**FINAL REPORT**

**FLAVOUR-FUSION**

**TEAM ID:**LTVIP2026TMIDS65419

**1. INTRODUCTION**

**1.1 Project Overview**

Flavor Fusion is a web-based Generative AI application designed to automate the creation of structured recipe blog posts. The system leverages Google’s Gemini (gemini-flash-latest) model to generate complete blog-style recipes based on user input.

The application allows users to:

* Enter a recipe topic
* Select desired word count (100–2000 words)
* Generate a structured recipe blog
* Download the output as a Markdown (.md) file

The solution is implemented using Streamlit for the frontend interface and Python for application logic, with Gemini API integration for AI-driven content generation.

**1.2 Purpose**

The purpose of this project is to:

* Reduce the time required to write structured recipe blogs
* Provide AI-assisted content generation for food enthusiasts and bloggers
* Demonstrate practical integration of Generative AI into a real-world web application
* Deliver a lightweight, scalable, and deployable AI-based system

**2. IDEATION PHASE**

**2.1 Problem Statement**

Food bloggers and home cooks spend significant time writing structured recipe blog posts. Drafting engaging introductions, organizing ingredients, and formatting instructions requires writing skills and effort.

Traditional recipe websites provide static content and do not assist users in generating customized, blog-ready content instantly.

Flavor Fusion addresses this gap by providing AI-powered automated recipe blog generation.



**2.2 Empathy Map Canvas**

**Target User**

Aspiring food blogger or home cook with limited writing time.

**Says**

* “Writing takes more time than cooking.”
* “I want professional-looking blogs.”

**Thinks**

* “My content isn’t engaging enough.”
* “I need to post consistently.”

**Does**

* Searches online for blog format examples
* Spends hours formatting content

**Feels**

* Frustrated by writing effort
* Insecure about content quality

**Gains Expected**

* Faster content creation
* Structured blog output
* Increased publishing confidence

**2.3 Brainstorming**

During ideation, the team evaluated multiple AI-based content generation ideas including:

* Travel blog generator
* Fitness plan generator
* Academic summary tool
* AI recipe generator

The AI recipe blog generator was selected due to:

* Clear user problem
* Feasible implementation
* Strong demonstration of Generative AI capability
* Practical real-world relevance

Features were prioritized based on impact and development effort.

Graphical user interface, application

Description automatically generated

**3. REQUIREMENT ANALYSIS**

**3.1 Customer Journey Map**

1. User opens web application
2. Enters recipe topic
3. Selects word count
4. Clicks “Generate Recipe”
5. Views AI-generated blog
6. Downloads Markdown file

Pain Point Addressed:  
Manual content drafting replaced with instant AI generation.

**3.2 Solution Requirement**

**Functional Requirements**

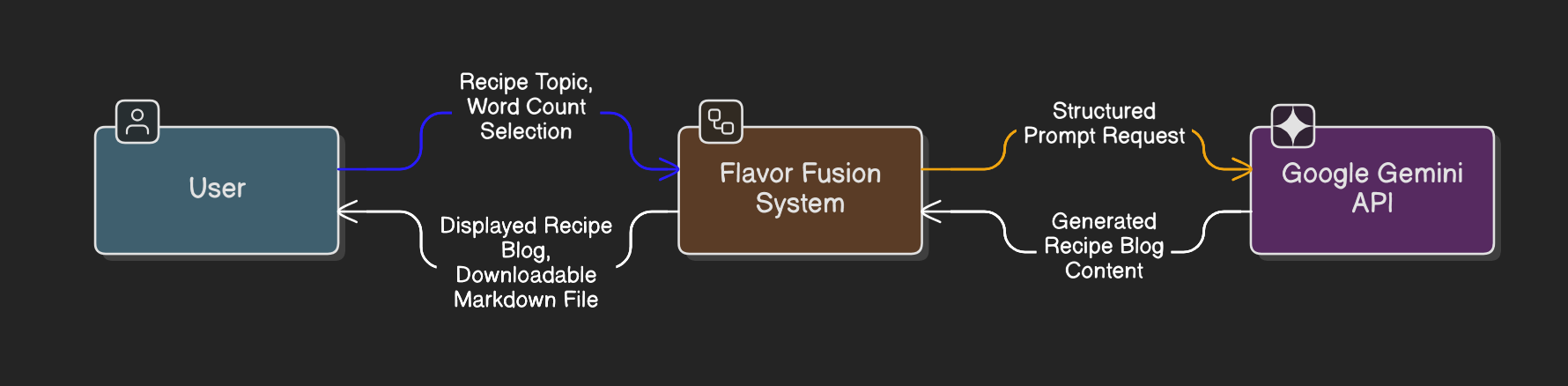
* Accept recipe topic input
* Accept word count selection
* Generate structured recipe blog via Gemini API
* Display generated content
* Provide Markdown download
* Show loading feedback
* Handle API errors gracefully

**Non-Functional Requirements**

* Usability: Simple and intuitive UI
* Performance: Generation within acceptable API latency
* Reliability: No system crash during API failure
* Security: API key stored securely
* Scalability: Cloud deployable architecture

**3.3 Data Flow Diagram (Description)**

User → Streamlit UI → Application Logic → Gemini API  
Gemini API → Application Logic → Content Formatter → Markdown Generator → User



DFD Level 0 (Industry Standard)

DFD Level 1 (Industry Standard)



The system follows a stateless architecture with no persistent database.

**3.4 Technology Stack**

Frontend: Streamlit  
Backend: Python  
AI Model: Google Gemini (gemini-flash-latest)  
File Format: Markdown (.md)  
Deployment: Local / Streamlit Cloud

**4. PROJECT DESIGN**

**4.1 Problem–Solution Fit**

Problem:  
Time-consuming manual recipe blog writing.

Solution:  
AI-powered automated blog generation with structured formatting.

Fit Justification:  
The solution directly reduces writing effort while maintaining professional output quality.

Calendar

Description automatically generated

**4.2 Proposed Solution**

Flavor Fusion is a lightweight AI-powered web application that:

* Dynamically generates structured recipe blogs
* Provides customizable word length
* Offers export-ready Markdown files
* Enhances user engagement via interactive UI

The system bridges business need (content automation) with AI technology.

**4.3 Solution Architecture**

The architecture consists of four layers:

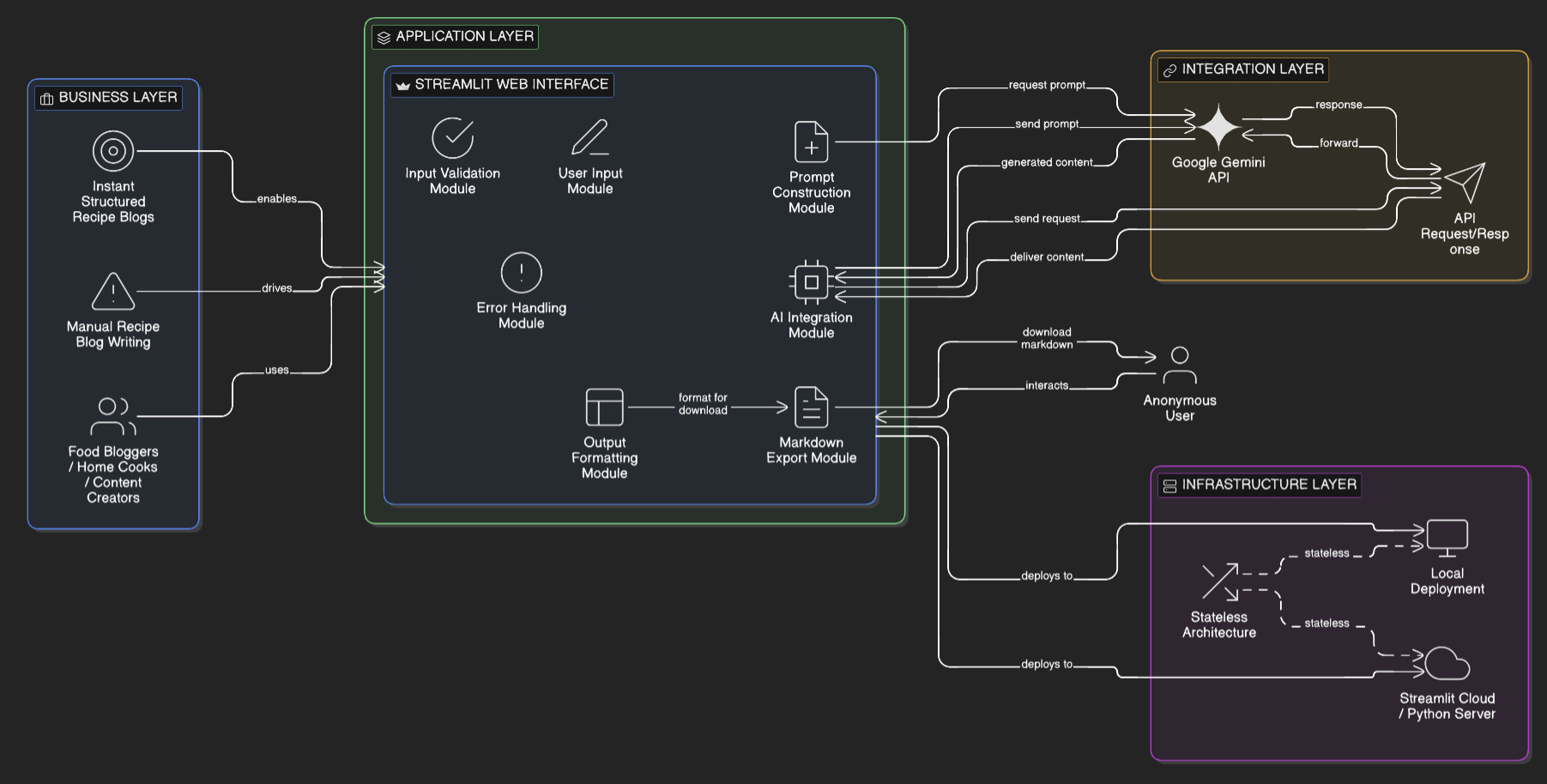
Business Layer:  
User need for faster blog creation.

Application Layer:  
Streamlit UI + Python logic modules.

Integration Layer:  
Google Gemini API.

Infrastructure Layer:  
Local or cloud deployment environment.

The application is stateless and scalable through horizontal cloud deployment.



**5. PROJECT PLANNING & SCHEDULING**

**5.1 Project Planning**

Development was completed in two sprints:

Sprint 1:

* UI development
* Gemini API integration
* Prompt engineering

Sprint 2:

* Markdown export
* Loading indicator and joke display
* Error handling
* Testing and deployment

Average team velocity: 19 Story Points per sprint.

**6. FUNCTIONAL AND PERFORMANCE TESTING**

**6.1 Performance Testing**

* AI generation time tested under multiple word count conditions.
* Short output (~200 words): 3–5 seconds.
* Long output (~1500 words): 5–8 seconds depending on API latency.
* No application crash during API timeout simulation.

System confirmed stable for UAT.

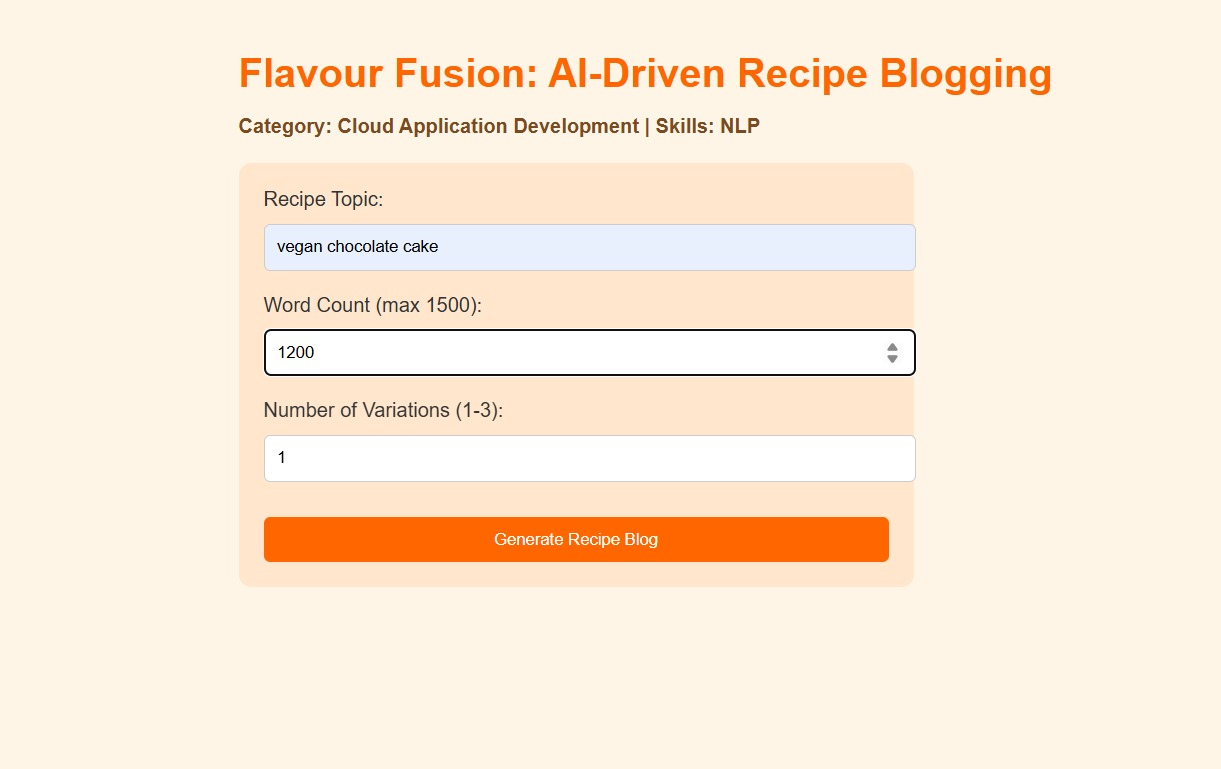
**7. RESULTS**

**7.1 Output Screenshots**

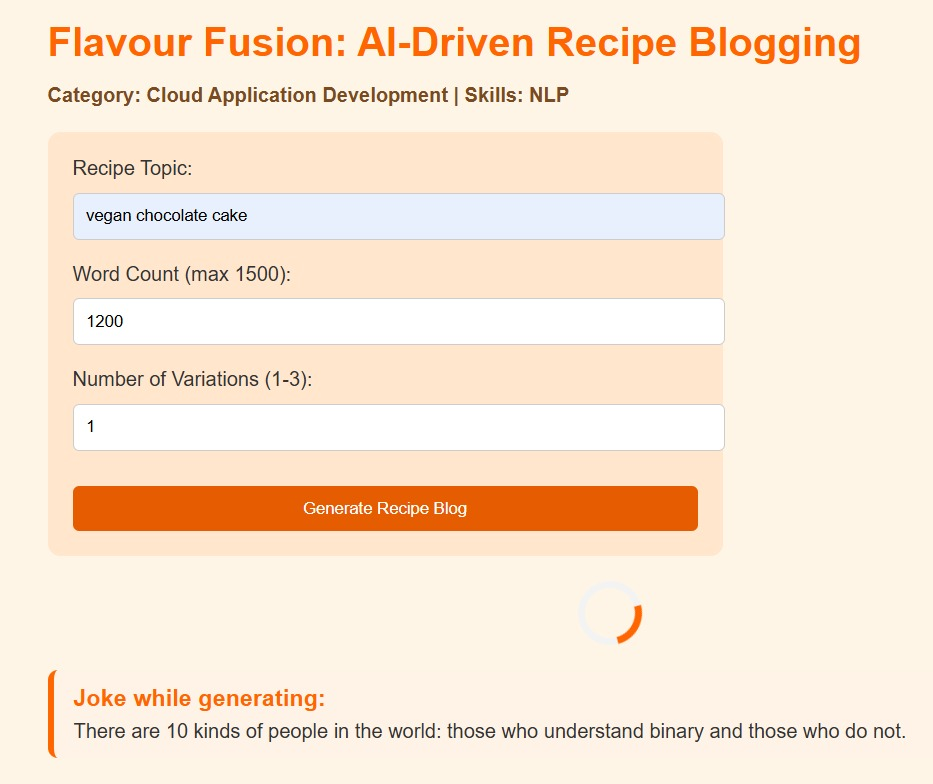
* Main interface screen

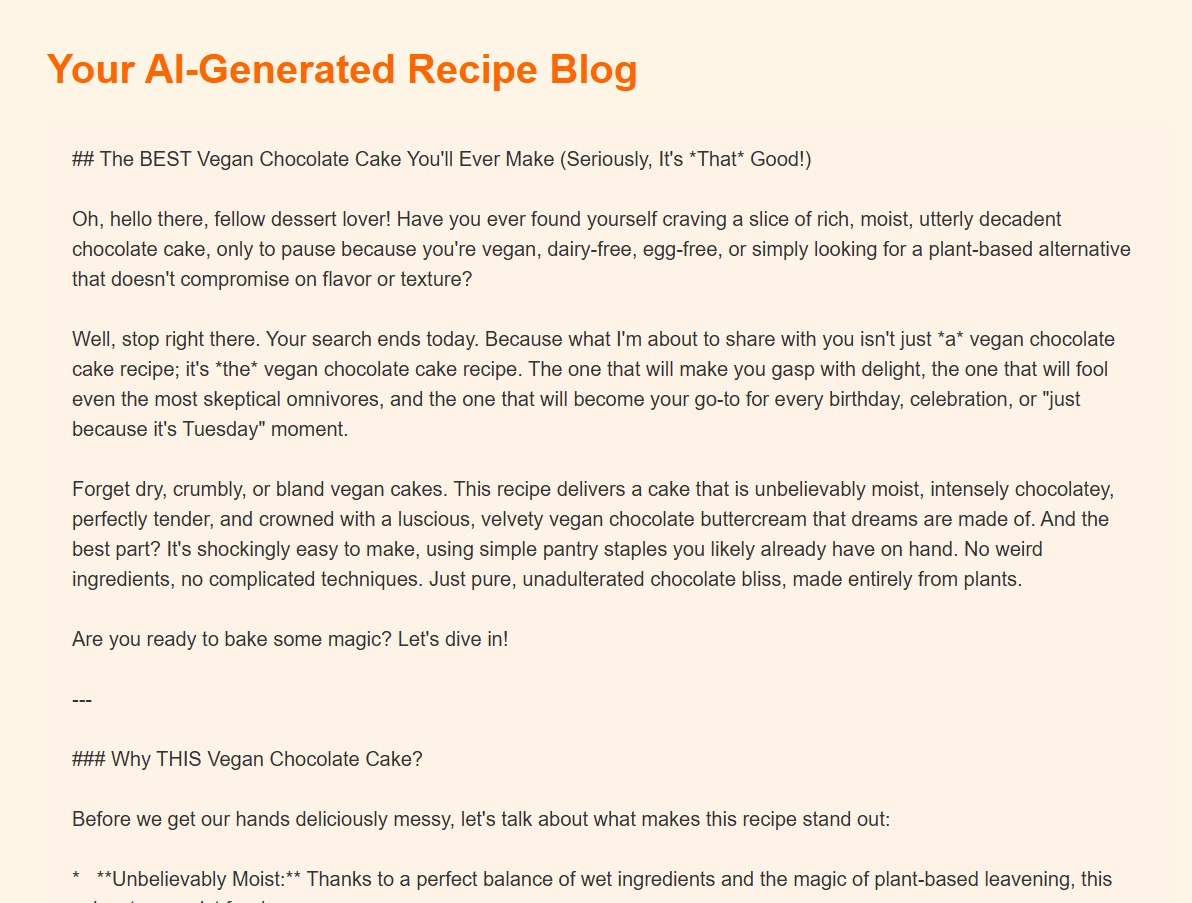


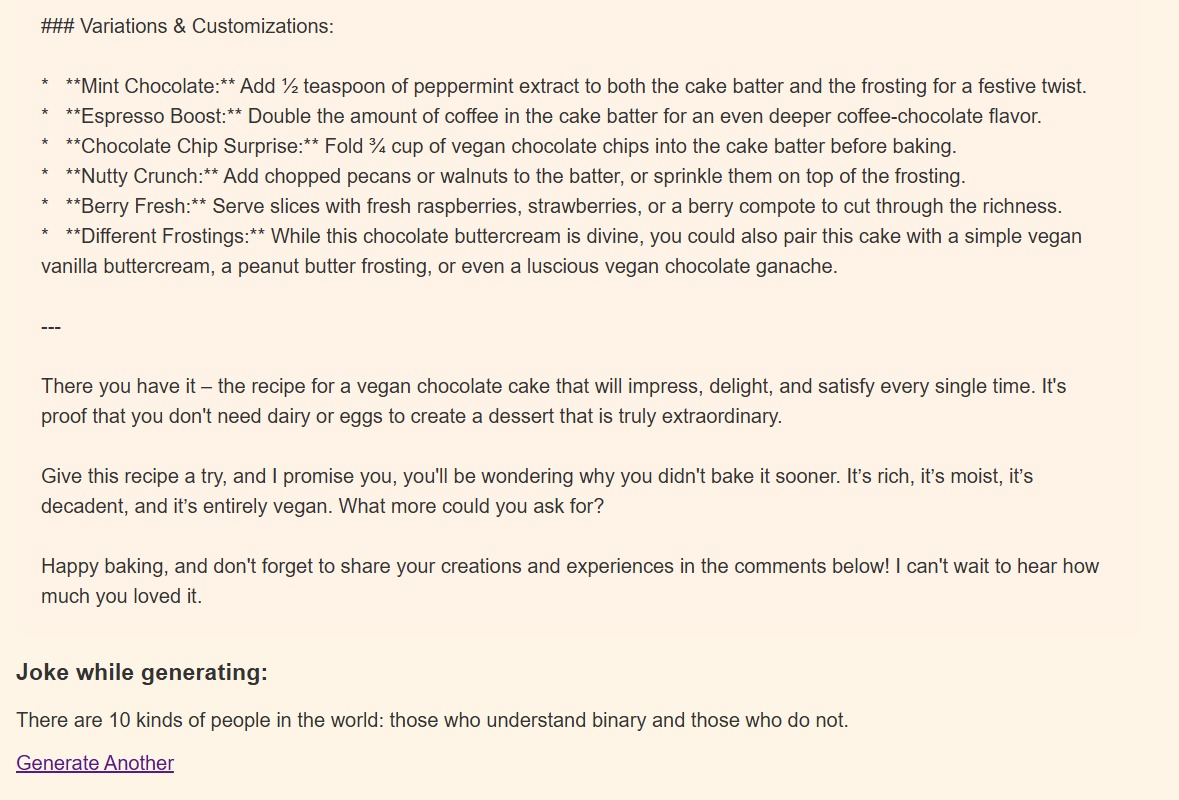
* Input section



* Waiting Time



* Generated recipe output
* Generate another recipe confirmation



Generated outputs demonstrated:

* Structured blog format
* Correct word count range
* Professional presentation

**8. ADVANTAGES & DISADVANTAGES**

**Advantages**

* Significant time savings
* Structured professional output
* Lightweight architecture
* Easy deployment
* Beginner-friendly UI

**Disadvantages**

* Dependent on external AI API
* Requires internet connection
* API latency may vary
* No persistent user storage

**9. CONCLUSION**

Flavor Fusion successfully demonstrates practical integration of Generative AI into a real-world web application.

The system reduces manual effort in content creation while maintaining structured output quality. It validates the feasibility of AI-driven automation for creative blogging tasks.

The project achieves strong problem–solution alignment and demonstrates scalable AI application architecture.

**10. FUTURE SCOPE**

* Multi-language recipe generation
* SEO keyword optimization
* AI image generation for recipes
* Nutrition analysis integration
* User account and recipe history storage
* Cloud auto-scaling deployment

**11. APPENDIX**

**Source Code (app.py):**

import os

from flask import Flask, render\_template, request

from dotenv import load\_dotenv

try:

    # Preferred SDK (actively maintained)

    from google import genai as google\_genai

    \_GENAI\_MODE = "new"

except ImportError:

    # Backward-compatible fallback for older environments

    import google.generativeai as google\_genai

    \_GENAI\_MODE = "legacy"

# Load environment variables

load\_dotenv()

app = Flask(\_\_name\_\_)

# Get API key

API\_KEY = os.getenv("GOOGLE\_API\_KEY", "").strip()

print(f"API key loaded: {'yes' if API\_KEY else 'no'}")

# Configure legacy Gemini SDK

if API\_KEY and \_GENAI\_MODE == "legacy":

    google\_genai.configure(api\_key=API\_KEY)

# Programmer jokes

PROGRAMMER\_JOKES = [

    "Why do programmers prefer dark mode? Because light attracts bugs!",

    "A SQL query walks into a bar, walks up to two tables, and asks: Can I join you?",

    "There are 10 kinds of people in the world: those who understand binary and those who do not.",

    "Debugging is like being the detective in a crime movie where you are also the murderer.",

]

def fallback\_recipe(topic: str, word\_count: int) -> str:

    return (

        f"{topic} - Simple Home Recipe Guide\n\n"

        "Ingredients:\n"

        "- 2 cups main ingredient of choice\n"

        "- 1 tbsp oil or butter\n"

        "- 1 tsp salt (adjust to taste)\n"

        "- 1 tsp spices/herbs\n"

        "- Optional garnish\n\n"

        "Steps:\n"

        "1. Prep all ingredients and preheat cookware.\n"

        "2. Cook base ingredients on medium heat until aromatic.\n"

        "3. Add seasonings and simmer until textures are just right.\n"

        "4. Plate, garnish, and serve warm.\n\n"

        "Cooking Tips:\n"

        "- Taste and adjust seasoning near the end.\n"

        "- Keep heat moderate to avoid burning.\n"

        "- Pair with a fresh side salad or bread.\n\n"

        f"Requested target length: ~{word\_count} words."

    )

def call\_gemini(prompt: str) -> str:

    if not API\_KEY:

        raise RuntimeError("GOOGLE\_API\_KEY is not configured.")

    if \_GENAI\_MODE == "new":

        client = google\_genai.Client(api\_key=API\_KEY)

        model\_candidates = ["gemini-2.5-flash", "gemini-2.0-flash", "gemini-1.5-flash"]

        last\_error = None

        for model\_name in model\_candidates:

            try:

                response = client.models.generate\_content(

                    model=model\_name,

                    contents=prompt,

                )

                text = (response.text or "").strip()

                if text:

                    return text

            except Exception as exc:

                last\_error = exc

        raise RuntimeError(f"Gemini new SDK failed for all model candidates: {last\_error}")

    model\_candidates = ["gemini-2.0-flash", "gemini-1.5-flash", "gemini-1.5-flash-latest"]

    last\_error = None

    for model\_name in model\_candidates:

        try:

            model = google\_genai.GenerativeModel(model\_name)

            response = model.generate\_content(prompt)

            text = (response.text or "").strip()

            if text:

                return text

        except Exception as exc:

            last\_error = exc

    raise RuntimeError(f"Gemini legacy SDK failed for all model candidates: {last\_error}")

def generate\_recipe(topic: str, word\_count: int) -> str:

    print(f"Generating recipe for topic: {topic}")

    prompt = (

        f"Write a detailed recipe blog about '{topic}' "

        f"with approximately {word\_count} words. "

        "Include ingredients, numbered steps, prep/cook times, "

        "and practical cooking tips."

    )

    try:

        ai\_text = call\_gemini(prompt)

        if ai\_text:

            return ai\_text

        raise RuntimeError("Gemini response did not include text.")

    except Exception as exc:

        print(f"Gemini generation failed: {exc}")

        return fallback\_recipe(topic, word\_count)

@app.route("/")

def home():

    return render\_template("index.html", jokes=PROGRAMMER\_JOKES)

@app.route("/generate", methods=["POST"])

def generate():

    topic = request.form.get("topic", "Quick and Easy Cooking")

    word\_count\_raw = request.form.get("word\_count", 300)

    variations\_raw = request.form.get("variations", 1)

    selected\_joke = request.form.get("joke") or PROGRAMMER\_JOKES[0]

    try:

        word\_count = int(word\_count\_raw)

    except (TypeError, ValueError):

        word\_count = 300

    word\_count = max(50, min(1500, word\_count))

    try:

        variations = int(variations\_raw)

    except (TypeError, ValueError):

        variations = 1

    variations = max(1, min(3, variations))

    blogs = [generate\_recipe(topic, word\_count) for \_ in range(variations)]

    return render\_template("result.html", blog=blogs, joke=selected\_joke)

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

**GitHub Repository**

https://github.com/Nehapriya30/Flavour-Fusion-AI-Driven-Recipe-Blogging

**Project Demo Link**

https://drive.google.com/drive/folders/1\_DviBK6TFRJokXzs2ZzlddlmzVLZCMdh?usp=sharing