CHAT APPLICATION USING LANGCHAIN MODEL

Github→ https://github.com/Ranjith2012/chatapplication-FAI.git

Workflow of Application:

This code is a streamlit app that allows users to upload multiple PDF documents and engage in a conversation with a language model. The app processes the uploaded PDFs, extracts their text content, and splits it into smaller chunks. It then uses an OpenAI language model to generate responses to user questions about the content of the PDFs. The app displays the conversation history and provides a retry mechanism in case of errors during the conversation process.

Streamlit is a Python library that allows you to create web applications for data science and machine learning projects with minimal effort.Stream It provides a simple and intuitive API, enabling developers to quickly build web applications without the need for extensive knowledge of web development technology.

The uploaded PDF documents are processed using the PYPDF2 library to extract their text content.

Process 1:

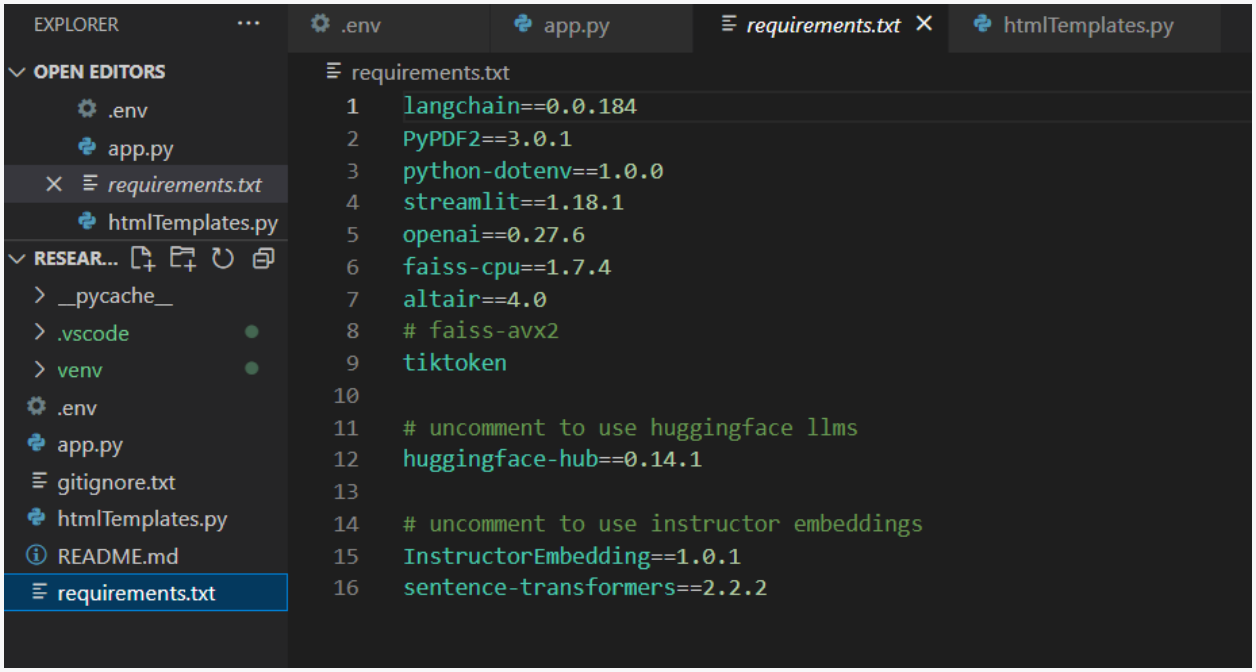
Creating the file directory for our project and creating the virtual environment is a feature provided by Python that allows developers to create isolated and self-contained environments for their projects. It helps address the issue of conflicting dependencies

Create Local Environment:

```
Stopping...
(venv) PS C:\Users\Win11\researchapp> venv\Scripts\activate
```

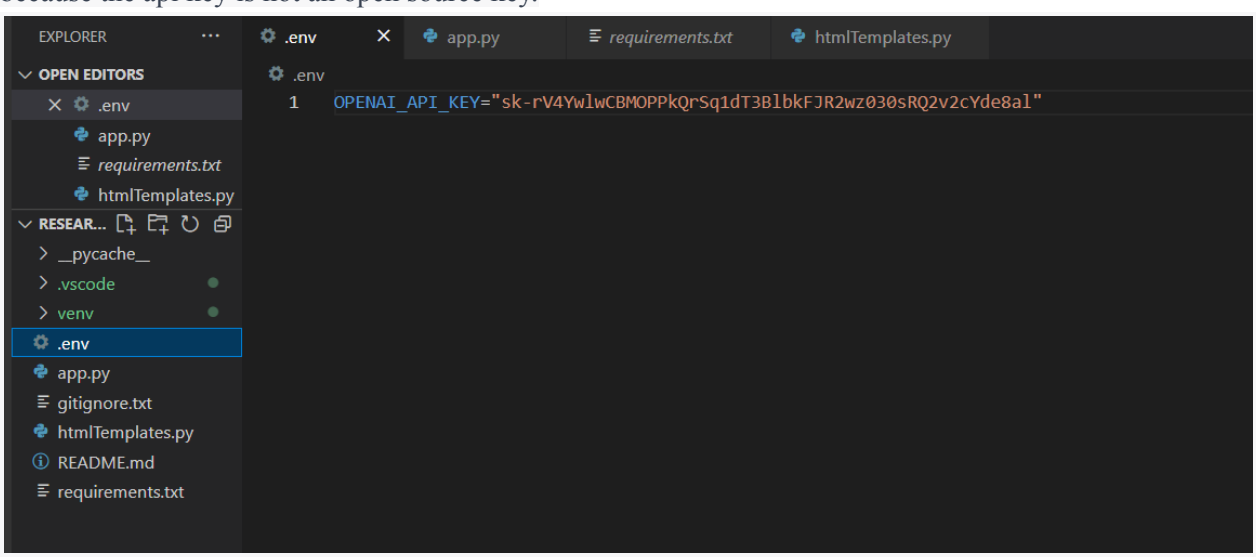
Process 2:

A requirements.txt file is a text file used in Python projects to specify the dependencies required to run the project. The different versions of packages are installed in the Virtual Environment.



Process 3:

Create the .env file to redirect the OPENAI_API_KEy to the our app.py file to access the variable because the api key is not an open source key.



Process 4:

Research Paper pdf processing:

The code is a Python function named get_pdf_text that takes a list of PDF documents as input. It uses the PyPDF2library to read and process each PDF, extracting the text content from all the pages. The extracted text from all the PDFs is then concatenated into a single string. The function returns this concatenated text, providing a convenient way to obtain the combined text content from multiple PDFs for further processing or analysis.

Text Processing:

- This function takes a single input parameter text, which is a long string of text content. It uses a custom CharacterTextSplitter to split the text into smaller chunks based on certain criteria.

- a chunk size of 1000 characters, and a chunk overlap of 200 characters. The length_function parameter is likely a custom function to calculate the length of the text.

- This function takes a list of text chunks, typically the output of the get_text_chunks function. It uses OpenAI's OpenAIEmbeddings to obtain embeddings vector representationsof the text chunks. The OpenAIEmbeddings is likely using an API key OPENAI_API_KEY to interact with OpenAI's services for obtaining embeddings.

- The code defines a function called get_conversation_chain that sets up a conversational retrieval chain for interactive conversations with a language model. It takes a vector store containing embeddings of text chunks as input, likely generated using OpenAI's embeddings and FAISS. The function creates a conversation chain using a language model instance, the vector store converted to a retriever, and a conversation memory.

- The main function sets up the Streamlit web application. It configures the page title, icon, and custom CSS. It initializes st.session_state.conversation and st.session_state.chat_history to None if they do not exist.

- The main part of the app is displayed with a header, a text input for user questions, and a sidebar for uploading PDF documents. When the user enters a question, the handle_userinput function is called, and the conversation is displayed.

- The sidebar allows users to upload PDF documents, and a button triggers the PDF processing. The text content of the PDFs is split into chunks, converted into embeddings, and used to create a conversation chain with the language model.

Process5:

To create the front end using htmlTemplates css variable contains a string representing CSS styles for the chat messages. It defines the appearance of user and bot messages in the chat interface. The styling includes padding, border-radius, background colors, and avatar positioning.

The bot_template variable contains an HTML template for displaying a bot message in the chat interface. It includes an avatar on the left side with an image of a robot, and the message content is displayed on the right side. The {{MSG}} placeholder is used to dynamically insert the bot's message text.

The user_template variable contains an HTML template for displaying a user message in the chat interface. It includes an avatar on the left side with an image of a user icon, and the message content is displayed on the right side. The {{MSG}} placeholder is used to dynamically insert the user's message text.

APPLICATION: