

# GOOGLE CLOUD GENERATIVE AI

PRESENTED BY

FACULTY MENTOR

MR. GANESH

TEAM ID : LTVIP2025TMID27925

TEAM MEMBERS : 1

21E91A0301 : GOLLAMUDI YEHOSHUVA

## **DocuQuery: AI-Powered PDF Knowledge Assistant Using Google PALM**

### **Scenario 1: Price List Analyzer**

Companies often deal with multiple price lists from various suppliers or vendors. With our tool, users can upload multiple price lists, and the tool will extract prices of items listed in each document. Additionally, users can obtain detailed information about the items present in the price lists, including descriptions, quantities, and any other relevant information. This feature streamlines the process of comparing prices across different suppliers and facilitates informed decision-making in procurement.

### **Scenario 2: Research Paper Simplifier**

Researchers often encounter lengthy and complex research papers in their field of study. Our tool simplifies the process of digesting and understanding these papers by providing concise summaries. Researchers can upload research papers, and the tool will generate summarized versions that capture the key insights and findings. Furthermore, users can ask questions related to the content of the research papers, and the tool will provide accurate answers, facilitating deeper understanding and analysis of the research material.

### Scenario 3: Resume Matcher for Hiring

Companies receive numerous resumes from job applicants for various positions. Our tool streamlines the hiring process by automating the screening of resumes and matching candidates with the required skillset and qualifications. Companies can upload multiple resumes, and the tool will analyze the content to identify candidates who meet the specified criteria. This feature accelerates the hiring process, allowing companies to identify suitable candidates more efficiently and make informed hiring decisions.

## Project Flow

- Users interact with the web application UI to upload PDF documents and ask questions related to the content.
- The uploaded PDF documents are processed by the backend, where the text is extracted from each document using PyPDF2.
- The extracted text is split into smaller chunks to optimize processing and analysis.
- The text chunks are passed to the Google Palm Embeddings module to generate embeddings for each chunk.
- The embeddings are stored in a FAISS vector store for efficient retrieval during question-answering.
- The user's question is passed to the Conversational Retrieval Chain, which uses the pre-trained Google Palm model to generate responses.
- The response from the model is displayed to the user via the web application UI, providing answers to their questions based on the content of the uploaded PDF documents.

To accomplish this, we have to complete all the activities listed below,

- Setting Up Google API Key
  - Generate PALM API key
- Installation and Importing of Libraries and Adding API Key
  - Install and Import necessary libraries for the project
  - Add the Google API Key to the code
- PDF Text Processing
  - Extract Text from PDF Documents
  - Split Text into Chunks
- Conversational Chain Setup
  - Setting Up Conversational Retrieval Chain
- Application Setup and Integration
  - Configure Streamlit UI
  - Sidebar Settings
  - Document Processing
  - Run the Application

## Project Structure

Create the Project folder which contains files as shown below

- We are building a Streamlit application in which we will use Python script app.py for scripting.
- The requirements.txt file contains all the libraries required for the project.
- Venv folder is the environment created on which the project is developed.

## Setting Up Google API Key

For initializing the model we need to generate PALM API.

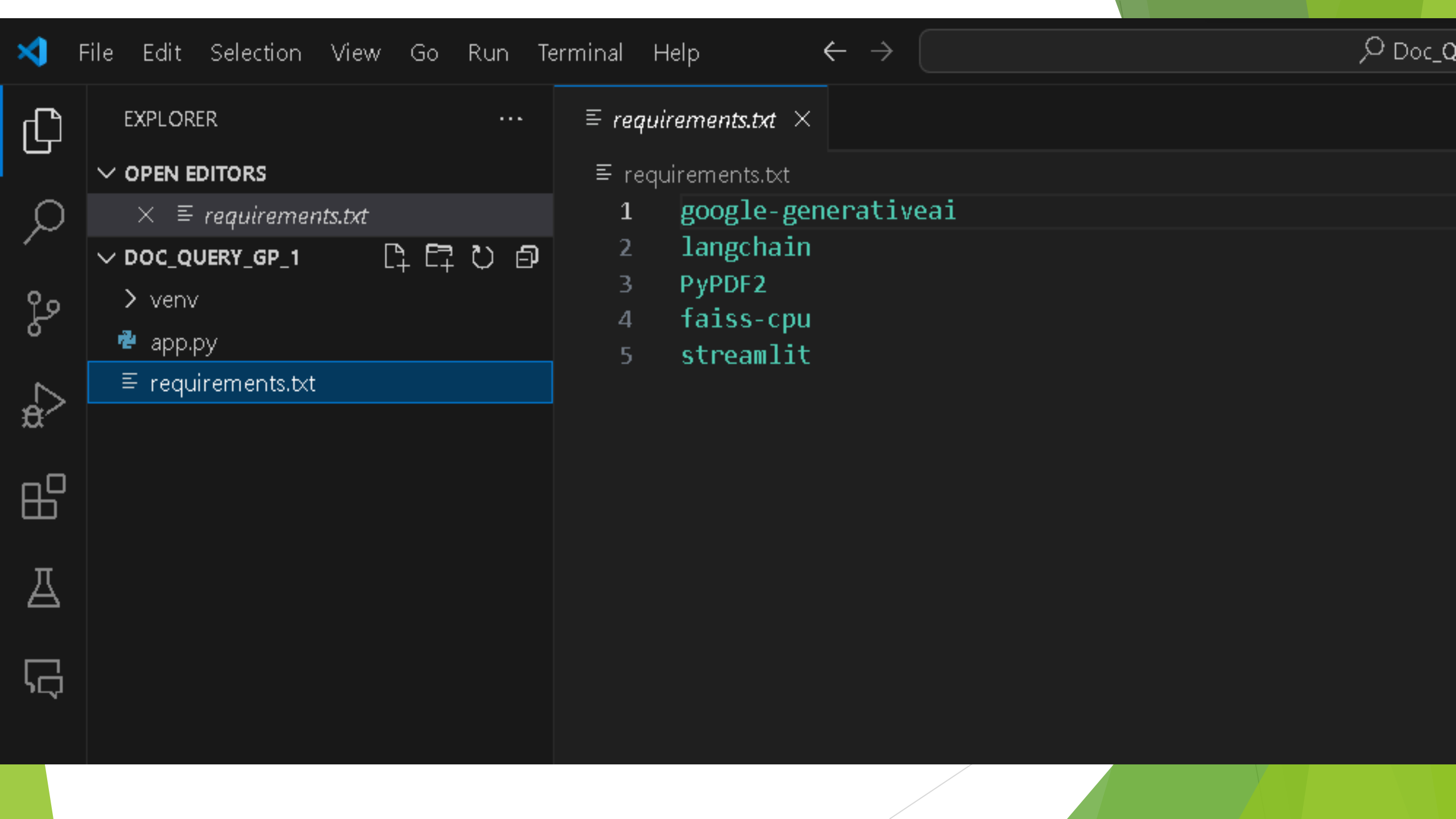
## Installation and Importing of Libraries and Adding API Key

In this milestone, we will import necessary libraries and add the API Key generated from the above milestone.

### Install and Import necessary libraries for the project

Before installing activate the environment for developing the project.

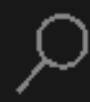
- Firstly install the libraries present in the requirement.txt file  
Streamlit (streamlit): Web application framework for building interactive web apps.
  - PyPDF2 (PyPDF2): Library for reading and extracting text from PDF files.
  - LangChain Text Splitter (langchain.text\_splitter): Module for splitting text into smaller chunks.
  - Google Generative AI (google.generativeai): Google's API for accessing generative AI models.
  - LangChain Embeddings (langchain.embeddings): Module for generating embeddings from text data.
  - LangChain LLMs (langchain.llms): Module for accessing pre-trained language models.
  - LangChain Vectorstores (langchain.vectorstores): Module for storing and retrieving vectors.
  - LangChain Chains (langchain.chains): Module for building conversational retrieval chains.
  - LangChain Memory (langchain.memory): Module for managing conversation history.
- The extracted text for each page is appended to the list `text_`. Additionally, it prints the text of each page along with the page number.



EXPLORER



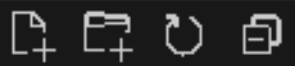
OPEN EDITORS



requirements.txt



DOC\_QUERY\_GP\_1



venv

app.py

requirements.txt



requirements.txt

requirements.txt

```
1 google-generativeai
2 langchain
3 PyPDF2
4 faiss-cpu
5 streamlit
```



app.py X

app.py > ...

```
1 import streamlit as st
2 from PyPDF2 import PdfReader
3 from langchain.text_splitter import RecursiveCharacterTextSplitter
4 import google.generativeai as palm
5 from langchain.embeddings import GooglePalmEmbeddings
6 from langchain.llms import GooglePalm
7 from langchain.vectorstores import FAISS
8 from langchain.chains import ConversationalRetrievalChain
9 from langchain.memory import ConversationBufferMemory
10 import os
11
```

## Add the Google API Key to the code

```
11  
12 os.environ['GOOGLE_API_KEY'] = 'Your_Api_Key_Here'  
13
```

.Set the environment variable GOOGLE\_API\_KEY to the obtained API key value.  
Example: `os.environ['GOOGLE_API_KEY'] = 'YOUR_API_KEY_HERE'`


## PDF Text Processing

## Extract Text from PDF Documents

```
14 def get_pdf_text(pdf_docs):  
15     text=""  
16     for pdf in pdf_docs:  
17         pdf_reader= PdfReader(pdf)  
18         for page in pdf_reader.pages:  
19             text+= page.extract_text()  
20     return text  
21
```

## Split Text into Chunks

```
21  
22 def get_text_chunks(text):  
23     text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=20)  
24     chunks = text_splitter.split_text(text)  
25     return chunks
```

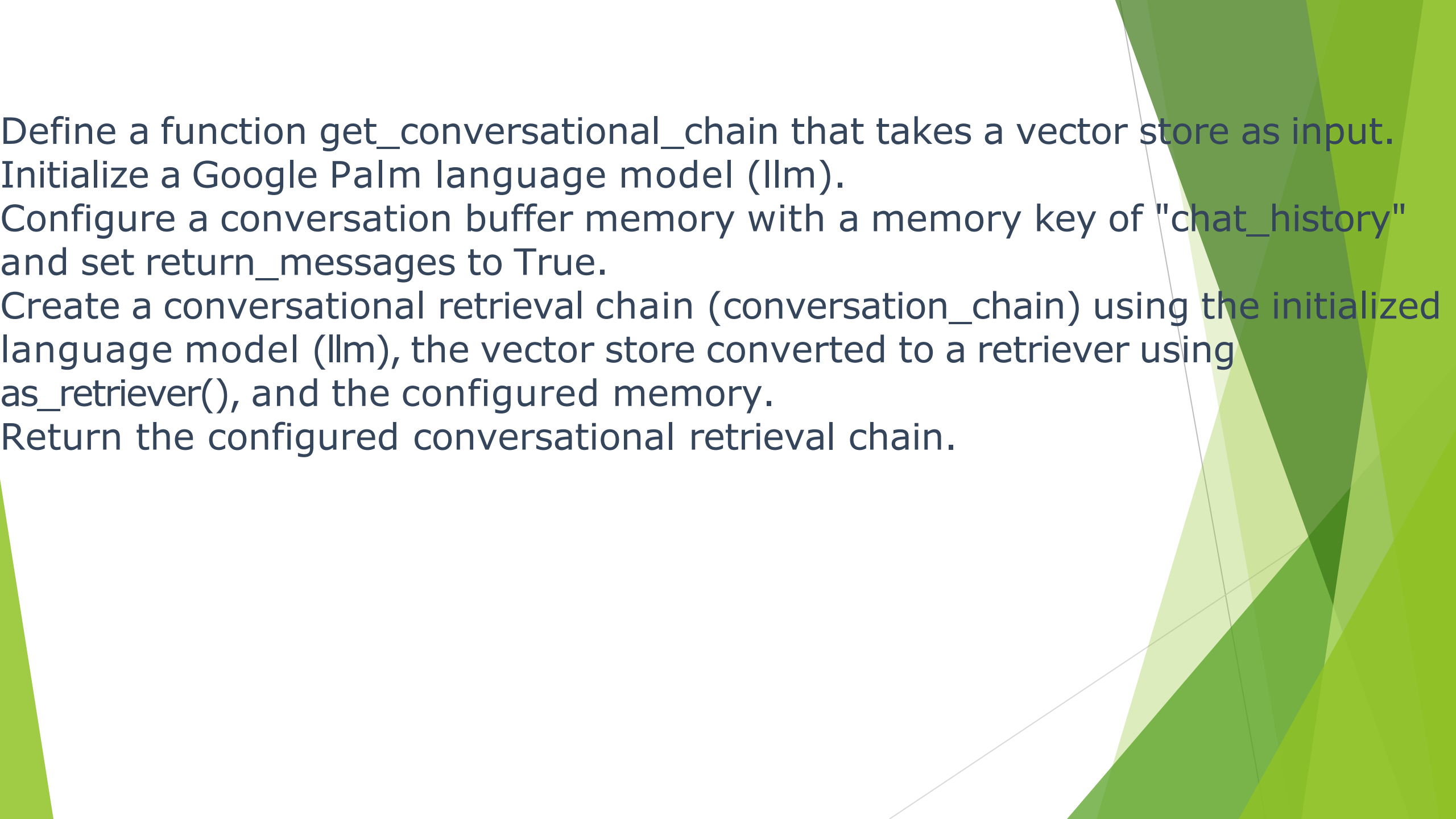


Define a function `get_pdf_text` that takes a list of PDF documents as input.  
Iterate over each PDF document in the list.  
Use `PdfReader` from `PyPDF2` library to read the PDF document.  
Iterate over each page in the PDF document.  
Extract text from each page and concatenate it to the text variable.  
Return the concatenated text as the output.

## Conversational Chain Setup

### Setting Up Conversational Retrieval Chain

```
31
32 def get_conversational_chain(vector_store):
33     llm=GooglePalm()
34     memory = ConversationBufferMemory(memory_key = "chat_history", return_messages=True)
35     conversation_chain = ConversationalRetrievalChain.from_llm(llm=llm, retriever=vector_store.as_retriever(), memory=memory)
36     return conversation_chain
37
```

The background of the slide features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.

Define a function `get_conversational_chain` that takes a vector store as input.  
Initialize a Google Palm language model (llm).  
Configure a conversation buffer memory with a memory key of "chat\_history" and set `return_messages` to True.  
Create a conversational retrieval chain (`conversation_chain`) using the initialized language model (llm), the vector store converted to a retriever using `as_retriever()`, and the configured memory.  
Return the configured conversational retrieval chain.

```
37
38 def user_input(user_question):
39     response = st.session_state.conversation({'question': user_question})
40     st.session_state.chatHistory = response['chat_history']
41     for i, message in enumerate(st.session_state.chatHistory):
42         if i%2 == 0:
43             st.write("Human: ", message.content)
44         else:
45             st.write("Bot: ", message.content)
```



**• Handling User Input and Generating Responses** Define a function `user_input` that takes a user question as input.

Call the `conversation` method of `st.session_state` with the user question as a parameter to initiate a conversation.

Store the conversation history returned by the `conversation` method in the `chatHistory` session state variable.

Iterate over the messages in the conversation history:

- If the index (`i`) is even (indicating a message from the user), display "Human: " followed by the content of the message using `st.write`.
- If the index (`i`) is odd (indicating a message from the bot), display "Bot: " followed by the content of the message using `st.write`.

```
37
38 def user_input(user_question):
39     response = st.session_state.conversation({'question': user_question})
40     st.session_state.chatHistory = response['chat_history']
41     for i, message in enumerate(st.session_state.chatHistory):
42         if i%2 == 0:
43             st.write("Human: ", message.content)
44         else:
45             st.write("Bot: ", message.content)
```

## • **Configure Streamlit UI**

Define a function main to set up the Streamlit UI for the DocuQuery application.

Set the page configuration to display the title "DocuQuery: AI-Powered PDF Knowledge Assistant".

Add a header to the UI displaying the application title.

Include a text input field for users to ask questions from the PDF files.

Check if the "conversation" and "chatHistory" session state variables exist; if not, initialize them to None.

If a user question is provided, call the user\_input function to handle the question.

```
45     st.write(st.session_state.messages)
46 def main():
47     st.set_page_config("DocuQuery: AI-Powered PDF Knowledge Assistant")
48     st.header("DocuQuery: AI-Powered PDF Knowledge Assistant")
49     user_question = st.text_input("Ask a Question from the PDF Files")
50     if "conversation" not in st.session_state:
51         st.session_state.conversation = None
52     if "chatHistory" not in st.session_state:
53         st.session_state.chatHistory = None
54     if user_question:
55         user_input(user_question)
```

## Sidebar Settings

- Within the sidebar, add a title "Settings".
- Include a subheader "Upload your Documents" to guide users.
- Provide a file uploader component for users to upload PDF files.
- Add a button labeled "Process" to initiate document processing.

```
55     user_input(user_question)
56     with st.sidebar:
57         st.title("Settings")
58         st.subheader("Upload your Documents")
59         pdf_docs = st.file_uploader("Upload your PDF Files and Click on the Process Button", accept_multiple_files=True)
60         if st.button("Process"):
```

# Document Processing

```
60     if st.button("Process"):
61         with st.spinner("Processing"):
62             raw_text = get_pdf_text(pdf_docs)
63             text_chunks = get_text_chunks(raw_text)
64             vector_store = get_vector_store(text_chunks)
65             st.session_state.conversation = get_conversational_chain(vector_store)
66             st.success("Done")
67
68
```

- Upon clicking the "Process" button, trigger the processing of uploaded PDF files.
- Display a spinner indicating that processing is in progress.
- Extract text from the uploaded PDF files using the `get_pdf_text` function.
- Split the extracted text into smaller chunks using the `get_text_chunks` function.
- Generate embeddings for the text chunks and create a vector store using the `get_vector_store` function.
- Initialize and configure the conversational retrieval chain using the vector store with the `get_conversational_chain` function.
- Upon completion of processing, display a success message to indicate that processing is done.

## Run the Application

```
68  
69  
70  if __name__ == "__main__":  
71      |    main()  
72
```

Use the `__name__` variable to ensure that the main function is executed when the script is run as the main program.

Run the DocuQuery application by calling the main function.

The output of the DocuQuery project, considering all three scenarios, would involve an interactive web application interface where users can perform various tasks related to PDF documents. Here's how the output would look like for each scenario:

Activate the environment by copying the relative path of activate inside venv folder.

`Recording_Chat_Multi_PDF__PALM2` is the environment name

Then run the application using `"streamlit run app.py"`



```
D:\Streamlit practice\LargeLanguageModelsProjects-main\Doc_Query_GP_1>venv\Scripts\activate
```

```
(Recording_Chat_Multi_PDF_PaLM2) D:\Streamlit practice\LargeLanguageModelsProjects-main\Doc_Query_GP_1>
```

```
(Recording_Chat_Multi_PDF_PaLM2) D:\Streamlit practice\LargeLanguageModelsProjects-main\Doc_Query_GP_1>streamlit run app.py
```

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8501>

Network URL: <http://192.168.0.200:8501>

## Scenario 1: Resume Matcher for Hiring

Companies can upload multiple resumes from job applicants. The tool analyzes the content of the resumes to identify candidates with the required skillset and qualifications specified by the company.

The output includes a list of candidates who meet the specified criteria, along with their relevant details extracted from the resumes.

Companies can review the matched candidates and make informed hiring decisions more efficiently.

## Scenario 2: Price List Analyzer

Users can upload multiple price lists from different suppliers or vendors.

Users can obtain detailed information about the items present in the price lists, including descriptions, quantities, and other relevant information.

The output may include tables or lists displaying the extracted prices and item details for each price list uploaded.

## Scenario 3: Research Paper Simplifier

Users can upload research papers or scholarly articles.

The tool generates concise summaries of the uploaded research papers, capturing the key insights and findings.

Users can ask questions related to the content of the research papers, and the tool provides accurate answers based on the summarized content.

The output may include summarized versions of the research papers, along with the questions asked and the corresponding answers provided by the tool.