

Important Note on Swiggy APIs: Swiggy does *not* provide public APIs for developers. Scraping them in real-time is unstable and difficult. For this project, I recommend **simulating** the Swiggy part using a seed database (mock data) or a one-time scrape of local restaurants to populate your own database. This allows you to control the data structure for the AI.

Here is your detailed Product Requirement Document (PRD) and Implementation Guide.

Project Name: NutriDel (Nutrition + Delivery)

1. High-Level Overview

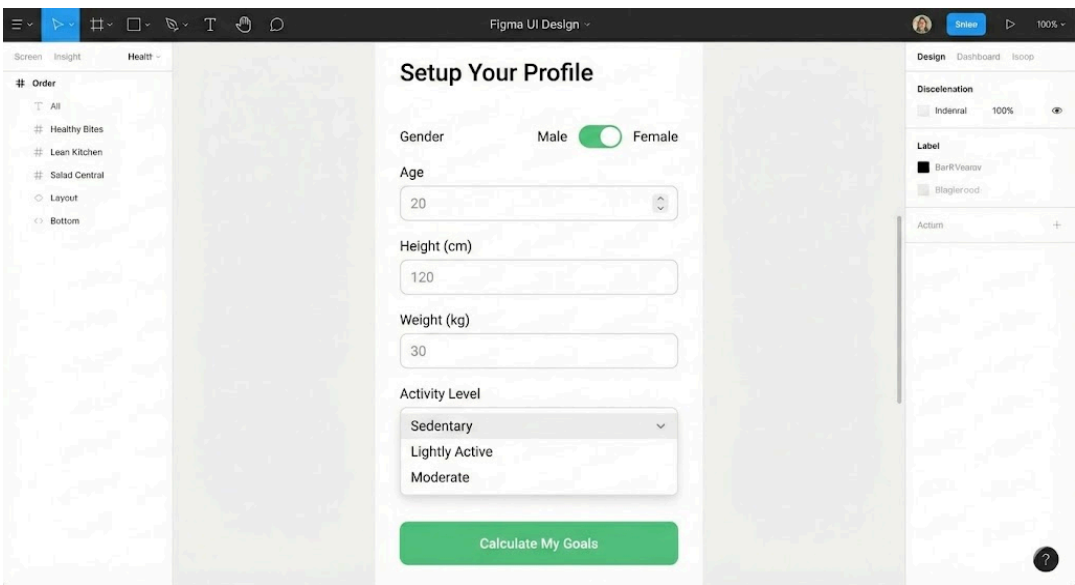
Goal: A food delivery platform that acts as a personal nutritionist. The system doesn't just list food; it understands the nutritional content of every menu item and recommends meals based on the user's specific biological data and goals using an LLM (Large Language Model).

Target Audience: Health-conscious individuals, students, and fitness enthusiasts.

2. Functional Requirements

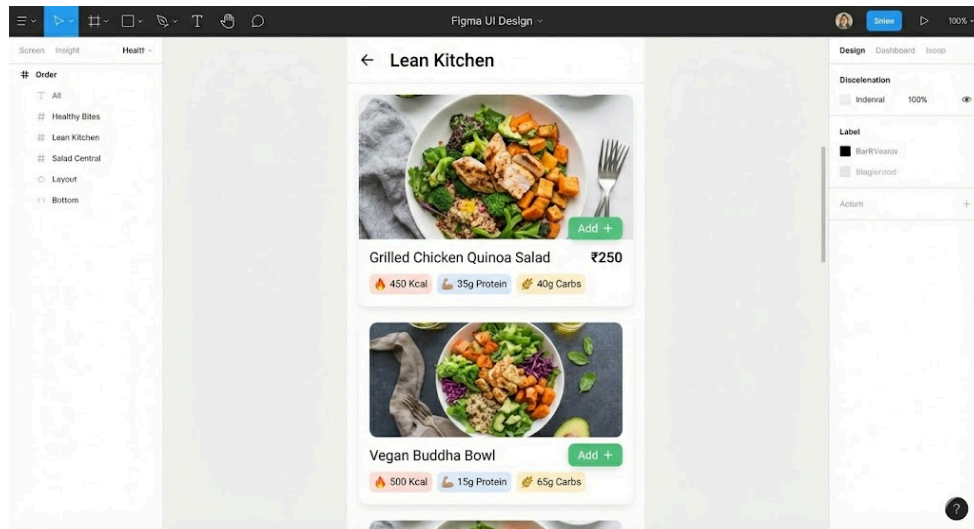
Module A: User Profile & Health Engine (The "HealthifyMe" Part)

- **Onboarding:** Input Gender, Age, Height, Weight, Activity Level (Sedentary, Active, etc.).
- **Calculators:** System auto-calculates BMI (Body Mass Index) and BMR (Basal Metabolic Rate).
- **Goal Setting:** User sets a goal (Weight Loss, Muscle Gain, Maintenance).
- **Dashboard:** Visual tracking of daily calories consumed vs. target.



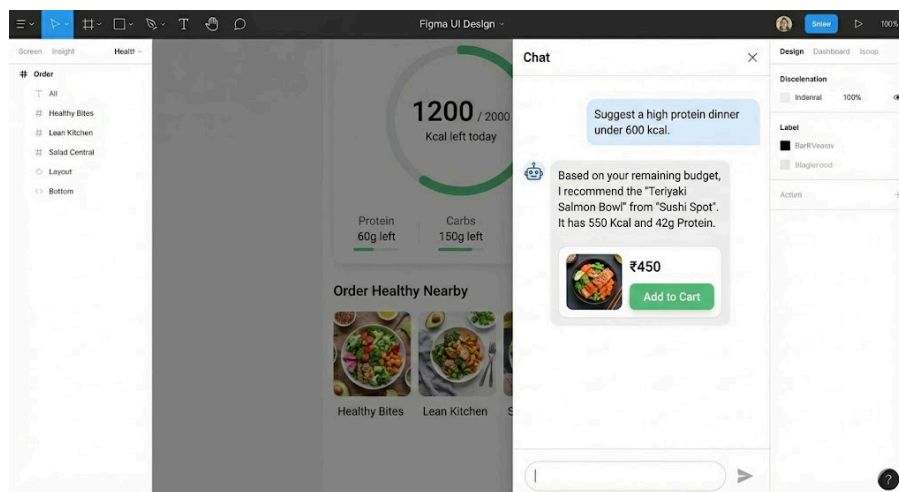
Module B: Food Delivery Core (The "Swiggy" Part)

- **Restaurant Listing:** List of mock restaurants.
- **Menu Display:** Dishes with price, image, and **nutritional breakdown** (Calories, Protein, Carbs, Fats). *Note: The AI needs this data to work.*
- **Cart & Mock Checkout:** Add items and place an order (no real payment gateway needed; use a mock success screen).



Module C: AI Nutritional Chatbot (The Integration)

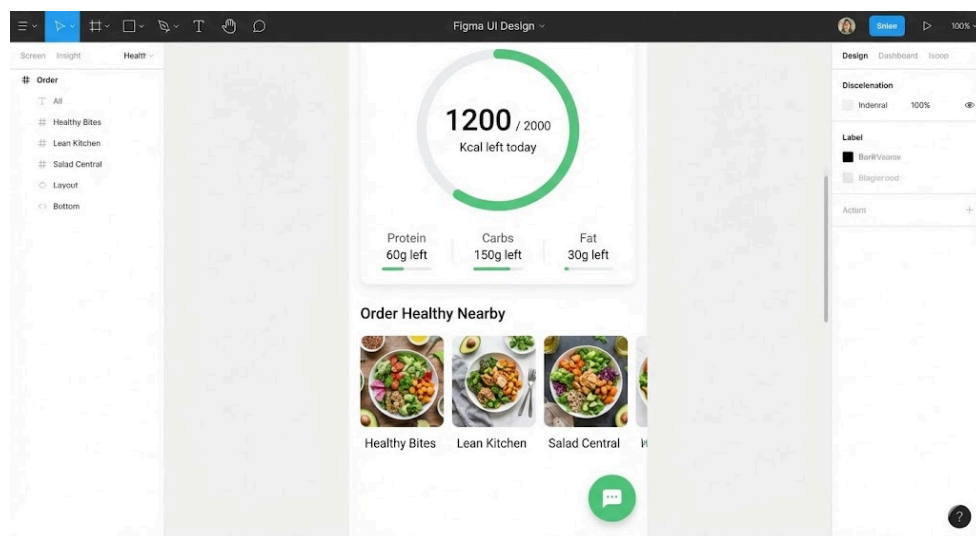
- **Context Aware:** The bot knows the user's BMR and current calorie intake for the day.
- **Menu Aware:** The bot has access to the database of restaurants.
- **RAG Implementation:** The backend searches the menu database for matching items, feeds them to the LLM, and the LLM generates a recommendation.
- **Query Example:** "I have 500 calories left today. Suggest a high-protein dinner from a nearby restaurant."
- **Output:** "I recommend the 'Grilled Chicken Salad' from 'Healthy Bites'. It has 450 kcal and 35g protein."



Module D: The Main Dashboard (The Hybrid View)

- **Requirement:** A single view that combines nutritional tracking status with the ability to order food.
- **Top Section:** Visual tracking of daily calorie balance (Calories consumed vs. Total daily target).
- **Bottom Section:** A standard restaurant listing grid (simulating Swiggy).
- **Floating Action Button (FAB):** A persistent button to access the AI chat.

Figma Design Reference: Main Dashboard Note the calorie counter at the top and the restaurant list at the bottom, with the chat button in the corner.



3. System Design & Architecture

Since you want to learn backend, we will use a robust flow.

A. Tech Stack Recommendation

- **Backend:** Node.js (Express) or Python (FastAPI/Django). *Python is better for AI integration, Node is better if you know JS.*
- **Database:** MongoDB (Great for flexible menu structures) or PostgreSQL.
- **AI/LLM:** OpenAI API (gpt-4o-mini is cheap) or Gemini API (often has a free tier).
- **Vector Database:** Pinecone or MongoDB Atlas Vector Search (crucial for the chatbot to "search" the menu).

B. High-Level Architecture Diagram

1. **Frontend (React/Next.js):** User Interface.
2. **API Server:** Handles Auth, Orders, and User Data.
3. **AI Service:** Middleware that constructs the prompt for the LLM.
4. **Knowledge Base:** A database containing menu items with nutritional metadata.

4. Implementation Details (The "How-To")

Phase 1: Authentication & Session Management

Security is the first step in backend learning.

Method: JWT (JSON Web Tokens).

1. **Login:** User sends credentials -> Server verifies -> Server signs a `Status: 200` and sends a `token`.
2. **Storage:** Store the token in `HttpOnly Cookies` (more secure than `LocalStorage`).
3. **Middleware:** Create a function `verifyToken` that runs before every private API call (like `GET /my-profile`).
4. **Expiration:** Tokens should expire in 24 hours.

Phase 2: The Data Layer (Simulating Swiggy)

You need a database of food. Do not hardcode this.

- **Schema (MongoDB Example):**
- JSON

```
// MenuItem Collection
{
  "_id": "...",
  "name": "Chicken Biryani",
  "restaurant_id": "rest_123",
  "price": 250,
  "nutrition": {
    "calories": 600,
    "protein": 25,
    "carbs": 80,
    "fats": 20
  },
  "tags": ["indian", "high-carb", "spicy"]
}
```

-
-
- **Implementation:** Create a JSON file with 50 diverse food items. Write a script to "seed" (upload) this to your database. This is your "Swiggy API."

Phase 3: The AI Implementation (RAG - Retrieval Augmented Generation)

This is the most technical and impressive part. You cannot feed the *entire* database to ChatGPT every time. You use **RAG**.

Step-by-Step Flow:

1. **Vectorization:** When you seed your database, use an embedding model (like OpenAI `text-embedding-3-small`) to convert the food name and description into a vector (a list of numbers). Store this in your DB.
2. **User Query:** User asks: "I need high protein food."
3. **Search:** Your backend converts the query to a vector and searches your DB for "nearest neighbors" (foods mathematically similar to 'high protein').
4. **Prompt Engineering:** The backend constructs a prompt:
"You are a nutritionist. The user needs 500kcal high protein. Here are the 5 closest items found in our database: [List of items found in step 3]. Recommend the best one and explain why."
5. **Response:** Send this to Gemini/ChatGPT API. Return the answer to the frontend.

Phase 4: Lifestyle Calculations

This is pure logic (Backend Service).

- BMR Formula (Mifflin-St Jeor):
$$BMR = (10 \times \text{weight}) + (6.25 \times \text{height}) - (5 \times \text{age}) + 5$$

(for men).
- **Implementation:** Create a utility function `calculateStats(user)` that runs whenever the user updates their profile. Store the result in the User DB so you don't recalculate every login.

5. UI/UX & Visual References

For the Figma design, you want a "Clean" and "Green" aesthetic.

A. Dashboard / Home

- **Top:** Greeting + "Calories Left Today" (Progress circular bar).
- **Middle:** "Order Food" (Grid of food images like Swiggy).
- **Bottom:** Floating Action Button (FAB) for the AI Chat.

B. The AI Chat Interface

- **Right Side Drawer or Pop-up:**
- **Look:** Like WhatsApp or ChatGPT.
- **Feature:** When the bot suggests a food (e.g., "Try the Greek Salad"), there should be an **"Add to Cart" button directly inside the chat bubble.**

6. Detailed System Components (Backend)

Here is exactly what you need to build in your backend folder structure:

1. `server.js`: Entry point.
2. `/config`: Database connection logic.
3. `/models`:

- `User.js` (Stores name, auth, height, weight, daily_logs).
 - `Restaurant.js` (Stores details).
 - `MenuItem.js` (Stores food info + nutrition data).
4. **`/routes`:**
- `/auth` (Login/Register).
 - `/user` (Update weight, get stats).
 - `/food` (Get menu, Search food).
 - `/ai` (The endpoint where the chat logic lives).
5. **`/controllers`:** The logic for the routes.
6. **`/utils`:**
- `aiClient.js` (Setup for Gemini/OpenAI).
 - `calculations.js` (The BMI/BMR math).

7. A Next Step for You

This is a big project, but very manageable if broken down.

Would you like me to generate the detailed Mongoose (MongoDB) Schema code for the `User` and `MenuItem` models so you can start setting up the database today?