# Runbook

# Vander Lindes Conversational AI

This runbook provides detailed instructions on setting up, deploying, and maintaining the **Vander-lindes Conversational AI** project, a multi-agent conversational AI system designed for natural, context-aware, and policy-compliant customer support for the airline industry.

## Table of Contents

- Pre-requisites
- Environment Setup
- Running the Application
- Troubleshooting
- Maintenance
- Deployment
- Backup & Restore

## Pre-requisites

Before you begin, ensure you have the following:

1. **Python 3.9+**
2. **Git**: To clone the repository.
3. **LLM API Key**: (e.g., from Google AI Studio) for response generation.

## Environment Setup

❖ **Clone the Repository**
Clone the GitHub repository using the following command:

```
git clone https://github.com/Surya-S-17/Vander-lindes-Conversational-AI.git
cd Vander-lindes-Conversational-AI
```

❖ **Create a Virtual Environment**
It's recommended to set up a Python virtual environment to isolate dependencies:

```
python -m venv venv
source venv/bin/activate  # On Windows use `venv\Scripts\activate`
```

- **Install Dependencies**

  Install the required Python dependencies listed in
  Requirements.txt
  ```
  pip install -r requirements.txt
  ```

- **Set Environment Variables**

  Create a .env file in the root directory to store your API keys and other sensitive information:
  ```
  LLM_API_KEY=your_api_key_here
  ```
  Replace your_api_key_here with your actual API key.

## Running the Application

The application consists of several backend services and a frontend demo. You will need to run each component in separate terminals.

### Backend Services

You need to run each of the four backend services in separate terminals:

- **Intent Classifier Service** (Terminal 1):
  ```
  uvicorn intent_classify:app --reload
  ```
- **Context Management Service** (Terminal 2):
  ```
  uvicorn context_management:app --reload
  ```
- **Policy Retrieval (RAG) Service** (Terminal 3):
  ```
  uvicorn Policy_retrival_RAG:app --reload
  ```
- **Response Generation Service** (Terminal 4):
  ```
  uvicorn response_generation:app --reload
  ```

### Frontend Demo

Once the backend services are running, you can start the frontend:

- **Launch Streamlit Demo** (Terminal 5):
  ```
  streamlit run demo.py
  ```
- **Open the Demo in Browser**:

  Navigate to http://localhost:8501 in your browser to interact with the chatbot.

## Troubleshooting

Here are some common issues and their solutions:

### Issue 1: `ModuleNotFoundError` for required libraries

- **Solution**: Ensure that all dependencies are installed:
  ```
  pip install -r requirements.txt
  ```

### Issue 2: Backend services fail to start

- **Solution**: Ensure all backend services are running in separate terminals. If a service fails, check the logs for specific errors (e.g., incorrect configuration, missing dependencies).

### Issue 3: Cannot connect to the frontend

- **Solution**: Make sure the backend services are running. The frontend requires all backend services to be active to function properly.

### Issue 4: `LLM_API_KEY` not set correctly

- **Solution**: Check the `.env` file for correctness. Ensure the API key is properly assigned:
  ```
  LLM_API_KEY=your_api_key_here
  ```

## Maintenance

- **Update Dependencies**
  If new updates to dependencies are released, run:
  ```
  pip install --upgrade -r requirements.txt
  ```
- **Monitor Backend Services**
  Ensure the backend services are running smoothly. You can set up logging to track errors in services like:
  ```
  intent_classify:app
  context_management:app
  Policy_retrival_RAG:app
  response_generation:app
  ```
- **Check for Errors in Response Generation**
  If the responses from the bot seem off or incorrect, ensure the underlying LLM model is correctly configured and the API key is active.
- **Database & RAG Maintenance**
  If using a vector database (e.g., ChromaDB), ensure it is up to date with relevant data for Policy Retrieval. You may need to reindex periodically to improve search and retrieval performance.

## Deployment

For deployment to a production environment (e.g., AWS, Azure, Google Cloud), follow these steps:

1. **Containerization with Docker**
   You can containerize the application for easier deployment using Docker. Create a Dockerfile for each microservice, or use a single docker-compose.yml file to orchestrate the containers.

2. **Configure Web Server**
   Deploy the backend services on a web server like **Nginx** or **Gunicorn** to handle production traffic. If using **FastAPI**, deploy via uvicorn with Gunicorn for better performance.

3. **Set up SSL/TLS**:
   For secure communication, ensure SSL/TLS is set up on your web server for HTTPS.

4. **Load Balancer**:
   If deploying multiple instances for scalability, configure a load balancer to distribute the traffic across instances.

5. **Environment Configuration**:
   Set up environment variables in your production environment (e.g., AWS Lambda, Google Cloud Functions, or Heroku).

## Backup & Restore

1. **Backup Backend Data**:
   If using a database for policy storage (e.g., ChromaDB), schedule regular backups to avoid data loss. You can use cloud backup services or manual backups based on your storage configuration.

2. **Backup Configuration Files**:
   Make sure to backup critical configuration files such as .env, requirements.txt, and any custom scripts that define the logic of your agents.

3. **Restore from Backup**:
   To restore from a backup, follow the reverse process to restore the configuration files and data backups to the correct locations.

### Conclusion

This runbook will guide you through the setup, troubleshooting, deployment, and maintenance of the **Vander-lindes Conversational AI** project. Follow these steps for a smooth operational experience and refer back to this document whenever necessary for guidance.