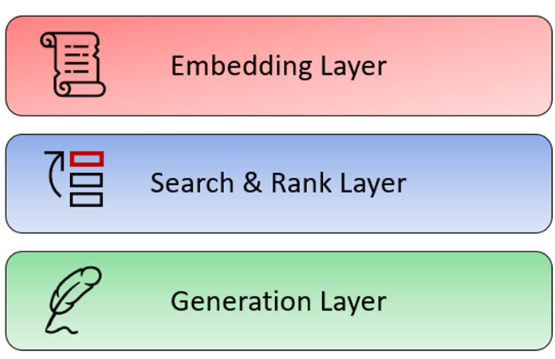
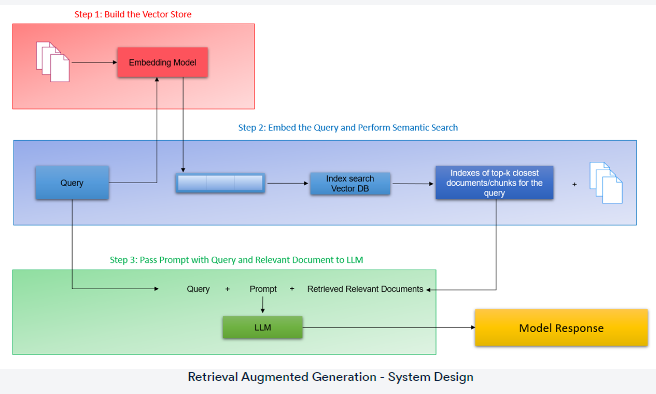
**Project Report**

**Objective** : To improvise the Mr.HelpMate AI.

**Design :** Below in the Diagram is the Design of the system





Step 1: Build the vector store: The first step is to build a vector store that can store documents along with metadata. The documents are converted to raw text and then split into chunks. Each chunk is then represented as a vector using an embedding model. The vector store is then populated with these vectors.

Step 2: Embed the query and perform semantic search: The next step is to embed the user query into the same vector space as the documents in the vector store. This is done using an embedding model. Once the query is embedded, a semantic search is performed to find the closest embedding from the vector store. The entries with the highest semantic overlap with the query are retrieved.

Step 3: Pass the prompt with the query and the relevant documents to the LLM: The final step is to pass the prompt, which is a concatenation of the query and the retrieved documents, to the LLM. The LLM generates a response based on the context of the query, the system prompt and the relevant documents passed from the search layer. The retrieved documents serve as the knowledge bank and provide the necessary context for the query to the LLM, which helps it generate a more accurate and relevant response.

**Implementation:**

I have created 3 functions for the 3 layers

rag\_function

cross\_encoder\_fn(query,results\_df)

generate\_response(query, top\_3\_RAG)

query = input()

results\_df=rag\_function(query)

print(results\_df)

if results\_df is None:

  print("No results found")

else:

  top\_3\_RAG=cross\_encoder\_fn(query,results\_df)

  response = generate\_response(query, top\_3\_RAG)

print("query==>",query)

print("\n".join(response))

**Challenges and Lessons Learned:**

We have to find a way to prompt in a better manner.