**Project Documentation: Interactive QA Bot Using RAG and Docker Deployment**

**Overview**

This project builds a **Retrieval-Augmented Generation (RAG) model** for a Question Answering (QA) bot using LangChain and Google Generative AI. The bot retrieves document embeddings and generates coherent answers based on user queries. The frontend interface is built using **Streamlit**, allowing users to upload PDF documents, query them in real time, and get responses

**Pipeline Overview**

**1. Document Ingestion and Processing**

* **Input**: A PDF document uploaded by the user.
* **Processing**:
  + The PDF is read using the PyPDFLoader from LangChain.
  + The document is split into smaller chunks using TokenTextSplitter to handle large documents and improve the accuracy of the retrieval and answer generation process.
* **Output**: A list of document chunks that are used for embedding generation and question-answering.

**2. Embedding Creation**

* **Embedding Model**: The project uses **GoogleGenerativeAIEmbeddings** from LangChain.
* **Vector Store**: These embeddings are stored in a vector database, **FAISS**, which allows efficient retrieval of relevant document chunks based on the user’s query.

**3. Query Processing and Answer Generation**

* **RAG Model**:
  + The user's query is processed and relevant document chunks are retrieved from the FAISS index.
  + The generative model **ChatGoogleGenerativeAI** (Gemini-1.5 Pro) generates answers based on the retrieved chunks.
* **Refinement**: Questions and answers can be refined using custom PromptTemplate definitions.

**4. Frontend Interface**

* **Streamlit**:
  + The app provides an interface for users to upload PDF documents.
  + Users can enter queries related to the content of the document, and the bot will respond in real-time with the generated answers.
  + Streamlit runs the app locally or in a containerized environment.

### ****File Structure****

project-directory/

│

├── app.py # Main Streamlit app

├── src/

│ ├── helper.py # Helper functions (document processing, FAISS index, retrieval chain)

│ ├── prompt.py # Custom prompt templates for LangChain

├── requirements.txt # Dependencies

├── .env # Environment variables (API keys, etc.)

└── README.md

* Here is the sample image of the output of the frontend of bot:

