Design Document

Team Number: 11
Team Leader: Zhuo Chen
Team Members: Peng Liang, Wenting Shi, Yik Fei Wong, Hon Kwan Wu, Tianyi Zhang
Project Name: What Does the Board Say?

1. Purpose (1 point): Briefly explain the system you are designing.

We are designing a mobile application that combines the functionalities of synchronized whiteboard and voice chat together. Users can write on the whiteboard through mobile devices while chatting. The content as well as voice will be synchronized and displayed on other users' whiteboard. The application is target to small group conference, including 3-6 participants.

2. Non-Functional requirements (2 points): Explain the non-functional requirements and provide justifications for your choices.

(1) Smooth of processing whiteboard:

The whiteboard itself needs to provide a smooth working environment without delayed response. For example, when user manipulates tools such as eraser, pens and etc. the application need to respond immediately. When a new document is created, users don't need to wait a long time for the system to response. Additionally, all the functions the application provided need be used instinctively.

(2) Optimize delay time:

Since users working/meeting remotely want to receive others updates/responses immediately in order to decide what they should do next. As the designers, we must optimize the delay time of the whiteboard process as well as the voice function to satisfy our customers.

(3) Application stability for multi-users:

The server device is going to handle multiple users, therefore the stability for multi-users is important. The limited RAM, processors, and other resources of mobile devices will be considered while implementing the networking part of the application, making it more stable even if the number of users increases.

(4) Good design of User Interfaces:

Our GUI will have a simple and clear interface containing a main whiteboard and a toolbar with several function keys, which includes but not limits to the buttons: "settings", "Pencil", "Eraser", "Text", "Import Graphic", "Save & Load", "Clear", "Undo/Redo", "Voice", etc. Our goal is not only to provide the desired functionalities for our users, more importantly, to provide them easier way to access these functionalities and better user experiences for our application.

(5) Icon designs:

It is important to design good looking and easily understandable icons for users. It will not only make new users to start using the application more easily and quickly, but also enhance their experience during remote working process.

(6) Smooth loading-screen with tips

During loading time, tips of using the whiteboard will be provided on screen and user can identify the tablet is functioning by a loading bar and percentage of how much the loading has finished.

3. Design Outline (4 points):

Initial part:

`Main`: A main interface letting the user create or join a whiteboard. Then program may go into two modules: host or client. Once the user choose to create a whiteboard, the program will run as a host, on the other hand, the program will run in client mode if the user choose to join a whiteboard.

Mainframe:

`Workspace`: The `Workspace` allows users to make changes on it. All the changes made on the Workspace, i.e. the whiteboard, will be synchronized to other users. Changes can be made by using `FunctionButtons`.

`FunctionButtons:` The `FunctionButtons` provide different functional buttons that allow users to make different changes on the `WorkSpace`. Functional buttons includes pencil, eraser, voice chat, etc.

Host module:

`Create`: Shows the IP address of the host and provides an invitation interface for the host user to invite other users.

`Server`: The Host will act as the server when using the whiteboard. All the changes made on the `Workspace` will be sent to the Host. The Host will be responsible to synchronize all the changes to the `Client`.

Client module:

`Join`: Input the IP address and password provided by the host in order to enter the `Workspace`.

`Client`: When any changes are made by a Client, the changes will be sent to the Server.

4. Design Issues (2 points):

a. Mention at least two issues (1 point for each one).

b. The issue title, the options you have to solve it (0.5 point), and justification of choice (0.5 point).

1. Peer to peer connection or client server connection?

   Solution: Client server architecture

   (1) Synchronization: our program requires a high quality of synchronization between users, client-server architecture will satisfy this since every client will download the exact same data that the server had collected. Whereas peer to peer network sometimes results in poor connection between two particular user, and eventually fails to synchronous with others.

   (2) Data management: as all data are stored in server, managing data becomes much easier. Furthermore, in case when crashes happen in programs it can be recovered easily and efficiently. While in peer computing, it has to take back up at every workstation.

   (3) Security: Server will define password for all users during startup time, password and IP will be organized. Server also can change user privilege to enhance whiteboard environment.

2. Fat server or thin server? Branch from client-server connection

   Solution: Fat server

   (1) Network: As we move on with client-server connection, our project coordinator mentioned a variable point that the server may not handle the amount of data stream need to send. We solve this by using Fat server. When many data gets received on server, the server will merge and compress the data before sending them. Instead of just simply receive and send them. This can lower the network pressure exchange with clock cycles combining data.

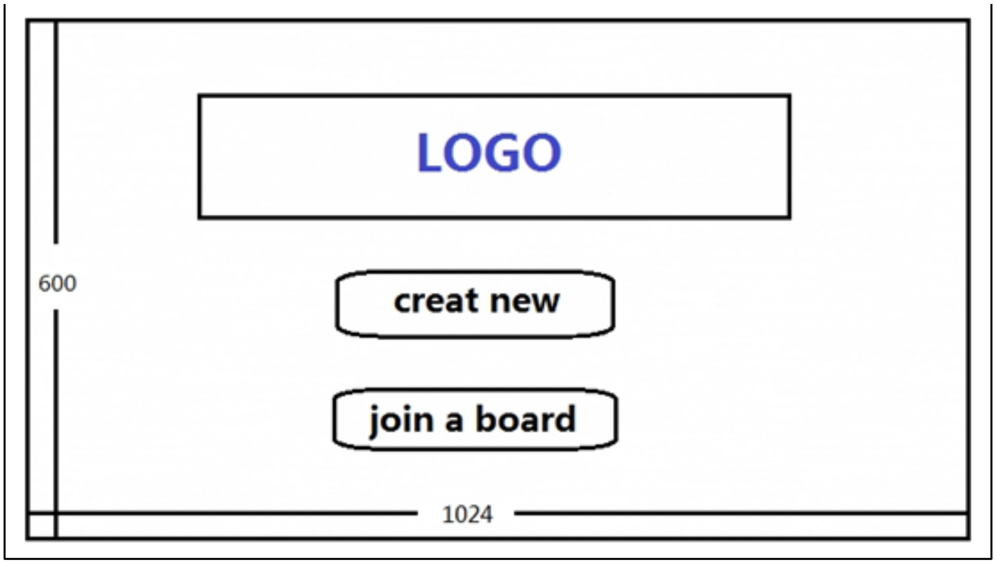
5. Design Details (6 Points):

a. Description of the classes and interactions between the classes (3 points).

Main.class:

This class is the entry class of the entire application. It will launch a startup frame with a “create button” and a “join button”. There is also an action listener in this class to decide whether the Create object will be created or the Join object will be created later.

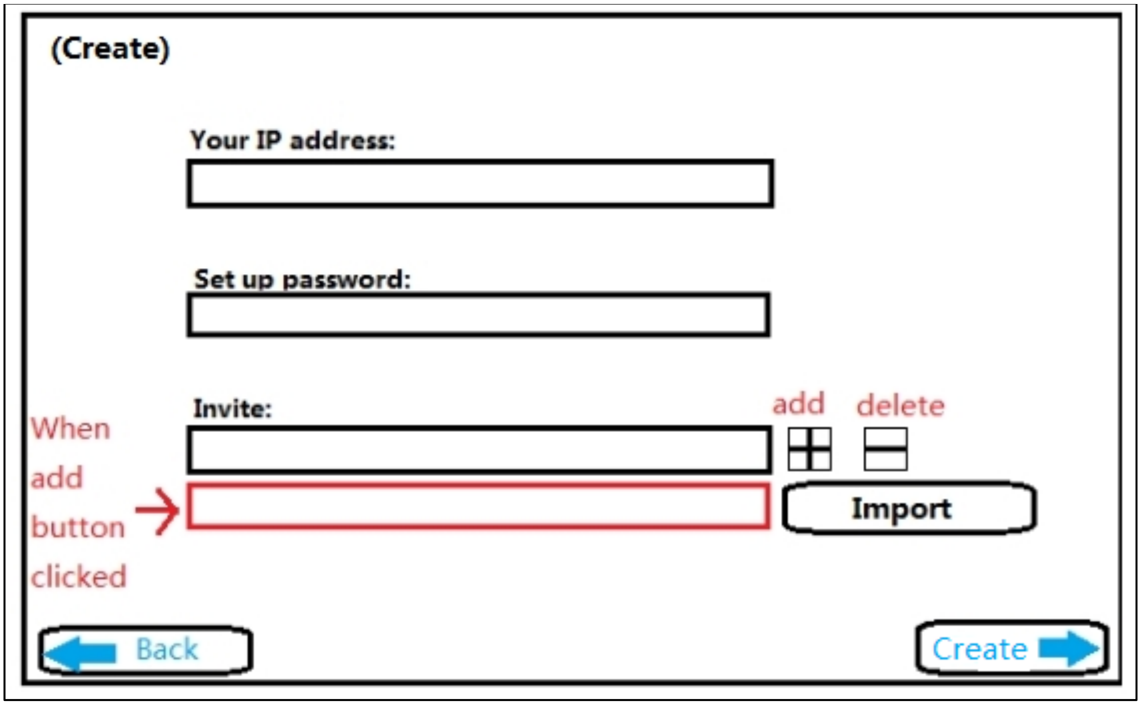
The frame of the startup is shown below:



Create.class:

This class will launch a create frame with the host's IP address, the set up password text field, and the invite text field. Also, there are buttons of "+ (Add)", "-(Delete)", "Create" or "Back". After filling all the email addresses of users the host wants to invite, the host can click the "Create" button. A Server object will then be created with the host IP address, password string and invite list as arguments.

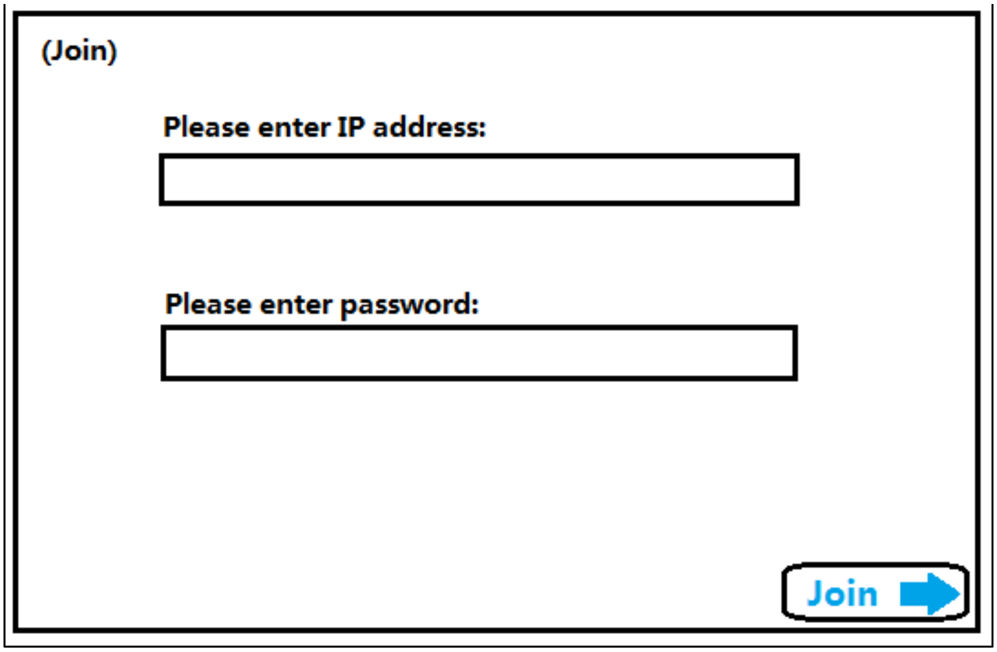
The frame of the create is shown below:



Join.class:

This class will launch a join frame with the enter IP text field, the enter password text field. Also, there is a button of “Join”. After filling the host IP address and password, the user can click the “Join” button. A client object will then be created with host IP address, password as arguments.

The frame of the create is shown below:



Server.class:

This backend class will communicate as a server with client classes. UDP (User Datagram Protocol) is the network transmission protocol we are going to use. This class will receive the data from each client, composing it and send the data to other clients. When any changes are made by any users in their front end Mainframe, the changes will be packaged and send to their backup, finally to the server. The server will then send the package to other clients and changes will be reflected on their whiteboard. Also, the server has the function to modify privileges of each client.

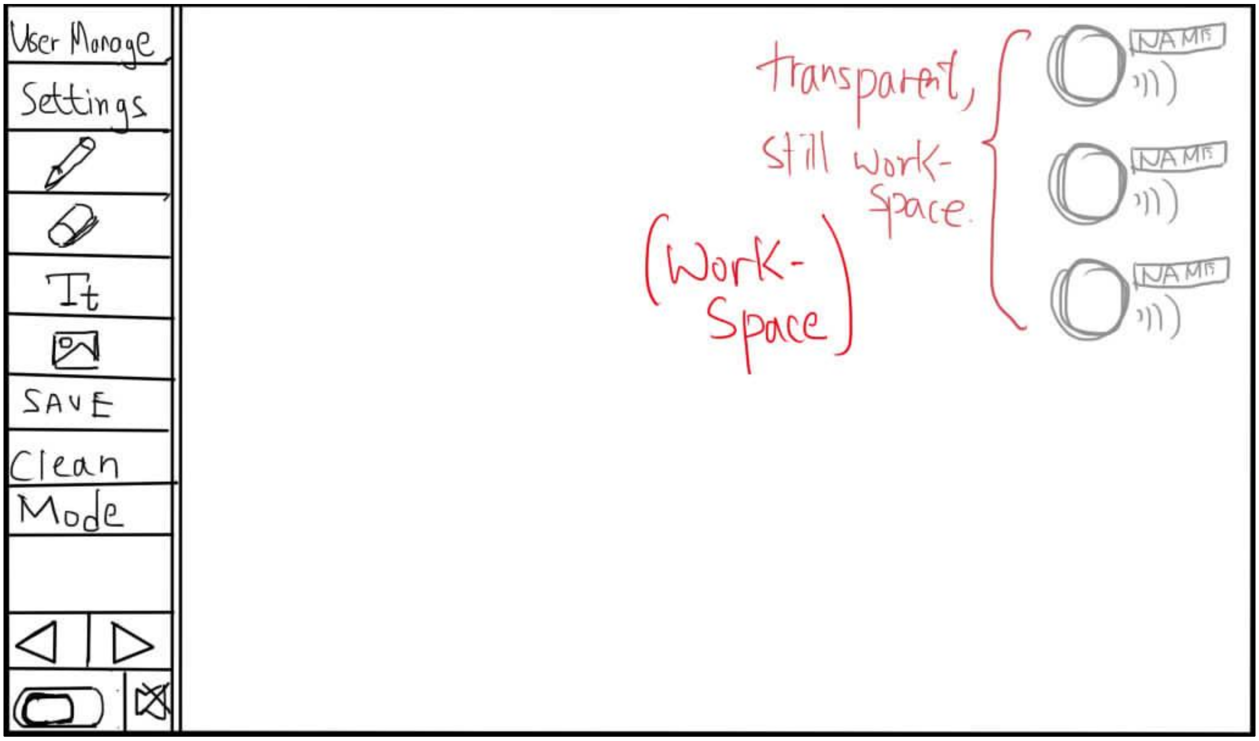
Client.class:

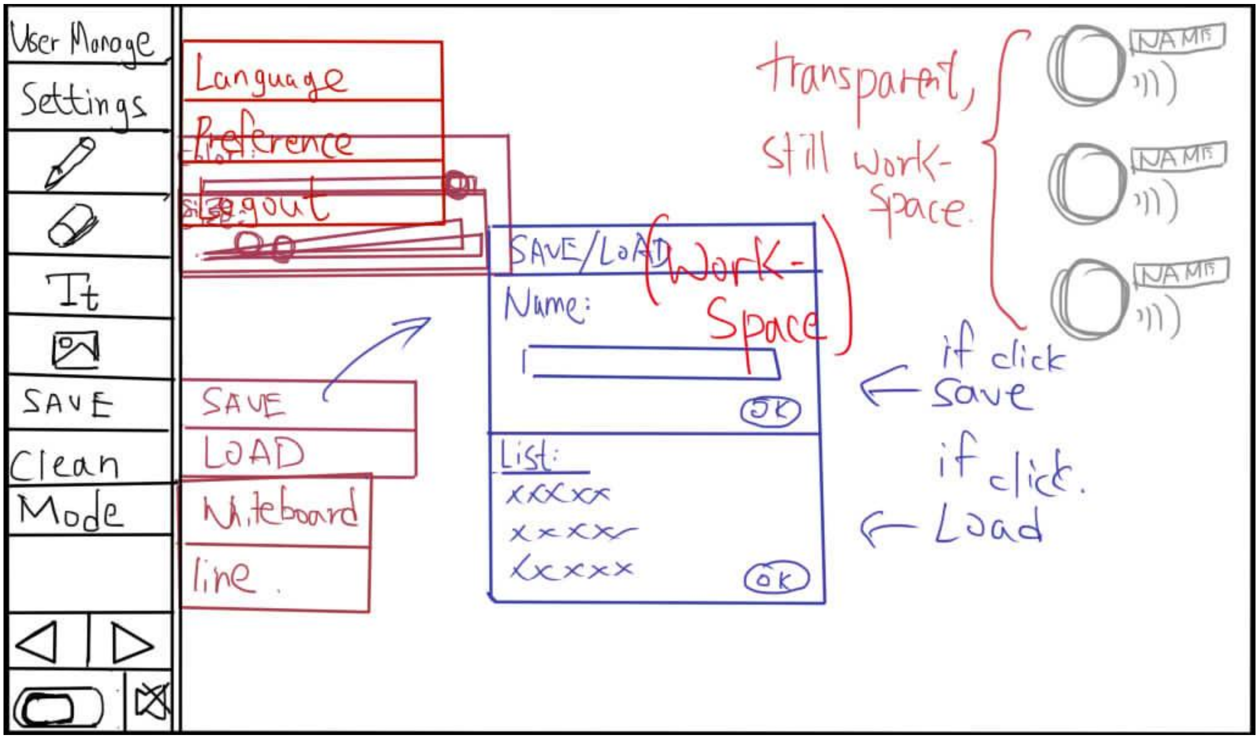
This backend class will communicate with the server class. We will use UDP Server-Client implementation. When changes are made, the changes will be packed into a data package and send the package to the server. It will also receive data package from the server, decompose the package and reflect any changes made by other clients on the Mainframe.

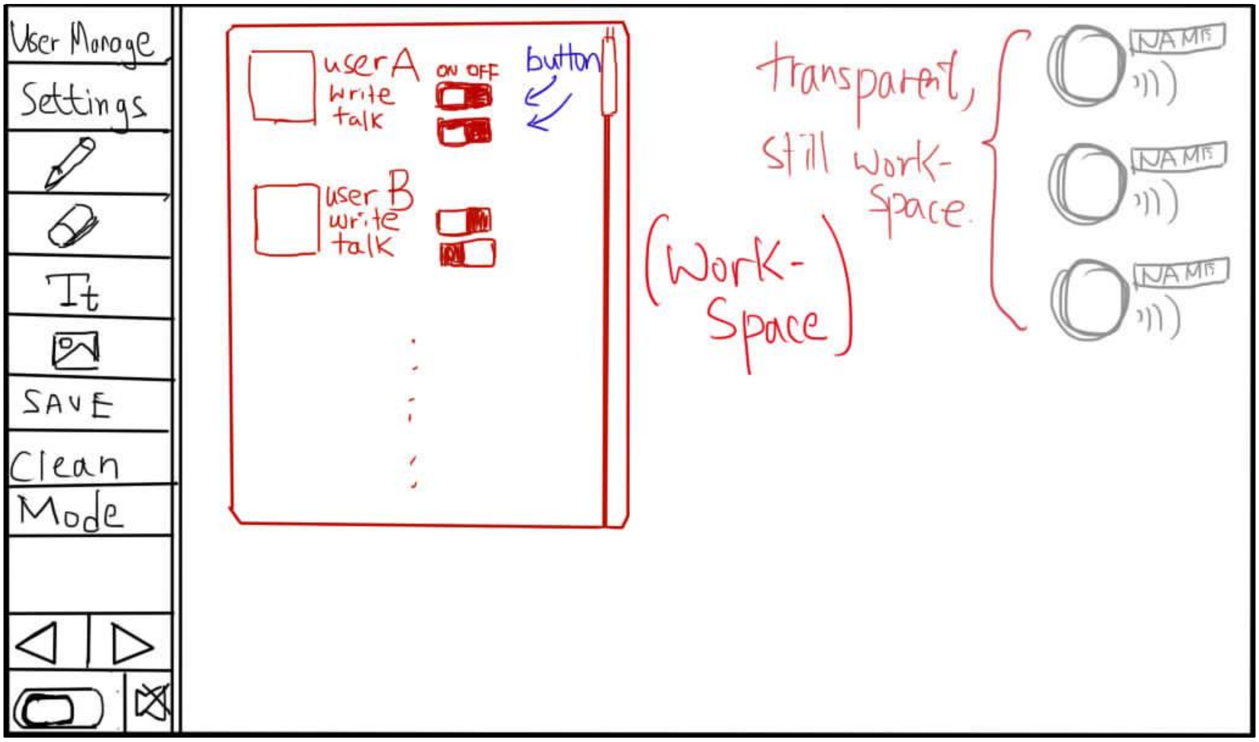
Mainframe.class:

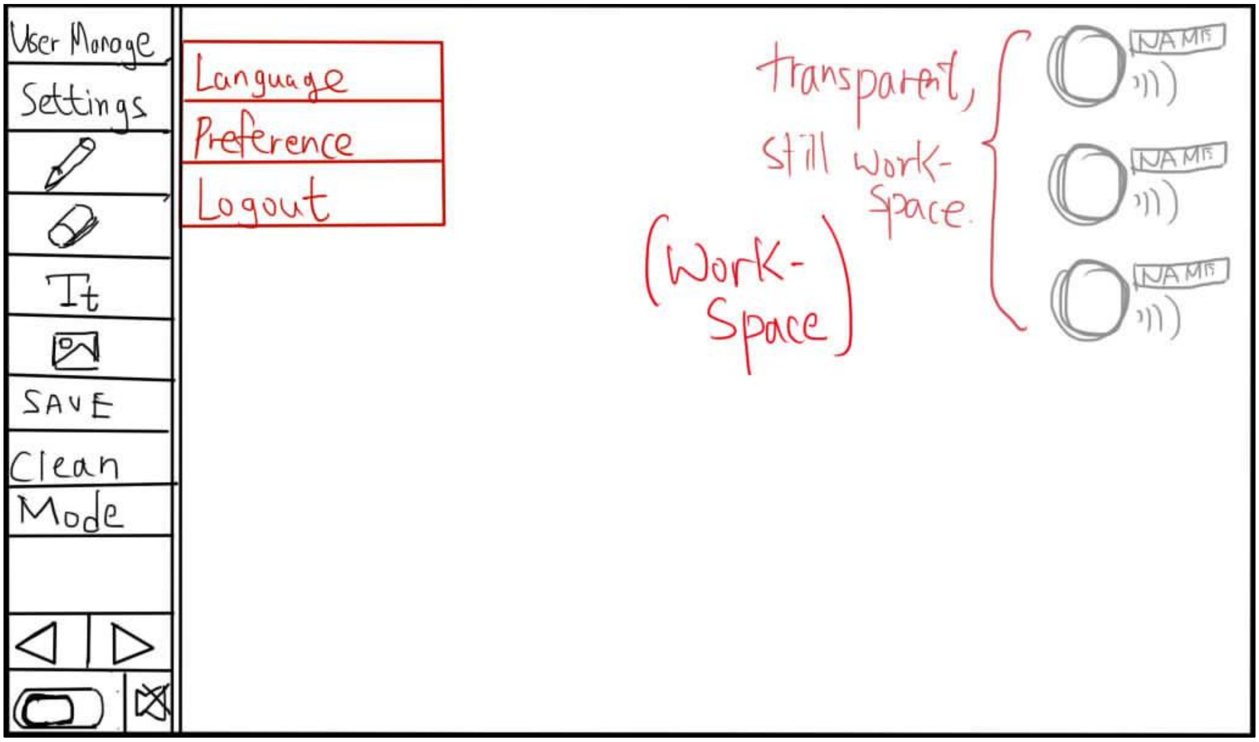
This frontend class will be created by the back end server class or client class, which will be a container of multiple classes such as “Workspace”, “”, etc. Every changes made in this frontend class will be packaged and send to the corresponding backend class, finally to others via network. Also, the Mainframe will receive data from its backend class from other users.

The Mainframe is shown below:

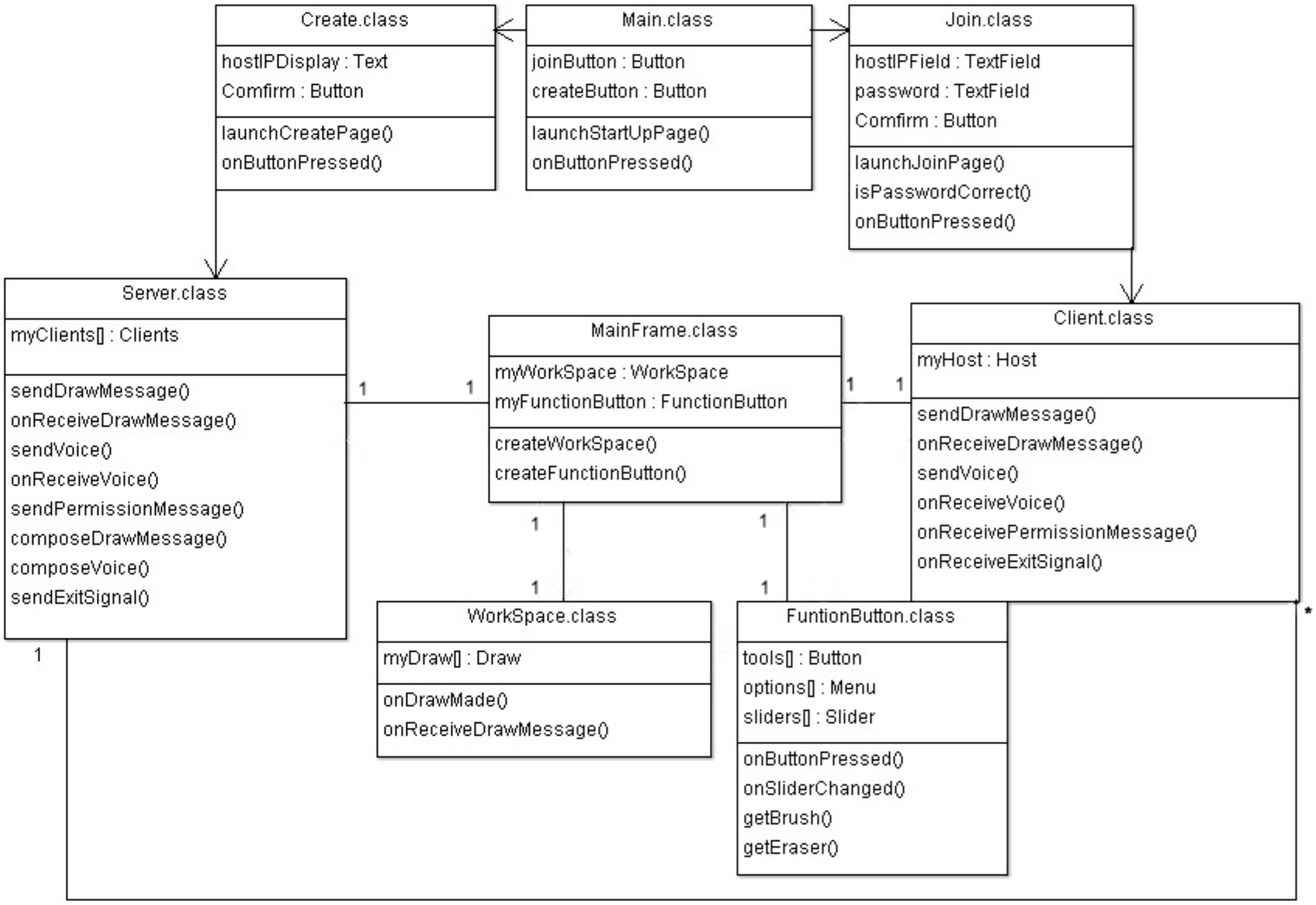





User Manage
Settings
It
SAVE
Clean
Mode
userA
write
talk
ON OFF
button
userB
write
talk
transparent,
still work-space.
(Work-Space)
NAME
NAME
NAME


User Manage
Settings
It
SAVE
Clean
Mode
Language
Preference
Logout
transparent,
still work-space.
(Work-Space)
NAME
NAME
NAME

b. UML diagram that illustrates the classes and interactions (3 points)


Create.class
hostIPDisplay : Text
Comfirm : Button
launchCreatePage()
onButtonPressed()
Main.class
joinButton : Button
createButton : Button
launchStartUpPage()
onButtonPressed()
Join.class
hostIPField : TextField
password : TextField
Comfirm : Button
launchJoinPage()
isPasswordCorrect()
onButtonPressed()
Server.class
myClients[] : Clients
sendDrawMessage()
onReceiveDrawMessage()
sendVoice()
onReceiveVoice()
sendPermissionMessage()
composeDrawMessage()
composeVoice()
sendExitSignal()
MainFrame.class
myWorkSpace : WorkSpace
myFunctionButton : FunctionButton
createWorkSpace()
createFunctionButton()
Client.class
myHost : Host
sendDrawMessage()
onReceiveDrawMessage()
sendVoice()
onReceiveVoice()
onReceivePermissionMessage()
onReceiveExitSignal()
WorkSpace.class
myDraw[] : Draw
onDrawMade()
onReceiveDrawMessage()
FuntionButton.class
tools[] : Button
options[] : Menu
sliders[] : Slider
onButtonPressed()
onSliderChanged()
getBrush()
getEraser()
1
1
1
1
1
1
1
1
1
*