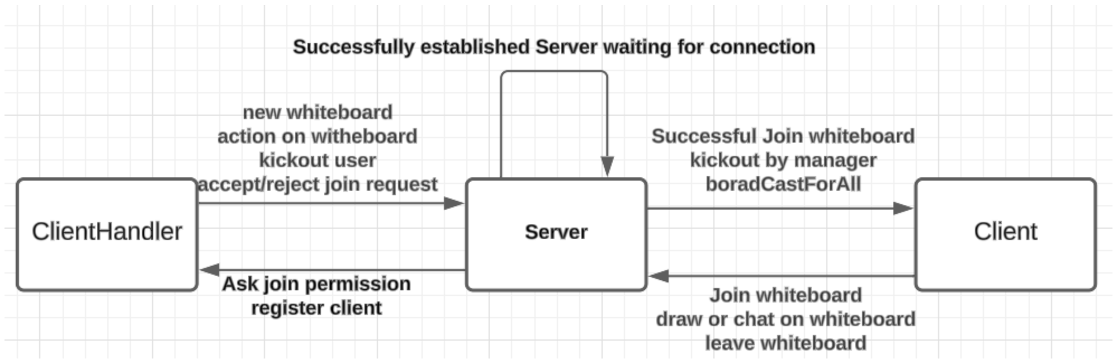
COMP90015 Project Report

Zhuoyang Liu 917183

System architecture

There is a central server, which will store all the whiteboard and User's information. Different users such as Manager and client can use the whiteboard differently. The server has RMIT objects which is ready for clients to use, and a TCP request will be send from clients' operations.

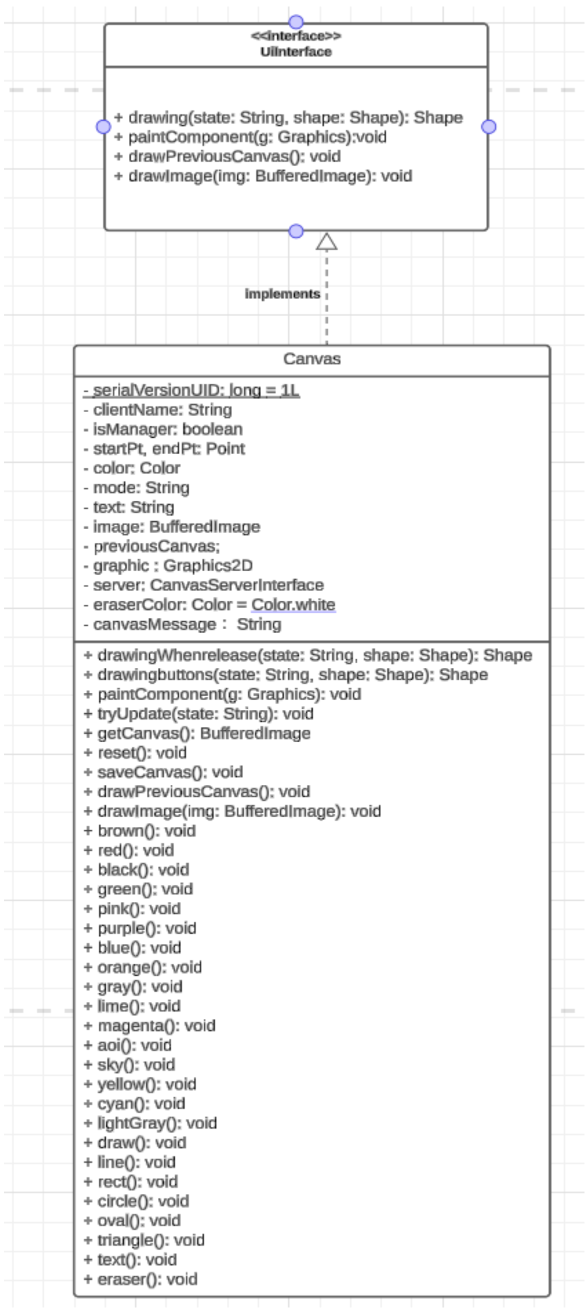


The advantage of using central server architecture will be showed as follows:

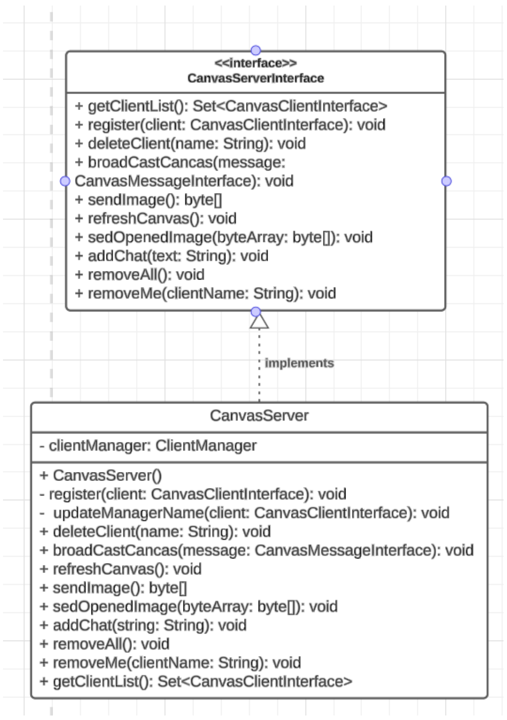
1. Central server architecture is easy to physically secure. Due to the location of server, the pressure of secure and service of server and client has been reduced to minimum.
2. User experience is much smoother. Each client will have their own dedicated system, which can be modified to suit custom need.
3. Dedicated resources (memory, CPU cores, etc.)
4. Less cost for small systems. Compared to other architecture, central systems take less funds to set up.
5. Since the server are located at on machine, it is much easier to updates.
6. Much easier to operate on client node. Since the client and server are separated, it only needs to remove the connection of the client node from the server voila.

Communication protocols

The RMI has been set as the communication protocol to handle remote object. In our system there are only one remote object has been passed through the RMI.



The drawing class on the whiteboard uses RMI for the system. The remote object will be client received when they are using the lookup remote object by the key from the RMI registry. The draw function is now available for the client by calling the remote methods.


<<interface>>
CanvasServerInterface
+ getClientList(): Set<CanvasClientInterface>
+ register(client: CanvasClientInterface): void
+ deleteClient(name: String): void
+ broadCastCancas(message: CanvasMessageInterface): void
+ sendImage(): byte[]
+ refreshCanvas(): void
+ sedOpenedImage(byteArray: byte[]): void
+ addChat(text: String): void
+ removeAll(): void
+ removeMe(clientName: String): void
implements
CanvasServer
- clientManager: ClientManager
+ CanvasServer()
- register(client: CanvasClientInterface): void
- updateManagerName(client: CanvasClientInterface): void
+ deleteClient(name: String): void
+ broadCastCancas(message: CanvasMessageInterface): void
+ refreshCanvas(): void
+ sendImage(): byte[]
+ sedOpenedImage(byteArray: byte[]): void
+ addChat(string: String): void
+ removeAll(): void
+ removeMe(clientName: String): void
+ getClientList(): Set<CanvasClientInterface>

The client manager class will be holding the client's information uses RMI for the system. Same as drawing class, the client is able to receive the remote object when they are using the lookup remote object with the key from the RMI registry. The client can take the action such as adding new user or remove current user etc. through the connections.

The following advantages encourage me to use the RMI for the component objects:

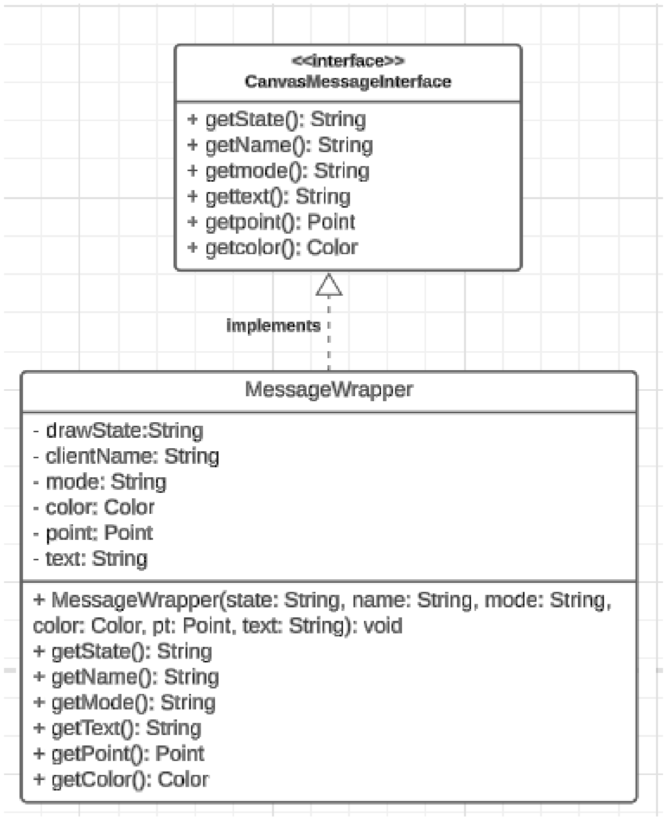
1. Managers threads and sockets to facilitate communication at the core level, which allowed developers to explore more possibilities.
2. Server-side alterations can be mode seamlessly, shielding clients from the need for updates.
3. Extending an RMI solution is a breeze, User can easily introduce new classes or extend existing ones as in a standalone application.

As we are expected to extend the functionality of the canvas such as add more shapes or add more operation buttons, using RMI for the canvas object and userList object for storage user information will be beneficial. Compared to TCP connection, it will be very difficult to make sure the consensus of the data. Although Overhead of marshaling and unmarshaling and overhead of object serialization are two important advantages for RMI. But due to the object is simple, they may not affect too much.

I have also tried to use the TCP as the communication between client and servers. It will allow server to process the request such as Create whiteboard, kickout user, User leave Whiteboard etc. However, due to those disadvantages I have chosen to use RMI 1): Delay in speed of communication. 2): Complexity.

Message formats:

At the first, the message is Json formatted. The message format has been set as {“message type”, “action takes”, “Color”} and other key, value pair as the server request. But the Json formatted still need lost of time to split and decompose the data passed to the server and process them. So, I tried to encapsulate the information that needs to be sent into a class, and then send this class to the backend through the information communication system. This change greatly reduces the complexity of the back-end code and reduces the degree of coupling.



Design Diagrams:

Class diagram:

The detail has been shows in the class diagram. Get inspiration from the explanation of RMI architecture in workshop 7. Server and client will be two classes that start the entire system, and they correspond to the central server and client respectively. It is worth noting that the central server will automatically set the first user connected to the server as the manager, which greatly reduces the coupling of the code and improves future scalability.

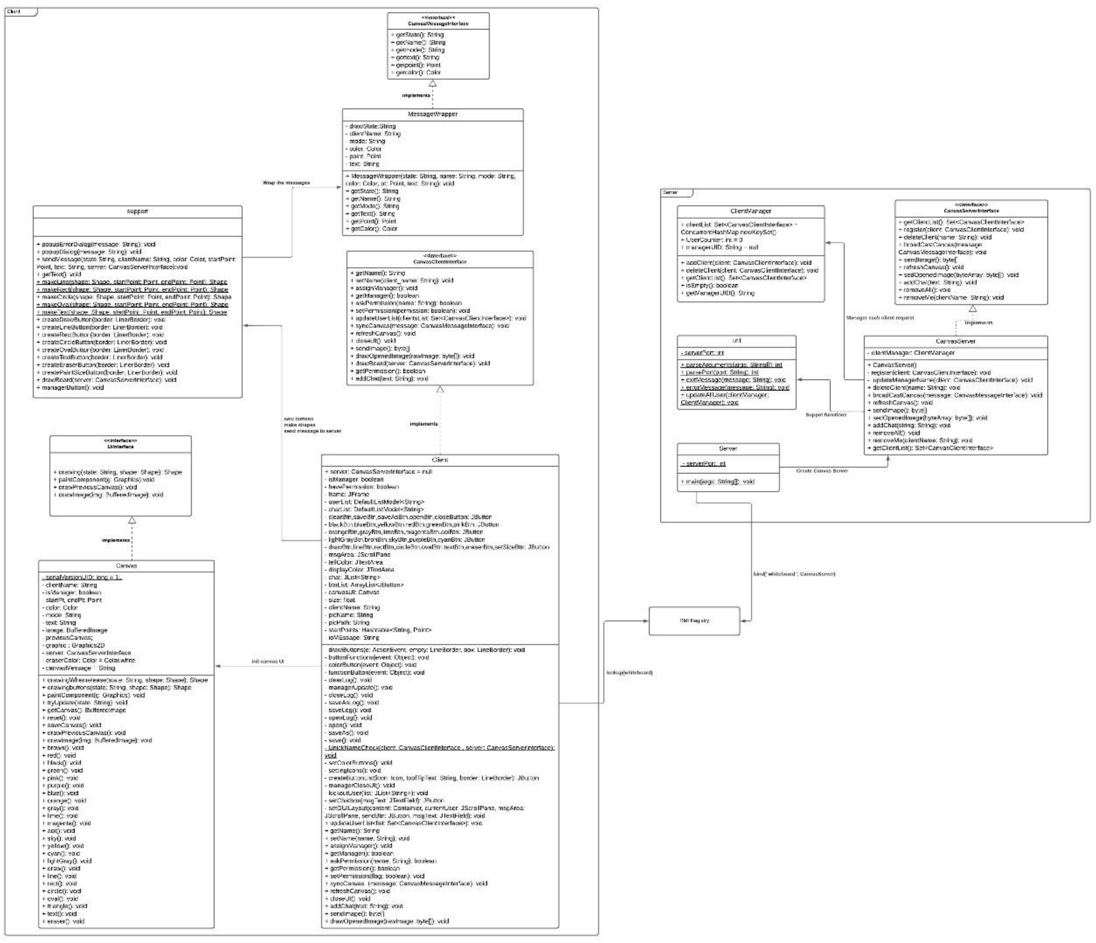
After the server is set up, Client will start look up the RMI remote canvas and user list from the server. At the same time, some general auxiliary methods are also placed in the util class. Some simpler logical tasks are placed in the support class. The Client class will also handle requests sent by other clients and send responses. It will implement multiple clients by creating new threads.

At the same time, the client contains the front-end GUI layout and logic classes. It will initialize the UI components, and if the user is an administrator, it will open functions such as File Menu to it and allow it to perform functions such as kicking. If it is an ordinary user, these functions that are only open to administrators will be hidden, and only basic drawing, chatting and other functions will be allowed. At the same time, the Support class will also assist the client class. Simple new buttons and simple drawing board logic will be implemented in this class. Then the functions of integrating the shapes recorded in the canvas and some common pop-up messages were implemented in the util class.

In the Canvas class. The canvas GUI will be initiated by this class. And it will also handle the mouse click, mouse drag and mouse release action. Those actions will be saved and an object and transfer to Client class which have the handler for each action.

In the remote package. There are three interface which will be implemented in both server and client packages. Those interfaces are mainly forcing on the use RMI for client server communication. The Canvas will be drawn using a BufferedImage. However, the BufferedImage class in JAVA is not serializable, so it cannot be transferred in RMI communication. So, I tried to use the Shape function to serialize the BufferedImage so that the BufferedImage can be used as a whiteboard canvas, which will allow it to be communicated to the server by RMI.

At the same time, there is a class named synchronization in the client class. It will record the shape drawn by the user in the Canvas and share it with every user recorded in the User list through RMI communication. The user list is initialized in CanvasServer and controlled by the clientManager class. It will allow users to add, delete, share and other changes. At the same time, he will also send a request to the manager for permission.

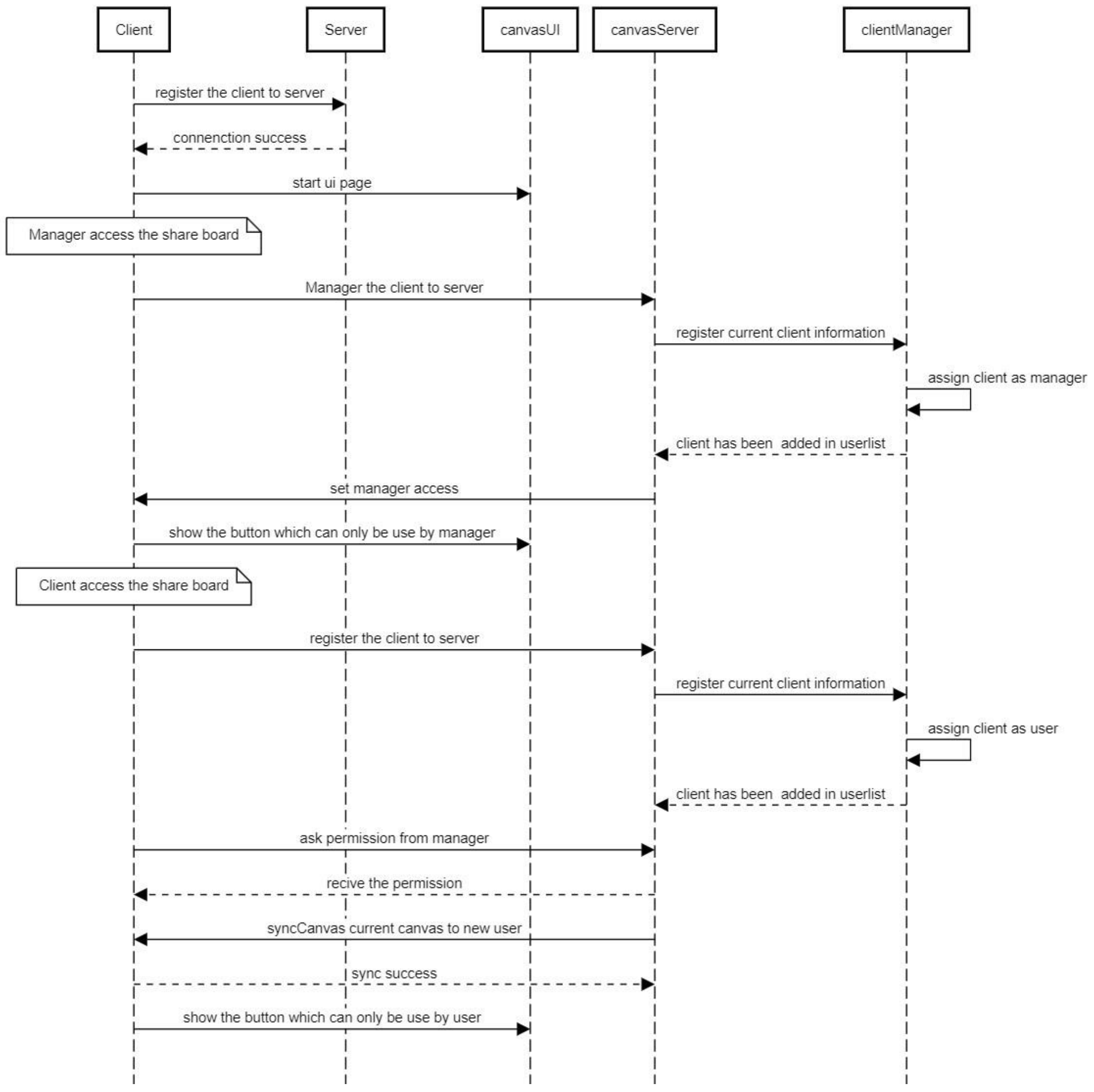

Client
<<interface>> CanvasMessageInterface
implements
MessageWrapper
Wrap the messages
support
<<interface>> CanvasClientInterface
implements
new buttons make shapes send message to server
<<interface>> UIinterface
implements
Canvas
Client
init canvas UI
Server
ClientManager
<<interface>> CanvasServerInterface
Manager each client request
implements
util
CanvasServer
Support functions
Server
Create Canvas Server
bind("whiteboard", CanvasServer)
RMI Registry
lookup(whiteboard)

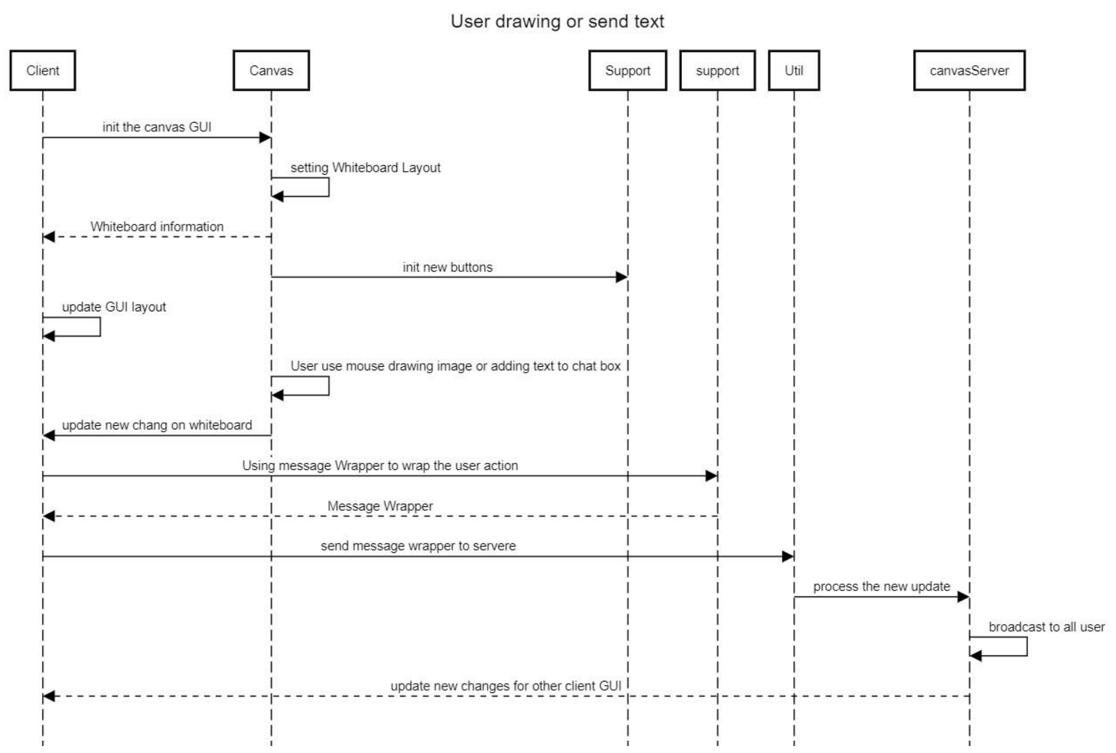
Sequence diagram:

The first sequence diagram is the process for the client to establish a link with the server. Then the server performs different operations on administrators and different users by calling canvasServer class and clientManager class.

The second sequence diagram is the process for the manager or user to start drawing an image on a shared artboard

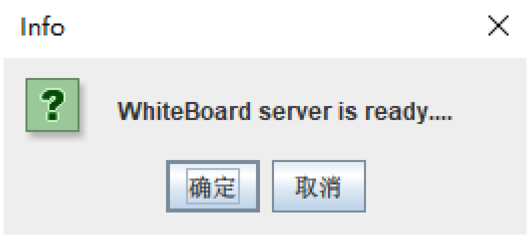
Client acces Shared Whiteboard


Client
Server
canvasUI
canvasServer
clientManager
register the client to server
connenction success
start ui page
Manager access the share board
Manager the client to server
register current client information
assign client as manager
client has been added in userlist
set manager access
show the button which can only be use by manager
Client access the share board
register the client to server
register current client information
assign client as user
client has been added in userlist
ask permission from manager
recive the permission
syncCanvas current canvas to new user
sync success
show the button which can only be use by user


User drawing or send text
Client
Canvas
Support
support
Util
canvasServer
init the canvas GUI
setting Whiteboard Layout
Whiteboard information
init new buttons
update GUI layout
User use mouse drawing image or adding text to chat box
update new chang on whiteboard
Using message Wrapper to wrap the user action
Message Wrapper
send message wrapper to servere
process the new update
broadcast to all user
update new changes for other client GUI

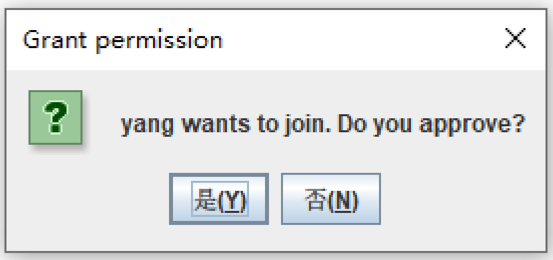
Implementation details:

When server is running it will pop up a message dialog.


Info

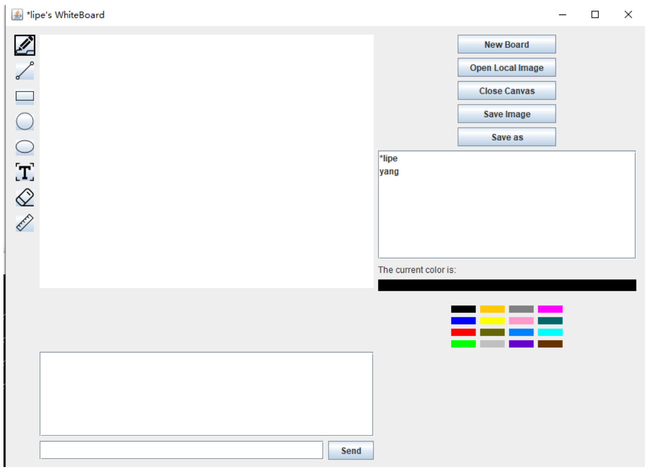
WhiteBoard server is ready....

The manager can accept or reject a new user to join the Canvas.

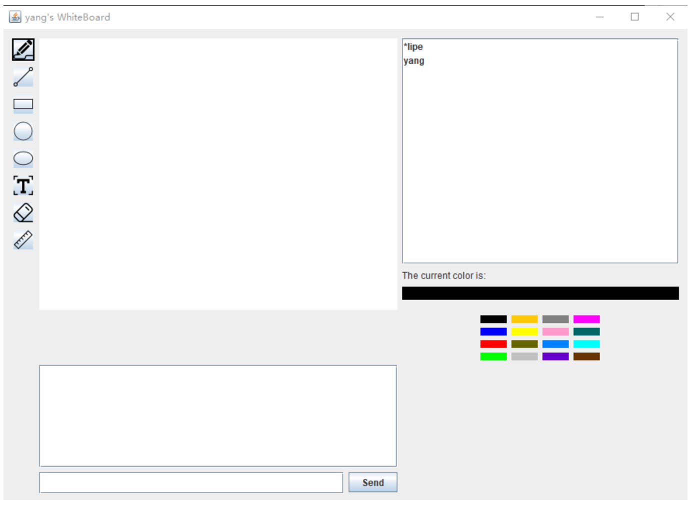

Grant permission

yang wants to join. Do you approve?

The first user who connect to this server will be assign as the manager and they will be highlighted with a “*” after their name. and the Manager will have the access to he function buttons: New Board, Open Local Image, Close Canvas, Save Image, Save as.

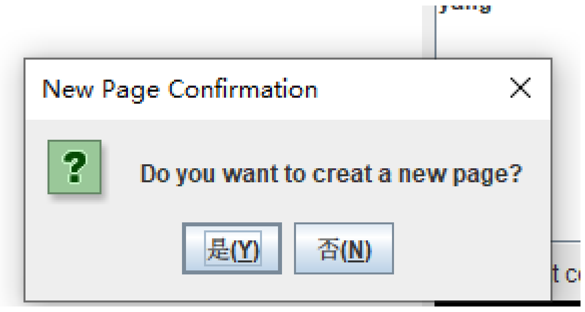


(Manager GUI)



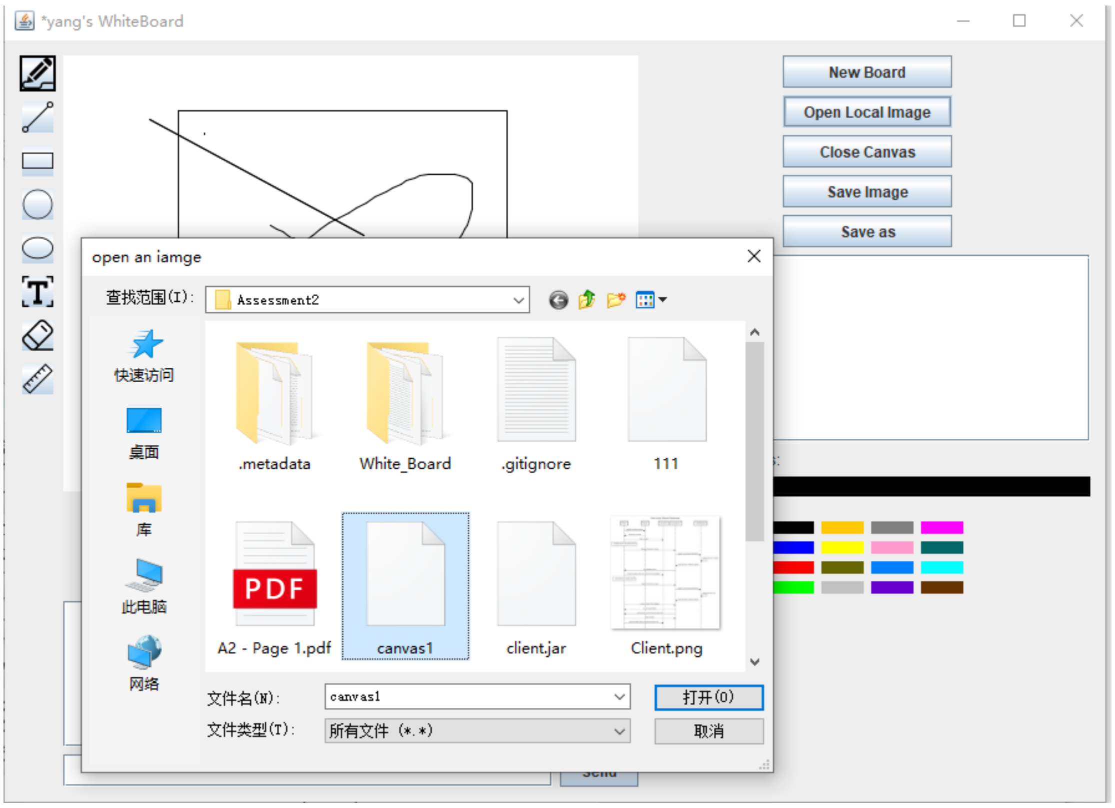
(Client GUI)

There will be message notification when manager want to new a white board.

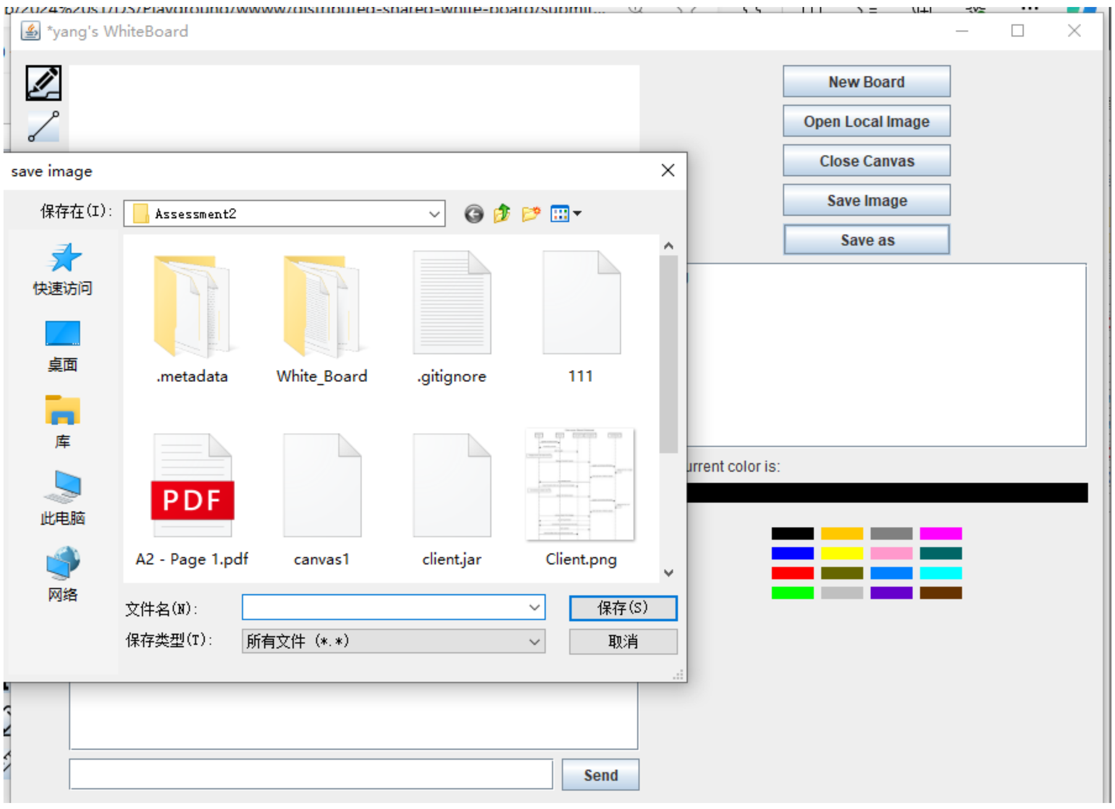

New Page Confirmation

Do you want to creat a new page?

Manager can open a saved whiteboard. And if a new board have not been saved before the manager can save the whiteboard by specifying the path and the file name.

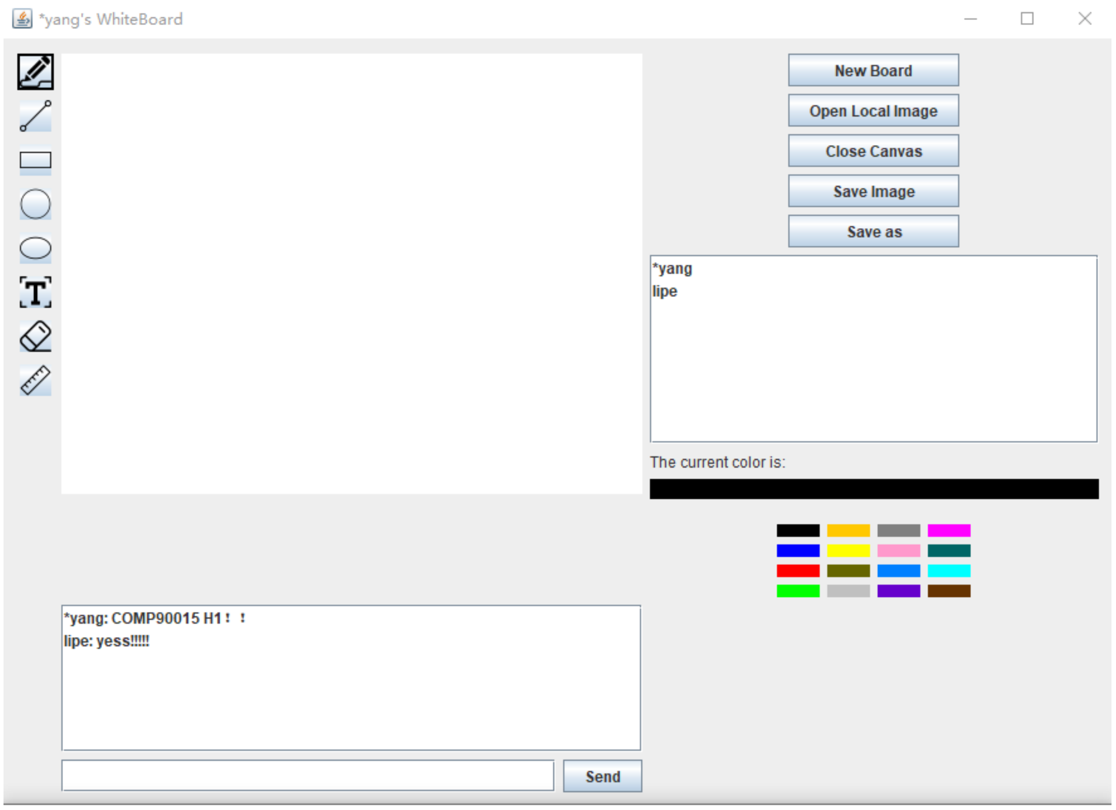


(Open a local image)



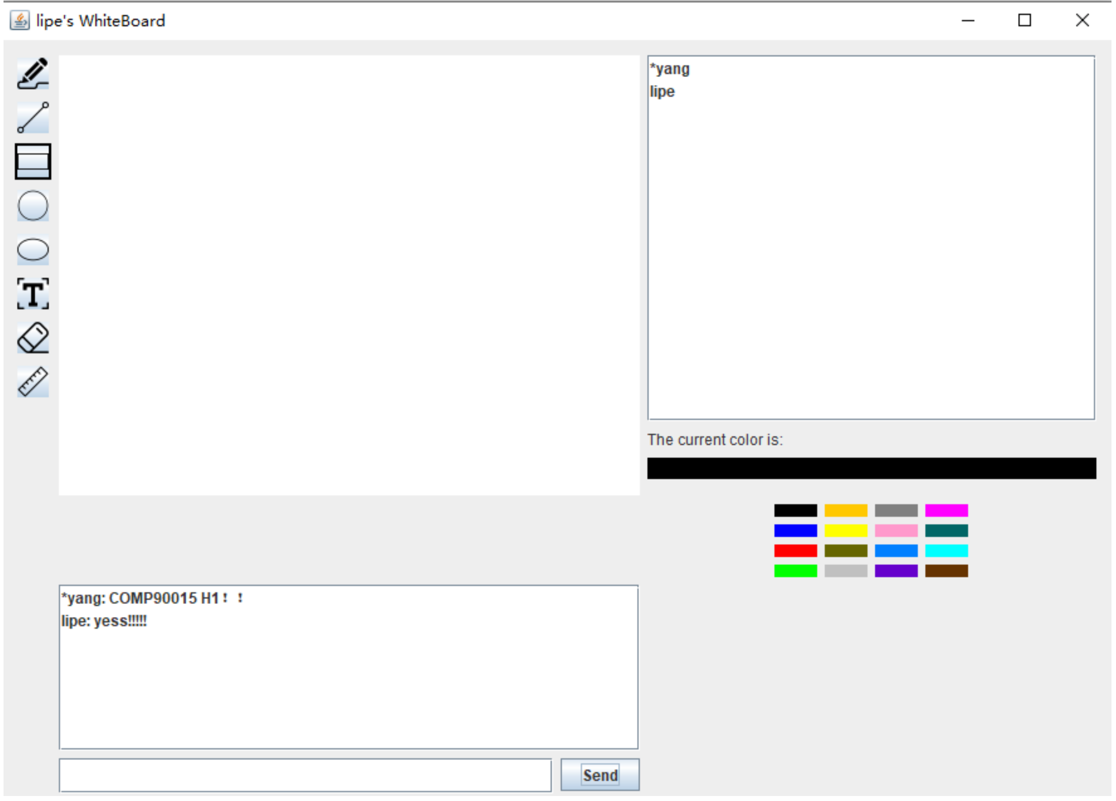
(First time click Save Image or Save as)

User can talk to each other through this GUI.



*yang
lipe

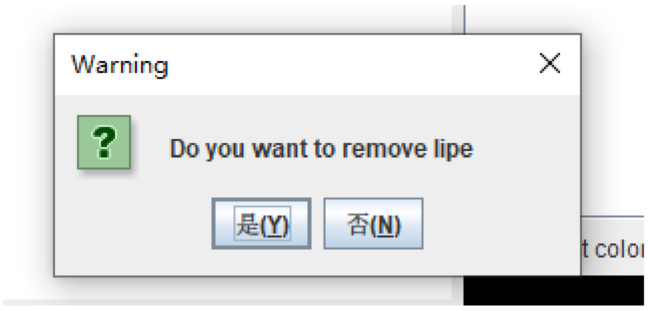
*yang: COMP90015 H1！ ！
lipe: yess!!!!!



*yang
lipe

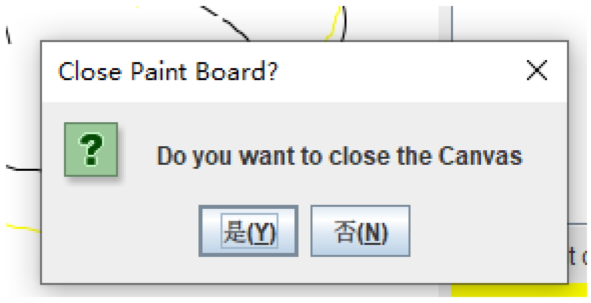
*yang: COMP90015 H1！ ！
lipe: yess!!!!!

And the manager is able to kick any one current in the Canvas.


Warning

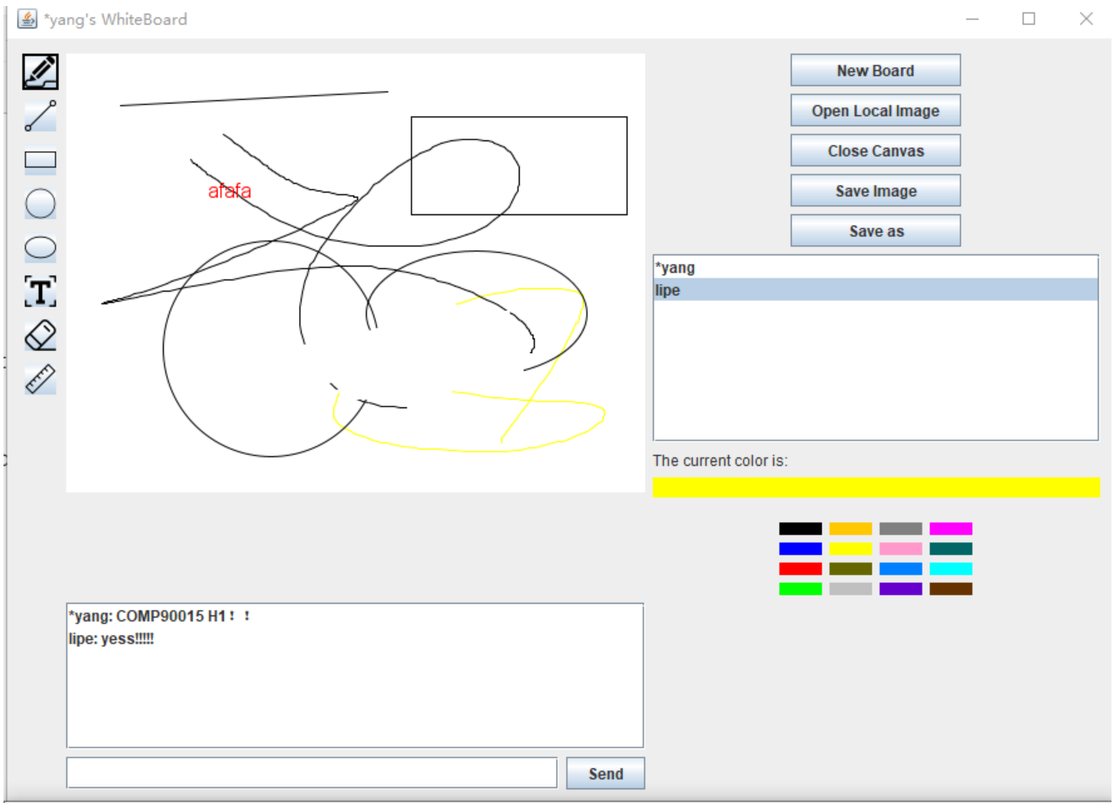
Do you want to remove lipe

The manager are also able to close the Canvas


Close Paint Board?

Do you want to close the Canvas

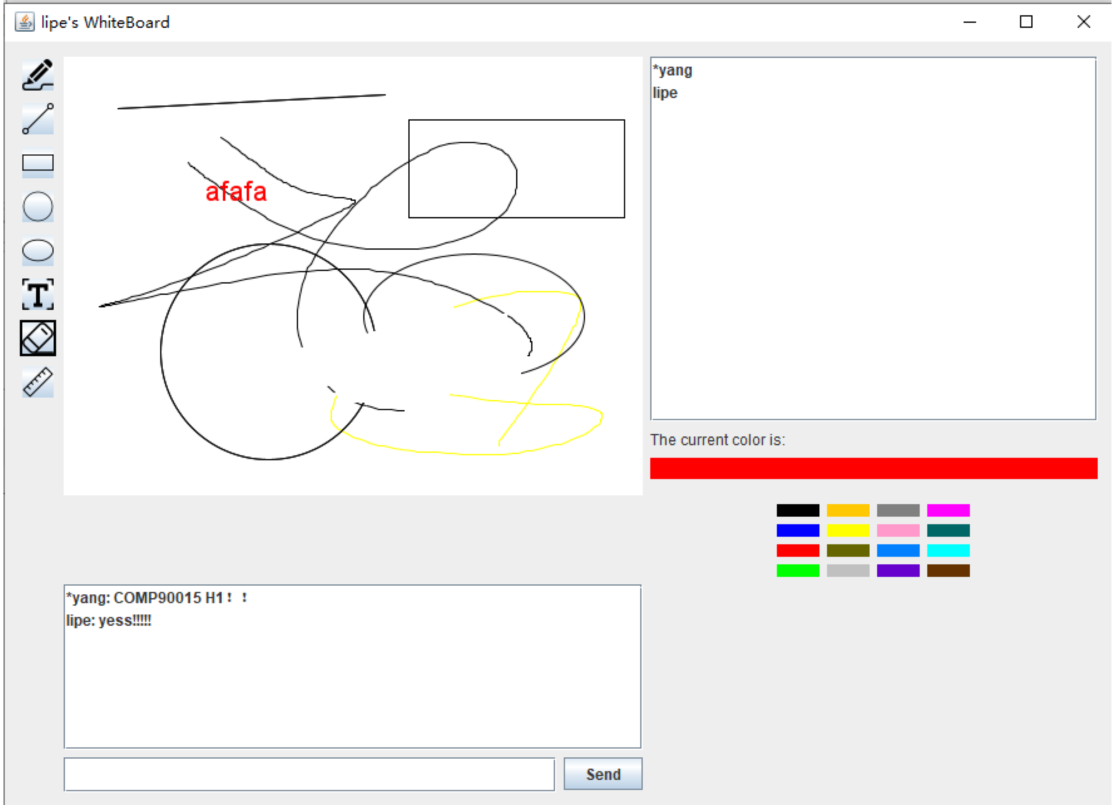
Different color and shapes will be available for all user and the canvas will be update with no delay.



*yang
lipe

The current color is:

*yang: COMP90015 H1！！
lipe: yess!!!!!

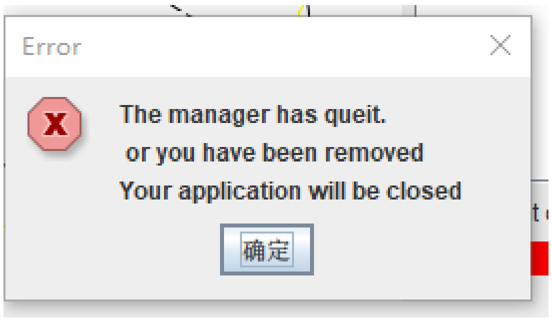


*yang
lipe

The current color is:

*yang: COMP90015 H1！！
lipe: yess!!!!!

And if the user has been kickout or canvas has been closed. There is a notification will be sent to them.



Error

The manager has queit.
 or you have been removed
Your application will be closed