Docu Query: AI-Powered PDF Knowledge Assistant Using Google PALM

An AI-Powered PDF Knowledge Assistant using Google PALM (Pathways Language Model) represents a highly advanced solution for extracting, interacting with, and Analysing information from PDF documents. The goal of such an assistant is to make PDF documents more accessible and easier to interact with by leveraging powerful AI models like PALM, which provide sophisticated language understanding and reasoning capabilities.

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 analyse 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 Stream lit UI
    - Sidebar Settings
    - Document Processing
    - Run the Application

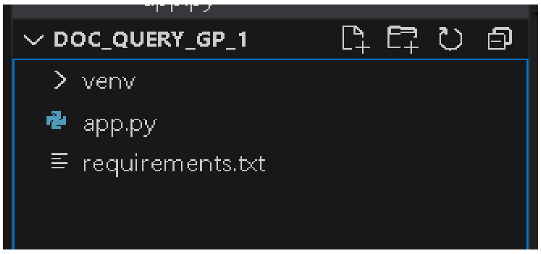
Prior Knowledge:

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
- Stream lit: 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/

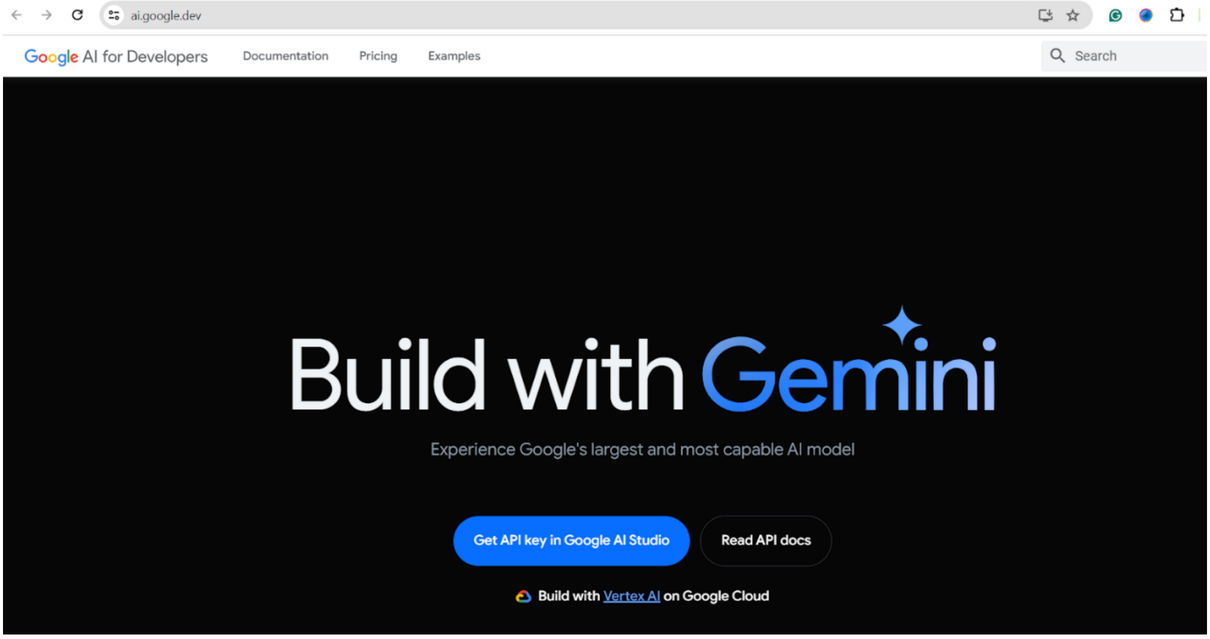
Project Structure:

Create the Project folder which contains files as shown below

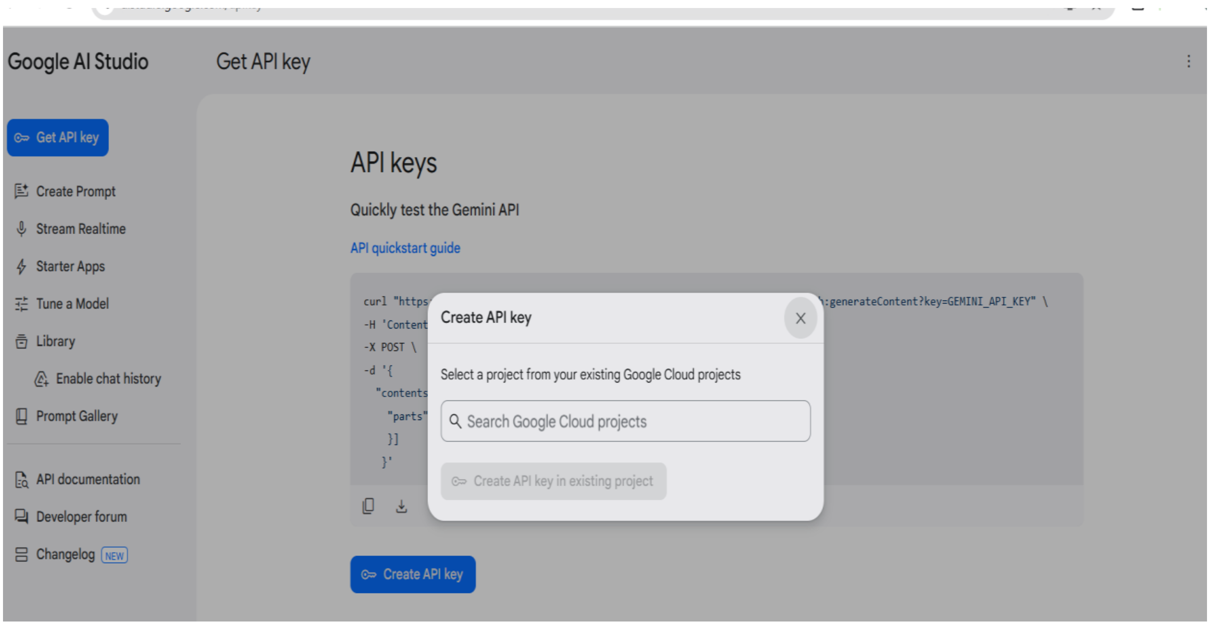


```
DOC_QUERY_GP_1
  > venv
    app.py
    requirements.txt
```

Generate PALM API:



ai.google.dev
Google AI for Developers
Documentation
Pricing
Examples
Search
Build with Gemini
Experience Google's largest and most capable AI model
Get API key in Google AI Studio
Read API docs
Build with Vertex AI on Google Cloud



API keys

Quickly test the Gemini API

API quickstart guide

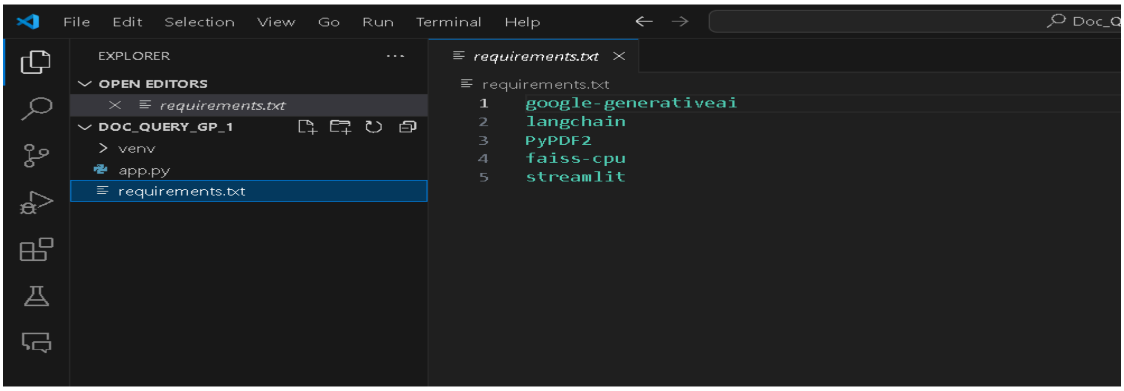
```
curl "https                    :generateContent?key=GEMINI_API_KEY" \
-H 'Content
-X POST \
-d '{
  "contents
    "parts"
    }]
   }'
```

Create API key

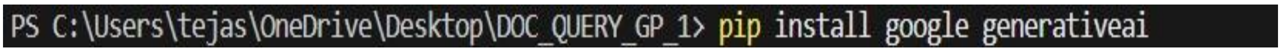
Select a project from your existing Google Cloud projects

Search Google Cloud projects
Create API key in existing project
Create API key

Install and import necessary libraries for the project:



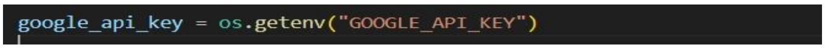
```
google-generativeai
langchain
PyPDF2
faiss-cpu
streamlit
```



```
PS C:\Users\tejas\OneDrive\Desktop\DOC_QUERY_GP_1> pip install google generativeai
```

- PyPDF2 (PyPDF2): Library for reading and extracting text from PDF files.
- Lang Chain 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.

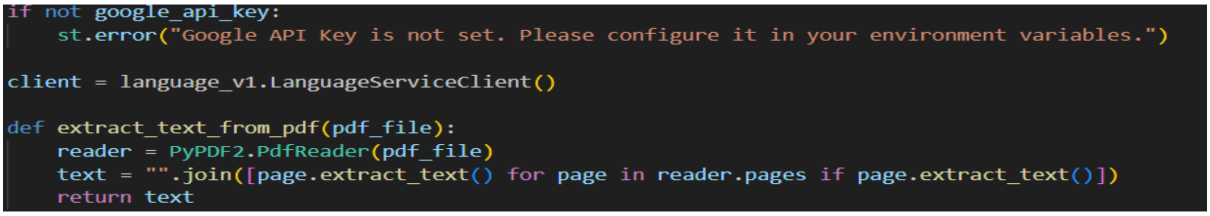
Add the Google API Key to the code:



```
google_api_key = os.getenv("GOOGLE_API_KEY")
```

- Set the environment variable GOOGLE_API_KEY to the obtained API key value.
- Example: os.environ['GOOGLE_API_KEY'] = 'YOUR_API_KEY_HERE'

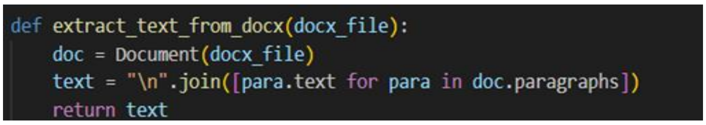
Extract Text from PDF Documents:



```
if not google_api_key:
    st.error("Google API Key is not set. Please configure it in your environment variables.")

client = language_v1.LanguageServiceClient()

def extract_text_from_pdf(pdf_file):
    reader = PyPDF2.PdfReader(pdf_file)
    text = "".join([page.extract_text() for page in reader.pages if page.extract_text()])
    return text
```

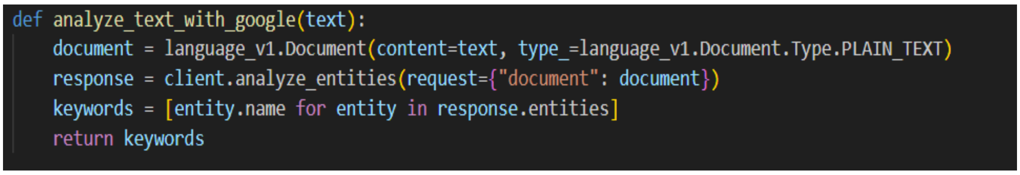
Define a function get_pdf_text that takes a list of PDF documents as input.

- 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.



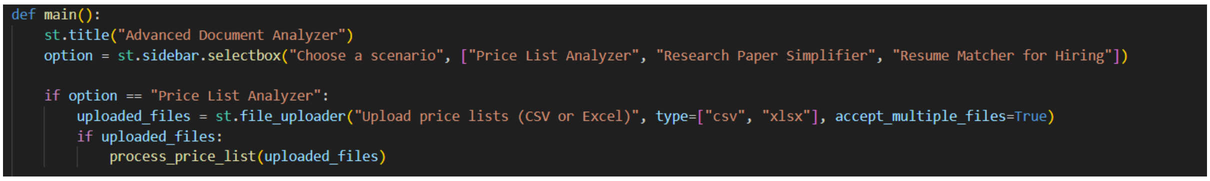
```
def extract_text_from_docx(docx_file):
    doc = Document(docx_file)
    text = "\n".join([para.text for para in doc.paragraphs])
    return text
```

- Define a function extract text from docx 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 docx file.



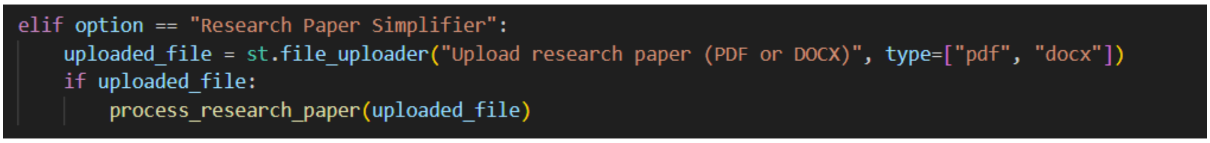
```python
def analyze_text_with_google(text):
    document = language_v1.Document(content=text, type_=language_v1.Document.Type.PLAIN_TEXT)
    response = client.analyze_entities(request={"document": document})
    keywords = [entity.name for entity in response.entities]
    return keywords
```

- Define a function analyse text with google that takes a user question as input.
- Call the conversation method of language_v1. Document with the user question as a parameter to initiate a conversation.
- Store the conversation history returned by the conversation method in the client.analyse_entities session state variable.



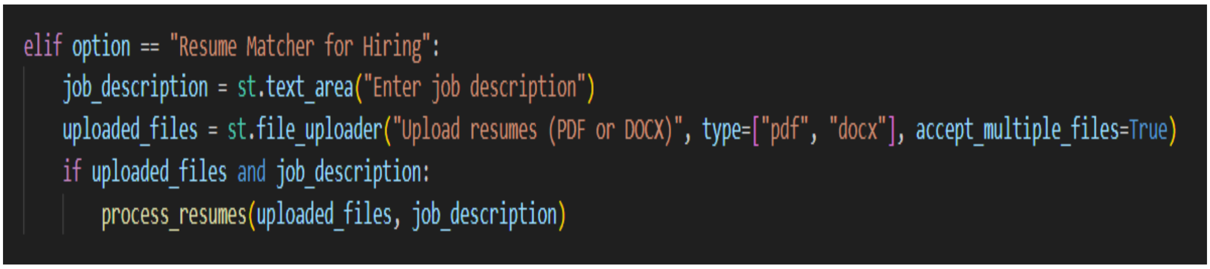
```python
def main():
    st.title("Advanced Document Analyzer")
    option = st.sidebar.selectbox("Choose a scenario", ["Price List Analyzer", "Research Paper Simplifier", "Resume Matcher for Hiring"])

    if option == "Price List Analyzer":
        uploaded_files = st.file_uploader("Upload price lists (CSV or Excel)", type=["csv", "xlsx"], accept_multiple_files=True)
        if uploaded_files:
            process_price_list(uploaded_files)
```

- 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.



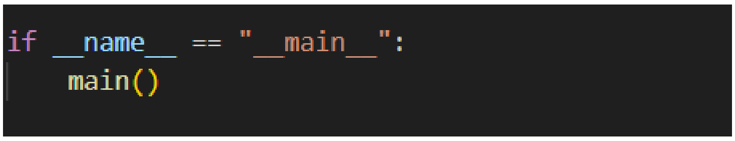
```python
elif option == "Research Paper Simplifier":
    uploaded_file = st.file_uploader("Upload research paper (PDF or DOCX)", type=["pdf", "docx"])
    if uploaded_file:
        process_research_paper(uploaded_file)
```

- 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.



```
elif option == "Resume Matcher for Hiring":
    job_description = st.text_area("Enter job description")
    uploaded_files = st.file_uploader("Upload resumes (PDF or DOCX)", type=["pdf", "docx"], accept_multiple_files=True)
    if uploaded_files and job_description:
        process_resumes(uploaded_files, job_description)
```

- 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_area function.
- Generate embeddings for the text chunks and create a vector store using the get st.file_uploader 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.

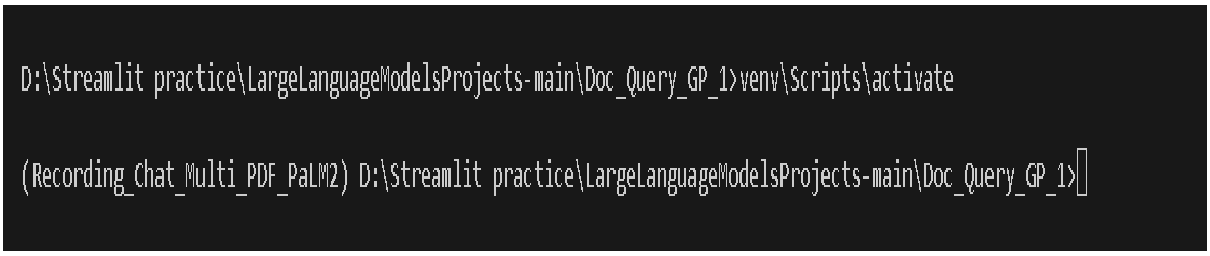


```
if __name__ == "__main__":
    main()
```

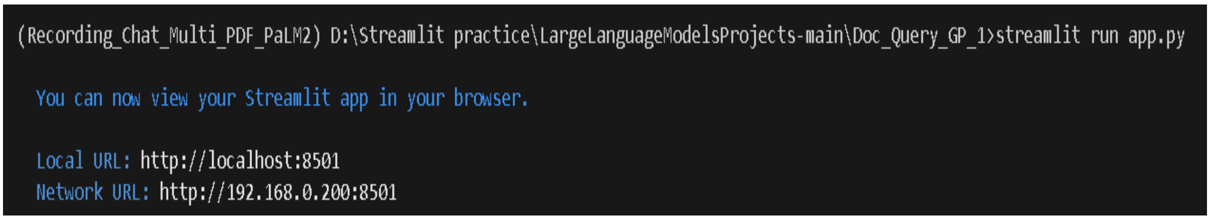
- Use the __name__ variable to ensure that the main function is executed when the script is run as the main program.
- Run the DocuQuery application by calling the main function.

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”



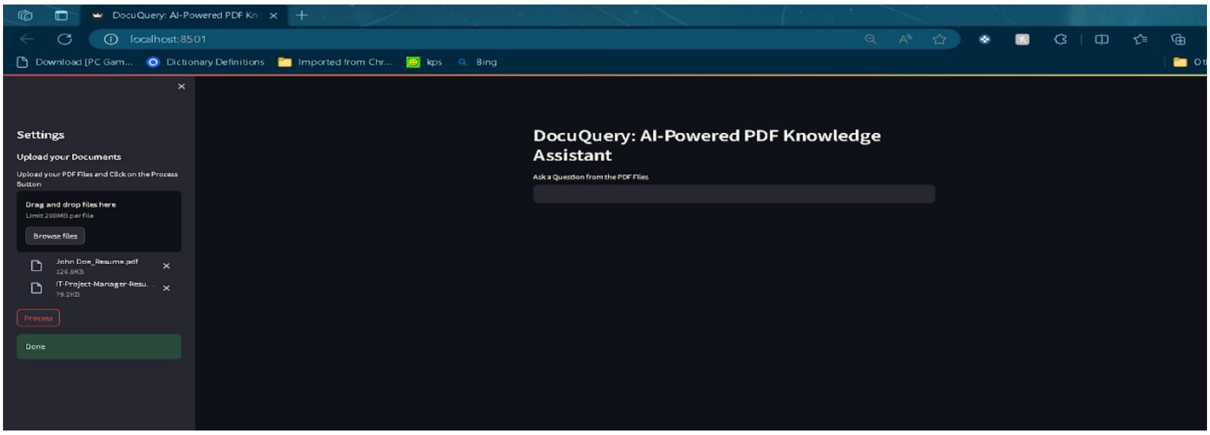
```
D:\Streamlit practice\LargeLanguageModelsProjects-main\Doc_Query_GP_1>venv\Scripts\activate

(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

Browse files

John Doe_Resume.pdf 126.8KB

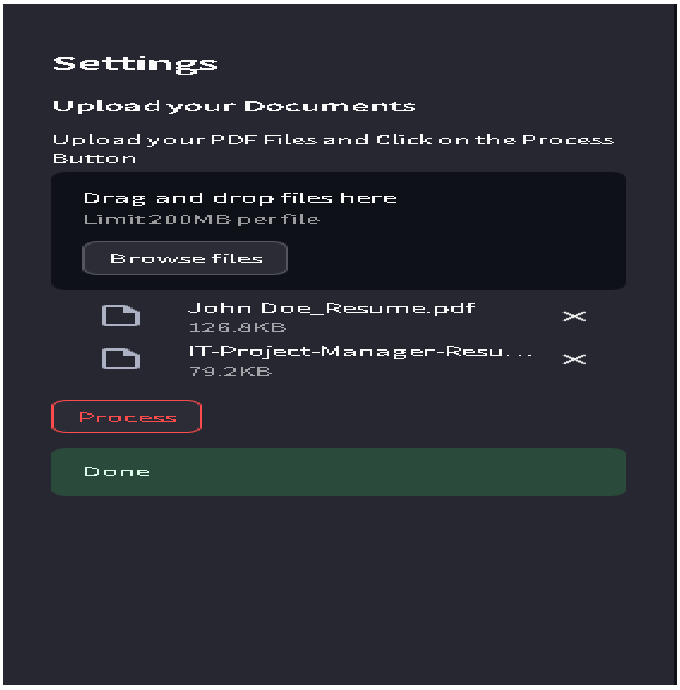
IT-Project-Manager-Resu... 79.2KB

Process

Done

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

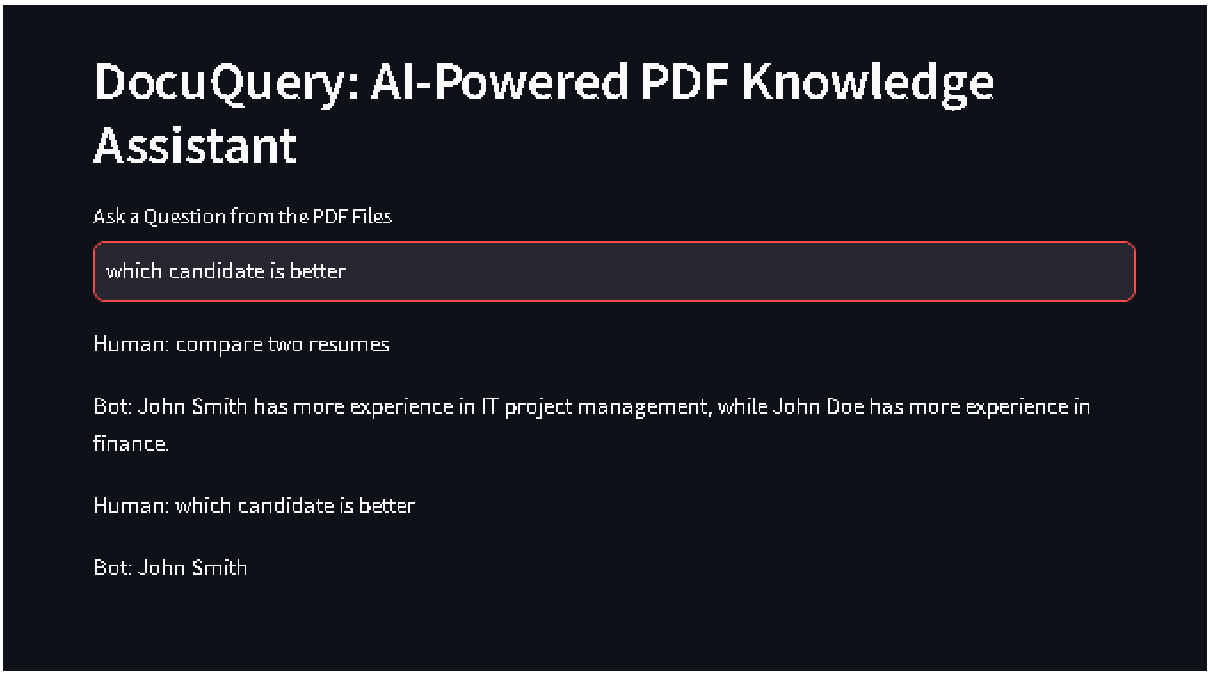
Browse files

John Doe_Resume.pdf 126.8KB

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

Process

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