

| | |
|----------------------|---|
| Project Name: | <i>Nutrition App Using Gemini Pro</i> |
| Team ID: | <i>SWTID1720075968</i> |
| Team Members: | <i>Ramya Rajesh Nair</i> <i>Sunku Peda Akshay</i> <i>Pendekanti Saikarthik</i> <i>Sowmya Chowdary Gogadi</i> |

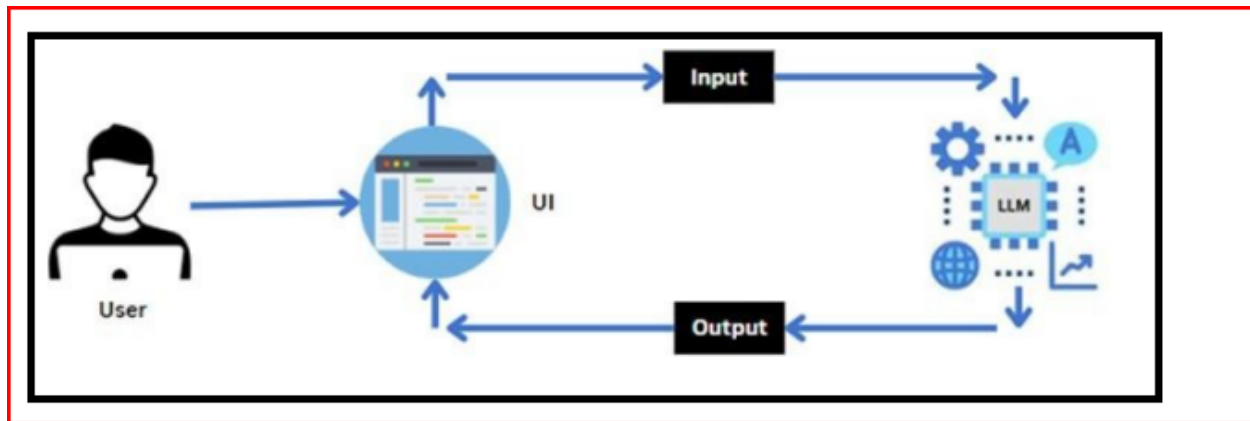
Nutrition App Using Gemini Pro

AIM:-

To develop Nutritionist AI, a state-of-the-art mobile application utilizing Gemini Pro AI technology, with the objective of delivering highly personalized dietary recommendations and holistic wellness solutions to empower users in achieving sustainable health goals. The app will integrate advanced machine learning algorithms to analyze user data, including dietary preferences, health metrics, and lifestyle habits, to generate customized meal plans, provide real-time nutritional insights, and offer educational

resources. By leveraging Gemini Pro's capabilities, Nutritionist AI aims to revolutionize the way individuals manage their nutrition and wellness, fostering healthier lifestyles and improved overall well-being.

Technical Architecture:-



Project Flow:

User Interaction with the UI:

- Users interact with a user-friendly mobile application interface designed to provide intuitive access to various features related to nutrition and health.
- The UI includes options for users to input their dietary preferences, health goals, current physical activity levels, dietary restrictions (if any), and other relevant personal information.

Collection and Transmission of User Input:

- Once the user inputs their information through the UI, the data is collected and formatted within the frontend application.

- The collected user input is then securely transmitted to the backend server using an API endpoint. This transmission is facilitated using a secure communication protocol and is protected by authentication mechanisms, including the use of Google API keys for access control and security.

Forwarding Input to Gemini Pro Pre-trained Model:

- Upon receiving the user input data, the backend server processes this information and prepares it for transmission to the Gemini Pro pre-trained model.
- The backend sends an API request to the Gemini Pro service, passing along the formatted user input data. This API call includes the necessary authentication credentials and parameters required to interact with the Gemini Pro model.

Processing by Gemini Pro Pre-trained Model:

- The Gemini Pro pre-trained model receives the incoming API request containing the user input data. This model is designed to leverage advanced artificial intelligence and machine learning techniques to analyze and process the input.
- Using its pre-trained algorithms, the Gemini Pro model interprets the user's dietary preferences, health objectives, and other relevant factors to generate personalized recommendations.

Generation and Return of Results:

- After processing the user input, the Gemini Pro model generates output based on its analysis. This output typically includes personalized dietary recommendations, meal plans, nutritional insights, and possibly educational resources tailored to the user's specific needs.

- The results generated by Gemini Pro are then returned to the backend server via another API call response. This response includes structured data or formatted text, depending on the application's design requirements.

Formatting and Display on the Frontend:

- Finally, the formatted results from Gemini Pro are received by the frontend application.
- The frontend processes these results for display, ensuring that the information is presented in a clear and user-friendly manner within the UI.
- Depending on the design, the frontend may include charts, graphs, textual recommendations, interactive elements, or other visual aids to enhance user understanding and engagement.

To accomplish this, we have to complete all the activities listed below:

Requirements Specification:-

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

- **Purpose:** This file lists all Python libraries that are necessary for the project to run.
- **Content:** Include libraries such as Flask (for web framework), requests (for making API calls), numpy (for numerical computations), pandas (for data handling), and any specific AI or NLP libraries required for interfacing with Gemini Pro.

Install the required libraries:

- **Execution:** Once the requirements.txt file is created, execute `pip install -rrequirements.txt` in the command line.
- **Purpose:** This command installs all dependencies listed in requirements.txt, ensuring that the project has all necessary libraries to function correctly.

Initialization of Google API Key

Generate Google API Key:

- **Process:** Obtain an API key from the Google Cloud Console by creating a new project and enabling the required APIs (such as Google Drive API for accessing PDF content).
- **Security:** Ensure the API key is stored securely and not exposed in public repositories or client-side code to prevent unauthorized use.

Initialize Google API Key:

- **Implementation:** Store the API key securely in a configuration file or environment variable.
- **Access Control:** Ensure proper access controls and permissions are set to restrict usage based on the project's requirements.

Interfacing with Pre-trained Model

Load the Gemini Pro pre-trained model:

- **Setup:** Use the appropriate library or framework (such as TensorFlow or PyTorch) to load the Gemini Pro model weights and architecture.

- **Integration:** Ensure the model is initialized correctly to accept input data and generate predictions.

Implement a function to get Gemini response:

- **Functionality:** Create a function that takes user input, formats it appropriately (e.g., tokenization for NLP models), and sends it to the Gemini Pro model for processing.
- **Response Handling:** Capture the response from Gemini Pro and prepare it for further processing or display.

Implement a function to read PDF content:

- **Function Design:** Develop a function that reads PDF files uploaded by users or accessed from specified locations (e.g., Google Drive).
- **Text Extraction:** Utilize libraries like PyPDF2 or pdfplumber to extract text content from PDF files accurately.

Write a prompt for Gemini model:

- **Prompt Definition:** Craft a specific prompt that guides the Gemini Pro model to generate relevant outputs based on user inputs.
- **Contextualization:** Ensure the prompt encapsulates the user's dietary preferences, health goals, and any other pertinent information required for personalized recommendations.

Model Deployment

Integrate with Web Framework:

- **Web Framework Selection:** Choose a suitable web framework like Flask or Django for hosting the application.

Endpoint Definition: Define endpoints within the framework to handle user requests, interact with the Gemini Pro model, and serve responses back to users.

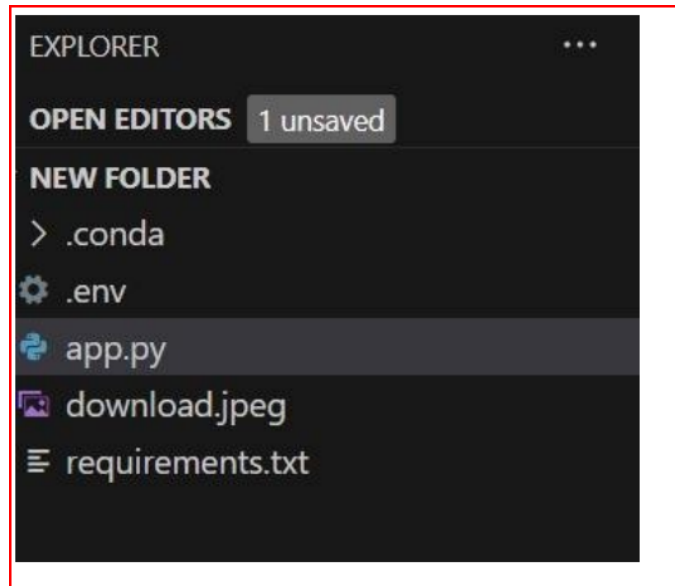
- **API Design:** Design API endpoints to accept inputs (e.g., user data or PDF uploads), process them using implemented functions, and return results.

Host the Application:

- **Deployment Strategy:** Deploy the web application to a hosting platform like Google Cloud Platform (GCP) or AWS.
- **Scalability Considerations:** Ensure the deployment is scalable to handle potential increases in traffic or user interactions.
- **Monitoring and Maintenance:** Implement monitoring tools and procedures to track application performance and maintain uptime

Project Structure:

Create the Project folder which contains files as shown below:



Images Folder:

- This folder is typically created within your project directory to store images that are used in the user interface (UI). Organizing images in a dedicated folder helps maintain clarity and makes it easier to manage assets related to the UI components of your application.

.env File:

- The .env file is a configuration file used to store sensitive data such as API keys, database credentials, and other environment variables. It is important to keep this file secure and out of version control by adding it to your .gitignore file (or equivalent for your version control system) to prevent exposing sensitive information publicly.

app.py:

- app.py serves as the primary application file in your project. It commonly houses both the backend logic (such as data processing or machine

learning model handling) and the frontend code (such as UI generation using frameworks like Streamlit or Flask). Keeping this file well-organized and structured is crucial for the maintainability and scalability of your application.

requirements.txt:

- This file lists all Python libraries that your application depends on. It specifies the exact versions of each library to ensure consistency across different environments and deployments. You can generate this file using `pip freeze > requirements.txt` after installing all necessary libraries for your project.

File Organization Best Practices:

- Adopting a structured approach to organizing files and directories within your project is essential for clarity and maintainability.

For instance:

- **Separation of Concerns:** Keep different types of files (like data, code, configuration) in separate directories.
- **Naming Conventions:** Use meaningful names for directories and files to easily understand their purpose.
- **Version Control:** Use a version control system like Git to track changes, collaborate with others, and maintain a history of your project's development.

Version Control Best Practices:

- Follow these practices to effectively manage your codebase:
 - **Regular Commits:** Make frequent, small commits with clear

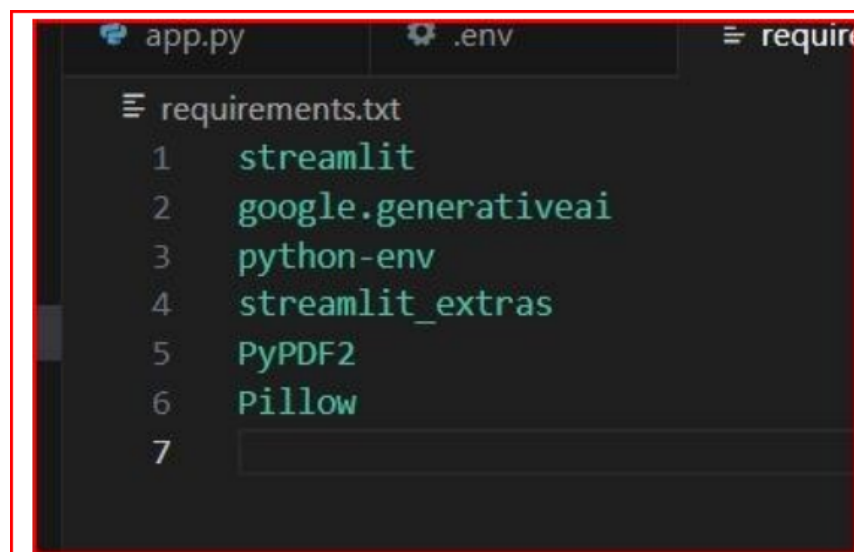
messages to document changes.

- **Branching Strategy:** Use branches for feature development, bug fixes, and experimentation, and merge them back to the main branch (e.g., main or master) after review.
- **Git Ignore:** Exclude files that shouldn't be tracked (e.g., .env, temporary files) using .gitignore.

Milestone1: 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.

Activity1: Create a requirements.txt file to list the required libraries.

A screenshot of a code editor with a dark background. The editor has three tabs at the top: 'app.py', '.env', and 'requirements.txt'. The 'requirements.txt' tab is active, showing a list of libraries. The text is as follows:

```
requirements.txt
1  streamlit
2  google.generativeai
3  python-env
4  streamlit_extras
5  PyPDF2
6  Pillow
7
```

Streamlit: Streamlit is a popular Python framework used for building interactive web applications for machine learning and data science

projects. It allows developers to create UI components using simple Python scripts, making it easy to deploy and sharedata-driven applications.

streamlit_extras: streamlit_extras typically refers to additional utilities and enhancements that extend the functionality of Streamlit applications beyond its corefeatures.

google-generativeai: This Python client library is used to interact with GenerativeAPIs provided by Google. These APIs typically involve pre-trained language models likeGemini Pro, enabling applications to generate text or perform natural language processing tasks.

python-dotenv: python-dotenv is a Python library used for managing environment variables stored in a .env file. It simplifies loading environment variables into your Python projects, which is particularly useful for handling sensitive information such asAPI keys and database credentials.

PyPDF2: PyPDF2 is a Python library for working with PDF documents. It provides functionalities for extracting text, merging or splitting PDFs, encrypting or decryptingPDFs, and more.

Pillow: Pillow is a fork of the Python Imaging Library (PIL), offering a wide range offunctionalities for opening, manipulating, and saving various image file formats in Python.

Activity2: Install the required libraries

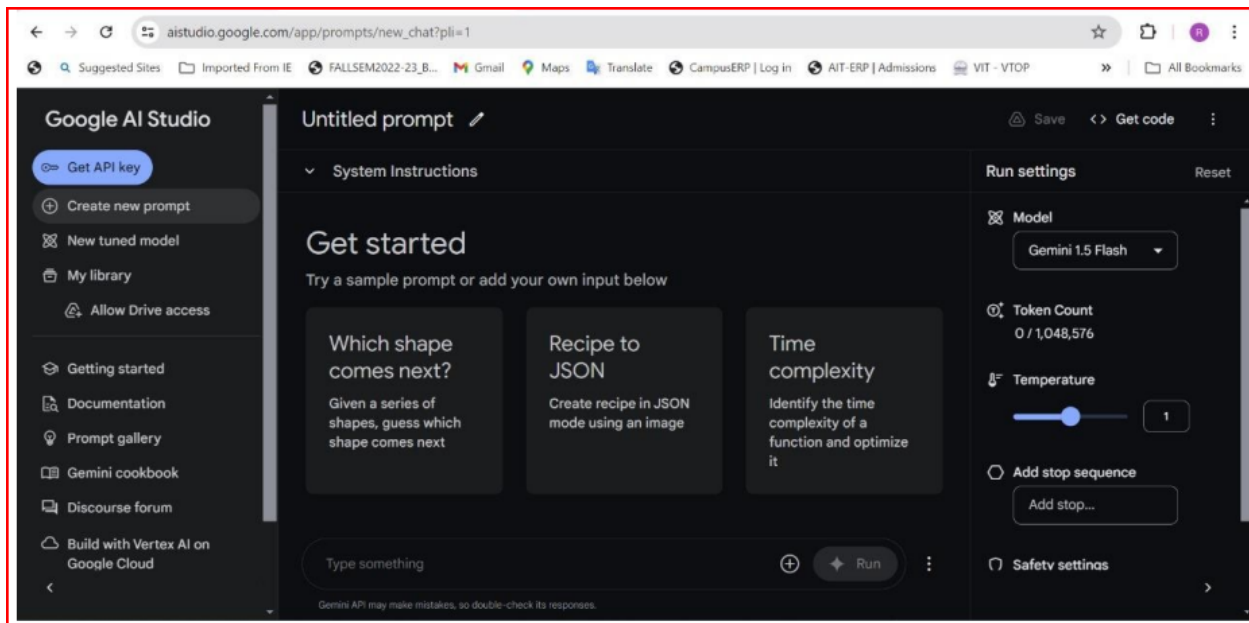
- Open the terminal and run the command: `pipinstall-rrequirements.txt`

- This command installs all the libraries listed in the requirements.txt file.

Milestone2: Initialization of Google API key

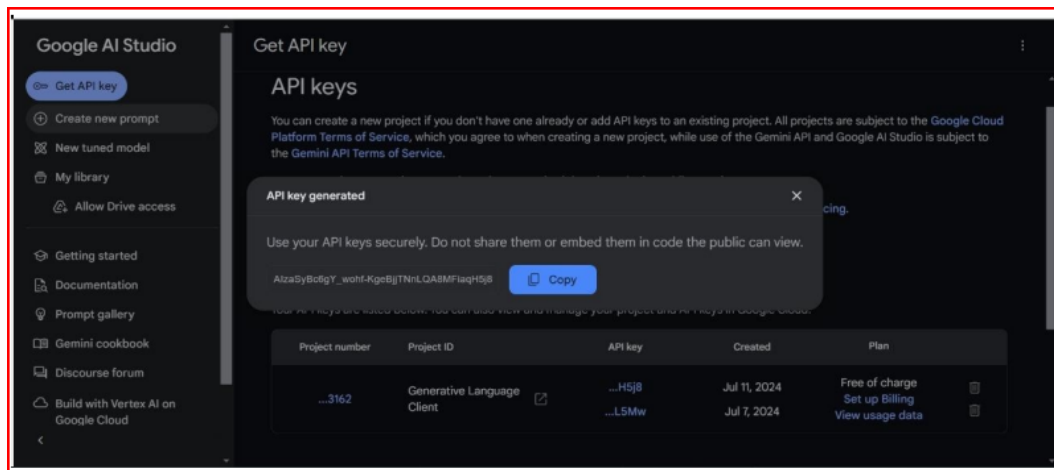
The Google API key is a secure access token provided by Google, enabling developers to authenticate and interact with various Google APIs. It acts as a form of identification, allowing users to access specific Google services and resources. This key plays a crucial role in authorizing and securing API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

Activity1: Generate Google API key



After signing in to your account, navigate to the 'Get an API Key' option. Clicking on this option will redirect you to another web page as shown

below.



Next, click on 'Create API Key' and choose the generative language client as the project. Then, select 'Create API key in existing project'.

Copy the newly generated API key as it is required for loading the Gemini Pro pre-trained model.

Activity2: Initialize Google API Key



1. Create a .env file and define a variable named GOOGLE_API_KEY.
2. Assign the copied Google API key to this variable.

3. Paste the API key obtained from the previous steps here.

Milestone3: 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.

Activity1: Load the Gemini Pro API

```
app.py > ...
1  ### Health Management APP
2  from dotenv import load_dotenv
3
4  load_dotenv() ## load all the environment variables
5  import streamlit as st
6  import os
7  import google.generativeai as genai
8  from PIL import Image
9
10 genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
11
12 ## Function to load Google Gemini Pro Vision API And get respon
```

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.generative ai for utilizing Google's Generative AI capabilities, and PIL.Image for image processing. The `gen ai.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.

Activity2: Implement a function to get Gemini response

```
## Function to load Google Gemini Pro Vision API And get response

def get_gemini_repsonse(input,image,prompt):
    model=genai.GenerativeModel('gemini-pro-vision')
    response=model.generate_content([input,image[0],prompt])
    return response.text
```

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

Activity3: Implement a function to read the Image and set the imageformat for Gemini Pro model Input

```

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

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

##initialize our streamlit app

```

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 `File Not Found Error`, 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.

Activity4: Write a prompt for Gemini model


```
input_prompt="""
You are an expert in nutritionist where you need to see the food items from the image
and calculate the total calories, also provide the details of every food items with calories intake
is below format

1. Item 1 - no of calories
2. Item 2 - no of calories
----
----
"""
```

The variable `input_prompt` is a multi-line string designed as a prompt for a nutritionist AI model. It instructs the model to analyze an image of food items, identify each food item, and calculate the total calories. Additionally, the model list 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.

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.

Activity1: Integrate with Web Framework

```

st.set_page_config(page_title="AI Nutritionist App")

st.header("AI Nutritionist App")
input=st.text_input("Input Prompt: ",key="input")
uploaded_file = st.file_uploader("Choose an image...", 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("Tell me the total calories")

```

1. If "Tell me the total calories but on click":

```

62  ## If submit button is clicked
63
64  if submit:
65      image_data=input_image_setup(uploaded_file)
66      response=get_gemini_repsonse(input_prompt,image_data,input)
67      st.subheader("The Response is")
68      st.write(response)
69

```

This code initializes a Streamlit application titled "AI Nutritionist App" 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 "Tell me the total calories" 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.

Activity2: Host the Application

Launching the Application:

- a. To host the application, got other terminal ,type-streamlit runapp.py
- b. Here app.py refers to a python script.

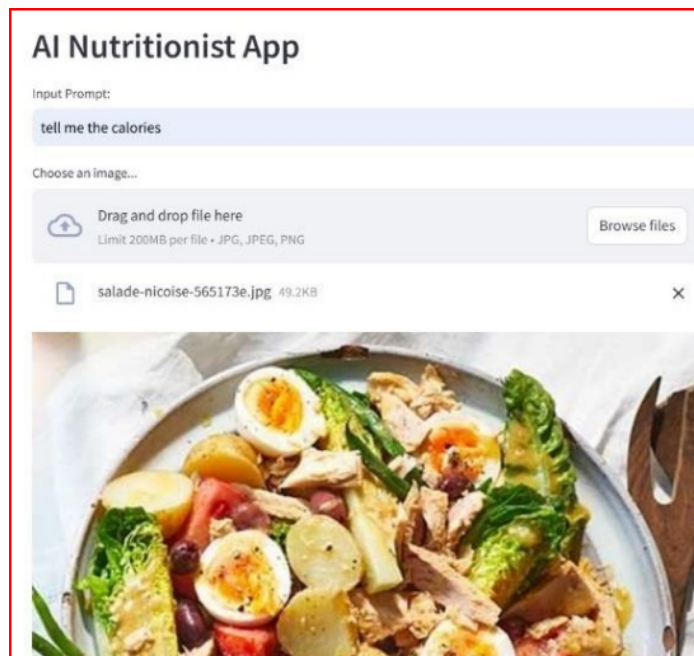
You can now view your Streamlit app in your browser.

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

Network URL: `http://192.168.29.80:8501`

Run the command to get the below results

Input 1:



Output-1:

The Response is

The image contains a salad with the following ingredients:

1. Lettuce
2. Tomatoes
3. Potatoes
4. Eggs
5. Tuna
6. Green beans
7. Olives
8. Olive oil

The calorie counts for each ingredient are as follows:

1. Lettuce: 5 calories per cup
2. Tomatoes: 25 calories per cup
3. Potatoes: 110 calories per cup
4. Eggs: 70 calories per egg
5. Tuna: 180 calories per cup
6. Green beans: 40 calories per cup
7. Olives: 10 calories per olive
8. Olive oil: 120 calories per tablespoon

The total number of calories in the salad is 765 calories.

Input-2:

AI Nutritionist App

Input Prompt:

tell me the calories

Choose an image...



Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files



download.jpeg.jpg 9.6KB



Output-2:



Uploaded image.

Tell me the total calories

The Response is

1. lasagna 366 calories
2. carrot 40 calories
3. spinach 7 calories
4. cheese 113 calories