# COMP90015 Distributed Systems
# **Distributed Shared White Board**

Puja Shome | 1205745

## 1. Problem Context

This report documents the details of a Distributed Shared White Board, implemented based on client- server architecture. Collaborative or shared whiteboards allow multiple participants to draw simultaneously on a canvas. This implementation of collaborative whiteboard provides features such as : **freehand drawing, inputing text, drawing shapes such as lines, circles, triangles, rectangles**. It also provides a file menu with **New, Open, Save** and **SaveAs** options, to be controlled only by the manager (in this implementation it is the Server). The application also provides a **chat box** function to enable communication between users. Sockets and threads are the lowest level of abstraction used in this project for network communication and concurrency.

## 2. System Architecture

This project is implemented based on client-server architecture. The main components of this architecture are : Clients and a Server.

### 2.1 Server or The Manager

The Server can be invoked in 2 ways- (i) By running java -jar StartWhiteBoard.jar <IP> <port> <username> in the command prompt or (i) By running java -jar StartWhiteBoard.jar if no arguments are passed in which case the default IP, port and name are 127.0.0.1, 4444 and Server respectively. On invoking the server, the whiteboard GUI (*Fig 1*) opens up and the manager can draw on the blank canvas. The server/manager has special privileges to perform actions like : creating a new file, opening an existing file, saving a file or saving file to a new place or with a new title. These features are only exclusive to the manager, and clients are not allowed access to these features. On invoking the server, the server also creates a server socket on the specified port and listens for requests from clients to establish connection. When a clients makes a request to connect, the server has the authority to either accept or decline the request. If the server disconnects, the application is terminated and all users are notified.

### 2.2 Clients or Users

The Client can also be invoked in 2 ways- (i) By running java -jar JoinWhiteBoard.jar <IP> <port> <username> or (i) By running java -jar JoinWhiteBoard.jar if no arguments are passed in which case the default IP, port and name are 127.0.0.1, 4444 and “no name”. The Client uses the server's IP address and port number to send a request to establish a connection. If the server accepts the client's request,
the whiteboard GUI (*Fig 1*) opens up and the user can draw on the blank canvas. If the canvas has been drawn on prior to the user's joining, the whiteboard GUI is updated to its most

current state with all the changes made to the canvas visible to this new user. Clients can also connect and disconnect anytime.

## 3. System Design Features

### 3.1 TCP Sockets and Thread-per-Connection Architecture

The distributed shared whiteboard implemented in this project follows a Thread-per-Connection architecture and all communication between the server and clients is enabled via TCP sockets. When a client establishes connection with the server, the server creates a thread for that client and that thread is alive until the client is disconnected. The client can make multiple requests while the connection is active and all the requests are handled by the same thread. The server creates a new thread for every new client and their requests are handled concurrently by means of multithreading.

### 3.2 JSON Data Exchange Format

The message exchange protocol used in this project is JSON. When the manager or a user draws on his own canvas, the data of each such is packed into a JSON object and passed on to all other active whiteboards as a JSON string thus enabling all whiteboard GUI canvases to be updated simultaneously (*Table 1*). For the chat box JSON message format is used a well.

| **Key** | **Definition** | **Applicable for** |
|---|---|---|
| shapeName | Name of the shape to be drawn | Line, Pen, Oval, Triangle, Rectangle, Eraser, Text |
| shapeInitialX | X coordinate of mouse is press | Line, Pen, Oval, Triangle, Rectangle, Eraser, Text |
| shapeInitialY | Y coordinate of mouse is press | Line, Pen, Oval, Triangle, Rectangle, Eraser, Text |
| shapeFinalX | X coordinate of mouse release | Line, Pen, Oval, Triangle, Rectangle, Eraser, Text |
| shapeFinalY | Y coordinate of mouse release | Line, Pen, Oval, Triangle, Rectangle, Eraser, Text |
| shapeWidth | Thickness of shape outline | Line, Pen, Oval, Triangle, Rectangle, Eraser, Text |
| shapeColor | Colour for drawing | Line, Pen, Oval, Triangle, Rectangle, Text |
| isFill | True if fill shape with colour is chosen | Oval, Triangle, Rectangle |
| shapeText | The text message typed by user | Only for text |
| shapeFrameWidth | Width of Font Frame | Only for text |
| shapeFontWidth | Font size for text | Only for text |
| shapeStyle | Font style for text | Only for text |

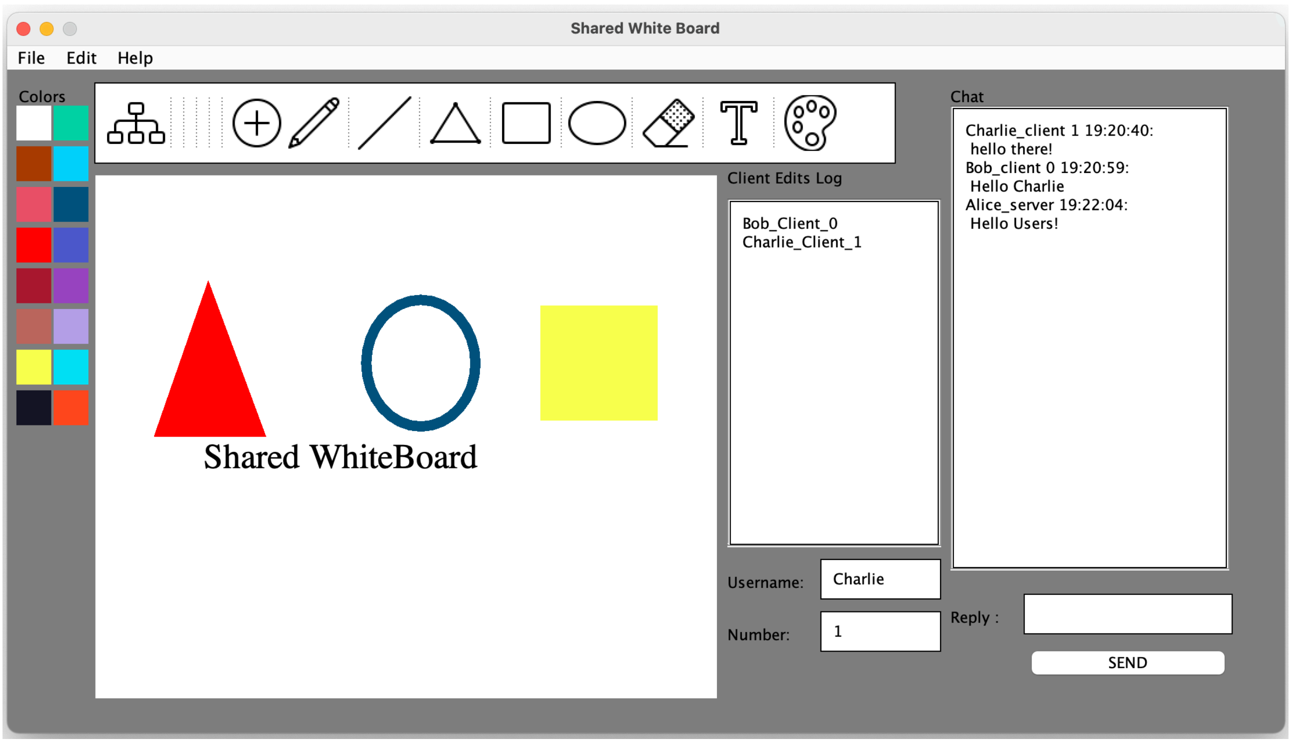
Table 1 : JSON Data Exchange Format for drawing on canvas

### 3.3 Robust Failure Model

All operation and connection errors are properly managed by means of exception handling on both the client side and the server side. The system is guaranteed communication reliability as TCP maintains order of the messages sent as well as prevents omission failures and data corruption. All errors such as incorrect input parameters, I/O errors and defects in network communication are properly handled.

### 3.4 Synchronised Access to Data

The system is fit to avoid any kind of inconsistent state by synchronising access to the data, i.e, if one thread tries to read the data and other thread tries to update the same data, only one of the threads is allowed access to that data at that time, thus maintaining consistency within the system.



Bob_Client_0
Charlie_Client_1

Charlie_client 1 19:20:40:
hello there!
Bob_client 0 19:20:59:
Hello Charlie
Alice_server 19:22:04:
Hello Users!

Username: Charlie
Number: 1
Reply :

Fig 1 : The WhiteBoard GUI

## 4. Implementation Details (Basic Features)

### 4.1 Concurrency

The Distributed Shared White Board application supports a single whiteboard to be shared between all of the clients (users) and the server (manager). When a client joins the whiteboard, the whiteboard GUI is updated to its most current state with all the changes made to the canvas visible to this new client i.e, all users see the same image of the whiteboard and have the privilege of doing all the drawing operations. Users can draw together in real time, without appreciable delays between making and observing edits. If one client makes edits to the canvas, all active whiteboard GUI canvases are updated simultaneously thus demonstrating concurrency.

### 4.2 Unique Identification of Users

Clients provide a username through the command prompt (java -jar JoinWhiteBoard.jar <IP> <port> <username>) to join the whiteboard. If no username is provided then the clients are

assigned unique numbers so that they can still be uniquely identified. Eg. : "Charlie_client 1" or "no name_client 2".
The Server or Manager can also provide a username through the command prompt (java -jar StartWhiteBoard.jar <IP> <port> <username>). Eg. : "Server_server" or "Alice_server".

## 4.3 Manager Authorise Entry

To enter the whiteboard all clients need permission from the Manager or the Server. When a client sends a request for connection, the manager receives a message saying that a user wants to join the whiteboard. If the manager accepts, the client joins the whiteboard, else the connection request is rejected.

## 4.4 Drawing Features

The application supports features such as **freehand drawing, inputing text, drawing shapes such as lines, circles, triangles, rectangles** and users can choose from **16 colours**. Apart from the 16 colours, users can also choose other colours from the Colour Palette. The buttons and their features are discussed in table 2. Right click action on the buttons reveal other features of the shapes such as line thickness or font style.

| Button | Function | Features (on right click) | Client / Server Function |
|---|---|---|---|
| 🖧 | Remove participant | N/A | Server |
| ⊕ | Create a new page | N/A | Server |
| ✏ | Draw with pencil | Size | Both |
| ╱ | Draw a Line | Line Style, Size | Both |
| △ | Draw a Triangle | Size, Fill | Both |
| □ | Draw a Rectangle | Size, Fill | Both |
| ⬭ | Draw an Oval/Circle | Size, Fill | Both |
| ⌫ | Eraser | Big, Medium, Small | Both |
| T | Text Inputting | Frame Size, Font Size, Style, Bold, Italic | Both |
| 🎨 | Colour Palette | N/A | Both |

Table 2 : The whiteboard Features(Basic)

## 4.5 Peer List

A peer list is also maintained by the whiteboard user interface that shows all the active clients using the application.

## 5. Implementation Details (Advanced Features)

### 5.1 Chat Window

A chat window has been implemented in this application to enable users to communicate with each other. The data exchange protocol is JSON. When a user sends a message in the chat box, all the data (the message, the client's username, the client number and the time when the message) is packed into a JSON object and passed on to all other active whiteboards as a JSON string. The message is print in the chat box in the format : Name + Client Number + Time + Message.

### 5.2 File Menu and Edit Menu

A **File** menu with **New**, **Open**, **Save** and **SaveAs** controls is also provided. These functions are only accessible to the managers, and not the users. The details of the functions are provided in the table below.

| | |
|---|---|
| New | To open a new File : a new canvas |
| Open | To open an existing file |
| Save | To save a file |
| SaveAs | To save a file to a new location or with a new title |

Table 3 : File Menu Functions

The **Edit** menu provides two functionalities : **Undo** and **Redo**.

### 5.3 Removing a Participant

The Manager also has the right to remove a participant at any time. The icon opens up a dialog box for the Manager to input the number of client to be removed.

## 6. System Analysis

### 6.1 Advantages

- This project uses TCP sockets which guarantee communication reliability by maintaining the order of the messages and preventing omission failures and data corruption.
- Thread-per-connection architecture has low overheads. This projects uses Thread-per-connection architecture where on thread is created per client and the client can make multiple requests via that thread.
- JSON is lightweight and easy to read /write. It is compact - an average JSON string is about two thirds of the size of the same data in XML.
- The system's failure model is robust and equipped to handle all kinds of errors including I/O errors, Network communication errors, input parameter errors, etc, on both the server and the client side.

## 6.2 Disadvantages

- Thread-per-connection architecture causes unbalanced load which can impact scalability.
- The system has only been run within test environments, with only a few clients. Thus load and response testing could not be possible.

## 7. Conclusion

In conclusion, a Distributed Shared White Board has been successfully implemented in this project with sockets and threads being the lowest level of abstraction. The whiteboard GUI is user friendly.  All the design choices made have been thought through thoroughly and the project has been tested against different kinds of errors.