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

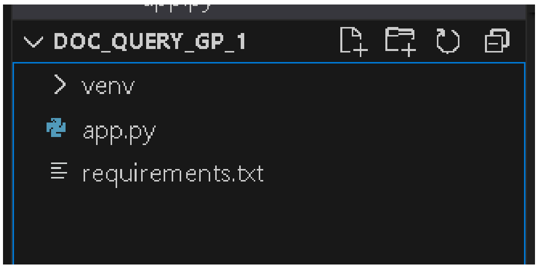
Prior Knowledge

You must have prior knowledge of the following topics to complete this project.

- NLP: https://www.tutorialspoint.com/natural_language_processing/index.htm
- RAG: https://www.datacamp.com/blog/what-is-retrieval-augmented-generation-rag
- LLM & PALM: https://cloud.google.com/vertex-ai/docs/generative-ai/learn-resources
- Streamlit: Create interactive web applications. https://docs.streamlit.io/
- Google Generative AI: https://aistudio.google.com/
- Embeddings and Vector Stores Facebook AI Similarity Search (FAISS): https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/
- PDF Processing: Extract text from PDF documents. https://pypi.org/project/PyPDF2/
- Langchain: https://python.langchain.com/v0.1/docs/get_started/introduction/

Project Structure

Create the Project folder which contains files as shown below



DOC_QUERY_GP_1

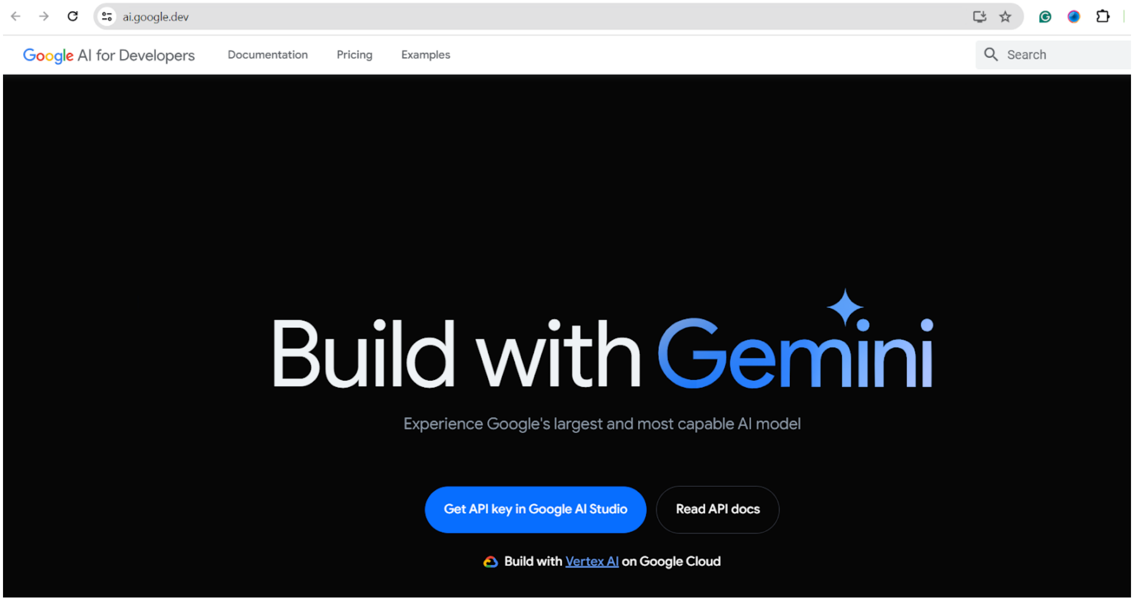
- venv
- app.py
- requirements.txt

- 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.

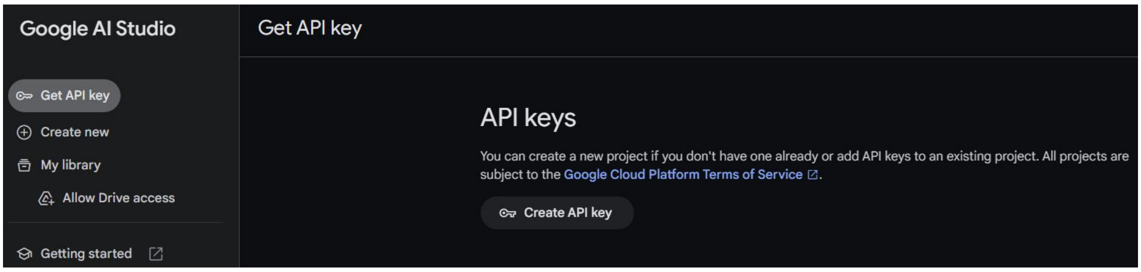
Generate PALM API

- Click on the link (https://developers.generativeai.google/).
- Then click on “Get API key in Google AI Studio”.
- Click on “Get API key” from the right navigation menu.
- Now click on “Create API key”. (Refer the below images)
- Copy the API key.



Build with Gemini

Experience Google's largest and most capable AI model



API keys

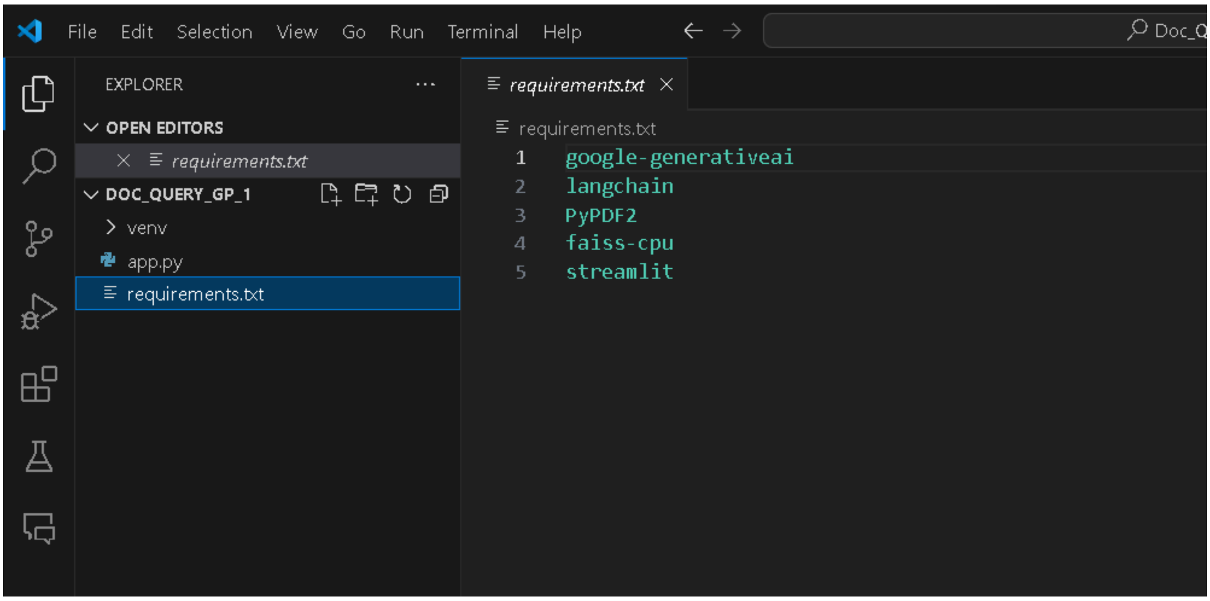
You can create a new project if you don't have one already or add API keys to an existing project. All projects are subject to the Google Cloud Platform Terms of Service.

Installation and Importing of Libraries and Adding API Key

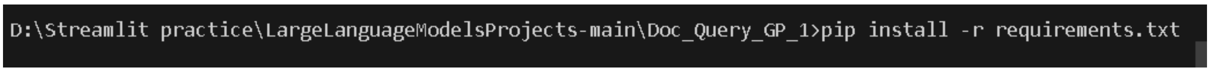
In this milestone, we will import necessary libraries and add the API Key generated from the above milestone.

Install and Import necessary libraries for the project

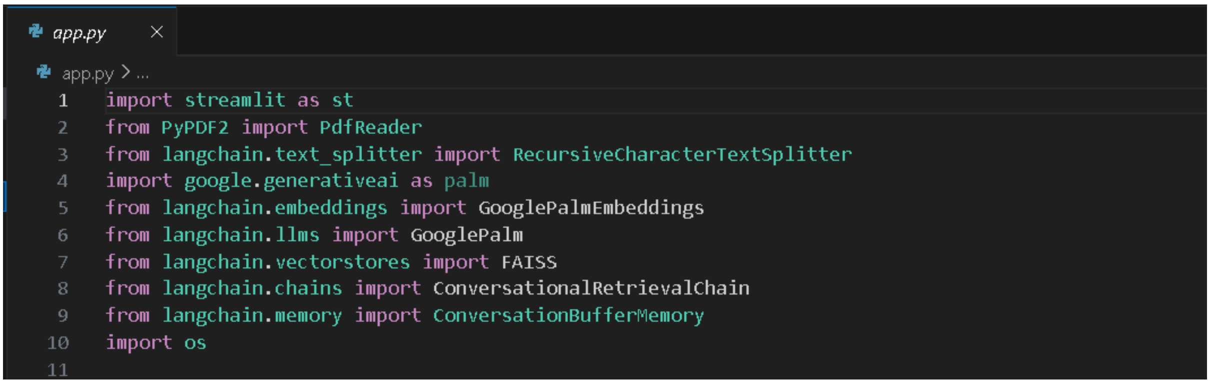
- Before installing activate the environment for developing the project.
- Firstly install the libraries present in the requirement.txt file



```
google-generativeai
langchain
PyPDF2
faiss-cpu
streamlit
```



```
D:\Streamlit practice\LargeLanguageModelsProjects-main\Doc_Query_GP_1>pip install -r requirements.txt
```

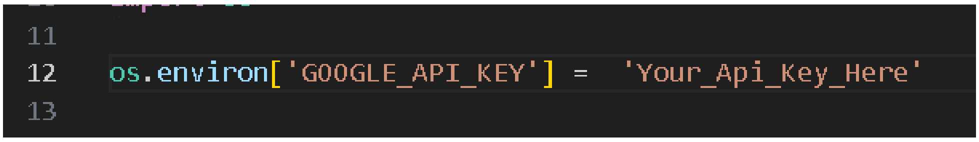


```python
import streamlit as st
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
import google.generativeai as palm
from langchain.embeddings import GooglePalmEmbeddings
from langchain.llms import GooglePalm
from langchain.vectorstores import FAISS
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
import os

```

- 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.method. 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.

Add the Google API Key to the code

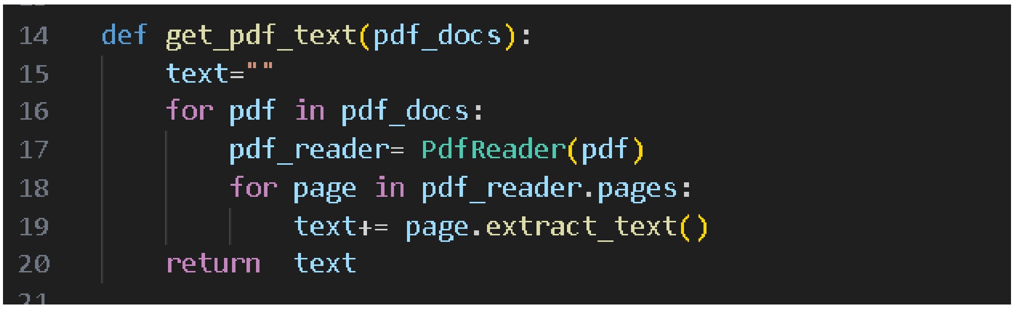


```python

os.environ['GOOGLE_API_KEY'] =  'Your_Api_Key_Here'

```

- 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

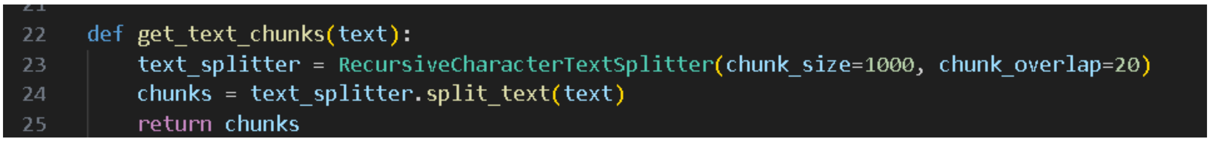


```python
def get_pdf_text(pdf_docs):
    text=""
    for pdf in pdf_docs:
        pdf_reader= PdfReader(pdf)
        for page in pdf_reader.pages:
            text+= page.extract_text()
    return  text
```

- 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.

Split Text into Chunks

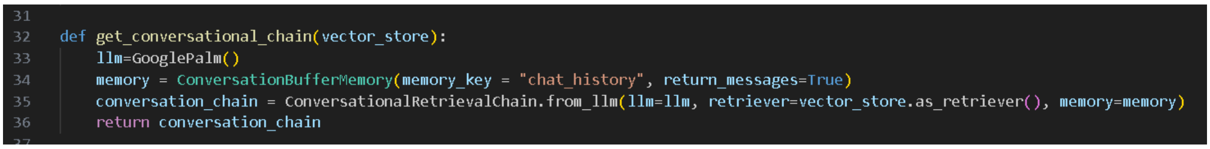


```python
def get_text_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=20)
    chunks = text_splitter.split_text(text)
    return chunks
```

- Define a function get_text_chunks that takes the extracted text as input.
- Initialize a RecursiveCharacterTextSplitter object with chunk size of 1000 characters and overlap of 20 characters.
- Use the splitter to split the text into smaller chunks.
- Return the list of text chunks as the output.

Conversational Chain Setup

Setting Up Conversational Retrieval Chain

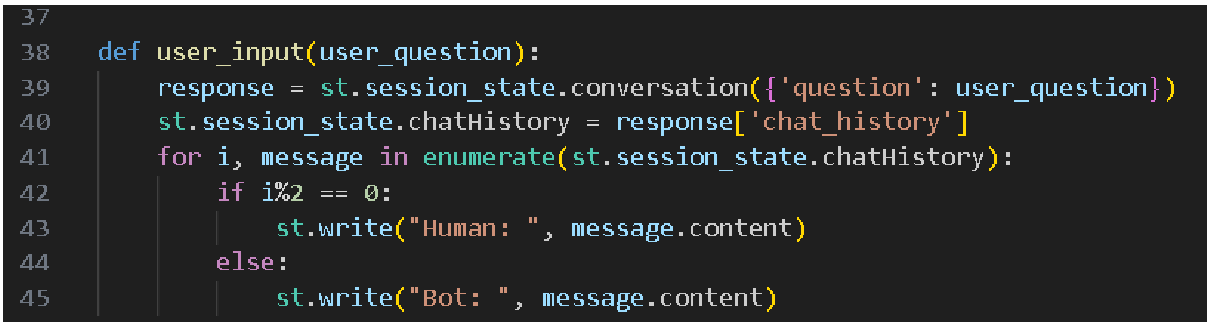


```python

def get_conversational_chain(vector_store):
    llm=GooglePalm()
    memory = ConversationBufferMemory(memory_key = "chat_history", return_messages=True)
    conversation_chain = ConversationalRetrievalChain.from_llm(llm=llm, retriever=vector_store.as_retriever(), memory=memory)
    return conversation_chain
```

- 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.

User Interaction and Response Handling

Handling User Input and Generating Responses

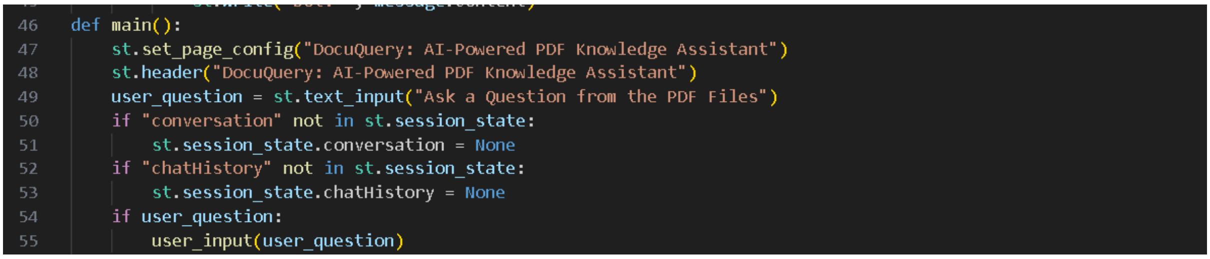


```python

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

- 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.

Application Setup and Integration

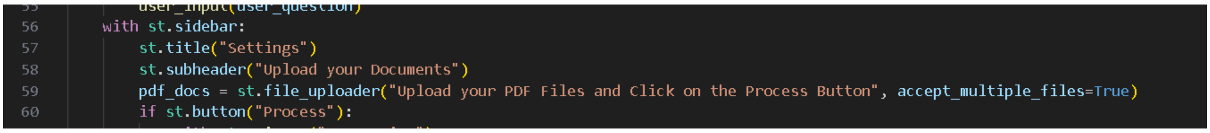
Configure Streamlit UI



```python
def main():
    st.set_page_config("DocuQuery: AI-Powered PDF Knowledge Assistant")
    st.header("DocuQuery: AI-Powered PDF Knowledge Assistant")
    user_question = st.text_input("Ask a Question from the PDF Files")
    if "conversation" not in st.session_state:
        st.session_state.conversation = None
    if "chatHistory" not in st.session_state:
        st.session_state.chatHistory = None
    if user_question:
        user_input(user_question)
```

- 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.

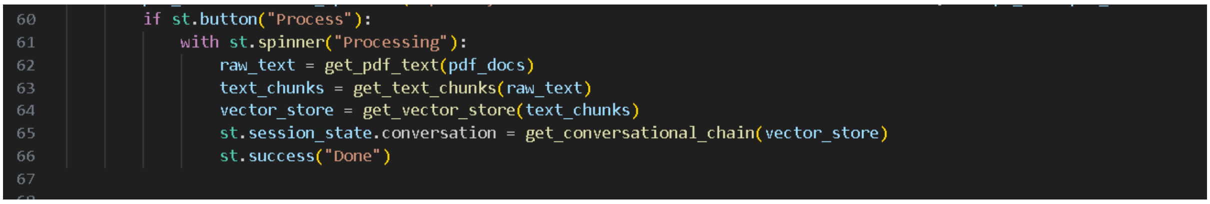
Sidebar Settings



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

- 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.

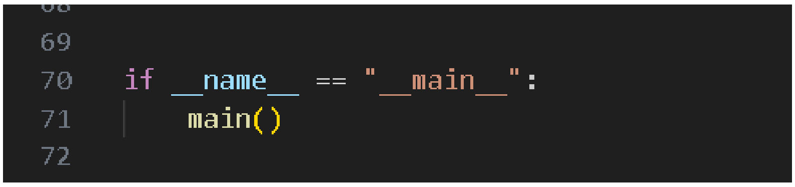
Document Processing



```
        if st.button("Process"):
            with st.spinner("Processing"):
                raw_text = get_pdf_text(pdf_docs)
                text_chunks = get_text_chunks(raw_text)
                vector_store = get_vector_store(text_chunks)
                st.session_state.conversation = get_conversational_chain(vector_store)
                st.success("Done")
```

- 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

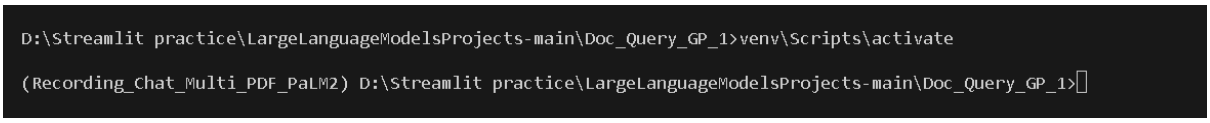


```
if __name__ == "__main__":
    main()
```

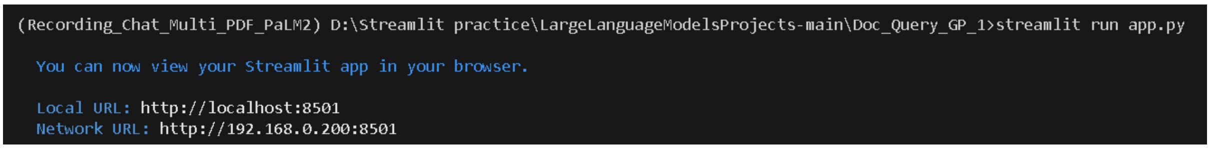
- Use the __name__ variable to ensure that the main function is executed when the script is run as the main program.
- Run the DocuQuery application by calling the main function.

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”



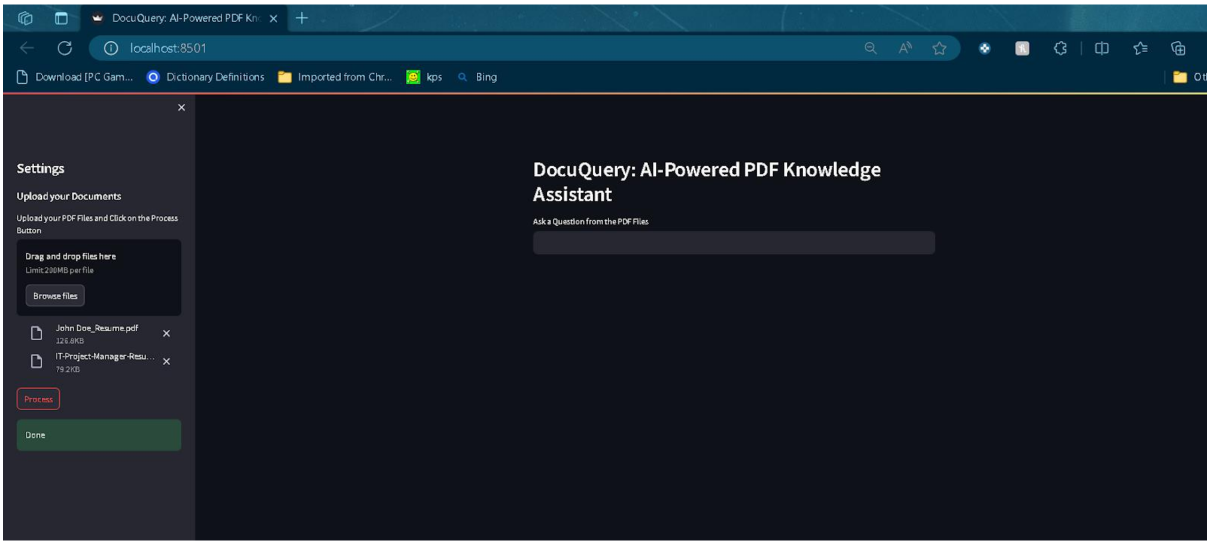
```
D:\Streamlit practice\LargeLanguageModelsProjects-main\Doc_Query_GP_1>venv\Scripts\activate

(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.



Settings

Upload your Documents

Upload your PDF Files and Click on the Process Button

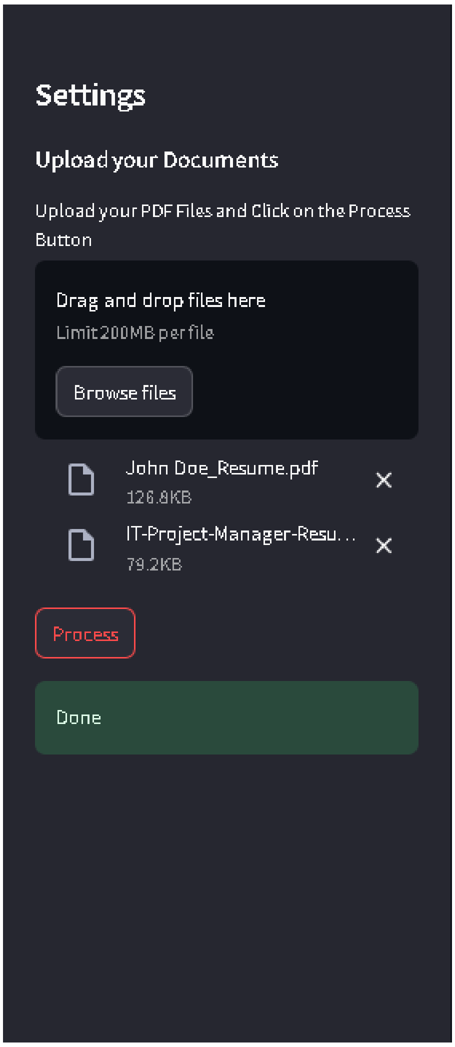
Drag and drop files here
Limit 200MB per file

John Doe_Resume.pdf 126.8KB

IT-Project-Manager-Resu... 79.2KB

DocuQuery: AI-Powered PDF Knowledge Assistant

Ask a Question from the PDF Files



Settings

Upload your Documents

Upload your PDF Files and Click on the Process Button

Drag and drop files here

Limit 200MB per file

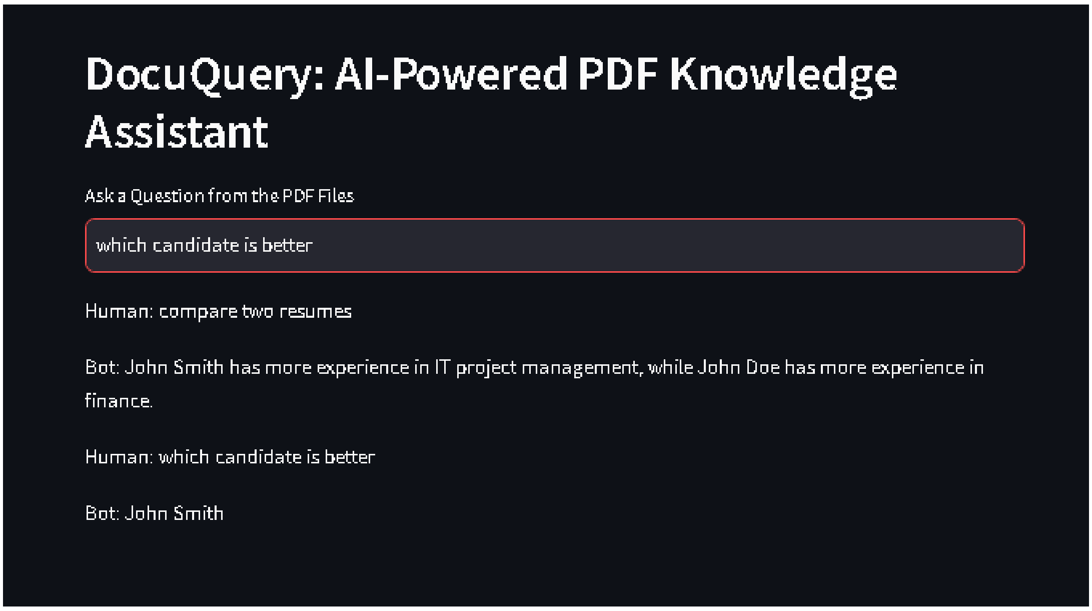
John Doe_Resume.pdf

126.8KB

IT-Project-Manager-Resu...

79.2KB

Done



DocuQuery: AI-Powered PDF Knowledge Assistant

Ask a Question from the PDF Files

which candidate is better

Human: compare two resumes

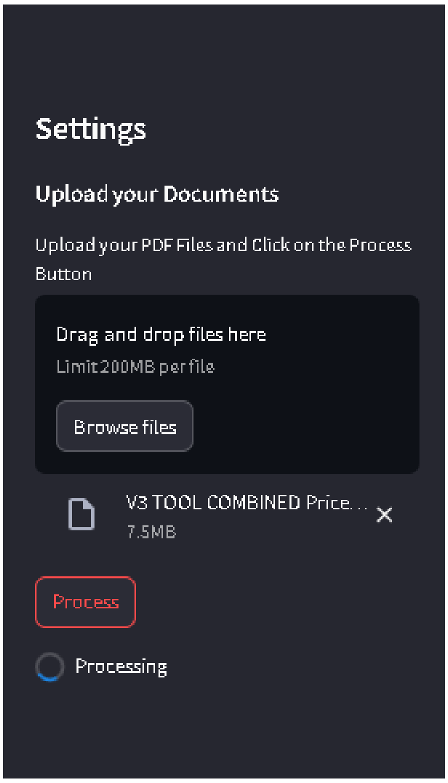
Bot: John Smith has more experience in IT project management, while John Doe has more experience in finance.

Human: which candidate is better

Bot: John Smith

Scenario 2: Price List Analyzer

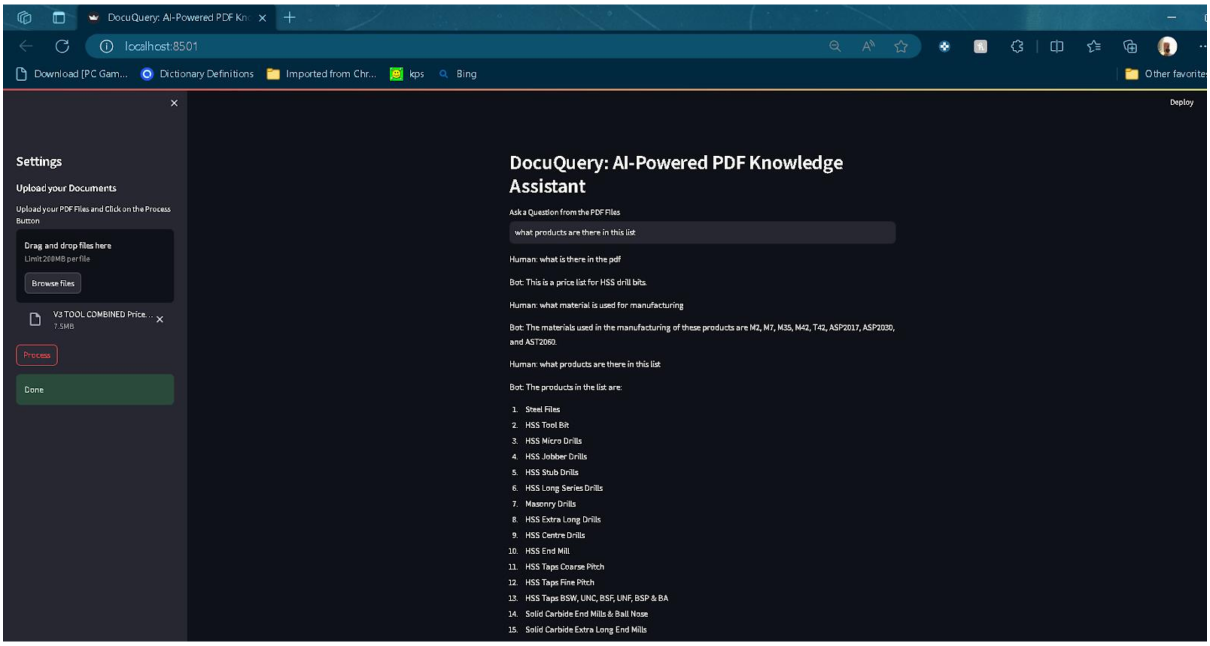
- Users can upload multiple price lists from different suppliers or vendors.
- Users can obtain detailed information about the items present in the price lists, including descriptions, quantities, and other relevant information.
- The output may include tables or lists displaying the extracted prices and item details for each price list uploaded.



Settings

Upload your Documents

Upload your PDF Files and Click on the Process Button



Settings

Upload your Documents

Upload your PDF Files and Click on the Process Button

Drag and drop files here
Limit 200MB per file

DocuQuery: AI-Powered PDF Knowledge Assistant

Ask a Question from the PDF Files

what products are there in this list

Human: what is there in the pdf

Bot: This is a price list for HSS drill bits.

Human: what material is used for manufacturing

Bot: The materials used in the manufacturing of these products are M2, M7, M35, M42, T42, ASP2017, ASP2030, and AST2060.

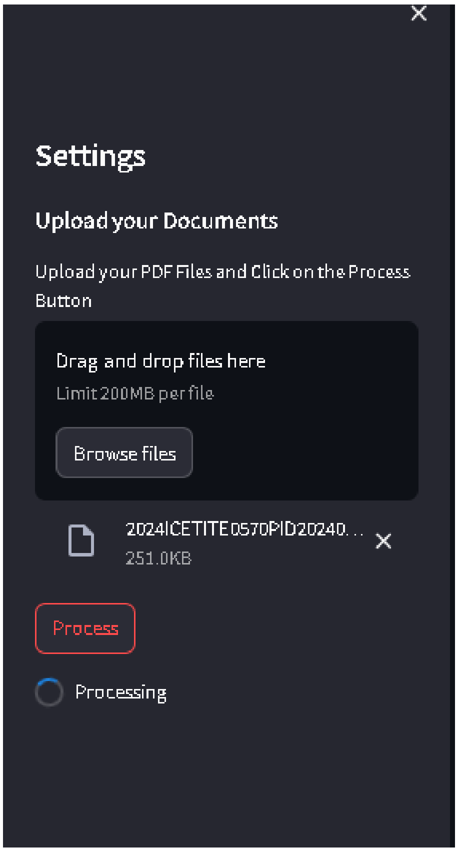
Human: what products are there in this list

Bot: The products in the list are:

1. Steel Files
2. HSS Tool Bit
3. HSS Micro Drills
4. HSS Jobber Drills
5. HSS Stub Drills
6. HSS Long Series Drills
7. Masonry Drills
8. HSS Extra Long Drills
9. HSS Centre Drills
10. HSS End Mill
11. HSS Taps Coarse Pitch
12. HSS Taps Fine Pitch
13. HSS Taps BSW, UNC, BSF, UNF, BSP & BA
14. Solid Carbide End Mills & Ball Nose
15. Solid Carbide Extra Long End Mills

Scenario 3: Research Paper Simplifier

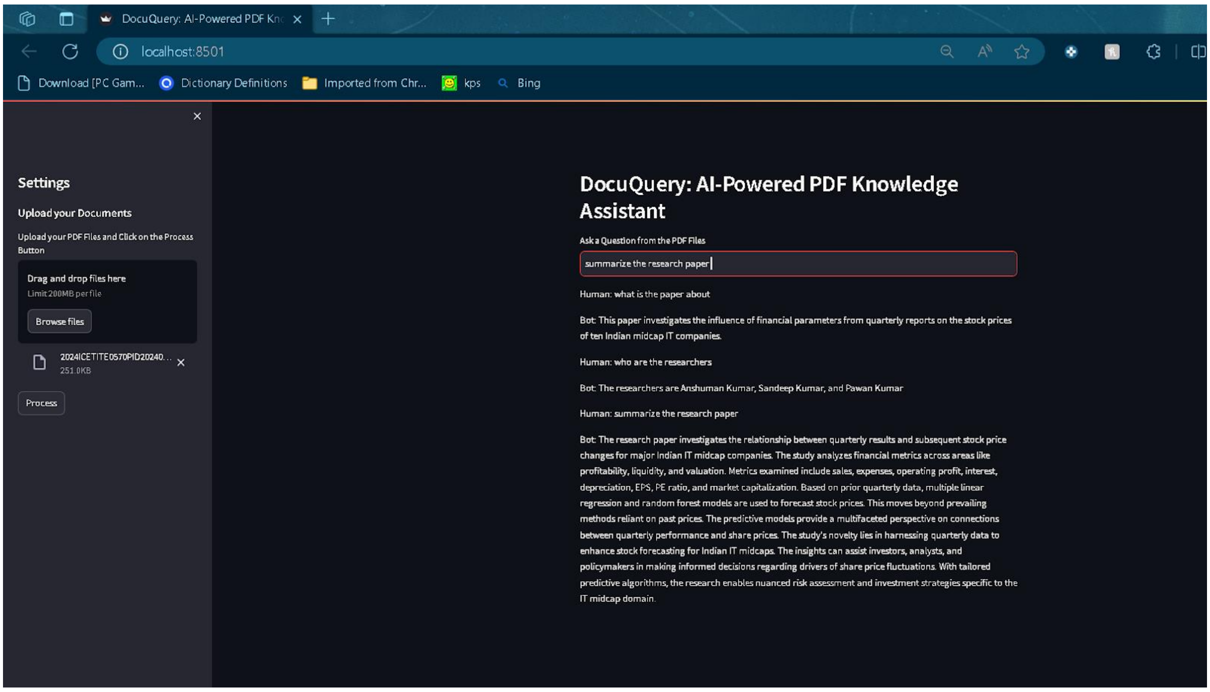
- Users can upload research papers or scholarly articles.
- The tool generates concise summaries of the uploaded research papers, capturing the key insights and findings.
- Users can ask questions related to the content of the research papers, and the tool provides accurate answers based on the summarized content.
- The output may include summarized versions of the research papers, along with the questions asked and the corresponding answers provided by the tool.



Settings

Upload your Documents

Upload your PDF Files and Click on the Process Button



DocuQuery: AI-Powered PDF Knowledge Assistant

Ask a Question from the PDF Files

summarize the research paper

Human: what is the paper about

Bot: This paper investigates the influence of financial parameters from quarterly reports on the stock prices of ten Indian midcap IT companies.

Human: who are the researchers

Bot: The researchers are Anshuman Kumar, Sandeep Kumar, and Pawan Kumar

Human: summarize the research paper

Bot: The research paper investigates the relationship between quarterly results and subsequent stock price changes for major Indian IT midcap companies. The study analyzes financial metrics across areas like profitability, liquidity, and valuation. Metrics examined include sales, expenses, operating profit, interest, depreciation, EPS, PE ratio, and market capitalization. Based on prior quarterly data, multiple linear regression and random forest models are used to forecast stock prices. This moves beyond prevailing methods reliant on past prices. The predictive models provide a multifaceted perspective on connections between quarterly performance and share prices. The study's novelty lies in harnessing quarterly data to enhance stock forecasting for Indian IT midcaps. The insights can assist investors, analysts, and policymakers in making informed decisions regarding drivers of share price fluctuations. With tailored predictive algorithms, the research enables nuanced risk assessment and investment strategies specific to the IT midcap domain.