## A brief overview of the implementation, including tools used and key assumptions

**Description:**

An AI & Data Engineer Technical Task to develop a simple Retrieval-Augmented Generation (RAG) application that processes user queries based on a small dataset of documents. The system use

FAISS for efficient document retrieval and utilize LangChain with OpenAI to generate responses.

**Tools/Frameworks:**

VS Code

Faiss

LangChain

OpenAI

**Key assumptions:**

- FAISS is trusted to efficiently retrieve semantically relevant documents from a large vector index which assumes high-quality embeddings from OpenAI

with low retrieval latency for real-time use.

-LangChain Orchestrates Seamless Integration to manage the flow between query → retrieval → generation, handle prompt construction and context injection and support modularity (e.g. swapping LLMs)

-The OpenAI models (like GPT-3.5) benefit from additional context retrieved from external sources, especially for domain-specific or time-sensitive queries.

**The Design:**

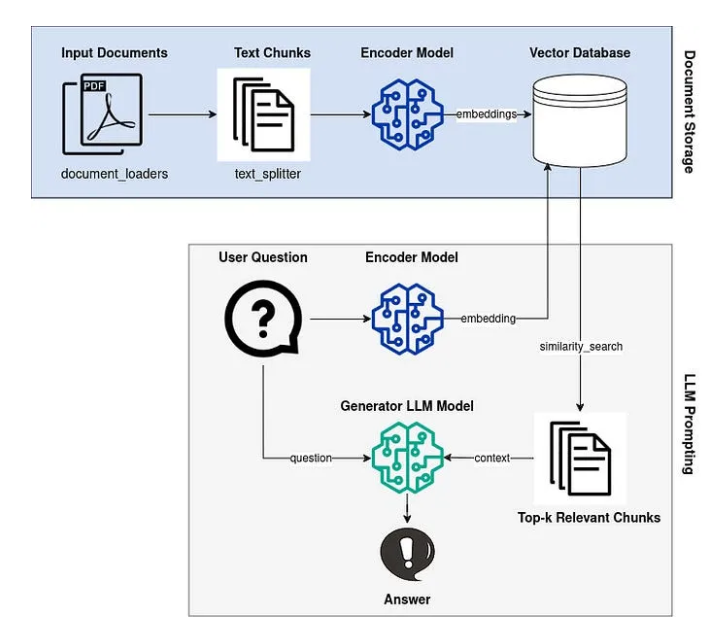
The design includes a small dataset of documents which are used to create embeddings with OpenAI models and a FAISS vector store index for efficient similarity search including LangChain.

Blockers: OpenAI licence

**SDLC:**

Using Agile methology for App creation

**The High-Level design** :



Steps:

* -Setting the environment dependancy ( .env file and CheetSheet)
* - Building the App

Phases:

= import requirenment

= Load environment variables from .env file

= Faiss settings ( inutualizing,loading/opening, save etc. )

= Functions

1. to load documents from a directory
2. to split text into chunks
3. To split documents into chunks
4. to generate embeddings using OpenAI API
5. To generate embeddings for the document chunks
6. To Add to FAISS vector store
7. to query documents using FAISS
8. to generate a response from OpenAI
9. to generate answers

**Scalability Considerations** (optional but recommended) - Outline how the system could

handle larger datasets or more complex use cases.

- overwriting the documents to improve the output

-Scaling for Larger Datasets ( advanced indexing) using partition data into clusters, improving accuracy, speed (tune retrieval nprobes)

-Memory Efficiency (apply 8-bit or 16-bit quantization to reduce memory usage while maintaining decent retrieval precision)

-GPU Acceleration for Faiss (to improves performance on massive datasets)

- use batch Embedding and Indexing Process embeddings ( to reduce API calls and latency when using OpenAI’s embedding models)

-Use Cases optimization

-Hybrid Search Techniques Combine semantic search via embeddings (with keyword-based filtering or metadata constraints to improve relevance)

-Second layer/model Reranking to retrieved documents before passing them to the LLM for generation

-Improvements regarding LangChain templetes (custom prompt templates) that guide the LLM more effectively (especially when dealing with multi-step reasoning or domain-specific queries)

-Caching and Preprocessing Cache improvenments regarding frequently accessed queries and preprocess documents to remove noise, improving both speed and accuracy