Project - High Level Design
On
ProjectTitle

Sports 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	Ayushi Parsai	EN22CS301257
2	Devansh Mittal	EN22CS301326
3	Ayush Kourav	EN22CS301248
4	Deepesh Vyas	EN22CS301319

Group Name:06D3

Project Number:GAI-30

Industry Mentor Name: Prof Suraj Nayak

University Mentor Name: Prof. Vineeta Rathore

Academic Year: 2025-26

Table of Contents

1. Introduction

The project titled **"Sports Content Generator"** is a Generative AI-powered application developed to assist sports professionals in generating structured, professional-quality sports content automatically. The system is built using Python and deployed through Streamlit, with integration to the Google Gemini API for advanced text generation.

The application enables users to transform simple sports-related inputs such as match statistics, player performances, or tournament summaries into high-quality outputs including:

- Match Recaps
- Player Performance Reports
- Pre-Match Analysis
- Post-Match Reviews
- Tournament Summaries

The system ensures:

- Consistent tone and professional formatting
- Industry-standard sports terminology
- Reduced manual drafting effort
- Fast content generation

1.1 Scope of the Document

This document describes the overall system architecture, design components, data flow, API structure, and non-functional requirements of the Sports Content Generator. It provides a high-level understanding of how the system is built and how different modules interact.

1.2 Intended Audience

This document is intended for:

- Software Developers

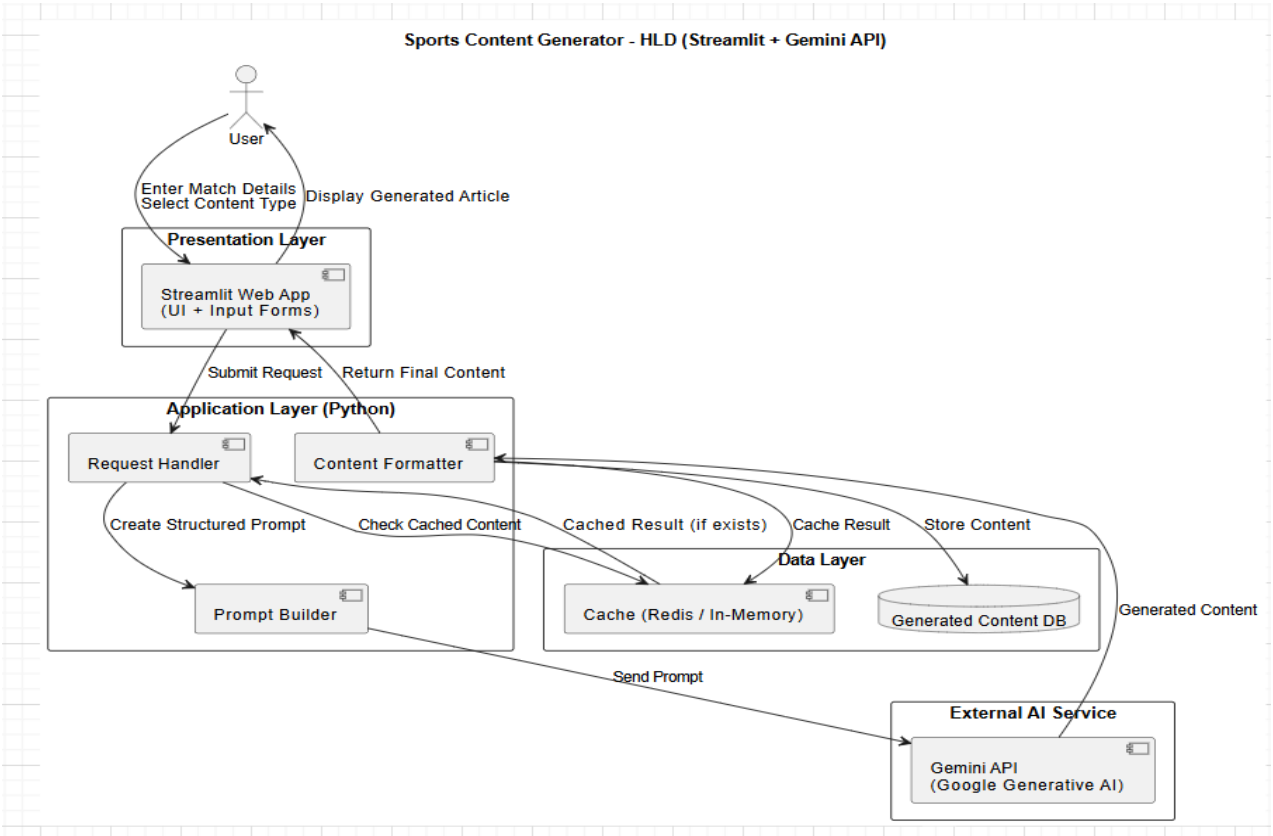
- Technical Interview Panels
- DevOps Engineers
- QA Engineers

1.3 System Overview

The Sports Content Generator consists of:

- Frontend Web Interface (User Input + Display)
- Backend Application Server (Business Logic)
- Sports Data API Integration
- LLM API Integration (Hugging Face / OpenAI)

2. System Design



2.1 Application Design

The system follows a two-layer architecture:

1) Presentation Layer – Streamlit UI

- Built using Streamlit
- Provides:
    - Text input fields
    - Dropdown selectors
    - Content type selection
    - Generate button
    - Output display panel

2) Application & AI Layer – Python + Gemini API

- Prompt Engineering Module
- API Communication Module
- Response Formatting Module
- Error Handling Module

2.2 Process Flow

1. User opens the Streamlit web application
2. User selects type of content (e.g., Match Recap)
3. User enters basic match details
4. Python backend validates inputs
5. System constructs structured prompt

6. Prompt sent to Gemini API
7. Output formatted for readability
8. Content displayed on Streamlit interface

2.3 Information Flow

User Input (Streamlit UI)
↓
Python Backend
↓
Prompt Engineering Engine
↓
Gemini API
↓
Generated Response
↓
Formatter
↓
Streamlit Display

All API calls are made securely using an API key stored in environment variables.

2.4 Components Design

1. Streamlit Interface Module

- User Input Forms
- Dropdown menus
- Output Display Panel
- Download Option

2. Prompt Engineering Module

-

- Predefined templates
- Dynamic variable injection
- Tone control instructions
- Formatting constraints

3. Gemini API Integration Module

- API key authentication
- HTTPS request handling
- Response parsing

4. Response Formatter

- Adds headings
- Formats paragraphs
- Ensures professional tone

2.5 Key Design Considerations

- Secure storage of Gemini API key
- Consistent sports journalism tone
- Minimal latency in API calls
- Clean and user-friendly interface
- Error handling for API failures
- Scalability for future sports types

3. Data Design

3.1 Data Model

Since the Sports Content Generator is primarily an AI-driven content generation tool, it does not require a highly complex relational database. However, minimal structured data storage can be implemented for:

- User session data
- Prompt templates
- Generated content history
- Logs and analytics

Logical Data Entities

1. **User (Optional)**
    - user_id
    - username
    - email
    - role
2. **Prompt Template**
    - template_id
    - content_type
    - template_text
    - created_at
3. **Generated Content**
    - content_id
    - user_id
    - input_data
    - generated_output
    - timestamp

4. **API Logs**
    - request_id
    - response_time
    - status

For a lightweight implementation, storage can be handled using:

- SQLite
- JSON files
- Or in-memory session state (Streamlit session state)

3.2 Data Access Mechanism

Data can be accessed using:

- Python file handling (for JSON storage)
- SQLite queries (if database used)
- Streamlit session state for temporary storage

Access Flow:

User Request
↓
Python Backend
↓
Read/Write Data
↓
Return Response

3.3 Data Retention Policies

- Generated content stored for limited duration (optional)
- API logs retained for debugging

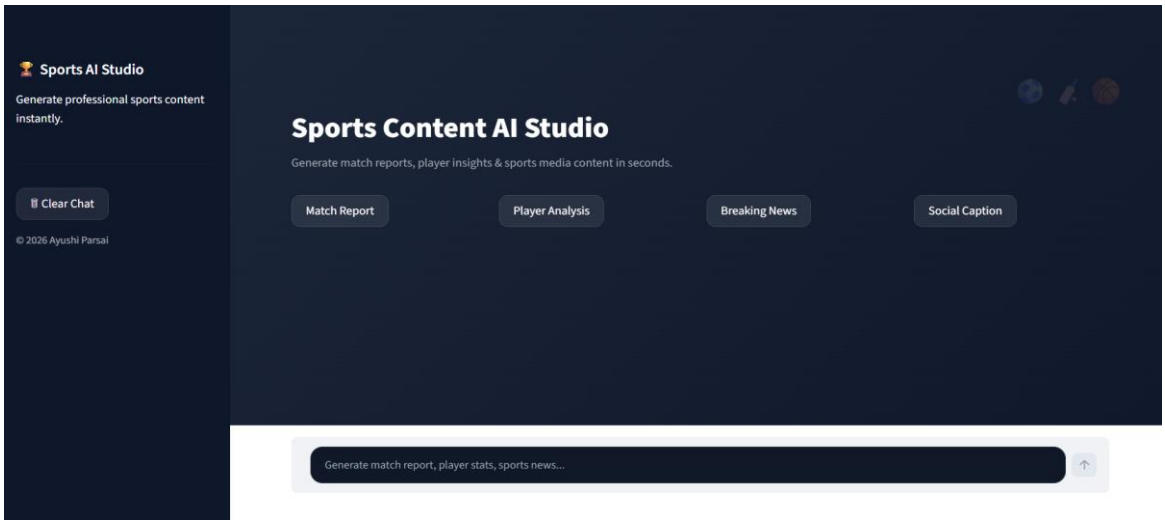
- No sensitive personal data stored
- API key stored securely in environment variables

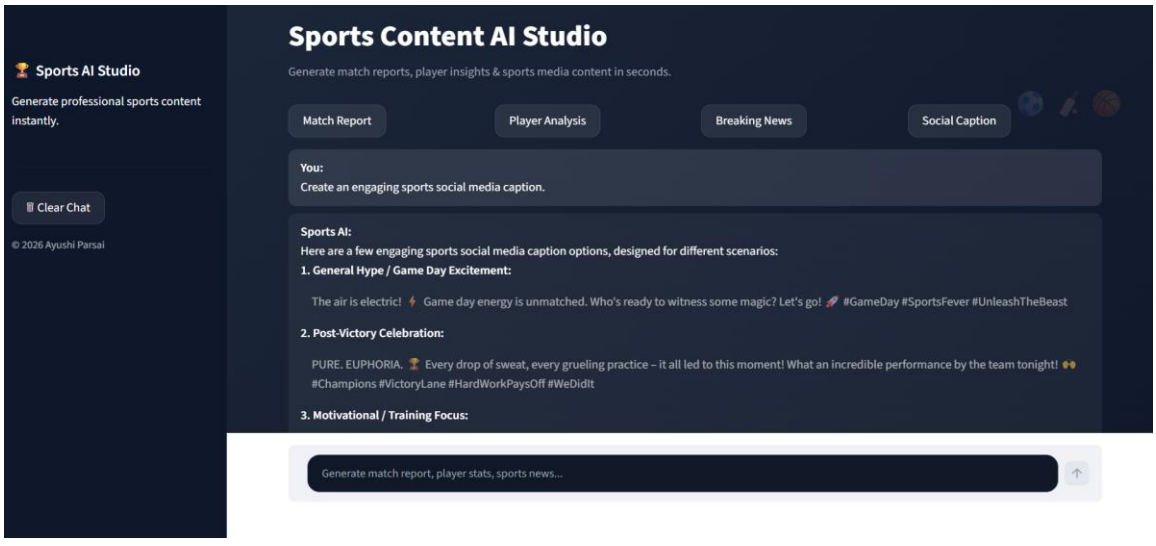
3.4 Data Migration

Currently not required due to lightweight architecture.
Future migration can include:

- Moving from SQLite to PostgreSQL

4. Interfaces





4.1 User Interface

The user interface is developed using Streamlit and follows a modern dashboard-style layout. It is divided into two major sections: the sidebar panel and the main content area.

The sidebar panel includes the project title "Sports AI Studio," a short description stating "Generate professional sports content instantly," and a Clear Chat button. The Clear Chat button resets the session state and clears previous interactions. The sidebar also contains footer information such as developer credits.

The main content area displays the heading "Sports Content AI Studio" along with a subtitle explaining the purpose of the application. Below the heading, four content selection buttons are provided:

Match Report
Player Analysis
Breaking News
Social Caption

Each button allows the user to choose the type of sports content they want to generate. Based on the selected option, a specific prompt template is activated in the backend.

The interface follows a chat-based interaction model. User inputs are displayed in one block, and AI-generated responses appear in a separate block. This conversational format enhances usability and provides a structured content generation experience.

At the bottom of the interface, a persistent input field allows users to enter match details, player statistics, or custom instructions. When the user submits the input, the request is processed and the generated content is displayed in the chat section.

5. State and Session Management

Streamlit session state is used to:

- Store user inputs temporarily
- Store generated outputs
- Maintain UI state between refreshes

Example:

- st.session_state["generated_text"]

Session data is cleared once application is restarted.

6. Caching

Caching is implemented to:

- Reduce repeated API calls
- Improve performance

Streamlit provides caching decorators such as:

- st.cache_data
- st.cache_resource

- Use Case: If the same input is provided, cached output is returned instead of calling Gemini API again.

7. Non-Functional Requirements

7.1 Security Aspects

- API key stored in environment variables
- HTTPS communication with Gemini API
- No sensitive personal data storage
- Input validation to prevent injection attacks
- Error handling without exposing system details

7.2 Performance Aspects

- Average response time: 2–5 seconds (depending on API latency)
- Lightweight architecture ensures low system load
- Efficient prompt construction
- Caching reduces redundant calls
- Scalable to cloud deployment (Streamlit Cloud / AWS / GCP)

8. References

1.Streamlit Documentation
Streamlit Inc. (2024). *Streamlit Documentation – Build Data Apps in Python.*
Available at: https://docs.streamlit.io
(Accessed: 2026)

2.Google Gemini API Documentation
Google AI. (2024). *Gemini API Documentation – Google Generative AI.*
Available at: https://ai.google.dev
(Accessed: 2026)

3.Python Official Documentation
Python Software Foundation. (2024). *Python 3 Documentation.*
Available at: https://docs.python.org/3/
(Accessed: 2026)

4.PlantUML Documentation (For Diagrams)

PlantUML. (2024). *PlantUML Language Reference Guide.*

Available at: https://plantuml.com

(Accessed: 2026)