Project Development Phase

Development & Integration

Date	10-02-2026
Team ID	LTVIP2026TMIDS65437
Project Name	Flavour Fusion: -Ai-Driven Recipe Blogging
Maximum Marks	3 Marks

1.Frontend development (Streamlit / Flask / Web UI):

Frontend development for an AI-driven recipe blog focused on "Flavour Fusion" involves creating an intuitive, interactive user interface (UI) that bridges user ingredient inputs with AI-generated culinary creations. The goal is to allow users to input available ingredients, dietary preferences, and desired flavor fusions (e.g., Italian-Indian) and receive custom recipes instantly.

1. Streamlit: Rapid Prototyping & AI App Frontend

Streamlit is suitable for quick app building, requiring no prior frontend experience (HTML/CSS/JS) and is based entirely on Python.

2. Flask: Backend & Custom Web UI

Flask is a Python web framework used when more control, custom routing, or a traditional, highly styled website is needed.

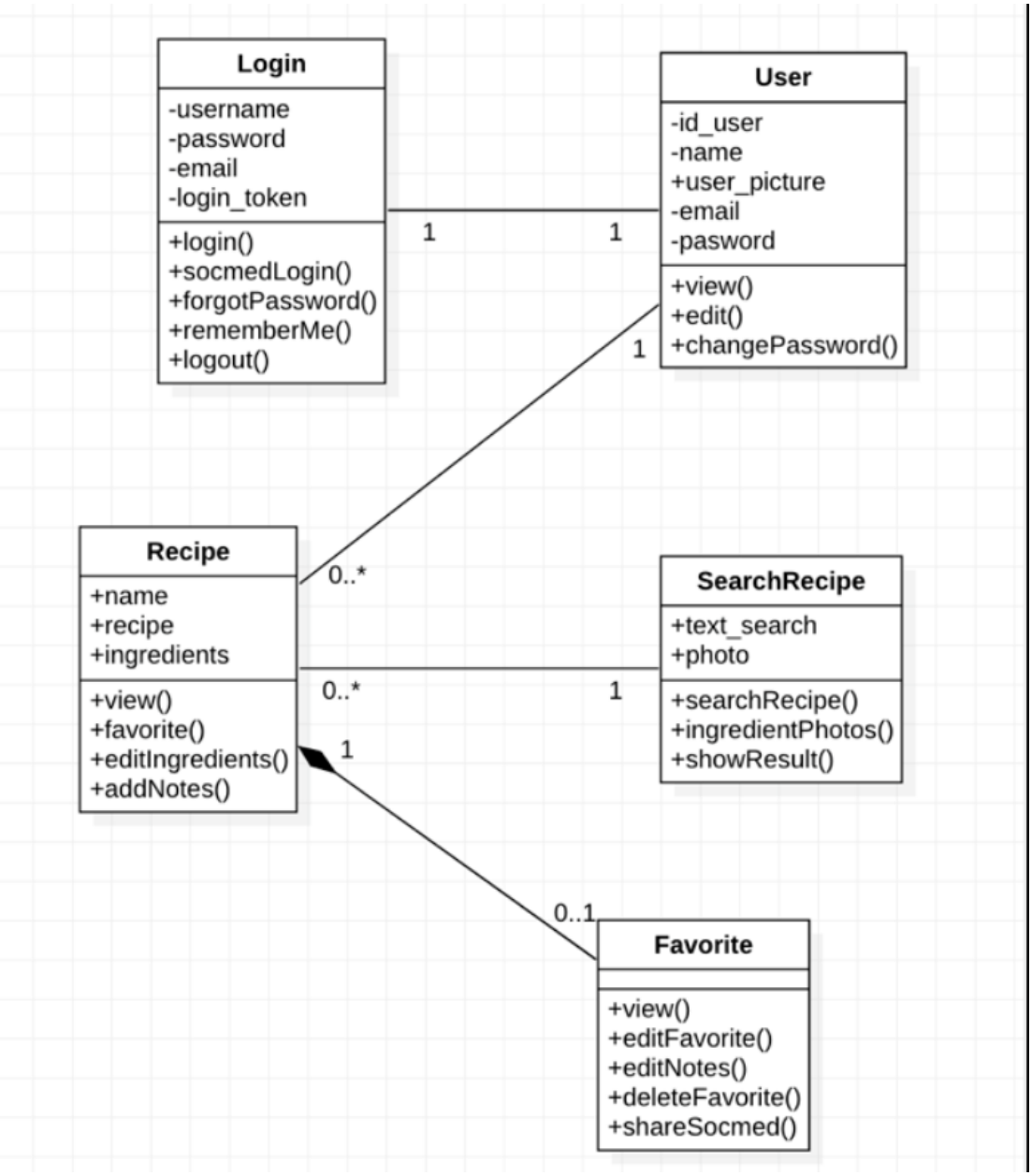
3. Key Frontend Features for AI Recipe Blogging

Regardless of the framework, the UI must include

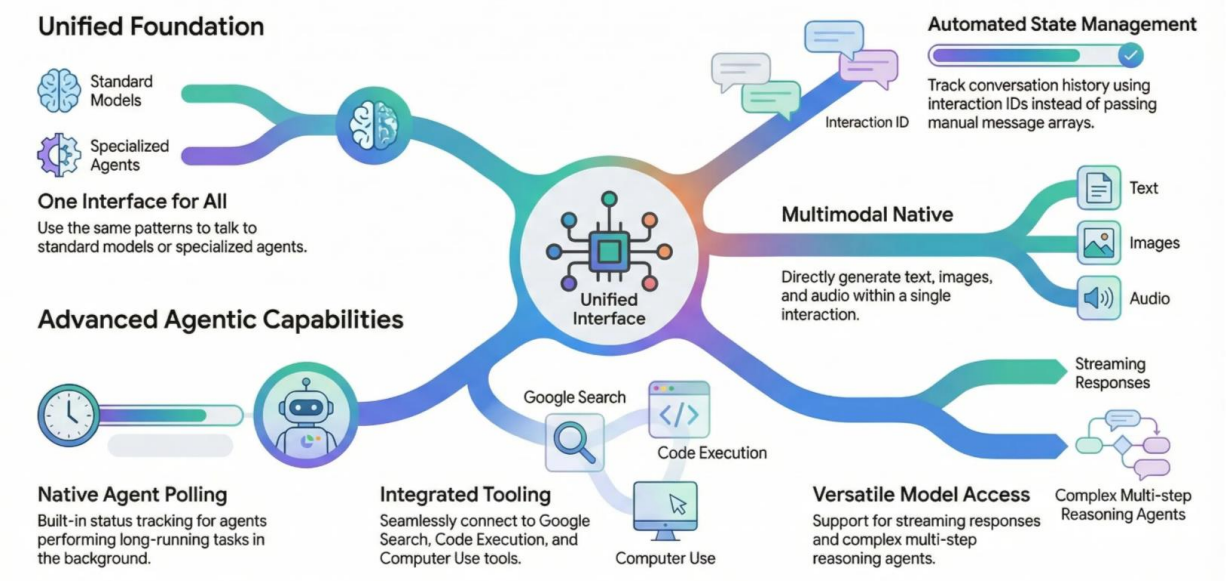
Feature	Streamlit	Flask + Custom Web UI
Development Speed	Extremely Fast (minutes/hours)	Slower (requires HTML/CSS)
Skill Required	Pure Python	Python + HTML + CSS + JS
Customization	Limited to built-in components	Highly Customizable
Best Use Case	Rapid Prototyping/MVP	Production-Ready/Full Blog

2.Backend logic implementation:

Backend logic for an AI-driven "Flavour Fusion" recipe blogging platform combines ingredient-based search, Large Language Model (LLM) generation, and structured database management. The goal is to create, store, and serve unique, personalized recipes. The backend connects user inputs (available ingredients, cuisine preferences) with generative AI APIs (like OpenAI or Gemini) to create new, blended recipes.



3.Gemini API integration:



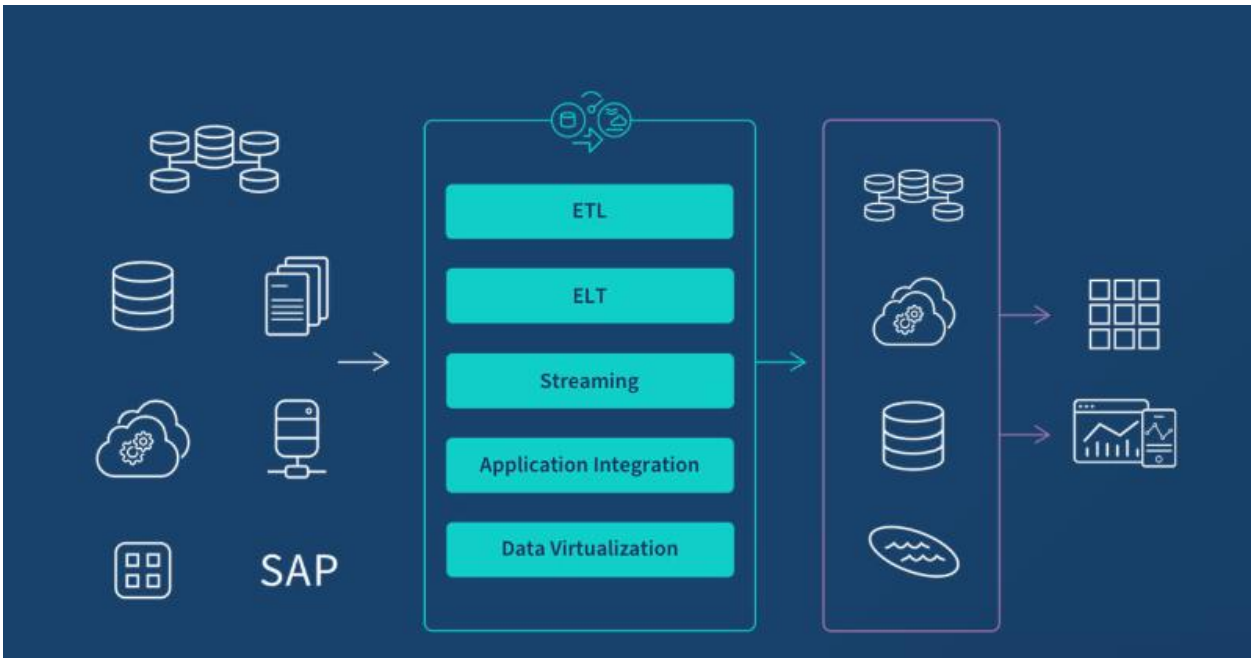
4.Database integration (if used):

The Role of Database Integration in Flavour Fusion

Instead of just storing text, the database integrates with machine learning models (like CNNs for image recognition and LLMs for text generation) to:

- **Store Structured Knowledge:** Maintain a repository of ingredients, nutritional data, cooking methods, and flavor profiles.

- **Fuel Personalized AI Generation:** Use user history (past searches, dietary needs) to feed into the AI model, which then generates custom "fused" recipes.
- **Bridge Visual Inputs:** In projects like "Flavour Fusion," a visual recognition model processes food photos, while the database provides the corresponding recipe metadata for that dish.
- **Power RAG (Retrieval-Augmented Generation):** The database stores vast, accurate recipe data, which the AI retrieves first to generate more reliable, fact-checked recipes rather than guessing.



5.Modular code structure:

A modular code structure for an AI-driven "Flavor Fusion" recipe blog divides functionality into independent "modules". This allows for easier updates, debugging, and scaling.

Core Modules

Using a modular approach, the system is broken down into these core modules:

- **Frontend User Interface:** Allows users to interact with the blog.
- **Image Processing Module:** Identifies ingredients in uploaded images.
- **Flavor Fusion Engine:** Creates new recipes by combining ingredients and techniques.
- **Vector Database:** Stores and retrieves recipes.

CMS Module: Manages blog content

6. Error handling:

Error handling in an AI-driven "Flavor Fusion" application—which combines ingredients from different cuisines to generate new recipes—is crucial to prevent the generation of inedible, dangerous, or nonsense recipes. Because AI models are probabilistic (often "hallucinating" facts), the error-handling framework must be robust, validating inputs, AI outputs, and external API dependencies.

```
participant User as Client (Browser/Mobile)

    participant Server as Your Secure Backend (Node.js/Python)

    participant Secret as Environment Variables (.env)

    participant AI as AI API (OpenAI/Gemini)



    User->>Server: Requ
```

7. API key security:

By using a **backend-for-frontend (BFF)** or a **proxy server**. The client (user's browser) Securing API keys is the most critical aspect of building an AI-driven application like **Flavour Fusion** (AI recipe blogging), as exposed keys can lead to unexpected financial costs, unauthorized access to your account, and depletion of your usage quotas.

For a recipe app using AI (e.g., OpenAI API), the key must **never** be exposed in client-side code (JavaScript in the browser or mobile app).

The Core Principle: Backend-for-Frontend (BFF) Pattern

The only truly secure way to use API keys makes a request to your server, which then makes the call to the AI service, hiding the key from the public.

```
    "Generate Vegan Recipe" (No Key)

    Note right of Server: Validates user session
```

Server->>Secret: Retrieve API Key securely

Secret-->>Server: Return API Key

Server->>AI: API Call (Includes Secret Key + Prompt)

AI-->>Server: JSON Recipe Data

Server-->>User: Structured Recipe Data