Assignment: Building a Retrieval-Augmented Generation (RAG) System with Ollama, Kubernetes, GitHub Actions, and a Chatbot UI

Objective

Design and implement an advanced Retrieval-Augmented Generation (RAG) system for a flight information service. The system should enable users to ask questions about flights, retrieve relevant data from a mock database, and generate natural language responses using a locally hosted Ollama server. Additionally, the application should be deployed on a Kubernetes cluster using Minikube, include CI tests via GitHub Actions, and provide a simple web-based chat bot interface for user interaction.

Key Features

- **Mock Database:**
    - Store flight data in a flat list of dictionaries.
    - Each dictionary should include details like flight number, origin, destination, and time.
- **Query Handling:**
    - Parse user questions related to flight queries.
    - Search and retrieve relevant flight information from the mock database.
    - Format the retrieved data for further processing.
- **Ollama API Integration:**
    - Using locally running Ollama server.
    - Use Python's `ollama.py/langchain` library to send the formatted flight data and user query to the Ollama server.
    - Handle API responses and errors appropriately.
- **Chatbot UI:**
    - Develop a simple web-based chat bot interface.

    - The interface should allow users to enter flight-related questions and display the responses.
    - The UI may be built using lightweight frameworks such as Gradio.

- **Kubernetes Deployment:**
    - Prepare Kubernetes YAML configuration files for deployment on a Minikube cluster.
    - Create deployment and service YAML files to manage your backend (query handling and Ollama integration) and, if separate, the chatbot UI.
    - Define environment variables and necessary configurations.
- **Continuous Integration Testing:**
    - Set up GitHub Actions to run automated tests.
    - Include unit tests for query parsing, database search, API integration, and, optionally, basic UI interactions.
    - The CI workflow should automatically build the project and run tests on each push or pull request.

Implementation Steps

1. Mock Database

- **File:** `mock_database.py`
- **Requirements:**
    - Create a flat list (or similar structure) of dictionaries containing sample flight information.

1. Example entry:

```
 flights = [
    {"flight_number": "NY100", "origin": "New York", "destination":
"London", "time": "2025-05-01 08:00"},
    # ... additional flight records
]
```

2. Query Handler

- **File:** `query_handler.py`
- **Requirements:**
    - Implement functions to:
        - Parse the user's question.
        - Search the mock database for matching flight details.
        - Prepare and format the search results for sending to the Ollama server.
        - Handle cases where no flights are found or the query is ambiguous.

3. Ollama Server Integration

- **File:** `ollama_api.py`
- **Requirements:**
    - Use the `requests` library to send HTTP requests to the locally running Ollama server.
    - Pass the formatted flight data along with the user's query.
    - Retrieve and process the response from the Ollama server.
    - Include error handling for potential API failures.

4. Chatbot UI

- **Files:**
    - A backend file (e.g., `app.py`) to serve the API and UI.
- **Requirements:**
    - Develop a simple web interface where users can:
        - Input their flight-related queries.
        - View the natural language responses generated by the system.
    - Ensure smooth communication between the frontend and the backend API.

5. Kubernetes Deployment on Minikube

- **Files:**
    - Deployment YAML file (e.g., `deployment.yaml`)
    - Service YAML file (e.g., `service.yaml`)
- **Requirements:**
    - Define deployments for the backend service (and the chatbot UI if separate).
    - Expose your services via Kubernetes Service objects.
    - Configure environment variables and any necessary settings.
    - Provide instructions in the documentation on how to deploy to a Minikube cluster.

6. CI Testing with GitHub Actions

- **File:** `.github/workflows/test.yml`
- **Requirements:**
    - Set up a GitHub Actions workflow to automatically build the project and run tests.
    - Include unit tests for:
        - Query parsing and handling.
        - Ollama API integration (with mocked responses if needed).
        - Optionally, tests for the chatbot UI functionality.
    - Ensure the workflow triggers on pushes and pull requests.

7. Documentation

- **File:** `README.md`
- **Requirements:**
    - **Project Overview:** Describe the purpose and overall architecture of the project.
    - **Setup Instructions:**
        - Detail how to install dependencies.
        - Explain how to set up and run the local Ollama server.
        - Provide instructions for setting up the Kubernetes environment with Minikube.
    - **Deployment Steps:**
        - Instructions for applying Kubernetes YAML files.
        - Steps to expose and access the services.
    - **Usage:**
        - Guide users on how to interact with the chatbot UI.
        - Provide example queries (e.g., “What are the flights from New York to London?”).
    - **CI/CD Explanation:**
        - Detail the GitHub Actions configuration and testing workflow.

Deliverables

- **Code Files:**
    - `mock_database.py` – Contains the mock flight database.
    - `query_handler.py` – Parses queries and retrieves flight data.
    - `ollama_api.py` – Handles API calls to the locally running Ollama server.
    - UI-related files (e.g., `app.py`).
- **Kubernetes Configuration Files:**
    - Deployment YAML file (e.g., `deployment.yaml`).
    - Service YAML file (e.g., `service.yaml`).
- **CI Configuration:**

    - GitHub Actions workflow file (`.github/workflows/test.yml`).
- **Documentation:**
    - `README.md` – Detailed documentation for setup, deployment, and usage.
- **Additional Files:**
    - `requirements.txt` – A list of all Python dependencies.

Evaluation Criteria

- **Functionality:**
    - Correctly parses user queries, retrieves flight data from the mock database, and generates coherent responses using the Ollama server.
    - Provides an intuitive chatbot UI for user interaction.
- **Code Quality:**
    - Modular, clean, and well-commented code.
    - Robust error handling for both database queries and API interactions.
- **Deployment:**
    - Successful deployment on a Minikube Kubernetes cluster using provided YAML files.
    - Correct exposure and functionality of all services.
- **Testing:**
    - Automated tests running in GitHub Actions.
    - Adequate coverage for key functionalities such as query handling and API integration.
- **Documentation:**
    - A clear, concise README.md with all necessary instructions for building, deploying, and using the system.
    - Detailed explanations of the CI/CD process and Kubernetes deployment steps.