**Project Title: Intelligent Document Finder with Llama Index**

**Overview of system:**

This system enables user to query on documents of any kind (ex. PDF, PPT, Word Documents, etc) which are uploaded in Google Drive or OneDrive and user can expect response of query along with the metadata like file name, page number, author etc.

**Setup and Installation:**

* Create Virtual Environment with the help of virtualenv library(steps are given below):
  + Run the following commands for creating virtualenvironment:

1. Pip install virtualenv
2. virtualenv venv
3. venv\Scripts\activate

* Install required libraries which are mentioned in requirments.txt
* Create following directory inside root of project folder: cache\_model, chroma\_db, model\_cache, pipeline\_store
* Create GoogleAI API key from here: <https://aistudio.google.com/app/apikey>
* Store the GoogleAI API key in .env file
* To setup Google Drive integration follow below steps:

1. Create a Google Cloud Platform (GCP) Project:

* Go to the Google Cloud Console (<https://console.cloud.google.com/>).
* Create a new project or select an existing project.

1. Enable the Google Drive API:

* In the Cloud Console, navigate to the "APIs & Services" > "Library" section.
* Search for "Google Drive API" and click on it.
* Click the "Enable" button to enable the API for your project.

1. Set Up OAuth 2.0 Credentials:

* In the Cloud Console, navigate to the "APIs & Services" > "Credentials" section.
* Click on "Create credentials" and select "OAuth client ID."
* Select "Web application" as the application type.
* Add the authorized redirect URIs (e.g., <http://localhost:8080/auth/google/callback>).
* Click "Create" and note down the client ID and client secret.

1. Configure Google Drive API Access:

* In your Google Drive, create a folder where users will upload documents.
* Share the folder with the service account email address (e.g., example@your-project-id.iam.gserviceaccount.com) generated for your GCP project.

1. Get the credentials.json file and add it in project folder(here is sample structure of the file.)

{

"type": "service\_account",

"project\_id": "your\_project\_id",

"private\_key\_id": "your\_private\_key\_id",

"private\_key": "your\_private\_key",

"client\_email": "your\_client\_email",

"client\_id": "your\_client\_id",

"auth\_uri": "https://accounts.google.com/o/oauth2/auth",

"token\_uri": "https://accounts.google.com/o/oauth2/token",

"auth\_provider\_x509\_cert\_url": "https://www.googleapis.com/oauth2/v1/certs",

"client\_x509\_cert\_url": "your\_client\_x509\_cert\_url"

}

* To setup One Drive integration follow below steps:

1. Sign in to the Azure portal: Go to the [Azure portal](https://portal.azure.com/) and sign in with your Microsoft account.
2. Register an application: Once signed in, navigate to the "Microsoft Entra ID" service, and then search "App registrations".

1. Register a new application: Click on the "New registration" button. Enter a name for your application, and choose the appropriate Supported account types.

1. Configure the application: After registering, you'll be taken to your application's overview page. Here, note down the "Application (client) ID". You'll also need to configure a few more settings:

* Redirect URI: Under the "Authentication" section, add a redirect URI. For testing purposes.
* API permissions: Under the "API permissions" section, add the necessary permissions for accessing OneDrive in our case it is: Microsoft Graph --> Delegated Permission -- > Files.Read.All, Microsoft Graph --> Delegated Permission -- > Users.Read.All, Microsoft Graph --> Delegated Permission -- > Users.Read

1. Grant admin consent: if permissions require admin consent. We can do this by clicking on the "Grant admin consent" button under the "API permissions" section.
2. Get client secret: Under the "Certificates & secrets" section, create a new client secret. Note down the value of the client secret.
3. Sharing the OneDrive folder with app registration: share the one drive folder from which you want to fetch data to the service account email which is in structure like this: “client[id@tenant.onmicrosoft.com](mailto:id@tenant.onmicrosoft.com)”.
4. Now you can provide client id in OneDriveReader from llamahub to load data from particular folder.

**Module Information:**

app.py:

* This code sets up a streamlit interface for signup, login and providing google drive or one drive link and based on that querying a document and displaying the response along with metadata.

main.py:

* This file handles backend processing for SignUp, Login and Verifying user.

auth.py:

* API endpoints are defined in this file for SignUp, Login and Verifying user.

database\_con.py:

* This module sets up a SQLAlchemy engine to connect to a PostgreSQL database and creates a sessionmaker for managing database sessions.

models.py:

* This module defines a SQLAlchemy model class called Users that represents a table named 'users' in the database. It has three columns: id (integer, primary key), username (string, unique, not nullable), and hash\_password (string, indexed).

extract\_id\_from\_onedrive.py:

* This module provides a function, that extracts the folder ID from a OneDrive folder URL by parsing the query string. It returns the extracted folder ID, encoded to handle special characters, or None if the folder ID is not found in the URL.

load\_drive\_files.py:

* this file contains function for get files data from google drive.

load\_onedrive\_files.py:

* This module provides a function, that loads documents from a specified OneDrive folder using the OneDriveReader class. It utilizes the dotenv module to load the OneDrive client ID from the environment variables, ensuring secure authentication.

setup\_embedding\_model.py:

* In this file huggingface’s opensource model is defined to use it for embeddings.

setup\_chromadb.py:

* this file contains function to create chromadb vector database so we can use that to store vectors.

setup\_ingestion\_pipeline.py:

* this file contains function to create ingestion pipeline for processing of input data, including text splitting, embedding, and storing vectors and documents.

index\_creater.py:

* This module provides a function, that creates a new VectorStoreIndex object using a provided vector store and embedding model. The VectorStoreIndex is used for indexing and querying vectors in the vector store.

get\_gdrive\_fie\_ids.py:

* This Python module uses the Google Drive API to authenticate users and list files and folders in user’s Google Drive. It checks for existing credentials, handles authentication, and then recursively lists files and folders, collecting their IDs based on specific criteria such as visibility and parent folder.

data\_node\_ingestion\_gdrive.py:

* This module defines an ingestion function that loads data from Google Drive based on the provided fileIDs, creates or loads an ingestion pipeline, and then processes the documents using the pipeline.

data\_node\_ingestion\_onedrive.py:

* This module defines an ingestion function that loads data from One Drive, creates or loads an ingestion pipeline, and then processes the documents using the pipeline.

**System Usage:**

Once your done with above setup and installation process then run main.py file to start backend server so signup and login and other functionality work properly. After that run app.py file with this command: “streamlit run app.py” and now you can see interface and in that you can select option from selectbox for SignUp, login, and search and you can use the app. Signup first if you are new user and then login and then select search from selectbox and now you can connect your google drive or onedrive with the system by clicking on appropriate button.once it display success message then query whatever you want based on files you have provided.

**Authentication working with the system:**

* Signup form takes two input username and password and in username user should provide his/her emailId and it do hashing on password and then username and hashed password will be stored in database.
* Login form takes two input same as signup and at backend provided username and password will be matched for authentication if it will successful then JWT encoded token will be generated.
* When user will provide google drive folder ID at that time encoded JWT token and Folder ID both will be send to server then at the backend emaiID will be extracted from folderID and token will be decoded and from token username(which in our case is emailid) will be fetched and it will be compared with emailID fetched from google drive folder ID and if it matches successfully then data will be fetched from user provided folder and now user can query.

**Resources:**

* for fetching files from Google drive:

1. <https://llamahub.ai/l/readers/llama-index-readers-google?from=readers>
2. <https://docs.llamaindex.ai/en/stable/examples/ingestion/ingestion_gdrive.html>

* for creation of ChromaDB:

1. <https://docs.llamaindex.ai/en/stable/examples/vector_stores/ChromaIndexDemo.html>

* for implementing ingestion pipeline:

1. <https://docs.llamaindex.ai/en/stable/examples/ingestion/ingestion_gdrive.html>

* for integrating streamlit interface:

1. <https://docs.streamlit.io/>

* for integrating onedrive interface:

1. <https://learn.microsoft.com/en-us/entra/fundamentals/create-new-tenant>
2. <https://llamahub.ai/l/readers/llama-index-readers-microsoft-onedrive?from=readers>
3. https://learn.microsoft.com/en-us/answers/questions/1180698/unauthorized-client-the-client-does-not-exist-or-i