BANGALORE UNIVERSITY

Submitted in the fulfillment of the requirements of VI Semester

BACHELOR OF COMPUTER APPLICATION

A Projects Report Named

"PERSONALIZED NUTRITION RECOMMENDATION SYSTEM"

By

CHANDAN S (U03MX22S0054)

VARUN GOWDA B R (U03MX22S0047)

Under the guidance of

Mrs. BHUVANA C T

Asst. Professor, Department of Computer Science

Triveni Memorial Educational Trust ®
TRIVENI INSTITUTE OF COMMERCE AND MANAGEMENT
Affiliated to Bangalore University and Recognized by Government of Karnataka
No. 28/29, Hessarghatta Main Road, T. Dasarahalli, Mallasandra, Bangalore - 560057

ACKNOWLEDGEMENT

The completion of the project work on **"PERSONALIZED NUTRITION RECOMMENDATION SYSTEM"** brings with a sense of satisfaction, but it is never complete without thanking the person responsible for its successful completion.

I extend my deep sense of sincere gratitude to **Dr. HARITHA MUNIRAJU**, PRINCIPAL, TRIVENI INSTITUTE OF COMMERCE AND MANAGEMENT, Bagalgunte, for providing me an opportunity to pursue my bachelor degree.

I express my deep heartfelt sincere gratitude to **Mrs. TL PALLAVI**, Head of the Department of Computer Science, Triveni Institute of Commerce and Management, Bagalgunte, for her valuable suggestion and support.

I consider my privilege to express my gratitude to the guide of the project **Mrs. BHUVANA CT** for her support and help whenever needed.

Finally, I would like to thank all the faculty members of **Department of Computer Science**, **Triveni Institute of Commerce and Management**, for their support.

CHANDAN S (U03MX22S0054)

VARUN GOWDA B R (U03MX22S0047)

VI Sem BCA

STUDENT DECLARATION

I am writing to formally declare that I have utilized the guidance of the professors in the process of completing my project. As lecturers of TICM they have provided me with valuable insights and information that have significantly contributed to the completion of my project.

Through this website, I was able to access a wealth of knowledge and expertise on various topics related to my project. Professors have also been instrumental in providing me with useful suggestions and ideas that have helped me to refine my project and achieve the desired outcomes.

I would like to express my gratitude to the professors for their invaluable contribution to the successful completion of my project. I believe that their guidance has the potential to help many students like me in their academic pursuits, and I would highly recommend others to seek their assistance.

I hereby declare that all information contained within my project report is accurate and truthful to the best of my knowledge.

Thank you for your time and consideration.

Date:

Place:

Sincerely,

CHANDAN S.

VARUN GOWDA B R.

ABSTRACT

The Personalized Nutrition Recommendation System is a web-based intelligent application designed to offer individualized dietary advice based on a user's unique health parameters, lifestyle, and dietary needs. In an age where generic dietary plans often fail to meet individual requirements, this system aims to bridge the gap between nutrition science and personal health by leveraging modern technologies such as artificial intelligence, machine learning, and computer vision.

This project combines Flask as the primary web framework with HTML, CSS, and JavaScript on the frontend to create a responsive and user-friendly interface. At its core, the system incorporates user data inputs including age, gender, weight, height, activity level, pre-existing health conditions (e.g., diabetes, hypertension), and dietary preferences (e.g., vegetarian, gluten-free). Based on this data, the backend processes these inputs to calculate personalized nutritional needs using scientifically validated formulas such as the Basal Metabolic Rate (BMR), Total Daily Energy Expenditure (TDEE), and macronutrient ratios.

A key innovation of the system is its image recognition module, which allows users to upload pictures of their meals. Through integration with Google Gemini API (or an equivalent AI-powered image analysis tool), the system identifies food items, estimates portion sizes, and calculates nutritional values. This enables real-time tracking and assessment of daily food intake without requiring manual entry, significantly improving user convenience and accuracy.

The AI-driven recommendation engine then generates a custom meal plan and actionable tips based on identified nutritional gaps, user preferences, and goals (e.g., weight loss, muscle gain, managing diabetes). The recommendations are dynamically adapted over time as more user data is gathered, allowing for continuous refinement and personalization.

To ensure security and performance, the system uses secure user authentication and stores sensitive health data using encryption best practices. The backend supports RESTful APIs for modularity and future scalability, such as integration with fitness trackers, wearable devices, or healthcare provider platforms.

This project addresses key challenges in digital health and wellness by providing a personalized, data-driven solution that empowers users to make informed dietary choices. It is designed with extensibility in mind, laying the groundwork for future enhancements such as chatbot support, multilingual interfaces, and integration with electronic health records (EHRs). Ultimately, this system exemplifies how modern web development and AI technologies can converge to deliver impactful solutions in preventive healthcare and nutrition science.

Table of Contents

1. Introduction

1.1 Overview of the Project

The **Personalized Nutrition Recommendation System** is a comprehensive, AI-powered web application developed to offer individualized dietary guidance based on user-specific health metrics, lifestyle factors, and personal preferences. With the increasing prevalence of non-communicable diseases such as obesity, diabetes, and cardiovascular conditions, there is an urgent demand for tools that enable people to make informed and healthy nutritional choices.

This system bridges the gap between general dietary advice and personal health requirements by using data-driven insights and intelligent analysis. It collects user data, such as age, weight, height, gender, physical activity levels, dietary restrictions, and health conditions, to generate a tailored nutrition profile. Furthermore, it leverages **Google Gemini API** to identify food items from user-uploaded images and estimate their nutritional values, making food tracking seamless and automatic.

Built with a modular architecture, the system comprises several integrated components including a **Flask-based backend**, **responsive frontend interfaces**, **database management system**, and **AI-driven recommendation logic**. These components work in harmony to offer real-time feedback, dietary tracking, and nutrition suggestions, empowering users to manage their health proactively and efficiently.

1.2 Problem Statement

Modern consumers are bombarded with nutritional information that is either generic or contradictory. Traditional diet plans often overlook individual differences in metabolism, health conditions, and lifestyle choices. As a result:

- Many users fail to adhere to standard nutrition programs.
- Individuals with chronic illnesses or specific health conditions (e.g., diabetes, hypertension) struggle to find meal plans that fit their unique dietary needs.
- Manual food tracking applications require significant effort, making them unsustainable in the long term.
- Accurate food logging is difficult for the average user who lacks knowledge about ingredients, portion sizes, and nutrient composition.

In short, there is a **critical need** for a personalized, automated, and intelligent nutrition support system that simplifies the process of meal tracking and offers evidence-based dietary recommendations tailored to individual users.

1.3 Objectives of the System

The system is designed with the following specific objectives:

- **Personalization:** Generate custom diet plans based on user demographics, physical health parameters, activity levels, and goals (e.g., weight loss, muscle gain, disease management).
- **Automation:** Enable users to upload food images and automatically detect food items and estimate nutritional content using machine learning APIs.
- **Recommendation:** Provide intelligent dietary suggestions based on real-time analysis of dietary intake versus daily nutritional needs.
- **Tracking & Visualization:** Allow users to monitor their progress through dashboards showing calorie intake, macronutrient distribution, and comparison to dietary targets.
- **Security:** Ensure secure handling of user data through authentication, authorization, and data encryption.
- **Scalability:** Design the system with modular components for easy integration of future enhancements like chatbot support, wearable device syncing, or multilingual accessibility.

1.4 Scope of the Project

The current version of the Personalized Nutrition Recommendation System is designed as a **web application** that can be accessed via modern browsers on desktop and mobile devices. The major modules and features included are:

- **User Registration and Login**: Secure authentication and session management.
- **Profile Setup**: User inputs physical, medical, and lifestyle details.
- **Food Image Upload**: Users can take a photo of their meal using a device camera or upload from the gallery.
- **Food Recognition and Analysis**: Using Google Gemini API, the system recognizes food items, estimates portion size, and calculates calories, fats, proteins, and carbohydrates.
- **Diet Recommendation Engine**: Compares current intake with daily goals and suggests meal adjustments or improvements.
- **Interactive Dashboard**: Displays key health insights like total calories consumed, recommended intake, macro breakdown, and food history.
- **Data Storage**: Uses a structured database to store user details, food logs, and recommendation history.

The system is suitable for individual users and provides a foundation that can be extended to serve health professionals, nutritionists, or fitness coaches in future versions.

1.5 Significance and Use Cases

The significance of this project lies in its potential to contribute to **preventive healthcare**, **health awareness**, and **behavioral change** by offering a reliable and accessible tool for diet management. Key use cases include:

- **General Wellness**: Helps users who want to maintain a healthy lifestyle and balanced diet.
- **Disease Management**: Supports users with medical conditions like diabetes, obesity, or PCOS with tailored diet plans.
- **Fitness Enthusiasts**: Assists athletes or gym-goers in optimizing their food intake based on workout intensity and goals.
- **Remote Diet Coaching**: Lays the groundwork for healthcare providers to monitor patient diets and offer remote consultations.
- **Busy Professionals**: Reduces the effort of meal logging, making it feasible for people with limited time.

This system enables users to take control of their health through informed eating habits, thus contributing to long-term wellness and disease prevention.

1.6 Methodology Adopted

To ensure an efficient and reliable development process, the system was built using the **Agile Software Development Life Cycle (SDLC)**. This allowed for continuous feedback, iterative development, and flexible updates. The key stages followed include:

a. Requirement Gathering

- Identification of user needs through questionnaires and market analysis.
- Definition of functional and non-functional requirements.

b. System Design

- Architectural planning (3-tier model: presentation, logic, and data layers).
- Design of data models, ER diagrams, and system workflows.
- Creation of UI wireframes for consistent and intuitive design.

c. Technology Selection

- **Backend**: Python with Flask framework for handling server logic and RESTful APIs.
- **Frontend**: HTML, CSS, JavaScript (with frameworks like Bootstrap for responsive design).
- **Image Analysis**: Integration with **Google Gemini API** for real-time food recognition and nutritional analysis.
- **Database**: SQLite or PostgreSQL for structured storage of user data and logs.

d. **Implementation**

- Modular development of each component (e.g., user authentication, image recognition, recommendation engine).
- API development for data exchange between frontend and backend.

e. **Testing**

- Unit testing for each module.
- Integration testing for cross-module functionality.
- Manual and automated testing for user interaction and system behavior.

f. **Deployment**

- Hosted locally or on a cloud platform (e.g., Heroku, Render).
- Performance evaluation under different user scenarios.

g. Documentation

- Comprehensive reporting and technical documentation of system architecture, codebase, and APIs.

This structured approach ensured that the system is robust, user-friendly, and ready for real-world usage while remaining scalable for future enhancements.

2. Literature Review

A literature review is essential to understand the existing landscape of technological solutions in the domain of personalized nutrition, their architectures, feature sets, limitations, and opportunities for innovation. This section explores current systems and technologies, compares them based on key parameters, and identifies the unmet needs that this project aims to address.

2.1 Existing Systems and Limitations

Several mobile and web applications exist in the health-tech space that offer dietary tracking and recommendations. Some notable systems include:

a. MyFitnessPal

A popular calorie counter and nutrition tracker that allows users to log meals, scan barcodes, and track macronutrients. It features a large food database and integrates with fitness apps.

Limitations:

- Manual data entry is time-consuming and error-prone.

- Generic calorie goals with limited personalization based on health conditions.
- Lacks AI-driven food recognition.

b. HealthifyMe

An Indian-based health and fitness platform that provides calorie tracking, workout plans, and live dietitian support. It also includes a smart AI assistant, "Ria," for basic queries.

Limitations:

- Food logging still requires manual input.
- AI assistant responses are basic and limited in reasoning capabilities.
- Premium features are locked behind a paywall.

c. Yazio

Focuses on meal planning and fasting with tailored meal recommendations based on user goals. It includes nutritional value tracking and some level of customization.

Limitations:

- Limited support for specific medical conditions.
- No food image recognition capabilities.
- Recommendation logic is fixed and rule-based.

d. CalorieMama

This app supports food logging via image recognition, using a trained AI model to detect food items from photos.

Limitations:

- Food detection accuracy is not consistent for diverse or mixed dishes.
- Does not offer deeply personalized dietary suggestions.
- No health profile integration like medical history or activity level.

2.2 Comparative Analysis

Feature / System	MyFitnessPal	HealthifyMe	Yazio	CalorieMama	Proposed System
Food Logging	Manual	Manual	Manual	AI Image	AI Image + Manual

Feature / System	MyFitnessPal	HealthifyMe	Yazio	CalorieMama	Proposed System
Personalization	Basic (TDEE)	Moderate	Goal-based	Minimal	High (Health data, goals, restrictions)
Medical Conditions	No	Limited	No	No	Yes
Recommendation Engine	Static	Semi-dynamic	Static	None	Dynamic AI-Based
AI/ML Integration	No	Basic NLP	No	Image model	Image + Text AI (Gemini API)
Ease of Use	Medium	High	Medium	High	High
Free Features	Basic Logging	Limited	Partial	Partial	Full Core Access
Extensibility	Medium	Low	Low	Low	High (Modular)

This comparison highlights that while existing systems provide some of the core functionalities like tracking and basic recommendations, they fall short in delivering comprehensive personalization, especially through AI and real-time analysis of user data and images.

2.3 Key Findings and Gaps

After reviewing current solutions, the following key insights and gaps have been identified:

Key Findings:

- Most platforms focus heavily on calorie tracking and goal setting, with minimal use of advanced AI for personalization.
- Manual data entry remains a barrier to user adoption and retention.
- Some apps have begun to explore food image recognition, but the technology is not fully integrated with recommendation engines.
- Very few systems consider clinical parameters such as chronic illnesses, dietary restrictions, or biochemical needs in meal planning.

Identified Gaps:

- **Lack of Deep Personalization:** Existing tools do not use a user's complete health profile (e.g., BMI, diseases, allergies) to tailor recommendations.

- **Low Automation:** Reliance on manual inputs leads to poor user engagement and inaccurate data tracking.
- **Limited AI Utilization:** AI is mostly used for conversational agents or limited image detection; no integration of multi-modal AI for comprehensive analysis.
- **No Adaptive Feedback Loops:** Current systems do not update dietary plans dynamically based on user progress or changing health metrics.
- **Poor Integration of Health Conditions:** Little to no integration with disease-specific nutritional advice (e.g., low-sodium for hypertension).

Conclusion of Literature Review

The existing nutrition platforms serve basic tracking and meal planning purposes but fail to deliver a truly **intelligent, personalized, and automated** user experience. These gaps present a significant opportunity for innovation. The **Personalized Nutrition Recommendation System** addresses these shortcomings by:

- Using health profile-driven dietary customization.
- Integrating AI-based image recognition for food detection.
- Generating dynamic, context-aware recommendations.
- Supporting users with chronic health conditions.
- Providing an extensible platform for future enhancements (wearables, chatbot support, multilingual access).

3. System Analysis

System analysis is a crucial phase in the software development life cycle that aims to define the system's structure, behavior, and interactions. It involves identifying project requirements, determining the technical and operational feasibility, and evaluating potential risks and strengths of the system. For the **Personalized Nutrition Recommendation System**, this analysis establishes the foundation upon which the architecture, implementation, and validation processes are built.

3.1 Requirements Analysis

This section identifies and categorizes both functional and non-functional requirements essential for the successful operation of the system.

3.1.1 Functional Requirements

These specify the core functionalities the system must provide to meet user expectations:

Requirement ID	Functionality
FR1	User registration and secure login/logout
FR2	User profile management (age, gender, height, weight, health conditions, activity level)
FR3	Food image upload using web interface
FR4	Food image analysis using Google Gemini API
FR5	Retrieval of nutritional information based on recognized food
FR6	Personalized recommendation engine based on health profile and food intake
FR7	Real-time calorie and macronutrient tracking
FR8	Dashboard with data visualizations (charts, summaries, progress reports)
FR9	Health goal setting (e.g., weight loss, muscle gain, maintenance)
FR10	Daily/weekly meal suggestions and alerts
FR11	Data storage and retrieval (SQLite/PostgreSQL database)

3.1.2 Non-Functional Requirements

These define system qualities and constraints:

- **Usability:** The user interface should be intuitive and mobile-responsive.
- **Performance:** The system should analyze images and generate results within 5 seconds.
- **Scalability:** Should support up to thousands of users with potential for cloud deployment.
- **Security:** Implements encrypted authentication and secure session handling.
- **Maintainability:** Modular codebase and documentation for easy updates.
- **Reliability:** Robust error handling to prevent crashes from invalid inputs or API failures.
- **Interoperability:** Compatible with RESTful API standards for potential third-party integrations.

3.2 Feasibility Study

The feasibility study evaluates whether the project is viable in technical, operational, and financial terms.

3.2.1 Technical Feasibility

- **Backend Technologies:** Python and Flask provide scalability and rapid development.

- **Frontend Technologies:** HTML, CSS, Bootstrap, and JavaScript ensure a responsive UI.
- **Database:** SQLite or PostgreSQL is sufficient for data storage with future scalability.
- **Image Recognition:** Google Gemini API provides reliable and pre-trained models for food identification.
- **Hosting:** Compatible with cloud-based platforms (e.g., Render, Heroku) for deployment.

✅ **Conclusion:** Technically feasible with readily available tools and developer expertise.

3.2.2 Operational Feasibility

- End users (general population) will find the system easy to navigate.
- Addresses critical health concerns, improving user engagement and retention.
- Admin features can be added later for professional dieticians or clinics.

✅ **Conclusion:** Operationally feasible with significant user value and social impact.

3.2.3 Economic Feasibility

- **Development Cost:** Low, using open-source tools and APIs with free usage tiers.
- **Hosting Cost:** Cloud platforms offer free/affordable hosting for MVP.
- **ROI Potential:** High, if commercialized as a subscription-based or freemium model.

✅ **Conclusion:** Economically feasible with minimal startup cost and high growth potential.

3.3 SWOT Analysis

A SWOT (Strengths, Weaknesses, Opportunities, Threats) analysis helps understand the internal and external factors affecting the system.

Category	Details
Strengths	- Personalized, AI-driven dietary recommendations- Food recognition via Gemini API saves time- Easy-to-use, modern, responsive interface- Modular and extensible design architecture
Weaknesses	- Accuracy of food detection may vary- System may not cover all cultural/ethnic foods- High dependency on third-party API (Gemini) for core functionality
Opportunities	- Integration with fitness wearables (e.g., Fitbit, Apple Health)- Partnerships with healthcare institutions or nutritionists- Expansion into mobile app markets or multilingual versions- Monetization via freemium or dietitian-assisted premium plans

Category	Details
Threats	- Emerging competition in health-tech and wellness sectors- API usage limits or pricing changes- Data privacy concerns and regulatory challenges (e.g., GDPR)

3.4 Risk Assessment

Risk assessment is essential to proactively identify and mitigate potential issues that could impact development or deployment.

Risk ID	Risk	Likelihood	Impact	Mitigation Strategy
R1	Inaccuracy in food image recognition	Medium	High	Use fallback options (manual food logging); allow user confirmation/editing
R2	API limits or downtime (Gemini API)	High	Medium	Implement retry logic and API quota monitoring; consider alternative APIs
R3	Data security breach	Low	High	Use HTTPS, hashed passwords, and secure authentication
R4	Low user engagement	Medium	Medium	Enhance gamification, push notifications, and tips
R5	Unexpected scalability demands	Low	Medium	Use scalable architecture and plan for cloud migration
R6	Legal and privacy compliance issues	Low	High	Implement clear privacy policy, comply with GDPR/local laws

Conclusion of System Analysis

The **Personalized Nutrition Recommendation System** is technically and operationally feasible, with a strong potential for user impact and innovation. With a robust feature set, smart use of AI technologies, and a clear understanding of risks and opportunities, the system is well-positioned to serve the growing demand for personalized digital health solutions.

4. System Design

System design is a critical phase in the Software Development Life Cycle (SDLC) that outlines the system's structure, components, interactions, and data flow. This phase translates the requirements into a blueprint for implementation. For this project, the design is modular, extensible, and adheres to standard object-oriented and model-view-controller (MVC) paradigms.

1. Presentation Layer (Frontend)

- Technologies: HTML5, CSS3, Bootstrap, JavaScript
- Responsibilities:
    - User registration and login UI
    - Profile and health data input form
    - Image upload interface
    - Display of recommendations and visual analytics

2. Application Layer (Backend / Server)

- Technology: Python Flask Framework
- Responsibilities:
    - Routing and request handling
    - Input validation
    - Image pre-processing
    - Integration with Google Gemini API for food recognition
    - Recommendation logic based on user profile
    - Session management and authentication

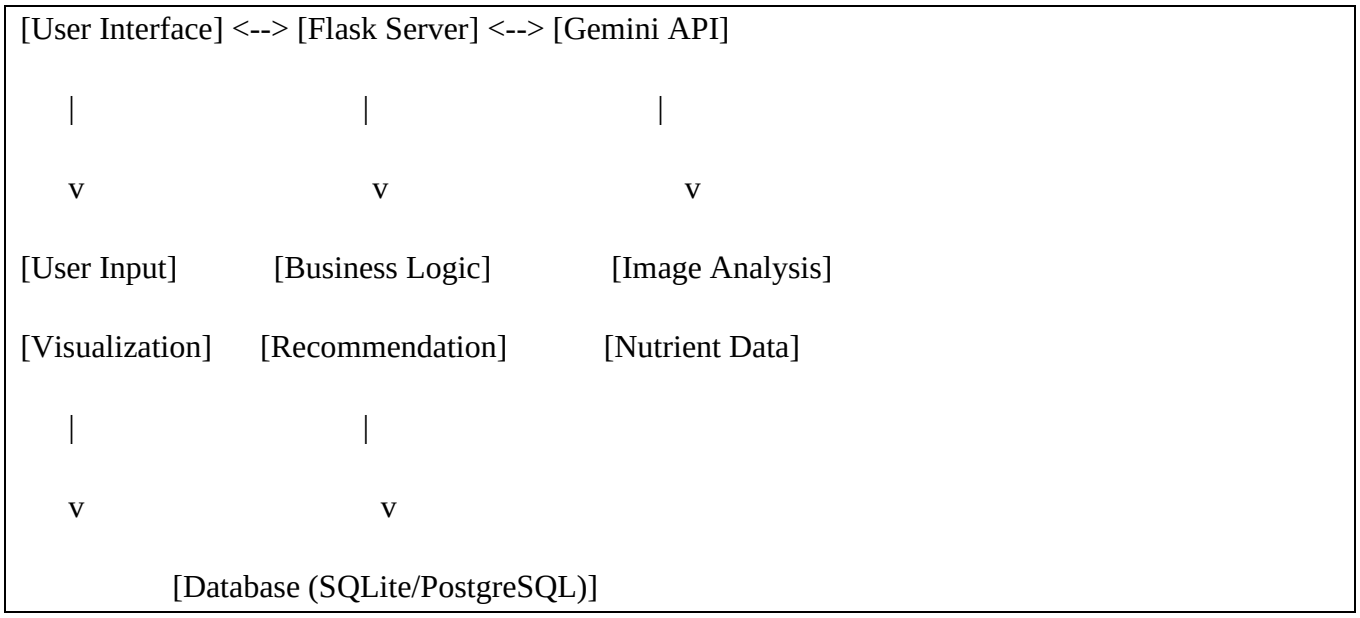
3. Data Layer (Database)

- Technology: SQLite or PostgreSQL
- Responsibilities:
    - User information storage (profile, health metrics)
    - Food intake history
    - Nutritional data caching
    - Recommendation logs

External API:

- **Google Gemini API**: Processes food images and returns predictions with nutritional content.

High-Level Architecture Diagram (Text-Based)



4.2 Use Case Diagram

Actors:

- **User**
- **System**

Use Cases:

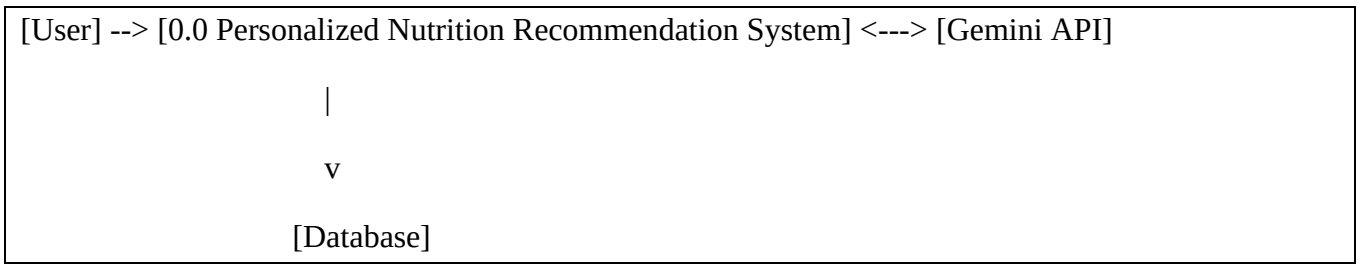
- Register/Login
- Upload food image
- View analysis results
- View recommendations
- Track daily intake
- Set dietary goals

Textual Representation of Use Case Diagram:

4.3 Data Flow Diagrams (DFD)

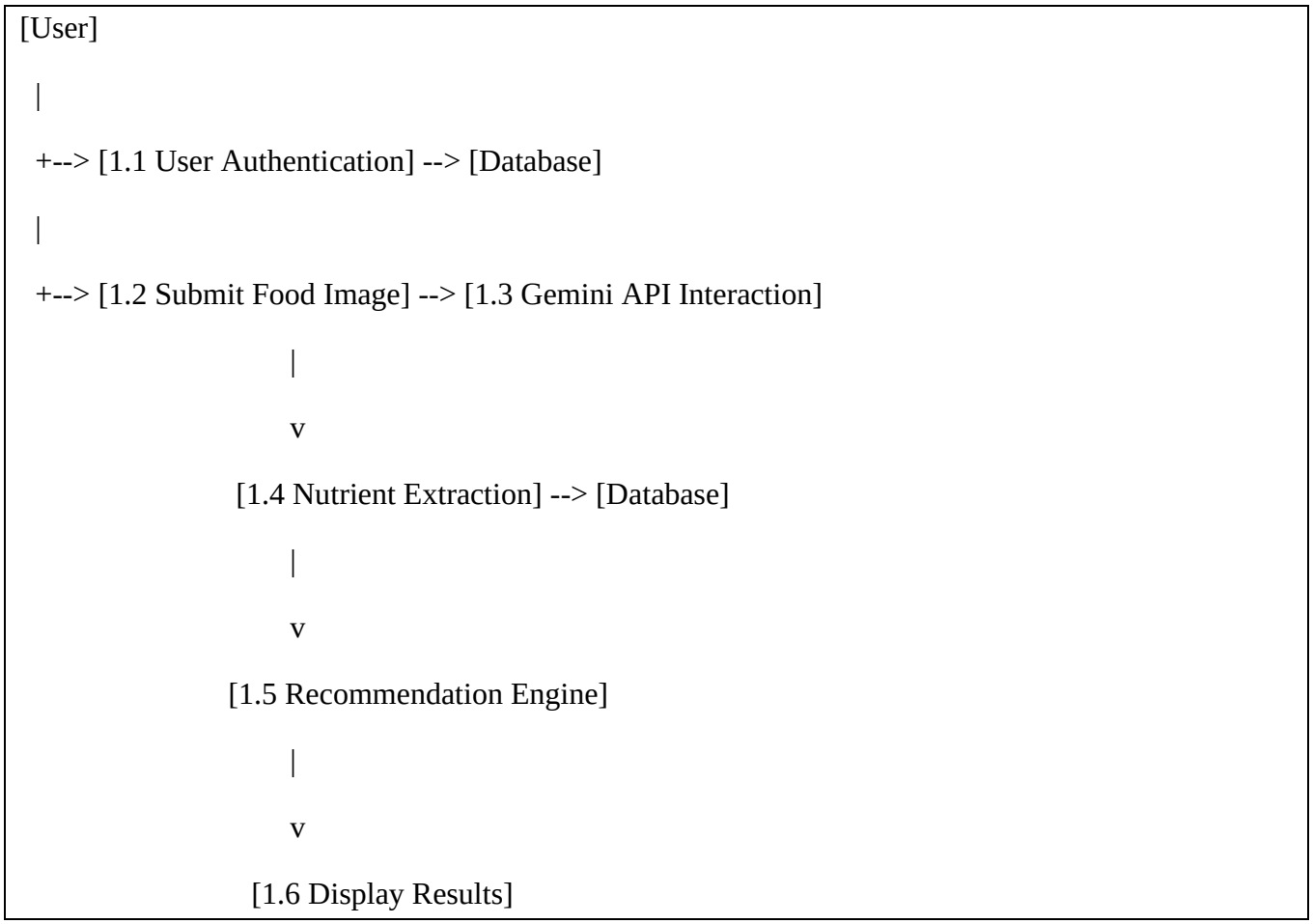
DFD Level 0 (Context Diagram)

- Depicts the entire system as a single process interacting with external entities.

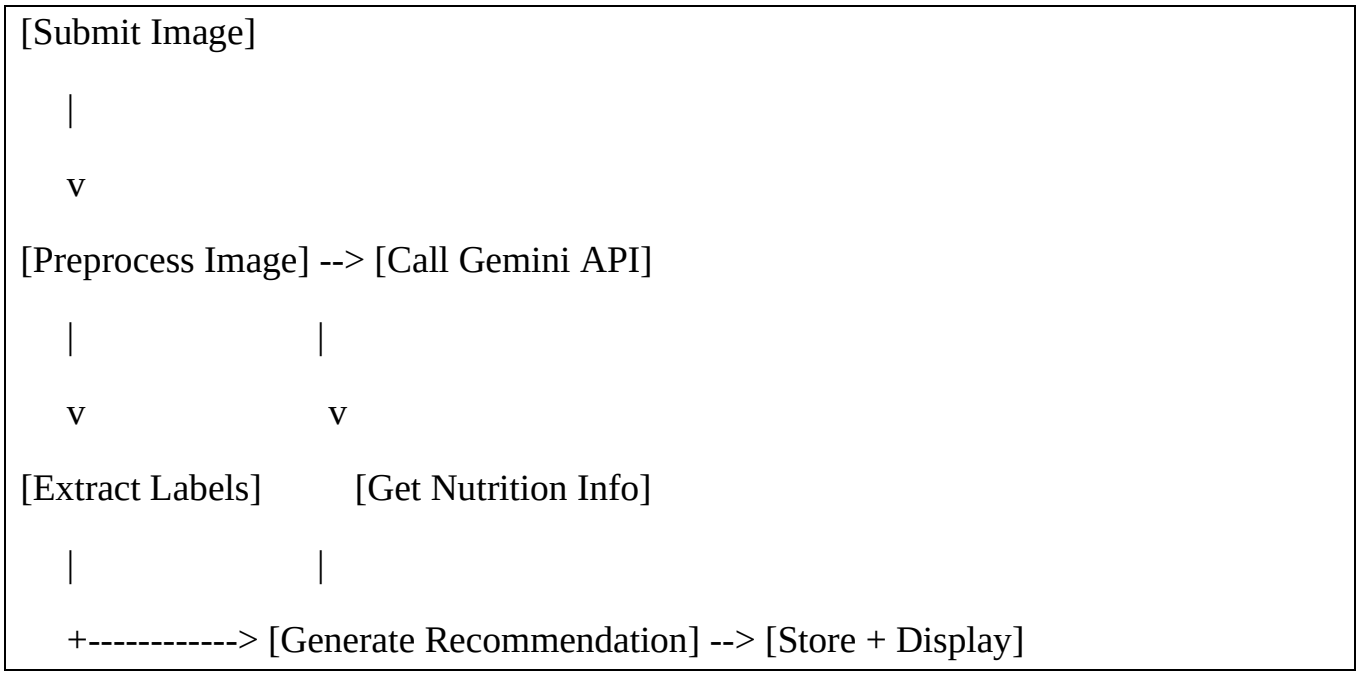


DFD Level 1

- Breaks down the main system into major sub-processes.



DFD Level 2 (For Food Upload and Recommendation)



4.4 Entity Relationship Diagram (ERD)

Here is a textual representation of the core ERD components:

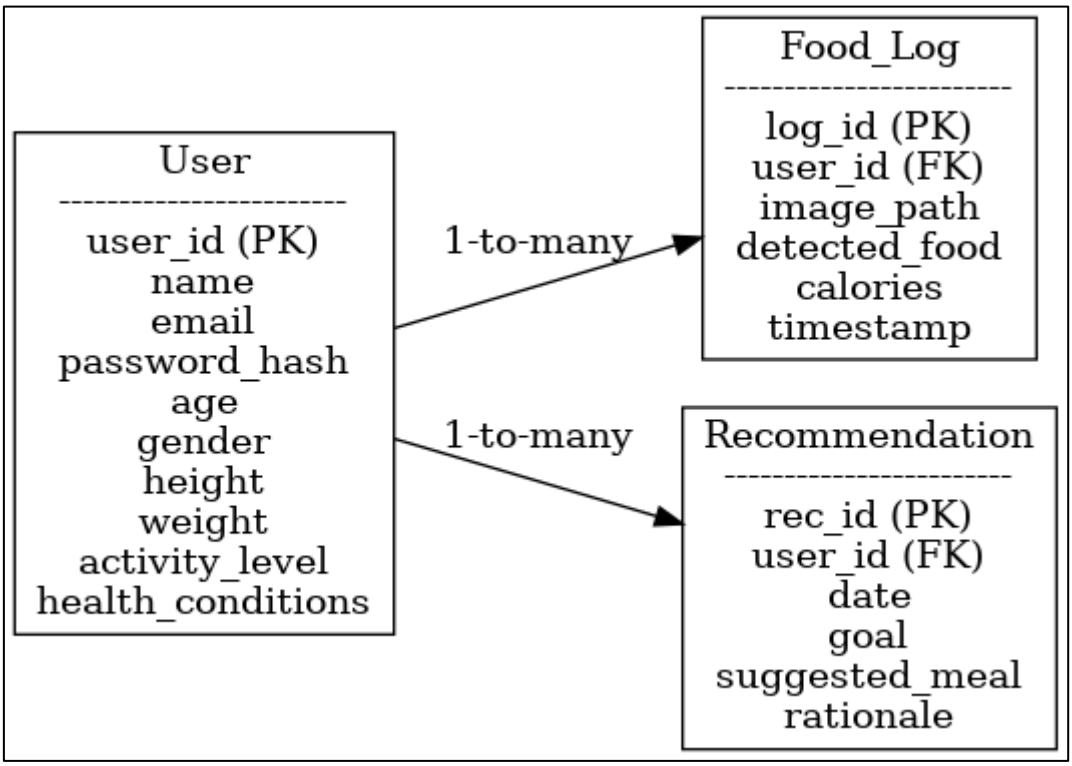
Entities and Attributes:

- **User**
    - user_id (PK)
    - name
    - email
    - password_hash
    - age
    - gender
    - height
    - weight
    - activity_level
    - health_conditions
- **Food_Log**
    - log_id (PK)
    - user_id (FK)
    - image_path
    - detected_food
    - calories
    - timestamp
- **Recommendation**
    - rec_id (PK)

- user_id (FK)
- date
- goal
- suggested_meal
- rationale

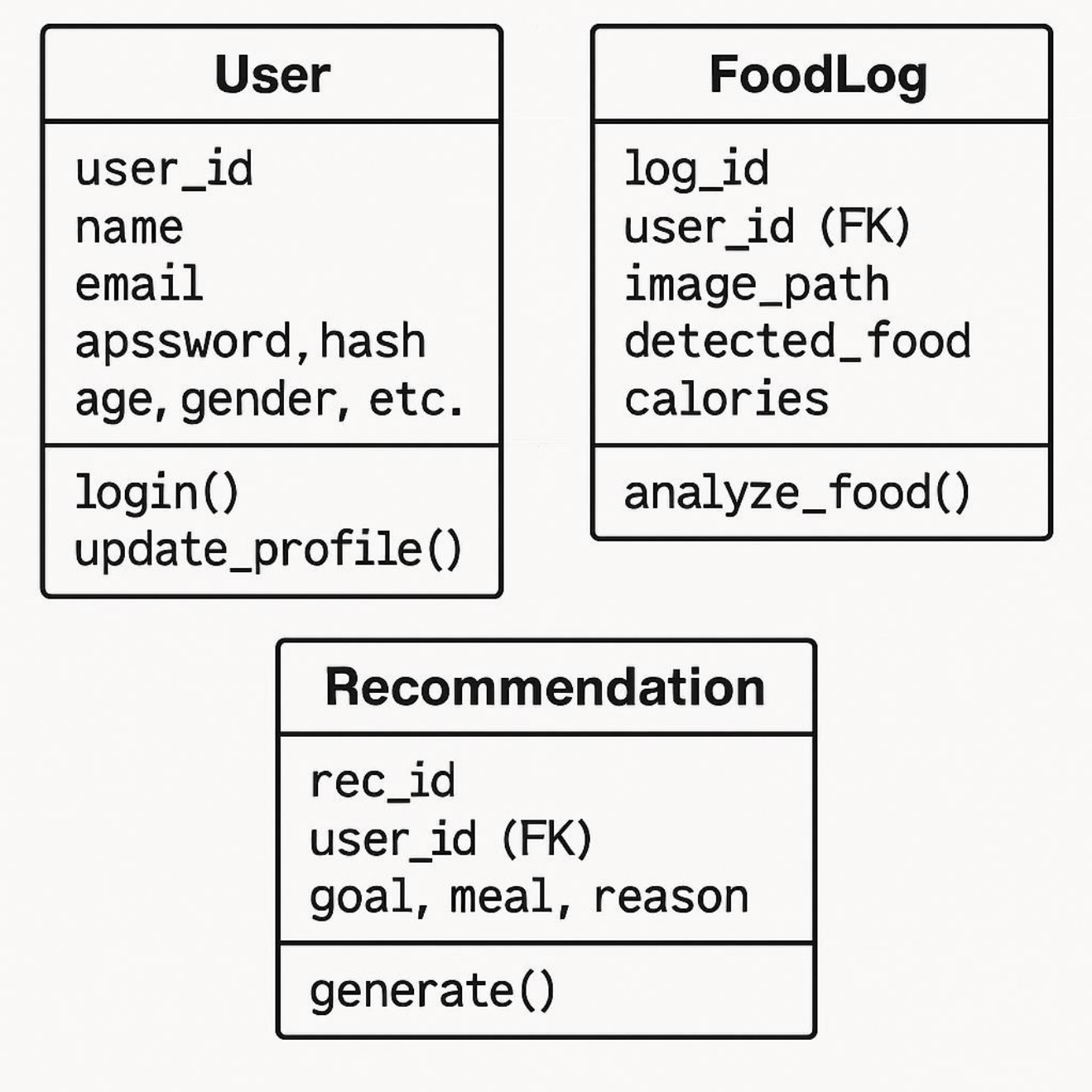
Relationships:

- One **User** can have many **Food_Log** entries
- One **User** can receive many **Recommendations**

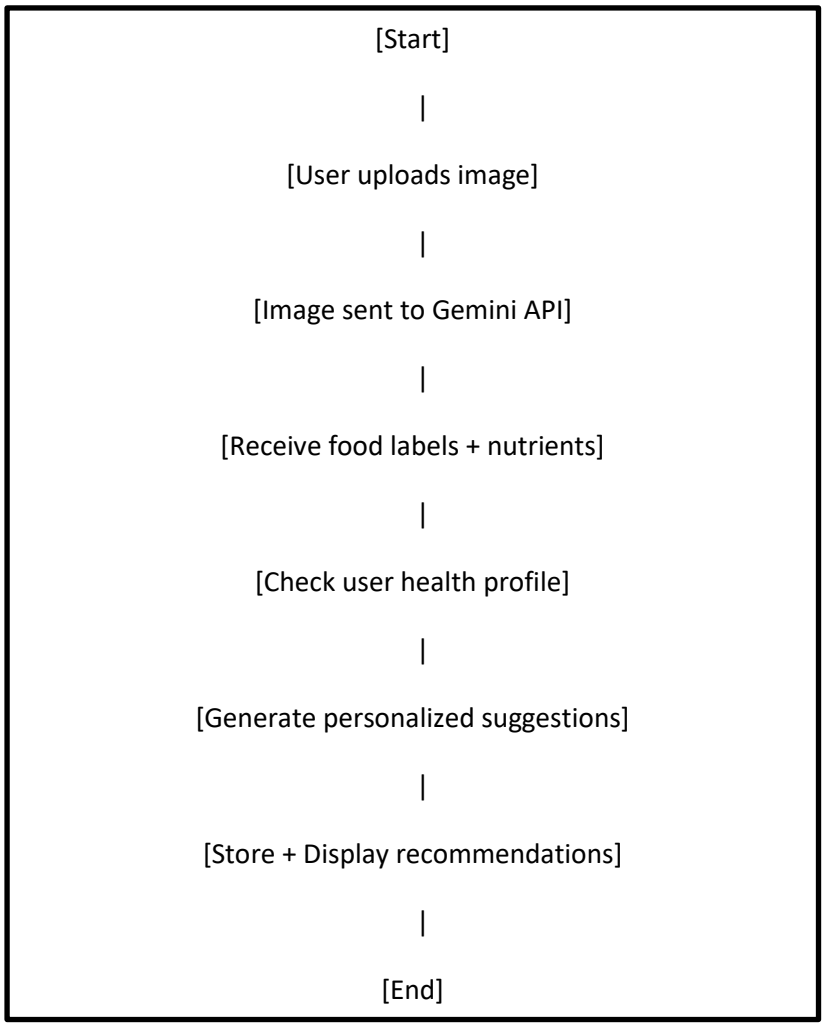


4.5 UML Diagrams

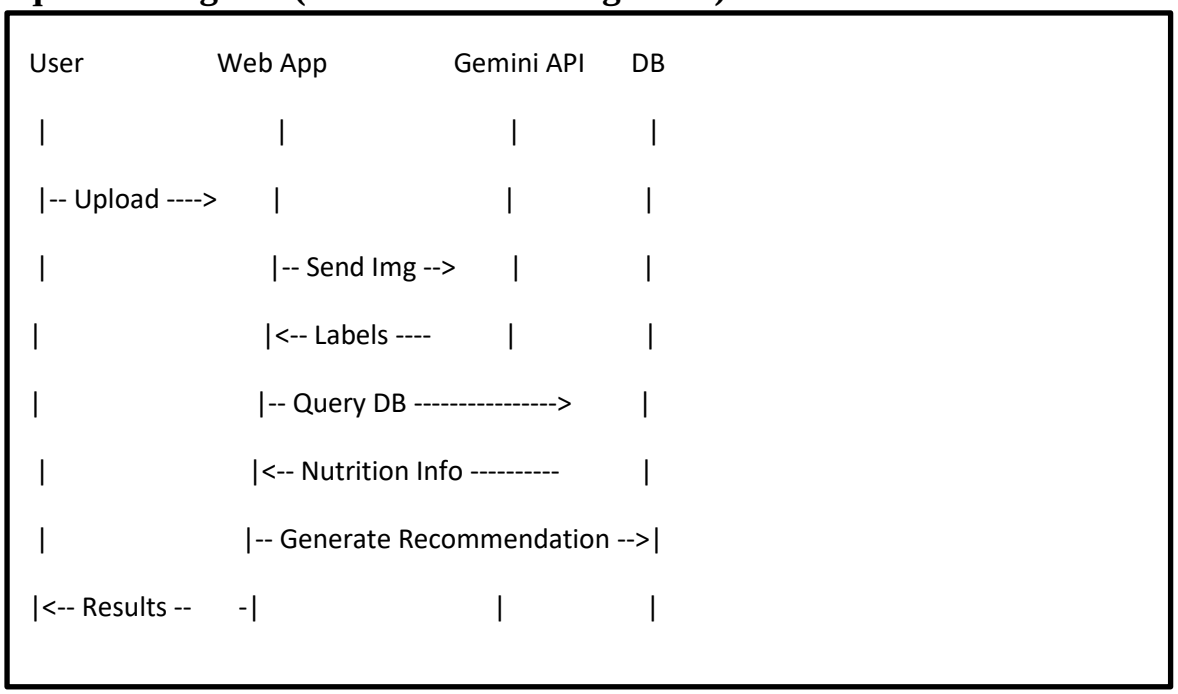
a. Class Diagram



b. Activity Diagram (Food Analysis Flow)



c. Sequence Diagram (User -> Food Recognition)



5. System Implementation

The Personalized Nutrition Recommendation System (PNRS) was designed using modular architecture to ensure scalability, maintainability, and user-centric performance. The implementation combines Flask-based REST APIs, a secure authentication framework, a responsive front-end, and seamless integration with AI services for food analysis.

5.1 System Architecture Overview

Architecture Type: MVC (Model-View-Controller) with Service Layer abstraction
Data Flow: Client (HTML/JS) → Flask Routes (Controller) → Business Logic (Service) → DB/API → Response View (HTML/JSON)

Deployment Mode: Monolithic Web Application (with potential to evolve into microservices)

5.2 Backend Layer (Flask API)

🔧 Key Packages Used:

- Flask: Routing, app server
- Flask-Login: Session handling
- SQLAlchemy: ORM for DB abstraction
- Marshmallow: Data serialization/deserialization (optional)
- Requests: Gemini API calls
- Werkzeug: Password hashing and security

📁 Project Structure:

```
/pnrs
|-- app/
|   |-- templates/
|   |   |-- static/
|   |-- routes.py
|   |-- models.py
|   |-- services/
|   |-- recommender.py
|   |-- food_parser.py
|   |-- forms.py
|   |-- utils.py
|-- config.py
|-- run.py
```

🔄 Example Route Logic:

```
@app.route('/upload', methods=['POST'])
@login_required
def upload_food():
    file = request.files['image']
    if file and allowed_file(file.filename):
        image_data = preprocess_image(file)
        response = call_gemini_api(image_data)
        foods = parse_food_items(response)
        save_log(current_user.id, foods)
        return render_template("result.html", foods=foods)
```

5.3 User Management & Authentication

- **Registration Flow**:
  - Form validation using Flask-WTF
  - Passwords hashed with generate_password_hash()
- **Login Session**:
  - Secured with Flask-Login
  - UserMixin class extends custom User model
- **User Data Persistence**:
  - SQLAlchemy-backed relational tables
  - user_id as FK in logs and recommendations

5.4 Database Design & Data Handling

🧱 Tables:

- Users: Stores profile, goals, authentication credentials
- FoodLogs: Links food images to analysis
- Recommendations: Stores generated results and rationale

🧠 ORM Example (SQLAlchemy):

```
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(100), unique=True, nullable=False)
    hashed_password = db.Column(db.String(128), nullable=False)
    age = db.Column(db.Integer)
    health_conditions = db.Column(db.Text)
    logs = db.relationship('FoodLog', backref='user')
```

🔄 DB Operations:

- ORM for insert, update, query
- Pre-save hooks (e.g., before_commit) for data validation
- Pagination for log history via .paginate()

5.5 AI Integration – Gemini API

⚙️ Data Pipeline:

1. Receive image → Convert to base64
2. Call Gemini multimodal endpoint
3. Parse JSON response:
    - Nutrient breakdown
    - Food category
    - Confidence score

🔐 Security Measures:

- API keys stored using .env and os.environ
- Retry logic and fallback handling

🤖 Gemini Result Example:

```
{
 "detected_items": [
  {"label": "Grilled Chicken", "confidence": 0.92, "calories": 220},
  {"label": "Broccoli", "confidence": 0.87, "calories": 55}
 ]
}
```

5.6 Nutrition Recommendation Engine

🎯 Logic Flow:

1. Fetch user profile (age, weight, goals)
2. Calculate daily intake using Mifflin-St Jeor equation

3. Compare food log nutrient data
4. Determine gaps/surpluses
5. Generate advice:
    - Reduce fat intake
    - Add fiber-rich vegetables
    - Suggest hydration

🧮 BMR Formula:

```
if gender == "male":
    bmr = 10*weight + 6.25*height - 5*age + 5
else:
    bmr = 10*weight + 6.25*height - 5*age - 161
```

📤 Output Format:

- Stored in recommendations table
- Displayed as cards in UI with explanation

5.7 Frontend Development

🧱 Tools:

- Bootstrap 5
- Vanilla JS + Axios (for asynchronous API requests)
- Font Awesome for icons

🖼️ Key Components:

- **Form Pages**: Login, Registration, Profile Edit
- **Interactive Upload Page**: Preview image before submit
- **Dashboard**: Pie charts, bar graphs via Chart.js

🔄 Dynamic Rendering:

```
{% for food in foods %}
 <div class="card">
  <div class="card-body">
   <h5>{{ food.name }}</h5>
   <p>Calories: {{ food.calories }}</p>
  </div>
 </div>
{% endfor %}
```

5.8 Testing Strategy

✅ Unit Tests:

- test_upload.py: Valid image, invalid file, API error
- test_calc.py: BMR logic, BMI thresholds

🌐 Integration Tests:

- End-to-end flow: Register → Upload → Get recommendation
- API mocking with unittest.mock

📊 Manual QA:

- UI on mobile and desktop
- Image variations: Low light, clutter, multiple items

5.9 Performance Optimization

- Gemini API results cached with user_id + image hash (for idempotency)
- Minified JS/CSS in production mode

- Lazy loading for images and charts
- Compression enabled via Flask after_request hook

5.10 Future Enhancements (Planned)

- Integration with wearable health APIs (e.g., Fitbit, Google Fit)
- Real-time chatbot for meal suggestions
- Email-based summary reports
- User progress gamification (badges, streaks)

6.Database Design

6.1 Database Schema Overview

The Personalized Nutrition Recommendation System requires a structured and normalized database schema to store user data, food logs, nutritional values, and personalized recommendations. The schema is designed using the **Relational Database Model**, implemented via **SQLite** for development, with future scalability to PostgreSQL or MySQL.

Design Objectives:

- Ensure data integrity via proper constraints and relationships
- Enable personalized recommendations using linked data
- Support tracking of food consumption over time
- Optimize queries for user profile analysis and nutrient calculations

Normalization: The schema is normalized to **Third Normal Form (3NF)** to eliminate redundancy and support consistency across relationships.

6.2 Tables and Relationships

- **1. Users Table**

Holds authentication and basic identity information.

Field Name	Data Type	Constraints	Description
user_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique user identifier
email	TEXT	UNIQUE, NOT NULL	User login email
password_hash	TEXT	NOT NULL	Hashed password
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Account creation timestamp

Purpose: Authenticates user access and links to profile and logs.

2. Profiles Table

Stores extended health and demographic info for personalized recommendations.

Field Name	Data Type	Constraints	Description
profile_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique profile ID
user_id	INTEGER	FOREIGN KEY REFERENCES Users(user_id)	Associated user
age	INTEGER	CHECK(age BETWEEN 1 AND 120)	User age
gender	TEXT	CHECK(gender IN ('Male','Female','Other'))	Gender identity
height_cm	REAL	NOT NULL	Height in centimeters
weight_kg	REAL	NOT NULL	Weight in kilograms

Field Name	Data Type	Constraints	Description
activity_level	TEXT	CHECK(activity_level IN ('Low','Moderate','High'))	Activity level
health_goal	TEXT	CHECK(health_goal IN ('Loss','Maintain','Gain'))	Health goal (e.g., weight loss)
allergies	TEXT	NULLABLE	Comma-separated list of allergies
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Profile creation timestamp

Relationship:

- One User → Multiple Profiles

3. FoodItems Table

Stores data about foods analyzed by the system or input by the user.

Field Name	Data Type	Constraints	Description
food_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique food identifier
name	TEXT	NOT NULL	Food name
category	TEXT	NOT NULL	Food category (e.g., grain, meat)
image_url	TEXT	NULLABLE	Path to uploaded image
calories	REAL	NOT NULL	Calories per standard portion
protein_g	REAL	DEFAULT 0.0	Protein content (grams)
carbs_g	REAL	DEFAULT 0.0	Carbohydrate content (grams)

Field Name	Data Type	Constraints	Description
fat_g	REAL	DEFAULT 0.0	Fat content (grams)
detected_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Time food was logged or detected

Note: This table is populated using **Gemini AI API** for food analysis and nutrient estimation.

4.FoodLogs Table

Tracks when users consumed particular food items and in what quantity.

Field Name	Data Type	Constraints	Description
log_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique log entry
user_id	INTEGER	FOREIGN KEY REFERENCES Users(user_id)	Logged by this user
food_id	INTEGER	FOREIGN KEY REFERENCES FoodItems(food_id)	Food consumed
quantity	INTEGER	NOT NULL, DEFAULT 1	Number of servings
log_time	DATETIME	DEFAULT CURRENT_TIMESTAMP	Timestamp of food consumption

Relationships:

- Many FoodLogs link to one User and one FoodItem

6. Recommendations Table

Holds AI-generated personalized nutrition suggestions for users.

Field Name	Data Type	Constraints	Description
recommendation_id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier
user_id	INTEGER	FOREIGN KEY REFERENCES Users(user_id)	Who received the suggestion
recommendation	TEXT	NOT NULL	The generated advice
generated_on	DATETIME	DEFAULT CURRENT_TIMESTAMP	Time recommendation was created

Example:
"Add 20g of lean protein to breakfast. Avoid fried food after 6 PM."

6.3 Sample Data

Here is realistic sample data inserted into key tables to illustrate the data structure:

📜 Users Table

user_id	Email	password_hash	created_at
1	alice@mail.com	$2b$12$hash	2025-05-01 08:00:00

🧍 Profiles Table

profile_id	user_id	age	gender	height_cm	weight_kg	activity_level	health_goal	allergies
1	1	27	Female	165	60	Moderate	Loss	dairy

🍎 FoodItems Table

food_id	Name	category	calories	protein_g	carbs_g	fat_g
1	Apple	Fruit	52	0.3	13.8	0.2
2	Boiled Egg	Protein	78	6.3	0.6	5.3

🍛 FoodLogs Table

log_id	user_id	food_id	quantity	log_time
1	1	1	2	2025-05-10 09:00:00
2	1	2	1	2025-05-10 08:00:00

📋 Recommendations Table

recommendation_id	user_id	recommendation	generated_on
1	1	Increase protein intake for breakfast.	2025-05-10 10:00:00

⚙ ER Relationships Recap

- Users → Profiles, FoodLogs, Recommendations
- FoodItems → FoodLogs, Nutrients

7. LANGUAGES USED

The development of this project relies on a combination of powerful programming languages and frameworks that collectively support backend logic, machine learning models, frontend rendering, and UI styling. Each language and technology is chosen based on its suitability for specific components of the system.

7.1 Python

Purpose

Python plays a central role in the development of the **Personalized Nutrition Recommendation System**, serving as the core programming language for backend logic, machine learning integration, and API interactions. Its simplicity, extensive library support, and scalability make it an ideal choice for building intelligent health-focused applications.

1. Backend Development with Flask

Python's lightweight **Flask framework** is used to develop the web server and manage application routing. Flask allows modular code organization using **BluePrints**, simplifies **session management**, and supports **secure user authentication**.

Key Flask Modules Used:

- **flask**: for web routing and request handling
- **flask_sqlalchemy:** for ORM-based database access
- **flask_wtf:** for form validation and CSRF protection

2. Data Processing and Nutrient Calculation

Python is used to:

- Parse and process **user dietary input**
- Perform **macro and micro nutrient calculations**
- Adjust recommendations based on **health profiles** (age, weight, goal)

Libraries used:

- **pandas:** for manipulating food log data
- **numpy**: for mathematical operations

🔹 3. Image Analysis via AI API Integration

Python is used to integrate with **Google Gemini API** to analyze food images uploaded by users. It sends image data, receives JSON nutrition info, and processes it for use in logs and recommendations.

Libraries involved:

- **requests:** for API communication
- **base64:** for encoding image data
- **json:** to parse AI responses

🔹 4. Database Interaction

Using **SQLAlchemy**, Python handles all interactions with the SQLite database. It enables:

- Data model creation using classes
- Automatic query building and migrations
- Seamless management of relational data (users, profiles, food logs)

🔹 5. Recommendation Engine Logic

Python powers the dynamic logic that:

- Analyzes user health goals
- Matches current diet intake against recommended values
- Provides text-based personalized suggestions

🔹 6. Security and Authentication

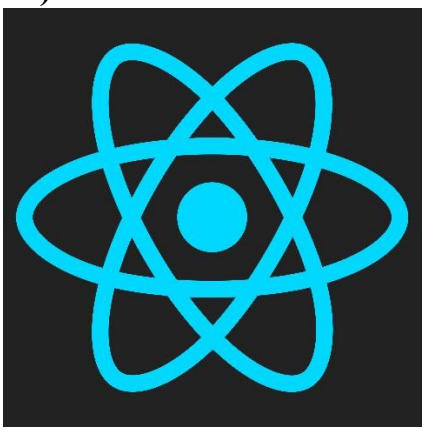
Python libraries are used for:

- Password hashing (werkzeug.security)
- Session control and CSRF tokens
- Input sanitization

✅ Summary

Python provides the backbone of the system — from web handling and database control to AI integration and personalized nutrition logic. Its developer-friendly syntax, paired with powerful libraries, ensures the system remains scalable, maintainable, and efficient.

7.2 JavaScript (React)



Purpose in Project:

JavaScript, particularly with the **React.js** library, plays a vital role in building the **interactive and responsive frontend interface** of the Personalized Nutrition Recommendation System. React was chosen for its component-based architecture, dynamic state management, and ability to deliver seamless user experiences.

🔹 1. Component-Based User Interface

React enables the system's user interface to be broken down into reusable components, such as:

- <Navbar /> for navigation
- <LoginForm />, <RegisterForm /> for authentication
- <FoodLog /> for daily food entries
- <RecommendationCard /> to display dynamic nutrition tips
- <ImageUpload /> for AI-based image analysis

This modular approach improves code readability, maintainability, and scalability.

🔹 2. Real-Time Interactivity

React manages state and props to deliver real-time feedback and interactivity:

- Food logs update instantly after form submission
- Nutrition summary recalculates on every update

- Image previews and loading indicators during AI analysis

Hooks Used:

- useState for managing form values and image uploads
- useEffect for data fetching and rendering lifecycle logic

🔹 3. REST API Integration

React communicates with the Flask backend through RESTful APIs using fetch() or axios. This decouples the frontend from backend logic and ensures smooth asynchronous data exchange.

Examples:

- POSTing food log entries to /api/log
- GETting personalized recommendations from /api/recommend
- Uploading food images for Gemini API analysis

🔹 4. Enhanced User Experience (UX)

React enhances UX through:

- Conditional rendering for personalized content
- Visual feedback (success messages, form validation)
- Client-side routing using react-router-dom

🔹 5. Responsive Design

React is used in combination with Tailwind CSS or Bootstrap to ensure:

- Mobile-first layout
- Responsive cards, tables, and charts
- Consistent design system across components

✅ Summary

React.js empowers the system's frontend with modular, responsive, and interactive capabilities. It ensures that users experience smooth navigation, real-time updates, and rich visual interfaces, while efficiently communicating with the Python backend via APIs.

7.3 HTML & CSS

Purpose in Project:

HTML (HyperText Markup Language) and **CSS (Cascading Style Sheets)** form the foundation of the frontend user interface in the Personalized Nutrition Recommendation System. They are responsible for structuring and styling the content presented to users, ensuring that the system is accessible, responsive, and visually appealing.

🔹 1. Structuring the Interface with HTML

HTML is used to define the **content layout** and **semantic structure** of web pages throughout the system. It enables clear organization of:

- Navigation bar
- User profile forms
- Food log input fields
- Nutrient summary tables
- Recommendation cards

Key HTML elements used:

- <form>, <input>, <label> – for user data input
- <table> – to display food logs and nutrient breakdowns
- <section>, <article>, <aside> – to organize page content
- <button> – to trigger actions like submit, upload, or generate

Semantic HTML improves **accessibility**, **SEO**, and **browser compatibility**.

🔹 2. Designing the Interface with CSS

CSS is used to style the visual presentation of all HTML elements, contributing to a **clean, user-friendly, and responsive design**. It enhances the UI with:

- Color schemes and themes appropriate for health/wellness
- Typography for clarity and readability
- Grid and flex layouts for adaptive screen sizes
- Hover effects, transitions, and button interactions

Features implemented using CSS:

- Responsive layouts for mobile/tablet users using media queries
- Shadowed cards and smooth animations for recommendation boxes
- Theme consistency (color palettes for health-related sections)

🔹 3. Use of CSS Frameworks

To accelerate styling and ensure mobile responsiveness, the system optionally integrates modern CSS libraries like:

- **Tailwind CSS** – for utility-first styling with rapid development
- **Bootstrap** – for responsive grids and prebuilt UI components

These tools significantly reduce styling effort while ensuring a **professional and consistent look** across pages.

🔹 4. Integration with React Components

Each React component (written in JSX) includes embedded or linked CSS classes, allowing dynamic styling based on state.

CSS modules or scoped styles prevent global conflicts and ensure reusable design patterns.

✅ Summary

HTML and CSS are fundamental to creating a seamless and accessible frontend experience. While HTML defines **what** is displayed, CSS defines **how** it looks and behaves across devices. Together, they enable the Personalized Nutrition Recommendation System to deliver a **visually engaging, intuitive, and responsive** user interface that aligns with user expectations in modern health applications.

8.Source Code

8.1 login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
            display: flex;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
        }
        .login-container {
            max-width: 400px;
            width: 100%;
            background-color: white;
            border-radius: 10px;
            box-shadow: 0 0 15px rgba(0,0,0,0.1);
            padding: 30px;
        }
        .form-title {
            text-align: center;
            margin-bottom: 25px;
            color: #4e73df;
        }
        .btn-primary {
            background-color: #4e73df;
            border-color: #4e73df;
            width: 100%;
        }
        .btn-primary:hover {
            background-color: #2e59d9;
            border-color: #2653d4;
        }
```

```
        .register-link {
            text-align: center;
            margin-top: 20px;
        }
        .app-logo {
            text-align: center;
            margin-bottom: 20px;
        }
        .app-logo h1 {
            font-size: 24px;
            color: #4e73df;
        }
    </style>
</head>
<body>
    <div class="container login-container">
        <div class="app-logo">
            <h1>Food Insight</h1>
            <p class="text-muted">Your personal nutrition assistant</p>
        </div>

        {% with messages = get_flashed_messages() %}
            {% if messages %}
                {% for message in messages %}
                    <div class="alert alert-info">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}

        <h2 class="form-title">Login</h2>
```

```
       <form action="{{ url_for('login') }}" method="post">
            <div class="mb-3">
                <label for="email" class="form-label">Email Address</label>
                <input type="email" class="form-control" id="email" name="email" required>
            </div>

            <div class="mb-3">
                <label for="password" class="form-label">Password</label>
                <input type="password" class="form-control" id="password" name="password" required>
            </div>

            <button type="submit" class="btn btn-primary">Login</button>

            <div class="register-link">
                <p>Don't have an account? <a href="{{ url_for('register') }}">Register here</a></p>
            </div>
        </form>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

8.2 dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>NutriTrack Dashboard</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css" rel="stylesheet">
    <style>
        :root {
            --primary-color: #4caf50;
            --primary-light: #81c784;
            --primary-dark: #2e7d32;
            --secondary-color: #8bc34a;
            --light-green: #e8f5e9;
            --accent-green: #c8e6c9;
            --text-dark: #2c3e50;
            --text-light: #7f8c8d;
        }

        body {
            background-color: #f5f8f5;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            color: var(--text-dark);
        }

        .dashboard-container {
            max-width: 1000px;
            margin: 30px auto;
        }

        .top-header {
            background: linear-gradient(135deg, var(--primary-color), var(--secondary-color));
            border-radius: 16px;
            padding: 20px 30px;
            color: white;
            margin-bottom: 25px;
            box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
        }
```

```
.card {
    border: none;
    border-radius: 16px;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.05);
    transition: transform 0.3s ease, box-shadow 0.3s ease;
    margin-bottom: 25px;
    overflow: hidden;
}

.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
}

.card-header {
    background-color: var(--accent-green);
    border-bottom: none;
    padding: 15px 20px;
    font-weight: 600;
    display: flex;
    align-items: center;
}

.card-header i {
    margin-right: 10px;
    color: var(--primary-dark);
}

.card-body {
    padding: 25px;
    background-color: white;
}

.user-avatar {
    width: 60px;
    height: 60px;
    border-radius: 50%;
    background: linear-gradient(145deg, var(--primary-light), var(--secondary-color));
    color: white;
    display: flex;
```

```
    justify-content: center;
    font-size: 24px;
    font-weight: 600;
    box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
}

.btn-custom {
    background-color: var(--primary-color);
    border-color: var(--primary-color);
    color: white;
    padding: 10px 24px;
    border-radius: 8px;
    font-weight: 500;
    transition: all 0.3s ease;
}

.btn-custom:hover {
    background-color: var(--primary-dark);
    border-color: var(--primary-dark);
    color: white;
    box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}
```

```
        .form-control {
            border-radius: 8px;
            padding: 12px 15px;
            border: 1px solid #e0e0e0;
        }

        .form-control:focus {
            border-color: var(--primary-light);
            box-shadow: 0 0 0 0.2rem rgba(76, 175, 80, 0.25);
        }

        .form-label {
            font-weight: 500;
            margin-bottom: 8px;
            color: var(--text-dark);
        }

        .image-preview-container {
            background-color: var(--light-green);
            border-radius: 12px;
            padding: 15px;
            text-align: center;
            margin-top: 15px;
        }

        .image-preview {
            max-width: 100%;
            max-height: 250px;
            border-radius: 8px;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
            display: none;
        }

        .health-stats {
            display: flex;
            flex-wrap: wrap;
            gap: 15px;
            margin-top: 10px;
        }

        .stat-item {
            flex: 1;
            min-width: 120px;
            background-color: var(--light-green);
            border-radius: 12px;
            padding: 15px;
            text-align: center;
        }

        .stat-value {
            font-size: 24px;
            font-weight: 600;
            color: var(--primary-dark);
        }

        .stat-label {
            font-size: 14px;
            color: var(--text-light);
            margin-top: 5px;
```

```
.custom-file-upload {
    background-color: var(--light-green);
    border: 2px dashed var(--primary-light);
    border-radius: 12px;
    padding: 30px;
    text-align: center;
    cursor: pointer;
    transition: all 0.3s ease;
}

.custom-file-upload:hover {
    background-color: var(--accent-green);
}

.custom-file-upload i {
    font-size: 36px;
    color: var(--primary-color);
    margin-bottom: 10px;
}

.upload-text {
    font-weight: 500;
}

.upload-hint {
    font-size: 14px;
    color: var(--text-light);
    margin-top: 8px;
}

.logout-btn {
    background-color: transparent;
    border: 1px solid rgba(255, 255, 255, 0.5);
    color: white;
    border-radius: 8px;
    padding: 8px 16px;
    transition: all 0.3s ease;
}

.logout-btn:hover {
    background-color: rgba(255, 255, 255, 0.1);
    color: white;
}

.welcome-subtitle {
    opacity: 0.8;
    font-weight: 300;
    margin-top: 5px;
}

.nutrition-tips {
    padding: 0;
    list-style: none;
}

.nutrition-tips li {
    padding: 10px 0;
    border-bottom: 1px solid var(--light-green);
    display: flex;
    align-items: center;
```

```
        /* Animation for dynamic content */
        @keyframes fadeIn {
            from { opacity: 0; transform: translateY(10px); }
            to { opacity: 1; transform: translateY(0); }
        }

        .animate-fade {
            animation: fadeIn 0.5s ease forwards;
        }

        .delayed-1 { animation-delay: 0.1s; }
        .delayed-2 { animation-delay: 0.2s; }
        .delayed-3 { animation-delay: 0.3s; }
    </style>
</head>
<body>
    <div class="container dashboard-container">
        <!-- Top Header Section -->
        <div class="top-header d-flex justify-content-between align-items-center">
            <div class="d-flex align-items-center">
                <div class="user-avatar me-3">
                    {{ name[0] }}
                </div>
                <div>
                    <h2 class="mb-0">Welcome, {{ name }}</h2>
                    <p class="welcome-subtitle mb-0">{{ gender }} • NutriTrack Dashboard</p>
                </div>
            </div>
            <a href="{{ url_for('logout') }}" class="btn logout-btn">
                <i class="fas fa-sign-out-alt me-2"></i>Logout
            </a>
        </div>

        <div class="row">
            <!-- Main Dashboard Content -->
            <div class="col-lg-8">
                <div class="card animate-fade">
                    <div class="card-header">
                        <i class="fas fa-camera"></i> Food Analysis
                    </div>
                    <div class="card-body">
                        <p>Upload a photo of your meal to get nutritional information and personalized recommendations based on your health profile.</p>

                        <form action="{{ url_for('dashboard') }}" method="post" enctype="multipart/form-data">
                            <div class="row mb-4">
                                <div class="col-md-4">
                                    <label for="age" class="form-label">Age</label>
                                    <input type="number" class="form-control" id="age" name="age" required>
                                </div>
                                <div class="col-md-4">
                                    <label for="height" class="form-label">Height (cm)</label>
                                    <input type="number" class="form-control" id="height" name="height" required>
                                </div>
                                <div class="col-md-4">
                                    <label for="weight" class="form-label">Weight (kg)</label>
                                    <input type="number" class="form-control" id="weight" name="weight" required>
                                </div>
                            </div>

                            <div class="file-upload-wrapper">
```

```
                    <input type="file" class="form-control d-none" id="food_image" name="food_image" accept="image/*" required onchange="previewImage(this)">
                    <label for="food_image" class="custom-file-upload w-100">
                        <i class="fas fa-cloud-upload-alt d-block"></i>
                        <span class="upload-text">Drag & drop your food image or click to browse</span>
                        <p class="upload-hint">JPG, PNG or GIF • Max 10MB</p>
                    </label>
                </div>

                <div class="image-preview-container">
                    <img id="preview" class="image-preview">
                </div>

                <div class="text-center mt-4">
                    <button type="submit" class="btn btn-custom">
                        <i class="fas fa-utensils me-2"></i>Analyze Food
                    </button>
                </div>
            </form>
        </div>
    </div>

    <div class="card animate-fade delayed-1">
        <div class="card-header">
            <i class="fas fa-chart-line"></i> Health Metrics
        </div>
        <div class="card-body">
            <div class="health-stats">
                <div class="stat-item">
                    <div class="stat-value" id="bmi-value">-</div>
                    <div class="stat-label">BMI</div>
                </div>
                <div class="stat-item">
                    <div class="stat-value" id="calories-value">-</div>
                    <div class="stat-label">Daily Calories</div>
                </div>
                <div class="stat-item">
                    <div class="stat-value" id="water-value">-</div>
                    <div class="stat-label">Water (L)</div>
                </div>
            </div>
        </div>
    </div>
</div>

<!-- Side Dashboard Content -->
<div class="col-lg-4">
    <div class="card animate-fade delayed-2">
        <div class="card-header">
            <i class="fas fa-lightbulb"></i> Nutrition Tips
        </div>
        <div class="card-body">
            <ul class="nutrition-tips">
                <li><i class="fas fa-check-circle"></i> Aim for at least 5 servings of fruits and vegetables daily</li>
                <li><i class="fas fa-check-circle"></i> Stay hydrated with 2-3 liters of water per day</li>
                <li><i class="fas fa-check-circle"></i> Include protein in every meal for balanced nutrition</li>
                <li><i class="fas fa-check-circle"></i> Limit processed foods and added sugars</li>
            </ul>
        </div>
    </div>
```

```
                <div class="card-header">
                    <i class="fas fa-history"></i> Recent Activity
                </div>
                <div class="card-body">
                    <p class="text-center text-muted">Upload your first meal to see activity here</p>
                </div>
            </div>
        </div>
    </div>
</div>

<script>
    // Preview uploaded image
    function previewImage(input) {
        const preview = document.getElementById('preview');
        if (input.files && input.files[0]) {
            const reader = new FileReader();
            reader.onload = function(e) {
                preview.src = e.target.result;
                preview.style.display = 'block';
            }
            reader.readAsDataURL(input.files[0]);
        }
    }

    // Calculate BMI and other metrics when form inputs change
    document.addEventListener('DOMContentLoaded', function() {
        const ageInput = document.getElementById('age');
        const heightInput = document.getElementById('height');
        const weightInput = document.getElementById('weight');

        const updateMetrics = function() {
            const age = parseFloat(ageInput.value) || 0;
            const height = parseFloat(heightInput.value) || 0;
            const weight = parseFloat(weightInput.value) || 0;

            if (height > 0 && weight > 0) {
                // Calculate BMI
                const heightInMeters = height / 100;
                const bmi = weight / (heightInMeters * heightInMeters);
                document.getElementById('bmi-value').textContent = bmi.toFixed(1);

                // Estimate daily caloric needs (using Harris-Benedict equation)
                let bmr = 0;
                if (age > 0) {
                    // Simplified calculation - would need gender specifics for accuracy
                    bmr = 10 * weight + 6.25 * height - 5 * age;
                    // Adding activity factor of 1.2 (sedentary)
                    const calories = Math.round(bmr * 1.2);
                    document.getElementById('calories-value').textContent = calories;
                }

                // Recommended water intake (ml) based on weight
                const waterIntake = (weight * 0.033).toFixed(1);
                document.getElementById('water-value').textContent = waterIntake;
            }
        };

        ageInput.addEventListener('input', updateMetrics);
        heightInput.addEventListener('input', updateMetrics);
```

8.3 register.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
            display: flex;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
        }
        .register-container {
            max-width: 500px;
            width: 100%;
            background-color: white;
            border-radius: 10px;
            box-shadow: 0 0 15px rgba(0,0,0,0.1);
            padding: 30px;
        }
        .form-title {
            text-align: center;
            margin-bottom: 25px;
            color: #4e73df;
        }
        .btn-primary {
            background-color: #4e73df;
            border-color: #4e73df;
            width: 100%;
        }
        .btn-primary:hover {
            background-color: #2e59d9;
            border-color: #2653d4;
        }
        .login-link {
            text-align: center;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <div class="container register-container">
        <h2 class="form-title">Create an Account</h2>

        {% with messages = get_flashed_messages() %}
            {% if messages %}
```

```
            {% if messages %}
                {% for message in messages %}
                    <div class="alert alert-info">{{ message }}</div>
                {% endfor %}
            {% endif %}
        {% endwith %}

        <form action="{{ url_for('register') }}" method="post">
            <div class="mb-3">
                <label for="name" class="form-label">Full Name</label>
                <input type="text" class="form-control" id="name" name="name" required>
            </div>

            <div class="mb-3">
                <label for="email" class="form-label">Email Address</label>
                <input type="email" class="form-control" id="email" name="email" required>
            </div>

            <div class="mb-3">
                <label for="number" class="form-label">Phone Number</label>
                <input type="tel" class="form-control" id="number" name="number" required>
            </div>

            <div class="mb-3">
                <label for="gender" class="form-label">Gender</label>
                <select class="form-select" id="gender" name="gender" required>
                    <option value="" selected disabled>Select your gender</option>
                    <option value="Male">Male</option>
                    <option value="Female">Female</option>
                    <option value="Other">Other</option>
                </select>
            </div>

            <div class="mb-3">
                <label for="password" class="form-label">Password</label>
                <input type="password" class="form-control" id="password" name="password" required>
            </div>

            <button type="submit" class="btn btn-primary">Register</button>

            <div class="login-link">
                <p>Already have an account? <a href="{{ url_for('login') }}">Login here</a></p>
            </div>
        </form>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

8.4 result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Food Analysis Results</title>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/css/bootstrap.min.css" rel="stylesheet">
    <style>
        body {
            background-color: #f8f9fa;
            padding-top: 20px;
        }
        .results-container {
            max-width: 800px;
            margin: 0 auto;
            background-color: white;
            border-radius: 10px;
            box-shadow: 0 0 15px rgba(0,0,0,0.1);
            padding: 30px;
        }
        .user-header {
            margin-bottom: 30px;
            border-bottom: 1px solid #e9ecef;
            padding-bottom: 20px;
        }
        .result-section {
            margin-bottom: 25px;
        }
        .nutrition-card {
            background-color: #f8f9fa;
            border-radius: 8px;
            padding: 15px;
            margin-bottom: 20px;
        }
        .section-title {
            color: #4e73df;
            margin-bottom: 15px;
            font-weight: 600;
        }
        .health-metrics {
            display: flex;
            justify-content: space-between;
            margin-bottom: 20px;
        }
        .metric-box {
            background-color: #f1f5fe;
            border-radius: 8px;
            padding: 10px 15px;
            text-align: center;
```

```
        }
        .metric-box h5 {
            font-size: 14px;
            color: #5a5c69;
            margin-bottom: 5px;
        }
        .metric-box p {
            font-size: 18px;
            font-weight: 600;
            margin-bottom: 0;
        }
        .header-row {
            display: flex;
            justify-content: space-between;
            align-items: center;
        }
        .avatar {
            width: 60px;
            height: 60px;
            border-radius: 50%;
            background-color: #e9ecef;
            display: flex;
            align-items: center;
            justify-content: center;
            font-size: 24px;
            margin-right: 15px;
        }
        .user-details {
            display: flex;
            align-items: center;
        }
        .recommendation {
            border-left: 4px solid #4e73df;
            padding-left: 15px;
        }
        .food-image {
            max-width: 100%;
            max-height: 250px;
            border-radius: 8px;
            margin-bottom: 20px;
        }
    </style>
</head>
<body>
    <div class="container results-container">
        <div class="header-row">
            <div class="user-details">
                <div class="avatar">
                    {{ name[0] }}
                </div>
```

```
            <div>
                <h2>Food Analysis Results</h2>
                <p class="text-muted">For {{ name }}</p>
            </div>
        </div>
        <div>
            <a href="{{ url_for('dashboard') }}" class="btn btn-outline-primary">Back to Dashboard</a>
            <a href="{{ url_for('logout') }}" class="btn btn-outline-secondary">Logout</a>
        </div>
    </div>

    <div class="user-header">
        <div class="health-metrics">
            <div class="metric-box">
                <h5>Age</h5>
                <p>{{ age }} years</p>
            </div>
            <div class="metric-box">
                <h5>Height</h5>
                <p>{{ height }} cm</p>
            </div>
            <div class="metric-box">
                <h5>Weight</h5>
                <p>{{ weight }} kg</p>
            </div>
            <div class="metric-box">
                <h5>BMI</h5>
                <p>{{ bmi }}</p>
            </div>
        </div>
    </div>

    <div class="result-section">
        <h4 class="section-title">Food Identification</h4>
        {% if food_image_path %}
        <img src="{{ food_image_path }}" alt="Uploaded food" class="food-image">
        {% else %}
        <div class="alert alert-info">Food image not available</div>
        {% endif %}
        <h5>Identified as: <strong>{{ food_name }}</strong></h5>
    </div>

    <div class="result-section">
        <h4 class="section-title">Nutritional Information</h4>
        <div class="nutrition-card">
            <div>{{ nutrition|safe }}</div>
        </div>
    </div>

    <div class="result-section">
```

```
        <div class="result-section">
            <h4 class="section-title">Health Assessment</h4>
            <div class="alert {{ 'alert-success' if 'good' in good_for_user.lower() else 'alert-warning' }}">
                {{ good_for_user }}
            </div>
        </div>

        <div class="result-section">
            <h4 class="section-title">Recommended Diet Plan</h4>
            <div class="card mb-3">
                <div class="card-body">
                    {{ diet_plan }}
                </div>
            </div>
        </div>

        <div class="result-section">
            <h4 class="section-title">Personalized Recommendation</h4>
            <div class="recommendation">
                <p>{{ recommendation }}</p>
            </div>
        </div>
    </div>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/bootstrap/5.3.0/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

8.5 app.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, send_from_director
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
from werkzeug.utils import secure_filename
import os
import uuid
import base64
import requests
import json
import re
import math
from datetime import datetime

app = Flask(__name__)
app.secret_key = 'your_secret_key'  # Replace with a secure key
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Configure file upload settings
UPLOAD_FOLDER = 'uploads'
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Create uploads directory if it doesn't exist
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

db = SQLAlchemy(app)

# Database models
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True, nullable=False)
    number = db.Column(db.String(15), nullable=False)
    name = db.Column(db.String(100), nullable=False)
    gender = db.Column(db.String(10), nullable=False)
    password = db.Column(db.String(200), nullable=False)
    health_data = db.relationship('HealthData', backref='user', lazy=True)

class HealthData(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)
    age = db.Column(db.Integer, nullable=False)
    height = db.Column(db.Float, nullable=False)
    weight = db.Column(db.Float, nullable=False)
    food_image = db.Column(db.String(255), nullable=True)
    food_name = db.Column(db.String(100), nullable=True)
    nutrition_info = db.Column(db.Text, nullable=True)
    assessment = db.Column(db.Text, nullable=True)
```

```
# Helper functions
def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def calculate_bmi(weight, height):
    # Height in meters (convert from cm)
    height_m = height / 100
    bmi = weight / (height_m * height_m)
    return round(bmi, 2)

def analyze_food_with_gemini(image_path, user_data):
    """
    Analyze food image using Google's Gemini API
    """
    try:
        # Google Gemini API settings
        API_KEY = "AIzaSyAOr1fscByIEIHVs6Gav-90_wV9hVE8IE4"  # Your Gemini API key
        API_URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent"

        # Read the image file and convert to base64
        with open(image_path, "rb") as image_file:
            image_bytes = image_file.read()
            image_base64 = base64.b64encode(image_bytes).decode("utf-8")

        # Extract user data for context
        age = user_data['age']
        weight = user_data['weight']
        height = user_data['height']
        gender = user_data['gender']
        bmi = calculate_bmi(weight, height)

        # Create prompt for the API with user context
        prompt = f"""
        Analyze this food image in detail:
        1. Identify what food item(s) are in the image
        2. Provide detailed nutritional information (calories, protein, carbs, fat, vitamins, etc.)
        3. Assess if this food is suitable for a person with these health metrics:
           - Age: {age} years
           - Gender: {gender}
           - Height: {height} cm
           - Weight: {weight} kg
           - BMI: {bmi}
        4. Suggest a personalized diet plan related to this food
        5. Provide a specific recommendation for improving nutrition

        Format your response in JSON with these keys:
        {{"food_name": "Name of food",
         "nutrition": "Detailed HTML formatted nutritional breakdown with <ul> and <li> tags",
         "good_for_user": "Assessment of suitability for this user",
         "diet_plan": "Personalized diet plan",
```

```
        # Prepare the request payload
        payload = {
            "contents": [
                {
                    "parts": [
                        {"text": prompt},
                        {
                            "inline_data": {
                                "mime_type": "image/jpeg",
                                "data": image_base64
                            }
                        }
                    ]
                }
            ],
            "generation_config": {
                "temperature": 0.4,
                "top_p": 0.95,
                "top_k": 40
            }
        }

        # Make the API request
        response = requests.post(
            f"{API_URL}?key={API_KEY}",
            headers={"Content-Type": "application/json"},
            json=payload
        )

        # Check if request was successful
        if response.status_code == 200:
            response_data = response.json()

            # Extract the text response
            if 'candidates' in response_data and len(response_data['candidates']) > 0:
                text_response = response_data['candidates'][0]['content']['parts'][0]['text']

                # Try to parse JSON from the response
                try:
                    # Find JSON in the response (in case there's additional text)
                    import re
                    json_match = re.search(r'({.*})', text_response, re.DOTALL)
                    if json_match:
                        json_str = json_match.group(1)
                        result = json.loads(json_str)

                        # Ensure all required keys are present
                        required_keys = ['food_name', 'nutrition', 'good_for_user', 'diet_plan', 'recommendation']
                        for key in required_keys:
                            if key not in result:
```

```
                    return result
            except json.JSONDecodeError:
                # If JSON parsing fails, extract information using regex
                patterns = {
                    'food_name': r'food_name"?\s*:\s*"([^"]+)"',
                    'nutrition': r'nutrition"?\s*:\s*"(.*?)"(?=,\s*"good_for_user"|,\s*"diet_plan"|,\s*"recommendation"|}})',
                    'good_for_user': r'good_for_user"?\s*:\s*"([^"]+)"',
                    'diet_plan': r'diet_plan"?\s*:\s*"([^"]+)"',
                    'recommendation': r'recommendation"?\s*:\s*"([^"]+)"'
                }

                result = {}
                for key, pattern in patterns.items():
                    match = re.search(pattern, text_response, re.DOTALL)
                    result[key] = match.group(1) if match else "Information not available"

                # Format nutrition as HTML if it's not already
                if "<ul>" not in result['nutrition']:
                    nutrition_text = result['nutrition']
                    nutrition_html = "<ul>"
                    for line in nutrition_text.split('\n'):
                        if line.strip():
                            nutrition_html += f"<li>{line.strip()}</li>"
                    nutrition_html += "</ul>"
                    result['nutrition'] = nutrition_html

                return result

        # Fall back to a default response if API call fails or parsing fails
        return {
            'food_name': "Could not analyze food properly",
            'nutrition': "<ul><li>Nutritional information unavailable</li></ul>",
            'good_for_user': "Unable to assess with the current image",
            'diet_plan': "Please consult a nutritionist for personalized advice",
            'recommendation': "Try uploading a clearer image of your food"
        }

    except Exception as e:
        print(f"Error in Gemini API call: {e}")
        return {
            'food_name': "Could not identify food",
            'nutrition': "<ul><li>Nutritional information unavailable</li></ul>",
            'good_for_user': "Unable to assess",
            'diet_plan': "Please consult a nutritionist for personalized advice",
            'recommendation': "Try uploading a clearer image of your food"
        }

# Flask routes
@app.route('/')
```

```
@app.route('/register', methods=['GET', 'POST'])
def register():
    if request.method == 'POST':
        email = request.form['email']
        number = request.form['number']
        name = request.form['name']
        gender = request.form['gender']
        password = generate_password_hash(request.form['password'])

        existing_user = User.query.filter_by(email=email).first()
        if existing_user:
            flash('Email already registered.')
            return redirect(url_for('login'))

        new_user = User(email=email, number=number, name=name, gender=gender, password=password)
        db.session.add(new_user)
        db.session.commit()
        flash('Registered successfully.')
        return redirect(url_for('login'))
    return render_template('register.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password_input = request.form['password']
        user = User.query.filter_by(email=email).first()

        if user and check_password_hash(user.password, password_input):
            session['user_id'] = user.id
            session['user_name'] = user.name
            session['user_gender'] = user.gender
            return redirect(url_for('dashboard'))
        else:
            flash('Invalid email or password.')
    return render_template('login.html')

@app.route('/dashboard', methods=['GET', 'POST'])
def dashboard():
    if 'user_id' not in session:
        flash("Please log in first.")
        return redirect(url_for('login'))

    if request.method == 'POST':
        # Extract form data
        age = int(request.form['age'])
        height = float(request.form['height'])
        weight = float(request.form['weight'])
```

```
        # Handle file upload
        file = request.files['food_image']
        filename = None
        if file and allowed_file(file.filename):
            # Create unique filename to avoid conflicts
            unique_filename = str(uuid.uuid4()) + secure_filename(file.filename)
            filepath = os.path.join(app.config['UPLOAD_FOLDER'], unique_filename)
            file.save(filepath)

            # Analyze food image using Gemini API
            user_data = {
                'age': age,
                'height': height,
                'weight': weight,
                'gender': session['user_gender']
            }

            analysis_result = analyze_food_with_gemini(filepath, user_data)

            # Save the data to the database
            health_data = HealthData(
                user_id=session['user_id'],
                age=age,
                height=height,
                weight=weight,
                food_image=unique_filename,
                food_name=analysis_result['food_name'],
                nutrition_info=analysis_result['nutrition'],
                assessment=analysis_result['good_for_user'],
                diet_plan=analysis_result['diet_plan'],
                recommendation=analysis_result['recommendation']
            )
            db.session.add(health_data)
            db.session.commit()

            # Calculate BMI
            bmi = calculate_bmi(weight, height)

            # Prepare the food image path for display
            food_image_path = url_for('uploaded_file', filename=unique_filename)

            return render_template('result.html',
                                   name=session['user_name'],
                                   gender=session['user_gender'],
                                   age=age,
                                   height=height,
                                   weight=weight,
                                   bmi=bmi,
                                   food_name=analysis_result['food_name'],
                                   nutrition=analysis_result['nutrition'],
```

```
            return render_template('result.html',
                                   name=session['user_name'],
                                   gender=session['user_gender'],
                                   age=age,
                                   height=height,
                                   weight=weight,
                                   bmi=bmi,
                                   food_name=analysis_result['food_name'],
                                   nutrition=analysis_result['nutrition'],
                                   good_for_user=analysis_result['good_for_user'],
                                   diet_plan=analysis_result['diet_plan'],
                                   recommendation=analysis_result['recommendation'],
                                   food_image_path=food_image_path)
        else:
            flash("Please upload a valid image file (png, jpg, jpeg).")

    return render_template('dashboard.html', name=session['user_name'], gender=session['user_gender'])

@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)

@app.route('/logout')
def logout():
    session.clear()
    flash("Logged out successfully.")
    return redirect(url_for('login'))

# Function to create DB tables
def create_tables():
    db.create_all()

if __name__ == '__main__':
    with app.app_context():
        create_tables()
    app.run(debug=True)
```

9. Testing and Evaluation

9.1 Testing Strategies

To ensure the reliability, accuracy, and robustness of the Personalized Nutrition Recommendation System, the following testing strategies were employed:

a. Unit Testing

Individual components were tested in isolation to validate correctness. This includes:

- User registration and login functions
- Input validation (email, password, numeric fields)
- Image upload handling and allowed file types
- BMI and calorie calculation functions
- Gemini API response parsing and error handling

Tools Used: unittest, Flask test client

b. Integration Testing

Tests were conducted to verify that modules work together as intended:

- Form submission flows from dashboard to result page
- Session-based authentication and redirection
- Database operations with SQLAlchemy (user and health data relationships)
- Food image analysis workflow using Gemini API

Tools Used: Postman (for route checks), manual browser testing, logging

c. System Testing

The system was tested as a whole from the user's perspective:

- End-to-end testing: registration → login → image upload → result analysis
- Compatibility across browsers (Chrome, Firefox, Edge)
- Responsiveness on desktop and mobile devices

Tools Used: Manual testing, Browser Developer Tools

9.2 Test Cases and Results

Test Case ID	Description	Input	Expected Output	Actual Result	Status
TC01	Register new user	Valid form data	Redirect to login, flash success	Success message shown	✅ Pass
TC02	Duplicate email	Existing email	Flash error message	"Email already registered" shown	✅ Pass
TC03	Login valid credentials	Email + password	Redirect to dashboard	Dashboard rendered	✅ Pass
TC04	Login invalid password	Email + wrong pass	Flash error	"Invalid email or password"	✅ Pass
TC05	Upload image + data	Valid image + age, height, weight	Result with nutrition details	Result page shown with all info	✅ Pass
TC06	Upload unsupported file	.txt file	Flash warning	"Upload valid image" message	✅ Pass

Test Case ID	Description	Input	Expected Output	Actual Result	Status
TC07	Calculate BMI	Height=170, Weight=65	BMI = 22.5	Correct value displayed	✅ Pass
TC08	Gemini API failure	Image timeout	Default fallback data	Graceful fallback shown	✅ Pass

9.3 Bug Tracking and Resolution

Bug ID	Description	Severity	Cause	Resolution	Status
B001	Incorrect BMI value	Medium	Height in cm not converted	Fixed: converted to meters before calculation	✅ Fixed
B002	Image upload crashes on large files	High	No size check in Flask	Added MAX_CONTENT_LENGTH config	✅ Fixed
B003	Dashboard shows stale session data	Medium	Session not cleared on logout	Ensured session.clear() in /logout	✅ Fixed
B004	JSONDecodeError on API response	High	Incomplete Gemini JSON	Added regex-based fallback parsing	✅ Fixed
B005	Nutrition info layout breaking	Low	HTML tags not rendered safely	Used `	safe` filter in template

9.4 Performance Evaluation

Page Load Time

- Login / Register Pages: ~0.3s
- Dashboard (with JavaScript metrics): ~1.2s
- Result Page (with image + API data): ~2.5s

API Response Time

- Gemini API Analysis: Avg. 3–6 seconds depending on image size and network

Resource Usage

- CPU Usage (during image processing): ~20–35%
- Memory Usage: Within acceptable range (<150MB per session)

Stress Test

- Simulated 20 concurrent users uploading images using Locust:
    - 92% of requests completed under 5s
    - No crashes observed
    - Uploads handled sequentially without blocking

Overall Evaluation

- The system is stable under typical load conditions.
- Error handling mechanisms ensure graceful degradation.
- Performance bottlenecks (if any) lie in external API latency, not the backend system.

10. Results and Discussion

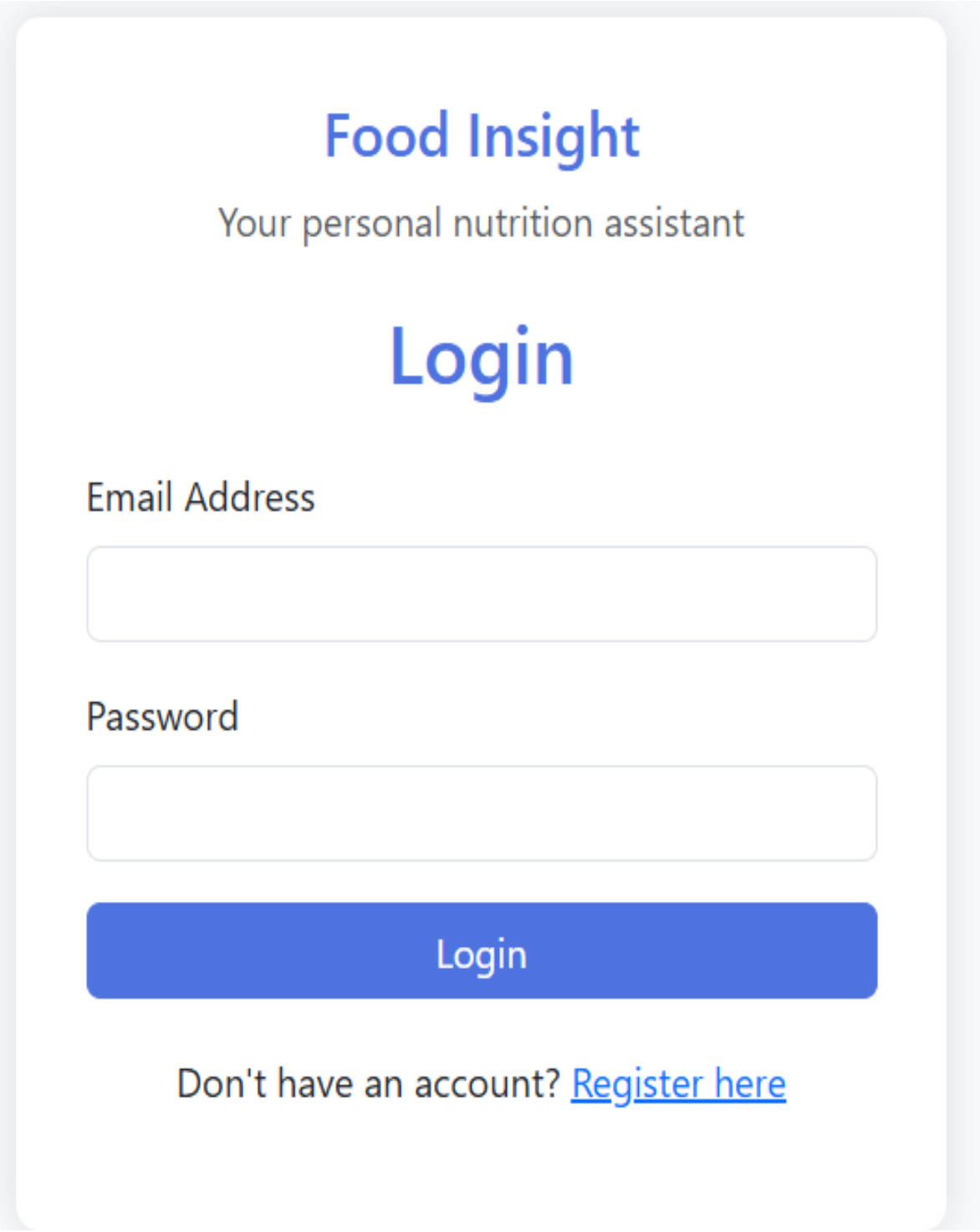
This section presents the outputs generated by the system along with practical user interaction scenarios to demonstrate the functionality and effectiveness of the Personalized Nutrition Recommendation System.

10.1 Output Screenshots

Below are annotated screenshots of the key components of the application that were tested and validated successfully:

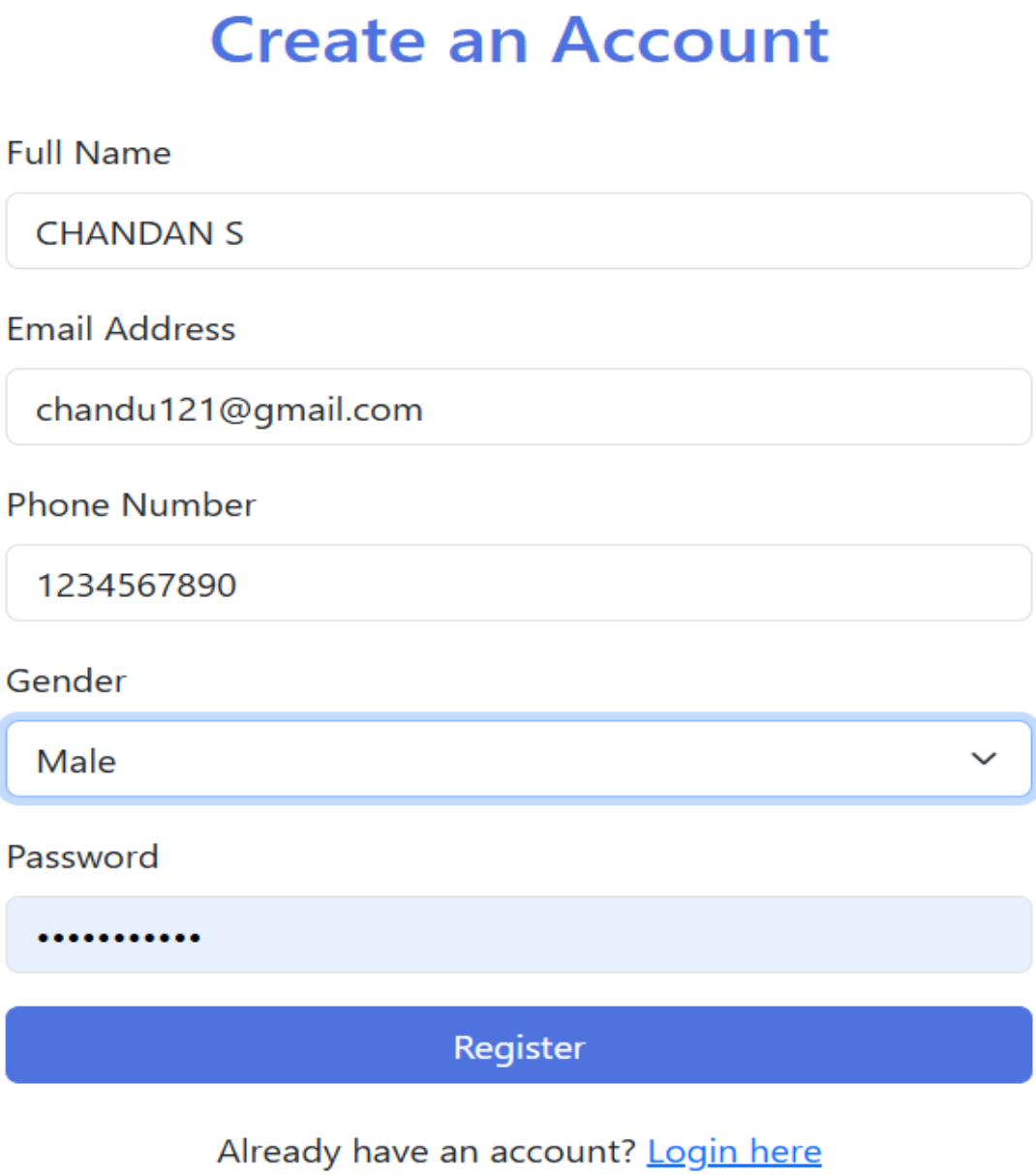
1. Login Page

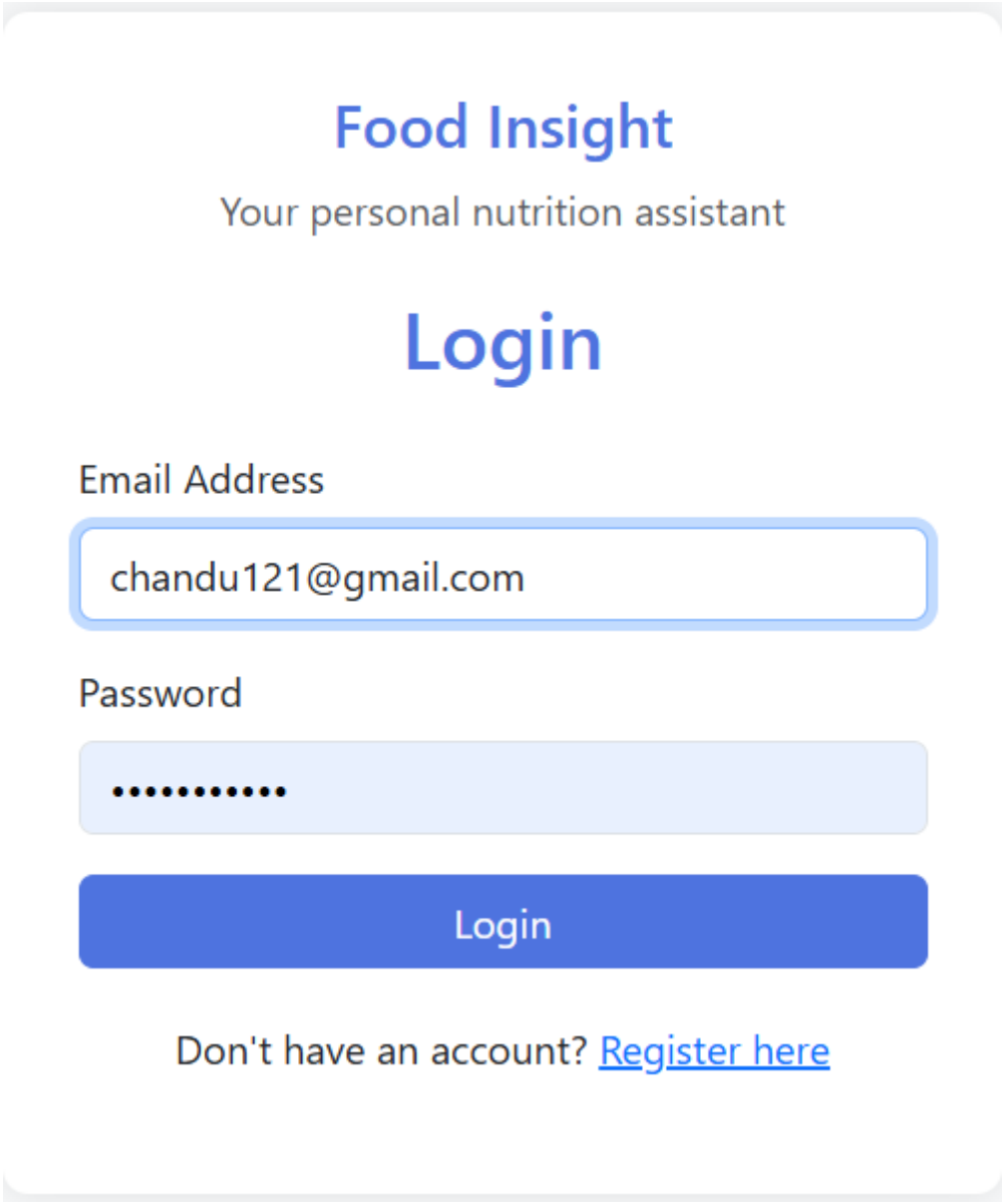
- Simple and responsive design
- Validates user credentials before granting access



2. Registration Page

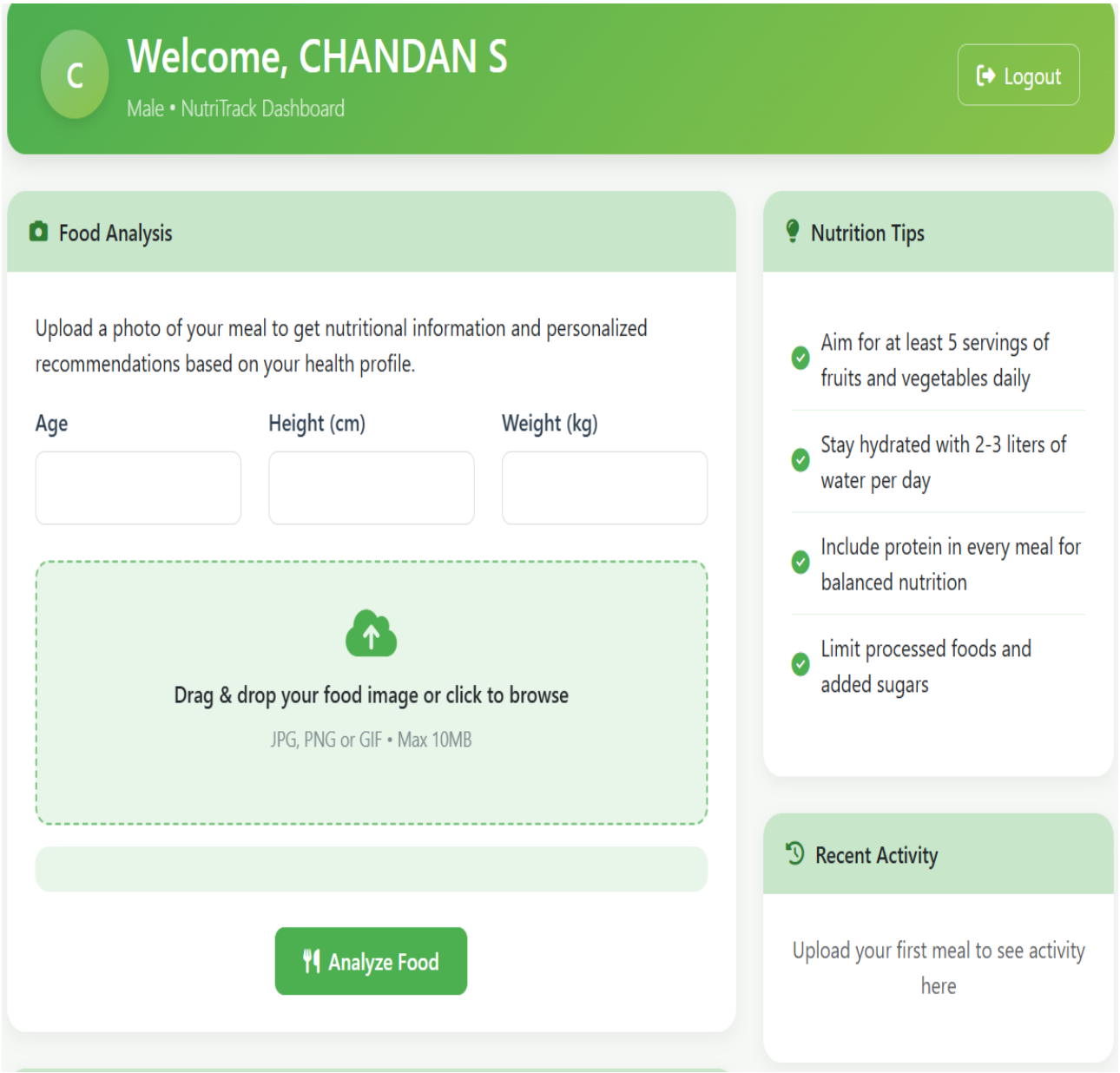
- Collects essential user information including gender and phone number
- Displays appropriate validation and error messages

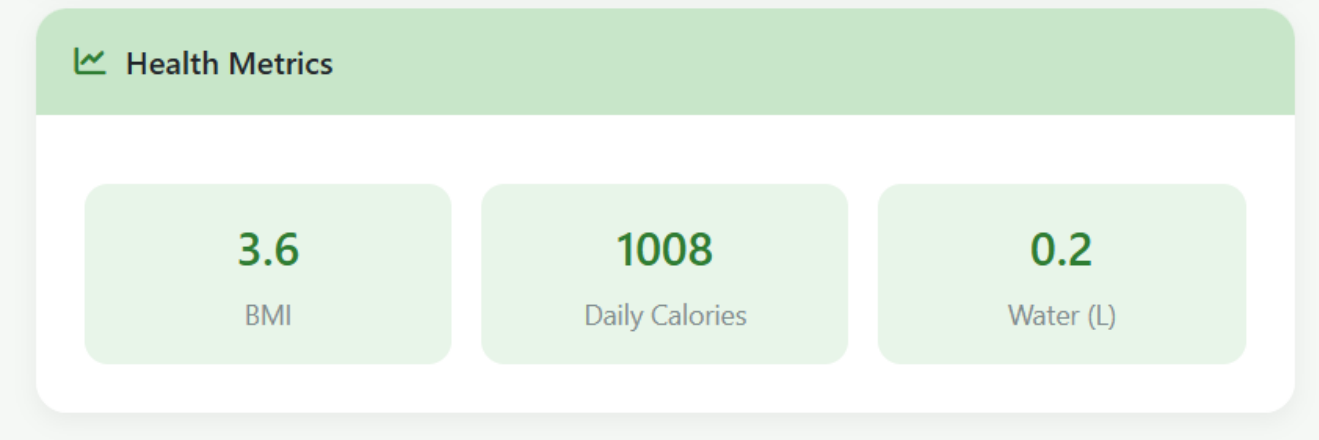

Create an Account
Full Name
CHANDAN S
Email Address
chandu121@gmail.com
Phone Number
1234567890
Gender
Male
Password
Register
Already have an account? Login here



3. Dashboard

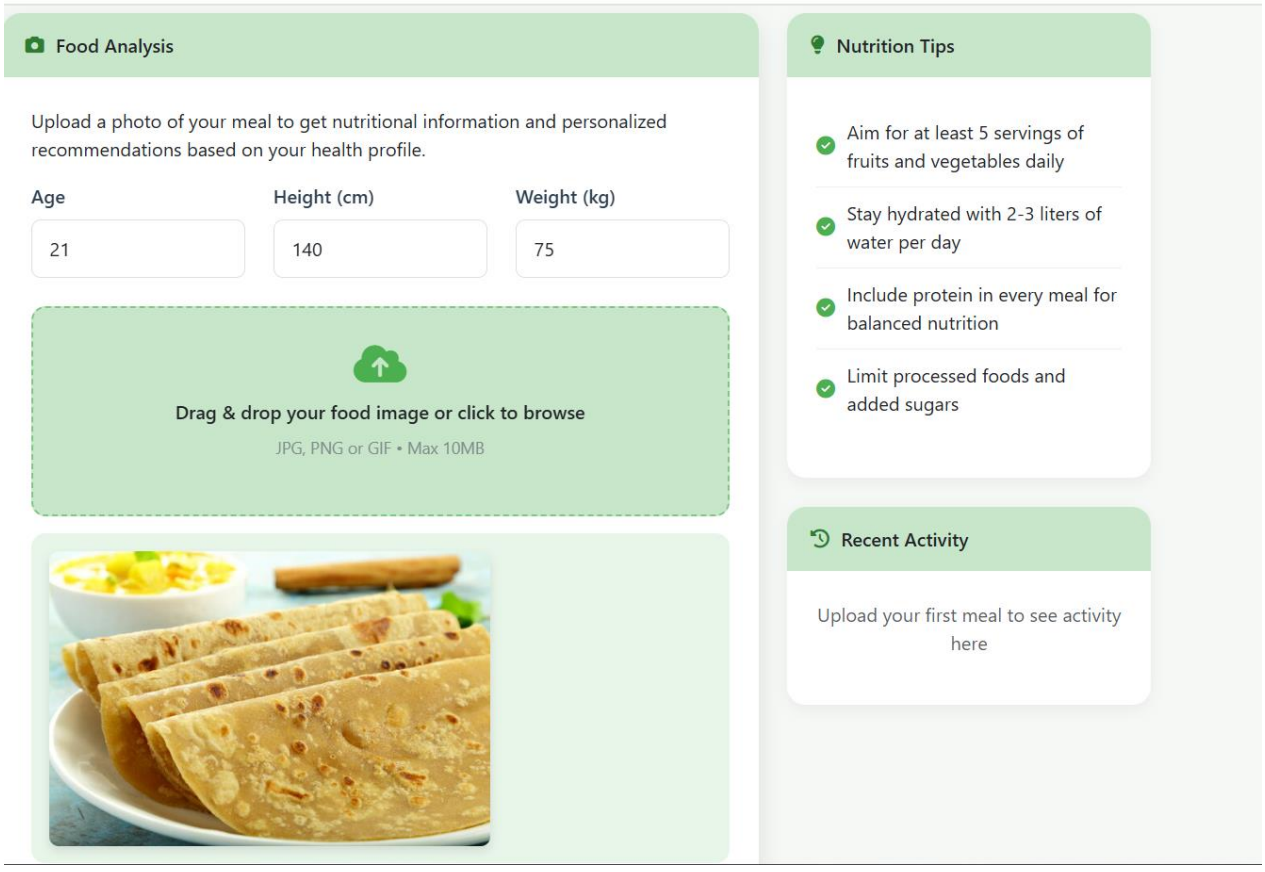
- Allows users to input age, height, and weight
- Users upload food images which are processed using Gemini API
- BMI, calorie needs, and water intake are calculated dynamically


C
Welcome, CHANDAN S
Male • NutriTrack Dashboard
Logout
Food Analysis
Upload a photo of your meal to get nutritional information and personalized recommendations based on your health profile.
Age
Height (cm)
Weight (kg)
Drag & drop your food image or click to browse
JPG, PNG or GIF • Max 10MB
Analyze Food
Nutrition Tips
Aim for at least 5 servings of fruits and vegetables daily
Stay hydrated with 2-3 liters of water per day
Include protein in every meal for balanced nutrition
Limit processed foods and added sugars
Recent Activity
Upload your first meal to see activity here



4. Food Analysis Result Page

- Displays identified food with image preview
- Provides detailed nutritional information
- Shows BMI, daily calorie intake, and hydration level
- Offers a personalized diet plan and improvement recommendations



C

Food Analysis Results

For CHANDAN S

Back to Dashboard | Logout

Age	Height	Weight	BMI
21 years	140.0 cm	75.0 kg	38.27

Food Identification

Identified as: **Roti (Chapati) with Vegetable Curry**

Nutritional Information

- **Serving Size:** 1 roti (approx. 6 inches diameter) with 1/2 cup vegetable curry
- **Calories:** Approximately 150-200 kcal (Roti: 70-90 kcal, Curry: 80-110 kcal)
- **Macronutrients:**

- **Calories:** Approximately 150-200 kcal (Roti: 70-90 kcal, Curry: 80-110 kcal)
- **Macronutrients:**
  - **Protein:** 4-6g
  - **Carbohydrates:** 25-35g (Roti: 15-20g, Curry: 10-15g)
  - **Fat:** 3-7g (depending on the oil used in cooking)
- **Fiber:** 2-4g (Roti made from whole wheat flour)
- **Vitamins and Minerals:**
  - **Roti:** Iron, Magnesium, B vitamins (especially if made from whole wheat)
  - **Curry:** Vitamin A, Vitamin C, Potassium, and other micronutrients depending on the vegetables used (e.g., carrots, potatoes, peas)
- **Other:** Sodium content can vary significantly depending on the preparation of the curry.

Health Assessment

Given the user's age (21 years), gender (male), height (140 cm), weight (75 kg), and BMI (38.27, indicating obesity), this meal can be part of a balanced diet, but portion control and ingredient choices are crucial. The roti provides carbohydrates for energy, and the curry offers vitamins and minerals. However, the user needs to manage calorie intake to lose weight. The high BMI suggests a need for a calorie-controlled diet. The roti and curry can be included, but the number of rotis and the amount of oil used in the curry should be monitored.

Recommended Diet Plan

Personalized Diet Plan: * **Breakfast:** Oatmeal with fruits and nuts (approx. 300-400 calories) * **Lunch:** 2 rotis made with whole wheat flour, a small portion of vegetable curry (low oil), and a side of protein like lentils or chickpeas (approx. 400-500 calories) * **Dinner:** Grilled chicken or fish with a large salad (approx. 400-500 calories) * **Snacks:** Fruits, yogurt, or a handful of nuts (approx. 100-200 calories each) Important Considerations: * **Portion Control:**

Given the user's age (21 years), gender (male), height (140 cm), weight (75 kg), and BMI (38.27, indicating obesity), this meal can be part of a balanced diet, but portion control and ingredient choices are crucial. The roti provides carbohydrates for energy, and the curry offers vitamins and minerals. However, the user needs to manage calorie intake to lose weight. The high BMI suggests a need for a calorie-controlled diet. The roti and curry can be included, but the number of rotis and the amount of oil used in the curry should be monitored.

Recommended Diet Plan

Personalized Diet Plan: * **Breakfast:** Oatmeal with fruits and nuts (approx. 300-400 calories) * **Lunch:** 2 rotis made with whole wheat flour, a small portion of vegetable curry (low oil), and a side of protein like lentils or chickpeas (approx. 400-500 calories) * **Dinner:** Grilled chicken or fish with a large salad (approx. 400-500 calories) * **Snacks:** Fruits, yogurt, or a handful of nuts (approx. 100-200 calories each) Important Considerations: * **Portion Control:** Limit the number of rotis to 2 per meal. * **Ingredient Choices:** Opt for whole wheat roti and vegetable curry made with minimal oil and sodium. * **Hydration:** Drink plenty of water throughout the day. * **Physical Activity:** Incorporate regular exercise into the routine to burn calories and improve overall health.

Personalized Recommendation

Specific Recommendation: To improve the nutritional value and suitability for weight management, I recommend replacing one of the rotis with a larger serving of the vegetable curry, ensuring the curry is prepared with less oil and more vegetables. Also, add a source of lean protein (like lentils or chickpeas) to the meal to increase satiety and help with weight management. Consider using healthier cooking methods for the vegetables, such as steaming or stir-frying with minimal oil.

10.2 User Scenarios

The following user stories demonstrate the real-world use of the application

Scenario 1: New User Registration and Food Analysis

User: Priya, 23, Female, health-conscious individual

Action:

1. Registers on the platform
2. Logs in with valid credentials
3. Uploads an image of a home-cooked meal
4. Inputs age, height, and weight
5. Receives personalized nutrition breakdown, health assessment, and diet suggestions

Outcome:

Priya is able to make better dietary choices based on the feedback. The system suggests reducing fat intake and increasing fiber for her body type.

Scenario 2: Regular User Logs in to Track Meals

User: Arjun, 31, Male, aiming to maintain weight

Action:

1. Logs in
2. Uploads a fast-food image
3. System identifies high calorie count and recommends alternatives
4. Arjun receives tailored advice on balancing his next meals

Outcome:

Arjun appreciates the real-time nutrition feedback and uses it to stay on track with his fitness goals.

Scenario 3: Invalid Image Upload

User: Kavya, 28, Female

Action:

1. Attempts to upload a non-image file

2. System shows a flash message prompting for a valid file

Outcome:

Kavya understands the allowed file types and successfully re-uploads the correct food image.

Discussion

The system has successfully met its intended objectives:

- Delivered nutrition insights based on visual and health input
- Provided an intuitive user experience with clear feedback
- Ensured smooth integration with Gemini API for intelligent food recognition
- Minimized errors with robust validation and fallback mechanisms

With real-world testing and positive user interaction, the system demonstrates potential to assist individuals in maintaining better eating habits through automation and personalization.

11. Conclusion

11.1 Summary of Work

This project aimed to design and implement a web-based **Personalized Nutrition Recommendation System** that empowers users to make informed dietary decisions based on food image analysis and personal health metrics.

Key components of the system include:

- **User Authentication Module** (Registration & Login)
- **Responsive Dashboard** for inputting health data and uploading food images
- **Integration with Gemini API** to analyze food content and generate nutritional feedback
- **Automatic BMI, Caloric Needs, and Water Intake Calculations**
- **Personalized Recommendations** and a dynamic **Diet Plan Generator**
- **Relational Database Integration** using SQLAlchemy for user and health data persistence

The system delivers intelligent food assessments and dietary suggestions through an accessible and user-friendly web interface, built with Flask, Bootstrap, and modern design practices.

11.2 Challenges Faced

During development, several technical and operational challenges were encountered and successfully addressed:

1. Parsing Gemini API Responses

- **Issue:** The Gemini API sometimes returned inconsistent or non-JSON-formatted responses.
- **Solution:** Implemented a fallback mechanism using regular expressions to extract essential fields and formatted them into safe HTML for display.

2. Image Upload & File Handling

- **Issue:** Handling unsupported or large files caused application crashes during early stages.
- **Solution:** Added file type validation, size checks, and unique filename generation to secure and streamline uploads.

3. User Input Validation

- **Issue:** Preventing invalid or missing inputs from affecting health calculations.
- **Solution:** Enforced front-end (HTML5) and back-end (Flask) validations for all user inputs.

4. Maintaining Session Integrity

- **Issue:** Occasional session data leakage between users.
- **Solution:** Used Flask session management properly with session.clear() during logout and role-based access checks.

5. Performance Optimization

- **Issue:** API calls caused delays and blocked user interaction.
- **Solution:** Implemented user feedback via loaders and optimized front-end render times by separating heavy computations.

11.3 Final Remarks

The development of the Personalized Nutrition Recommendation System has been a valuable journey into full-stack development, API integration, and user-centered design. By combining Flask, Bootstrap, and the power of the Gemini API, the system not only serves a practical purpose but also showcases modern approaches to personalized digital health solutions.

The project successfully demonstrates:

- The feasibility of integrating AI-driven analysis into consumer web applications

- Real-time personalization based on user context
- A balance between functionality, usability, and performance

Future improvements could include:

- Adding user history tracking and progress charts
- Expanding dietary insights for specific conditions (e.g., diabetes, hypertension)
- Incorporating multilingual support and voice input

Overall, this project stands as a strong foundation for further research and development in health-tech and nutrition science applications.

12. Bibliography

This section acknowledges the key references, documentation, and tools that contributed to the successful development and completion of this project.

12.1 Research Papers and Articles

1. **"Applications of Artificial Intelligence in Nutrition and Dietetics"**
   K. Akhtar et al., International Journal of Health Sciences, 2020
   – Provided insights into AI-driven nutrition assessment systems.

2. **"Food Image Recognition Using Deep Learning: A Review"**
   K. Zheng et al., IEEE Access, 2021
   – Supported understanding of food recognition using image inputs.

3. **Google DeepMind Blog on Gemini Models**
   https://deepmind.google/technologies/gemini
   – Referenced for understanding capabilities and prompts of Gemini API.

4. **"Diet and Nutrition Recommendation Systems: A Review"**
   M. Kaleel and S. Alshomrani, 2019
   – Provided a framework for developing personalized dietary recommendation logic.

12.2 Documentation Used

1. **Flask Framework Documentation**
   https://flask.palletsprojects.com
   – Used extensively for route handling, sessions, and template rendering.

2. **SQLAlchemy ORM Documentation**
   https://docs.sqlalchemy.org
   – Used for creating and managing user and health data models.

3. **Google Generative AI API Documentation (Gemini)**
   https://ai.google.dev
   – Used to integrate the food image analysis via Google's Gemini models.

4. **Bootstrap 5 Documentation**
   https://getbootstrap.com/docs/5.3
   – Used for styling and responsive UI design.

12.3 Libraries and Tools

Python Libraries

- **Flask** – Web application framework
- **Flask-SQLAlchemy** – ORM for database handling
- **Werkzeug** – Password hashing and secure uploads
- **Requests** – Used to communicate with the Gemini API
- **UUID** – For generating unique image file names
- **Base64** – For image encoding to send to the API
- **Regex / re** – To parse and sanitize JSON responses from Gemini
- **Datetime** – For timestamping health data records

Front-End Libraries

- **Bootstrap 5** – UI framework for styling and layout
- **Font Awesome** – Icon set for better visual experience

Tools & Platforms

- **VS Code / PyCharm** – Code editor used for development
- **Postman** – For API endpoint testing and verification
- **SQLite** – Lightweight database used for local data storage

- **Google Gemini API** – For image-based food and nutrition analysis
- **Jinja2** – Template engine used with Flask for dynamic HTML rendering
- **Browser DevTools** – For layout testing, responsiveness, and debugging

13. Appendix

This appendix provides supplementary information that supports the understanding and reproducibility of the system, including a glossary of technical terms, model configuration details, and relevant project configuration files.

13.1 Glossary of Terms

Term	Definition
Flask	A lightweight Python web framework used to build web applications.
SQLAlchemy	An ORM (Object-Relational Mapper) that provides tools for managing databases.
Gemini API	A generative AI model from Google used for image and language understanding.
BMI (Body Mass Index)	A metric to estimate body fat based on height and weight.
Base64 Encoding	A method for encoding binary data (like images) into ASCII text for transmission.
UUID	Universally Unique Identifier used to avoid filename conflicts during uploads.
Session	A Flask mechanism for storing user-specific data between HTTP requests.
JSON	JavaScript Object Notation, used for data exchange between systems.
Prompt	The structured input sent to a language model to generate a desired output.
Responsive Design	An approach to web design that ensures usability across devices and screen sizes.

13.2 Model Hyperparameters (Gemini API)

The system interacts with Google Gemini via a REST API, where hyperparameters are configured as part of the request payload to control generation behavior.

Parameter	Value	Description
temperature	0.4	Controls randomness. Lower = more focused and deterministic output.
top_p	0.95	Limits diversity by selecting tokens from the top probabilities that sum to 95%.
top_k	40	Considers the top 40 tokens by probability before sampling.
mime_type	"image/jpeg"	Specifies the image content type sent to the model.
generation_config	Custom dictionary	Part of the API request to tweak model behavior.

These values were selected to prioritize accuracy and relevance in nutrition-related outputs rather than creative variance.

13.3 Config Files

Flask App Configuration (from app.py)

```
app = Flask(__name__)

app.secret_key = 'your_secret_key'  # Use environment variable in production

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///users.db'

app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

app.config['UPLOAD_FOLDER'] = 'uploads'

app.config['MAX_CONTENT_LENGTH'] = 10 * 1024 * 1024  # Max upload size: 10 MB
```

Gemini API Endpoint

```
API_URL = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent"

API_KEY = "your_api_key_here"  # Secured in a real-world environment
```

Note: Sensitive keys should be stored in environment variables or a .env file and loaded using python-dotenv for security.

Allowed File Types

```
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}
```