Email Integartaion Using LangGraph

Deadline	@July 25, 2025
Status	Done

Project Overview

This modular system combines:

- LangChain (for RAG)
- Qdrant (vector storage)
- OpenAI embeddings
- SMTP-based email automation
- LangGraph (to orchestrate flow)

It performs RAG on a document, answers user queries, and sends responses via email.

Folder Structure

```
bash
CopyEdit
INTEGRATE EMAIL/
│
├── config.py          # Env vars and Qdrant client setup
├── embedder.py        # Embedding model setup
├── emil.py            # Email sending logic
├── graph_flow.py      # LangGraph pipeline definition
├── loader.py          # Load text file as string
├── main.py            # Main orchestration script
├── splitter.py        # Text splitter using LangChain
```

```
├── vectorstore.py        # Qdrant vectorstore init
├── .env                  # Environment secrets (EMAIL_USER, PASSWORD)
└── DOCUNMENT.txt         # Source document to be queried
```

Setup Instructions

1. Install Dependencies

```
bash
CopyEdit
pip install langchain langgraph openai python-dotenv qdrant-client
```

2. Set Up `.env`

```
env
CopyEdit
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_app_password  # Gmail app password, not regular p
assword
```

Enable 2FA on Gmail and create an **App Password** at:

👉 https://myaccount.google.com/apppasswords

Module Descriptions

`config.py`

Handles environment loading and Qdrant setup.

```
python
CopyEdit
```

```
import os
from dotenv import load_dotenv
from qdrant_client import QdrantClient

load_dotenv()

QDRANT_HOST = "localhost"
QDRANT_PORT = 6333
COLLECTION_NAME = "DOCUNMENT.txt"

EMAIL_USER = os.getenv("EMAIL_USER")
EMAIL_PASSWORD = os.getenv("EMAIL_PASSWORD")

qdrant_client = QdrantClient(host=QDRANT_HOST, port=QDRANT_PORT)
```

---

embedder.py

Returns the OpenAI embedding model.

```
python
CopyEdit
from langchain_openai import OpenAIEmbeddings

def get_embedder():
    return OpenAIEmbeddings()
```

---

emil.py

Sends an email using SMTP.

```
python
CopyEdit
import smtplib
```

```
from email.mime.text import MIMEText
from config import EMAIL_USER, EMAIL_PASSWORD

def send_email(to, subject, body):
    msg = MIMEText(body)
    msg["Subject"] = subject
    msg["From"] = EMAIL_USER
    msg["To"] = to

    try:
        with smtplib.SMTP("smtp.gmail.com", 587) as smtp:
            smtp.starttls()
            smtp.login(EMAIL_USER, EMAIL_PASSWORD)
            smtp.send_message(msg)
            print("✅ Email sent successfully!")
    except Exception as e:
        print(f"❌ Failed to send email: {e}")
```

`graph_flow.py`

Builds the LangGraph pipeline.

```
python
CopyEdit
from typing import TypedDict
from langchain_openai import ChatOpenAI
from langgraph.graph import StateGraph, END
from langchain_core.runnables import RunnableLambda
from emil import send_email

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

```
    recipient: str

def build_graph(db) → StateGraph:
    llm = ChatOpenAI(model="gpt-3.5-turbo")

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

    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,
            "recipient": state["recipient"]
        }

    def email_node(state: GraphState):
        subject = f"Response to your query: {state['question'][:50]}"
        body = state["answer"]
        if state.get("recipient"):
            send_email(state["recipient"], subject, body)
        return state

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

```
    graph.add_node("send_email", RunnableLambda(email_node))

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

    return graph.compile()
```

---

loader.py

Reads plain text from file.

```
python
CopyEdit
def load_txt_as_string(txt_path: str) → str:
    with open(txt_path, 'r', encoding='utf-8') as f:
        return f.read()
```

---

splitter.py

Splits large text into chunks.

```
python
CopyEdit
from langchain.text_splitter import RecursiveCharacterTextSplitter

def split_text_to_chunks(text: str, chunk_size=1000, chunk_overlap=200):
    splitter = RecursiveCharacterTextSplitter(chunk_size=chunk_size, chunk_ov
erlap=chunk_overlap)
    return splitter.split_text(text)
```

---

vectorstore.py

Creates Qdrant vector store and adds embedded chunks.

```
python
CopyEdit
from langchain_core.documents import Document
from langchain_community.vectorstores import Qdrant
from qdrant_client.http.models import VectorParams, Distance
from config import qdrant_client, COLLECTION_NAME
from embedder import get_embedder

def create_qdrant_vectorstore(chunks: list) → Qdrant:
    qdrant_client.recreate_collection(
        collection_name=COLLECTION_NAME,
        vectors_config=VectorParams(size=1536, distance=Distance.COSINE),
    )

    documents = [Document(page_content=chunk) for chunk in chunks]
    embedder = get_embedder()
    db = Qdrant(
        client=qdrant_client,
        collection_name=COLLECTION_NAME,
        embeddings=embedder
    )
    db.add_documents(documents)
    return db
```

---

main.py

The main script that drives the system.

```
python
CopyEdit
from loader import load_txt_as_string
```

```
from splitter import split_text_to_chunks
from vectorstore import create_qdrant_vectorstore
from graph_flow import build_graph

if __name__ == "__main__":
    file_path = r"C:\Users\hp\Documents\INTEGRATE EMAIL\DOCUNMENT.txt"

    raw_text = load_txt_as_string(file_path)
    chunks = split_text_to_chunks(raw_text)
    print(f"✅ Loaded and split {len(chunks)} text chunks.")

    db = create_qdrant_vectorstore(chunks)
    print("📤 Uploaded chunks to Qdrant.")

    graph_app = build_graph(db)

    inputs = {
        "question": "how to signup",
        "recipient": "shahzain0141@gmail.com"
    }

    result = graph_app.invoke(inputs)

    print("\n📩 Final Answer:")
    print(result["answer"])
```

🧪 Output Example

```
bash
CopyEdit
✅ Loaded and split 7 text chunks.
📤 Uploaded chunks to Qdrant.
✅ Email sent successfully!
```

📩 Final Answer:
To sign up, follow these steps...