# Project Documentation and User Guide

## Overview

This project is a Streamlit-based web application that integrates with LangChain and Google Generative AI to provide an interactive chatbot capable of answering academic-related questions. The application processes user queries, retrieves relevant documents, and generates responses based on the conversation history.

## Project Components

1. **Dependencies:**
   - langchain_community: For document loaders, text splitting, and vector stores.
   - sentence_transformers: For embedding texts.
   - streamlit: For the web interface.
   - langchain_google_genai: For integrating Google Generative AI.
2. **Main Script:**
   - Loads and processes a PDF document.
   - Embeds document texts and stores them in a vector store.
   - Initializes a chat history with Streamlit.
   - Sets up a prompt template and response chain using Google Generative AI.
   - Handles user inputs and displays responses in the Streamlit app.

## Setup Instructions

1. **Environment Setup:**
   - Install the required libraries:

     ```
     pip install langchain_community sentence_transformers streamlit langchain_google_genai python-dotenv
     ```

   - Create a .env file in your project directory with the following content:

     ```
     GOOGLE_API_KEY=your_google_api_key
     LANGCHAIN_API_KEY=your_langchain_api_key
     ```

2. **Running the Application:**
   - Place your PDF file (e.g., SQL tutorials.pdf) in the specified path.
   - Run the Streamlit app:

     ```
     streamlit run app.py
     ```

## File Descriptions

### 1. app.py

#### Imports and Environment Setup

```
from langchain_community.document_loaders import PyPDFLoader
```

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
from sentence_transformers import SentenceTransformer
from langchain_community.vectorstores import Chroma
from langchain.docstore.document import Document
from typing import List
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.output_parsers import StrOutputParser
from langchain_community.chat_message_histories import StreamlitChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory
import streamlit as st
import os
from dotenv import load_dotenv
from langchain_google_genai import ChatGoogleGenerativeAI

load_dotenv()

google_api_key = os.getenv("GOOGLE_API_KEY")
langchain_api_key = os.getenv("LANGCHAIN_API_KEY")

if not google_api_key or not langchain_api_key:
    st.error("API keys are missing. Please check your .env file.")

os.environ["GOOGLE_API_KEY"] = google_api_key
os.environ["LANGCHAIN_TRACING_V2"] = "true"
os.environ["LANGCHAIN_API_KEY"] = langchain_api_key
```

## Document Loading and Embedding

```
loader = PyPDFLoader('/path/to/your/SQL tutorials.pdf')
docs = loader.load()

text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
docs = text_splitter.split_documents(docs)

hf_model = SentenceTransformer("all-MiniLM-L6-v2")

class HuggingFaceEmbeddings:
    def __init__(self, model):
        self.model = model

    def embed_documents(self, texts: List[str]) -> List[List[float]]:
        embeddings = self.model.encode(texts)
        return embeddings.tolist()

    def embed_query(self, text: str) -> List[float]:
        return self.model.encode([text])[0].tolist()

hf_embedder = HuggingFaceEmbeddings(hf_model)

docs = [Document(page_content=doc) if isinstance(doc, str) else doc for doc in docs]

texts = [doc.page_content for doc in docs]

db = Chroma.from_texts(texts=texts, embedding=hf_embedder)
```

## Chat Interface and Response Handling

```
msgs = StreamlitChatMessageHistory(key="langchain_messages")
if len(msgs.messages) == 0:
    msgs.add_ai_message("How can I help you?")

prompt = ChatPromptTemplate.from_messages(
    [
        ("system", "You're an assistant knowledgeable about academic topics. Only answer academic-related questions. Provide answers based on previous chat."),
        MessagesPlaceholder(variable_name="history"),
        ("user", "Question: {question}")
    ]
)

llm = ChatGoogleGenerativeAI(model="gemini-1.5-pro")
output_parser = StrOutputParser()

chain = prompt | llm | output_parser
chain_with_history = RunnableWithMessageHistory(
    chain,
    lambda session_id: msgs,
    input_messages_key="question",
    history_messages_key="history",
)

for msg in msgs.messages:
    st.chat_message(msg.type).write(msg.content)

if prompt_text := st.chat_input():
    st.chat_message("user").write(prompt_text)

    query_embedding = hf_embedder.embed_query(prompt_text)
    relevant_docs = db.similarity_search(query_embedding, k=3)  # Adjust k as needed

    config = {"configurable": {"session_id": "any"}}
    response = chain_with_history.invoke({"question": prompt_text, "relevant_docs": relevant_docs}, config)

    st.chat_message("ai").write(response)

view_messages = st.expander("View the message contents in session state")
with view_messages:
    view_messages.json(st.session_state.langchain_messages)
```

## 2. no_rag_app.py

### Imports and Environment Setup

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_core.output_parsers import StrOutputParser
from langchain_community.chat_message_histories import StreamlitChatMessageHistory
from langchain_core.runnables.history import RunnableWithMessageHistory

import streamlit as st
import os
from dotenv import load_dotenv

from langchain_google_genai import ChatGoogleGenerativeAI
```

```
load_dotenv()

msgs = StreamlitChatMessageHistory(key="langchain_messages")
if len(msgs.messages) == 0:
    msgs.add_ai_message("How can I help you?")

view_messages = st.expander("View the message contents in session state")

# Access environment variables
google_api_key = os.getenv("GOOGLE_API_KEY")
langchain_api_key = os.getenv("LANGCHAIN_API_KEY")

if not google_api_key or not langchain_api_key:
    st.error("API keys are missing. Please check your .env file.")

os.environ["GOOGLE_API_KEY"]=google_api_key
os.environ["LANGCHAIN_TRACING_V2"]="true"
os.environ["LANGCHAIN_API_KEY"]=langchain_api_key

# Prompt Template
prompt = ChatPromptTemplate.from_messages(
    [
        ("system","You're an assistant knowledgeable about academic topics. Only answer academic-related questions. Provide answers
based on previous chat."),
        MessagesPlaceholder(variable_name="history"),
        ("user", "Question:{question}")
    ]
)

llm = ChatGoogleGenerativeAI(model="gemini-1.5-pro")
output_parser = StrOutputParser()
chain = prompt | llm | output_parser

chain_with_history = RunnableWithMessageHistory(
    chain,
    lambda session_id: msgs,
    input_messages_key="question",
    history_messages_key="history",
)

for msg in msgs.messages:
    st.chat_message(msg.type).write(msg.content)

if prompt := st.chat_input():
    st.chat_message("user").write(prompt)
    config = {"configurable": {"session_id": "any"}}
    response = chain_with_history.invoke({"question": prompt}, config)
    st.chat_message("ai").write(response)

with view_messages:
    view_messages.json(st.session_state.langchain_messages)
```

# User Guide

1. **Starting the Application:**
    - Make sure your environment is properly set up with all dependencies installed.
    - Place the PDF file in the specified path.

    - Run the Streamlit app with `streamlit run app.py`.
2. **Using the Chatbot:**
    - Open the Streamlit application in your web browser.
    - Type your academic-related questions in the input field.
    - The chatbot will process your question, retrieve relevant information from the PDF, and provide an answer.
3. **Viewing Chat History:**
    - Expand the "View the message contents in session state" section to see the chat history and message details.

# Notes

- Adjust the document path in the `app.py` script as needed.
- Ensure your API keys are correctly set in the `.env` file to enable Google Generative AI integration.
- Modify the `k` parameter in the `similarity_search` method to change the number of relevant documents retrieved.