Firstly, I thoroughly reviewed the project and understand what needs to be done and how to execute it. Afterward, I broke down the project into small tasks, which I am detailing stepwise.

**Setup instructions:**

1. First clone the repo then create virtual environment in same directory.
2. Create a conda virtual environment of python version 3.10.0

conda create -p venvIntell python==3.10.0

conda activate path/of/venvIntell

1. Install the requirements.txt by using

pip install -r requirements.txt

1. Please create .env file to store openai API KEY
2. Please change the folder\_id of Automate\_Data\_Storage\_and\_Indexing.py in line no. 35

docs = load\_data(folder\_id="1uCxh7jmHBzU0ZUNix901qq2qkjkYHJPL")

**Step 2: Fetching Documents from Google Drive**

1. Developed a separate file for this task.

2. Retrieved various file types from Google Drive using the GoogleDriveReader.

3. Authenticated with Google and saved credentials.

* Downloaded the `credentials.json` file following the instructions (https://developers.google.com/drive/api/v3/quickstart/python).
* Copied and renamed the `credentials.json` file to `client\_secrets.json` for use by PyDrive.
* Note: Both files are essentially the same but required with different names according to the libraries used.

4. Followed the instructions in [base.py]

(https://github.com/run-llama/llama\_index/blob/main/llama-index-integrations/readers/llama-index-readers-google/llama\_index/readers/google/drive/base.py) for authentication.

Run the quickstart.py .

After running this token.json will be created.

1. Then run quickstart.py to authenticate with google drive
2. Next run main.py
3. Wait for some time then you are ready to chat with your own documents
4. Now you can ask your questions

**Step 1: Setting Up Environment**

1. Created a Conda virtual environment with Python version 3.10.0.

2. Installed the required dependencies from the `requirements.txt` file.

3. Stored the OpenAI API KEY in the `.env` file.

4. Utilized GPT-3.5 LLM for this project.

**Step 2: Fetching Documents from Google Drive**

1. Developed a separate file for this task.

2. Retrieved various file types from Google Drive using the GoogleDriveReader.

3. Authenticated with Google and saved credentials.

* Downloaded the `credentials.json` file following the instructions (https://developers.google.com/drive/api/v3/quickstart/python).
* Copied and renamed the `credentials.json` file to `client\_secrets.json` for use by PyDrive.
* Note: Both files are essentially the same but required with different names according to the libraries used.

4. Followed the instructions in [base.py]

(https://github.com/run-llama/llama\_index/blob/main/llama-index-integrations/readers/llama-index-readers-google/llama\_index/readers/google/drive/base.py) for authentication.

Run the quickstart.py .

After running this token.json will be created.

5. Created a function (`load\_data`) to load documents by folder ID and return the documents.

**Step 3: Indexing the Documents**

1. Developed a separate file for this task.

2. Used the `VectorStoreIndex` to index the returned documents.

3. Returned the indexing part with the help of a function.

from To\_Fetch\_Multiples\_documents\_from\_drive import load\_data

Created a custom function using `@st.cache\_resource(show\_spinner=False)` as a decorator to cache the resource and avoid unnecessary computations during subsequent executions. The `show\_spinner=False` parameter hides the spinner during caching.

import\_docs\_by\_fetching\_documents\_from\_drive()

A function for importing documents by fetching them from Google Drive:

* Used `with st.spinner(text="Loading and indexing your docs – hang tight! This should take 1-2 minutes.")` to display a spinner during the loading process with a specified loading message.

**Step 4: Frontend Part and Extraction of Metadata**

Developed a separate file for this step. In this phase, users can ask questions about their documents, and the model will provide answers along with relevant metadata (file\_name, file\_type, page\_number, author, etc.).

* Provided a text input box with Streamlit to allow users to ask questions.
* Explained on the UI what prompts to write for better answers.

Configured Streamlit page settings, including title, icon, layout, and initial sidebar state. Displayed the title for the chat application.

* If the chat messages history is not present in the session state, initialized it with a default message.
* Imported documents and built an index using the `import\_docs\_by\_fetching\_documents\_from\_drive` function.
* If the chat engine is not present in the session state, initialized it using the Llama Index with specified chat mode and verbosity.

Prompted the user for input using Streamlit's `chat\_input`.

* Iterated through prior chat messages and displayed them.
* If the last message is not from the assistant, generated a response using the chat engine.
* Displayed the response and appended it to the chat messages history.
* Displayed metadata for each source node in the response.

**Flow chart**

Suppose that we are using open source model

Display response with metadata

User can ask their queries from their documents

Develop fronted using streamlit for asking user queries

Index the loaded data by using VectorStoreIndex

By using GoogleDriveReader

Load documents in text format

User have to add folder\_id of google drive