**Lesson 3 Demo 2**

**IT Helpdesk Chatbot: Enabling Memory and Context Awareness**

**Objective:** To develop an AI-powered IT support chatbot that automates responses, remembers past issues, and provides relevant troubleshooting steps to improve efficiency

You are building an AI-driven IT support chatbot to assist users with common technical issues. IT support teams often receive repetitive queries, leading to delays and inefficiencies. By automating responses and retaining past interactions, the chatbot can provide quicker, more relevant troubleshooting steps, reducing the support team's workload and improving response times. The goal is to:

a. Configure an AI agent to provide troubleshooting steps based on past interactions

b. Implement a Streamlit-based web UI for real-time user interactions

c. Extend the chatbot for multi-agent collaboration or integration with IT ticketing systems

**Prerequisites**

1. Create a virtual environment
2. Activate the virtual environment
3. Install required dependencies

**Steps to be followed:**

Step 1: Setup the environment

Step 2: Load Environment Variables and Set Up OpenAI Client
Step 3: Define the IT Support Chatbot Class with Memory Capability
Step 4: Create a Streamlit Web App for Real-Time Interaction

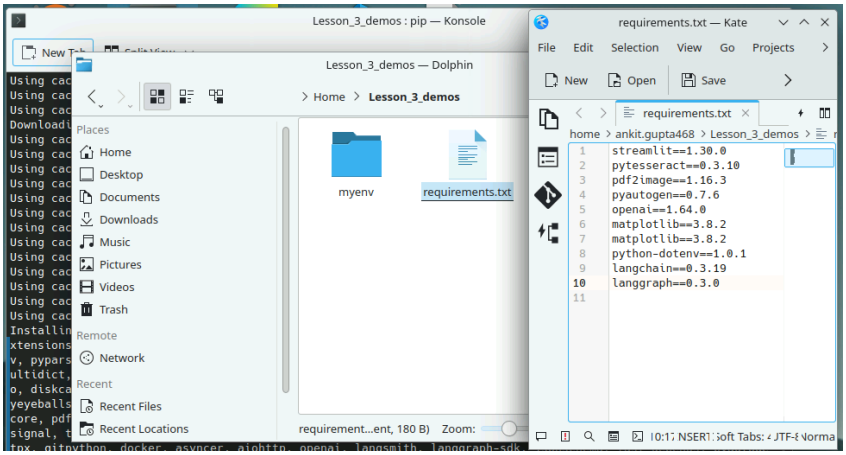
Step 5: Run the code

## 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**

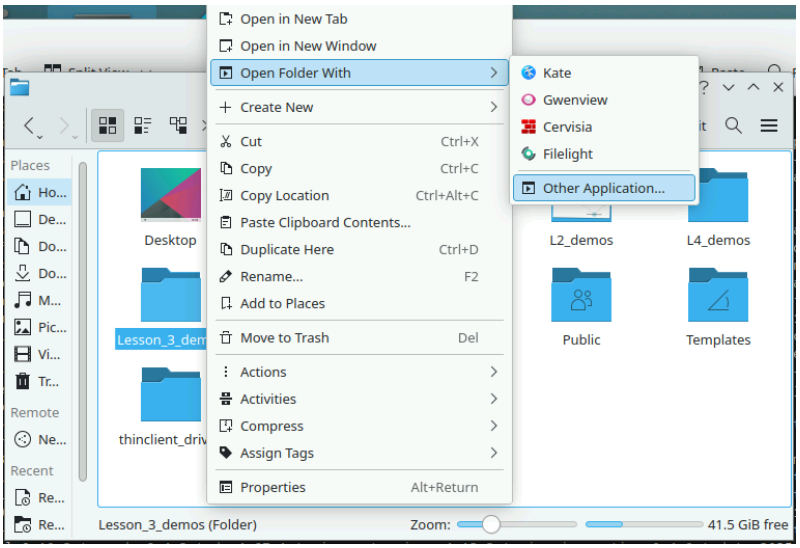
1.2 After this, activate the virtual environment using the command below:
**python3 -m venv venv** (not needed if the virtual env. is already created in Demo1)
**source venv/bin/activate**

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 "**Demo2**".

**Step 2: Load Environment Variables and Set Up OpenAI Client**

Start by importing necessary libraries and configuring the connection to the Azure OpenAI endpoint using the openai.AzureOpenAI client.
This setup authenticates your application to send requests to the GPT model securely.

```
import autogen

import openai

import streamlit as st

from dotenv import load_dotenv

import os

client = openai.AzureOpenAI(


api_key="2ABecnfxzhRg4M5D6pBKiqxXVhmGB2WvQ0aYKkbTCPsj0JLKsZPfJQQJ99BDAC77bzfXJ3w3AAABACOGi
3sC",

    api_version="2023-12-01-preview",

    # azure_endpoint="https://openai-api-management-gw.azure-api.net/"

    azure_endpoint="https://openai-api-management-gw.azure-api.net/deployments/gpt-4o-
mini/chat/completions?api-version=2023-12-01-preview"

)
```

**Step 3: Define the IT Support Chatbot Class with Memory Capability**

Create a custom chatbot class ITSupportBot that inherits from autogen.AssistantAgent. This bot will store past user issues using an internal memory dictionary, allowing it to maintain conversation context across multiple queries.

```
# Step 2: Define an IT Support chatbot that remembers past issues
class ITSupportBot(autogen.AssistantAgent):
    def __init__(self, name, memory=None, model="gpt-4o-mini"):
        super().__init__(name=name)
        self.memory = memory if memory is not None else {}  # Stores past user issues
        self.model = model  # Specifies GPT model

    def generate_reply(self, message, sender, **kwargs):
        """
        Step 3: Generates a response based on user input and past issues.
        - Retrieves past issues if available
        - Stores the latest issue in memory
        - Calls the GPT model to generate a response
        """
        context = self.memory.get(sender, "")  # Retrieves past issue if available
        self.memory[sender] = message  # Stores latest issue in memory
        response = self._get_gpt_response(message, context)  # Calls GPT for reply
        return response

    def _get_gpt_response(self, message, context):
        prompt = f"User's previous issue: {context}\nNew issue: {message}\nIT Support
Response:"
        response = client.chat.completions.create(
            model=self.model,
            messages=[
                {"role": "system", "content": "You are a helpful IT support assistant
providing troubleshooting steps."},
                {"role": "user", "content": prompt}
            ]
        )
        return response.choices[0].message.content.strip()
```

**Step 4: Create a Streamlit Web App for Real-Time Interaction**

Use Streamlit to build a simple web interface:

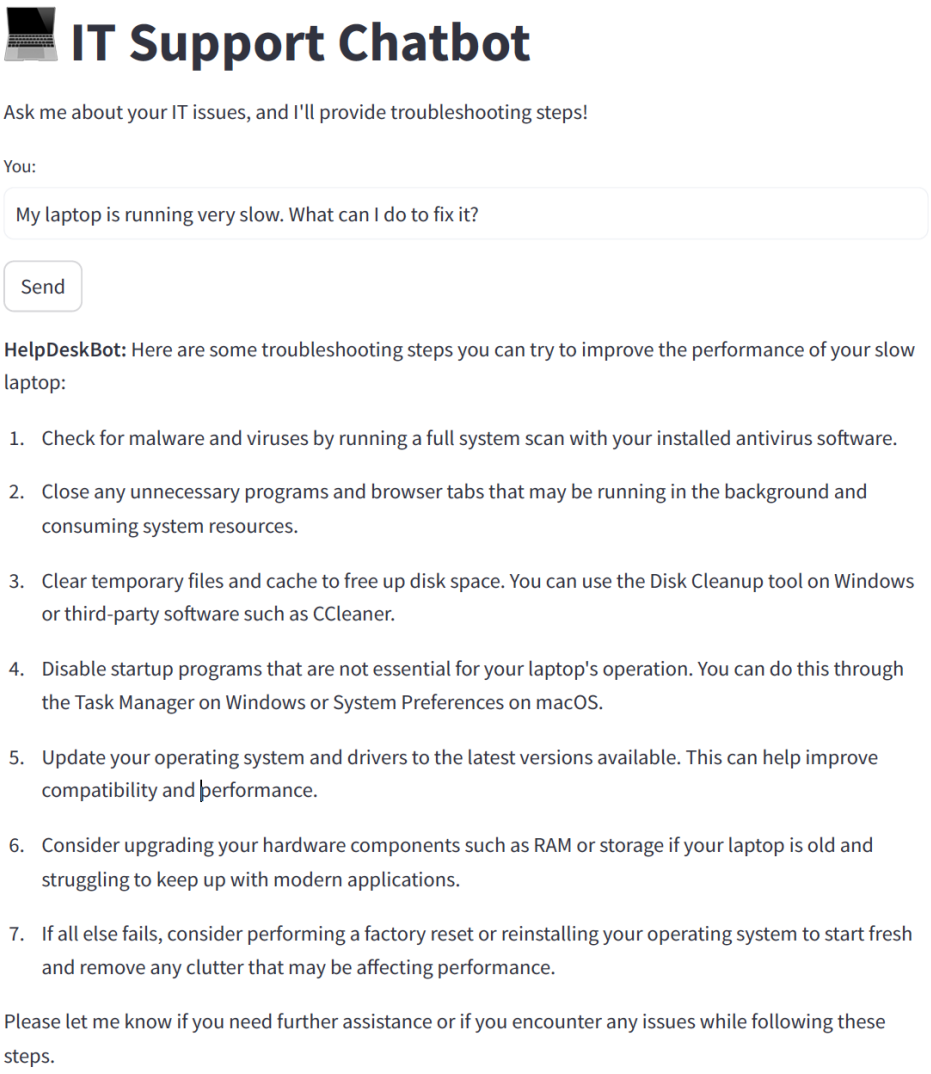
4.1 Display the chatbot title and instructions.
4.2 Provide a text input field for users to submit their issues.
4.3 Display the bot's response after clicking the "Send" button.
4.4 Maintain the chatbot instance across interactions using st.session_state to ensure memory persists throughout the session.

```
# Step 5: Streamlit UI for real-time chatbot interaction

st.title("□ IT Support Chatbot")

st.write("Ask me about your IT issues, and I'll provide troubleshooting steps!")


# Initialize chatbot

if "chatbot" not in st.session_state:

    st.session_state.chatbot = ITSupportBot(name="HelpDeskBot")


# Input field for user query

user_input = st.text_input("You:", "")


if st.button("Send"):

    if user_input:

        response = st.session_state.chatbot.generate_reply(user_input, "User1")

        st.write(f"**HelpDeskBot:** {response}")
```

**Step 5: Run the code:**

5.1 Save the file and then run the streamlit webapp from command prompt using the command given below:
**streamlit run Demo2.py**

**Output:**




By following the above-mentioned steps, you have successfully highlighted how AI agents can leverage memory and context awareness to enhance user interactions. By remembering past issues, the chatbot avoids repetitive questions and provides more relevant troubleshooting steps. Context-aware responses ensure that conversations feel natural and adaptive rather than isolated exchanges. Leveraging GPT-4, the bot generates dynamic replies tailored to each user's history, making interactions more efficient and human-like. This approach is invaluable for IT support, customer service, and virtual assistants that require continuous, context-rich engagement.