Lesson 2 Demo 5

End-to-End Automated Client Inquiry Workflow Demo

Objective: To streamline and automate the entire client inquiry processing workflow by integrating orchestration, evaluation, and task-specific agents

Imagine a business receiving hundreds of client inquiries each day. Each inquiry requires a series of steps—gathering client details, evaluating the request, scheduling appointments, and updating the CRM. Without automation, these tasks are handled manually by different departments or employees. This manual approach results in delays, errors, inconsistent processes, and a lack of tracking, causing missed opportunities, frustrated clients, and wasted resources. In such a scenario, there is a pressing need for a streamlined, automated solution that ensures all inquiries are processed efficiently and accurately from start to finish.

Prerequisites:

1. Create a virtual environment
2. Activate the virtual environment
3. Install the libraries in `requirements.txt`

Steps to be followed:

1. Set up the environment
2. Creation of state.py
    a. Defining the InquiryState class
    b. Default values for optional attributes
3. Creation of agents.py
    a. Defining the intake agent
    b. Defining the evaluation agent
    c. Defining the scheduling agent
    d. Defining the CRM update agent
4. Creation of workflow.py
    a. Creating the state graph
    b. Connecting the agents
    c. Compiling the workflow
5. Creation of main.py
    a. Setting up the Streamlit UI
    b. Invoking the workflow
    c. Displaying the results
6. Run the webapp

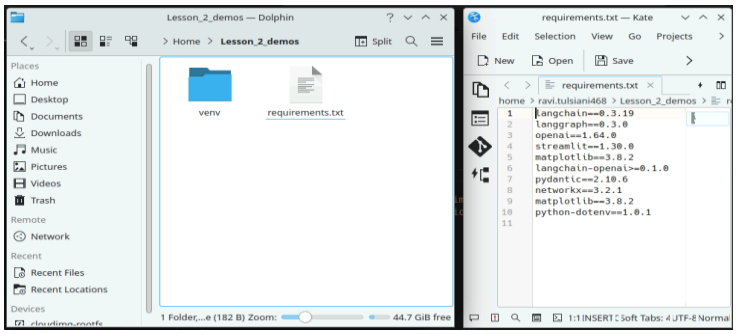
Step 1: Set up the environment

1.1 Open command prompt and go to the "**Lesson_2_demos**" folder (which we created in Demo_1) using the command given below:
mkdir Lesson_2_demos (not needed if the folder is already created in Demo1)
cd Lesson_2_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.

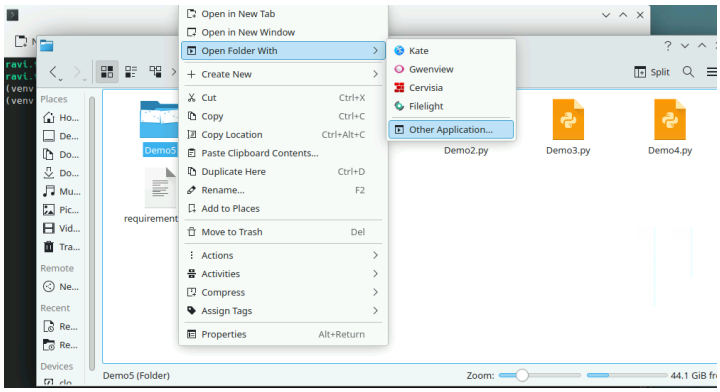
1.3 Now, create another folder inside the "**Lesson_2_demos**" folder named "**Demo5**" using the command given below:
mkdir Demo5
cd Demo5

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



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

1.6 Now, open the folder named "**Demo5**" using VS Code editor:



1.7 After this, open 4 new Python file using the "**New File**" option and name them as "**state**", "**agents**", "**workflow**", and "**main**".

Step 2: Creation of state.py

2.1 In this step, the state.py file is created to define the data structure for handling client inquiry information. This structure holds essential details related to each inquiry, ensuring that all necessary data is captured and maintained throughout the workflow. It provides a unified format for tracking the status and progress of an inquiry from intake through to CRM update.

2.2 The `InquiryState` class is defined to capture the client's name, email, request details, approval status, evaluation notes, appointment time, CRM logs, and activity logs.

2.3 Default values are assigned to certain attributes, such as setting `is_approved` to False initially and initializing `activity_log` as an empty list to track the actions performed on the inquiry.

```
# ---- state.py ----
from dataclasses import dataclass, asdict, field
from typing import List

@dataclass
class InquiryState:
    """Defines the structure of inquiry state, tracking client details, process
status, and logs."""
    client_name: str
    client_email: str
    request_details: str
    is_approved: bool = False
    evaluation_notes: str = ""
    appointment_time: str = ""
    crm_log: str = ""
    activity_log: List[str] = field(default_factory=list)
```

Step 3: Creation of agents.py

3.1 In this step, the agents.py file is developed to define different agents that process the client inquiries. Each agent handles a specific task in the workflow, such as intake, evaluation, scheduling, and CRM updates. These agents interact with the InquiryState to update its attributes based on the progress of the request. The goal is to automate the decision-making process and ensure that each part of the inquiry is handled consistently.

3.2 The intake agent handles the initial reception of the inquiry, logging the request and recording the time of submission.

3.3 The evaluation agent evaluates the request using an AI model to determine whether it should be approved or denied based on predefined criteria.

3.4 The scheduling agent sets up an appointment time if the request is approved or logs that no appointment is set if the request is denied.

3.5 The CRM update agent updates the CRM system with relevant information and marks the workflow as complete.

(Code on next page)

```
# ---- agents.py ----
import datetime
from langgraph.graph import END
from dataclasses import asdict
from state import InquiryState
from dotenv import load_dotenv
import openai
import os

# Load environment variables
load_dotenv()

# Initialize Azure OpenAI client
client = openai.AzureOpenAI(

api_key="2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AA
ABACOGi3sC",
    api_version="2023-12-01-preview",
    azure_endpoint="https://openai-api-management-gw.azure-api.net/deployments/gpt-
4o-mini/chat/completions?api-version=2023-12-01-preview"
)

def intake_agent(state: InquiryState) -> dict:
    """Handles the intake process and logs the request."""
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    state.activity_log.append(f"Request received from {state.client_name} at
{timestamp}.")
    return asdict(state)

def evaluation_agent(state: InquiryState) -> dict:
    """Uses an Azure OpenAI LLM to determine if the inquiry is approved."""
    try:
        prompt = (
            f"Analyze the following request for approval. Return only 'approved' or
'denied'.\n"
            f"Request: {state.request_details}"
        )
        response = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=[
                {"role": "system", "content": "You are a helpful assistant that
makes approval decisions."},
                {"role": "user", "content": prompt}
            ],
            temperature=0.3,
            max_tokens=100
        )
        decision = response.choices[0].message.content.strip().lower()
        state.is_approved = "approved" in decision
        state.evaluation_notes = (
            "Request meets approval criteria." if state.is_approved
            else "Request does not meet approval criteria."
        )
        state.activity_log.append("Evaluation outcome: " + state.evaluation_notes)
    except Exception as e:
        state.activity_log.append(f"Evaluation error: {str(e)}")
        state.is_approved = False
        state.evaluation_notes = "Error occurred during evaluation."

    return asdict(state)

def scheduling_agent(state: InquiryState) -> dict:
    """Schedules a meeting if the request is approved."""
    if state.is_approved:
        meeting_time = datetime.datetime.now() + datetime.timedelta(days=1)
        state.appointment_time = meeting_time.strftime("%Y-%m-%d %H:%M:%S")
        state.activity_log.append("Meeting set for " + state.appointment_time)
    else:
        state.activity_log.append("No appointment set as request was denied.")
    return asdict(state)

def crm_update_agent(state: InquiryState) -> dict:
    """Handles CRM update and properly ends the workflow."""
    state.crm_log = "CRM updated with request details."
    state.activity_log.append("CRM updated successfully.")
    return asdict(state)
```

Step 4: Creation of workflow.py

4.1 In this step, the workflow.py file is created to design the flow of the inquiry process. The goal is to connect the different agents (defined in agents.py) in a structured sequence, ensuring that each step in the process is executed in the correct order. The workflow orchestrates the processing of inquiries by defining how data flows between the agents and how the overall process is managed.

4.2 The state graph is created to represent the sequence of actions and interactions between agents in the workflow.

4.3 The agents are connected in the graph to ensure that after one agent completes its task, the next agent in the sequence is triggered automatically.

4.4 The workflow is compiled to produce a final, executable process that integrates all the agents and ensures that the client inquiry moves smoothly from intake to CRM update.

(Code on next page)

```
# ----workflow.py----
from langgraph.graph import StateGraph
import networkx as nx
import matplotlib.pyplot as plt
from agents import intake_agent, evaluation_agent, scheduling_agent,
crm_update_agent
from state import InquiryState
from dataclasses import asdict

def construct_graph():
    """Builds the state graph for inquiry processing."""
    workflow = StateGraph(InquiryState)  # ☑ Use InquiryState

    # ☑ Add agent nodes
    workflow.add_node("intake", intake_agent)
    workflow.add_node("evaluation", evaluation_agent)
    workflow.add_node("scheduling", scheduling_agent)
    workflow.add_node("crm_update", crm_update_agent)

    # ☑ Add an "end" node as a dummy function
    workflow.add_node("end", lambda state: asdict(state))  # No processing,
just a pass-through

    # ☑ Define the workflow structure
    workflow.set_entry_point("intake")
    workflow.add_edge("intake", "evaluation")
    workflow.add_edge("evaluation", "scheduling")
    workflow.add_edge("scheduling", "crm_update")

    # ☑ Connect "crm_update" to the "end" node
    workflow.add_edge("crm_update", "end")

    # ☑ Mark "end" as the final step
    workflow.set_finish_point("end")

    return workflow.compile()



def visualize_graph():
    """Generates a visualization of the workflow graph."""
    graph = nx.DiGraph()
    edges = [
        ("START", "intake"),
        ("intake", "evaluation"),
        ("evaluation", "scheduling"),
        ("scheduling", "crm_update")  # ☑ Last node (no outgoing edges)
    ]
    graph.add_edges_from(edges)

    plt.figure(figsize=(8, 5))
    nx.draw(graph, with_labels=True, node_color='lightblue',
edge_color='gray', node_size=2000, font_size=10, font_weight='bold')
    plt.savefig("workflow.png")
```

Step 5: Creation of main.py

5.1 In this step, the main.py file is created to serve as the user interface and entry point for the inquiry processing system. The primary objective of this file is to allow users to interact with the system, input client inquiry data, trigger the workflow, and view the results. It integrates the workflow created in workflow.py with a front-end interface to provide an easy-to-use platform for handling inquiries.

5.2 The **Streamlit UI** is set up to accept input from the user, such as the client's name, email, and request details.

5.3 The **workflow is invoked** with the user input, passing the data through the sequence of agents for processing.

5.4 The **results are displayed** to the user, including evaluation outcomes, appointment details (if applicable), CRM updates, and an activity log of the entire process.

5.5 This step ensures that the system is accessible to users and facilitates seamless interaction with the inquiry workflow, making the process automated and transparent.

(Code on next page)

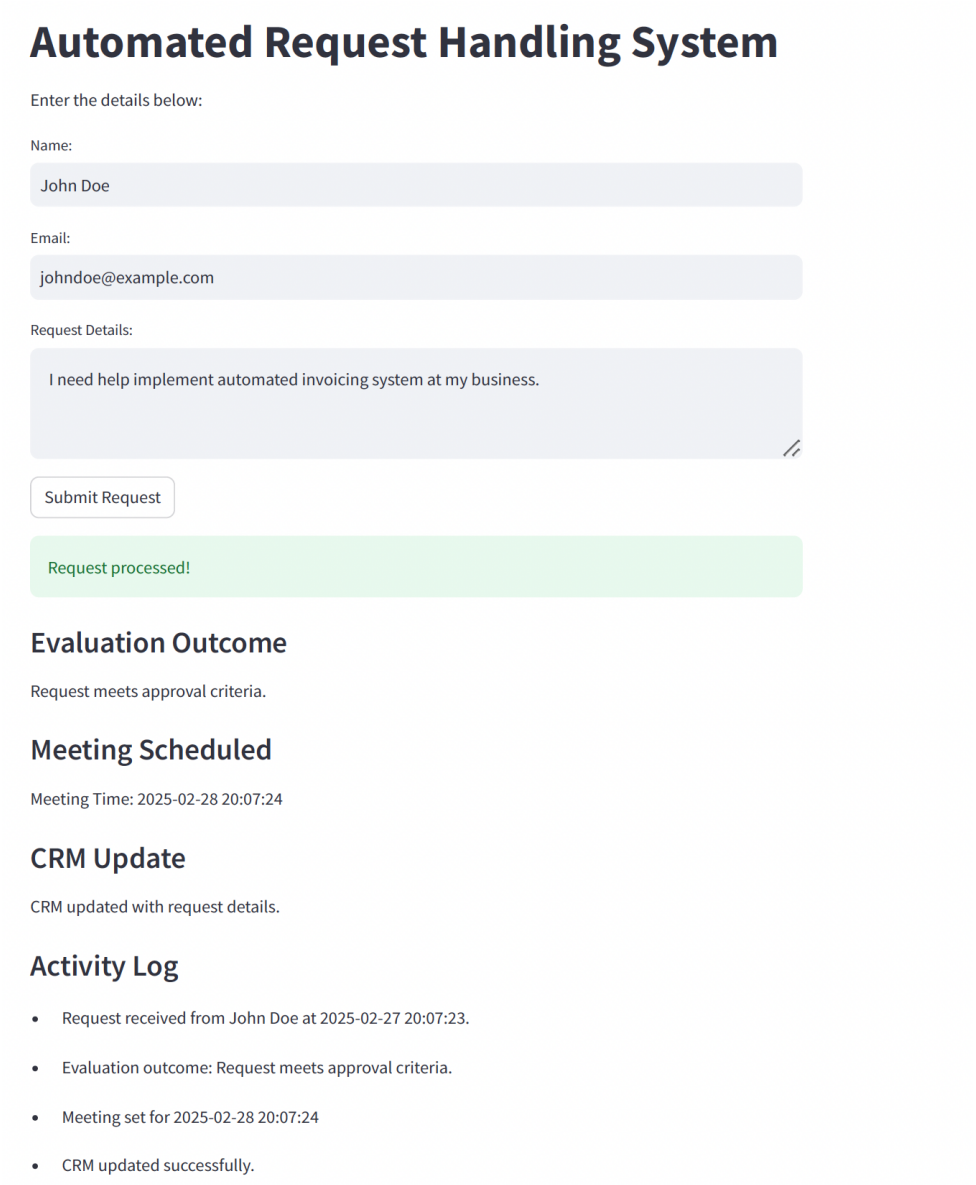
```python
# ----main.py----
import streamlit as st
from state import InquiryState
from workflow import construct_graph, visualize_graph
from dataclasses import asdict
import matplotlib.pyplot as plt

def main():
    """Streamlit UI for inquiry processing system."""
    st.title("Automated Request Handling System")
    st.write("Enter the details below:")

    client_name = st.text_input("Name:")
    client_email = st.text_input("Email:")
    request_details = st.text_area("Request Details:")

    if st.button("Submit Request"):
        inquiry_state = InquiryState(
            client_name=client_name,
            client_email=client_email,
            request_details=request_details
        )
        state_dict = asdict(inquiry_state)

        process_graph = construct_graph()
        final_state = process_graph.invoke(state_dict)

        st.success("Request processed!")
        st.markdown("### Evaluation Outcome")
        st.write(final_state.get("evaluation_notes", ""))
        if final_state.get("is_approved"):
            st.markdown("### Meeting Scheduled")
            st.write(f"Meeting Time: {final_state.get('appointment_time', '')}")
        st.markdown("### CRM Update")
        st.write(final_state.get("crm_log", ""))
        st.markdown("### Activity Log")
        for entry in final_state.get("activity_log", []):
            st.write(f"- {entry}")

    # st.markdown("### Workflow Graph")
    graph_path = visualize_graph()
    plt.savefig('Workflow.png')
    # st.image(graph_path, caption="Process Workflow")

if __name__ == "__main__":
    main()
```

Step 6: Run the webapp

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

streamlit run main.py

Output:



By following the above-mentioned steps, you have successfully automated the client inquiry management system and efficiently handled client requests by integrating state management, agent tasks, and workflow orchestration. It ensures accurate, consistent, and timely processing through AI-powered evaluation and a user-friendly interface. By automating the entire process, the system reduces manual errors, increases operational efficiency, and provides a scalable solution for managing growing business needs.