Project: Chatbot for User and Group Information on z/OS and z/VM Servers

Defining the Objective

Objective:
The objective of this chatbot is to provide detailed and accessible information about users and groups on z/OS (and in the future, z/VM) servers to company employees. The chatbot should be able to answer questions like "Who is the owner of user XYZ?" or "Which groups is user ABC connected to?".

Problem to Solve:
The lack of an efficient and easy-to-use tool for obtaining information about users and groups on z/OS and z/VM servers. Currently, users must search for information in multiple places and documents, which can be time-consuming and frustrating.

Type of Information Provided:

- **Users:**
  - Owner of the user
  - User attributes (OMVS, TSO, etc.)
  - Groups the user is connected to
- **Groups:**
  - Owner of the group
  - Group members
  - Group attributes

Type of Interaction:

1. **Starting the Conversation:**
   - The user initiates the conversation with the chatbot and selects whether they want information about a user or a group.
2. **Selecting the Environment:**

- The chatbot asks the user about the specific environment (e.g., server XYZ) where the information is needed.

3. **Verifying the Environment:**
   - The chatbot checks if the selected environment is in the Ansible inventory and looks for matches.
4. **Confirming the Environment:**
   - If a match is found, the chatbot asks the user if the environment is correct.
5. **Executing the Playbook:**
   - If the answer is positive, the chatbot executes an Ansible playbook in the selected environment to obtain the requested information.
6. **Displaying the Information:**
   - The chatbot displays the requested information to the user clearly and comprehensively, allowing the user to ask additional questions about the obtained information.

Ansible Inventory:

The Ansible inventory is a file containing the list of servers and their respective details, such as location and server type. In this project, the inventory contains the list of z/OS (and in the future, z/VM) servers and their associated details.

Ansible Tower API:

The Ansible Tower API allows programmatic interaction with Ansible Tower, executing playbooks, obtaining information about servers and inventories, among other functionalities.

LU and LG Commands:

The chatbot will execute LU (List User) and LG (List Group) commands on the z/OS server to obtain the requested information about users and groups. These commands are essential for accessing the required data on z/OS servers.

Technical Implementation

Key Functionalities:

1. **Loading Environment Variables:**
   - Loads the necessary variables from an `.env` file to configure API credentials and endpoints.
2. **User Interaction:**
   - The chatbot interacts with the user to obtain the entity type (user or group) and the environment.

3. **Ansible Execution:**
   - Executes an Ansible playbook to retrieve information about the user or group on the z/OS server.
4. **Processing and Humanizing Output:**
   - Processes and humanizes the output from Ansible using Watsonx.ai.
5. **Additional Interaction:**
   - The user can ask additional questions about the retrieved information and provide feedback on the generated responses.
6. **Future Functionality:**
   - While not yet implemented, the plan is to extend this functionality to z/VM servers.

7. **Chat with Watsonx:**
   - Finally, the user can have a conversation with Watsonx about the information provided by Ansible.

Workflow

1. **Loading Environment Variables:** The necessary variables are loaded from the `.env` file.
2. **User Interaction:** The chatbot interacts with the user to get the type of entity (user or group) and the environment.
3. **Ansible Execution:** An Ansible playbook is executed to retrieve information about the user or group on the z/OS server.
4. **Processing and Humanizing Output:** The output from Ansible is processed and humanized using Watsonx.ai.

5. **Additional Interaction:** The user can ask additional questions about the retrieved information and provide feedback on the responses generated.
6. **Future Functionality:** Extend functionality to z/VM servers.
7. **Chat with Watsonx:** Finally, the user can have a conversation with Watsonx about the information provided by Ansible.

Implementation

Main.py

```python
import sys
import os
import time
import csv
from dotenv import load_dotenv
from utils import read_default_systems, send_to_watsonxai, ansible_playbook, process_and_humani

def load_environment():
    print("Cargando variables de entorno...")
    load_dotenv(os.path.join(os.path.dirname(__file__), '..', 'config', '.env'))
    env_vars = {
        "API_KEY": os.getenv("API_KEY"),
        "IBM_CLOUD_URL": os.getenv("IBM_CLOUD_URL"),
        "PROJECT_ID": os.getenv("PROJECT_ID"),
        "ANSIBLE_TOWER_URL": os.getenv("ANSIBLE_TOWER_URL"),
        "ANSIBLE_TOWER_TOKEN": os.getenv("ANSIBLE_TOWER_TOKEN"),
        "WATSONX_ACCESS_TOKEN": os.getenv("WATSONX_ACCESS_TOKEN")
    }

    missing_vars = [key for key, value in env_vars.items() if value is None]
    if missing_vars:
        raise Exception(f"Ensure all environment variables are set in the .env file. Missing va

    print("Variables de entorno cargadas:")
    for key, value in env_vars.items():
        print(f"{key}: {value}")

    return env_vars

def save_feedback(feedback_log):
    feedback_file = os.path.join(os.path.dirname(__file__), '..', 'feedback', 'feedback_log.csv
    os.makedirs(os.path.dirname(feedback_file), exist_ok=True)

    with open(feedback_file, mode='a', newline='', encoding='utf-8') as file:
        writer = csv.writer(file)
        writer.writerow(["question", "response", "feedback", "comment"])
        for entry in feedback_log:
            writer.writerow([entry["question"], entry["response"], entry["feedback"], entry["co

def chatbot_interaction(env_vars):
    humanized_info = ""
    conversation_history = ""
    feedback_log = []
```

```
while True:
    try:
        if not humanized_info:
            print("\nWelcome to the z/OS User and Group Information Chatbot")
            entity_type = input("Do you want information about a 'user' or a 'group'? (Type
            if entity_type == 'exit':
                print("Thank you for using the chatbot. Goodbye!")
                save_feedback(feedback_log)
                break
            if entity_type not in ['user', 'group']:
                print("Invalid input. Please select 'user' or 'group'.")
                continue

            environment = input("Please provide the environment (e.g., server XYZ) or type

            csv_path = os.path.join(os.path.dirname(__file__), '..', 'config', 'default_syst
            matches = read_default_systems(csv_path, environment)

            if matches.empty:
                print(f"The environment {environment} is not in the system list.")
                continue

            if environment.lower() == 'info':
                for index, row in matches.iterrows():
                    print(f"System: {row['system']}, Host: {row['host']}")
                continue

            host = None
            for index, row in matches.iterrows():
                system_name = row['system']
                host = row['host']
                confirmation = input(f"Do you mean '{system_name}'? (yes/no): ").strip().lo
                if confirmation == "yes":
                    environment = system_name
                    break

            if host is None:
                print("No host was found for the provided environment.")
                continue

            entity_name = input(f"Please provide the {entity_type} name: ").strip()
```

```
            print("Querying the information, this may take a few moments...")
            start_time = time.time()
            output_file_path = ansible_playbook(host, entity_type, entity_name, env_vars['AN
            end_time = time.time()

            elapsed_time = end_time - start_time
            print(f"Query completed in approximately {elapsed_time:.2f} seconds.")

            if output_file_path:
                print(f"Ansible output file found: {output_file_path}")

                # Attempt to process the output and humanize it
                humanized_info, new_access_token = process_and_humanize_racf_output(
                    output_file_path, entity_type, entity_name,
                    env_vars['API_KEY'], env_vars['IBM_CLOUD_URL'],
                    env_vars['PROJECT_ID'], env_vars['WATSONX_ACCESS_TOKEN']
                )
                env_vars['WATSONX_ACCESS_TOKEN'] = new_access_token
                conversation_history += f"Humanized Information: {humanized_info}\n"
                print("\nYou can now ask questions about the information retrieved.")
            else:
                print("No information was obtained from the z/OS server.")
                continue
        else:
            user_question = input("Ask about the retrieved information (or type 'exit' to te
            if user_question.lower() == 'exit':
                print("Thank you for using the chatbot. Goodbye!")
                save_feedback(feedback_log)
                break

            prompt = f"Based on the following information: {humanized_info}\nQuestion: {user
            response, new_access_token = call_ibm_watsonx(env_vars['API_KEY'], env_vars['WAT
            env_vars['WATSONX_ACCESS_TOKEN'] = new_access_token

            if 'results' in response and 'generated_text' in response['results'][0]:
                ai_response = response['results'][0]['generated_text']
                conversation_history += f"User: {user_question}\nAI: {ai_response}\n"

                # Verificar y corregir la respuesta si es necesario
                if ai_response.lower() == 'none' or 'error' in ai_response.lower():
                    ai_response = "The response was unclear. Please ask again or rephrase yo

                print(f"AI: {ai_response}")
```

```
                    # Preguntar si le gustó la respuesta
                    feedback = input("Did you like the response? (yes/no): ").strip().lower()
                    if feedback == 'no':
                        comment = input("Please provide your feedback to improve the response: '
                        feedback_log.append({
                            "question": user_question,
                            "response": ai_response,
                            "feedback": feedback,
                            "comment": comment
                        })
                    elif feedback == 'yes':
                        feedback_log.append({
                            "question": user_question,
                            "response": ai_response,
                            "feedback": feedback,
                            "comment": ""
                        })
                else:
                    print(f"Error in response: {response}")

        except Exception as e:
            print(f"An error occurred: {str(e)}")

if __name__ == "__main__":
    env_vars = load_environment()
    chatbot_interaction(env_vars)
```

Utils.py

```
import os
import time
import requests
from requests.packages.urllib3.exceptions import InsecureRequestWarning
from ibm_watson_machine_learning.foundation_models import Model
from ibm_watson_machine_learning.metanames import GenTextParamsMetaNames as GenParams
import pandas as pd
from bs4 import BeautifulSoup

requests.packages.urllib3.disable_warnings(InsecureRequestWarning)

def read_default_systems(csv_path, environment):
    if not os.path.isfile(csv_path):
        raise FileNotFoundError(f"The file {csv_path} does not exist.")
    df = pd.read_csv(csv_path)
    matches = df[df.apply(lambda row: environment.lower() in row.astype(str).str.lower().values,
    return matches

def send_to_watsonxai(prompts, api_key, ibm_cloud_url, project_id,
                      model_name="ibm/granite-13b-chat-v2",
                      decoding_method="greedy",
                      max_new_tokens=100,
                      min_new_tokens=30,
                      temperature=1.0,
                      repetition_penalty=1.0):
    creds = {
        "url": ibm_cloud_url,
        "apikey": api_key
    }

    assert not any(map(lambda prompt: len(prompt) < 1, prompts)), "Make sure none of the prompts

    model_params = {
        GenParams.DECODING_METHOD: decoding_method,
        GenParams.MIN_NEW_TOKENS: min_new_tokens,
        GenParams.MAX_NEW_TOKENS: max_new_tokens,
        GenParams.RANDOM_SEED: 42,
        GenParams.TEMPERATURE: temperature,
        GenParams.REPETITION_PENALTY: repetition_penalty,
    }

    model = Model(
        model_id=model_name,
```

```
        params=model_params,
        credentials=creds,
        project_id=project_id
    )

    responses = []
    for prompt in prompts:
        response = model.generate_text(prompt)
        responses.append(response)
    return responses

def get_access_token(api_key):
    url = "https://iam.cloud.ibm.com/identity/token"
    headers = {
        "Content-Type": "application/x-www-form-urlencoded",
        "Accept": "application/json"
    }
    data = {
        "grant_type": "urn:ibm:params:oauth:grant-type:apikey",
        "apikey": api_key
    }
    response = requests.post(url, headers=headers, data=data)
    response.raise_for_status()
    return response.json()["access_token"]

def call_ibm_watsonx(api_key, access_token, project_id, input_text):
    url = "https://us-south.ml.cloud.ibm.com/ml/v1/text/generation?version=2023-05-29"
    body = {
        "input": input_text,
        "parameters": {
            "decoding_method": "greedy",
            "max_new_tokens": 900,
            "repetition_penalty": 1.05
        },
        "model_id": "ibm/granite-13b-chat-v2",
        "project_id": project_id
    }

    headers = {
        "Accept": "application/json",
        "Content-Type": "application/json",
        "Authorization": f"Bearer {access_token}"
    }
```

```
    response = requests.post(url, headers=headers, json=body)

    if response.status_code == 401 and "authentication_token_expired" in response.text:
        print("Token expired, refreshing token...")
        access_token = get_access_token(api_key)
        headers["Authorization"] = f"Bearer {access_token}"
        response = requests.post(url, headers=headers, json=body)

    if response.status_code != 200:
        raise Exception("Non-200 response: " + str(response.text))

    return response.json(), access_token

def ansible_playbook(host, entity_type, entity_name, ansible_tower_url, ansible_tower_token):
    try:
        headers = {
            'Authorization': f'Bearer {ansible_tower_token}',
            'Content-Type': 'application/json'
        }

        payload = {
            'extra_vars': {
                'host': host,
                'entity_type': entity_type,
                'entity_name': entity_name
            }
        }

        print("Payload sent to Ansible Tower:")
        print(payload)

        response = requests.post(f'{ansible_tower_url}/api/v2/job_templates/5097/launch/', heade
        response.raise_for_status()
        job_id = response.json().get('id')

        while True:
            job_response = requests.get(f'{ansible_tower_url}/api/v2/jobs/{job_id}/', headers=he
            job_response.raise_for_status()
            job_status = job_response.json().get('status')
            if job_status in ['successful', 'failed']:
                break
            print("Working on the query, please wait...")
```

```
            time.sleep(10)

        time.sleep(15)

        if job_status == 'successful':
            job_output_url = f'{ansible_tower_url}/api/v2/jobs/{job_id}/stdout/?format=html'
            job_output_response = requests.get(job_output_url, headers=headers, verify=False)
            job_output_response.raise_for_status()
            full_output_html = job_output_response.text

            soup = BeautifulSoup(full_output_html, 'html.parser')
            full_output = soup.get_text(separator="\n")

            start_marker = "TASK [Print RACF output parts]"
            end_marker = "PLAY RECAP"
            start_index = full_output.find(start_marker)
            end_index = full_output.find(end_marker)

            if start_index != -1 and end_index != -1:
                relevant_output = full_output[start_index:end_index]

                cleaned_output = []
                for line in relevant_output.split('\n'):
                    if "msg" in line:
                        cleaned_line = line.split("msg")[1].strip(": ").strip("\"")
                        cleaned_output.append(cleaned_line)

                cleaned_output_str = "\n".join(cleaned_output)

                output_file_path = os.path.join(os.path.dirname(__file__), '..', 'outputs', 'ans
                with open(output_file_path, 'w') as f:
                    f.write(cleaned_output_str)

                print(f"Cleaned Ansible output saved to {output_file_path}")
                return output_file_path
            else:
                print("No relevant output found in the Ansible output.")
                return None
        else:
            print("The playbook execution failed.")
            return None

    except Exception as e:
```

```
        print(f"An error occurred while executing the playbook: {str(e)}")
        return None

def process_and_humanize_racf_output(file_path, entity_type, entity_name, api_key, ibm_cloud_ur
    try:
        with open(file_path, 'r') as f:
            relevant_output = f.read()

        prompt = f"The following information is about a user from a z/OS server. Please provide
        print(f"Debug Prompt: {prompt}")

        # Send the relevant output to Watsonx
        response, new_access_token = call_ibm_watsonx(api_key, access_token, project_id, prompt
        print(f"API Response: {response}")  # Debug print

        if 'results' in response and 'generated_text' in response['results'][0]:
            humanized_response = response['results'][0]['generated_text']
            print(f"Humanized information for {entity_type} {entity_name}:")
            print(humanized_response)
            return humanized_response, new_access_token
        else:
            print(f"Key 'generated_text' not found in response: {response}")

        return None, new_access_token

    except Exception as e:
        print(f"An error occurred while processing the Ansible output file: {str(e)}")
        return None, access_token
```

Report on Time Savings Percentage for a Query System in zOS and zVM Environments

This report aims to evaluate the impact of automation on the time required to perform daily queries in zOS and zVM environments. The queries performed include searches for users, groups, resources, and datasets. The comparison is made between the time it takes a person to perform these tasks manually and the time it takes a generative artificial intelligence (AI) to carry out the same tasks.

Manual Process

In a typical zOS and zVM environment, a person dedicated to performing queries on users, groups, resources, and datasets can take approximately 4 hours daily. This time includes data collection, analysis, and the generation of necessary reports for system administration.

Automated Process

With the implementation of generative artificial intelligence, the same set of tasks can be completed more efficiently. The AI can perform queries, analyze data, and generate reports in approximately 30 minutes. This is due to the AI's capability to process large volumes of data simultaneously and without errors, eliminating the need for human intervention in repetitive tasks.

Percentage of Time Saved

The estimated manual time to perform the queries is 240 minutes, while the time with AI is 30 minutes. Therefore, the percentage of time saved is 87.50%. This demonstrates a significant reduction in the time required to complete these daily tasks.

Explanation for 100% Time Saved

If the process is fully automated with AI, achieving 100% time savings, it means that human intervention is no longer necessary. The AI can handle all tasks of querying, analyzing, and generating reports without human intervention. This not only frees up human resources for more complex and strategic tasks but also reduces the risk of human errors and increases operational efficiency.

Detailed Report on Time Savings Calculation for a Query System in zOS and zVM Environments

This report provides a detailed explanation of how the time savings percentage was calculated for performing daily queries in zOS and zVM environments using generative artificial intelligence (AI). The queries include searches for users, groups, resources, and datasets. The calculation is based on a comparison between the time taken by a person to perform these tasks manually and the time taken by the AI to complete the same tasks.

Manual Process Time Calculation

In a typical zOS and zVM environment, a person spends approximately 4 hours daily performing queries on users, groups, resources, and datasets. This time includes data collection, analysis, and report generation. For calculation purposes, the time spent is converted to minutes.

- **Manual time spent per day:** 4 hours
- Since 1 hour = 60 minutes, the total manual time is calculated as:
  - 4 hours * 60 minutes/hour = 240 minutes

Automated Process Time Calculation

With the implementation of AI, the same tasks can be performed more efficiently, reducing the time required to approximately 30 minutes.

- **Automated time spent per day:** 30 minutes

Time Savings Percentage Calculation

The percentage of time saved is calculated by comparing the manual time with the automated time using the following formula:

$$\text{Percentage of time saved} = \left(\frac{\text{Manual time} - \text{Automated time}}{\text{Manual time}} \right) \times 100$$

Substituting the values, we get:

$$\text{Percentage of time saved} = \left(\frac{240 \text{ minutes} - 30 \text{ minutes}}{240 \text{ minutes}} \right) \times 100$$

$$\text{Percentage of time saved} = \left(\frac{210 \text{ minutes}}{240 \text{ minutes}} \right) \times 100$$

$$\text{Percentage of time saved} = 0.875 \times 100$$

$$\text{Percentage of time saved} = 87.5\%$$

Conclusion

The calculation shows that the implementation of AI results in an 87.5% time savings in performing daily queries in zOS and zVM environments. This significant reduction highlights the efficiency and effectiveness of AI in handling repetitive tasks, freeing up human resources for more complex activities.