NutriPro

A Nutrition App Using Gemini Pro :

Your Guide to Healthy Eating and Well-being

by

Harshith YVS

1. Project Description

Nutritionist AI is a cutting-edge mobile app created to offer customized diet suggestions and nutrition guidance by utilizing the advanced features of the Gemini Pro model. The application utilizes artificial intelligence to examine user information, dietary choices, and health objectives, providing customized meal schedules, nutritional advice, and wellness suggestions. Nutritionist AI's main goal is to encourage healthier eating behaviours and enhance overall wellness with intelligent suggestions based on data.

2. Case Studies/Use Cases

Scenario 1: Weight Loss Journey

Sarah, a 28-year-old with a goal to lose 15 pounds, uses Nutritionist AI to aid her in her weight loss journey. As a vegetarian with a moderate activity level, she inputs her dietary preferences and health goals into the app. Nutritionist AI creates a calorie-controlled, nutrient-dense meal plan tailored to her vegetarian diet. Sarah logs her meals by taking photos or scanning barcodes, and the app provides feedback on her calorie intake and nutritional balance, suggesting necessary adjustments. By syncing her fitness tracker, the app integrates her physical activity data, offering comprehensive insights to help Sarah stay on track with her weight loss while maintaining proper nutrition.

Scenario 2: Managing Diabetes

John, a 45-year-old with Type 2 Diabetes, relies on Nutritionist AI to manage his condition through diet. He inputs his low-carb dietary preference and diabetes condition, and the app generates meal plans that focus on low carbohydrate and high fibre content to help control his blood sugar levels. John uses the app to log his meals, receiving immediate feedback on their suitability for his diabetes management. Detailed nutritional breakdowns highlight carbohydrate content and glycaemic index, aiding John in making informed food choices. Additionally, the app provides educational resources about managing diabetes through diet, keeping John well-informed and empowered to handle his condition better.

Scenario 3: Building the Muscle

Emily, a 30-year-old strength training enthusiast, uses Nutritionist AI to support her goal of gaining muscle mass. With a preference for high-protein meals and an intense workout regime, she inputs her dietary preferences and fitness goals into the app. Nutritionist AI generates meal plans rich in protein and essential nutrients necessary for muscle growth. Emily benefits from a variety of high-protein recipes that cater to her needs, with each recipe including detailed instructions and nutritional information. By connecting her fitness tracker, the app accounts for her caloric expenditure and provides

insights on balancing her protein intake with her workouts, optimizing her muscle-building efforts.

3. Project Architecture
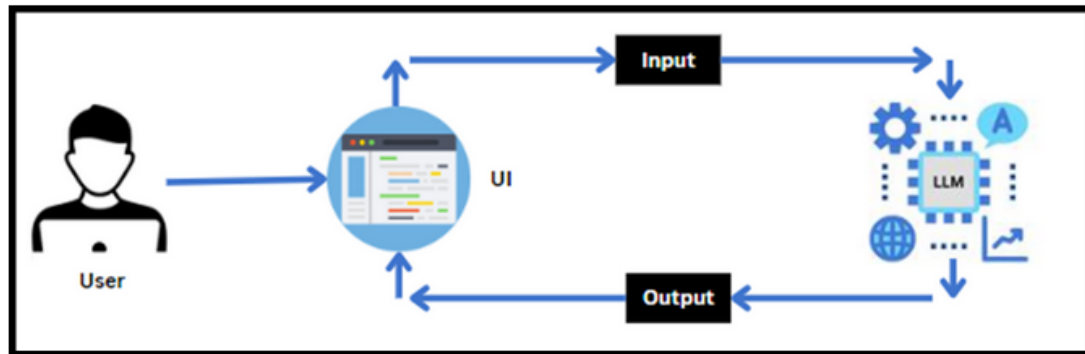
The above image depicts a general architecture of a system that uses a Large Language Model (LLM) to process user input and generate output. Here's a breakdown of the process:

- The user interacts with the system, providing input through a user interface (UI). This input can be in various forms, such as text, voice, or images.
- The UI receives the user input and processes it, potentially performing tasks like validation, normalization, or tokenization.
- The processed input is then sent to the LLM.
- The LLM processes the input using its vast knowledge and understanding of language, generating a response based on the provided context and patterns.
- The generated response from the LLM is sent back to the UI.
- The UI then presents the LLM's response to the user in a suitable format, such as text, voice, or visual output.
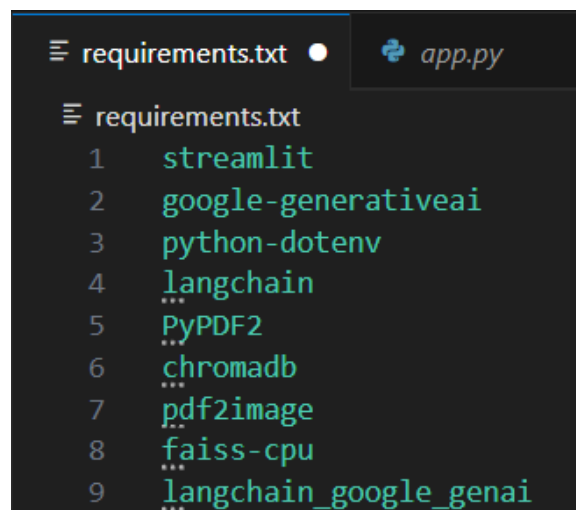
4. Project Flow

Milestone 1: Requirements Specification

Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

Activity 1: Create a requirements.txt file to list the required libraries.

- **streamlit:** Streamlit is a powerful framework for building interactive web applications with Python.

- **streamlit_extras:** Additional utilities and enhancements for Streamlit applications.

- **google-generativeai:** Python client library for accessing the GenerativeAI API, facilitating interactions with pre-trained language models like Gemini Pro.

- **python-dotenv:** Python-dotenv allows you to manage environment variables stored in a .env file for your Python projects.

- **Pillow:** Pillow is a Python Imaging Library (PIL) fork that adds support for opening, manipulating, and saving many different images file formats.

```
requirements.txt ●        app.py

 requirements.txt
    1   streamlit
    2   google-generativeai
    3   python-dotenv
    4   langchain
    5   PyPDF2
    6   chromadb
    7   pdf2image
    8   faiss-cpu
    9   langchain_google_genai
```

Activity 2: Install the required libraries

- Open the terminal.
- Run the command:
  `pip install -r requirements.txt`
- This command installs all the libraries listed in the requirements.txt file.

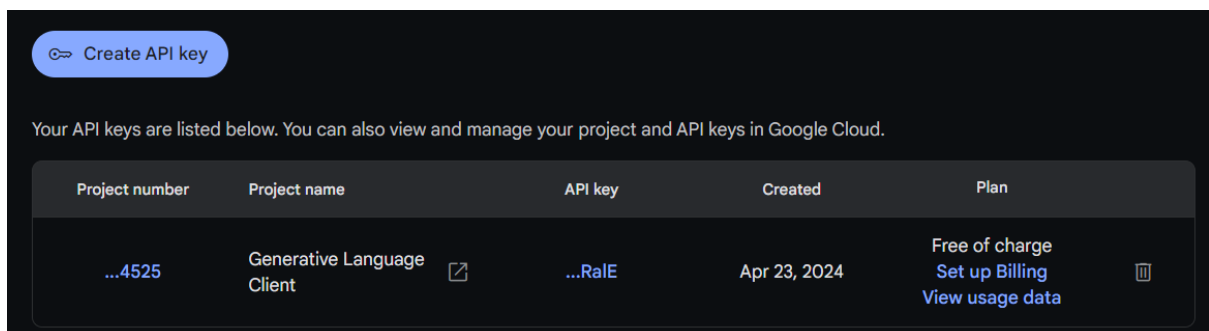**Milestone 2: Initialization of Google API Key**

The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs. It acts as a form of identification, allowing users to access specific Google services and resources. This key plays a crucial role in authorizing and securing API requests, ensuring that only authorized users can access and utilize Google's services.

Activity 1: Generate Google API Key

1) Click the provided link to access the following webpage.

   Link: https://ai.google.dev/gemini-api/docs/api-key

2) After signing in to your account, navigate to the 'Get an API Key' option. Clicking on this option will redirect you to another webpage as shown below.
3) Next, click on 'Create API Key' and choose the generative language client as the project.
4) Then, select 'Create API key in existing project'.
5) Copy the newly generated API key as it is required for loading the Gemini Pro pre-trained model.

Activity 2: Initialize Google API Key

- Create a .env file and define a variable named GOOGLE_API_KEY.
- Assign the copied Google API key to this variable.
- Paste the API key obtained from the previous steps here.
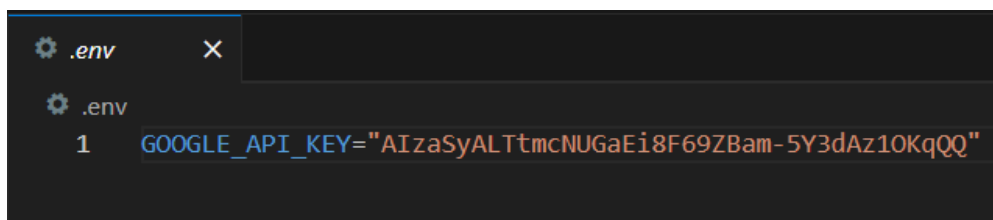
Milestone 3: Interfacing with Pre-trained Model

To interface with the pre-trained model, we'll start by creating an app.py file, which will contain both the model and Streamlit UI code.

Activity 1: Load the Gemini Pro API

```python
# NutriPro
import streamlit as st
import os
import google.generativeai as genai
from PIL import Image
from dotenv import load_dotenv

load_dotenv()  # loading the environment variables

genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
model = genai.GenerativeModel('gemini-1.5-flash')
```

This code snippet is for initializing a health management application using Streamlit, an open-source app framework, and Google Generative AI services. The script starts by loading environment variables from a .env file using the load_dotenv() function from the dotenv package. It then imports necessary libraries: streamlit for creating the web app interface, os for accessing environment variables, google.generativeai for utilizing Google's Generative AI capabilities, and PIL.Image for image processing. The genai.configure() function is called to set up the Google Generative AI API with the API key retrieved from the environment variables, ensuring secure and authorized access to the AI services.

Activity 2: Implement a function to get gemini response

- The function get_gemini_response takes an input text as a parameter.
- It calls the generate_content method of the model object to generate a response.
- The generated response is returned as text.

```python
# function to load Google Gemini Pro model and get response
def get_gemini_response(prompt_text, image_data, prompt_description):
    # Use gemini-1.5-flash instead of gemini-pro-vision
    model = genai.GenerativeModel('gemini-1.5-flash')
    response = model.generate_content([prompt_text, image_data[0], prompt_description])
    return response.text
```

Activity 3: Implement a function to read the Image and set the image format for Gemini Pro model Input

The function input_image_setup processes an uploaded image file for a health management application. It first checks if a file has been uploaded. If a file is present, it reads the file's content into bytes and creates a dictionary containing the file's MIME type and its byte data. This dictionary is then stored in a list named image_parts, which is returned by the function. If no file is uploaded, the function raises a FileNotFoundError, indicating that an image file is required but not provided. This setup ensures that the uploaded image is correctly formatted and ready for further processing or analysis in the application.

```python
# check if a file has been uploaded
def input_image_setup(uploaded_file):
    if uploaded_file is not None:
        bytes_data = uploaded_file.getvalue()  # read the file into bytes

        image_parts = [
            {
                "mime_type": uploaded_file.type,
                "data": bytes_data
            }
        ]
        return image_parts
    else:
        raise FileNotFoundError("No file uploaded")

st.set_page_config(page_title="NutriPro")
st.header('NutriPro with Google Gemini')
```

Activity 4: Write a prompt for gemini model

The variable input_prompt is a multi-line string designed as a prompt for a Gemini Health App model. It instructs the model to analyse an image of food items, identify each food item, and calculate the total calories. Additionally, the model is to provide a detailed breakdown of each food item with its respective calorie count. The expected output format is a numbered list where each item is listed alongside its calorie content, ensuring clarity and structured information for the user. This prompt is likely used in conjunction with an AI service that can process images and generate nutritional information based on the visual data provided.

```python
input_prompt = """You have to identify different types of food in images.
The system should accurately detect and label various foods displayed in the image,
providing the name of the food and its location within the image (e.g., bottom left, right corner, etc.).
Additionally, the system should extract nutritional information and
categorize the type of food (e.g., fruits, vegetables, grains, etc.) based on the detected items.
The output should include a comprehensive report or display showing the identified foods, their positions, names,
and corresponding nutritional details."""
```

Milestone 4: Model Deployment

We deploy our model using the Streamlit framework, a powerful tool for building and sharing data applications quickly and easily. With Streamlit, we can create interactive web applications that allow users to interact with our models in real-time, providing an intuitive and seamless experience.

Activity 1: Integrate with Web Framework

NutriPro Application:

```python
st.set_page_config(page_title="NutriPro")
st.header('NutriPro with Google Gemini')

# Avoid using 'input' as a variable name to prevent conflict with built-in function
user_input = st.text_input("Input prompt: ", key='user_input')
uploaded_file = st.file_uploader("Choose an image of the food or food table", type=["jpg", 'jpeg', 'png'])
image = ""

if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image, caption="Uploaded Image", use_column_width=True)

submit = st.button("Scan & Analyze")
```

If "Scan and Analyze button is clicked":

```python
if submit:
    try:
        image_data = input_image_setup(uploaded_file)  # Use correct function to get image data
        response = get_gemini_response(user_input, image_data, input_prompt)
        st.subheader("NutriPro Scan Results: ")
        st.write(response)
    except Exception as e:
        st.error(f"Error processing the image: {e}")

st.markdown("<small>© 2024 Harshith YVS. All rights reserved.</small>", unsafe_allow_html=True)
```

This code initializes a Streamlit application titled "NutriPro" by setting the page title and creating the app's header. It includes a text input field for users to enter a custom prompt and a file uploader for users to upload an image in JPG, JPEG, or PNG format. If an image is uploaded, it is opened using the PIL library and displayed within the app with a caption. A button labeled "Scan & Analyze" is also provided, which users can click to trigger the application's functionality for analyzing the uploaded image to calculate and display the total calorie content of the food items depicted.

Activity 2: Host the Application

Launching the Application:

- To host the application, go to the terminal, type:

```
streamlit run app.py
```

- Here app.py refers to a python script.

```
PS C:\Users\Harshith Y\Desktop\NutriPro> streamlit run app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.29.177:8501
```

5. References

[1]   https://www.tutorialspoint.com/natural_language_processing/index.htm
[2]   https://en.wikipedia.org/wiki/Generative_artificial_intelligence
[3]   https://deepmind.google/technologies/gemini/#introduction
[4]   https://ai.google.dev/gemini-api/docs/get-started/python
[5]   https://colab.research.google.com/github/google/generative-ai-
       docs/blob/main/site/_en/gemini-api/docs/get-started/python.ipynb
[6]   https://www.geeksforgeeks.org/a-beginners-guide-to-streamlit/

APPENDIX

GitHub Link: https://github.com/harshith7422/NutriPro

RESULTS:

NutriPro Scan Results:

The image contains a bowl of Dal Makhani, a popular lentil-based Indian dish. The dish is a thick, creamy stew made with lentils, butter, cream, and spices. It is typically served with rice or naan bread. The Dal Makhani is topped with cilantro and chili peppers, and it is served in a bowl.

Here is a breakdown of the food items and their nutritional information:

- **Dal Makhani:** A rich and flavorful lentil-based stew, packed with protein, fiber, and essential vitamins and minerals. It is a good source of iron, folate, and magnesium.
- **Cilantro:** A fresh herb with a bright flavor, rich in vitamins A, C, and K. It also contains antioxidants and has anti-inflammatory properties.
- **Chili Peppers:** These spicy peppers add a kick to the dish. They are a good source of vitamin C and capsaicin, a compound with potential health benefits.

Food Type: Grains, Vegetables, and Legumes

Position: The Dal Makhani is in the center of the image. The cilantro is on top of the dish, while the chili peppers are scattered throughout the stew.

The image highlights a delicious and nutritious Indian dish, Dal Makhani. It is a flavorful and satisfying meal that can be enjoyed by people of all ages.

CODE:

app.py

```python
# NutriPro
import streamlit as st
import os
import google.generativeai as genai
from PIL import Image
from dotenv import load_dotenv

load_dotenv()  # loading the environment variables

genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
model = genai.GenerativeModel('gemini-1.5-flash')

# function to load Google Gemini Pro model and get response
def get_gemini_response(prompt_text, image_data, prompt_description):
    # Use gemini-1.5-flash instead of gemini-pro-vision
    model = genai.GenerativeModel('gemini-1.5-flash')
    response = model.generate_content([prompt_text, image_data[0], prompt_description])
    return response.text

# check if a file has been uploaded
def input_image_setup(uploaded_file):
    if uploaded_file is not None:
        bytes_data = uploaded_file.getvalue()  # read the file into bytes

        image_parts = [
            {
                "mime_type": uploaded_file.type,
                "data": bytes_data
            }
        ]
        return image_parts
    else:
        raise FileNotFoundError("No file uploaded")

st.set_page_config(page_title="NutriPro")
st.header('NutriPro with Google Gemini')
```

```python
38    # Avoid using 'input' as a variable name to prevent conflict with built-in function
39    user_input = st.text_input("Input prompt: ", key='user_input')
40    uploaded_file = st.file_uploader("Choose an image of the food or food table", type=["jpg", 'jpeg', 'png'])
41    image = ""
42
43    if uploaded_file is not None:
44        image = Image.open(uploaded_file)
45        st.image(image, caption="Uploaded Image", use_column_width=True)
46
47    submit = st.button("Scan & Analyze")
48
49    input_prompt = """You have to identify different types of food in images.
50    The system should accurately detect and label various foods displayed in the image,
51    providing the name of the food and its location within the image (e.g., bottom left, right corner, etc.).
52    Additionally, the system should extract nutritional information and
53    categorize the type of food (e.g., fruits, vegetables, grains, etc.) based on the detected items.
54    The output should include a comprehensive report or display showing the identified foods, their positions, names,
55    and corresponding nutritional details."""
56
57    if submit:
58        try:
59            image_data = input_image_setup(uploaded_file)  # Use correct function to get image data
60            response = get_gemini_response(user_input, image_data, input_prompt)
61            st.subheader("NutriPro Scan Results: ")
62            st.write(response)
63        except Exception as e:
64            st.error(f"Error processing the image: {e}")
65
66    st.markdown("<small>© 2024 Harshith YVS. All rights reserved.</small>", unsafe_allow_html=True)
```

requirements.txt

```
requirements.txt ●        app.py

requirements.txt
1    streamlit
2    google-generativeai
3    python-dotenv
4    langchain
5    PyPDF2
6    chromadb
7    pdf2image
8    faiss-cpu
9    langchain_google_genai
```