# Phase 2: Backend and Database Setup

---

## Overview

This phase focuses on building the backend infrastructure and database required to manage user data, support AI recommendations, and integrate app functionality. It lays the foundation for seamless communication between the frontend, backend, and AI systems while ensuring scalability and reliability.

---

## Goals

1. Create a robust backend to handle user management, AI requests, and data storage.
2. Design a scalable database to store structured (user profiles, meal/workout plans) and flexible data (AI outputs).
3. Ensure secure and efficient APIs for communication between the frontend and backend.
4. Implement basic unit testing to validate functionality.

---

## Key Deliverables

1. **Backend Framework**:
    - User authentication and profile management.
    - API endpoints for meal and workout plans.
    - Communication endpoints for AI-generated recommendations.
2. **Database**:
    - Schema to handle structured and unstructured data.
    - Integration with PostgreSQL for core data storage.
    - Plans to incorporate MongoDB for dynamic data in future phases.
3. **Testing**:
    - Functional tests for APIs and database interactions.
4. **Deployment**:
    - Deploy the backend to AWS or another hosting provider.

---

## Backend Design

### 1. Framework Selection

- **Django REST Framework**:
    - Provides built-in tools for API development.
    - Streamlined integration with PostgreSQL.
    - Secure and scalable for long-term growth.

**2. Core Functionalities**

- **User Management**:
    - Registration, login, and profile updates.
    - Secure password handling using bcrypt.
    - JWT-based authentication for session management.
- **API Endpoints**:
    - `/register`: Register a new user.
    - `/login`: Authenticate users.
    - `/profile`: Fetch or update user details.
    - `/meal-plan`: Retrieve AI-generated meal plans.
    - `/workout-plan`: Retrieve AI-generated workout plans.
    - `/generate-plan`: Endpoint to send user inputs to the AI and receive personalized recommendations.
- **AI Communication**:
    - Middleware for sending user data to the AI model and receiving outputs.
    - Ensure timeout and fallback mechanisms for AI unavailability.

---

## Database Design

**1. Primary Database**

- **PostgreSQL** (Relational Database):
    - Handles structured data, such as:
        - User profiles, preferences, and progress.
        - Meal and workout plans.
        - User logs and progress tracking.

**2. Future Database Integration**

- **MongoDB** (Non-relational Database):
    - To be used for flexible and dynamic data, such as:
        - AI-generated plans stored in JSON format.
        - Logs of user interactions and queries.

**3. Database Schema Design**

**Tables for PostgreSQL:**

1. **Users Table**:
    - `id`: Primary key.
    - `name`, `email`, `password`, `age`, `height`, `weight`, `fitness_goal`, `dietary_preferences`.
2. **Meal Plans Table**:
    - `id`: Primary key.
    - `user_id`: Foreign key referencing `users`.
    - `date`: The date for the meal plan.
    - `breakfast`, `lunch`, `dinner`, `snacks`: Text fields for meal details.
3. **Workout Plans Table**:
    - `id`: Primary key.
    - `user_id`: Foreign key referencing `users`.
    - `date`: The date for the workout plan.
    - `exercises`: JSONB for flexible exercise details (sets, reps, duration).
4. **Progress Logs Table**:
    - `id`: Primary key.
    - `user_id`: Foreign key referencing `users`.
    - `date`: The date of the log.
    - `weight`, `calories_consumed`, `workouts_completed`: Tracking user progress.

---

## API Design

**Core APIs:**

1. `/register`:
    - Accepts user information and creates a new profile.
    - Validates email uniqueness and password strength.
2. `/login`:
    - Authenticates user credentials and returns a JWT token.
3. `/profile`:
    - Retrieves or updates user profile details.
4. `/meal-plan` and `/workout-plan`:
    - Fetches pre-generated plans for a specific user and date.
5. `/generate-plan`:
    - Sends user data (e.g., goals, preferences, availability) to the AI and retrieves a dynamic plan.

---

## Testing

**1. API Testing:**

- Use tools like **Postman** to validate endpoints for:
    - Input validation.
    - Proper response formats.
    - Authentication errors.

**2. Unit Tests:**

- Write unit tests for all major API functionalities using tools like **PyTest** or **unittest**.
- Test scenarios:
    - User registration with invalid email/password.
    - Correct retrieval of meal/workout plans.

**3. Database Testing:**

- Validate CRUD operations on PostgreSQL tables.
- Ensure proper foreign key relationships.

---

## Deployment

**1. Hosting:**

- Use **AWS** (EC2 or Elastic Beanstalk) for deploying the backend.
- Configure autoscaling to handle variable user loads.

**2. Environment Management:**

- Use `.env` files for sensitive credentials (e.g., database URLs, JWT secret keys).
- Implement CI/CD pipelines using GitHub Actions or AWS CodePipeline.

**3. Monitoring:**

- Set up basic monitoring tools:
    - **Prometheus** or **Datadog** for API performance metrics.
    - **CloudWatch Logs** for debugging.

---

## Timeline

**Week 1:**

- Finalize database schema.
- Set up Django REST Framework.
- Implement user registration and login APIs.

**Week 2:**

- Complete meal/workout plan APIs.
- Test API and database interactions.
- Deploy backend to AWS.

---

## Future Enhancements

1. Implement caching for frequently accessed data using **Redis**.
2. Introduce real-time updates with **WebSockets** (e.g., for progress tracking).
3. Expand MongoDB usage for storing dynamic AI-generated data.