Group 7： “Flower bottle" vases

Group Members: Yixin Chen**,** Zhiyang She, Jue Wang**,** Wenjiao Guo

Github link: https://github.com/sophiashe123456/chatsystem_sp_ass.git

# Chat System function introduce

This is a simple socket-based chat system based on python programming language that supports the following features:

- Broadcast messages
- Private messages
- File transfer
- List online users
- List groups
- Group chat
- Group file transfer
- List group members
- User exit from the chat system

# Commands Overview

- **Public Message**: Just type the message and press Enter.
- **Private Message**: /msg <username> <message>
- **File Transfer**: /file <username> <filename>
- **List Online Users**: /list
- **List Groups**: /showgroups
- **Create a Group**: /create <groupname>
- **Join a Group**: /join <groupname>
- **Send a Group Message**: /group <groupname> <message>
- **Quit**: /quit
- **List group members name:** /showgroupmembers <groupname>
- **Send a file within the group:** /groupfile <groupname> <filename>

# Prepare

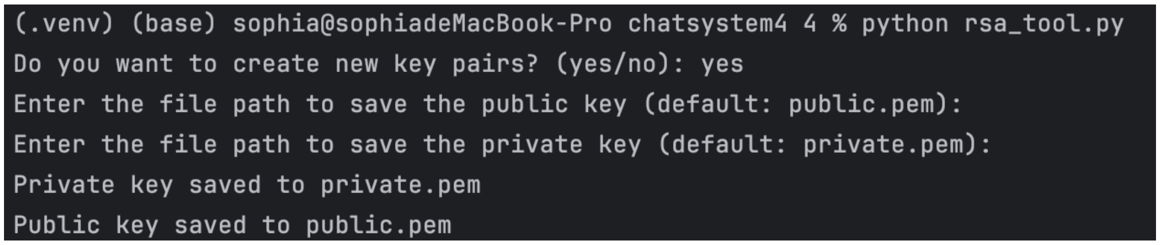
1. Clone or download the project files, including `server.py` and `client.py` etc.
2. Ensure `server.py` and `client.py` are in the same directory.
3. Run the chat system from the terminal or select other appropriate environment (VS Code, PyCharm etc.).
4. Ensure you have all the needed libraried:

```
pip install websocket
pip install cryptography
pip install pycryptodome
pip install aioconsole
```

5. Generate RSA keys in the same directory:

```
python rsa_tool.py
```

--------- As following --------



```
(.venv) (base) sophia@sophiadeMacBook-Pro chatsystem4 4 % python rsa_tool.py
Do you want to create new key pairs? (yes/no): yes
Enter the file path to save the public key (default: public.pem):
Enter the file path to save the private key (default: private.pem):
Private key saved to private.pem
Public key saved to public.pem
```

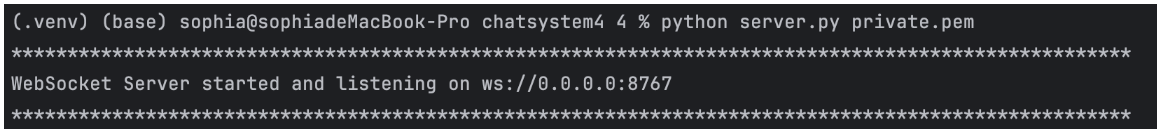
6. Check your own IP address, the following use 192.168.1.104 as example.

# Instruction

● Starting the Server

- **Start the Server**: Run the server script (server.py) use following code:
  python server.py private.pem
- You should see a message indicating that: WebSocket server started and listening on ws://0.0.0.0:8767.

--------- As following --------

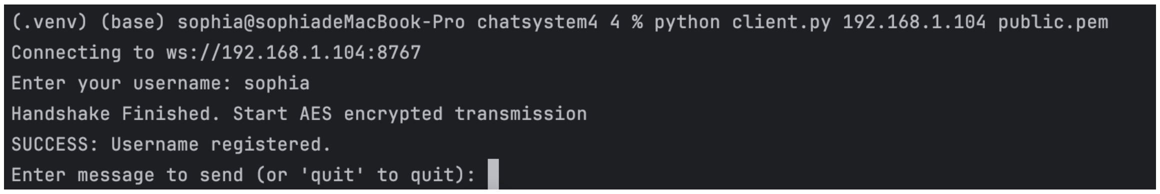


```
(.venv) (base) sophia@sophiadeMacBook-Pro chatsystem4 4 % python server.py private.pem
*********************************************************************************************
WebSocket Server started and listening on ws://0.0.0.0:8767
*********************************************************************************************
```

● Connecting Clients

- **Start Client for User sophia**: Run the client script (client1.py) for the first user:
  python client.py 192.168.1.104 public.pem (change to your server IP address)
- **Enter the username when prompted:** Enter your username: sophia
- **Start Client for User jojo**: Run another instance of the client script for the second user: python client.py 192.168.1.104 public.pem
- **Enter the username when prompted:** Enter your username: jojo

--------- As following --------



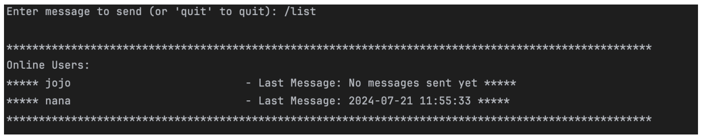
```
(.venv) (base) sophia@sophiadeMacBook-Pro chatsystem4 4 % python client.py 192.168.1.104 public.pem
Connecting to ws://192.168.1.104:8767
Enter your username: sophia
Handshake Finished. Start AES encrypted transmission
SUCCESS: Username registered.
Enter message to send (or 'quit' to quit):
```

● Using the Chat System

Once both clients are connected, you can use the following commands to interact:

➢ **Check online users**

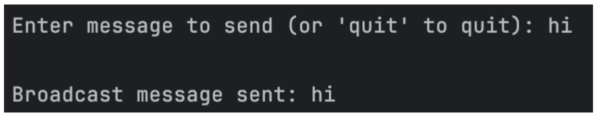
- ✓ Show all active online users: Use ***/list*** command. Example:



```
Enter message to send (or 'quit' to quit): /list

**************************************************************************************************
Online Users:
***** jojo                          - Last Message: No messages sent yet *****
***** nana                          - Last Message: 2024-07-21 11:55:33 *****
**************************************************************************************************
```

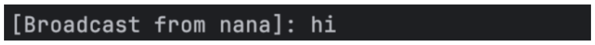
➢ Public Messages

- ✓ **Send a Public Message**: Any message that does not start with a command (/msg, /file, /list, /group, /create, /join, /quit.) will be broadcast to all connected users. Example:
  - User nana types： hi
  - All other users should see: [Broadcast from nana]: hi



```
Enter message to send (or 'quit' to quit): hi

Broadcast message sent: hi
```

(nana send)

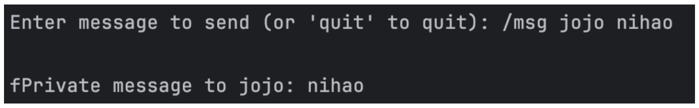


```
[Broadcast from nana]: hi
```

(others receive)

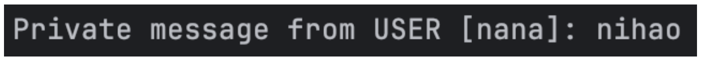
➢ Private Messages

- ✓ **Send a Private Message**: Use the '/msg receive_client_name message' command to send a private message to a specific user. Example:
  - User sophia types: ***/msg*** jojo How are you?
  - Only User jojo should see: Private message from sophia: How are you?
  - User sophia will see a confirmation: Private message to jojo: How are you?



```
Enter message to send (or 'quit' to quit): /msg jojo nihao

fPrivate message to jojo: nihao
```

(nana send)

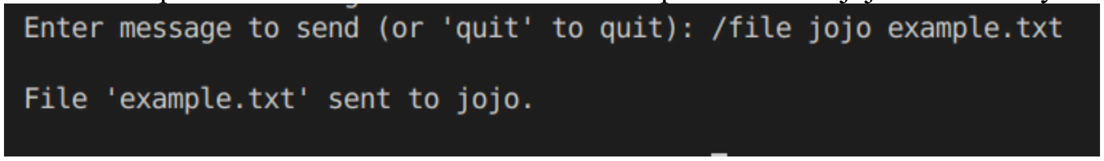


```
Private message from USER [nana]: nihao
```

(jojo receive)

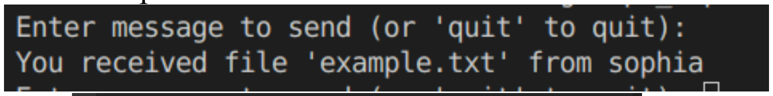
➢ File Transfer

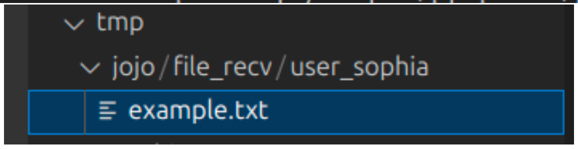
- ✓ **Send a File**: Use the '***/file*** receive_client_name path/file_name' command to send a file to a specific user. Example:
  - User sophia wants to send a file named example.txt to User jojo. User sophia types: /file jojo example.txt
  - User sophia will see a confirmation: File example.txt sent to jojo successfully.



```
Enter message to send (or 'quit' to quit): /file jojo example.txt

File 'example.txt' sent to jojo.
```

- User jojo will receive the file, automatically download in her computer and open the file in a separate window and receive a notification: You received file 'example.txt' from sophia.

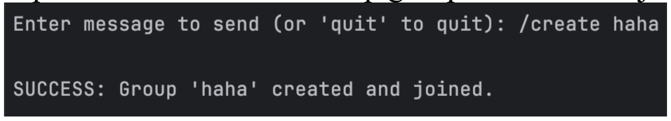

```
Enter message to send (or 'quit' to quit):
You received file 'example.txt' from sophia
```


```
∨ tmp
  ∨ jojo / file_recv / user_sophia
      example.txt
```

➢ Group Chat

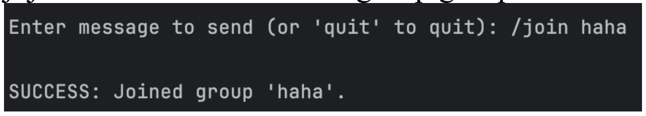
✓ **Create a group**: Use the ***/create*** command to create a chat group. Example:

- User sophia wants to create a chat-group groupa. User sophia types the following command and will automatically join the group herself: */create groupa*
- sophia should receive: Group groupa created and joined.

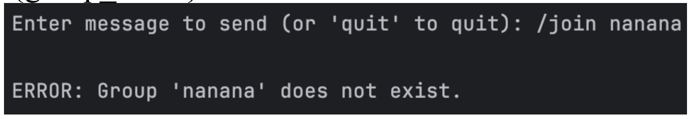

```
Enter message to send (or 'quit' to quit): /create haha

SUCCESS: Group 'haha' created and joined.
```

✓ **Join a group**: Use the ***/join*** command to join a chat group. Example:

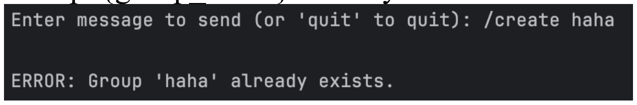
- Anyone else that wants to chat with all the members in groupa at the same time need to join the group. If jojo wants to join the group, she types: */join groupa.*
- jojo should receive: Joined group groupa.


```
Enter message to send (or 'quit' to quit): /join haha

SUCCESS: Joined group 'haha'.
```

- If clients join some group does not exit, they should receive: Group '{group_name}' does not exist.

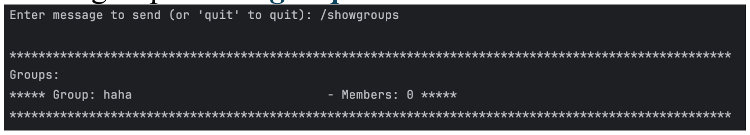

```
Enter message to send (or 'quit' to quit): /join nanana

ERROR: Group 'nanana' does not exist.
```

- If clients create a group using the exiting group name, they should receive: Group '{group_name}' already exists.


```
Enter message to send (or 'quit' to quit): /create haha

ERROR: Group 'haha' already exists.
```

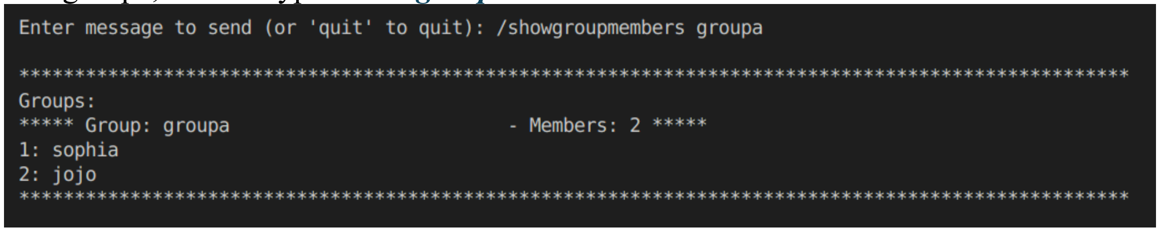
✓ **Show group details**. Example:

- If clients want to check what groups are there, need to type the following to show all the groups: ***/showgroups***



```
Enter message to send (or 'quit' to quit): /showgroups

****************************************************************************************************
Groups:
***** Group: haha                           - Members: 0 *****
****************************************************************************************************
```

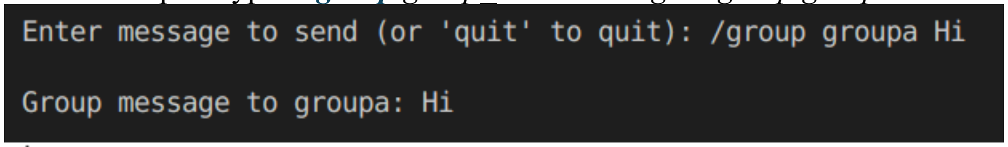
- If clients joined groupa and want to check the name of all the members in the groupa, need to type: ***/showgroupmembers***



```
Enter message to send (or 'quit' to quit): /showgroupmembers groupa

****************************************************************************************************
Groups:
***** Group: groupa                           - Members: 2 *****
1: sophia
2: jojo
****************************************************************************************************
```

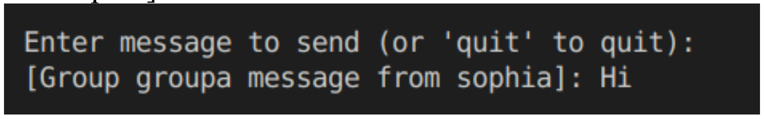
✓ **Send a group message.** Example:

- User sophia types '***/group*** *group_name message*': */group gourpa Hi.*



```
Enter message to send (or 'quit' to quit): /group groupa Hi

Group message to groupa: Hi
```

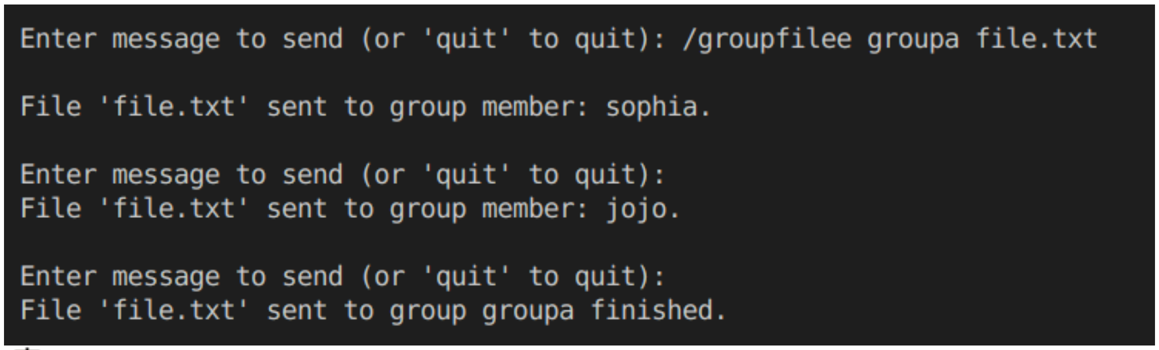
- Only group member in groupa should see: [Receive group groupa message from sophia]: Hi.



```
Enter message to send (or 'quit' to quit):
[Group groupa message from sophia]: Hi
```

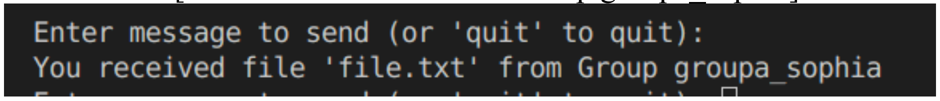
✓ **Send a file within the group.** Example:

- User sophia types '***/groupfile*** *group_name path/file_name*': /groupfile gourpa file.txt.



```
Enter message to send (or 'quit' to quit): /groupfilee groupa file.txt

File 'file.txt' sent to group member: sophia.

Enter message to send (or 'quit' to quit):
File 'file.txt' sent to group member: jojo.

Enter message to send (or 'quit' to quit):
File 'file.txt' sent to group groupa finished.
```

- All the group members in groupa should receive the file, automatically download in their computer and open the file in a separate window and receive a notification: [You receive file.txt from Group groupa_sophia].



```
Enter message to send (or 'quit' to quit):
You received file 'file.txt' from Group groupa_sophia
```

# Example Chat Session

Here is an example chat session with the commands, sophia and jojo are the only clients connected with server1 at the moment:

1 **Public Message**:
- **Sophia** types: Hello everyone!
- **Jojo** sees: [Broadcast from sophia]: Hello everyone!

2 **List clients**:
- **Sophia** types: /list
- **Sophia** sees: sophia, jojo

3 **Private Message**:
- **Sophia** types: /msg jojo How are you?
- **Jojo** sees: Private message from sophia: How are you?
- **Sophia** sees: Private message to jojo: How are you?

4 **File Transfer**:
- **Sophia** types: /file jojo example.txt
- **Sophia** sees a notification: File example.txt sent to jojo successfully.
- **Jojo** receives the file, file download in jojo's computer and opened in a separate window, and jojo sees a notification: You received file example.txt from sophia.

5 **Group Message**:
- **Sophia** types: /create haha
- **Sophia** sees: Group haha created and joined.
- **Jojo** types: /showgroups
- **Jojo** sees: haha
- **Jojo** types: /join haha
- **Jojo** sees: Joined group haha
- **Jojo** types: /showgroupmembers
- **Jojo** sees: jojo sophia
- **Jojo** types: /group haha Hello there!
- **Sophia** sees: [Receive group haha message from jojo]: Hello there!
- **Sophia** types: /groupfile haha file.txt
- **Jojo** sees: [Receive file.txt from Group haha sophia].

If client nana is in the group, she should receive the message and file same as jojo and sophia at the same time.

# Testing plan

1. Testing Objectives

Verify the functionality, security, performance, and code quality of the chat system, ensuring it meets expectations and identifying potential vulnerabilities and areas for improvement.

2. Testing Methods

- Functional

- Message Functionality: Test sending and receiving text messages, private and group chats, special characters, and emojis
- User Management: Test user registration, login, logout, and ensure usernames do not contain spaces
- File Transfer: Test file upload and download, performance, and stability of large file transfers

- Security

  - Encryption/Decryption: Test RSA and AES encryption/decryption functionality, upgrade from AES-128 to AES-256
  - Key Exchange: Test the integrity and authentication of the key exchange process, simulate man-in-the-middle attacks
  - Input Validation: Validate and sanitize user inputs and filenames to prevent injection and directory traversal attacks
  - Intentional Backdoors: Identify and exploit five intentional backdoor vulnerabilities

- Performance

  - Concurrent User Handling: Simulate high concurrent user operations, test system response and stability
  - File Transfer Efficiency: Test speed and success rate of transferring files of different sizes, optimize transfer methods

- Code Quality

  - Code Style: Use Pylint to check code style consistency, address line length issues and unused variables
  - Modularity: Break down large functions into smaller units, standardize string formatting (prefer f-strings)
  - Documentation and Comments: Ensure detailed comments throughout the code, enhance README file with setup instructions and security warning

3. Testing Tools

Tool Type	**Specific Tools**
Automated Testing Tools	Pylint, DeepSource, Snyk
Manual Review Tools	Code Review, Penetration Testing
Performance Testing Tools	Apache JMeter, Locust

4. Testing Process

- Preparation

  - Set up testing environment: Configure test servers and clients, install testing tools
  - Prepare test data: Create test users and files, prepare test cases and scripts

- Execution

  - Functional Testing: Execute message functionality, user management, and file transfer tests

- Security Testing: Perform encryption/decryption, key exchange, input validation, and intentional backdoor tests
- Performance Testing: Use tools to simulate high concurrent user operations, record response times and performance data
- Code Quality Testing: Use Pylint to check code quality, manually review comments and documentation

- Analysis and Improvement

  - Analyze Test Results: Evaluate completeness and correctness of functionalities, security, performance, and code quality
  - Submit Test Report: Summarize identified issues and improvement suggestions, submit a detailed report
  - Issue Resolution and Retesting: Fix issues based on the report, optimize code, and retest to ensure problems are resolved