COMP90015 Distributed Systems
Assignment 2 – Report

Summary

The task was to design and implement a shared white board application that a manager can host, and multiple clients can join to draw on the session's canvas while updating the canvas in real-time for other users. The canvas must include the features to draw lines, shapes such as circles, ovals, and rectangles, and allow free hand drawing with pencil. At least 16 different colours are available for the users to pick from to fill or draw with. The canvas includes text that can be scaled. The host of the white board session will have access to privileged functions such as

Design

Server Architecture

The server architecture chosen for this assignment is a central server system in which the central server processes all request made by clients/hosts and manages the system state as displayed on the Figure 1. The transport protocol used is TCP for this assignment for the reliability of the protocol.

The advantages of using a central server architecture as opposed to peer to peer:

- Easy to physically secure as there is one server in which all users connect to. In a peer-to-peer a connection may be compromised since the data is shared on all peers.
- The server can have dedicated resources for scaling if needed. Whereas in a peer-to-peer model, not all peers will have the same number of resources dedicated for the software.
- Quicker updates are available depending on the location of the server and the latency between user and server.
- Central governance is easier to manage if all data are only stored within the server, thus if government, law, or company policies is required to enforce certain type of "pictures" or "data" shared between users are shared, they can be monitored within the server.

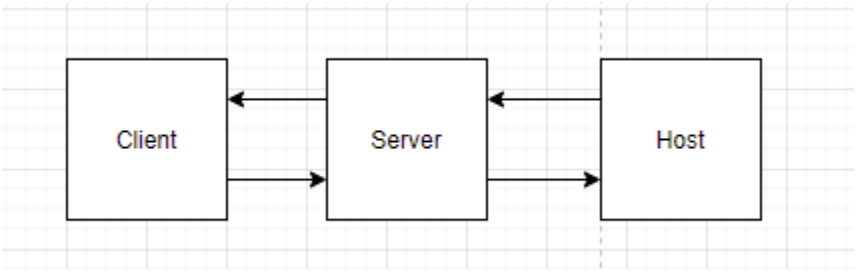

Figure 1: Server Architecture

The server allows multiple "sessions" of white board to be hosted concurrently using a thread-per-request model where a thread is created per connection listened on the server socket. Each individual session's key is the host's username where all usernames are unique within the server. Once the request has been processed, both the request thread and socket input/output streams are closed, however, if a request from a user with the protocol of either a "CREATE" or "JOIN" request, the socket will remain open until the client send a "EXIT" request to the server. This ensures all updates are sent to the correct users when a session is changed.

On the client side, on the application start, they will request to either create a new white board session or join an existing session. Once the server has accepted this request, a listener thread is run on client

application to listen for updates from the server in the event of graphical, user, or chat message changes. This thread is maintained on a server handler that processes all actions of the user. Since the username is used for the key on the server to process request, a new socket connection is used for the user actions.

Both white board sessions hosted by the server and users connected to the server is stored in hash map. Each session is a Session object which includes information such as connected clients, host, and the current state of the canvas.

Message Exchange Protocol

A custom Message Exchange Protocol was designed for this assignment for the communication between Server and Users. The protocol designed have similarities with the HTTP protocol. The Server Request and Response headers are shown in Figure 2. The Client Request and Response Header are shown in Figure 3, including the headers “KeepAlive” that notifies the server to keep the TCP socket open since the client will be listen until the closer of the application.

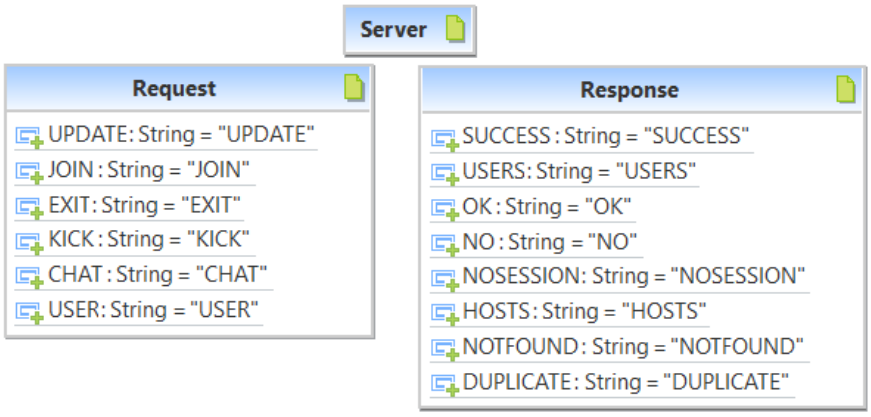

Figure 2: Server Protocol Headers

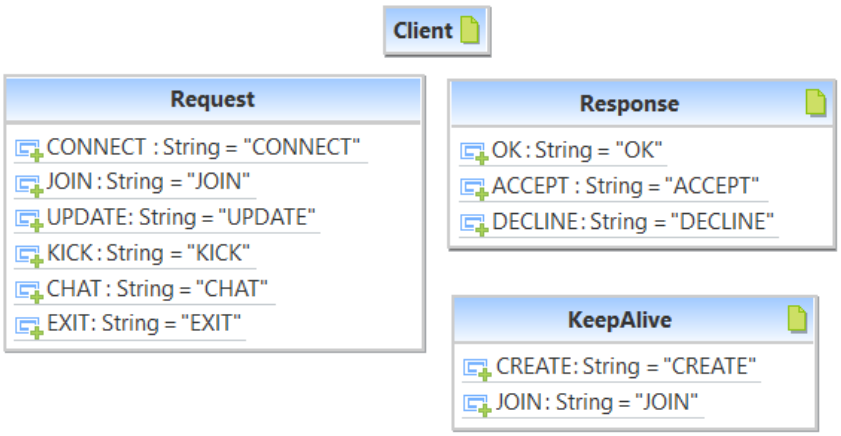

Figure 3: Client Protocol Headers

Message Exchange Payload

Using the Message Exchange Protocols as headers a payload class is created that is used for the communication between Server and User. Using Java’s Object input and output streams, all transmitted object must be serializable, thus the payload stores objects of Serialised Chat Message, Serialised Canvas image, and Serialised String Array.

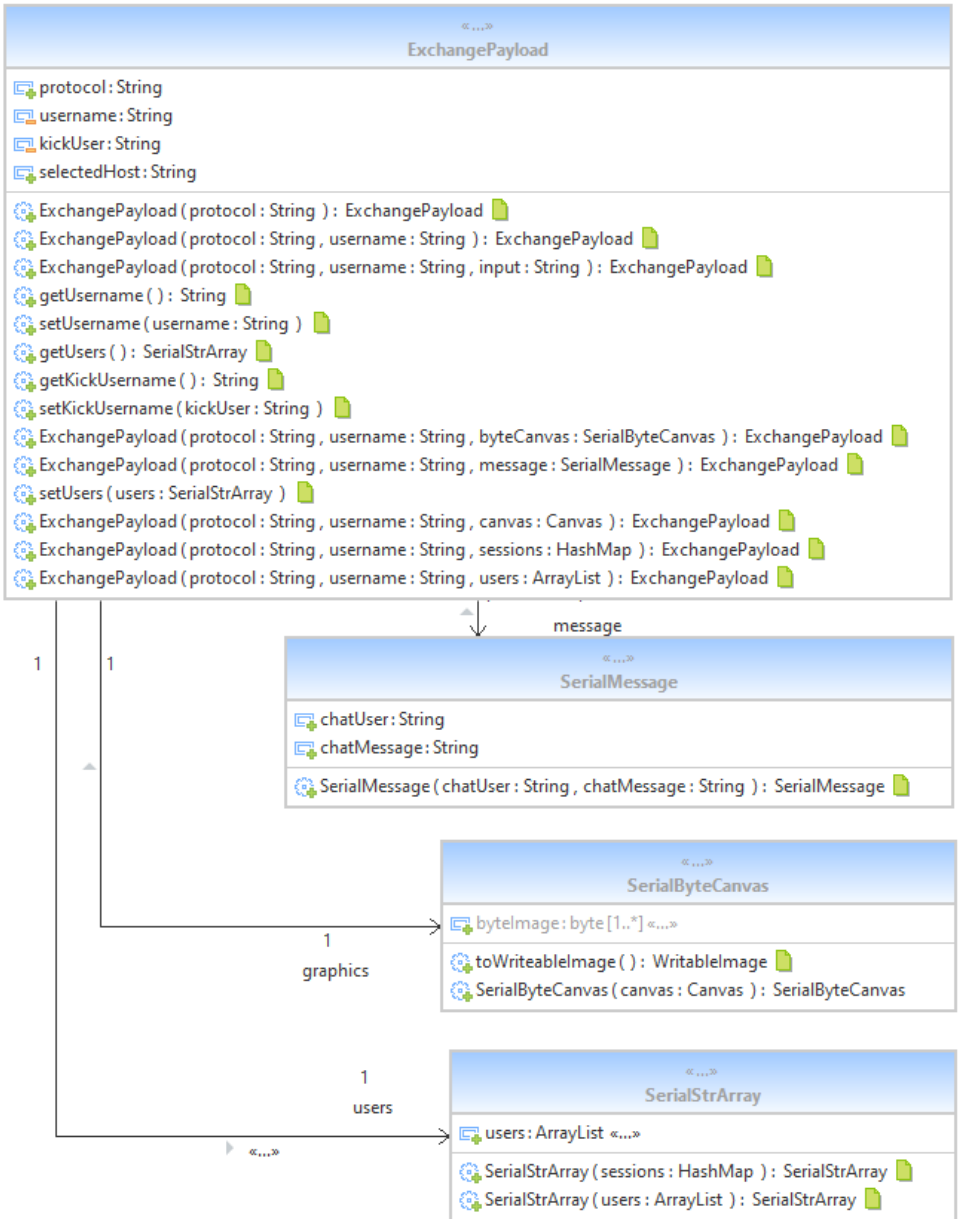

Figure 4: Message Exchange Payload

Class Design

The White Board Server UML class diagram is shown in Figure 5. The extra connection shown are the interaction to the client. The Whole diagram can be found within the submission folder.

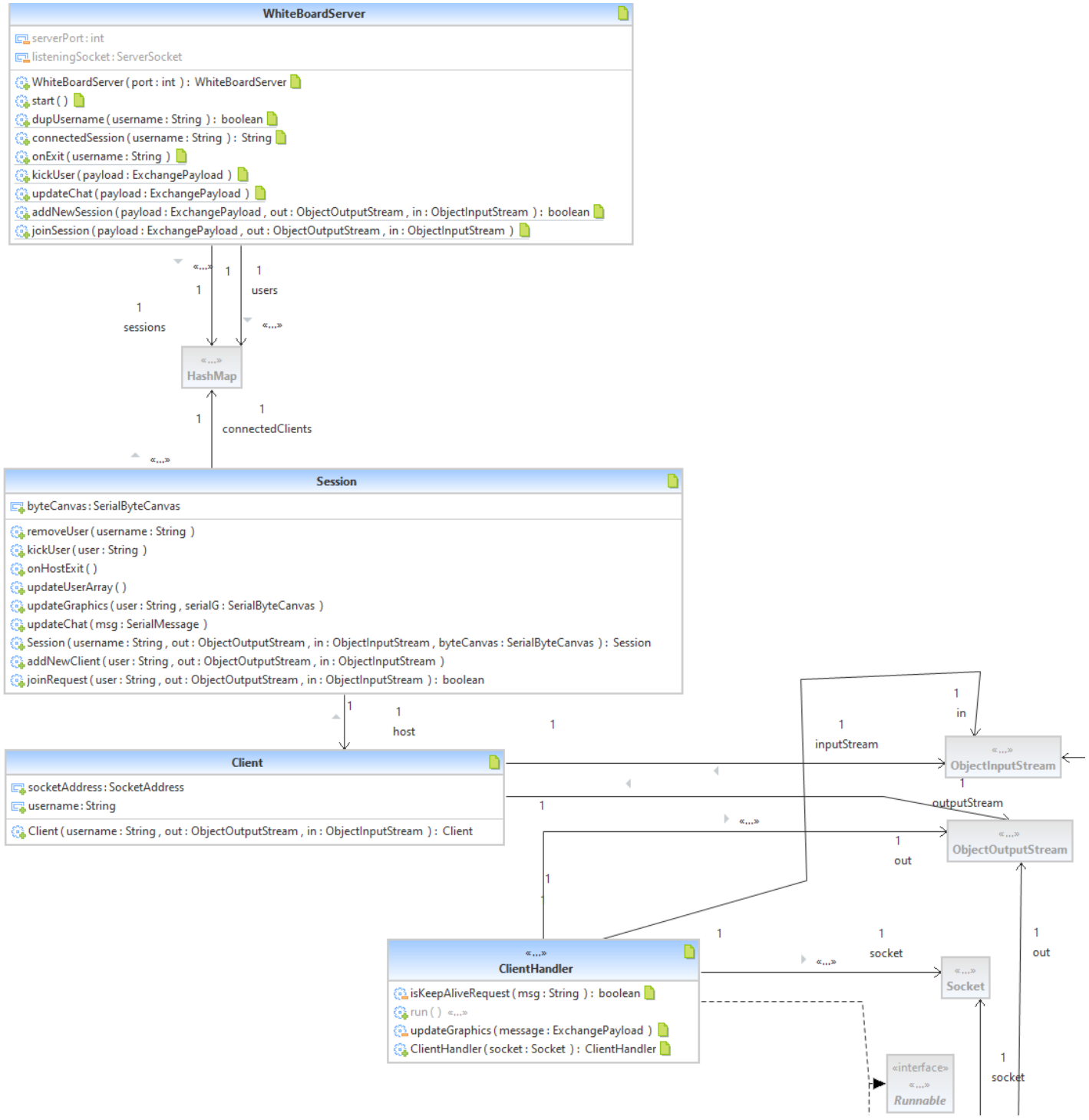

Figure 5: Server UML Class Diagram

The White Board Client UML class diagram is shown in Figure 6. The extra connection shown are the interaction to the client. The Whole diagram can be found within the submission folder.

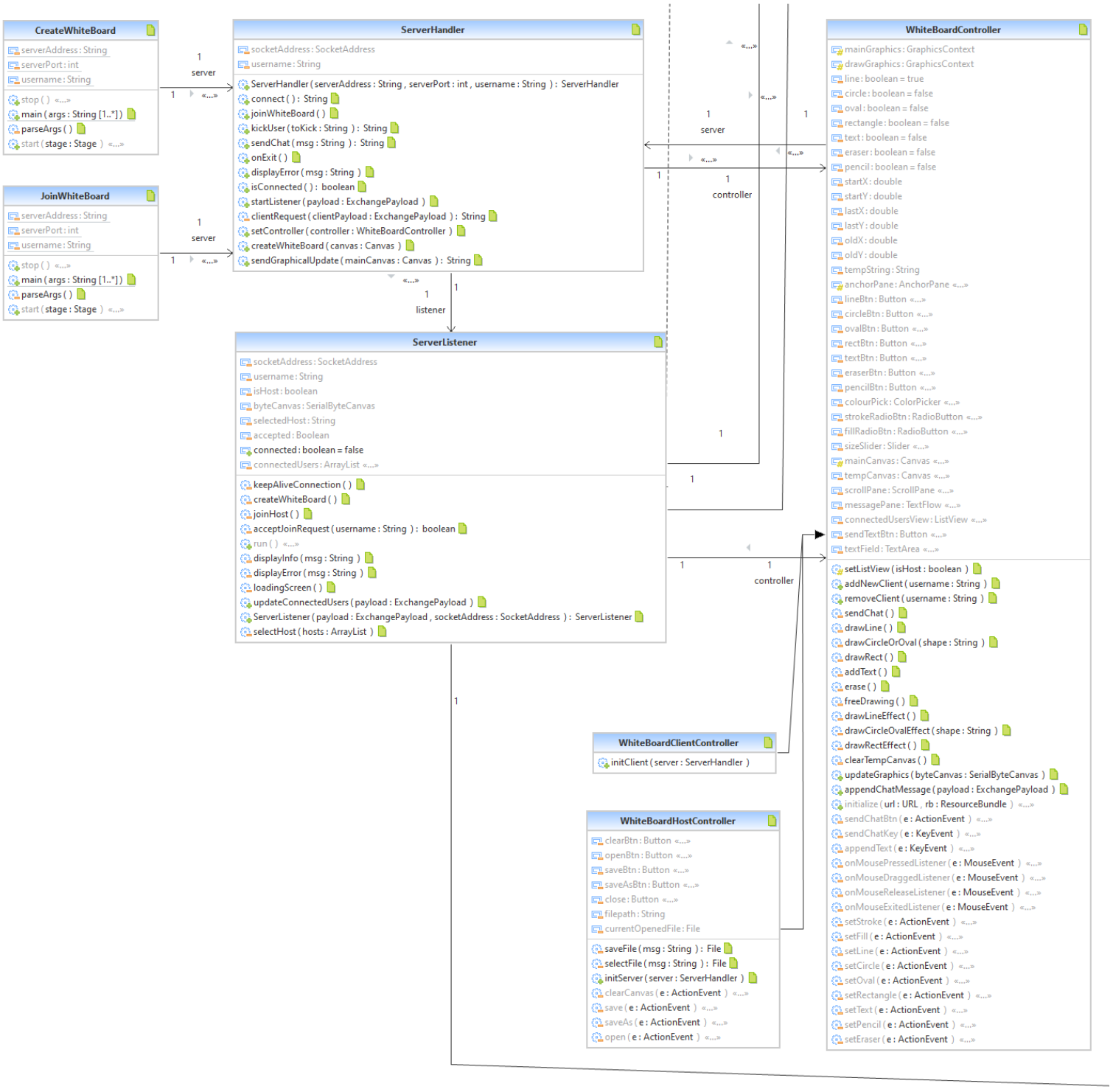

Figure 6: Client UML Class Diagram

Implementation

White Board Server

Server Architecture

The implementation of the thread-per-connection of white board server was programmed with the inbuilt java library. When a **White Board Server** is started, a socket is created bounded to the port initially inputted at before the start of program. Within a while loop, it'll listen for a new connection and start a new thread to handle the new connection and it's requests, see code snippet below.

```
//Create a server socket listening on port
listeningSocket = new ServerSocket(serverPort);

System.out.println("Server listening on port " +
        serverPort + " for requests");

//Listen for incoming connections for ever
while (true)
{
    // thread by connection, create a new thread per client
    Thread t = new Thread(new ClientHandler(listeningSocket.accept()));
    t.start();
}
```

The **ClientHandler** class implements a Runnable class. This is where the server will read the header of the Client Request Message and reply with if the Server was able to successfully complete the Client's request. The steps of how the server would handle each request in **ONE** single thread from start to finish:

1. Read the exchange payload through a ObjectInputStream
2. Retrieve the required information – request protocol.
3. Through multiple if statements check request Header to decide what operation the server will do. For example, if clientReq == CONNECT,
4. Check if duplicate username already exist in the users hash map
5. If no duplicate found, reply with "SUCCESS", if found, reply with "DUPLICATE"
6. reset() the ObjectOutputStream
7. Finally, close the socket if client has not.

Furthermore, when a client has connected to the client, the client must then request either to create or join a session. When a user request to "CREATE", a new session is created within the server. When a user request to "JOIN", a reply with a list of available sessions is sent back for the user to select from.

If no sessions are being hosted, then the server will reply with an error once the user selects a host to join. The server will send a request for confirmation to accept or decline the join request to the Host, using the socket stored upon "CREATE" request. Once confirmed, the user will be added to the session.

For any update request made by client, for example, if an update (canvas change) request is invoked by a client, the server will access the session which the client resides in. Then appropriately send an update request to all clients connected to the session using the open connection stored at "CREATE" and "JOIN" request. See code snippet below.

```
public synchronized void updateGraphics(String user,
        SerialByteCanvas serialG) throws Exception
{
    // set the current white board to be this
    this.byteCanvas = serialG;

    ObjectOutputStream objectOut;

    // for each client connected to this session, update their canvas
    for (Client client : connectedClients.values())
    {
        if(client.getUsername().equals(user))
            continue;

        System.out.println("Updating canvas of user :" +client.getUsername());
        objectOut = client.getOutputStream();
        objectOut.writeObject(new ExchangePayload(
                Protocols.Server.Request.UPDATE, user, this.byteCanvas));
        objectOut.reset();
    }
}
```

Another example is when the host exit out of the current session. An “EXIT” request will be sent to the session to shut down the session that the host was hosting, but first it will check if the user who sent the request was a host by checking the sessions hash map. Once confirmed the onHostExit() is invoked on the session object. All connected clients will be kicked out of their session with an error message. See code snippet below.

```
public void onHostExit() throws IOException
{
    ObjectOutputStream objectOut;

    // for each client connected to this session, send EXIT request to each
    for (Client client : connectedClients.values())
    {
        if(client.getUsername().equals(host.getUsername()))
            continue;
        objectOut = client.getOutputStream();
        objectOut.writeObject(new ExchangePayload(
                Protocols.Server.Request.EXIT));

        objectOut.close();
    }
}
```

White Board Client

The white board’s GUI is implemented using Javafx solely because of the availability of scene builder that provides a better UI editor than Java Swing.

Client UI and features

When joining a white board session, a list of hosts who the user can connect to is displayed see figure 7.

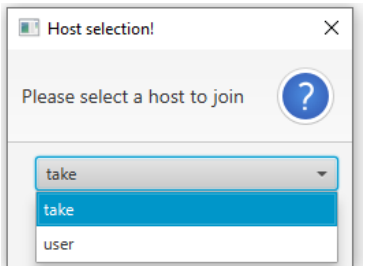

Figure 7: Host Selection Dialog

Once the host has accepted the join request. The white board is then editable. First, it will load the current canvas of the session. Figure 8 displays the tools available for a “Client” user.

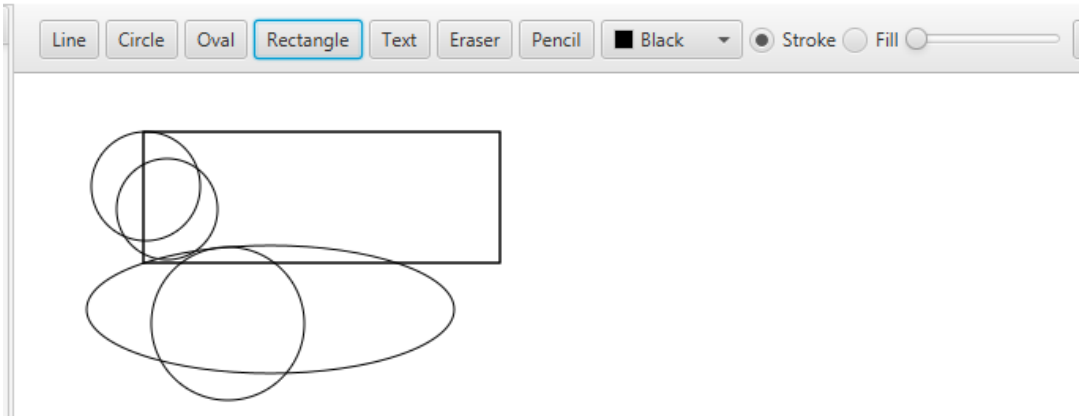

Figure 8: White Board Client UI

When a user edits the current canvas, the ServerHandler’s updateGraphics() method is called when mouse is released and the shape is drawn onto the canvas, or for every character entered when using the Text tool. The method sends the current image of the canvas to the server that will process the request. Then all connected clients will be updated with the changes in real-time. When the user exits out of the windows, stop() method is invoked and will send a “EXIT” request to the server to remove the client from the session.

When a new user is connected to the session, the connected client UI will be updated with the new user shown on Figure 9.

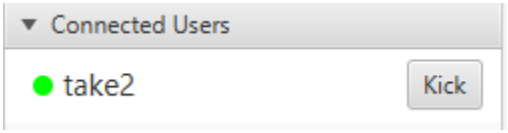

Figure 9: Connected users UI.

A chat feature is available, shown in Figure 10. This chat is updated to all connected clients in the session using the same sequence as a canvas change update request. The Connected Users and Chat is on a split pane thus, their size can be adjusted by the slider between the two panes.

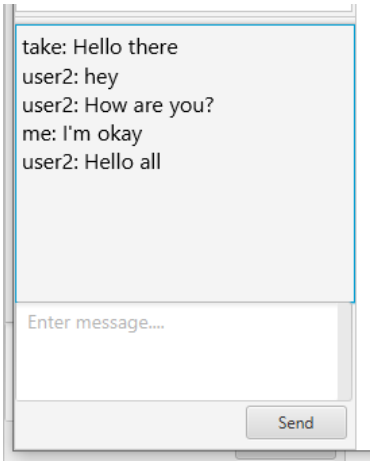

Figure 10: Chat Feature

Host UI and features

When a white board session is created, it is assumed that the user who created this is the host, thus using the host controller that is for processing the actions of UI changes/clicks. The host GUI is displayed to the user. The host have all features that was available to the normal client but includes more features. A different view with these features is displayed, such as “New”, “Save”, “SaveAs” and “Open” button are visible on the toolbar shown on figure 11.

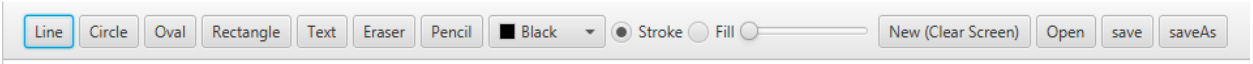

Figure 11: Host Toolbar features

“New” will clear the screen.

“Open” will open a file chooser for the user to load an image or previously saved state of the canvas into the current canvas.

“save” will save the current state of the canvas into the opened file. If no opened file was selected, the UI will request for a new save.

“saveAs” will save the current state of the canvas into an image, that the user can save. Two image file extension are available, “jpg” and “png”.

Another feature is on the Connected User’s UI, the host may kick a user if they would like to, shown on figure 12.

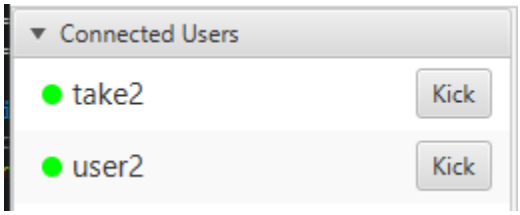

Figure 12: Host Connected Users UI

A confirmation dialog will display when the host clicks “Kick”, if the host click yes then a “KICK” request will be sent to the server to kick the user off this session.

Running the application

Requirements

- Java JDK 16 – if unsure run “java -version” in command line to check
- Javafx downloaded otherwise, it is provided with the source code.

Running each component

Since Javafx requires the library path and modules to be added to the command, the guide will use the library path provided with the uploaded code.

1. Navigate to the root folder of the submitted files.
2. Run the server.
   - java -p "javafx\lib" --add-modules "javafx.controls,javafx.fxml,javafx.swing" -jar server.jar <portnumber>
3. Create/host a new white board session.

- java -p "javafx\lib" --add-modules "javafx.controls,javafx.fxml,javafx.swing" -jar CreateWhiteBoard.jar <serveraddress> <portnumber> <username>

4. Join a white board session.

- java -p "javafx\lib" --add-modules "javafx.controls,javafx.fxml,javafx.swing" -jar JoinWhiteBoard.jar <serveraddress> <portnumber> <username>

Conclusion

In conclusion, central server architecture model is a suitable choice for a white board application in which a company may want to use since the control of policies can be enforced within the server. When allocated with enough resource the server will be able to process updates even faster. In terms of technology, another framework would have been far more suitable than using standard java libraries such as Java's Remote Method Invocation dedicated to transfer and distribute Java classes through a network.