Smart Nutrition Server Software

1. Introduction

This project aims to create a system that helps generate personalized diet plans for users based on their individual information and feedback. The system handles client requests, processes data, and stores user information and feedback for future reference. While some components are fully implemented, others are designed for future enhancements.

2. Design Overview

The project consists of several key components that interact to achieve the desired functionality. These components include classes for handling food items, categories, diseases, patients, and diet plans, as well as utility functions for data management and a message handler for processing client requests. When the server receives a diet plan request from a user, it detaches a thread to process the user's information and generate the plan. A mutex is used to protect the shared resource, which includes users_data.json and its equivalent data structure in the code.

3. Class Descriptions

FoodItem

- **Attributes:**
    - string name
    - int calories
    - float carbs
    - float protein
    - float fat
    - vector<string> nutrients
- **Purpose:**
    - Stores information about each food item.
    - Intended for future diet plan generation, though the logic for generating diet plans is not yet implemented.

FileUtils

- **Functions:**
    - saveDataToFile
    - loadDataFromFile
- **Purpose:**
    - Logs data in external files to ensure data persistence.

FoodCategory

- **Attributes:**
  - string name
  - vector<FoodItem> items
- **Purpose:**
  - Categorizes food items to ensure a balanced diet.

Disease

- **Attributes:**
  - string name
  - string description
- **Purpose:**
  - Holds disease information for future diet plan customization.
  - Designed for further improvement of diet plan generation based on specific diseases.

Patient

- **Attributes:**
  - string userId
  - string name
  - int age
  - string gender
  - float height
  - float weight
  - vector<Disease> diseases
  - float feedback
- **Methods:**
  - Getters and setters for all attributes.
  - float getBMI()
- **Purpose:**
  - Holds patient information, feedback, and BMI.
  - Aids in generating diet plans by providing relevant user data.

DataInit

- **Purpose:**
  - Initializes the food_data.json file.
  - Populates data structures with a set of food categories (carbs, protein, sweets, veggies).

DietPlan

- **Purpose:**
  - Generates diet plans based on the information provided by other classes.
  - Note: The logic for diet plan generation is still under development.

MessageHandler

- **Purpose:**
  - Handles client requests by processing data, creating diet plans, and logging user data.
  - Manages threading to handle multiple client requests concurrently.

4. Functionality and Usage

Initialization

1. **Data Initialization:**
   - Run the DataInit component to initialize the food_data.json file with categories and food items.

Handling Requests

2. **Client Request Handling:**
   - The MessageHandler processes incoming client requests.
   - Upon receiving a request, it detaches a thread to handle the request, processes the user data, generates a diet plan, and sends it back along with a user ID.
   - The client can then send feedback, which is stored in the user's record.

Providing Feedback

3. **Feedback Processing:**
   - The client sends a feedback score along with the user ID.
   - The feedback is updated in the user's record and logged in the users_data.json file.

5. Example Workflows

Example Scenario

1. **User Registration:**
   - User sends their personal information (name, age, gender, height, weight, diseases) to the server.
   - The server processes this data, generates a diet plan, and sends it back along with a user ID.
2. **Providing Feedback:**
   - User sends feedback on the diet plan using their user ID.
   - The server updates the feedback score in the user's record.

6. Future Improvements

Implement the logic for generating diet plans based on the information stored in FoodItem and Disease.

Enhance the Disease class with more comprehensive descriptions and their impacts on diet plans.

7. Technical Details

- Dependancies

  C++17 Standard Library

  Pistache HTTP library

  nlohmann/json for JSON handling

- Build and Run

  ➔ g++ -std=c++17 -o nutrition_server main.cpp DataInit.cpp FileUtils.cpp FoodCategory.cpp FoodItem.cpp message_handler.cpp Patient.cpp DietPlan.cpp Disease.cpp -lpistache -lboost_system -ljsoncpp

  ➔ ./nutrition_server

8. Appendices

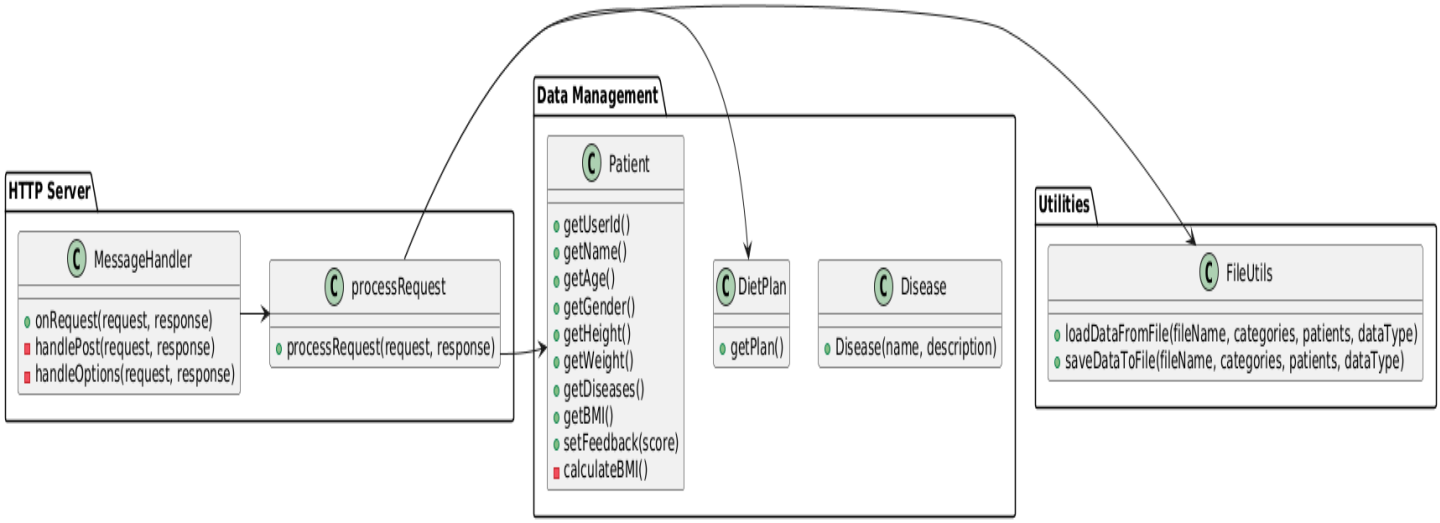
Project Repository

https://github.com/Mohamed-Badr-Saad/nutrition-system

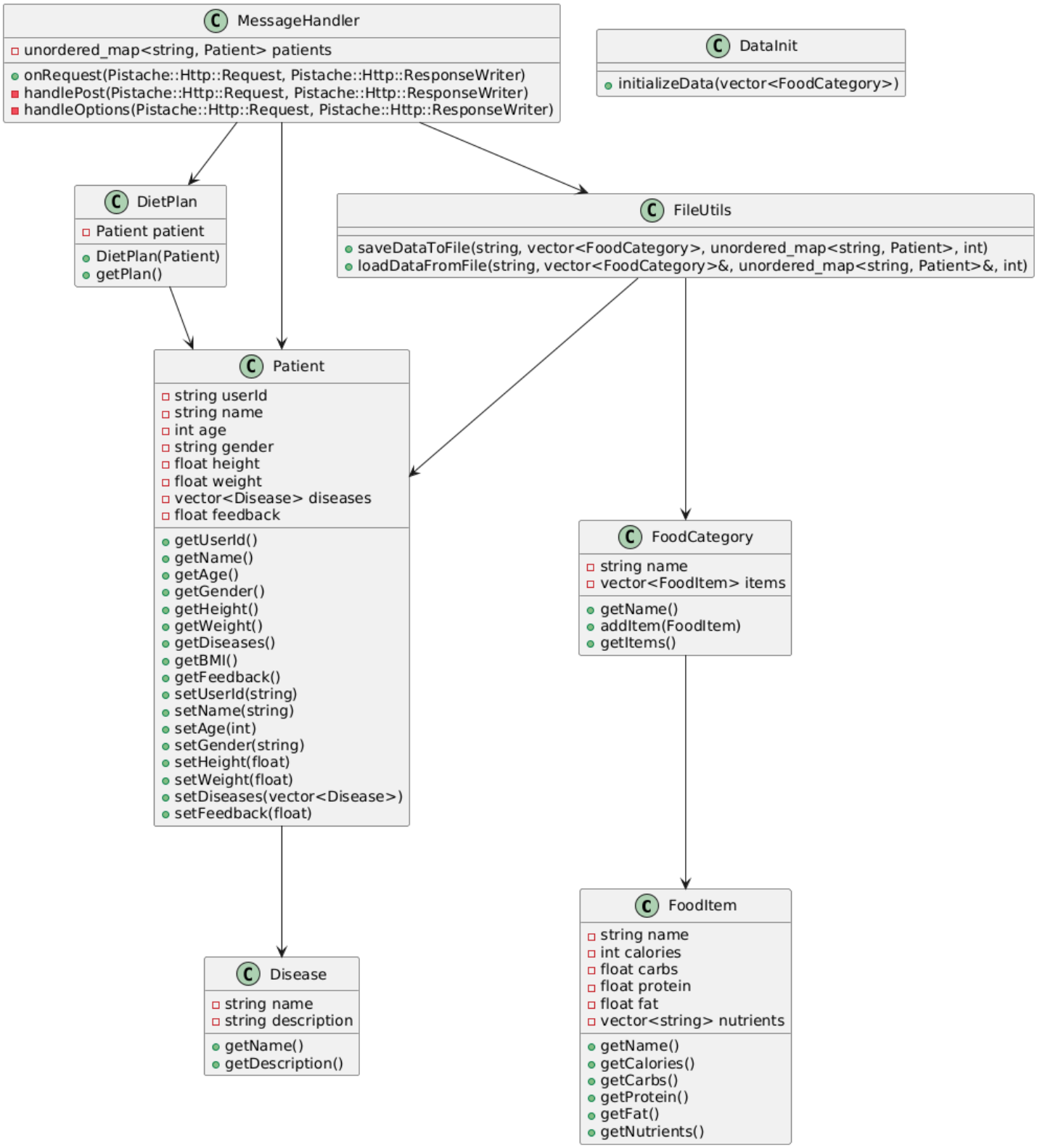
https://drive.google.com/drive/folders/1pO28HcND6EyDSKmOHKHZWX8-qYqCyQF_?usp=drive_link

9.Diagrams

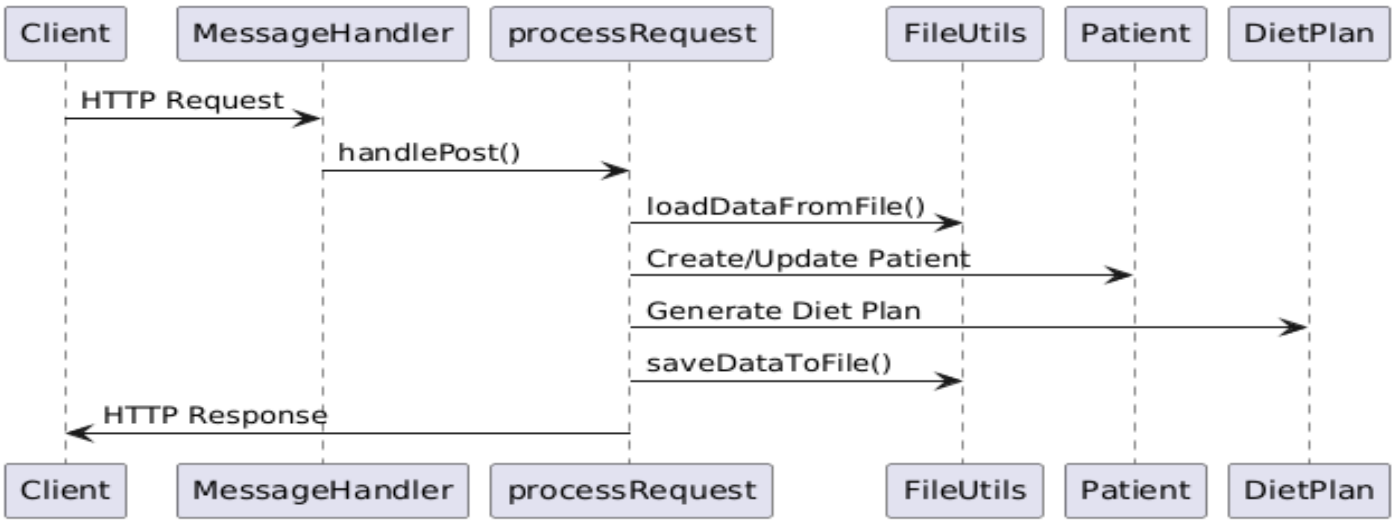
High-Level System Overview



Classes Diagram



Data Flow (Sequence Diagram)



Users_Data log file example

users_data.json

pistache_server > {} users_data.json > [ ] patients > {} 0 > # feedback


```
{
    "patients": [
        {
            "age": 30,
            "dietPlan": "{\n    \"Friday\": {\n        \"Breakfast\": [\n            \"Blueberry\",\n            \"Chicken Breast\"\n        ],\n        \"Dinner\":
            "diseases": [
                "headache"
            ],
            "feedback": 6.0,
            "gender": "Female",
            "height": 168.0,
            "name": "Sara",
            "userId": "user189841",
            "weight": 65.0
        },
        {
            "age": 24,
            "dietPlan": "{\n    \"Friday\": {\n        \"Breakfast\": [\n            \"Broccoli\",\n            \"Salmon\"\n        ],\n        \"Dinner\": [\n
            "diseases": [
                "None"
            ],
            "feedback": 6.0,
            "gender": "Male",
            "height": 178.0,
            "name": "Mohamed Badr",
            "userId": "user103954",
            "weight": 75.0
        }
```

Patient Software

1. Introduction

The client-side application is a web-based interface built using React.js and Bootstrap. It allows users to enter their personal information, request a diet plan from the server, view the generated diet plan, and provide feedback on the plan. The application handles user interactions, communicates with the server via HTTP requests, and updates the UI accordingly.

2. Components Overview

The application consists of several React components that manage different parts of the user interface and functionality:

- App: The main component that orchestrates the overall flow.
- PatientForm: Collects user information.
- DietPlanDisplay: Displays the generated diet plan.
- FeedbackForm: Collects user feedback on the diet plan.

3. Component Descriptions

App

- State:
    - dietPlan (string): Holds the diet plan data.
    - userId (string): Holds the user ID.
    - feedback (number | null): Holds the feedback score.
- Functions:
    - handlePatientSubmit: Updates state with diet plan data and user ID.
    - handleFeedbackSubmit: Sends feedback to the server.
- Purpose:
    - Manages the overall application state and handles the main workflow.

PatientForm

- Props:
  - onSubmit: Function to handle form submission.
- State:
  - name, age, gender, height, weight, diseases: User input fields.
- Functions:
  - handleSubmit: Sends user information to the server and updates the parent component with the diet plan and user ID.
- Purpose:
  - Collects user information and sends it to the server for diet plan generation.

DietPlanDisplay

- Props:
  - dietPlan: The diet plan data to display.
- Purpose:
  - Displays the diet plan in a structured format.

FeedbackForm

- Props:
  - userId: The user ID.
  - onSubmit: Function to handle feedback submission.
- State:
  - feedbackScore: User input for feedback.
  - submissionStatus: Status of feedback submission (success or error).
- Functions:
  - handleSubmit: Sends feedback to the server and updates the parent component.
- Purpose:
  - Collects user feedback and sends it to the server.

4. Functionality and Usage

User Registration and Diet Plan Request

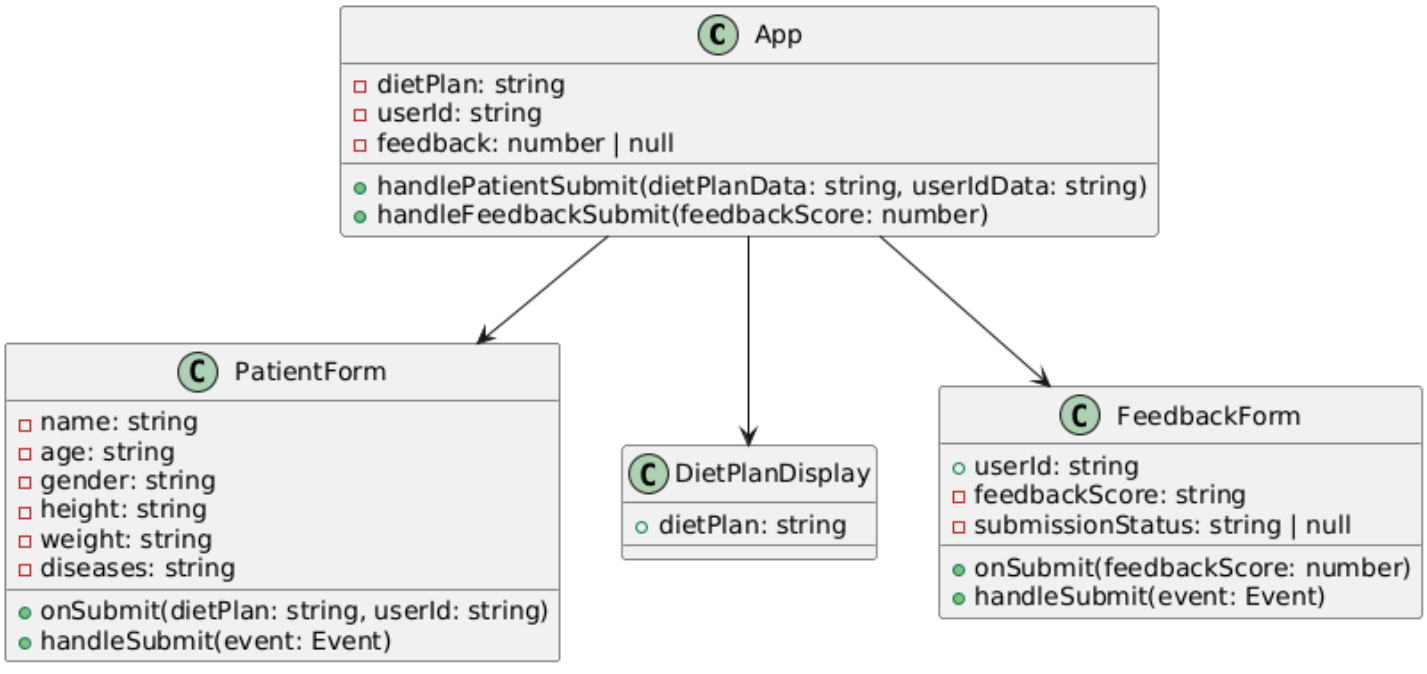
1. User enters personal information:
    - The user fills out the PatientForm with their details.
2. Form submission:
    - The PatientForm sends the data to the server.
    - The server processes the data and returns a diet plan and user ID.
3. Display diet plan:
    - The App component updates its state with the received diet plan and user ID.
    - The DietPlanDisplay component renders the diet plan.

Providing Feedback

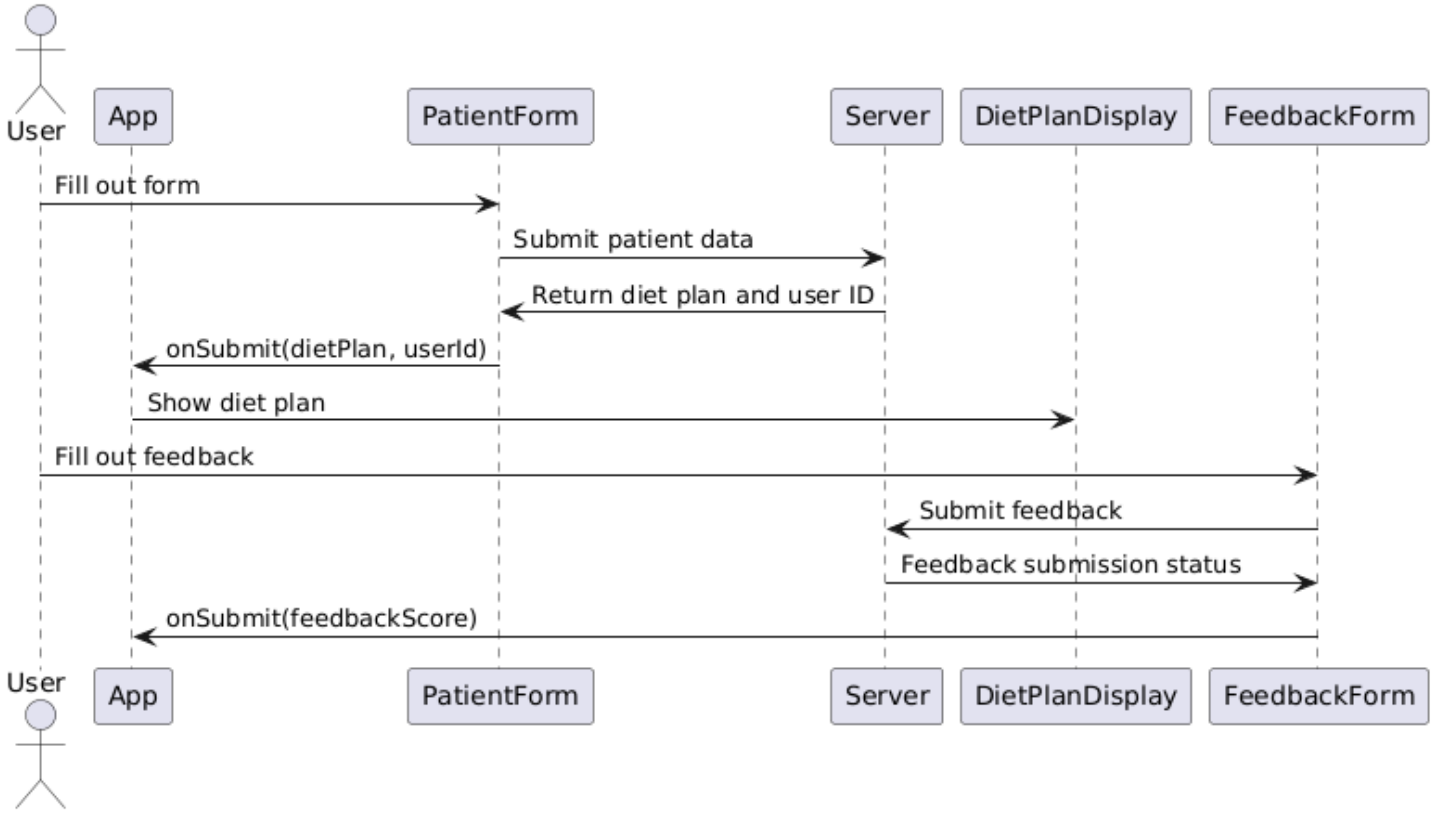
4. User enters feedback:
    - The user fills out the FeedbackForm.
5. Feedback submission:
    - The FeedbackForm sends the feedback to the server.
    - The server processes the feedback and updates the user record.

5. Diagrams

Component hierarchy



Sequence Diagram



6. Future Improvements

- Implement a more detailed diet plan structure and visualization.
- Add validation and better form controls for user inputs.
- Improve styling and responsiveness of the UI using more advanced Bootstrap features.

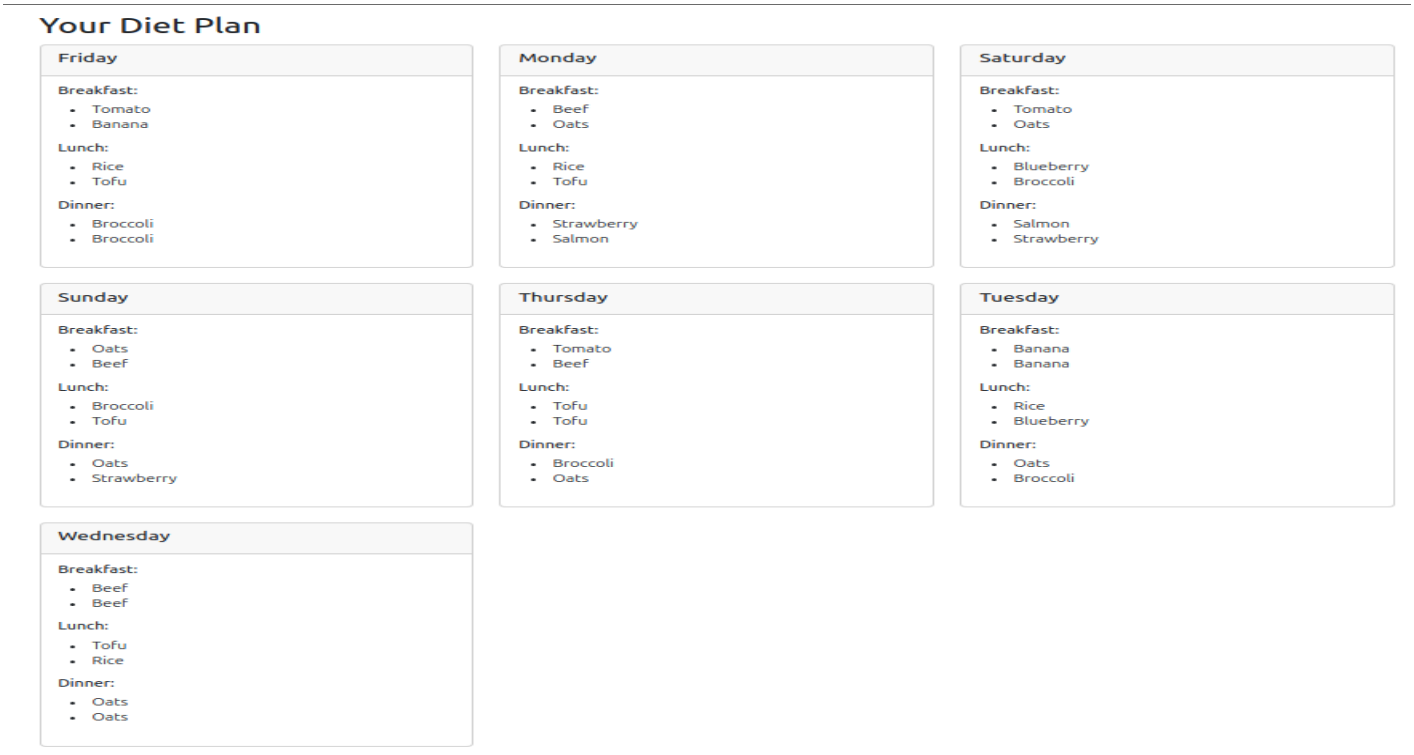
7. Technical Details

- Dependencies:
    - React.js
    - Bootstrap
- Setup and Run:
    - Npx create-react-app nutrition_system_client
    - npm install to install dependencies.
    - npm start.

8. Screenshots

Fill your Info

Getting Diet Plan



Submitting Feedback