|  |  |
| --- | --- |
| **Project Title** | **Building a RAG-Based Question Answering System Using MongoDB Movies Dataset** |
| **Skills take away From This Project** | * MongoDB to SQL data migration using Python * SQL schema design and relational data modeling * Data preprocessing and transformation * LangChain-based RAG architecture setup * Prompt-based intelligent response generation * Integration of LLMs with structured data |
| **Domain** | **Media & Entertainment / AI-Powered Chat Systems** |

**Problem Statement:**

Design and develop an intelligent system that can answer user queries based on the sample\_mflix movie dataset by using a Retrieval-Augmented Generation (RAG) architecture. The project should involve migrating NoSQL data (from MongoDB) into a SQL structure and leveraging that structured data in a prompt-based QA system.

**Business Use Cases:**

AI chat assistant for movie recommendation platforms (e.g., Netflix, Amazon Prime)

Intelligent FAQ system for cinema chains or OTT apps

Semantic search for media metadata and user feedback

Insights from user reviews and ratings for business decisions

**Approach:**

1. **Extract & Transform Data:**
   * Use Python to extract all collections (movies, comments, users, sessions, etc.) from sample\_mflix MongoDB.
   * Normalize/flatten nested documents.
   * Create relational schemas in a SQL database (MySQL/PostgreSQL).
   * Push transformed data into SQL using Python and SQLAlchemy or Psycopg2.
2. **Preprocessing & Embedding:**
   * Combine relevant fields (title, plot, reviews) into documents.
   * Use OpenAI embeddings or Hugging Face transformers to vectorize documents.
   * Store embeddings in a FAISS vector store.
3. **RAG Architecture:**
   * Use LangChain or similar framework.
   * Implement retrieval logic to fetch relevant text chunks from the vector store.
   * Use OpenAI GPT or other LLMs to generate responses based on user prompts.
4. **Build QA Interface:**
   * Accept user prompts, retrieve relevant chunks, and generate responses.
   * Deploy via Streamlit, Flask, or any web-based UI.

**Results:**

 A fully functional movie QA system capable of understanding user queries.

 Data successfully migrated and normalized from MongoDB to SQL.

 Searchable vector store of movie plots, reviews, and metadata.

 Real-time, accurate, and context-aware answers to queries.

**Project Evaluation metrics:**

| **Metric** | **Description** |
| --- | --- |
| Data migration completeness | All collections properly mapped and pushed |
| Embedding accuracy | Similar queries return similar results |
| Response relevance | Human evaluation of generated answers (precision) |
| System latency | Time taken from prompt to final answer |
| Usability of the interface | User feedback on the prompt/response system |

**Technical Tags:**

MongoDB, SQL, LangChain, Python, FAISS, Embeddings, RAG, LLM, Vector Search, Data Engineering, NLP

**Data Set:**

**Source:** sample\_mflix dataset (MongoDB Atlas sample data)

**Collections Used:** movies, comments, embedded\_movies, users, sessions, theaters

**Format:** JSON/Document (MongoDB), migrated to relational tables

**Variables:** Movie metadata, user reviews, login data, theater geolocation, etc.

**Data Set Explanation:**

 The dataset simulates a movie streaming platform.

 **movies** contains metadata about films.

 **comments** links users to movies and includes textual reviews.

 **users** and **sessions** model authentication data.

 **Preprocessing Steps:**

* Flatten nested documents
* Create foreign key relationships (e.g., comments.movie\_id ↔ movies.\_id)
* Join relevant content (plot, reviews, cast) for embeddings

**Project Deliverables:**

 SQL schema and ER diagram

 Python scripts for MongoDB to SQL migration

 FAISS vector store creation scripts

 LangChain RAG implementation with prompt/response logic

 Final user interface (e.g., Streamlit app)

 Documentation (README, setup guide, deployment instructions)

**Project Guidelines:**

 Use Git for version control (main, dev branches)

 Use PEP8 standards for Python coding

 Modularize scripts for ETL, embedding, QA logic

 Include exception handling and logging

 Use .env for managing API keys and DB credentials

**Timeline:**

| **Week** | **Task** |
| --- | --- |
| 1 | Dataset exploration, schema design, MongoDB to SQL migration |
| 1 | Data preprocessing and embeddings (OpenAI/HuggingFace + FAISS) |
| 2 | RAG system integration with LangChain |
| 2 | QA testing, prompt engineering, UI development |
| 2 | Final testing, deployment, and documentation |

**PROJECT DOUBT CLARIFICATION SESSION ( PROJECT AND CLASS DOUBTS)**

**About Session:** The Project Doubt Clarification Session is a helpful resource for resolving questions and concerns about projects and class topics. It provides support in understanding project requirements, addressing code issues, and clarifying class concepts. The session aims to enhance comprehension and provide guidance to overcome challenges effectively.

**Note: Book the slot at least before 12:00 Pm on the same day**

**Timing: Tuesday, Thursday, Saturday (5:00PM to 7:00PM)**

**Booking link :<https://forms.gle/XC553oSbMJ2Gcfug9>**

**LIVE EVALUATION SESSION (CAPSTONE AND FINAL PROJECT)**

**About Session:** The Live Evaluation Session for Capstone and Final Projects allows participants to showcase their projects and receive real-time feedback for improvement. It assesses project quality and provides an opportunity for discussion and evaluation.

**Note: This form will Open on Saturday and Sunday Only on Every Week**

**Timing: Monday-Saturday (11:30PM to 12:30PM)**

**Booking link :** [**https://forms.gle/1m2Gsro41fLtZurRA**](https://forms.gle/1m2Gsro41fLtZurRA)