COMP90015
Distributed Systems

Assignment #2 - Distributed White Board

Xing Yang Goh
#1001969
May 25, 2023

1 Problem Description

A client-server architecture for a real-time distributed whiteboard service will be developed, handling multiple concurrent client requests over the network and ensuring this shared interactive canvas is consistent between the clients without appreciable delays between making and observing the edits. This whiteboard service will formally allow the following functionalities:

1. Allow for multiple clients to draw on the shared interactive canvas
2. Clients can input lines, circles, oval rectangle shapes and text on the canvas
3. Free-drawing pen capabilities and an eraser tool to edit the canvas
4. Clients can choose from a variety of colours (16+) and can choose a stroke size for the edits
5. A user window bar for users to see the clients connected to the whiteboard
6. A chat window to allow the users to communicate with each other
7. A file menu for the manager to create a new canvas, open an existing canvas, save/saveAs the current canvas state and close the whiteboard.
8. A kick option for the manager to remove specified users.

2 System Overview

The client-server connection is handled using the Remote Method Interface (RMI), which provides remote communication between programs written in Java. As such, the Application and Server are written with Java, using the JavaFX library for the GUI and application. To initialise the server, the user can provide the appropriate host and port details, which will initialise an RMI registry at this port and rebind a new instance of the whiteboard server to this registry. To connect a client to this server, the app is initialised using a user-specified host and port details, along with a username that has to be alphanumerical or underscore characters with a length range of 4-19 characters. If this is the first user connected to the server, the client is granted manager status to allow file and kick functionalities. Note that each client has to have a unique username on the server. Once the client has been initialised, the list of clients connected to the server is updated, and all the subsequent non-manager clients will have their whiteboards automatically updated to the current state.

To handle this interactive drawing experience, every time a user edits the board, this action is encapsulated into a Shape object in figure 3, that has all the required information to locally reproduce this edit on every client's whiteboard by "re-drawing" this Shape is instantaneously produced on every client's app instance. This method is used as opposed to encoding and decoding the entire board state at every edit as this sends much less data with each edit, which is very important when sending freehand drawings and erase action as this information will be sent on every pixel drag. These drawings are performed using the various event handlers and stroke commands included in the JavaFX application library. Note that all the server methods utilise a synchronized keyword, this ensures that only one thread executes at a time for all the synchronized methods by using a lock associated with a monitor that encapsulates the shared resource and allows only one process at any instance to effectively handle the concurrency with multiple users.

3 System Design Diagram

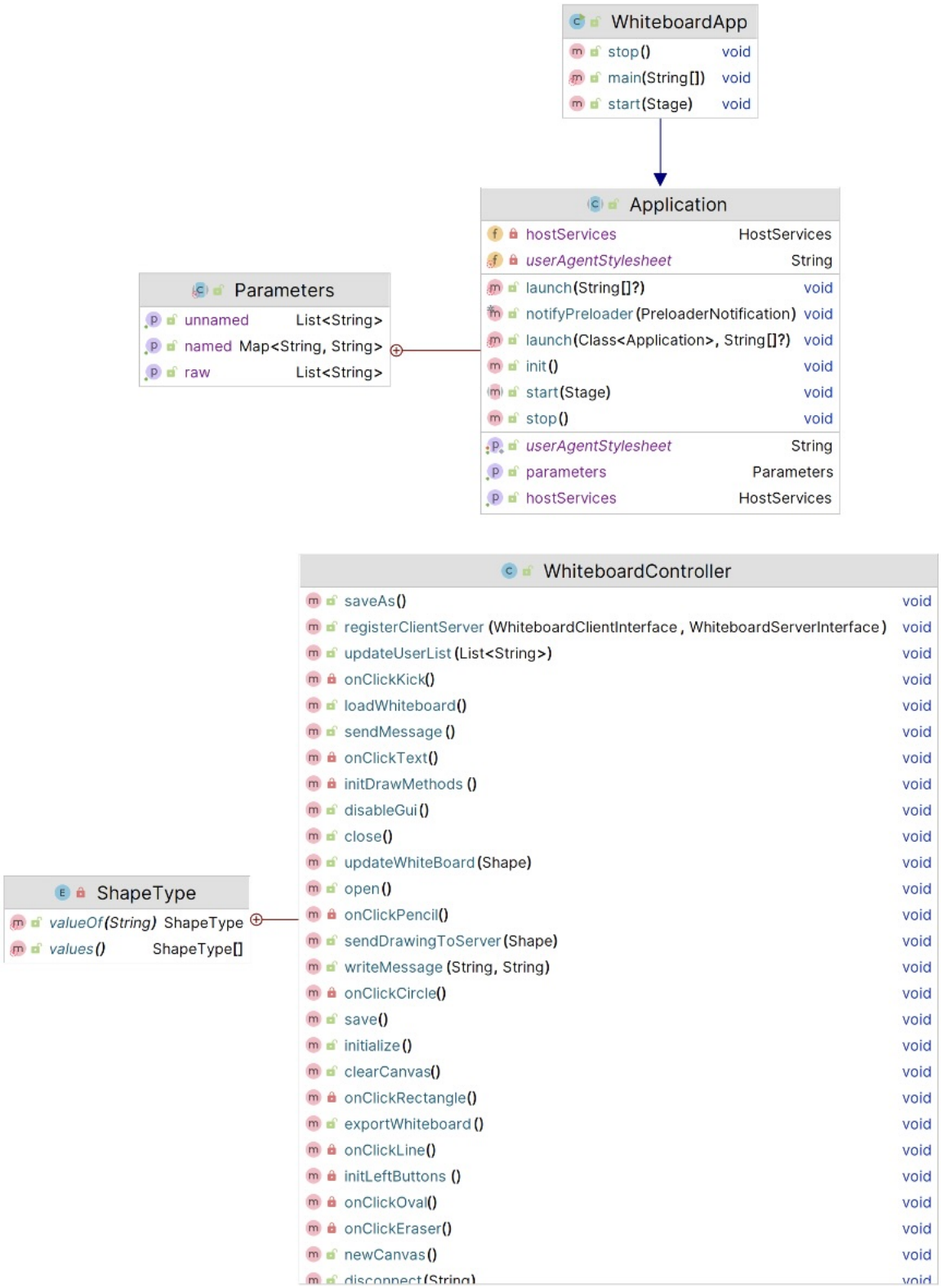

Figure 1: UML Diagram for the Whiteboard App and Controller classes

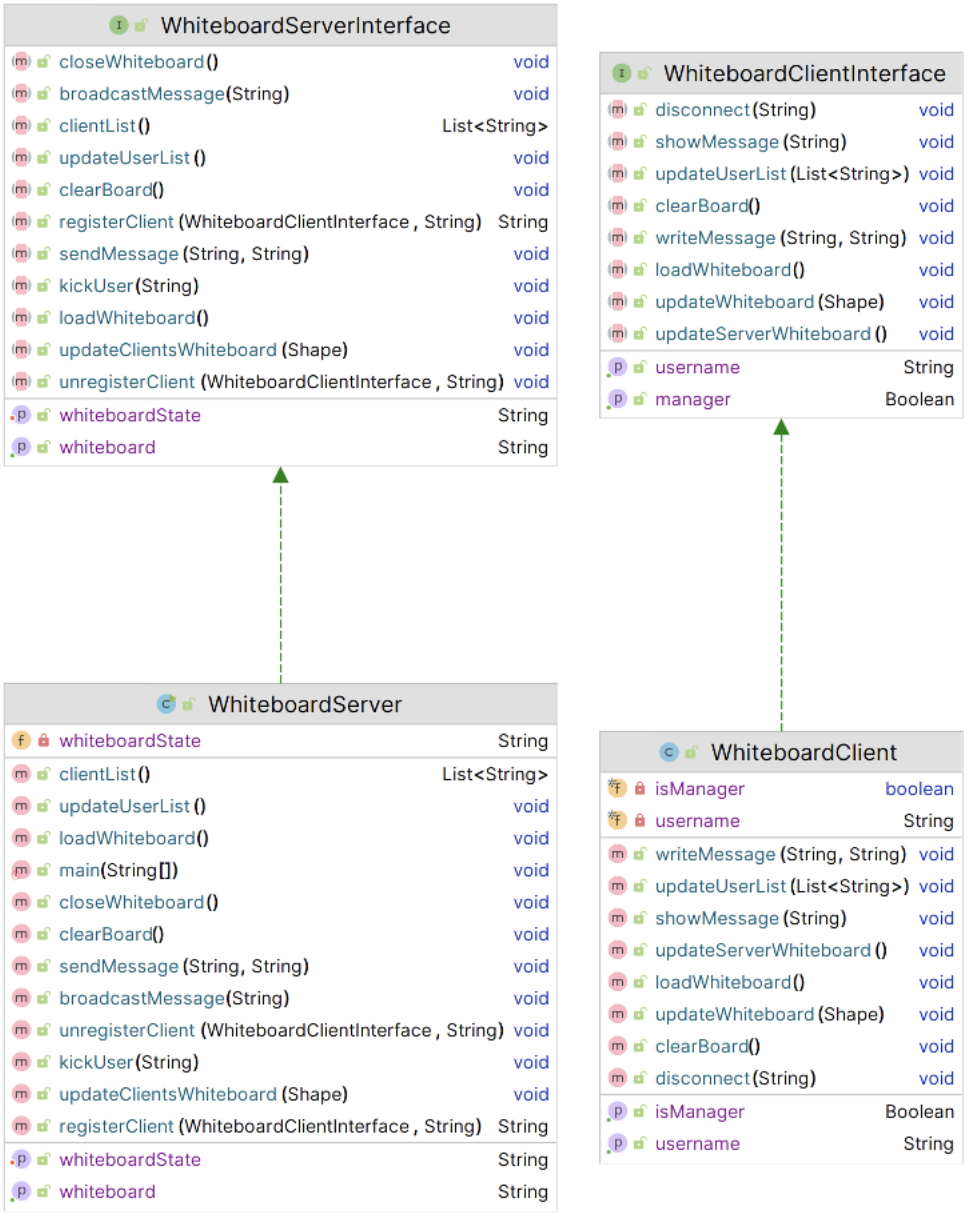

Figure 2: UML Diagram for the Whiteboard Server and Client classes

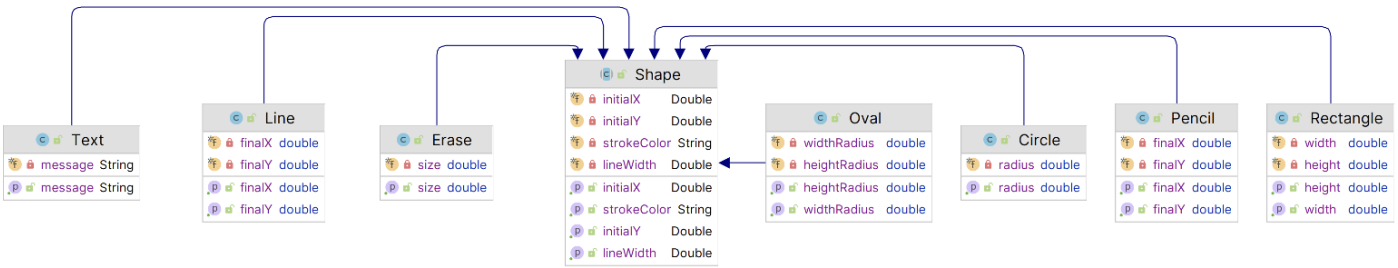

Figure 3: UML Diagram for the various Whiteboard Shape classes

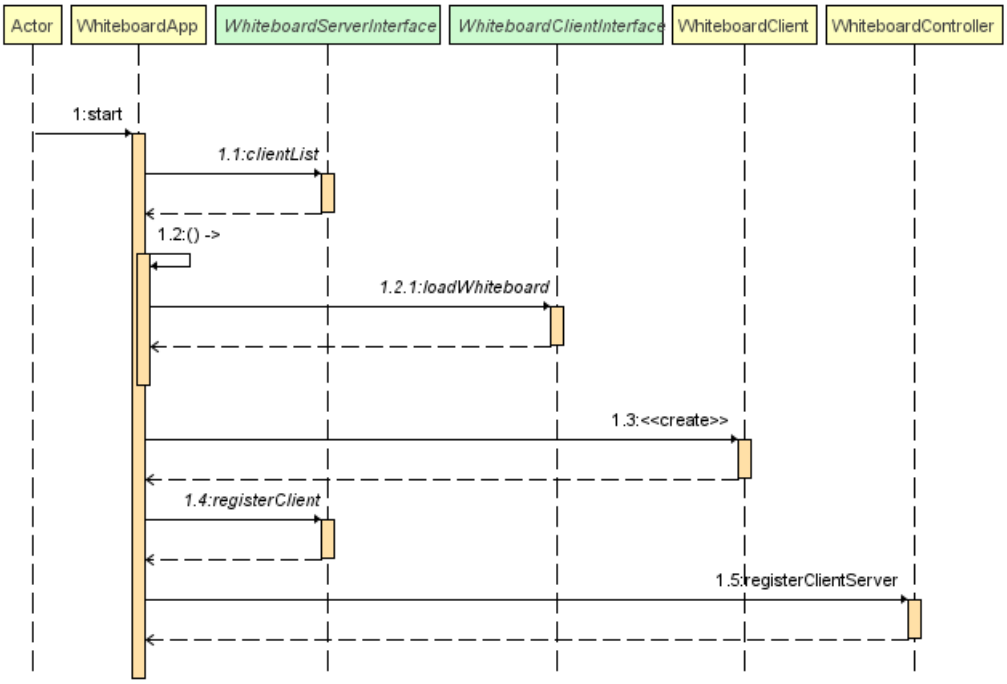

Figure 4: Whiteboard Service sequence diagram

This sequence diagram demonstrates the initialisation of the whiteboard app, when it starts, it acquires the whiteboard server from the registry with the given host and port arguments. From this, it acquires the current list of clients from the server ensuring the username is valid and loads the whiteboard state (either a new canvas if the first user or the current canvas if subsequent). This loading of the current canvas state is done by updating Base64 encoded whiteboard stored on the server from the manager client and sending this to all the clients as a runnable, which is posted to the event queue of the JavaFX application controller instance for this current app thread. Now the client is created by passing it the controller, username and manager status, and this client is registered to the server. Now when the JavaFX application begins execution, the runnable event is triggered which saves the current whiteboard state from the manager client to the server with Base64 encoding, and this whiteboard state is updated for all the clients, effectively initialising the new app instance to the existing whiteboard state. When a new user has joined, they will be added to the list of users and a message on the chat box will non-intrusively alert all the clients that this new user has joined.

4 Implementation

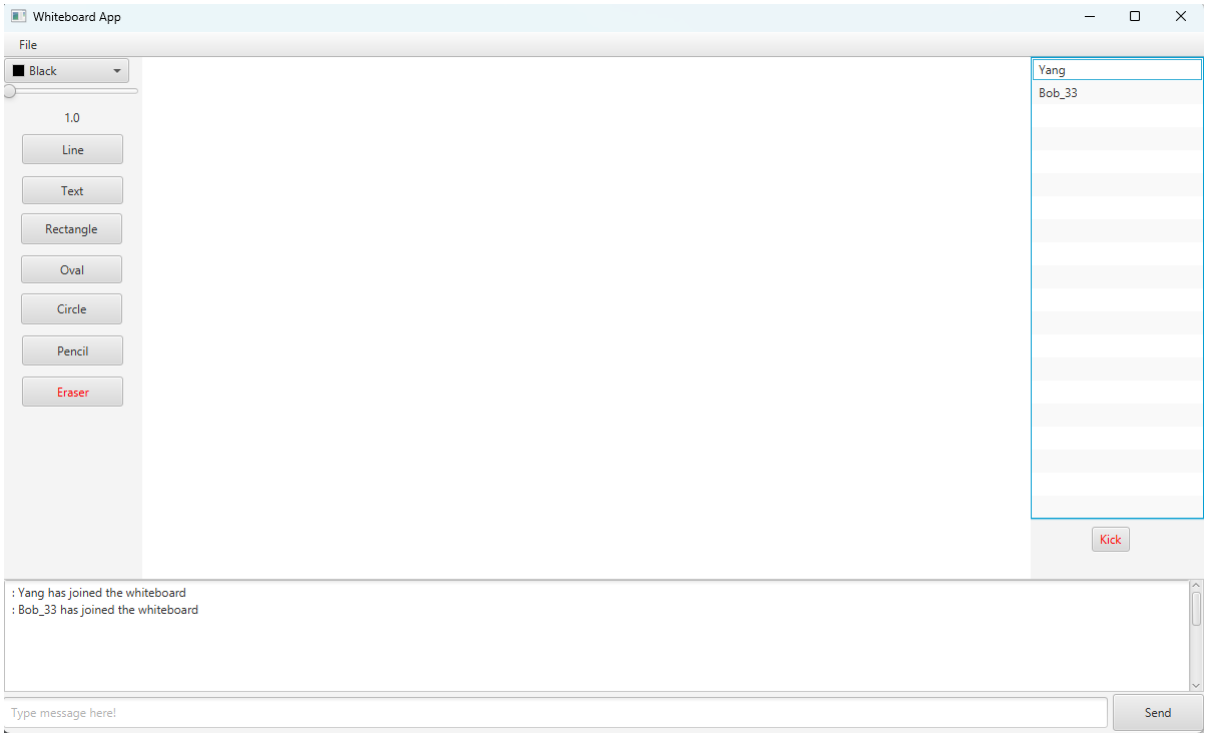

Figure 5: Manager GUI

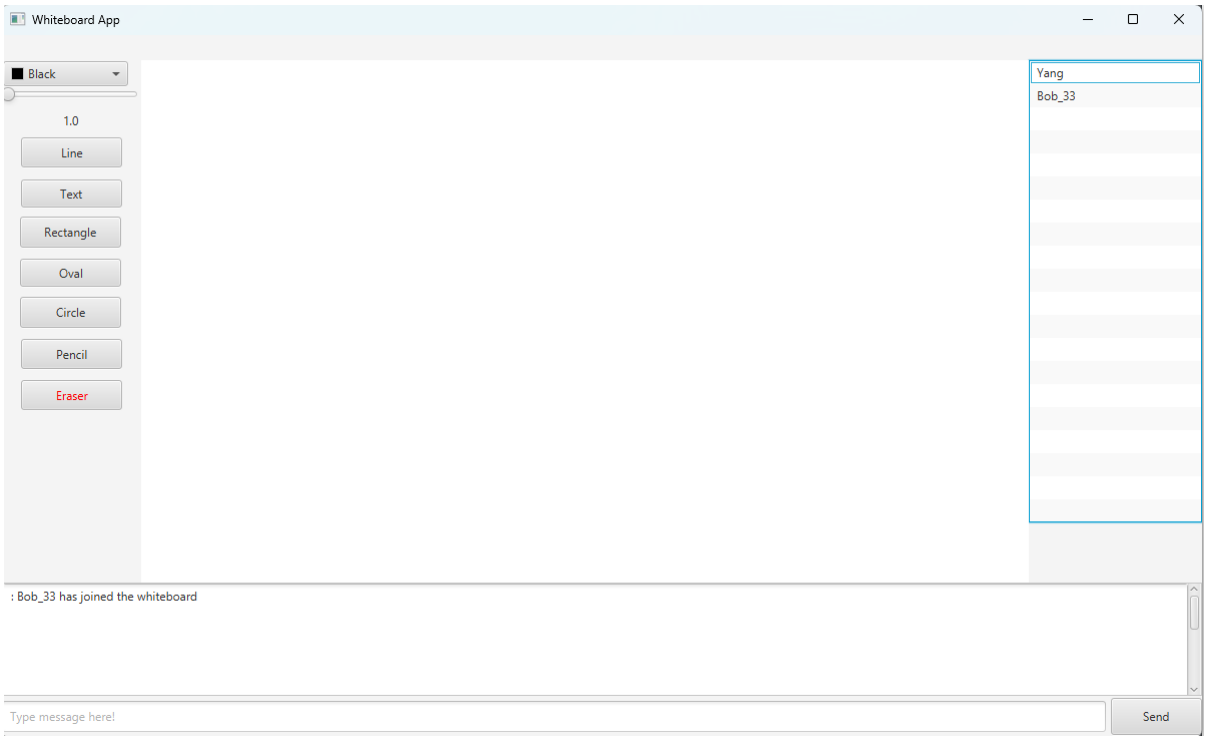

Figure 6: Client GUI

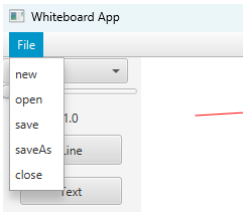

Figure 7: Manager file options GUI

Here are the two different GUI instances, one for the manager and one for the client, the only difference between these instances is that the manager GUI has a file option with the new, open, save, saveAs and close options; and the kick button below the user list.

In general, both GUIs have the following features:

1. Buttons for the various editing tools for the canvas
2. Colour picker to select the desired colour
3. Slider tool to choose the stroke size
4. A list of users currently connected to the server
5. Chat message box with a slider to view messages sent
6. A field to type a message, and a send button (can be clicked or triggered with return key).

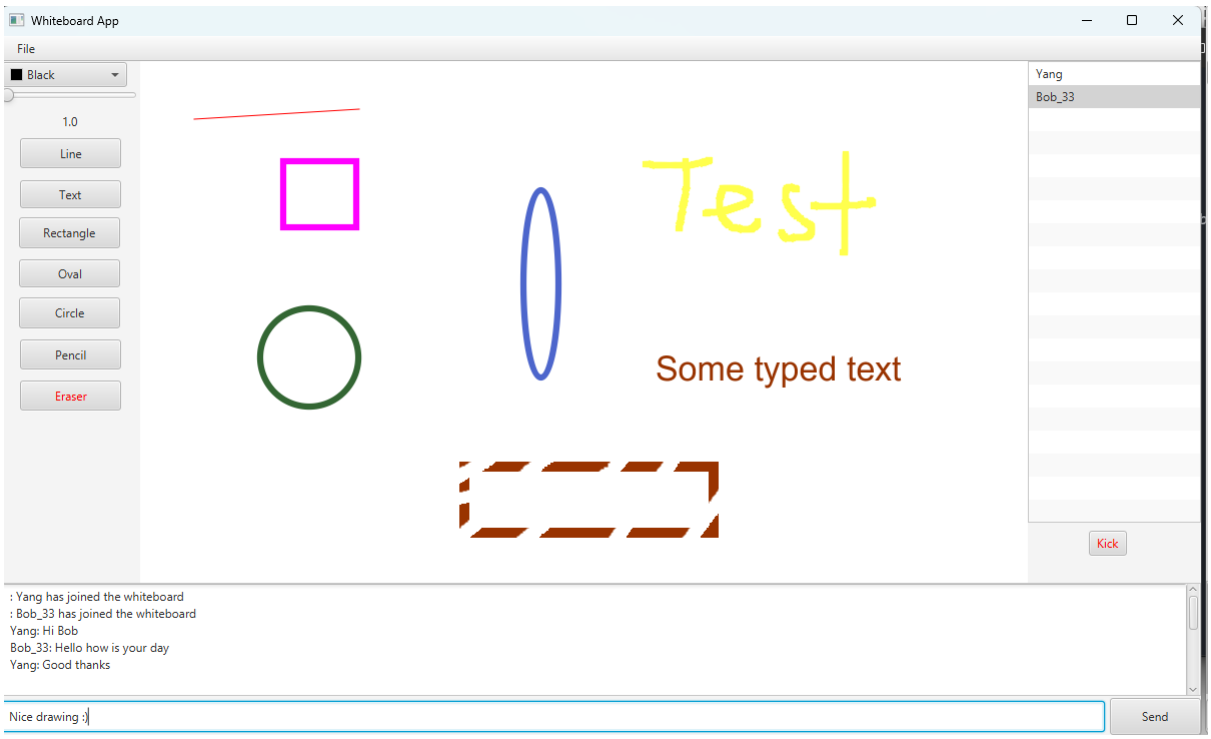

Figure 8: Example of the different functionalities being utilised

Figure 8 demonstrates the various features that are available to the clients. The various different editing tools are demonstrated using different colours and stroke sizes. The chat box with the history of messages and a new message typed out to be sent.

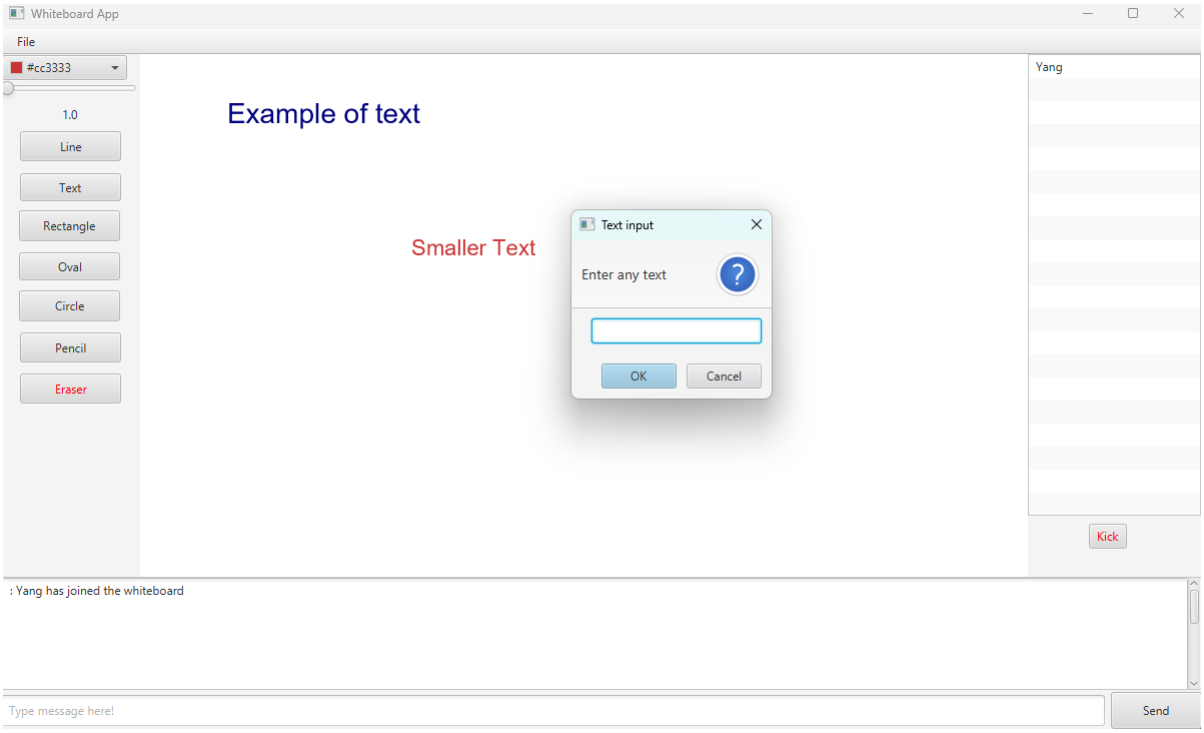

Figure 9: Inputting text onto the whiteboard

To input text onto the whiteboard, an alert box prompting the user to enter text will be displayed when the user clicks on a location on the canvas after clicking on the text button. This text also uses colour and the slider bar to determine the size of the text. Note that this text has a maximum width of 500 pixels and will be cut off after that point.

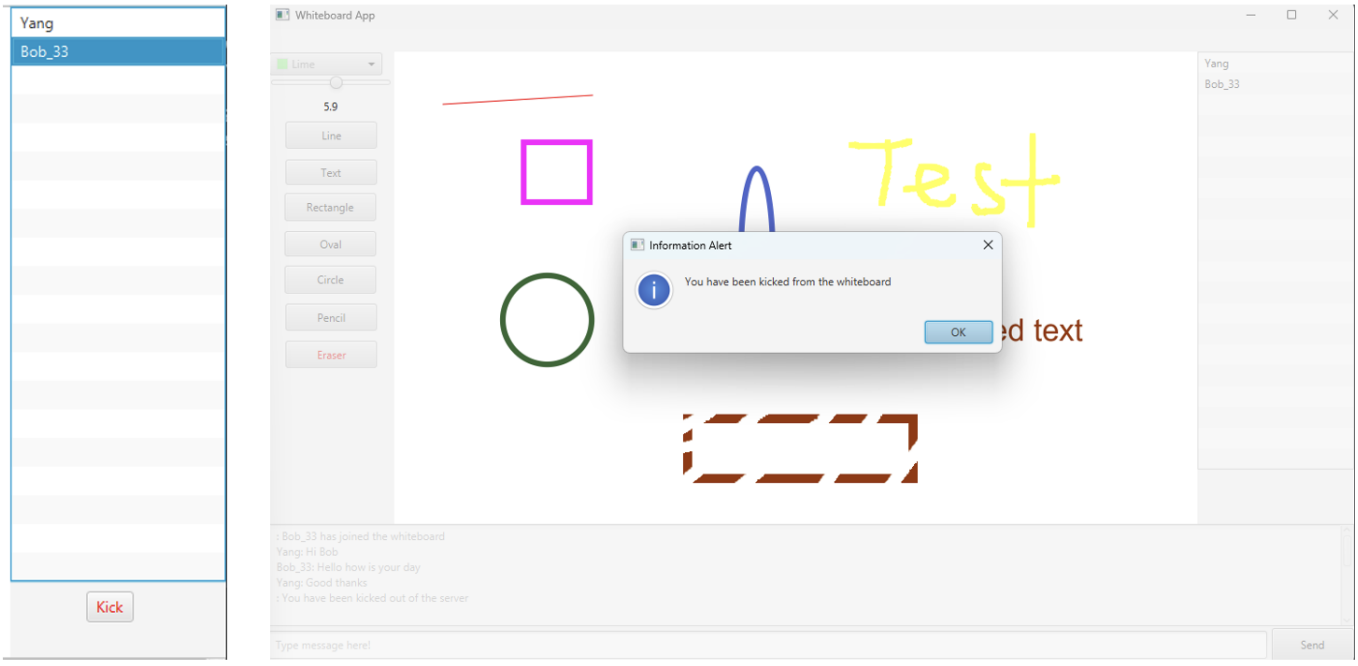

Figure 10: Manager kicking a user out

The manager also has the ability to kick users out, this is done through selecting the user on the user list and clicking on the kick button. When this is performed, the user will receive an alert that they have been kicked from the whiteboard and all the GUI elements are disabled, and they will be unregistered from the list of clients on the server.

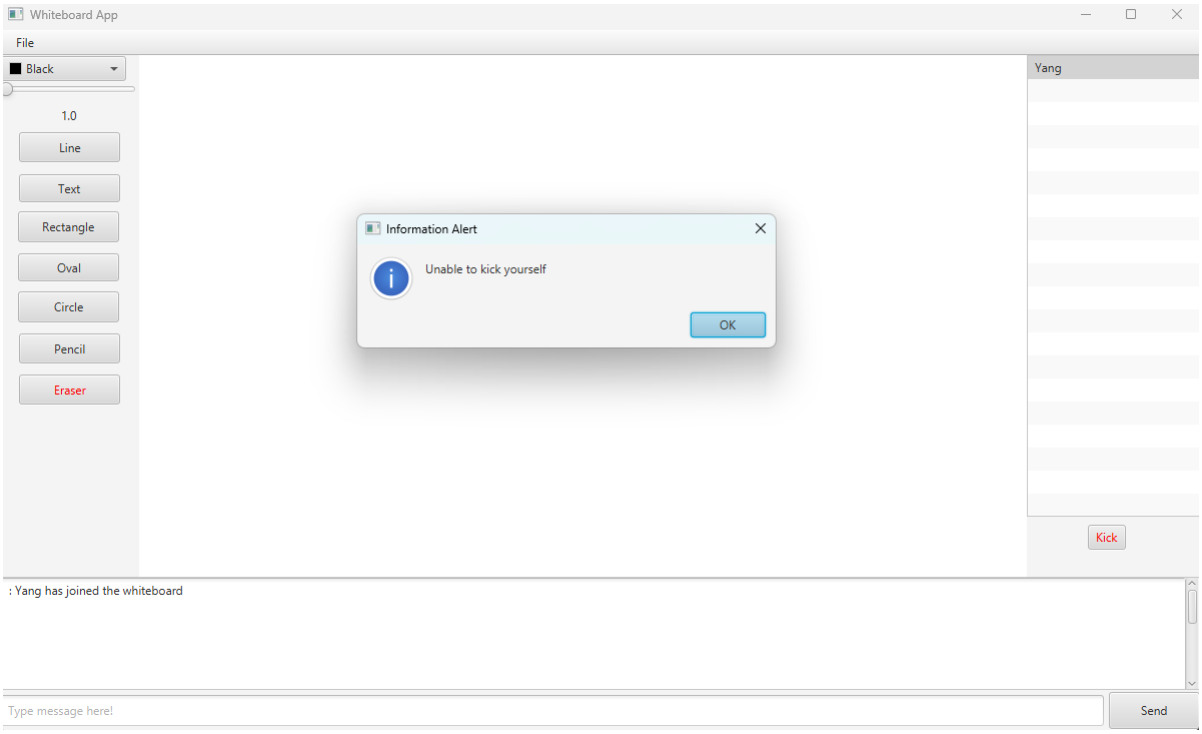

Figure 11: Manager attempting to kick themself

Note that the manager is unable to kick themself, and will receive an alert regarding this when attempted to let them know of the issue.

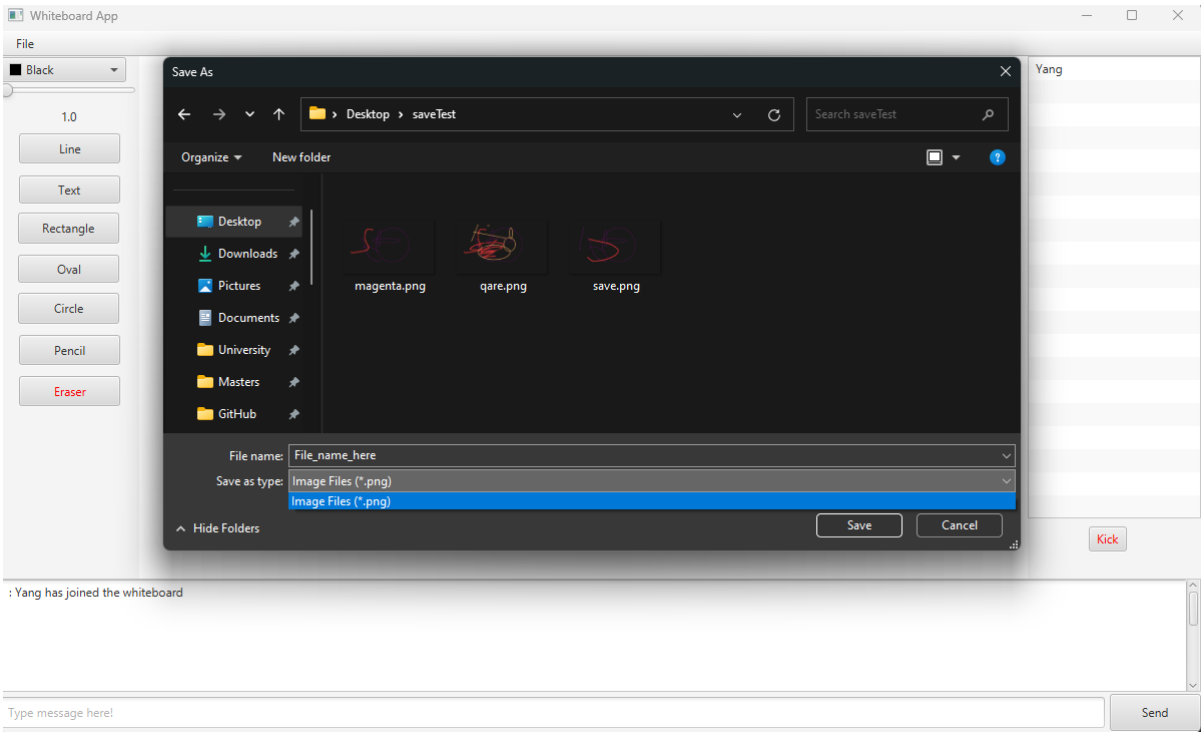

Figure 12: Manager saving and loading functionality

For the file options, when the manager attempts to use the load or saveAs command, they will be prompted with a file chooser for the user to specify where they want to load or save the file from (limited to *.png files). And a similar decoding/encoding Base64 method is used to load/save the file similar to when a new user joins an active whiteboard. Note that if the user tries to use the save command before a file has been opened, it will default to the saveAs functionality as the file path has not been set. Similarly, pressing the new command will clear the canvas for all users and reset this file path, so any future save commands will do a saveAs even if there has been an open command previously.

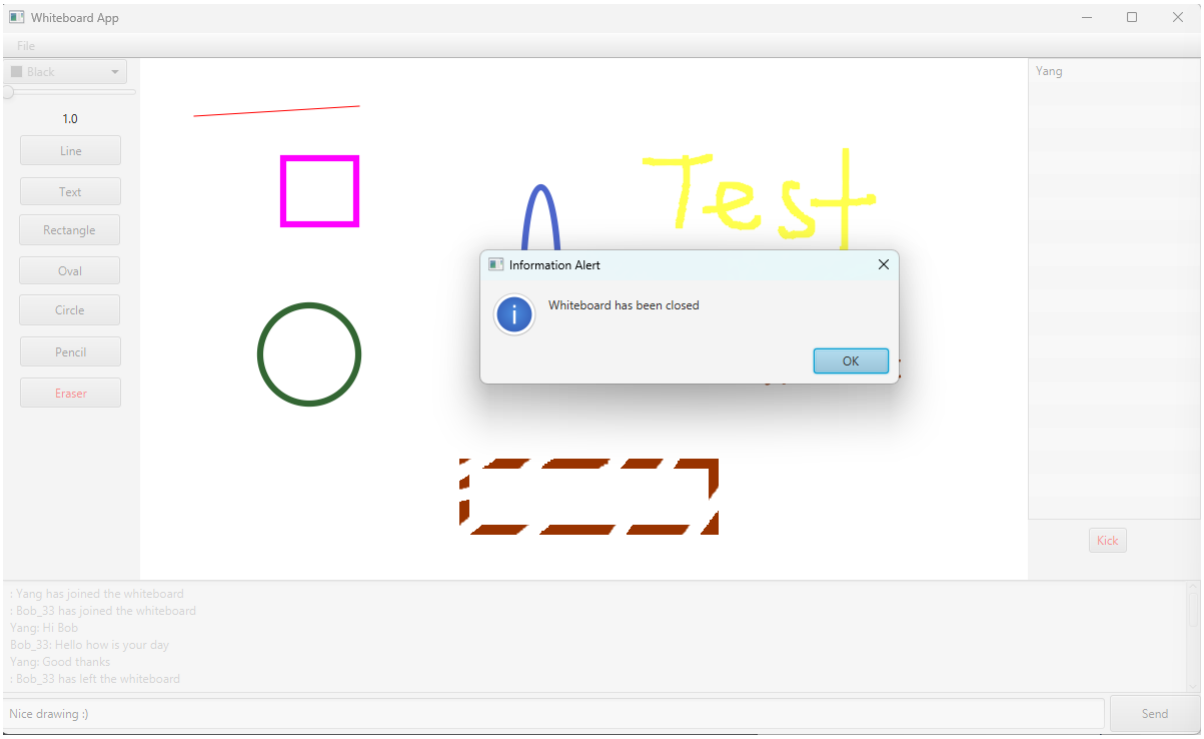

Figure 13: Manager close functionality

When the manager hits the close command or closes the application, all the users are removed from the server and have their GUIs disabled.