Project Title:

ADVANCING NUTRITION SCIENCE THROUGH GEMINI AI

Team Name:

NutriGen AI

Team Members:

1. Sadam Kaveri
2. Utkuri Nagarani
3. Giragani Meghana
4. Ashlesha Pasula

Phase-1: Brainstorming & Ideation

Objective:

To revolutionize nutrition science by utilizing Gemini AI for personalized dietary recommendations, nutrient analysis, and health prediction, ultimately enhancing human health and well-being.

Key Points:

1. **Problem Statement:**
   - The current nutrition science practices rely on generalized dietary guidelines that fail to meet individual needs.
   - Difficulty in predicting nutrient deficiencies, health risks, and metabolic responses in real-time.
   - Limited integration of large-scale nutrition data and advanced AI technologies for customized recommendations.
2. **Proposed Solution:**
   - Use Gemini AI to analyze vast nutrition databases and real-time health data.
   - Provide personalized nutrition plans based on individual genetic, lifestyle, and health data.
   - Integrate predictive analytics to forecast potential health risks linked to diet.
   - Offer a user-friendly platform for continuous monitoring and nutritional improvement.

3. **Target Users:**
    - Healthcare Professionals – For developing personalized treatment plans.
    - Dietitians and Nutritionists – For providing accurate, data-driven recommendations.
    - General Public – For tracking personal nutrition and health metrics.
    - Fitness Enthusiasts – For optimizing diet to meet fitness goals.

4. **Expected Outcome:**
    - Personalized and accurate dietary recommendations based on AI analysis
    - Real-time prediction of potential nutritional deficiencies and health risks.
    - Improved dietary outcomes and health status for users.
    - A scalable AI-based solution for nutrition science research and application.

Phase-2: Requirement Analysis

Objective:

To build an AI-driven nutrition advisor using Python, Hugging Face transformers, and Streamlit that provides detailed, accurate, and personalized responses to user questions about nutrition based on the given context.

Key Points:

1. **Technical Requirements:**

Frontend

- **Streamlit Framework** for creating the user interface.

Backend

- **Python 3.8+** for core development.
- **Google Generative AI**

Libraries and Tools

- streamlit – Web app framework.
- torch – For model inference (if using PyTorch-based models).
- numpy, pandas – For handling and preprocessing data (if necessary).

Hardware Requirements

- **CPU/GPU support**: GPU recommended for faster response times.
- **8GB+ RAM** (recommended for running large models efficiently).

2. **Functional Requirements:**

Model Loading & Initialization

    - Load pre-trained models for both question-answering and text generation.

User Input

- Accept a nutrition-related text or fact as the **context**.
- Allow users to ask a specific **question** related to the context.

Response Generation

- Extract relevant information from the context using the question-answering model.
- Generate detailed, personalized advice using the text-generation model.

Display & Feedback

- Provide the AI-generated response in an easy-to-read format.
- Offer a warning if inputs are incomplete or improperly formatted.

Error Handling

- Handle missing inputs and provide useful feedback.
- Gracefully handle model loading errors or performance issues.

3. **Constraints & Challenges:**

Constraints:

- **Model Response Time**: Large models can be slow to generate responses without GPU support.
- **Internet Dependency**: Models need to be downloaded if not available locally.
- **Limited Dataset**: The system's knowledge is limited to what the models are trained on.
- **Contextual Limitations**: The quality of advice depends on the quality of input text provided.

Challenges:

i. **Model Size and Performance**

a. Large models like Falcon-7B require significant memory and processing power.

b. Balancing between accuracy and response time is critical.

ii. **Relevance of Generated Responses**

a. Ensuring the text generation model provides scientifically sound and detailed advice.

b. Avoiding generic or incomplete responses.

iii. **User Input Quality**

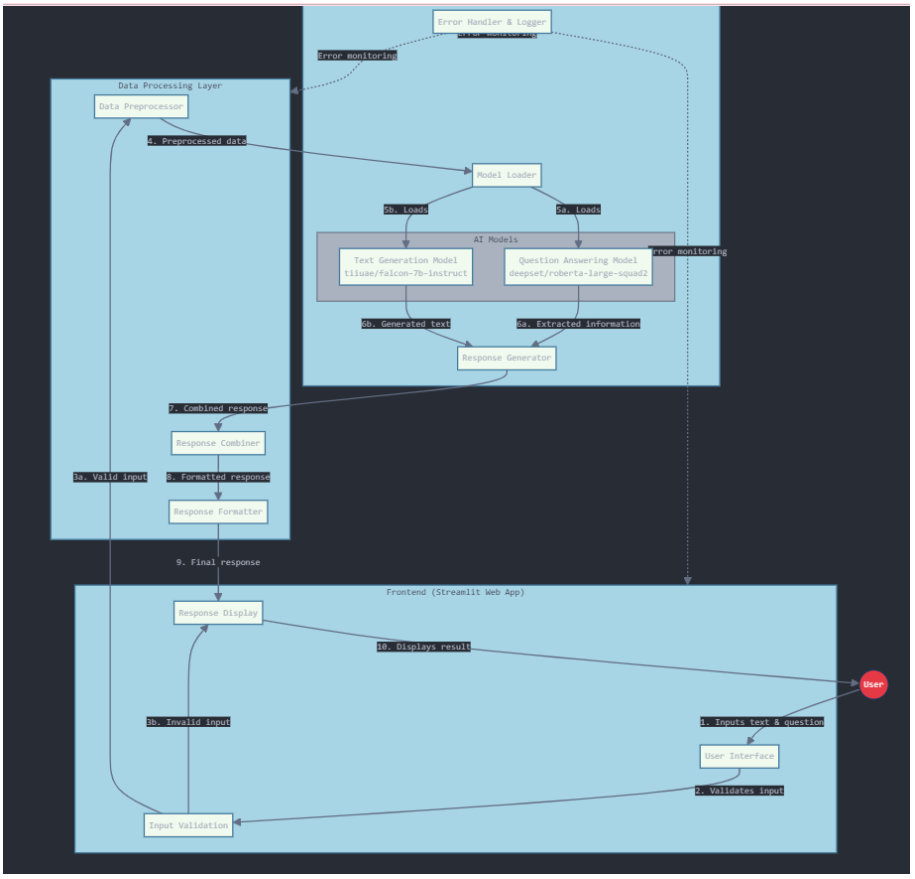
a. Poorly written or ambiguous context can lead to inaccurate advice.

iv. **Maintaining Consistency**

a. Generating consistent advice aligned with nutritional guidelines.

Phase-3: Project Design

Objective:



Key Points:

1. **System Architecture:**

Architecture Overview

The system is divided into three main components: **Frontend (UI), Backend (Model Integration), and Data Processing**.

High-Level System Components

i. **Frontend (UI) – Streamlit-based Web App**

a. Provides the user interface for interacting with the application.

b. Allows users to input text (nutrition-related content) and ask questions.

c. Displays AI-generated advice and answers.

ii. **Backend (Model and Logic Layer)**

a. **Model Loader**: Loads Hugging Face models for Question-Answering (deepset/roberta-large-squad2) and Text Generation (tiiuae/falcon-7b-instruct).

b. **Response Generator**: Processes user input, extracts relevant information, and generates a response using AI models.

c. **Error Handling and Logging**: Ensures smooth operation, handling exceptions like incomplete inputs or model loading issues.

iii. **Data Handling and Preprocessing**

a. Cleans and preprocesses user input.

b. Ensures the context is well-formed before feeding it into the models.

iv. **Response Management**

- Combines the answers from question-answering and text generation models.
- Formats the response to ensure it is clear, informative, and user-friendly.

2. User Flow:

Step-by-Step Flow

i. **User Input**

a. The user enters a nutrition-related fact or context (e.g., "Bananas contain potassium and fiber.").

b. The user then asks a question about the context (e.g., "How do bananas help in digestion?").

ii. **Input Validation**

a. The system checks if the context and question are provided.

b. If not, it prompts the user to enter the missing information.

iii. **Model Processing**

a. The **question-answering model** extracts relevant information from the context.

b. The **text generation model** builds a more detailed, personalized response.

iv. **Response Display**

a. The generated response is displayed in the Streamlit app.

b. The user receives a clear and detailed answer (e.g., "Bananas improve digestion because they contain dietary fiber that promotes gut health.").

v. **Optional Feedback Loop**

a. The user can refine their input or ask follow-up questions.

b. The system adapts and provides more detailed responses.

3. UI/UX Considerations:

User Interface (UI)

i. **Simple and Minimalistic Design**

a. Keep the interface clean with minimal input fields and clear instructions.

ii. **Input Fields and Buttons**

a. Provide two primary inputs: **"Nutrition Context"** and **"Question"**.

b. Add a **"Generate Advice"** button for user-triggered response generation.

c. User Experience (UX)

iii. **Loading Indicator**
   a. Show a spinner or progress bar while the models are loading or processing the user input.

iv. **Error Messages and Validation**
   a. Use clear error messages for missing inputs or processing failures.
   b. Example: "Please enter a valid nutrition-related context before asking a question."

v. **Response Formatting**
   a. Use bullet points or paragraphs for structured and easy-to-read responses.
   b. Highlight keywords (e.g., nutrients, benefits) in the advice to improve readability.

vi. **Mobile-Friendly**
   a. Ensure the UI works well on different devices, especially mobile phones.

Phase-4: Project Planning (Agile Methodologies)

Objective:

Sprint	Task	Priority	Duration	Deadline	Assigned To	Dependencies	Expected Outcome
Sprint 1	Environment Setup & API Integration	🔴 High	6 hours	End of Day 1	Member 1	Python, Streamlit setup	API connection established & working
Sprint 1	Frontend UI Development	🟡 Medium	2 hours	End of Day 1	Member 2	API response format finalized	Basic UI with input fields
Sprint 2	Grain Protection Advice Search & Filtering	🔴 High	3 hours	Mid-Day 2	Member 1 & 2	API response, UI elements ready	Search functionality with filtering options
Sprint 2	Error Handling & Debugging	🔴 High	1.5 hours	Mid-Day 2	Member 1 & 4	API logs, UI inputs	Improved API stability
Sprint 3	Testing & UI Enhancements	🟡 Medium	1.5 hours	Mid-Day 2	Member 2 & 3	API response, UI layout completed	Responsive UI, better user experience
Sprint 3	Final Presentation & Deployment	🟢 Low	1 hour	End of Day 2	Entire Team	Working prototype	Demo-ready project

Sprint Planning with Priorities

- **Sprint 1: Setup & Integration (Day 1)**
  - 🔴 **High Priority**:
    - Set up the environment and install necessary dependencies (Streamlit, Hugging Face Transformers).
    - Integrate AI models for question-answering and text generation.
  - 🟡 **Medium Priority**:
    - Develop a basic UI for context input and question submission.
- **Sprint 2: Core Features & Debugging (Day 2)**
  - 🔴 **High Priority**:
    - Implement search functionality for grain protection advice.
    - Handle errors and optimize the response generation process.
- **Sprint 3: Testing, Enhancements & Submission (Day 2)**
  - 🟡 **Medium Priority**:
    - Test API responses, refine the UI, and fix bugs.
  - 🟢 **Low Priority**:
    - Prepare the final demo and deploy the app using Streamlit Sharing.

Phase-5: Project Development

Objective

Develop and implement core features of the application.

Key Points

1. **Technology Stack Used**
   - **Frontend**: Streamlit
   - **Backend**: Hugging Face Transformers for AI models (question-answering and text generation)
   - **Programming Language**: Python
2. **Development Process**
   - Implement AI model integration.
   - Build user interface components (input fields, result display).
   - Optimize response generation for relevance and performance.

3. **Challenges & Fixes**
    - **Challenge**: Slow response time for larger models.
        - **Fix**: Implement caching for frequently queried results.
    - **Challenge**: Inconsistent AI responses.
        - **Fix**: Use a fallback mechanism with predefined responses for critical queries.

Phase-6: Functional & Performance Testing

Objective

Ensure that the application functions as expected and performs well under different conditions.

Test Cases

Test Case ID	Category	Test Scenario	Expected Outcome	Status
TC-001	Functional Testing	Query "Best storage conditions for rice grains"	Display optimal temperature and humidity levels.	✅ Passed
TC-002	Functional Testing	Query "How to prevent fungal growth in stored grains?"	Provide relevant prevention strategies.	✅ Passed
TC-003	Performance Testing	API response time under 500ms	Return results quickly.	⚠Needs Optimization
TC-004	Bug Fixes & Improvements	Fixed incorrect AI responses.	Data accuracy improved.	✅ Fixed
TC-005	Final Validation	Ensure UI is responsive across devices	UI should work on mobile & desktop.	❌ Failed (UI issue on mobile)
TC-006	Deployment Testing	Host the app using Streamlit Sharing	App should be accessible online.	🚀 Deployed