1. Set Up App Databases

Tasks:

Choose a Database Type: Decide on SQL (PostgreSQL, MySQL) or NoSQL (MongoDB, Firebase) based on your data structure.

In our case we are to use mongoDB as our primary database and Firebase as a supportive storage and authentication.

Design Database Schema:

- Nutritional data: Food items, nutritional values.
- Meal plans: User-specific meal plans.
- User profiles: User information, preferences.

Create the Database:

- Set up the database server (local or cloud).
- Use migration tools to create tables/collections.

Resources/Tools:

DBMS: MongoDB and Firebase.

Design Tools: Lucidchart.

Hosting: AWS RDS (Alternate use), Google Cloud SQL (Primary one to use), Firebase.

2. Implement Algorithms for Meal Plans and Food Recommendations

Tasks:

Research Nutritional Guidelines: Familiarize yourself with dietary needs.

Develop Recommendation Algorithms:

Content-based filtering based on user preferences.

Collaborative filtering for shared meal plans.

Optimize Algorithms: Test with different datasets to improve accuracy.

Resources/Tools:

Programming Languages: Python, JavaScript, typescript.

Libraries: 1. NumPy, Pandas for data handling; Scikit-learn for machine learning.

2. Express js for all backend processes.

3. Develop APIs for Backend-Frontend Connection

Tasks:

Define API Endpoints: Create RESTful endpoints for fetching and storing data.

Implement API Logic: Use a suitable web framework to build your API.

Handle Requests and Responses: Ensure data is sent and received correctly.

Resources/Tools:

Frameworks: Express.js (Node.js), Flask/Django (Python).

API Testing: Postman, Insomnia.

Documentation: Swagger, Postman.

4. Implement User Authentication and Security Protocols

Tasks:

Choose Authentication Method: Implement OAuth2 or JWT for secure logins.

In our case case, we are choosing Oauth2 as our primary method for secure logins and firebase auth as backup auth

Set Up Security Measures:

HTTPS for data transmission.

Data encryption (e.g., bcrypt for passwords).

Input validation to prevent attacks (e.g., SQL injection).

Resources/Tools:

Libraries: Passport.js (Node.js), Flask-JWT (Python).

Encryption Tools: bcrypt, Argon2.

Security Guidelines: OWASP best practices.

5. Integrate AI Tools for Custom Responses

Tasks:

Select AI Tools: Choose from OpenAI, Dialogflow, or custom models.

Integrate AI into APIs: Develop logic to send user queries to the AI and return responses.

Test AI Responses: Ensure they are accurate and relevant.

Resources/Tools:

AI APIs: OpenAI API, Google Cloud AI.

Integration Libraries: Axios (for API requests), TensorFlow.js if needed.

6. Test Backend Infrastructure for Scalability and Reliability

Tasks:

Set Up Testing Frameworks: Use unit and integration tests to validate API functionality.

Load Testing: Simulate user traffic to assess performance and scalability.

Monitoring: Implement logging and monitoring to track server performance.

Resources/Tools:

Testing Frameworks: Jest, Mocha (JavaScript); pytest (Python).

Load Testing Tools: Apache JMeter, Locust.

Monitoring Tools: Grafana, Prometheus, New Relic.

General Procedures

1. Project Management:

Use Agile methodologies (Scrum).

Tools: Jira, Trello for task tracking.

2. Version Control:

Use Git for version control and collaboration (GitHub, GitLab).

3. Documentation:

Maintain thorough documentation of APIs, algorithms, and setup instructions using Markdown or Confluence.