Implementation of Distributed Shared Whiteboard Using Web Socket Protocol

Project Report submitted in partial fulfilment of the requirement for the degree of

Bachelor of Technology.

in

Computer Science & Engineering

Under the Supervision of

Dr. Rajni Mohana

By

Kush Aditya (111335)

to

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "**Implementation Of Distributed Shared Whiteboard Using Web socket Protocol**", submitted by **Kush Aditya(111335)** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date: 13-05-2015

Supervisor's Name: Dr.Rajni Mohana

Designation: Assistant Professor (Senior Grade)

Acknowledgement

Before I get into the project description, I would like to add a few heartfelt words for the people who were a part of this project. In numerous ways, people who gave unending support right from the stage project idea were conceived.

I sincerely expressed deep gratitude to Dr. Rajni Mohana faculty of Computer Science and Engineering. I thank you for the moral support, benevolent guidance, helpful suggestions and constant encouragement which make my project successful.

My sincere thanks also goes to my friends for their co-operation to achieve this goal.

Date: 13-05-2015 Name of the student: Kush Aditya(111335)

Table of Content

List of Figures

Abstract

Our proposal on Shared Whiteboard is the online platform bringing together the learners either local or distant and the teachers within one class through the help of web based system. It is referred as a web based system as it requires computer and internet connection. Shared Whiteboard has the features of chatting as well as video and audio conferencing between the teacher and the students. The teacher can register new students, set the agenda i.e. what he is going to teach, he will give the brief outline of topics to the students. Then he will start the presentation, he can also upload the resources like images, text file etc. After giving presentation the teacher will end the presentation by clicking on "end presentation" button, and then he will be redirected to the login page of the system. Similarly, the students can apply for the registration in the whiteboard system; he can attend the presentation and download the resources which are available on the whiteboard. First student will login into the system using username and password, after successful login he will be redirected to the page from where he will click on the "attend class" button and then he will be redirected to the shared whiteboard page. Both teacher and students can use different functionality of the shared whiteboard system like both the users can use different buttons to draw different types of figures or they can use pencil to write or draw anything. Similarly, whiteboard is provided for the sharing of ideas or any important messages among the teachers and students. As shared whiteboard is the representation of real classroom virtually, students are fully benefited as per in the real classroom. They can share the ideas, knowledge, and subject matters with the teacher and among the other students.

CHAPTER 1

Introduction to Shared Whiteboard

1.1 Concept of Shared Whiteboard

Just as the term virtual means a simulation of the real thing, Shared Whiteboard is a whiteboard shared via Internet, which provides a convenient communication environment for distance learners just like traditional face-to-face classroom. A Shared Whiteboard allows learners to attend a class from anywhere in the world and aims to provides a learning experience that is similar to a real classroom.

When we go to college we have a schedule of lectures, which we must attend. Student must arrive on time, and when he enters the classroom, he finds a teacher, fellow learners, a blackboard or whiteboard, LCD projector, optionally a television screen with videos. Likewise, a Shared Whiteboard session is a scheduled, online, teacher-led training session where teachers and learners interact together using computers linked to a network such as the Internet. A Shared Whiteboard enables to bring learners from around the world together online in highly interactive virtual classes while greatly reducing the travel, time, and expense of on-site teaching/training programs. It can be used as a solution for live delivery and interaction that addresses the entire process of creating and managing our teaching-learning process. It facilitates instructor and student in teaching-learning events, such as a seminar, online discussion or a live training for employees in company. As in traditional classroom, there are professor and fellow learners present with the student; we have many participants present in virtual classroom. They can talk with each other as in the traditional classroom via chat. Similarly presenter uses whiteboard, gives notes/resources, gives presentation as given in traditional one.

Thus, Shared Whiteboard can be visualized as a board on which a lecture or session is conducted using Internet. Now, that we have some idea about Shared Whiteboard, we will discuss some advantages that Shared Whiteboard offers over traditional Whiteboard.

1.2 Advantages of Shared Whiteboard

Following are some of the advantages of Shared Whiteboard over traditional Whiteboard model:

• Removal of geographical barriers (Anywhere learning)

A Shared Whiteboard allows learners and teachers to attend a single live training session from any place in the world, provided they have a computer and Internet connection.

• Sessions can be recorded

If learners miss a traditional classroom-based training session, they have very little opportunity to engage in the learning experience that took place.

A virtual classroom has a facility to record the session so learners or teachers can replay it afterwards. Teachers too get an opportunity to review their own or their colleagues' performance.

• Quicker to organize

Training can be organized more quickly than traditional classroom-based training. Classrooms and projectors do not need to be reserved, materials do not need to be distributed.

The sessions are easier to schedule or reschedule since attendees will not be traveling to the venue of the session.

• One to one communication

With the help of Shared Whiteboard, learners can talk to the teacher and to each other, and although this communication is not as rich in a traditional classroom, it still can help learners, since it is one to one.

Due to these advantages, concept of Shared Whiteboard is getting very popular. Since it allows learners to attend sessions from anywhere in world, it is very useful for distant learners and for peoples who cannot meet face to face because of lack of time.

Though it gives lots of advantages, it has some pitfalls also. Following section describes some limitations of Shared Whiteboard.

1.3 Limitations of Shared Whiteboard

Following are some of the limitations of Shared Whiteboard over Traditional Whiteboard

• *Teachers and students need to become familiar with the tools*:

Teachers and students are familiar with the workings of a traditional classroom, that is, they understand the concepts of hand raising, the whiteboard, assignments, and so forth. With a Shared Whiteboard, all attendees must become familiar with the way the Shared Whiteboard works before training starts.

• *Time dependency for Live Sessions:*

Attending Shared Whiteboard training is restricted to a certain scheduled time.

• *Infrastructure for the participants PC needs to be prepared:*

Shared Whiteboard sessions need to be scheduled, teachers need to be invited, and participants' PCs need to be prepared.

• *Technical Limitations:*

Technical issues such as bandwidth, speed of the connection or power failure may create problem while presentation is going on.

1.4 Different Users of Shared Whiteboard

There are different classes of users of Shared Whiteboard based on the roles that they play. When presenter uses a Shared Whiteboard, he has different work to do than participant. Depending on the user type, Shared Whiteboard takes different form for each user. These forms can be categorized depending on the user's role. These roles are as follows:

• Presenter (or Administrator)

• Participant

Facilities provided by presenter interface to Faculty:

• *Register new users*

Presenter as an administrator has to register new users that will be attending the session.

• *Create a session*

Presenter has to decide the session time, users that will be invited for the session. While creating a session, he can specify the time and users of the particular session.

• *Cancel a user registration*

If any registered user does not want to attend the session, presenter cancels his registration.

• *Conduct Online presentation*

As a presenter, he conducts the session for participants. During presentation he performs various activities in the classroom. He can load the presentation slide that will be displayed to participants.

• *Share Resources*

Presenter can add various resources to the session. It may be a file or just a simple web page link that participant can download at their end.

• *Conduct Poll*

He can create a poll for participants. Also he can chat with participants.

• Explain concepts using Whiteboard

He can use whiteboard to explain some of the topic, which may not be able to explain via presentations, or to solve any particular doubt asked by the participant.

Facilities provided by participant interface to students:

• *View online presentation*

When a participant joins the session, they can view the presentation, which are conducted by the presenter on the Shared Whiteboard. The presentation may include the PowerPoint presentation slides or it,may also include the snapshot of the whiteboard on which the presenter can explain the concepts to the participants.

• *Public/Private Chat*

The participants can have a conversation with fellow participants publicly or privately via the chat feature available in the interface, the chat allows the participant to send the instant messages to the participants who are also attending the session. Participants can also send private messages to any of the participant.

• *Download Resources*

The resources that are been shared by the presenter can be downloaded by the participants at their machine. The resources can be the files which may include course material, e-book's etc, or it may be also web links which presenter may want the participants should refer.

CHAPTER 2

Features of Shared Whiteboard

2.1 Common Features

We can extract the following common features of Shared Whiteboard.

• Directing Messaging among participants or between presenter and participants is possible using chat feature in Shared Whiteboard.

• Audio and Video can be used in session. Using audio, presenter can deliver voice-based lectures in a classroom.

• Shared whiteboard, are used by instructors and students to view images, presentations, or other application.

• Resource sharing is possible between presenter and participants. Presenter uploads files (notes) to the session. Participants at their end can download these resources.

• Presenter creates polls. These polls are useful for presenter to get feedback from participants.

• Virtual hands up, to indicate that participant has questions to ask.When participant does his hands up, presenter can chat with him.

2.2 Scope of the Proposed System

When we talk about scope, we are talking about developing a common understanding as to what is included in, or excluded from, a project i.e. proposed system. Scope can be defined in terms of deliverables, functionalities that the proposed system will perform.

Scope of our system- Shared Whiteboard is defined in terms of tools or features that the proposed system will have. The features of the proposed system can be listed as:

· Agenda:

Before starting the session, presenter has to create agenda for the session. Agenda briefly describes the topics for the discussion. Thus it gives an outline of the session to all participants

· Presentation display:

The presenter can show PowerPoint presentation in the session.For that he can load presentation file or any other file on the whiteboard. He is able navigate between different slides using appropriate buttons.

· Whiteboard:

Whiteboard is used to write while session is going on. Authority of using whiteboard will be with Presenter only. As a participant in the session, he can view whiteboard screens. Presenter can write on screen, or he can draw different geometrical shapes.

· Chat:

Using this feature presenter and participants can send short text messages to each other. Participants are allowed to send message to presenter using Hands-up facility. To give any message to participant,presenter can use chat tool.

· Resource Sharing:

Presenter uploads the resources that can be useful to the particular session. Participants can download these resources. Also, presenter can give any resource link on the web. Participants can see that web page separately at their end.

Real Time audio: Using this feature the presenter and participants can talk to each other using microphone.

CHAPTER 3

Approach

Our approach is to build a system that would facilitate conducting classes on Internet for participants from anywhere in world. Users can get many advantages by using Shared Whiteboard. He can save a lot of time by using the Shared Whiteboard to attend the session. Shared Whiteboard will have two types of users, as presenter and participant. When a user enters in the Shared Whiteboard, he will be asked whether he want to join the classroom as presenter or participant. Presenter is the one who conducts the session, and participant is the one who attends the session. Presenter when enters the classroom, he can also perform administrative functionalities. The users of Shared Whiteboard (presenter and participant) has different interface of the classroom, when they enter in the classroom. Depending on the user type, the features of the classroom will be different for:

· Presenter

· Participant

The presenter can create the session in the beginning & will conduct online presentation in the session. The participants can attend this session using the name of the session and view the presentation made by the presenter. In the session of the classroom, various functionalities that will be performed by the users are as follows

· **Agenda:**

The presenter creates the agenda in the beginning of the session. This agenda specifies an outline of the session.

The participants can only view the agenda in their main window. The agenda will give him the idea about the topics that will be covered in the session.

· **Presentation Area:**

In the presenter interface, the presentation area allows presenter to upload the presentation file. It also allows him to navigate between the slides. In the participant window, the presentation area will display the slide that the presenter is explaining.

· **Whiteboard:**

The presenter will be able to write, draw and highlight a particular area on the whiteboard. The white board will be consists of various components such as various shapes, lines, eraser, pointer, etc, using which he will be able explain the particular topic to the participants.The participants can view the whiteboard in their interface, but they will not be allowed to use it (they will not be allowed to make any changes to whiteboard). The changes made by the presenter on the whiteboard will be displayed to the participants.

· **Shared Resources & Web links:**

The presenter can share their resources using the load resource Facility that will be present in the presenter interface. The resources will be files of type (doc, rtf, pdf) or web links (URL).The participants can download these shared resources on their machines, also they can refer the given web links using their web browser

· **Chat:**

The presenter will use the chat functionality to send text messages to the participants. He can send a message to particular participant or to all participants.The participants will be able to send text messages to each other.But, he will not be able to send message to presenter directly. He can do it by using the hands-up facility.

· **Participant list:**

In the presenter interface, there will be the list of participants, which will consist of the names of the participants that will be attending the session. If the presenter has to eject any particular participant, then he can use the eject facility that will be available in presenter interface. In the participant interface, the list of participants will only show the names of the fellow participants that are attending the session.

CHAPTER 4

Software/Hardware Requirement

4.1 Hardware Requirement

- CPU.
- Free hard drive space.
- RAM.
- Internet connection.

4.2 Software Requirement

- Operating system (OS): Windows 2000/XP/Vista
- Software required:
    1. Internet Explorer 6.0/7.0
    2. Netbeans IDE 7.3.1, 7.4, 8.0, Java EE version
    3. Java Development Kit(JDK) version 7 or 8
    4. Glassfish Server open source edition 4

CHAPTER 5

Technologies Used

5.1 Client Side Scripting/Coding

- HTML
- CSS

5.2 Server Side Scripting/Coding

- Java
- JavaScript

5.3 Web Socket API

The Web Socket specification defines an API establishing "socket" connections between a web browser and a server. In plain words: There is an persistent connection between the client and the server and both parties can start sending data at any time.

5.4 JDBC API

The JDBC API allows you invoke database SQL commands from Java programming language methods. You can use the JDBC API in a servlet, JSP technology page, or an enterprise bean when you need to access the database. The JDBC API has two parts: an application-level interface that application components use to access a database and a service provider interface to attach a JDBC driver to the Java EE platform.

CHAPTER 6

Web Sockets

6.1 What are Web Sockets?

Web Socket is a protocol providing full-duplex communications channels over a single TCP connection. The Web Socket protocol was standardized by the IETF as RFC 6455 in 2011, and the Web Socket API in Web IDL is being standardized by the W3C.

Web Socket is designed to be implemented in web browsers and web servers, but it can be used by any client or server application. The Web Socket Protocol is an independent TCP-based protocol. Its only relationship to HTTP is that its handshake is interpreted by HTTP servers as an Upgrade request. The Web Socket protocol makes more interaction between a browser and a web site possible, facilitating live content and the creation of real-time games. This is made possible by providing a standardized way for the server to send content to the browser without being solicited by the client, and allowing for messages to be passed back and forth while keeping the connection open. In this way a two-way (bi-directional) ongoing conversation can take place between a browser and the server. A similar effect has been achieved in non-standardized ways using stop-gap technologies such as Comet.

In addition, the communications are done over TCP port number 80, which is of benefit for those environments which block non-web Internet connections using a firewall. [2]

6.2 Uses of Web sockets

6.2.1 Social Feeds

One of the benefits of social apps is about knowing what all your friends are doing when they do it. Sure, it's a little creepy, but we all love it. You don't want to wait minutes to find out a family member won a pie-baking contest or a friend has become engaged. You're online, so your feed should update in real time. [6]

6.2.2 Multiplayer Games

The Web is quickly coming into its own as a gaming platform. Without having to rely on plug-ins, Web developers are now able to implement and experiment with high-performance gaming in the browser. Whether you're dealing with DOM elements, CSS animations, HTML5 canvas or you're experimenting with WebGL, efficient interaction between players is crucial. [6]

6.2.3 Collaborative Editing

We are living in the age of distributed development teams. Working on a copy of a document used to suffice, but then you had to figure out a way to merge all the edited copies together. With a collaborative solution like Web Sockets, we can work on the same document and skip all the merges. It's easy to see who is editing what and if you're working on the same portion of a document as someone else. [6]

6.2.4 Multimedia Chat

While there's no substitute for holding a meeting in person, videoconferences are about as good as it gets when we can't get everybody in the same room. The videoconference route is plug-in heavy, though, and full of proprietary "goodness." so using Web Sockets with API's and the HTML5 audio and video elements is an obvious win. [6]

6.2.5 Online Education

School keeps getting more expensive, while the Internet keeps getting faster and cheaper. Online education can be a great way to learn, especially if you can interact with teachers and other students. Web Sockets is the natural choice, allowing for multimedia chat, text chat, and other perks like collaborative drawing on a digital communal chalkboard. [6]

6.3 How Web Sockets works

Web Sockets provide a persistent connection between a client and server that both parties can use to start sending data at any time. The client establishes a Web Socket connection through a process known as the Web Socket handshake. This process

starts with the client sending a regular HTTP request to the server. An Upgrade header is included in this request that informs the server that the client wishes to establish a Web Socket connection.

Once the handshake gets complete the initial HTTP connection is replaced by a Web Socket connection that uses the same underlying TCP/IP connection. At this point either party can start sending data.

With Web Sockets you can transfer as much data as you like without incurring the overhead associated with traditional HTTP requests. Data is transferred through a Web Socket as *messages*, each of which consists of one or more *frames* containing the data you are sending (the payload). In order to ensure the message can be properly reconstructed when it reaches the client each frame is prefixed with 4-12 bytes of data about the payload. Using this frame-based messaging system helps to reduce the amount of non-payload data that is transferred, leading to significant reductions in latency.[1]

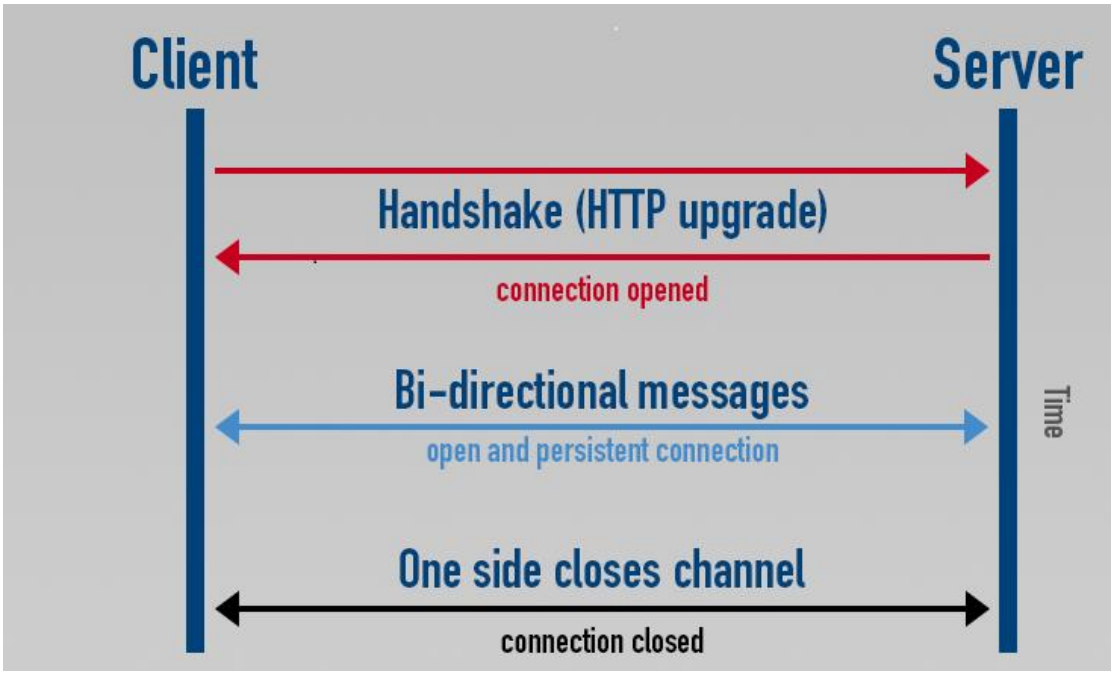

Figure 6.3: Mechanism of Handshaking

CHAPTER 7

Design

Software design is the activity where software requirements are analyzed in order to produce a description of the internal structure and organization of the system that will serve as the basis for its construction. There are two activities:

· *Software architectural design:* the top-level structure and organization of the system is described and various components are identified (how the system is decomposed and organized into components and must describe the interfaces between these components.

· *Software implementation design:* each component is sufficiently described to allow for its coding.The software design objectives can be listed as follows:

· To produce various models that can be analyzed and evaluated to determine if they will allow the various requirements to be fulfilled.

· To examine and evaluate various alternative solutions and tradeoffs,

· To plan the subsequent development activities.

Software system design results in the following products:

· A list of design goals derived from the nonfunctional requirements

· Software architecture

o Subsystem decomposition in terms of

Responsibilities.

Dependencies.

Mapping to hardware.

o Major policy decision such as

Control flow.

Access control.

Data storage.

Access control.

Boundary condition.

7.1 Uses-Case Diagram

A use case is a technique for capturing the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal.

Use case diagrams depict:

· *Use cases:* A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.

· *Actors:* An actor is a person, organization, or external system that plays a role in one or more interactions with your system. Actors are drawn as stick figures.

· *Associations:* Associations between actors and use cases are indicated in use case diagrams by solid lines. An association exists whenever an actor is involved with an interaction described by a use case. Associations are modeled as lines connecting use cases and actors to one another, with an optional arrowhead on one end of the line. The arrowhead is often used to indicate the direction of the initial invocation of the relationship or to indicate the primary actor within the use case.

· *System boundary boxes (optional):* We can draw a rectangle around the use cases, called the system boundary box, to indicate the scope of the system. Anything within the box represents functionality that is in scope and anything outside the box is not.System boundary boxes are rarely used.

· *Packages (optional):* Packages are UML constructs that enable us to organize model elements (such as use cases) into groups. Packages are depicted as file folders and can be used on any of the UMLdiagrams, including both use case diagrams and class diagrams. [9]

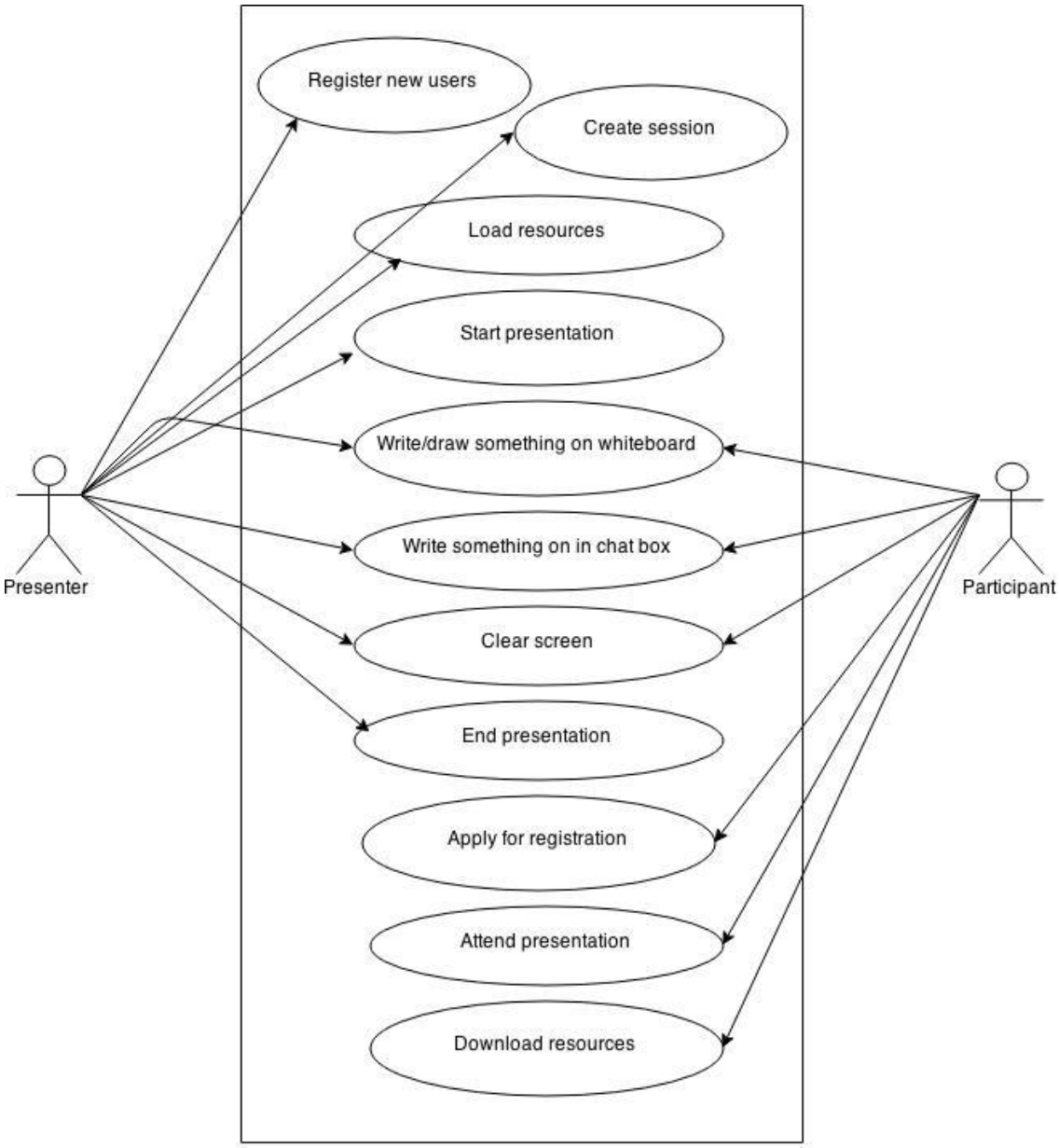

Figure 7.1: Use Case Diagram for Shared Whiteboard:-

As seen in use case diagram, in virtual classroom, there are two entities, Presenter and Participant, which interacts with the system. These two entities can be identified as actors in the System. Activities performed by the presenter:

· Registering new Users

· Create session

· Load Resources to the Session

· Start the Presentation

· Write/Draw on whiteboard

· Write in chat box

· Clear Screen.

· End the presentation

The functionalities of presenter are identified as use-cases in the system.

· Activities performed by the participant:

· Apply for the registration

· Attend the presentation

· Download Resources

Participant can apply for registration for the presentation, also he attends presentation. Therefore, use-cases identified for him in the system are as apply for registration and attend the presentation.

Use Case: Register new users

Actor: Student, Instructor

Purpose: To take the details of the new instructor or student and to register them.

Actor Actions	**System Response**
1. The user will click on the register Button and he will fill the form.	
2. After form filling user will click on the submit button.	
	3. The new entries of the user will get loaded into the database.

Use Case: System Login

Actor: Student, Instructor, Admin

Purpose: Verify UserName and Password to enter the system

Actor Actions	**System Response**
1. The user selects the user type and writes username and password	
	2. The database is searched in order to verify user info.
	3. If verification is OK than go to the related page else notify user

Use Case: Clear Screen

Actor: Student, Instructor

Purpose: System allows users to clear their whiteboard area.

Actor Actions	**System Response**
1.User click on the clear screen Button.	**2.** The whiteboard area of the users will get cleared.

Use Case: Download Resources

Actor: Student

Purpose: If there exist available documents in the database student may download

Actor Actions	**System Response**
1. Student selects the link	

	2. System checks the database if there exists available materials then lists them
3. Student downloads the desired material	

Use Case: Start Presentation

Actor: Instructor

Purpose: Instructor can start presentation

Actor Actions	**System Response**
1. Instructor clicks on the start presentation button.	
	2. System takes the instructor to the main page i.e. the whiteboard page.
3. Instructor will start giving presentation.	
	4. System will start transferring data to the connected clients.

Use Case: Load resources

Actor: Instructor

Purpose: Instructor may put some supplementary material and homework on the whiteboard.

Actor Actions	**System Response**
1. Instructor selects the link	
	2.System shows a dialog box

3. Instructor specifies the files and uploads

4. System will automatically upload the document on the whiteboard.

Use Case: Start Presentation

Actor: Instructor

Purpose: The instructor starts the presentation

Actor Actions	**System Response**
1. Instructor clicks on the "start" button	
	2. System takes the instructor to the main page

Use Case: Draw/write on the Whiteboard

Actor: Instructor, Student

Purpose: The instructor will use the whiteboard during the lecture in order to make points more clear. If instructor gives permission to the student then student will also use whiteboard.

Actor Actions	**System Response**
1. Instructor selects drawing tool and selects other options like size, color.	

2 . Instructor draws something on the board

3. The events(e.g. mouse move, drawn points coordinates) that instructor has done recorded by the system and they are send to all users in a certain time interval.

4. When the packets receive to the students their whiteboards will reflect the changes to the student.

5. If a student raises hand the instructor sends a signal to the server and that student then can use whiteboard and the same events take place.

Use Case: End Presentation

Actor: Instructor

Purpose: Instructor can end the presentation.

Actor Actions	**System Response**
1. Instructor clicks on the end presentation button.	
	2. System takes the instructor to the login page.
3.Instructor will again login to start the Presentation.	

Use Case: Write Something in Chat Box

Actor: Student

Purpose: This tool will provide a real time discussion environment.

Actor Actions	**System Response**
1. Student writes something on the chat console and submits "send" button	
	2. Server takes the message and looks IP of all online users.
	3. System sends the message to all online users.

Use Case: Apply for registration.

Actor: Student

Purpose: The students will give all the details to register.

Actor Actions	**System Response**
1. The student will click on register button.	
	2. System will take the student to the page where all details have to be filled.
3. Student will fill all the entries and then Clicks on submit button.	**4.** System adds all the details to database.

Use Case: Attend Presentation

Actor: Student

Purpose: A student can attend the presentation.

Actor Actions	**System Response**

1. The student will login to the system	**2.**System will take the student to main page i.e. whiteboard page.

7.2 Class Diagram

A class diagram consists of a group of classes and interfaces reflecting important entities of the business domain of the system being modelled, and the relationships between these classes and interfaces. The classes and interfaces in the diagram represent the members of a family tree and the relationships between the classes are analogous to relationships between members in a family tree. Interestingly, classes in parent class (the grand patriarch or matriarch of the family, as the case may be) and related child classes under the parent classes. Similarly, a software application is comprised of classes and a diagram depicting the relationship between each of these classes would be the class diagram. Thus, a class diagram is a pictorial representation of the detailed system design. [4]

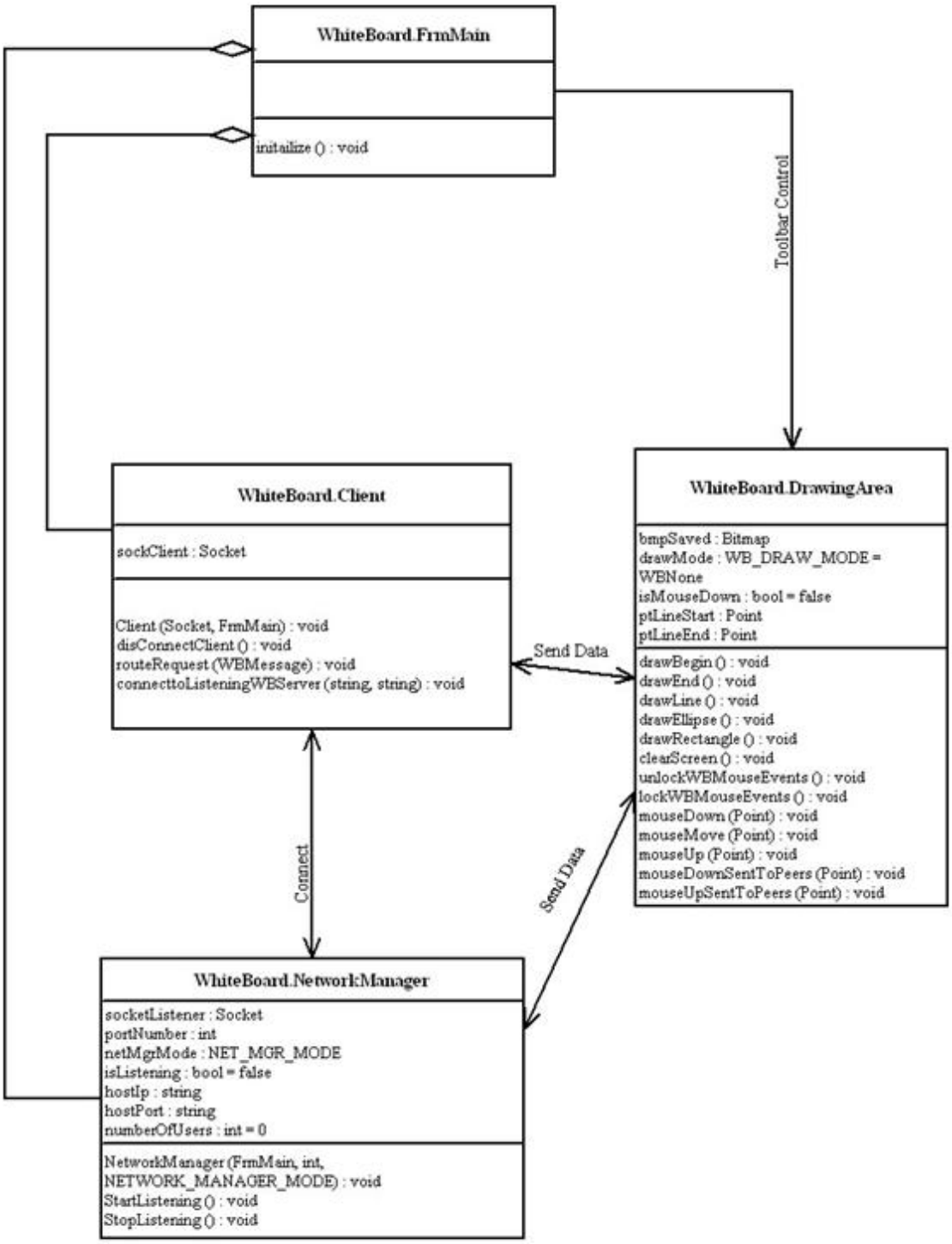

Figure 7.2.1: Class Diagram for Whiteboard Module

This component starts when the user enters the shared whiteboard system and finishes when the instructor or students exits the shared whiteboard system. As you can see from the class diagram above, this component contains 4 classes.

Class Name: FrmMain

This class contains the variables and methods used in the creation of the form (Buttons, labels, etc.)

Methods:

initialize (): This method initializes whiteboard.

Class Name: Network Manager

This class provides necessary methods for networking management. It controls the server-side operations and communicates with the clients. It also stores the number of clients connected to the server.

Attributes:

1) **socketListener: Socket**
2) **portNumber: int**
3) **netMgrMode: NET_MGR_MODE(*)**
4) **isListening: bool**
5) **hostIP: string**
6) **hostPort: string**
7) **numberOfUsers: int**

Methods:

1) **NetworkManager (FrmMain mForm, int portN, NET_MGR_MODE wbMode):** This is the constructor of the NetworkManager class.
2) **StartListening ():** This method starts listening to the port created by the NetworkManager.
3) **StopListening ():** This method forces the server to stop listening to a port. Therefore the network traffic managed by the server is broken and all the clients connected to the server are forced to disconnect.

Class Name: Client

This class contains methods used for the client-side operations of the whiteboard.

Attributes:

1) **sockClient: Socket**

Methods:

1) **Client (Socket socketClient, FrmMain ClientForm):** This is the constructor of the Client class. socketClient is the socket and ClientForm is the form created for the use of the client.
2) **ConnecttoListeningServer (string HostIP, string HostPort):** This method connects the client to the server listening to port HostPort and having the IP address HostIP.
3) **disConnectClient ():** This method disconnects the client that is connected to the server.
4) **routeRequest (WBMessage(**)):** This method maintains the singleton property of the whiteboard that is when a user is drawing on the whiteboard, no other users can access to the drawing methods of the whiteboard. This method calls the method lockWBMouseEvents () of DrawingArea class to ensure that the whiteboards of all the other users is locked when a user is drawing on the whiteboard.

Class Name: DrawingArea

This class creates a bitmap image to store the drawings. This image is overridden each a user draws an image.

Attributes:

1) **bmpSaved: Bitmap**
2) **drawMode: WB_DRAW_MODE(***)**
3) **isMouseDown: bool**
4) **ptLineStart: Point**
5) **ptLineEnd: Point**

Methods:

1) **drawBegin ():** This method sends the information that a user started to draw something on the whiteboard, to the network manager.
2) **drawEnd ():** This method sends the information that a user finished drawing something on the whiteboard, to the network manager.
3) **drawLine ():** This method is called when the user pressed DrawLine button on the toolbar of the whiteboard. It tells the system that the user is or will be drawing a line.

4) **drawEllipse ():**This method is called when the user pressed DrawEllipse button on the toolbar of the whiteboard. It tells the system that the user is or will be drawing an ellipse.
5) **DrawRectangle ():** This method is called when the user pressed DrawRectangle button on the toolbar of the whiteboard. It tells the system that the user is or will be drawing a rectangle.
6) **clearScreen ():**This method is called when the user pressed clearScreen button on the toolbar of the whiteboard. It clears the screen of the whiteboard.
7) **unlockWBMouseEvents ():** This method is called to unlock the whiteboard so that the user can access the drawing functions of the whiteboard.
8) **lockWBMouseEvents ():** This method is called to lock the whiteboard so that the user cannot access the drawing functions of the whiteboard. This method together with the lockWBMouseEvents method maintains the singleton property of the whiteboard.
9) **mouseDown (Point pt):** This method stores the coordinates of the point when the left mouse button is pressed.
10) **mouseMove ():** This method is called to draw on the whiteboard when the left mouse button is pressed.
11) **mouseUp ():**This method stores the coordinates of the point when the left mouse button is released, that is it stores the end point of the drawing.
12) **mouseDownSentToPeers (Point pt):** This method sends the coordinates of the point when the left mouse button is pressed to the connected peers so that the start point of the drawing is identified by the other users.
13) **mouseUpSentToPeers (Point pt):** This method sends the coordinates of the point when the left mouse button is released to the connected peers so that the end point of the drawing is identified by the other users.

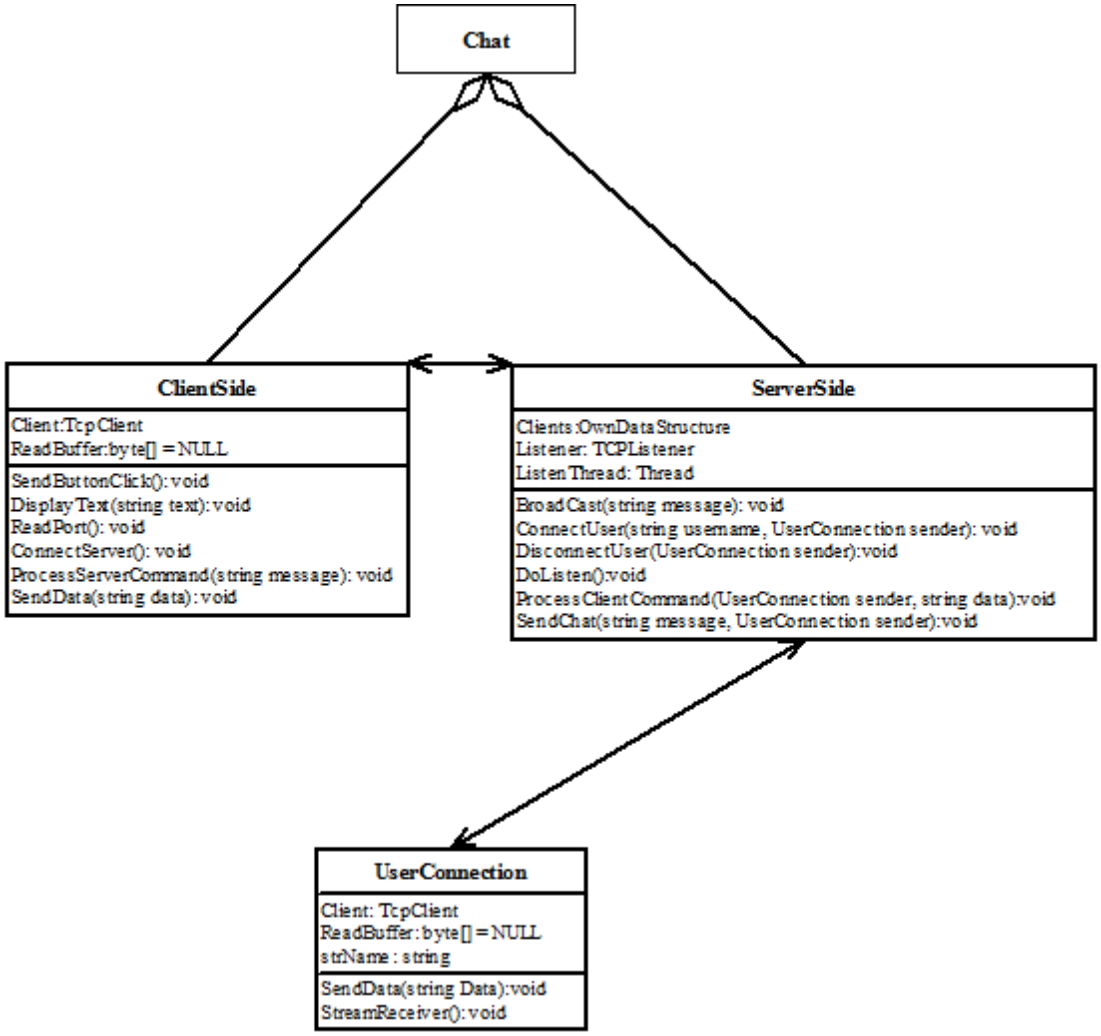

Figure 7.2.2: Chat Module

We have designed the "chat tool" with three different class structures. The first class is the "ClientSide" that will execute in clients and the second and third class "ServerSide" and "UserConnection" will execute in the server side.

Class Name: ClientSide

Attributes:

1) **Client:TcpClient**
2) **ReadBuffer: Byte[]**

Methods:

1) **ReadPort ():** This method will read the incoming data to the ReadBuffer array.

2) **ProcessServerCommand (string message):** This method will split the incoming message and will determine the type of the message that is a chat message or any information message like "x joined the class".
3) **DisplayText ():**This method will show the incoming or sent message in the clients chat console.
4) **SendData ():** This will send the written message to the server and server will handle the message.
5) **SendButtonClick ():** This is the button event takes place when user clicks the "send" button.
6) **ConnectServer ():** This is the method called when a new user connects to the classroom.

Class Name: ServerSide

Attributes:

1) **ListenThread: Thread**
2) **Clients: Our data structure**
3) **Listener: TCPListener**

Methods:

1) **Broadcast (string Message):** Sends the message to all attached clients
2) **DoListen ():** This method will listen the specified port for incoming messages from clients.
3) **ConnectUser(string username, UserConnection sender) :** This method will inform all the clients if a new user joins the classroom.
4) **DisconnectUser (string username, UserConnection sender):** This method will inform all the clients if a user exits the classroom.
5) **ProcessClientCommand (string username, UserConnection sender):** This method will split the incoming message of a client and will act according to that command
6) **SendChat (string message, UserConnection sender):** This method will send the message to the clients with sender information.

Class Name: UserConnection

In our ServerSide class a thread will work named ListenThread and if any connection is found a new UserConnection object will be created.

Attributes:

1) **Client: TCPClient**
2) **ReadBuffer: Byte[]**
3) **StrName: String**

Methods:

1) **StreamReceiver ():** This method will begin a asynchronous read from the stream
2) **SendData (string message):** This method will send the message to the user.

7.3 Data Flow Diagram

7.3.1 Shared Whiteboard System DFD Level 0:

As you can see in the following graph representation of our Shared whiteboard component, we have 3 inputs and 4 outputs. Besides that, we have one main process of our system, which control all of the tools of the system. All process works under this main process of our system. Our inputs of this component are student, instructor and video/audio streaming device. And outputs are student list, chat box, whiteboard, and audio. The important processes of this component are Whiteboard, Chat tool, and video/audio streaming system. All of them receive some data from users than all of them is distributed all of the members by this system.

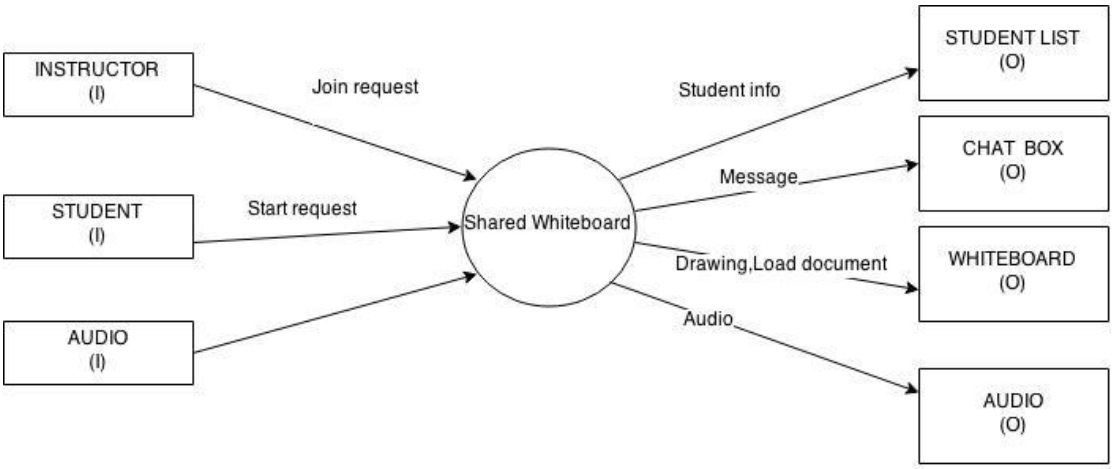

Figure 7.3.1

7.3.2 Shared Whiteboard System DFD Level 1:

In this system we have 2 main processes, which are whiteboard system, chat system, in the whiteboard instructor and student can use this tool .All of the users can use the chat tool. We also have 2 inputs (Instructor and student) and 2 output(whiteboard and chat box).

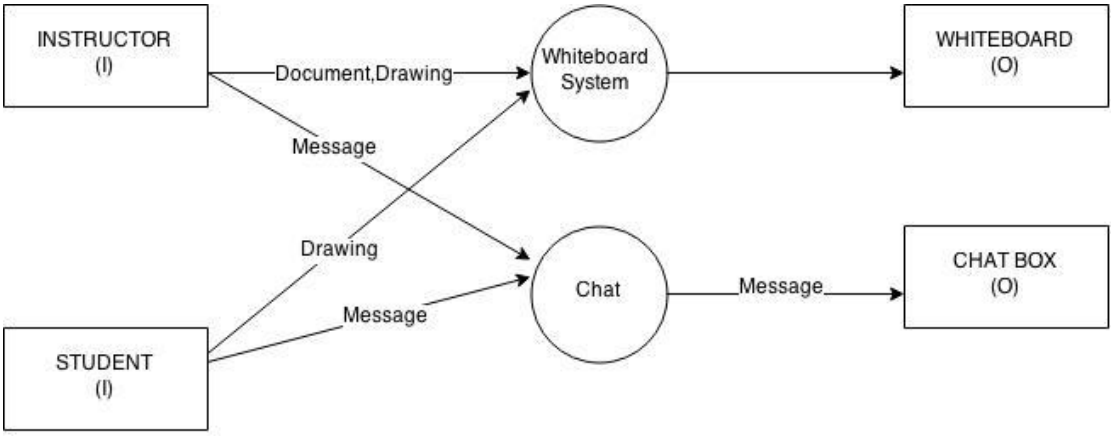

Figure7.3.2

7.3.3 Whiteboard DFD Level 2:

As you can see the following graphical representation of the whiteboard system, the instructor and student can use the whiteboard. All of the drawings and writing is saved in specific place then it is sent all of the users as an output. In the corner of the whiteboard there is a drawing toolbar, which makes the drawing easier. Moreover, instructor can also upload their documents, such as power point file, image or text document, on whiteboard.

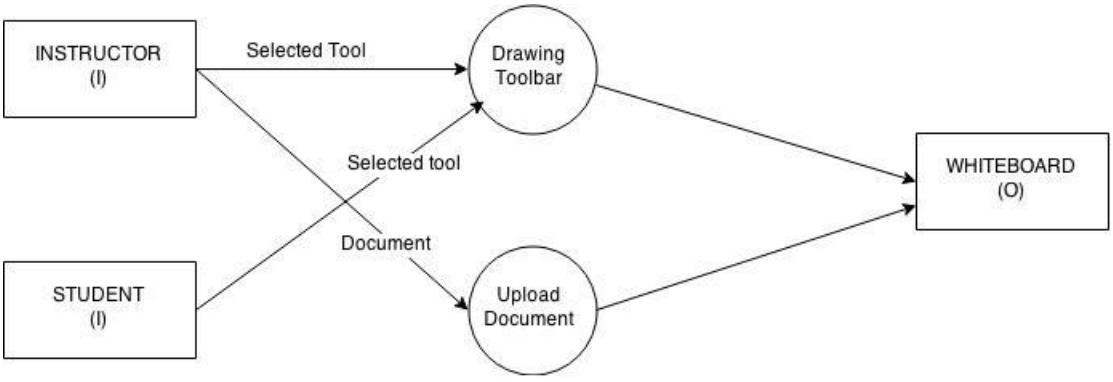

Figure 7.3.3

7.3.4 ChatTool DFD Level 2:

We have two main processes of this chat tool. These are general message and private message. Only the instructor uses the private message. But general message can be used by all of the users.

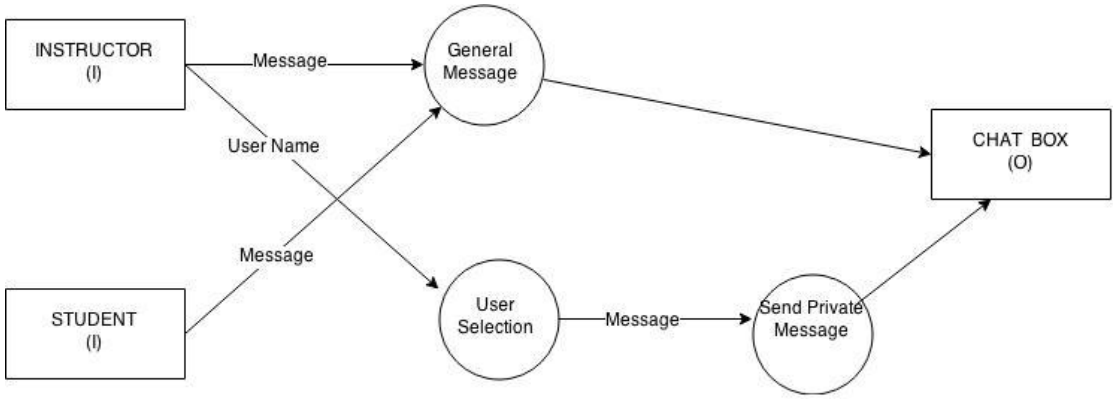

Figure 7.3.4

7.4 Sequence Diagram

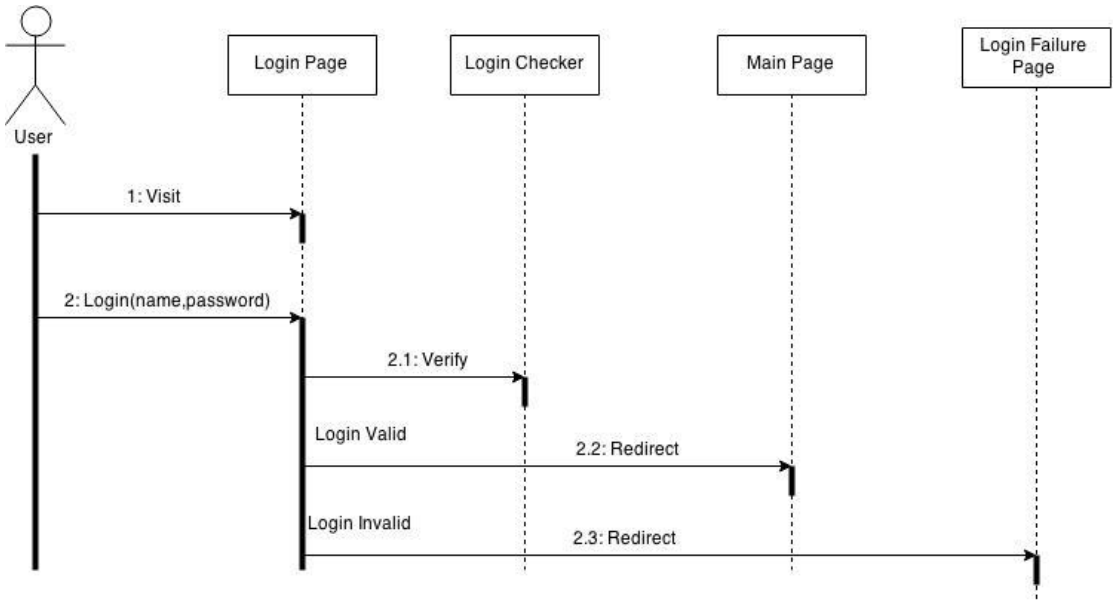

Figure 7.4.1: Login System

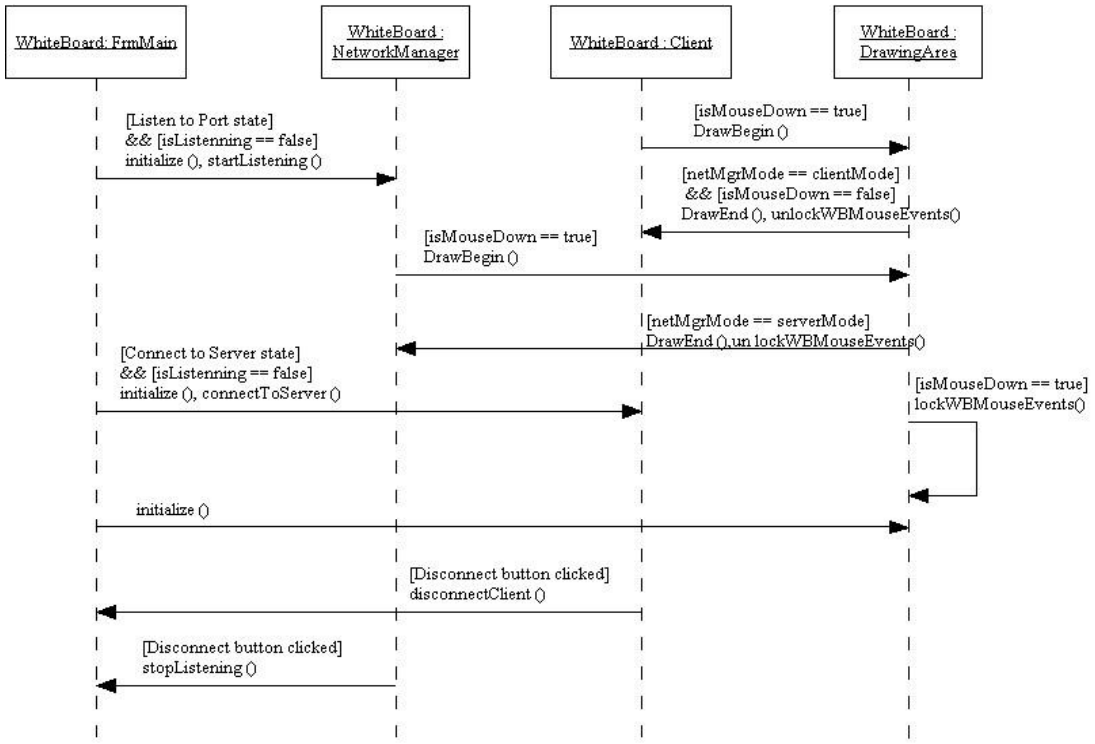

Figure 7.4.2: Whiteboard System

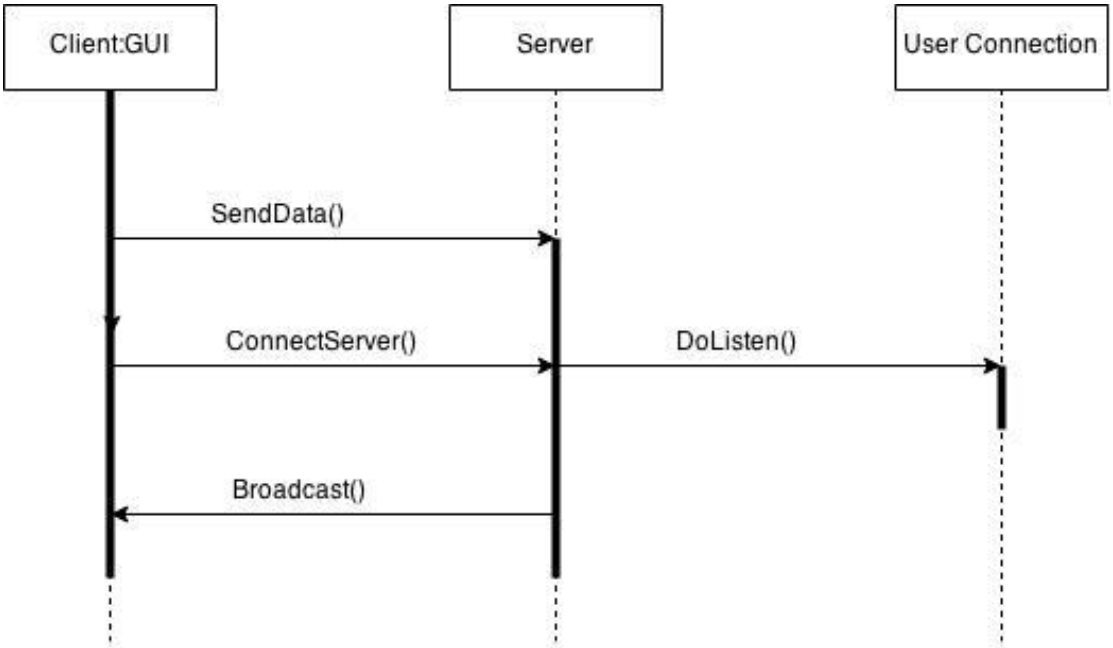

Figure 7.4.3: Chat System

CHAPTER 8

Implementation

8.1 White Board (GUI)

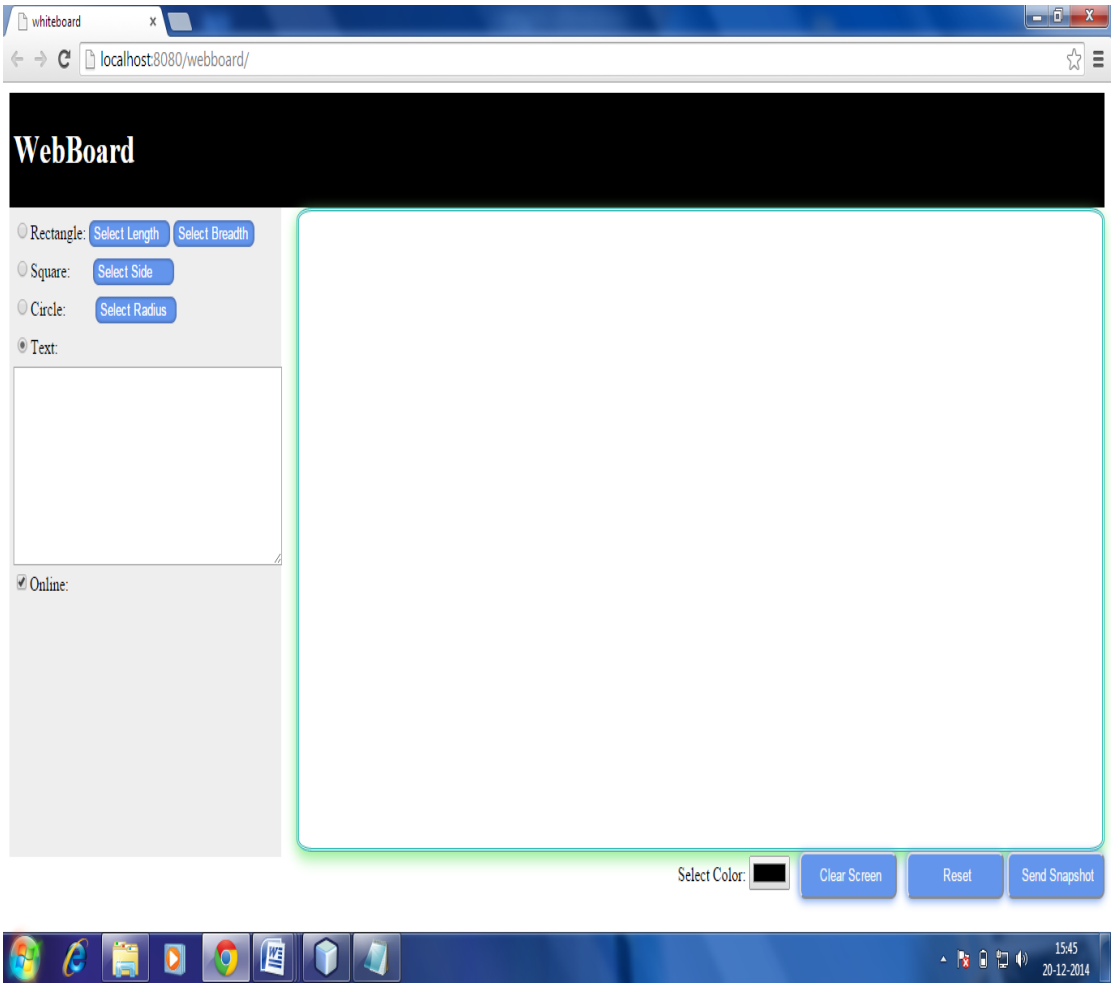

Figure: 8.1

This is the snapshot of the whiteboard system with two panels one on the left side, other at the bottom and the white area which is shown is the whiteboard area or drawing area on which we can draw. The left Panel contains radio buttons for rectangle, square, circle and text. It also has one check box named as online and drop down list for selecting dimensions of the figure. The functionality of these buttons is as follows:

1. **Rectangle:** - This radio button is used to draw rectangle of selected length and breadth on the whiteboard area. The figure will be drawn by clicking on the whiteboard area.
2. **Square:** - This button is used to draw square of selected side.
3. **Circle:** - This button is used to draw circles of selected radius.
4. Text: - This button is used to write any text on the whiteboard area .The text is first written in the text area then by clicking on the whiteboard it can be transferred to whiteboard.
5. **Online**: - This checkbox is used to transfer the content of the presenter whiteboard to the participant's whiteboard. If it is checked then the content of the presenter whiteboard will be transferred otherwise not.

On the bottom panel we have three buttons and one colour palette. The functionality of each button is:-

1. **Clear Screen**: - This button is used to clear the whiteboard area for further drawing/writing.
2. **Reset**: - Reset button is used to reset all the values to default values.
3. **Send Snapshot**: - This button is used to send the snapshot of the whiteboard area to all the connected clients.
4. **Select Colour:** - This palette is used to choose the colour.

8.2 Presenter and Participant (GUI)

Presenter's Graphical user Interface:-

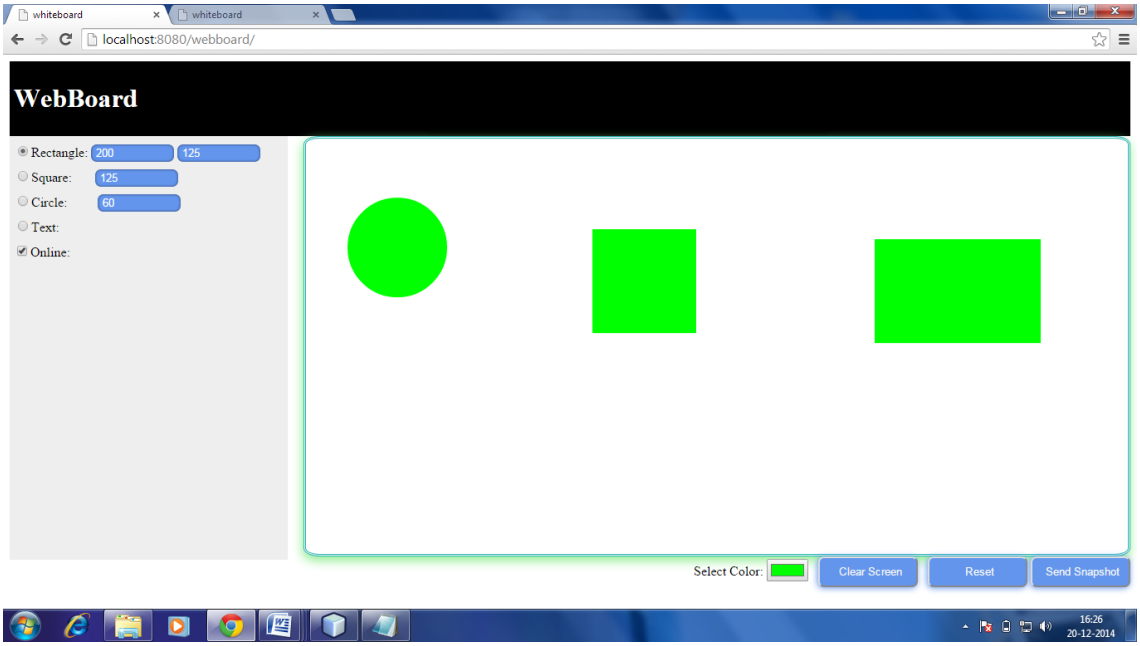

Figure: 8.2.1

Participant 1 Graphical user Interface:-

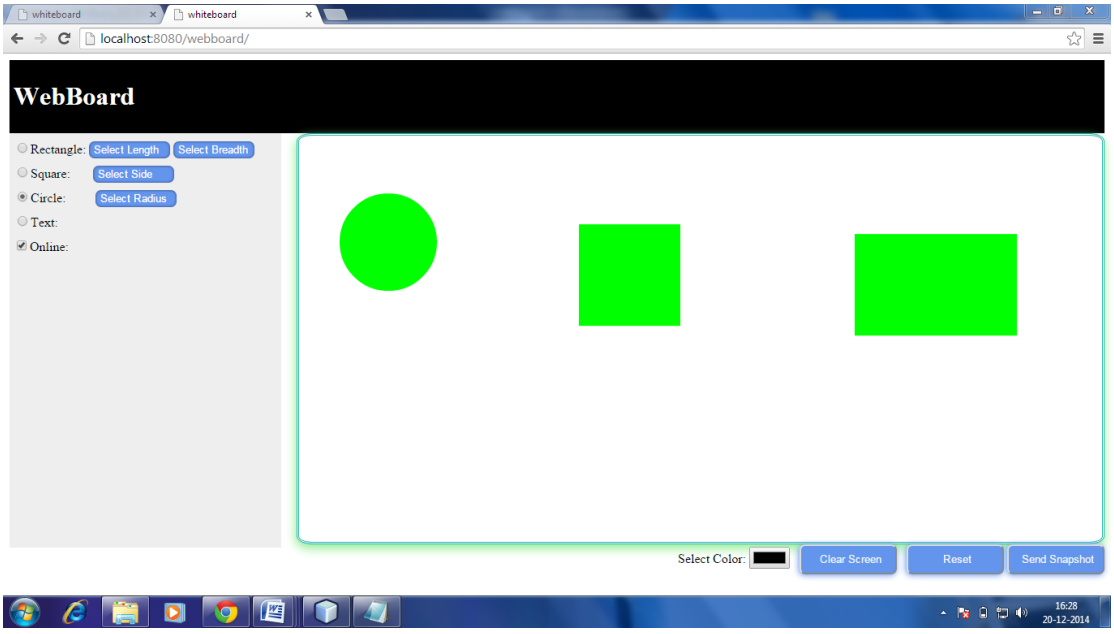

Figure: 8.2.2

Participant 2 Graphical User Interface:-

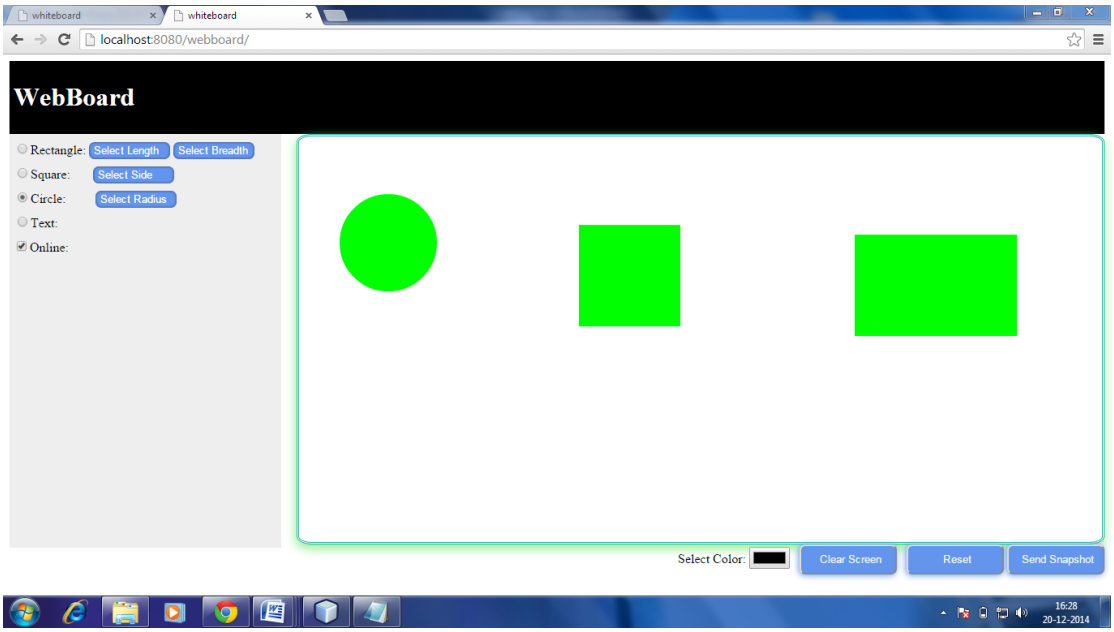

Figure: 8.2.3

Presenter's Graphical User Interface:-

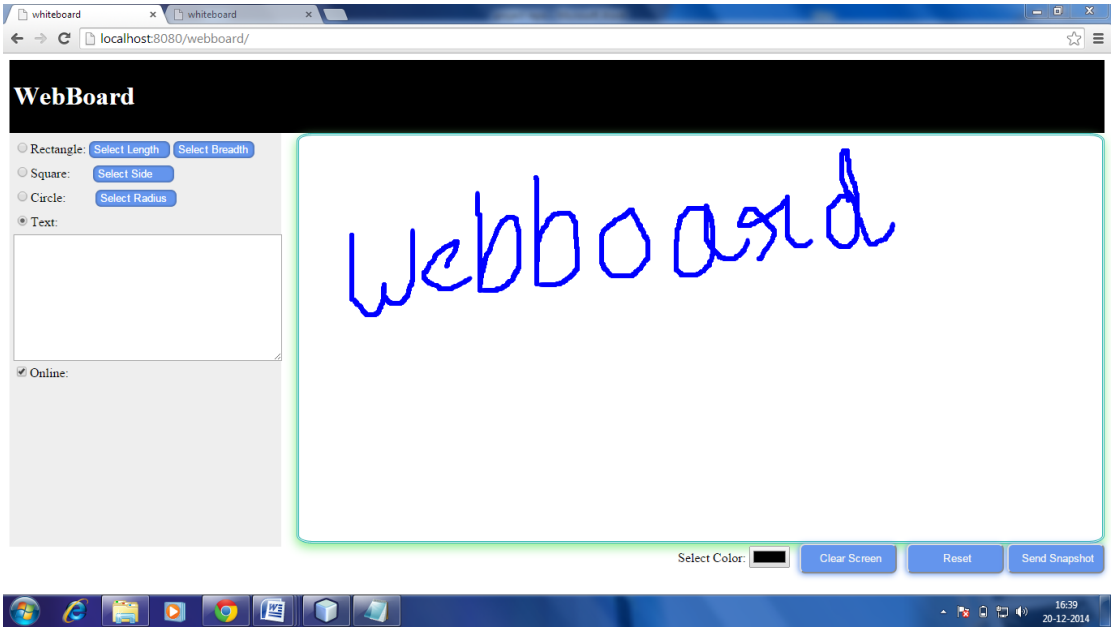

Figure: 8.2.4

Participant 1 Graphical User Interface:-

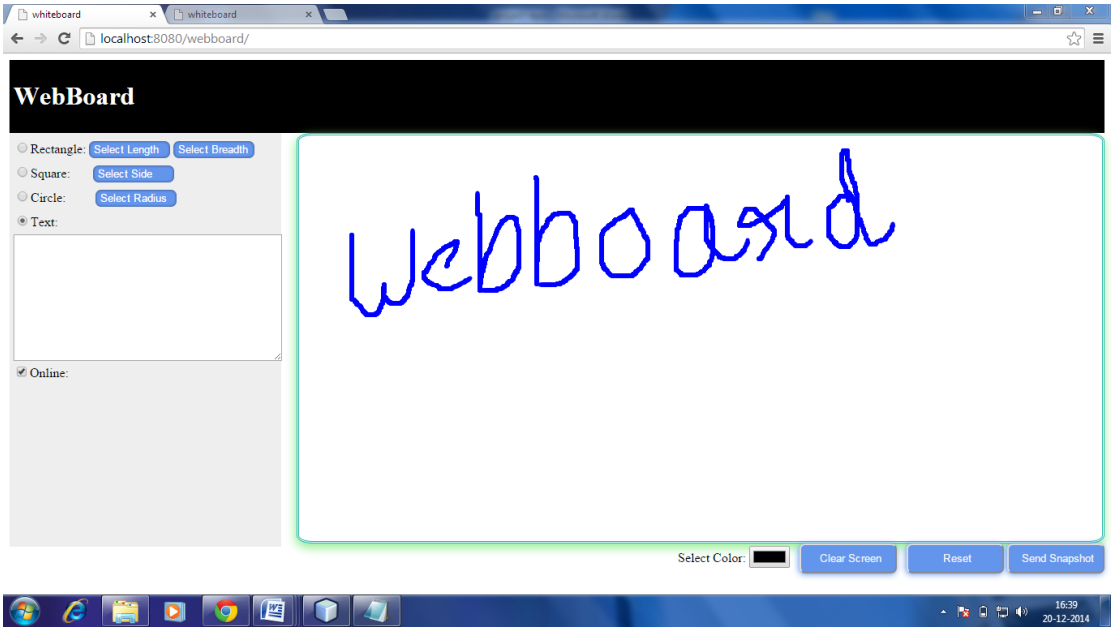

Figure: 8.2.5

Participant 2 Graphical User Interface:-

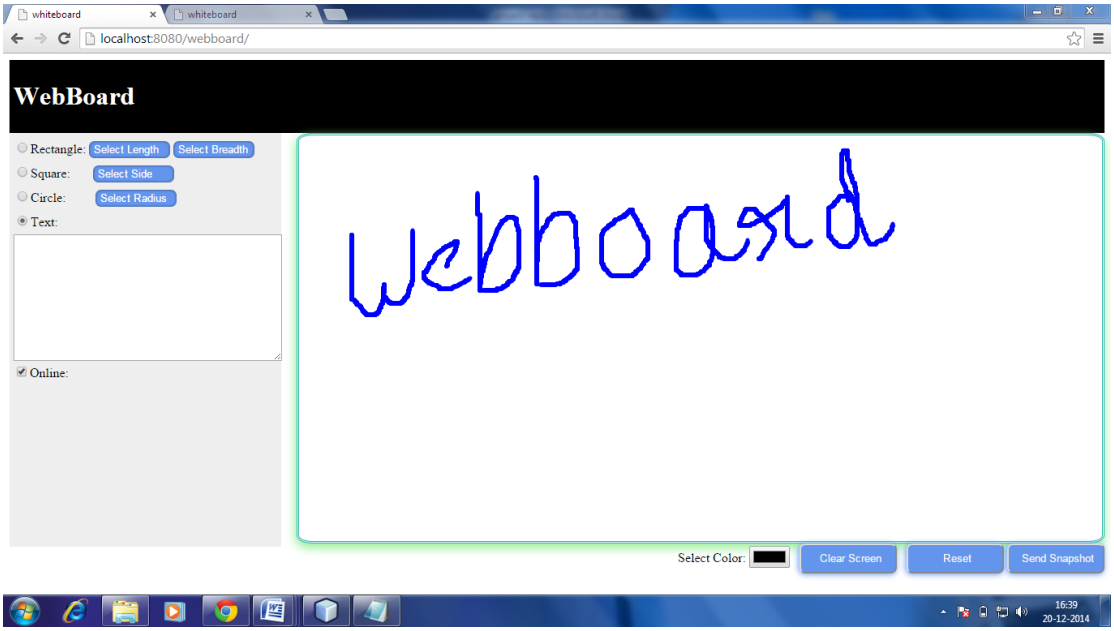

Figure: 8.2.6

As we can see above that we have one GUI of presenter or teacher and we have two GUI of participants or students connected to the server, so whatever the teacher is drawing or writing on his whiteboard area it is getting reflected to all the connected clients or students.

Presenter's Graphical User Interface:-

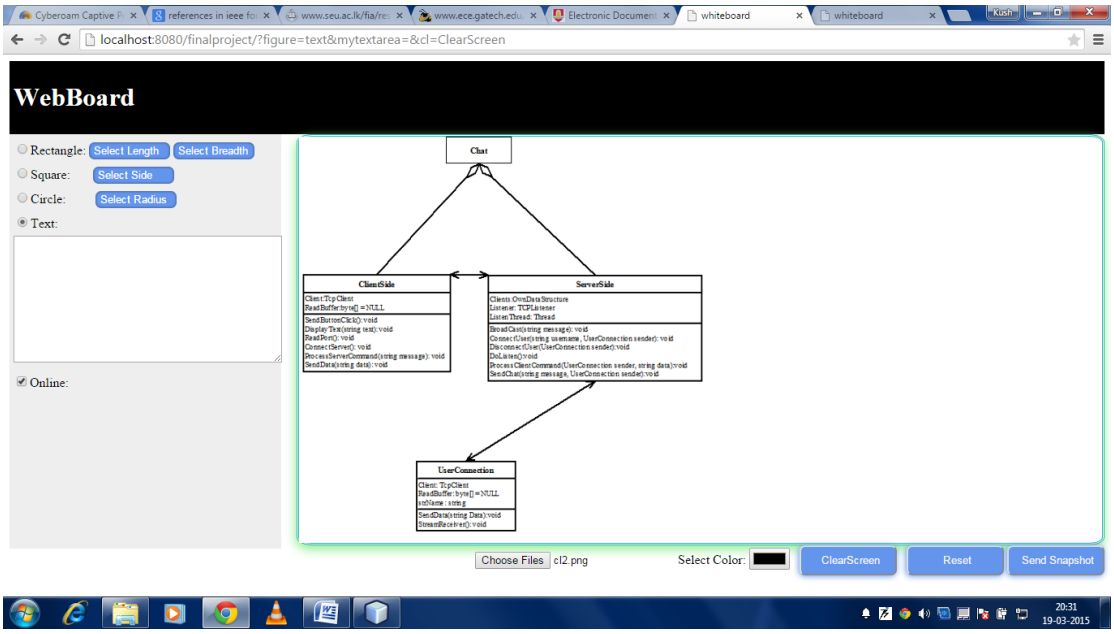

Figure: 8.2.7

Participant 1 Graphical User Interface:-

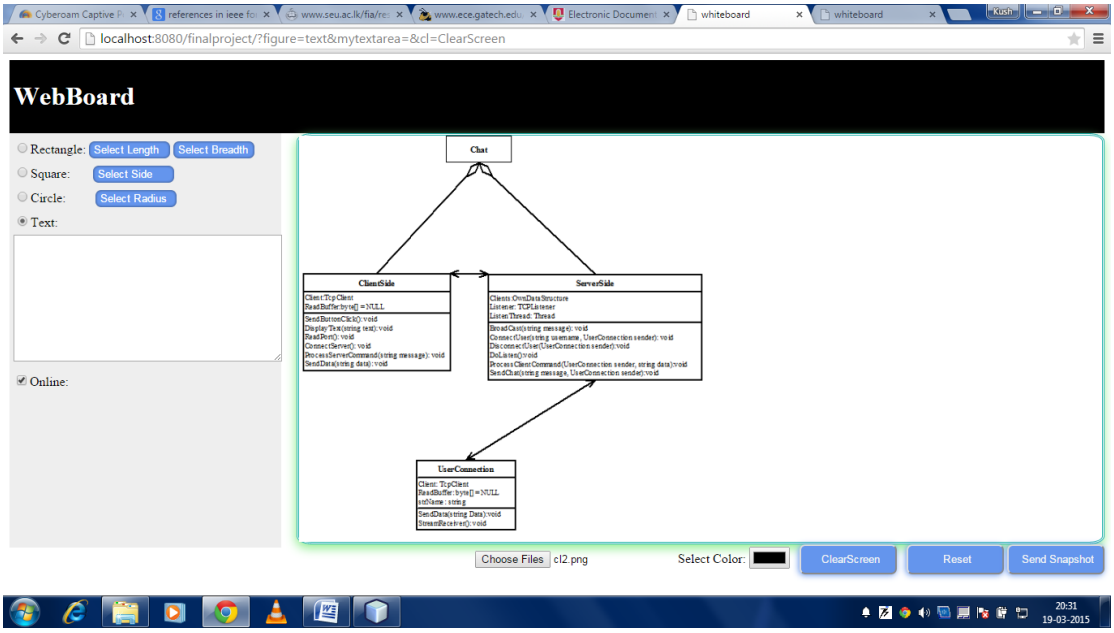

Figure: 8.2.8

Participant 2 Graphical User Interface:-

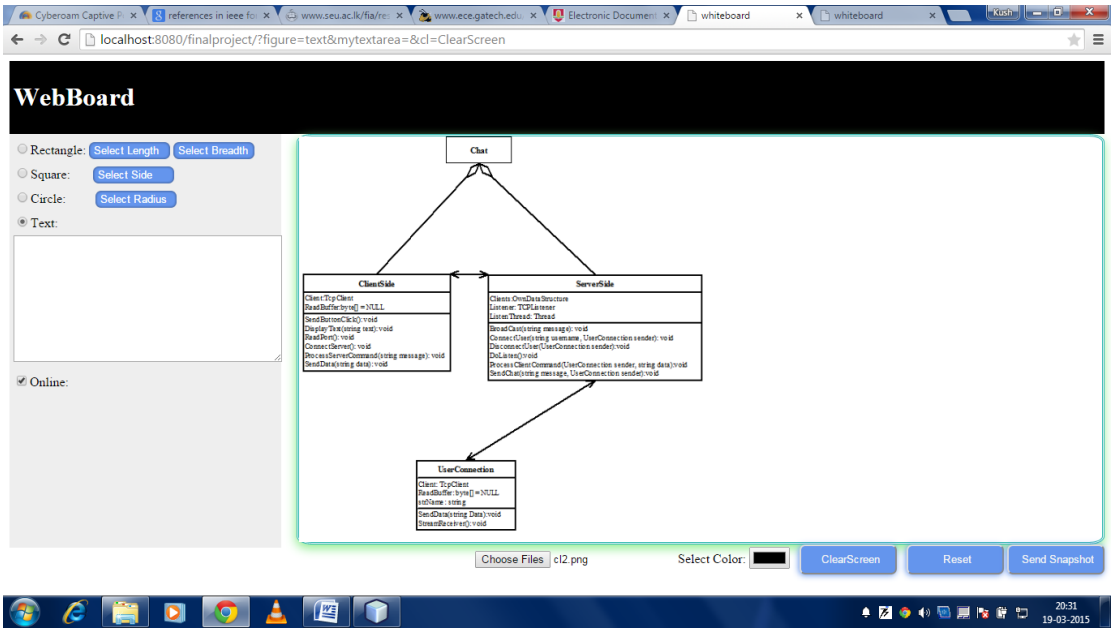

Figure: 8.2.9

As we can see above that we have one GUI of presenter or teacher and we have two GUI of participants or students connected to the server, so whatever image the teacher is uploading on his whiteboard area it is getting reflected to all the connected clients or students.

Presenter's Graphical user Interface:-

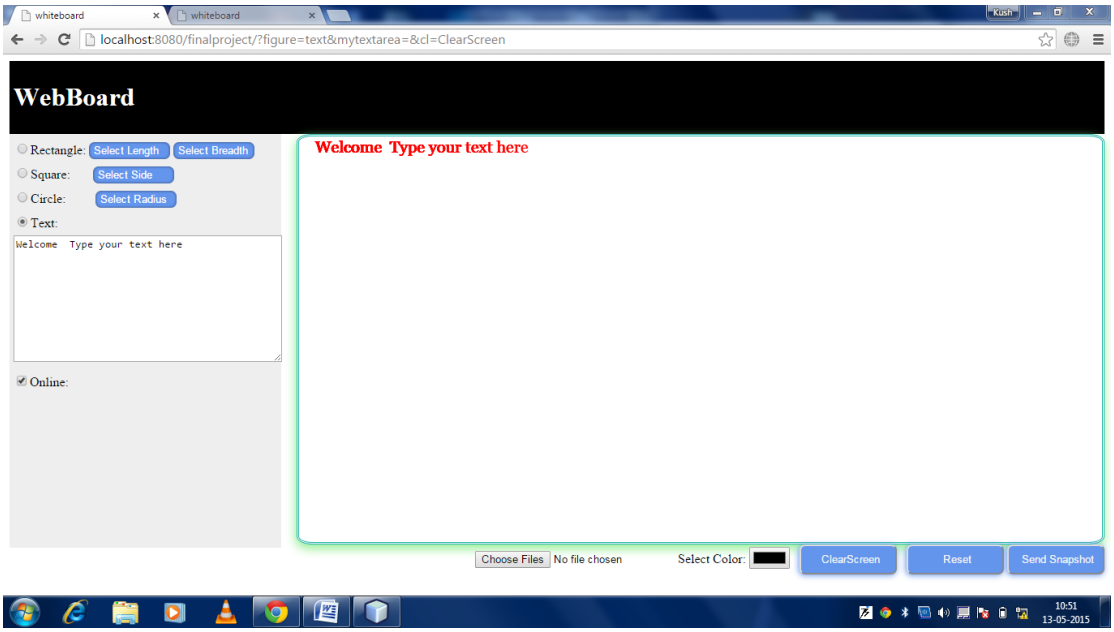

Figure: 8.2.10

Participant 1 Graphical User Interface:-

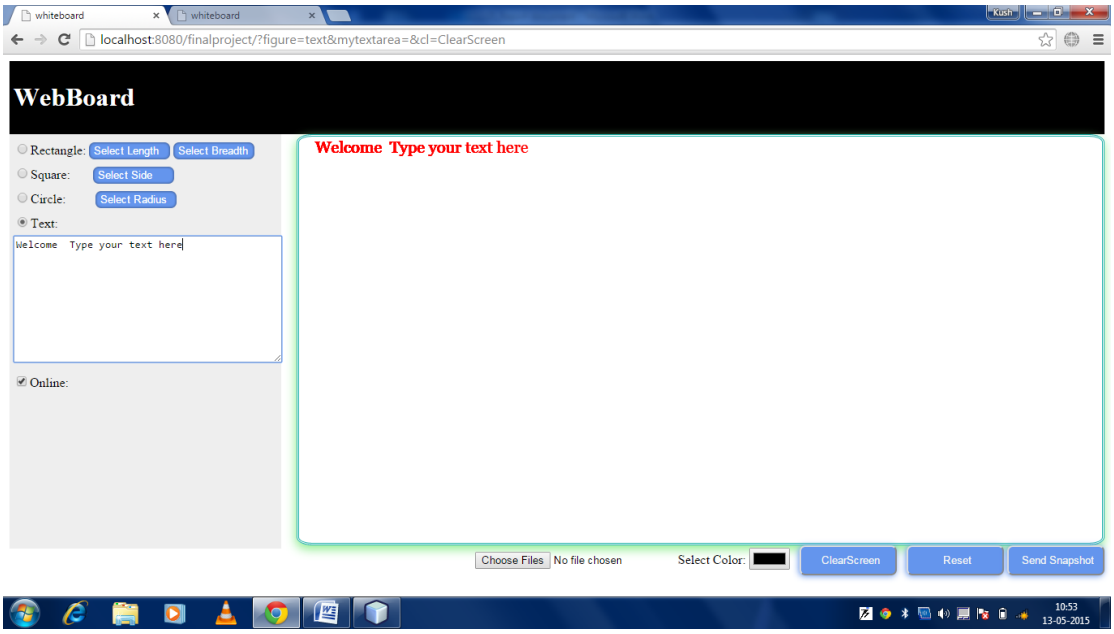

Figure: 8.2.11

Participant 2 Graphical User Interface:-

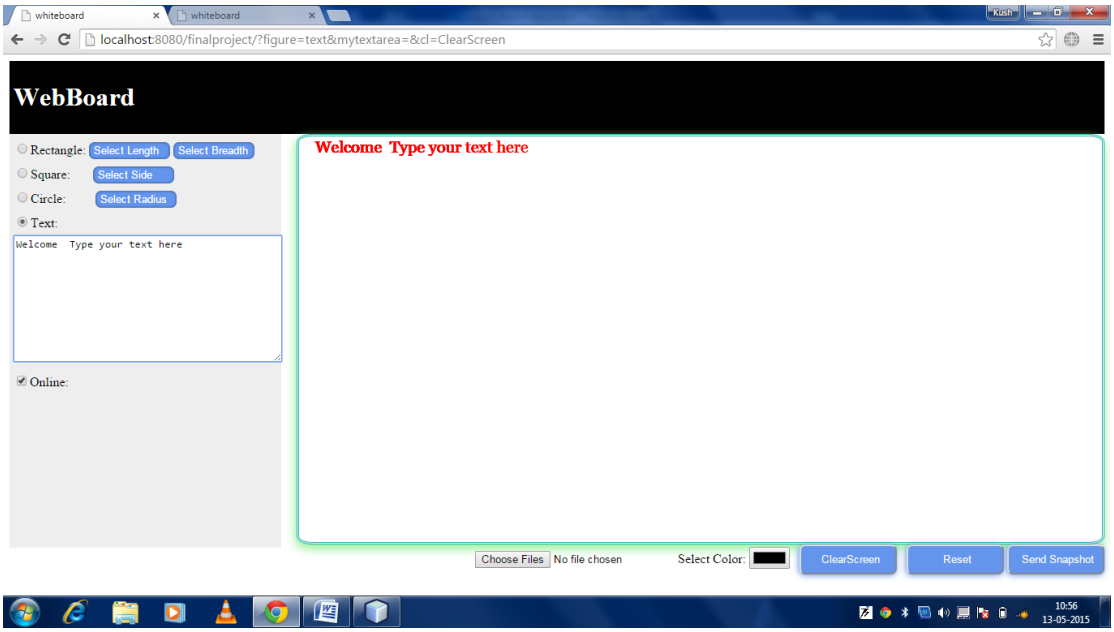

Figure: 8.2.12

As we can see above that we have one GUI of presenter or teacher and we have two GUI of participants or students connected to the server, so whatever text the teacher is typing through the keyboard it is getting reflected to all the connected clients or students.

Conclusion and Future Work

Conclusion

We have implemented most of the functionalities of the proposed Shared Whiteboard. Our Shared Whiteboard will be used for conducting online classes. It has mainly two users, presenter and participant. A presenter will conduct a class for participants, which will attend the class from anywhere in the world.

Presenter can conduct classes for participants. As a presenter he can upload files to the whiteboard, he can give presentation to participants, also he can use white board, he can answer participants doubts using chat facility.
When participant enters in classroom, he can attend the class. He is able to download the resources submitted by presenter; he can chat with other participants in classroom
Also the presenter can type the text using the keyboard, which will get displayed on the whiteboard, and will also get transferred to all the connected clients.

Future Work

The Shared Whiteboard developed is quite useful in many aspects. Through this, classes can be conducted on the Internet and participants from all over world can attend it. Though this is currently usable, it can be improved in many dimensions. As the facility of real time audio can also be implemented, so this system could be extended in this dimension. It can also be extended by implementing the feature of video chat.

References

1. “9 Uses of Web sockets”. [Online].Available: http://www.infoworld.com/article/2609720/application-development/9-killer-uses-for-websockets.html [Accessed: Dec 14, 2014].
2. “An Introduction to Web Sockets”. [Online].Available: http://blog.teamtreehouse.com/an-introduction-to-websockets .[Accessed: Nov 20,2014].
3. “An Online Virtual Learning Environment For Higher Education”. [Online].Available:http://www.academia.edu/889403/An_Online_Virtual_Learning_Environment_for_Higher_Education [Accessed: Nov 28, 2014].
4. “Characteristics of a Virtual Classroom”. [Online].Available: http://www.learndash.com/characteristics-of-a-virtual-classroom/ [Accessed: Dec 14, 2014].
5. **“**Common Features of a Virtual Classroom Software Program**”.** [Online].Available: https://www.td.org/Publications/Newsletters/Learning-Circuits/Learning-Circuits-Archives/2010/04/Common-Features-of-a-Virtual-Classroom-Software-Program [Accessed: Dec 18, 2014].
6. “The UML Class Diagram: Part 1”. [Online].Available: http://www.developer.com/design/article.php/2206791/The-UML-Class-Diagram-Part-1.htm [Accessed: Dec 11, 2014].
7. “Visual Use case”. [Online].Available: http://www.visualusecase.com/use-case/use-case.html [Accessed: Dec 18, 2014].
8. “Web Application Technologies”. [Online].Available: http://www.extropia.com/presentations/birznieks/pdf/web_application_technologies.pdf [Accessed: Dec 11, 2014].
9. “Web Socket”. [Online].Available:http://en.wikipedia.org/wiki/WebSocket [Accessed: Nov 20,2014].
10. “What Are Web sockets?” [Online].Available: http://www.pubnub.com/blog/what-are-websockets/ [Accessed: Dec 18, 2014].

Appendix

Index.html

```
<%--
    Document   : draw
    Created on : 10 Feb, 2015, 2:28:50 AM
    Author     : kush
--%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>whiteboard</title>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <style>
            #header {
                background-color:black;
                color:white;
                text-align:left;
                padding:5px;
            }
            #nav {
                line-height:30px;
                background-color:#eeeeee;
                height:500px;
                width:325px;
                float:left;
                padding:5px;
            }
            #section {
                width:1000px;
                height:500px;
                //  float:right;
```

```
    // padding:10px;
    box-shadow: -1px 5px 18px lightgreen;
    border-color: lightgray;height:500px;
    width: 990px;
    padding: 0px 0px 0px 0px;
    border-style: double;
    border-width: 3px;
    border-color: lightseagreen;
    border-radius: 2% 2% 2% 2%;
}
#footer {
    background-color:black;
    color:white;
    clear:both;
    text-align:center;
    padding:10px;
}
.button{
    text-indent: 5px;
    //     text-rendering: optimizeSpeed;
    color: whitesmoke;
    background-color: cornflowerblue;
    box-shadow: -1px 1px 10px cornflowerblue;
    border-radius: 7px 7px 7px 7px;
    width: 120px;
    height: 37px;
}
select {
    border: 0 !important;  /*Removes border*/
    -webkit-appearance: none;  /*Removes default chrome and safari style*/
    //  -moz-appearance: none; /* Removes Default Firefox style*/
    background-color:cornflowerblue ;
    width: 100px; /*Width of select dropdown to give space for arrow image*/
    text-indent: 0.01px; /* Removes default arrow from firefox*/
```

```
            text-overflow: "";  /*Removes default arrow from firefox*/ /*My custom style for fonts*/
            color: #FFF;
            border-radius: 7px;
            padding: 2px;
            box-shadow: inset 0 0 5px rgba(000,000,000, 0.5);
          }
      </style>
   </head>
   <body>
      <div id="header">
         <h1>WebBoard</h1>
      </div>
      <div id="section" style="float:right;">
         <div id="sketch">
            <canvas id="myCanvas" width="1000" height="500" ></canvas>
         </div>
      </div>
      <form name="inputForm">
         <div id="nav" style="float: left;">
            <input type="radio" id="r1" name="figure" value="rectangle" onclick="hidetextarea();" >Rectangle:
            <select id="mySelect1">
               <option>Select Length</option>
               <option>125</option>
               <option>150</option>
               <option>200</option>
               <option>250</option>
               <option>300</option>
               <option>350</option>
               <option>400</option>
            </select>
            <select id="mySelect2">
               <option>Select Breadth</option>
```

```
                <option>125</option>
                <option>150</option>
                <option>200</option>
                <option>250</option>
                <option>300</option>
                <option>350</option>
                <option>400</option>
            </select>              <br>
            <input type="radio" id="r2" name="figure" value="square"
onclick="hidetextarea();">Square:
             
             
             
            <select id="mySelect3">
                <option>Select Side</option>
                <option>125</option>
                <option>150</option>
                <option>200</option>
                <option>250</option>
                <option>300</option>
                <option>350</option>
                <option>400</option>
            </select>
            <br>
            <input type="radio" id="r3" name="figure" value="circle"
onclick="hidetextarea();">Circle:
             
             
             
             
            <select id="mySelect4">
                <option>Select Radius</option>
                <option>25</option>
                <option>50</option>
```

```
                <option>60</option>
                <option>70</option>
                <option>80</option>
                <option>90</option>
                <option>100</option>
            </select>
            <br>
<input type="radio" id="r4" name="figure" checked value="text"
onclick="showtextarea();">Text:
            <br>
            <textarea title="textarea" rows="10" cols="44" id="x13"
name="mytextarea"></textarea>
<input type="checkbox" id="instant" value="Online" checked="true">Online:
        </div>
        <div id="nav5" style="float:right;">
            <input type="file" id="files" name="files[]" multiple />
             
            Select Color: <input type="color" id="myColor">
             
            <input type="submit" id="clear" name="cl" value="ClearScreen"
class="button" onclick="clearscreen();
                return false;">
             
            <button type="reset" value="Reset" class="button"
onClick="hidetextarea()">Reset</button>
            <input type="submit" value="Send Snapshot" class="button"
onclick="defineImageBinary();
                return false;">
        </div>
    </form>
    <script type="text/javascript" src="websocket.js"></script>
    <script type="text/javascript" src="whiteboard.js"></script>
  </body>
</html>
```

Websocket.js

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
var wsUri = "ws://" + document.location.host + document.location.pathname + "whiteboardendpoint";
var websocket = new WebSocket(wsUri);
websocket.binaryType = "arraybuffer";
websocket.onmessage = function (evt) {
   onMessage(evt);
};
websocket.onerror = function (evt) {
   onError(evt);
};
function sendText(json) {
   console.log("sending text: " + json);
   websocket.send(json);
}
function cle(json1) {
   websocket.send(json1);
}
function sendBinary(bytes) {
   console.log("sending binary: " + Object.prototype.toString.call(bytes));
   websocket.send(bytes);
}
function onMessage(evt) {
   console.log("received: " + evt.data);
   if (typeof evt.data == "string") {
      drawImageText(evt.data);
      clsc(evt.data);
   } else {
```

```
        drawImageBinary(evt.data);
    }
}
function onError(evt) {
    writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
}
```

Whiteboard.js

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
var color = document.inputForm.myColor.value;
var canvas = document.getElementById("myCanvas");
var context = canvas.getContext("2d");
//canvas.addEventListener('touchmove', defineImageBinary, false);
canvas.addEventListener("click", defineImageBinary, false);
canvas.addEventListener("click", defineImage, false);
var text = document.getElementById("x13");
text.addEventListener('keyup', textBoxChanged, false);
text.addEventListener("keyup", defineImageBinary, false);
function textBoxChanged(e) {
    var target = e.target;
    message = target.value;
    write();
}
function write() {
    //var font1 = document.inputForm.mySelect5;
    context.font = "20px Georgia";
    context.fillStyle = "#FF0000";
    context.fillText(message, 20, 20);
}
```

```
function getCurrentPos(evt) {
    var rect = canvas.getBoundingClientRect();
    return {
        x: evt.clientX - rect.left,
        y: evt.clientY - rect.top
    };
}
function defineImage(evt) {
    var currentPos = getCurrentPos(evt);
    for (i = 0; i < document.inputForm.figure.length; i++) {
        if (document.inputForm.figure[i].checked) {
            var shape = document.inputForm.figure[i];
            break;
        }
    }
    var color = document.inputForm.myColor;
    var size1 = document.inputForm.mySelect1;
    var size2 = document.inputForm.mySelect2;
    var size3 = document.inputForm.mySelect3;
    var size4 = document.inputForm.mySelect4;
    var json = JSON.stringify({
        "shape": shape.value,
        "color": color.value,
        "size1": size1.value,
        "size2": size2.value,
        "size3": size3.value,
        "size4": size4.value,
        "coords": {
            "x": currentPos.x,
            "y": currentPos.y
        }

    });
    drawImageText(json);
```

```
    if (document.getElementById("instant").checked) {
        sendText(json);
    }
}
/*function clearscreen()
 {
 var clr = document.inputForm.clear;
 var json1 = JSON.stringify({
 "clr": clr.value
 });
 clsc(json1);
 cle(json1);
 }*/
/*function clsc(image1) {
 var json1 = JSON.parse(image1);
 if(json1.clr.value.equals("ClearScreen")){
 var canvas = document.getElementById("myCanvas");
 var context = canvas.getContext("2d");
 context.clearRect(0, 0, 1100, 500);
 }
  }*/
function drawImageText(image) {
//    console.log("drawImageText");
    var json = JSON.parse(image);
    context.fillStyle = json.color;
    switch (json.shape) {
        case "rectangle":
            context.fillRect(json.coords.x, json.coords.y, json.size1, json.size2);
            // rect = {},
            //         drag = false;
            //canvas.addEventListener('mousedown', startpoints, false);
            //canvas.addEventListener('mouseup', endpoints, false);
            // canvas.addEventListener('mousemove', drawrect, false);
            // function startpoints(e) {
```

```
            //  rect.startX = e.pageX - this.offsetLeft;
            //  rect.startY = e.pageY - this.offsetTop;
            // drag = true;
            //}
            // function endpoints() {
            //    drag = false;
            //}
            ////function drawrect(e) {
            //  if (drag) {
            //     rect.w = (e.pageX - this.offsetLeft) - rect.startX;
            //     rect.h = (e.pageY - this.offsetTop) - rect.startY;
            //   //ctx.clearRect(0,0,canvas.width,canvas.height);
            //   context.fillRect(rect.startX, rect.startY, rect.w, rect.h);
            // }
            //}
               break;
         case "square":
         default:
               context.fillRect(json.coords.x, json.coords.y, json.size3, json.size3);
               break;
         case "circle":
               context.beginPath();
               context.arc(json.coords.x, json.coords.y, json.size4, 0, 2 * Math.PI, false);
               context.fill();
               break;
      }
}
function showtextarea() {
   document.getElementById("x13").style.display = 'block';
}
function hidetextarea() {
   document.getElementById("x13").style.display = 'none';
}
```

```
function drawImageBinary(blob) {
    var bytes = new Uint8Array(blob);
//    console.log('drawImageBinary (bytes.length): ' + bytes.length);
    var imageData = context.createImageData(canvas.width, canvas.height);
    for (var i = 8; i < imageData.data.length; i++) {
        imageData.data[i] = bytes[i];
    }
    context.putImageData(imageData, 0, 0);
    var img = document.createElement('img');
    img.height = canvas.height;
    img.width = canvas.width;
    img.src = canvas.toDataURL();
}
function defineImageBinary() {
    var image = context.getImageData(0, 0, canvas.width, canvas.height);
    var buffer = new ArrayBuffer(image.data.length);
    var bytes = new Uint8Array(buffer);
    for (var i = 0; i < bytes.length; i++) {
        bytes[i] = image.data[i];
    }
    sendBinary(buffer);
}
document.getElementById('files').addEventListener('change', doUpload, false);
function doUpload(e) {
    // The user might upload multiple files, we'll take the first
    var file = e.target.files[0];
// Check that there is a file uploaded
    if (file) {
        // We need to use a FileReader to actually read the file.
        var reader = new FileReader();
// When it's loaded, we'll send the image data to the canvas method
        reader.onload = function (event) {
            canvasLoadImage(event.target.result);
        };
```

```
// Pass the reader the file to read and give us the DataURL
        reader.readAsDataURL(file);
    }
}
// Handle the returned image data
function canvasLoadImage(imgData) {
    // Create a New Imgae
    var img = new Image();
// Assign the image data as the source - as we are using a data URL
    img.src = imgData;
// Always use onload with images!
    img.onload = function () {
// Load the Canvas & Context
        var canvas = document.getElementById('myCanvas');
        var context = canvas.getContext('2d');
// Draw the image
        context.drawImage(img, 0, 0, 500, 500);
       };
}
(function () {
    var canv = document.querySelector('#myCanvas');
    var ctx = canv.getContext('2d');
    var sketch = document.querySelector('#sketch');
    var sketch_style = getComputedStyle(sketch);
    canv.width = parseInt(sketch_style.getPropertyValue('width'));
    canv.height = parseInt(sketch_style.getPropertyValue('height'));
    var mouse = {x: 0, y: 0};
    /* Mouse Capturing Work */
    canv.addEventListener('mousemove', function (e) {
        mouse.x = e.pageX - this.offsetLeft;
        mouse.y = e.pageY - this.offsetTop;
    }, false);
    /* Drawing on Paint App */
```

```
    ctx.lineWidth = 8;
    ctx.lineJoin = 'round';
    ctx.lineCap = 'round';
    ctx.strokeStyle = "blue";
    canv.addEventListener('mousedown', function (e) {
        ctx.beginPath();
        ctx.moveTo(mouse.x, mouse.y);
        canv.addEventListener('mousemove', onPaint, false);
    }, false);
    canv.addEventListener('mouseup', function () {
        canv.removeEventListener('mousemove', onPaint, false);
    }, false);
    var onPaint = function () {
        ctx.lineTo(mouse.x, mouse.y);
        ctx.stroke();
    };
}());
```

Coordinate.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package whiteboardapp;

/**
 *
 * @author nb
 */
public class Coordinates {
        private float x;
```

```
    private float y;
public Coordinates() {
    }
    public Coordinates(float x, float y) {
        this.x = x;
        this.y = y;
    }
    public float getX() {
        return x;
    }
    public void setX(float x) {
        this.x = x;
    }
    public float getY() {
        return y;
    }
    public void setY(float y) {
        this.y = y;
    }
    }
```

Figure.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package whiteboardapp;
import java.io.StringWriter;
import javax.json.Json;
import javax.json.JsonObject;

/**
```

```
 *
 * @author nb
 */
public class Figure {
    private JsonObject json;
    public Figure() {
    }
        @Override
    public String toString() {
        StringWriter writer = new StringWriter();
        Json.createWriter(writer).write(json);
        return writer.toString();
    }
public Figure(JsonObject json) {
        this.json = json;
    }
    public JsonObject getJson() {
        return json;
    }
    public void setJson(JsonObject json) {
        this.json = json;
    }

}
```

FigureDecoder.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package whiteboardapp;
```

```
import java.io.StringReader;
import javax.json.Json;
import javax.json.JsonException;
import javax.json.JsonObject;
import javax.websocket.DecodeException;
import javax.websocket.Decoder;
import javax.websocket.EndpointConfig;

/**
 *
 * @author nb
 */
public class FigureDecoder implements Decoder.Text<Figure> {
    @Override
    public Figure decode(String string) throws DecodeException {
        System.out.println("decoding: " + string);
        JsonObject jsonObject = Json.createReader(new
StringReader(string)).readObject();
        return new Figure(jsonObject);
    }
    @Override
    public boolean willDecode(String string) {
        try {
            Json.createReader(new StringReader(string)).readObject();
            return true;
        } catch (JsonException ex) {
            ex.printStackTrace();
            return false;
        }
    }
    @Override
    public void init(EndpointConfig config) {
        System.out.println("init");
    }
```

```
    @Override
    public void destroy() {
        System.out.println("destroy");
    }
    }
```

FigureEncoder.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package whiteboardapp;
import javax.websocket.EncodeException;
import javax.websocket.Encoder;
import javax.websocket.EndpointConfig;

/**
 *
 * @author nb
 */
public class FigureEncoder implements Encoder.Text<Figure> {
@Override
    public String encode(Figure figure) throws EncodeException {
        return figure.getJson().toString();
    }
    @Override
    public void init(EndpointConfig config) {
        System.out.println("init");
    }
    @Override
    public void destroy() {
        System.out.println("destroy");
    }
```

}

MyWhiteboard.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package whiteboardapp;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;
import javax.websocket.EncodeException;
import javax.websocket.OnClose;
import javax.websocket.OnMessage;
import javax.websocket.OnOpen;
import javax.websocket.Session;
import javax.websocket.server.ServerEndpoint;

/**
 *
 * @author kush
 */
@ServerEndpoint(value = "/whiteboardendpoint",
        encoders = {FigureEncoder.class},
        decoders = {FigureDecoder.class})
public class MyWhiteboard {
    private static final Set<Session> peers = Collections.synchronizedSet(new
HashSet<Session>());
    @OnOpen
```

```
    public void onOpen(Session peer) {
        peers.add(peer);
    }
        @OnClose
    public void onClose(Session peer) {
        peers.remove(peer);
    }
@OnMessage
    public void broadcastFigure(Figure figure, Session session) throws IOException,
EncodeException {
        System.out.println("broadcastFigure: " + figure);
        for (Session peer : peers) {
            if (!peer.equals(session)) {
                peer.getBasicRemote().sendObject(figure);
            }
        }
    }
    @OnMessage
    public void broadcastSnapshot(ByteBuffer data, Session session) throws
IOException {
        System.out.println("broadcastBinary: " + data);
        for (Session peer : peers) {
            if (!peer.equals(session)) {
                peer.getBasicRemote().sendBinary(data);
            }
        }
    }
}
```