Project Specification: Autonomous Email Customer Agent

Assignment Type: Individual Project

Estimated Time: 10 Hours

Core Technologies: Python, LangGraph, Vector Database, LLM (OpenAI/Gemini), FastAPI, Gmail API, Docker

1. Project Goal

The objective of this project is to build an autonomous agent that can handle customer support emails. You will create a production-style application that reads incoming emails from a Gmail inbox, uses a Retrieval-Augmented Generation (RAG) system to find relevant information from a policy document, drafts a reply, validates its own response for accuracy and safety, and sends the email. The entire workflow will be orchestrated using LangGraph.

2. Learning Objectives

Upon successful completion of this project, you will be able to:

- **Design Agentic Workflows:** Implement a multi-step agent using LangGraph, including conditional logic and loops.
- **Build a RAG System:** Ingest a document, store it in a vector database, and perform semantic searches to retrieve relevant context.
- **Integrate Real-World APIs:** Securely connect to the Gmail API using OAuth to read and send emails.
- **Implement Autonomous Logic:** Create an autonomous draft -> validate -> rewrite loop to improve response quality before sending.
- **Apply AI Safety:** Integrate guardrails to check for policy compliance and redact Personally Identifiable Information (PII).
- **Build and Deploy an Application:** Wrap the agent in a FastAPI backend with a simple web interface for monitoring and containerize it with Docker.
- **Ensure Transparency:** Generate explainability logs that trace the agent's decision-making process.

2.1. Test scenarios

Here are business-centric test cases restricted entirely to the scenarios and policies available in the airlines policy document.

Each test case includes the **Source of Truth** from the airlines_policy.md (FAQ) that the agent must use to formulate its response.

Category 1: Standard Policy Questions

These scenarios test the agent's ability to find and correctly relay information that is explicitly stated in the policy document.

Test Case ID	Scenario	Input Email Body	Source of Truth (from airlines_policy.md)	Expected Agent Action	Key Elements of Expected Response
BIZ-S-01	**Baggage by Fare**	"Hi, I've booked an Economy Classic ticket. How many checked bags can I bring?"	## Baggage Allowance -> Economy Classic: 1 piece up to 23 kg	**Send Reply**	1. States that an Economy Classic ticket includes one checked bag up to 23 kg.
BIZ-S-02	**Hand Luggage**	"Hello, can I bring my guitar on the plane with me?"	## Hand Baggage -> Musical instruments… can be taken into the cabin… if they do not exceed the hand baggage dimensions.	**Send Reply**	1. Informs that a guitar can be taken as hand baggage if it meets the standard dimensions (55x40x23 cm).
BIZ-S-03	**Special Meals**	"I need to order a vegetarian meal for my upcoming flight. How do I do that?"	## Meals and Drinks -> Special meals can be ordered free of charge up to 24 hours before departure.	**Send Reply**	1. Advises that a vegetarian meal can be ordered for free. 2. Specifies it must be done at least 24 hours before the flight.
BIZ-S-04	**Unaccompanied Child**	"I need to book a flight for my 10-year-old daughter to fly	## Unaccompanied Minors -> Our service for unaccompanied minors is mandatory for children aged 5-11 travelling alone.	**Send Reply**	1. Confirms that a 10-year-old can travel alone using the mandatory unaccompanied minor service.

Test Case ID	Scenario	Input Email Body	Source of Truth (from airlines_policy.md)	Expected Agent Action	Key Elements of Expected Response
		alone. Is this possible?"			

Category 2: Policy Denials & Boundaries

These scenarios test the agent's ability to identify requests that violate or fall outside the stated policy, and to politely deny them based on the FAQ.

Test Case ID	Scenario	Input Email Body	Source of Truth (from airlines_policy.md)	Expected Agent Action	Key Elements of Expected Response
BIZ-D-01	**No Free Bag**	"I'm flying Economy Light. I assume I get one free checked bag?"	## Baggage Allowance -> Economy Light: No free checked baggage allowance.	**Send Reply**	1. Politely clarifies that the Economy Light fare does not include a free checked bag. 2. Mentions that baggage can be added for a fee.
BIZ-D-02	**Refund Denial**	"I booked an Economy Light flight but my plans have changed. Please cancel and refund my ticket."	## Fare Conditions -> Economy Light: Non-refundable.	**Send Reply**	1. States that according to the fare conditions, Economy Light tickets are non-refundable. 2. Does **not** offer a refund.
BIZ-D-03	**Pet Too Large**	"Can I bring my golden retriever in the cabin with me? He is very well-behaved and weighs 30 kg."	## Travelling with Pets -> Small dogs and cats up to 8 kg (including carrier) may travel in the cabin.	**Send Reply**	1. Explains that only small pets up to 8 kg (including the carrier) are permitted in the cabin. 2. Suggests the option of transporting the pet in the cargo hold.

Test Case ID	Scenario	Input Email Body	Source of Truth (from airlines_policy.md)	Expected Agent Action	Key Elements of Expected Response
BIZ-D-04	**Child Too Young**	"I want my 4-year-old to fly solo to visit his grandparents."	## Unaccompanied Minors -> ...mandatory for children aged 5-11 travelling alone.	**Send Reply**	1. Informs the customer that the service for children travelling alone is only available for those aged 5 and older.

Category 3: Escalations & Safety

These scenarios test the agent's ability to recognize when a query is outside its knowledge base (the FAQ) or when it contains sensitive information, requiring escalation to a human.

Test Case ID	Scenario	Input Email Body	Source of Truth (from airlines_policy.md)	Expected Agent Action	Reason for Action / Expected Behavior
BIZ-E-01	**Out-of-Scope**	"My flight was cancelled. How do I claim compensation under EU261 regulations?"	**(Not in FAQ)** The document has no information on flight cancellations or compensation regulations.	**Escalate**	The agent must recognize that its knowledge base does not cover compensation claims and hand off to a human agent.
BIZ-E-02	**PII Handling**	"Can you check the price for an extra bag? My booking code is X1Y2Z3 and my credit card is 4444..."	**(Not in FAQ)** THE AIRLINES_POLICY.MD (FAQ) has pricing, but not a facility to check specific bookings.	**Send Reply & Escalate**	1. **Response:** Provides the general cost of an extra bag from the FAQ. 2. **Behavior:** The reply must **not** include the PII. The system should flag the PII and escalate for a human to ensure the booking is okay without risk.
BIZ-E-03	**Ambiguous**	"The website is broken. I	**(Not in FAQ)** THE AIRLINES_POLICY.MD (FAQ) explains *when* to check in, but not how to	**Escalate**	The query is too vague and relates to a technical issue outside the FAQ's

Test Case ID	Scenario	Input Email Body	Source of Truth (from airlines_policy.md)	Expected Agent Action	Reason for Action / Expected Behavior
		can't check in."	troubleshoot website errors.		scope. It requires human-led troubleshooting.

3. Technical Stack

COMPONENT	REQUIRED TECHNOLOGY	NOTES
ORCHESTRATION	LangGraph	The core framework for building the agent.
LANGUAGE	Python	
BACKEND API	FastAPI	To serve the application and expose monitoring endpoints.
WEB UI	Simple HTML/CSS/JavaScript or Streamlit	For monitoring the agent's activity. A polished UI is a plus.
VECTOR DATABASE	Your choice (e.g., ChromaDB, FAISS, Weaviate)	Must be abstracted behind an interface to remain pluggable.
LLM & EMBEDDINGS	OpenAI (e.g., gpt-4o, text-embedding-3-small) or Google Gemini	Your application should be configurable to switch between providers.
EMAIL INTEGRATION	Gmail API with OAuth 2.0	For reading from and sending to a real inbox.
SAFETY GUARDRAILS	LLMGuard (or a custom implementation of its principles)	For validating LLM outputs.
CONTAINERIZATION	Docker	A Dockerfile and run scripts are required.
STATE PERSISTENCE	LangGraph MemorySaver	To persist the agent's state in-memory across requests within a single run.

---

4. High-Level Flow

Your agent will follow a stateful, cyclical graph structure. An incoming email triggers the graph, which then moves through stages of retrieval, drafting, validation, and sending.

This is given as indicative start so that you can get some inspiration. Feel free to improve and make it better.

1. **Email Retrieval**: The system starts by fetching new emails from a Gmail inbox with the 'agent_inbox' label.
2. **Context Retrieval**: The agent retrieves relevant context from a Vector Database to inform the response.
3. **Draft Generation**: Using an LLM, the system drafts a reply based on the email content and retrieved context.
4. **Validation**: The drafted response goes through guardrail validation to ensure quality and appropriateness.
5. **Outcome Handling**:
   - If validation passes: The reply is sent via Gmail API
   - If validation fails with <3 retries: The system rewrites the draft
   - If validation fails with ≥3 retries: The email is escalated for manual review
6. **Logging**: Each critical step is logged for monitoring and debugging purposes.
7. **Completion**: Processed emails are marked as completed to prevent reprocessing.

The system includes a retry mechanism (up to 3 attempts) for response generation and fails safely to manual review when automated processing isn't successful.

5. What is given below is just some pseudo code. Do your own design

Step 1: Gmail API Integration

Your agent needs to interact with a real inbox.

1. **Setup Google Cloud Project:** Enable the Gmail API and create OAuth 2.0 credentials. Store the credentials.json file securely.
2. **Authenticate:** Write a utility function that handles the OAuth flow to get user consent and store tokens. The google-auth-oauthlib library is excellent for this.
3. **Implement Email Functions:**
   - read_inbox(): A function that scans for unread emails with a specific label (e.g., agent_inbox).
   - send_reply(to, subject, body): A function that sends an email using the authenticated client.
   - mark_as_processed(email_id): A function that removes the agent_inbox label and adds a processed_by_agent label.

4. **Safety First:** Hardcode checks to ensure the agent **only** replies to the original sender and **only** processes emails with the agent_inbox label.

Step 2: Data Ingestion and RAG Setup

Your agent's knowledge will come from a provided THE AIRLINES_POLICY.MD (FAQ) document. You must create a script to process this document and load it into your chosen vector database.

1. **Fetch the Data:** Download the Airlines THE AIRLINES_POLICY.MD (FAQ) markdown file.

Python

```
import requests

import re


# URL to the policy document

# local file airlines_policy.md

url = "...."


response = requests.get(url)

response.raise_for_status()

faq_text = response.text
```

2. **Chunk the Document:** Split the markdown file into meaningful chunks. Splitting by headers (##) is a good starting point.

Python

```
# Split the text into documents based on markdown headers

chunks = re.split(r"(?=\n##\s)", faq_text)

documents = [{"page_content": chunk.strip()} for chunk in chunks if chunk.strip()]

print(f"Created {len(documents)} document chunks.")
```

3. **Create a Vector Database Interface:** Design a simple class or interface to handle interactions with the vector database. This makes your code modular and easy to adapt.

4. **Index the Documents:** Use an embedding model (e.g., OpenAI text-embedding-3-small or a Gemini equivalent) to convert the text chunks into vectors and store them in the database.

Step 3: Build the LangGraph Agent

This is the core of the project. You will define a state graph with nodes representing each step in the process.

1. **Define the State:** Create a TypedDict or BaseModel to represent the agent's state as it moves through the graph.

Python

```
from typing import List, TypedDict


class AgentState(TypedDict):
  email_content: str
  original_sender: str
  retrieved_docs: List[str]
  draft_reply: str
  validation_result: dict
  rewrite_count: int
  final_reply: str
  log: List[str]
```

2. **Implement Graph Nodes:** Each function below will be a node in your LangGraph.
   - **retrieve_context**: Takes the email content from the state, queries the vector DB, and adds the retrieved documents back to the state.
   - **draft_reply**: Takes the email content and retrieved docs, uses an LLM with a carefully crafted prompt to generate a draft response, and adds it to the state.
   - **validate_reply**: This is a critical guardrail. It uses an LLM (or LLMGuard) to check the draft against the retrieved documents for factual consistency, checks for PII, and ensures a polite tone. The result ({ "is_valid": True/False, "reason": "..." }) is added to the state.
   - **rewrite_reply**: If validation fails, this node takes the validation feedback and asks the LLM to generate a revised draft.
   - **send_email**: If validation passes, this node uses the Gmail API client to send the draft_reply.
   - **escalate**: If the draft fails validation after a set number of retries (e.g., 2), this node marks the email for human review.
3. **Define Conditional Edges:** Use conditional logic to create the draft -> validate -> rewrite loop.
   - After the validate_reply node, an edge should route to send_email if is_valid is true.
   - If is_valid is false and rewrite_count is less than 2, route to rewrite_reply.

  - If is_valid is false and rewrite_count is 2 or more, route to escalate.

4. **Configure Persistence:** Initialize your graph with MemorySaver to maintain state for the duration of a single email processing workflow.

Step 4: Create the FastAPI Application and UI

Expose your agent's functionality through a web server.

1. **API Endpoints:**
   - POST /trigger-run: Manually trigger the agent to check the inbox and process one email.
   - GET /logs: An endpoint to view the explainability logs generated by the agent.
   - GET /status: A simple endpoint showing the agent's current status (e.g., idle, processing).
2. **Background Worker:** The email polling logic should run in a background task (e.g., using FastAPI-Scheduler or a simple asyncio task) that periodically checks the Gmail inbox.
3. **Simple Web UI:** Create a single-page frontend that:
   - Shows the agent's status.
   - Displays a real-time log of agent actions (e.g., "Fetched email from user@example.com", "Drafted reply", "Validation PASSED", "Sent reply").
   - Presents the full explainability log for each processed email, including the retrieved documents that informed the reply.

6. Prompts and Guardrails

The quality of your agent depends heavily on its prompts.

- **Drafting Prompt: Instruct the LLM to act as a Airlines customer support agent. It should be told to use the provided context, be polite, concise, and <u>never make up information</u>.**
- You are a helpful and polite customer support agent for Swiss Air. Use the following retrieved policy documents to answer the customer's question. If the documents do not contain the answer, state that you do not have enough information and will escalate the ticket to a human agent.

- Retrieved Documents:

  {retrieved_docs}

- Customer Email:

{email_content}

- Your Reply:

- **Validation Prompt:** Create a prompt that asks an LLM to act as a reviewer. It should return structured JSON output.
- You are a quality assurance validator. Your task is to evaluate a draft email reply based on provided context. The reply must be factually consistent with the context and contain no PII. Respond with JSON only.

- Context Documents:

  {retrieved_docs}

- Draft Reply:

  {draft_reply}

- Return a JSON object with two keys: "is_valid" (boolean) and "reason" (a string explaining your decision, especially if invalid).

7. Deliverables

You must submit a link to a private GitHub repository containing the following:

1. **Source Code:** All Python source code for the agent, FastAPI app, and utilities.
2. **README.md:** A comprehensive document that includes:
   - Detailed setup instructions (API keys (don't give your keys), OAuth setup, Python environment).
   - Instructions on how to build and run the application using Docker.
   - A brief explanation of your design choices.
3. **Dockerfile:** The file to containerize the application.

4. **requirements.txt:** A list of all Python dependencies.
5. **Configuration File:** A config.yaml or .env.example file showing what environment variables are needed.
6. **Demo Video:** A short (2-4 minute) video demonstrating your agent successfully processing at least two different emails: one it can answer and one it escalates.

8. Grading Rubric (100 Points)

- **Core Functionality (60%)**
  - (15) LangGraph agent is correctly implemented with a stateful graph and conditional logic.
  - (15) RAG pipeline correctly ingests, chunks, and retrieves from the vector DB.
  - (15) Gmail API integration works for reading and sending emails autonomously and safely.
  - (15) The draft -> validate -> rewrite/escalate loop is functional and robust.
- **Application & Deployment (25%)**
  - (10) The application is served correctly via FastAPI.
  - (10) A functional web UI is present for monitoring and viewing logs.
  - (5) The application is fully containerized with Docker and includes run scripts.
- **Code Quality & Documentation (15%)**
  - (5) The code is clean, modular, and well-commented.
  - (5) The README.md file is clear and comprehensive.
  - (5) The demo video clearly showcases the project's functionality.

Integrity Note: All code must be your own. While you may use libraries and frameworks as specified, the core logic connecting them must be your original work. Be sure to use a test Gmail account and never commit sensitive credentials to your repository. Good luck!