

## **Flavour Fusion: AI-Driven Recipe Blogging**

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

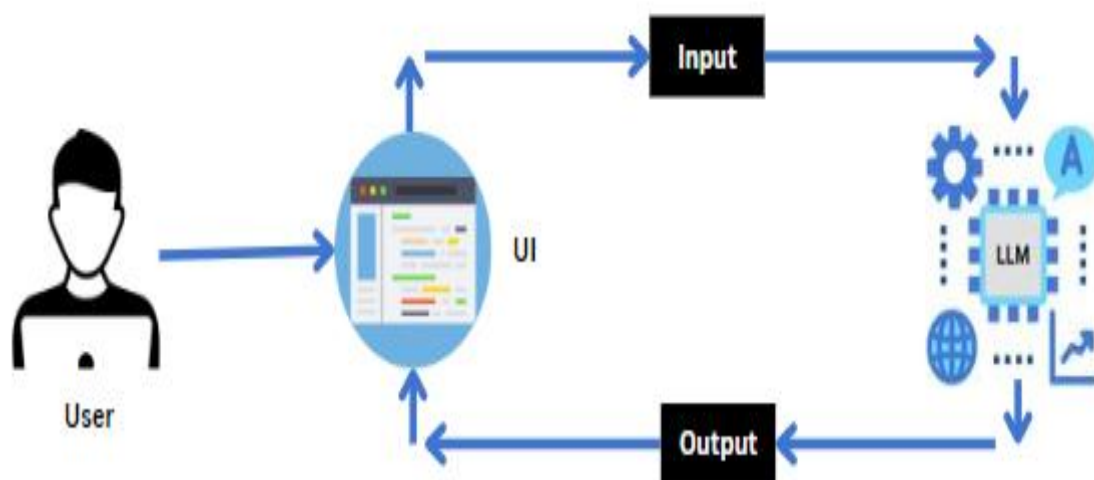
A food blogger specializing in vegan recipes opens the Flavour Fusion app and inputs "Vegan Chocolate Cake" with a 1200-word count. As the app generates the content, it entertains them with a programmer joke. The AI quickly delivers a detailed and creative recipe. The blogger reviews the well-crafted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



## Project Flow

### 1. User Input via Streamlit UI:

Users input a prompt (e.g., topic, keywords) and specify parameters such as the desired length, tone, or style through the Streamlit interface.

### 2. Backend Processing with Generative AI Model:

The input data is sent to the backend, where it interfaces with the selected Generative AI model (e.g., GPT-4, Gemini, etc.).

The model processes the input, generating text based on the specified parameters and user input.

### 6. Content Generation:

The AI model autonomously creates content tailored to the user's specifications.

This could be a blog post, poem, article, or any other form of text.

### 7. Return and Display Generated Content:

The generated content is sent back to the frontend for display on the Streamlit app.

The app presents the content to the user in an easily readable format.

### 11. Customization and Finalization:

Users can further customize the generated content through the Streamlit UI if desired. This might include editing text, adjusting length, or altering tone.

### 12. Export and Usage:

Once satisfied, users can export or copy the content for their use, such as saving it to a file or directly sharing it.

## 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>

## 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.

### Requirements Specification

- Install the libraries
- pip install streamlit
- pip install google.generativeai

Install the required libraries.

## 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.

The image shows a web browser window with the address bar displaying `ai.google.dev`. The page header includes the Google AI for Developers logo and navigation links for Documentation, Pricing, and Examples, along with a search bar. The main content area features a large heading "Build with Gemini" with a subtitle "Experience Google's largest and most capable AI model". Below this are two buttons: "Get API key in Google AI Studio" and "Read API docs". A link at the bottom says "Build with Vertex AI on Google Cloud".

The second part of the image shows the "Get API key" page in Google AI Studio. The left sidebar contains links for "Get API key", "Create new", "My library", "Allow Drive access", and "Getting started". The main content area is titled "API keys" and includes a paragraph explaining that users can create a new project or add API keys to an existing one, with a link to the Google Cloud Platform Terms of Service. A "Create API key" button is located at the bottom.

Google AI for Developers   Documentation   Pricing   Examples   Search

# Build with Gemini

Experience Google's largest and most capable AI model

[Get API key in Google AI Studio](#)   [Read API docs](#)

[Build with Vertex AI on Google Cloud](#)

## Google AI Studio

### Get API key

[Get API key](#)   [Create new](#)   [My library](#)   [Allow Drive access](#)

## API keys

You can create a new project if you don't have one already or add API keys to an existing project. All projects are subject to the [Google Cloud Platform Terms of Service](#).

[Create API key](#)

[Getting started](#)

## Joke Generation (get\_joke() Function)

```
# Function to generate a joke
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 sharp.",
        "Why was the JavaScript developer sad? Because he didn't know how to 'null' his feelings.",
        "Why don't programmers like nature? It has too many bugs.",
        "Why do programmers prefer dark mode? Because light attracts bugs!",
        "Why do Java developers wear glasses? Because they don't see sharp.",
        "Why was the JavaScript developer sad? Because he didn't know how to 'null' his feelings.",
        "Why do Python programmers prefer using snake_case? Because it's easier to read!",
        "How many programmers does it take to change a light bulb? None, that's a hardware problem.",
        "Why did the developer go broke? Because he used up all his cache.",
        "Why do programmers always mix up Christmas and Halloween? Because Oct 31 == Dec 25.",
        "Why did the programmer get kicked out of the beach? Because he kept using the 'C' language!",
        "Why was the computer cold? It left its Windows open."
    ]
    return random.choice(jokes)
```

1. This function selects and returns a random programming joke from a predefined list
2. A list of jokes is stored in the jokes list.
3. The random.choice(jokes) function is used to randomly select and return a joke from this list.
4. This function is called within the recipe generation function to provide a light-hearted joke while the recipe is being created.

## Recipe Generation (recepie\_generation() Function)


```
def recepie_generation(user_input, word_count):
    """
    Function to generate a blog based on user input and word count.
    Parameters:
    user_input (str): The topic for the blog.
    word_count (int): The desired number of words for the blog.
    Returns:
    str: The generated blog content.
    """

    # Display a message while the blog is being generated
    st.write("### 🌀 Generating your recepie...")
    st.write(f"While I work on creating your blog, here's a little joke to keep you entertained:\n\n**{get_joke()}**")
    # Start a chat session with the input topic and word count
    chat_session = model.start_chat(
        history=[
            {
                "role": "user",
                "parts": [
                    f"Write a recepie based on the input topic: {user_input} and number of words: {word_count}\n",
                ],
            },
        ]
    )
    try:
        # Generate a response for the new input
        response = chat_session.send_message(user_input)
        st.success("👉 Your recepie is ready!")
        return response.text
    except Exception as e:
        st.error(f"Error generating blog: {e}")
        return None
```

- Generates a recipe based on user input and a specified word count.
- user\_input (str): The topic or theme for the recipe.
- word\_count (int): The desired length of the recipe Generates a recipe based on user input and a specified word count.
- in words.
- Displays a message indicating that the recipe is being generated.
- Calls get\_joke() to display a joke to the user while waiting.
- Starts a chat session using the Gemini 1.5 Flash model, passing the user input and word count.
- Attempts to generate the recipe and returns the generated text if successful.
- Handles any exceptions by displaying an error message.



# RecepieMaster: AI-Powered Blog Generation

 Hello! I'm recepieMaster, your friendly robot. Let's create a fantastic recepie together! ↻

Topic

malai kofta

Number of words

555

- +


Generate recepie

## Generating your recepie...

While I work on creating your blog, here's a little joke to keep you entertained:

Why do Java developers wear glasses? Because they don't see sharp.

Why do Java developers wear glasses? Because they don't see sharp.

 Your recepie is ready!

## Indulge in Creamy Heaven: Malai Kofta Recipe

Serves: 4-6 Prep time: 45 minutes Cook time: 30 minutes

Ingredients:

For the Kofta:

- 2 large potatoes, boiled and mashed
- 1 cup paneer (Indian cottage cheese), grated
- 1/2 cup finely chopped mixed vegetables (carrots, peas, beans)
- 1/4 cup cashew nuts, finely chopped
- 2 tbsp raisins
- 1 tbsp ginger-garlic paste
- 1 tsp green chili, finely chopped
- 1/2 tsp red chili powder
- 1/2 tsp garam masala powder
- 1/4 tsp turmeric powder
- 1/4 cup fresh coriander leaves, chopped
- 2 tbsp cornflour
- Salt to taste
- Oil for deep frying

**For the Gravy:**

- 2 tbsp oil
- 1 large onion, finely chopped
- 1 inch ginger, grated
- 4-5 garlic cloves, minced
- 2 medium tomatoes, pureed
- 1 tsp red chili powder
- 1 tsp coriander powder
- 1/2 tsp turmeric powder
- 1/2 tsp garam masala powder
- 1/2 cup fresh cream
- 1/4 cup cashew paste (soak 1/4 cup cashews in warm water for 30 minutes, then blend to a smooth paste)
- 1/4 cup water
- 1 tbsp sugar
- Salt to taste
- Fresh coriander leaves for garnish

**Instructions:****Making the Kofta:**

1. In a large bowl, combine the mashed potatoes, grated paneer, chopped vegetables, cashew nuts, raisins, ginger-garlic paste, green chili, red chili powder, garam masala powder, turmeric powder, coriander leaves, cornflour and salt.
2. Mix well to form a smooth and pliable dough. If the dough feels too sticky, add a little more cornflour.
3. Divide the dough into equal-sized portions and shape them into smooth, round balls.
4. Heat oil in a deep pan over medium heat. Carefully slide the kofta balls into the hot oil and deep fry until golden brown and crispy.
5. Remove the koftas from the oil and place them on a plate lined with paper towels to absorb excess oil.

**Making the Gravy:**

1. In a separate pan, heat oil over medium heat. Add the chopped onions and sauté until translucent.
2. Add the grated ginger and minced garlic, and sauté for another minute until fragrant.
3. Add the tomato puree, red chili powder, coriander powder, turmeric powder, and garam masala powder. Cook for 5-7 minutes, stirring frequently, until the oil separates from the masala.
4. Stir in the cashew paste and water, and bring the gravy to a boil. Reduce the heat to low and simmer for 10-15 minutes, stirring occasionally, until the gravy thickens.
5. Add the fresh cream and sugar to the gravy, and mix well. Adjust the salt to your liking.
6. Gently add the fried koftas to the gravy and let them simmer for another 5 minutes, allowing them to soak up the flavors of the rich gravy.



**Serving:**

Garnish with fresh coriander leaves and serve the Malai Kofta hot with naan, roti, or jeera rice for a truly indulgent and unforgettable meal.

**Tips:**

- For extra richness, you can add a tablespoon of butter to the gravy along with the cream.
- Adjust the spice level to your preference by adding more or less green chilies and chili powder.
- You can also shallow fry the koftas instead of deep frying for a lighter option.
- To make the koftas ahead of time, you can fry them and store them in an airtight container in the refrigerator for up to 2 days. Reheat them in the gravy before serving.

Enjoy the creamy, flavorful goodness of Malai Kofta!