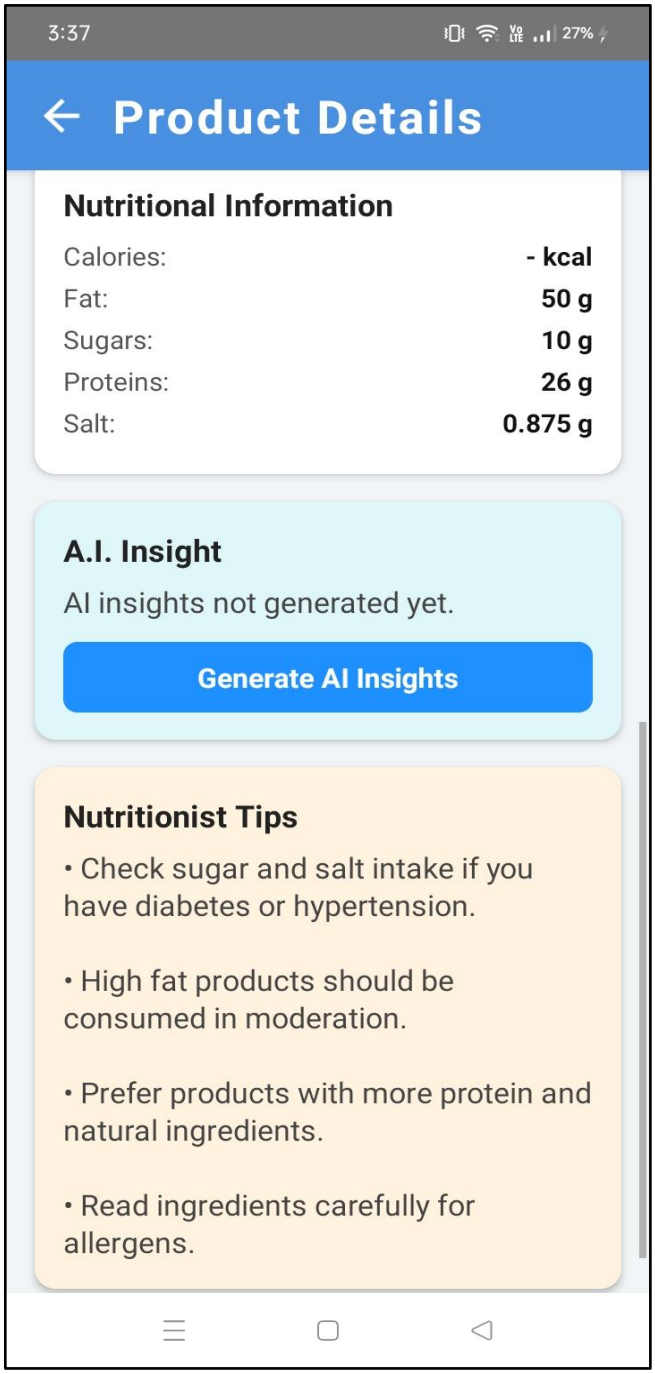


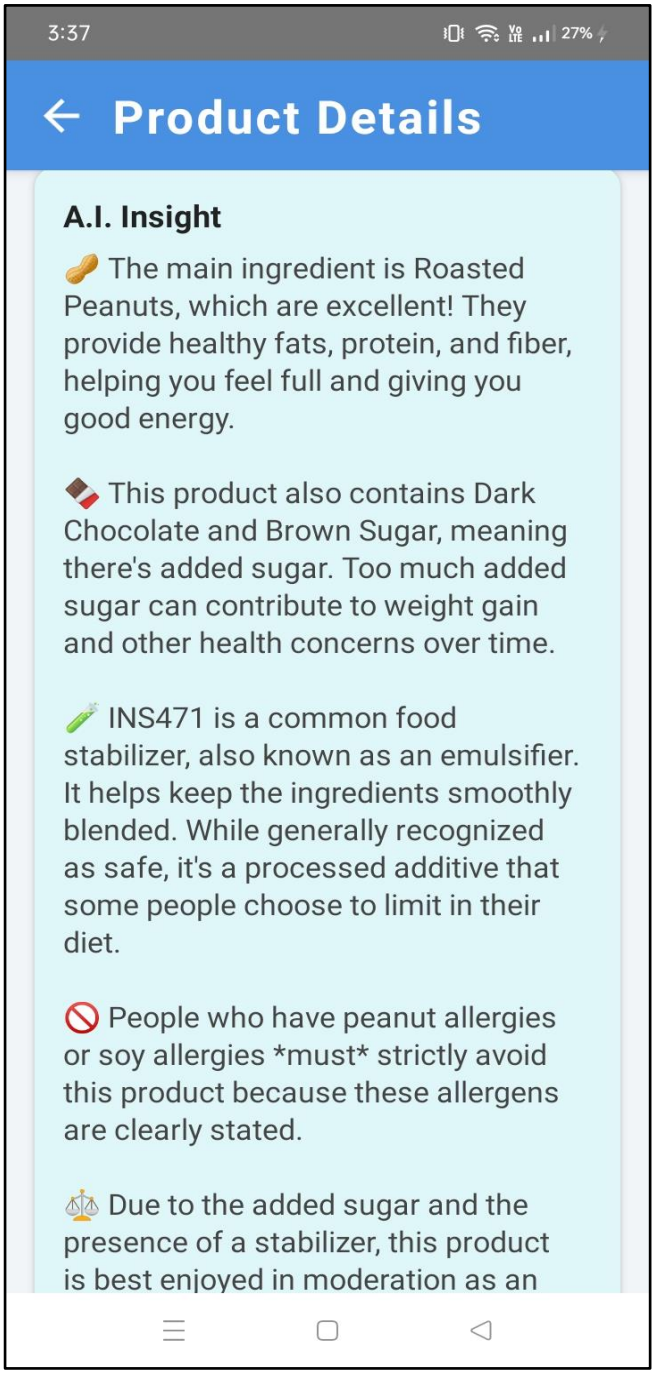


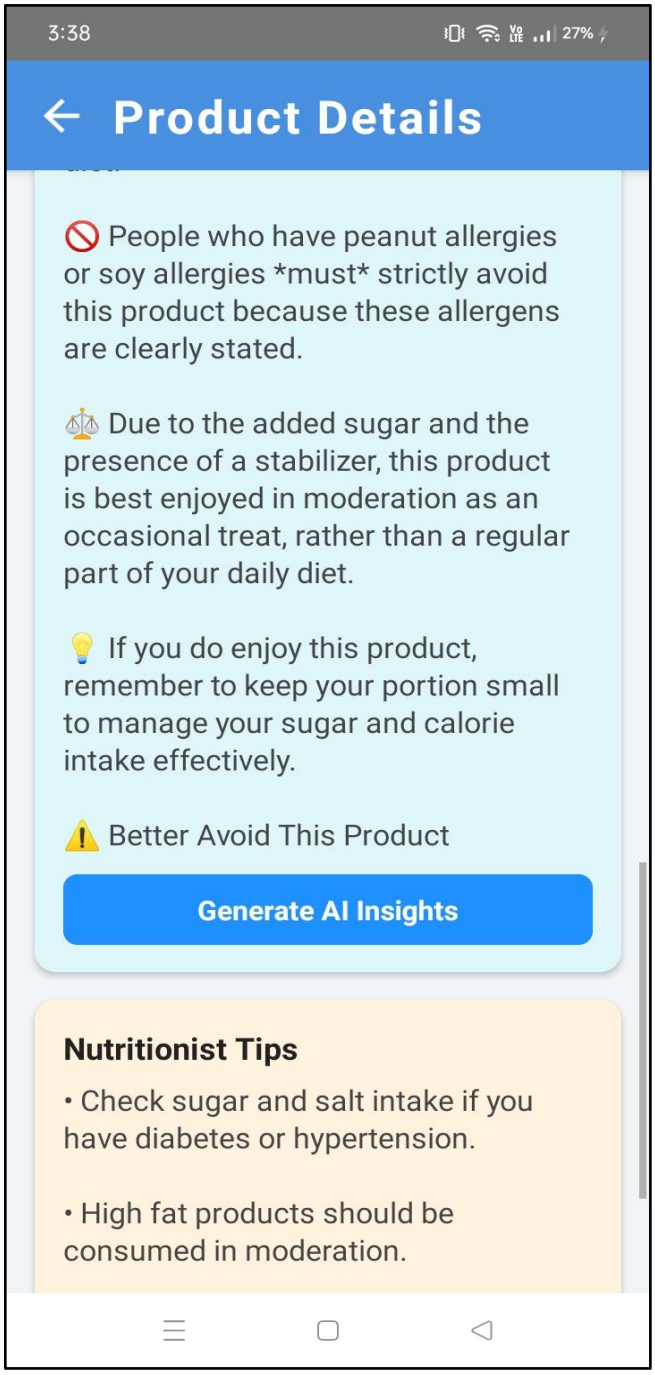

3:36
27%
Lic.No 10015021001816
Lic.No 10722999001593
For Batch No., Production Date,
Expiry Date, MRP (Incl. of all
taxes) and USP, see Cap/Bottom
Storage Instructions:
Store in a cool and dry place,
away from direct sunlight.
Do not accept if seal is tampered.
Oil Separation is Natural,
Stir Well Before Use.
Scanned Code
8904327600405
OK
Scanned Code: 8904327600405
Know More About the Product


3:36
27%
Scanned Code: 8904327600405
Know More About the Product


3:37
27%
Product Details
MYFITNESS
CHOCOLATE
PEANUT BUTTER
Smooth
brown sugar
pink himalayan salt
dark chocolate
Net Weight : 510g (18oz)
www.myfitness.in
Myfitness Chocolate Peanut Butter
Brand: Myfitness
Categories: Plant-based foods and beverages, Plant-based foods, Legumes and their products, Spreads, Nuts and their products, Plant-based spreads, Oilseed purees, Legume butters, Nut butters, Peanut butters
Ingredients
Roasted Peanuts, Dark Chocolate, Brown Sugar, Salt, Stabilizer (INS471). Allergen Info: Contains Peanuts And Soya Peanut Butter (Proprietary Food-4.2.2.5)


3:37
27%
Product Details
Nutritional Information
Calories: - kcal
Fat: 50 g
Sugars: 10 g
Proteins: 26 g
Salt: 0.875 g
A.I. Insight
AI insights not generated yet.
Generate AI Insights
Nutritionist Tips
• Check sugar and salt intake if you have diabetes or hypertension.
• High fat products should be consumed in moderation.
• Prefer products with more protein and natural ingredients.
• Read ingredients carefully for allergens.


3:37
27%
Product Details
A.I. Insight
🥜 The main ingredient is Roasted Peanuts, which are excellent! They provide healthy fats, protein, and fiber, helping you feel full and giving you good energy.
🍫 This product also contains Dark Chocolate and Brown Sugar, meaning there's added sugar. Too much added sugar can contribute to weight gain and other health concerns over time.
🧪 INS471 is a common food stabilizer, also known as an emulsifier. It helps keep the ingredients smoothly blended. While generally recognized as safe, it's a processed additive that some people choose to limit in their diet.
🚫 People who have peanut allergies or soy allergies *must* strictly avoid this product because these allergens are clearly stated.
⚖️ Due to the added sugar and the presence of a stabilizer, this product is best enjoyed in moderation as an


3:38
27%
Product Details
🚫 People who have peanut allergies or soy allergies *must* strictly avoid this product because these allergens are clearly stated.
⚖️ Due to the added sugar and the presence of a stabilizer, this product is best enjoyed in moderation as an occasional treat, rather than a regular part of your daily diet.
💡 If you do enjoy this product, remember to keep your portion small to manage your sugar and calorie intake effectively.
⚠️ Better Avoid This Product
Generate AI Insights
Nutritionist Tips
• Check sugar and salt intake if you have diabetes or hypertension.
• High fat products should be consumed in moderation.

CHAPTER 8. CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

The development of the AI Nutritionist application demonstrates how technology can play a vital role in improving health and lifestyle management. By integrating Artificial Intelligence with nutrition science, the system provides users with personalized dietary recommendations, AI-driven health diagnosis, barcode-based food scanning, and meal planning features. Unlike generic diet applications, this system adapts to each user's health profile, dietary preferences, and restrictions, thereby ensuring tailored solutions.

The project successfully addresses the problem of poor dietary awareness and lack of accessibility to personalized nutrition guidance. Through its interactive and user-friendly design, the application not only educates users about their food choices but also helps them maintain long-term health goals. Overall, the AI Nutritionist App bridges the gap between healthcare, technology, and everyday lifestyle management.

8.2 Advantages

1. Personalization – Provides diet plans and health tips tailored to individual profiles.
2. AI-Driven Insights – Uses machine learning for better accuracy in nutrition diagnosis.
3. User-Friendly Interface – Easy to navigate for both beginners and advanced users.
4. Barcode Scanning – Quickly identifies food contents and potential health risks.
5. Data Storage – Securely stores user profiles, diet plans, and scan history.

8.3 Limitations

1. Dependence on User Input – Accuracy of diagnosis relies on correct and complete profile information.
2. Limited Medical Scope – Provides nutritional guidance but cannot replace professional medical advice.
3. Database Dependency – Food analysis relies heavily on the food database, and sometimes ingredients may not be available or updated, leading to incomplete results when scanning food packets.

4. No Offline Support (Current Version) – Requires internet access for AI diagnosis and scanning.
5. Initial Language Limitation – May not support all regional languages at present.

8.4 Future Scope

While the application provides a strong foundation, there are several opportunities for enhancement and expansion in the future:

1. Integration with Wearable Devices – Syncing with smartwatches or fitness bands to track real-time health parameters such as heart rate, blood pressure, and calories burned.
2. Voice Assistant Support – Adding speech-based interactions for users with low literacy or accessibility challenges.
3. AI-Powered Food Image Recognition – Allowing users to take pictures of food items instead of only scanning barcodes.
4. Multilingual and Regional Language Support – Making the application accessible to a wider population.
5. Integration with Healthcare Professionals – Providing channels for dietitians and doctors to review and approve diet plans.
6. Nutritional Trend Analysis – Offering insights based on cumulative food intake to detect long-term risks such as obesity or nutrient deficiencies.
7. Gamification Features – Introducing reward points, streaks, or challenges to motivate users to maintain healthy eating habits.
8. Cloud-Based Data Sharing – Allowing users to securely store and share their nutritional data across devices.[9]

CHAPTER 9. REFERENCES

1. https://www.thelancet.com/article/S01406736%2819%2930041-8/fulltext - Problem Statement
2. https://www.ndtv.com/health/boost-your-nutritional-intake-by-making-these-small-changes-to-your-diet-8470927 - Aim
3. https://www.mdpi.com/2076-3417/15/9/4911 - Objective
4. https://www.geeksforgeeks.org/software-engineering/software-engineering-spiral-model/ - Spiral Model
5. https://www.investopedia.com/terms/f/feasibility-study.asp - Feasibility Study
6. https://reactnative.dev/ - React Native
7. https://www.typescriptlang.org/docs/ - Typescript
8. https://www.geeksforgeeks.org/system-design/system-design-tutorial - System Design
9. https://www.nature.com/articles/s41598-024-65438-x - Future scope