Project - High Level Design

on

Entertainment Content Generator (Script Craft)

Course Name: Generative AI

Institution Name: Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1.	Bhupendra Singh Sisodiya	EN22CS301281
2.	Ayush Meena	EN22CS301249
3.	Damita Pathak	EN22CS301306
4.	Deshansh Sharma	EN22CS301320
5.	Bhavesh Khatri	EN22CS301262

Group Name: 10D3

Project Number: GAI-34

Industry Mentor Name: Mr. Suraj Nayak

University Mentor Name: Prof. Vineeta Rathore

Academic Year: 2025-26

Table of Contents

1. Introduction

1.1 Scope of the Document

This document details the architecture, data flow, and system design of the ScriptCraft AI application. It is a specialized GenAI content generation tool tailored for Entertainment professionals designed to automate and format complex documentation like script scenes.

1.2 Intended Audience

- **System Architects:** To understand the Vector DB integration and Prompt Engineering orchestration.
- **UI/UX Developers:** To manage the Streamlit interface and real-time generation previews.
- **Backend Developers:** To review Python logic and LLM API integrations.

1.3 System Overview

ScriptCraft AI leverages sophisticated Prompt Engineering techniques and a Retrieval-Augmented Generation (RAG) architecture. By utilizing a Vector Database for context and memory management and the Gemini LLM API as the language generation engine , it transforms basic topics into high-quality, professional-grade outputs.

---

2. System Design

2.1 Application Design

The application is built on a Streamlit framework for rapid UI development with Python. The backend relies on Python to orchestrate the flow between the user's input, the Vector Database , and the Gemini LLM API.

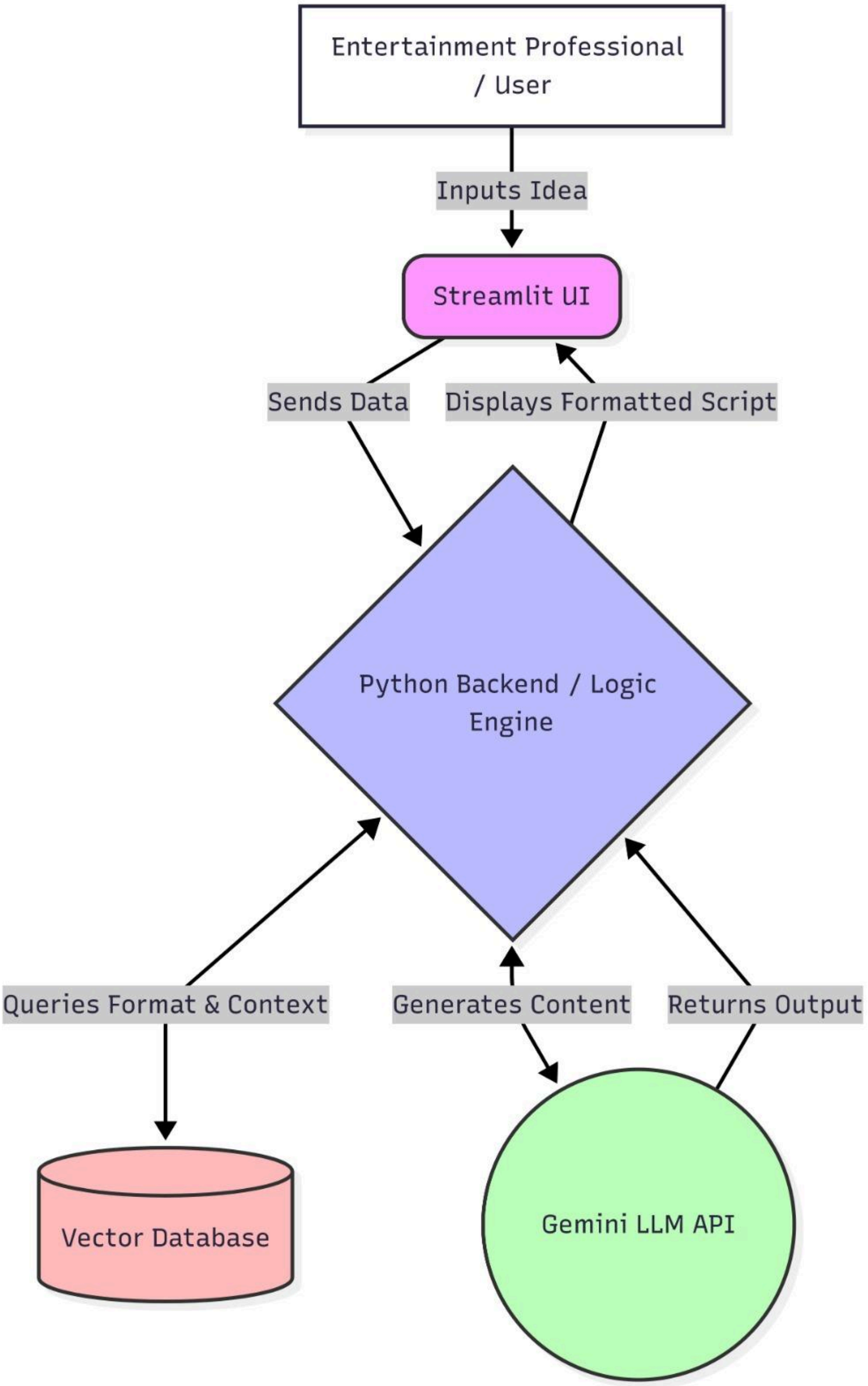

Fig.1 System Architecture Diagram

2.2 Process Flow

1. **User Input:** User provides a basic topic, idea, or plot concept.
2. **Prompt Engineering:** Enhance & structure the input.
3. **Vector DB Query:** Retrieve relevant context & examples.
4. **LLM Generation:** AI creates professional content.
5. **Formatted Output:** Industry-standard script scene is delivered.

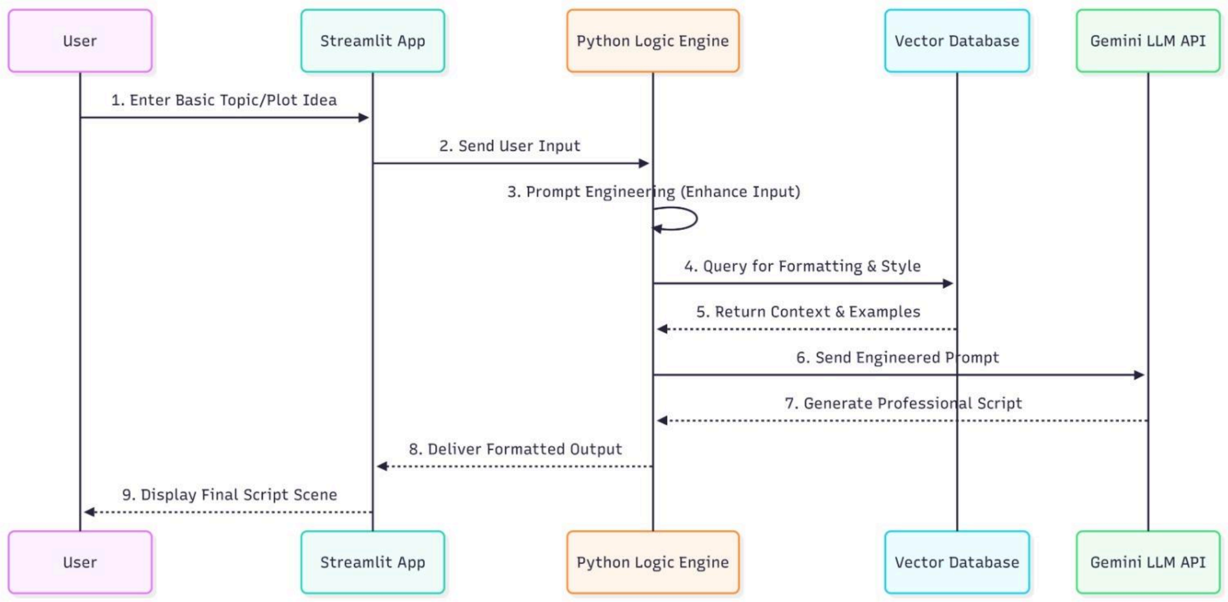
User
Streamlit App
Python Logic Engine
Vector Database
Gemini LLM API
1. Enter Basic Topic/Plot Idea
2. Send User Input
3. Prompt Engineering (Enhance Input)
4. Query for Formatting & Style
5. Return Context & Examples
6. Send Engineered Prompt
7. Generate Professional Script
8. Deliver Formatted Output
9. Display Final Script Scene
User
Streamlit App
Python Logic Engine
Vector Database
Gemini LLM API

Fig.2 Process Flow (Sequence) Diagram

2.3 Information Flow

The flow operates sequentially with retrieval augmentation: User Input -> Prompt Engineering Layer -> Vector DB Retrieval -> LLM Generation -> Output Formatting -> Streamlit UI Rendering

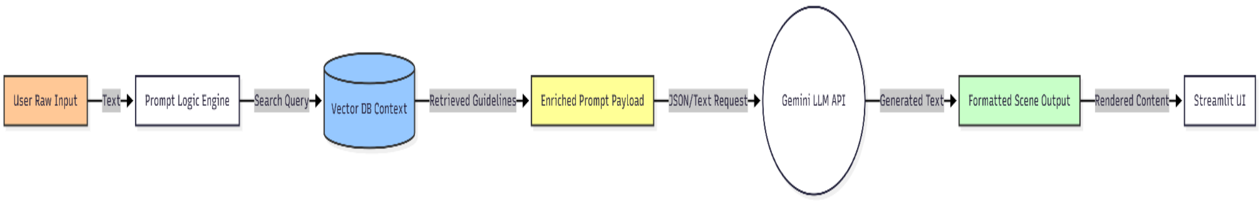

Fig.3 Information Flow Diagram

2.4 Components Design

- **Creative Workspace (UI):** A Streamlit interface for interactive and responsive design.
- **Context Manager (Vector DB):** Handles efficient semantic search and retrieval for example scripts and scenes.
- **Prompt Orchestrator (Python):** Dynamically constructs context-rich prompts using extensive libraries for data processing.
- **Generation Engine (Gemini LLM API):** Provides advanced natural language generation fine-tuned for creative writing tasks.

2.5 Key Design Considerations

- **Consistency:** Maintains narrative voice and emotional tone throughout generated content and achieves 95%+ format accuracy.
- **Efficiency Gain:** Reduces drafting time by 70-80%.

2.6 API Catalogue

Provider	Model	Purpose
Google	Gemini 3 Flash	Language Generation Engine , Context-aware content creation
Vector DB	Embedding Model	Efficient semantic search and retrieval

3. Data Design

3.1 Data Model

The system operates on a transient JSON-based data model for scene generation:

JSON

```
{
  "scene_heading": "string",
  "action_lines": ["list"],
  "characters": [
{
"name": "string",
    "dialogue": "string",
    "parenthetical": "string"
  }
 ],
 "tone_metadata": "string"
}
```

3.2 Data Access Mechanism

Data is routed via API calls to the Google Gemini endpoint and Vector DB using secure environment tokens

3.3 Data Retention Policies

Currently, the application uses Volatile Retention. Data exists only within the `st.session_state` and is cleared upon browser refresh or session termination.

3.4 Data Migration

N/A (Stateless application).

---

4. Interfaces

- **GUI:** Streamlit-based web interface.
- **External APIs:**
  - google.generativeai for LLM interaction.
  - Vector DB client for context retrieval.

---

5. State and Session Management

The application utilizes `st.session_state` to store user inputs and generated content. This ensures that when the UI reruns (a standard Streamlit behavior), the generated design is not lost

---

6. Caching

The persistence of the result in session state acts as a manual cache for the duration of the user's visit. Static formatting rules from the Vector DB can be cached using `@st.cache_data`.

---

7. Non-Functional Requirements

7.1 Security Aspects

- **Token Management:** API keys are retrieved via `os.environ.get`, ensuring no secrets are hardcoded.

7.2 Performance Aspects

- **Latency:** Average generation time is 3-5 seconds , dependent on the response times of the remote endpoints.
  +1
- **Concurrency:** The application is synchronous; the user sees a loading state during generation.

---

8. References

- **Streamlit Documentation:** docs.streamlit.io
- **Google Gemini API Guide:** ai.google.dev/gemini-api/docs