Lesson 3 Demo 8

Implementation of AutoGen in LangGraph Workflows

Objective: To demonstrate how AutoGen can be integrated into LangGraph workflows to build an interactive AI assistant for handling FAQs, order inquiries, and escalations

You are developing a conversational AI assistant for a retail company to enhance customer service by automating responses to frequently asked questions and handling order inquiries. The assistant will leverage AutoGen within LangGraph workflows to provide instant responses, retrieve order statuses from an external API, and escalate complex issues to a human representative when necessary. The goal is to improve efficiency while maintaining a seamless customer experience.

Prerequisites:

1. Create a virtual environment
2. Install dependencies

Steps to be followed:

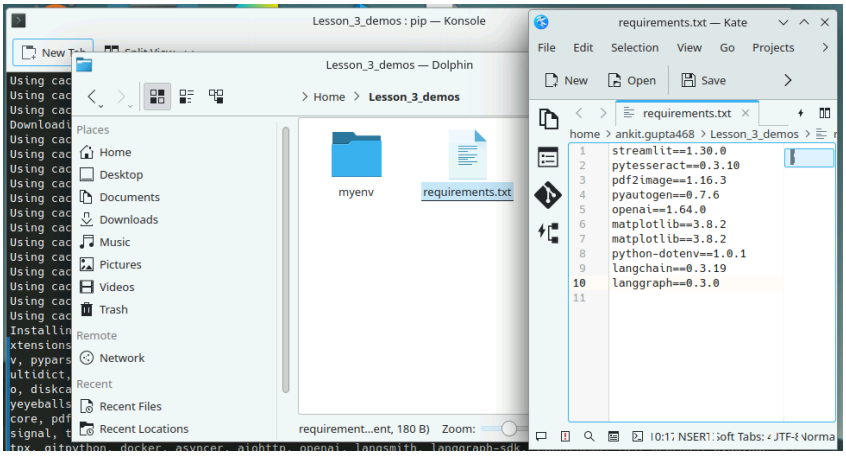
1. Setup the environment
2. Define the State for LangGraph
3. Create the FAQ Handler Function
4. Create the Escalation Flow
5. Build the LangGraph Workflow
6. Set up the Streamlit Interface
7. Run the code

Step 1: Set up the environment

1.1 Open command prompt and go to the "**Lesson_3_demos**" folder (which we created in Demo_1) using the command given below:
mkdir Lesson_3_demos (not needed if the folder is already created in Demo1)
cd Lesson_3_demos

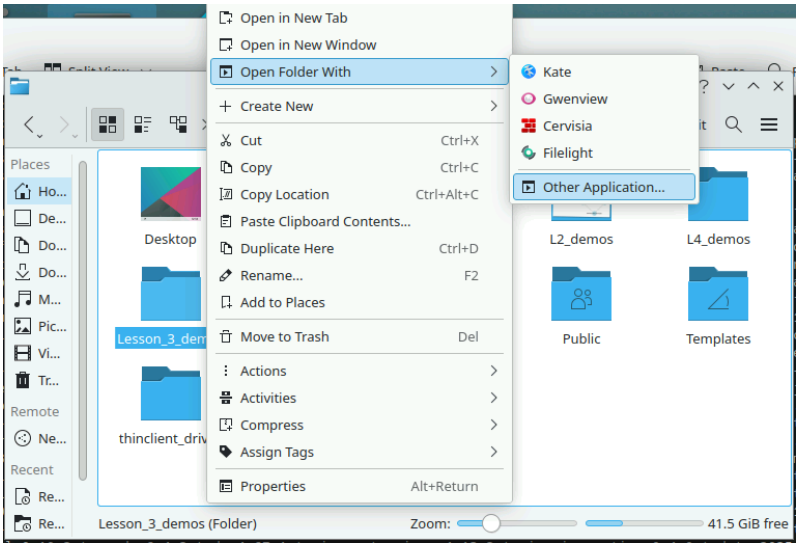
1.2 After this, activate the virtual environment using the command below:
python3 -m venv venv (not needed if the virtual env. is already created in Demo1)
source venv/bin/activate

1.3 Now, create a `requirements.txt` file inside the folder with required libraries (not needed if already done in Demo1):



1.4 Install all the required libraries using the command below:
pip install -r requirements.txt (not needed if already done in Demo1)

1.5 Now, open the folder using VS Code editor:



After this, open a new Python file using the "**New File**" option and name it as "**Demo8**".

Step 2: Define the State for LangGraph

2.1 Define a **QueryState** class using TypedDict to capture user input and bot response.

2.2 This state holds the information for processing user queries and handling responses in the workflow.

```
import streamlit as st
from typing import TypedDict
from autogen import AssistantAgent, UserProxyAgent
from langgraph.graph import StateGraph, START


# Define the state for LangGraph
class QueryState(TypedDict):
    user_input: str
    response: str
```

Step 3: Create the FAQ Handler Function

3.1 Implement a function faq_handler that uses the **FAQ_Bot** assistant to handle FAQs and order status queries.

3.2 This function calls the assistant agent with the user's input and retrieves the bot's response, returning it along with the original user input.

```
# Define AutoGen Assistant Agent
define_assistant = AssistantAgent(
    name="FAQ_Bot",
    llm_config={
        "config_list": [
            {
                "api_type": "azure",
                "azure_endpoint": "https://openai-api-management-gw.azure-
api.net/deployments/gpt-4o-mini/chat/completions?api-version=2023-12-01-preview",
                "api_version": "2023-12-01-preview",
                "api_key":
"2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAABACOGi3sC",
                # "deployment_name": "gpt-4o-mini",  # corrected key name
                "model": "gpt-4o-mini",  # optional, but good to keep
            }
        ],
        "temperature": 0.7,
    },
    system_message="I can answer FAQs and check order statuses.",
    code_execution_config={'use_docker': False}
)
```

Step 4: Create the Escalation Flow

4.1 Implement the escalate_to_human function for more complex queries that cannot be resolved by the AI.
4.2 The function returns a message indicating that the query is being escalated to human support for further assistance.

```
# Function to handle FAQs

def faq_handler(state: QueryState) -> QueryState:

    response = define_assistant.run(state["user_input"])  # Generate response

    return {"user_input": state["user_input"], "response": response}


# Function to escalate complex queries

def escalate_to_human(state: QueryState) -> QueryState:

    return {"user_input": state["user_input"], "response": "Escalating to human support.
Please wait."}
```

Step 5: Build the LangGraph Workflow

5.1 Create a **StateGraph** for managing the workflow, which includes nodes for **FAQ_Handler** and **Escalation**.
5.2 Add edges to connect the start state to the FAQ handler, and from the FAQ handler to the escalation flow when necessary.
5.3 Compile the graph to manage the query handling and escalation.

```
# Create LangGraph Workflow

graph = StateGraph(QueryState)

graph.add_node("FAQ_Handler", faq_handler)

graph.add_node("Escalation", escalate_to_human)


graph.add_edge(START, "FAQ_Handler")

graph.add_edge("FAQ_Handler", "Escalation")


workflow = graph.compile()
```

Step 6: Set Up the Streamlit Interface

6.1 Build a **Streamlit** UI that allows users to input their query and submit it for processing.

6.2 Once the user submits their query, the workflow is invoked, and the response (either from the bot or human support) is displayed.

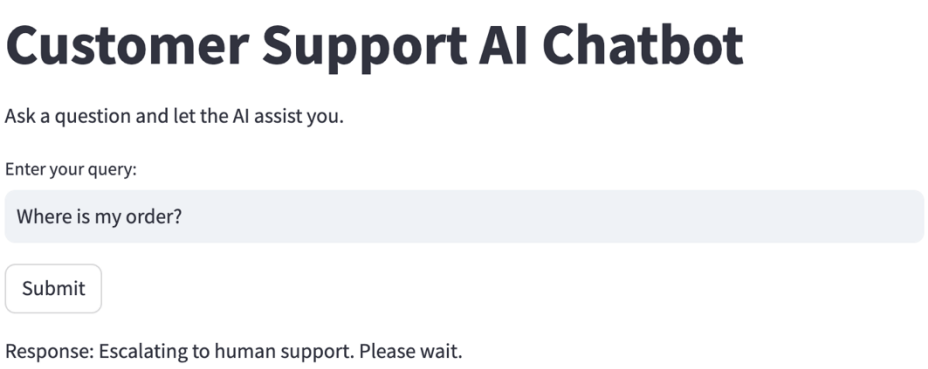
```
# Streamlit UI
st.title("Customer Support AI Chatbot")
st.write("Ask a question and let the AI assist you.")

user_input = st.text_input("Enter your query:")
if st.button("Submit"):
    query = QueryState(user_input=user_input)
    result = workflow.invoke(query)
    st.write("Response:", result.response)
```

Step 7: Run the code:

7.1 Save the file and then run the streamlit webapp from command prompt using the command given below:

streamlit run Demo8.py

Output:



By following the above-mentioned steps, you have successfully demonstrated how AutoGen, LangGraph, and Streamlit can be integrated to build an interactive and intelligent AI-powered customer support chatbot. By leveraging AutoGen's AssistantAgent, the chatbot can efficiently respond to frequently asked questions while using LangGraph to determine when a query requires human escalation. The Streamlit UI provides a simple and user-friendly interface, enabling real-time interactions.