

Flavour Fusion: AI-Driven Recipe Blogging

Project Description:

Flavour Fusion: AI-Driven Recipe Blogging is a web application that leverages Google's Generative AI to create unique and customized recipe blogs. The app provides users with the ability to input a topic and specify the desired word count for their recipe blog. Using the specified parameters, the AI generates detailed and engaging recipe content. Additionally, the app includes a fun feature where it tells a programmer joke to entertain users while the AI is generating the content.

Scenario 1: Creating a Vegan Recipe Blog

A food blogger specializing in vegan recipes opens the Flavour Fusion app and inputs "Vegan Chocolate Cake" with a 1200-word count. As the app generates the content, it entertains them with a programmer joke. The AI quickly delivers a detailed and creative recipe. The blogger reviews the well-crafted content and incorporates it into their blog, ready to share with their audience.

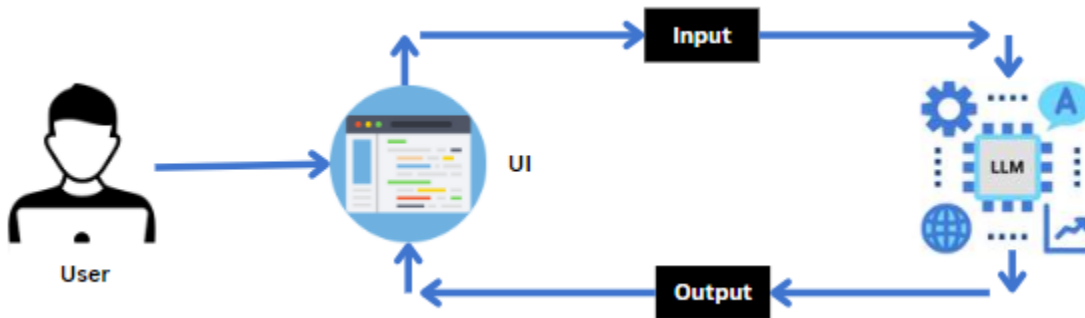
Scenario 2: Crafting a Quick Weeknight Dinner Recipe Blog

A busy professional looking for quick dinner ideas uses the Flavour Fusion app, inputting "Quick Weeknight Dinners" and specifying an 800-word count. The app provides a lighthearted joke while generating the content. The AI produces a concise yet practical recipe blog filled with quick and easy dinner ideas. The user finds the suggestions useful and incorporates them into their weeknight meal planning.

Scenario 3: Developing a Gluten-Free Baking Recipe Blog

A baker specializing in gluten-free recipes accesses the Flavor Fusion app to generate new content for their blog. They enter "Gluten-Free Bread" as the topic and select a 1500-word count. The app entertains with a joke during the content creation process. The AI delivers a comprehensive and well-detailed recipe. The baker reviews the high-quality content and publishes it on their gluten-free baking blog, confident it will be valuable to their readers.

Architecture:



Project Flow:

1. Users input a topic and specify the desired length of the blog post through the Streamlit UI.
2. The input topic and length are sent to the Gemini 1.5 Flash language model, which is integrated into the backend.
3. Gemini 1.5 Flash processes the input and generates a blog post based on the user's specifications.
4. The model autonomously creates a well-structured, engaging blog post tailored to the specified topic and word count.
5. The generated blog post is sent back to the frontend for display on the Streamlit app.
6. Users can customize the blog post further if desired and export or copy the content for their use.

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

- Initialize Gemini 1.5 Flash:
- Generate Gemini 1.5 Flash API
- Initialize the pre-trained model
- Interfacing with Pre-trained Model
 - Blog Generation
- Model Deployment
 - Deploy the application using Streamlit

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- LLM & Gemini 1.5 Flash :

A large language model is a type of artificial intelligence algorithm that applies neural network techniques with lots of parameters to process and understand human languages or text using self-supervised learning techniques. Tasks like text generation, machine translation, summary writing, image generation from texts, machine coding, chat-bots, or Conversational AI are applications of the Large Language Model. Examples of such LLM models are Chat GPT by open AI, BERT (Bidirectional Encoder Representations from Transformers) by Google, etc.

<https://www.geeksforgeeks.org/large-language-model-llm/>

<https://cloud.google.com/vertex-ai/docs/generative-ai/learn-resources>

- Streamlit:

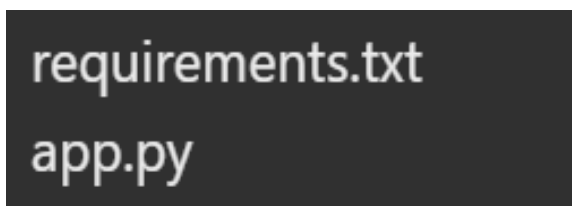
Basic knowledge of building interactive web applications using Streamlit.

Understanding of Streamlit's UI components and how to integrate them with backend logic.

<https://www.datacamp.com/tutorial/streamlit>

Project Structure:

Create the Project folder which contains application file as shown below



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.

```
≡ requirements.txt
1  streamlit
2  google.generativeai
3
```

Activity 2: Install the required libraries.

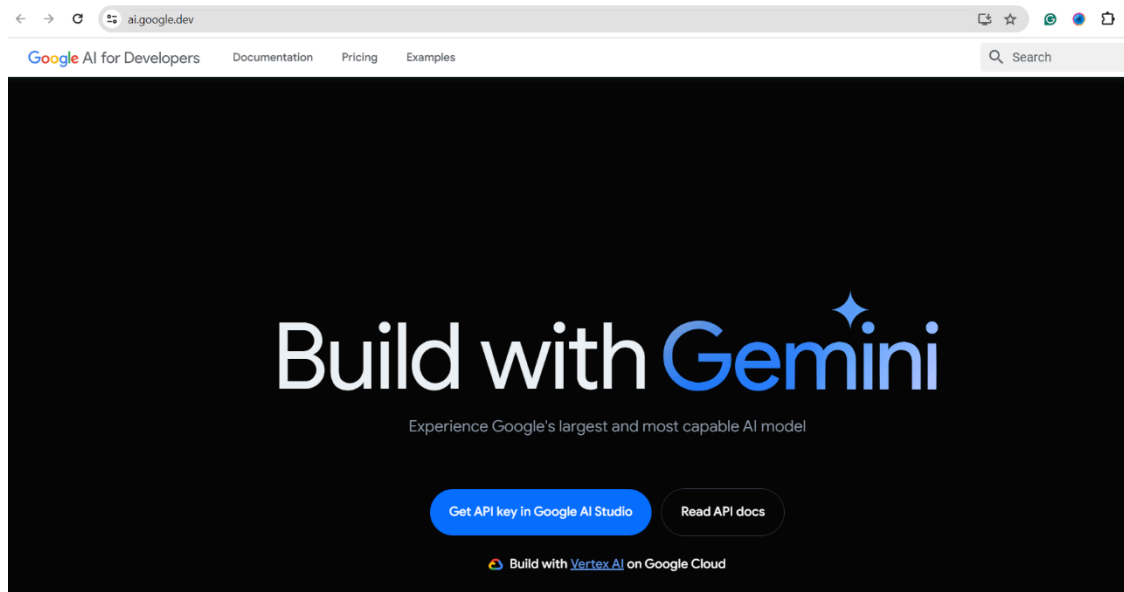
```
(myenv) C:\genai>pip install -r requirements.txt
```

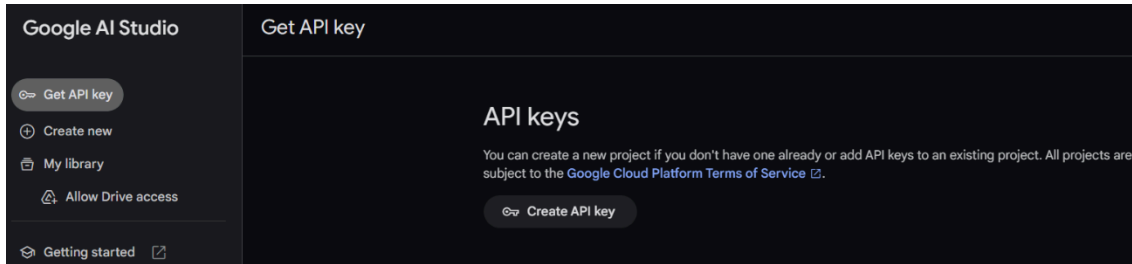
Milestone 2: Initializing the Model

For initializing the model we need to generate Google API key.

Activity 1: Generate Google API key

- Click on the link (<https://developers.generativeai.google/>).
- Then click on “Get API key in Google AI Studio”.
- Click on “Get API key” from the right navigation menu.
- Now click on “Create API key”. (Refer the below images)
- Copy the API key.





Activity 2: Initialize the pre-trained model

Activity 2.1: Import necessary files

```
import streamlit as st
import google.generativeai as genai
import random
```

- Streamlit, a popular Python library, is imported as **st**, enabling the creation of user interfaces directly within the Python script.
- **Google Generative AI (genai)**: Imported to interact with Google's Gemini 1.5 Flash model, allowing AI-generated text responses.
- A built-in Python module that provides functions to generate 'random' choices, such as selecting a joke from a predefined list.

Activity 2.2: Configuration of the Gemini Pro API

```
api_key = "AIzaSyCr3KHAWQYxUqDJ3U6i1gPEBKcZMn4VeOQ"
genai.configure(api_key=api_key)
```

- Configuring the API key: The configure function is used to set up or configure the Google API with an API key. The provided API key, in this case, is "AIzaSyCr3KHAWQYxUqDJ3U6i1gPEBKcZMn4VeOQ".

This ensures that the application can communicate with Google's Gemini 1.5 Flash model.

Activity 2.3: Define the model to be used

An instance of GenerativeModel is created with the model name "gemini-1.5-flash"

```
generation_config = {  
    "temperature": 0.75,    # Controls creativity  
    "top_p": 0.95,         # Controls diversity of responses  
    "top_k": 64,           # Limits the number of high-probability w  
    "max_output_tokens": 8192, # Limits response length  
    "response_mime_type": "text/plain",  
}
```

Milestone 3: Interfacing with Pre-trained Model

In this milestone, we will build a function that interacts with the **Gemini 1.5 Flash model** to generate recipe blogs dynamically. Additionally, we enhance user engagement by displaying a **random programming joke** while the model processes the recipe request.

Activity 1: Create a function for random jokes

```
def get_joke():  
    jokes = [  
        "Why don't programmers like nature? It has too many bugs.",  
        "Why do Java developers wear glasses? Because they don't see",  
        "How many programmers does it take to change a light bulb? 1",  
        "Why did the developer go broke? Because he used up all his",  
        "Why do programmers prefer dark mode? Because light attract",  
    ]  
    return random.choice(jokes)
```

A list of programming-related jokes is defined.

The random.choice() function selects one joke at random from the list.

This joke will be displayed before the AI-generated recipe is shown.

Activity 2: Create a Function to Generate Recipes

```
def generate_recipe(topic, word_count):
    try:
        # Display processing message
        st.write("🌀 Generating your recipe...")
        st.write(f"🤖 While I work on your blog, here's a joke for you: \n\n {get_joke()}")

        # Initialize the model for content generation
        chat_session = genai.GenerativeModel("gemini-1.5-flash").start_chat()

        # Formulate the prompt for AI
        prompt = f"Write a recipe blog on '{topic}' with {word_count} words"

        # Generate the content
        response = chat_session.send_message(prompt)

        # Display success message
        st.success("✅ Your recipe is ready!")
    except Exception as e:
        st.error(f"An error occurred: {e}")
    return response.text
```

Explanation:

- **User Notification:**
 - `st.write("🌀 Generating your recipe...")` informs users that processing has started.
 - `st.write(f"🤖 While I work on your blog, here's a joke for you: \n\n {get_joke()}")` displays a joke to engage users.
- **AI Model Interaction:**
 - A new chat session is created using `genai.GenerativeModel("gemini-1.5-flash").start_chat()`.
 - The prompt specifies that the model should generate a recipe blog based on user inputs.
- **Response Handling:**
 - `response = chat_session.send_message(prompt)` sends the request and retrieves the AI-generated recipe.
 - `st.success("✅ Your recipe is ready!")` confirms that the process is complete.
 - If an error occurs, it is caught by `except Exception as e`, and an error message is displayed.

Milestone 4: Model Deployment

In this milestone, we are deploying the created model using streamlit. Model deployment using Streamlit involves creating a user-friendly web interface, enabling users to interact with the model through a browser. Streamlit provides easy-to-use tools for developing and deploying data-driven applications, allowing for seamless integration of models into web-based applications.

Activity 1: Set Up the Streamlit

```
def main():
    st.title("Flavour Fusion: AI-Driven Recipe Blogging 🍳📝")
    st.write("### Generate AI-powered recipe blogs with ease!")

    topic = st.text_input("Enter your recipe topic:", placeholder="e.g., Vegetarian")
    word_count = st.number_input("Word count:", min_value=100, max_value=2000)

    if st.button("Generate Recipe"):
        if topic and word_count:
            recipe = generate_recipe(topic, word_count)
            if recipe:
                st.text_area("Generated Recipe:", recipe, height=300)
                st.download_button("Download Recipe", recipe, file_name=f"{topic}.txt")
            else:
                st.warning("Please enter a topic and word count.")
```

Activity 2: Running the Web Application

Start the Streamlit App

1. Open Anaconda Prompt or a terminal.
2. Navigate to the project folder where `app.py` is saved.
3. Run the command:

```
streamlit run app.py
```

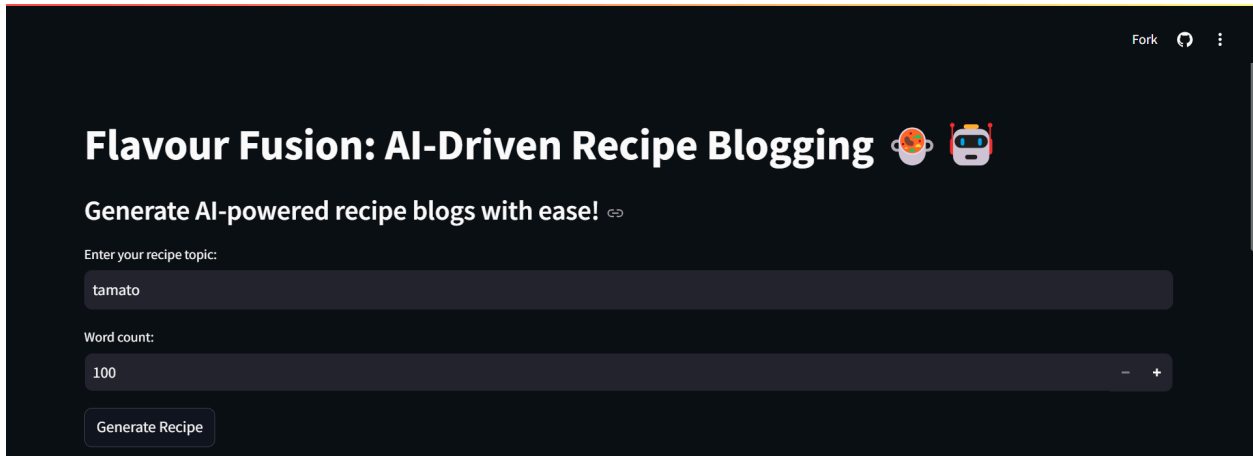
4. Open the **localhost URL** that appears to access the web application.

Now, the application will open in the web browser,

Expected Output:

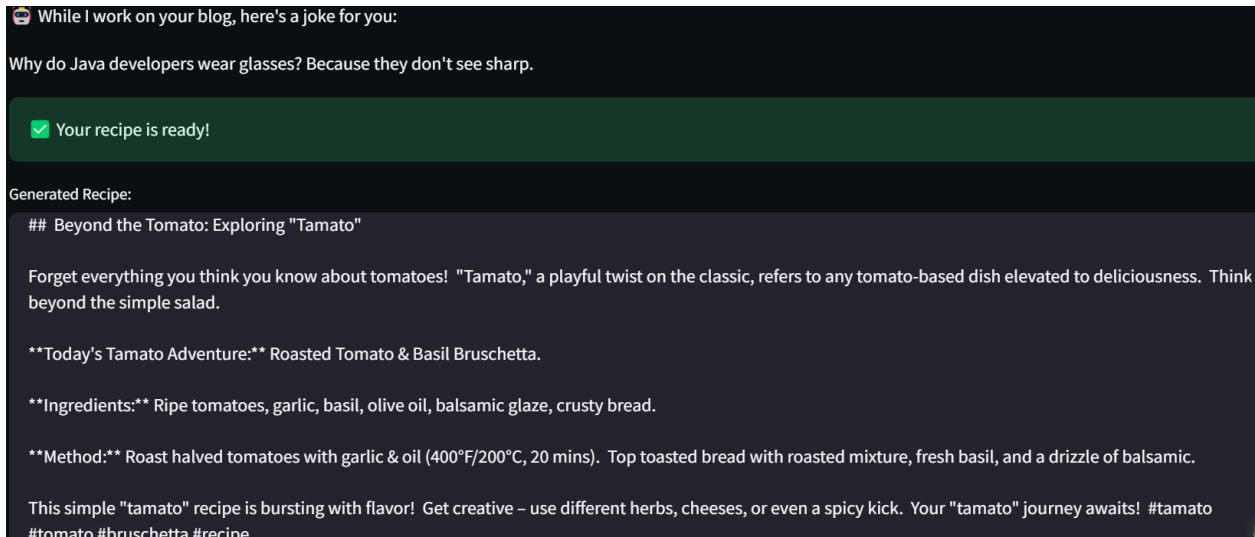
- Users enter ingredients like "tomato, cheese, basil".
- Selects Word Count
- Click "Generate Recipe".

After giving the input:



The screenshot shows a web application titled "Flavour Fusion: AI-Driven Recipe Blogging" with a dark theme. At the top right, there are links for "Fork", a refresh icon, and a menu icon. The main heading is "Flavour Fusion: AI-Driven Recipe Blogging" followed by two robot icons. Below this is a sub-heading "Generate AI-powered recipe blogs with ease!" with a double arrow icon. The form has two input fields: "Enter your recipe topic:" with the text "tamato" and "Word count:" with the value "100". A "Generate Recipe" button is at the bottom left of the form area.

The Output generated:



The screenshot displays the output of the application. It starts with a message from a robot icon: "While I work on your blog, here's a joke for you: Why do Java developers wear glasses? Because they don't see sharp." Below this is a green notification bar with a checkmark icon and the text "Your recipe is ready!". The main section is titled "Generated Recipe:" and contains the following text: "## Beyond the Tomato: Exploring "Tamato"" followed by a paragraph: "Forget everything you think you know about tomatoes! "Tamato," a playful twist on the classic, refers to any tomato-based dish elevated to deliciousness. Think beyond the simple salad." Then, "**Today's Tamato Adventure:**" Roasted Tomato & Basil Bruschetta." followed by "**Ingredients:**" Ripe tomatoes, garlic, basil, olive oil, balsamic glaze, crusty bread." and "**Method:**" Roast halved tomatoes with garlic & oil (400°F/200°C, 20 mins). Top toasted bread with roasted mixture, fresh basil, and a drizzle of balsamic." The output concludes with a paragraph: "This simple "tamato" recipe is bursting with flavor! Get creative – use different herbs, cheeses, or even a spicy kick. Your "tamato" journey awaits! #tamato #tomato #bruschetta #recipe".

Conclusion:

The **Flavour Fusion: AI-Driven Recipe Blogging** project successfully demonstrates the power of **Google's Generative AI** in creating personalized and engaging recipe blogs. By leveraging **Streamlit** for an intuitive user interface and **Gemini 1.5 Flash** for AI-powered content generation, we built a seamless and interactive web application.