System Architecture and Logic

Overview

The Weight Loss Coaching System combines advanced AI technologies, structured logic, and an intuitive user experience to deliver personalized health improvement plans. This document outlines the system's architecture, backend logic, and key workflows.

Table of Contents

1. System Architecture

The system follows a modular architecture, allowing for scalability and flexibility in implementing new features.

High-Level Architecture

1. Frontend: User interface for accessing personalized plans, tracking progress, and gamification.

• Built with React or similar frameworks for responsive design.

• Multi-language support (English and Spanish).

2. Backend: API-powered engine handling data processing, plan generation, and progress tracking.

• Built using Node.js and Express.js.

• Interfaces with OpenAI for AI-driven personalization.

3. Database: Storage for user data, progress photos, supplement links, and analytics.

• Recommended: MongoDB for flexibility or PostgreSQL for relational data.

4. Third-Party Integrations:

• OpenAI: Personalized content generation.

• Wearables (e.g., Fitbit, Apple Watch): Syncing user activity data.

• Supplement distributors for tracking purchases.

5. Hosting and Deployment:

• Deployed on Replit, Render, or AWS for scalability and persistent uptime.

2. Core Components

a. User Onboarding

• Collects user inputs via a conversational flow.

• Stores user preferences, activity levels, dietary restrictions, and supplement habits.

b. Plan Generation

• Uses OpenAI to generate personalized diet, activity, and supplement plans.

• Logic adapts to user feedback and changing goals.

c. Progress Tracking

• Includes weekly photo uploads, weight logging, and activity tracking.

• Provides visual feedback and progress reports.

d. Gamification

• Tracks streaks, rewards, and leaderboards.

• Encourages daily engagement and healthy competition.

e. Admin and Distributor Tools

• Analytics for monitoring user progress and supplement sales.

• Management tools for customizing plans and overseeing operations.

3. Backend Logic and Data Flow

1. Data Collection:

• Inputs from users are gathered via the frontend or API.

• Example: { preferences: "keto", activityLevel: "moderate", supplementation: "none" }.

2. Data Validation:

• Middleware validates incoming data for completeness and accuracy.

3. Plan Processing:

• Data is passed to the OpenAI API for personalized plan generation.

• Additional logic customizes outputs based on user roles (e.g., distributor-specific supplements).

4. Storage:

• Plans and progress data are stored securely in the database.

• Photos are uploaded to a cloud storage solution.

5. Feedback Loop:

• Users provide feedback, which triggers plan adjustments.

• Admins can manually override or refine plans.

6. API Responses:

• Returns structured JSON objects for frontend consumption.

• Example response for /api/generate-plan:

{

```
  "diet": "low carb",
  "activities": ["30 mins walking", "10 mins stretching"],
  "supplements": ["Omega-3 capsules"]
}
```

4. OpenAI Integration

Role of OpenAI

- Generates personalized plans based on user data.
- Supports dynamic feedback to refine plans over time.

Endpoint Usage

- Calls to OpenAI are structured as:

```
const response = await openai.Completion.create({
  model: "text-davinci-003",
  prompt: "Generate a keto-friendly diet plan...",
  max_tokens: 500
});
```

Error Handling

- Logs errors when OpenAI fails to generate outputs.

• Returns fallback responses to ensure user continuity.

5. Bolt.new Configuration

Purpose

• Provides a guided setup for collecting user inputs efficiently.

Key Logic

1. Question Flows:

• Questions are structured to align with OpenAI input requirements (e.g., dietary preferences, activity levels).

2. Data Mapping:

• Maps user responses to JSON objects for API processing.

Example Flow

• Question: “What is your dietary preference?”

• User Answer: “Keto.”

• JSON Output: { "preferences": "keto" }.

6. Plan Adjustment Workflow

Weekly Updates

• Plans are revised every Sunday based on:

• Weight and activity progress.

• Feedback provided by users.

Admin Adjustments

• Admins can override plans or add supplemental recommendations.

Flowchart

1. User provides feedback → Backend processes changes → OpenAI generates a new plan → Updated plan is stored in the database.

7. Security Considerations

Data Encryption

• Use HTTPS for all API communications.

• Encrypt sensitive data (e.g., user photos, API keys).

Authentication

• Implement role-based access control (RBAC) for users, admins, and distributors.

• Require API keys for all external integrations.

Error Logging

• Log errors securely without exposing sensitive information.

• Use monitoring tools like Sentry or LogRocket for real-time alerts.

This comprehensive architecture and logic document ensures all stakeholders understand the system's inner workings and future scalability.