Distributed systems report

Zimo Peng

1143743

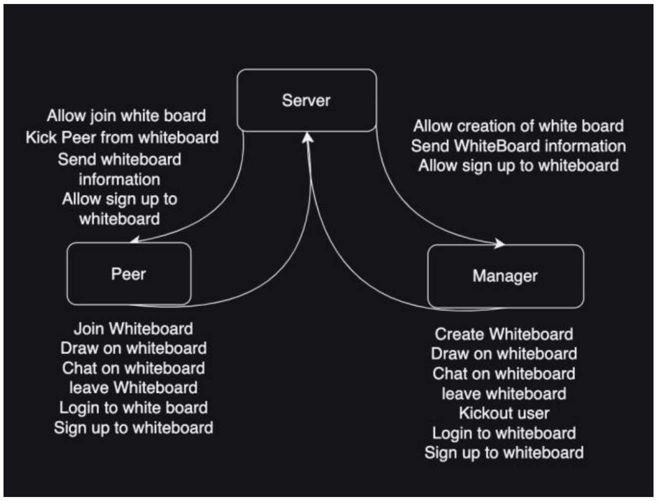
1.System architecture:

The system architecture that I went for was to have a central server, which store the drawings, shapes, texts and other users on the white board, and also all the functions that each client is able to perform on the server. A client can become a manager or a peer, but each whiteboard can only have 1 manager, and in order for there to be peers, a manager must first create the whiteboard. The server uses remote invocation to communicate objects from the server to the client, which sets up a TCP Connection.

The reasons why I chose a central server are:

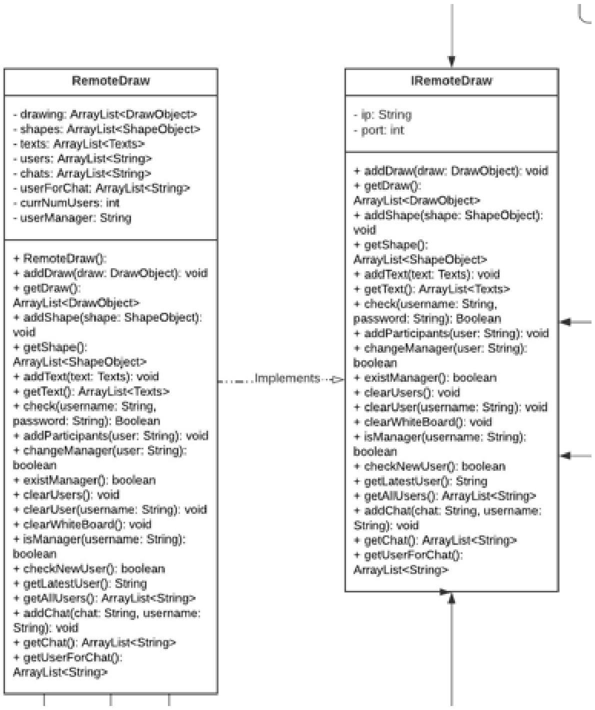
- Easy to detach a client from the server, as all you need to do is remove the user from the server variable, and connection closes.
- Every node updates to the same server, so the server is always in a consistent state.
- No need to fetch information from other servers when information is needed, so is efficient
- Cost efficient since all the variables and functions are on one server.

Here is the system architecture diagram:



2.Communication and message protocols:

The communication protocol RMI acts as the underlying communication protocol between the clients and the server. RMI allows clients to invoke methods on remote objects as if they were local objects, making network communicatin much simpler. I have one remote object named RemoteDraw, and the interface for the remote object is IRemoteDraw. This object holds all the arraylists of the drawings, shapes, texts, and users. It also holds all the functions that you can carry out on the whiteboard. Here is the class diagram:


RemoteDraw
- drawing: ArrayList<DrawObject>
- shapes: ArrayList<ShapeObject>
- texts: ArrayList<Texts>
- users: ArrayList<String>
- chats: ArrayList<String>
- userForChat: ArrayList<String>
- currNumUsers: int
- userManager: String
+ RemoteDraw():
+ addDraw(draw: DrawObject): void
+ getDraw(): ArrayList<DrawObject>
+ addShape(shape: ShapeObject): void
+ getShape(): ArrayList<ShapeObject>
+ addText(text: Texts): void
+ getText(): ArrayList<Texts>
+ check(username: String, password: String): Boolean
+ addParticipants(user: String): void
+ changeManager(user: String): boolean
+ existManager(): boolean
+ clearUsers(): void
+ clearUser(username: String): void
+ clearWhiteBoard(): void
+ isManager(username: String): boolean
+ checkNewUser(): boolean
+ getLatestUser(): String
+ getAllUsers(): ArrayList<String>
+ addChat(chat: String, username: String): void
+ getChat(): ArrayList<String>
+ getUserForChat(): ArrayList<String>
Implements
IRemoteDraw
- ip: String
- port: int
+ addDraw(draw: DrawObject): void
+ getDraw(): ArrayList<DrawObject>
+ addShape(shape: ShapeObject): void
+ getShape(): ArrayList<ShapeObject>
+ addText(text: Texts): void
+ getText(): ArrayList<Texts>
+ check(username: String, password: String): Boolean
+ addParticipants(user: String): void
+ changeManager(user: String): boolean
+ existManager(): boolean
+ clearUsers(): void
+ clearUser(username: String): void
+ clearWhiteBoard(): void
+ isManager(username: String): boolean
+ checkNewUser(): boolean
+ getLatestUser(): String
+ getAllUsers(): ArrayList<String>
+ addChat(chat: String, username: String): void
+ getChat(): ArrayList<String>
+ getUserForChat(): ArrayList<String>

The reasons I chose to use RMI are:

- Easy to communicate shared objects with clients, all clients access the same instance of the variables on the server.
- Easy to use shared objects. RMI allows us to use shared objects just as if they were our own local objects.
- It handles threads and sockets for communication
- RMI solutions are usually easy to extend by just adding new classes. Since, in the implementation, we have to constantly add new shapes, drawings, objects, this was a big aid to progressively making the whiteboard.
- RMI also has in built security features, meaning you won’t need to create them yourself.

We are using at-most-once invocation meaning the remote procedure call is executed exactly once. So the caller receives a response or it receives an exception saying it was not executed at all.

Using TCP socket and thread programming would be a lot more complicated, as we would have to find a message format to send through, and every time you add a new class or object, you would have to alter the message on the server and client so that it is received correctly.

Some disadvantages of RMI are:

- Need to unmarshal and marshal objects that are sent from node to node
- Need to make sure the objects that are sent are serializable, if they are new classes, you will have to extend the serializable class.
- All Remote objects must throw the remote exception, which needs to be caught on the client side.

But these disadvantages were negligible, because we were building a small scale distributed system, and the advantages greatly outweighed the disadvantages.

To communicate changes to all the clients, my whiteboards implement a timer based approach, where every 100 milliseconds, a server update is done so that all the local objects are updated to the state of the remote object. This keeps the whiteboard in a consistent state, and keeps up with all the updates within a very reasonable delay.

Synchronization Protocol: To handle real time collaboration of the whiteboard, a lot of issues were brought up. What if 2 clients used the same function at the same time? This was dealt with the addition of the synchronized keyword for all the relevant functions in IRemoteDraw and RemoteDraw

Error handling: Due to the usage of remote objects, Exception handling was a necessity since it is possible that a remote exception could occur. Or not bound exceptions since we are binding the registry to a port.

3.Implementation Details:

To open the server, run rmiregistry first, then java –jar RMIServer.jar portnumber

To access the whiteboard when the server is open, use java –jar CreateWhiteBoard.jar portnumber ipaddress . Or if you are a peer use java –jar JoinWhiteBoard.jar portnumber ipaddress.

We were made to implement a collaborative whiteboard where many clients can access the same whiteboard. I implemented one manager per whiteboard, and any number of peers for that whiteboard.

Before entering the whiteboard, the client needs to sign in to the whiteboard using a set username and password. If you are not a user yet, you have the option to sign up to the whiteboard.

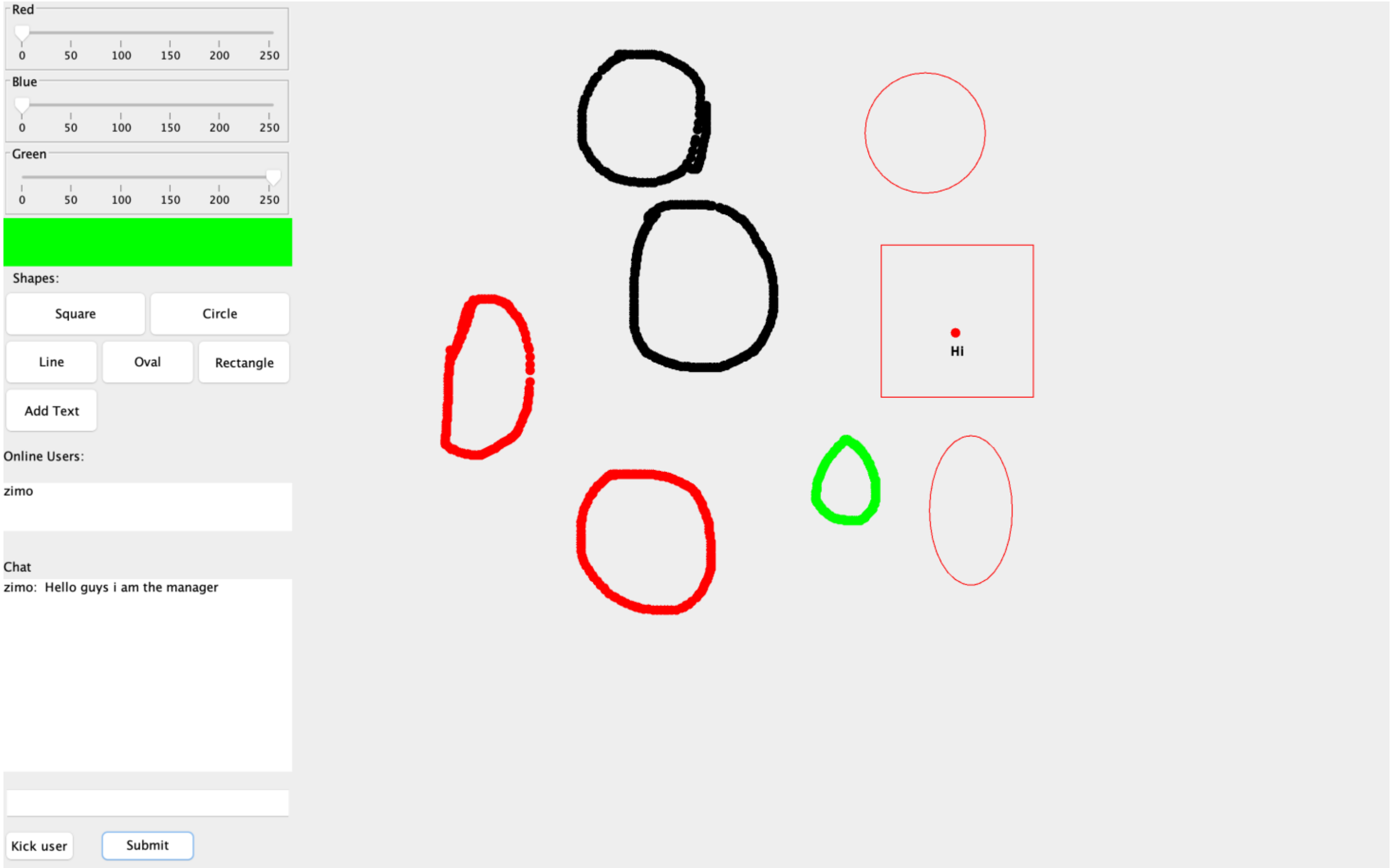
The manager of the white board has special commands that only he/she can use. They can click the kick user button, and kick a peer out of the network, which will exit the frame of that peer.

Managers are also the only ones that are allowed to create a whiteboard. Only 1 whiteboard can be created per server. When the manager leaves the whiteboard, all the users immediately leave the Server

Here are some of the important features on the whiteboard that I implemented:

- Freehand drawing: Client is allowed to freely draw on the whiteboard
- Shapes: Client is allowed to draw shapes, which they resize to add to the board. The available shapes are square, rectangle, circle, oval and line
- Text: Clients can add text to the board
- Colors: Client can choose to draw freehand and shapes in any color they wish, by choosing what they want on the slider.
- ChatBox: Clients can chat through the chat box which everybody can access.
- Online users: an online box which tells you everybody that is connected to the server.
- All methods are synchronized so that concurrent access to shared variables can be dealt with.

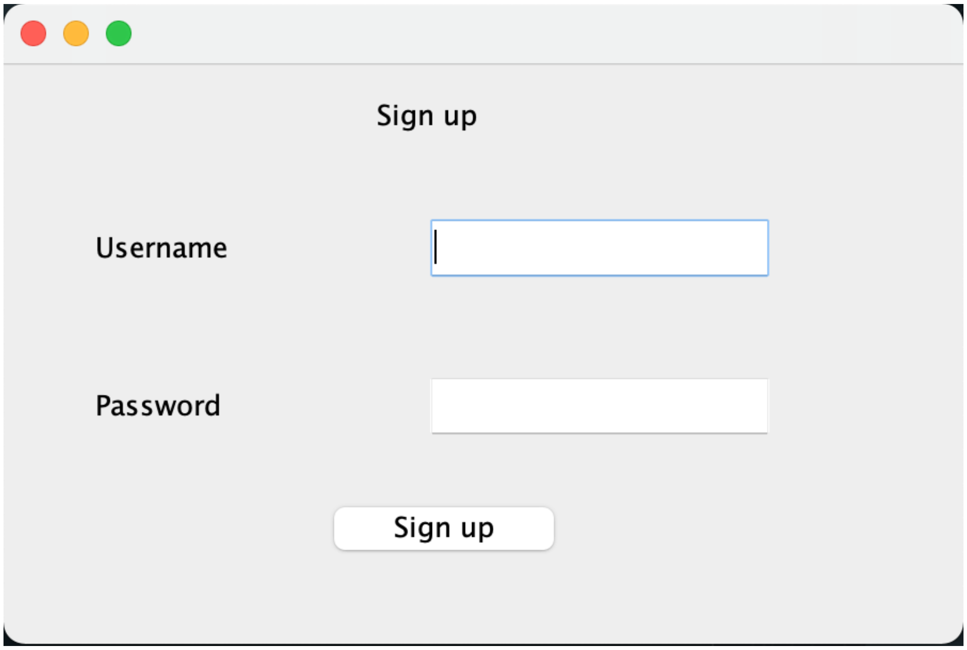
Here is a picture of the Client Manager whiteboard:



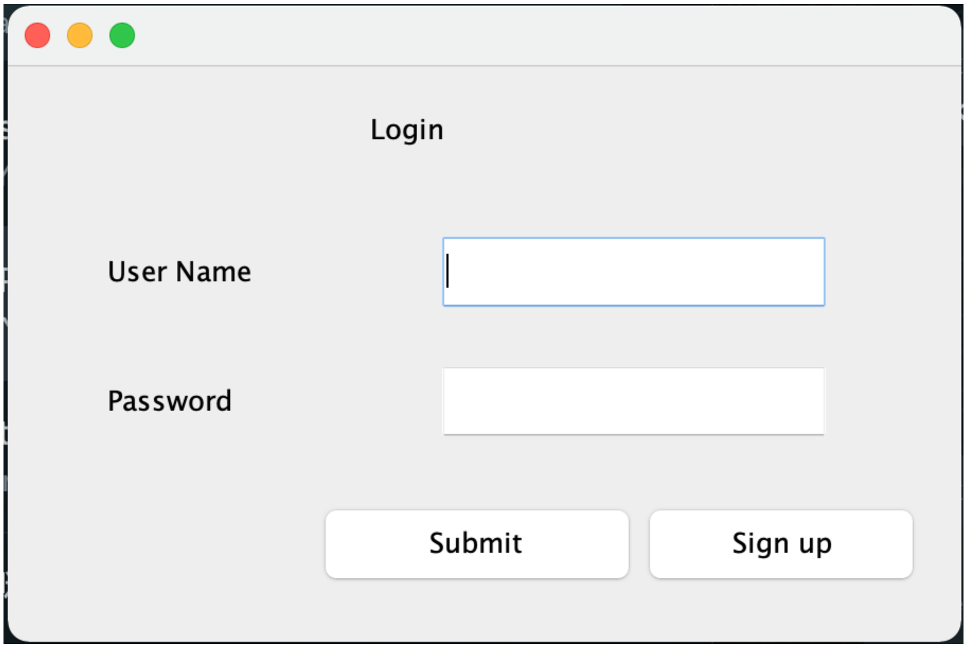
zimo

zimo: Hello guys i am the manager

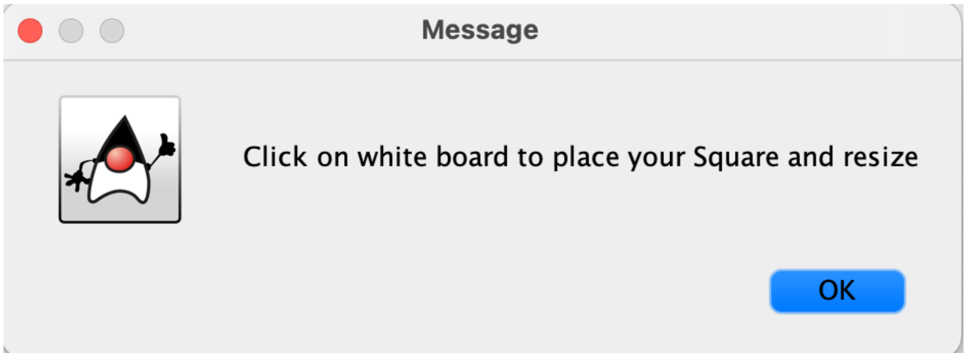
This is the sign up page, once you click the sign up button, the page closes and you will be able to login with your credentials. If you type in an already used username, you will encounter an error message and you will have to try another username



Here is the Login page, error message pops up if you type in the wrong password or username, or if the user is already part of the whiteboard:

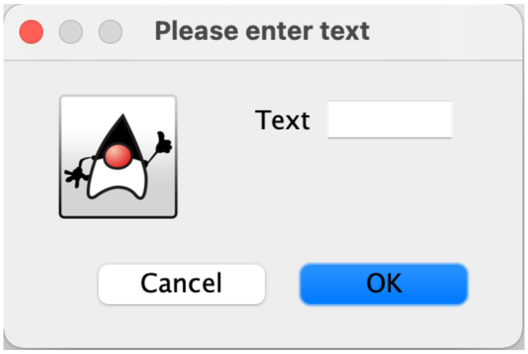


To Draw shapes, click on the shapes button and the following message will appear, you can size the shape how you wish:

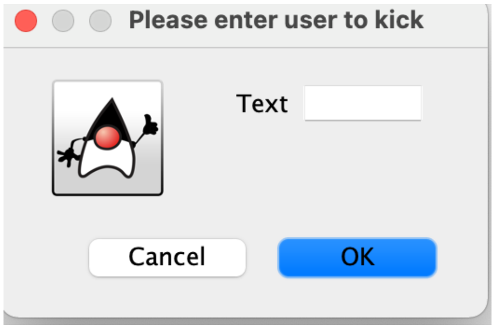


Click on white board to place your Square and resize

When you add a text, it prompts you to enter the text. Once you press ok, you can place that text anywhere on the whiteboard.



To kick a user, enter the username to kick, if it exists then the user will be kicked and the relevant peer will shut down the whiteboard, if it doesn’t exist, then the user will



Future implementation:

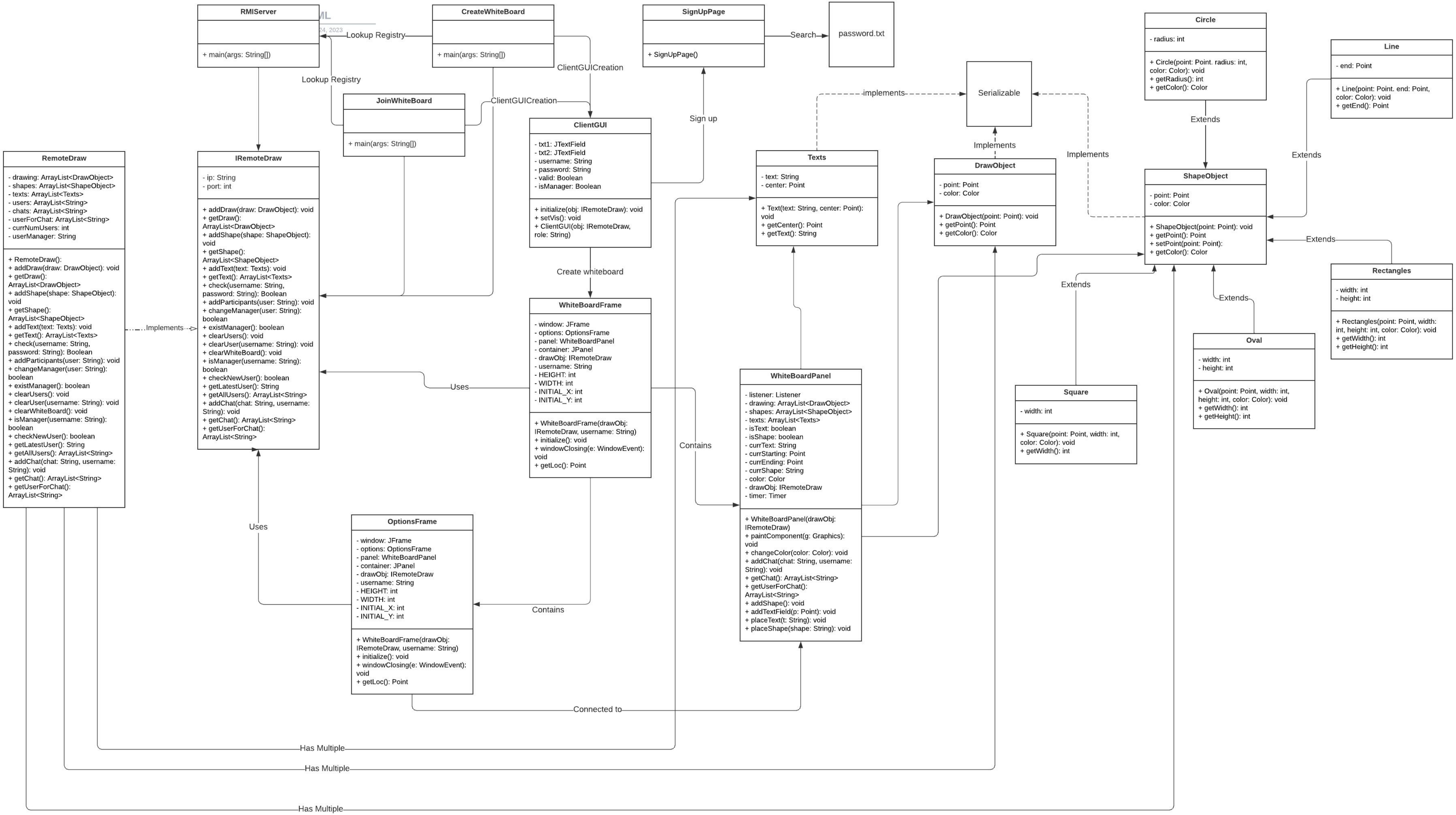
- In the future I would like to implement a more event based system, so that when an event happens, the change is updated to all clients. Currently I have a timer based system. When a change occurs, it is updated to the system. Every client has a timer, which every time the timer runs, the server state is given to the clients to update.
- I would like to implement the create, save and open file operations, but I was not able to due to time constraints.

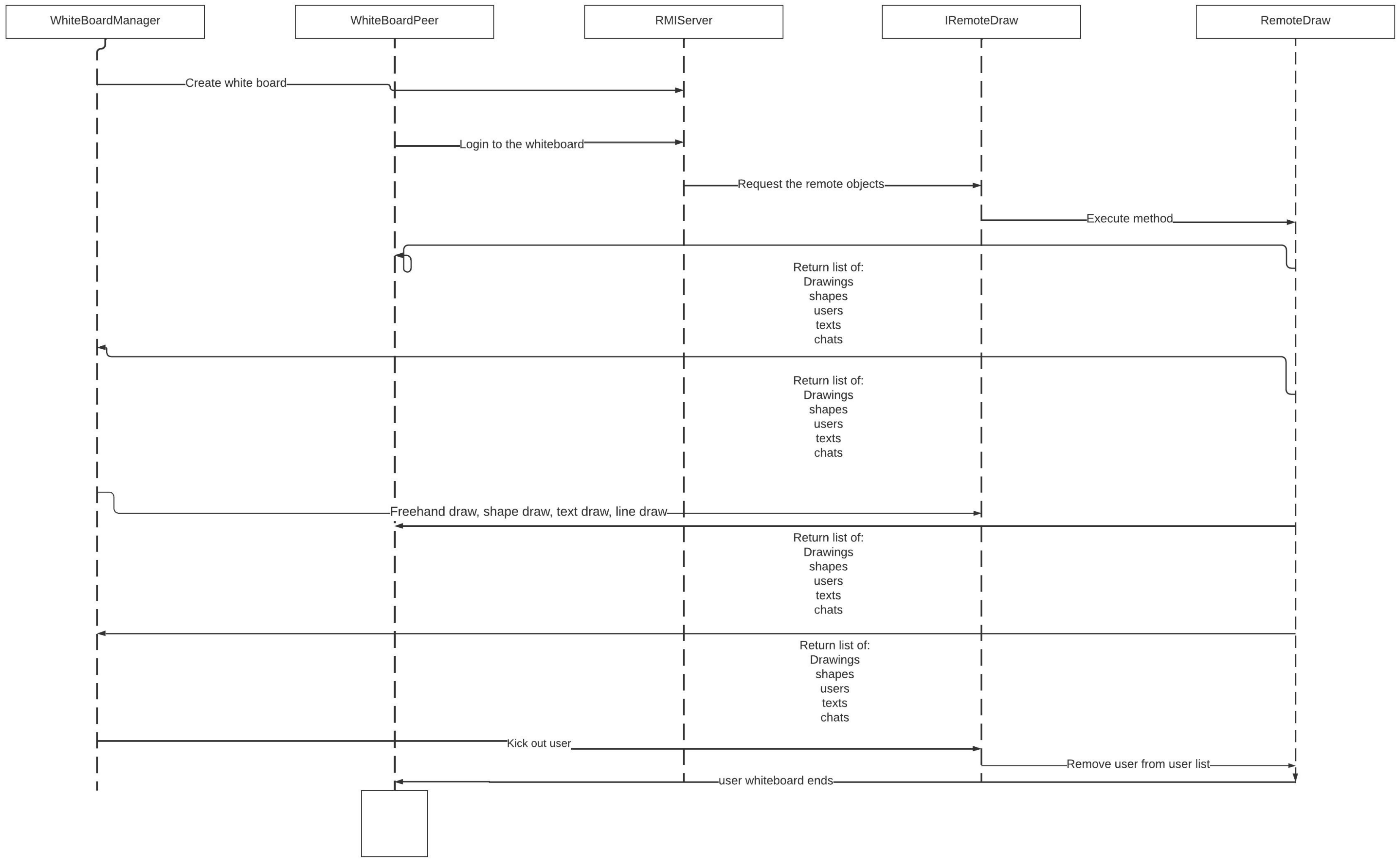
4.New innovations:

Some of the new innovations and creative ideas that I implemented were:

- Login and sign up page: This prevents the manager to accept peers when they join, since they will have to sign up to the server to access the whiteboard, otherwise they will not be able to access it. Both these pages require a username and a password.
- Colour sliders: Instead of having individual colors, I had 3 sliders, which indicated values of red, blue and green, which can be used to make any possible color. This means that every possible color can be used on the server.
- For Shapes and drawings I implemented classes to hold the different shapes that could be used. ShapeObject is the superclass, and oval, rectangle, circle etc. are the classes which extend the superclass. This allows the shapes to have their own behaviors. In the future if the shape behaviors want to be extended, then it is easy to do so. An example could be coloring only a certain arc of the square. This is better than holding strings of which shape is drawn because it is much more informative.
- Using a timer based system so that all whiteboards will always be in a consistent state. There will not be elements missing (within a small delay).

5.Design Diagrams:


RMIServer
+ main(args: String[])
CreateWhiteBoard
+ main(args: String[])
SignUpPage
+ SignUpPage()
password.txt
Search
Lookup Registry
Lookup Registry
ClientGUICreation
ClientGUICreation
JoinWhiteBoard
+ main(args: String[])
ClientGUI
- txt1: JTextField
- txt2: JTextField
- username: String
- password: String
- valid: Boolean
- isManager: Boolean
+ initialize(obj: IRemoteDraw): void
+ setVis(): void
+ ClientGUI(obj: IRemoteDraw, role: String)
Sign up
Create whiteboard
Serializable
implements
Implements
Implements
Circle
- radius: int
+ Circle(point: Point. radius: int, color: Color): void
+ getRadius(): int
+ getColor(): Color
Line
- end: Point
+ Line(point: Point. end: Point, color: Color): void
+ getEnd(): Point
Extends
Extends
RemoteDraw
- drawing: ArrayList<DrawObject>
- shapes: ArrayList<ShapeObject>
- texts: ArrayList<Texts>
- users: ArrayList<String>
- chats: ArrayList<String>
- userForChat: ArrayList<String>
- currNumUsers: int
- userManager: String
+ RemoteDraw():
+ addDraw(draw: DrawObject): void
+ getDraw(): ArrayList<DrawObject>
+ addShape(shape: ShapeObject): void
+ getShape(): ArrayList<ShapeObject>
+ addText(text: Texts): void
+ getText(): ArrayList<Texts>
+ check(username: String, password: String): Boolean
+ addParticipants(user: String): void
+ changeManager(user: String): boolean
+ existManager(): boolean
+ clearUsers(): void
+ clearUser(username: String): void
+ clearWhiteBoard(): void
+ isManager(username: String): boolean
+ checkNewUser(): boolean
+ getLatestUser(): String
+ getAllUsers(): ArrayList<String>
+ addChat(chat: String, username: String): void
+ getChat(): ArrayList<String>
+ getUserForChat(): ArrayList<String>
Implements
IRemoteDraw
- ip: String
- port: int
+ addDraw(draw: DrawObject): void
+ getDraw(): ArrayList<DrawObject>
+ addShape(shape: ShapeObject): void
+ getShape(): ArrayList<ShapeObject>
+ addText(text: Texts): void
+ getText(): ArrayList<Texts>
+ check(username: String, password: String): Boolean
+ addParticipants(user: String): void
+ changeManager(user: String): boolean
+ existManager(): boolean
+ clearUsers(): void
+ clearUser(username: String): void
+ clearWhiteBoard(): void
+ isManager(username: String): boolean
+ checkNewUser(): boolean
+ getLatestUser(): String
+ getAllUsers(): ArrayList<String>
+ addChat(chat: String, username: String): void
+ getChat(): ArrayList<String>
+ getUserForChat(): ArrayList<String>
Texts
- text: String
- center: Point
+ Text(text: String, center: Point): void
+ getCenter(): Point
+ getText(): String
DrawObject
- point: Point
- color: Color
+ DrawObject(point: Point): void
+ getPoint(): Point
+ getColor(): Color
ShapeObject
- point: Point
- color: Color
+ ShapeObject(point: Point): void
+ getPoint(): Point
+ setPoint(point: Point):
+ getColor(): Color
Extends
Rectangles
- width: int
- height: int
+ Rectangles(point: Point, width: int, height: int, color: Color): void
+ getWidth(): int
+ getHeight(): int
Extends
Extends
WhiteBoardFrame
- window: JFrame
- options: OptionsFrame
- panel: WhiteBoardPanel
- container: JPanel
- drawObj: IRemoteDraw
- username: String
- HEIGHT: int
- WIDTH: int
- INITIAL_X: int
- INITIAL_Y: int
+ WhiteBoardFrame(drawObj: IRemoteDraw, username: String)
+ initialize(): void
+ windowClosing(e: WindowEvent): void
+ getLoc(): Point
Uses
Contains
WhiteBoardPanel
- listener: Listener
- drawing: ArrayList<DrawObject>
- shapes: ArrayList<ShapeObject>
- texts: ArrayList<Texts>
- isText: boolean
- isShape: boolean
- currText: String
- currStarting: Point
- currEnding: Point
- currShape: String
- color: Color
- drawObj: IRemoteDraw
- timer: Timer
+ WhiteBoardPanel(drawObj: IRemoteDraw)
+ paintComponent(g: Graphics): void
+ changeColor(color: Color): void
+ addChat(chat: String, username: String): void
+ getChat(): ArrayList<String>
+ getUserForChat(): ArrayList<String>
+ addShape(): void
+ addTextField(p: Point): void
+ placeText(t: String): void
+ placeShape(shape: String): void
Square
- width: int
+ Square(point: Point, width: int, color: Color): void
+ getWidth(): int
Oval
- width: int
- height: int
+ Oval(point: Point, width: int, height: int, color: Color): void
+ getWidth(): int
+ getHeight(): int
Uses
OptionsFrame
- window: JFrame
- options: OptionsFrame
- panel: WhiteBoardPanel
- container: JPanel
- drawObj: IRemoteDraw
- username: String
- HEIGHT: int
- WIDTH: int
- INITIAL_X: int
- INITIAL_Y: int
+ WhiteBoardFrame(drawObj: IRemoteDraw, username: String)
+ initialize(): void
+ windowClosing(e: WindowEvent): void
+ getLoc(): Point
Contains
Connected to
Has Multiple
Has Multiple
Has Multiple


WhiteBoardManager
WhiteBoardPeer
RMIServer
IRemoteDraw
RemoteDraw
Create white board
Login to the whiteboard
Request the remote objects
Execute method
Return list of:
Drawings
shapes
users
texts
chats
Return list of:
Drawings
shapes
users
texts
chats
Freehand draw, shape draw, text draw, line draw
Return list of:
Drawings
shapes
users
texts
chats
Return list of:
Drawings
shapes
users
texts
chats
Kick out user
Remove user from user list
user whiteboard ends