**Technical Specification Document**

**1. Overview**

The **AI Meal Planner** is a web-based application that provides personalized meal plans using **AI** based on user preferences, dietary restrictions, and nutritional goals. This document describes the architecture, technologies, tools, and integrations required to build the platform.

**2. Tech Stack Summary**

* **Frontend**: Vue.js (for UI/UX)
* **Backend**: .NET 8 (for API and business logic)
* **Database**: PostgreSQL (for data storage)
* **Cloud Provider**: AWS (for hosting and services)
* **AI Integration**: OpenAI API with Semantic Kernel
* **Authentication**: AWS Cognito or IdentityServer for user management
* **Containerization**: Docker (for deployment and portability)

**3. System Architecture**

**Architecture Diagram**

1. **Frontend (Vue.js)**: User Interface for meal planning and interaction.
2. **Backend (ASP.NET Core 8)**: Exposes RESTful APIs for frontend communication and handles business logic.
3. **Database (PostgreSQL)**: Stores user data, meal plans, recipes, preferences, and logs.
4. **AI Service**: Integrates with **OpenAI API** or **custom AI models** hosted on AWS Lambda or SageMaker.
5. **AWS Services**:
   * **EC2 or ECS**: Host the backend.
   * **S3**: Store recipe images and static assets.
   * **Cognito**: Manage user authentication.
   * **RDS (PostgreSQL)**: Manage database hosting.
6. **Monitoring**: CloudWatch (for logging and performance monitoring).

**4. Frontend (Vue.js with Vite and Vuetify)**

**Purpose:**

* User-facing UI for creating and viewing meal plans.
* Supports **responsive design** to work on desktops, tablets, and smartphones.

**Key Features:**

* **User Dashboard**: Displays current meal plan and nutritional breakdown.
* **Meal Selection**: Allows users to swap and customize meals.
* **Grocery List Generation**: Generates and displays grocery lists.

**Tools and Libraries:**

* **Vue Router**: For client-side routing.
* **Pinia**: For state management.
* **Axios**: For API communication with the backend.
* **Vuetify**: For styling components.

**5. Backend (ASP.NET Core 8)**

**Purpose:**

* Handles business logic, processes API requests, and communicates with the database and AI services.

**Key Responsibilities:**

1. **User Management**: Login, registration, and authentication.
2. **Meal Plan Generation**: Logic for creating meal plans using **AI models** or **OpenAI API**.
3. **Database Operations**: CRUD operations for user profiles, meal plans, and recipes.
4. **Payment and Subscription**: Integrate with AWS Cognito or third-party payment gateways.

**Endpoints Example:**

* **GET /api/mealplans**: Fetch user’s meal plan.
* **POST /api/mealplans/generate**: Generate a new meal plan based on user preferences.
* **POST /api/grocerylist**: Generate a grocery list for the user

**6. Database (PostgreSQL)**

**Purpose:**

Store user profiles, meal plans, dietary preferences, recipes, and logs.

**Schema Design (Simplified):**

* **Users**: Stores user details and preferences.sql Copy code   CREATE TABLE Users (
* Id SERIAL PRIMARY KEY,
* Email VARCHAR(255) UNIQUE NOT NULL,
* PasswordHash TEXT NOT NULL,
* Preferences JSONB
* );
* **Meals**: Stores meal data and nutritional info.sql Copy code   CREATE TABLE Meals (
* Id SERIAL PRIMARY KEY,
* Name VARCHAR(255),
* Calories INT,
* Ingredients JSONB
* );
* **MealPlans**: Links users to their meal plans.sql Copy code   CREATE TABLE MealPlans (
* Id SERIAL PRIMARY KEY,
* UserId INT REFERENCES Users(Id),
* CreatedAt TIMESTAMP DEFAULT NOW(),
* Meals JSONB
* );

**7. AI Integration**

**Options for AI Integration:**

* **OpenAI API**:
  1. Use **GPT-based models** to recommend meal plans based on user preferences and previous behavior.
  2. Example Request to OpenAI:json Copy code   {
  3. "prompt": "Generate a 7-day vegan meal plan with 1500 calories per day.",
  4. "max\_tokens": 1000
  5. }
* **Custom AI Model**:
  1. Train a **custom recommendation model** using AWS SageMaker with user data.
  2. This requires **historical data** and **training pipelines** but offers more control over meal recommendations.

**Integration Plan:**

1. **OpenAI**: Use a RESTful client in .NET to call OpenAI’s API for meal recommendations.
2. **Custom AI**: Expose the AI model as a **microservice** on AWS Lambda and call it from the backend.
3. **We will be using Semantic Kernel for AI integration**

**8. Authentication and User Management**

* Use **AWS Cognito** to manage **user authentication, authorization**, and **password resets**.
* Alternatively, use **IdentityServer4** for a self-hosted authentication solution.

**9. Hosting and Deployment on AWS**

* **Backend Hosting**:
  1. Deploy the **.NET Core 8 API** using **AWS ECS** (Elastic Container Service) or **EC2** instances.
  2. Use **Docker** containers to package the backend service.
* **Frontend Hosting**:
  1. Deploy the **Vue.js frontend** to **AWS S3** with **CloudFront** for CDN distribution.
* **Database Hosting**:
  1. Use **AWS RDS (PostgreSQL)** to host the application database with automatic backups.
* **Static Assets**:
  1. Store recipe images and assets in **S3 buckets**.

**10. DevOps and CI/CD**

* **Version Control**: GitHub or Bitbucket for source control.
* **CI/CD Pipeline**:
  1. Use **GitHub Actions** or **AWS CodePipeline** to automate builds and deployments.
  2. Automate **unit tests** and **end-to-end tests** before deployment.

**11. Monitoring and Logging**

* **AWS CloudWatch**: Monitor API requests, errors, and performance.
* **Sentry**: Capture frontend exceptions and performance bottlenecks.
* **Health Checks**: Implement **/health** endpoint to monitor backend availability.

**12. Security and Compliance**

1. **SSL/TLS**: Use **Let’s Encrypt** or AWS Certificate Manager for HTTPS.
2. **Data Encryption**: Encrypt sensitive user data in PostgreSQL.
3. **Authentication**: Secure endpoints with **JWT tokens** issued by AWS Cognito.
4. **Rate Limiting**: Protect the API using **rate limits** to prevent abuse.

**13. Testing Strategy**

* **Unit Tests**: Test individual components (e.g., meal generation logic).
* **Integration Tests**: Test database and API interactions.
* **End-to-End Tests**: Simulate user workflows with tools like **Cypress** or **Playwright**.

**14. Timeline and Milestones**

1. **Month 1-2**: Frontend and backend setup, basic API endpoints, and database schema.
2. **Month 3**: AI integration with OpenAI API.
3. **Month 4**: Authentication and subscription management.
4. **Month 5**: Deploy to AWS and begin testing.
5. **Month 6**: Full launch with monitoring and CI/CD in place.

**15. Conclusion**

The **AI Meal Planner** leverages modern technologies (Vue.js, .NET 8, PostgreSQL, AWS) to create a scalable, secure, and user-friendly platform. With **AI-powered meal recommendations** and AWS infrastructure, the platform aims to provide seamless and personalized meal planning for users.

This technical specification ensures a clear development path, providing flexibility to use **OpenAI or custom AI models** as the recommendation engine evolves.

4o