# Auto Email Responder with LangGraph + Qdrant + GPT

A **Retrieval-Augmented Generation (RAG)** system that reads unseen emails, processes them through LangGraph and Qdrant, generates contextual replies using OpenAI's GPT, and sends automated email responses — **in real time**.

## Libraries Used

| Library | Purpose |
|---|---|
| `dotenv` | Loads Gmail credentials securely from `.env` |
| `langchain_core.documents.Document` | Wraps chunks of documents for ingestion |
| `langchain_community.vectorstores.Qdrant` | Interfaces with Qdrant vector database |
| `langchain.text_splitter.RecursiveCharacterTextSplitter` | Splits large text into overlapping chunks |
| `qdrant_client` | Python client to communicate with Qdrant server |
| `qdrant_client.http.models.VectorParams, Distance` | Defines vector size and similarity metric |
| `langgraph.graph.StateGraph` | Builds and controls the LangGraph workflow |
| `langchain_core.runnables.RunnableLambda` | Wraps callable functions into LangGraph nodes |
| `langchain_openai.ChatOpenAI` | Accesses OpenAI LLM (GPT-3.5-turbo) for response generation |
| `langchain_openai.OpenAIEmbeddings` | Converts text into vector embeddings |
| `yagmail` | Sends emails through Gmail using OAuth or App Password |
| `os` | Fetches system-level environment variables |
| `imaplib` | Connects to Gmail and reads emails via IMAP |

| Library | Purpose |
|---|---|
| `email` | Parses raw email messages and headers |
| `re` | Extracts sender's email address using regex |

## System Behavior

- Connects to Gmail and reads **unseen** messages.
- Retrieves context from documents stored in **Qdrant**.
- Generates a **friendly and helpful** response using **GPT-3.5 Turbo**.
- Sends the response **back via email**.
- Repeats this process **every 60 seconds**.

## Step-by-Step Breakdown

### Step 1: Load Environment Variables

```
python
CopyEdit
from dotenv import load_dotenv
load_dotenv()
```

- Loads `EMAIL_USER` and `EMAIL_PASSWORD` from `.env`.

### Step 2: Load and Split Text File

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

```
python
CopyEdit
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]
```

- Loads your RAG knowledge base (e.g., FAQs or support content).
- Chunks it into overlapping pieces to preserve context.

## Step 3: Convert Text to Embeddings

```
python
CopyEdit
embedding_function = OpenAIEmbeddings()
```

- Converts each document chunk into a **1536-dimensional vector** using OpenAI's embedding model.

## Step 4: Initialize Qdrant

```
python
CopyEdit
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 your local Qdrant instance running in Docker.
- Recreates a collection with cosine similarity.

### Step 5: Upload Documents to Vector Store

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

- Stores all chunks and their embeddings in Qdrant.

### Step 6: Define Shared State for LangGraph

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

- This shared `state` is passed across each LangGraph node.

### Step 7: Read Inbox Node

```
python
CopyEdit
def read_inbox(state: GraphState) → GraphState:
    ...
```

- Uses `imaplib` to access unseen emails in Gmail.
- Parses sender's email and subject using `email` and `decode_header`.
- Returns extracted email content, subject, and sender email.

## Step 8: Retrieve Node

```
python
CopyEdit
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,
        ...
    }
```

- Retrieves top relevant chunks from Qdrant based on query similarity.

## Step 9: Generate Node (GPT Answering)

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

def generate(state: GraphState):
    prompt = f"""..."""
    response = llm.invoke(prompt)
    return {
        "answer": response.content,
        ...
    }
```

- Uses GPT-3.5 to generate a helpful, polite reply to the question using the retrieved context.

## Step 10: Send Email Node

```
python
CopyEdit
def send_email(state: GraphState):
    yag = yagmail.SMTP(user=os.getenv("EMAIL_USER"), password=os.getenv
("EMAIL_PASSWORD"))
    yag.send(to=recipient, subject=subject, contents=body)
```

- Uses `yagmail` to send the LLM's response as a reply email to the original sender.

## Step 11: Define LangGraph Workflow

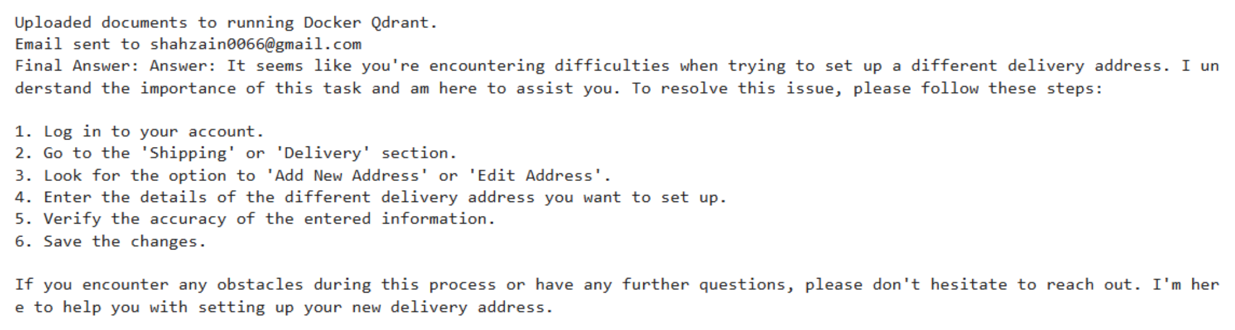
```
python
CopyEdit
graph = StateGraph(GraphState)
graph.add_node("read_inbox", RunnableLambda(read_inbox))
graph.add_node("retrieve", RunnableLambda(retrieve))
graph.add_node("generate", RunnableLambda(generate))
graph.add_node("send_email", RunnableLambda(send_email))

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

app = graph.compile()
```

- Constructs a LangGraph DAG (Directed Acyclic Graph) with 4 key steps:
  1. `read_inbox`
  2. `retrieve`
  3. `generate`
  4. `send_email`

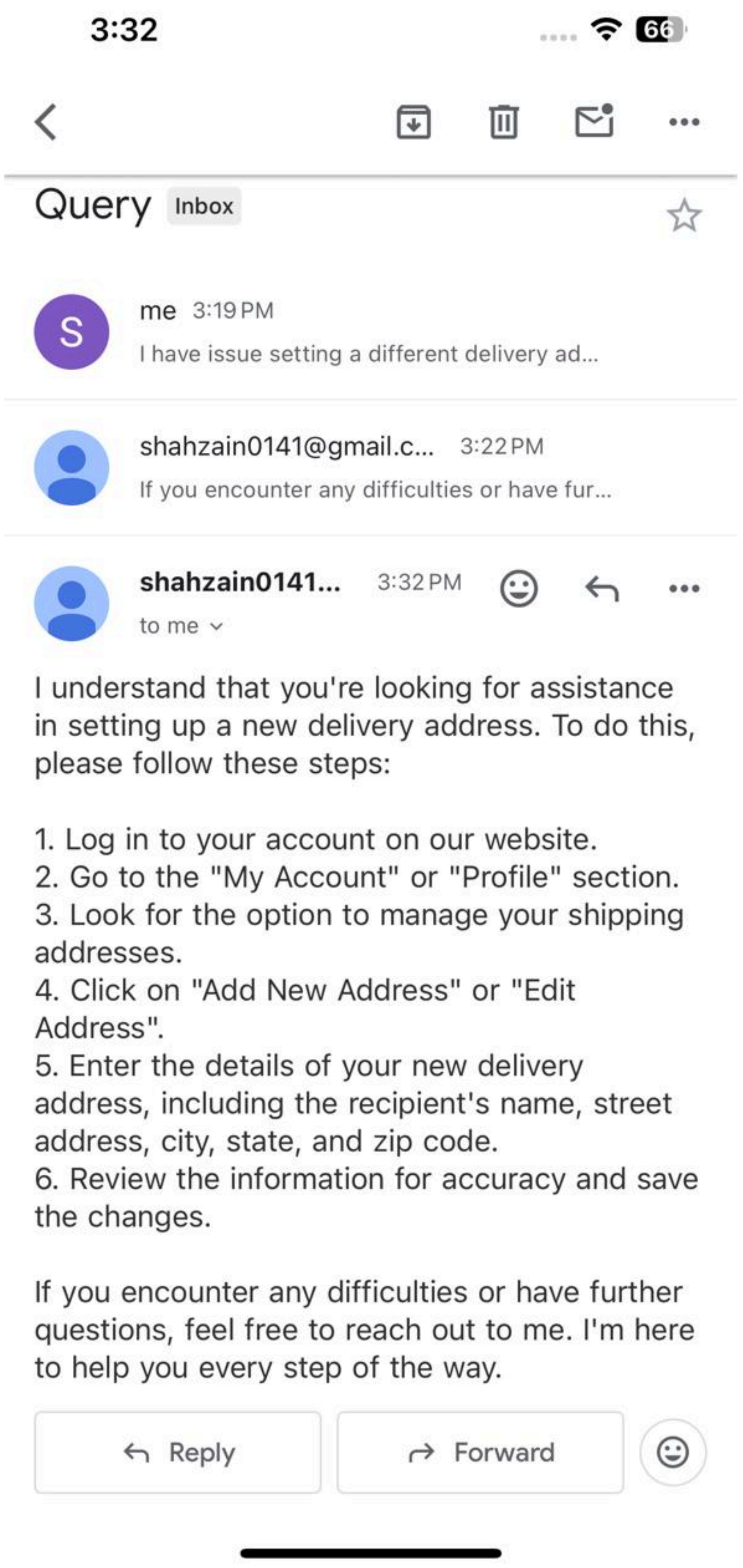
**Output:**



```
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.
```



me 3:19 PM
I have issue setting a different delivery ad...

shahzain0141@gmail.c... 3:22 PM
If you encounter any difficulties or have fur...

**shahzain0141...** 3:32 PM
to me

I understand that you're looking for assistance in setting up a new delivery address. To do this, please follow these steps:

1. Log in to your account on our website.
2. Go to the "My Account" or "Profile" section.
3. Look for the option to manage your shipping addresses.
4. Click on "Add New Address" or "Edit Address".
5. Enter the details of your new delivery address, including the recipient's name, street address, city, state, and zip code.
6. Review the information for accuracy and save the changes.

If you encounter any difficulties or have further questions, feel free to reach out to me. I'm here to help you every step of the way.