**NutriAssist – Produce diet plans**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science**

**School of Engineering and**

**Sciences**

Submitted by

**V. Bhagya Satya Sri (AP22110010083)**

**A picture containing text

Description automatically generated**

Under the Guidance of

**Syed Arshad**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**[April, 2025]**

**1. Introduction**

NutriAssist is a personalized nutrition management web application designed to help individuals achieve healthier lifestyles. By analyzing users' health goals, fitness parameters, and dietary preferences, NutriAssist generates customized meal plans and tracks daily intake. Users can also log their meals, monitor progress, and consult certified nutritionists for expert advice. The platform includes an admin panel for managing user accounts, nutritionist profiles, and generating progress reports, making NutriAssist a complete solution for promoting better eating habits and overall well-being.

**2. Project Overview**

**Purpose:**

NutriAssist is a powerful web application built to help users maintain healthy lifestyles by providing tailored nutrition advice based on their health goals, preferences, and medical details. This application targets individuals who are looking to improve their dietary habits and overall health through personalized meal plans that are generated by analyzing their age, weight, fitness goals, and dietary preferences. Moreover, the app includes the ability for users to log meals, track their daily intake, and consult certified nutritionists for professional guidance. The application also comes with an admin panel, where administrators can manage user accounts, nutritionist profiles, and generate detailed reports on users’ progress. The underlying goal is to support individuals in making healthier food choices that align with their personalized needs, and ultimately, improve their well-being through a structured meal plan system.

**Goals:**

* Personalized Diet Plans: The app provides users with meal plans specifically tailored to their health conditions and dietary preferences. These plans are designed to help users achieve their individual health goals such as weight loss, muscle gain, or maintaining a balanced diet.
* Meal Logging: Users can log their daily meals to track their calorie intake and monitor their nutritional values. This feature allows users to get a better understanding of their eating habits and areas where they might need to improve.
* Nutritionist Consultation: Through NutriAssist, users can communicate with certified nutritionists for more detailed and customized advice. This adds a professional layer to the app’s health recommendations.
* Admin Dashboard: The admin dashboard enables administrators to manage various aspects of the app, such as adding and editing nutritionist profiles, reviewing user health details, and generating comprehensive reports on users’ progress.
* Authentication: NutriAssist employs JWT (JSON Web Tokens) authentication, which ensures secure login and access control for both users and admins. The authentication mechanism ensures that only authorized individuals can access specific features of the application.

**3. Architecture**

**Frontend (React):**

The frontend of NutriAssist is built using React, a widely used JavaScript library that facilitates the creation of interactive user interfaces. React provides a component-based architecture, which means the application is built out of modular, reusable components. This architecture enhances maintainability and scalability of the application as different features (e.g., logging meals, viewing diet plans) are encapsulated into independent components that can be easily modified or replaced. React’s declarative nature simplifies the development process by making the UI predictable and easy to debug. React Router is integrated to enable smooth navigation between different pages like the dashboard, diet plan page, and consultation page without the need for full-page reloads.

The app also employs React Context API to manage global states like user authentication and roles (admin or user). This makes it easier to share state across components, especially when dealing with complex authentication flows and user-specific data. For styling, CSS Modules or Styled-components are used to scope CSS to individual components, which avoids potential issues with global CSS styles and keeps the UI consistent across different parts of the application. Axios, a promise-based HTTP client, is used for making API requests to the backend.

**Backend (Node.js + Express):**

The backend of NutriAssist is powered by Node.js, a runtime environment built on Chrome’s V8 JavaScript engine. Node.js is ideal for building fast and scalable applications, and it is especially well-suited for handling asynchronous I/O operations, which are common when dealing with web servers. The backend application leverages Express.js, a lightweight framework for building RESTful APIs. Express.js simplifies routing, request handling, and middleware integration, which enhances the overall structure and readability of the application’s backend code.

Node.js and Express work together to create a flexible, scalable, and fast backend architecture for NutriAssist. The backend handles requests for various operations, such as user registration, login, diet plan generation, meal logging, and user profile management. JWT Authentication is implemented to secure sensitive routes and ensure that only authorized users and admins can access specific features. When users log in, the backend generates a JWT, which is sent to the frontend and stored locally. This token is used to authenticate subsequent requests by the user.

**Database (MongoDB):**

For data storage, NutriAssist uses MongoDB, a database that provides flexibility in storing unstructured data. MongoDB is well-suited for the application because it allows for schema-less data, meaning the structure of the data can evolve over time without the need for migrations. This is particularly beneficial in a dynamic environment like NutriAssist, where user profiles, diet plans, and health details might change based on various factors.

The database is organized into collections that represent different entities in the app, such as Users, Admins, Diet Plans, and Meals. Each collection is designed to store documents with specific attributes. For instance:

* Users store personal details, health goals, and meal logs.
* Diet Plans are linked to user profiles and contain customized meal recommendations.
* Meals store nutritional information about each meal (calories, proteins, etc.).

MongoDB’s document-oriented structure makes it easier to handle nested data and relationships between collections. Additionally, NutriAssist leverages Mongoose, an Object Data Modeling (ODM) library for MongoDB, which provides a higher-level abstraction for interacting with the database. Mongoose enables validation, query building, and relationship management between collections, ensuring data integrity and seamless interactions.

**4. Setup Instructions**

**Prerequisites:**

Before setting up the NutriAssist application, ensure you have the following software installed:

* Node.js (v14 or above): The runtime environment required to run the backend server.
* MongoDB: You can either set up a local MongoDB instance or use a cloud-based service like MongoDB Atlas for database management.
* Git: For cloning the repository.

**Installation:**

1. Install Backend Dependencies:

Navigate to the server directory:

* + cd server

Install the backend dependencies:

* + npm install

1. Install Frontend Dependencies:

Navigate to the client directory:

* + cd client

Install the frontend dependencies:

* + npm install

1. Configure Environment Variables: Create .env files in both the client and server directories:
   * Server .env:
     + MONGO\_URI: mongodb+srv://myAtlasDBUser:2005@myatlasclusteredu.11maf.mongodb.net/nutritionassistant?retryWrites=true&w=majority&appName=myAtlasClusterEDU
     + JWT\_SECRET: Secret key for signing JWT tokens.
   * Client .env:
     + REACT\_APP\_API\_URL: The API base URL (e.g., http://localhost:5000).
2. Run the Application:

To run the frontend locally, navigate to the client directory and use the following command:

* + npm start

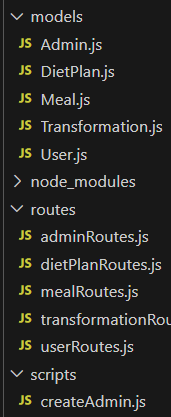
To start the backend locally, navigate to the server directory and use the following command:

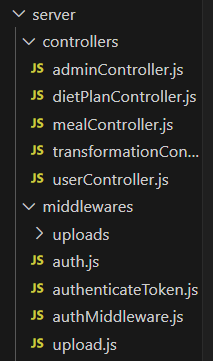
* + npm start

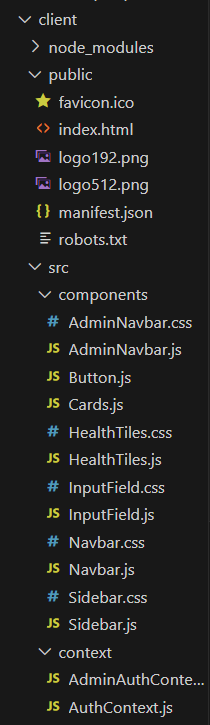
**5. Folder Structure**

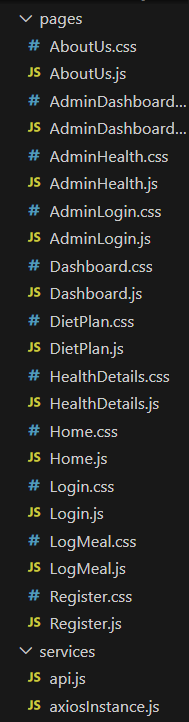
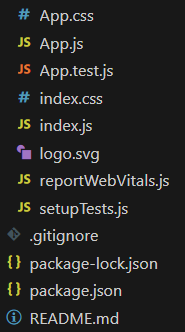
The project is divided into two main directories: client (for the frontend) and server (for the backend). Below is an overview of the folder structure for both parts of the application.

**Backend (Server):**

****



**Frontend (Client):**

****

**6. Running the Application**

Once you have the application set up, you can run the frontend and backend locally:

* Frontend:
  1. Navigate to the client directory and run:
  2. npm start
  3. This will start the React app on http://localhost:3000.
* Backend:
  1. Navigate to the server directory and run:
  2. npm start
  3. This will start the Express server on http://localhost:5000.

**7. API Documentation**

API Documentation for NutriAssist

The following documentation outlines the API endpoints used in the NutriAssist application, which provides personalized nutrition advice and meal planning. These endpoints manage user profiles, health details, meals, and diet plans, with JWT-based authentication.

**1. User Endpoints**

**POST /api/users/register – Register a New User**

**Description: Registers a new user on the platform.**

**Request Body:**

{

"username": "john\_doe",

"email": "john.doe@example.com",

"password": "strongPassword123"

}

**Response:**

{

"message": "User registered successfully",

"user": {

"id": "user123",

"username": "john\_doe",

"email": "john.doe@example.com"

}

}

**POST /api/users/login – User Login**

**Description: Allows a user to log into the application.**

**Request Body:**

{

"email": "john.doe@example.com",

"password": "strongPassword123"

}

**Response:**

{

"message": "Login successful",

"token": "jwt.token.here",

"user": {

"id": "user123",

"username": "john\_doe"

}

}

**GET /api/users/profile – Get User Profile**

**Description: Fetches the profile information of the logged-in user.**

**Headers:**

**Authorization: Bearer <JWT\_TOKEN>**

**Response:**

{

"user": {

"id": "user123",

"username": "john\_doe",

"email": "john.doe@example.com"

}

}

**Admin Endpoints**

**POST /api/admins/login – Admin Login**

**Description: Admin login using email and password.**

**Request Body:**

{

"email": "admin@example.com",

"password": "adminPassword123"

}

**Response:**

{

"message": "Login successful",

"token": "jwt.admin.token.here",

"admin": {

"id": "admin123",

"email": "admin@example.com"

}

}

**GET /api/admins/users – List All Users**

**Description: Admin can view a list of all users.**

**Headers:**

**Authorization: Bearer <JWT\_TOKEN>**

**Response:**

{

"users": [

{

"id": "user123",

"username": "john\_doe",

"email": "john.doe@example.com"

},

{

"id": "user124",

"username": "jane\_smith",

"email": "jane.smith@example.com"

}

]

}

**Health Details Endpoints**

**GET /api/healthdetails/:userId – Get Health Details of a User**

**Description: Fetches health details for a specific user.**

**Response:**

{

"healthDetails": {

"userId": "user123",

"height": "5'8\"",

"weight": "75kg",

"BMI": 28,

"goal": "weight loss"

}

}

**POST /api/healthdetails/:userId – Create or Update Health Details**

**Description: Allows the user to update or provide their health details.**

**Request Body:**

{

"height": "5'9\"",

"weight": "78kg",

"BMI": 28,

"goal": "muscle gain"

}

**Response:**

{

"message": "Health details updated successfully",

"healthDetails": {

"userId": "user123",

"height": "5'9\"",

"weight": "78kg",

"BMI": 28,

"goal": "muscle gain"

}

}

**Meal Endpoints**

**POST /api/meals/create – Create a New Meal**

**Description: Allows a user or admin to create a new meal for a user’s meal plan.**

**Request Body:**

{

"userId": "user123",

"mealType": "lunch",

"mealName": "Grilled Chicken Salad",

"day": "Monday"

}

**Response:**

{

"message": "Meal created successfully",

"meal": {

"userId": "user123",

"mealType": "lunch",

"mealName": "Grilled Chicken Salad",

"day": "Monday"

}

}

**GET /api/meals/:userId – Get All Meals for a User**

**Description: Fetches all the meals associated with a user’s meal plan.**

**Response:**

{

"meals": [

{

"mealType": "lunch",

"mealName": "Grilled Chicken Salad",

"day": "Monday"

},

{

"mealType": "dinner",

"mealName": "Baked Salmon",

"day": "Monday"

}

]

}

**Diet Plan Endpoints**

**POST /api/dietplan/create – Create a New Diet Plan**

**Description: Creates a personalized diet plan for a user based on their health details.**

**Request Body:**

{

"userId": "user123",

"mealPlan": [

{

"mealType": "lunch",

"mealName": "Grilled Chicken Salad",

"day": "Monday"

},

{

"mealType": "dinner",

"mealName": "Baked Salmon",

"day": "Monday"

}

]

}

**Response:**

{

"message": "Diet plan created successfully",

"dietPlan": {

"userId": "user123",

"mealPlan": [

{

"mealType": "lunch",

"mealName": "Grilled Chicken Salad",

"day": "Monday"

},

{

"mealType": "dinner",

"mealName": "Baked Salmon",

"day": "Monday"

}

]

}

}

**GET /api/dietplan/:userId – Get Diet Plan for a User**

**Description: Retrieves the diet plan for a user based on their health details and preferences.**

**Response:**

{

"dietPlan": {

"userId": "user123",

"mealPlan": [

{

"mealType": "lunch",

"mealName": "Grilled Chicken Salad",

"day": "Monday"

},

{

"mealType": "dinner",

"mealName": "Baked Salmon",

"day": "Monday"

}

]

}

}

**8. Authentication**

JWT Token Authentication

* Description: This application uses JWT (JSON Web Tokens) for user and admin authentication. Tokens are used to verify the identity of the user or admin in subsequent API requests.
* Flow:
  + A user or admin logs in by sending their credentials via the /login endpoint.
  + Upon successful login, the server generates a JWT and sends it back to the client.
  + The client includes the JWT in the Authorization header for all protected routes like /users/profile, /admin/users, etc.

Example of Authorization Header:

Authorization: Bearer <JWT\_TOKEN>

**9. User Interface**

**Admin Dashboard:**

* View a list of all users, including their health details, meal logs, and diet plans.
* Create, update, and delete diet plans.
* View user progress reports and health insights.

**User Dashboard:**

* Access a personalized diet plan based on health details like weight, age, and fitness goals.
* Log meals and track nutritional intake.
* Communicate with nutritionists for additional advice and guidance.

**10. Testing**

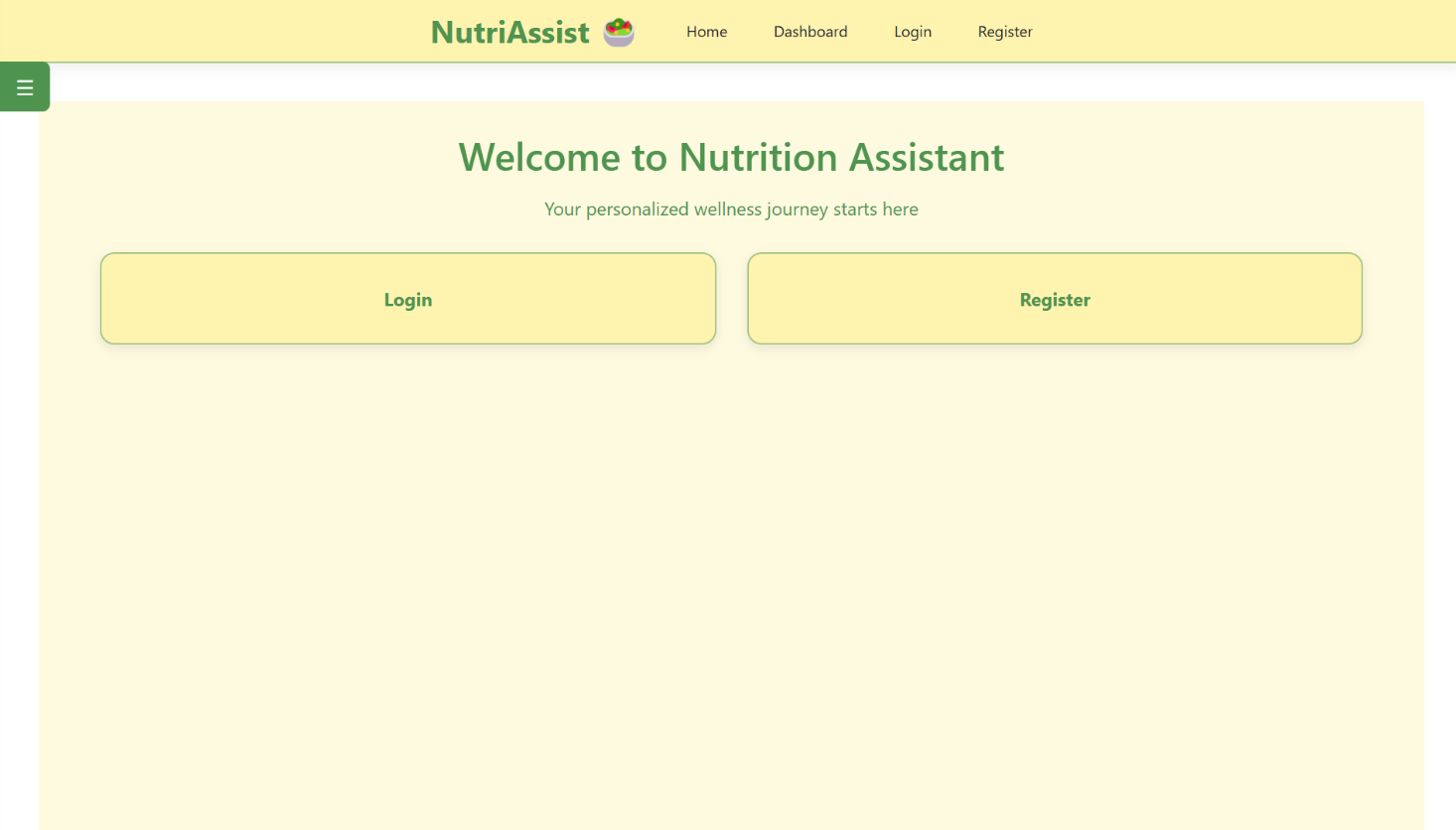
**Manual Testing:**

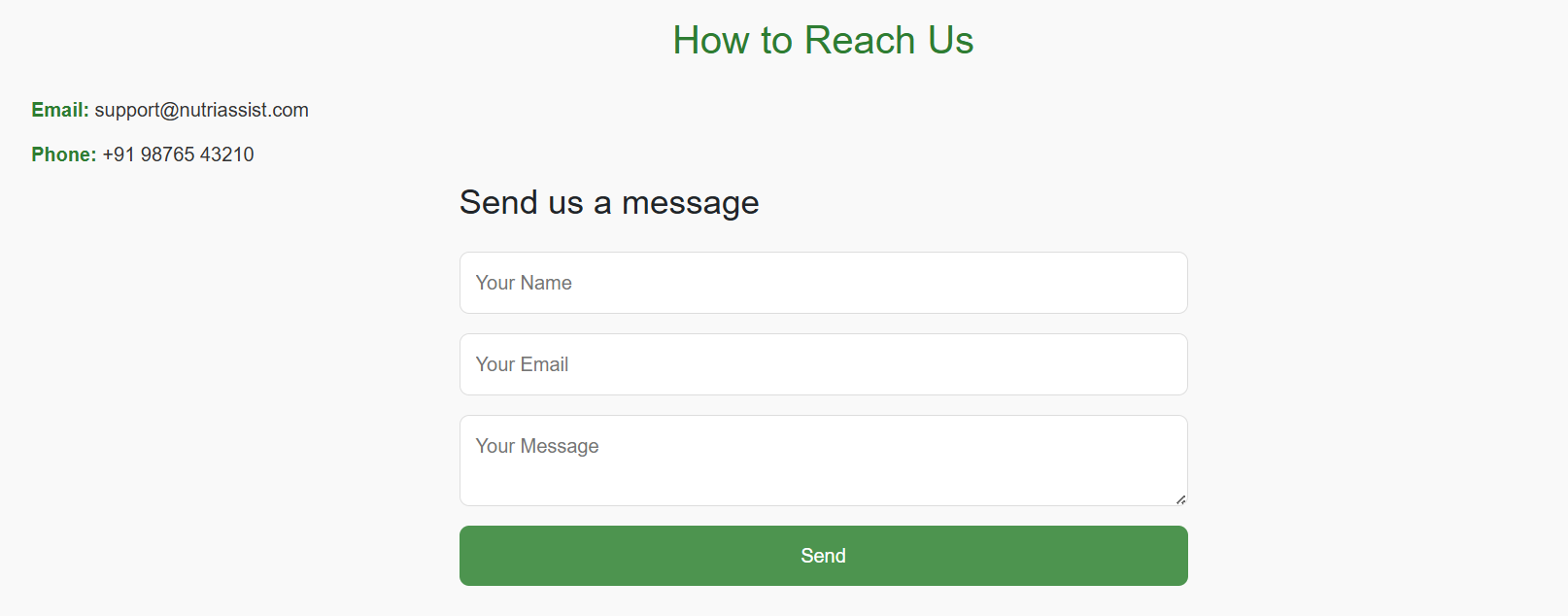
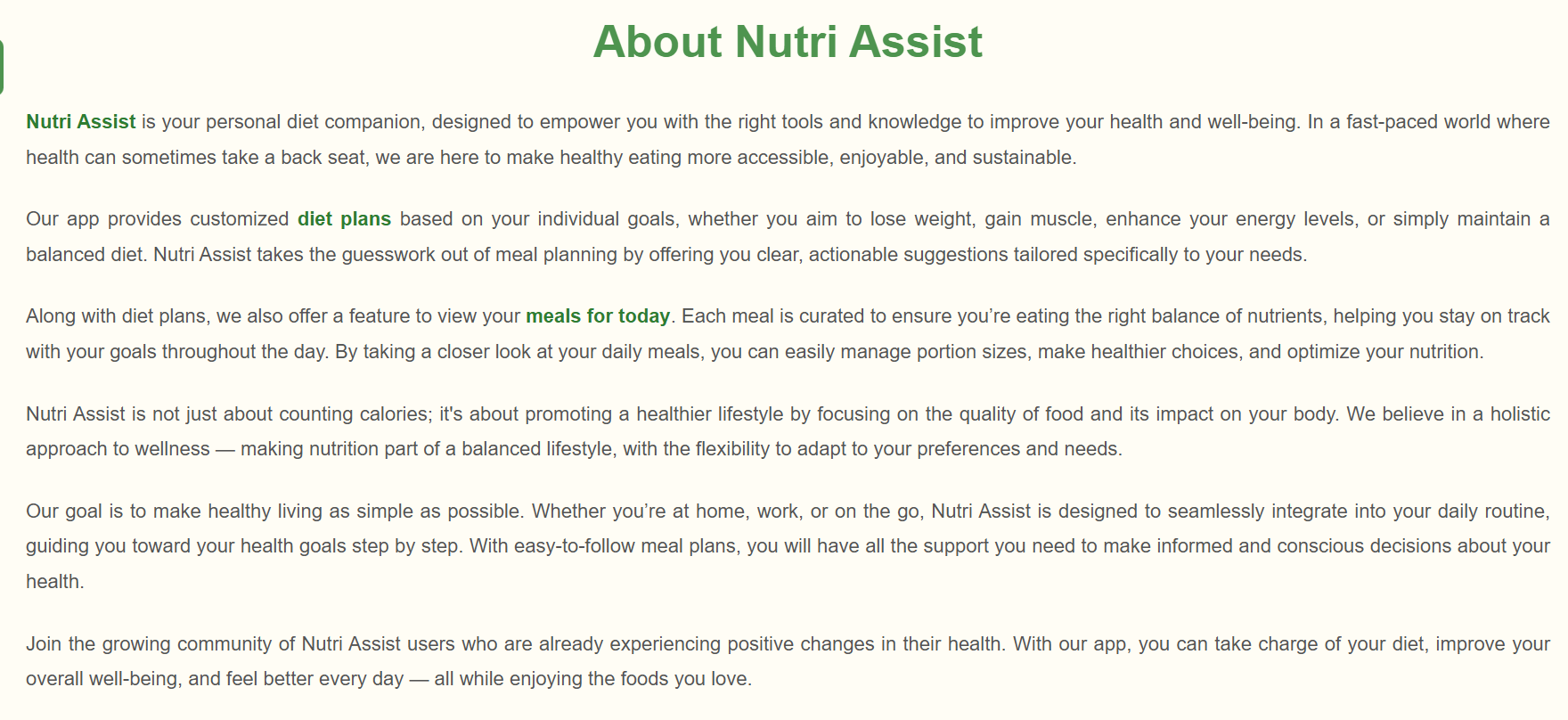
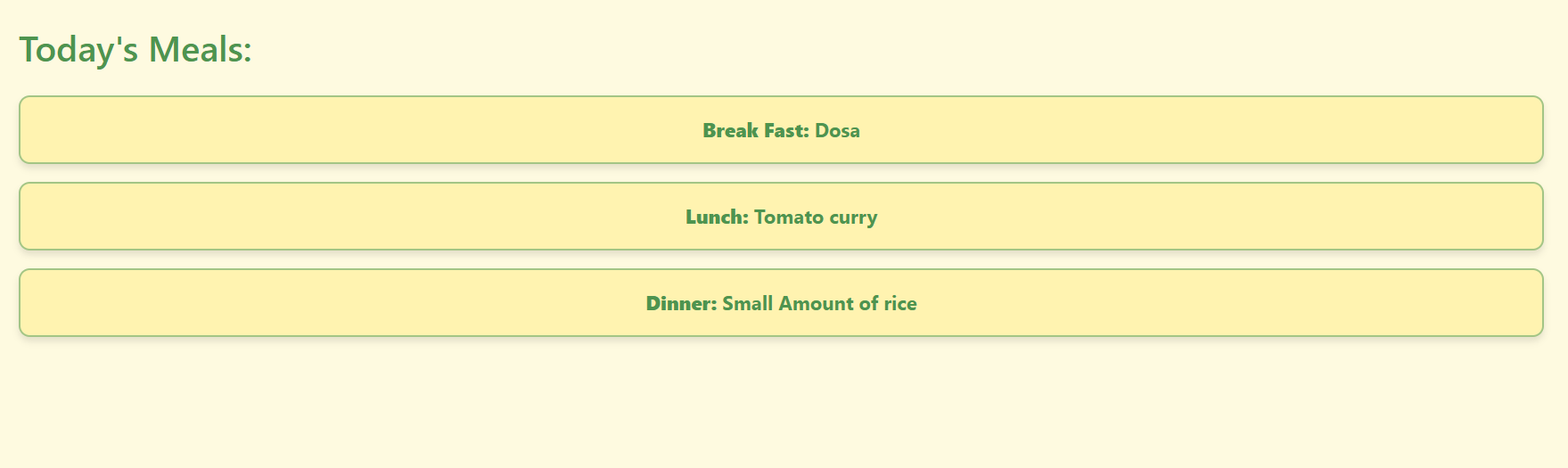
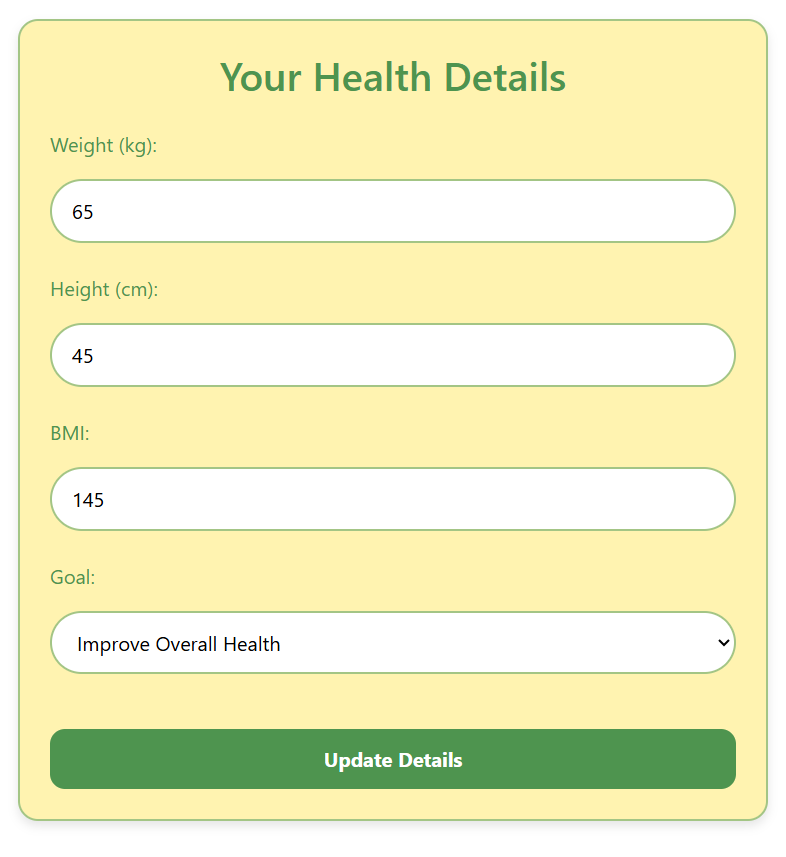
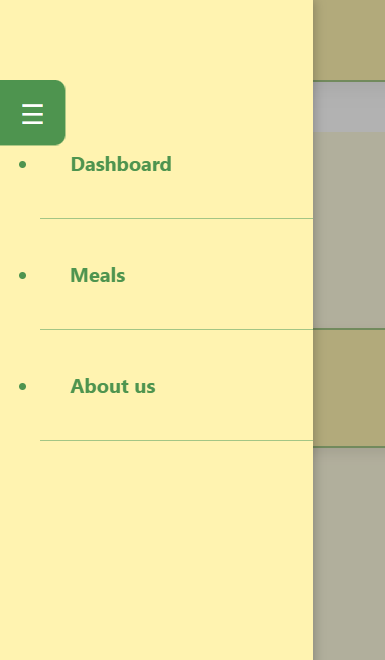
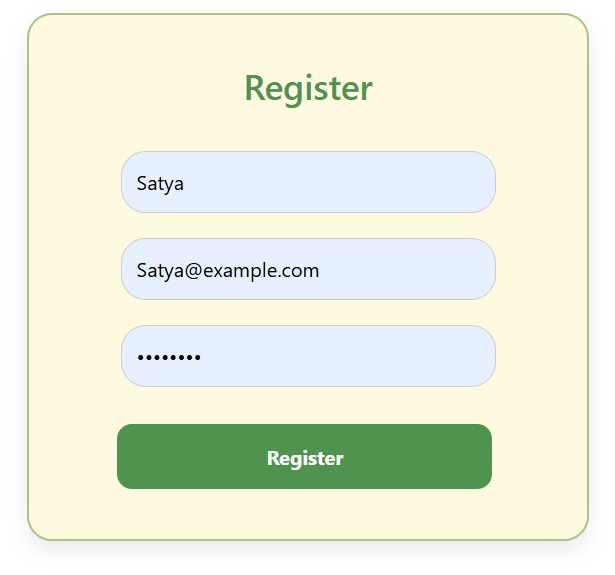
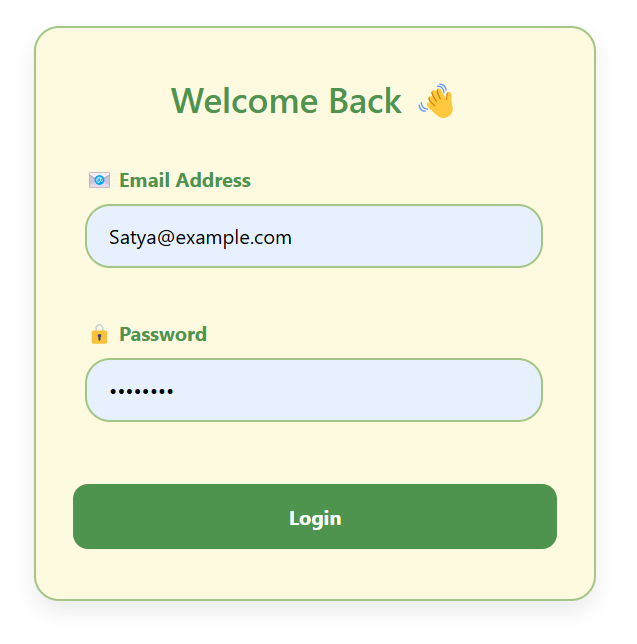
All features are manually tested for functionality, usability, and performance. Test cases include logging in, meal logging, viewing diet plans.

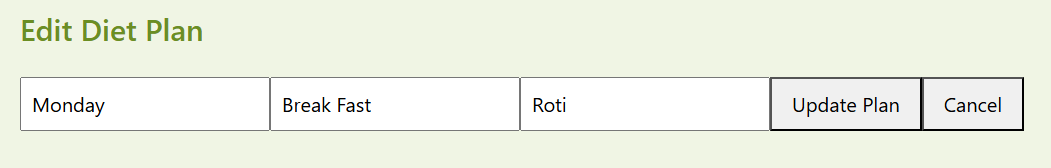
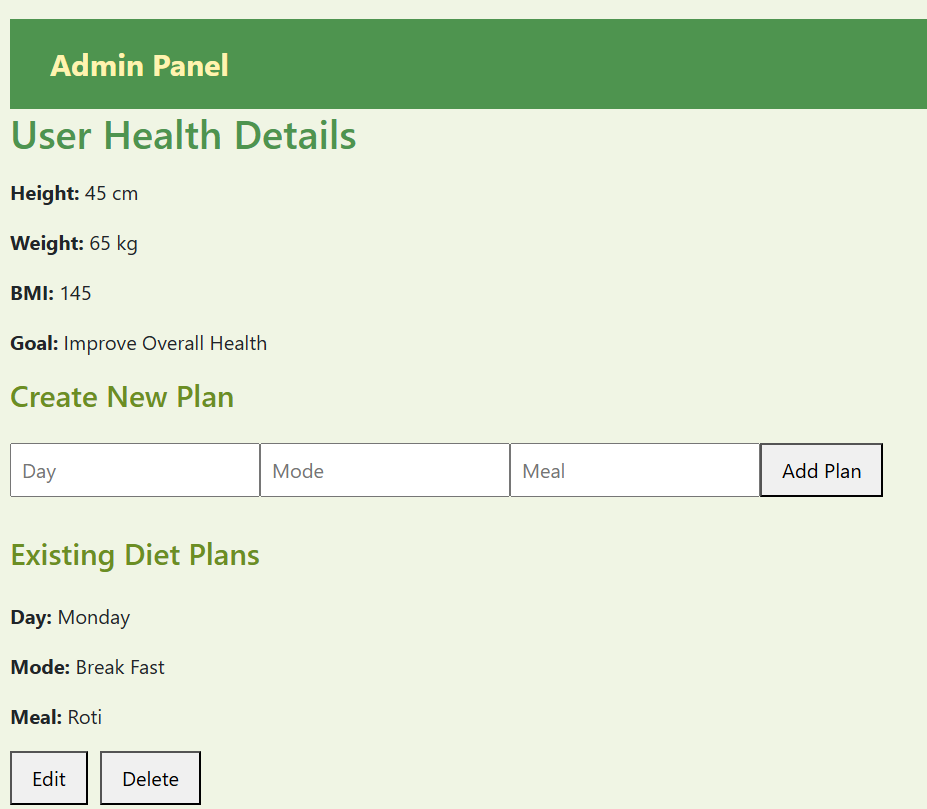
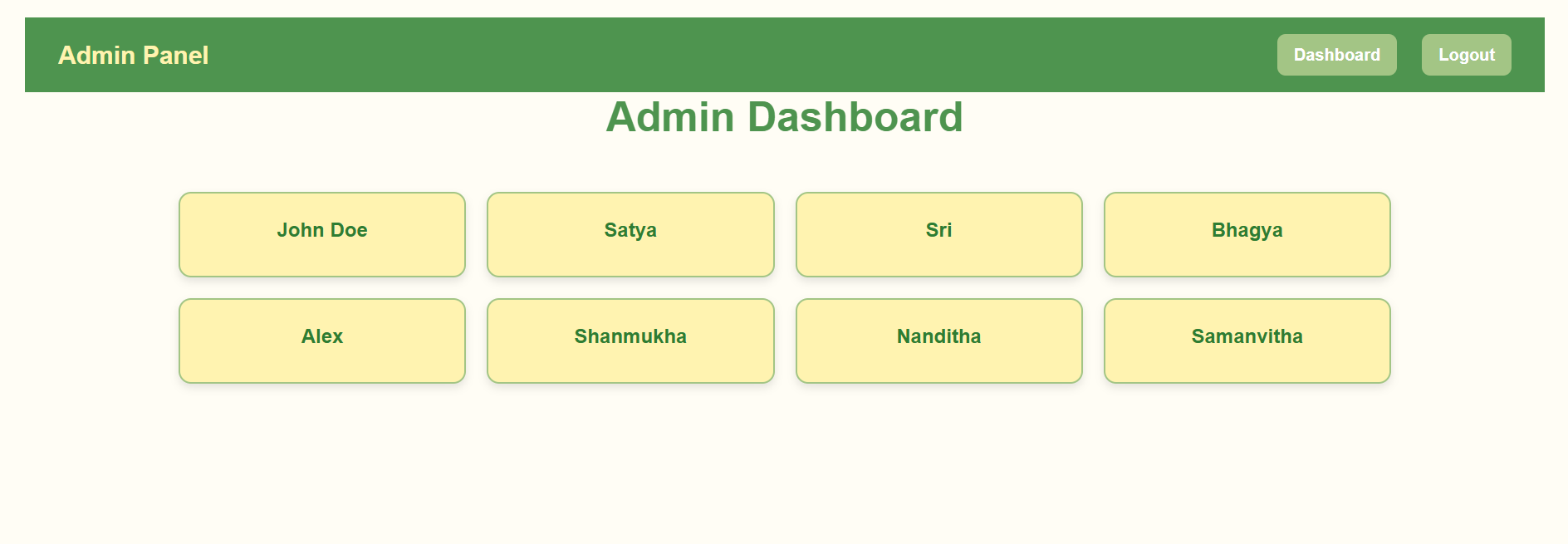
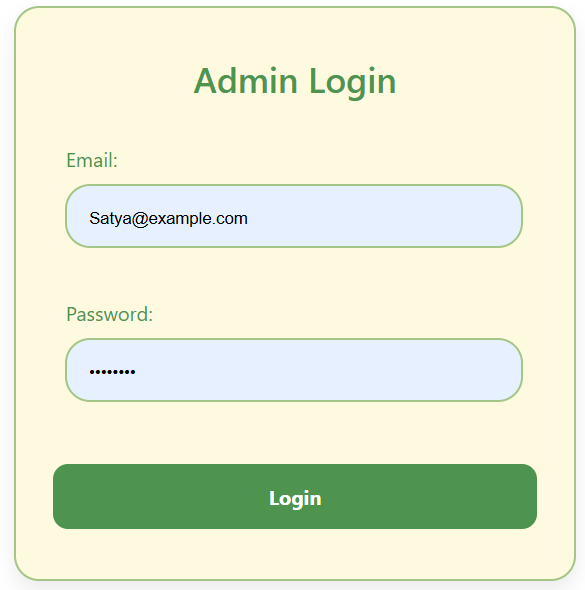
Unit Testing:

* Jest is used for frontend component testing.
* Mocha and Chai are used for backend API testing.

**11. Screenshots and Demo**

**Screenshots:**

****

****

**Video Links:**

**1.Frontend Link:**

**https://drive.google.com/file/d/1sB7NxeOO0XX9wRPjEE2HsCO1fjuizY-t/view?usp=drive\_link**

**2.Backend Link:**

**https://drive.google.com/file/d/1MngC0cNOGWSCqnfGEvGfiwr\_9DVPf09M/view?usp=sharing**

**3. DEMO Link : https://drive.google.com/file/d/1RD9I732IyjyyTf0DIO7md\_syB7URfH5C/view?usp=sharing**

**12. Known Issues**

* Minor glitches when using the app on mobile screens.
* Delay in fetching personalized diet plans.

**13. Future Enhancements**

* AI-Driven Meal Recommendations: Leverage AI algorithms to further personalize meal plans based on users’ past behavior, preferences, and health goals.
* Real-time Chat with Nutritionists: Add live chat functionality for real-time interactions between users and nutritionists.

\*\*\*THANK YOU\*\*\*