## Aim

To develop a Retrieval-Augmented Generation (RAG) system using the GEMMA GROQ model for document-based question answering. The system integrates various AI tools and libraries to process, store, and retrieve relevant document information efficiently.

## Objective

1. Implement a document ingestion and embedding system using Google Generative AI embeddings.
2. Create a retrieval mechanism to find relevant document chunks based on user queries.
3. Develop a user-friendly interface using Streamlit to interact with the system.
4. Integrate the GEMMA GROQ model to generate accurate answers to user queries based on the retrieved document context.

## Summary

This project focuses on building a RAG system using GEMMA GROQ and various Langchain components. The system ingests documents from a specified directory, processes them into vector embeddings, and stores them using FAISS. Users can input questions through a Streamlit interface, and the system retrieves relevant document chunks to generate accurate responses using the GEMMA GROQ model.

## Tools and Libraries Used

- **Streamlit**: For building the user interface.
- **Langchain**: For managing document processing and retrieval.
- **FAISS**: For storing and retrieving vector embeddings.
- **PyPDF2**: For loading PDF documents.
- **Google Generative AI**: For generating embeddings.
- **dotenv**: For loading API keys from the environment.
- **GEMMA GROQ**: For generating responses to user queries.

## Procedure

1. **Setup and Configuration**:
   - Load environment variables and API keys.
   - Initialize the Streamlit interface.
2. **Document Processing**:
   - Load documents from a specified directory using PyPDFDirectoryLoader.
   - Split documents into chunks using RecursiveCharacterTextSplitter.

3. **Embedding and Vector Storage**:
    - Generate embeddings for document chunks using GoogleGenerativeAIEmbeddings.
    - Store embeddings in a FAISS vector store.
4. **User Interaction**:
    - Provide a text input field for users to ask questions.
    - Embed documents into the vector store when prompted.
5. **Query Processing**:
    - Create a document chain using the GEMMA GROQ model and a prompt template.
    - Retrieve relevant document chunks using the FAISS vector store.
    - Generate and display responses based on retrieved documents.

**CODE:**

```
import streamlit as st
import os
from langchain_groq import ChatGroq
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.chains.combine_documents import create_stuff_documents_chain
from langchain_core.prompts import ChatPromptTemplate
from langchain.chains import create_retrieval_chain
from langchain_community.vectorstores import FAISS
from langchain_community.document_loaders import PyPDFDirectoryLoader
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from dotenv import load_dotenv
import os
load_dotenv()

## load the GROQ And OpenAI API KEY
groq_api_key=os.getenv('GROQ_API_KEY')
os.environ["GOOGLE_API_KEY"]=os.getenv("GOOGLE_API_KEY")
```

```
st.title("Gemma Model Document Q&A")


llm=ChatGroq(groq_api_key=groq_api_key,
             model_name="Gemma-7b-it")


prompt=ChatPromptTemplate.from_template(
"""
Answer the questions based on the provided context only.
Please provide the most accurate response based on the question
<context>
{context}
<context>
Questions:{input}

"""
)


def vector_embedding():


    if "vectors" not in st.session_state:


        st.session_state.embeddings=GoogleGenerativeAIEmbeddings(model = "models/embedding-001")
        st.session_state.loader=PyPDFDirectoryLoader("./data") ## Data Ingestion
        st.session_state.docs=st.session_state.loader.load() ## Document Loading
        st.session_state.text_splitter=RecursiveCharacterTextSplitter(chunk_size=1000,chunk_overlap=200)
## Chunk Creation
        st.session_state.final_documents=st.session_state.text_splitter.split_documents(st.session_state.do
cs[:20]) #splitting
        st.session_state.vectors=FAISS.from_documents(st.session_state.final_documents,st.session_state.e
mbeddings) #vector OpenAI embeddings
```

```
prompt1=st.text_input("Enter Your Question From Doduments")


if st.button("Documents Embedding"):
    vector_embedding()
    st.write("Vector Store DB Is Ready")


import time



if prompt1:
    document_chain=create_stuff_documents_chain(llm,prompt)
    retriever=st.session_state.vectors.as_retriever()
    retrieval_chain=create_retrieval_chain(retriever,document_chain)
    start=time.process_time()
    response=retrieval_chain.invoke({'input':prompt1})
    print("Response time :",time.process_time()-start)
    st.write(response['answer'])

    # With a streamlit expander
    with st.expander("Document Similarity Search"):
        # Find the relevant chunks
        for i, doc in enumerate(response["context"]):
            st.write(doc.page_content)
            st.write("--------------------------------")
```

**Highlights**

- **Integration of GEMMA GROQ Model**: The system utilizes the GEMMA GROQ model to generate precise answers based on the provided document context.
- **Vector Embeddings with Google Generative AI**: Embeddings are generated using state-of-the-art models, ensuring high-quality vector representations.
- **Efficient Document Retrieval**: FAISS is used to efficiently store and retrieve vector embeddings, facilitating quick access to relevant document chunks.
- **User-Friendly Interface**: Streamlit provides an intuitive interface for users to interact with the system, ask questions, and view responses.

**Conclusion**

The RAG system effectively combines advanced AI models and tools to provide accurate document-based question answering. By leveraging the power of GEMMA GROQ, Google Generative AI, and Langchain, the project demonstrates a robust solution for information retrieval and generation. The use of Streamlit ensures a seamless user experience, making the system accessible and easy to use.