CHAT APPLICATION

TEAM MEMBERS:

2203A54007	-	K. THRISHA
2203A54016	-	P. CHANDANA
2203A54017	-	P. SRIJA
2203A54030	-	P. MEGHANA REDDY

PROBLEM STATEMENT:

Chat Application with Multi-User Roles and Features -

Design and implement a Python-based Chat Application that facilitates communication between multiple users in both group chats and personal chats. The system should support role-based access, allowing users with different roles (Admin and Member) to perform specific actions. The application should also include features like message translation, message management, and chat moderation.

ABSTRACT:

This project presents a **Chat Application** implemented in Python, featuring role-based functionality for Admins and Members. The application supports both **group chats** and **personal chats**, providing a robust platform for user interaction. The primary objectives are to enable real-time communication, ensure role-based access control, and integrate message management features.

Admins can manage chat participants, delete any message, and perform moderation tasks like kicking or promoting users, whereas Members can send messages, translate received content, and delete their own messages. Both roles are seamlessly integrated into the chat system, ensuring user-specific functionalities.

A unique feature of this application is the integration of **message translation**, powered by the googletrans library, enabling users to send and receive messages in multiple languages. This enhances usability in multilingual environments.

The project employs an **object-oriented programming (OOP)** approach, with a modular design incorporating classes such as User, Admin, Member, Message, Chat, and ChatRoom. This structure ensures code reusability, maintainability, and scalability.

The system is interactive, using a console-based menu to facilitate actions like creating chats, sending messages, viewing chat history, and managing participants. Error handling ensures smooth operation by validating inputs and restricting unauthorized actions.

This application serves as a foundation for building more advanced communication platforms, highlighting key features like **role management**, **multilingual support**, and **user-friendly interaction**.

CODE:

```
import datetime
from googletrans import Translator


# User Class (Base Class)
class User:
    def __init__(self, username, role):
        self.username = username
        self.role = role

    def send_message(self, chat, content, target_language=None):
        if target_language:
            content = self.translate_message(content, target_language)
        message = Message(self, content)
        chat.add_message(message)
        print(f"{self.username} sent the message: {content}")

    def receive_message(self, message):
```

```
        print(f"{self.username} received message: {message.content}")

    def delete_own_message(self, chat, message):
        if message.sender == self:
            chat.remove_message(message)
            print(f"{self.username} deleted their message: {message.content}")
        else:
            print("You can only delete your own messages.")

    def translate_message(self, content, target_language):
        translator = Translator()
        translation = translator.translate(content, dest=target_language)
        return translation.text

    def translate_received_message(self, message, target_language):
        translator = Translator()
        translation = translator.translate(message.content, dest=target_language)
        print(f"Translated message: {translation.text}")

# Admin Class (Inheriting from User Class)
class Admin(User):
    def __init__(self, username):
        super().__init__(username, role="admin")
        self.privileges = ["kick", "delete", "promote", "add", "remove"]

    def delete_message(self, chat, message):
        if message in chat.messages:
```

```
            chat.remove_message(message)
            print(f"Admin {self.username} deleted the message:
{message.content}")
        else:
            print("Message not found in the chat.")

    def kick_user(self, chat, user):
        if user in chat.participants:
            chat.participants.remove(user)
            print(f"Admin {self.username} kicked {user.username} from the chat.")
        else:
            print(f"User {user.username} is not in the chat.")

    def promote_user(self, user):
        if user.role != "admin":
            user.__class__ = Admin
            print(f"Admin {self.username} promoted {user.username} to Admin.")
        else:
            print(f"{user.username} is already an Admin.")

    def add_user_to_chat(self, chat, user):
        if user not in chat.participants:
            chat.participants.append(user)
            print(f"Admin {self.username} added {user.username} to the chat.")
        else:
            print(f"{user.username} is already a participant in the chat.")

    def remove_user_from_chat(self, chat, user):
```

```
        if user in chat.participants:
            chat.participants.remove(user)
            print(f"Admin {self.username} removed {user.username} from the chat.")
        else:
            print(f"{user.username} is not a participant in the chat.")

    def view_chat_history(self, chat):
        print("\nChat History:")
        for message in chat.messages:
            print(f"[{message.timestamp}] {message.sender.username}: {message.content}")

# Member Class (Inheriting from User Class)
class Member(User):
    def __init__(self, username):
        super().__init__(username, role="member")

# Message Class
class Message:
    def __init__(self, sender, content):
        self.sender = sender
        self.content = content
        self.timestamp = datetime.datetime.now()

    def edit_message(self, new_content):
        self.content = new_content
        print(f"Message edited: {self.content}")
```

```
# Chat Class
class Chat:
    def __init__(self, chat_id, participants):
        self.chat_id = chat_id
        self.participants = participants
        self.messages = []

    def add_message(self, message):
        self.messages.append(message)
        self.broadcast_message(message)

    def broadcast_message(self, message):
        for participant in self.participants:
            if participant != message.sender:  # Exclude the sender
                participant.receive_message(message)

    def remove_message(self, message):
        if message in self.messages:
            self.messages.remove(message)
        else:
            print("Message not found in chat.")

    def view_chat_history(self):
        return self.messages

# ChatRoom Class
```

```
class ChatRoom:
    def __init__(self):
        self.chats = []

    def create_group_chat(self, chat_name, participants):
        chat = Chat(chat_name, participants)
        self.chats.append(chat)
        return chat

    def create_personal_chat(self, user1, user2):
        chat_id = f"{user1.username}-{user2.username}"
        chat = Chat(chat_id, [user1, user2])
        self.chats.append(chat)
        return chat

    def get_chat(self, chat_id):
        for chat in self.chats:
            if chat.chat_id == chat_id:
                return chat
        return None

    def list_chats(self):
        return [chat.chat_id for chat in self.chats]

# Main Application Flow
def main():
    chatroom = ChatRoom()
```

```
# Create Admin and Members
admin = Admin("Thrisha")
member1 = Member("Meghana")
member2 = Member("Srija")

# Add Users to a list for interaction
users = [admin, member1, member2]

while True:
    print("\n=== Chat Application ===")
    print("1. Create a Group Chat")
    print("2. Join a Chat")
    print("3. Send Message to Group Chat")
    print("4. Send Message to Personal Chat")
    print("5. View Chat History")
    print("6. Translate a Message")
    print("7. Delete Your Own Message")
    print("8. Admin: Delete Any Message")
    print("9. Add User to Group Chat (Admin Only)")
    print("10. Remove User from Group Chat (Admin Only)")
    print("11. Exit")
    choice = input("Enter your choice: ")

    if choice == "1":
        chat_name = input("Enter the group chat name: ")
        participant_names = input("Enter usernames of participants (comma-
separated): ").split(",")
```

```
            participants = [u for u in users if u.username in participant_names]
            chat = chatroom.create_group_chat(chat_name, participants)
            print(f"Group chat '{chat_name}' created.")

        elif choice == "2":
            print("Available Chats:")
            available_chats = chatroom.list_chats()
            for idx, chat_id in enumerate(available_chats, 1):
                print(f"{idx}. {chat_id}")
            chat_choice = int(input("Select a chat by number: ")) - 1
            if 0 <= chat_choice < len(available_chats):
                chat_id = available_chats[chat_choice]
                print(f"You joined the chat: {chat_id}")
            else:
                print("Invalid chat selection.")

        elif choice == "3":
            username = input("Enter your username: ")
            user = next((u for u in users if u.username == username), None)
            if user:
                chat_name = input("Enter the chat name: ")
                chat = chatroom.get_chat(chat_name)
                if chat:
                    message = input("Enter your message: ")
                    target_language = input("Enter the target language for translation
(or leave blank for no translation): ")
                    user.send_message(chat, message, target_language if
target_language else None)
```

```
            else:
                print("Chat not found.")
        else:
            print("User not found.")


    elif choice == "4":
        sender_name = input("Enter your username: ")
        recipient_name = input("Enter recipient username: ")
        sender = next((u for u in users if u.username == sender_name), None)
        recipient = next((u for u in users if u.username == recipient_name),
None)
        if sender and recipient:
            chat_id = f"{sender.username}-{recipient.username}"
            chat = chatroom.get_chat(chat_id)
            if not chat:
                chat = chatroom.get_chat(f"{recipient.username}-
{sender.username}")
            if chat:
                message = input("Enter your message: ")
                target_language = input("Enter the target language for translation
(or leave blank for no translation): ")
                sender.send_message(chat, message, target_language if
target_language else None)
            else:
                print("No personal chat exists. Creating one.")
                chat = chatroom.create_personal_chat(sender, recipient)
                message = input("Enter your message: ")
                target_language = input("Enter the target language for translation
(or leave blank for no translation): ")
```

```
                sender.send_message(chat, message, target_language if
target_language else None)
            else:
                print("Invalid usernames.")


        elif choice == "5":
            chat_type = input("View (group/personal) chat history? ")
            if chat_type == "group":
                chat_name = input("Enter the chat name: ")
                chat = chatroom.get_chat(chat_name)
                if chat:
                    print("\nGroup Chat History:")
                    for msg in chat.view_chat_history():
                        print(f"[{msg.timestamp}] {msg.sender.username}:
{msg.content}")
                else:
                    print("Chat not found.")
            elif chat_type == "personal":
                chat_id = input("Enter the chat ID (e.g., Alice-Bob): ")
                chat = chatroom.get_chat(chat_id)
                if chat:
                    print(f"\nChat History for {chat_id}:")
                    for msg in chat.view_chat_history():
                        print(f"[{msg.timestamp}] {msg.sender.username}:
{msg.content}")
                else:
                    print("Chat not found.")
            else:
```

```
            print("Invalid option.")

        elif choice == "6":
            username = input("Enter your username: ")
            user = next((u for u in users if u.username == username), None)
            if user:
                chat_name = input("Enter the chat name: ")
                chat = chatroom.get_chat(chat_name)
                if chat:
                    print("\nChat Messages:")
                    for idx, msg in enumerate(chat.view_chat_history(), 1):
                        print(f"{idx}. [{msg.timestamp}] {msg.sender.username}:
{msg.content}")
                    message_idx = int(input("Enter the number of the message to
translate: ")) - 1
                    if 0 <= message_idx < len(chat.messages):
                        message = chat.messages[message_idx]
                        target_language = input("Enter the target language: ")
                        user.translate_received_message(message, target_language)
                    else:
                        print("Invalid message selection.")
                else:
                    print("Chat not found.")
            else:
                print("User not found.")

        elif choice == "7":
            username = input("Enter your username: ")
```

```
            user = next((u for u in users if u.username == username), None)
            if user:
                chat_name = input("Enter the chat name: ")
                chat = chatroom.get_chat(chat_name)
                if chat:
                    print("\nChat Messages:")
                    for idx, msg in enumerate(chat.view_chat_history(), 1):
                        print(f"{idx}. [{msg.timestamp}] {msg.sender.username}: {msg.content}")
                    message_idx = int(input("Enter the number of the message you want to delete: ")) - 1
                    if 0 <= message_idx < len(chat.messages):
                        message = chat.messages[message_idx]
                        user.delete_own_message(chat, message)
                    else:
                        print("Invalid message selection.")
                else:
                    print("Chat not found.")
            else:
                print("User not found.")

        elif choice == "8":
            admin_name = input("Enter admin username: ")
            admin_user = next((u for u in users if u.username == admin_name and isinstance(u, Admin)), None)
            if admin_user:
                chat_name = input("Enter the chat name: ")
                chat = chatroom.get_chat(chat_name)
```

```
            if chat:
                print("\nChat Messages:")
                for idx, msg in enumerate(chat.view_chat_history(), 1):
                    print(f"{idx}. [{msg.timestamp}] {msg.sender.username}: {msg.content}")
                message_idx = int(input("Enter the number of the message you want to delete: ")) - 1
                if 0 <= message_idx < len(chat.messages):
                    message = chat.messages[message_idx]
                    admin_user.delete_message(chat, message)
                else:
                    print("Invalid message selection.")
            else:
                print("Chat not found.")
        else:
            print("Admin not found or invalid credentials.")


    elif choice == "9":
        admin_name = input("Enter admin username: ")
        admin_user = next((u for u in users if u.username == admin_name and isinstance(u, Admin)), None)
        if admin_user:
            chat_name = input("Enter the chat name: ")
            chat = chatroom.get_chat(chat_name)
            if chat:
                new_member_name = input("Enter the username to add: ")
                new_member = next((u for u in users if u.username == new_member_name), None)
                if not new_member:
```

```
                print("User not found. Creating a new member.")
                new_member = Member(new_member_name)
                users.append(new_member)
            admin_user.add_user_to_chat(chat, new_member)
        else:
            print("Chat not found.")

    elif choice == "10":
        admin_name = input("Enter admin username: ")
        admin_user = next((u for u in users if u.username == admin_name and
isinstance(u, Admin)), None)
        if admin_user:
            chat_name = input("Enter the chat name: ")
            chat = chatroom.get_chat(chat_name)
            if chat:
                member_name = input("Enter the username to remove: ")
                member = next((u for u in users if u.username == member_name),
None)
                if member:
                    admin_user.remove_user_from_chat(chat, member)
                else:
                    print("User not found.")
            else:
                print("Chat not found.")

    elif choice == "11":
        print("Exiting the chat application.")
        break
```

```
        else:
            print("Invalid choice. Please try again.")


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

TECHNOLOGIES USED:

1. **Programming Language:**
    - **Python**: The core programming language used to implement the chat application due to its simplicity, flexibility, and rich library support.
2. **Libraries and Frameworks:**
    - **datetime**: Used to manage and format timestamps for messages.
    - **googletrans**: A library used for integrating message translation functionality, enabling multilingual support.
3. **Object-Oriented Programming (OOP):**
    - Employed to design modular and reusable code with classes like User, Admin, Member, Message, Chat, and ChatRoom.
4. **Command-Line Interface (CLI):**
    - A text-based interface to allow users to interact with the application.
5. **Data Structures:**
    - **Lists**: Used to manage participants, chats, and messages efficiently.
6. **Error Handling:**
    - Python's built-in exception handling mechanisms ensure smooth application behavior during invalid inputs or errors.

This combination of technologies ensures the application is efficient, user-friendly, and extensible for further enhancements.

CONCLUSION:

The Chat Application demonstrates the effective use of **Python** and **object-oriented programming** to design a modular, scalable, and feature-rich communication platform. By integrating role-based functionality, the system ensures appropriate access control, allowing Admins and Members to perform distinct tasks. The inclusion of **message translation** using the googletrans library highlights the application's ability to cater to multilingual environments, making it highly versatile.

The code successfully achieves the following objectives:

1. Provides seamless communication through group and personal chats.
2. Implements robust role-based operations for Admins and Members.
3. Facilitates dynamic message management, including translation, deletion, and broadcasting.
4. Ensures user-friendly interaction via a command-line interface.

This project lays a strong foundation for more advanced chat systems with potential enhancements such as real-time messaging, graphical user interfaces, and database integration for persistence. The application showcases how simple yet effective design principles and Python's capabilities can be leveraged to create practical software solutions for communication needs.