COMP90015 Assignment 2 Report

Content

Haofeng Chen
(1400978)

1. System Architecture

In my design, my distributed shared whiteboard enables the multiple users including the manager and the other peers to draw simultaneously on a whiteboard. The image in the whiteboard, manager name and other usernames will be stored in the server. Moreover, the manager has his own access such as opening a saved image, saving an image, kicking out a user and so on. The RMI structure is used to control the image in the whiteboard, manager name and other usernames. And TCP connection is also used to deal with the client's request such as creating a new whiteboard or kicking out a user. There are totally four threads. The first thread can be used to deal with user requests while the second and third thread can be used to deal with manager actions and user actions respectively. The fourth thread is used to update the user list in the whiteboard. Synchronization is used as well to ensure each thread won't interfere with others when executing the commands.

2. Communication Protocols and Message Format

One communication protocol I used is RMI, handling with one remote object call RemoteCanvas for drawing shapes in the whiteboard and the other remote object called RemoteUser for getting and setting manager name with adding usernames as well. Then I implement two remote interfaces in a class called WhiteboardImpl which represents the actual remote objects whose methods can be invoked remotely. Meanwhile, binding needs to be mentioned because the server needs to bind its remote object implementation to the name in RMI registry, allowing users to look up remote object through its names "RemoteCanvas" and "RemoteUser". (Figure 1) Furthermore, in order to solve the problem how to keep the image concurrency, I serialize the image, which means that I convert the image to a byte format to communicate between client and server, and in the other client sides, the image will be composed of bytes.

```
RemoteCanvas remoteCanvas = new WhiteboardImpl();
RemoteUser remoteUser = new WhiteboardImpl();
Registry registry = LocateRegistry.createRegistry(port);
registry.rebind( name: "RemoteCanvas", remoteCanvas);
registry.rebind( name: "RemoteUser", remoteUser);
```

(Figure 1)

There are some drawbacks of RMI. The one is that RMI heavily depends on the network communication, so some issues related to Internet like latency and network failures will lead to the failure of RMI. The other drawback can be not scale well for large-scale systems. With an increasing number of clients and servers, the RMI may be hard to manage a multitude of network connections between clients and servers effectively and efficiently.

However, when it comes to the advantages of RMI, RMI provides a simple and effective way of implementing distributed systems in java. It reduces the complexity of network communication, so that the developers can put more emphasis on the system details and logic. Moreover, RMI offers different security mechanisms to protect the communication between client and server, integrating with Java's security infrastructure to enhance access control.

The other communication protocol is TCP which I learned in the first part of this semester. I mainly use TCP sockets to communicate the request from user and manager, such as creating a new whiteboard or joining an existing whiteboard, which will be necessary to send request to the server and receive relative response. I will list where I use TCP below.

Server-side:

1. Create a new whiteboard.
2. Join an existing whiteboard.
3. The result of joining whiteboard.
4. Kick out a user.
5. User closes the whiteboard.
6. Manager closes the whiteboard.
7. Manager opens a file.
8. Deal with messages sending from various users.

Manager-side:

1. Response to the joining request.
2. Receive the messages from various users.

User-side:

1. Receive kick out notification.
2. Receive the notification of manager closing the whiteboard.
3. Receive the notification of manager opening a file.
4. Receive the messages from the manager and various users.

The message format is JSON string format. For example, through getting JSON string "Request" part, the server will know how to execute the next command. If the manager's name exists, the managerExist method will be executed. Otherwise, createWhiteboard function will be called.(Figure 2 &3)

```
Send: {"Request":"Create Whiteboard","Manager Name":"Kenny"}
Receive: {"Response":"Created Successfully","Manager Name":"Kenny"}
```

(Figure 2)

```
String requestType = (String) object.get("Request");
switch (requestType) {
    case Connection.createWhiteboard -> {
        if (manager.getManagerName() != null) {
            socket.managerExist();
        } else {
            String managerName = manager.addNewUser((String) object.get("Manager Name"));
            manager.setManagerName(managerName);
            manager.addUserSocket(managerName, socket);
            socket.createWhiteboard(managerName);
        }
        break;
    }
```

(Figure 3)

The advantages of TCP are that TCP provides reliable connection between client and server, guarantying that the data packets can arrive the destination without any errors, and TCP can detect errors in the data packets in case that the affected packets may still be under transmission. On the other hand, there are a few disadvantages of TCP

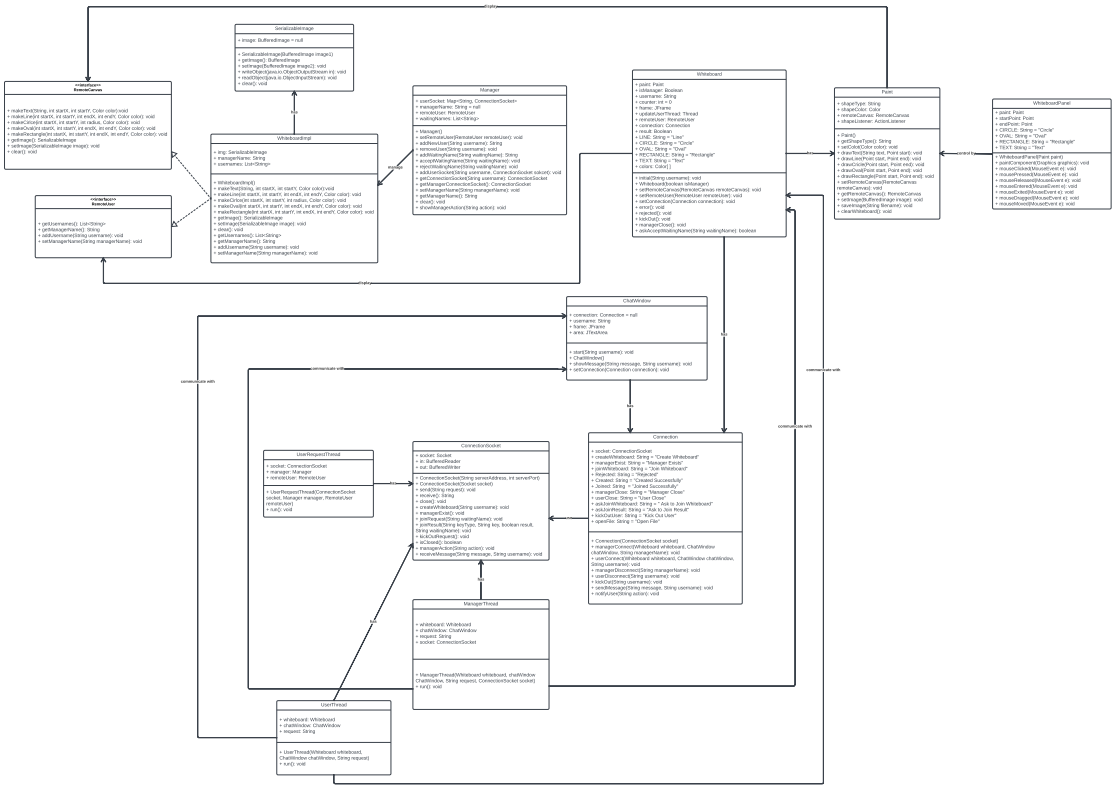
such as latency during the data transmission and limited scalability in highly distributed and scalable systems.

3. Design Diagrams

In the figure 4, the WhiteboardImpl class implements two interfaces named RemoteCanvas for drawing shapes and RemoteUser for getting and setting manager name with adding new user. There is a SerializableImage class to convert the buffered image to byte format in order to transfer data. There is also a Manager class to perform the management actions regarding remote users.

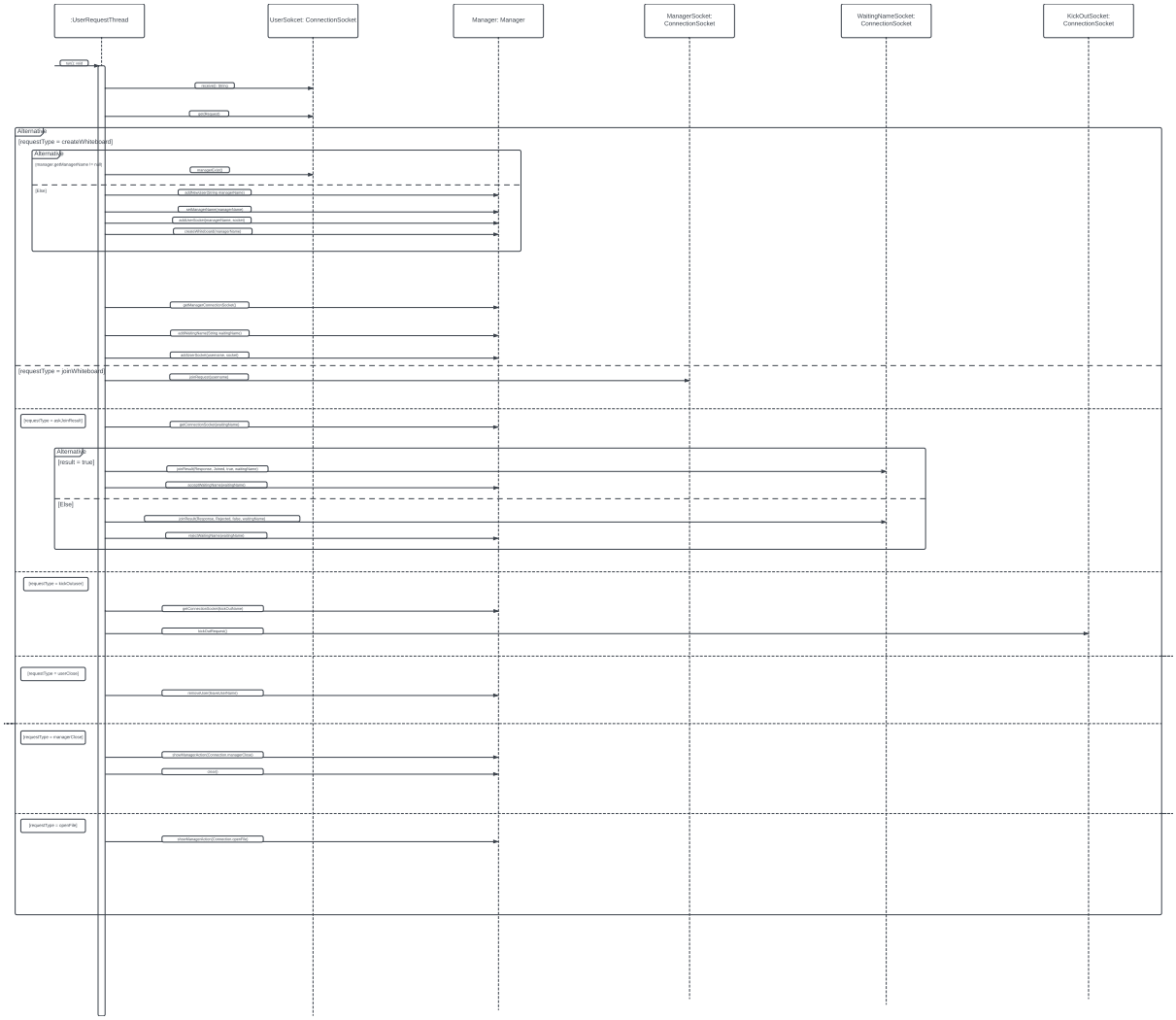
Whiteboard class can be seen the most essential class because it owns the Paint class which has the various methods of drawing different shapes and setting different colours. Meanwhile, the Paint class is controlled by WhiteboardPanel class which has different mouse actions. The Whiteboard also displays the remote users including the manager's name and other peer names separately.

When it comes to the connection part, there is a UserRequestThread working at the server to deal with the users' requests and sending response through the methods in ConnectionSocket class. Connection class is used to maintain the connection or execute the disconnection on the manager and user sides. ManagerThread class means receiving the request from server and providing a response to the server while UserThread class represents receiving the request from server and sending response to the server.

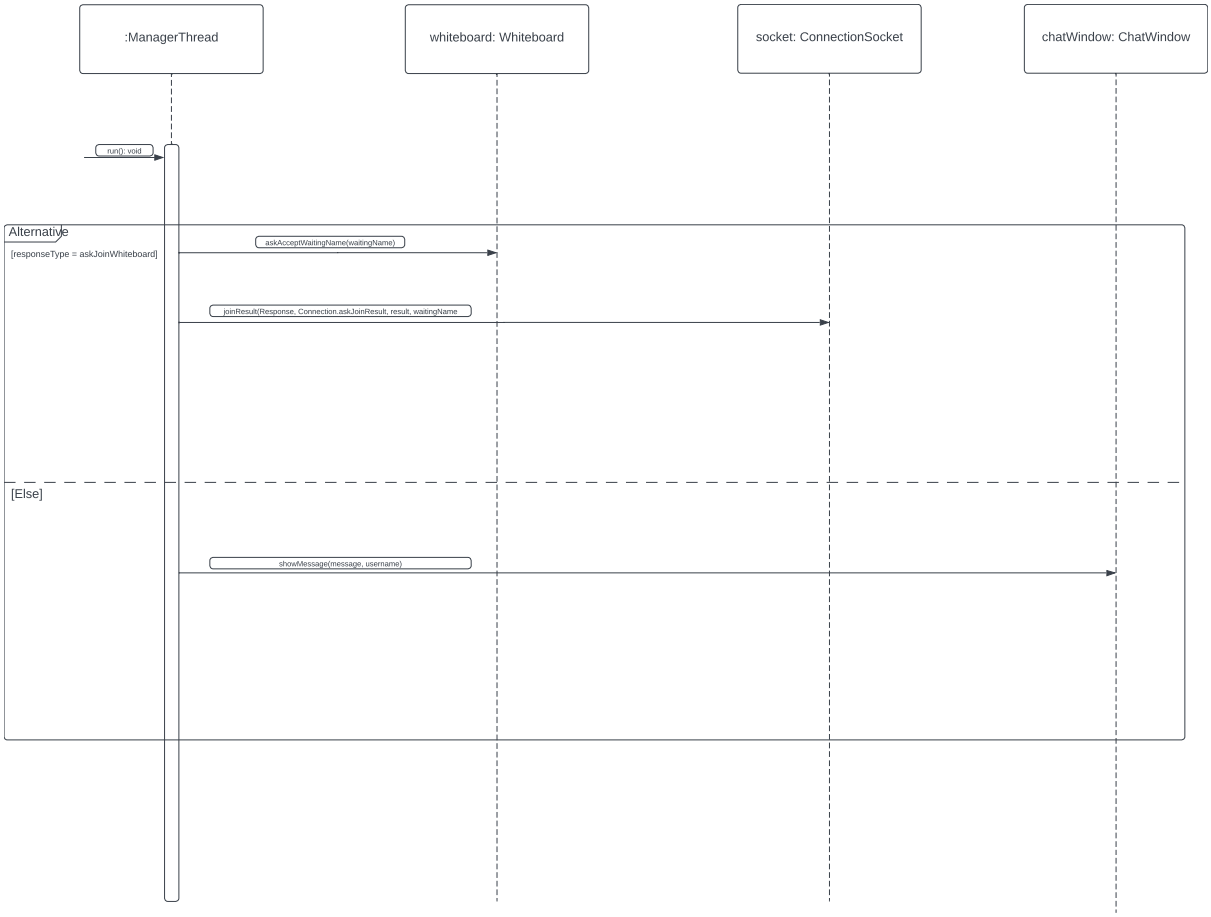


(Figure 4)

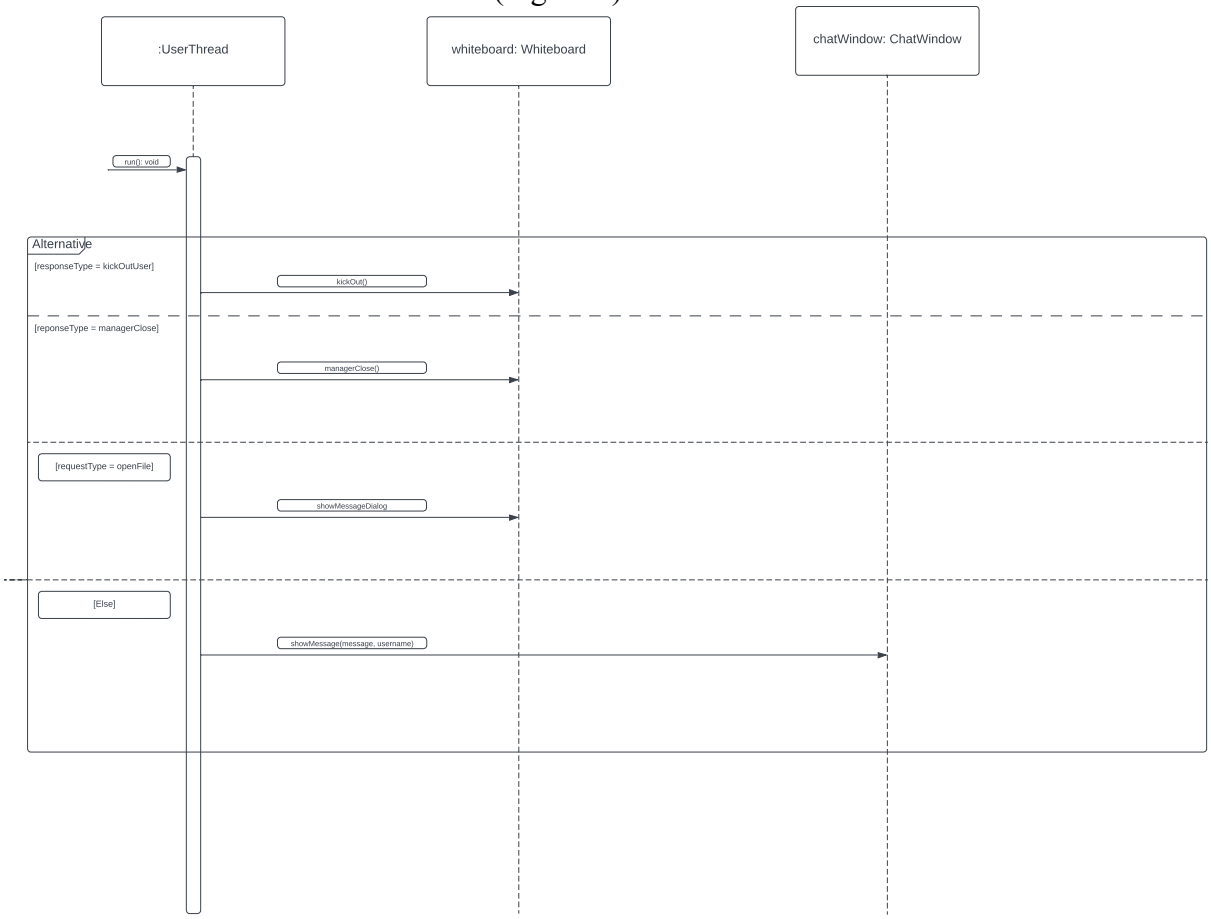
Figure 5 demonstrates the communication of UserRequestThread class.
Figure 6 demonstrates the communication of ManagerThread class.
Figure 7 indicates the communication of UserThread class.
Figure 8 indicates the communication of WhiteboardPanel class.


:UserRequestThread
UserSokcet: ConnectionSocket
Manager: Manager
ManagerSocket: ConnectionSocket
WaitingNameSocket: ConnectionSocket
KickOutSocket: ConnectionSocket
Alternative
[requestType = createWhiteboard]
Alternative
[else]
[requestType = joinWhiteboard]
[requestType = askJoinResult]
Alternative
[result = true]
[Else]
[requestType = kickOutUser]
[requestType = userClose]
[requestType = managerClose]
[requestType = openWB]

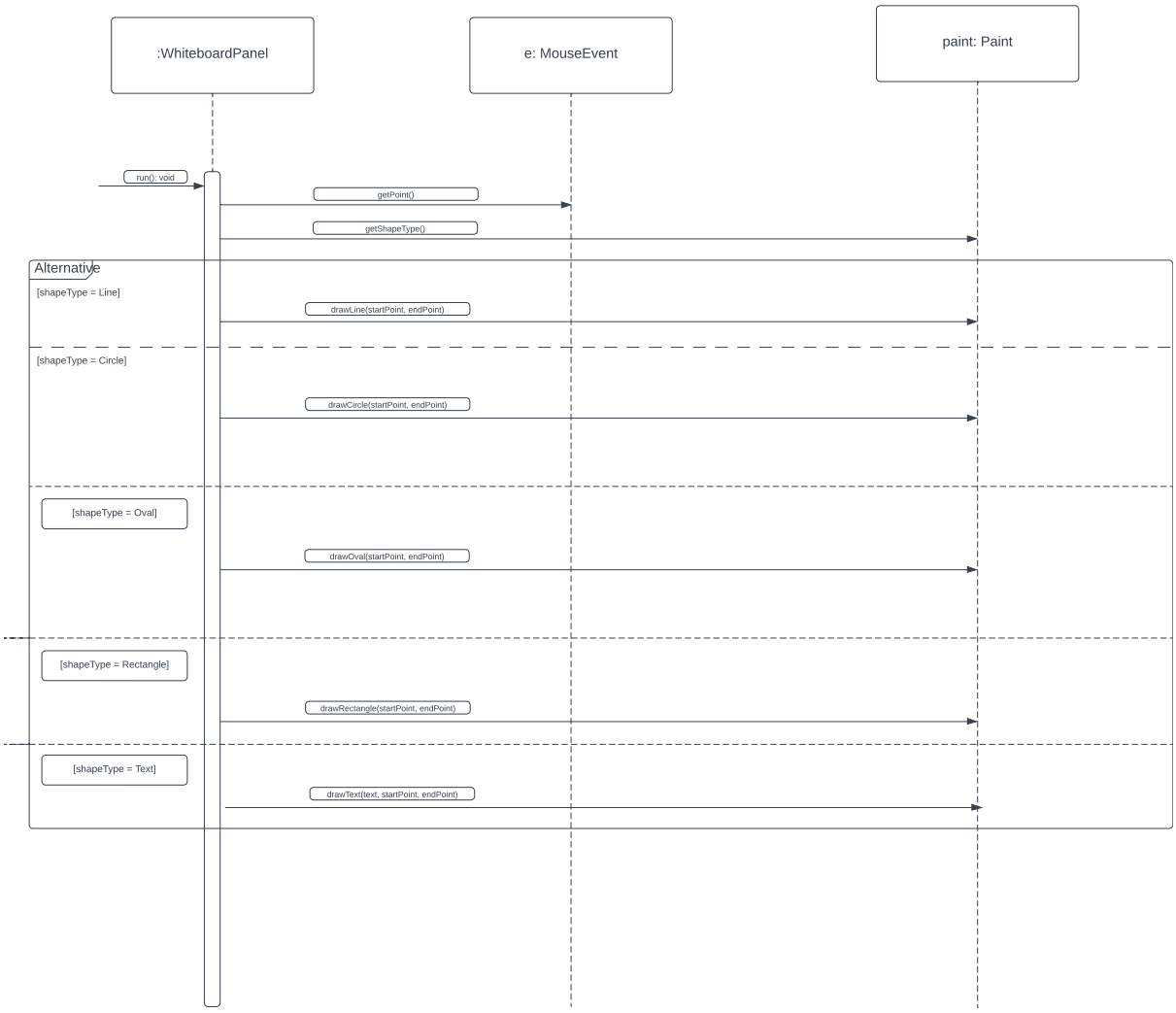
(Figure 5)


:ManagerThread
whiteboard: Whiteboard
socket: ConnectionSocket
chatWindow: ChatWindow
run(): void
Alternative
[responseType = askJoinWhiteboard]
askAcceptWaitingName(waitingName)
joinResult(Response, Connection.askJoinResult, result, waitingName
[Else]
showMessage(message, username)

(Figure 6)


:UserThread
whiteboard: Whiteboard
chatWindow: ChatWindow
run(): void
Alternative
[responseType = kickOutUser]
kickOut()
[reponseType = managerClose]
managerClose()
[requestType = openFile]
showMessageDialog
[Else]
showMessage(message, username)

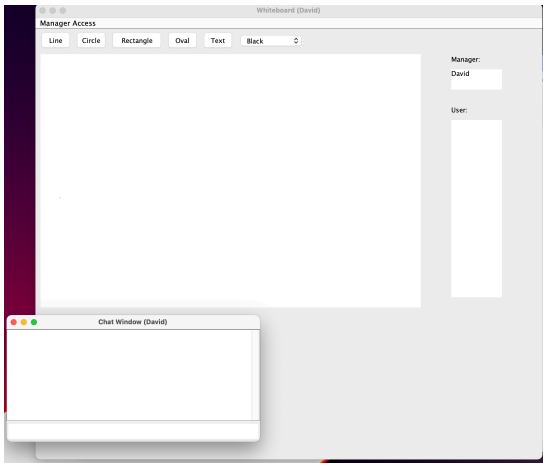
(Figure 7)



(Figure 8)

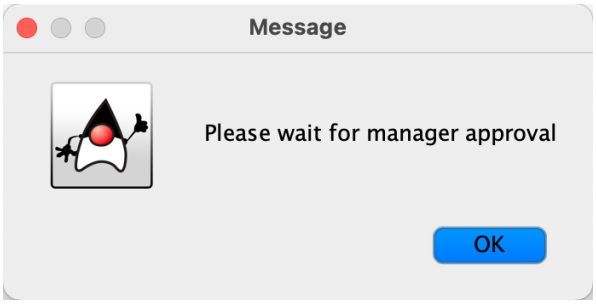
4. Implementation Details

After launching the server and CreateWhiteboard, a whiteboard with a chat window in the manager's view will be created. Manager name will be shown in the Manager area. (Figure 9)



(Figure 9)

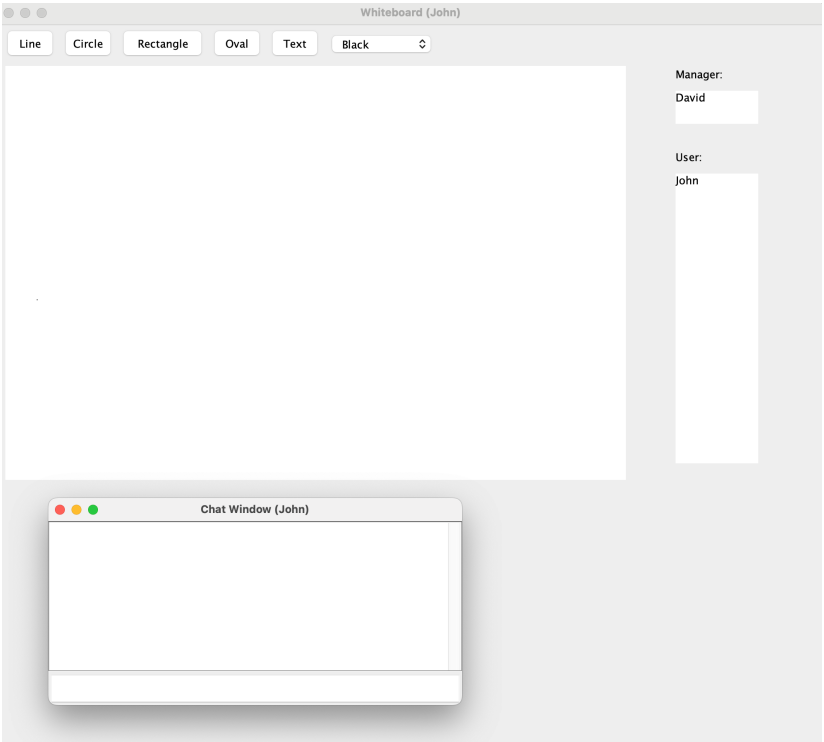
When a user wants to join the whiteboard, he will be notified “Please wait for manager approval”. (Figure 10) Meanwhile, in the manager view, the manager will receive a join request and he can choose Yes or No. (Figure 11)



(Figure 10)

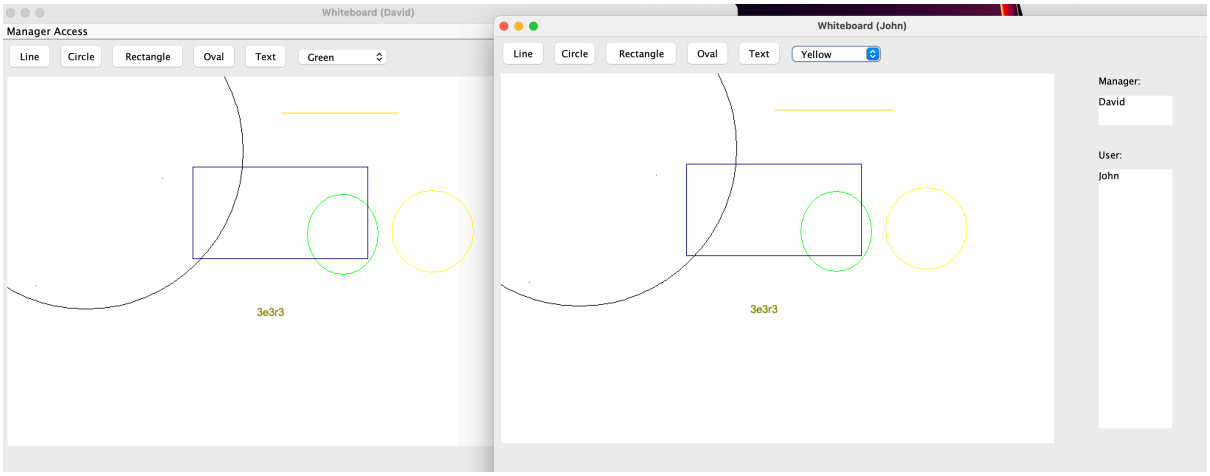
(Figure 11)

If the manager approves the join request, a whiteboard with a chat window in the user’s view will be created. The username will be displayed in the user area. (Figure 12)

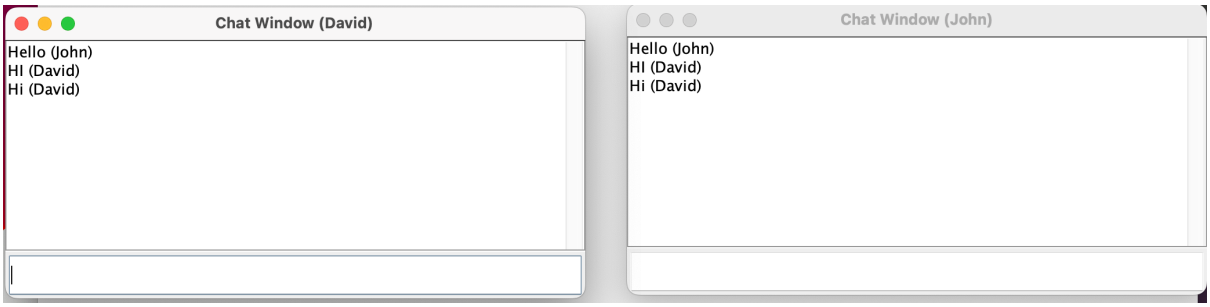


(Figure 12)

Then the manager and users can draw shapes in the whiteboard simultaneously. And they can also chat with each other via chat window. (Figure 13 & 14)

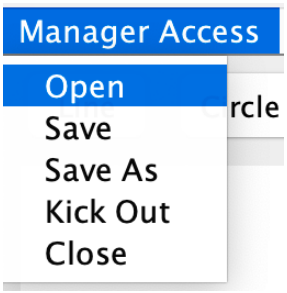


(Figure 13)



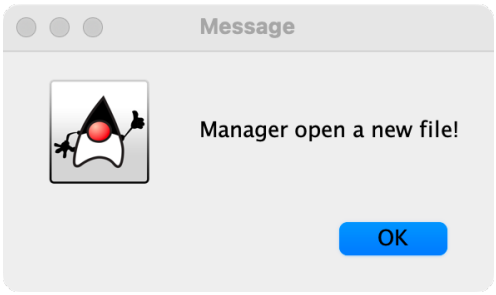
(Figure 14)

In the manager's view, he has his own access like open, save, save as, kick out and close. (Figure 15)

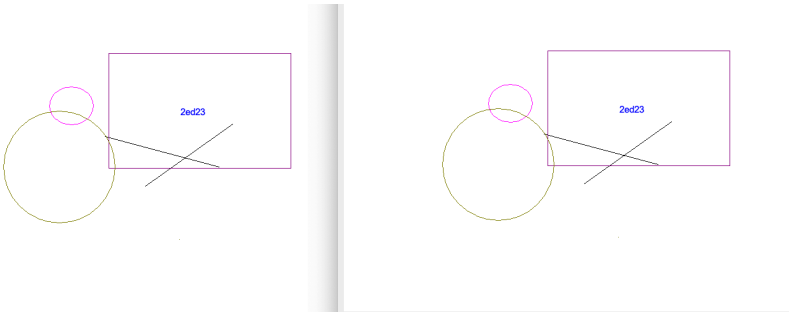


(Figure 15)

If the manager clicks the "Open" button, the users will receive the message "Manager open a new file!" and the whiteboard will display the image. (Figure 16 & 17)

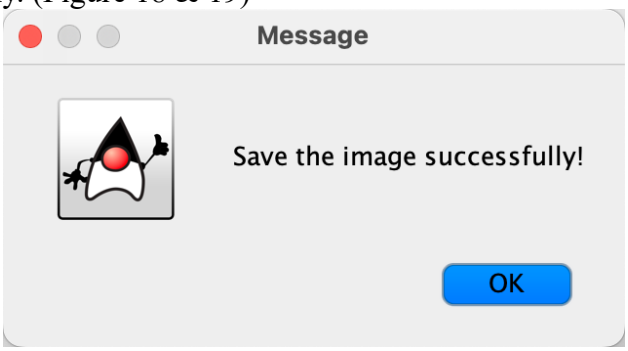


(Figure 16)

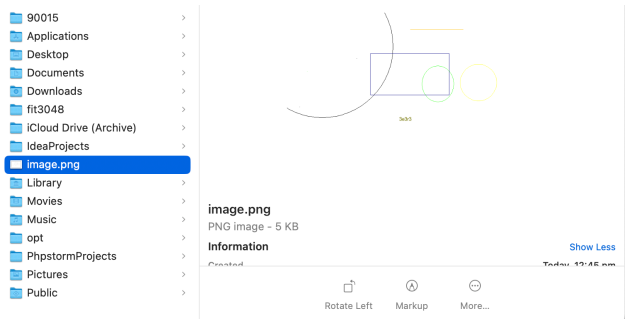


(Figure 17)

If the manager click “Save” button, the image will be saved and then the window will pop up a message “Save the image successfully!”. And the image will be store in the finder automatically. (Figure 18 & 19)

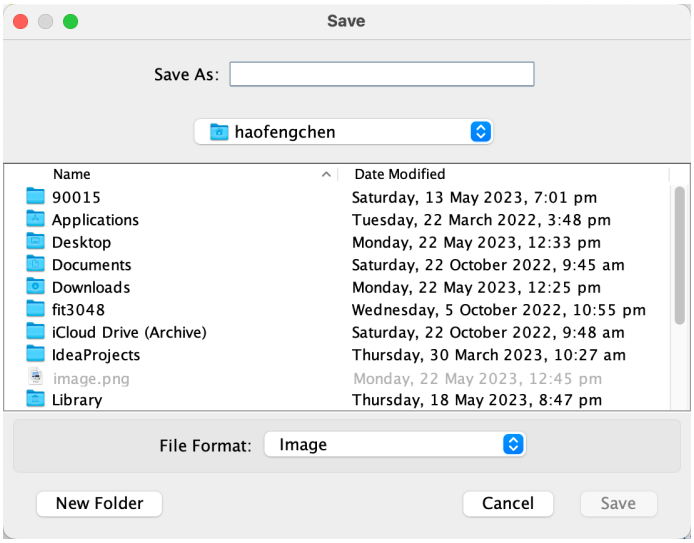


(Figure 18)



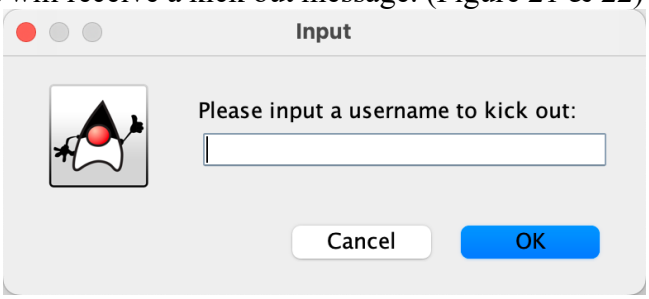
(Figure 19)

If the manager clicks the “Save as” button, a save interface will be shown and the manager can choose where to save the image. (Figure 20)

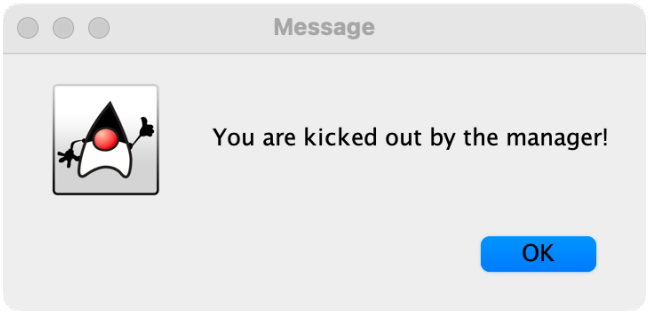


(Figure 20)

If the manager clicks “Kick Out” button, an input dialog will pop up to allow manager to input a username to kick out. If the username exists in the user list, the user who will be kicked out will receive a kick out message. (Figure 21 & 22)

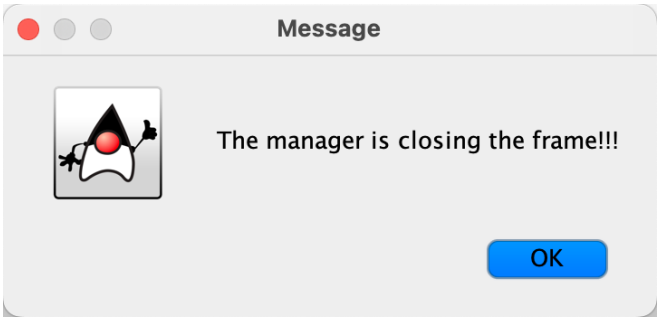


(Figure 21)



(Figure 22)

If the manager clicks the “close” button, all the whiteboards will be closed, and the users can receive the message “The manager is closing the frame!”. (Figure 23)



(Figure 23)