Project - High Level Design

On

Entertainment Content Generator

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	Aman Sharma	EN22CS301109
2	Anusha Singh Panwar	EN22CS301179
3	Ananya Bhatia	EN22CS301117
4	Ankit Nagar	EN22CS301134
5	Arpit Patidar	EN22CS301199

Group Name: Group 02D2

Project Number: GAI-14

Industry Mentor Name:

University Mentor Name: Dr. Hemlata Patel

Academic Year:

1. Introduction

The **Entertainment Content Generator** is a Generative AI-based system designed to automate the creation of structured entertainment content such as concepts, loglines, pitches, outlines, character sketches, and screenplay scenes.

The system leverages Large Language Models (LLMs) along with semantic vector memory to ensure narrative consistency across multiple content generation stages. It provides a guided creative workflow that transforms a seed idea into professionally formatted entertainment material.

The system is built using:

- Streamlit (Frontend)
- Google Gemini LLM
- FAISS Vector Database
- SentenceTransformer Embeddings

1.1 Scope of the document.

This document provides a High-Level Design (HLD) of the Entertainment Content Generator system. It covers:

- System architecture
- Application design
- Process and information flow
- Component breakdown
- API usage
- Data handling mechanisms
- Non-functional requirements
- Security and performance aspects

1.2 Intended Audience

This High-Level Design (HLD) document is intended for stakeholders involved in the evaluation, development, and enhance
ment of the Entertainment Content Generator system.
The primary audience includes:

- **University Mentors and Academic Evaluators** – To review the system architecture,

design approach, and implementation strategy for academic assessment.

- **Industry Mentors** – To evaluate the practical applicability, architectural soundness, and real-world relevance of the solution.
- **Developers and Project Team Members** – To understand the overall system structure, component interactions, and workflow for development, maintenance, and future enhancements.
- **System Architects and Technical Reviewers** – To assess design patterns, modularity, scalability, and integration of Generative AI components.
- **Quality Assurance (QA) Personnel** – To understand system flow and identify areas for functional and performance testing.

This document provides a structured overview of the system's design to ensure clarity, maintainability, and technical transparency.

1.3 System overview.

The **Entertainment Content Generator** is a Generative AI-based application designed to assist users in creating structured and professional entertainment content from a simple seed

idea. The system transforms user input into multiple stages of narrative development, ensuring consistency, coherence, and industry-standard formatting throughout the creative process.
The application integrates a Large Language Model (LLM) with a semantic memory mechanism to maintain contextual continuity across different stages of content generation.

1.3.1 Purpose of the System

The primary purpose of the system is to:

- Convert a basic idea into structured entertainment content
- Automate multi-stage narrative development
- Maintain consistency in tone, genre, and character development
- Provide professionally formatted outputs suitable for creative industries

1.3.2 Core Functionality

The system generates content in the following structured stages:

1. **Concept Development**
2. **Logline Creation**
3. **Elevator Pitch Generation**
4. **Story Outline Development**
5. **Character Profile Creation**
6. **Scene Writing**

Each stage builds upon the previous one to create a complete narrative framework.

1.3.3 Key Features

- Multi-stage content generation pipeline
- Context-aware generation using semantic vector memory
- Retrieval-based continuity mechanism

- Regeneration capability for iterative refinement
- Structured and industry-aligned output formatting
- Support for both full pipeline execution and partial generation

1.3.4 High-Level Architecture

The system follows a layered architecture consisting of:

- **Presentation Layer** – Streamlit-based user interface
- **Application Layer** – Content pipeline orchestrating generation stages
- **AI Layer** – Google Gemini Large Language Model
- **Memory Layer** – FAISS-based vector database with sentence embeddings

These components work together to deliver a seamless and structured AI-powered content generation experience.

2.System Design.

2.1 Application Design

The application consists of modular components:

2.1.1 Frontend (Streamlit)

Responsibilities:

- Accept user input (idea, genre, tone, output type)
- Display generated results
- Manage session state
- Trigger regeneration
- Display stored memory

2.1.2 Content Pipeline (Core Engine)

Responsibilities:

- Control stage execution
- Fetch contextual memory
- Build prompts
- Call LLM
- Store outputs in vector database

It provides:

- run_stage()
- run_full_pipeline()

2.1.3 LLM Client

Responsibilities:

- Connect to Google Gemini API
- Apply system-level prompt constraints
- Handle retries and rate limits
- Manage temperature configuration

2.1.4 Vector Store (Memory Layer)

Technology:

- FAISS
- IndexFlatL2
- SentenceTransformer (all-MiniLM-L6-v2)

Responsibilities:

- Store generated outputs
- Generate embeddings
- Retrieve top-k similar context
- Maintain metadata (stage, content)

2.2 Process Flow.

2.3 Information Flow

The system follows a structured information flow to ensure contextual continuity and modular processing across all content generation stages.

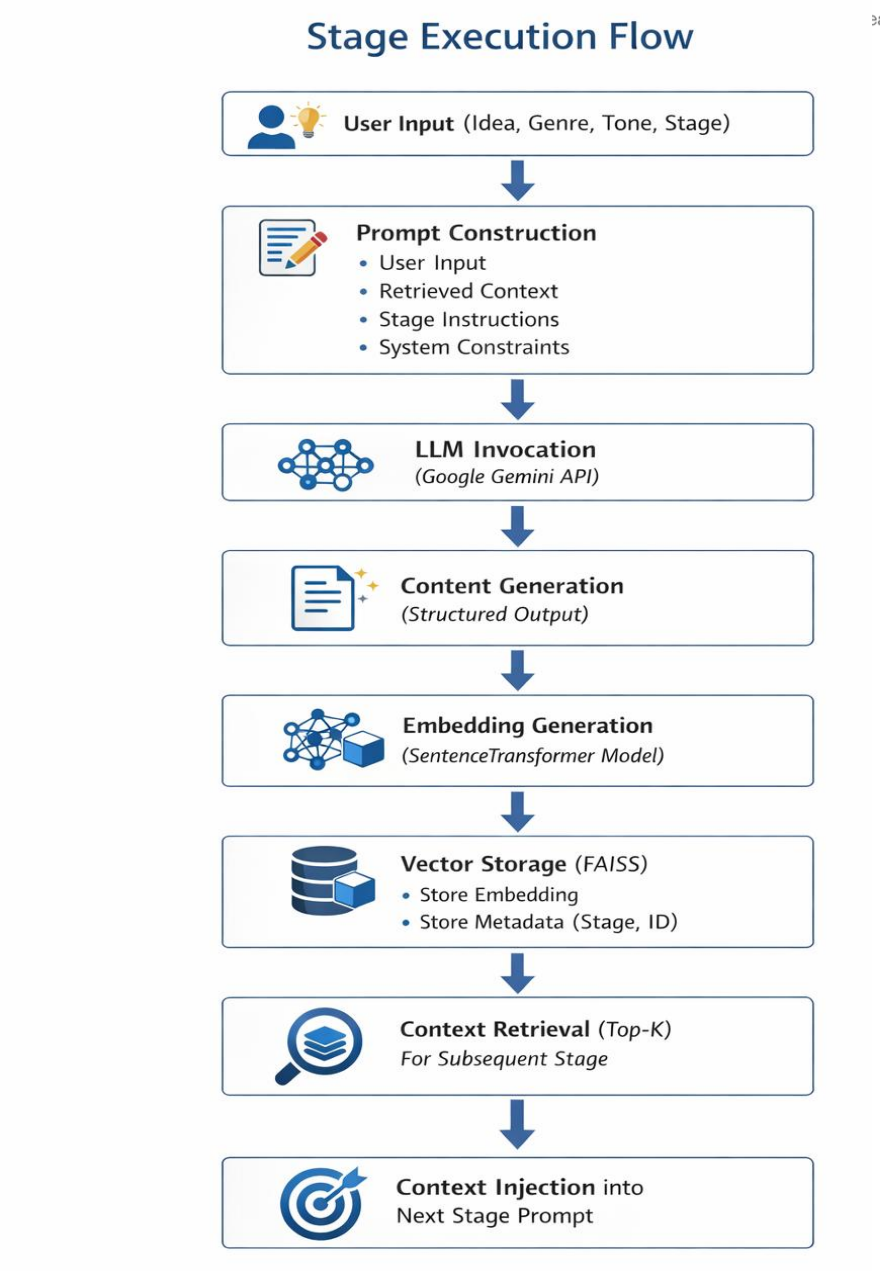
2.3.1 Stage Execution Flow

The execution of each stage follows a standardized pipeline:

1. **User Input Submission**
   The user provides a seed idea along with optional parameters such as genre, tone, and output type.
2. **Prompt Construction**
   The application layer constructs a structured prompt by combining:

- User input
- Retrieved contextual memory
- Stage-specific formatting instructions
- System-level constraints

3. **LLM Invocation**
   The structured prompt is sent to the Google Gemini API through the LLM client.
4. **Content Generation**
   The LLM generates structured output for the requested stage.
5. **Embedding Generation**
   The generated output is converted into vector embeddings using SentenceTransformer (all-MiniLM-L6-v2).
6. **Vector Storage**
   The embedding is stored in the FAISS IndexFlatL2 vector database along with metadata (stage name, timestamp, content ID).
7. **Context Retrieval for Next Stage**
   For subsequent stages, the system retrieves top-k semantically similar embeddings to maintain narrative consistency.
8. **Context Injection**
   Retrieved contextual content is injected into the next stage prompt to ensure coherence and continuity.



2.4 Component Diagram Explanation

The Component Diagram represents the high-level structural organization of the Entertainment Content Generator system. It illustrates how major modules interact with each other to perform structured AI-based content generation.

The system is divided into four primary layers:

- Presentation Layer
- Application Layer
- AI Integration Layer
- Memory Layer

Each component has clearly defined responsibilities to ensure modularity, maintainability, and scalability.

2.4.1 Presentation Layer – Streamlit Frontend

Component: User Interface (Streamlit)

Responsibilities:

- Accept user inputs (idea, genre, tone, stage selection)
- Display generated content outputs
- Provide regeneration options
- Manage session state
- Trigger pipeline execution

Interaction Flow:
The frontend sends user input to the Content Pipeline and receives structured output for display. It does not directly communicate with the LLM or vector database.

2.4.2 Application Layer – Content Pipeline (Core Engine)

Component: Content Orchestrator

Responsibilities:

- Manage execution of generation stages
- Construct structured prompts
- Retrieve contextual memory
- Invoke LLM client
- Store outputs in vector memory

Key Functions:

- run_stage(stage_name)
- run_full_pipeline()

Interaction Flow:

- Receives user request from frontend
- Requests relevant context from vector store
- Sends prompt to LLM client
- Receives generated output
- Stores embedding in memory layer

This component acts as the central controller of the system.

2.4.3 AI Integration Layer – LLM Client

Component: Google Gemini API Client

Responsibilities:

- Connect to Google Gemini API
- Apply system-level prompt constraints
- Handle API authentication
- Manage retries and rate limiting
- Control temperature and generation parameters

Interaction Flow:

- Accepts structured prompt from Content Pipeline
- Sends request to LLM
- Returns generated content

This layer abstracts external API communication to maintain loose coupling.

2.4.4 Memory Layer – Vector Store (FAISS)

Components:

- SentenceTransformer (Embedding Model)
- FAISS (IndexFlatL2)

Responsibilities:

- Convert generated text into vector embeddings
- Store embeddings with metadata
- Retrieve top-k semantically similar content
- Maintain narrative continuity

Stored Metadata Includes:

- Stage Name
- Content ID
- Timestamp
- Text Content

Interaction Flow:

- Receives text from Content Pipeline
- Generates embedding
- Stores in FAISS index
- Returns relevant context during retrieval queries

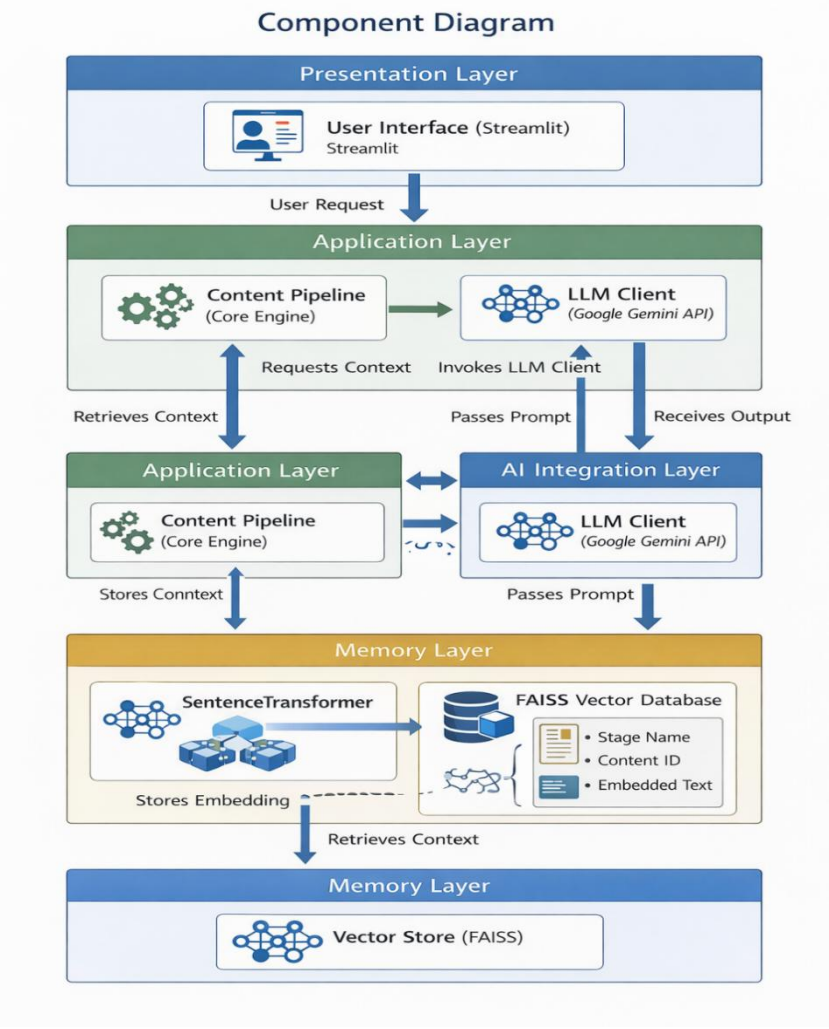
2.4.5 Component Interaction Summary

The interaction between components follows a controlled and modular pattern:

1. User interacts with Streamlit UI.
2. UI sends request to Content Pipeline.
3. Content Pipeline retrieves context from Vector Store.
4. Pipeline sends structured prompt to LLM Client.
5. LLM generates content.
6. Output is embedded and stored in FAISS.
7. Response is returned to UI for display.

This architecture ensures:

- Clear separation of concerns
- High modularity
- Scalable AI integration



2.5 Key Design Considerations

The design of the Entertainment Content Generator is guided by architectural principles to ensure scalability, maintainability, and performance.

2.5.1 Modularity

The system follows a layered and modular architecture:

- Frontend is isolated from AI logic.
- Content Pipeline acts as a central orchestrator.
- LLM client abstracts external API interaction.
- Memory layer is independent and replaceable.

This ensures easier maintenance and future upgrades.

2.5.2 Scalability

- Stateless frontend design.
- Decoupled AI integration layer.
- Vector database can be replaced with scalable alternatives.
- Pipeline supports stage-wise independent execution.

The architecture allows horizontal scaling if deployed in production.

2.5.3 Context Consistency

- Use of semantic vector embeddings.
- Top-k similarity retrieval.
- Metadata tagging by stage.

This ensures narrative coherence across multi-stage generation.

2.5.4 Extensibility

- New stages can be added to the pipeline.
- Prompt templates are configurable.
- Embedding model can be upgraded without changing frontend.

The system is designed for easy feature expansion.

2.5.5 Performance Optimization

- Efficient vector search using FAISS (IndexFlatL2).
- Controlled LLM temperature settings.
- Minimal redundant API calls.

This improves response time and system efficiency.

2.6 API Catalogue

- **Google Gemini API** – Generate structured content from prompts
- **run_stage()** – Execute single pipeline stage
- **run_full_pipeline()** – Execute all stages sequentially
- **retrieve_context()** – Fetch similar embeddings from memory
- **generate_embedding()** – Convert text into vector embeddings

3.1 Data Model

The data model now accounts for multiple LLM sources including **Google Gemini** and **Hugging Face**.

3.1.1 Data Types

1. **Generated Content Data**

   - **Content ID** – Unique identifier
   - **Stage** – Concept, Logline, Pitch, Outline, Character, Scene
   - **Text** – Generated content text
   - **Timestamp** – Creation/modification time
   - **Version** – For regeneration history
   - **LLM Source** – Google Gemini or Hugging Face

2. **Vector Embeddings Data**

   - **Embedding Vector** – Numerical representation of content
   - **Content ID** – Link to corresponding content
   - **Stage** – Stage for context relevance
   - **Metadata** – Genre, tone, output type, LLM source
   - **Top-k Retrieval Reference** – Index for semantic search

Storage Mechanism:

- FAISS vector database (IndexFlatL2) for embeddings
- JSON/NoSQL storage for metadata and original text

3.1.2 AI Layer

AI Integration Layer – LLM Client

Components:

- **Google Gemini API Client** – Primary content generation LLM
- **Hugging Face Model Client** – Optional alternative LLM for:

- Text generation
- Embedding creation
- Offline testing or experimentation

Responsibilities:

- Accept structured prompts from Content Pipeline
- Apply system-level constraints and temperature control
- Handle API calls (rate limiting, retries)
- Return structured content to Content Pipeline

3.2 Data Access Mechanism

The Entertainment Content Generator ensures secure, efficient, and structured access to both content and vector data for all stages of the pipeline.

3.2.1 Access Layers

1. **Frontend (Streamlit UI)**
    - Sends user input to Content Pipeline
    - Requests generated content for display
    - Triggers regeneration or partial stage execution
2. **Content Pipeline (Application Layer)**
    - Core orchestrator for data flow
    - Fetches context from vector store
    - Calls LLM Client for generation
    - Stores generated outputs in memory
3. **Memory Layer (FAISS + Metadata Storage)**
    - Embeddings stored in FAISS IndexFlatL2
    - Metadata (stage, genre, LLM source) in NoSQL/JSON
    - Top-k retrieval for context injection
4. **AI Layer (LLM Clients)**
    - Access structured prompts from pipeline
    - Return generated content
    - Supports Google Gemini and Hugging Face

3.2.2 Access Protocols

- **Read Access:** Retrieve top-k embeddings and metadata for context
- **Write Access:** Store generated content and embeddings after each stage

- **Controlled Access:** Only the Content Pipeline interacts with memory layer; frontend cannot directly access vectors
- **Versioning:** Each generated content version is stored to support regeneration and history

3.3 Data Retention Policies

The Entertainment Content Generator defines clear policies to manage the **lifecycle of generated content, embeddings, and metadata**.

3.3.1 Retention Duration

- **Generated Content:** Retained for **12 months** by default
- **Embeddings (FAISS):** Stored for **12 months** to ensure context continuity
- **Metadata:** Retained alongside content to maintain audit trail and stage tracking
- **Regenerated Versions:** Stored as separate versions, linked to original content ID

3.3.2 Storage & Archival

- Active data is stored in a **FAISS vector database** and associated **NoSQL/JSON storage**
- After the retention period, **older content is archived or deleted** automatically
- Optional **manual export** for backup or academic evaluation

3.3.3 Access Control

- Only **Content Pipeline and authorized admins** can access stored data
- Frontend users cannot directly access stored vectors or metadata
- LLM source (Google Gemini or Hugging Face) is logged for reproducibility

3.4 Data Mitigation

Data mitigation ensures that sensitive or unnecessary data is **minimized, anonymized, and securely handled** throughout the content generation pipeline.

3.4.1 Purpose

- Reduce storage of redundant or irrelevant data
- Protect user inputs and generated outputs
- Maintain compliance with privacy and security best practices

3.4.2 Mitigation Strategies

1. **Data Minimization**

    - Only necessary input data (idea, genre, tone) is stored
    - Unused or irrelevant intermediate data is not retained

2. **Anonymization**

    - User identifiers are separated from content and metadata
    - Content stored in FAISS/NoSQL contains no personally identifiable information (PII)

3. **Version Control & Cleanup**

    - Older or obsolete versions of generated content are flagged
    - Automatic deletion of expired content as per retention policies

4. **Access Restrictions**

    - Only the Content Pipeline and authorized admins access stored vectors and metadata
    - Frontend users cannot access internal embeddings or vector data

5. **Secure Storage & Transmission**

    - All embeddings and metadata are stored in secure vector/NoSQL databases
    - API communications are encrypted using HTTPS/TLS

3.4.3 Benefits

- Reduces risk of sensitive data exposure
- Ensures compliance with data protection principles
- Maintains **system efficiency** by limiting unnecessary storage
- Supports safe **regeneration and iterative refinement** without exposing raw input

4. Interfaces

The Entertainment Content Generator interacts with users and external services through clearly defined interfaces.

4.1 User Interface (UI)

The system provides a web-based interface developed using Streamlit. The interface allows users to:

- Enter a seed idea, genre, and tone
- Select output type (Full Pipeline or Partial Generation)
- View generated content in structured format
- Regenerate specific stages
- Monitor generated outputs

The UI ensures ease of use, structured presentation, and smooth interaction throughout the content generation workflow.

4.2 External API Interface

The system integrates with the Google Gemini API for AI-based content generation.

- Authentication: API Key-based
- Request Type: Prompt-based generation
- Response Type: Structured textual output

This external interface enables AI-powered processing while maintaining separation between the application logic and the AI model.

5. State and Session Management

The system manages application state using Streamlit's session management mechanism.

The following information is maintained during a session:

- Initialized content pipeline object
- Generated outputs of each stage
- API key validation status
- Selected execution mode
- Retrieved contextual memory

Session-based management ensures:

- Continuity during user interaction
- Support for regeneration functionality
- Temporary storage of outputs without persistent database dependency

Currently, state is maintained only for the active session.

6. Caching

The system does not implement a dedicated caching mechanism. However:

- FAISS enables efficient similarity search for quick retrieval of contextual memory.
- Embeddings are stored in memory for fast access during execution.

Future enhancements may include:

- Response caching to reduce repeated API calls
- Integration with caching systems such as Redis for improved scalability

7. Non-Functional Requirements

The system satisfies the following non-functional requirements:

7.1 Usability

- Simple and intuitive user interface
- Structured content presentation
- Clear stage-wise generation process

7.2 Maintainability

- Modular architecture
- Clear separation of concerns
- Easily extendable pipeline structure

7.3 Scalability

- Layered architecture allows integration of persistent databases
- Can be extended to API-based or cloud deployment

7.4 Reliability

- Retry mechanism implemented for API calls
- Controlled sequential stage execution

• Security Aspects

The system incorporates the following security measures:

- API key-based authentication for Gemini access
- No long-term storage of user data
- Limited exposure of backend components
- Secure communication with external AI services

Future improvements may include:

- Encrypted storage of API keys
- Role-based access control
- Secure deployment environment configuration

• Performance Aspects

System performance depends primarily on:

- LLM response time
- Network latency
- Input size and complexity

Performance optimizations include:

- Lightweight embedding model (MiniLM)
- Efficient FAISS vector search
- Sequential pipeline execution
- Controlled prompt structuring

The system is suitable for academic and prototype-scale deployment.

8. References

- Google Gemini API Documentation
- FAISS Documentation
- SentenceTransformers Documentation
- Streamlit Documentation