Text and Image LLM Project

```
# Q&A Chatbot
#from langchain.llms import OpenAI

from dotenv import load_dotenv

load_dotenv()  # take environment variables from .env.

import streamlit as st
import os
import pathlib
import textwrap
from PIL import Image


import google.generativeai as genai


os.getenv("GOOGLE_API_KEY")
```

```
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

## Function to load OpenAI model and get respones

def get_gemini_response(input,image):
    model = genai.GenerativeModel('gemini-1.5-flash')
    if input!="":
       response = model.generate_content([input,image])
    else:
       response = model.generate_content(image)
    return response.text

##initialize our streamlit app

st.set_page_config(page_title="Gemini Image Demo")

st.header("Gemini Application")
input=st.text_input("Input Prompt: ",key="input")
uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])
image=""
```

```
if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image, caption="Uploaded Image.", use_column_width=True)



submit=st.button("Tell me about the image")

## If ask button is clicked

if submit:

    response=get_gemini_response(input,image)
    st.subheader("The Response is")
    st.write(response)
```

Chat With Multiple PDF

```
import streamlit as st
from PyPDF2 import PdfReader
from langchain_text_splitters import RecursiveCharacterTextSplitter
import os
from langchain_google_genai import GoogleGenerativeAIEmbeddings
import google.generativeai as genai
from langchain_community.vectorstores import FAISS
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.chains.question_answering import load_qa_chain
from langchain_core.prompts import PromptTemplate
from dotenv import load_dotenv

load_dotenv()
os.getenv("GOOGLE_API_KEY")
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))
```

```
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



def get_text_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000, chunk_overlap=1000)
    chunks = text_splitter.split_text(text)
    return chunks
```

```
def get_vector_store(text_chunks):
    embeddings = GoogleGenerativeAIEmbeddings(model = "models/embedding-001")
    vector_store = FAISS.from_texts(text_chunks, embedding=embeddings)
    vector_store.save_local("faiss_index")



def get_conversational_chain():


    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-pro",
                                   temperature=0.3)

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

    return chain



def user_input(user_question):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")

    # Load the FAISS vector store with dangerous deserialization enabled
```

```
    try:
        new_db = FAISS.load_local("faiss_index", embeddings, allow_dangerous_deserialization=True)
    except Exception as e:
        st.error(f"Error loading vector store: {e}")
        return

    # Perform similarity search
    docs = new_db.similarity_search(user_question)

    # Get the conversational chain
    chain = get_conversational_chain()

    try:
        # Generate a response
        response = chain(
            {"input_documents": docs, "question": user_question},
            return_only_outputs=True
        )
        st.write("Reply:", response["output_text"])
    except Exception as e:
```

```
        st.error(f"Error generating response: {e}")



def main():
    st.set_page_config("Chat with PDF")
    st.header("Chat with PDF using 💁 ")

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

    if user_question:
        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()
```