**AI-Powered PDF Knowledge Assistant Using Google PALM**

**Project Description:**

This AI-powered assistant utilizes Google PALM to extract, analyze, and summarize content from PDF documents, making information retrieval seamless. It is designed to assist professionals across various domains by automating document understanding and answering queries in real-time.

**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.

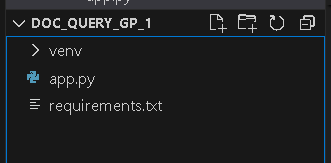
**Prior Knowledge**

 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/introducingon/

**Project Structure**

Create the Project folder which contains files as shown below



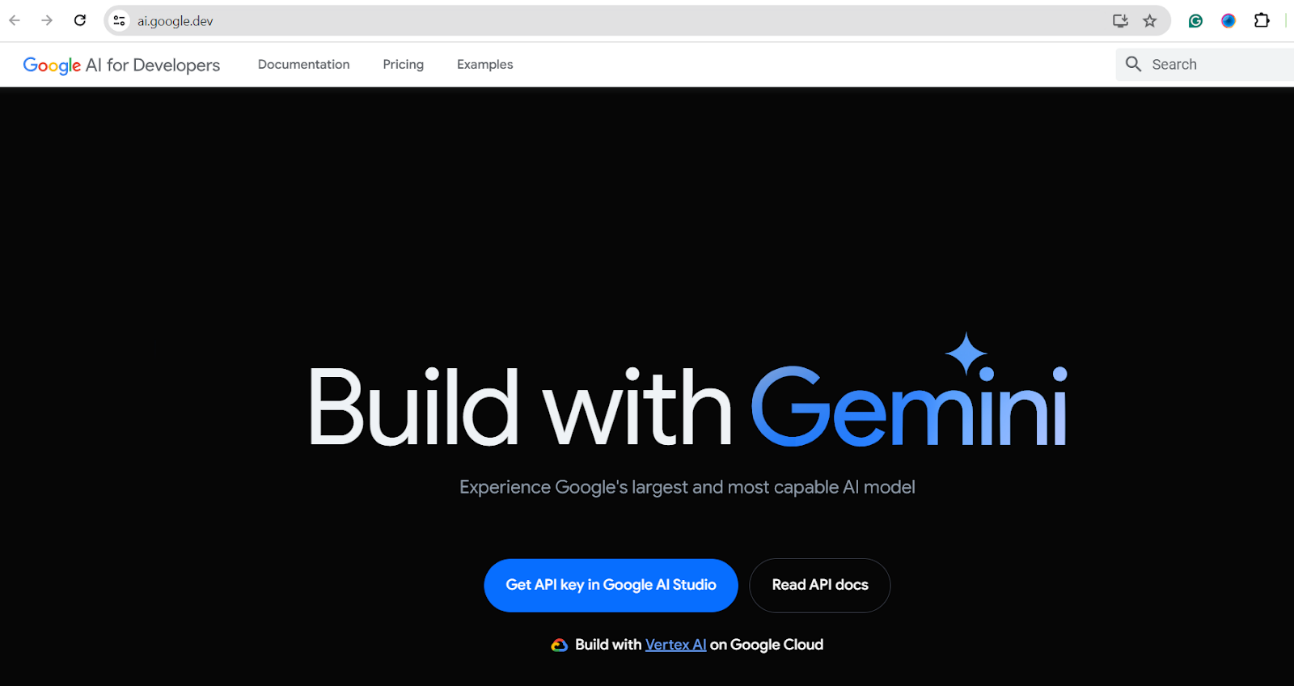
* We are building a Streamlit application in which we will use Python script.

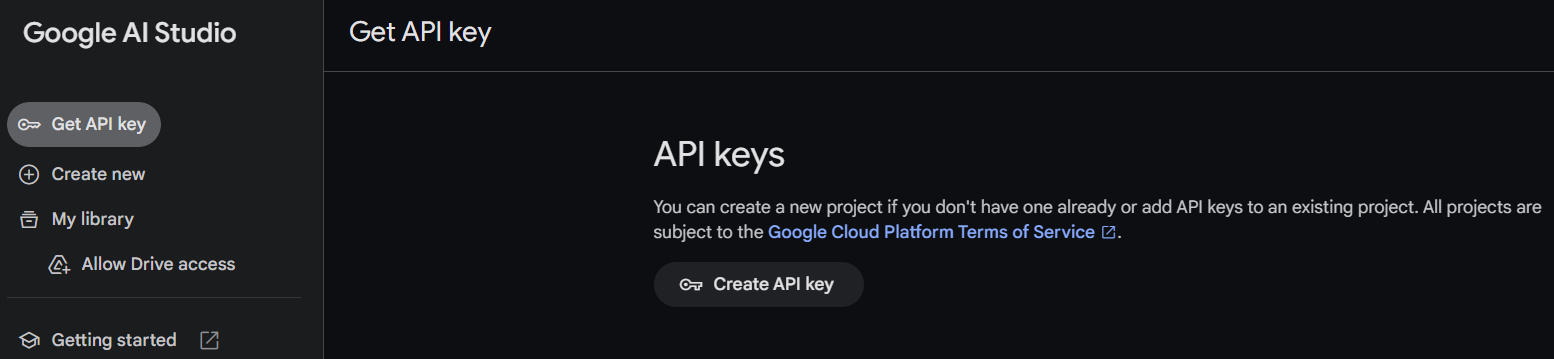
**Activity 1: Setting Up Google API Key**

* 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.



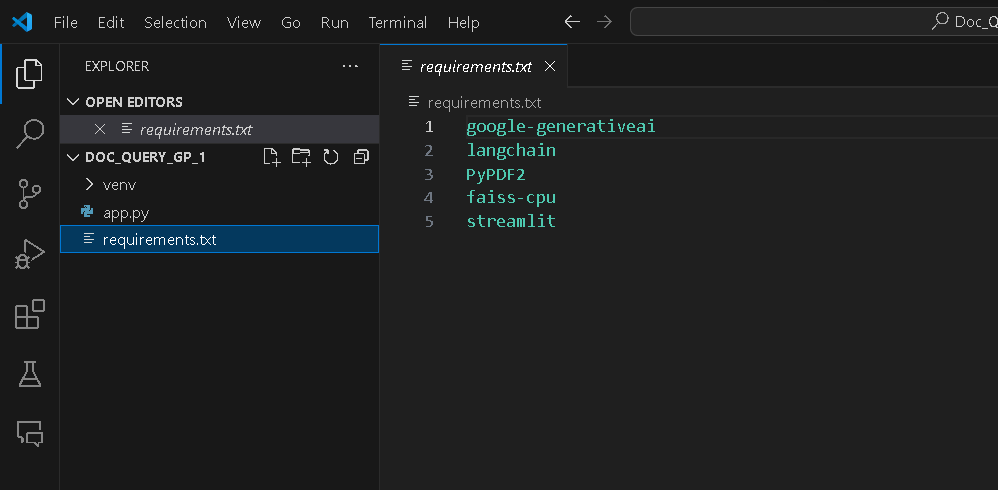


**Activity 2: Installation and Importing of Libraries and Adding API Key**

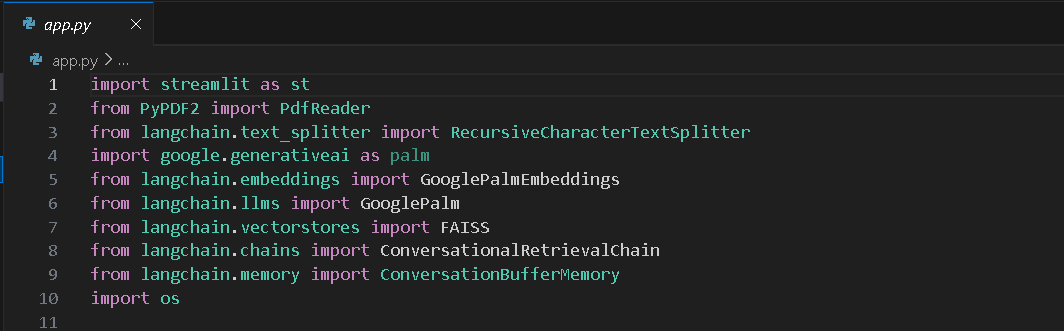
In this milestone, we will import necessary libraries and add the API Key generated from the above milestone.

**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

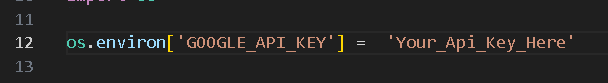






* Streamlit (streamlit): Web application framework for building interactive web apps.
* PyPDF2 (PyPDF2): Library for reading and extracting text from PDF files.
* LangChain Text Splitter (langchain.text\_splitter): Module for splitting text into smaller chunks.
* Google Generative AI (google.generativeai): 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.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.

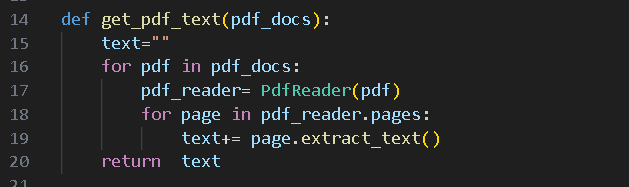
**Add the Google API Key to the code:**



* Set the environment variable GOOGLE\_API\_KEY to the obtained API key value.
* Example: os.environ['GOOGLE\_API\_KEY'] = 'YOUR\_API\_KEY\_HERE'

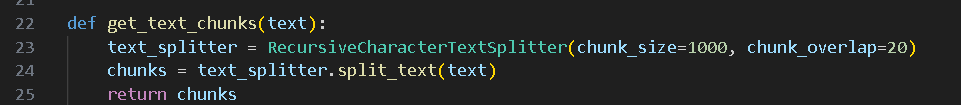
**Activity 3: PDF Text Processing**

**Extract Text from PDF Documents:**



* 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.

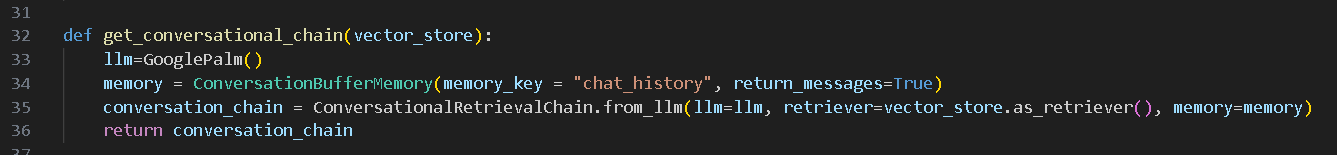
**Split Text into Chunks:**



* 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.

**Activity 3: Conversational Chain Setup**

**Setting Up Conversational Retrieval 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.

**Activity 4: User Interaction and Response Handling**

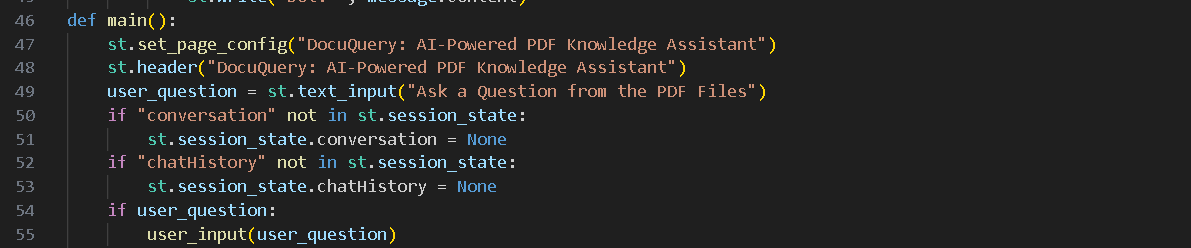
**Handling User Input and Generating Responses**



* 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.

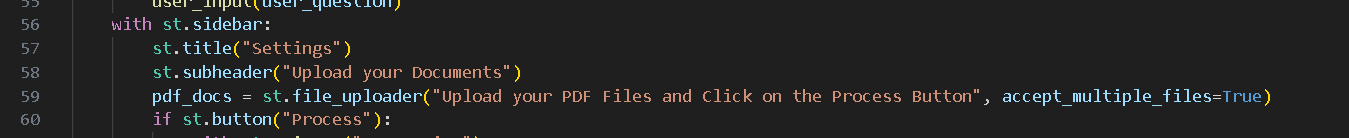
**Activity 5: Application Setup and Integration**

**Configure Streamlit UI:**



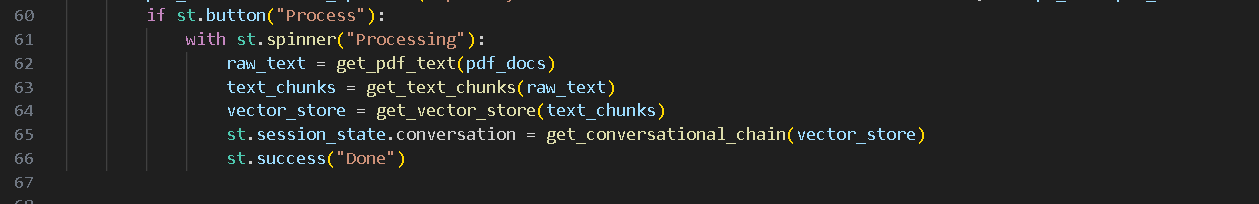
* Define a function main to set up the Streamlit UI for the DocuQuery application.
* Set the page configuration to display the title "DocuQuery: AI-Powered PDF Knowledge Assistant".
* Add a header to the UI displaying the application title.
* Include a text input field for users to ask questions from 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.

**Sidebar Settings:**



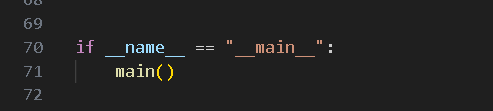
* Within the sidebar, add a title "Settings".
* Include a subheader "Upload your Documents" to guide users.
* Provide a file uploader component for users to upload PDF files.
* Add a button labeled "Process" to initiate document processing.

**Document Processing :**



* 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.

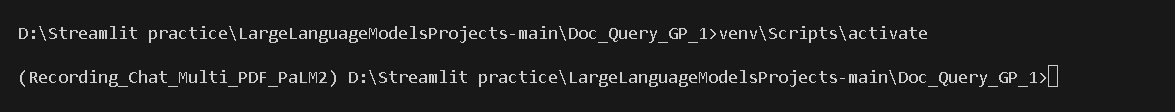
**Run the Application**

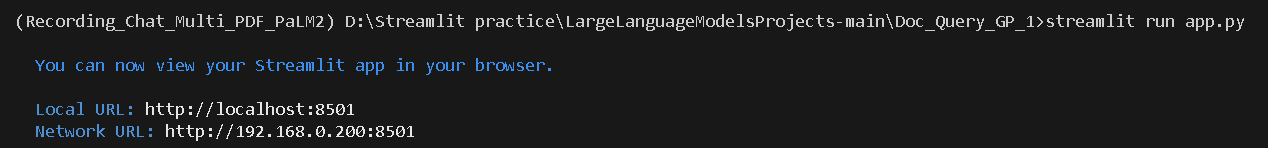


* Use the \_\_name\_\_ variable to ensure that the main function is executed when the script is run as the main program.
* Run the DocuQuery application by calling the main function.

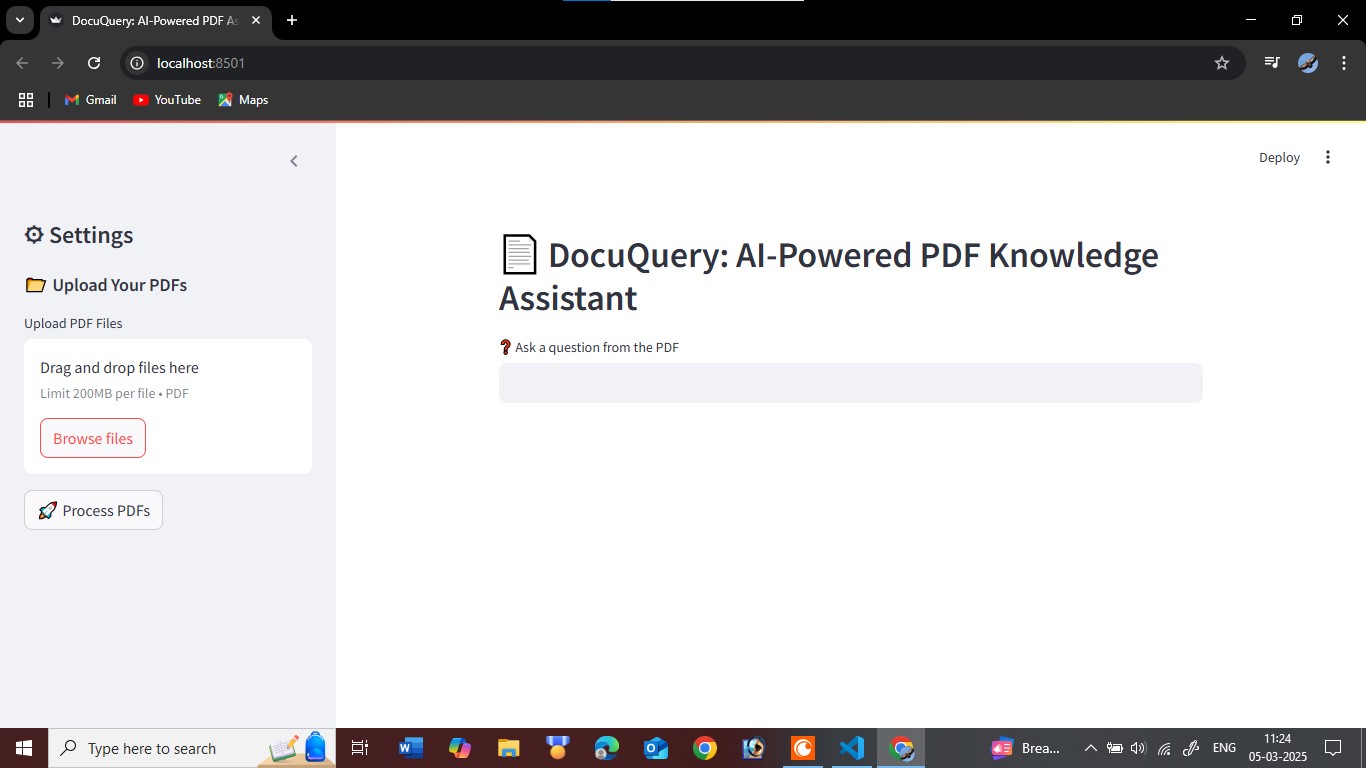
The output of the DocuQuery project, considering all three scenarios, would involve an interactive web application interface where users can perform various tasks related to PDF documents. Here's how the output would look like for each scenario:

* 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”

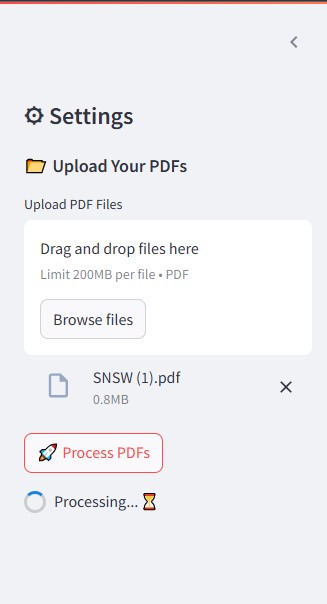




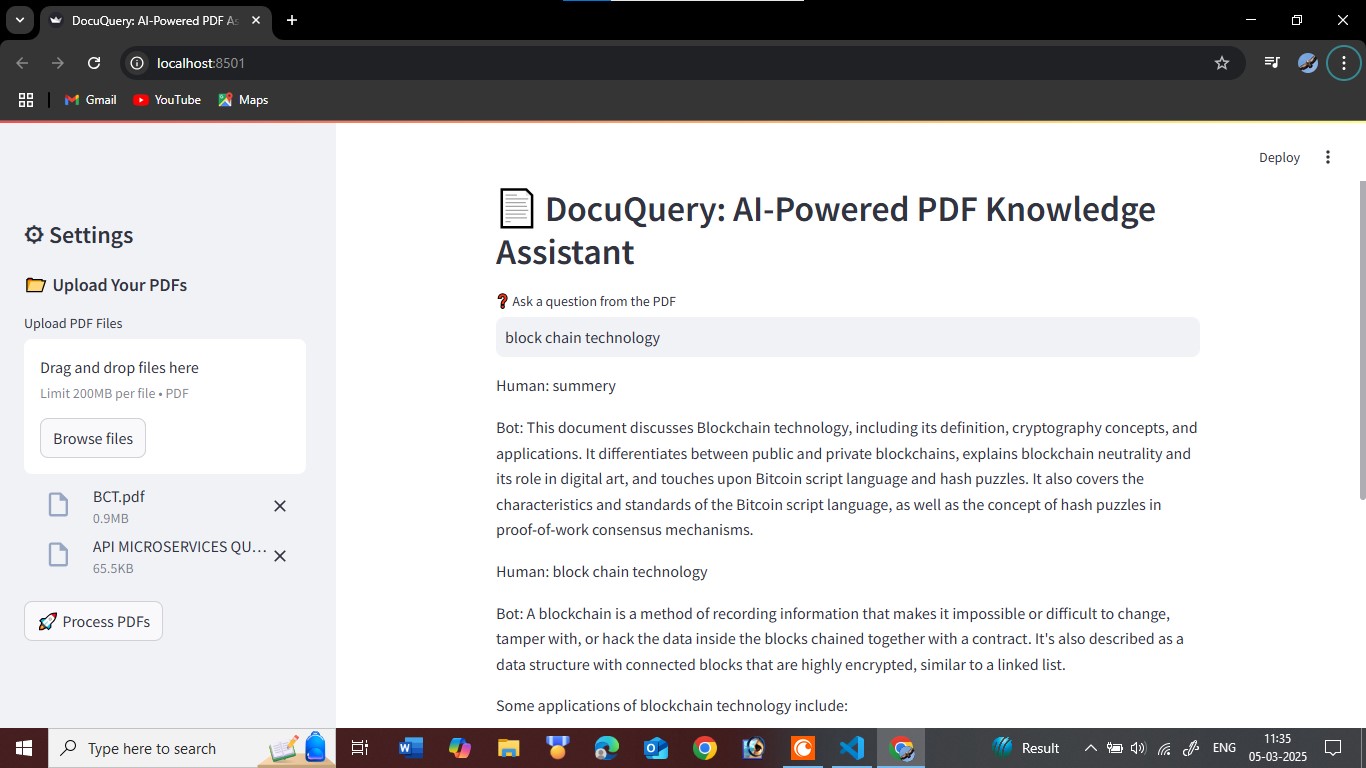
Now,the application will open in the web browser,



After giving the input:



The Output Generated:



**Conclusion:**

The AI-Powered PDF Knowledge Assistant Using Google PALM is a highly efficient and intelligent document processing tool designed to streamline data extraction, analysis, and retrieval from PDFs. By leveraging advanced AI capabilities, this assistant significantly enhances productivity across various domains, including procurement, research, and recruitment. Overall, this AI-driven tool revolutionizes document management by providing instant, accurate, and insightful information extraction, reducing manual efforts, and enhancing decision-making processes. It serves as a powerful solution for organizations and professionals looking to optimize their workflows and maximize efficiency.