Flavour Fusion : AI-Driven Recipe Blogging

1. INTRODUCTION

- **PROJECT TITLE**: Flavour Fusion : AI-Driven Recipe Blogging
- **TEAM MEMBERS :**
- **Team Leader :** Kontham Sri Chakradhar
- **Team member :** Kumari Kengava
- **Team member :** Lalam Deevena Kumar
- **Team member :** Sheik Sattar
- **Team member :** Sriramula Mahesh

2. PROJECT OVERVIEW

- **PURPOSE**

The purpose of this project is to develop an AI-powered recipe generator that creates detailed and structured recipes based on user input. This application uses Google Gemini Generative AI to automatically generate recipe content including introduction, ingredients, cooking instructions, tips, and conclusion. The goal of the project is to demonstrate the practical implementation of Generative AI, API integration, and web application development using Python and Streamlit. It also helps users quickly generate high-quality recipe blogs without manual writing, saving time and effort. This project serves as a learning tool for understanding modern AI technologies and building real-world AI applications.

- **FEATURES**

AI-powered recipe generation using Google Gemini API
• User input for recipe topic and word count
• Automatic generation of complete recipe blogs
• Interactive and user-friendly Streamlit web interface
• Displays random joke during recipe generation for better user experience
• Real-time content generation using Generative AI
• Error handling for smooth operation
• Fast and efficient response system
• Simple and clean application design
• Easy to use and scalable project structure

3. ARCHITECTURE

- **Frontend: Streamlit (Python-based UI)**

The frontend of this application is built using Streamlit, a Python library used to create interactive web applications. Streamlit provides a simple and efficient interface where users can enter the recipe topic and desired word count. It handles user input, displays jokes during processing, and shows the generated recipe in real time. Streamlit manages the entire user interface including buttons, text input fields, and output display without requiring HTML, CSS, or JavaScript.

Backend: Python with Gemini API Integration

- The backend is implemented using Python, which handles the core logic of the application. The backend receives user input from the Streamlit interface, processes it, and sends a prompt request to the Google Gemini Generative AI API. The Gemini model generates the recipe content based on the given input. The backend then receives the response from the API and sends it back to the frontend for display. Python manages prompt creation, API communication, response handling, and error management.
- **AI Service Integration: Google Gemini API**

The application integrates with the Google Gemini API to generate AI-powered recipe content. The API processes the input prompt and returns structured text output. This integration enables real-time content generation using advanced large language models.

- **Database: Not Required**

This project does not use a database because it generates content dynamically in real time. All data is processed temporarily during execution, and no user data is stored permanently. This makes the application lightweight, fast, and easy to maintain.

4. SETUP INSTRUCTIONS

- **Prerequisites**
- Before running this project, ensure the following software and tools are installed on your system:
- • Python 3.9 or higher
  • pip (Python package manager)
  • Streamlit library
  • Google GenAI Python SDK (`google-genai`)
  • Internet connection for Gemini API access
  • Google Gemini API key (from Google AI Studio)
  • Web browser (Chrome, Edge, or Firefox recommended)
- You can download Python from:
  https://www.python.org/downloads/
- You can create a Gemini API key from:
  https://aistudio.google.com/app/apikey
- **Installation**
- Follow these steps to set up and run the project:
- **Step 1: Clone the repository**
- Open Command Prompt or Terminal and run:
- `git clone https://github.com/YOUR_USERNAME/RecipeMaster-AI-Recipe-Generator.git`
- `cd RecipeMaster-AI-Recipe-Generator`
- **Step 2: Install required dependencies**
- Install required Python libraries using pip:
- `pip install streamlit`
- `pip install google-genai`
- **Step 3: Set up API key**
- Open the `app.py` file and replace:

- `client = genai.Client(api_key="YOUR_API_KEY")`
- with your actual Gemini API key:
- `client = genai.Client(api_key="AIzaSyXXXXXXXXXXXX")`
- **Step 4: Run the application**
- Start the Streamlit server using:
- `python -m streamlit run app.py`
- **Step 5: Open in browser**
- The application will automatically open in your browser at:
- `http://localhost:8501`
- **Environment Variables (Optional Best Practice)**
- Instead of hardcoding API key, you can use environment variable:
- Windows:
- `set GEMINI_API_KEY=your_api_key_here`
- Then modify code:
- `client = genai.Client(api_key=os.getenv("GEMINI_API_KEY"))`

5. FOLDER STRUCTURE

- This project follows a simple and clean folder structure since it is built using Python and Streamlit. Streamlit handles both frontend and backend functionalities within a single application file.

```
RecipeMaster/
│
├── app.py
├── requirements.txt
├── README.md
└── .gitignore
```

Application File (app.py)

The `app.py` file is the main file of the project and contains the entire application logic. It includes:

• Streamlit frontend user interface
• User input fields (recipe topic and word count)
• Integration with Google Gemini API
• Recipe generation function
• Joke generation function
• Output display logic
• Error handling

- This file acts as both frontend and backend of the application.
- **Requirements File (requirements.txt)**

- The `requirements.txt` file contains the list of required Python libraries needed to run the project, such as:• streamlit
  • google-genaiThis file helps install all dependencies easily using pip.
- **README File (README.md)**
- The README.md file contains project documentation including:
  - Project description
    • Installation instructions
    • Usage guide
    • Technologies use

- **.gitignore File**
  This file is used to exclude unnecessary files such as cache files, virtual environments, and API keys from being uploaded to GitHub.

6. RUNNING THE APPLICATION

- **Step 1: Open Project Folder**
  Open Command Prompt or Terminal and navigate to the project folder:
  `cd RecipeMaster`
  Example:
  `cd Desktop\FlavourFusion`

- **Step 2: Ensure Dependencies are Installed**
  Make sure required libraries are installed:
  `pip install streamlit`
  `pip install google-genai`
  If requirements.txt is available, run:
  `pip install -r requirements.txt`

- **Step 3: Add Gemini API Key**
  Open the `app.py` file and ensure your API key is added:
  `client = genai.Client(api_key="YOUR_API_KEY")`
  Replace with your actual API key.

- **Step 4: Run the Streamlit Application**
  Start the application using the following command:
  `python -m streamlit run app.py`

- **Step 5: Access the Application**
  After running the command, Streamlit will start a local server and automatically open the application in your default web browser.
  If it does not open automatically, open your browser and go to:
  `http://localhost:8501`

- **Step 6: Use the Application**
- • Enter a recipe topic
  • Enter the desired word count
  • Click on the "Generate Recipe" button
  • The AI will generate and display the recipe

7. API DOCUMENTATION

This project uses the Google Gemini API to generate AI-powered recipe content. The API processes user input and returns structured recipe text in real time.

API Name

Google Gemini Generative AI API

Base Provider

Google AI Studio

Website: https://aistudio.google.com/

Authentication

The API uses an API key for authentication. The API key is generated from Google AI Studio and is required to access the Gemini models.

Example:

client = genai.Client(api_key="YOUR_API_KEY")

The API key ensures secure access and identifies the application making the request.

Model Used

Model Name:

gemini-2.0-flash

This model is optimized for fast and efficient text generation.

API Request

The application sends a prompt to the Gemini API using the following method:

```
response = client.models.generate_content(
    model="gemini-2.0-flash",
    contents=prompt
)
```

Request Parameters

Parameter	Type	Description
model	string	Name of the Gemini model
contents	string	Input prompt containing recipe topic and instructions

API Response

The API returns generated text content.

Example:

response.text

Response includes:

• Recipe introduction
• Ingredients list
• Cooking instructions
• Tips
• Conclusion

Error Handling

The application handles API errors using try-except blocks:

try:

    recipe = generate_recipe(topic, words)

except Exception as e:

    st.error(e)

API Workflow

1. User enters recipe topic
2. Application creates prompt
3. Prompt is sent to Gemini API
4. Gemini generates recipe
5. Response is returned
6. Recipe is displayed to user

8. AUTHENTICATION

The AI Recipe Generator application uses API key authentication to securely access the Google Gemini Generative AI service. The API key is required to authorize requests and ensure that only valid users can access the AI model.

The API key is generated from Google AI Studio and is used in the application to establish a secure connection with the Gemini API. Each request sent to the Gemini model includes this API key for verification.

To authenticate, the API key is configured in the application using the Google GenAI client as shown below:

from google import genai

client = genai.Client(api_key="YOUR_API_KEY")

The API key acts as a unique identifier for the application and allows access to the Gemini AI model for content generation.

For security purposes, the API key should be kept confidential and should not be shared publicly. It is recommended to store the API key in environment variables instead of directly writing it in the source code.

Example using environment variable:

```
import os

from google import genai

client = genai.Client(api_key=os.getenv("GEMINI_API_KEY"))
```

This ensures secure authentication and protects the API key from unauthorized access.

9. USER INTERFACE

The user interface of the AI Recipe Generator is built using Streamlit, which provides a simple, interactive, and user-friendly web interface. The UI allows users to easily interact with the application and generate recipes without requiring any technical knowledge.

The interface consists of a clean layout with input fields, buttons, and output display sections. Users can enter the recipe topic and specify the desired word count using the provided input components. Once the user clicks the "Generate Recipe" button, the application sends the request to the Gemini AI model and displays the generated recipe on the same page.

To enhance user experience, a random programming joke is displayed while the recipe is being generated. The generated recipe is shown in a structured format, making it easy to read and understand.

Key UI Components:

- Title section displaying the application name
- Text input field for entering recipe topic
- Number input field for selecting word count
- Generate Recipe button to trigger AI generation
- Status messages (loading, success, error)
- Output display section for showing generated recipe
- Informational message area for displaying jokes

The Streamlit interface ensures fast interaction, real-time updates, and smooth user experience, making the application efficient and easy to use.

10. TESTING

Testing was performed to ensure that the AI Recipe Generator application works correctly and generates accurate recipe content based on user input. Both functional testing and user interface testing were conducted to verify the performance and reliability of the system.

Functional Testing

Functional testing was done to verify that all core features of the application work as expected.

Test cases included:

• Entering valid recipe topics and generating recipes
• Testing different word counts (100 to 2000 words)
• Verifying correct response from Gemini API
• Checking proper display of generated recipe
• Testing joke generation feature
• Verifying error handling when input is empty
• Testing API key authentication

Expected result:
The application successfully generated complete recipes based on user input.

User Interface Testing

UI testing was performed to ensure that all interface components function properly.

Tested components:

• Text input field
• Number input field
• Generate Recipe button
• Output display area
• Status messages (loading, success, error)

Expected result:
All UI components responded correctly and displayed proper output.

Performance Testing

The application was tested for response time and performance.

Results:

• Recipe generation time: 2–5 seconds (depending on word count)
• No crashes or freezes observed
• Smooth and responsive interface

Error Handling Testing

Tested scenarios:

• Empty input
• Invalid API key
• API quota exceeded

Result:
Application displayed appropriate error messages without crashing.

11. Screenshots or Demo

https://github.com/chakrii18/Flavour-Fusion-AI-Driven-Recipe-Blogging/blob/main/FlavourFusion%20Video%20Demo.mp4

12. Known Issues

- **API Quota Limit**

The Google Gemini API has usage limits in the free tier. If the quota is exceeded, the application may show an error such as "Resource Exhausted" or "Quota Exceeded." This can be resolved by waiting for the quota reset or using a new API key.

- **Internet Dependency**

The application requires an active internet connection to communicate with the Gemini API. Without internet access, the recipe generation feature will not work.

- **Response Time Variation**

The recipe generation time may vary depending on the word count and API response time. Larger word counts may take slightly longer to generate.

- **No Data Storage**

The application does not store generated recipes permanently. Once the application is closed or refreshed, the generated content will be lost unless saved manually.

- **API Key Security**

If the API key is exposed publicly, it may be misused. It is recommended to store the API key securely using environment variables.

- **Limited Customization**

Currently, the application only generates recipes based on topic and word count. Advanced customization features such as cuisine type, difficulty level, or dietary preferences are not included.

13. Future Enhancements

The AI Recipe Generator application can be further improved by adding new features and enhancing existing functionalities to provide a better user experience and increased functionality.

- **Recipe History Storage**

Future versions can include a database to store previously generated recipes. This will allow users to view, manage, and reuse their generated recipes without losing them after closing the application.

- **Download and Export Options**

A feature can be added to allow users to download recipes in different formats such as PDF, TXT, or DOCX. This will help users save and share recipes easily.

- **Advanced Input Options**

Additional input options such as cuisine type (Indian, Chinese, Italian), difficulty level (easy, medium, hard), cooking time, and dietary preferences (vegetarian, vegan, gluten-free) can be added to generate more customized recipes.

- **User Authentication System**

User login and registration features can be implemented to provide personalized experience and allow users to save their recipe history securely.

- **Improved User Interface**

The user interface can be enhanced with better styling, themes (dark/light mode), animations, and improved layout for a more professional look.

- **Multi-language Support**

The application can be extended to support multiple languages, allowing users to generate recipes in their preferred language.

- **Deployment to Cloud**

The application can be deployed on cloud platforms such as Streamlit Cloud, AWS, or Heroku, making it accessible from anywhere without local setup.