RAG System with Email Integration Using LangGraph

Overview

This project illustrates the creation of a **Retrieval-Augmented Generation (RAG)** pipeline with **LangChain** and **LangGraph** alongside **Qdrant** as the vector store with **email delivery** integration through `yagmail`. The architecture of the system begins with the user submitting a query and, with the help of OpenAI's language model, the system fetches the pertinent document, constructs a response, and emails the answer.

Libraries Used:

Library	Purpose
`dotenv`	To load environment variables (email credentials).
`langchain_core.documents.Document`	Represents documents to be stored and queried.
`langchain_community.vectorstores.Qdrant`	Interface for the Qdrant vector store.
`langchain.text_splitter.RecursiveCharacterTextSplitter`	Splits large documents into manageable chunks.
`qdrant_client`	Python client for connecting to a running Qdrant instance.
`langgraph.graph.StateGraph`	Defines the LangGraph workflow structure.
`langchain_openai`	For embeddings and language model (ChatOpenAI).
`yagmail`	Sends emails through Gmail using credentials from environment variables.
`os`	To interact with the file system and environment variables.

Step-by-Step Breakdown

Step 1: **Environment Setup**

```
from dotenv import load_dotenv
load_dotenv()
```

- Loads Gmail credentials (`EMAIL_USER`, `EMAIL_PASSWORD`) from a `.env` file for secure access.

Step 2: **Load & Preprocess Text File**

```
def load_txt_as_documents(txt_file):
    with open(txt_file, 'r', encoding='utf-8') as f:
        raw_text = f.read()
    return raw_text
```

- Reads text from a `.txt` file.
- Used to load the knowledge base for retrieval.

```
raw_text = load_txt_as_documents("rag_service.txt")
text_splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overla
p=200)
texts = text_splitter.split_text(raw_text)
documents = [Document(page_content=chunk) for chunk in texts]
```

- The document is split into **chunks of 1000 characters**, with **200-character overlap** for better context preservation.

Step 3: **Initialize Embedding Model**

```
embedding_function = OpenAIEmbeddings()
```

- Converts text into **vector embeddings** using OpenAI's API.
- Each vector has **1536 dimensions**.

Step 4: **Setup Qdrant Vector Store**

```
qdrant_client = QdrantClient(host="localhost", port=6333)
qdrant_client.recreate_collection(
    collection_name="rag_txt_collection",
    vectors_config=VectorParams(size=1536, distance=Distance.COSINE),
)
```

- Connects to a **running Qdrant Docker container**.
- Creates (or recreates) a collection named `"rag_txt_collection"` with cosine similarity metric.

```
db = Qdrant(
    client=qdrant_client,
    collection_name="rag_txt_collection",
    embeddings=embedding_function
)
db.add_documents(documents)
```

- Uploads all document chunks with their embeddings into Qdrant.

Step 5: **LangGraph State Definition**

```
from typing import TypedDict

class GraphState(TypedDict):
    question: str
    context: str
    answer: str
    recipient: str
```

- Defines the **shared state** between all LangGraph nodes (query, context, answer, recipient).

Step 6: **Retrieve Node**

```
def retrieve(state: GraphState):
    query = state["question"]
    retriever = db.as_retriever()
    docs = retriever.invoke(query)
    context = "\n\n".join([doc.page_content for doc in docs])
    return {"question": query, "context": context}
```

- Retrieves the most relevant chunks from Qdrant based on cosine similarity to the query.

Step 7: **Generate Node (LLM Answering)**

```
llm = ChatOpenAI(model="gpt-3.5-turbo")

def generate(state: GraphState):
    prompt = f"""Answer the question using this context:\n\n{state['context']}\n
\nQuestion: {state['question']}"""
    response = llm.invoke(prompt)
    return {
        "question": state["question"],
        "context": state["context"],
        "answer": response.content
    }
```

- Forms a prompt by combining context and the user query.
- Uses GPT-3.5 Turbo to generate an answer.

Step 8: **Send Email Node**

```
def send_email(state: GraphState):
    recipient = state.get("recipient")
    subject = f"Response to your query: {state['question'][:50]}"
    body = state["answer"]
    try:
        yag = yagmail.SMTP(user=os.getenv("EMAIL_USER"), password=os.gete
```

```
nv("EMAIL_PASSWORD"))
        yag.send(to=recipient, subject=subject, contents=body)
        print(f"Email sent to {recipient}")
    except Exception as e:
        print(f"Failed to send email: {e}")
    return state
```

- Uses `yagmail` to send the answer as an email.
- Authenticates using credentials from `.env`.

Step 9: **LangGraph Setup**

```
graph = StateGraph(GraphState)
graph.add_node("retrieve", RunnableLambda(retrieve))
graph.add_node("generate", RunnableLambda(generate))
graph.add_node("send_email", RunnableLambda(send_email))

graph.set_entry_point("retrieve")
graph.add_edge("retrieve", "generate")
graph.add_edge("generate", "send_email")
graph.add_edge("send_email", END)

app = graph.compile()
```

- Constructs a **LangGraph pipeline**:
    1. `retrieve` → 2. `generate` → 3. `send_email` → 4. `END`.

Step 10: **Run the Pipeline**

```
inputs = {
    "question": "I have an issue setting a different delivery address up",
    "recipient": "shahzain0066@gmail.com"
}
```

```
result = app.invoke(inputs)

print("Final Answer:", result["answer"])
```

- Sends a query + recipient email.
- The system retrieves, answers, and emails the reply.

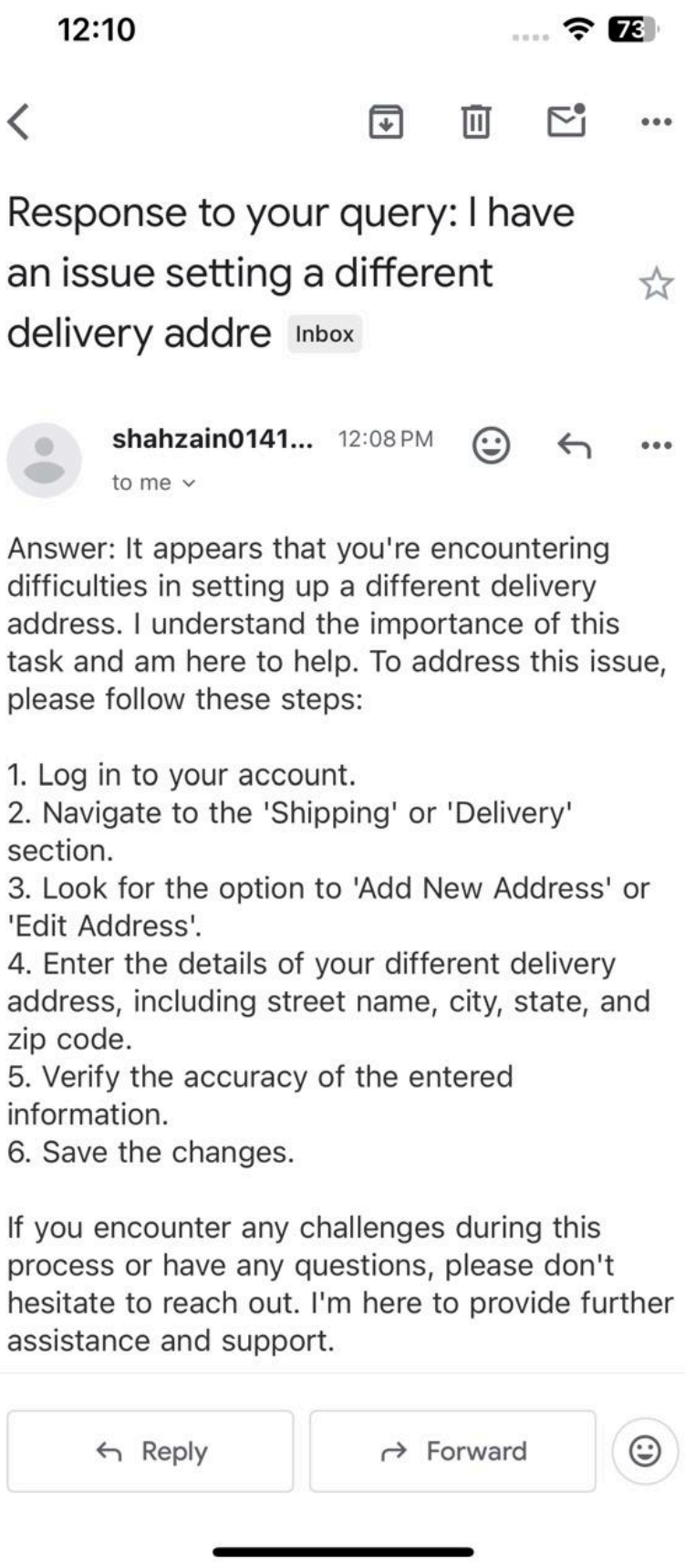
✅ Final Output

1. Documents uploaded to Qdrant.
2. Query processed.
3. Answer generated by GPT.
4. Email successfully sent to the recipient.

```
Uploaded documents to running Docker Qdrant.
Email sent to shahzain0066@gmail.com
Final Answer: Answer: It seems like you're encountering difficulties when trying to set up a different delivery address. I un
derstand the importance of this task and am here to assist you. To resolve this issue, please follow these steps:

1. Log in to your account.
2. Go to the 'Shipping' or 'Delivery' section.
3. Look for the option to 'Add New Address' or 'Edit Address'.
4. Enter the details of the different delivery address you want to set up.
5. Verify the accuracy of the entered information.
6. Save the changes.

If you encounter any obstacles during this process or have any further questions, please don't hesitate to reach out. I'm her
e to help you with setting up your new delivery address.
```


12:10
Response to your query: I have an issue setting a different delivery addre
Inbox
shahzain0141...
12:08 PM
to me
Answer: It appears that you're encountering difficulties in setting up a different delivery address. I understand the importance of this task and am here to help. To address this issue, please follow these steps:
1. Log in to your account.
2. Navigate to the 'Shipping' or 'Delivery' section.
3. Look for the option to 'Add New Address' or 'Edit Address'.
4. Enter the details of your different delivery address, including street name, city, state, and zip code.
5. Verify the accuracy of the entered information.
6. Save the changes.
If you encounter any challenges during this process or have any questions, please don't hesitate to reach out. I'm here to provide further assistance and support.
Reply
Forward