Generative AI RAG System Project Documentation

## Project Overview

This project is designed to build a Retrieval-Augmented Generation (RAG) system using Generative AI. The solution involves using LangChain, Faiss for vector storage, and FinBERT/SentenceTransformer for embeddings. The final implementation integrates document retrieval and chunking with a generation model like Flan-T5 or LLAMA for user query responses.

## Project Structure

Library Project  
 ├── Data # Folder containing input documents  
 │ ├── 2022.docx  
 │ ├── 2023.pdf  
 │ └── 2024.docx  
 ├── Embeddings # Folder for all embedding-related scripts  
 │ ├── embeddings\_FinBERT.py  
 │ ├── embeddings\_SentenceTransformer.py  
 │ └── PCA.py  
 ├── Generation # Folder for text generation models  
 │ ├── generation.py  
 │ └── generation\_llama.py  
 ├── Storing\_embedding # Folder for vector storage implementations  
 │ ├── storage\_elasticsearch.py  
 │ └── storage\_faiss.py  
 ├── Retrieval # Folder for document retrieval functions  
 │ └── retrieval.py  
 ├── Scripts # Core scripts for the overall project  
 │ ├── document\_loader.py  
 │ ├── preprocessing\_documents.py  
 │ ├── elastic\_handler.py  
 │ ├── Chunking.py  
 │ └── main.py # Main file to connect all modules  
 └── requirements.txt # List of required libraries for the project

## Detailed Project Steps

|  |  |  |  |
| --- | --- | --- | --- |
| Step | Description | Purpose | Outcome |
| Step 1: Loading Documents | Loads documents from multiple formats (PDF, DOCX) using LangChain. | To gather data for the system. | A structured document list is generated. |
| Step 2: Preprocessing | Cleans, tokenizes, removes stopwords, and standardizes the text. | To ensure the data is noise-free and in a consistent format. | Documents are in a clean, analyzable format. |
| Step 3: Chunking | Splits documents into smaller chunks of 300 tokens using recursive chunking. | To facilitate better embedding and search accuracy. | Documents split into chunks of uniform size. |
| Step 4: Generating Embeddings | Uses FinBERT and SentenceTransformer models to generate embeddings. | To represent text in vector format for effective similarity searches. | High-dimensional embeddings for each document. |
| Step 5: Storing Embeddings | Stores embeddings in Faiss vector storage. | To enable fast vector-based similarity search and retrieval. | Embeddings indexed and stored. |
| Step 6: Document Retrieval | Retrieves the top-k relevant chunks using Faiss and Elasticsearch. | To find the most contextually relevant documents for a query. | Top-k documents retrieved with similarity scores. |
| Step 7: Generation | Generates text responses using Flan-T5 or LLAMA models. | To provide coherent, contextually relevant responses to user queries. | Generated answers based on combined document chunks. |

## Key Components

1. \*\*Document Loader\*\*:  
- Reads documents of various formats (PDF, DOCX) and ensures they are in the correct structure.  
- Provides a unified document structure for downstream processing.

2. \*\*Preprocessing Module\*\*:  
- Preprocesses text using tokenization, stopword removal, and case standardization.  
- Integrates with LangChain's text chunking and splitting functions.

3. \*\*Chunking\*\*:  
- Uses LangChain's `RecursiveCharacterTextSplitter` for better chunk management.  
- Supports chunk overlap for preserving context between chunks.

4. \*\*Embeddings\*\*:  
- \*\*FinBERT\*\*: Financial domain-specific transformer for capturing financial language.  
- \*\*SentenceTransformer\*\*: General-purpose transformer for capturing sentence-level semantics.

5. \*\*Storage in Faiss and Elasticsearch\*\*:  
- Stores high-dimensional embeddings in Faiss for efficient similarity search.  
- Elasticsearch enables text-based and hybrid retrieval.

6. \*\*Document Retrieval\*\*:  
- Retrieves documents based on query embeddings.  
- Uses similarity metrics like cosine distance to rank and select the top-k relevant chunks.

7. \*\*Generation\*\*:  
- Generates natural language responses using the \*\*Flan-T5\*\* or \*\*LLAMA\*\* model.  
- Integrates chunked context for informed and context-rich generation.

## Requirements

torch==2.0.1  
transformers==4.25.1  
langchain==0.0.126  
numpy==1.23.3  
faiss-cpu==1.7.3  
scikit-learn==1.1.3  
elasticsearch==8.15.1  
pdfminer.six==20201018  
nltk==3.7  
unstructured==0.4.4  
pydantic==1.10.5  
requests==2.28.1

## Running the Project

1. \*\*Set up the environment\*\*:

```bash  
pip install -r requirements.txt  
```

2. \*\*Start Elasticsearch\*\* (if required for hybrid search):

```bash  
elasticsearch.bat  
```

3. \*\*Run the main script\*\*:

```bash  
python main.py  
```

## Future Enhancements

1. \*\*Integration with Other Vector DBs\*\*:  
- Explore integrations with Pinecone, Milvus, or Weaviate.

2. \*\*Improved Retrieval Algorithms\*\*:  
- Implement different hybrid search strategies combining vector and text-based search.

3. \*\*Advanced Generation Models\*\*:  
- Experiment with more advanced models like GPT-4, T5-XL, or BERT-large.

4. \*\*Scaling with Kubernetes\*\*:  
- Dockerize the entire system and deploy using Kubernetes for horizontal scaling.

5. \*\*UI Integration\*\*:  
- Create a frontend interface using Streamlit or Flask for user interaction.