Flavour Fusion AI-Driven Recipe Blogging

1. Introduction

• Project Title: Flavour Fusion AI-Driven Recipe Blogging

• Team ID: LTVIP2026TMIDS76104

• Team Members & Responsibities:

Name	**Role**	**Responsibilities**
Peetha Madhurima	Team Lead	Project planning, timeline management, task delegation, and final review of deliverables.
Nithin Tirukkovalluri	Frontend Developer	Implementation of the Streamlit user interface, layout design, and ensuring responsiveness.
Abhishek Kumar Sah	Backend Developer	Integration of Google Gemini API, prompt engineering, and core logic implementation in Python.
Dommeti Pujitha	Data/Requirement Analyst	Requirement gathering, defining user stories, and analyzing problem statements (Ideation Phase).
Indra Narayan Yadav	QA & Documentation	Manual testing of the application, identifying bugs, and preparing project documentation.

2. Project Overview

• Purpose:

Flavour Fusion is an AI-powered application designed to assist food bloggers, home cooks, and culinary enthusiasts in generating detailed and engaging recipe blogs. The system leverages advanced Generative AI to create structured recipe content instantly, saving time and effort in content creation.

• Problem Statement:

Creating high-quality recipe blogs requires significant time and creativity. Writers must structure content effectively, ensuring it includes ingredients, step-by-step instructions, and engaging introductions. Many creators struggle with writer's block or the repetitive nature of formatting recipes.

Objectives:

- To automate the generation of recipe blog content.
- To provide a user-friendly interface for generating recipes based on simple topics.
- To ensure generated content is structured, professional, and ready for publication.

Scope:
The current scope of the project includes text-based recipe generation. It covers:

- Generating titles, introductions, ingredient lists, and instructions.
- Customizing content length via word count.
- Providing cooking tips and variations.

Key Features:

- **Instant Recipe Generation:** Generates comprehensive recipes in seconds.
- **Customizable Length:** Users can specify the word count (100-2000 words).
- **Engaging UI:** Simple and clean interface with interactive elements.
- **Entertaining Wait Time:** Displays programmer jokes while the AI generates content.

Target Users:

- Food Bloggers
- Content Creators
- Home Cooks
- Culinary Students

Technologies Used:

- **Frontend & Backend:** Streamlit (Python framework for data apps)
- **AI Model:** Google Gemini (gemini-2.5-flash)
- **Language:** Python 3.x
- **Key Libraries:** google-generativeai, python-dotenv

3. Architecture

Architecture Type:
The application follows a **Monolithic Script Architecture** typical of Streamlit applications. It acts as a client-server model where the browser acts as the client and the Python script runs on the server (local or cloud).

System Components:

1. **User Interface (Streamlit):** Handles user inputs (topic, word count) and displays the generated content.
2. **Application Logic (**

   app.py**):**

   - Manages API configurations.

    - Constructs prompts for the AI model.
    - Handles data flow between the UI and the AI service.
3. **External Service (Google Gemini API):** Receives the prompt and generates the recipe content.

Data Flow:

1. **Input:** User enters "Recipe Topic" and "Word Count" in the browser.
2. **Request:** The application sends a structured prompt to the Google Gemini API.
3. **Processing:** Gemini processes the prompt and generates text.
4. **Response:** The API returns the generated text to the application.
5. **Display:** The application renders the text as Markdown in the browser.

4. Setup Instructions

Prerequisites:

- Python 3.8 or higher installed.
- A Google Cloud Project with the **Gemini API** enabled.
- An API Key for Google Gemini.

Installation Steps:

1. **Clone the Repository**

```
git clone <repository-url>
cd "Flavour Fusion AI-Driven Recipe Blogging"
```

2. **Install Dependencies:**

```
pip install -r requirements.txt
```

3. **Configure Environment Variables:**
    1. Create a file named env in the root directory.
    2. Add your Gemini API key:

```
GEMINI_API_KEY=your_actual_api_key_here
```

Build and Deployment

Since this is a Streamlit app, there is no traditional "build" step for the frontend. It is run directly by the Python interpreter. For deployment (e.g., on Streamlit Cloud or Heroku), ensure
requirements.txt is present and the GEMINI_API_KEY is set in the platform's secrets management.

5. Folder Structure

```
Flavour Fusion AI-Driven Recipe Blogging/
│
```

```
├── app.py                # Main application entry point and logic
├── requirements.txt      # Python dependencies
├── .env                  # Environment variables (API Key) - Not in source control
├── .gitignore            # Files to ignore in Git
│
├── Ideation Phase/       # Documentation regarding initial ideas
├── Requirement Analysis/   # User stories and requirements documents
├── Project Planning Phase/ # Project timelines and plans
└── Performance Testing/    # Logs or reports on performance
```

Key Module:

app.py

```
from dotenv import load_dotenv
import os
import streamlit as st
import google.generativeai as genai
import random

# ==================================
# Configure Gemini API Key
# ==================================
load_dotenv()
genai.configure(api_key=os.getenv("GEMINI_API_KEY"))


# ==================================
# Model Generation Configuration
# ==================================
generation_config = {
    "temperature": 0.7,
    "top_p": 0.9,
    "top_k": 40,
    "max_output_tokens": 2048,
}


# ==================================
# Initialize Gemini Model
# ==================================
model = genai.GenerativeModel(
    model_name="gemini-2.5-flash",
    generation_config=generation_config
)
```

```
# ==================================
# Joke Generation Function
# ==================================
def get_joke():
    jokes = [
        "Why do programmers prefer dark mode? Because light attracts bugs!",
        "Why did the programmer quit his job? Because he didn't get arrays!",
        "How many programmers does it take to change a light bulb? None. It's a
hardware problem!",
        "Why do Java developers wear glasses? Because they don't see sharp!",
        "Why was the computer cold? Because it forgot to close its Windows!",
        "There are only 10 types of people in the world: those who understand binary and
those who don't.",
        "Why did the programmer go broke? Because he used up all his cache!",
        "Debugging: Being the detective in a crime movie where you are also the
murderer."
    ]
    return random.choice(jokes)


# ==================================
# Recipe Generation Function
# ==================================
def recipe_generation(user_input, word_count):
    """
    Generates a recipe blog using Gemini AI
    """

    # Show loading message
    st.write("### ⏳ Generating your recipe blog...")
    st.write(f"💡 Here's a programmer joke while you wait:\n\n**{get_joke()}**")

    # Prompt for Gemini
    prompt = f"""
    Write a detailed and engaging recipe blog about: {user_input}.

    Requirements:
    - Word count: approximately {word_count} words
    - Include:
        • Introduction
        • Ingredients list
        • Step-by-step instructions
        • Cooking tips
        • Variations (optional)
        • Conclusion

    Make it friendly, engaging, and suitable for a food blog.
    """

    try:
        response = model.generate_content(prompt)
        st.success("✅ Your recipe blog is ready!")
        return response.text

    except Exception as e:
        st.error(f"❌ Error generating blog: {e}")
        return None


# ================================
# Streamlit User Interface
# ================================

# Page title
st.title("🍽️ Flavour Fusion: AI-Driven Recipe Blogging")

# Description
st.write("Generate creative and detailed recipe blogs using Gemini AI.")


# User input
topic = st.text_input("Enter Recipe Topic:", placeholder="Example: Vegan Chocolate
Cake")

word_count = st.number_input(
    "Enter Word Count:",
    min_value=100,
    max_value=2000,
    value=500,
    step=100
)


# Generate button
if st.button("Generate Recipe Blog"):

    if topic.strip() == "":
        st.warning("⚠️ Please enter a recipe topic.")
    else:
        blog = recipe_generation(topic, word_count)

        if blog:
            st.write("## 📖 Generated Recipe Blog")
            st.write(blog)


# Footer
st.write("---")
st.write("Made with ♡ using Streamlit and Gemini API")
```

requirements.txt

```
streamlit
google-generativeai
python-dotenv
```

6. Running the Application

Start the Application

Run the following command in the terminal from the project root:

```
streamlit run app.py
```

Accessing the App

- **Local URL:** http://localhost:8501
- **Network URL:** http://10.70.6.66:8501

The application will automatically open in your default web browser.

7. API Documentation

Note: This application primarily consumes an external API (Google Gemini) rather than exposing its own REST endpoints. However, the internal functions act as the logical interface.

Internal Functions

recipe_generation(user_input, word_count)

- **Description:** Generates a recipe blog post using the Gemini model.
- **Parameters:**
  - user_input (str): The topic of the recipe (e.g., "Pasta").
  - word_count (int): Approximate length of the blog post.
- **Returns:** str (The generated blog content in Markdown format) or None on error.

get_joke()

- **Description:** Returns a random programmer joke to entertain users during loading states.
- **Returns:** str

8. Authentication

User Authentication

- **Current Status:** No user authentication (Signcase/Login) is currently implemented. The application is open for public use when running.

System Authentication

- **API Security:** The application uses API Key authentication to communicate with Google Gemini.
- **Security Best Practice:** The API key is stored in a

  .env file and loaded using python-dotenv to prevent hardcoding sensitive credentials in the source code.

9. User Interface

Main Screen
The UI is divided into three main sections:

1. **Header:** Displays the title "🍽 Flavour Fusion: AI-Driven Recipe Blogging" and a brief description.
2. **Input Area:**
   - **Text Input:** "Enter Recipe Topic" (e.g., "Vegan Chocolate Cake").
   - **Number Input:** "Enter Word Count" (Range: 100-2000, Default: 500).
   - **Action Button:** "Generate Recipe Blog".
3. **Output Area:** Displays the generated blog post using Markdown formatting (Headers, Lists, Bold text).

Feedback Mechanisms

- **Loading State:** A spinner and a dynamic programmer joke appear while the request is being processed.
- **Success Message:** "✅ Your recipe blog is ready!" appears upon completion.
- **Error Handling:** Displays an error message if the API call fails or the input is empty.

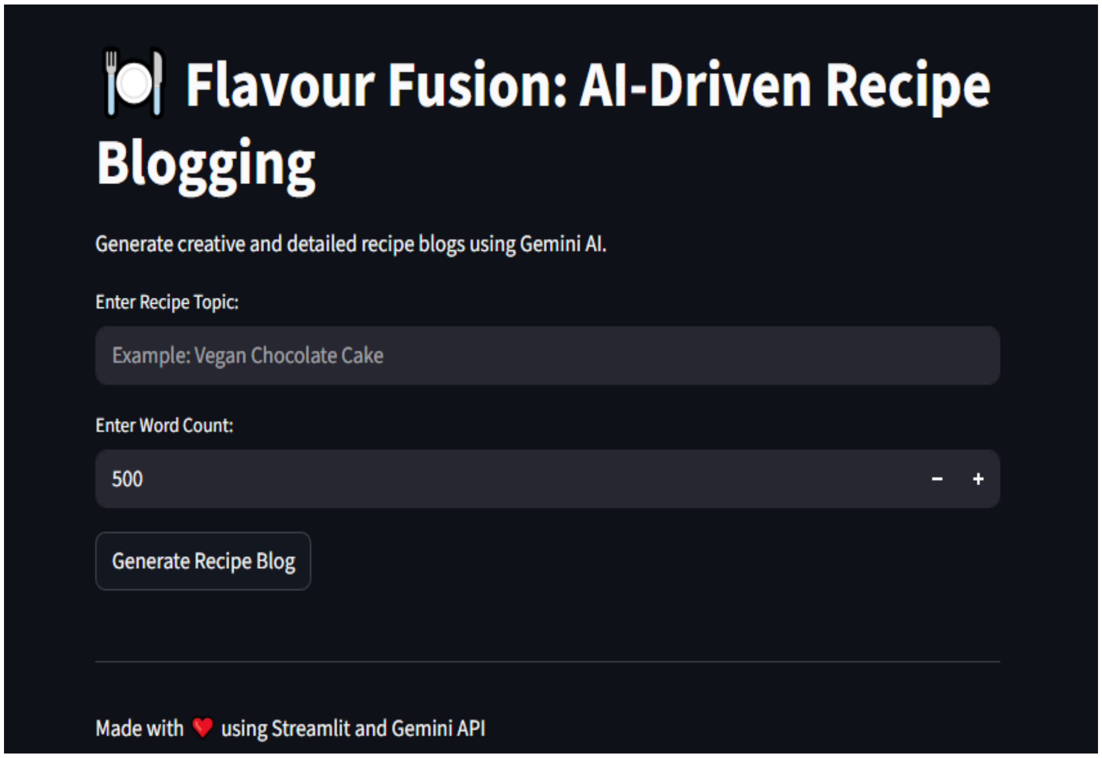
10. Testing

Testing Methods

- **Manual Testing:** The application has been manually tested to ensure:
  - API connectivity with valid and invalid keys.
  - Response to empty inputs (Validation checks).
  - Handling of different word counts.
  - UI responsiveness.

11. Screenshots or Demo

1.Home Page: Show the clean initial state with title and input fields.



🍽️ Flavour Fusion: AI-Driven Recipe Blogging

Generate creative and detailed recipe blogs using Gemini AI.

Enter Recipe Topic:

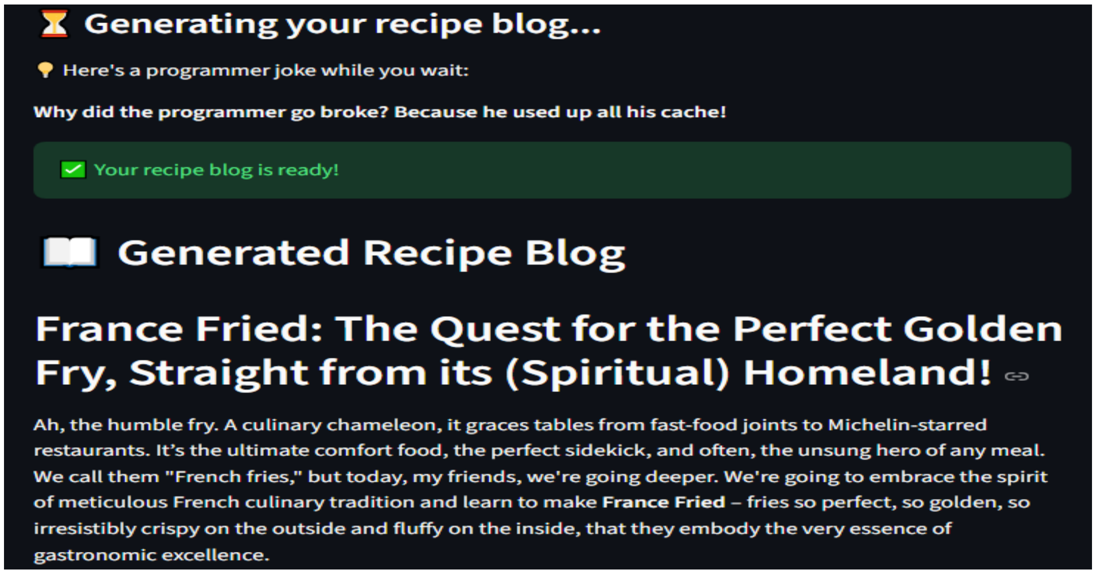
Example: Vegan Chocolate Cake

Enter Word Count:

500

Made with ❤️ using Streamlit and Gemini API

2.Generating State: Capture the interface showing the "Running..." spinner and the random joke.



⏳ Generating your recipe blog...

💡 Here's a programmer joke while you wait:

Why did the programmer go broke? Because he used up all his cache!

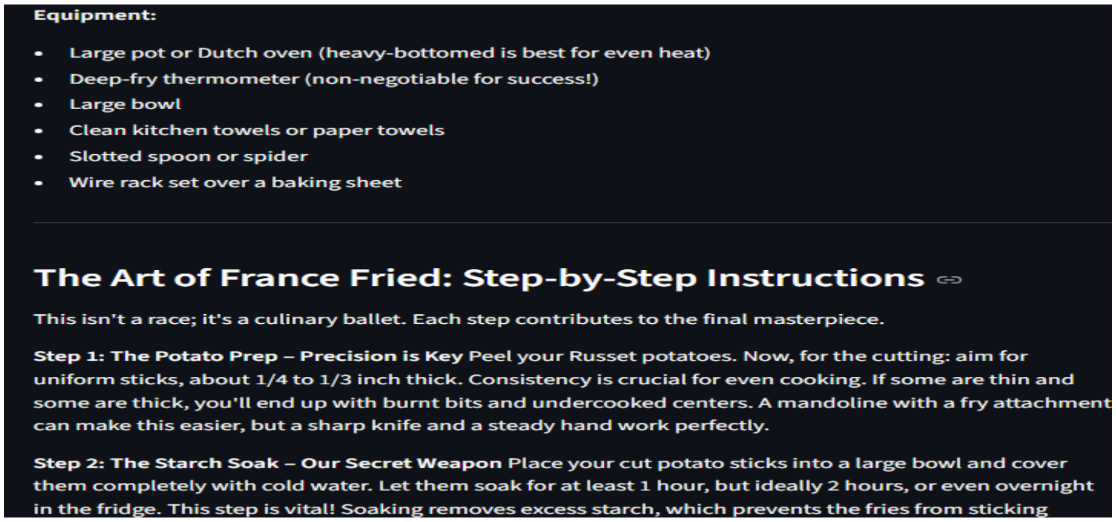
✅ Your recipe blog is ready!

📖 Generated Recipe Blog

France Fried: The Quest for the Perfect Golden Fry, Straight from its (Spiritual) Homeland!

Ah, the humble fry. A culinary chameleon, it graces tables from fast-food joints to Michelin-starred restaurants. It's the ultimate comfort food, the perfect sidekick, and often, the unsung hero of any meal. We call them "French fries," but today, my friends, we're going deeper. We're going to embrace the spirit of meticulous French culinary tradition and learn to make **France Fried** – fries so perfect, so golden, so irresistibly crispy on the outside and fluffy on the inside, that they embody the very essence of gastronomic excellence.

3.Result View: Show a fully generated recipe blog with clear formatting.



Equipment:

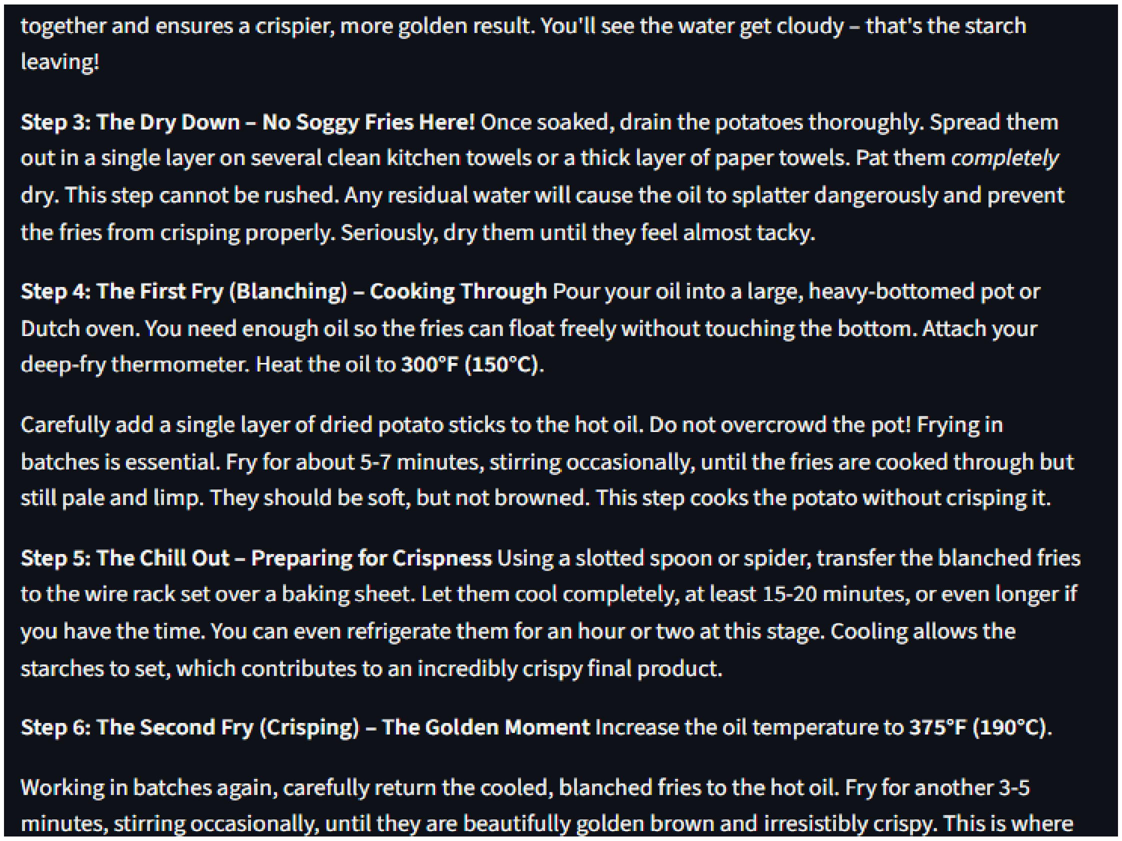
- Large pot or Dutch oven (heavy-bottomed is best for even heat)
- Deep-fry thermometer (non-negotiable for success!)
- Large bowl
- Clean kitchen towels or paper towels
- Slotted spoon or spider
- Wire rack set over a baking sheet

The Art of France Fried: Step-by-Step Instructions

This isn't a race; it's a culinary ballet. Each step contributes to the final masterpiece.

Step 1: The Potato Prep – Precision is Key Peel your Russet potatoes. Now, for the cutting: aim for uniform sticks, about 1/4 to 1/3 inch thick. Consistency is crucial for even cooking. If some are thin and some are thick, you'll end up with burnt bits and undercooked centers. A mandoline with a fry attachment can make this easier, but a sharp knife and a steady hand work perfectly.

Step 2: The Starch Soak – Our Secret Weapon Place your cut potato sticks into a large bowl and cover them completely with cold water. Let them soak for at least 1 hour, but ideally 2 hours, or even overnight in the fridge. This step is vital! Soaking removes excess starch, which prevents the fries from sticking



together and ensures a crispier, more golden result. You'll see the water get cloudy – that's the starch leaving!

Step 3: The Dry Down – No Soggy Fries Here! Once soaked, drain the potatoes thoroughly. Spread them out in a single layer on several clean kitchen towels or a thick layer of paper towels. Pat them *completely* dry. This step cannot be rushed. Any residual water will cause the oil to splatter dangerously and prevent the fries from crisping properly. Seriously, dry them until they feel almost tacky.

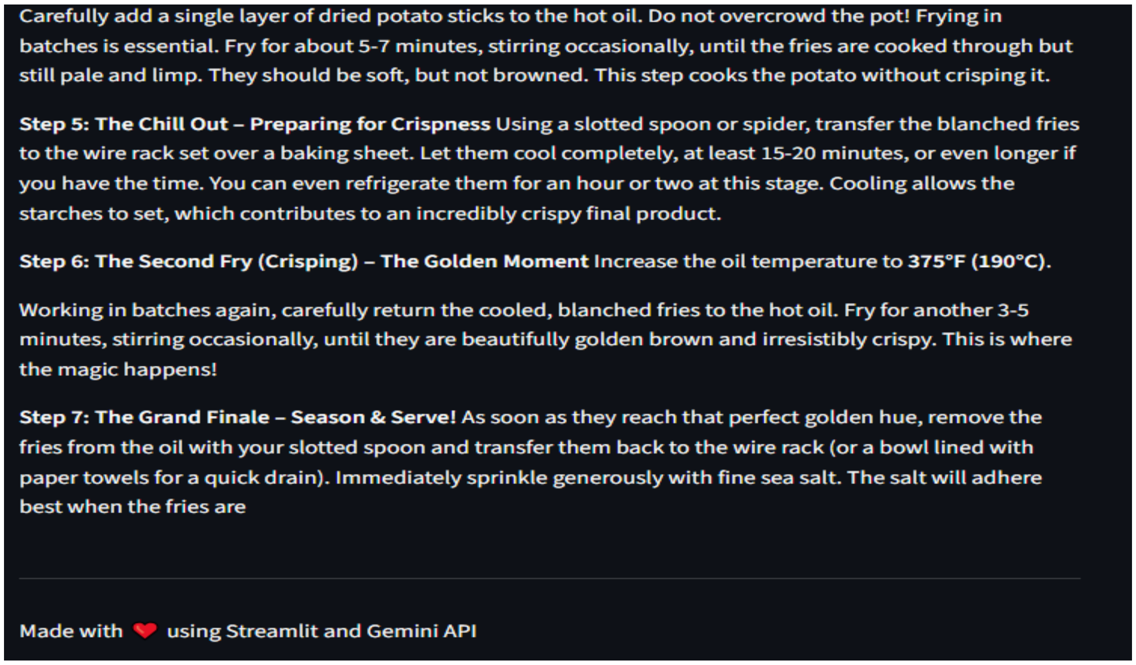
Step 4: The First Fry (Blanching) – Cooking Through Pour your oil into a large, heavy-bottomed pot or Dutch oven. You need enough oil so the fries can float freely without touching the bottom. Attach your deep-fry thermometer. Heat the oil to **300°F (150°C)**.

Carefully add a single layer of dried potato sticks to the hot oil. Do not overcrowd the pot! Frying in batches is essential. Fry for about 5-7 minutes, stirring occasionally, until the fries are cooked through but still pale and limp. They should be soft, but not browned. This step cooks the potato without crisping it.

Step 5: The Chill Out – Preparing for Crispness Using a slotted spoon or spider, transfer the blanched fries to the wire rack set over a baking sheet. Let them cool completely, at least 15-20 minutes, or even longer if you have the time. You can even refrigerate them for an hour or two at this stage. Cooling allows the starches to set, which contributes to an incredibly crispy final product.

Step 6: The Second Fry (Crisping) – The Golden Moment Increase the oil temperature to **375°F (190°C)**.

Working in batches again, carefully return the cooled, blanched fries to the hot oil. Fry for another 3-5 minutes, stirring occasionally, until they are beautifully golden brown and irresistibly crispy. This is where



Carefully add a single layer of dried potato sticks to the hot oil. Do not overcrowd the pot! Frying in batches is essential. Fry for about 5-7 minutes, stirring occasionally, until the fries are cooked through but still pale and limp. They should be soft, but not browned. This step cooks the potato without crisping it.

Step 5: The Chill Out – Preparing for Crispness Using a slotted spoon or spider, transfer the blanched fries to the wire rack set over a baking sheet. Let them cool completely, at least 15-20 minutes, or even longer if you have the time. You can even refrigerate them for an hour or two at this stage. Cooling allows the starches to set, which contributes to an incredibly crispy final product.

Step 6: The Second Fry (Crisping) – The Golden Moment Increase the oil temperature to 375°F (190°C).

Working in batches again, carefully return the cooled, blanched fries to the hot oil. Fry for another 3-5 minutes, stirring occasionally, until they are beautifully golden brown and irresistibly crispy. This is where the magic happens!

Step 7: The Grand Finale – Season & Serve! As soon as they reach that perfect golden hue, remove the fries from the oil with your slotted spoon and transfer them back to the wire rack (or a bowl lined with paper towels for a quick drain). Immediately sprinkle generously with fine sea salt. The salt will adhere best when the fries are

Made with ❤️ using Streamlit and Gemini API

12. Known Issues

- **API Latency:** Generation time depends on the Google Gemini API response time and may vary.
- **Context Limit:** Extremely specific or obscure recipe requests might yield generic results.
- **Session State:** The app does not currently save generated recipes; refreshing the page clears the data.

13. Future Enhancements

1. **Image Generation:** Integrate DALL-E or Gemini Pro Vision to generate realistic images of the recipes.
2. **Save Functionality:** Allow users to download the blog as a PDF or Markdown file.
3. **User Accounts:** Implement login using Firebase or SQL to verify users and save their history.
4. **History Tab:** A feature to view previously generated recipes within the session.
5. **Multi-language Support:** Add an option to generate recipes in different languages (Spanish, French, Hindi, etc.).