Google Drive integretion

Deadline	@July 29, 2025
Status	Done

📂 Project Report: Intelligent Document Question Answering System using Google Drive, LangChain, Qdrant, and OpenAI

✅ Overview

This project implements a fully automated Retrieval-Augmented Generation (RAG) pipeline that integrates:

- Google Drive for document storage
- LangChain for orchestration
- Qdrant as the vector database
- OpenAI (GPT-3.5) as the LLM

The system retrieves documents from Google Drive, chunks and embeds the content, stores it in Qdrant, and enables natural language question answering through a LangGraph-driven RAG workflow.

🔧 Tools & Technologies

Component	Technology
File Storage	Google Drive API
Document Parsing	Python (Text I/O, Google API)
Vector Store	Qdrant (local Docker instance)
Embeddings	OpenAIEmbeddings via LangChain

Component	Technology
LLM	ChatOpenAI (GPT-3.5-turbo)
RAG Graph Flow	LangGraph
Chunking	RecursiveCharacterTextSplitter

🔐 Step 1: Google Drive Authentication & File Listing

```
from googleapiclient.discovery import build
from google_auth_oauthlib.flow import InstalledAppFlow
from google.auth.transport.requests import Request
import pickle, os

SCOPES = ['https://www.googleapis.com/auth/drive.readonly']

def authenticate_google_drive():
    creds = None
    if os.path.exists('token.pkl'):
        with open('token.pkl', 'rb') as token:
            creds = pickle.load(token)

    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file('a.json', SCOPES)
            creds = flow.run_local_server(port=0)
        with open('token.pkl', 'wb') as token:
            pickle.dump(creds, token)

    return build('drive', 'v3', credentials=creds)
```

✅ Functionality:

- Authenticates Google user

- Reads `a.json` (OAuth client ID)
- Caches token in `token.pkl`

List Files:

```
def list_files(service):
    results = service.files().list(
        pageSize=10,
        fields="nextPageToken, files(id, name, mimeType)"
    ).execute()
    return results.get('files', [])
```

📄 Step 2: Read Local `.txt` File

```
def load_file(file_path):
    with open(file_path, 'r') as f:
        return f.read()
```

🔷 Step 3: Document Chunking

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

def chunk_text(document):
    splitter = RecursiveCharacterTextSplitter(
        chunk_size=1000,
        chunk_overlap=200
    )
    return splitter.create_documents([document])
```

💿 Step 4: Qdrant Vector Store Setup

```
from langchain.vectorstores import Qdrant
from qdrant_client import QdrantClient
from langchain.embeddings import OpenAIEmbeddings

qclient = QdrantClient(host="localhost", port=6333)
qclient.recreate_collection(
    collection_name="rag_docs",
    vectors_config=models.VectorParams(size=1536, distance=models.Distanc
e.COSINE)
)

vector_store = Qdrant.from_documents(
    documents=chunked_docs,
    embedding=OpenAIEmbeddings(),
    location="http://localhost:6333",
    collection_name="rag_docs",
    api_key=None
)
```

✅ Notes:

- Connects to local Qdrant instance (ensure Docker is running)
- Clears and re-creates collection "rag_docs"

🧰 Step 5: Define RAG Graph with LangGraph

```
from langgraph.graph import StateGraph, END
from langchain_core.runnables import RunnableLambda
from langchain.chains.combine_documents import create_stuff_documents_c
hain
from langchain.chains import create_retrieval_chain
from langchain.chat_models import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
```

```
retriever = vector_store.as_retriever()

prompt = ChatPromptTemplate.from_template("""
Answer the question based only on the following context:

<context>
{context}
</context>

Question: {input}
""")

docs_chain = create_stuff_documents_chain(
    llm=ChatOpenAI(model='gpt-3.5-turbo', temperature=0.3), prompt=prompt
)
rag_chain = create_retrieval_chain(retriever, docs_chain)

def retrieve_context(state):
    return {"context": retriever.invoke(state["query_text"]), **state}

def generate_answer(state):
    return {"answer": docs_chain.invoke(state), **state}

graph = StateGraph()
graph.add_node("context_retriever", RunnableLambda(retrieve_context))
graph.add_node("answer_generator", RunnableLambda(generate_answer))

graph.set_entry_point("context_retriever")
graph.add_edge("context_retriever", "answer_generator")
graph.set_finish_point("answer_generator")

app = graph.compile()
```

🚀 Step 6: Run the Full Chat Application

```
response = app.invoke({"query_text": "How can I change my shipping addres
s?"})
print(response["answer"])
```

🔢 Sample Output

```
You can change your shipping address in the account settings under the 'Deli
very Preferences' section. Ensure the address is serviceable within our curren
t delivery zones.
```

📅 Directory Structure

```
Google_Drive_Qdrant_RAG
├── a.json            # Google OAuth Client Credentials
├── token.pkl         # Saved access token
├── DOCUNMENT.txt     # Sample local document
├── main.py           # Complete pipeline script
└── qdrant_config.yaml  # Optional Docker config
```

📝 Future Improvements

- [ ] Add PDF/Google Docs parser and converter
- [ ] Deploy on FastAPI with Web UI
- [ ] Implement document metadata filtering
- [ ] Switch to hybrid search (semantic + keyword)
- [ ] Add user authentication for chat UI

🌟 Conclusion

This system offers an end-to-end intelligent pipeline that combines document storage, chunking, semantic search, and LLM-based generation into a unified Q&A platform.

It can be adapted for enterprise search, knowledge management, legal document queries, and customer support bots.