DISTRIBUTED SHARED WHITE BOARD

Assignment 2

Josh Feng

1266669

MAY 4, 2024

UNIVERSITY OF MELBOURNE

COMP90015 - Distributed System

1. System Architecture

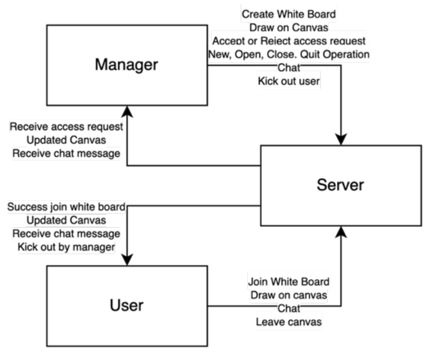


Figure 1: System Architecture Diagram

Figure 1 shows the system architecture of my program, I am using server-centric, there exists one server, one manager and multiple users in the system. Any interactions between the manager and the users will first be sent to the server for unified management scheduling and synchronization of information. This approach improves the manageability of the system because all message interactions are recorded and controlled by the server.

Server: The server is the core of the system, it is responsible for handling all request from managers and users and forwarding them accordingly. For example, if a manager sends a request to the server for kick out user1, the server will query to find user1 and kick it out and update the user list to all users and manager. In addition, the system used RMI server, and TCP as the protocol for remote method calls.

Manager and User: Manager and users have different permissions, but their actions are sent to the server. For example, when a user draws a line, the server will be receiving the action and updates the canvas to all users.

Benefit:

- It makes the system easy to manage, improves the controllability of the system and avoids causing some users to be unable to get the latest data.
- All data storage and processing are done on the central server, simplifying data maintenance to avoid the need for redundant storage of data in multiple locations. And centralized management facilitates data consolidation and updating and improves data integrity.
- All users have access to a consistent data and service experience. This is particularly important for providing consistent and reliable user services, especially for this project.

2. Communication Protocols and Message Formats

I used RMI as the communication protocol in my system, which is a java technology that allows the methods of an object to be invoked remotely between different JVMs. I have three remote objects in my system, canvas, client, and server (Figure 2).

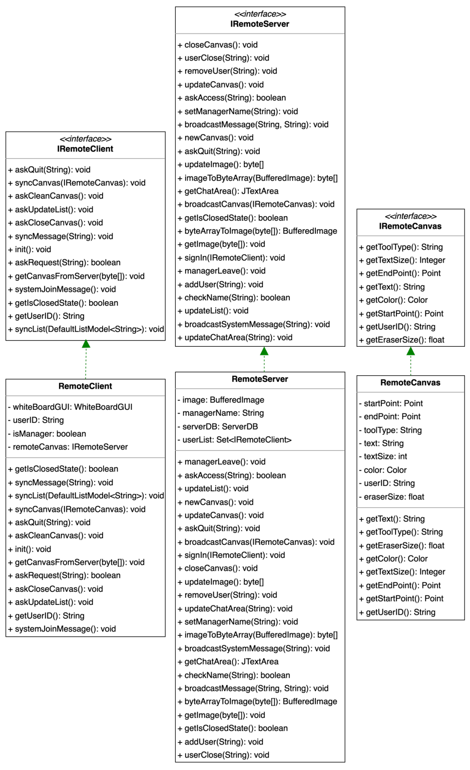

<<interface>>
IRemoteServer
+ closeCanvas(): void
+ userClose(String): void
+ removeUser(String): void
+ updateCanvas(): void
+ askAccess(String): boolean
+ setManagerName(String): void
+ broadcastMessage(String, String): void
+ newCanvas(): void
+ askQuit(String): void
+ updateImage(): byte[]
+ imageToByteArray(BufferedImage): byte[]
+ getChatArea(): JTextArea
+ broadcastCanvas(IRemoteCanvas): void
+ getIsClosedState(): boolean
+ byteArrayToImage(byte[]): BufferedImage
+ getImage(byte[]): void
+ signIn(IRemoteClient): void
+ managerLeave(): void
+ addUser(String): void
+ checkName(String): boolean
+ updateList(): void
+ broadcastSystemMessage(String): void
+ updateChatArea(String): void
<<interface>>
IRemoteClient
+ askQuit(String): void
+ syncCanvas(IRemoteCanvas): void
+ askCleanCanvas(): void
+ askUpdateList(): void
+ askCloseCanvas(): void
+ syncMessage(String): void
+ init(): void
+ askRequest(String): boolean
+ getCanvasFromServer(byte[]): void
+ systemJoinMessage(): void
+ getIsClosedState(): boolean
+ getUserID(): String
+ syncList(DefaultListModel<String>): void
<<interface>>
IRemoteCanvas
+ getToolType(): String
+ getTextSize(): Integer
+ getEndPoint(): Point
+ getText(): String
+ getColor(): Color
+ getStartPoint(): Point
+ getUserID(): String
+ getEraserSize(): float
RemoteClient
- whiteBoardGUI: WhiteBoardGUI
- userID: String
- isManager: boolean
- remoteCanvas: IRemoteServer
+ getIsClosedState(): boolean
+ syncMessage(String): void
+ syncList(DefaultListModel<String>): void
+ syncCanvas(IRemoteCanvas): void
+ askQuit(String): void
+ askCleanCanvas(): void
+ init(): void
+ getCanvasFromServer(byte[]): void
+ askRequest(String): boolean
+ askCloseCanvas(): void
+ askUpdateList(): void
+ getUserID(): String
+ systemJoinMessage(): void
RemoteServer
- image: BufferedImage
- managerName: String
- serverDB: ServerDB
- userList: Set<IRemoteClient>
+ managerLeave(): void
+ askAccess(String): boolean
+ updateList(): void
+ newCanvas(): void
+ updateCanvas(): void
+ askQuit(String): void
+ broadcastCanvas(IRemoteCanvas): void
+ signIn(IRemoteClient): void
+ closeCanvas(): void
+ updateImage(): byte[]
+ removeUser(String): void
+ updateChatArea(String): void
+ setManagerName(String): void
+ imageToByteArray(BufferedImage): byte[]
+ broadcastSystemMessage(String): void
+ getChatArea(): JTextArea
+ checkName(String): boolean
+ broadcastMessage(String, String): void
+ byteArrayToImage(byte[]): BufferedImage
+ getImage(byte[]): void
+ getIsClosedState(): boolean
+ addUser(String): void
+ userClose(String): void
RemoteCanvas
- startPoint: Point
- endPoint: Point
- toolType: String
- text: String
- textSize: int
- color: Color
- userID: String
- eraserSize: float
+ getText(): String
+ getToolType(): String
+ getEraserSize(): float
+ getColor(): Color
+ getTextSize(): Integer
+ getEndPoint(): Point
+ getStartPoint(): Point
+ getUserID(): String

Figure 2: RMI Objects

When the server starts, it will create a RemoteServer and binds it to the RMI registry, and all user communication needs to be transferred with the RemoteServer. All user operations, including sending messages and drawings, are transmitted to the RemoteServer, which processes them and finally synchronies them to all users.

When users (including manager) open the program, they need to get the RemoteServer object by using the lookup remote object by the key from the rmiregistry. They will create a RemoteClient object, the object contains user's information, it has access to users who can perform actions, and transfer the RemoteClient object to the server. Server can find each user by finding the RemoteClient and performing the operation on it.

There are many benefits to using RMI:

- RMI can handle threads and sockets for underlying communication.
- Server-side implementations can make changes without notifying the client.
- RMI have good scalability and flexibility to add new features to existing server.

- RMI hides the details of the network, making calling a remote object as easy as calling a local object.
- RMI provides an effective mechanism for handling exception in a distributed environment through RemoteException.

The program can let user sends messages via chat box and all users can see the content of the message. When a user sends a message, it sends the message to the server, which then synchronizes the message to the other users, the whole process is implemented through RMI.

At each operation the user drawing on the canvas, the program will pack up the user's operation and creates a RemoteCanvas to store it. The RemoteCanvas stores all the details of this user's operation, such as the tool type, colours, and coordinate points. After the operation is completed, the RemoteCanvas is sent to the server, which sends it to other users, who unpack it and draw if for synchronization. I didn't transfer the whole canvas to other users because it might not be transferred in time due to the network speed, thus other user can't get the latest canvas in real time. If transfer a canvas data to other users, the amount of data in the canvas need to transfer, will cause the user to feel lag when drawing.

After each user operation on the canvas, the server stores a copy of the latest canvas in Buffered Image format. When a new user connects to the server, the server sends user the latest canvas, which ensures that all users are operating on the same canvas.

3. Design Diagrams

3.1 Class Diagram

Figure 3 (Page 4) shows the class diagram of the whole system, there are two executable classes in the system CreateWhiteboard and JoinWhiteboard. Before running JoinWhiteBoard, CreateWhiteBoard must has run. When CreateWhiteBoard runs, it will first create a server and run the server, after that it will create a manager's whiteboard and bind it to the RMI server. In addition, there are 3 different packages in my system: remote, server and whiteboard.

In the remote package, there are three different RMI classes. The functions and associated with these three classes have been described in section 2.

In the whiteboard package, the classes are the core that make up the whiteboard GUI. My system uses java swing to create the GUI, the WhiteBoardGUI class is a shell which is responsible for creating the basic information of the GUI, such as title and size, there is also a menuBar in this class for the manager to perform addition operations on the board. The WhiteBoardGUI class will creates ToolBar, DrawPanel and ChatBox separately and adds them into its own JFrame. The ToolBar class is used to create brush types and colour sections. The user selects the appropriate brush type such as line and free draw in order to draw different kinds of patterns on the canvas. User can also select different colour to adjust the colour of the brush. Users achieve different effects by clicking on the relevant buttons. DrawPanel class creates a drawing canvas for the user to draw on, which is implemented using java Graphics2D. When the user draws, the canvas determines what type and colour the user has selected in the ToolBar and draws the corresponding content according to the user’s selection. The ChatBox class provides three functions, list of user box, chat history box and input box. User list display all usernames indicating that who are editing the same whiteboard currently, the chat history box displays the content of messages sent by the user, and the input box allows the current user to enter text and send it to everyone.

In server package, ServerDB class is used to store the chat log and user list, when a user sends a message, server add it into chat log before server broadcasts to other users, so that the newly joined users can also get the history of chat log. The user list is used to store which users are currently editing the canvas and is updated in real time when new users join or exist users leave.

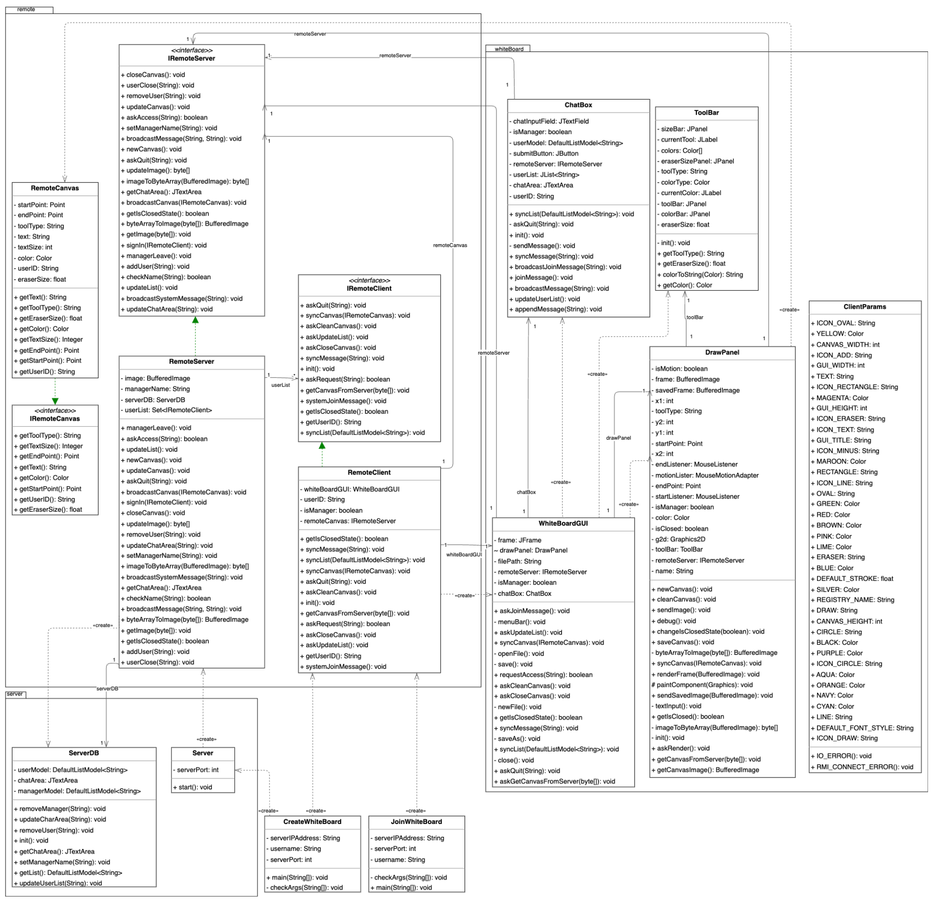


Figure 3: Class Diagram

3.2 Interaction Diagram

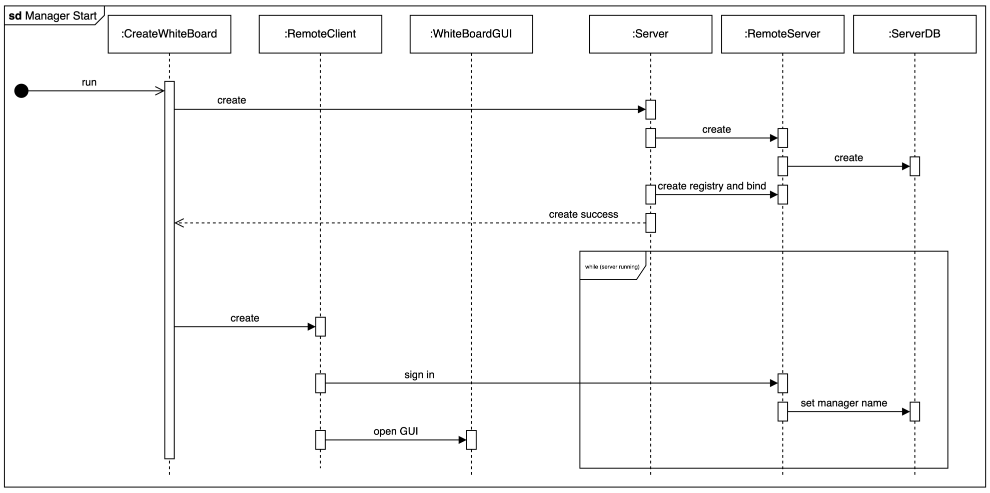


Figure 4: Sequence Diagram – Manager

When manager opens CreateWhiteBoard, the program first creates and opens the server, and then creates the manager's GUI. Figure 4 above shows the interaction logic in this case.

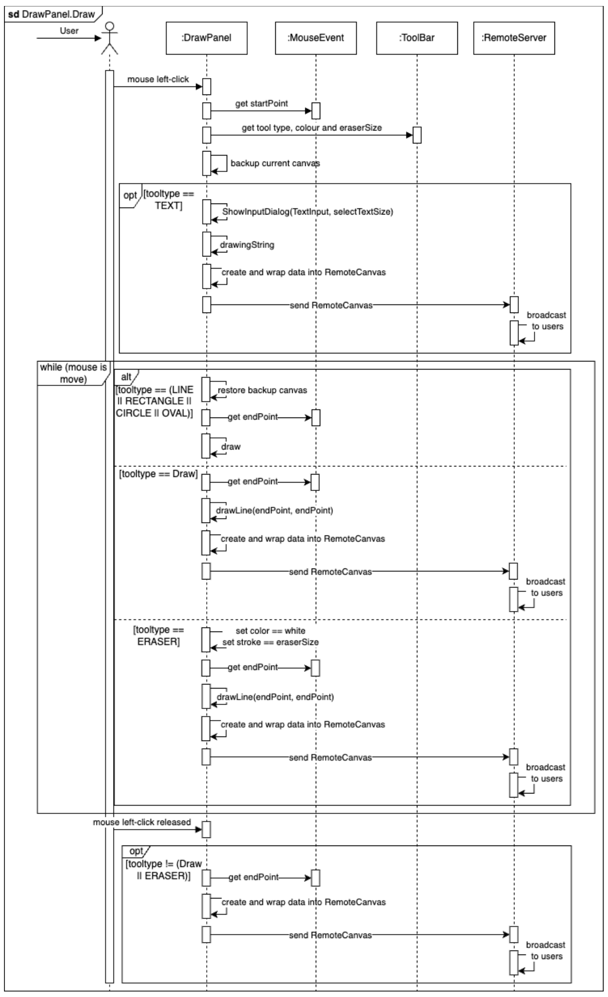


Figure 5: Sequence Diagram – Draw a shape.

Figure 5 above shows that the process when the user draws a shape on the canvas. When the toolType is LINE, RECTANGLE, CIRCLE and OVAL, the user can move the mouse and display the shape in real time, so that the user can determine the desired shape. When the toolType is DRAW and ERASER, whenever the user moves the mouse, it will draw the current point on canvas and send it to the server and synchronize it to other users. Other tool types of data as remoteCanvas send to the server and synchronize with other users only when the user released mouse left-click.

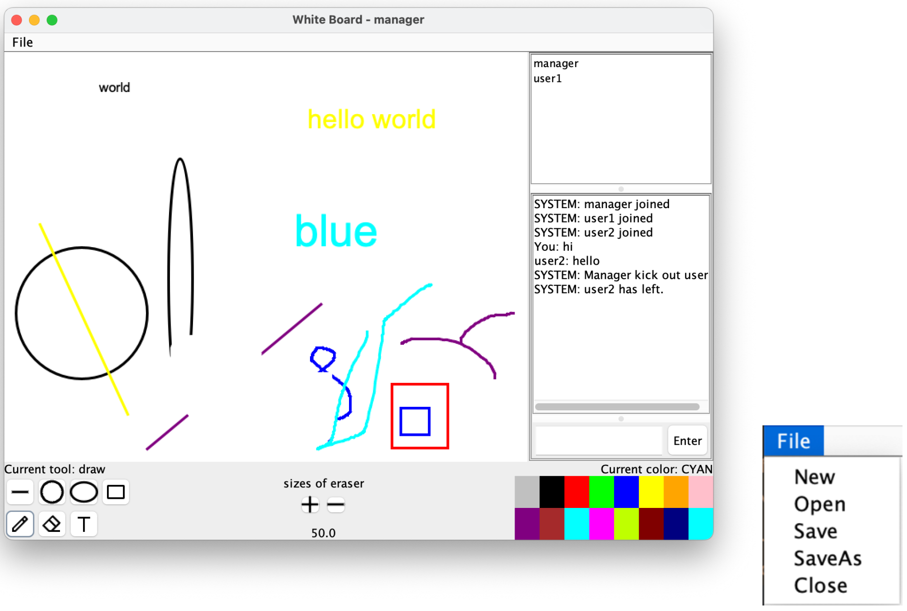
3. Implementation Details

I have provided two executables, CreateWhiteBoard.jar and JoinWhiteBoard.jar. When the first user wants to create whiteboard, he must start with CreateWhiteBoard.jar, which will create the rmi server and whiteboard. After that, other users only can use JointWhiteBoard.jar to join the manager's whiteboard, and other users can't get the same privileges as managers.

The specification must be used to run in terminal:

- ➢ java -jar CreateWhiteBoard.jar <serverIPAddress> <serverPort> <username>
- ➢ java -jar JoinWhiteBoard.jar <serverIPAddress> <serverPort> <username>

Also, SDK version must be (at least): 21.0.2



manager
user1

```
SYSTEM: manager joined
SYSTEM: user1 joined
SYSTEM: user2 joined
You: hi
user2: hello
SYSTEM: Manager kick out user
SYSTEM: user2 has left.
```

Figure 6: Whiteboard GUI and File functions button

Figure 6 shows the whiteboard GUI for the manager. The area in the centre is the draw panel. The size of the canvas is fixed, with width of 590 and a height of 465.

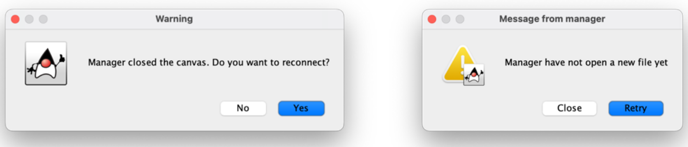
The bottom area of the GUI is the toolBar, the left side allows users to select the type of tool he wants to use, and there is a label showing the currently selected tool in real time. The middle area allows user change size of eraser by clicking the buttons. The right area allows the user to select the colour they want, providing 16 colours, and a label showing the currently selected colour in real time. When the whiteboard started the default tool is line and colour is black.

On the right is the ChatBox, which contains three sections. The top one is the user list, which shows all the users that are editing. In the case figure 6, there is a manager named Manager and has a user named user1. In the manager side, he can kick out user1 by double-clicking his name. Other users cannot kick someone. When some user leaves, the user list is refreshed to ensure that it always shows the latest content.

The box in the centre is chat log, which is visible to all. User can send message by typing text in the input text area below and press Enter, everyone will see your message. The format of the message is: Name: Context. When you send a message your GUI will not show your username but "You", other users GUI will be showing your name normally. This ensures that users don't misinterpret their messages.

The manager and users will use the same GUI, the only difference is that the manager have a menu bar with File button allows manager to perform new, open, save, saveAs and close operations. When the manger clicks on the new button, a new canvas is opened for all users. The open button allows manager to open an existing image with png format. Since the size of the whiteboard is fixed, if the manager opens an image that is larger than the size of the canvas it may cause the image to be incompletely displayed in the canvas. The save button allows the manager to save the current canvas locally, but the manager must have previously clicked the SaveAs button and selected a path, otherwise the whiteboard will ask the manager to get a path.

When the manager clicks the close button, the whiteboard first asks if it wants to save the current canvas, then the whiteboard clean the canvas, and the server notifies all other users that the manager has clicked the close button, user can close the whiteboard or request to reconnection. If the manager wants to continue drawing, he must click on the new or open button, otherwise he will not be able to draw. Also, if the user clicks on request reconnect, if the manager doesn't create or open a canvas, the whiteboard will tell the user that the manager didn't create or open the canvas and ask whether to retry (Figure 7 below).


Warning

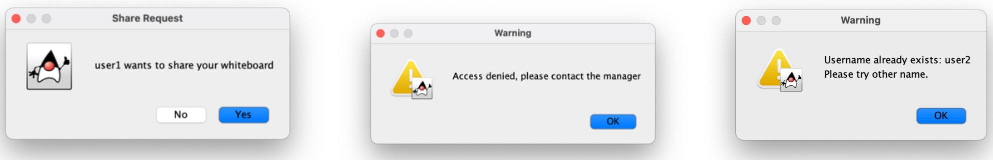
Manager closed the canvas. Do you want to reconnect?

Message from manager

Manager have not open a new file yet

Figure 7: relevant warning message from user side after close operation

When a new user starts the whiteboard, the manager's whiteboard will pop up a message that the manager needs to accept or reject this user's access request, and until then the manager can't do anything else on the whiteboard. This avoids request from being backlogged or ignored, which improves system responsiveness and user satisfaction. If manager clicks yes, this user will open whiteboard. If the manager clicks no, the user will open a message window indicating that access has been denied by the manager (Figure 8). If the new user's name already exists in the user list, it will ask for change to other name.



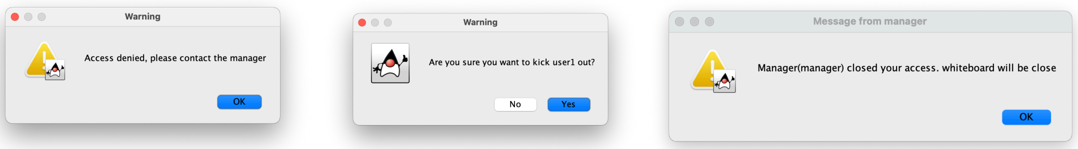
user1 wants to share your whiteboard

Access denied, please contact the manager

Username already exists: user2
Please try other name.

Figure 8: relevant message when new user request access

Figure 9 shows more information about the operation, such as the message user receives when manager denies user access to the canvas. When the manager clicks on a user in the user list, a message pops up confirming the action. When user is kicked out by the manager, a message will pop up to remind the user, and GUI will close.



Access denied, please contact the manager

Are you sure you want to kick user1 out?

Manager(manager) closed your access. whiteboard will be close

Figure 9: others relevant message

4. New Innovations

In this project, I provide a user-friendly GUI that user can use it easily, I have put all the interactions required by the user into the GUI including exceptions, such as popup message to tell the user when the RMI server connection failed. I handle all the possible exceptions that may occur and will alert the user when necessary. In addition, the user only needs to be in terminal when it is started, but at all other times it only needs to be in the GUI, which provides convenience for the user. Users can change the size of the eraser, added system information in the chat log, when the user joins or quits, manager open, new, or close, the server will send message in chat box and prompt for relevant information, so that users can clearly see what is happened in the whiteboard.