DocuQuery: AI-Powered PDF Knowledge Assistant Using Google PALM

Skills Required:

- Python

- NLP (Natural Language Processing)

- Streamlit for UI Development

- Google Gemini

Project Description

DocuQuery is an AI-powered tool that enhances document processing by leveraging Google PALM for natural language understanding. The tool enables users to interact with PDF documents through intelligent querying, automated summaries, and knowledge extraction.

Use Cases

Price List Analyzer

Businesses dealing with multiple supplier price lists can use DocuQuery to extract and compare item prices efficiently. Users can upload multiple price lists, and the tool will extract relevant details, facilitating better procurement decisions.

Research Paper Summarizer

Researchers can upload academic papers, and DocuQuery will generate concise summaries of key findings. Users can also pose questions related to the paper's content, improving research efficiency.

Resume Matcher for Hiring

Recruiters can upload resumes and specify job criteria. The tool will analyze and rank candidates based on their qualifications and experience, streamlining the hiring process.

Technical Implementation

Dependencies

- Streamlit for UI interaction

- PyPDF2 for extracting text from PDFs

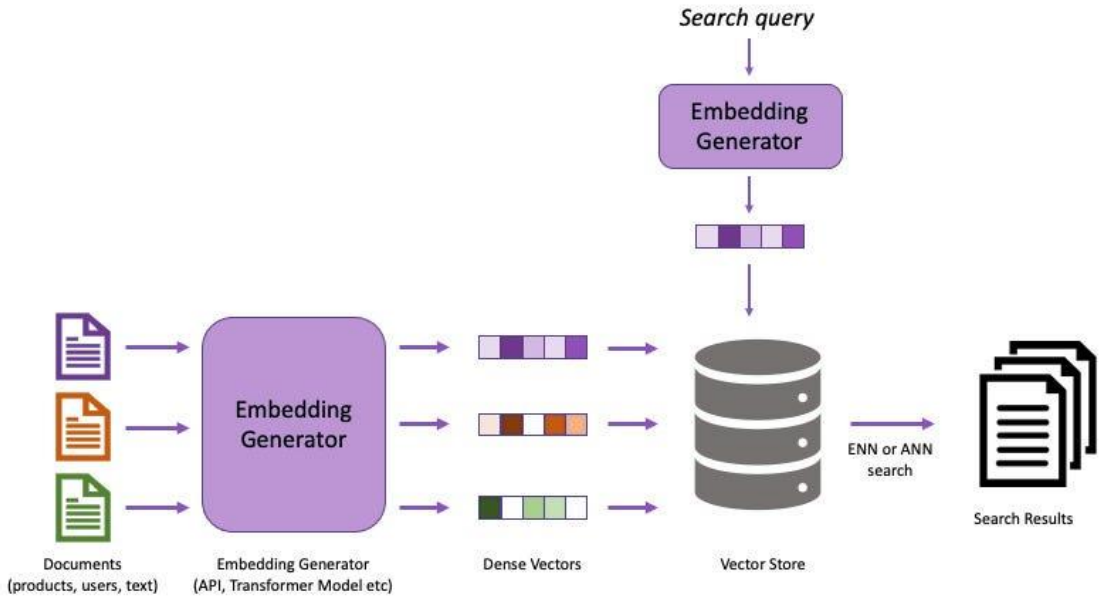
- Langchain for text processing and embedding

- Google Generative AI (PALM) for embeddings and conversational AI

- FAISS for vector storage and similarity search

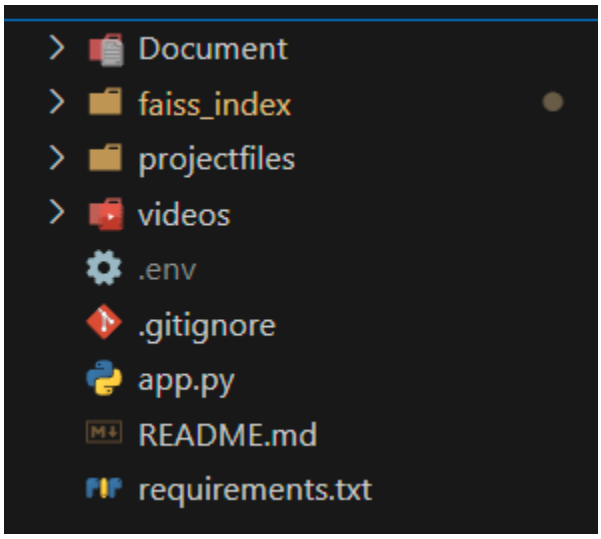
- dotenv for environment variable management

Architecture



Project Structure:

Create the Project folder which contains application file as shown below



Milestone 1: Requirements Specification

Specifying the required libraries in the requirements.txt file ensures seamless setup and reproducibility of the project environment, making it easier for others to replicate the development environment.

Activity 1: Create a requirements.txt file to list the required libraries.

```
streamlit
google-generativeai
python-dotenv
langchain
PyPDF2
chromadb
faiss-cpu
langchain_google_genai
```

Activity 2: Install the required libraries.

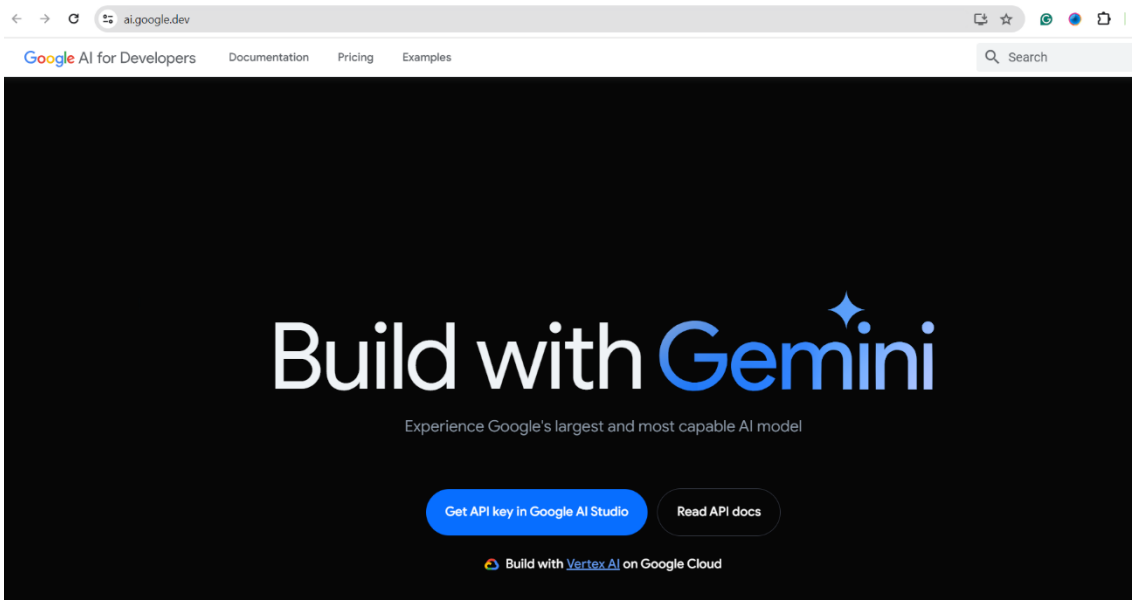
```
(gemini) PS D:\github\DocuQuery-AI-Powered-PDF-Knowledge-Assistant-Using-Google-PALM> pip install -r .\requirements.txt
```

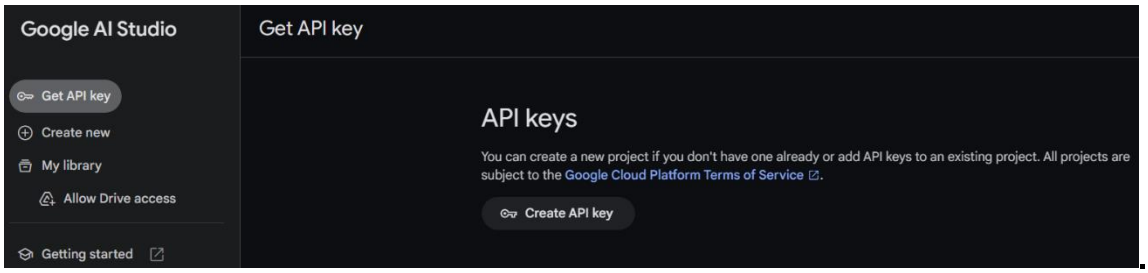
Milestone 2: Initializing the Model

For initializing the model we need to generate PALM API.

Activity 1: 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: Define App logic and components

Extracting PDF Text

from PyPDF2 import PdfReader

```
from PyPDF2 import PdfReader

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

Splitting Text into Chunks

from langchain.text_splitter import RecursiveCharacterTextSplitter

```
def get_text_chunks(text):
    """ Split text into chunks. We will use these chunks to create a vector store """
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=100_000, chunk_overlap=1000)
    chunks = text_splitter.split_text(text)      You, 5 days ago • palm hasn't worked so went
    return chunks
```

Creating a Vector Store

from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain_community.vectorstores import FAISS

```python
def get_vector_store(text_chunks):
    """ Create a vector store using the text chunks """
    embeddings = GoogleGenerativeAIEmbeddings(model = "models/embedding-001")
    vector_store = FAISS.from_texts(text_chunks, embedding=embeddings)
    vector_store.save_local("faiss_index")
```

Milestone 3: Interfacing with Pre-trained Model

In this milestone, we will build a prompt template to generate feedback based on the project details entered by the user.

Conversational AI with Gemini

from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.chains.question_answering import load_qa_chain
from langchain.prompts import PromptTemplate

```python
def get_conversational_chain():
    """ Create a conversational chain using the Gemini model """

    prompt_template = """
    Answer the question as detailed as possible from the provided context, make sure to provide all the details, if the answer is not in
    provided context just say, "answer is not available in the context", don't provide the wrong answer\n\n
    Context:\n {context}?\n
    Question: \n{question}\n

    Answer:
    """

    model = ChatGoogleGenerativeAI(model="gemini-2.0-flash",
                                   temperature=0.3)

    prompt = PromptTemplate(template = prompt_template, input_variables = ["context", "question"])
    chain = load_qa_chain(model, chain_type="stuff", prompt=prompt)

    return chain
```

Handling User Input

import streamlit as st

```python
def user_input(user_question):
    """ Get user question and return the response """
    embeddings = GoogleGenerativeAIEmbeddings(model = "models/embedding-001")

    new_db = FAISS.load_local("faiss_index", embeddings, allow_dangerous_deserialization=True)
    docs = new_db.similarity_search(user_question)

    chain = get_conversational_chain()


    response = chain(
        {"input_documents":docs, "question": user_question}
        , return_only_outputs=True)

    st.write("Reply: ", response["output_text"])
```

Milestone 4: Model Deployment

In this milestone, we are deploying the created model using streamlit. Model deployment using Streamlit involves creating a user-friendly web interface, enabling users to interact with the model through a browser. Streamlit provides easy-to-use tools for developing and deploying data-driven applications, allowing for seamless integration of models into web-based applications.

Main Application

import streamlit as st

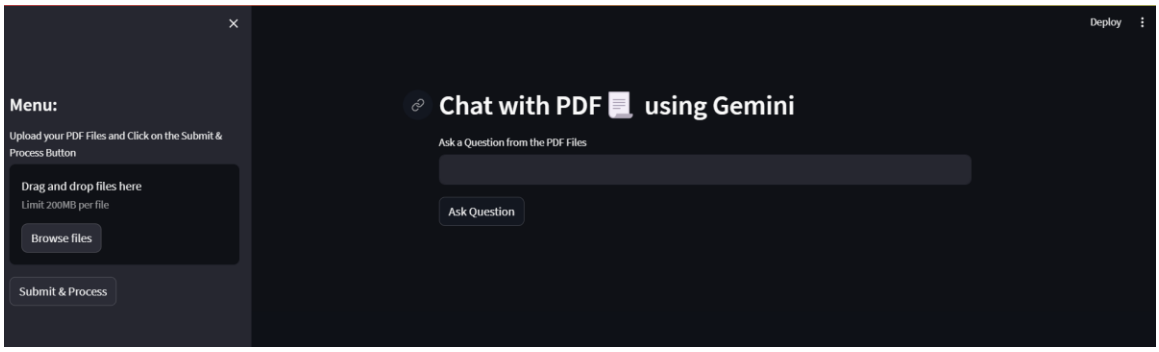
```
def main():
    """ Document Chat Application """
    st.set_page_config("Chat PDF")
    st.header("Chat with PDF  using Gemini")

    user_question = st.text_input("Ask a Question from the PDF Files")

    if st.button("Ask Question") and user_question:
        with st.spinner("Processing..."):
            user_input(user_question)

    with st.sidebar:
        st.title("Menu:")
        pdf_docs = st.file_uploader("Upload your PDF Files and Click on the Submit & Process Button", accept_multiple_files=True)
        if st.button("Submit & Process"):
            with st.spinner("Processing..."):
                raw_text = get_pdf_text(pdf_docs)
                text_chunks = get_text_chunks(raw_text)
                get_vector_store(text_chunks)
                st.success("Done")



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

Run the web application

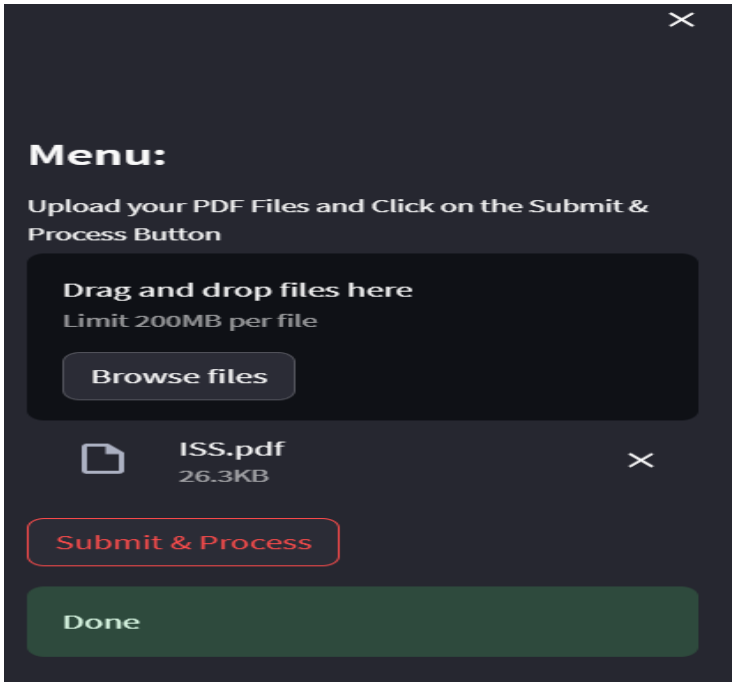
- Open the anaconda prompt from the start menu
- Navigate to the folder where your Python script is.
- Now type “streamlit run app.py” command
- Navigate to the localhost where you can view your web page

```
(gemini) PS D:\github\DocuQuery-AI-Powered-PDF-Knowledge-Assistant-Using-Google-PALM> streamlit run .\app.py

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.2:8501
```

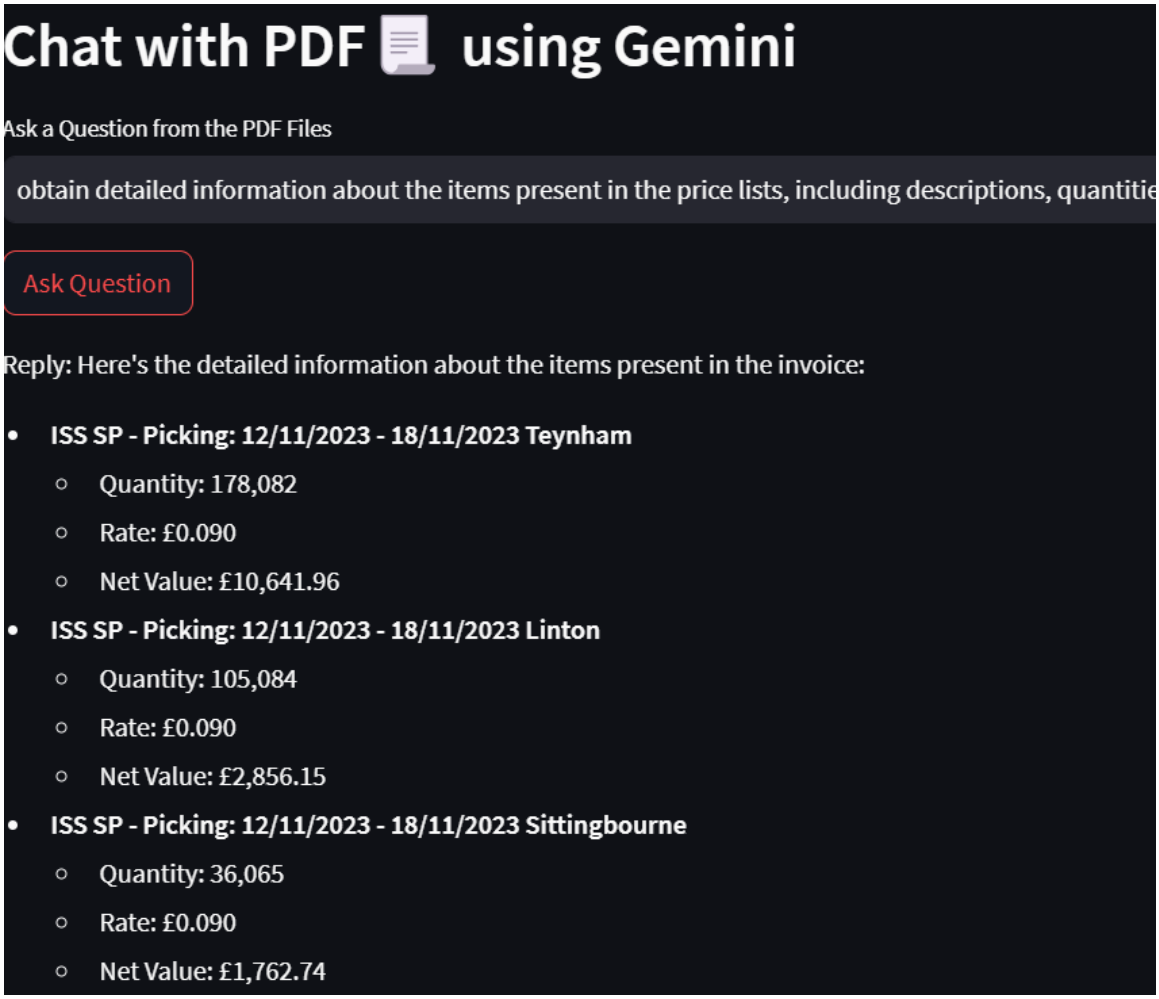
Now, the application will open in the web browser,



Upload the document and click on submit and process to create embedding and store in FAISS.



Now ask questions like retrieve info



This will retirive the information from the embeedings stored in the vectore store and give the relevant information.

Conclusion

DocuQuery is a smart tool that makes reading PDFs easy. Whether finding prices, summarizing papers, or choosing the best resume, it helps users get quick answers. With future improvements, it will become even more powerful. DocuQuery is like a helpful reading buddy for everyone!